UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

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

TEMA: "SISTEMA MULTIPLATAFORMA PARA LA GESTIÓN DEL FLUJO DE CLIENTES"

AUTOR: JUAN PABLO RUÍZ TIRIRA

DIRECTOR: ING. CARPIO PINEDA

IBARRA-ECUADOR
2015

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

La UNIVERSIDAD TÉCNICA DEL NORTE dentro del proyecto Repositorio Digital institucional determina la necesidad de disponer los textos completos de forma digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual ponemos a disposición la siguiente información:

DATOS DE CONTACTO	
CÉDULA DE IDENTIDAD:	100217296-1
APELLIDOS Y NOMBRES:	RUÍZ TIRIRA JUAN PABLO
DIRECCIÓN:	QUITO, Calle Carlos Mantilla y Pasaje Mantilla, sector San José de Morán.
EMAIL:	juanpa7900@gmail.com
TELÉFONO FIJO:	042921883
TELÉFONO MOVIL:	0997364934
DATOS DE LA OBRA	
TÍTULO:	"SISTEMA MULTIPLATAFORMA PARA LA GESTIÓN DEL FLUJO DE CLIENTES"
AUTOR:	RUÍZ TIRIRA JUAN PABLO
FECHA:	JUNIO 2015
PROGRAMA:	PREGRADO
TÍTULO POR EL QUE OPTA:	INGENIERO EN SISTEMAS COMPUTACIONALES
DIRECTOR:	ING. CARPIO PINEDA

.....

Nombre: Juan Pablo Ruíz Tirira

Cédula: 100217296-1

Ibarra a los 19 días del mes de Junio del 2015

CERTIFICACIÓN

Ibarra, 19 de Junio del 2015

Señores
UNIVERSIDAD TÉCNICA DEL NORTE
Presente

De mis consideraciones.-

Siendo auspiciantes del proyecto de tesis del egresado RUIZ TIRIRA JUAN PABLO con CI: 100217296-1 quien desarrolló su trabajo con el tema "SISTEMA MULTIPLATAFORMA PARA LA GESTIÓN DEL FLUJO DE CLIENTES", con el aplicativo: "DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN DEL FLUJO DE CLIENTES UTILIZANDO TECNOLOGIA J2EE" me es grato informar que se han superado con satisfacción las pruebas técnicas y la revisión de cumplimiento de los requerimientos funcionales, por lo que se recibe el proyecto como culminado y realizado por parte del egresado RUIZ TIRIRA JUAN PABLO. Una vez que hemos recibido la capacitación y documentación respectiva, nos comprometemos a continuar utilizando el mencionado aplicativo en beneficio de nuestra empresa.

El egresado RUIZ TIRIRA JUAN PABLO, puede hacer uso de este documento para los fines pertinentes en la Universidad Técnica del Norte.

Q-MATIC ECUADOR CIA. LTDA.
RUC: 1791432762001

.....

Ing. Jesús Gómez
Gerente General, Empresa Q-Matic Ecuador CIA. LTDA.

2. AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

Yo, RUÍZ TIRIRA JUAN PABLO, con cédula de identidad Nro. 100217296-1, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en forma digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y el uso del archivo digital en la biblioteca de la universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión, en concordancia con la Ley de Educación Superior Artículo 143.

EL AUTOR:

.....

RUÍZ TIRIRA JUAN PABLO

CI.: 100217296-1

UNIVERSIDAD TÉCNICA DEL NORTE

CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE INVESTIGACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, Juan Pablo Ruíz Tirira, con cédula de identidad Nro. 100217296-1 manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la ley de propiedad intelectual del Ecuador, artículos 4,5 y 6 en calidad de autor del trabajo de grado denominado: "SISTEMA MULTIPLATAFORMA PARA LA GESTIÓN DEL FLUJO DE CLIENTES", que ha sido desarrollado para optar por el título de: **Ingeniero en Sistemas Computacionales**, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En mi condición de autor me reservo los derechos morales de la obra antes citada.

En concordancia suscribo este documento en el momento en el que hago la entrega del trabajo final en formato impreso y digital a la biblioteca de la Universidad Técnica del Norte.

.....

Nombre: Juan Pablo Ruíz Tirira

Cédula: 100217296-1

Ibarra a los 19 días de Junio del 2015

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DEL ASESOR

Certifico que la Tesis **"SISTEMA MULTIPLATAFORMA PARA LA GESTIÓN DEL FLUJO DE CLIENTES"**, previo a la obtención del título de Ingeniero en Sistemas Computacionales, ha sido realizado en su totalidad por el señor Ruíz Tirira Juan Pablo portador de la cédula de identidad número: 100217296-1.

Ibarra a los 19 días de Junio del 2015

.....

Ing. Carpio Pineda

Director de Proyecto

DEDICATORIA

A mi madre por ser la persona que ha estado conmigo en los buenos y en los malos momentos, por ser una madre incondicional quien con su ejemplo de lucha me enseñó a seguir adelante y no decaer ante las circunstancias adversas de la vida.

A mi padre por inculcarme los valores morales que han hecho de mí una buena persona, por enseñarme que en la vida todo lo que se busca, con esfuerzo y perseverancia se consigue.

A mis hermanos por haberme brindado su apoyo en los momentos en los que más necesité.

A mis hijos Sol y Mateo por ser mi fuente de inspiración y la razón principal por la cual siempre doy mi mejor esfuerzo.

A mi esposa María Gabriela por ser mi compañera de la vida, la persona que nunca ha dejado de creer en mí y me impulsa siempre a luchar por mis sueños.

Juan Pablo.

AGRADECIMIENTO

A Dios por darme la vida, las fuerzas y la sabiduría para poder cumplir con mis sueños y mis objetivos en la vida.

A mis padres, mis hermanas y mis hermanos por todo el apoyo que me han brindado para poder culminar con éxito este trabajo.

