UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Ingeniería en Sistemas Informáticos Computacionales

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENÍERA EN SISTEMAS COMPUTACIONALES

TEMA:

Estudio de la metodología de desarrollo de Software Open Up (Open Unified Process), aplicado al desarrollo de aplicaciones Web mediante la utilización del Framework ZK-JSP.

APLICATIVO:

"Sistema de control y administración de citas médicas e historias clínicas (SISMED) para el departamento médico de la empresa productora y exportadora de flores "Inversiones Ponte Tresa S.A""

MANUAL TÉCNICO

Autor:

Diego Edwin Guerrón Coral

Director:

Ing. Pedro Granda

Ibarra – Ecuador

2016

PAQUETE com.product.authentication

CLASE MYUSERDETAILSSERVICE

```
package com.product.authentication;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.GrantedAuthorityImpl;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import com.product.business.service.CRUDService;
import com.product.domain.Users;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.msjUtilitarios;

@SuppressWarnings("deprecation")
public class MyUserDetailsService implements UserDetailsService {

	@Autowired
	private CRUDService crudService;

	/*
	 * You just have to make sure that the user-by-username-query returns
three
	 * fields. 1) the userName 2) the password 3) boolean for is the user
	 * active. If you don't have an active field, make your query always
return
	 * true for that third field.
	 */
	@Override
	public UserDetails loadUserByUsername(String userName)
			throws UsernameNotFoundException, DataAccessException {

		// Declare a null Spring User
		UserDetails user = null;

		try {

			// Search database for a user that matches the specified
username
			// You can provide a custom DAO to access your persistence
layer
			// Or use JDBC to access your database
			// DbUser is our custom domain user. This is not the same as
```

```
                    // Spring's User

                    final Map<String, Object> params = new HashMap<String,
Object>();
                    params.put("username", userName);
                    Users dbUser = crudService.GetUniqueEntityByNamedQuery(
                                "Users.findActiveUserByUserID", new Object[] {
userName, });

                    // Populate the Spring User object with details from the
dbUser
                    // Here we just pass the username, password, and access level
                    // getAuthorities() will translate the access level to the
correct
                    // role type
                    msjUtilitarios util= new msjUtilitarios();
                    String pass= util.Desencriptar(dbUser.getPassword());

                    user = new User(dbUser.getUserName(), pass, true,
                                true, true, true, getAuthorities());
                    FHSessionUtil.setCurrentUser(dbUser);

              } catch (Exception e) {
                    System.out.println(e);
                    throw new UsernameNotFoundException("Error in retrieving
user");
              }

              // Return user to Spring for processing.
              // Take note we're not the one evaluating whether this user is
              // authenticated or valid
              // We just merely retrieve a user that matches the specified
username

              return user;
        }

        /**
         * Retrieves the correct ROLE type depending on the access level, where
         * access level is an Integer. Basically, this interprets the access
value
         * whether it's for a regular user or admin.
         *
         * @param access
         *            an integer value representing the access of the user
         * @return collection of granted authorities
         */
        public Collection<GrantedAuthority> getAuthorities() {
              // Create a list of grants for this user
              List<GrantedAuthority> authList = new
ArrayList<GrantedAuthority>(2);
              // All users are granted with ROLE_USER access
```

```
            // Therefore this user gets a ROLE_USER by default
            authList.add(new GrantedAuthorityImpl("ROLE_USER"));
            // User has admin access
            authList.add(new GrantedAuthorityImpl("ROLE_ADMIN"));
            // Return list of granted authorities
            return authList;
      }
}
```

PAQUETE com.product.business.dao

CLASE AUTHDAO

```
package com.product.business.dao;

import java.util.List;

import com.product.webapp.utilities.MenusForThisUser;

public interface AuthDao {

      public List<MenusForThisUser> getMenuForThisUser(Long roleID);
      public List<MenusForThisUser> getMenuForSystemUser();
      public void deleteRolesMenu(Long roleID);
      public void deleteRolesMenu(List<Long> prAppMenuIDs, Long roleID);

}
```

PAQUETE com.product.business.dao

CLASE AUTHDAOIMPL

```
package com.product.business.dao;

import java.util.List;
import org.hibernate.Query;
import org.hibernate.SQLQuery;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.transform.Transformers;
import org.hibernate.type.IntegerType;
import org.hibernate.type.LongType;
import org.hibernate.type.StringType;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.product.webapp.utilities.MenusForThisUser;

@Repository
public class AuthDaoImpl implements AuthDao {
```

```
	@Autowired
	SessionFactory sessionFactory;

	protected final Session getCurrentSession() {
		return sessionFactory.getCurrentSession();
	}

	@Override
	@SuppressWarnings("unchecked")
	public List<MenusForThisUser> getMenuForSystemUser() {
		SQLQuery q1 = getCurrentSession()
				.createSQLQuery(
					"SELECT  a.menuCaption as mCaption,
a.menuOrder as mOrder , a.appMenuID as mMenuID, a.parentAppMenuID as mPrMenuID,
"
						+ "a.canAdd as canAdd,
a.canEdit as canEdit, a.CanDelete as canDelete, a.canView as canView, "
						+ "a.MenuLevel as menuLevel,
a.UniqueAppMenu as uniqueAppMenu, a.UniqueFileName as uniqueFileName "
						+ "FROM appmenus a     order
by a.menuOrder");

		q1.addScalar("mCaption", StringType.INSTANCE);
		q1.addScalar("mOrder", IntegerType.INSTANCE);
		q1.addScalar("mMenuID", LongType.INSTANCE);
		q1.addScalar("mPrMenuID", LongType.INSTANCE);
		q1.addScalar("canAdd", IntegerType.INSTANCE);
		q1.addScalar("canEdit", IntegerType.INSTANCE);
		q1.addScalar("canDelete", IntegerType.INSTANCE);
		q1.addScalar("canView", IntegerType.INSTANCE);
		q1.addScalar("menuLevel", IntegerType.INSTANCE);
		q1.addScalar("uniqueAppMenu", StringType.INSTANCE);
		q1.addScalar("uniqueFileName", StringType.INSTANCE);
		q1.setResultTransformer(Transformers
				.aliasToBean(MenusForThisUser.class));
		return q1.list();

	}

	@Override
	@SuppressWarnings("unchecked")
	public List<MenusForThisUser> getMenuForThisUser(Long roleID) {
		String SQL = "";
		SQL = "SELECT  a.menuCaption as mCaption, a.menuOrder as mOrder ,
b.PrAppMenuID as mMenuID, a.ParentAppMenuID as mPrMenuID, "
			+ "b.canAdd as canAdd, b.canEdit as canEdit,
b.CanDelete as canDelete, b.canView as canView, "
			+ "a.MenuLevel as menuLevel, a.UniqueAppMenu as
uniqueAppMenu, a.UniqueFileName as uniqueFileName "
			+ "FROM appmenus a,  rolesmenu b "
			+ "WHERE a.AppMenuID = b.PrAppMenuID "
			+ "AND "
```

```
                        + "b.roleID = ? ";

                SQL = SQL + " order by a.menuOrder  ";

                SQLQuery q1 = getCurrentSession().createSQLQuery(SQL);

                q1.addScalar("mCaption", StringType.INSTANCE);
                q1.addScalar("mOrder", IntegerType.INSTANCE);
                q1.addScalar("mMenuID", LongType.INSTANCE);
                q1.addScalar("mPrMenuID", LongType.INSTANCE);
                q1.addScalar("canAdd", IntegerType.INSTANCE);
                q1.addScalar("canEdit", IntegerType.INSTANCE);
                q1.addScalar("canDelete", IntegerType.INSTANCE);
                q1.addScalar("canView", IntegerType.INSTANCE);
                q1.addScalar("menuLevel", IntegerType.INSTANCE);
                q1.addScalar("uniqueAppMenu", StringType.INSTANCE);
                q1.addScalar("uniqueFileName", StringType.INSTANCE);
                q1.setParameter(0, roleID);
                q1.setResultTransformer(Transformers
                                .aliasToBean(MenusForThisUser.class));
                return q1.list();

        }
        @Override
        public void deleteRolesMenu(Long roleID) {
                String query = "";
                query = "delete from RolesMenu  where   roleID = :roleID ";
                Query q = getCurrentSession().createSQLQuery(query);
                q.setParameter("roleID", roleID);
                q.executeUpdate();
        }

        @Override
        public void deleteRolesMenu(List<Long> prAppMenuIDs, Long roleID) {
                String query = "delete from RolesMenu where prAppMenuID in
(:prAppMenuIDs) and role.ID =:roleID ";
                Query q = getCurrentSession().createQuery(query);
                q.setParameterList("prAppMenuIDs", prAppMenuIDs);
                q.setParameter("roleID", roleID);
                q.executeUpdate();

        }
}
```

PAQUETE com.product.business.dao

CLASE CRUDDAO

```
package com.product.business.dao;

import java.io.Serializable;
import java.util.List;
```

```
public interface CRUDDao {
	<T> List<T> getAll(Class<T> klass);

	<T> void Save(T klass);

	<T> void Update(T klass);

	<T> T findByPrimaryKey(Class<T> klass, Serializable id);

//	<T> T GetUniqueEntityByNamedQuery(String query, final Map<String, ?
extends Object> params);

	<T> T GetUniqueEntityByNamedQuery(String query, Object... params);
	<T> List<T> GetListByNamedQuery(String query, Object... params);

	<T> void delete(T klass);

	<T> Long getQueryCount(String query, Object... params);
}
```

PAQUETE com.product.business.dao

CLASE CRUDDAOIMPL

```
package com.product.business.dao;

import java.io.Serializable;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Repository;

@Repository
public class CRUDDaoImpl implements CRUDDao {

	@Autowired
	SessionFactory sessionFactory;
	@Override
	@SuppressWarnings("unchecked")
	public <T> List<T> getAll(Class<T> klass) {
		return getCurrentSession().createQuery("from " + klass.getName())
				.list();
	}

	protected final Session getCurrentSession() {
		return sessionFactory.getCurrentSession();
	}

	@Override
	public <T> void Save(T klass) throws DataAccessException {
```

```
			getCurrentSession().saveOrUpdate(klass);
		}

		@Override
		public <T> void delete(T klass) throws DataAccessException {
			getCurrentSession().delete(klass);

		}

		/**
		 * Retrieve an object that was previously persisted to the database using
		 * the indicated id as primary key
		 */
		@Override
		@SuppressWarnings("unchecked")
		public <T> T findByPrimaryKey(Class<T> klass, Serializable id) {
			return (T) getCurrentSession().get(klass, id);
		}
		@Override
		@SuppressWarnings("unchecked")
		public <T> List<T> GetListByNamedQuery(String query, Object... params) {
			Query q = getCurrentSession().getNamedQuery(query);

			int i = 1;
			String arg ="arg";
			if (params != null) {
				for (Object o : params) {
					if (o != null) {
						q.setParameter(arg+i, o);
						i++;
					}
				}
			}
			List<T> list = q.list();
			return list;
		}
		@Override
		@SuppressWarnings("unchecked")
		public <T> T GetUniqueEntityByNamedQuery(String query, Object... params)
{

			Query q = getCurrentSession().getNamedQuery(query);

			int i = 1;
			String arg ="arg";
			for (Object o : params) {
				q.setParameter(arg+i, o);
				i++;
			}

			List<T> results = q.list();
```

```
			T foundentity = null;
			if (!results.isEmpty()) {
				// ignores multiple results
				foundentity = results.get(0);
			}
			return foundentity;
		}
		@Override
		public <T> Long getQueryCount(String query, Object... params) {

			Query q = getCurrentSession().getNamedQuery(query);
			int i = 1;
			String arg ="arg";
			Long count = (long) 0;

			if (params != null) {
				for (Object o : params) {
					if (o != null) {
						q.setParameter(arg+i, o);
						i++;
					}
				}
			}
			count = (Long) q.uniqueResult();
			return count;
		}

		@Override
		public <T> void Update(T klass) {
			getCurrentSession().update(klass);

		}

}
```

PAQUETE com.product.business.service
CLASE AUTHSERVICE

```
package com.product.business.service;

import java.util.List;
import com.product.webapp.utilities.MenusForThisUser;

public interface AuthService extends CRUDService {

		public List<MenusForThisUser> getMenuForThisUser(Long roleID);
		public List<MenusForThisUser> getMenuForSystemUser();
		public void deleteRolesMenu(Long roleID);
		public void deleteRolesMenu(List<Long> prAppMenuIDs, Long roleID);

}
```

PAQUETE com.product.business.service
CLASE AUTHSERVICEIMPL

```
package com.product.business.service;

import java.io.Serializable;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.product.business.dao.AuthDao;
import com.product.business.dao.CRUDDao;
import com.product.webapp.utilities.MenusForThisUser;

@Service
public class AuthServiceImpl implements AuthService {

	@Autowired
	private CRUDDao CRUDDao;

	@Autowired
	private AuthDao AuthDao;

	@Override
	@Transactional(readOnly = true)
	public <T> List<T> getAll(Class<T> klass) {
		return CRUDDao.getAll(klass);
	}

	@Override
	@Transactional
	public <T> T findByPrimaryKey(Class<T> klass, Serializable id) {
		return CRUDDao.findByPrimaryKey(klass, id);
	}

	@Override
	@Transactional
	public <T> void Save(T klass) throws DataAccessException {
		CRUDDao.Save(klass);
	}

	@Override
	@Transactional
	public <T> void delete(T klass) throws DataAccessException {
		CRUDDao.delete(klass);
	}

	@Override
	public <T> T GetUniqueEntityByNamedQuery(String query, Object... params)
{
		return CRUDDao.GetUniqueEntityByNamedQuery(query, params);
```

```
	}

	@Override
	public <T> List<T> GetListByNamedQuery(String query, Object... params) {
		return CRUDDao.GetListByNamedQuery(query, params);
	}

	@Override
	@Transactional
	public <T> Long getQueryCount(String query, Object... params) {
		return CRUDDao.getQueryCount(query, params);
	}

	@Override
	@Transactional
	public List<MenusForThisUser> getMenuForThisUser(Long roleID) {
		return AuthDao.getMenuForThisUser(roleID);
	}

	@Override
	@Transactional
	public List<MenusForThisUser> getMenuForSystemUser() {
		return AuthDao.getMenuForSystemUser();
	}

	@Override
	@Transactional
	public void deleteRolesMenu(Long roleID) {
		AuthDao.deleteRolesMenu(roleID);
	}
	@Override
	@Transactional
	public void deleteRolesMenu(List<Long> prAppMenuIDs, Long roleID){
		AuthDao.deleteRolesMenu(prAppMenuIDs,roleID);
	}

	@Override
	public <T> void Update(T klass) {
		CRUDDao.Update(klass);

	}

}
```

PAQUETE com.product.business.service
CLASE CRUDSERVICE

```
package com.product.business.service;

import java.io.Serializable;
import java.util.List;
```

```
public interface CRUDService {
	<T> List<T> getAll(Class<T> klass);

	<T> void Save(T klass);

	<T> void Update(T klass);

	<T> T findByPrimaryKey(Class<T> klass, Serializable id);

	<T> void delete(T klass);

	public <T> T GetUniqueEntityByNamedQuery(String query, Object... params);

	<T> List<T> GetListByNamedQuery(String query, Object... params);

	<T> Long getQueryCount(String query, Object... params);
}
```

PAQUETE com.product.business.service
CLASE CRUDSERVICEIMPL

```
package com.product.business.service;

import java.io.Serializable;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.product.business.dao.CRUDDao;

@Service
public class CRUDServiceImpl implements CRUDService {

	@Autowired
	private CRUDDao CRUDDao;

	@Override
	@Transactional(readOnly = true)
	public <T> List<T> getAll(Class<T> klass) {
		return CRUDDao.getAll(klass);
	}

	@Override
	@Transactional
	public <T> void Save(T klass) throws DataAccessException {
		CRUDDao.Save(klass);
	}

	@Override
	@Transactional
```

```
	public <T> void delete(T klass) throws DataAccessException {
		CRUDDao.delete(klass);
	}

	@Override
	public <T> T GetUniqueEntityByNamedQuery(String query, Object... params)
{
		return CRUDDao.GetUniqueEntityByNamedQuery(query, params);
	}

	@Override
	public <T> List<T> GetListByNamedQuery(String query, Object... params) {
		return CRUDDao.GetListByNamedQuery(query, params);
	}

	@Override
	@Transactional(readOnly = true)
	public <T> Long getQueryCount(String query, Object... params) {
		return CRUDDao.getQueryCount(query, params);
	}

	@Override
	@Transactional(readOnly = true)
	public <T> T findByPrimaryKey(Class<T> klass, Serializable id) {
		return CRUDDao.findByPrimaryKey(klass, id);
	}

	@Override
	public <T> void Update(T klass) {
		CRUDDao.Update(klass);

	}

}
```

PAQUETE com.product.domain
CLASE APPMENU

```
package com.product.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

@Entity
@Table(name = "appmenus")
```

```
@NamedQueries({ @NamedQuery(name = "AppMenu.getAllMenus", query = "SELECT menus
FROM AppMenu  as menus  order by menuOrder") })
public class AppMenu {

	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="sec_appmenus")
	@SequenceGenerator(name="sec_appmenus", allocationSize=1,
sequenceName="sec_appmenus")
	@Column(name = "AppMenuID")
	private Long appMenuID;
	private String menuCaption;
	private Integer menuOrder;
	private Long parentAppMenuID;
	private Integer canAdd;
	private Integer canEdit;
	private Integer canDelete;
	private Integer canView;
	private String UniqueAppMenu;
	private String UniqueFileName;
	public Long getAppMenuID() {
		return appMenuID;
	}
	public void setAppMenuID(Long appMenuID) {
		this.appMenuID = appMenuID;
	}
	public String getMenuCaption() {
		return menuCaption;
	}
	public void setMenuCaption(String menuCaption) {
		this.menuCaption = menuCaption;
	}
	public Integer getMenuOrder() {
		return menuOrder;
	}
	public void setMenuOrder(Integer menuOrder) {
		this.menuOrder = menuOrder;
	}

	public Long getParentAppMenuID() {
		return parentAppMenuID;
	}
	public void setParentAppMenuID(Long parentAppMenuID) {
		this.parentAppMenuID = parentAppMenuID;
	}
	public Integer getCanAdd() {
		return canAdd;
	}
	public void setCanAdd(Integer canAdd) {
		this.canAdd = canAdd;
	}
	public Integer getCanEdit() {
```

```
		return canEdit;
	}
	public void setCanEdit(Integer canEdit) {
		this.canEdit = canEdit;
	}
	public Integer getCanDelete() {
		return canDelete;
	}
	public void setCanDelete(Integer canDelete) {
		this.canDelete = canDelete;
	}
	public Integer getCanView() {
		return canView;
	}
	public void setCanView(Integer canView) {
		this.canView = canView;
	}
	public String getUniqueAppMenu() {
		return UniqueAppMenu;
	}
	public void setUniqueAppMenu(String uniqueAppMenu) {
		UniqueAppMenu = uniqueAppMenu;
	}
	public String getUniqueFileName() {
		return UniqueFileName;
	}
	public void setUniqueFileName(String uniqueFileName) {
		UniqueFileName = uniqueFileName;
	}

}
```

PAQUETE com.product.domain
CLASE BASEENTIY

```
package com.product.domain;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Version;

@MappedSuperclass
public abstract class BaseEntity implements Serializable {

	private static final long serialVersionUID = 1L;
	@Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long ID;
@Version
private Long version;
private Long createdBy;
@Temporal(value = TemporalType.DATE)
private Date createdDate;
private Long updatedBy;
@Temporal(value = TemporalType.DATE)
private Date updatedDate;
private String deactivatedReason;
private String activatedReason;
private Integer active;

public long getID() {
	return ID;
}

public void setID(long iD) {
	ID = iD;
}

public Long getVersion() {
	return version;
}

public void setVersion(Long version) {
	this.version = version;
}

public Long getCreatedBy() {
	return createdBy;
}

public void setCreatedBy(Long createdBy) {
	this.createdBy = createdBy;
}

public Date getCreatedDate() {
	return createdDate;
}

public void setCreatedDate(Date createdDate) {
	this.createdDate = createdDate;
}

public Long getUpdatedBy() {
	return updatedBy;
}

public void setUpdatedBy(Long updatedBy) {
	this.updatedBy = updatedBy;
```

```
	}

	public Date getUpdatedDate() {
		return updatedDate;
	}

	public void setUpdatedDate(Date updatedDate) {
		this.updatedDate = updatedDate;
	}

	public String getDeactivatedReason() {
		return deactivatedReason;
	}

	public void setDeactivatedReason(String deactivatedReason) {
		this.deactivatedReason = deactivatedReason;
	}

	public String getActivatedReason() {
		return activatedReason;
	}

	public void setActivatedReason(String activatedReason) {
		this.activatedReason = activatedReason;
	}

	public Integer getActive() {
		return active;
	}

	public void setActive(Integer active) {
		this.active = active;
	}

}
```

PAQUETE com.product.domain
CLASE BASEENTIY GENERAL

```
package com.product.domain;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.MappedSuperclass;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Version;

@MappedSuperclass
public abstract class BaseEntityGeneral implements Serializable {

	private static final long serialVersionUID = 1L;
```

```
@Version
private Long version;
private Long createdBy;
@Temporal(value = TemporalType.DATE)
private Date createdDate;
private Long updatedBy;
@Temporal(value = TemporalType.DATE)
private Date updatedDate;
private Integer active;
private String deactivatedReason;
private String activatedReason;

public Long getVersion() {
	return version;
}

public void setVersion(Long version) {
	this.version = version;
}

public Long getCreatedBy() {
	return createdBy;
}

public void setCreatedBy(Long createdBy) {
	this.createdBy = createdBy;
}

public Date getCreatedDate() {
	return createdDate;
}

public void setCreatedDate(Date createdDate) {
	this.createdDate = createdDate;
}

public Long getUpdatedBy() {
	return updatedBy;
}

public void setUpdatedBy(Long updatedBy) {
	this.updatedBy = updatedBy;
}

public Date getUpdatedDate() {
	return updatedDate;
}

public void setUpdatedDate(Date updatedDate) {
	this.updatedDate = updatedDate;
}
```

```
	public Integer getActive() {
		return active;
	}

	public void setActive(Integer active) {
		this.active = active;
	}

	public String getDeactivatedReason() {
		return deactivatedReason;
	}

	public void setDeactivatedReason(String deactivatedReason) {
		this.deactivatedReason = deactivatedReason;
	}

	public String getActivatedReason() {
		return activatedReason;
	}

	public void setActivatedReason(String activatedReason) {
		this.activatedReason = activatedReason;
	}


}
```

PAQUETE com.product.domain
CLASE ROLES

```
package com.product.domain;

import javax.persistence.Entity;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import org.hibernate.validator.constraints.NotBlank;

@Entity
@Table(name = "roles")
@NamedQueries({ @NamedQuery(name = "Roles.getAllRoles", query = "SELECT roles
FROM Roles  as roles  order by roleName") })
public class Roles extends BaseEntity {

	private static final long serialVersionUID = 1L;
	@NotBlank(message = "Role Name cannot be empty")
	private String roleName;
	private String roleDesc;
	private Integer system;

	public String getRoleName() {
```

```
		return roleName;
	}

	public void setRoleName(String roleName) {
		this.roleName = roleName;
	}

	public String getRoleDesc() {
		return roleDesc;
	}

	public void setRoleDesc(String roleDesc) {
		this.roleDesc = roleDesc;
	}

	public Integer getSystem() {
		return system;
	}

	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public String toString() {
		return this.roleName;
	}

}
```

PAQUETE com.product.domain
CLASE ROLESMENU

```
package com.product.domain;

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
```

```
@Table(name = "rolesmenu")
@NamedQueries({ @NamedQuery(name = "RolesMenu.getAllMenusForRole", query =
"SELECT menus  FROM RolesMenu  as menus where menus.role.id = :arg1  ") })
public class RolesMenu {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)

	private long ID;
	private Long version;
	private long createdBy;
	@Temporal(value = TemporalType.DATE)
	private Date createdDate;
	private Long updatedBy;
	@Temporal(value = TemporalType.DATE)
	private Date updatedDate;

	@Column(name = "PrAppMenuID")
	private long prAppMenuID;

	@ManyToOne(fetch = FetchType.LAZY)
	@JoinColumn(name = "roleID")
	private Roles role;


	@Column(name = "canAdd")
	private Integer canAdd;

	@Column(name = "canEdit")
	private Integer canEdit;

	@Column(name = "canDelete")
	private Integer canDelete;

	@Column(name = "canView")
	private Integer canView;

	public long getID() {
		return ID;
	}

	public void setID(long iD) {
		ID = iD;
	}

	public Long getVersion() {
		return version;
	}

	public void setVersion(Long version) {
		this.version = version;
	}
```

```
public long getCreatedBy() {
	return createdBy;
}

public void setCreatedBy(long createdBy) {
	this.createdBy = createdBy;
}

public Date getCreatedDate() {
	return createdDate;
}

public void setCreatedDate(Date createdDate) {
	this.createdDate = createdDate;
}

public Long getUpdatedBy() {
	return updatedBy;
}

public void setUpdatedBy(Long updatedBy) {
	this.updatedBy = updatedBy;
}

public Date getUpdatedDate() {
	return updatedDate;
}

public void setUpdatedDate(Date updatedDate) {
	this.updatedDate = updatedDate;
}

public long getPrAppMenuID() {
	return prAppMenuID;
}

public void setPrAppMenuID(long prAppMenuID) {
	this.prAppMenuID = prAppMenuID;
}

public Integer getCanAdd() {
	if (this.canAdd == null)
		return 0;
	else
		return canAdd;
}

public void setCanAdd(Integer canAdd) {
	this.canAdd = canAdd;
}
```

```
	public Integer getCanEdit() {
		if (this.canEdit == null)
			return 0;
		else
			return canEdit;
	}

	public void setCanEdit(Integer canEdit) {
		this.canEdit = canEdit;
	}

	public Integer getCanDelete() {

		if (this.canDelete == null)
			return 0;
		else
			return canDelete;
	}

	public void setCanDelete(Integer canDelete) {
		this.canDelete = canDelete;
	}

	public Integer getCanView() {

		if (this.canView == null)
			return 0;
		else
			return canView;
	}

	public void setCanView(Integer canView) {
		this.canView = canView;
	}

	public Roles getRole() {
		return role;
	}

	public void setRole(Roles role) {
		this.role = role;
	}

}
```

PAQUETE com.product.domain
CLASE TCANTON

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1
```

```
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * TCanton generated by hbm2java
 */
@Entity
@Table(name = "t_canton", schema = "public")
@NamedQueries({

	@NamedQuery(name = "TCanton.getAllTCantonesUno", query = "SELECT cantones
FROM TCanton  as cantones  inner JOIN fetch cantones.provCanton as provcanton
WHERE cantones.provCanton.idprovincia = :arg1"),
	@NamedQuery(name = "TCanton.getAllTCantones", query = "SELECT cantones
FROM TCanton  as cantones  inner JOIN fetch cantones.provCanton as provcanton
order by nombrecanton") })
public class TCanton extends BaseEntityGeneral{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idcanton;


	@ManyToOne(targetEntity = TProvincia.class, fetch = FetchType.LAZY,
cascade = CascadeType.ALL, optional=false )
	@JoinColumn(name = "idprovincia")
	private TProvincia provCanton;

	@Column(name = "nombrecanton", length = 45)
	private String nombrecanton;
	private Integer system;


	public int getIdcanton() {
		return idcanton;
	}
```

```
	public void setIdcanton(int idcanton) {
		this.idcanton = idcanton;
	}


	public TProvincia getProvCanton() {
		return provCanton;
	}

	public void setProvCanton(TProvincia provCanton) {
		this.provCanton = provCanton;
	}

	public String getNombrecanton() {
		return nombrecanton;
	}

	public void setNombrecanton(String nombrecanton) {
		this.nombrecanton = nombrecanton;
	}


	public Integer getSystem() {
		return system;
	}

	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public String toString() {
		return this.nombrecanton;
	}

}
```

PAQUETE com.product.domain
CLASE TDETALLEHISTORIACLINICA

```
package com.product.domain;
```

```
// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.Valid;

/**
 * TDetalleHistoriaClinica generated by hbm2java
 */
@Entity
@Table(name = "t_detalle_historia_clinica", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TDetalleHistoriaClinica.getAllTDetalleUno", query =
"SELECT cantones  FROM TCanton  as cantones  inner JOIN fetch
cantones.provCanton as provcanton WHERE cantones.provCanton = :arg1"),
	@NamedQuery(name = "TDetalleHistoriaClinica.getAllTDetalle", query =
"SELECT detalle  FROM TDetalleHistoriaClinica  as detalle  inner JOIN fetch
detalle.THistoriaClinica as THistoriaClinica inner JOIN fetch detalle.medico as
medico inner JOIN fetch detalle.persona as persona order by iddetallehc desc")
})

public class TDetalleHistoriaClinica extends BaseEntityGeneral implements
Cloneable {

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="sec_detalle_historia_clinica")
	@SequenceGenerator(name="sec_detalle_historia_clinica", allocationSize=1,
sequenceName="sec_detalle_historia_clinica")
	private int iddetallehc;


	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "numerohistoria")
	private THistoriaClinica THistoriaClinica;
```

```
@ManyToOne(fetch = FetchType.LAZY, optional=false)
@JoinColumn(name = "idmedico")
private TMedico medico;

@Valid
@ManyToOne(fetch = FetchType.LAZY, optional=false)
@JoinColumn(name = "persona")
private TPersona persona;
@Temporal(TemporalType.DATE)
private Date fecharegistro;
private String motivoconsulta;
private Integer system;



public int getIddetallehc() {
	return iddetallehc;
}
public void setIddetallehc(int iddetallehc) {
	this.iddetallehc = iddetallehc;
}


public THistoriaClinica getTHistoriaClinica() {
	return THistoriaClinica;
}
public void setTHistoriaClinica(THistoriaClinica tHistoriaClinica) {
	THistoriaClinica = tHistoriaClinica;
}
public TMedico getMedico() {
	return medico;
}
public void setMedico(TMedico medico) {
	this.medico = medico;
}
public TPersona getPersona() {
	return persona;
}
public void setPersona(TPersona persona) {
	this.persona = persona;
}

public Date getFecharegistro() {
	return fecharegistro;
}
public void setFecharegistro(Date fecharegistro) {
	this.fecharegistro = fecharegistro;
}
public String getMotivoconsulta() {
	return motivoconsulta;
```

```
	}
	public void setMotivoconsulta(String motivoconsulta) {
		this.motivoconsulta = motivoconsulta;
	}
	public Integer getSystem() {
		return system;
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public Object clone() throws CloneNotSupportedException {
		return super.clone();
	}

	@Override
	public String toString()
	{
		return this.getMedico().getPersona().getCedulapersona();
	}

}
```

PAQUETE com.product.domain
CLASE TDETALLERECETA

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
```

```
import javax.persistence.NamedQuery;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

/**
 * TDetalleReceta generated by hbm2java
 */
@Entity
@Table(name = "t_detalle_receta", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TMedico.getAllTMedicosUno", query = "SELECT medicos
FROM TMedico as medicos  inner JOIN fetch medicos.persona as persona WHERE
medicos.persona.nombrespersona = :arg1 and medicos.active = 1"),
	@NamedQuery(name = "TDetalleReceta.getAllTDetalleReceta", query = "SELECT
detallereceta  FROM TDetalleReceta  as detallereceta  inner JOIN fetch
detallereceta.TRecetas as TRecetas  inner JOIN fetch detallereceta.TMedicamento
as TMedicamento order by iddetallereceta desc") })

public class TDetalleReceta extends BaseEntityGeneral implements Cloneable{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="sec_detalle_receta")
	@SequenceGenerator(name="sec_detalle_receta", allocationSize=1,
sequenceName="sec_detalle_receta")
	private int iddetallereceta;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idreceta")
	private TRecetas TRecetas;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idmedicamento")
	private TMedicamento TMedicamento;


	private String observaciones;
	private Integer system;

	public int getIddetallereceta() {
		return iddetallereceta;
	}
	public void setIddetallereceta(int iddetallereceta) {
		this.iddetallereceta = iddetallereceta;
	}
	public TRecetas getTRecetas() {
```

```
		return TRecetas;
	}
	public void setTRecetas(TRecetas tRecetas) {
		TRecetas = tRecetas;
	}

	public TMedicamento getTMedicamento() {
		return TMedicamento;
	}
	public void setTMedicamento(TMedicamento tMedicamento) {
		TMedicamento = tMedicamento;
	}
	public String getObservaciones() {
		return observaciones;
	}
	public void setObservaciones(String observaciones) {
		this.observaciones = observaciones;
	}
	public Integer getSystem() {
		return system;
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub

	}
	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public Object clone() throws CloneNotSupportedException {
		return super.clone();
	}
}
```

PAQUETE com.product.domain
CLASE TDIAGMEDICO

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.validation.Valid;

/**
 * TPreConsulta generated by hbm2java
 */
@Entity
@Table(name = "t_diag_medico", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TCanton.getAllTCantonesUno", query = "SELECT
cantones  FROM TCanton  as cantones  inner JOIN fetch cantones.provCanton as
provcanton WHERE cantones.provCanton = :arg1"),
	@NamedQuery(name = "TDiagMedico.getAllTDiagMedico", query = "SELECT
diagmedico  FROM TDiagMedico  as diagmedico inner JOIN fetch
diagmedico.detalleHC as detalleHC inner JOIN fetch diagmedico.TEnfermedad  as
enfermedad order by iddiagnostico") })

public class TDiagMedico extends BaseEntityGeneral implements Cloneable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="sec_diag_medico")
	@SequenceGenerator(name="sec_diag_medico", allocationSize=1,
sequenceName="sec_diag_medico")
	private int iddiagnostico;

	@Valid
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "iddetallehc")
	private TDetalleHistoriaClinica detalleHC;

	@Valid
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idenfermedad")
	private TEnfermedad TEnfermedad;

	private String observaciones;
	private Integer system;



	public TDetalleHistoriaClinica getDetalleHC() {
		return detalleHC;
```

```
	}
	public void setDetalleHC(TDetalleHistoriaClinica detalleHC) {
		this.detalleHC = detalleHC;
	}

	public int getIddiagnostico() {
		return iddiagnostico;
	}
	public void setIddiagnostico(int iddiagnostico) {
		this.iddiagnostico = iddiagnostico;
	}

	public TEnfermedad getTEnfermedad() {
		return TEnfermedad;
	}
	public void setTEnfermedad(TEnfermedad tEnfermedad) {
		TEnfermedad = tEnfermedad;
	}
	public String getObservaciones() {
		return observaciones;
	}
	public void setObservaciones(String observaciones) {
		this.observaciones = observaciones;
	}
	public Integer getSystem() {
		return system;
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public Object clone() throws CloneNotSupportedException {
		return super.clone();
	}
}
```

PAQUETE com.product.domain
CLASE TENFERMEDAD

```
package com.product.domain;
```

```
// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * TExamen generated by hbm2java
 */
@Entity
@Table(name = "t_enfermedad", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TMedico.getAllTMedicosUno", query = "SELECT medicos
FROM TMedico as medicos  inner JOIN fetch medicos.persona as persona WHERE
medicos.persona.nombrespersona = :arg1 and medicos.active = 1"),
	@NamedQuery(name = "TEnfermedad.getAllTEnfermedad", query = "SELECT
enfermedad  FROM TEnfermedad  as enfermedad order by idenfermedad asc") })

public class TEnfermedad extends BaseEntityGeneral {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idenfermedad;

	private String nombreenfermedad;
	private Integer system;

	public int getIdenfermedad() {
		return idenfermedad;
	}
	public void setIdenfermedad(int idenfermedad) {
		this.idenfermedad = idenfermedad;
	}
	public String getNombreenfermedad() {
		return nombreenfermedad;
	}
	public void setNombreenfermedad(String nombreenfermedad) {
		this.nombreenfermedad = nombreenfermedad;
	}
	public Integer getSystem() {
		return system;
```

```
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub
	}
	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public String toString()
	{
		return this.nombreenfermedad;
	}

}
```

PAQUETE com.product.domain
CLASE TESPECIALIDAD

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * TEspecialidad generated by hbm2java
 */
@Entity
@Table(name = "t_especialidad", schema = "public")
@NamedQueries({ @NamedQuery(name = "TEspecialidad.getAllTEspecialidad", query =
"SELECT cust  FROM TEspecialidad  as cust order by nomEspecialidad") })
public class TEspecialidad extends BaseEntityGeneral {



	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int idespecialidad;

@Column(name = "nom_especialidad", length = 45)
private String nomEspecialidad;

private Integer system;


public int getIdespecialidad() {
	return idespecialidad;
}

public void setIdespecialidad(int idespecialidad) {
	this.idespecialidad = idespecialidad;
}

public String getNomEspecialidad() {
	return nomEspecialidad;
}

public void setNomEspecialidad(String nomEspecialidad) {
	this.nomEspecialidad = nomEspecialidad;
}

public Integer getSystem() {
	return system;
}

public void setSystem(Integer system) {
	this.system = system;
}

@Override
public void setDeactivatedReason(String reason) {
	// TODO Auto-generated method stub

}

@Override
public void setActivatedReason(String reason) {
	// TODO Auto-generated method stub

}

@Override
public String toString() {
	return this.nomEspecialidad;
}
```

```
}
```

PAQUETE com.product.domain
CLASE TEXAMEN

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

/**
 * TExamen generated by hbm2java
 */
@Entity
@Table(name = "t_examen", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TMedico.getAllTMedicosUno", query = "SELECT medicos
FROM TMedico as medicos  inner JOIN fetch medicos.persona as persona WHERE
medicos.persona.nombrespersona = :arg1 and medicos.active = 1"),
	@NamedQuery(name = "TExamen.getAllTExamen", query = "SELECT examen  FROM
TExamen  as examen  inner JOIN fetch examen.detalleHC as detalleHC order by
idexamen asc") })

public class TExamen extends BaseEntityGeneral {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;


	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="sec_examen")
```

```
	@SequenceGenerator(name="sec_examen", allocationSize=1,
sequenceName="sec_examen")
	private int idexamen;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "iddetallehc")
	private TDetalleHistoriaClinica detalleHC;


	@OneToMany(fetch = FetchType.EAGER, mappedBy = "TExamen",cascade =
CascadeType.ALL, orphanRemoval=true)
	private List<TResultadoExamen> TResultadoExamens;


	public List<TResultadoExamen> getTResultadoExamens() {
		return TResultadoExamens;
	}
	public void setTResultadoExamens(List<TResultadoExamen>
tResultadoExamens) {
		TResultadoExamens = tResultadoExamens;
	}

	@Temporal(TemporalType.DATE)
	private Date fechasolicitudexamen;

	private String observacionesexamen;
	private Integer system;




	public int getIdexamen() {
		return idexamen;
	}
	public void setIdexamen(int idexamen) {
		this.idexamen = idexamen;
	}

	public TDetalleHistoriaClinica getDetalleHC() {
		return detalleHC;
	}
	public void setDetalleHC(TDetalleHistoriaClinica detalleHC) {
		this.detalleHC = detalleHC;
	}
	public Date getFechasolicitudexamen() {
		return fechasolicitudexamen;
	}
	public void setFechasolicitudexamen(Date fechasolicitudexamen) {
		this.fechasolicitudexamen = fechasolicitudexamen;
	}
	public String getObservacionesexamen() {
		return observacionesexamen;
```

```
	}
	public void setObservacionesexamen(String observacionesexamen) {
		this.observacionesexamen = observacionesexamen;
	}
	public Integer getSystem() {
		return system;
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub
	}
	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub
	}


	@Override
	public Object clone() throws CloneNotSupportedException {
		return super.clone();
	}