A mi esposa y mis hijos por el cariño y la confianza que han depositado en mí.

A mi amigo Gerry, quien de manera desinteresada ha compartido conmigo sus conocimientos, los cuales han aportado al desarrollo de este trabajo.

A la empresa Q-Matic Ecuador por haberme brindado todas las facilidades para poder desarrollar con éxito este trabajo.

Un sincero agradecimiento al Ing. Carpio Pineda por su tiempo y su valiosa ayuda.

Juan Pablo.

TABLA DE CONTENIDOS

INDICE DE GRÁFICOS

INDICE DE TABLAS

RESUMEN

En la actualidad existen empresas e instituciones tanto públicas como privadas donde la atención a los clientes es deficiente, las personas pierden su tiempo haciendo largas colas para ser atendidos.

Por otra parte, en empresas o instituciones donde se ha automatizado la atención mediante un sistema de administración del flujo de clientes, existe una deficiencia relacionada con los tiempos de respuesta del soporte técnico cuando el sistema falla, debido a que las empresas que desarrollan sistemas de administración del flujo de clientes en su mayoría se encuentran ubicadas en el extranjero, lo cual provoca pérdidas económicas para las empresas que implantan estas soluciones.

El Sistema de Gestión del Flujo de Clientes para la empresa Q-Matic Ecuador es una solución tecnológica acorde a los requerimientos y necesidades de las empresas que brindan atención al público las cuales están permanentemente diseñando estrategias que les permitan brindar una mejor atención al público a un costo menor.

Este sistema permitirá a la empresa Q-Matic Ecuador disponer de un sistema multiplataforma el cual podrá ser implementado en cualquier institución que brinde atención al público sin importar su tamaño.

El sistema permitirá gestionar las colas de espera de manera ágil y sin contratiempos relacionados al soporte técnico del sistema.

SUMMARY

Today there are companies and both public and private institution where the customer service is poor, people waste their time standing in long lines for help.

Moreover in companies or institutions where attention has been automated through a system of management of customer flow, there is a deficiency related to the response times of technical support when the system fails, because the companies that develop management systems customer flow are mostly located abroad, causing economic losses for those undertakings that implement these solutions

The management system flow of customers for the company Q-Matic Ecuador, is a technological solution according to the requirements and needs of companies who provide care to the public which are constantly devising strategies that enable them to provide better care at a public solution lower cost.

This will allow the company Q-Matic Ecuador have a multiplatform system which could be implemented in any institution providing care to the public regardless of system size.

The system will manage agile waiting lines and smoothly related to technical support system.

INTRODUCCIÓN

A. PROBLEMA

- **Antecedentes**

Los tiempos de respuesta muy largos en cuanto al soporte de software y hardware por parte de las empresas extranjeras proveedoras de sistemas de administración de colas de espera, conlleva pérdidas de tiempo que se traducen en pérdidas económicas para las empresas usuarias de estos tipos de sistemas.

Las empresas que no cuentan con un software de administración del flujo de clientes, no están en la capacidad de brindar una atención adecuada a los clientes de las mismas, pues el exceso de tiempo que esperan las personas para ser atendidas conlleva inconformidad y en ciertos casos deserción por parte de estas personas.

- **Situación actual**

En la actualidad existen empresas e instituciones tanto públicas como privadas donde la atención a los clientes es deficiente, las personas pierden su tiempo haciendo largas colas para ser atendidos.

- **Definición del problema**

En empresas o instituciones donde se ha automatizado la atención mediante un sistema de administración del flujo de clientes, existe una deficiencia relacionada con los tiempos de respuesta del soporte técnico cuando el sistema falla, debido a que las empresas que desarrollan sistemas de administración del flujo de clientes en su mayoría se encuentran ubicadas

en el extranjero, lo cual provoca pérdidas económicas para las empresas que implantan estas soluciones.

- **Prospectiva**

El desarrollo de un sistema de control de flujo de clientes, implementado con tecnología local permitirá brindar una mejor atención a los clientes que adquieren dichos sistemas. Permitirá el ahorro de recursos tanto humanos como económicos maximizando la rentabilidad de los usuarios de este sistema.

B. OBJETIVOS

- **Objetivo general**

Desarrollar un sistema multiplataforma para la gestión del flujo de clientes que permita mejorar el nivel de satisfacción mediante la automatización del proceso de atención.

- **Objetivos específicos**

  - Analizar la estructura, la clasificación y los tipos de colas de espera.

  - Determinar los requerimientos específicos de software y hardware que se necesitan para el desarrollo del sistema.

  - Definir las características importantes y las ventajas de la plataforma J2EE en el desarrollo de aplicaciones.

- Implementar un sistema multiplataforma para la administración del flujo de clientes.

C. ALCANCE Y LIMITACIONES

- **Alcance**

Se desarrollará un sistema de gestión del flujo de clientes, el mismo que estará estructurado de la siguiente manera:

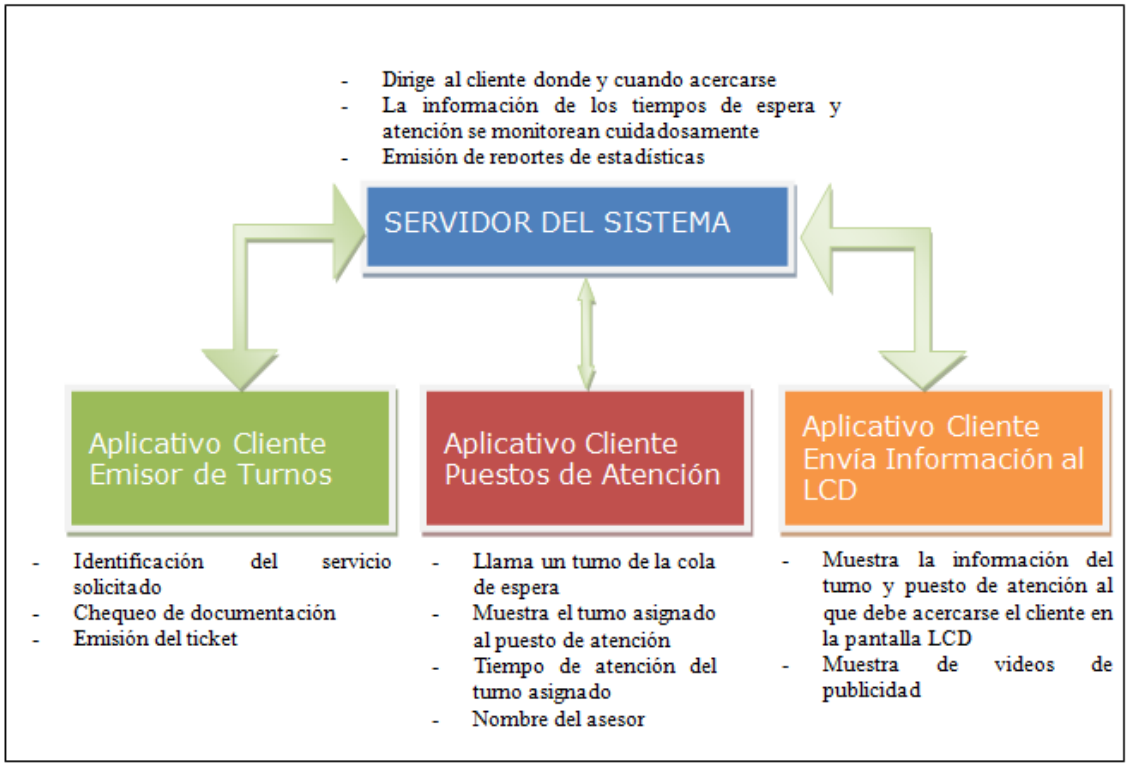

Figura 1. Componentes de Software de un Sistema Administrador de colas.

Fuente: Propia

El servidor del sistema será el encargado de la gestión y direccionamiento de los turnos. Permitirá el manejo de prioridades de espera, tiempos de espera, tiempos de atención, pantalla de supervisión de los turnos en espera,

pantalla de supervisión de los asesores conectados al sistema, permitirá la generación de alarmas en base a los tiempos de espera y tiempos de atención. La información se almacenará en una base de datos SQL Server, con esta información se emitirán los reportes estadísticos necesarios para la toma de decisiones por parte del administrador del sistema.

Aplicativo cliente que se instalará en cada puesto de atención, el mismo que se conectará al servidor y mostrará la información relacionada al puesto de atención tal como: Nombre del asesor, Identificativo del puesto de atención, turno asignado al asesor, tiempo de atención, opción para insertar un turno, opción para transferir a una cola diferente a la del turno actual, opción para conectarse al sistema, opción para desconectarse del sistema.

Aplicativo cliente que emitirá los turnos que se entregan al usuario que requiere ser atendido, el mismo que tendrá la opción para elegir el servicio deseado de acuerdo a la configuración solicitada por el cliente o institución donde se implante el sistema.

Aplicativo cliente que se encargará de mostrar el turno y el puesto de atención al cual debe dirigirse el usuario. La información receptada por este aplicativo se enviará a una pantalla LCD elegida para tal efecto.

- **Limitaciones**

La aplicación servidor estará desarrollada para funcionar en una sola agencia y las aplicaciones cliente para poder comunicarse con la aplicación servidor deben estar dentro de la intranet de la agencia y deben tener activado el protocolo TCP/IP .

D. JUSTIFICACIÓN

Las nuevas leyes impuestas a la importación de software extranjero, ha encarecido de manera significativa los costos de todo tipo de software proveniente del exterior, lo que ha hecho que la venta de software extranjero y particularmente de software de administración del flujo de clientes disminuya de manera significativa.

Desarrollar un sistema de administración del flujo de clientes que cumpla con las normas internacionales de calidad, permitirá disponer de un software de calidad y a costos adecuados para su venta tanto en el país como en el extranjero.

De igual manera permitirá bajar considerablemente los tiempos de respuesta relacionados al soporte y mantenimiento del sistema, ahorrando recursos tanto humanos como económicos.

Implantar un sistema de administración del flujo de clientes con soporte local permitirá optimizar el flujo de clientes, mejorar el servicio y crear una atmósfera más relajada tanto para el cliente como para el personal que está atendiendo. También optimiza los costos del personal, aumenta las ganancias y brinda a los usuarios de este tipo de sistemas un valor agregado al maximizar la rentabilidad de su empresa.

La plataforma J2EE permite desarrollar aplicaciones con arquitectura de N capas, basándose en componentes de software que se ejecutan sobre un servidor de aplicaciones.

La plataforma J2EE permite a los desarrolladores crear aplicaciones empresariales de manera más ágil, permitiéndoles aprovechar al máximo las bondades de los servidores de aplicaciones relacionadas con la velocidad, la seguridad y la fiabilidad.

Con J2EE es posible desarrollar todo tipo de aplicaciones informáticas, debido a que posee todas las especificaciones necesarias para desarrollar aplicaciones que se ejecutan en dispositivos móviles, computadores personales, etc.

Entre las especificaciones más importantes están las relacionadas con el manejo de los Enterprise JavaBeans, el acceso a las base de datos por medio de JDBC, Servlets para implementar funcionalidades en el lado del servidor, RMI para la comunicación remota entre servidores, JNDI para el manejo de servicios de directorio y la especificación JSF para la capa de presentación.

CAPÍTULO I

MARCO TEÓRICO

1.1 COLAS DE ESPERA

Las "colas de espera" son un aspecto de la vida cotidiana al que todas las personas se enfrentan todos los días, al momento de esperar el autobús, en los bancos, en los supermercados, en los hospitales, en todos los lugares donde se presta el servicio de atención al cliente están presentes las colas de espera.