}
```

PAQUETE com.product.domain
CLASE THISTORIACLINICA

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
```

```
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.Valid;


/**
 * THistoriaClinica generated by hbm2java
 */
@Entity
@Table(name = "t_historia_clinica", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "THistoriaClinica.getAllTHistoriasCUno", query =
"SELECT hc  FROM THistoriaClinica  as hc  inner JOIN fetch hc.persona as persona
WHERE hc.persona = :arg1"),
	@NamedQuery(name = "THistoriaClinica.getAllTHistoriasC", query = "SELECT
hc  FROM THistoriaClinica  as hc  inner JOIN fetch hc.persona as persona  order
by persona.nombrespersona") })
public class THistoriaClinica extends BaseEntityGeneral implements Cloneable  {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
//	@Id
//	@GeneratedValue(strategy = GenerationType.IDENTITY)

	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="sec_historia_clinica")
	@SequenceGenerator(name="sec_historia_clinica", allocationSize=1,
sequenceName="sec_historia_clinica")
	private int numerohistoria;

	@Valid
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idpersona")
	private TPersona persona;

	@Temporal(TemporalType.DATE)
	@Column(name = "fecharegistro")
	private Date fecharegistro;

	private Integer system;


	@OneToMany(fetch = FetchType.EAGER, mappedBy = "THistoriaClinica",cascade
= CascadeType.ALL, orphanRemoval=true)
	private List<TDetalleHistoriaClinica> TDetalleHistoriaClinicas;


	public List<TDetalleHistoriaClinica> getTDetalleHistoriaClinicas() {
```

```
            return TDetalleHistoriaClinicas;
      }

      public void setTDetalleHistoriaClinicas(List<TDetalleHistoriaClinica>
tDetalleHistoriaClinicas) {
            TDetalleHistoriaClinicas = tDetalleHistoriaClinicas;
      }

      public int getNumerohistoria() {
            return numerohistoria;
      }

      public void setNumerohistoria(int numerohistoria) {
            this.numerohistoria = numerohistoria;
      }

      public TPersona getPersona() {
            return persona;
      }

      public void setPersona(TPersona persona) {
            this.persona = persona;
      }

      public Date getFecharegistro() {
            return fecharegistro;
      }

      public void setFecharegistro(Date fecharegistro) {
            this.fecharegistro = fecharegistro;
      }

      public Integer getSystem() {
            return system;
      }

      public void setSystem(Integer system) {
            this.system = system;
      }

      @Override
      public void setDeactivatedReason(String reason) {
            // TODO Auto-generated method stub

      }

      @Override
      public void setActivatedReason(String reason) {
            // TODO Auto-generated method stub

      }
```

```
	@Override
	public Object clone() throws CloneNotSupportedException {
		return super.clone();
	}

}
```

PAQUETE com.product.domain
CLASE THORAS

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * THoras generated by hbm2java
 */
@Entity
@Table(name = "t_horas", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TMedico.getAllTMedicosUno", query = "SELECT medicos
FROM TMedico as medicos  inner JOIN fetch medicos.persona as persona WHERE
medicos.persona.nombrespersona = :arg1 and medicos.active = 1"),
	@NamedQuery(name = "THoras.getAllTHoras", query = "SELECT horas  FROM
THoras  as horas order by horas.idhora") })
public class THoras implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idhora;
	private String horas;


	public THoras() {
	}

	public THoras(int idhora) {
```

```
		this.idhora = idhora;
	}

	public THoras(int idhora, String horas) {
		this.idhora = idhora;
		this.horas = horas;
	}

	@Id
	@Column(name = "idhora", unique = true, nullable = false)
	public int getIdhora() {
		return this.idhora;
	}

	public void setIdhora(int idhora) {
		this.idhora = idhora;
	}

	@Column(name = "horas", length = 12)
	public String getHoras() {
		return this.horas;
	}

	public void setHoras(String horas) {
		this.horas = horas;
	}


	@Override
	public String toString() {
		return this.horas;
	}
}
```

PAQUETE com.product.domain
CLASE TMEDICAMENTO

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
```

```
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.Valid;

/**
 * TMedicamento generated by hbm2java
 */
@Entity
@Table(name = "t_medicamento", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TMedicamento.getAllTMedicamentosUno", query =
"SELECT medicamento  FROM TMedicamento  as medicamento  inner JOIN fetch
medicamento.tipomedicamento as tmedicamento WHERE medicamento.nombremedicamento
= :arg1 and medicamento.active = 1"),
	@NamedQuery(name = "TMedicamento.getAllTMedicamentos", query = "SELECT
medicamento  FROM TMedicamento  as medicamento  inner JOIN fetch
medicamento.tipomedicamento as tmedicamento  order by nombremedicamento") })
public class TMedicamento extends BaseEntityGeneral {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idmedicamento;

	@Valid
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idtipomedicamento")
	private TTipoMedicamentos tipomedicamento;

	@Column(name = "nombremedicamento", length = 50)
	private String nombremedicamento;

	private Integer system;

	public int getIdmedicamento() {
		return idmedicamento;
	}

	public void setIdmedicamento(int idmedicamento) {
		this.idmedicamento = idmedicamento;
	}

	public TTipoMedicamentos getTipomedicamento() {
		return tipomedicamento;
	}

	public void setTipomedicamento(TTipoMedicamentos tipomedicamento) {
```

```
			this.tipomedicamento = tipomedicamento;
		}

		public String getNombremedicamento() {
			return nombremedicamento;
		}

		public void setNombremedicamento(String nombremedicamento) {
			this.nombremedicamento = nombremedicamento;
		}

		public Integer getSystem() {
			return system;
		}

		public void setSystem(Integer system) {
			this.system = system;
		}

		@Override
		public void setDeactivatedReason(String reason) {
			// TODO Auto-generated method stub

		}

		@Override
		public void setActivatedReason(String reason) {
			// TODO Auto-generated method stub

		}

		@Override
		public String toString()
		{
			return this.getNombremedicamento();
		}

}
```

PAQUETE com.product.domain
CLASE TMEDICO

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * TMedico generated by hbm2java
 */
@Entity
@Table(name = "t_medico", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TMedico.getAllTMedicosUno", query = "SELECT medicos
FROM TMedico as medicos  inner JOIN fetch medicos.persona as persona WHERE
medicos.persona.nombrespersona = :arg1 and medicos.active = 1"),
	@NamedQuery(name = "TMedico.getAllTMedicos", query = "SELECT medicos
FROM TMedico  as medicos  inner JOIN fetch medicos.persona as persona  inner
JOIN fetch medicos.especialidad as especialidad order by
persona.nombrespersona") })

public class TMedico extends BaseEntityGeneral {



	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idmedico;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idpersona")
	private TPersona persona;
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idespecialidad")
	private TEspecialidad especialidad;

	@Column(name = "estadomedico", length = 20)
	private String estadomedico;

	private Integer system;

	public TMedico() {
	}

	public TMedico(int idmedico) {
```

```
		this.idmedico = idmedico;
	}

	public int getIdmedico() {
		return this.idmedico;
	}

	public void setIdmedico(int idmedico) {
		this.idmedico = idmedico;
	}

	public TPersona getPersona() {
		return persona;
	}

	public void setPersona(TPersona persona) {
		this.persona = persona;
	}

	public TEspecialidad getEspecialidad() {
		return especialidad;
	}

	public void setEspecialidad(TEspecialidad especialidad) {
		this.especialidad = especialidad;
	}

	public String getEstadomedico() {
		return this.estadomedico;
	}

	public void setEstadomedico(String estadomedico) {
		this.estadomedico = estadomedico;
	}


	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	public Integer getSystem() {
		return system;
	}
```

```
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public String toString() {
		return this.persona.getCedulapersona();//getNombrespersona();

	}

}
```

PAQUETE com.product.domain
CLASE TMEDICOCITAS

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * TMedicosCitas generated by hbm2java
 */
@Entity
@Table(name = "t_medicos_citas", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TMedico.getAllTMedicosUno", query = "SELECT medicos
FROM TMedico as medicos  inner JOIN fetch medicos.persona as persona WHERE
medicos.persona.nombrespersona = :arg1 and medicos.active = 1"),
	  @NamedQuery(name = "TMedicosCitas.getAllTMedicosCitas", query = "SELECT
medicosCitas  FROM TMedicosCitas  as medicosCitas  inner JOIN fetch
medicosCitas.turno as turno  inner JOIN fetch medicosCitas.persona as persona
inner JOIN fetch medicosCitas.medico as medico inner JOIN fetch
medicosCitas.especialidad as especialidad order by persona.nombrespersona") })
public class TMedicosCitas extends BaseEntityGeneral {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
```

```
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idmedicocita;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idturno")
	private TTurno turno;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idpersona")
	private TPersona persona;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "medico")
	private TMedico medico;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "especialidad")
	private TEspecialidad especialidad;

//	private String horacita;
	private String estadomedicocita;
//	@Temporal(TemporalType.DATE)
//	private Date fechacita;
	private Integer system;
	public int getIdmedicocita() {
		return idmedicocita;
	}
	public void setIdmedicocita(int idmedicocita) {
		this.idmedicocita = idmedicocita;
	}
	public TTurno getTurno() {
		return turno;
	}
	public void setTurno(TTurno turno) {
		this.turno = turno;
	}


	public TPersona getPersona() {
		return persona;
	}
	public void setPersona(TPersona persona) {
		this.persona = persona;
	}


	public TMedico getMedico() {
		return medico;
	}
	public void setMedico(TMedico medico) {
		this.medico = medico;
```

```
	}
//	public String getHoracita() {
//		return horacita;
//	}
//	public void setHoracita(String horacita) {
//		this.horacita = horacita;
//	}

	public TEspecialidad getEspecialidad() {
		return especialidad;
	}
	public void setEspecialidad(TEspecialidad especialidad) {
		this.especialidad = especialidad;
	}
	public String getEstadomedicocita() {
		return estadomedicocita;
	}
	public void setEstadomedicocita(String estadomedicocita) {
		this.estadomedicocita = estadomedicocita;
	}
	public Integer getSystem() {
		return system;
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}


}
```

PAQUETE com.product.domain
CLASE TPARROQUIA

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.Valid;

/**
 * TParroquia generated by hbm2java
 */
@Entity
@Table(name = "t_parroquia", schema = "public")
@NamedQueries({

	@NamedQuery(name = "TParroquia.getAllTParroquiasUno", query = "SELECT
parroquias  FROM TParroquia  as parroquias  inner JOIN fetch
parroquias.cantonParroquia as cantonParroquia WHERE parroquias.nombreparroquia =
:arg1 and parroquias.active = 1"),
	@NamedQuery(name = "TParroquia.getAllTParroquias", query = "SELECT
parroquias  FROM TParroquia  as parroquias  inner JOIN fetch
parroquias.cantonParroquia as cantonParroquia  order by nombreparroquia") })
public class TParroquia extends BaseEntityGeneral {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idparroquia;

	@Valid
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idcanton")
	private TCanton cantonParroquia;

	@Column(name = "nombreparroquia", length = 45)
	private String nombreparroquia;

	private Integer system;

	public int getIdparroquia() {
		return idparroquia;
	}

	public void setIdparroquia(int idparroquia) {
		this.idparroquia = idparroquia;
	}

	public TCanton getCantonParroquia() {
```

```
		return cantonParroquia;
	}

	public void setCantonParroquia(TCanton cantonParroquia) {
		this.cantonParroquia = cantonParroquia;
	}

	public String getNombreparroquia() {
		return nombreparroquia;
	}

	public void setNombreparroquia(String nombreparroquia) {
		this.nombreparroquia = nombreparroquia;
	}

	public Integer getSystem() {
		return system;
	}

	public void setSystem(Integer system) {
		this.system = system;
	}
	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stu
	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public String toString() {
		return this.nombreparroquia;
	}

}
```

PAQUETE com.product.domain
CLASE TPERSONA

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

/**
 * TPersona generated by hbm2java
 */
@Entity
@Table(name = "t_persona", schema = "public")
@NamedQueries({

	@NamedQuery(name = "TPersona.getAllTPersonasUno", query = "SELECT
personas  FROM TPersona  as personas  inner JOIN fetch personas.parroquias as
parroquias WHERE personas.nombrespersona = :arg1 and personas.active = 1"),
	@NamedQuery(name = "TPersona.getAllTPersonas", query = "SELECT personas
FROM TPersona  as personas  inner JOIN fetch personas.parroquias as parroquias
inner JOIN fetch personas.provincia as provincia inner JOIN fetch
personas.canton as canton  order by nombrespersona") })
public class TPersona extends BaseEntityGeneral implements Cloneable {


	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idpersona;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "idparroquia")
	private TParroquia parroquias;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "provincia")
	private TProvincia provincia;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "canton")
	private TCanton canton;


	private String cedulapersona;
```

```
private String nombrespersona;
private String sexopersona;
@Temporal(TemporalType.DATE)
private Date fechanacimiento;
private String estadocivil;
private String direccionpersona;
private String correopersona;
private String telefonopersona;
private String celularpersona;

@Lob
@Column(name = "fotopaciente")
private byte[] fotopaciente;


private Integer system;

public String getCedulapersona() {
	return cedulapersona;
}
public void setCedulapersona(String cedulapersona) {
	this.cedulapersona = cedulapersona;
}

public TParroquia getParroquias() {
	return parroquias;
}
public void setParroquias(TParroquia parroquias) {
	this.parroquias = parroquias;
}
public String getNombrespersona() {
	return nombrespersona;
}
public void setNombrespersona(String nombrespersona) {
	this.nombrespersona = nombrespersona;
}
public String getSexopersona() {
	return sexopersona;
}
public void setSexopersona(String sexopersona) {
	this.sexopersona = sexopersona;
}
public Date getFechanacimiento() {
	return fechanacimiento;
}
public void setFechanacimiento(Date fechanacimiento) {
	this.fechanacimiento = fechanacimiento;
}
public String getEstadocivil() {
	return estadocivil;
}
public void setEstadocivil(String estadocivil) {
```

```
		this.estadocivil = estadocivil;
	}
	public String getDireccionpersona() {
		return direccionpersona;
	}
	public void setDireccionpersona(String direccionpersona) {
		this.direccionpersona = direccionpersona;
	}
	public String getCorreopersona() {
		return correopersona;
	}
	public void setCorreopersona(String correopersona) {
		this.correopersona = correopersona;
	}
	public String getTelefonopersona() {
		return telefonopersona;
	}
	public void setTelefonopersona(String telefonopersona) {
		this.telefonopersona = telefonopersona;
	}
	public String getCelularpersona() {
		return celularpersona;
	}
	public void setCelularpersona(String celularpersona) {
		this.celularpersona = celularpersona;
	}

	public byte[] getFotopaciente() {
		return fotopaciente;
	}
	public void setFotopaciente(byte[] fotopaciente) {
		this.fotopaciente = fotopaciente;
	}
	public Integer getSystem() {
		return system;
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}
```

```
	public int getIdpersona() {
		return idpersona;
	}
	public void setIdpersona(int idpersona) {
		this.idpersona = idpersona;
	}
	public TProvincia getProvincia() {
		return provincia;
	}
	public void setProvincia(TProvincia provincia) {
		this.provincia = provincia;
	}
	public TCanton getCanton() {
		return canton;
	}
	public void setCanton(TCanton canton) {
		this.canton = canton;
	}
	@Override
	public String toString() {
		return this.nombrespersona;//.cedulapersona;
	}
}
```

PAQUETE com.product.domain
CLASE TPRECONSULTA

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.Valid;

/**
 * TPreConsulta generated by hbm2java
 */
@Entity
@Table(name = "t_pre_consulta", schema = "public")
```

```
@NamedQueries({

	//@NamedQuery(name = "TCanton.getAllTCantonesUno", query = "SELECT
cantones  FROM TCanton  as cantones  inner JOIN fetch cantones.provCanton as
provcanton WHERE cantones.provCanton = :arg1"),
	@NamedQuery(name = "TPreConsulta.getAllTPreConsulta", query = "SELECT
preconsulta  FROM TPreConsulta  as preconsulta  inner JOIN fetch
preconsulta.detalleHC as detalleHC") })

public class TPreConsulta extends BaseEntityGeneral implements Cloneable {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idpreconsulta;

	@Valid
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "iddetallehc")
	private TDetalleHistoriaClinica detalleHC;

	private String precionpaciente;
	private String tallapaciente;
	private String pesopaciente;
	private String temperaturapaciente;
	@Temporal(TemporalType.DATE)
	private Date fechapreconsulta;
	private Integer system;


	public int getIdpreconsulta() {
		return idpreconsulta;
	}
	public void setIdpreconsulta(int idpreconsulta) {
		this.idpreconsulta = idpreconsulta;
	}
	public TDetalleHistoriaClinica getDetalleHC() {
		return detalleHC;
	}
	public void setDetalleHC(TDetalleHistoriaClinica detalleHC) {
		this.detalleHC = detalleHC;
	}
	public String getPrecionpaciente() {
		return precionpaciente;
	}
	public void setPrecionpaciente(String precionpaciente) {
		this.precionpaciente = precionpaciente;
	}
```

```
	public String getTallapaciente() {
		return tallapaciente;
	}
	public void setTallapaciente(String tallapaciente) {
		this.tallapaciente = tallapaciente;
	}
	public String getPesopaciente() {
		return pesopaciente;
	}
	public void setPesopaciente(String pesopaciente) {
		this.pesopaciente = pesopaciente;
	}
	public String getTemperaturapaciente() {
		return temperaturapaciente;
	}
	public void setTemperaturapaciente(String temperaturapaciente) {
		this.temperaturapaciente = temperaturapaciente;
	}
	public Date getFechapreconsulta() {
		return fechapreconsulta;
	}
	public void setFechapreconsulta(Date fechapreconsulta) {
		this.fechapreconsulta = fechapreconsulta;
	}
	public Integer getSystem() {
		return system;
	}
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public Object clone() throws CloneNotSupportedException {
		return super.clone();
	}
}
```

PAQUETE com.product.domain
CLASE TPROVINCIA

```
package com.product.domain;
```

```
// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * TProvincia generated by hbm2java
 */
@Entity
@Table(name = "t_provincia", schema = "public")
@NamedQueries({ @NamedQuery(name = "TProvincia.getAllTProvincias", query =
"SELECT provincias  FROM TProvincia  as provincias order by nombreprovincia") })
public class TProvincia extends BaseEntityGeneral {


	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "idprovincia")
	private Integer idprovincia;

	@Column(name = "nombreprovincia", length = 45)
	private String nombreprovincia;
	private Integer system;



	public int getIdprovincia() {
		return idprovincia;
	}

	public void setIdprovincia(int idprovincia) {
		this.idprovincia = idprovincia;
	}

	public String getNombreprovincia() {
		return nombreprovincia;
	}

	public void setNombreprovincia(String nombreprovincia) {
		this.nombreprovincia = nombreprovincia;
```

```
	}

	public Integer getSystem() {
		return system;
	}

	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public String toString() {
		return this.nombreprovincia;
	}

}
```

PAQUETE com.product.domain
CLASE TRECETAS

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.SequenceGenerator;
```

```
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
/**
 * TRecetas generated by hbm2java
 */
@Entity
@Table(name = "t_recetas", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TRecetas.getAllTRecetasUno", query = "SELECT turnos
FROM TTurno  as turnos  inner JOIN fetch turnos.medico as medico WHERE
turnos.descripcionturno = :arg1 and turnos.active = 1"),
	@NamedQuery(name = "TRecetas.getAllTRecetas", query = "SELECT recetas
FROM TRecetas  as recetas  inner JOIN fetch recetas.TDetalleHistoriaClinica as
TDetalleHistoriaClinica") })

public class TRecetas extends BaseEntityGeneral implements Cloneable{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="sec_recetas")
	@SequenceGenerator(name="sec_recetas", allocationSize=1,
sequenceName="sec_recetas")
	private int idreceta;

	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "iddetallehc")
	private TDetalleHistoriaClinica TDetalleHistoriaClinica;


	@OneToMany(fetch = FetchType.EAGER, mappedBy = "TRecetas",cascade =
CascadeType.ALL, orphanRemoval=true)
	private List<TDetalleReceta> TDetalleRecetas;


	@Temporal(TemporalType.DATE)
	@Column(name = "fecha")
	private Date fecha;

	private Integer edad;
	private Integer system;



	public List<TDetalleReceta> getTDetalleRecetas() {
		return TDetalleRecetas;
```

```
}
public void setTDetalleRecetas(List<TDetalleReceta> tDetalleRecetas) {
	TDetalleRecetas = tDetalleRecetas;
}
public int getIdreceta() {
	return idreceta;
}
public void setIdreceta(int idreceta) {
	this.idreceta = idreceta;
}

public TDetalleHistoriaClinica getTDetalleHistoriaClinica() {
	return TDetalleHistoriaClinica;
}
public void setTDetalleHistoriaClinica(
		TDetalleHistoriaClinica tDetalleHistoriaClinica) {
	TDetalleHistoriaClinica = tDetalleHistoriaClinica;
}
public Date getFecha() {
	return fecha;
}
public void setFecha(Date fecha) {
	this.fecha = fecha;
}
public Integer getEdad() {
	return edad;
}
public void setEdad(Integer edad) {
	this.edad = edad;
}
public Integer getSystem() {
	return system;
}
public void setSystem(Integer system) {
	this.system = system;
}

@Override
public void setDeactivatedReason(String reason) {
	// TODO Auto-generated method stub

}

@Override
public void setActivatedReason(String reason) {
	// TODO Auto-generated method stub

}

@Override
public Object clone() throws CloneNotSupportedException {
	return super.clone();
```

```
	}

}
```

PAQUETE com.product.domain
CLASE TRESULTADOEXAMEN

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.Valid;


/**
 * TResultadoExamen generated by hbm2java
 */
@Entity
@Table(name = "t_resultado_examen", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TCanton.getAllTCantonesUno", query = "SELECT
cantones  FROM TCanton  as cantones  inner JOIN fetch cantones.provCanton as
provcanton WHERE cantones.provCanton = :arg1"),
	@NamedQuery(name = "TResultadoExamen.getAllTResultadoExamen", query =
"SELECT resultado  FROM TResultadoExamen  as resultado  inner JOIN fetch
resultado.TExamen as TExamen inner JOIN fetch resultado.tipoexamen as
tipoexamen") })
public class TResultadoExamen extends BaseEntityGeneral implements Cloneable{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int idresultadoexamen;
```

```
@Valid
@ManyToOne(fetch = FetchType.LAZY, optional=false)
@JoinColumn(name = "idexamen")
private TExamen TExamen;

@Valid
@ManyToOne(fetch = FetchType.LAZY, optional=false)
@JoinColumn(name = "idtipoexamen")
private TTipoExamen tipoexamen;
@Temporal(TemporalType.DATE)
private Date fechaentrega;
private String descripcion;
private String descripcionimagen;
private Integer system;



public String getDescripcionimagen() {
	return descripcionimagen;
}
public void setDescripcionimagen(String descripcionimagen) {
	this.descripcionimagen = descripcionimagen;
}
public int getIdresultadoexamen() {
	return idresultadoexamen;
}
public void setIdresultadoexamen(int idresultadoexamen) {
	this.idresultadoexamen = idresultadoexamen;
}
public TExamen getTExamen() {
	return TExamen;
}
public void setTExamen(TExamen tExamen) {
	TExamen = tExamen;
}
public Date getFechaentrega() {
	return fechaentrega;
}
public void setFechaentrega(Date fechaentrega) {
	this.fechaentrega = fechaentrega;
}
public String getDescripcion() {
	return descripcion;
}
public void setDescripcion(String descripcion) {
	this.descripcion = descripcion;
}

public TTipoExamen getTipoexamen() {
	return tipoexamen;
}
}
```

```
	public void setTipoexamen(TTipoExamen tipoexamen) {
		this.tipoexamen = tipoexamen;
	}
	public Integer getSystem() {
		return system;
	}

	public void setSystem(Integer system) {
		this.system = system;
	}
	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public String toString()
	{
		return this.getTipoexamen().getNombreexamen();
	}


	@Override
	public Object clone() throws CloneNotSupportedException {
		return super.clone();
	}
}
```

PAQUETE com.product.domain
CLASE TTIPOEXAMEN

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
```

```
/**
 * TTipoExamen generated by hbm2java
 */
@Entity
@Table(name = "t_tipo_examen", schema = "public")
@NamedQueries({ @NamedQuery(name = "TTipoExamen.getAllTipoExamenes", query =
"SELECT tipoexamenes  FROM TTipoExamen  as tipoexamenes  order by nombreexamen")
})
public class TTipoExamen extends BaseEntityGeneral {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "idtipoexamen")
	private int idtipoexamen;

	private String nombreexamen;
	private String descripcionexamen;

	private Integer system;

	public int getIdtipoexamen() {
		return idtipoexamen;
	}

	public void setIdtipoexamen(int idtipoexamen) {
		this.idtipoexamen = idtipoexamen;
	}

	public String getNombreexamen() {
		return nombreexamen;
	}

	public void setNombreexamen(String nombreexamen) {
		this.nombreexamen = nombreexamen;
	}

	public String getDescripcionexamen() {
		return descripcionexamen;
	}

	public void setDescripcionexamen(String descripcionexamen) {
		this.descripcionexamen = descripcionexamen;
	}

	public Integer getSystem() {
		return system;
	}
```

```
	public void setSystem(Integer system) {
		this.system = system;
	}

	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}

	@Override
	public String toString() {
		return this.nombreexamen;
	}


}
```

PAQUETE com.product.domain
CLASE TTIPOMEDICAMENTOS

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 * TTipoMedicamentos generated by hbm2java
 */
@Entity
@Table(name = "t_tipo_medicamentos", schema = "public")
@NamedQueries({ @NamedQuery(name = "TTipoMedicamentos.getAllTiposMedicamentos",
query = "SELECT tipomedicamento  FROM TTipoMedicamentos  as tipomedicamento
order by idtipomedicamento") })
public class TTipoMedicamentos extends BaseEntityGeneral {

	/**
	 *
```

```
 */
private static final long serialVersionUID = 1L;

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "idtipomedicamento")
private int idtipomedicamento;

@Column(name = "descripcion", length = 50)
private String descripcion;

private Integer system;


public TTipoMedicamentos() {
}


public int getIdtipomedicamento() {
	return idtipomedicamento;
}


public void setIdtipomedicamento(int idtipomedicamento) {
	this.idtipomedicamento = idtipomedicamento;
}


public String getDescripcion() {
	return descripcion;
}


public void setDescripcion(String descripcion) {
	this.descripcion = descripcion;
}
public Integer getSystem() {
	return system;
}


public void setSystem(Integer system) {
	this.system = system;
}

@Override
public void setDeactivatedReason(String reason) {
	// TODO Auto-generated method stub

}

@Override
```

```
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}
	@Override
	public String toString() {
		return this.descripcion;
	}


}
```

PAQUETE com.product.domain
CLASE TTURNOS

```
package com.product.domain;

// Generated 15-jun-2015 19:23:07 by Hibernate Tools 3.4.0.CR1

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.Valid;

/**
 * TTurno generated by hbm2java
 */
@Entity
@Table(name = "t_turno", schema = "public")
@NamedQueries({

	//@NamedQuery(name = "TTurno.getAllTTurnoUno", query = "SELECT turnos
FROM TTurno  as turnos  inner JOIN fetch turnos.medico as medico WHERE
turnos.descripcionturno = :arg1 and turnos.active = 1"),
	@NamedQuery(name = "TTurno.getAllTTurno", query = "SELECT turnos  FROM
TTurno  as turnos  inner JOIN fetch turnos.hora as hora inner JOIN fetch
turnos.persona as persona inner JOIN fetch turnos.persona as persona order by
idturno") })
public class TTurno extends BaseEntityGeneral {

	/**
```

```
     *
     */
    private static final long serialVersionUID = 1L;


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int idturno;

    @Valid
    @ManyToOne(fetch = FetchType.LAZY, optional=false)
    @JoinColumn(name = "idmedico")
    private TMedico medico;

    @Valid
    @ManyToOne(fetch = FetchType.LAZY, optional=false)
    @JoinColumn(name = "persona")
    private TPersona persona;

    @Valid
    @ManyToOne(fetch = FetchType.LAZY, optional=false)
    @JoinColumn(name = "idhora")
    private THoras hora;

//  private String descripcionturno;

    @Temporal(TemporalType.DATE)
    @Column(name = "fechaturno")
    private Date fechaturno;

    private String estado;
    private Integer system;


    public int getIdturno() {
        return idturno;
    }
    public void setIdturno(int idturno) {
        this.idturno = idturno;
    }
    public TMedico getMedico() {
        return medico;
    }
    public void setMedico(TMedico medico) {
        this.medico = medico;
    }
    public Integer getSystem() {
        return system;
    }
    public void setSystem(Integer system) {
        this.system = system;
    }
```

```
	public THoras getHora() {
		return hora;
	}
	public void setHora(THoras hora) {
		this.hora = hora;
	}
	public Date getFechaturno() {
		return fechaturno;
	}
	public void setFechaturno(Date fechaturno) {
		this.fechaturno = fechaturno;
	}
	public String getEstado() {
		return estado;
	}
	public void setEstado(String estado) {
		this.estado = estado;
	}
	@Override
	public void setDeactivatedReason(String reason) {
		// TODO Auto-generated method stub
	}

	@Override
	public void setActivatedReason(String reason) {
		// TODO Auto-generated method stub

	}
	public TPersona getPersona() {
		return persona;
	}
	public void setPersona(TPersona persona) {
		this.persona = persona;
	}

@Override
public String toString()
{
		return
this.getHora().getHoras();//getMedico().getPersona().getNombrespersona();
}

}
```

PAQUETE com.product.domain
CLASE USERS

```
package com.product.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
```

```
import javax.persistence.FetchType;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.Valid;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.NotBlank;

@Entity
@Table(name = "users")
@NamedQueries({
		@NamedQuery(name = "Users.findActiveUserByUserID", query = "SELECT
usr  FROM Users  as usr  inner JOIN fetch usr.userRole as usrole WHERE
usr.userName = :arg1  and usr.active = 1"),
		@NamedQuery(name = "Users.getAllUsers", query = "SELECT usr  FROM
Users  as usr  inner JOIN fetch usr.userRole as usrole  order by userName") })
public class Users extends BaseEntity {

	private static final long serialVersionUID = 1L;
	@NotBlank(message = "User Name cannot be empty")
	private String userName;
	@NotBlank(message = "Password cannot be empty")
	private String password;
	@NotBlank(message = "First Name cannot be empty")
	private String firstName;
	private String middleName;
	private String estado;
	private Integer system;

	@Lob
	@Column(name = "foto")
	private byte[] foto;


	// bi-directional many-to-one association to Practice
	@NotNull(message = "Users Role cannot be Empty")
	@Valid
	@ManyToOne(fetch = FetchType.LAZY, optional=false)
	@JoinColumn(name = "roleID")
	private Roles userRole;

	public Roles getUserRole() {
		return userRole;
	}

	public void setUserRole(Roles userRole) {
		this.userRole = userRole;
	}
```

```
public String getUserName() {
	return userName;
}

public void setUserName(String userName) {
	this.userName = userName;
}

public String getPassword() {
	return password;
}

public void setPassword(String password) {
	this.password = password;
}

public String getFirstName() {
	return firstName;
}

public void setFirstName(String firstName) {
	this.firstName = firstName;
}
public String getMiddleName() {
	return middleName;
}

public void setMiddleName(String middleName) {
	this.middleName = middleName;
}

public String getEstado() {
	return estado;
}

public void setEstado(String estado) {
	this.estado = estado;
}


public Integer getSystem() {
	return system;
}

public void setSystem(Integer system) {
	this.system = system;
}

public byte[] getFoto() {
	return foto;
}
```

```
	public void setFoto(byte[] foto) {
		this.foto = foto;
	}

}
```

PAQUETE com.product.webapp
CLASE ErrorDetailVM

```
package com.product.webapp;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.HashMap;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.select.Selectors;
public class ErrorDetailVM {

	private String errMessage;
	private Integer errCode;
	private Exception exception;
	private String errFrom;
	private String moreErrorInfo;

	public String getMoreErrorInfo() {
		return moreErrorInfo;
	}

	public void setMoreErrorInfo(String moreErrorInfo) {
		this.moreErrorInfo = moreErrorInfo;
	}

	public String getErrFrom() {
		return errFrom;
	}

	public void setErrFrom(String errFrom) {
		this.errFrom = errFrom;
	}

	public String getErrMessage() {
		return errMessage;
	}

	public void setErrMessage(String errMessage) {
		this.errMessage = errMessage;
	}
```

```
	public Integer getErrCode() {
		return errCode;
	}

	public void setErrCode(Integer errCode) {
		this.errCode = errCode;
	}

	@SuppressWarnings("unchecked")
	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view) {
		Selectors.wireComponents(view, this, false);

		final HashMap<String, Object> map = (HashMap<String, Object>)
Sessions
				.getCurrent().getAttribute("errordetail");
		this.errCode = (Integer) map.get("errCode");
		this.errMessage = (String) map.get("errMessage");
		this.errFrom = (String) map.get("errFrom");
		this.exception = (Exception) map.get("exception");
		this.moreErrorInfo = (String) map.get("moreErrorInfo");
		StringWriter sw = new StringWriter();
		PrintWriter pw = new PrintWriter(sw);
		exception.printStackTrace(pw);
		this.errMessage = sw.toString();

	}
}
```

PAQUETE com.product.webapp
CLASE ErrorDialogVM

```
package com.product.webapp;


import java.util.HashMap;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zul.Window;

public class ErrorDialogVM {

	@Wire("#errordialog")
```

```
	private Window win;

	private Integer errCode;
	private String errMessage;
	private String sorryMessage;
	private Exception exception;
	private String moreErrorInfo;

	public String getSorryMessage() {
		return sorryMessage;
	}

	public void setSorryMessage(String sorryMessage) {
		this.sorryMessage = sorryMessage;
	}

	public String getMoreErrorInfo() {
		return moreErrorInfo;
	}

	public void setMoreErrorInfo(String moreErrorInfo) {
		this.moreErrorInfo = moreErrorInfo;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("map") HashMap<String, Object> map) {
		Selectors.wireComponents(view, this, false);
		this.errCode = 0;
		this.errMessage = "";
		this.exception = (Exception) map.get("Exception");
		this.moreErrorInfo = (String) map.get("moreErrorInfo");
		this.sorryMessage = "\tNo se pudo ejecutar la transacción\n\t
!!!Error registros duplicados!!!\n\n Verifique la información y vuelva a
ingresar";
	}

	@Command
	public void closeThis() {
		win.detach();
	}

	@Command
	public void showErrorDetail() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("errCode", this.errCode);
		map.put("errMessage", this.errMessage);
		map.put("exception", this.exception);
		map.put("errFrom", "Application Error Handler");
		map.put("moreErrorInfo", moreErrorInfo);
		Sessions.getCurrent().setAttribute("errordetail", map);
		Executions.getCurrent().sendRedirect("/errordetail.zul", "_blank");
```

```
			win.detach();
		}

}
```

PAQUETE com.product.webapp
CLASE ErrorMsgVM

```
package com.product.webapp;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.HashMap;
import java.util.Map;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zul.Window;

import com.product.webapp.utilities.FHSessionUtil;

public class ErrorMsgVM {

	static final Logger LOG = LoggerFactory.getLogger(ErrorMsgVM.class);
	@Wire("#error")
	private Window win;

	@WireVariable("requestScope")
	private Map<String, Object> _requestScope;

	private Integer errCode;
	private String errMessage;
	private String sorryMessage;
	private Exception exception;

	public String getSorryMessage() {
		return sorryMessage;
	}

	public void setSorryMessage(String sorryMessage) {
		this.sorryMessage = sorryMessage;
	}
```

```
	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view) {
		Selectors.wireComponents(view, this, false);
		this.errCode = (Integer) _requestScope
				.get("javax.servlet.error.status_code");
		this.errMessage = (String) _requestScope
				.get("javax.servlet.error.message");
		this.exception = (Exception) _requestScope
				.get("javax.servlet.error.exception");
		StringWriter sw = new StringWriter();
		PrintWriter pw = new PrintWriter(sw);
		exception.printStackTrace(pw);
		LOG.error("User ID : {} \nError : {} ",
FHSessionUtil.getCurrentUser()
				.getID(), sw.toString());
	}

	@Command
	public void closeThis() {
		win.detach();
	}

	@Command
	public void showErrorDetail() {

		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("errCode", this.errCode);
		map.put("errMessage", this.errMessage);
		map.put("exception", this.exception);
		map.put("errFrom", "Un Handled Application Error");
		Sessions.getCurrent().setAttribute("errordetail", map);
		Executions.getCurrent().sendRedirect("/errordetail.zul", "_blank");
		win.detach();
	}

}
```

PAQUETE com.product.webapp
CLASE MainVM

```
package com.product.webapp;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Path;
import org.zkoss.zk.ui.event.ClientInfoEvent;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Borderlayout;
import org.zkoss.zul.Center;
import org.zkoss.zul.Popup;
import org.zkoss.zul.Window;

import com.product.business.service.AuthService;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuChildPredicate;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MenusForThisUser;

public class MainVM {

	@SuppressWarnings("unused")
	private static final long serialVersionUID = 1L;
	static final Logger LOG = LoggerFactory.getLogger(MainVM.class);

	@Wire("#popup")
	Popup popup;

	@Wire("#main")
	private Window win;

	@Wire("#mainlayout")
	private Borderlayout borderLayout;
	private String userName;
	private String nombreCompleto;
	private String cedula;
	private String estado;

	private List<MenusForThisUser> allmenu;
	private List<MenusForThisUser> topMenus;
	private List<MenuItems> nodes;
	private List<MenuItems> fullmenuitems;
	@WireVariable
	private AuthService AuthService;
	private String heading1;
	private String heading2;
```

```
private String heading3;

public String getEstado() {
	return estado;
}

public void setEstado(String estado) {
	this.estado = estado;
}

public String getCedula() {
	return cedula;
}

public void setCedula(String cedula) {
	this.cedula = cedula;
}

public String getNombreCompleto() {
	return nombreCompleto;
}

public void setNombreCompleto(String nombreCompleto) {
	this.nombreCompleto = nombreCompleto;
}

public String getHeading1() {
	return heading1;
}

public void setHeading1(String heading1) {
	this.heading1 = heading1;
}

public String getHeading2() {
	return heading2;
}

public void setHeading2(String heading2) {
	this.heading2 = heading2;
}

public String getHeading3() {
	return heading3;
}

public void setHeading3(String heading3) {
	this.heading3 = heading3;
}

public List<MenuItems> getNodes() {
	return nodes;
```

```
	}

	public void setNodes(List<MenuItems> nodes) {
		this.nodes = nodes;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}

	@SuppressWarnings("unchecked")
	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view)
			throws Exception {
		try {
			Selectors.wireComponents(view, this, false);
			this.userName = StringUtils.capitalize(FHSessionUtil
					.getCurrentUser().getUserRole().getRoleName());
			this.nombreCompleto = StringUtils.capitalize(FHSessionUtil
					.getCurrentUser().getFirstName())
					+ " "
					+
StringUtils.capitalize(FHSessionUtil.getCurrentUser()
                                        .getMiddleName());
			this.cedula =
StringUtils.capitalize(FHSessionUtil.getCurrentUser()
					.getUserName());
			this.estado =
StringUtils.capitalize(FHSessionUtil.getCurrentUser()
					.getEstado());
			if (!estado.equals("A")) {
				Infrastructure.showInformationMessage(
						"Debe Cambiar su contraseña", 250);

			}
			fullmenuitems = new ArrayList<MenuItems>();
			AuthService = (AuthService)
SpringUtil.getBean("AuthService");
			FHSessionUtil.setMainContainer(borderLayout.getCenter());

			this.heading1 = "INVERSIONES PONTE TRESA S.A.";
			this.heading2 = "GESTION DE CITAS MEDICA E HISTORIAS
CLINICAS\n\n";

			if (FHSessionUtil.getCurrentUser().getSystem() == 1
					&& FHSessionUtil.getCurrentUser().getSystem() ==
1) {
				allmenu = AuthService.getMenuForSystemUser();
```

```
                }

                else {
                        allmenu = AuthService.getMenuForThisUser(FHSessionUtil
                                        .getCurrentUser().getUserRole().getID());
                }
                if (allmenu.size() == 0) {
                        Infrastructure
                                        .showInformationMessage(
                                                        "El usuario no cuenta con
permisos para realizar esta solicitud",
                                                        600);
                        return;
                }

                topMenus = ((List<MenusForThisUser>) CollectionUtils.select(
                                allmenu, new MenuChildPredicate(0L)));

                nodes = new ArrayList<MenuItems>();

                for (Iterator<MenusForThisUser> i = topMenus.iterator(); i
                                .hasNext();) {
                        MenusForThisUser tmp = i.next();
                        MenuItems m1 = new MenuItems(tmp.getmCaption(),
                                        tmp.getUniqueAppMenu(),
tmp.getUniqueFileName(),
                                        tmp.getCanAdd(), tmp.getCanEdit(),
tmp.getCanDelete(),
                                        tmp.getCanView());
                        fullmenuitems.add(m1);
                        m1 = getChildren(m1, tmp.getmMenuID());
                        nodes.add(m1);
                }
        } catch (Exception e) {
                // TODO: handle exception

        }
    }

    @SuppressWarnings("unchecked")
    private MenuItems getChildren(MenuItems m1, Long prMenuID) {
        List<MenusForThisUser> childs;
        try {

                childs = ((List<MenusForThisUser>)
CollectionUtils.select(allmenu,
                                new MenuChildPredicate(prMenuID)));
                if (childs.size() == 0)
                        return m1;
                for (Iterator<MenusForThisUser> i = childs.iterator();
i.hasNext();) {
                        MenusForThisUser tmp = i.next();
```

```
                        MenuItems ch;
                        ch = new MenuItems(tmp.getmCaption(),
tmp.getUniqueAppMenu(),
                                    tmp.getUniqueFileName(), tmp.getCanAdd(),
                                    tmp.getCanEdit(), tmp.getCanDelete(),
tmp.getCanView());
                        m1.addChild(ch);
                        fullmenuitems.add(ch);
                        ch = getChildren(ch, tmp.getmMenuID());
                  }
                  return m1;
            } catch (Exception e) {
                  return m1;
            }
      }

      @Command
      public void menuClicked(@BindingParam("menuitem") MenuItems menuitem) {
            try {
                  showMenu(menuitem);
            } catch (Exception e) {

            }
      }

      public void showMenu(MenuItems menuitem) {
            try {
                  if (menuitem == null)
                        return;

                  borderLayout = (Borderlayout)
Path.getComponent("/main/mainlayout");

                  /* get an instance of the center area */
                  Center c1 = borderLayout.getCenter();

                  /* clear the center area */
                  c1.getChildren().clear();

                  final HashMap<String, Object> map = new HashMap<String,
Object>();
                  map.put("MenuDetails", menuitem);
                  FHSessionUtil.setMenuItems(menuitem);
                  map.put("container", c1);
                  Executions.createComponents(menuitem.getUrl(), c1, map);
            } catch (Exception e) {
            }
      }

      @Command
      public void Logout() {
            Executions.sendRedirect("/j_spring_security_logout");
```

```
	}

	@Command
	public void onClientInfo(
			@ContextParam(ContextType.TRIGGER_EVENT) ClientInfoEvent evt)
{
		FHSessionUtil.setDesktopHeight(evt.getDesktopHeight() / 65);
	}

}
```

PAQUETE com.product.webapp
CLASE ModalDialogVM

```
package com.product.webapp;

import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.select.SelectorComposer;
import org.zkoss.zk.ui.select.annotation.Listen;
import org.zkoss.zul.Window;

public class ModalDialogVM extends SelectorComposer<Component> {
	private static final long serialVersionUID = 1L;

	@Listen("onClick = #orderBtn")
	public void showModal(Event e) {
		//create a window programmatically and use it as a modal dialog.
		Window window = (Window)Executions.createComponents(
				"/widgets/window/reset_dialog.zul", null, null);
		window.doModal();
	}
}
```

PAQUETE com.product.webapp
CLASE ResetDialogVM

```
package com.product.webapp;

import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
```

```
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.web.fn.ServletFns;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.select.SelectorComposer;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Listen;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Window;
import com.product.business.service.CRUDService;
import com.product.domain.Users;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MailSender;
import com.product.webapp.utilities.msjUtilitarios;
public class ResetDialogVM extends SelectorComposer<Component> {
	private static final long serialVersionUID = 1L;

	@Wire
	private Window modalDialog;


	@WireVariable
	private CRUDService crudService;
	private Users selectedRecord = new Users();

	private String emailUsuario;




	public Window getModalDialog() {
		return modalDialog;
	}

	public void setModalDialog(Window modalDialog) {
		this.modalDialog = modalDialog;
	}

	public String getEmailUsuario() {
		return emailUsuario;
	}

	public void setEmailUsuario(String emailUsuario) {
		this.emailUsuario = emailUsuario;
	}

	public Users getSelectedRecord() {
		return selectedRecord;
```

```
	}

	public void setSelectedRecord(Users selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	@AfterCompose
	@NotifyChange("myImage")
	public void initSetup(@ContextParam(ContextType.VIEW) Component
view)throws Exception
	{
		super.doAfterCompose(view);
		  // Ponemos la imagen de fondo
		  String uri = ServletFns.encodeURL("/images/zk.png");
		  String format = "background-image:url('%1$s'); border: 'none';
background-repeat:repeat-y;";
		  String style = String.format(format, uri);
		  getModalDialog().setStyle(style);
		Selectors.wireComponents(view, this, false);
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		//this.emailUsuario = "Ingrese E-mail....";
		//this.selectedRecord = consultarUsuario(this.emailUsuario);
	}

	// //////CONSULTAR SI EXISTE EL REGISTRO DE PRECONSULTA
		public Users consultarUsuario(String usernames) {
			crudService = (CRUDService)
SpringUtil.getBean("crudService");
			List<Users> pconsultaAux =
crudService.GetListByNamedQuery("Users.getAllUsers");
			Users us = new Users();
			Iterator<Users> i = pconsultaAux.iterator();
			while (i.hasNext()) {
				us = i.next();
				if
(us.getUserName().toUpperCase().trim().equals(usernames.toUpperCase().trim())) {
					return  us;
				}
			}
			return us;
		}

		@Command
```

```
	public void saveThis(@BindingParam("action") Integer action)throws
IOException {
		try {
			this.selectedRecord =
consultarUsuario(this.emailUsuario);
			this.selectedRecord.setEstado("R");
			msjUtilitarios util= new msjUtilitarios();
			String pass= util.getCadenaAlfanumAleatoria(8);
			this.selectedRecord.setPassword(pass);
			MailSender m = new MailSender();
			crudService.Save(this.selectedRecord);
			m.EnviarEmailReseteo(this.emailUsuario, pass);
			Infrastructure.showSuccessmessageEmail();
			modalDialog.detach();
		} catch (Exception e) {
			//ExceptionHandler.handleException(e, "Users ",
this.selectedRecord, ": SaveThis");
		}
	}
	@Command
	public void showModalP() {
		modalDialog.detach();
	}

	@Listen("onClick = #closeBtn")
	public void showModal(Event e) {
		modalDialog.detach();
	}

}
```

PAQUETE com.product.webapp.admin
CLASE CANTONESCRUDVM

```
import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
```

```
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TProvincia;
import com.product.domain.TCanton;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;

public class CantonesCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#cantonesCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TCanton selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";
	private Component container;
	final HashMap<String, Object> callerArg = new HashMap<String, Object>();
	private List<TProvincia> provinciasList;


	public List<TProvincia> getProvinciasList() {
		return provinciasList;
	}

	public void setProvinciasList(List<TProvincia> provinciasList) {
		this.provinciasList = provinciasList;
	}

	public TCanton getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TCanton selectedRecord) {
```

```
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TCanton cantones,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);
		this.provinciasList =
crudService.GetListByNamedQuery("TProvincia.getAllTProvincias");

		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TCanton();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = cantones;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}
```

```
		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = cantones;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}

	}

	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {
			crudService.Save(this.selectedRecord);
			Infrastructure.showSuccessmessage();
			goBack();
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TCanton ",
this.selectedRecord,
					this.moreErrorInfo + ": SaveThis");
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)
				|| (StringUtils.isBlank(dirty.getSrc()))) {
```

```
                goBack();
                return;
            }
            if (StringUtils.isNotBlank(dirty.getSrc())) {
                Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
                            Consts.UNSAVED_TITLE, this,
                            MessageBoxConfirmType.USAVED_CONFIRMATION);
            }
      }

      @Override
      public void onConfirmClick(String code, int button) {
            if (code.equals("unSaved") && button == Messagebox.YES) {
                goBack();
            }
      }

      public void goBack() {
            container.getChildren().clear();
            win.detach();
            Executions.createComponents(ApplicationLinks.CantonesList,
container,
                        callerArg);

      }
}
```

PAQUETE com.product.webapp.admin
CLASE CANTONESLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
```

```
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TCanton;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class CantonesListVM implements ConfirmResponse {

	@Wire("#cantoneslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<TCanton> allReordsInDB = null;
	private List<TCanton> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TCanton selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
```

```
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TCanton.getAllTCantones");
		filteredRecords = (List<TCanton>)
CollectionUtils.select(allReordsInDB,
				new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(@BindingParam("selectedRecord") TCanton
cantones,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(cantones);
			allReordsInDB.add(cantones);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, cantones);
			allReordsInDB.set(this.selectedItemIndex, cantones);
		}
	}

	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<TCanton>();
		for (Iterator<TCanton> i = allReordsInDB.iterator(); i.hasNext();)
{
			TCanton tmp = i.next();
			if
(tmp.getNombrecanton().toLowerCase().indexOf(dataFilter.getName()) == 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
				}
			}
		}
	}

	@Command
	public void onExcelExport() {
```

```
            this.excelColumns = new ArrayList<ExcelColumns>();
            this.excelColumns.add(new ExcelColumns("idcanton", "Código
Cantón"));
            this.excelColumns.add(new ExcelColumns("nombrecanton", "Nombre
Canton", 9600));
            BeanToExcel beanToExcel = new BeanToExcel();
            beanToExcel.setExcelColumns(excelColumns);
            beanToExcel.setDataSheetName("Lista de Cantones");
            beanToExcel.setDataList(filteredRecords);
            beanToExcel.exportToExcel();
      }

      @Command
      public void onDeactivate() {
            ShowWindow.ShowDeactivateWin();
      }

      @Command
      public void onActivate() {
            ShowWindow.ShowActivateWin();
      }

      // note this will be executed from ActivateVM.Java
      @GlobalCommand
      @NotifyChange("selectedItem")
      public void activateThis(@BindingParam("reason") String reason) {
            this.selectedItem.setActive(1);
            this.selectedItem.setActivatedReason(reason);
            crudService.Save(this.selectedItem);
      }

      // note this will be executed from deactivateVM.Java
      @GlobalCommand
      @NotifyChange("selectedItem")
      public void deactivateThis(@BindingParam("reason") String reason) {
            this.selectedItem.setActive(0);
            this.selectedItem.setDeactivatedReason(reason);
            crudService.Save(this.selectedItem);
      }

      @Command
      public void onAddNew() {
            goToCRUD(RecordMode.ADD);
      }

      @Command
      public void openAsReadOnly() {
            goToCRUD(RecordMode.READ);
      }

      @Command
      public void onEdit() {
```

```
			goToCRUD(RecordMode.EDIT);
		}

		@Command
		public void onlinkOpen(@BindingParam("record") TCanton record)
				throws Exception, NoSuchFieldException {
			if (menu.getCanEdit() == 1)
				onEdit();
			else
				openAsReadOnly();
		}

		public void goToCRUD(final RecordMode mode) {
			if (mode.equals(RecordMode.ADD))
				CRUDargs.put("selectedRecord", null);
			else {
				CRUDargs.put("selectedRecord", selectedItem);
				setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
			}
			CRUDargs.put("recordMode", mode);
			CRUDargs.put("container", container);
			CRUDargs.put("MenuDetails", menu);
			container.getChildren().clear();
			ShowWindow.openZulFile(ApplicationLinks.CantonesCRUD, container,
CRUDargs);
		}

		@Command
		public void onDelete() {
			Infrastructure.confirm("deleteFirstConfirm", "El Elemento
seleccionado  \""
					+ this.selectedItem.getIdcanton() + "\" se
eliminara.?",
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.DELETE_CONFIRMATION);
		}

		public void deleteSelectedItem() {
			try {
				crudService.delete(selectedItem);
				Infrastructure.showSuccessmessage();
				allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

		filteredRecords.remove(filteredRecords.indexOf(selectedItem));
				BindUtils.postNotifyChange(null, null, this, "dataSet");
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "TCanton ",
this.selectedItem,
						"" + ": onConfirmClick");
			}
		}
```

```
	@Override
	@NotifyChange("dataSet")
	public void onConfirmClick(String code, int button) {
		if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
			Infrastructure
					.confirm(
							"deleteSecondConfirm",
							"El elemento seleccionado  \""
									+
this.selectedItem.getIdcanton()
									+ "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
							Consts.UNSAVED_TITLE, this,

	MessageBoxConfirmType.DELETE_CONFIRMATION);
		}
		if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
			deleteSelectedItem();
		}
	}

	public Window getWin() {
		return win;
	}

	public void setWin(Window win) {
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
```

```
        this.excelColumns = excelColumns;
    }

    public List<TCanton> getAllReordsInDB() {
        return allReordsInDB;
    }

    public void setAllReordsInDB(List<TCanton> allReordsInDB) {
        this.allReordsInDB = allReordsInDB;
    }

    public List<TCanton> getDataSet() {
        return filteredRecords;
    }

    public void setFilteredRecords(List<TCanton> filteredRecords) {
        this.filteredRecords = filteredRecords;
    }

    public Component getContainer() {
        return container;
    }

    public void setContainer(Component container) {
        this.container = container;
    }

    public DataFilter getDataFilter() {
        return dataFilter;
    }

    public void setDataFilter(DataFilter dataFilter) {
        this.dataFilter = dataFilter;
    }

    public MenuItems getMenu() {
        return menu;
    }

    public void setMenu(MenuItems menu) {
        this.menu = menu;
    }

    public Integer getSelectedItemIndex() {
        return selectedItemIndex;
    }

    public void setSelectedItemIndex(Integer selectedItemIndex) {
        this.selectedItemIndex = selectedItemIndex;
    }

    public Integer getPageSize() {
```

```
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TCanton getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TCanton selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE DETALLERECETACRUD

```
package com.product.webapp.admin;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TDetalleReceta;
import com.product.domain.TMedicamento;
```

```
public class DetalleRecetaCrud {

	@Wire("#win")
	private Window win;

	@WireVariable
	private CRUDService crudService;
	private TDetalleReceta selectedDetalle;
	private TDetalleReceta selectedDetalleOrg;

	private List<TMedicamento> medicamentoList;
	private String moreErrorInfo = "";

	private boolean makeAsReadOnly;
	private String recordMode;



	public TDetalleReceta getSelectedDetalle() {
		return selectedDetalle;
	}

	public void setSelectedDetalle(TDetalleReceta selectedDetalle) {
		this.selectedDetalle = selectedDetalle;
	}

	public TDetalleReceta getSelectedDetalleOrg() {
		return selectedDetalleOrg;
	}

	public void setSelectedDetalleOrg(TDetalleReceta selectedDetalleOrg) {
		this.selectedDetalleOrg = selectedDetalleOrg;
	}

	public List<TMedicamento> getMedicamentoList() {
		return medicamentoList;
	}

	public void setMedicamentoList(List<TMedicamento> medicamentoList) {
		this.medicamentoList = medicamentoList;
	}

	public String getMoreErrorInfo() {
		return moreErrorInfo;
	}

	public void setMoreErrorInfo(String moreErrorInfo) {
		this.moreErrorInfo = moreErrorInfo;
	}
```

```
	public String getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(String recordMode) {
		this.recordMode = recordMode;
	}


	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}
	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@Init
	@NotifyChange("selectedDetalle")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("sDetalleRecetas") TDetalleReceta
selectedDetalle,
			@ExecutionArgParam("recordMode") String recordMode) throws
CloneNotSupportedException {
		Selectors.wireComponents(view, this, false);
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.medicamentoList =
crudService.GetListByNamedQuery("TMedicamento.getAllTMedicamentos");
		setRecordMode(recordMode);


		if (recordMode.equals("NEW")) {
			this.selectedDetalle = new TDetalleReceta();

		}

		if (recordMode.equals("EDIT")) {
			System.out.println("Value is " +
selectedDetalle.getIddetallereceta());
			this.selectedDetalleOrg = selectedDetalle;
			this.selectedDetalle= (TDetalleReceta)
selectedDetalle.clone();
		}

		if (recordMode == "READ") {
			setMakeAsReadOnly(true);
			win.setTitle(win.getTitle() + " (Readonly)");
		}

	}
```

```
	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void save() {
		Map args = new HashMap();
		selectedDetalle.setActive(1);
		selectedDetalle.setSystem(0);
		args.put("pDetalleRecetas", this.selectedDetalle);
		args.put("recordMode", this.recordMode);
		BindUtils.postGlobalCommand(null, null, "updateRecetasInfo", args);
		//win.detach();
		closeThis() ;
}


	@Command
	public void closeThis() {
		win.detach();
	}

}
```

PAQUETE com.product.webapp.admin
CLASE DETALLERESULTADOEXAMENESCRUD

```
package com.product.webapp.admin;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.zkoss.bind.BindContext;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.io.Files;
import org.zkoss.util.media.AMedia;
import org.zkoss.util.media.Media;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.event.UploadEvent;
import org.zkoss.zk.ui.select.Selectors;
```

```
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Iframe;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TResultadoExamen;
import com.product.domain.TTipoExamen;
import com.product.webapp.utilities.EncryptionUtil;



public class DetalleResultadoExamenesCrud {

	@Wire("#win")
	private Window win;

	@WireVariable
	private CRUDService crudService;
	private TResultadoExamen selectedDetalle;
	private TResultadoExamen selectedDetalleOrg;
	private String moreErrorInfo = "";
	private String filePath;

	private boolean makeAsReadOnly;
	private String recordMode;
	private List<TTipoExamen> examenesList;




	public List<TTipoExamen> getExamenesList() {
		return examenesList;
	}

	public void setExamenesList(List<TTipoExamen> examenesList) {
		this.examenesList = examenesList;
	}

	public String getFilePath() {
		return filePath;
	}

	public void setFilePath(String filePath) {
		this.filePath = filePath;
	}

	public TResultadoExamen getSelectedDetalle() {
		return selectedDetalle;
```

```
	}

	public void setSelectedDetalle(TResultadoExamen selectedDetalle) {
		this.selectedDetalle = selectedDetalle;
	}

	public TResultadoExamen getSelectedDetalleOrg() {
		return selectedDetalleOrg;
	}

	public void setSelectedDetalleOrg(TResultadoExamen selectedDetalleOrg) {
		this.selectedDetalleOrg = selectedDetalleOrg;
	}

	public String getMoreErrorInfo() {
		return moreErrorInfo;
	}

	public void setMoreErrorInfo(String moreErrorInfo) {
		this.moreErrorInfo = moreErrorInfo;
	}


	public String getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(String recordMode) {
		this.recordMode = recordMode;
	}


	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}
	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@Init
	@NotifyChange("selectedDetalle")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("sDetalleResultadoExamen")
TResultadoExamen selectedDetalle,
			@ExecutionArgParam("recordMode") String recordMode) throws
CloneNotSupportedException {
		Selectors.wireComponents(view, this, false);
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.examenesList =
crudService.GetListByNamedQuery("TTipoExamen.getAllTipoExamenes");
		setRecordMode(recordMode);
```

```
		if (recordMode.equals("NEW")) {
			this.selectedDetalle = new TResultadoExamen();

		}

		if (recordMode.equals("EDIT")) {
			this.selectedDetalleOrg = selectedDetalle;
			this.selectedDetalle= (TResultadoExamen)
selectedDetalle.clone();
		}

		if (recordMode == "READ") {
			setMakeAsReadOnly(true);
			win.setTitle(win.getTitle() + " (Readonly)");
		}

	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void save() {
		Map args = new HashMap();
		selectedDetalle.setActive(1);
		selectedDetalle.setSystem(0);
		selectedDetalle.setDescripcionimagen(filePath);
		selectedDetalle.setFechaentrega(new Date());
		args.put("pDetalleResultadoExamen", this.selectedDetalle);
		args.put("recordMode", this.recordMode);
		BindUtils.postGlobalCommand(null, null,
"updateResultadoExamenInfo", args);
		//win.detach();
		closeThis() ;
}

	@Command
	@NotifyChange("fileuploaded")
	public void onUploadPDF(
			@ContextParam(ContextType.BIND_CONTEXT) BindContext ctx)
			throws IOException {

		UploadEvent upEvent = null;
		Object objUploadEvent = ctx.getTriggerEvent();
		if (objUploadEvent != null && (objUploadEvent instanceof
UploadEvent)) {
			upEvent = (UploadEvent) objUploadEvent;
		}
		if (upEvent != null) {
			Media media = upEvent.getMedia();
			Calendar now = Calendar.getInstance();
			int year = now.get(Calendar.YEAR);
			int month = now.get(Calendar.MONTH); // Note: zero based!
```

```
                    int day = now.get(Calendar.DAY_OF_MONTH);
                    filePath =
Executions.getCurrent().getDesktop().getWebApp().getRealPath("/ImgExamenes");
                    String yearPath = "\\ImgExamen" + "\\" + year + "_" + month +
"_"+ day + "\\";
                    filePath = filePath + yearPath;
                    File baseDir = new File(filePath);
                    if (!baseDir.exists()) {
                          baseDir.mkdirs();
                    }
                    Files.copy(new File(filePath + media.getName()),
                                media.getStreamData());
                     //Messagebox.show("File Sucessfully uploaded in the path
["+filePath + " ]");
                    //fileuploaded = true;
                    filePath = filePath + media.getName();
                    selectedDetalle.setDescripcionimagen(filePath);
                    //Messagebox.show(filePath);
              }
       }

       @Command
       public void showPDFOption1() throws IOException {
              Window win = (Window)
Executions.createComponents("zk/components/zkpdfviewer.zul", null, null);
              try {
                    String ruta= selectedDetalle.getDescripcionimagen();

                    Iframe frame = (Iframe) win.getFellow("reportframe");
                    File f = new File(ruta);               //
Messagebox.show(""+f);
                    byte[] buffer = new byte[(int) f.length()];
                    FileInputStream fs = new FileInputStream(f);
                    fs.read(buffer);
                    fs.close();
                    ByteArrayInputStream is = new ByteArrayInputStream(buffer);
                    AMedia amedia = new AMedia(filePath, "pdf",
"application/pdf", is);
                    frame.setContent(amedia);
                     Messagebox.show(ruta);

              } catch (Exception e) {
                    win.detach();
                    Messagebox.show("No se ha encontrado ninguna imagen");
              }

       }

       @Command
       public void showPDFOption2() throws IOException {
              String URL;
              URL = "zkpdfviewer.zul?filepath=";
```

```
			Messagebox.show("22  " + filePath);
			URL = URL + EncryptionUtil.encode(filePath);
			Executions.getCurrent().sendRedirect(URL, "_blank");

		}



		@Command
		public void closeThis() {
			win.detach();
		}

}
```

PAQUETE com.product.webapp.admin
CLASE DETALLESHCCRUDVM

```
package com.product.webapp.admin;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TDetalleHistoriaClinica;
import com.product.domain.THistoriaClinica;
import com.product.domain.TMedico;
import com.product.domain.TPreConsulta;



public class DetallesHCCRUDVM {

		@Wire("#win")
		private Window win;
```

```
@WireVariable
private CRUDService crudService;
private TDetalleHistoriaClinica selectedDetalle;
private TDetalleHistoriaClinica selectedDetalleOrg;

private List<TMedico> medicoList;
private List<THistoriaClinica> historiaCList;
private String moreErrorInfo = "";

private boolean makeAsReadOnly;
private String recordMode;



public String getMoreErrorInfo() {
	return moreErrorInfo;
}

public void setMoreErrorInfo(String moreErrorInfo) {
	this.moreErrorInfo = moreErrorInfo;
}

public List<THistoriaClinica> getHistoriaCList() {
	return historiaCList;
}

public void setHistoriaCList(List<THistoriaClinica> historiaCList) {
	this.historiaCList = historiaCList;
}

public List<TMedico> getMedicoList() {
	return medicoList;
}

public void setMedicoList(List<TMedico> medicoList) {
	this.medicoList = medicoList;
}

public String getRecordMode() {
	return recordMode;
}

public void setRecordMode(String recordMode) {
	this.recordMode = recordMode;
}

public TDetalleHistoriaClinica getSelectedDetalle() {
	return selectedDetalle;
}

public void setSelectedDetalle(TDetalleHistoriaClinica selectedDetalle) {
```

```
			this.selectedDetalle = selectedDetalle;
		}

		public TDetalleHistoriaClinica getSelectedDetalleOrg() {
			return selectedDetalleOrg;
		}

		public void setSelectedDetalleOrg(TDetalleHistoriaClinica
selectedDetalleOrg) {
			this.selectedDetalleOrg = selectedDetalleOrg;
		}

		public boolean isMakeAsReadOnly() {
			return makeAsReadOnly;
		}
		public void setMakeAsReadOnly(boolean makeAsReadOnly) {
			this.makeAsReadOnly = makeAsReadOnly;
		}

		@Init
		@NotifyChange("selectedDetalle")
		public void initSetup(@ContextParam(ContextType.VIEW) Component view,
				@ExecutionArgParam("sDetalleHC") TDetalleHistoriaClinica
selectedDetalle,
				@ExecutionArgParam("sPreconsulta") TPreConsulta
selectedPreconsulta,
				@ExecutionArgParam("recordMode") String recordMode) throws
CloneNotSupportedException {
			Selectors.wireComponents(view, this, false);
			crudService = (CRUDService) SpringUtil.getBean("crudService");
			this.medicoList =
crudService.GetListByNamedQuery("TMedico.getAllTMedicos");
			setRecordMode(recordMode);


			if (recordMode.equals("NEW")) {
				this.selectedDetalle = new TDetalleHistoriaClinica();

			}

			if (recordMode.equals("EDIT")) {
				System.out.println("Value is " +
selectedDetalle.getIddetallehc());
				this.selectedDetalleOrg = selectedDetalle;
				this.selectedDetalle= (TDetalleHistoriaClinica)
selectedDetalle.clone();
			}

			if (recordMode == "READ") {
				setMakeAsReadOnly(true);
				win.setTitle(win.getTitle() + " (Readonly)");
			}
```

```
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void save() {
		Map args = new HashMap();
		selectedDetalle.setActive(1);
		selectedDetalle.setSystem(0);

	this.selectedDetalle.setPersona(this.selectedDetalle.getMedico().getPerso
na());//getTHistoriaClinica().getPersona());
		args.put("pDetalleHC", this.selectedDetalle);
		args.put("recordMode", this.recordMode);
		BindUtils.postGlobalCommand(null, null, "updateEmployeeInfo",
args);
		//win.detach();
		closeThis() ;
}


	@Command
	public void closeThis() {
		win.detach();
	}

}
```

PAQUETE com.product.webapp.admin
CLASE DIAGNOSTICOCRUD

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.Init;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Window;
```

```
import com.product.business.service.CRUDService;
import com.product.domain.TDetalleHistoriaClinica;
import com.product.domain.TDiagMedico;
import com.product.domain.TEnfermedad;



public class DiagnosticoCrud {

	@Wire("#win")
	private Window win;

	private TDetalleHistoriaClinica texto1;
	private String texto2;

	@WireVariable
	private CRUDService crudService;
	private TDiagMedico selectedTDiagMedico;
	private TDiagMedico selectedTDiagMedicoOrg;
	private List<TDetalleHistoriaClinica> detalleHCList;

	private List<TDiagMedico> diagMedico = new ArrayList<TDiagMedico>();
	private List<TEnfermedad> enfermedadList=new ArrayList<TEnfermedad>();

	private TDiagMedico curSelectedTDiagMedico;
	private Integer curSelectedTDiagMedicoIndex;



	public List<TEnfermedad> getEnfermedadList() {
		return enfermedadList;
	}

	public void setEnfermedadList(List<TEnfermedad> enfermedadList) {
		this.enfermedadList = enfermedadList;
	}

	public List<TDiagMedico> getDiagMedico() {
		return diagMedico;
	}

	public void setDiagMedico(List<TDiagMedico> diagMedico) {
		this.diagMedico = diagMedico;
	}

	public TDiagMedico getCurSelectedTDiagMedico() {
		return curSelectedTDiagMedico;
	}

	public void setCurSelectedTDiagMedico(TDiagMedico curSelectedTDiagMedico)
{
```

```
		this.curSelectedTDiagMedico = curSelectedTDiagMedico;
	}

	public Integer getCurSelectedTDiagMedicoIndex() {
		return curSelectedTDiagMedicoIndex;
	}

	public void setCurSelectedTDiagMedicoIndex(Integer
curSelectedTDiagMedicoIndex) {
		this.curSelectedTDiagMedicoIndex = curSelectedTDiagMedicoIndex;
	}


	private boolean makeAsReadOnly;
	private String recordMode;



	public List<TDetalleHistoriaClinica> getDetalleHCList() {
		return detalleHCList;
	}

	public void setDetalleHCList(List<TDetalleHistoriaClinica> detalleHCList)
{
		this.detalleHCList = detalleHCList;
	}

	public String getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(String recordMode) {
		this.recordMode = recordMode;
	}

	public TDiagMedico getSelectedTDiagMedico() {
		return selectedTDiagMedico;
	}

	public void setSelectedTDiagMedico(TDiagMedico selectedTDiagMedico) {
		this.selectedTDiagMedico = selectedTDiagMedico;
	}

	public TDiagMedico getSelectedTDiagMedicoOrg() {
		return selectedTDiagMedicoOrg;
	}

	public void setSelectedTDiagMedicoOrg(TDiagMedico selectedTDiagMedicoOrg)
{
		this.selectedTDiagMedicoOrg = selectedTDiagMedicoOrg;
	}
```

```
	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}
	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	public TDetalleHistoriaClinica getTexto1() {
		return texto1;
	}

	public void setTexto1(TDetalleHistoriaClinica texto1) {
		this.texto1 = texto1;
	}

	public String getTexto2() {
		return texto2;
	}

	public void setTexto2(String texto2) {
		this.texto2 = texto2;
	}

	@SuppressWarnings({ "unchecked", "unused"})
	@Init
//	@NotifyChange("selectedDetalle")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("sDiagnostico") TDiagMedico
selectedPreconsulta,
			@ExecutionArgParam("recordMode") String recordMode) throws
CloneNotSupportedException {
		Selectors.wireComponents(view, this, false);
		///OBTENGO PARAMETRO DESDE HISTORIACCRUDVM DESDE SESSION
		final HashMap<String, Object> map = (HashMap<String, Object>)
Sessions.getCurrent().getAttribute("allmydiagnostico");
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.enfermedadList =
crudService.GetListByNamedQuery("TEnfermedad.getAllTEnfermedad");
		///ASIGNO VALORES DE SESSION
		this.texto1 = (TDetalleHistoriaClinica) map.get("DiagMedico");
		this.texto2 = ""+texto1.getIddetallehc();

		setRecordMode(recordMode);

		if(consultarPersona(texto1.getIddetallehc())==null)
			recordMode="NEW";
		else
			recordMode="EDIT";

		if (recordMode.equals("NEW")) {
			this.selectedTDiagMedico= new TDiagMedico();
		}
```

```
		if (recordMode.equals("EDIT")) {
			List <TDiagMedico> Consulta= new ArrayList<TDiagMedico>();
			selectedPreconsulta =
consultarPersona(texto1.getIddetallehc());
			this.selectedTDiagMedicoOrg = selectedPreconsulta;
			this.selectedTDiagMedico= (TDiagMedico)
selectedPreconsulta.clone();

		}

		if (recordMode == "READ") {
			setMakeAsReadOnly(true);
			win.setTitle(win.getTitle() + " (Readonly)");
		}

	}

	@Command
	public void save() {
		selectedTDiagMedico.setActive(1);
		selectedTDiagMedico.setSystem(0);
		selectedTDiagMedico.setDetalleHC(this.texto1);
		crudService.Save(selectedTDiagMedico);
		closeThis() ;
}

	@Command
	public void closeThis() {
		win.detach();
	}

	@Command
	public void editThisDiagMedico() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("texto1", this.curSelectedTDiagMedico);

	setCurSelectedTDiagMedicoIndex(diagMedico.indexOf(curSelectedTDiagMedico)
);
		map.put("recordMode", "EDIT");
		Executions.createComponents("/zk/master/diagnosticoCrud.zul", null,
map);
	}

	////////CONSULTAR SI EXISTE EL REGISTRO DE PRECONSULTA
	public TDiagMedico  consultarPersona(int idDetalle)
	{
		List<TDiagMedico> pconsultaAux =
crudService.GetListByNamedQuery("TDiagMedico.getAllTDiagMedico");
		TDiagMedico precon= new TDiagMedico();
		TDiagMedico preconAux= new TDiagMedico();
		Iterator<TDiagMedico> i = pconsultaAux.iterator();
```

```
            while (i.hasNext()) {
                  precon = i.next();
                  if (precon.getDetalleHC().getIddetallehc() == idDetalle) {
                        return preconAux=precon;
                  }
            }
            return preconAux;
      }
}
```

PAQUETE com.product.webapp.admin
CLASE ESPECIALIDADCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TEspecialidad;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;

public class EspecialidadCRUDVM implements ConfirmResponse {
```

```
	@WireVariable
	private CRUDService crudService;

	@Wire("#especialidadCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TEspecialidad selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";
	private Component container;
	final HashMap<String, Object> callerArg = new HashMap<String, Object>();


	public TEspecialidad getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TEspecialidad selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TEspecialidad
especialidad,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
```

```
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);


		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TEspecialidad();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new
Date().getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = especialidad;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = especialidad;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}
```

```
	}

	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {
			crudService.Save(this.selectedRecord);
			Infrastructure.showSuccessmessage();
			goBack();
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TEspecialidad ",
this.selectedRecord,
					this.moreErrorInfo + ": SaveThis");
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)
				|| (StringUtils.isBlank(dirty.getSrc()))) {
			goBack();
			return;
		}
		if (StringUtils.isNotBlank(dirty.getSrc())) {
			Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.USAVED_CONFIRMATION);
		}
	}

	@Override
	public void onConfirmClick(String code, int button) {
		if (code.equals("unSaved") && button == Messagebox.YES) {
			goBack();
		}
	}

	public void goBack() {
		container.getChildren().clear();
		win.detach();
		Executions.createComponents(ApplicationLinks.EspecialidadesList,
container,
				callerArg);

	}
}
```

PAQUETE com.product.webapp.admin
CLASE ESPECIALIDADESLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TEspecialidad;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class EspecialidadesListVM implements ConfirmResponse {

	@Wire("#especialidadlist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;
```

```
	@WireVariable
	protected CRUDService crudService;


	private List<ExcelColumns> excelColumns = null;
	private List<TEspecialidad> allReordsInDB = null;
	private List<TEspecialidad> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TEspecialidad selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TEspecialidad.getAllTEspecialidad");
		filteredRecords = (List<TEspecialidad>)
CollectionUtils.select(allReordsInDB,new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(@BindingParam("selectedRecord")
TEspecialidad especialidad,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(especialidad);
			allReordsInDB.add(especialidad);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, especialidad);
			allReordsInDB.set(this.selectedItemIndex, especialidad);
		}
	}
```

```
	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<TEspecialidad>();
		for (Iterator<TEspecialidad> i = allReordsInDB.iterator();
i.hasNext();) {
			TEspecialidad tmp = i.next();
			if
(tmp.getNomEspecialidad().toLowerCase().indexOf(dataFilter.getName()) == 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
				}
			}
		}
	}

	@Command
	public void onExcelExport() {

		this.excelColumns = new ArrayList<ExcelColumns>();
		this.excelColumns.add(new ExcelColumns("userName", "User Name"));
		this.excelColumns
				.add(new ExcelColumns("firstName", "First Name",
9600));
		this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
		BeanToExcel beanToExcel = new BeanToExcel();
		beanToExcel.setExcelColumns(excelColumns);
		beanToExcel.setDataSheetName("TEspecialidadList");
		beanToExcel.setDataList(filteredRecords);
		beanToExcel.exportToExcel();
	}

	@Command
	public void onDeactivate() {
		ShowWindow.ShowDeactivateWin();
	}

	@Command
	public void onActivate() {
		ShowWindow.ShowActivateWin();
	}

	// note this will be executed from ActivateVM.Java
	@GlobalCommand
	@NotifyChange("selectedItem")
	public void activateThis(@BindingParam("reason") String reason) {
		this.selectedItem.setActive(1);
		this.selectedItem.setActivatedReason(reason);
```

```
			crudService.Save(this.selectedItem);
		}

		// note this will be executed from deactivateVM.Java
		@GlobalCommand
		@NotifyChange("selectedItem")
		public void deactivateThis(@BindingParam("reason") String reason) {
			this.selectedItem.setActive(0);
			this.selectedItem.setDeactivatedReason(reason);
			crudService.Save(this.selectedItem);
		}

		@Command
		public void onAddNew() {
			goToCRUD(RecordMode.ADD);
		}

		@Command
		public void openAsReadOnly() {
			goToCRUD(RecordMode.READ);
		}

		@Command
		public void onEdit() {
			goToCRUD(RecordMode.EDIT);
		}

		@Command
		public void onlinkOpen(@BindingParam("record") TEspecialidad record)
				throws Exception, NoSuchFieldException {
			if (menu.getCanEdit() == 1)
				onEdit();
			else
				openAsReadOnly();
		}

		public void goToCRUD(final RecordMode mode) {
			if (mode.equals(RecordMode.ADD))
				CRUDargs.put("selectedRecord", null);
			else {
				CRUDargs.put("selectedRecord", selectedItem);
				setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
			}
			CRUDargs.put("recordMode", mode);
			CRUDargs.put("container", container);
			CRUDargs.put("MenuDetails", menu);
			container.getChildren().clear();
			ShowWindow.openZulFile(ApplicationLinks.EspecialidadesCRUD,
container, CRUDargs);
		}

		@Command
```

```
	public void onDelete() {
		Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado \""
				+ this.selectedItem.getIdespecialidad() + "\" se
eliminará.?",
				Consts.UNSAVED_TITLE, this,
				MessageBoxConfirmType.DELETE_CONFIRMATION);
	}

	public void deleteSelectedItem() {
		try {
			crudService.delete(selectedItem);
			Infrastructure.showSuccessmessage();
			allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

	filteredRecords.remove(filteredRecords.indexOf(selectedItem));
			BindUtils.postNotifyChange(null, null, this, "dataSet");
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "Domain ",
this.selectedItem,
					"" + ": onConfirmClick");
		}
	}

	@Override
	@NotifyChange("dataSet")
	public void onConfirmClick(String code, int button) {
		if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
			Infrastructure
					.confirm(
							"deleteSecondConfirm",
							"El Elemento seleccionado  \""
									+
this.selectedItem.getIdespecialidad()
									+ "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
							Consts.UNSAVED_TITLE, this,

	MessageBoxConfirmType.DELETE_CONFIRMATION);
		}
		if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
			deleteSelectedItem();
		}
	}

	public Window getWin() {
		return win;
	}

	public void setWin(Window win) {
```

```
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
		this.excelColumns = excelColumns;
	}

	public List<TEspecialidad> getAllReordsInDB() {
		return allReordsInDB;
	}

	public void setAllReordsInDB(List<TEspecialidad> allReordsInDB) {
		this.allReordsInDB = allReordsInDB;
	}

	public List<TEspecialidad> getDataSet() {
		return filteredRecords;
	}

	public void setFilteredRecords(List<TEspecialidad> filteredRecords) {
		this.filteredRecords = filteredRecords;
	}

	public Component getContainer() {
		return container;
	}

	public void setContainer(Component container) {
		this.container = container;
	}

	public DataFilter getDataFilter() {
```

```
		return dataFilter;
	}

	public void setDataFilter(DataFilter dataFilter) {
		this.dataFilter = dataFilter;
	}

	public MenuItems getMenu() {
		return menu;
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public Integer getSelectedItemIndex() {
		return selectedItemIndex;
	}

	public void setSelectedItemIndex(Integer selectedItemIndex) {
		this.selectedItemIndex = selectedItemIndex;
	}

	public Integer getPageSize() {
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TEspecialidad getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TEspecialidad selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

Paquete com.product.webapp.admin
CLASE EXAMENESCRUD