Sin embargo la espera no es algo que a las personas les agrade. Las personas en la actualidad buscan un servicio rápido, eficiente y sin espera. Las organizaciones que hacen esperar a sus clientes corren el riesgo de perder a sus clientes, lo cual se traduce en perdida de dinero.

El problema de la espera tiende a agudizarse en aquellas organizaciones donde el proceso de atención es muy engorroso y no existe un mecanismo automático de gestión del flujo de clientes.

1.1.1 DEFINICIÓN

Según (Larry Ritzman, Lee Krajewski y Manoj Malhotra, 2008, pág. 292) "se conoce como filas de espera, a una hilera formada por uno o varios clientes que aguardan para recibir un servicio. Las filas de espera se forman a causa de un desequilibrio temporal entre la demanda de un servicio y la capacidad del sistema para suministrarlo. "

"Las situaciones de líneas de espera también se denominan problemas de Teoría de Colas, lo cual se debe al término británico "queue" que quiere decir cola" (Schroeder, 1995).

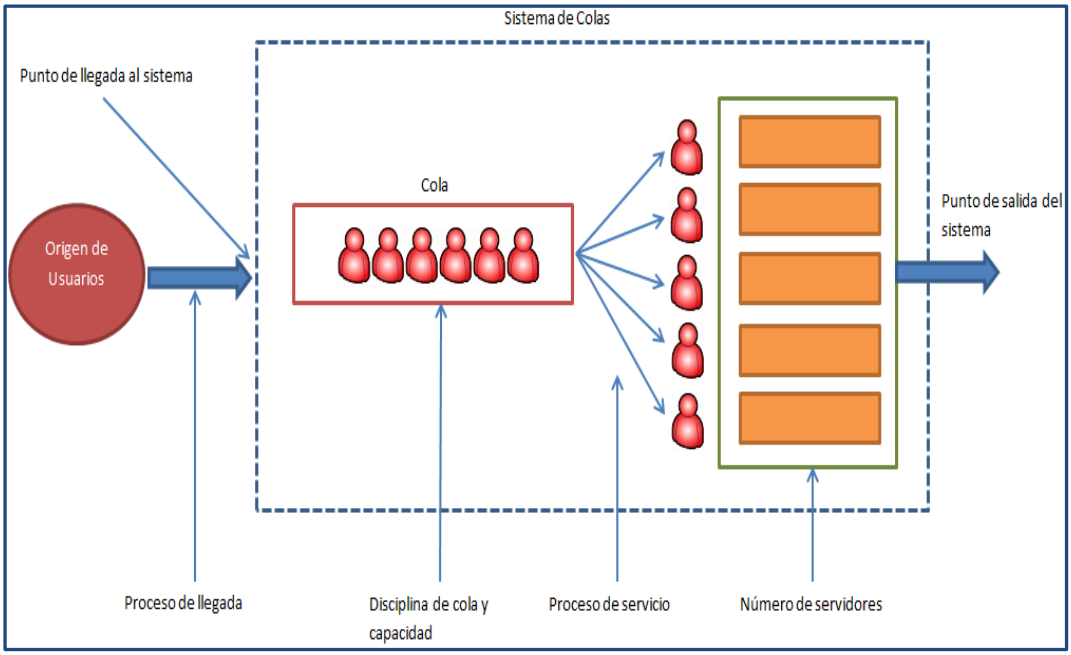

Figura 1.1 Modelo de un sistema de colas
Fuente: Propia

1.1.2 ANTECEDENTES HISTÓRICOS

El origen de la Teoría de Colas, según el enfoque actual, se encuentra en los trabajos de Agner Kraup en 1909 para analizar el tráfico telefónico o la congestión de llamadas, con el objetivo de cumplir la demanda incierta de servicios en el sistema telefónico de Copenhague. Sus investigaciones acabaron en una nueva teoría denominada teoría de colas o de líneas de espera. Esta teoría paso a ser una herramienta muy importante en simulación y ayudó a solucionar muchos problemas prácticos que tenían como característica llegadas y salidas.

1.1.3 TIPOS DE COLAS DE ESPERA

Las instalaciones donde se presta el servicio al cliente consisten en el personal y los equipos necesarios para proporcionar atención al cliente. Se debe elegir un modelo de Teoría de Colas adecuado tomando en cuenta el volumen de los clientes y el carácter de los servicios ofrecidos.

El prestador del servicio es conocido como canal, y las disposiciones del servicio o los pasos necesarios para proporcionar el servicio al cliente se conoce como fase.

La figura 1.2 muestra algunos ejemplos de los cinco tipos básicos de disposiciones para las instalaciones de servicio. Los gerentes deben elegir una disposición adecuada según el volumen de sus clientes y el carácter de los servicios ofrecidos. Algunos servicios requieren un solo paso, también conocido como fase, en tanto que otros requieren una secuencia de pasos.