```
package com.product.webapp.admin;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.view.JasperViewer;

import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TDetalleHistoriaClinica;
import com.product.domain.TExamen;
import com.product.domain.TResultadoExamen;
import com.product.webapp.utilities.Infrastructure;

public class ExamenesCrud {

	@Wire("#examenCrud")
	private Window win;
	private Connection con;
	@WireVariable
	private CRUDService crudService;
	private boolean makeAsReadOnly;
```

```
	private TExamen selectedTExamenes;
	private List<TResultadoExamen> detalleResultadoExamenes = new
ArrayList<TResultadoExamen>();
	private TResultadoExamen curSelectedDetalleResultadoExamenes;
	private Integer curSelectedDetalleResultadoExamenesIndex;
	private String recordMode;
	private TDetalleHistoriaClinica texto1;
	private String texto2;


	public ExamenesCrud()
	{}
	private List<TResultadoExamen> detalleResultadoExamenesList;


	private List<TDetalleHistoriaClinica> detalleHCList;

	public TExamen getSelectedTExamenes() {
		return selectedTExamenes;
	}

	public void setSelectedTExamenes(TExamen selectedTExamenes) {
		this.selectedTExamenes = selectedTExamenes;
	}

	public List<TResultadoExamen> getallDetalleRE() {
		return detalleResultadoExamenes;
	}

	public List<TResultadoExamen> getDetalleResultadoExamenes() {
		return detalleResultadoExamenes;
	}

	public void setDetalleResultadoExamenes(
			List<TResultadoExamen> detalleResultadoExamenes) {
		this.detalleResultadoExamenes = detalleResultadoExamenes;
	}

	public TResultadoExamen getCurSelectedDetalleResultadoExamenes() {
		return curSelectedDetalleResultadoExamenes;
	}

	public void setCurSelectedDetalleResultadoExamenes(
			TResultadoExamen curSelectedDetalleResultadoExamenes) {
		this.curSelectedDetalleResultadoExamenes =
curSelectedDetalleResultadoExamenes;
	}

	public Integer getCurSelectedDetalleResultadoExamenesIndex() {
		return curSelectedDetalleResultadoExamenesIndex;
	}
```

```
	public void setCurSelectedDetalleResultadoExamenesIndex(
			Integer curSelectedDetalleResultadoExamenesIndex) {
		this.curSelectedDetalleResultadoExamenesIndex =
curSelectedDetalleResultadoExamenesIndex;
	}

	public TDetalleHistoriaClinica getTexto1() {
		return texto1;
	}

	public void setTexto1(TDetalleHistoriaClinica texto1) {
		this.texto1 = texto1;
	}

	public void setDetalleHCList(List<TDetalleHistoriaClinica> detalleHCList)
{
		this.detalleHCList = detalleHCList;
	}

	public List<TDetalleHistoriaClinica> getDetalleHCList() {
		return detalleHCList;
	}

	public List<TResultadoExamen> getDetalleResultadoExamenesList() {
		return detalleResultadoExamenesList;
	}

	public void setDetalleResultadoExamenesList(
			List<TResultadoExamen> detalleResultadoExamenesList) {
		this.detalleResultadoExamenesList = detalleResultadoExamenesList;
	}



	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}


	public String getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(String recordMode) {
		this.recordMode = recordMode;
	}

	public String getTexto2() {
```

```
		return texto2;
	}

	public void setTexto2(String texto2) {
		this.texto2 = texto2;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@SuppressWarnings({"unchecked" })
	@Init
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("sExamenes") TExamen r1,
			@ExecutionArgParam("recordMode") String recordMode) throws
CloneNotSupportedException {
		Selectors.wireComponents(view, this, false);
		// /OBTENGO PARAMETRO DESDE HISTORIACCRUDVM DESDE SESSION
		final HashMap<String, Object> map = (HashMap<String, Object>)
Sessions.getCurrent().getAttribute("allmyExamenes");
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		// /ASIGNO VALORES DE SESSION
		this.texto1 = (TDetalleHistoriaClinica) map.get("Examen");
		this.texto2 = "" + texto1.getIddetallehc();
		setRecordMode(recordMode);

		if (consultarPersonaExamen(texto1.getIddetallehc()) == null)
			recordMode = "NEW";
		else
			recordMode = "EDIT";

		if (recordMode.equals("NEW")) {
			this.selectedTExamenes = new TExamen();

		}

		if (recordMode.equals("EDIT")) {
			this.selectedTExamenes =
consultarPersonaExamen(texto1.getIddetallehc());//(TRecetas) r1.clone();

	setDetalleResultadoExamenes(selectedTExamenes.getTResultadoExamens());
		}

		if (recordMode == "READ") {
			setMakeAsReadOnly(true);
			win.setTitle(win.getTitle() + " (Readonly)");
		}
```

```
	}

	@Command
	public void addNewDetalleResusltadoExamenes() {

		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sDetalleResultadoExamen", null);
		map.put("recordMode", "NEW");

	Executions.createComponents("/zk/master/detallesResultadosExamenes.zul",
null, map);
	}

	@Command
	public void editThisDetalleResultadoExamenes() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sDetalleResultadoExamen",
this.curSelectedDetalleResultadoExamenes);

	setCurSelectedDetalleResultadoExamenesIndex(detalleResultadoExamenes.inde
xOf(curSelectedDetalleResultadoExamenes));
		map.put("recordMode", "EDIT");

	Executions.createComponents("/zk/master/detallesResultadosExamenes.zul",
null, map);
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void deleteThisDetalleResultadoExamenes() {

		String str = "El \""
				+
this.curSelectedDetalleResultadoExamenes.getIdresultadoexamen()
				+ "\" se eliminará de forma permanente y la acción no
se puede deshacer.";

		Messagebox.show(str, "Confirm", Messagebox.OK | Messagebox.CANCEL,
				Messagebox.QUESTION, new EventListener() {
					@Override
					public void onEvent(Event event) throws
Exception {
						if (((Integer)
event.getData()).intValue() == Messagebox.OK) {

	detalleResultadoExamenes.remove(curSelectedDetalleResultadoExamenes);
							BindUtils.postNotifyChange(null,
null,
								ExamenesCrud.this,
"allDetalleRE");
						}
```

```
                                }
                        });
        }

        @Command
        public void save() {
                selectedTExamenes.setTResultadoExamens(detalleResultadoExamenes);
                selectedTExamenes.setActive(1);
                selectedTExamenes.setSystem(0);
                selectedTExamenes.setDetalleHC(this.texto1);
                selectedTExamenes.setCreatedDate(new Date());
                this.selectedTExamenes.setFechasolicitudexamen(new Date());
                crudService.Save(this.selectedTExamenes);
                Infrastructure.showSuccessmessage();
                //closeThis();
        }

        @Command
        public void closeThis() {
                win.focus();
                win.detach();
        }

        @GlobalCommand
        @NotifyChange("allDetalleRE")
        public void updateResultadoExamenInfo(
                        @BindingParam("pDetalleResultadoExamen") TResultadoExamen
dR1,
                        @BindingParam("recordMode") String recordMode) {

                if (recordMode.equals("EDIT")) {
                        dR1.setTExamen(selectedTExamenes);

        detalleResultadoExamenes.set(this.curSelectedDetalleResultadoExamenesInde
x, dR1);
                }

                if (recordMode.equals("NEW")) {
                        dR1.setTExamen(selectedTExamenes);
                        detalleResultadoExamenes.add(dR1);

                }
        }

        // //////CONSULTAR SI EXISTE EL REGISTRO DE PRECONSULTA
        public TExamen consultarPersonaExamen(int idDetalle) {
                List<TExamen> pconsultaAux =
crudService.GetListByNamedQuery("TExamen.getAllTExamen");
                TExamen precon = new TExamen();
                TExamen preconAux = new TExamen();
                Iterator<TExamen> i = pconsultaAux.iterator();
                while (i.hasNext()) {
```

```
                precon = i.next();
                if (precon.getDetalleHC().getIddetallehc() == idDetalle) {
                        return preconAux = precon;
                }
        }
        return preconAux;
    }


    public  String rutaImagen(String idRegistro)
    {
        List<TResultadoExamen> listaImagen=
crudService.GetListByNamedQuery("TResultadoExamen.getAllTResultadoExamen");
        TResultadoExamen im =  new TResultadoExamen();
        Iterator<TResultadoExamen> i = listaImagen.iterator();
        while (i.hasNext()) {
                im = i.next();
                if (im.getIdresultadoexamen()== Integer.parseInt(idRegistro))
{
                        return im.getDescripcionimagen();
                }
        }
        return " ";
    }

    @SuppressWarnings({ "unchecked", "rawtypes" })
    @Command
    public void imprimirExamen() throws JRException {

        try {
                Class.forName("org.postgresql.Driver");
                con =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/sismed",
"postgres","root");
        } catch (Exception e) {
                e.printStackTrace();
        }
        try {
                        HashMap parametros = new HashMap();
                        int idReceta= Integer.parseInt(getTexto2());
                        String reportFile =
Executions.getCurrent().getDesktop().getWebApp().getRealPath("/Reportes");
                                reportFile = reportFile + "/Examen.jrxml";
                                parametros.put("idexamen", idReceta);
                        JasperReport jas =
JasperCompileManager.compileReport(reportFile);
                        JasperPrint jasprint =
JasperFillManager.fillReport(jas,parametros, con);
                        JasperViewer.viewReport(jasprint, false);
                        con.close();
        } catch (Exception e) {
                e.printStackTrace();
```

```
            }

      }
}
```

PAQUETE com.product.webapp.admin
CLASE EXAMENESLIST

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;

import com.product.business.service.CRUDService;
import com.product.domain.TExamen;



public class ExamenesList {

      private List<TExamen> examen= new ArrayList<TExamen>();
      private TExamen curSelectedTExamen;
      private Integer curSelectedTExamenIndex;

      @WireVariable
      protected CRUDService crudService;


      public List<TExamen> getExamen() {
            return examen;
      }

      public void setExamen(List<TExamen> examen) {
            this.examen = examen;
```

```
	}

	public TExamen getCurSelectedTExamen() {
		return curSelectedTExamen;
	}

	public void setCurSelectedTExamen(TExamen curSelectedTExamen) {
		this.curSelectedTExamen = curSelectedTExamen;
	}

	public Integer getCurSelectedTExamenIndex() {
		return curSelectedTExamenIndex;
	}

	public void setCurSelectedTExamenIndex(Integer curSelectedTExamenIndex) {
		this.curSelectedTExamenIndex = curSelectedTExamenIndex;
	}

	@Init
	public void initSetup() {

		crudService = (CRUDService) SpringUtil.getBean("crudService");
		examen= crudService.GetListByNamedQuery("TExamen.getAllTExamen");

	}

	@Command
	public void addNewExamen() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sExamen", null);
		map.put("recordMode", "NEW");
		Executions.createComponents("/zk/master/examenCrud.zul", null,
map);
	}

	@Command
	public void editThisExamenes()
	{
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sExamenes", this.curSelectedTExamen);
		map.put("recordMode", "EDIT");
		setCurSelectedTExamenIndex(examen.indexOf(curSelectedTExamen));
		Executions.createComponents("/zk/master/examenCrud.zul", null,
map);
	}



	@Command
	public void openAsReadOnly()
	{
		examen.get(this.curSelectedTExamenIndex);
```

```
	}

	//The following method will be called from DepartmentCRUDVM.java after
the save
	@GlobalCommand
	@NotifyChange("allExamenes")
	public void updateExamenesInfo(TExamen r1,
			@BindingParam("recordMode") String recordMode) {

		if (recordMode.equals("EDIT")) {
			examen.set(this.curSelectedTExamenIndex, r1);
		}

		if (recordMode.equals("NEW")) {
			examen.add(r1);
		}
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void deleteThisExamenes()
	{
		int OkCancel;

		String str = "El elemento seleccionado  \"" +
curSelectedTExamen.getIdexamen()
				+ "\" se eliminará";
		OkCancel = Messagebox.show(str, "Confirm", Messagebox.OK
				| Messagebox.CANCEL, Messagebox.QUESTION);
		if (OkCancel == Messagebox.CANCEL) {
			return;
		}

		str = "El elemento \""
				+ curSelectedTExamen.getIdexamen()
				+ "\" Se eliminará de forma permanente y la acción no
se puede deshacer.";

		Messagebox.show(str, "Confirm", Messagebox.OK | Messagebox.CANCEL,
				Messagebox.QUESTION, new EventListener() {
					@Override
					public void onEvent(Event event) throws
Exception {
						if (((Integer)
event.getData()).intValue() == Messagebox.OK) {

	crudService.delete(curSelectedTExamen);

	examen.remove(examen.indexOf(curSelectedTExamenIndex));
						BindUtils.postNotifyChange(null,
null,
```

```
                                                                ExamenesList.this,
"allRecetas");
                                        }
                                    }
                                });
        }


}
```

PAQUETE com.product.webapp.admin
CLASE HISTORIACRUDVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TDetalleHistoriaClinica;
import com.product.domain.THistoriaClinica;
import com.product.domain.TPersona;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
```

```
public class HistoriaCCRUDVM {

	@Wire("#historiasClinicas")
	private Window win;

	@WireVariable
	private CRUDService crudService;
	private boolean makeAsReadOnly;
	private THistoriaClinica selectedTHistoria;
	private List<TDetalleHistoriaClinica> detalleHC = new
ArrayList<TDetalleHistoriaClinica>();
	private TDetalleHistoriaClinica curSelectedDetalleHC;
	private Integer curSelectedDetalleHCIndex;
	private String recordMode;
	private List<TPersona> personaList;
	private MenuItems menu;

	private boolean rolAdministrador = false;
	private boolean rolMedico = false;
	private boolean rolEnfermera = false;
	private boolean rolPacientes = false;

	public void verificaRolControl() {
		String RolUsuario = StringUtils.capitalize(FHSessionUtil
				.getCurrentUser().getUserRole().getRoleName());
		switch (RolUsuario) {
		case "Admin": {
			setRolAdministrador(true);
			setRolMedico(false);
			setRolPacientes(false);
			setRolEnfermera(false);
			break;
		}
		case "Paciente": {
			setRolAdministrador(false);
			setRolMedico(false);
			setRolPacientes(true);
			setRolEnfermera(false);
			break;
		}
		case "Medico": {
			setRolAdministrador(false);
			setRolMedico(true);
			setRolPacientes(false);
			setRolEnfermera(false);
			break;
		}
		case "Enfermera": {
			setRolAdministrador(false);
			setRolEnfermera(true);
			setRolMedico(false);
```

```
                setRolPacientes(false);
                break;
        }
        default: {
                setRolAdministrador(false);
                setRolMedico(false);
                setRolPacientes(false);
                setRolEnfermera(false);
                break;
        }
        }
}

public boolean isRolAdministrador() {
        return rolAdministrador;
}

public void setRolAdministrador(boolean rolAdministrador) {
        this.rolAdministrador = rolAdministrador;
}

public boolean isRolMedico() {
        return rolMedico;
}

public void setRolMedico(boolean rolMedico) {
        this.rolMedico = rolMedico;
}

public boolean isRolEnfermera() {
        return rolEnfermera;
}

public void setRolEnfermera(boolean rolEnfermera) {
        this.rolEnfermera = rolEnfermera;
}

public boolean isRolPacientes() {
        return rolPacientes;
}

public void setRolPacientes(boolean rolPacientes) {
        this.rolPacientes = rolPacientes;
}

public MenuItems getMenu() {
        return menu;
}

public void setMenu(MenuItems menu) {
        this.menu = menu;
}
```

```
	public List<TPersona> getPersonaList() {
		return personaList;
	}

	public void setPersonaList(List<TPersona> personaList) {
		this.personaList = personaList;
	}

	public List<TDetalleHistoriaClinica> getallDetalleHC() {
		return detalleHC;
	}

	public void setDetalleHC(List<TDetalleHistoriaClinica> detalleHC) {
		this.detalleHC = detalleHC;
	}

	public String getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(String recordMode) {
		this.recordMode = recordMode;
	}

	public THistoriaClinica getSelectedTHistoria() {
		return selectedTHistoria;
	}

	public void setSelectedTHistoria(THistoriaClinica selectedTHistoria) {
		this.selectedTHistoria = selectedTHistoria;
	}

	public TDetalleHistoriaClinica getCurSelectedDetalleHC() {
		return curSelectedDetalleHC;
	}

	public void setCurSelectedDetalleHC(
			TDetalleHistoriaClinica curSelectedDetalleHC) {
		this.curSelectedDetalleHC = curSelectedDetalleHC;
	}

	public Integer getCurSelectedDetalleHCIndex() {
		return curSelectedDetalleHCIndex;
	}

	public void setCurSelectedDetalleHCIndex(Integer
curSelectedDetalleHCIndex) {
		this.curSelectedDetalleHCIndex = curSelectedDetalleHCIndex;
	}

	public boolean isMakeAsReadOnly() {
```

```
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@Init
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("sHistoriaC") THistoriaClinica hc1,
			@ExecutionArgParam("recordMode") String recordMode,
			@ExecutionArgParam("MenuDetails") MenuItems menu)
			throws CloneNotSupportedException {
		try {
			crudService = (CRUDService)
SpringUtil.getBean("crudService");
			this.personaList = crudService

	.GetListByNamedQuery("TPersona.getAllTPersonas");
			Selectors.wireComponents(view, this, false);
			this.menu = menu;
			verificaRolControl();
			setRecordMode(recordMode);
			if (recordMode.equals("NEW")) {
				this.selectedTHistoria = new THistoriaClinica();
				this.selectedTHistoria.setFecharegistro(new Date());
			}

			if (recordMode.equals("EDIT")) {
				this.selectedTHistoria = (THistoriaClinica)
hc1.clone();
				setDetalleHC(hc1.getTDetalleHistoriaClinicas());
			}

			if (recordMode == "READ") {
				setMakeAsReadOnly(true);
				win.setTitle(win.getTitle() + " (Readonly)");
			}
		} catch (Exception e) {
			// manejo de error
		}

	}

	@Command
	public void addNewDetalleHC() {
		try {
			final HashMap<String, Object> map = new HashMap<String,
Object>();
			map.put("sDetalleHC", null);
			map.put("recordMode", "NEW");
```

```
                  Executions.createComponents("/zk/master/detallesHC.zul",
null, map);
            } catch (Exception e) {
                  // manejo de error
            }
      }

      @Command
      public void editThisDetalleHC() {
            try {
                  final HashMap<String, Object> map = new HashMap<String,
Object>();
                  map.put("sDetalleHC", this.curSelectedDetalleHC);
                  setCurSelectedDetalleHCIndex(detalleHC
                              .indexOf(curSelectedDetalleHC));
                  map.put("recordMode", "EDIT");
                  Executions.createComponents("/zk/master/detallesHC.zul",
null, map);
            } catch (Exception e) {
                  // manejo de error
            }
      }

      @SuppressWarnings({ "unchecked", "rawtypes" })
      @Command
      public void deleteThisDetalleHC() {
            try {
                  String str = "El \""
                              + this.curSelectedDetalleHC.getIddetallehc()
                              + "\" se eliminará de forma permanente y la
acción no se puede deshacer.";

                  Messagebox.show(str, "Confirm", Messagebox.OK |
Messagebox.CANCEL,
                              Messagebox.QUESTION, new EventListener() {
                                    @Override
                                    public void onEvent(Event event) throws
Exception {
                                          if (((Integer)
event.getData()).intValue() == Messagebox.OK) {

      detalleHC.remove(curSelectedDetalleHC);

      BindUtils.postNotifyChange(null, null,

      HistoriaCCRUDVM.this, "allDetalleHC");
                                          }
                                    }
                              });
            } catch (Exception e) {
                  // manejo de error
            }
```

```
	}

	@Command
	@Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor =
Exception.class)
	public void save() {
		try {
			selectedTHistoria.setTDetalleHistoriaClinicas(detalleHC);
			selectedTHistoria.setActive(1);
			selectedTHistoria.setSystem(0);
			selectedTHistoria.setCreatedDate(new Date());
			crudService.Save(this.selectedTHistoria);
			Infrastructure.showSuccessmessage();
			//closeThis();
		} catch (Exception e) {
			e.getMessage();
			//closeThis();
		}
	}

	@Command
	@Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor =
Exception.class)
	public void saveCombo() {
		try {
			selectedTHistoria.setTDetalleHistoriaClinicas(detalleHC);
			selectedTHistoria.setActive(1);
			selectedTHistoria.setSystem(0);
			selectedTHistoria.setCreatedDate(new Date());
			crudService.Save(this.selectedTHistoria);
			Infrastructure.showSuccessmessage();
			//closeThis();
		} catch (Exception e) {
//			e.getMessage();
//			closeThis();
		}
	}

	@Command
	public void closeThis() {
		try {
			win.focus();
			win.detach();
		} catch (Exception e) {
			// manejo de error
		}
	}

	@GlobalCommand
	@NotifyChange("allDetalleHC")
	public void updateEmployeeInfo(
			@BindingParam("pDetalleHC") TDetalleHistoriaClinica dHC1,
```

```
                        @BindingParam("recordMode") String recordMode) {
                try {
                        if (recordMode.equals("EDIT")) {
                                dHC1.setTHistoriaClinica(selectedTHistoria);
                                detalleHC.set(this.curSelectedDetalleHCIndex, dHC1);
                        }

                        if (recordMode.equals("NEW")) {
                                dHC1.setTHistoriaClinica(selectedTHistoria);
                                detalleHC.add(dHC1);

                        }
                } catch (Exception e) {
                        // manejo de error
                }
        }

        // ///////Preconsulta///////////////

        // /ENVIO PARAMETRO DESDE HISTORIACCRUDVM A PRECONSULTACRUD EN SESSION
        // SESSION
        @Command
        public void addNewPreconsulta() {
                final HashMap<String, Object> map = new HashMap<String, Object>();
                map.put("preconsulta", this.curSelectedDetalleHC);// CAMPO
SELECCIONADO
                map.put("recordMode", "EDIT");
                Sessions.getCurrent().setAttribute("allmyvalues", map);
                Executions.createComponents("/zk/master/preconsulta.zul", null,
map);
        }

        // ////////////////////////RECETA/////////////////////////7
        @Command
        public void addNewReceta() {
                final HashMap<String, Object> map = new HashMap<String, Object>();
                map.put("Recetas", this.curSelectedDetalleHC);
                map.put("recordMode", "EDIT");
                Sessions.getCurrent().setAttribute("allmyRecetas", map);
                Executions.createComponents("/zk/master/recetasCrud.zul", null,
map);
        }

        // ////////////////////////DIAGNOSTICO/////////////////////////7
        @Command
        public void addNewDiagnostico() {
                final HashMap<String, Object> map = new HashMap<String, Object>();
                map.put("DiagMedico", this.curSelectedDetalleHC);
                map.put("recordMode", "EDIT");
                Sessions.getCurrent().setAttribute("allmydiagnostico", map);
                Executions
```

```
                        .createComponents("/zk/master/diagnosticoCrud.zul",
null, map);
	}

	// //////////////////////////RECETA/////////////////////////7
	@Command
	public void addNewExamenes() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("Examen", this.curSelectedDetalleHC);
		map.put("recordMode", "EDIT");
		Sessions.getCurrent().setAttribute("allmyExamenes", map);
		Executions.createComponents("/zk/master/examenCrud.zul", null,
map);
	}

}
```

PAQUETE com.product.webapp.admin
CLASE HISTORIASLIST

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Transactional;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
```

```
import com.product.domain.TDetalleHistoriaClinica;
import com.product.domain.THistoriaClinica;
import com.product.domain.TPersona;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;

public class HistoriaCCRUDVM {

	@Wire("#historiasClinicas")
	private Window win;

	@WireVariable
	private CRUDService crudService;
	private boolean makeAsReadOnly;
	private THistoriaClinica selectedTHistoria;
	private List<TDetalleHistoriaClinica> detalleHC = new
ArrayList<TDetalleHistoriaClinica>();
	private TDetalleHistoriaClinica curSelectedDetalleHC;
	private Integer curSelectedDetalleHCIndex;
	private String recordMode;
	private List<TPersona> personaList;
	private MenuItems menu;

	private boolean rolAdministrador = false;
	private boolean rolMedico = false;
	private boolean rolEnfermera = false;
	private boolean rolPacientes = false;

	public void verificaRolControl() {
		String RolUsuario = StringUtils.capitalize(FHSessionUtil
				.getCurrentUser().getUserRole().getRoleName());
		switch (RolUsuario) {
		case "Admin": {
			setRolAdministrador(true);
			setRolMedico(false);
			setRolPacientes(false);
			setRolEnfermera(false);
			break;
		}
		case "Paciente": {
			setRolAdministrador(false);
			setRolMedico(false);
			setRolPacientes(true);
			setRolEnfermera(false);
			break;
		}
		case "Medico": {
			setRolAdministrador(false);
			setRolMedico(true);
			setRolPacientes(false);
			setRolEnfermera(false);
```

```
                break;
        }
        case "Enfermera": {
                setRolAdministrador(false);
                setRolEnfermera(true);
                setRolMedico(false);
                setRolPacientes(false);
                break;
        }
        default: {
                setRolAdministrador(false);
                setRolMedico(false);
                setRolPacientes(false);
                setRolEnfermera(false);
                break;
        }
        }
}

public boolean isRolAdministrador() {
        return rolAdministrador;
}

public void setRolAdministrador(boolean rolAdministrador) {
        this.rolAdministrador = rolAdministrador;
}

public boolean isRolMedico() {
        return rolMedico;
}

public void setRolMedico(boolean rolMedico) {
        this.rolMedico = rolMedico;
}

public boolean isRolEnfermera() {
        return rolEnfermera;
}

public void setRolEnfermera(boolean rolEnfermera) {
        this.rolEnfermera = rolEnfermera;
}

public boolean isRolPacientes() {
        return rolPacientes;
}

public void setRolPacientes(boolean rolPacientes) {
        this.rolPacientes = rolPacientes;
}

public MenuItems getMenu() {
```

```
		return menu;
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public List<TPersona> getPersonaList() {
		return personaList;
	}

	public void setPersonaList(List<TPersona> personaList) {
		this.personaList = personaList;
	}

	public List<TDetalleHistoriaClinica> getallDetalleHC() {
		return detalleHC;
	}

	public void setDetalleHC(List<TDetalleHistoriaClinica> detalleHC) {
		this.detalleHC = detalleHC;
	}

	public String getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(String recordMode) {
		this.recordMode = recordMode;
	}

	public THistoriaClinica getSelectedTHistoria() {
		return selectedTHistoria;
	}

	public void setSelectedTHistoria(THistoriaClinica selectedTHistoria) {
		this.selectedTHistoria = selectedTHistoria;
	}

	public TDetalleHistoriaClinica getCurSelectedDetalleHC() {
		return curSelectedDetalleHC;
	}

	public void setCurSelectedDetalleHC(
			TDetalleHistoriaClinica curSelectedDetalleHC) {
		this.curSelectedDetalleHC = curSelectedDetalleHC;
	}

	public Integer getCurSelectedDetalleHCIndex() {
		return curSelectedDetalleHCIndex;
	}
```

```
	public void setCurSelectedDetalleHCIndex(Integer
curSelectedDetalleHCIndex) {
		this.curSelectedDetalleHCIndex = curSelectedDetalleHCIndex;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@Init
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("sHistoriaC") THistoriaClinica hc1,
			@ExecutionArgParam("recordMode") String recordMode,
			@ExecutionArgParam("MenuDetails") MenuItems menu)
			throws CloneNotSupportedException {
		try {
			crudService = (CRUDService)
SpringUtil.getBean("crudService");
			this.personaList = crudService

	.GetListByNamedQuery("TPersona.getAllTPersonas");
			Selectors.wireComponents(view, this, false);
			this.menu = menu;
			verificaRolControl();
			setRecordMode(recordMode);
			if (recordMode.equals("NEW")) {
				this.selectedTHistoria = new THistoriaClinica();
				this.selectedTHistoria.setFecharegistro(new Date());
			}

			if (recordMode.equals("EDIT")) {
				this.selectedTHistoria = (THistoriaClinica)
hc1.clone();
				setDetalleHC(hc1.getTDetalleHistoriaClinicas());
			}

			if (recordMode == "READ") {
				setMakeAsReadOnly(true);
				win.setTitle(win.getTitle() + " (Readonly)");
			}
		} catch (Exception e) {
			// manejo de error
		}

	}

	@Command
	public void addNewDetalleHC() {
```

```
		try {
			final HashMap<String, Object> map = new HashMap<String,
Object>();
			map.put("sDetalleHC", null);
			map.put("recordMode", "NEW");
			Executions.createComponents("/zk/master/detallesHC.zul",
null, map);
		} catch (Exception e) {
			// manejo de error
		}
	}

	@Command
	public void editThisDetalleHC() {
		try {
			final HashMap<String, Object> map = new HashMap<String,
Object>();
			map.put("sDetalleHC", this.curSelectedDetalleHC);
			setCurSelectedDetalleHCIndex(detalleHC
					.indexOf(curSelectedDetalleHC));
			map.put("recordMode", "EDIT");
			Executions.createComponents("/zk/master/detallesHC.zul",
null, map);
		} catch (Exception e) {
			// manejo de error
		}
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void deleteThisDetalleHC() {
		try {
			String str = "El \""
				+ this.curSelectedDetalleHC.getIddetallehc()
				+ "\" se eliminará de forma permanente y la
acción no se puede deshacer.";

			Messagebox.show(str, "Confirm", Messagebox.OK |
Messagebox.CANCEL,
				Messagebox.QUESTION, new EventListener() {
					@Override
					public void onEvent(Event event) throws
Exception {
						if (((Integer)
event.getData()).intValue() == Messagebox.OK) {

	detalleHC.remove(curSelectedDetalleHC);

	BindUtils.postNotifyChange(null, null,

	HistoriaCCRUDVM.this, "allDetalleHC");
					}
```

```
                                        }
                                  });
                  } catch (Exception e) {
                        // manejo de error
                  }
            }

            @Command
            @Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor =
Exception.class)
            public void save() {
                  try {
                        selectedTHistoria.setTDetalleHistoriaClinicas(detalleHC);
                        selectedTHistoria.setActive(1);
                        selectedTHistoria.setSystem(0);
                        selectedTHistoria.setCreatedDate(new Date());
                        crudService.Save(this.selectedTHistoria);
                        Infrastructure.showSuccessmessage();
                        //closeThis();
                  } catch (Exception e) {
                        e.getMessage();
                        //closeThis();
                  }
            }

            @Command
            @Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor =
Exception.class)
            public void saveCombo() {
                  try {
                        selectedTHistoria.setTDetalleHistoriaClinicas(detalleHC);
                        selectedTHistoria.setActive(1);
                        selectedTHistoria.setSystem(0);
                        selectedTHistoria.setCreatedDate(new Date());
                        crudService.Save(this.selectedTHistoria);
                        Infrastructure.showSuccessmessage();
                        //closeThis();
                  } catch (Exception e) {
//                      e.getMessage();
//                      closeThis();
                  }
            }

            @Command
            public void closeThis() {
                  try {
                        win.focus();
                        win.detach();
                  } catch (Exception e) {
                        // manejo de error
                  }
            }
```

```
	@GlobalCommand
	@NotifyChange("allDetalleHC")
	public void updateEmployeeInfo(
			@BindingParam("pDetalleHC") TDetalleHistoriaClinica dHC1,
			@BindingParam("recordMode") String recordMode) {
		try {
			if (recordMode.equals("EDIT")) {
				dHC1.setTHistoriaClinica(selectedTHistoria);
				detalleHC.set(this.curSelectedDetalleHCIndex, dHC1);
			}

			if (recordMode.equals("NEW")) {
				dHC1.setTHistoriaClinica(selectedTHistoria);
				detalleHC.add(dHC1);

			}
		} catch (Exception e) {
			// manejo de error
		}
	}

	// ///////Preconsulta///////////////

	// /ENVIO PARAMETRO DESDE HISTORIACCRUDVM A PRECONSULTACRUD EN SESSION
	// SESSION
	@Command
	public void addNewPreconsulta() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("preconsulta", this.curSelectedDetalleHC);// CAMPO
SELECCIONADO
		map.put("recordMode", "EDIT");
		Sessions.getCurrent().setAttribute("allmyvalues", map);
		Executions.createComponents("/zk/master/preconsulta.zul", null,
map);
	}

	// ////////////////////////RECETA/////////////////////////7
	@Command
	public void addNewReceta() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("Recetas", this.curSelectedDetalleHC);
		map.put("recordMode", "EDIT");
		Sessions.getCurrent().setAttribute("allmyRecetas", map);
		Executions.createComponents("/zk/master/recetasCrud.zul", null,
map);
	}

	// ////////////////////////DIAGNOSTICO/////////////////////////7
	@Command
	public void addNewDiagnostico() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
```

```
            map.put("DiagMedico", this.curSelectedDetalleHC);
            map.put("recordMode", "EDIT");
            Sessions.getCurrent().setAttribute("allmydiagnostico", map);
            Executions
                        .createComponents("/zk/master/diagnosticoCrud.zul",
null, map);
      }

      // ////////////////////////RECETA/////////////////////////7
      @Command
      public void addNewExamenes() {
            final HashMap<String, Object> map = new HashMap<String, Object>();
            map.put("Examen", this.curSelectedDetalleHC);
            map.put("recordMode", "EDIT");
            Sessions.getCurrent().setAttribute("allmyExamenes", map);
            Executions.createComponents("/zk/master/examenCrud.zul", null,
map);
      }

}
```

PAQUETE com.product.webapp.admin
CLASE MEDICAMENTOSCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TMedicamento;
import com.product.domain.TTipoMedicamentos;
import com.product.webapp.utilities.ApplicationLinks;
```

```
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;

public class MedicamentosCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#medicamentosCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TMedicamento selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";
	private Component container;
	final HashMap<String, Object> callerArg = new HashMap<String, Object>();
	private List<TTipoMedicamentos> tipoMedicamentosList;



	public List<TTipoMedicamentos> getTipoMedicamentosList() {
		return tipoMedicamentosList;
	}

	public void setTipoMedicamentosList(List<TTipoMedicamentos>
tipoMedicamentosList) {
		this.tipoMedicamentosList = tipoMedicamentosList;
	}


	public TMedicamento getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TMedicamento selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}
```

```
	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TMedicamento
medicamento,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);
		this.tipoMedicamentosList =
crudService.GetListByNamedQuery("TTipoMedicamentos.getAllTiposMedicamentos");

		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TMedicamento();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = medicamento;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = medicamento;
		}
```

```
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}

	}

	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {
			crudService.Save(this.selectedRecord);
			Infrastructure.showSuccessmessage();
			goBack();
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TMedicamento ",
this.selectedRecord,
					this.moreErrorInfo + ": SaveThis");
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)
				|| (StringUtils.isBlank(dirty.getSrc()))) {
			goBack();
			return;
		}
		if (StringUtils.isNotBlank(dirty.getSrc())) {
			Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
```

```
                                Consts.UNSAVED_TITLE, this,
                                MessageBoxConfirmType.USAVED_CONFIRMATION);
                }
        }

        @Override
        public void onConfirmClick(String code, int button) {
                if (code.equals("unSaved") && button == Messagebox.YES) {
                        goBack();
                }
        }

        public void goBack() {
                container.getChildren().clear();
                win.detach();
                Executions.createComponents(ApplicationLinks.MedicamentosList,
container,
                                callerArg);

        }
}
```

PAQUETE com.product.webapp.admin
CLASE MEDICAMENTOSLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;
```

```
import com.product.business.service.CRUDService;
import com.product.domain.TMedicamento;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class MedicamentosListVM implements ConfirmResponse {

	@Wire("#medicamentoslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<TMedicamento> allReordsInDB = null;
	private List<TMedicamento> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TMedicamento selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
```

```
            allReordsInDB =
crudService.GetListByNamedQuery("TMedicamento.getAllTMedicamentos");
            filteredRecords = (List<TMedicamento>)
CollectionUtils.select(allReordsInDB,
                        new PredicateActive());

            zulName.setFocus(true);

      }

      @GlobalCommand
      @NotifyChange("dataSet")
      public void UsersListVMRefresh(@BindingParam("selectedRecord")
TMedicamento medicamentos,
                  @BindingParam("recordMode") RecordMode mode) {

            if (mode.equals(RecordMode.ADD)) {
                  filteredRecords.add(medicamentos);
                  allReordsInDB.add(medicamentos);
            }
            if (mode.equals(RecordMode.EDIT)) {
                  filteredRecords.set(this.selectedItemIndex, medicamentos);
                  allReordsInDB.set(this.selectedItemIndex, medicamentos);
            }
      }

      @Command
      @NotifyChange("dataSet")
      public void doFilter() {
            filteredRecords = new ArrayList<TMedicamento>();
            for (Iterator<TMedicamento> i = allReordsInDB.iterator();
i.hasNext();) {
                  TMedicamento tmp = i.next();
                  if
(tmp.getNombremedicamento().toLowerCase().indexOf(dataFilter.getName()) == 0) {
                        if (dataFilter.getIsActive().intValue() == 0)
                              filteredRecords.add(tmp);
                        else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
                              filteredRecords.add(tmp);
                        }
                  }
            }
      }

      @Command
      public void onExcelExport() {

            this.excelColumns = new ArrayList<ExcelColumns>();
            this.excelColumns.add(new ExcelColumns("userName", "User Name"));
            this.excelColumns
```

```
                        .add(new ExcelColumns("firstName", "First Name",
9600));
            this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
            BeanToExcel beanToExcel = new BeanToExcel();
            beanToExcel.setExcelColumns(excelColumns);
            beanToExcel.setDataSheetName("MedicamentoList");
            beanToExcel.setDataList(filteredRecords);
            beanToExcel.exportToExcel();
      }

      @Command
      public void onDeactivate() {
            ShowWindow.ShowDeactivateWin();
      }

      @Command
      public void onActivate() {
            ShowWindow.ShowActivateWin();
      }

      // note this will be executed from ActivateVM.Java
      @GlobalCommand
      @NotifyChange("selectedItem")
      public void activateThis(@BindingParam("reason") String reason) {
            this.selectedItem.setActive(1);
            this.selectedItem.setActivatedReason(reason);
            crudService.Save(this.selectedItem);
      }

      // note this will be executed from deactivateVM.Java
      @GlobalCommand
      @NotifyChange("selectedItem")
      public void deactivateThis(@BindingParam("reason") String reason) {
            this.selectedItem.setActive(0);
            this.selectedItem.setDeactivatedReason(reason);
            crudService.Save(this.selectedItem);
      }

      @Command
      public void onAddNew() {
            goToCRUD(RecordMode.ADD);
      }

      @Command
      public void openAsReadOnly() {
            goToCRUD(RecordMode.READ);
      }

      @Command
      public void onEdit() {
            goToCRUD(RecordMode.EDIT);
```

```
	}

	@Command
	public void onlinkOpen(@BindingParam("record") TMedicamento record)
			throws Exception, NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		container.getChildren().clear();
		ShowWindow.openZulFile(ApplicationLinks.MedicamentosCRUD,
container, CRUDargs);
	}

	@Command
	public void onDelete() {
		Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
				+ this.selectedItem.getIdmedicamento() + "\" se
eliminará.?",
				Consts.UNSAVED_TITLE, this,
				MessageBoxConfirmType.DELETE_CONFIRMATION);
	}

	public void deleteSelectedItem() {
		try {
			crudService.delete(selectedItem);
			Infrastructure.showSuccessmessage();
			allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

	filteredRecords.remove(filteredRecords.indexOf(selectedItem));
			BindUtils.postNotifyChange(null, null, this, "dataSet");
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TMedicamento ",
this.selectedItem,
					"" + ": onConfirmClick");
		}
	}

	@Override
```

```
	@NotifyChange("dataSet")
	public void onConfirmClick(String code, int button) {
		if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
			Infrastructure
					.confirm(
							"deleteSecondConfirm",
							"El elemento seleccionado  \""
								+
this.selectedItem.getIdmedicamento()
								+ "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
							Consts.UNSAVED_TITLE, this,

	MessageBoxConfirmType.DELETE_CONFIRMATION);
		}
		if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
			deleteSelectedItem();
		}
	}

	public Window getWin() {
		return win;
	}

	public void setWin(Window win) {
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
		this.excelColumns = excelColumns;
```

```
}

public List<TMedicamento> getAllReordsInDB() {
	return allReordsInDB;
}

public void setAllReordsInDB(List<TMedicamento> allReordsInDB) {
	this.allReordsInDB = allReordsInDB;
}

public List<TMedicamento> getDataSet() {
	return filteredRecords;
}

public void setFilteredRecords(List<TMedicamento> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
}

public void setMenu(MenuItems menu) {
	this.menu = menu;
}

public Integer getSelectedItemIndex() {
	return selectedItemIndex;
}

public void setSelectedItemIndex(Integer selectedItemIndex) {
	this.selectedItemIndex = selectedItemIndex;
}

public Integer getPageSize() {
	return pageSize;
```

```
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TMedicamento getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TMedicamento selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE MEDICOSCITASCRUDVM

```
package com.product.webapp.admin;

import java.io.IOException;
import java.sql.Timestamp;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.SelectorComposer;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
```

```
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
//import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TEspecialidad;
import com.product.domain.TMedico;
import com.product.domain.TMedicosCitas;
import com.product.domain.TPersona;
import com.product.domain.TTurno;
import com.product.domain.Users;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MailSender;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;


public class MedicosCitasCRUDVM extends SelectorComposer<Component> implements
        ConfirmResponse {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @WireVariable
    private CRUDService crudService;

    @Wire("#medicosCitasCRUD")
    private Window win;

    @Wire("#dirty")
    private Image dirty;

    private TMedicosCitas selectedRecord;
    private TTurno turno;
    private RecordMode recordMode;
    private boolean makeAsReadOnly;
    private String moreErrorInfo = "";
    private Component container;
    final HashMap<String, Object> callerArg = new HashMap<String, Object>();
    private List<TPersona> personasList;
```

```
private List<TTurno> turnoList;
private List<TMedico> medicoList;

private List<TEspecialidad> especialidadList;
private TEspecialidad selectEspecialidad = new TEspecialidad();
private TMedico selectMedico = new TMedico();

private String itmHora = "";

private Date fechacita;

public TTurno getTurno() {
	return turno;
}

public void setTurno(TTurno turno) {
	this.turno = turno;
}

public List<TMedico> getMedicoList() {
	return medicoList;
}

public void setMedicoList(List<TMedico> medicoList) {
	this.medicoList = medicoList;
}

public List<TEspecialidad> getEspecialidadList() {
	return especialidadList;
}

public void setEspecialidadList(List<TEspecialidad> especialidadList) {
	this.especialidadList = especialidadList;
}

public List<TPersona> getPersonasList() {

	return personasList;
}

public void setPersonasList(List<TPersona> personasList) {
	this.personasList = personasList;
}

public List<TTurno> getTurnoList() {
	return turnoList;
}

public void setTurnoList(List<TTurno> turnoList) {
	this.turnoList = turnoList;
}
```

```
public TMedicosCitas getSelectedRecord() {
	return selectedRecord;
}

public void setSelectedRecord(TMedicosCitas selectedRecord) {
	this.selectedRecord = selectedRecord;
}

public RecordMode getRecordMode() {
	return recordMode;
}

public void setRecordMode(RecordMode recordMode) {
	this.recordMode = recordMode;
}

public boolean isMakeAsReadOnly() {
	return makeAsReadOnly;
}

public void setMakeAsReadOnly(boolean makeAsReadOnly) {
	this.makeAsReadOnly = makeAsReadOnly;
}

public TEspecialidad getSelectEspecialidad() {
	return selectEspecialidad;
}

public void setSelectEspecialidad(TEspecialidad selectEspecialidad) {
	this.selectEspecialidad = selectEspecialidad;
}

public TMedico getSelectMedico() {
	return selectMedico;
}

public void setSelectMedico(TMedico selectMedico) {
	this.selectMedico = selectMedico;
}

public Date getFechacita() {
	return fechacita;
}

public void setFechacita(Date fechacita) {
	this.fechacita = fechacita;
}

public String getItmHora() {
	return itmHora;
}
```

```
	public void setItmHora(String itmHora) {
		this.itmHora = itmHora;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TMedicosCitas
medicoCitas,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container)
			throws IOException {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);

		String Rol = StringUtils.capitalize(FHSessionUtil.getCurrentUser()
				.getUserRole().getRoleName());
		if (Rol.trim().equals("Paciente"))
			this.personasList = buscarPersona(StringUtils

	.capitalize(FHSessionUtil.getCurrentUser().getMiddleName()));
		else
			this.personasList = crudService

	.GetListByNamedQuery("TPersona.getAllTPersonas");
		this.especialidadList = crudService

	.GetListByNamedQuery("TEspecialidad.getAllTEspecialidad");
		this.fechacita = new Date();
		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TMedicosCitas();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = medicoCitas;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}
```

```
		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = medicoCitas;
		}
	}

	// //////CONSULTAR SI EXISTE EL REGISTRO DE PRECONSULTA
			public Users consultarUsuario(String usernames) {
				crudService = (CRUDService)
SpringUtil.getBean("crudService");
				List<Users> pconsultaAux =
crudService.GetListByNamedQuery("Users.getAllUsers");
				Users us = new Users();
				Iterator<Users> i = pconsultaAux.iterator();
				while (i.hasNext()) {
					us = i.next();
					if
(us.getMiddleName().toUpperCase().trim().equals(usernames.toUpperCase().trim()))
{
						return  us;
					}
				}
				return us;
			}



	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}

	}
```

```
	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {
			Users usr=
consultarUsuario(this.selectedRecord.getPersona().getCedulapersona());

			MailSender m = new MailSender();

	this.selectedRecord.setEspecialidad(this.selectedRecord.getTurno().getMed
ico().getEspecialidad());

	this.selectedRecord.setMedico(this.selectedRecord.getTurno().getMedico())
;
			crudService.Save(this.selectedRecord);
			Infrastructure.showSuccessmessage();
			m.EnviarEmail(usr.getUserName());
			goBack();
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TMedicosCitas ",
					this.selectedRecord, this.moreErrorInfo + ":
SaveThis");
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)
				|| (StringUtils.isBlank(dirty.getSrc()))) {
			goBack();
			return;
		}
		if (StringUtils.isNotBlank(dirty.getSrc())) {
			Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.USAVED_CONFIRMATION);
		}
	}

	@Override
	public void onConfirmClick(String code, int button) {
		if (code.equals("unSaved") && button == Messagebox.YES) {
			goBack();
		}
	}

	public void goBack() {
		container.getChildren().clear();
		win.detach();
		Executions.createComponents(ApplicationLinks.MedicosCitasList,
```

```
                        container, callerArg);

        }

        public List<TPersona> buscarPersona(String cedula) {
                List<TPersona> listaPersonas = crudService
                                .GetListByNamedQuery("TPersona.getAllTPersonas");
                List<TPersona> listaPersonaAux = new ArrayList<TPersona>();
                TPersona persona = new TPersona();
                String cedulaPersona = "";
                Iterator<TPersona> i = listaPersonas.iterator();
                while (i.hasNext()) {
                        persona = i.next();
                        cedulaPersona = persona.getCedulapersona().trim();
                        if (cedulaPersona.equals(cedula.trim()))
                                listaPersonaAux.add(persona);
                }
                return listaPersonaAux;
        }

        public List<TMedico> retornarMedicoEspecialidad(int codigoEspecialidad) {
                List<TMedico> listadoMedicos = crudService
                                .GetListByNamedQuery("TMedico.getAllTMedicos");
                List<TMedico> listaMedicosAux = new ArrayList<TMedico>();
                TMedico medico = new TMedico();
                Iterator<TMedico> i = listadoMedicos.iterator();
                while (i.hasNext()) {
                        medico = i.next();
                        if (medico.getEspecialidad().getIdespecialidad() ==
codigoEspecialidad) {
                                listaMedicosAux.add(medico);
                        }
                }
                return listaMedicosAux;
        }

        public List<TTurno> retornarTurnoMedicoFecha(Date fecha, int idmedico) {
                List<TTurno> listadoTurnos = crudService
                                .GetListByNamedQuery("TTurno.getAllTTurno");
                List<TTurno> listadoTurnosAux = new ArrayList<TTurno>();
                TTurno turno = new TTurno();
                String formattedDateParam = "";
                String formattedDateTurno = "";
                SimpleDateFormat format;
                format = new SimpleDateFormat("yyyy-MM-dd");
                formattedDateParam = format.format(fecha);
                Iterator<TTurno> i = listadoTurnos.iterator();
                while (i.hasNext()) {
                        turno = i.next();
                        formattedDateTurno = format.format(turno.getFechaturno());
                        if (turno.getMedico().getIdmedico() == idmedico)
                        {
```

```
                        if (formattedDateTurno.compareTo(formattedDateParam)
== 0)
                        {
                              if
(!(turno.getEstado().trim().equals("Reservado")== true) )
                              {
                                    listadoTurnosAux.add(turno);
                              }
                        }
                  }
            }
            return listadoTurnosAux;
      }

      @Command
      @NotifyChange("medicoList")
      public void ShowSelectedEspecialidad() {
            this.medicoList =
retornarMedicoEspecialidad(this.selectEspecialidad
                        .getIdespecialidad());
      }

      @Command
      @NotifyChange("turnoList")
      public void ShowSelectedTurnosMedico() {
            this.turnoList = retornarTurnoMedicoFecha(this.getFechacita(),
                        this.selectMedico.getIdmedico());
      }

}
```

PAQUETE com.product.webapp.admin
CLASE MEDICOSCITASLISTVM

```
package com.product.webapp.admin;

import java.sql.Connection;
import java.sql.DriverManager;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.view.JasperViewer;
```

```
import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TMedico;
import com.product.domain.TMedicosCitas;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class MedicosCitasListVM implements ConfirmResponse {

	@Wire("#medicosCitaslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;
	private Connection con;

	@WireVariable
	protected CRUDService crudService;
	SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
```

```
	private List<ExcelColumns> excelColumns = null;
	private List<TMedicosCitas> allReordsInDB = null;
	private List<TMedicosCitas> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TMedicosCitas selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	private List<TMedico> medicoList;
	private TMedico medico = new TMedico();
	private String cedulamedico;

	// ///////////////PARAMETROS PARA REPORTES
	private String cedula;
	private Date fechaInicio;
	private Date fechaFin;

	public String getCedula() {
		return cedula;
	}

	public void setCedula(String cedula) {
		this.cedula = cedula;
	}

	public Date getFechaInicio() {
		return fechaInicio;
	}

	public void setFechaInicio(Date fechaInicio) {
		this.fechaInicio = fechaInicio;
	}

	public Date getFechaFin() {
		return fechaFin;
	}

	public void setFechaFin(Date fechaFin) {
		this.fechaFin = fechaFin;
	}

	// ////

	public String getCedulamedico() {
		return cedulamedico;
	}

	public void setCedulamedico(String cedulamedico) {
```

```
		this.cedulamedico = cedulamedico;
	}

	public TMedico getMedico() {
		return medico;
	}

	public void setMedico(TMedico medico) {
		this.medico = medico;
	}

	public List<TMedico> getMedicoList() {
		return medicoList;
	}

	public void setMedicoList(List<TMedico> medicoList) {
		this.medicoList = medicoList;
	}

	@SuppressWarnings({ "unchecked", "deprecation" })
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB = crudService

	.GetListByNamedQuery("TMedicosCitas.getAllTMedicosCitas");
		this.medicoList = crudService
				.GetListByNamedQuery("TMedico.getAllTMedicos");
		filteredRecords = (List<TMedicosCitas>) CollectionUtils.select(
				allReordsInDB, new PredicateActive());
		Date fecI = new Date();// fecha Actual
		Date temp;// fecha Inicial
		temp = new Date(fecI.getYear(), fecI.getMonth(), 1);
		zulName.setFocus(true);
		this.setFechaInicio(temp);
		this.setFechaFin(fecI);
	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(
			@BindingParam("selectedRecord") TMedicosCitas medicoCitas,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
```

```
                filteredRecords.add(medicoCitas);
                allReordsInDB.add(medicoCitas);
            }
            if (mode.equals(RecordMode.EDIT)) {
                filteredRecords.set(this.selectedItemIndex, medicoCitas);
                allReordsInDB.set(this.selectedItemIndex, medicoCitas);
            }
        }

        @Command
        @NotifyChange("dataSet")
        public void doFilter() {
            filteredRecords = new ArrayList<TMedicosCitas>();
            String Rol = StringUtils.capitalize(FHSessionUtil.getCurrentUser()
                    .getUserRole().getRoleName());
            String cedulaPer = StringUtils.capitalize(FHSessionUtil
                    .getCurrentUser().getMiddleName());
            if (Rol.trim().equals("Paciente")) {
                dataFilter.setName(cedulaPer);
                for (Iterator<TMedicosCitas> i = allReordsInDB.iterator(); i
                        .hasNext();) {
                    TMedicosCitas tmp = i.next();
                    if (tmp.getPersona().getCedulapersona().trim()
                            .equals(cedulaPer.trim())) {
                        filteredRecords.add(tmp);
                        // break;
                    }
                }
            } else {
                if (Rol.trim().equals("Medico")) {
                    dataFilter.setName(cedulaPer);
                    for (Iterator<TMedicosCitas> i =
allReordsInDB.iterator(); i
                            .hasNext();) {
                        TMedicosCitas tmp = i.next();
                        if
(tmp.getMedico().getPersona().getCedulapersona().trim()
                                .equals(cedulaPer.trim())) {
                            filteredRecords.add(tmp);
                            // break;
                        }
                    }
                } else {
                    filteredRecords = new ArrayList<TMedicosCitas>();
                    for (Iterator<TMedicosCitas> i =
allReordsInDB.iterator(); i
                            .hasNext();) {
                        TMedicosCitas tmp = i.next();
                        if
(tmp.getPersona().getCedulapersona().toLowerCase()
                                .indexOf(dataFilter.getName()) ==
0) {
```

```
                                        if (dataFilter.getIsActive().intValue()
== 0)
                                              filteredRecords.add(tmp);
                                        else if (tmp.getActive().equals(
                                                    dataFilter.getIsActive())) {
                                              filteredRecords.add(tmp);
                                        }
                                  }
                            }
                      }
                }

        }

        @Command
        public void onExcelExport() {

                this.excelColumns = new ArrayList<ExcelColumns>();
                this.excelColumns.add(new ExcelColumns("userName", "User Name"));
                this.excelColumns
                            .add(new ExcelColumns("firstName", "First Name",
9600));
                this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
                BeanToExcel beanToExcel = new BeanToExcel();
                beanToExcel.setExcelColumns(excelColumns);
                beanToExcel.setDataSheetName("MedicosCitasList");
                beanToExcel.setDataList(filteredRecords);
                beanToExcel.exportToExcel();
        }

        @Command
        public void onDeactivate() {
                ShowWindow.ShowDeactivateWin();
        }

        @Command
        public void onActivate() {
                ShowWindow.ShowActivateWin();
        }

        // note this will be executed from ActivateVM.Java
        @GlobalCommand
        @NotifyChange("selectedItem")
        public void activateThis(@BindingParam("reason") String reason) {
                this.selectedItem.setActive(1);
                this.selectedItem.setActivatedReason(reason);
                crudService.Save(this.selectedItem);
        }

        // note this will be executed from deactivateVM.Java
        @GlobalCommand
```

```
	@NotifyChange("selectedItem")
	public void deactivateThis(@BindingParam("reason") String reason) {
		this.selectedItem.setActive(0);
		this.selectedItem.setDeactivatedReason(reason);
		crudService.Save(this.selectedItem);
	}

	@Command
	public void onAddNew() {
		goToCRUD(RecordMode.ADD);
	}

	@Command
	public void openAsReadOnly() {
		goToCRUD(RecordMode.READ);
	}

	@Command
	public void onEdit() {
		goToCRUD(RecordMode.EDIT);
	}

	@Command
	public void onlinkOpen(@BindingParam("record") TMedicosCitas record)
			throws Exception, NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		container.getChildren().clear();
		ShowWindow.openZulFile(ApplicationLinks.MedicosCitasCRUD,
container,
				CRUDargs);
	}

	@Command
	public void onDelete() {
		Infrastructure.confirm(
				"deleteFirstConfirm",
				"El elemento seleccionado  \""
```

```
                                        + this.selectedItem.getIdmedicocita()
                                        + "\" se eliminará.?",
Consts.UNSAVED_TITLE, this,
                        MessageBoxConfirmType.DELETE_CONFIRMATION);
        }

        public void deleteSelectedItem() {
                try {
                        crudService.delete(selectedItem);
                        Infrastructure.showSuccessmessage();
                        allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

        filteredRecords.remove(filteredRecords.indexOf(selectedItem));
                        BindUtils.postNotifyChange(null, null, this, "dataSet");
                } catch (Exception e) {
                        ExceptionHandler.handleException(e, "TMedicosCitas ",
                                        this.selectedItem, "" + ": onConfirmClick");
                }
        }

        @Override
        @NotifyChange("dataSet")
        public void onConfirmClick(String code, int button) {
                if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
                        Infrastructure
                                        .confirm(
                                                        "deleteSecondConfirm",
                                                        "El elemento seleccionado \""
                                                                        +
this.selectedItem.getIdmedicocita()
                                                                        + "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
                                                        Consts.UNSAVED_TITLE, this,

        MessageBoxConfirmType.DELETE_CONFIRMATION);
                }
                if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
                        deleteSelectedItem();
                }
        }

        public Window getWin() {
                return win;
        }

        public void setWin(Window win) {
                this.win = win;
        }

        public Textbox getZulName() {
```

```
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
		this.excelColumns = excelColumns;
	}

	public List<TMedicosCitas> getAllReordsInDB() {
		return allReordsInDB;
	}

	public void setAllReordsInDB(List<TMedicosCitas> allReordsInDB) {
		this.allReordsInDB = allReordsInDB;
	}

	public List<TMedicosCitas> getDataSet() {
		return filteredRecords;
	}

	public void setFilteredRecords(List<TMedicosCitas> filteredRecords) {
		this.filteredRecords = filteredRecords;
	}

	public Component getContainer() {
		return container;
	}

	public void setContainer(Component container) {
		this.container = container;
	}

	public DataFilter getDataFilter() {
		return dataFilter;
	}

	public void setDataFilter(DataFilter dataFilter) {
```

```
		this.dataFilter = dataFilter;
	}

	public MenuItems getMenu() {
		return menu;
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public Integer getSelectedItemIndex() {
		return selectedItemIndex;
	}

	public void setSelectedItemIndex(Integer selectedItemIndex) {
		this.selectedItemIndex = selectedItemIndex;
	}

	public Integer getPageSize() {
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TMedicosCitas getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TMedicosCitas selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void imprimirReporte() throws JRException {

		try {
			Class.forName("org.postgresql.Driver");
			con = DriverManager.getConnection(
					"jdbc:postgresql://localhost:5432/sismed",
"postgres",
```

```
                                    "root");
                  } catch (Exception e) {
                        e.printStackTrace();
                  }
                  try {
                        @SuppressWarnings("unused")
                        String formattedDate = "";
                        @SuppressWarnings("unused")
                        String formattedDate1 = "";
                        Date fecha = getFechaInicio();// Inicio
                        Date dFechaActual = getFechaFin();// Fin
                        formattedDate = format.format(fecha);
                        formattedDate1 = format.format(dFechaActual);
                        if (fecha.after(dFechaActual))// fecha1 mayor que fecha 2
                        {
                              Messagebox.show(
                                          "Fecha inicial no puede ser mayor a fecha
Final",
                                          "Error de fecha", 1, "alert");
                        } else {
                              HashMap parametros = new HashMap();
                              String reportFile =
Executions.getCurrent().getDesktop().getWebApp().getRealPath("/Reportes");


                              if (getCedula().isEmpty() || getCedula().equals(null)
                                          || getCedula().length() == 0) {

                              } else {
                                    reportFile = reportFile +
"/ReporteMedicoCitasAtendidas.jrxml";
                                    parametros.put("cedulamedico", getCedula());
                                    parametros.put("fechainicial",
getFechaInicio());
                                    parametros.put("fechafinal", getFechaFin());
                              }
                              JasperReport jas =
JasperCompileManager.compileReport(reportFile);

                              JasperPrint jasprint =
JasperFillManager.fillReport(jas,parametros, con);
                              JasperViewer.viewReport(jasprint, false);
                              con.close();
                        }
                  } catch (Exception e) {
                        e.printStackTrace();
                  }

            }

}
```

PAQUETE com.product.webapp.admin
CLASE MEDICOSCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TEspecialidad;
import com.product.domain.TMedico;
import com.product.domain.TPersona;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;

public class MedicosCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#medicosCRUD")
	private Window win;

	@Wire("#dirty")
```

```
private Image dirty;

private TMedico selectedRecord;
private RecordMode recordMode;
private boolean makeAsReadOnly;
private String moreErrorInfo = "";
private Component container;
final HashMap<String, Object> callerArg = new HashMap<String, Object>();
private List<TPersona> personasList;
private List<TEspecialidad> especialidadList;



public List<TPersona> getPersonasList() {
	return personasList;
}

public void setPersonasList(List<TPersona> personasList) {
	this.personasList = personasList;
}

public List<TEspecialidad> getEspecialidadList() {
	return especialidadList;
}

public void setEspecialidadList(List<TEspecialidad> especialidadList) {
	this.especialidadList = especialidadList;
}

public TMedico getSelectedRecord() {
	return selectedRecord;
}

public void setSelectedRecord(TMedico selectedRecord) {
	this.selectedRecord = selectedRecord;
}

public RecordMode getRecordMode() {
	return recordMode;
}

public void setRecordMode(RecordMode recordMode) {
	this.recordMode = recordMode;
}

public boolean isMakeAsReadOnly() {
	return makeAsReadOnly;
}

public void setMakeAsReadOnly(boolean makeAsReadOnly) {
	this.makeAsReadOnly = makeAsReadOnly;
}
```

```
	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TMedico medico,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);
		this.personasList =
crudService.GetListByNamedQuery("TPersona.getAllTPersonas");

	this.especialidadList=crudService.GetListByNamedQuery("TEspecialidad.getA
llTEspecialidad");

		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TMedico();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = medico;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = medico;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
```

```
			case 27:
				s = "ESC";
				break;
			case 121:
				s = "F10";
				break;
			}
			if (s.equalsIgnoreCase("F10")) {
				if (!this.makeAsReadOnly)
					saveThis(0);
			}
			if (s.equalsIgnoreCase("ESC")) {
				cancel();
			}

		}

		@Command
		public void saveThis(@BindingParam("action") Integer action) {
			if (Libs.IsValidBean(this.selectedRecord) == false) {
				return;
			}
			try {
				crudService.Save(this.selectedRecord);
				Infrastructure.showSuccessmessage();
				goBack();
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "TMedico ",
this.selectedRecord,
						this.moreErrorInfo + ": SaveThis");
			}
		}

		@Command
		public void cancel() {
			if (recordMode.equals(RecordMode.READ)
					|| (StringUtils.isBlank(dirty.getSrc()))) {
				goBack();
				return;
			}
			if (StringUtils.isNotBlank(dirty.getSrc())) {
				Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
						Consts.UNSAVED_TITLE, this,
						MessageBoxConfirmType.USAVED_CONFIRMATION);
			}
		}

		@Override
		public void onConfirmClick(String code, int button) {
			if (code.equals("unSaved") && button == Messagebox.YES) {
				goBack();
			}
```

```
	}

	public void goBack() {
		container.getChildren().clear();
		win.detach();
		Executions.createComponents(ApplicationLinks.MedicosList,
container,
				callerArg);

	}
}
```

PAQUETE com.product.webapp.admin
CLASE MEDICOSLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TMedico;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
```

```
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class MedicosListVM implements ConfirmResponse {

	@Wire("#medicoslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<TMedico> allReordsInDB = null;
	private List<TMedico> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TMedico selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TMedico.getAllTMedicos");
		filteredRecords = (List<TMedico>)
CollectionUtils.select(allReordsInDB,
				new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
```

```
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(@BindingParam("selectedRecord") TMedico
medico,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(medico);
			allReordsInDB.add(medico);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, medico);
			allReordsInDB.set(this.selectedItemIndex, medico);
		}
	}

	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<TMedico>();
		for (Iterator<TMedico> i = allReordsInDB.iterator(); i.hasNext();)
{
			TMedico tmp = i.next();
			if
(tmp.getPersona().getCedulapersona().toLowerCase().indexOf(dataFilter.getName())
== 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
				}
			}
		}
	}

	@Command
	public void onExcelExport() {

		this.excelColumns = new ArrayList<ExcelColumns>();
		this.excelColumns.add(new ExcelColumns("userName", "User Name"));
		this.excelColumns
				.add(new ExcelColumns("firstName", "First Name",
9600));
		this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
		BeanToExcel beanToExcel = new BeanToExcel();
		beanToExcel.setExcelColumns(excelColumns);
		beanToExcel.setDataSheetName("MedicosList");
		beanToExcel.setDataList(filteredRecords);
		beanToExcel.exportToExcel();
	}
```

```
@Command
public void onDeactivate() {
	ShowWindow.ShowDeactivateWin();
}

@Command
public void onActivate() {
	ShowWindow.ShowActivateWin();
}

// note this will be executed from ActivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void activateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(1);
	this.selectedItem.setActivatedReason(reason);
	crudService.Save(this.selectedItem);
}

// note this will be executed from deactivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void deactivateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(0);
	this.selectedItem.setDeactivatedReason(reason);
	crudService.Save(this.selectedItem);
}

@Command
public void onAddNew() {
	goToCRUD(RecordMode.ADD);
}

@Command
public void openAsReadOnly() {
	goToCRUD(RecordMode.READ);
}

@Command
public void onEdit() {
	goToCRUD(RecordMode.EDIT);
}

@Command
public void onlinkOpen(@BindingParam("record") TMedico record)
		throws Exception, NoSuchFieldException {
	if (menu.getCanEdit() == 1)
		onEdit();
	else
		openAsReadOnly();
}
```

```
	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		container.getChildren().clear();
		ShowWindow.openZulFile(ApplicationLinks.MedicosCRUD, container,
CRUDargs);
	}

	@Command
	public void onDelete() {
		Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
				+ this.selectedItem.getIdmedico() + "\" se
eliminará.?",
				Consts.UNSAVED_TITLE, this,
				MessageBoxConfirmType.DELETE_CONFIRMATION);
	}

	public void deleteSelectedItem() {
		try {
			crudService.delete(selectedItem);
			Infrastructure.showSuccessmessage();
			allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

	filteredRecords.remove(filteredRecords.indexOf(selectedItem));
			BindUtils.postNotifyChange(null, null, this, "dataSet");
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TMedico ",
this.selectedItem,
					"" + ": onConfirmClick");
		}
	}

	@Override
	@NotifyChange("dataSet")
	public void onConfirmClick(String code, int button) {
		if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
			Infrastructure
					.confirm(
							"deleteSecondConfirm",
							"El elemento seleccionado \""
									+
this.selectedItem.getIdmedico()
```

```
                                                                + "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
                                                Consts.UNSAVED_TITLE, this,

        MessageBoxConfirmType.DELETE_CONFIRMATION);
                }
                if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
                        deleteSelectedItem();
                }
        }

        public Window getWin() {
                return win;
        }

        public void setWin(Window win) {
                this.win = win;
        }

        public Textbox getZulName() {
                return zulName;
        }

        public void setZulName(Textbox zulName) {
                this.zulName = zulName;
        }

        public CRUDService getCrudService() {
                return crudService;
        }

        public void setCrudService(CRUDService crudService) {
                this.crudService = crudService;
        }

        public List<ExcelColumns> getExcelColumns() {
                return excelColumns;
        }

        public void setExcelColumns(List<ExcelColumns> excelColumns) {
                this.excelColumns = excelColumns;
        }

        public List<TMedico> getAllReordsInDB() {
                return allReordsInDB;
        }

        public void setAllReordsInDB(List<TMedico> allReordsInDB) {
                this.allReordsInDB = allReordsInDB;
        }
```

```
public List<TMedico> getDataSet() {
	return filteredRecords;
}

public void setFilteredRecords(List<TMedico> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
}

public void setMenu(MenuItems menu) {
	this.menu = menu;
}

public Integer getSelectedItemIndex() {
	return selectedItemIndex;
}

public void setSelectedItemIndex(Integer selectedItemIndex) {
	this.selectedItemIndex = selectedItemIndex;
}

public Integer getPageSize() {
	return pageSize;
}

public void setPageSize(Integer pageSize) {
	this.pageSize = pageSize;
}

public TMedico getSelectedItem() {
	return selectedItem;
}
```

```
	public void setSelectedItem(TMedico selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE PARROQUIASCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TCanton;
import com.product.domain.TParroquia;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
```

```
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;

public class ParroquiasCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#parroquiasCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TParroquia selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";
	private Component container;
	final HashMap<String, Object> callerArg = new HashMap<String, Object>();
	private List<TCanton> cantonList;


	public List<TCanton> getCantonList() {
		return cantonList;
	}

	public void setCantonList(List<TCanton> cantonList) {
		this.cantonList = cantonList;
	}

	public TParroquia getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TParroquia selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}
```

```
	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TParroquia parroquias,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);
		this.cantonList =
crudService.GetListByNamedQuery("TCanton.getAllTCantones");

		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TParroquia();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = parroquias;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = parroquias;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
```

```
			case 27:
				s = "ESC";
				break;
			case 121:
				s = "F10";
				break;
			}
			if (s.equalsIgnoreCase("F10")) {
				if (!this.makeAsReadOnly)
					saveThis(0);
			}
			if (s.equalsIgnoreCase("ESC")) {
				cancel();
			}

		}

		@Command
		public void saveThis(@BindingParam("action") Integer action) {
			if (Libs.IsValidBean(this.selectedRecord) == false) {
				return;
			}
			try {
				crudService.Save(this.selectedRecord);
				Infrastructure.showSuccessmessage();
				goBack();
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "TParroquia ",
this.selectedRecord,
						this.moreErrorInfo + ": SaveThis");
			}
		}

		@Command
		public void cancel() {
			if (recordMode.equals(RecordMode.READ)
					|| (StringUtils.isBlank(dirty.getSrc()))) {
				goBack();
				return;
			}
			if (StringUtils.isNotBlank(dirty.getSrc())) {
				Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
						Consts.UNSAVED_TITLE, this,
						MessageBoxConfirmType.USAVED_CONFIRMATION);
			}
		}

		@Override
		public void onConfirmClick(String code, int button) {
			if (code.equals("unSaved") && button == Messagebox.YES) {
				goBack();
			}
```

```
	}

	public void goBack() {
		container.getChildren().clear();
		win.detach();
		Executions.createComponents(ApplicationLinks.ParroquiasList,
container,
				callerArg);

	}
}
```

PAQUETE com.product.webapp.admin
CLASE PARROQUIASLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TParroquia;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
```

```
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class ParroquiasListVM implements ConfirmResponse {

	@Wire("#parroquiaslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<TParroquia> allReordsInDB = null;
	private List<TParroquia> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TParroquia selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TParroquia.getAllTParroquias");
		filteredRecords = (List<TParroquia>)
CollectionUtils.select(allReordsInDB,
				new PredicateActive());

		zulName.setFocus(true);

	}
```

```
	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(@BindingParam("selectedRecord") TParroquia
parroquias,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(parroquias);
			allReordsInDB.add(parroquias);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, parroquias);
			allReordsInDB.set(this.selectedItemIndex, parroquias);
		}
	}

	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<TParroquia>();
		for (Iterator<TParroquia> i = allReordsInDB.iterator();
i.hasNext();) {
			TParroquia tmp = i.next();
			if
(tmp.getNombreparroquia().toLowerCase().indexOf(dataFilter.getName()) == 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
				}
			}
		}
	}

	@Command
	public void onExcelExport() {

		this.excelColumns = new ArrayList<ExcelColumns>();
		this.excelColumns.add(new ExcelColumns("userName", "User Name"));
		this.excelColumns.add(new ExcelColumns("firstName", "First Name",
9600));
		this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
		BeanToExcel beanToExcel = new BeanToExcel();
		beanToExcel.setExcelColumns(excelColumns);
		beanToExcel.setDataSheetName("UsersList");
		beanToExcel.setDataList(filteredRecords);
		beanToExcel.exportToExcel();
	}

	@Command
```

```
public void onDeactivate() {
	ShowWindow.ShowDeactivateWin();
}

@Command
public void onActivate() {
	ShowWindow.ShowActivateWin();
}

// note this will be executed from ActivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void activateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(1);
	this.selectedItem.setActivatedReason(reason);
	crudService.Save(this.selectedItem);
}

// note this will be executed from deactivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void deactivateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(0);
	this.selectedItem.setDeactivatedReason(reason);
	crudService.Save(this.selectedItem);
}

@Command
public void onAddNew() {
	goToCRUD(RecordMode.ADD);
}

@Command
public void openAsReadOnly() {
	goToCRUD(RecordMode.READ);
}

@Command
public void onEdit() {
	goToCRUD(RecordMode.EDIT);
}

@Command
public void onlinkOpen(@BindingParam("record") TParroquia record)
		throws Exception, NoSuchFieldException {
	if (menu.getCanEdit() == 1)
		onEdit();
	else
		openAsReadOnly();
}

public void goToCRUD(final RecordMode mode) {
```

```
            if (mode.equals(RecordMode.ADD))
                  CRUDargs.put("selectedRecord", null);
            else {
                  CRUDargs.put("selectedRecord", selectedItem);
                  setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
            }
            CRUDargs.put("recordMode", mode);
            CRUDargs.put("container", container);
            CRUDargs.put("MenuDetails", menu);
            container.getChildren().clear();
            ShowWindow.openZulFile(ApplicationLinks.ParroquiasCRUD, container,
CRUDargs);
      }

      @Command
      public void onDelete() {
            Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
                        + this.selectedItem.getIdparroquia() + "\" se
eliminará.?",
                        Consts.UNSAVED_TITLE, this,
                        MessageBoxConfirmType.DELETE_CONFIRMATION);
      }

      public void deleteSelectedItem() {
            try {
                  crudService.delete(selectedItem);
                  Infrastructure.showSuccessmessage();
                  allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

      filteredRecords.remove(filteredRecords.indexOf(selectedItem));
                  BindUtils.postNotifyChange(null, null, this, "dataSet");
            } catch (Exception e) {
                  ExceptionHandler.handleException(e, "Domain ",
this.selectedItem,
                              "" + ": onConfirmClick");
            }
      }

      @Override
      @NotifyChange("dataSet")
      public void onConfirmClick(String code, int button) {
            if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
                  Infrastructure
                              .confirm(
                                          "deleteSecondConfirm",
                                          "El elemento seleccionado  \""
                                                      +
this.selectedItem.getIdparroquia()
                                                      + "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
```

```
                                        Consts.UNSAVED_TITLE, this,
        MessageBoxConfirmType.DELETE_CONFIRMATION);
              }
              if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
                    deleteSelectedItem();
              }
        }

        public Window getWin() {
              return win;
        }

        public void setWin(Window win) {
              this.win = win;
        }

        public Textbox getZulName() {
              return zulName;
        }

        public void setZulName(Textbox zulName) {
              this.zulName = zulName;
        }

        public CRUDService getCrudService() {
              return crudService;
        }

        public void setCrudService(CRUDService crudService) {
              this.crudService = crudService;
        }

        public List<ExcelColumns> getExcelColumns() {
              return excelColumns;
        }

        public void setExcelColumns(List<ExcelColumns> excelColumns) {
              this.excelColumns = excelColumns;
        }

        public List<TParroquia> getAllReordsInDB() {
              return allReordsInDB;
        }

        public void setAllReordsInDB(List<TParroquia> allReordsInDB) {
              this.allReordsInDB = allReordsInDB;
        }

        public List<TParroquia> getDataSet() {
              return filteredRecords;
```

```
}

public void setFilteredRecords(List<TParroquia> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
}

public void setMenu(MenuItems menu) {
	this.menu = menu;
}

public Integer getSelectedItemIndex() {
	return selectedItemIndex;
}

public void setSelectedItemIndex(Integer selectedItemIndex) {
	this.selectedItemIndex = selectedItemIndex;
}

public Integer getPageSize() {
	return pageSize;
}

public void setPageSize(Integer pageSize) {
	this.pageSize = pageSize;
}

public TParroquia getSelectedItem() {
	return selectedItem;
}

public void setSelectedItem(TParroquia selectedItem) {
	this.selectedItem = selectedItem;
```

```
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE PERSONASCRUDVM

```
package com.product.webapp.admin;

import java.io.IOException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.zkoss.bind.BindContext;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.image.AImage;
import org.zkoss.image.Image;
import org.zkoss.util.media.Media;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.event.UploadEvent;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TCanton;
import com.product.domain.TParroquia;
import com.product.domain.TPersona;
```

```
import com.product.domain.TProvincia;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;

//import org.zkoss.zul.Image;

public class PersonasCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;
	@Wire("#personasCRUD")
	private Window win;
	// @Wire("#codetypeCant")
	// private Combobox codetypeCant;
	@Wire("#dirty")
	private Image dirty;
	private TPersona selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";
	private Component container;
	final HashMap<String, Object> callerArg = new HashMap<String, Object>();
	private List<TParroquia> parroquiasList;
	private List<TProvincia> provinciaList;
	private List<TCanton> cantonList;
	public int idprov = 1;
	private AImage myImage;
	private TProvincia selectProvincia = new TProvincia();
	private TCanton selectCanton = new TCanton();

	public TCanton getSelectCanton() {
		return selectCanton;
	}

	public void setSelectCanton(TCanton selectCanton) {
		this.selectCanton = selectCanton;
	}

	public TProvincia getSelectProvincia() {
		return selectProvincia;
	}

	public void setSelectProvincia(TProvincia selectProvincia) {
		this.selectProvincia = selectProvincia;
```

```
}

public List<TProvincia> getProvinciaList() {
	return provinciaList;
}

public void setProvinciaList(List<TProvincia> provinciaList) {
	this.provinciaList = provinciaList;
}

public List<TCanton> getCantonList() {
	return cantonList;
}

public void setCantonList(List<TCanton> cantonList) {
	this.cantonList = cantonList;
}

public List<TParroquia> getParroquiasList() {
	return parroquiasList;
}

public void setParroquiasList(List<TParroquia> parroquiasList) {
	this.parroquiasList = parroquiasList;
}

public TPersona getSelectedRecord() {
	return selectedRecord;
}

public void setSelectedRecord(TPersona selectedRecord) {
	this.selectedRecord = selectedRecord;
}

public RecordMode getRecordMode() {
	return recordMode;
}

public void setRecordMode(RecordMode recordMode) {
	this.recordMode = recordMode;
}

public boolean isMakeAsReadOnly() {
	return makeAsReadOnly;
}

public void setMakeAsReadOnly(boolean makeAsReadOnly) {
	this.makeAsReadOnly = makeAsReadOnly;
}

public int getIdprov() {
	return idprov;
```

```
	}

	public void setIdprov(int idprov) {
		this.idprov = idprov;
	}

	public AImage getMyImage() {
		return myImage;
	}

	public void setMyImage(AImage myImage) {
		this.myImage = myImage;
	}

	@AfterCompose
	@NotifyChange("myImage")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TPersona persona,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container)
			throws IOException {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);

		myImage = new
AImage(Executions.getCurrent().getDesktop().getWebApp()
				.getRealPath("/images")
				+ "/male.png");
		this.provinciaList = crudService
				.GetListByNamedQuery("TProvincia.getAllTProvincias");
		if (mode.equals(RecordMode.ADD)) {
			try {
				this.selectedRecord = new TPersona();
				this.selectedRecord.setSystem(0);
				this.selectedRecord.setActive(1);
				this.selectedRecord.setCreatedBy(FHSessionUtil
						.getLoggedInUserID());
				this.selectedRecord.setCreatedDate(new Timestamp(new
Date()
						.getTime()));