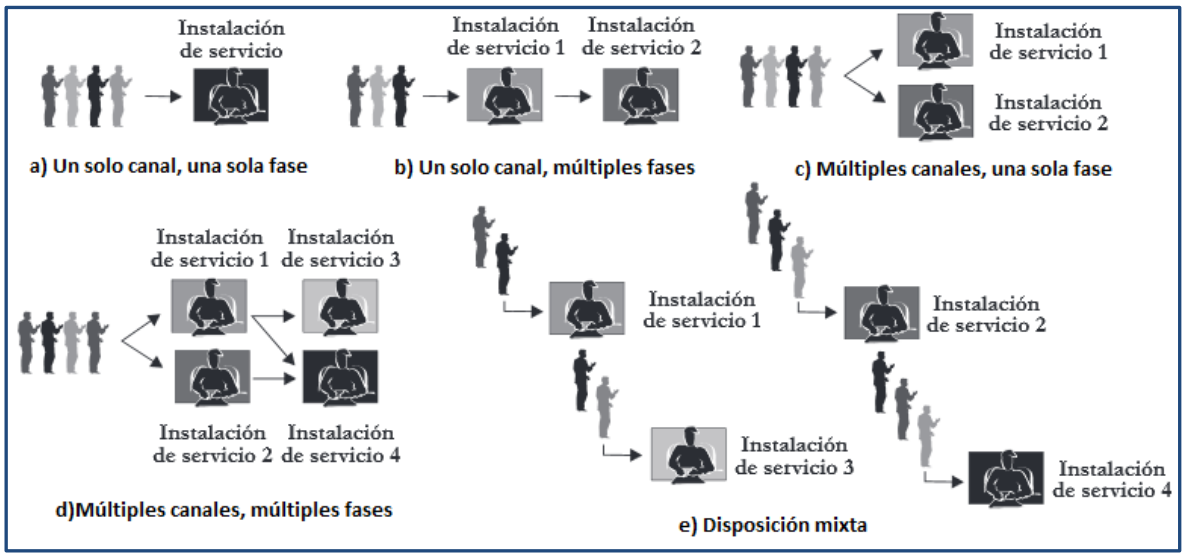

Figura 1.2 Ejemplos de disposiciones para instalaciones de servicio
Fuente: (Roberto Carro y Daniel Gonzáles, s.f, pág. 4)

1.1.3.1 MODELO SIMPLE DE TEORÍA DE COLAS

En el sistema de un solo canal y una sola fase, todos los servicios solicitados por un cliente son atendidos por un solo servidor. En este caso, los clientes forman una sola fila y van pasando al puesto de atención en el mismo orden en que fueron llegando. Ejemplo de este sistema son los autobancos donde los autos hacen fila mientras les toca el turno para pasar por el cajero.

Este modelo de colas debe tener las siguientes condiciones:

a) Las llegadas son atendidas sobre la base primero en entrar, primero en salir, y cada una de las llegadas espera el servicio, haciendo caso omiso de la longitud de la cola.
b) Cada entrada es independiente de la anterior.
c) Las llegadas son descritas por una distribución de probabilidad de Poisson y provienen de una población infinita.
d) Los tiempos de servicio varían de un cliente al siguiente y son independientes unos de los otros.
e) Los tiempos de servicio ocurren de acuerdo con la distribución de probabilidad exponencial.

El modelo simple, se conoce como M/M/1, es decir, que es un modelo de espera con llegadas aleatorias, distribución de servicio aleatorio y un sólo canal de servicio.

A partir de las condiciones anteriores se pueden desarrollar las siguientes ecuaciones:

p = utilización promedio del sistema = $\frac{\lambda}{\mu}$

Donde:

λ= número promedio de llegadas de clientes por periodo

μ= número promedio de clientes que completan el servicio por periodo

$P_{0=}$ probabilidad de que cero clientes estén en el sistema = 1 – λ/μ

$$Lq = \frac{\lambda^2}{\mu(\mu - \lambda)}$$

Donde:

$L_{q=}$ número promedio de clientes en la fila de espera

$$Ls = \frac{\lambda}{\mu - \lambda}$$

Donde:

$L_{s=}$ número promedio de clientes en el sistema

$$Wq = \frac{\lambda}{\mu(\mu - \lambda)}$$

Donde:

$W_{q=}$ tiempo promedio de espera en la fila

$$Ws = \frac{1}{\mu - \lambda}$$

Donde:

$W_{S=}$ tiempo promedio que un cliente pasa en el sistema

EJEMPLO:

Ecuauto proporciona un servicio de un solo canal de cambio de aceite y lubricante de automóviles. Las llegadas nuevas ocurren a una tasa de 3,5 automóviles por hora y la tasa media de servicio es de 6 automóviles por hora. Suponiendo que las llegadas siguen una distribución de probabilidad de

poisson y que los tiempos de servicio siguen una distribución de probabilidad exponencial.

a. ¿Cuál es la cantidad promedio de automóviles en el sistema?

b. ¿Cuál es el tiempo promedio que espera un automóvil para ser atendido?

c. ¿Cuál es el tiempo promedio que pasa un automóvil en el sistema?

Solución:

λ = 3,5 auto/hora

μ = 6 auto/hora

a)

$$L_q = \frac{\lambda^2}{\mu(\mu - \lambda)}$$

$$L_q = \frac{(3{,}5)^2}{6(6 - 3{,}5)}$$

$$L_q = 0{,}81$$

$$L_s = L_{q+}\frac{\lambda}{\mu}$$

$$L_s = 0{,}81 + \frac{3{,}5}{6}$$

$$L_s = 1{,}39$$

b)

$$W_q = \frac{L_q}{\lambda}$$

$$W_q = \frac{0{,}81}{3{,}5}$$

$$W_q = 0{,}2$$

c)

$$Ws = \frac{1}{\mu - \lambda}$$

$$Ws = \frac{1}{6 - 3{,}5}$$

$$Ws = 0{,}4 \text{ horas}$$

1.1.3.2 MODELO DE COLAS DE UN SOLO CANAL Y MÚLTIPLES FASES

En el sistema con un solo canal y múltiples fases los servicios se atienden en secuencia. Los clientes forman una sola fila y son atendidos en forma secuencial, pasando de una instalación de servicio a la siguiente. Un ejemplo de este sistema es el registro civil donde el primer puesto de atención revisa los documentos del cliente, el segundo puesto de atención toma la fotografía y finalmente el tercer puesto de atención entrega la cédula.

1.1.3.3 MODELO DE COLAS DE MÚLTIPLES CANALES Y UNA SOLA FASE

En el sistema de múltiples canales y una sola fase el mismo servicio es atendido por varios puestos de atención o cada puesto de atención atiende un servicio diferente. Los clientes forman una o varias filas dependiendo del diseño. En el diseño de una sola fila, los clientes son atendidos por el primer puesto de atención disponible como sucede en las cajas de los bancos. Si cada canal tiene su propia fila de espera, los clientes esperan hasta que el puesto de atención de su fila pueda atenderlos.