			} catch (Exception e) {
				// TODO: handle exception
			}

		}
```

```
		if (mode.equals(RecordMode.EDIT)) {
			try {
				this.selectedRecord = persona;
				byte[] temp = this.selectedRecord.getFotopaciente();
				this.selectedRecord.setUpdatedBy(FHSessionUtil
						.getLoggedInUserID());
				this.selectedRecord.setUpdatedDate(new Timestamp(new
Date()
						.getTime()));
				if (temp != null)
					myImage = new AImage("userPhoto", temp);
			} catch (Exception e) {
				// TODO: handle exception
				e.printStackTrace();
			}

		}

		if (mode.equals(RecordMode.READ)) {
			try {
				setMakeAsReadOnly(true);
				this.selectedRecord = persona;
				if (this.selectedRecord.getFotopaciente() != null)
					myImage = new AImage("userPhoto",

	this.selectedRecord.getFotopaciente());
			} catch (Exception e) {
				// TODO: handle exception
				if (this.selectedRecord.getFotopaciente() != null)
					myImage = new AImage("userPhoto",

	this.selectedRecord.getFotopaciente());
			}
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
```

```
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}

	}

	@Command
	public void saveThis(@BindingParam("action") Integer action) {

		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {
			if (validacionCedula(this.selectedRecord.getCedulapersona()))
{

				if (myImage != null) {
					byte[] bFile = myImage.getByteData();
					this.selectedRecord.setFotopaciente(bFile);
				} else
					this.selectedRecord.setFotopaciente(null);
				this.selectedRecord.setProvincia(this.selectedRecord

	.getParroquias().getCantonParroquia().getProvCanton());
				this.selectedRecord.setCanton(this.selectedRecord
							.getParroquias().getCantonParroquia());
				crudService.Save(this.selectedRecord);
				Infrastructure.showSuccessmessage();
				goBack();
			}
			else
			{
				Infrastructure.showSuccessmessageValidarCedula();
			}

		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TPersona ",
					this.selectedRecord, this.moreErrorInfo + ":
SaveThis");
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)) {
			goBack();
			return;
		} else {
```

```
                    Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
                                Consts.UNSAVED_TITLE, this,
                                MessageBoxConfirmType.USAVED_CONFIRMATION);
            }
      }

      @Override
      public void onConfirmClick(String code, int button) {
            if (code.equals("unSaved") && button == Messagebox.YES) {
                  goBack();
            }
      }

      public void goBack() {
            // container.getChildren().clear();
            win.detach();
            Executions.createComponents(ApplicationLinks.PersonasList,
container,
                        callerArg);

      }

      @Command
      @NotifyChange("myImage")
      public void removeImage() {

            myImage = null;
      }

      @Command
      @NotifyChange("myImage")
      public void upload(@ContextParam(ContextType.BIND_CONTEXT) BindContext
ctx) {

            UploadEvent upEvent = null;
            Object objUploadEvent = ctx.getTriggerEvent();
            if (objUploadEvent != null && (objUploadEvent instanceof
UploadEvent)) {
                  upEvent = (UploadEvent) objUploadEvent;
            }
            if (upEvent != null) {
                  Media media = upEvent.getMedia();
                  int lengthofImage = media.getByteData().length;
                  if (media instanceof Image) {
                        if (lengthofImage > 500 * 1024) {
                              Messagebox
                                          .show("Please Select a Image of
size less than 500Kb.");
                              return;
                        } else {
                              myImage = (AImage) media;// Initialize the bind
object to
```

```
                                                                                //
show image in zul page and
                                                                                //
Notify it also
                        }
                } else {
                        Messagebox.show("The selected File is not an image.");
                }
        }
    }

    public List<TCanton> retornarCantonesProvincia(int codigoProvincia) {

        List<TCanton> listadoCantonProv = crudService
                .GetListByNamedQuery("TCanton.getAllTCantones");
        List<TCanton> listaCantonesAux = new ArrayList<TCanton>();
        TCanton canton = new TCanton();
        Iterator<TCanton> i = listadoCantonProv.iterator();
        while (i.hasNext()) {
            canton = i.next();
            if (canton.getProvCanton().getIdprovincia() ==
codigoProvincia) {
                listaCantonesAux.add(canton);
            }
        }
        return listaCantonesAux;
    }

    public List<TParroquia> retornarParroquiaCantones(int codigoCanton) {

        List<TParroquia> listadoParroquiaCanton = crudService
                .GetListByNamedQuery("TParroquia.getAllTParroquias");
        List<TParroquia> listaParroquias = new ArrayList<TParroquia>();
        TParroquia parroquias = new TParroquia();
        Iterator<TParroquia> i = listadoParroquiaCanton.iterator();
        while (i.hasNext()) {
            parroquias = i.next();
            if (parroquias.getCantonParroquia().getIdcanton() ==
codigoCanton) {
                listaParroquias.add(parroquias);
            }
        }
        return listaParroquias;
    }

    @Command
    @NotifyChange("cantonList")
    public void ShowSelectedProvincia()

    {
        this.cantonList = retornarCantonesProvincia(this.selectProvincia
                .getIdprovincia());
```

```
	}

	@Command
	@NotifyChange("parroquiasList")
	public void ShowSelectedCantones()

	{
		this.parroquiasList = retornarParroquiaCantones(this.selectCanton
					.getIdcanton());
	}

	// // METODO PARA COMPROBAR LAS CEDULAS
	public Boolean validacionCedula(String cedula) {
		int num_provincias = 24;
		int prov = Integer.parseInt(cedula.substring(0, 2));

		if (!((prov > 0) && (prov <= num_provincias)))
		{
			Messagebox.show("Error: cedula mal ingresada");
			return false;
		}
		int[] d = new int[10];
		for (int i = 0; i < d.length; i++) {
			d[i] = Integer.parseInt(cedula.charAt(i) + "");
		}

		int imp = 0;
		int par = 0;

		for (int i = 0; i < d.length; i += 2) {
			d[i] = ((d[i] * 2) > 9) ? ((d[i] * 2) - 9) : (d[i] * 2);
			imp += d[i];
		}

		for (int i = 1; i < (d.length - 1); i += 2) {
			par += d[i];
		}

		int suma = imp + par;

		int d10 = Integer.parseInt(String.valueOf(suma + 10).substring(0,
1)
					+ "0")
					- suma;

		d10 = (d10 == 10) ? 0 : d10;

		if (d10 == d[9]) {
			return true;
		} else {
			return false;
		}
```

```
	}
}
```

PAQUETE com.product.webapp.admin
CLASE PERSONASLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TPersona;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class PersonasListVM implements ConfirmResponse {
```

```
	@Wire("#personaslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<TPersona> allReordsInDB = null;
	private List<TPersona> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TPersona selectedItem;
	private Textbox name;

	public Textbox getName() {
		return name;
	}

	public void setName(Textbox name) {
		this.name = name;
	}

	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;

	dataFilter.setName(StringUtils.capitalize(FHSessionUtil.getCurrentUser().
getMiddleName()));
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TPersona.getAllTPersonas");
		filteredRecords = (List<TPersona>)
CollectionUtils.select(allReordsInDB, new PredicateActive());

		zulName.setFocus(true);
```

```
	}


	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(
			@BindingParam("selectedRecord") TPersona persona,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(persona);
			allReordsInDB.add(persona);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, persona);
			allReordsInDB.set(this.selectedItemIndex, persona);
		}
	}

	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<TPersona>();
		String Rol =
StringUtils.capitalize(FHSessionUtil.getCurrentUser().getUserRole().getRoleName(
));
		String cedulaPer =
StringUtils.capitalize(FHSessionUtil.getCurrentUser().getMiddleName());

		if (Rol.trim().equals("Paciente"))
		{
			dataFilter.setName(cedulaPer);
			for (Iterator<TPersona> i = allReordsInDB.iterator();
i.hasNext();) {
				TPersona tmp = i.next();
				if
(tmp.getCedulapersona().trim().equals(cedulaPer.trim())) {
					filteredRecords.add(tmp);
					break;
				}
			}
		}

		else {
			for (Iterator<TPersona> i = allReordsInDB.iterator();
i.hasNext();) {
				TPersona tmp = i.next();
				if (tmp.getCedulapersona().toLowerCase()
						.indexOf(dataFilter.getName()) == 0) {
					if (dataFilter.getIsActive().intValue() == 0)
```

```
                                        filteredRecords.add(tmp);
                                    else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
                                        filteredRecords.add(tmp);
                                    }
                                }
                            }
                        }
        }

        @Command
        public void onExcelExport() {

                this.excelColumns = new ArrayList<ExcelColumns>();
                this.excelColumns.add(new ExcelColumns("userName", "User Name"));
                this.excelColumns.add(new ExcelColumns("firstName", "First Name",
9600));
                this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
                BeanToExcel beanToExcel = new BeanToExcel();
                beanToExcel.setExcelColumns(excelColumns);
                beanToExcel.setDataSheetName("PersonasList");
                beanToExcel.setDataList(filteredRecords);
                beanToExcel.exportToExcel();
        }

        @Command
        public void onDeactivate() {
                ShowWindow.ShowDeactivateWin();
        }

        @Command
        public void onActivate() {
                ShowWindow.ShowActivateWin();
        }

        // note this will be executed from ActivateVM.Java
        @GlobalCommand
        @NotifyChange("selectedItem")
        public void activateThis(@BindingParam("reason") String reason) {
                this.selectedItem.setActive(1);
                this.selectedItem.setActivatedReason(reason);
                crudService.Save(this.selectedItem);
        }

        // note this will be executed from deactivateVM.Java
        @GlobalCommand
        @NotifyChange("selectedItem")
        public void deactivateThis(@BindingParam("reason") String reason) {
                this.selectedItem.setActive(0);
                this.selectedItem.setDeactivatedReason(reason);
                crudService.Save(this.selectedItem);
```

```
	}

	@Command
	public void onAddNew() {
		goToCRUD(RecordMode.ADD);
	}

	@Command
	public void openAsReadOnly() {
		goToCRUD(RecordMode.READ);
	}

	@Command
	public void onEdit() {
		goToCRUD(RecordMode.EDIT);
	}

	@Command
	public void onlinkOpen(@BindingParam("record") TPersona record)
			throws Exception, NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		container.getChildren().clear();
		ShowWindow.openZulFile(ApplicationLinks.PersonasCRUD,
container,CRUDargs);
	}

	@Command
	public void onDelete() {
		Infrastructure.confirm(
				"deleteFirstConfirm",
				"El elemento seleccionado  \""
					+ this.selectedItem.getIdpersona()
					+ "\" se eliminará.?",
Consts.UNSAVED_TITLE, this,
				MessageBoxConfirmType.DELETE_CONFIRMATION);
	}
```

```
	public void deleteSelectedItem() {
		try {
			crudService.delete(selectedItem);
			Infrastructure.showSuccessmessage();
			allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

	filteredRecords.remove(filteredRecords.indexOf(selectedItem));
			BindUtils.postNotifyChange(null, null, this, "dataSet");
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TPersona ",
this.selectedItem,
					"" + ": onConfirmClick");
		}
	}

	@Override
	@NotifyChange("dataSet")
	public void onConfirmClick(String code, int button) {
		if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
			Infrastructure
					.confirm(
							"deleteSecondConfirm",
							"El elemento seleccionado \""
									+
this.selectedItem.getIdpersona()
									+ "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
							Consts.UNSAVED_TITLE, this,

	MessageBoxConfirmType.DELETE_CONFIRMATION);
		}
		if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
			deleteSelectedItem();
		}
	}

	public Window getWin() {
		return win;
	}

	public void setWin(Window win) {
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
```

```
}

public CRUDService getCrudService() {
	return crudService;
}

public void setCrudService(CRUDService crudService) {
	this.crudService = crudService;
}

public List<ExcelColumns> getExcelColumns() {
	return excelColumns;
}

public void setExcelColumns(List<ExcelColumns> excelColumns) {
	this.excelColumns = excelColumns;
}

public List<TPersona> getAllReordsInDB() {
	return allReordsInDB;
}

public void setAllReordsInDB(List<TPersona> allReordsInDB) {
	this.allReordsInDB = allReordsInDB;
}

public List<TPersona> getDataSet() {
	return filteredRecords;
}

public void setFilteredRecords(List<TPersona> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
```

```
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public Integer getSelectedItemIndex() {
		return selectedItemIndex;
	}

	public void setSelectedItemIndex(Integer selectedItemIndex) {
		this.selectedItemIndex = selectedItemIndex;
	}

	public Integer getPageSize() {
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TPersona getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TPersona selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE PRECONSULTACRUD

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
```

```
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.Init;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TDetalleHistoriaClinica;
import com.product.domain.TPreConsulta;



public class PreConsultaCrud {

	@Wire("#win")
	private Window win;

	private TDetalleHistoriaClinica texto1;
	private String texto2;

	@WireVariable
	private CRUDService crudService;
	private TPreConsulta selectedPreconsulta;
	private TPreConsulta selectedPreconsultaOrg;
	private List<TDetalleHistoriaClinica> detalleHCList;

	private List<TPreConsulta> preconsulta = new ArrayList<TPreConsulta>();

	private TPreConsulta curSelectedPreconsulta;
	private Integer curSelectedPreconsultaIndex;



	public List<TPreConsulta> getPreconsulta() {
		return preconsulta;
	}

	public void setPreconsulta(List<TPreConsulta> preconsulta) {
		this.preconsulta = preconsulta;
	}

	public TPreConsulta getCurSelectedPreconsulta() {
		return curSelectedPreconsulta;
	}
```

```
	public void setCurSelectedPreconsulta(TPreConsulta
curSelectedPreconsulta) {
		this.curSelectedPreconsulta = curSelectedPreconsulta;
	}

	public Integer getCurSelectedPreconsultaIndex() {
		return curSelectedPreconsultaIndex;
	}

	public void setCurSelectedPreconsultaIndex(Integer
curSelectedPreconsultaIndex) {
		this.curSelectedPreconsultaIndex = curSelectedPreconsultaIndex;
	}

	private boolean makeAsReadOnly;
	private String recordMode;



	public List<TDetalleHistoriaClinica> getDetalleHCList() {
		return detalleHCList;
	}

	public void setDetalleHCList(List<TDetalleHistoriaClinica> detalleHCList)
{
		this.detalleHCList = detalleHCList;
	}

	public String getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(String recordMode) {
		this.recordMode = recordMode;
	}


	public TPreConsulta getSelectedPreconsulta() {
		return selectedPreconsulta;
	}

	public void setSelectedPreconsulta(TPreConsulta selectedPreconsulta) {
		this.selectedPreconsulta = selectedPreconsulta;
	}

	public TPreConsulta getSelectedPreconsultaOrg() {
		return selectedPreconsultaOrg;
	}

	public void setSelectedPreconsultaOrg(TPreConsulta
selectedPreconsultaOrg) {
```

```
			this.selectedPreconsultaOrg = selectedPreconsultaOrg;
		}

		public boolean isMakeAsReadOnly() {
			return makeAsReadOnly;
		}
		public void setMakeAsReadOnly(boolean makeAsReadOnly) {
			this.makeAsReadOnly = makeAsReadOnly;
		}

		public TDetalleHistoriaClinica getTexto1() {
			return texto1;
		}

		public void setTexto1(TDetalleHistoriaClinica texto1) {
			this.texto1 = texto1;
		}

		public String getTexto2() {
			return texto2;
		}

		public void setTexto2(String texto2) {
			this.texto2 = texto2;
		}

		@SuppressWarnings({ "unchecked", "unused"})
		@Init
//		@NotifyChange("selectedDetalle")
		public void initSetup(@ContextParam(ContextType.VIEW) Component view,
				@ExecutionArgParam("sPreconsulta") TPreConsulta
selectedPreconsulta,
				@ExecutionArgParam("recordMode") String recordMode) throws
CloneNotSupportedException {
			Selectors.wireComponents(view, this, false);
			///OBTENGO PARAMETRO DESDE HISTORIACCRUDVM DESDE SESSION
			final HashMap<String, Object> map = (HashMap<String, Object>)
Sessions.getCurrent().getAttribute("allmyvalues");
			crudService = (CRUDService) SpringUtil.getBean("crudService");
			//this.detalleHCList =
crudService.GetListByNamedQuery("TDetalleHistoriaClinica.getAllTDetalle");
			///ASIGNO VALORES DE SESSION
			this.texto1 = (TDetalleHistoriaClinica) map.get("preconsulta");
			this.texto2 = ""+texto1.getIddetallehc();

			setRecordMode(recordMode);

			if(consultarPersona(texto1.getIddetallehc())==null)
				recordMode="NEW";
			else
				recordMode="EDIT";
```

```
		if (recordMode.equals("NEW")) {
			this.selectedPreconsulta = new TPreConsulta();
		}

		if (recordMode.equals("EDIT")) {
			//System.out.println("Value is " + selectedPreconsulta.);
			List <TPreConsulta> Consulta= new ArrayList<TPreConsulta>();
			selectedPreconsulta =
consultarPersona(texto1.getIddetallehc());
			this.selectedPreconsultaOrg = selectedPreconsulta;
			this.selectedPreconsulta= (TPreConsulta)
selectedPreconsulta.clone();

		}

		if (recordMode == "READ") {
			setMakeAsReadOnly(true);
			win.setTitle(win.getTitle() + " (Readonly)");
		}

	}

	@Command
	public void save() {
		selectedPreconsulta.setActive(1);
		selectedPreconsulta.setSystem(0);
		selectedPreconsulta.setDetalleHC(texto1);
		this.selectedPreconsulta.setFechapreconsulta(new Date());
		crudService.Save(selectedPreconsulta);
		closeThis() ;
}


	@Command
	public void closeThis() {
		win.detach();
	}

	@Command
	public void editThisPreconsulta() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("texto1", this.curSelectedPreconsulta);

	setCurSelectedPreconsultaIndex(preconsulta.indexOf(curSelectedPreconsulta
));
		map.put("recordMode", "EDIT");
		Executions.createComponents("/zk/master/preconsulta.zul", null,
map);
	}


	////////CONSULTAR SI EXISTE EL REGISTRO DE PRECONSULTA
```

```
	public TPreConsulta  consultarPersona(int idDetalle)
	{
		List<TPreConsulta> pconsultaAux =
crudService.GetListByNamedQuery("TPreConsulta.getAllTPreConsulta");
		TPreConsulta precon= new TPreConsulta();
		TPreConsulta preconAux= new TPreConsulta();
		Iterator<TPreConsulta> i = pconsultaAux.iterator();
		while (i.hasNext()) {
			precon = i.next();
			if (precon.getDetalleHC().getIddetallehc() == idDetalle) {
				return preconAux=precon;
			}
		}
		return preconAux;
	}
}
```

PAQUETE com.product.webapp.admin
CLASE PROVINCIASCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TProvincia;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
```

```
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;
public class ProvinciasCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#provinciasCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TProvincia selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";

	public TProvincia getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TProvincia selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TProvincia provincias,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
```

```
		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TProvincia();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = provincias;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = provincias;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}

	}

	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
```

```
                        return;
                }
                try {
                        crudService.Save(this.selectedRecord);
                        final HashMap<String, Object> map = new HashMap<String,
Object>();
                        map.put("selectedRecord", selectedRecord);
                        map.put("recordMode", recordMode);
                        BindUtils.postGlobalCommand(null, null,
"TProvinciaListVMRefresh", map);
                        Infrastructure.showSuccessmessage();
                        closeThis();
                } catch (Exception e) {
                        ExceptionHandler.handleException(e, "TProvincia ",
this.selectedRecord,
                                        this.moreErrorInfo + ": SaveThis");
                }
        }

        @Command
        public void cancel() {
                if (recordMode.equals(RecordMode.READ)
                                || (StringUtils.isBlank(dirty.getSrc()))) {
                        closeThis();
                        return;
                }
                if (StringUtils.isNotBlank(dirty.getSrc())) {
                        Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
                                        Consts.UNSAVED_TITLE, this,
                                        MessageBoxConfirmType.USAVED_CONFIRMATION);
                }
        }

        @Command
        public void closeThis() {
                win.detach();
        }

        @Override
        public void onConfirmClick(String code, int button) {
                if (code.equals("unSaved") && button == Messagebox.YES) {
                        closeThis();
                }

        }
}
```

PAQUETE com.product.webapp.admin
CLASE PROVINCIASLISTVM

```
package com.product.webapp.admin;
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TProvincia;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class ProvinciasListVM implements ConfirmResponse {

	@Wire("#provinciaslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;
```

```
	private List<ExcelColumns> excelColumns = null;
	private List<TProvincia> allReordsInDB = null;
	private List<TProvincia> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TProvincia selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TProvincia.getAllTProvincias");
		filteredRecords = (List<TProvincia>)
CollectionUtils.select(allReordsInDB,new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(@BindingParam("selectedRecord") TProvincia
provincia,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(provincia);
			allReordsInDB.add(provincia);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, provincia);
			allReordsInDB.set(this.selectedItemIndex, provincia);
		}
	}

	@Command
	@NotifyChange("dataSet")
```

```
	public void doFilter() {
		filteredRecords = new ArrayList<TProvincia>();
		for (Iterator<TProvincia> i = allReordsInDB.iterator();
i.hasNext();) {
			TProvincia tmp = i.next();
			if
(tmp.getNombreprovincia().toLowerCase().indexOf(dataFilter.getName()) == 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
				}
			}
		}
	}

	@Command
	public void onExcelExport() {

		this.excelColumns = new ArrayList<ExcelColumns>();
		this.excelColumns.add(new ExcelColumns("userName", "User Name"));
		this.excelColumns
				.add(new ExcelColumns("firstName", "First Name",
9600));
		this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
		BeanToExcel beanToExcel = new BeanToExcel();
		beanToExcel.setExcelColumns(excelColumns);
		beanToExcel.setDataSheetName("TProvinciaList");
		beanToExcel.setDataList(filteredRecords);
		beanToExcel.exportToExcel();
	}

	@Command
	public void onDeactivate() {
		ShowWindow.ShowDeactivateWin();
	}

	@Command
	public void onActivate() {
		ShowWindow.ShowActivateWin();
	}

	// note this will be executed from ActivateVM.Java
	@GlobalCommand
	@NotifyChange("selectedItem")
	public void activateThis(@BindingParam("reason") String reason) {
		this.selectedItem.setActive(1);
		this.selectedItem.setActivatedReason(reason);
		crudService.Save(this.selectedItem);
	}
```

```
	// note this will be executed from deactivateVM.Java
	@GlobalCommand
	@NotifyChange("selectedItem")
	public void deactivateThis(@BindingParam("reason") String reason) {
		this.selectedItem.setActive(0);
		this.selectedItem.setDeactivatedReason(reason);
		crudService.Save(this.selectedItem);
	}

	@Command
	public void onAddNew() {
		goToCRUD(RecordMode.ADD);
	}

	@Command
	public void openAsReadOnly() {
		goToCRUD(RecordMode.READ);
	}

	@Command
	public void onEdit() {
		goToCRUD(RecordMode.EDIT);
	}

	@Command
	public void onlinkOpen(@BindingParam("record") TProvincia record)
			throws Exception, NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		container.getChildren().clear();
		ShowWindow.openZulFile(ApplicationLinks.ProvinciasCRUD, container,
CRUDargs);
	}

	@Command
	public void onDelete() {
```

```
			Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
					+ this.selectedItem.getIdprovincia() + "\" se
eliminará.?",
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.DELETE_CONFIRMATION);
		}

		public void deleteSelectedItem() {
			try {
				crudService.delete(selectedItem);
				Infrastructure.showSuccessmessage();
				allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

		filteredRecords.remove(filteredRecords.indexOf(selectedItem));
				BindUtils.postNotifyChange(null, null, this, "dataSet");
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "TProvincia ",
this.selectedItem,
						"" + ": onConfirmClick");
			}
		}

		@Override
		@NotifyChange("dataSet")
		public void onConfirmClick(String code, int button) {
			if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
				Infrastructure
						.confirm(
								"deleteSecondConfirm",
								"El elemento seleccionado  \""
										+
this.selectedItem.getIdprovincia()
										+ "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
								Consts.UNSAVED_TITLE, this,

		MessageBoxConfirmType.DELETE_CONFIRMATION);
			}
			if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
				deleteSelectedItem();
			}
		}

		public Window getWin() {
			return win;
		}

		public void setWin(Window win) {
			this.win = win;
```

```
}

public Textbox getZulName() {
	return zulName;
}

public void setZulName(Textbox zulName) {
	this.zulName = zulName;
}

public CRUDService getCrudService() {
	return crudService;
}

public void setCrudService(CRUDService crudService) {
	this.crudService = crudService;
}

public List<ExcelColumns> getExcelColumns() {
	return excelColumns;
}

public void setExcelColumns(List<ExcelColumns> excelColumns) {
	this.excelColumns = excelColumns;
}

public List<TProvincia> getAllReordsInDB() {
	return allReordsInDB;
}

public void setAllReordsInDB(List<TProvincia> allReordsInDB) {
	this.allReordsInDB = allReordsInDB;
}

public List<TProvincia> getDataSet() {
	return filteredRecords;
}

public void setFilteredRecords(List<TProvincia> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
```

```
	}

	public void setDataFilter(DataFilter dataFilter) {
		this.dataFilter = dataFilter;
	}

	public MenuItems getMenu() {
		return menu;
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public Integer getSelectedItemIndex() {
		return selectedItemIndex;
	}

	public void setSelectedItemIndex(Integer selectedItemIndex) {
		this.selectedItemIndex = selectedItemIndex;
	}

	public Integer getPageSize() {
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TProvincia getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TProvincia selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}


}
```

PAQUETE com.product.webapp.admin
CLASE RECETALIST

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;

import com.product.business.service.CRUDService;
import com.product.domain.TRecetas;



public class RecetaList {

	private List<TRecetas> recetas= new ArrayList<TRecetas>();
	private TRecetas curSelectedRecetas;
	private Integer curSelectedRecetastIndex;

	@WireVariable
	protected CRUDService crudService;


	public List<TRecetas> getRecetas() {
		return recetas;
	}

	public void setRecetas(List<TRecetas> recetas) {
		this.recetas = recetas;
	}

	public TRecetas getCurSelectedRecetas() {
		return curSelectedRecetas;
	}

	public void setCurSelectedRecetas(TRecetas curSelectedRecetas) {
		this.curSelectedRecetas = curSelectedRecetas;
	}

	public Integer getCurSelectedRecetastIndex() {
```

```
		return curSelectedRecetastIndex;
	}

	public void setCurSelectedRecetastIndex(Integer curSelectedRecetastIndex)
{
		this.curSelectedRecetastIndex = curSelectedRecetastIndex;
	}

	public List<TRecetas> getallRecetas() {
		return recetas;
	}

	@Init
	public void initSetup() {

		crudService = (CRUDService) SpringUtil.getBean("crudService");
		recetas=
crudService.GetListByNamedQuery("TRecetas.getAllTRecetas");

	}

	@Command
	public void addNewReceta() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sRecetas", null);
		map.put("recordMode", "NEW");
		Executions.createComponents("/zk/master/recetasCrud.zul", null,
map);
	}

	@Command
	public void editThisRecetas()
	{
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sRecetas", this.curSelectedRecetas);
		map.put("recordMode", "EDIT");
		setCurSelectedRecetastIndex(recetas.indexOf(curSelectedRecetas));
		Executions.createComponents("/zk/master/recetas.zul", null, map);
	}



	@Command
	public void openAsReadOnly()
	{
		recetas.get(this.curSelectedRecetastIndex);

	}

	//The following method will be called from DepartmentCRUDVM.java after
the save
	@GlobalCommand
```

```
	@NotifyChange("allRecetas")
	public void updateRecetasInfo(TRecetas r1,
			@BindingParam("recordMode") String recordMode) {

		if (recordMode.equals("EDIT")) {
			recetas.set(this.curSelectedRecetastIndex, r1);
		}

		if (recordMode.equals("NEW")) {
			recetas.add(r1);
		}
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void deleteThisRecetas()
	{
		int OkCancel;

		String str = "El elemento seleccionado  \"" +
curSelectedRecetas.getIdreceta()
				+ "\" se eliminará";
		OkCancel = Messagebox.show(str, "Confirm", Messagebox.OK
				| Messagebox.CANCEL, Messagebox.QUESTION);
		if (OkCancel == Messagebox.CANCEL) {
			return;
		}

		str = "El elemento \""
				+ curSelectedRecetas.getIdreceta()
				+ "\" Se eliminará de forma permanente y la acción no
se puede deshacer.";

		Messagebox.show(str, "Confirm", Messagebox.OK | Messagebox.CANCEL,
				Messagebox.QUESTION, new EventListener() {
					@Override
					public void onEvent(Event event) throws
Exception {
						if (((Integer)
event.getData()).intValue() == Messagebox.OK) {

	crudService.delete(curSelectedRecetas);

	recetas.remove(recetas.indexOf(curSelectedRecetastIndex));
							BindUtils.postNotifyChange(null,
null,
								RecetaList.this,
"allRecetas");
						}
					}
				});
	}
```

```
}
```

PAQUETE com.product.webapp.admin
CLASE RECETASCRUD

```
package com.product.webapp.admin;

import java.sql.Connection;
import java.sql.DriverManager;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.view.JasperViewer;

import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.Init;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.Sessions;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TDetalleHistoriaClinica;
import com.product.domain.TDetalleReceta;
import com.product.domain.TRecetas;
import com.product.webapp.utilities.Infrastructure;
```

```
public class RecetasCrud {

	@Wire("#recetasCrud")
	private Window win;
	private Connection con;
	@WireVariable
	private CRUDService crudService;
	private boolean makeAsReadOnly;
	private TRecetas selectedTRecetas;
	private List<TDetalleReceta> detalleRecetas = new
ArrayList<TDetalleReceta>();
	private TDetalleReceta curSelectedDetalleReceta;
	private Integer curSelectedDetalleRecetaIndex;
	private String recordMode;
	private TDetalleHistoriaClinica texto1;
	private String texto2;

	private List<TDetalleHistoriaClinica> detalleHCList;

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public TRecetas getSelectedTRecetas() {
		return selectedTRecetas;
	}

	public void setSelectedTRecetas(TRecetas selectedTRecetas) {
		this.selectedTRecetas = selectedTRecetas;
	}

	public List<TDetalleReceta> getallDetalleRecetas() {
		return detalleRecetas;
	}

	public List<TDetalleReceta> getDetalleRecetas() {
		return detalleRecetas;
	}

	public void setDetalleRecetas(List<TDetalleReceta> detalleRecetas) {
		this.detalleRecetas = detalleRecetas;
	}

	public TDetalleReceta getCurSelectedDetalleReceta() {
		return curSelectedDetalleReceta;
	}

	public void setCurSelectedDetalleReceta(
```

```
                  TDetalleReceta curSelectedDetalleReceta) {
            this.curSelectedDetalleReceta = curSelectedDetalleReceta;
      }

      public Integer getCurSelectedDetalleRecetaIndex() {
            return curSelectedDetalleRecetaIndex;
      }

      public void setCurSelectedDetalleRecetaIndex(
                  Integer curSelectedDetalleRecetaIndex) {
            this.curSelectedDetalleRecetaIndex = curSelectedDetalleRecetaIndex;
      }

      public List<TDetalleHistoriaClinica> getDetalleHCList() {
            return detalleHCList;
      }

      public void setDetalleHCList(List<TDetalleHistoriaClinica> detalleHCList)
{
            this.detalleHCList = detalleHCList;
      }

      public String getRecordMode() {
            return recordMode;
      }

      public void setRecordMode(String recordMode) {
            this.recordMode = recordMode;
      }

      public TDetalleHistoriaClinica getTexto1() {
            return texto1;
      }

      public void setTexto1(TDetalleHistoriaClinica texto1) {
            this.texto1 = texto1;
      }

      public String getTexto2() {
            return texto2;
      }

      public void setTexto2(String texto2) {
            this.texto2 = texto2;
      }

      public boolean isMakeAsReadOnly() {
            return makeAsReadOnly;
      }

      public void setMakeAsReadOnly(boolean makeAsReadOnly) {
            this.makeAsReadOnly = makeAsReadOnly;
```

```
	}

	@SuppressWarnings({ "unchecked" })
	@Init
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("sRecetas") TRecetas r1,
			@ExecutionArgParam("recordMode") String recordMode) throws
CloneNotSupportedException {
		Selectors.wireComponents(view, this, false);
		// /OBTENGO PARAMETRO DESDE HISTORIACCRUDVM DESDE SESSION
		final HashMap<String, Object> map = (HashMap<String, Object>)
Sessions
				.getCurrent().getAttribute("allmyRecetas");
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		// /ASIGNO VALORES DE SESSION
		this.texto1 = (TDetalleHistoriaClinica) map.get("Recetas");
		this.texto2 = "" + texto1.getIddetallehc();
		setRecordMode(recordMode);

		if (consultarPersonaReceta(texto1.getIddetallehc()) == null)
			recordMode = "NEW";
		else
			recordMode = "EDIT";

		if (recordMode.equals("NEW")) {
			this.selectedTRecetas = new TRecetas();

		}

		if (recordMode.equals("EDIT")) {
			this.selectedTRecetas =
consultarPersonaReceta(texto1.getIddetallehc());//(TRecetas) r1.clone();
			setDetalleRecetas(selectedTRecetas.getTDetalleRecetas());
		}

		if (recordMode == "READ") {
			setMakeAsReadOnly(true);
			win.setTitle(win.getTitle() + " (Readonly)");
		}

	}

	@Command
	public void addNewDetalleRecetas() {

		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sDetalleRecetas", null);
		map.put("recordMode", "NEW");
		Executions.createComponents("/zk/master/detallesRecetas.zul", null,
map);
	}
```

```
	@Command
	public void editThisDetalleRecetas() {
		final HashMap<String, Object> map = new HashMap<String, Object>();
		map.put("sDetalleRecetas", this.curSelectedDetalleReceta);
		setCurSelectedDetalleRecetaIndex(detalleRecetas
				.indexOf(curSelectedDetalleReceta));
		map.put("recordMode", "EDIT");
		Executions
				.createComponents("/zk/master/detallesRecetas.zul",
null, map);
	}

	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void deleteThisDetalleRecetas() {

		String str = "El \""
				+ this.curSelectedDetalleReceta.getIddetallereceta()
				+ "\" se eliminará de forma permanente y la acción no
se puede deshacer.";

		Messagebox.show(str, "Confirm", Messagebox.OK | Messagebox.CANCEL,
				Messagebox.QUESTION, new EventListener() {
					@Override
					public void onEvent(Event event) throws
Exception {
						if (((Integer)
event.getData()).intValue() == Messagebox.OK) {

	detalleRecetas.remove(curSelectedDetalleReceta);
							BindUtils.postNotifyChange(null,
null,
								RecetasCrud.this,
"allDetalleHC");
						}
					}
				});
	}

	@Command
	public void save() {
		selectedTRecetas.setTDetalleRecetas(detalleRecetas);
		selectedTRecetas.setActive(1);
		selectedTRecetas.setSystem(0);
		selectedTRecetas.setCreatedDate(new Date());
		selectedTRecetas.setTDetalleHistoriaClinica(this.texto1);
		this.selectedTRecetas.setFecha(new Date());
		crudService.Save(this.selectedTRecetas);
		Infrastructure.showSuccessmessage();
		//closeThis();
	}
```

```
	@Command
	public void closeThis() {
		win.focus();
		win.detach();
	}

	@GlobalCommand
	@NotifyChange("allDetalleRecetas")
	public void updateRecetasInfo(
			@BindingParam("pDetalleRecetas") TDetalleReceta dR1,
			@BindingParam("recordMode") String recordMode) {

		if (recordMode.equals("EDIT")) {
			dR1.setTRecetas(selectedTRecetas);
			detalleRecetas.set(this.curSelectedDetalleRecetaIndex, dR1);
		}

		if (recordMode.equals("NEW")) {
			dR1.setTRecetas(selectedTRecetas);
			detalleRecetas.add(dR1);

		}
	}

	// //////CONSULTAR SI EXISTE EL REGISTRO DE PRECONSULTA
	public TRecetas consultarPersonaReceta(int idDetalle) {
		List<TRecetas> pconsultaAux = crudService
				.GetListByNamedQuery("TRecetas.getAllTRecetas");
		TRecetas precon = new TRecetas();
		TRecetas preconAux = new TRecetas();
		Iterator<TRecetas> i = pconsultaAux.iterator();
		while (i.hasNext()) {
			precon = i.next();
			if (precon.getTDetalleHistoriaClinica().getIddetallehc() ==
idDetalle) {
				return preconAux = precon;
			}
		}
		return preconAux;
	}
	@SuppressWarnings({ "unchecked", "rawtypes" })
	@Command
	public void imprimirReceta() throws JRException {

		try {
			Class.forName("org.postgresql.Driver");
			con =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/sismed",
"postgres","root");
		} catch (Exception e) {
			e.printStackTrace();
		}
```

```
            try {
                        HashMap parametros = new HashMap();
                        int idReceta= Integer.parseInt(getTexto2());
                        String reportFile =
Executions.getCurrent().getDesktop().getWebApp().getRealPath("/Reportes");
                              reportFile = reportFile + "/Receta.jrxml";
                              parametros.put("idreceta", idReceta);
                        JasperReport jas =
JasperCompileManager.compileReport(reportFile);
                        JasperPrint jasprint =
JasperFillManager.fillReport(jas,parametros, con);
                        JasperViewer.viewReport(jasprint, false);
                        con.close();
            } catch (Exception e) {
                  e.printStackTrace();
            }

      }

}
```

PAQUETE com.product.webapp.admin
CLASE ROLECRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.Roles;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
```

```
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;
public class RoleCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#roleCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private Roles selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";

	public Roles getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(Roles selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") Roles roles,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
```

```
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");

		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new Roles();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = roles;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = roles;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}

	}
```

```
	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {
			crudService.Save(this.selectedRecord);
			final HashMap<String, Object> map = new HashMap<String,
Object>();
			map.put("selectedRecord", selectedRecord);
			map.put("recordMode", recordMode);
			BindUtils.postGlobalCommand(null, null, "RolesListVMRefresh",
map);
			Infrastructure.showSuccessmessage();
			closeThis();
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "Role ",
this.selectedRecord,
					this.moreErrorInfo + ": SaveThis");
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)
				|| (StringUtils.isBlank(dirty.getSrc()))) {
			closeThis();
			return;
		}
		if (StringUtils.isNotBlank(dirty.getSrc())) {
			Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.USAVED_CONFIRMATION);
		}
	}

	@Command
	public void closeThis() {
		win.detach();
	}

	@Override
	public void onConfirmClick(String code, int button) {
		if (code.equals("unSaved") && button == Messagebox.YES) {
			closeThis();
		}

	}
}
```

PAQUETE com.product.webapp.admin
CLASE ROLEPERMISSIONCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import org.apache.commons.collections.BidiMap;
import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.collections.bidimap.DualHashBidiMap;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.event.SelectEvent;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Checkbox;
import org.zkoss.zul.Tree;
import org.zkoss.zul.TreeModel;
import org.zkoss.zul.Treeitem;
import org.zkoss.zul.Window;

import com.product.business.service.AuthService;
import com.product.domain.AppMenu;
import com.product.domain.Roles;
import com.product.domain.RolesMenu;
import com.product.webapp.utilities.AppMenuChildPredicate;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.RoleTreeMode;
import com.product.webapp.utilities.RoleTreeModel;
import com.product.webapp.utilities.ZKTreeUtil;

@SuppressWarnings("rawtypes")
public class RolePermissionCRUDVM {
```

```
	static final Logger LOG =
LoggerFactory.getLogger(RolePermissionCRUDVM.class);

	@Wire("#rolepermissionCRUD")
	private Window win;

	@Wire("#mytree")
	private Tree mytree;

	@Wire("#updateUsers")
	private Checkbox updateUsersChkBox;

	@WireVariable
	private AuthService AuthService;
	private List<AppMenu> allAppMenus = null;

	private List<RolesMenu> allRolesMenu = null;
	private Roles selectedRole;
	private String moreErrorInfo = "";
	private Set<RoleTreeMode> SelectedItems = new HashSet<RoleTreeMode>();
	private Set<RoleTreeMode> unSelectedItems = new HashSet<RoleTreeMode>();
	private String existingRoleMenus = "";
	private String existingRoleCRUDs = "";
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	BidiMap bidiMap = new DualHashBidiMap();
	BidiMap existingUnSelectedRoleLevelMenus = new DualHashBidiMap();

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	public Roles getSelectedRole() {
		return selectedRole;
	}

	public void setSelectedRole(Roles selectedRole) {
		this.selectedRole = selectedRole;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}
```

```
	TreeModel _model;

	public TreeModel getModel() {
		if (_model == null) {
			RoleTreeModel a = new RoleTreeModel(getRoot());
			a.setMultiple(true);
			_model = a;
		}
		return _model;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") Roles selectedRole,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("container") Component container) {

		Selectors.wireComponents(view, this, false);
		moreErrorInfo = this.getClass().getName();
		mytree.setMultiple(true);
		this.recordMode = mode;

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			win.setTitle(win.getTitle() + "(Read Only)");
		}

		this.selectedRole = selectedRole;
		AuthService = (AuthService) SpringUtil.getBean("AuthService");

		allAppMenus =
AuthService.GetListByNamedQuery("AppMenu.getAllMenus");
		storeExistingMenusForThisRole();
		mytree.setFocus(true);
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
```

```
                if (!this.makeAsReadOnly)
                        save();
                }
                if (s.equalsIgnoreCase("ESC")) {
                        closeThis();
                }

        }

        @SuppressWarnings("unchecked")
        public RoleTreeMode getRoot() {
                RoleTreeMode superRoot = new RoleTreeMode(null, "Menu", null);
                RoleTreeMode root = new RoleTreeMode(null, "Menu", null);
                superRoot.appendChild(root);
                for (Iterator<AppMenu> i = allAppMenus.iterator(); i.hasNext();) {

                        AppMenu tmp = i.next();

                        if (tmp.getParentAppMenuID() == null) {
                                RoleTreeMode firstLevelNode = new RoleTreeMode(root,
                                                tmp.getMenuCaption(), tmp);

                                List<AppMenu> childs;
                                childs = ((List<AppMenu>)
CollectionUtils.select(allAppMenus,
                                                new
AppMenuChildPredicate(tmp.getAppMenuID())));
                                getChildren(firstLevelNode, childs);
                                root.appendChild(firstLevelNode);

                        }
                }
                return superRoot;
        }

        @SuppressWarnings("unchecked")
        public void getChildren(RoleTreeMode parent, List<AppMenu> childs) {

                for (Iterator<AppMenu> i = childs.iterator(); i.hasNext();) {
                        AppMenu tmp = i.next();
                        RoleTreeMode firstLevelNode = new RoleTreeMode(parent,
                                        tmp.getMenuCaption(), tmp);

                        List<AppMenu> subchilds;
                        subchilds = ((List<AppMenu>)
CollectionUtils.select(allAppMenus,
                                        new AppMenuChildPredicate(tmp.getAppMenuID())));

                        if (subchilds.size() > 0) {
                                getChildren(firstLevelNode, subchilds);
                        } else {
                                addCRUDOperationNode(firstLevelNode);
```

```
                }
                parent.appendChild(firstLevelNode);
            }
        }

        public void addCRUDOperationNode(RoleTreeMode Parent) {

            if (Parent.getAppMenu().getCanAdd() == 1) {
                //RoleTreeMode CRUD = new RoleTreeMode(Parent, "Can Add",
null);
                RoleTreeMode CRUD = new RoleTreeMode(Parent, "Añadir", null);
                CRUD.setCRUDOperation(true);
                Parent.appendChild(CRUD);
            }

            if (Parent.getAppMenu().getCanEdit() == 1) {
                //RoleTreeMode CRUD = new RoleTreeMode(Parent, "Can Edit",
null);
                RoleTreeMode CRUD = new RoleTreeMode(Parent, "Editar", null);
                CRUD.setCRUDOperation(true);
                Parent.appendChild(CRUD);
            }

            if (Parent.getAppMenu().getCanDelete() == 1) {
                //RoleTreeMode CRUD = new RoleTreeMode(Parent, "Can Delete",
null);
                RoleTreeMode CRUD = new RoleTreeMode(Parent, "Eliminar",
null);
                CRUD.setCRUDOperation(true);
                Parent.appendChild(CRUD);
            }

            if (Parent.getAppMenu().getCanView() == 1) {
                //RoleTreeMode CRUD = new RoleTreeMode(Parent, "Can View",
null);
                RoleTreeMode CRUD = new RoleTreeMode(Parent, "Vizualizar",
null);
                CRUD.setCRUDOperation(true);
                Parent.appendChild(CRUD);
            }

        }

        @Command
        public void onFulfill() {
            ZKTreeUtil.doCollapseExpandAll(mytree, true);
            SetSelectItems(mytree);
            ZKTreeUtil.doCollapseExpandAll(mytree, false);
            ZKTreeUtil.expandOneLevels(mytree);
            Treeitem treeItem = mytree.getItems().iterator().next();
            if (StringUtils.isNotBlank(existingRoleMenus))
                treeItem.setSelected(true);
```

```
	}

	@Command
	public void save() {

		SelectedItems = new HashSet<RoleTreeMode>();
		unSelectedItems = new HashSet<RoleTreeMode>();
		getSelectItems(mytree, true);

		LOG.debug(" No of SelectedItems {}", SelectedItems.size());
		LOG.debug(" No of Un SelectedItems {}", unSelectedItems.size());

		deleteEntries();
		List<Long> newMenuIDs = new ArrayList<Long>();
		if (SelectedItems.size() == 0) {
			Infrastructure.showSuccessmessage();
			closeThis();
		}
		for (Iterator<RoleTreeMode> i = SelectedItems.iterator();
i.hasNext();) {
			RoleTreeMode tmp = i.next();

			Boolean roleMenuExists = false;
			RolesMenu existingMenu = null;
			String s1 = "";
			String s2 = "";

			if (tmp.getAppMenu() == null)
				continue;

			if (StringUtils.containsIgnoreCase(existingRoleMenus, "Mnu"
					+ tmp.getAppMenu().getAppMenuID())) {
				roleMenuExists = true;
				existingMenu = (RolesMenu) bidiMap.get("Mnu"
						+ tmp.getAppMenu().getAppMenuID());
				if (existingMenu == null)
					roleMenuExists = false;
			}
			try {
				RolesMenu p1;
				if (roleMenuExists == false) {
					p1 = new RolesMenu();

	p1.setPrAppMenuID(tmp.getAppMenu().getAppMenuID());
					p1.setRole(selectedRole);
					p1.setCreatedDate(new Timestamp(new
Date().getTime()));

	p1.setCreatedBy(FHSessionUtil.getLoggedInUserID());
					p1.setCanAdd(tmp.getCanAdd());
					p1.setCanEdit(tmp.getCanEdit());
					p1.setCanDelete(tmp.getCanDelete());
```

```
                                p1.setCanView(tmp.getCanView());
                                AuthService.Save(p1);
                                newMenuIDs.add(tmp.getAppMenu().getAppMenuID());
                          }
                          if (roleMenuExists == true) {
                                s1 = "MenuID : " +
tmp.getAppMenu().getAppMenuID()
                                            + ", CRUD : " +
tmp.getCanAdd().toString()
                                            + tmp.getCanEdit().toString()
                                            + tmp.getCanDelete().toString() +
tmp.getCanView();
                                s2 = "MenuID : " + existingMenu.getPrAppMenuID()
                                            + ", CRUD : " +
existingMenu.getCanAdd().toString()
                                            +
existingMenu.getCanEdit().toString()
                                            +
existingMenu.getCanDelete().toString()
                                            + existingMenu.getCanView();

                                p1 = existingMenu;
                                p1.setUpdatedDate(new Timestamp(new
Date().getTime()));

      p1.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
                                p1.setCanAdd(tmp.getCanAdd());
                                p1.setCanEdit(tmp.getCanEdit());
                                p1.setCanDelete(tmp.getCanDelete());
                                p1.setCanView(tmp.getCanView());

                                if (StringUtils.equalsIgnoreCase(s1, s2) ==
false) {

                                      LOG.debug(
                                                  " Existing Menu in the Role
Level , But Some CRUD Operation has been Changed. Changed {}",
                                                  s1);
                                      LOG.debug(
                                                  " Existing Menu in the Role
Level, But Some CRUD Operation has been Changed. Existing {}",
                                                  s2);
                                      AuthService.Save(p1);
                                }
                          }

                          Infrastructure.showSuccessmessage();
                          closeThis();
                    } catch (Exception e) {
                          ExceptionHandler.handleException(e, "Role Rights ",
                                      selectedRole, moreErrorInfo + ": save");
                    }
```

```
			}
		}

		public void deleteEntries() {

			List<Long> itemIDs = new ArrayList<Long>();

			LOG.debug("existingUnSelectedRoleLevelMenus "
					+ existingUnSelectedRoleLevelMenus);
			LOG.debug("Currently unselected Items " + unSelectedItems);
			for (Iterator<RoleTreeMode> i = unSelectedItems.iterator();
i.hasNext();) {
				RoleTreeMode tmp = i.next();
				if (existingUnSelectedRoleLevelMenus.get(tmp.getAppMenu()
						.getAppMenuID()) == null)
					itemIDs.add(tmp.getAppMenu().getAppMenuID());
			}

			if (itemIDs.size() == 0) {
				LOG.debug(" Nothing to Delete in deleteEntries");
				return;
			}
			if (itemIDs.size() > 0) {

				try {
					LOG.debug("Some Menus are unselected, so going to
delete {}",
							itemIDs);

					AuthService.deleteRolesMenu(itemIDs,
selectedRole.getID());

					LOG.debug(
							"UnSelected Menus are Removed from
practiceroleappmenu {}",
							itemIDs);

					if (SelectedItems.size() == 0) {
						// If all the rights for the roles removed, then
remove the
						// user
						// role mapping for this role. All the users
mapped under
						// this
						// role will be removed.

		AuthService.deleteRolesMenu(selectedRole.getID());
						LOG.debug("Since there are no rights available
for this Role, so User Role Mapping has been removed");
					}

				} catch (Exception e) {
```

```
                        ExceptionHandler.handleException(e, "Role Rights ",
                                    selectedRole, moreErrorInfo + ":
deleteEntries");
                  }
            }
      }

      @Command
      public void onSelect(@ContextParam(ContextType.TRIGGER_EVENT) SelectEvent
a) {
            Treeitem treeItem = (Treeitem) a.getReference();
            if (this.recordMode.equals(RecordMode.READ)) {
                  if (treeItem.isSelected())
                        treeItem.setSelected(false);
                  else
                        treeItem.setSelected(true);
                  return;
            }
            ZKTreeUtil.onSelect(treeItem);
      }

      private void getSelectItems(Component component, boolean aufklappen) {
            if (component instanceof Treeitem) {

                  Treeitem treeitem = (Treeitem) component;
                  if (treeitem.isSelected() == true)
                        SelectedItems.add((RoleTreeMode) treeitem.getValue());
                  else {
                        if (((RoleTreeMode) treeitem.getValue()).getAppMenu()
!= null)
                              unSelectedItems.add((RoleTreeMode)
treeitem.getValue());
                  }
                  RoleTreeMode m1 = treeitem.getValue();
                  if (m1.isCRUDOperation() == true)
                        SetCRUDOperation(treeitem, treeitem.isSelected());

            }
            Collection<?> com = component.getChildren();
            if (com != null) {
                  for (Iterator<?> iterator = com.iterator();
iterator.hasNext();) {
                        getSelectItems((Component) iterator.next(),
aufklappen);
                  }
            }

      }

      private void SetCRUDOperation(Treeitem treeitem, boolean isSelected) {
            Integer val;
```

```
            if (isSelected == true)
                  val = 1;
            else
                  val = 0;

            if (treeitem.getLabel().equalsIgnoreCase("Can Add"))
                  ((RoleTreeMode)
treeitem.getParentItem().getValue()).setCanAdd(val);

            if (treeitem.getLabel().equalsIgnoreCase("Can Edit"))
                  ((RoleTreeMode) treeitem.getParentItem().getValue())
                              .setCanEdit(val);

            if (treeitem.getLabel().equalsIgnoreCase("Can Delete"))
                  ((RoleTreeMode) treeitem.getParentItem().getValue())
                              .setCanDelete(val);

            if (treeitem.getLabel().equalsIgnoreCase("Can View"))
                  ((RoleTreeMode) treeitem.getParentItem().getValue())
                              .setCanView(val);

      }

      private void SetSelectItems(Component component) {
            if (component instanceof Treeitem) {
                  Treeitem treeitem = (Treeitem) component;
                  RoleTreeMode m1 = treeitem.getValue();
                  if (m1.getAppMenu() != null) {
                        RolesMenu existingMenu = null;
                        existingMenu = (RolesMenu) bidiMap.get("Mnu"
                                    + m1.getAppMenu().getAppMenuID());
                        if (existingMenu != null) {
                              treeitem.setSelected(true);
                        } else

      existingUnSelectedRoleLevelMenus.put(m1.getAppMenu()
                                          .getAppMenuID(), m1.getAppMenu()
                                          .getAppMenuID());
                  } else
                        SetCheckedForCRUDItems(m1, treeitem);
            }
            Collection<?> com = component.getChildren();
            if (com != null) {
                  for (Iterator<?> iterator = com.iterator();
iterator.hasNext();) {
                        SetSelectItems((Component) iterator.next());

                  }
            }
      }

      private void SetCheckedForCRUDItems(RoleTreeMode m1, Treeitem treeitem) {
```

```
			if (m1.isCRUDOperation() == true) {
				String toCheck;
				toCheck = "CRUD:"
						+ m1.getParent().getAppMenu().getAppMenuID() +
":"
						+ m1.getName();
				if (StringUtils.containsIgnoreCase(existingRoleCRUDs,
toCheck))
					treeitem.setSelected(true);
			}
		}

		public void storeExistingMenusForThisRole() {
			allRolesMenu = AuthService
					.GetListByNamedQuery(
							"RolesMenu.getAllMenusForRole",
							new Object[] { selectedRole.getID() });

			for (Iterator<RolesMenu> i = allRolesMenu.iterator(); i.hasNext();)
{
				RolesMenu tmp = i.next();
				existingRoleMenus = existingRoleMenus + "," + "Mnu"
						+ tmp.getPrAppMenuID();

				bidiMap.put("Mnu" + tmp.getPrAppMenuID(), tmp);

				if (tmp.getCanAdd() == 1)
					existingRoleCRUDs = existingRoleCRUDs + "," + "CRUD:"
							+ tmp.getPrAppMenuID() + ":Can Add";

				if (tmp.getCanEdit() == 1)
					existingRoleCRUDs = existingRoleCRUDs + "," + "CRUD:"
							+ tmp.getPrAppMenuID() + ":Can Edit";

				if (tmp.getCanDelete() == 1)
					existingRoleCRUDs = existingRoleCRUDs + "," + "CRUD:"
							+ tmp.getPrAppMenuID() + ":Can Delete";

				if (tmp.getCanView() == 1)
					existingRoleCRUDs = existingRoleCRUDs + "," + "CRUD:"
							+ tmp.getPrAppMenuID() + ":Can View";

			}
		}

		@Command
		public void closeThis() {
			win.detach();
		}

}
```

PAQUETE com.product.webapp.admin
CLASE ROLESLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.Roles;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class RolesListVM implements ConfirmResponse {

	@Wire("#roleslist")
	private Window win;
```

```
	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<Roles> allReordsInDB = null;
	private List<Roles> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private Roles selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("Roles.getAllRoles");
		filteredRecords = (List<Roles>)
CollectionUtils.select(allReordsInDB,
				new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void RolesListVMRefresh(@BindingParam("selectedRecord") Roles
roles,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(roles);
			allReordsInDB.add(roles);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, roles);
			allReordsInDB.set(this.selectedItemIndex, roles);
```

```
			}
		}

		@Command
		@NotifyChange("dataSet")
		public void doFilter() {
			filteredRecords = new ArrayList<Roles>();
			for (Iterator<Roles> i = allReordsInDB.iterator(); i.hasNext();) {
				Roles tmp = i.next();
				if
(tmp.getRoleName().toLowerCase().indexOf(dataFilter.getName()) == 0) {
					if (dataFilter.getIsActive().intValue() == 0)
						filteredRecords.add(tmp);
					else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
						filteredRecords.add(tmp);
					}
				}
			}
		}

		@Command
		public void onExcelExport() {

			this.excelColumns = new ArrayList<ExcelColumns>();
			this.excelColumns.add(new ExcelColumns("roleName", "Role Name"));
			this.excelColumns.add(new ExcelColumns("roleDesc", "Role Desc",
9600));
			BeanToExcel beanToExcel = new BeanToExcel();
			beanToExcel.setExcelColumns(excelColumns);
			beanToExcel.setDataSheetName("RolesList");
			beanToExcel.setDataList(filteredRecords);
			beanToExcel.exportToExcel();
		}

		@Command
		public void onDeactivate() {
			ShowWindow.ShowDeactivateWin();
		}

		@Command
		public void onActivate() {
			ShowWindow.ShowActivateWin();
		}

		// note this will be executed from ActivateVM.Java
		@GlobalCommand
		@NotifyChange("selectedItem")
		public void activateThis(@BindingParam("reason") String reason) {
			this.selectedItem.setActive(1);
			this.selectedItem.setActivatedReason(reason);
			crudService.Save(this.selectedItem);
```

```
	}

	// note this will be executed from deactivateVM.Java
	@GlobalCommand
	@NotifyChange("selectedItem")
	public void deactivateThis(@BindingParam("reason") String reason) {
		this.selectedItem.setActive(0);
		this.selectedItem.setDeactivatedReason(reason);
		crudService.Save(this.selectedItem);
	}

	@Command
	public void onAddNew() {
		goToCRUD(RecordMode.ADD);
	}

	@Command
	public void openAsReadOnly() {
		goToCRUD(RecordMode.READ);
	}

	@Command
	public void onEdit() {
		goToCRUD(RecordMode.EDIT);
	}

	@Command
	public void onlinkOpen(@BindingParam("record") Roles record) throws
Exception,
			NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		ShowWindow.openZulFile(ApplicationLinks.RoleCRUD, this.win,
CRUDargs);
	}

	@Command
	public void onDelete() {
```

```
			Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
					+ this.selectedItem.getRoleName() + "\" se
eliminará.?",
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.DELETE_CONFIRMATION);
		}

		public void deleteSelectedItem() {
			try {
				crudService.delete(selectedItem);
				Infrastructure.showSuccessmessage();
				allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

		filteredRecords.remove(filteredRecords.indexOf(selectedItem));
				BindUtils.postNotifyChange(null, null, this, "dataSet");
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "Role",
this.selectedItem,
						"" + ": onConfirmClick");
			}
		}

		@Override
		@NotifyChange("dataSet")
		public void onConfirmClick(String code, int button) {
			if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
					Infrastructure
							.confirm(
									"deleteSecondConfirm",
									"El elemento seleccionado
\""
											+
this.selectedItem.getRoleName()
											+ "\" Se
eliminará de forma permanente y la acción no se puede deshacer..?",
									Consts.UNSAVED_TITLE, this,

		MessageBoxConfirmType.DELETE_CONFIRMATION);
				}
				if (code.equals("deleteSecondConfirm") && button ==
Messagebox.YES) {
					deleteSelectedItem();
				}
		}

		public Window getWin() {
			return win;
		}

		public void setWin(Window win) {
```

```
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
		this.excelColumns = excelColumns;
	}

	public List<Roles> getAllReordsInDB() {
		return allReordsInDB;
	}

	public void setAllReordsInDB(List<Roles> allReordsInDB) {
		this.allReordsInDB = allReordsInDB;
	}

	public List<Roles> getDataSet() {
		return filteredRecords;
	}

	public void setFilteredRecords(List<Roles> filteredRecords) {
		this.filteredRecords = filteredRecords;
	}

	public Component getContainer() {
		return container;
	}

	public void setContainer(Component container) {
		this.container = container;
	}

	public DataFilter getDataFilter() {
```

```
		return dataFilter;
	}

	public void setDataFilter(DataFilter dataFilter) {
		this.dataFilter = dataFilter;
	}

	public MenuItems getMenu() {
		return menu;
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public Integer getSelectedItemIndex() {
		return selectedItemIndex;
	}

	public void setSelectedItemIndex(Integer selectedItemIndex) {
		this.selectedItemIndex = selectedItemIndex;
	}

	public Integer getPageSize() {
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public Roles getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(Roles selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE ROLESPERMISSIONLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.Roles;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class RolesPermissionListVM  {

	@Wire("#rolepermissionlist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;
	private List<Roles> allReordsInDB = null;
	private List<Roles> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer pageSize;
	private Roles selectedItem;
```

```
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);

		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("Roles.getAllRoles");
		filteredRecords = (List<Roles>)
CollectionUtils.select(allReordsInDB,
				new PredicateActive());
		zulName.setFocus(true);

	}


	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<Roles>();
		for (Iterator<Roles> i = allReordsInDB.iterator(); i.hasNext();) {
			Roles tmp = i.next();
			if
(tmp.getRoleName().toLowerCase().indexOf(dataFilter.getName()) == 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
				}
			}
		}
	}




	@Command
	public void openAsReadOnly() {
		goToCRUD(RecordMode.READ);
	}
```

```
	@Command
	public void onEdit() {
		goToCRUD(RecordMode.EDIT);
	}

	@Command
	public void onlinkOpen(@BindingParam("record") Roles record) throws
Exception,
			NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {

		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		ShowWindow.openZulFile(ApplicationLinks.RolePermissionCRUD,
this.win, CRUDargs);
	}



	public Window getWin() {
		return win;
	}

	public void setWin(Window win) {
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}
```

```
public void setCrudService(CRUDService crudService) {
	this.crudService = crudService;
}


public List<Roles> getAllReordsInDB() {
	return allReordsInDB;
}

public void setAllReordsInDB(List<Roles> allReordsInDB) {
	this.allReordsInDB = allReordsInDB;
}

public List<Roles> getDataSet() {
	return filteredRecords;
}

public void setFilteredRecords(List<Roles> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
}

public void setMenu(MenuItems menu) {
	this.menu = menu;
}


public Integer getPageSize() {
	return pageSize;
}

public void setPageSize(Integer pageSize) {
	this.pageSize = pageSize;
```

```
	}

	public Roles getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(Roles selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE TIPOENFERMEDADESCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TEnfermedad;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
```

```
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;
public class TipoEnfermedadesCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#tipoenfermedadesCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TEnfermedad selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";



	public TEnfermedad getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TEnfermedad selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TEnfermedad
tipoenfermedades,
			@ExecutionArgParam("recordMode") RecordMode mode,
```

```
                  @ExecutionArgParam("container") Component container) {
            Selectors.wireComponents(view, this, false);
            setRecordMode(mode);
            moreErrorInfo = this.getClass().getName();
            crudService = (CRUDService) SpringUtil.getBean("crudService");

            if (mode.equals(RecordMode.ADD)) {
                  this.selectedRecord = new TEnfermedad();
                  this.selectedRecord.setSystem(0);
                  this.selectedRecord.setActive(1);

      this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
                  this.selectedRecord.setCreatedDate(new Timestamp(new Date()
                              .getTime()));
            }

            if (mode.equals(RecordMode.EDIT)) {
                  this.selectedRecord = tipoenfermedades;

      this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
                  this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
                              .getTime()));
            }

            if (mode.equals(RecordMode.READ)) {
                  setMakeAsReadOnly(true);
                  this.selectedRecord = tipoenfermedades;
            }
      }

      @Command
      public void doCtrlKeyAction(
                  @org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

            int keyCode = Integer.parseInt(ctrlKeyCode);
            String s = "";
            switch (keyCode) {
            case 27:
                  s = "ESC";
                  break;
            case 121:
                  s = "F10";
                  break;
            }
            if (s.equalsIgnoreCase("F10")) {
                  if (!this.makeAsReadOnly)
                        saveThis(0);
            }
            if (s.equalsIgnoreCase("ESC")) {
                  cancel();
            }
```

```
	}

	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {
			crudService.Save(this.selectedRecord);
			final HashMap<String, Object> map = new HashMap<String,
Object>();
			map.put("selectedRecord", selectedRecord);
			map.put("recordMode", recordMode);
			BindUtils.postGlobalCommand(null, null,
"TipoEnfermedadesListVMRefresh", map);
			Infrastructure.showSuccessmessage();
			closeThis();
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TEnfermedades ",
this.selectedRecord,
					this.moreErrorInfo + ": SaveThis");
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)
				|| (StringUtils.isBlank(dirty.getSrc()))) {
			closeThis();
			return;
		}
		if (StringUtils.isNotBlank(dirty.getSrc())) {
			Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.USAVED_CONFIRMATION);
		}
	}

	@Command
	public void closeThis() {
		win.detach();
	}

	@Override
	public void onConfirmClick(String code, int button) {
		if (code.equals("unSaved") && button == Messagebox.YES) {
			closeThis();
		}

	}
```

```
}
```

PAQUETE com.product.webapp.admin
CLASE TIPOENFERMEDDADESLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TEnfermedad;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class TipoEnfermedadesListVM implements ConfirmResponse {

	@Wire("#tipoenfermedadeslist")
```

```
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;


	private List<ExcelColumns> excelColumns = null;
	private List<TEnfermedad> allReordsInDB = null;
	private List<TEnfermedad> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TEnfermedad selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TEnfermedad.getAllTEnfermedad");
		filteredRecords = (List<TEnfermedad>)
CollectionUtils.select(allReordsInDB,new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(@BindingParam("selectedRecord")
TEnfermedad tipoenfermedades,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(tipoenfermedades);
			allReordsInDB.add(tipoenfermedades);
		}
		if (mode.equals(RecordMode.EDIT)) {
```

```
                    filteredRecords.set(this.selectedItemIndex,
tipoenfermedades);
                    allReordsInDB.set(this.selectedItemIndex, tipoenfermedades);
             }
      }

      @Command
      @NotifyChange("dataSet")
      public void doFilter() {
             filteredRecords = new ArrayList<TEnfermedad>();
             for (Iterator<TEnfermedad> i = allReordsInDB.iterator();
i.hasNext();) {
                    TEnfermedad tmp = i.next();
                    if
(tmp.getNombreenfermedad().toLowerCase().indexOf(dataFilter.getName()) == 0) {
                           if (dataFilter.getIsActive().intValue() == 0)
                                  filteredRecords.add(tmp);
                           else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
                                  filteredRecords.add(tmp);
                           }
                    }
             }
      }

      @Command
      public void onExcelExport() {

             this.excelColumns = new ArrayList<ExcelColumns>();
             this.excelColumns.add(new ExcelColumns("userName", "User Name"));
             this.excelColumns
                           .add(new ExcelColumns("firstName", "First Name",
9600));
             this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
             BeanToExcel beanToExcel = new BeanToExcel();
             beanToExcel.setExcelColumns(excelColumns);
             beanToExcel.setDataSheetName("TTipoMedicamentosList");
             beanToExcel.setDataList(filteredRecords);
             beanToExcel.exportToExcel();
      }

      @Command
      public void onDeactivate() {
             ShowWindow.ShowDeactivateWin();
      }

      @Command
      public void onActivate() {
             ShowWindow.ShowActivateWin();
      }
```

```
// note this will be executed from ActivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void activateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(1);
	this.selectedItem.setActivatedReason(reason);
	crudService.Save(this.selectedItem);
}

// note this will be executed from deactivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void deactivateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(0);
	this.selectedItem.setDeactivatedReason(reason);
	crudService.Save(this.selectedItem);
}

@Command
public void onAddNew() {
	goToCRUD(RecordMode.ADD);
}

@Command
public void openAsReadOnly() {
	goToCRUD(RecordMode.READ);
}

@Command
public void onEdit() {
	goToCRUD(RecordMode.EDIT);
}

@Command
public void onlinkOpen(@BindingParam("record") TEnfermedad record)
		throws Exception, NoSuchFieldException {
	if (menu.getCanEdit() == 1)
		onEdit();
	else
		openAsReadOnly();
}

public void goToCRUD(final RecordMode mode) {
	if (mode.equals(RecordMode.ADD))
		CRUDargs.put("selectedRecord", null);
	else {
		CRUDargs.put("selectedRecord", selectedItem);
		setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
	}
	CRUDargs.put("recordMode", mode);
	CRUDargs.put("container", container);
	CRUDargs.put("MenuDetails", menu);
```

```
            container.getChildren().clear();
            ShowWindow.openZulFile(ApplicationLinks.TipoEnfermedadesCRUD,
container, CRUDargs);
      }

      @Command
      public void onDelete() {
            Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
                        + this.selectedItem.getIdenfermedad() + "\" se
eliminará.?",
                        Consts.UNSAVED_TITLE, this,
                        MessageBoxConfirmType.DELETE_CONFIRMATION);
      }

      public void deleteSelectedItem() {
            try {
                  crudService.delete(selectedItem);
                  Infrastructure.showSuccessmessage();
                  allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

      filteredRecords.remove(filteredRecords.indexOf(selectedItem));
                  BindUtils.postNotifyChange(null, null, this, "dataSet");
            } catch (Exception e) {
                  ExceptionHandler.handleException(e, "TEnfermedad ",
this.selectedItem,
                              "" + ": onConfirmClick");
            }
      }

      @Override
      @NotifyChange("dataSet")
      public void onConfirmClick(String code, int button) {
            if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
                  Infrastructure
                              .confirm(
                                          "deleteSecondConfirm",
                                          "El elemento seleccionado  \""
                                                      +
this.selectedItem.getIdenfermedad()
                                                      + "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
                                          Consts.UNSAVED_TITLE, this,

      MessageBoxConfirmType.DELETE_CONFIRMATION);
            }
            if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
                  deleteSelectedItem();
            }
      }
```

```
public Window getWin() {
	return win;
}

public void setWin(Window win) {
	this.win = win;
}

public Textbox getZulName() {
	return zulName;
}

public void setZulName(Textbox zulName) {
	this.zulName = zulName;
}

public CRUDService getCrudService() {
	return crudService;
}

public void setCrudService(CRUDService crudService) {
	this.crudService = crudService;
}

public List<ExcelColumns> getExcelColumns() {
	return excelColumns;
}

public void setExcelColumns(List<ExcelColumns> excelColumns) {
	this.excelColumns = excelColumns;
}

public List<TEnfermedad> getAllReordsInDB() {
	return allReordsInDB;
}

public void setAllReordsInDB(List<TEnfermedad> allReordsInDB) {
	this.allReordsInDB = allReordsInDB;
}

public List<TEnfermedad> getDataSet() {
	return filteredRecords;
}

public void setFilteredRecords(List<TEnfermedad> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}
```

```
public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
}

public void setMenu(MenuItems menu) {
	this.menu = menu;
}

public Integer getSelectedItemIndex() {
	return selectedItemIndex;
}

public void setSelectedItemIndex(Integer selectedItemIndex) {
	this.selectedItemIndex = selectedItemIndex;
}

public Integer getPageSize() {
	return pageSize;
}

public void setPageSize(Integer pageSize) {
	this.pageSize = pageSize;
}

public TEnfermedad getSelectedItem() {
	return selectedItem;
}

public void setSelectedItem(TEnfermedad selectedItem) {
	this.selectedItem = selectedItem;
}

public HashMap<String, Object> getCRUDargs() {
	return CRUDargs;
}

public void setCRUDargs(HashMap<String, Object> cRUDargs) {
	CRUDargs = cRUDargs;
}
```

```
}
```

PAQUETE com.product.webapp.admin
CLASE TIPOEXAMENESCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TTipoExamen;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;
public class TipoExamenesCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#tipoexamenesCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TTipoExamen selectedRecord;
```

```
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";

	public TTipoExamen getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TTipoExamen selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TTipoExamen
tipoexamenes,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");

		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TTipoExamen();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = tipoexamenes;
```

```
		this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
				this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
						.getTime()));
			}

			if (mode.equals(RecordMode.READ)) {
				setMakeAsReadOnly(true);
				this.selectedRecord = tipoexamenes;
			}
		}

		@Command
		public void doCtrlKeyAction(
				@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

			int keyCode = Integer.parseInt(ctrlKeyCode);
			String s = "";
			switch (keyCode) {
			case 27:
				s = "ESC";
				break;
			case 121:
				s = "F10";
				break;
			}
			if (s.equalsIgnoreCase("F10")) {
				if (!this.makeAsReadOnly)
					saveThis(0);
			}
			if (s.equalsIgnoreCase("ESC")) {
				cancel();
			}

		}

		@Command
		public void saveThis(@BindingParam("action") Integer action) {
			if (Libs.IsValidBean(this.selectedRecord) == false) {
				return;
			}
			try {
				crudService.Save(this.selectedRecord);
				final HashMap<String, Object> map = new HashMap<String,
Object>();
				map.put("selectedRecord", selectedRecord);
				map.put("recordMode", recordMode);
				BindUtils.postGlobalCommand(null, null,
"TipoExamenesListVMRefresh", map);
				Infrastructure.showSuccessmessage();
				closeThis();
```

```
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "TTipoExamen ",
this.selectedRecord,
					this.moreErrorInfo + ": SaveThis");
			}
		}

		@Command
		public void cancel() {
			if (recordMode.equals(RecordMode.READ)
					|| (StringUtils.isBlank(dirty.getSrc()))) {
				closeThis();
				return;
			}
			if (StringUtils.isNotBlank(dirty.getSrc())) {
				Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.USAVED_CONFIRMATION);
			}
		}

		@Command
		public void closeThis() {
			win.detach();
		}

		@Override
		public void onConfirmClick(String code, int button) {
			if (code.equals("unSaved") && button == Messagebox.YES) {
				closeThis();
			}

		}
}
```

PAQUETE com.product.webapp.admin
CLASE TIPOEXAMENESLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
```

```
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TTipoExamen;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class TipoExamenesListVM implements ConfirmResponse {

	@Wire("#tipoexameneslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<TTipoExamen> allReordsInDB = null;
	private List<TTipoExamen> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TTipoExamen selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();
```

```
	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TTipoExamen.getAllTipoExamenes");
		filteredRecords = (List<TTipoExamen>)
CollectionUtils.select(allReordsInDB,
				new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void RolesListVMRefresh(@BindingParam("selectedRecord")
TTipoExamen tipoexamenes,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(tipoexamenes);
			allReordsInDB.add(tipoexamenes);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex, tipoexamenes);
			allReordsInDB.set(this.selectedItemIndex, tipoexamenes);
		}
	}

	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<TTipoExamen>();
		for (Iterator<TTipoExamen> i = allReordsInDB.iterator();
i.hasNext();) {
			TTipoExamen tmp = i.next();
			if
(tmp.getNombreexamen().toLowerCase().indexOf(dataFilter.getName()) == 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
```

```
                        }
                    }
                }
        }

        @Command
        public void onExcelExport() {

                this.excelColumns = new ArrayList<ExcelColumns>();
                this.excelColumns.add(new ExcelColumns("roleName", "Role Name"));
                this.excelColumns.add(new ExcelColumns("roleDesc", "Role Desc",
9600));
                BeanToExcel beanToExcel = new BeanToExcel();
                beanToExcel.setExcelColumns(excelColumns);
                beanToExcel.setDataSheetName("TipoExamenesList");
                beanToExcel.setDataList(filteredRecords);
                beanToExcel.exportToExcel();
        }

        @Command
        public void onDeactivate() {
                ShowWindow.ShowDeactivateWin();
        }

        @Command
        public void onActivate() {
                ShowWindow.ShowActivateWin();
        }

        // note this will be executed from ActivateVM.Java
        @GlobalCommand
        @NotifyChange("selectedItem")
        public void activateThis(@BindingParam("reason") String reason) {
                this.selectedItem.setActive(1);
                this.selectedItem.setActivatedReason(reason);
                crudService.Save(this.selectedItem);
        }

        // note this will be executed from deactivateVM.Java
        @GlobalCommand
        @NotifyChange("selectedItem")
        public void deactivateThis(@BindingParam("reason") String reason) {
                this.selectedItem.setActive(0);
                this.selectedItem.setDeactivatedReason(reason);
                crudService.Save(this.selectedItem);
        }

        @Command
        public void onAddNew() {
                goToCRUD(RecordMode.ADD);
        }
```

```
	@Command
	public void openAsReadOnly() {
		goToCRUD(RecordMode.READ);
	}

	@Command
	public void onEdit() {
		goToCRUD(RecordMode.EDIT);
	}

	@Command
	public void onlinkOpen(@BindingParam("record") TTipoExamen record) throws
Exception,
			NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		ShowWindow.openZulFile(ApplicationLinks.TipoExamenesCRUD, this.win,
CRUDargs);
	}

	@Command
	public void onDelete() {
		Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
				+ this.selectedItem.getIdtipoexamen() + "\" se
eliminará.?",
				Consts.UNSAVED_TITLE, this,
				MessageBoxConfirmType.DELETE_CONFIRMATION);
	}

	public void deleteSelectedItem() {
		try {
			crudService.delete(selectedItem);
			Infrastructure.showSuccessmessage();
			allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

	filteredRecords.remove(filteredRecords.indexOf(selectedItem));
			BindUtils.postNotifyChange(null, null, this, "dataSet");
```

```
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "TTipoExamen",
this.selectedItem,
					"" + ": onConfirmClick");
			}
		}

		@Override
		@NotifyChange("dataSet")
		public void onConfirmClick(String code, int button) {
			if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
				Infrastructure
						.confirm(
								"deleteSecondConfirm",
								"El elemento seleccionado
\""
+
this.selectedItem.getIdtipoexamen()
+ "\" Se
eliminará de forma permanente y la acción no se puede deshacer..?",
Consts.UNSAVED_TITLE, this,

		MessageBoxConfirmType.DELETE_CONFIRMATION);
			}
			if (code.equals("deleteSecondConfirm") && button ==
Messagebox.YES) {
				deleteSelectedItem();
			}
		}

		public Window getWin() {
			return win;
		}

		public void setWin(Window win) {
			this.win = win;
		}

		public Textbox getZulName() {
			return zulName;
		}

		public void setZulName(Textbox zulName) {
			this.zulName = zulName;
		}

		public CRUDService getCrudService() {
			return crudService;
		}

		public void setCrudService(CRUDService crudService) {
```

```
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
		this.excelColumns = excelColumns;
	}

	public List<TTipoExamen> getAllReordsInDB() {
		return allReordsInDB;
	}

	public void setAllReordsInDB(List<TTipoExamen> allReordsInDB) {
		this.allReordsInDB = allReordsInDB;
	}

	public List<TTipoExamen> getDataSet() {
		return filteredRecords;
	}

	public void setFilteredRecords(List<TTipoExamen> filteredRecords) {
		this.filteredRecords = filteredRecords;
	}

	public Component getContainer() {
		return container;
	}

	public void setContainer(Component container) {
		this.container = container;
	}

	public DataFilter getDataFilter() {
		return dataFilter;
	}

	public void setDataFilter(DataFilter dataFilter) {
		this.dataFilter = dataFilter;
	}

	public MenuItems getMenu() {
		return menu;
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public Integer getSelectedItemIndex() {
```

```
		return selectedItemIndex;
	}

	public void setSelectedItemIndex(Integer selectedItemIndex) {
		this.selectedItemIndex = selectedItemIndex;
	}

	public Integer getPageSize() {
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TTipoExamen getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TTipoExamen selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE TIPOMEDICAMENTOSCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.zk.ui.Component;
```

```
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TTipoMedicamentos;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;
public class TipoMedicamentosCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#tipomedicamentosCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TTipoMedicamentos selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";

	public TTipoMedicamentos getSelectedRecord() {
		return selectedRecord;
	}

	public void setSelectedRecord(TTipoMedicamentos selectedRecord) {
		this.selectedRecord = selectedRecord;
	}

	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
```

```
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TTipoMedicamentos
tipomedicamentos,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("container") Component container) {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");

		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TTipoMedicamentos();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = tipomedicamentos;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}

		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = tipomedicamentos;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
```

```
			case 121:
				s = "F10";
				break;
			}
			if (s.equalsIgnoreCase("F10")) {
				if (!this.makeAsReadOnly)
					saveThis(0);
			}
			if (s.equalsIgnoreCase("ESC")) {
				cancel();
			}

		}

		@Command
		public void saveThis(@BindingParam("action") Integer action) {
			if (Libs.IsValidBean(this.selectedRecord) == false) {
				return;
			}
			try {
				crudService.Save(this.selectedRecord);
				final HashMap<String, Object> map = new HashMap<String,
Object>();
				map.put("selectedRecord", selectedRecord);
				map.put("recordMode", recordMode);
				BindUtils.postGlobalCommand(null, null,
"TTipoMedicamentosListVMRefresh", map);
				Infrastructure.showSuccessmessage();
				closeThis();
			} catch (Exception e) {
				ExceptionHandler.handleException(e, "TTipoMedicamentos ",
this.selectedRecord,
						this.moreErrorInfo + ": SaveThis");
			}
		}

		@Command
		public void cancel() {
			if (recordMode.equals(RecordMode.READ)
					|| (StringUtils.isBlank(dirty.getSrc()))) {
				closeThis();
				return;
			}
			if (StringUtils.isNotBlank(dirty.getSrc())) {
				Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
						Consts.UNSAVED_TITLE, this,
						MessageBoxConfirmType.USAVED_CONFIRMATION);
			}
		}

		@Command
		public void closeThis() {
```

```
			win.detach();
		}

		@Override
		public void onConfirmClick(String code, int button) {
			if (code.equals("unSaved") && button == Messagebox.YES) {
				closeThis();
			}

		}


}
```