Las ecuaciones a utilizar en este tipo de modelo son más complejas que las del modelo M/M/1, este modelo se conoce como M/M/S, es decir, que es un modelo de espera con llegadas aleatorias, distribución de servicio aleatorio y múltiples canales de servicio.

$$p = \frac{\lambda}{s\mu}$$

$$P_0 = \frac{1}{\sum_{n=0}^{s-1}\left(\frac{\lambda}{\mu}\right)^n \frac{1}{n!} + \frac{1}{S!}\left(\frac{\lambda}{\mu}\right)^s \frac{1}{1-\frac{\lambda}{S\mu}}}$$

$$L_q = \frac{\left(\frac{\lambda}{\mu}\right)^{S+1} P_0}{(S-1)!\left(S-\frac{\lambda}{\mu}\right)^2}$$

$$W_q = \frac{L_q}{\lambda}$$

$$Ws = Wq + 1/u$$

$$Ls = \lambda(Ws)$$

En donde:

S= número de canales de servicio.

Estas fórmulas son para llegadas tipo Poisson, tiempo de servicio exponencial, se aplica la disciplina de atender primero al que llega primero, todas las llegadas esperan en la cola y la cola tiene longitud infinita.

1.1.3.4 MODELO DE COLAS DE MÚLTIPLES CANALES Y MÚLTIPLES FASES

En el sistema de múltiples canales y múltiples fases los clientes pueden ser atendidos por uno de los puestos de atención de la primera fase, pero después requieren ser atendidos por uno de los puestos de atención de la segunda fase, y así sucesivamente. Ejemplo de este sistema son las lavadoras donde la primera fase corresponde a las máquinas lavadoras y la segunda fase corresponde a las secadoras.

A continuación se presentan las siguientes figuras con los modelos de Teoría de Colas que según Barry Render, son los más comunes.

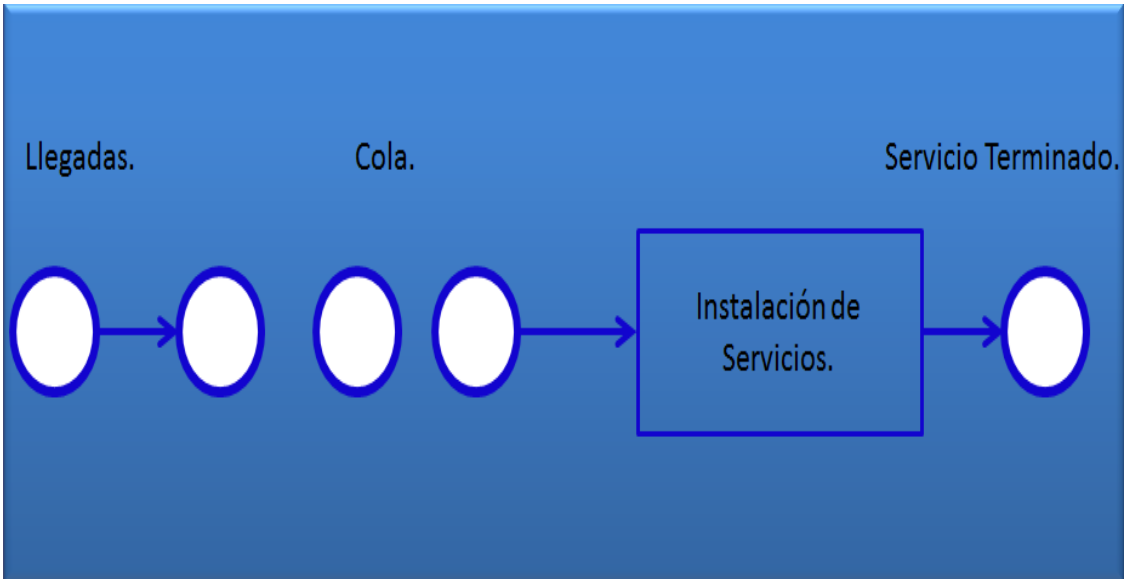

Figura 1.3 Un solo canal, una sola fase
Fuente: (Barry Render, 2012, pág. 504)

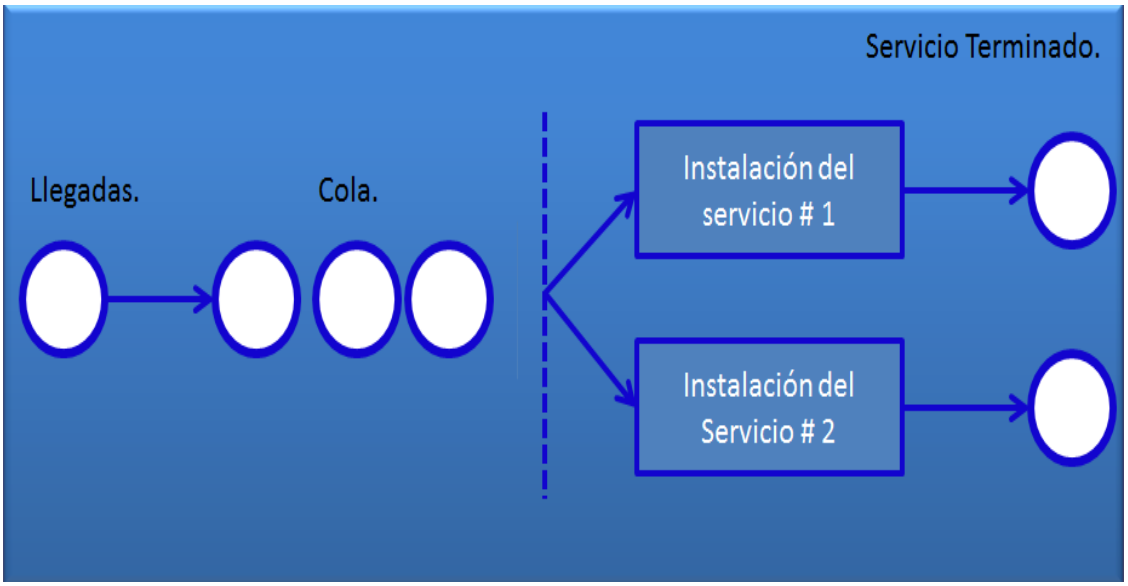

Figura 1.4 Canales múltiples, una sola fase

Fuente: (Barry Render, 2012, pág. 504)