PAQUETE com.product.webapp.admin
CLASE TIPOMEDICAMENTOSLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TTipoMedicamentos;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
```

```
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class TipoMedicamentosListVM implements ConfirmResponse {

	@Wire("#tipomedicamentoslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;


	private List<ExcelColumns> excelColumns = null;
	private List<TTipoMedicamentos> allReordsInDB = null;
	private List<TTipoMedicamentos> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TTipoMedicamentos selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TTipoMedicamentos.getAllTiposMedicamentos");
		filteredRecords = (List<TTipoMedicamentos>)
CollectionUtils.select(allReordsInDB,new PredicateActive());

		zulName.setFocus(true);

	}
```

```
	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(@BindingParam("selectedRecord")
TTipoMedicamentos tipomedicamentos,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(tipomedicamentos);
			allReordsInDB.add(tipomedicamentos);
		}
		if (mode.equals(RecordMode.EDIT)) {
			filteredRecords.set(this.selectedItemIndex,
tipomedicamentos);
			allReordsInDB.set(this.selectedItemIndex, tipomedicamentos);
		}
	}

	@Command
	@NotifyChange("dataSet")
	public void doFilter() {
		filteredRecords = new ArrayList<TTipoMedicamentos>();
		for (Iterator<TTipoMedicamentos> i = allReordsInDB.iterator();
i.hasNext();) {
			TTipoMedicamentos tmp = i.next();
			if
(tmp.getDescripcion().toLowerCase().indexOf(dataFilter.getName()) == 0) {
				if (dataFilter.getIsActive().intValue() == 0)
					filteredRecords.add(tmp);
				else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
					filteredRecords.add(tmp);
				}
			}
		}
	}

	@Command
	public void onExcelExport() {

		this.excelColumns = new ArrayList<ExcelColumns>();
		this.excelColumns.add(new ExcelColumns("userName", "User Name"));
		this.excelColumns
				.add(new ExcelColumns("firstName", "First Name",
9600));
		this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
		BeanToExcel beanToExcel = new BeanToExcel();
		beanToExcel.setExcelColumns(excelColumns);
		beanToExcel.setDataSheetName("TTipoMedicamentosList");
		beanToExcel.setDataList(filteredRecords);
		beanToExcel.exportToExcel();
```

```
}

@Command
public void onDeactivate() {
	ShowWindow.ShowDeactivateWin();
}

@Command
public void onActivate() {
	ShowWindow.ShowActivateWin();
}

// note this will be executed from ActivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void activateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(1);
	this.selectedItem.setActivatedReason(reason);
	crudService.Save(this.selectedItem);
}

// note this will be executed from deactivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void deactivateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(0);
	this.selectedItem.setDeactivatedReason(reason);
	crudService.Save(this.selectedItem);
}

@Command
public void onAddNew() {
	goToCRUD(RecordMode.ADD);
}

@Command
public void openAsReadOnly() {
	goToCRUD(RecordMode.READ);
}

@Command
public void onEdit() {
	goToCRUD(RecordMode.EDIT);
}

@Command
public void onlinkOpen(@BindingParam("record") TTipoMedicamentos record)
		throws Exception, NoSuchFieldException {
	if (menu.getCanEdit() == 1)
		onEdit();
	else
		openAsReadOnly();
```

```
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		container.getChildren().clear();
		ShowWindow.openZulFile(ApplicationLinks.TipoMedicamentosCRUD,
container, CRUDargs);
	}

	@Command
	public void onDelete() {
		Infrastructure.confirm("deleteFirstConfirm", "El elemento
seleccionado  \""
				+ this.selectedItem.getIdtipomedicamento() + "\" se
eliminará.?",
				Consts.UNSAVED_TITLE, this,
				MessageBoxConfirmType.DELETE_CONFIRMATION);
	}

	public void deleteSelectedItem() {
		try {
			crudService.delete(selectedItem);
			Infrastructure.showSuccessmessage();
			allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

	filteredRecords.remove(filteredRecords.indexOf(selectedItem));
			BindUtils.postNotifyChange(null, null, this, "dataSet");
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TTipoMedicamentos ",
this.selectedItem,
					"" + ": onConfirmClick");
		}
	}

	@Override
	@NotifyChange("dataSet")
	public void onConfirmClick(String code, int button) {
		if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
			Infrastructure
					.confirm(
							"deleteSecondConfirm",
							"El elemento seleccionado  \""
```

```
                                                                +
this.selectedItem.getIdtipomedicamento()
                                                                + "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
                                                Consts.UNSAVED_TITLE, this,

        MessageBoxConfirmType.DELETE_CONFIRMATION);
                }
                if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
                        deleteSelectedItem();
                }
        }

        public Window getWin() {
                return win;
        }

        public void setWin(Window win) {
                this.win = win;
        }

        public Textbox getZulName() {
                return zulName;
        }

        public void setZulName(Textbox zulName) {
                this.zulName = zulName;
        }

        public CRUDService getCrudService() {
                return crudService;
        }

        public void setCrudService(CRUDService crudService) {
                this.crudService = crudService;
        }

        public List<ExcelColumns> getExcelColumns() {
                return excelColumns;
        }

        public void setExcelColumns(List<ExcelColumns> excelColumns) {
                this.excelColumns = excelColumns;
        }

        public List<TTipoMedicamentos> getAllReordsInDB() {
                return allReordsInDB;
        }

        public void setAllReordsInDB(List<TTipoMedicamentos> allReordsInDB) {
                this.allReordsInDB = allReordsInDB;
```

```
}

public List<TTipoMedicamentos> getDataSet() {
	return filteredRecords;
}

public void setFilteredRecords(List<TTipoMedicamentos> filteredRecords) {
	this.filteredRecords = filteredRecords;
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
}

public void setMenu(MenuItems menu) {
	this.menu = menu;
}

public Integer getSelectedItemIndex() {
	return selectedItemIndex;
}

public void setSelectedItemIndex(Integer selectedItemIndex) {
	this.selectedItemIndex = selectedItemIndex;
}

public Integer getPageSize() {
	return pageSize;
}

public void setPageSize(Integer pageSize) {
	this.pageSize = pageSize;
}

public TTipoMedicamentos getSelectedItem() {
	return selectedItem;
```

```
	}

	public void setSelectedItem(TTipoMedicamentos selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}


}
```

PAQUETE com.product.webapp.admin
CLASE TURNOSCRUDVM

```
package com.product.webapp.admin;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import javax.servlet.ServletException;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Image;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TEspecialidad;
```

```
import com.product.domain.THoras;
import com.product.domain.TMedico;
import com.product.domain.TTurno;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;

public class TurnosCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#turnosCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;

	private TTurno selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;;
	private Component container;
	final HashMap<String, Object> callerArg = new HashMap<String, Object>();
	private List<TMedico> medicosList;
	private List<THoras> horasList;
	private List<TEspecialidad> especialidadList;

	private TEspecialidad selectEspecialidad = new TEspecialidad();
	private TMedico selectMedico = new TMedico();
	private THoras selectHora = new THoras();

	private List<TTurno> TurnosFecha;

	public List<TTurno> getTurnosFecha() {
		return TurnosFecha;
	}

	public void setTurnosFecha(List<TTurno> turnosFecha) {
		TurnosFecha = turnosFecha;
	}

	public List<TEspecialidad> getEspecialidadList() {
		return especialidadList;
	}

	public void setEspecialidadList(List<TEspecialidad> especialidadList) {
```

```
        this.especialidadList = especialidadList;
    }

    public TEspecialidad getSelectEspecialidad() {
        return selectEspecialidad;
    }

    public void setSelectEspecialidad(TEspecialidad selectEspecialidad) {
        this.selectEspecialidad = selectEspecialidad;
    }

    public TMedico getSelectMedico() {
        return selectMedico;
    }

    public void setSelectMedico(TMedico selectMedico) {
        this.selectMedico = selectMedico;
    }

    public THoras getSelectHora() {
        return selectHora;
    }

    public void setSelectHora(THoras selectHora) {
        this.selectHora = selectHora;
    }

    //
    public List<THoras> getHorasList() {
        return horasList;
    }

    public void setHorasList(List<THoras> horasList) {
        this.horasList = horasList;
    }

    public List<TMedico> getMedicosList() {
        return medicosList;
    }

    public void setMedicosList(List<TMedico> medicosList) {
        this.medicosList = medicosList;
    }

    public TTurno getSelectedRecord() {
        return selectedRecord;
    }

    public void setSelectedRecord(TTurno selectedRecord) {
        this.selectedRecord = selectedRecord;
    }
```

```
	public RecordMode getRecordMode() {
		return recordMode;
	}

	public void setRecordMode(RecordMode recordMode) {
		this.recordMode = recordMode;
	}

	public boolean isMakeAsReadOnly() {
		return makeAsReadOnly;
	}

	public void setMakeAsReadOnly(boolean makeAsReadOnly) {
		this.makeAsReadOnly = makeAsReadOnly;
	}

	@AfterCompose
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("selectedRecord") TTurno turnos,
			@ExecutionArgParam("recordMode") RecordMode mode,
			@ExecutionArgParam("MenuDetails") MenuItems menu,
			@ExecutionArgParam("container") Component container) throws
ServletException {
		Selectors.wireComponents(view, this, false);
		setRecordMode(mode);
		//moreErrorInfo = this.getClass().getName();
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.container = container;
		callerArg.put("container", container);
		callerArg.put("MenuDetails", menu);
		this.especialidadList=
crudService.GetListByNamedQuery("TEspecialidad.getAllTEspecialidad");
		this.horasList =
crudService.GetListByNamedQuery("THoras.getAllTHoras");
		if (mode.equals(RecordMode.ADD)) {
			this.selectedRecord = new TTurno();
			this.selectedRecord.setSystem(0);
			this.selectedRecord.setActive(1);
			this.selectedRecord.setEstado("Disponible");

	this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setCreatedDate(new Timestamp(new
Date().getTime()));
		}

		if (mode.equals(RecordMode.EDIT)) {
			this.selectedRecord = turnos;

	this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
			this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
					.getTime()));
		}
```

```
		if (mode.equals(RecordMode.READ)) {
			setMakeAsReadOnly(true);
			this.selectedRecord = turnos;
		}
	}

	@Command
	public void doCtrlKeyAction(
			@org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

		int keyCode = Integer.parseInt(ctrlKeyCode);
		String s = "";
		switch (keyCode) {
		case 27:
			s = "ESC";
			break;
		case 121:
			s = "F10";
			break;
		}
		if (s.equalsIgnoreCase("F10")) {
			if (!this.makeAsReadOnly)
				saveThis(0);
		}
		if (s.equalsIgnoreCase("ESC")) {
			cancel();
		}
	}

	@Command
	public void saveThis(@BindingParam("action") Integer action) {
		if (Libs.IsValidBean(this.selectedRecord) == false) {
			return;
		}
		try {

	this.selectedRecord.setPersona(this.selectedRecord.getMedico().getPersona
());
			//this.selectedRecord.setHora(this.selectedRecord.getHora());
			crudService.Save(this.selectedRecord);
			Infrastructure.showSuccessmessage();
			goBack();
		} catch (Exception e) {
			Messagebox.show("\t\t"+Consts.NO_SUCCESS_MESSAGE+"\n\n\tYa
existe Registro para este Turno..!!","Ingreso Turno Médico",1,
Messagebox.ERROR);
			goBack();
		}
	}
```

```
	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)
				|| (StringUtils.isBlank(dirty.getSrc()))) {
			goBack();
			return;
		}
		if (StringUtils.isNotBlank(dirty.getSrc())) {
			Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.USAVED_CONFIRMATION);
		}
	}

	@Override
	public void onConfirmClick(String code, int button) {
		if (code.equals("unSaved") && button == Messagebox.YES) {
			goBack();
		}
	}

	public void goBack() {
		container.getChildren().clear();
		win.detach();
		Executions.createComponents(ApplicationLinks.TurnosList, container,
				callerArg);

	}

	//seleccionar los medicos por especialidad
	public List<TMedico> retornarMedicoEspecialidad(int codigoEspecialidad) {
		List<TMedico> listadoMedicos =
crudService.GetListByNamedQuery("TMedico.getAllTMedicos");
		List<TMedico> listaMedicosAux = new ArrayList<TMedico>();
		TMedico medico = new TMedico();
		Iterator<TMedico> i = listadoMedicos.iterator();
		while (i.hasNext())
		{
			medico = i.next();
				if (medico.getEspecialidad().getIdespecialidad() ==
codigoEspecialidad) {
					listaMedicosAux.add(medico);
				}
		}
		return listaMedicosAux;
	}
	@Command
	@NotifyChange("medicosList")
	public void ShowSelectedEspecialidad()
	{
		this.medicosList =
retornarMedicoEspecialidad(this.selectEspecialidad.getIdespecialidad());
```

```
	}
}
```

PAQUETE com.product.webapp.admin
CLASE TURNOSLISTVM

```
package com.product.webapp.admin;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.TTurno;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;
```

```
public class TurnosListVM implements ConfirmResponse {

	@Wire("#turnoslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<TTurno> allReordsInDB = null;
	private List<TTurno> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private TTurno selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
		allReordsInDB =
crudService.GetListByNamedQuery("TTurno.getAllTTurno");
		filteredRecords = (List<TTurno>)
CollectionUtils.select(allReordsInDB,
				new PredicateActive());

		zulName.setFocus(true);

	}

	@GlobalCommand
	@NotifyChange("dataSet")
	public void UsersListVMRefresh(
			@BindingParam("selectedRecord") TTurno turnos,
			@BindingParam("recordMode") RecordMode mode) {

		if (mode.equals(RecordMode.ADD)) {
			filteredRecords.add(turnos);
```

```
                allReordsInDB.add(turnos);
            }
            if (mode.equals(RecordMode.EDIT)) {
                filteredRecords.set(this.selectedItemIndex, turnos);
                allReordsInDB.set(this.selectedItemIndex, turnos);
            }
        }

        @Command
        @NotifyChange("dataSet")
        public void doFilter() {
            filteredRecords = new ArrayList<TTurno>();
            String fech = "";
            int cont = 1;
            String formattedDate = "";
            SimpleDateFormat format;
            String formattedDate1 = "";
            for (Iterator<TTurno> i = allReordsInDB.iterator(); i.hasNext();) {
                TTurno tmp = i.next();
                fech = String.valueOf(tmp.getFechaturno());
                Date fecha = dataFilter.getFechaFiltro();
                Date dFechaActual = dataFilter.getFechaFiltroFin();
                format = new SimpleDateFormat("yyyy-MM-dd");
                formattedDate = format.format(fecha);
                formattedDate1 = format.format(dFechaActual);
                if ((tmp.getFechaturno().before(dFechaActual) &&
tmp.getFechaturno().after(fecha))|| (fech.compareTo(formattedDate1) == 0)||
(fech.compareTo(formattedDate) == 0))
                {
                    if
(tmp.getMedico().getPersona().getCedulapersona().indexOf(dataFilter.getName())
== 0)
                    {
                        System.out.println("::" + cont++ + "----------
>" + tmp.getFechaturno());
                        if (dataFilter.getIsActive().intValue() == 0)
                            filteredRecords.add(tmp);
                        else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
                            filteredRecords.add(tmp);
                        }
                    }
                }
            }
        }

        @Command
        public void onExcelExport() {

            this.excelColumns = new ArrayList<ExcelColumns>();
            this.excelColumns.add(new ExcelColumns("userName", "User Name"));
```

```
            this.excelColumns.add(new ExcelColumns("firstName", "First Name",
9600));
            this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
            BeanToExcel beanToExcel = new BeanToExcel();
            beanToExcel.setExcelColumns(excelColumns);
            beanToExcel.setDataSheetName("UsersList");
            beanToExcel.setDataList(filteredRecords);
            beanToExcel.exportToExcel();
      }

      @Command
      public void onDeactivate() {
            ShowWindow.ShowDeactivateWin();
      }

      @Command
      public void onActivate() {
            ShowWindow.ShowActivateWin();
      }

      // note this will be executed from ActivateVM.Java
      @GlobalCommand
      @NotifyChange("selectedItem")
      public void activateThis(@BindingParam("reason") String reason) {
            this.selectedItem.setActive(1);
            this.selectedItem.setActivatedReason(reason);
            crudService.Save(this.selectedItem);
      }

      // note this will be executed from deactivateVM.Java
      @GlobalCommand
      @NotifyChange("selectedItem")
      public void deactivateThis(@BindingParam("reason") String reason) {
            this.selectedItem.setActive(0);
            this.selectedItem.setDeactivatedReason(reason);
            crudService.Save(this.selectedItem);
      }

      @Command
      public void onAddNew() {
            goToCRUD(RecordMode.ADD);
      }

      @Command
      public void openAsReadOnly() {
            goToCRUD(RecordMode.READ);
      }

      @Command
      public void onEdit() {
            goToCRUD(RecordMode.EDIT);
```

```
	}

	@Command
	public void onlinkOpen(@BindingParam("record") TTurno record)
			throws Exception, NoSuchFieldException {
		if (menu.getCanEdit() == 1)
			onEdit();
		else
			openAsReadOnly();
	}

	public void goToCRUD(final RecordMode mode) {
		if (mode.equals(RecordMode.ADD))
			CRUDargs.put("selectedRecord", null);
		else {
			CRUDargs.put("selectedRecord", selectedItem);
			setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
		}
		CRUDargs.put("recordMode", mode);
		CRUDargs.put("container", container);
		CRUDargs.put("MenuDetails", menu);
		container.getChildren().clear();
		ShowWindow
				.openZulFile(ApplicationLinks.TurnosCRUD, container,
CRUDargs);
	}

	@Command
	public void onDelete() {
		Infrastructure.confirm("deleteFirstConfirm",
				"El elemento seleccionado  \"" +
this.selectedItem.getIdturno()
						+ "\" se eliminará.?",
Consts.UNSAVED_TITLE, this,
				MessageBoxConfirmType.DELETE_CONFIRMATION);
	}

	public void deleteSelectedItem() {
		try {
			crudService.delete(selectedItem);
			Infrastructure.showSuccessmessage();
			allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

	filteredRecords.remove(filteredRecords.indexOf(selectedItem));
			BindUtils.postNotifyChange(null, null, this, "dataSet");
		} catch (Exception e) {
			ExceptionHandler.handleException(e, "TTurno ",
this.selectedItem,
					"" + ": onConfirmClick");
		}
	}
```

```
	@Override
	@NotifyChange("dataSet")
	public void onConfirmClick(String code, int button) {
		if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
			Infrastructure
					.confirm(
							"deleteSecondConfirm",
							"El elemento seleccionado  \""
									+
this.selectedItem.getIdturno()
									+ "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
							Consts.UNSAVED_TITLE, this,

	MessageBoxConfirmType.DELETE_CONFIRMATION);
		}
		if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
			deleteSelectedItem();
		}
	}

	public Window getWin() {
		return win;
	}

	public void setWin(Window win) {
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
```

```
		this.excelColumns = excelColumns;
	}

	public List<TTurno> getAllReordsInDB() {
		return allReordsInDB;
	}

	public void setAllReordsInDB(List<TTurno> allReordsInDB) {
		this.allReordsInDB = allReordsInDB;
	}

	public List<TTurno> getDataSet() {
		return filteredRecords;
	}

	public void setFilteredRecords(List<TTurno> filteredRecords) {
		this.filteredRecords = filteredRecords;
	}

	public Component getContainer() {
		return container;
	}

	public void setContainer(Component container) {
		this.container = container;
	}

	public DataFilter getDataFilter() {
		return dataFilter;
	}

	public void setDataFilter(DataFilter dataFilter) {
		this.dataFilter = dataFilter;
	}

	public MenuItems getMenu() {
		return menu;
	}

	public void setMenu(MenuItems menu) {
		this.menu = menu;
	}

	public Integer getSelectedItemIndex() {
		return selectedItemIndex;
	}

	public void setSelectedItemIndex(Integer selectedItemIndex) {
		this.selectedItemIndex = selectedItemIndex;
	}

	public Integer getPageSize() {
```

```
		return pageSize;
	}

	public void setPageSize(Integer pageSize) {
		this.pageSize = pageSize;
	}

	public TTurno getSelectedItem() {
		return selectedItem;
	}

	public void setSelectedItem(TTurno selectedItem) {
		this.selectedItem = selectedItem;
	}

	public HashMap<String, Object> getCRUDargs() {
		return CRUDargs;
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```

PAQUETE com.product.webapp.admin
CLASE USERCRUDVM

```
package com.product.webapp.admin;

import java.io.IOException;
import java.sql.Timestamp;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindContext;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.image.AImage;
import org.zkoss.image.Image;
import org.zkoss.util.media.Media;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.Executions;
import org.zkoss.zk.ui.event.UploadEvent;
```

```
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.Roles;
import com.product.domain.Users;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.ExceptionHandler;
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.Libs;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.msjUtilitarios;

public class UserCRUDVM implements ConfirmResponse {

	@WireVariable
	private CRUDService crudService;

	@Wire("#userCRUD")
	private Window win;

	@Wire("#dirty")
	private Image dirty;
	private boolean rolAdministrador = false;
	private Users selectedRecord;
	private RecordMode recordMode;
	private boolean makeAsReadOnly;
	private String moreErrorInfo = "";
	private Component container;
	final HashMap<String, Object> callerArg = new HashMap<String, Object>();
	private List<Roles> rolesList;
	private AImage myImage;

	public boolean isRolAdministrador() {
		return rolAdministrador;
	}

	public void setRolAdministrador(boolean rolAdministrador) {
		this.rolAdministrador = rolAdministrador;
	}

	public List<Roles> getRolesList() {
		return rolesList;
```

```
}

public void setRolesList(List<Roles> rolesList) {
	this.rolesList = rolesList;
}

public Users getSelectedRecord() {
	return selectedRecord;
}

public void setSelectedRecord(Users selectedRecord) {
	this.selectedRecord = selectedRecord;
}

public RecordMode getRecordMode() {
	return recordMode;
}

public void setRecordMode(RecordMode recordMode) {
	this.recordMode = recordMode;
}

public boolean isMakeAsReadOnly() {
	return makeAsReadOnly;
}

public void setMakeAsReadOnly(boolean makeAsReadOnly) {
	this.makeAsReadOnly = makeAsReadOnly;
}

public AImage getMyImage() {
	return myImage;
}

public void setMyImage(AImage myImage) {
	this.myImage = myImage;
}

@AfterCompose
@NotifyChange("myImage")
public void initSetup(@ContextParam(ContextType.VIEW) Component view,
		@ExecutionArgParam("selectedRecord") Users user,
		@ExecutionArgParam("recordMode") RecordMode mode,
		@ExecutionArgParam("MenuDetails") MenuItems menu,
		@ExecutionArgParam("container") Component container)
		throws IOException {

	verificaRolControl();
	Selectors.wireComponents(view, this, false);
	setRecordMode(mode);
	moreErrorInfo = this.getClass().getName();
	crudService = (CRUDService) SpringUtil.getBean("crudService");
```

```
            this.container = container;
            callerArg.put("container", container);
            callerArg.put("MenuDetails", menu);

            myImage = new
AImage(Executions.getCurrent().getDesktop().getWebApp()
                        .getRealPath("/images")
                        + "/male.png");
            this.rolesList =
crudService.GetListByNamedQuery("Roles.getAllRoles");

            if (mode.equals(RecordMode.ADD)) {
                  this.selectedRecord = new Users();
                  this.selectedRecord.setSystem(0);
                  this.selectedRecord.setActive(1);

      this.selectedRecord.setCreatedBy(FHSessionUtil.getLoggedInUserID());
                  this.selectedRecord.setCreatedDate(new Timestamp(new Date()
                              .getTime()));
            }
            if (mode.equals(RecordMode.EDIT)) {
                  this.selectedRecord = user;
                  this.selectedRecord.setEstado("A");

      this.selectedRecord.setUpdatedBy(FHSessionUtil.getLoggedInUserID());
                  this.selectedRecord.setUpdatedDate(new Timestamp(new Date()
                              .getTime()));
            }

            if (mode.equals(RecordMode.READ)) {
                  try {
                        setMakeAsReadOnly(true);
                        this.selectedRecord = user;
                        if (this.selectedRecord.getFoto() != null)
                              myImage = new AImage("userPhoto",
                                          this.selectedRecord.getFoto());
                  } catch (Exception e) {
                        // TODO: handle exception
                        if (this.selectedRecord.getFoto() != null)
                              myImage = new AImage("userPhoto",
                                          this.selectedRecord.getFoto());
                  }
            }
      }

      // ////////////////////////
      public void verificaRolControl() {
            String RolUsuario = StringUtils.capitalize(FHSessionUtil
                        .getCurrentUser().getUserRole().getRoleName());
            switch (RolUsuario) {
            case "Admin": {
                  setRolAdministrador(false);
```

```
                        break;
                }
                default: {
                        setRolAdministrador(true);
                        break;
                }
                }
        }

        // ////////////////////////////////

        @Command
        public void doCtrlKeyAction(
                        @org.zkoss.bind.annotation.BindingParam("code") String
ctrlKeyCode) {

                int keyCode = Integer.parseInt(ctrlKeyCode);
                String s = "";
                switch (keyCode) {
                case 27:
                        s = "ESC";
                        break;
                case 121:
                        s = "F10";
                        break;
                }
                if (s.equalsIgnoreCase("F10")) {
                        if (!this.makeAsReadOnly)
                                saveThis(0);
                }
                if (s.equalsIgnoreCase("ESC")) {
                        cancel();
                }

        }

        @Command
        public void saveThis(@BindingParam("action") Integer action) {
                if (Libs.IsValidBean(this.selectedRecord) == false) {
                        return;
                }
                try {
                        msjUtilitarios msj= new msjUtilitarios();
                        String pass = msj.Encriptar(selectedRecord.getPassword());
                        selectedRecord.setPassword(pass);
                        crudService.Save(this.selectedRecord);
                        Infrastructure.showSuccessmessage();
                        goBack();
                } catch (Exception e) {
                        ExceptionHandler.handleException(e, "Users ",
this.selectedRecord,
                                        this.moreErrorInfo + ": SaveThis");
```

```
		}
	}

	@Command
	public void cancel() {
		if (recordMode.equals(RecordMode.READ)) {
			goBack();
			return;
		} else {
			Infrastructure.confirm("unSaved", Consts.UNSAVED_CONFIRM,
					Consts.UNSAVED_TITLE, this,
					MessageBoxConfirmType.USAVED_CONFIRMATION);
		}
	}

	@Override
	public void onConfirmClick(String code, int button) {
		if (code.equals("unSaved") && button == Messagebox.YES) {
			goBack();
		}
	}

	public void goBack() {
		container.getChildren().clear();
		win.detach();
		Executions.createComponents(ApplicationLinks.UsersList, container,
				callerArg);
	}
	@Command
	@NotifyChange("myImage")
	public void removeImage() {

		myImage = null;
	}
	@Command
	@NotifyChange("myImage")
	public void upload(@ContextParam(ContextType.BIND_CONTEXT) BindContext
ctx) {
		UploadEvent upEvent = null;
		Object objUploadEvent = ctx.getTriggerEvent();
		if (objUploadEvent != null && (objUploadEvent instanceof
UploadEvent)) {
			upEvent = (UploadEvent) objUploadEvent;
		}
		if (upEvent != null) {
			Media media = upEvent.getMedia();
			int lengthofImage = media.getByteData().length;
			if (media instanceof Image) {
				if (lengthofImage > 500 * 1024) {
					Messagebox
							.show("Please Select a Image of
size less than 500Kb.");
```

```
                                return;
                          } else {
                                myImage = (AImage) media;
                          }
                    } else {
                          Messagebox.show("The selected File is not an image.");
                    }
              }
        }
}
```

PAQUETE com.product.webapp.admin
CLASE USERSLISTVM

```
package com.product.webapp.admin;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.lang3.StringUtils;
import org.zkoss.bind.BindUtils;
import org.zkoss.bind.annotation.AfterCompose;
import org.zkoss.bind.annotation.BindingParam;
import org.zkoss.bind.annotation.Command;
import org.zkoss.bind.annotation.ContextParam;
import org.zkoss.bind.annotation.ContextType;
import org.zkoss.bind.annotation.ExecutionArgParam;
import org.zkoss.bind.annotation.GlobalCommand;
import org.zkoss.bind.annotation.NotifyChange;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.Selectors;
import org.zkoss.zk.ui.select.annotation.Wire;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zkplus.spring.SpringUtil;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

import com.product.business.service.CRUDService;
import com.product.domain.Users;
import com.product.webapp.excelexport.BeanToExcel;
import com.product.webapp.excelexport.ExcelColumns;
import com.product.webapp.utilities.ApplicationLinks;
import com.product.webapp.utilities.ConfirmResponse;
import com.product.webapp.utilities.Consts;
import com.product.webapp.utilities.DataFilter;
import com.product.webapp.utilities.ExceptionHandler;
```

```
import com.product.webapp.utilities.FHSessionUtil;
import com.product.webapp.utilities.Infrastructure;
import com.product.webapp.utilities.MenuItems;
import com.product.webapp.utilities.MessageBoxConfirmType;
import com.product.webapp.utilities.PredicateActive;
import com.product.webapp.utilities.RecordMode;
import com.product.webapp.utilities.ShowWindow;

public class UsersListVM implements ConfirmResponse {

	@Wire("#userslist")
	private Window win;

	@Wire("#name")
	private Textbox zulName;

	@WireVariable
	protected CRUDService crudService;

	private List<ExcelColumns> excelColumns = null;
	private List<Users> allReordsInDB = null;
	private List<Users> filteredRecords = null;
	private Component container;
	private DataFilter dataFilter = new DataFilter();
	private MenuItems menu;
	private Integer selectedItemIndex;
	private Integer pageSize;
	private Users selectedItem;
	protected HashMap<String, Object> CRUDargs = new HashMap<String,
Object>();

	@SuppressWarnings("unchecked")
	@AfterCompose
	@NotifyChange("dataSet")
	public void initSetup(@ContextParam(ContextType.VIEW) Component view,
			@ExecutionArgParam("container") Component container,
			@ExecutionArgParam("MenuDetails") MenuItems menu) {
		Selectors.wireComponents(view, this, false);
		this.container = container;
		this.menu = menu;
		String Roles =
StringUtils.capitalize(FHSessionUtil.getCurrentUser().getUserRole().getRoleName(
));
		if (!Roles.trim().equals("Admin"))
		{

	dataFilter.setName(StringUtils.capitalize(FHSessionUtil.getCurrentUser().
getMiddleName()));
		}
		crudService = (CRUDService) SpringUtil.getBean("crudService");
		this.pageSize = FHSessionUtil.getDesktopHeight();
```

```
            allReordsInDB =
crudService.GetListByNamedQuery("Users.getAllUsers");
            filteredRecords = (List<Users>)
CollectionUtils.select(allReordsInDB,
                        new PredicateActive());

            zulName.setFocus(true);

      }

      @GlobalCommand
      @NotifyChange("dataSet")
      public void UsersListVMRefresh(@BindingParam("selectedRecord") Users
user,
                  @BindingParam("recordMode") RecordMode mode) {

            if (mode.equals(RecordMode.ADD)) {
                  filteredRecords.add(user);
                  allReordsInDB.add(user);
            }
            if (mode.equals(RecordMode.EDIT)) {
                  filteredRecords.set(this.selectedItemIndex, user);
                  allReordsInDB.set(this.selectedItemIndex, user);
            }
      }

      @Command
      @NotifyChange("dataSet")
      public void doFilter() {
            filteredRecords = new ArrayList<Users>();
            String Rol =
StringUtils.capitalize(FHSessionUtil.getCurrentUser().getUserRole().getRoleName(
));
            String cedulaPer =
StringUtils.capitalize(FHSessionUtil.getCurrentUser().getMiddleName());
            if (!Rol.trim().equals("Admin"))
            {
                  Users tmp;
                  dataFilter.setName(cedulaPer);
                  for (Iterator<Users> i = allReordsInDB.iterator();
i.hasNext();) {
                        tmp = i.next();
                        if
(tmp.getMiddleName().trim().equals(cedulaPer.trim())) {
                              filteredRecords.add(tmp);
                              break;
                        }
                  }
            }
            else {
                  Users tmp;
```

```
                    for (Iterator<Users> i = allReordsInDB.iterator();
i.hasNext();) {
                            tmp = i.next();
                            if
(tmp.getMiddleName().toLowerCase().indexOf(dataFilter.getName()) == 0) {
                                    if (dataFilter.getIsActive().intValue() == 0)
                                            filteredRecords.add(tmp);
                                    else if
(tmp.getActive().equals(dataFilter.getIsActive())) {
                                            filteredRecords.add(tmp);
                                    }
                            }
                    }
            }
      }

      @Command
      public void onExcelExport() {

            this.excelColumns = new ArrayList<ExcelColumns>();
            this.excelColumns.add(new ExcelColumns("userName", "User Name"));
            this.excelColumns
                        .add(new ExcelColumns("firstName", "First Name",
9600));
            this.excelColumns.add(new ExcelColumns("lastName", "Last Name",
9600));
            BeanToExcel beanToExcel = new BeanToExcel();
            beanToExcel.setExcelColumns(excelColumns);
            beanToExcel.setDataSheetName("UsersList");
            beanToExcel.setDataList(filteredRecords);
            beanToExcel.exportToExcel();
      }

      @Command
      public void onDeactivate() {
            ShowWindow.ShowDeactivateWin();
      }

      @Command
      public void onActivate() {
            ShowWindow.ShowActivateWin();
      }


      // note this will be executed from ActivateVM.Java
      @GlobalCommand
      @NotifyChange("selectedItem")
      public void activateThis(@BindingParam("reason") String reason) {
            this.selectedItem.setActive(1);
            this.selectedItem.setActivatedReason(reason);
            crudService.Save(this.selectedItem);
      }
```

```
// note this will be executed from deactivateVM.Java
@GlobalCommand
@NotifyChange("selectedItem")
public void deactivateThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(0);
	this.selectedItem.setDeactivatedReason(reason);
	crudService.Save(this.selectedItem);
}

@GlobalCommand
@NotifyChange("selectedItem")
public void cambioClaveThis(@BindingParam("reason") String reason) {
	this.selectedItem.setActive(1);
	this.selectedItem.setPassword(reason);
	crudService.Save(this.selectedItem);
}

@Command
public void onAddNew() {
	goToCRUD(RecordMode.ADD);
}

@Command
public void openAsReadOnly() {
	goToCRUD(RecordMode.READ);
}

@Command
public void onEdit() {
	goToCRUD(RecordMode.EDIT);
}

@Command
public void onlinkOpen(@BindingParam("record") Users record)
		throws Exception, NoSuchFieldException {
	if (menu.getCanEdit() == 1)
		onEdit();
	else
		openAsReadOnly();
}

public void goToCRUD(final RecordMode mode) {
	if (mode.equals(RecordMode.ADD))
		CRUDargs.put("selectedRecord", null);
	else {
		CRUDargs.put("selectedRecord", selectedItem);
		setSelectedItemIndex(filteredRecords.indexOf(selectedItem));
	}
	CRUDargs.put("recordMode", mode);
	CRUDargs.put("container", container);
	CRUDargs.put("MenuDetails", menu);
```

```
            container.getChildren().clear();
            ShowWindow.openZulFile(ApplicationLinks.UsersCRUD, container,
CRUDargs);
      }

      @Command
      public void onDelete() {
            Infrastructure
                        .confirm(
                                    "deleteFirstConfirm",
                                    "El elemento seleccionado  \""
                                                +
this.selectedItem.getUserName()
                                                + "\" se eliminará.?",
Consts.UNSAVED_TITLE,
                                    this,
MessageBoxConfirmType.DELETE_CONFIRMATION);
      }

      public void deleteSelectedItem() {
            try {
                  crudService.delete(selectedItem);
                  Infrastructure.showSuccessmessage();
                  allReordsInDB.remove(allReordsInDB.indexOf(selectedItem));

      filteredRecords.remove(filteredRecords.indexOf(selectedItem));
                  BindUtils.postNotifyChange(null, null, this, "dataSet");
            } catch (Exception e) {
                  ExceptionHandler.handleException(e, "Users ",
this.selectedItem, ""
                              + ": onConfirmClick");
            }
      }

      @Override
      @NotifyChange("dataSet")
      public void onConfirmClick(String code, int button) {
            if (code.equals("deleteFirstConfirm") && button == Messagebox.YES)
{
                  Infrastructure
                              .confirm(
                                          "deleteSecondConfirm",
                                          "El elmento seleccionado  \""
                                                      +
this.selectedItem.getUserName()
                                                      + "\" Se eliminará de
forma permanente y la acción no se puede deshacer..?",
                                          Consts.UNSAVED_TITLE, this,

      MessageBoxConfirmType.DELETE_CONFIRMATION);
            }
```

```
		if (code.equals("deleteSecondConfirm") && button == Messagebox.YES)
{
			deleteSelectedItem();
		}
	}

	public Window getWin() {
		return win;
	}

	public void setWin(Window win) {
		this.win = win;
	}

	public Textbox getZulName() {
		return zulName;
	}

	public void setZulName(Textbox zulName) {
		this.zulName = zulName;
	}

	public CRUDService getCrudService() {
		return crudService;
	}

	public void setCrudService(CRUDService crudService) {
		this.crudService = crudService;
	}

	public List<ExcelColumns> getExcelColumns() {
		return excelColumns;
	}

	public void setExcelColumns(List<ExcelColumns> excelColumns) {
		this.excelColumns = excelColumns;
	}

	public List<Users> getAllReordsInDB() {
		return allReordsInDB;
	}

	public void setAllReordsInDB(List<Users> allReordsInDB) {
		this.allReordsInDB = allReordsInDB;
	}

	public List<Users> getDataSet() {
		return filteredRecords;
	}

	public void setFilteredRecords(List<Users> filteredRecords) {
		this.filteredRecords = filteredRecords;
```

```
}

public Component getContainer() {
	return container;
}

public void setContainer(Component container) {
	this.container = container;
}

public DataFilter getDataFilter() {
	return dataFilter;
}

public void setDataFilter(DataFilter dataFilter) {
	this.dataFilter = dataFilter;
}

public MenuItems getMenu() {
	return menu;
}

public void setMenu(MenuItems menu) {
	this.menu = menu;
}

public Integer getSelectedItemIndex() {
	return selectedItemIndex;
}

public void setSelectedItemIndex(Integer selectedItemIndex) {
	this.selectedItemIndex = selectedItemIndex;
}

public Integer getPageSize() {
	return pageSize;
}

public void setPageSize(Integer pageSize) {
	this.pageSize = pageSize;
}

public Users getSelectedItem() {
	return selectedItem;
}

public void setSelectedItem(Users selectedItem) {
	this.selectedItem = selectedItem;
}

public HashMap<String, Object> getCRUDargs() {
	return CRUDargs;
```

```
	}

	public void setCRUDargs(HashMap<String, Object> cRUDargs) {
		CRUDargs = cRUDargs;
	}

}
```