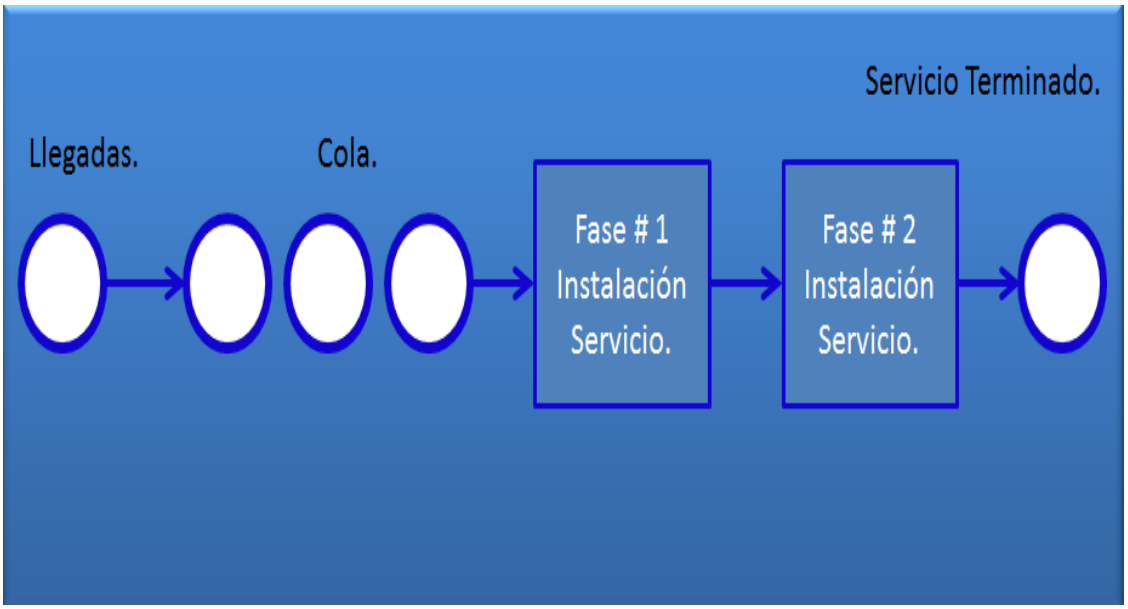

Figura 1.5 Un solo canal, fases múltiples

Fuente: (Barry Render, 2012, pág. 504)

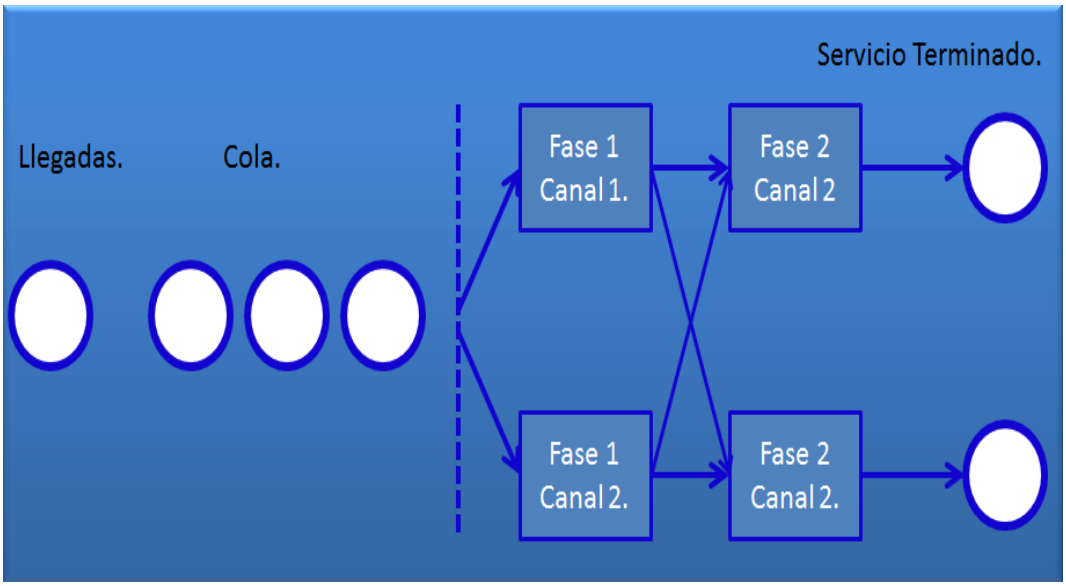

Figura 1.6 Canales múltiples, fases múltiples
Fuente: (Barry Render, 2012, pág. 504)

1.2 J2EE

La plataforma J2EE define un conjunto de contenedores, conectores y componentes necesarios para el diseño, desarrollo, implementación y despliegue de una aplicación empresarial independiente del sistema operativo en el que se instala.

La plataforma J2EE permite disponer de los siguientes recursos:

- Un modelo de desarrollo de componentes web entre los que se encuentran los Servlets, las páginas JSP, los componentes activos EJB los cuales se encuentran o se conocen como APIs de Java.
- Un conjunto de servicios que permiten realizar diferentes operaciones dentro de una aplicación web como son JDBC, JNDI, RMI, JavaMail, XML, etc.

- Un modelo para la creación de módulos en una aplicación web (.war), de módulos EJB (.jar) y de módulos corporativos (.ear), los cuales se asocian por medio de descriptores de despliegue en formato XML.

- Proporciona las herramientas que se necesitan para desarrollar aplicaciones J2EE así como los contenedores web y EJB que se necesitan para desarrollar los componentes mencionados anteriormente.

Entre las ventajas que se logran mediante el desarrollo de una aplicación en J2EE están las siguientes:

- Una aplicación que cumple con la especificación J2EE es independiente del sistema operativo en el que se ejecuta, pues solo será necesario tener instalada la máquina virtual de java para poder ejecutar la aplicación.

- Completo soporte para Servicios Web. La plataforma J2EE provee un framework para desarrollar y desplegar servicios web en la plataforma Java.

- El uso de contenedores permite simplificar el desarrollo, lo cual acelera el tiempo de entrega de una solución J2EE.

- La plataforma J2EE simplifica la conectividad entre aplicaciones y sistemas disponibles haciendo fácil la migración de una aplicación de escritorio a un ambiente web, a una aplicación para dispositivos móviles y otros dispositivos.

1.2.1 TECNOLOGÍAS J2EE

La plataforma J2EE se basa en una arquitectura multicapas, donde la lógica de la aplicación se divide en componentes de acuerdo a la función que desempeña y a los diferentes componentes que la conforman.

La especificación J2EE establece los siguientes componentes:

- Los componentes de la capa Cliente como los applets que corren en la máquina cliente.
- Los componentes de la capa Web como los servlets y JavaServer Pages que corren en el servidor de aplicaciones.
- Los componentes de la capa de negocio como los JavaBeans que corren en el servidor de aplicaciones.

Aunque una aplicación J2EE puede estar formada de tres de las cuatro capas que se muestran en la figura 1.7, las aplicaciones de tres capas son generalmente consideras aplicaciones J2EE multicapa porque están distribuidas sobre tres diferentes localizaciones: la máquina cliente, la máquina servidor J2EE y la máquina donde reside la base de datos.

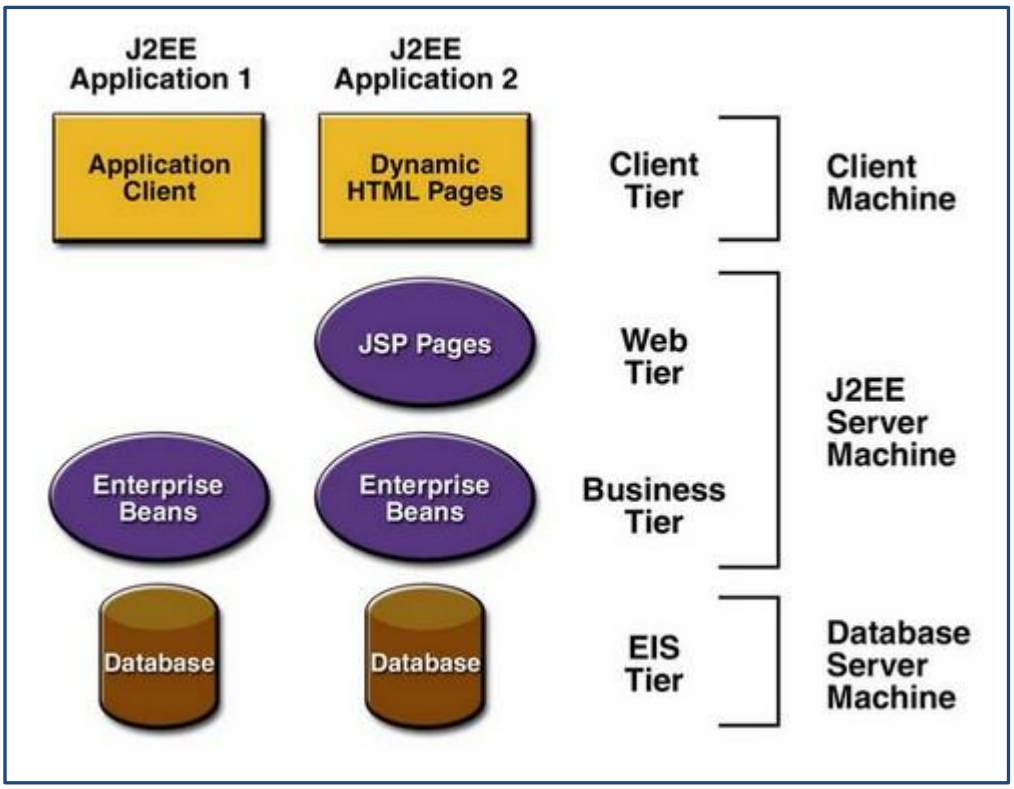

Figura 1.7 Aplicaciones Multitarea.

Fuente: Sun Microsystems. 2002. The J2EE Tutorial [Figura 1-1]

Todos estos componentes J2EE son ensamblados en una aplicación J2EE, donde el servidor de aplicaciones verifica que estén bien formados y cumplen con la especificación J2EE, se despliegan en el entorno de producción, donde se ejecutan y se gestionan por el servidor de aplicaciones J2EE.

1.2.1.1 CLIENTES J2EE

Un cliente J2EE puede ser un cliente web o una aplicación cliente.

Un cliente web está formado por páginas web dinámicas que son generadas por los componentes que corren en la capa web y los navegadores web que renderizan las páginas recibidas desde el servidor.

Algunas páginas web están compuestas por applets, lo cual puede ser una desventaja porque un sistema de este tipo necesita tener instalado un plugin de java en el navegador y configurar los archivos de seguridad para ejecutar applets en un navegador web correctamente.

Una aplicación cliente J2EE se ejecuta en una máquina cliente, típicamente tiene una interfaz gráfica de usuario pero también es posible una interfaz desde la línea de comandos.

1.2.1.2 COMPONENTES WEB

Las páginas JSP o los servlets que se ejecutan del lado del servidor son considerados como componentes web J2EE.

La especificación J2EE no considera a las paginas HTML como componentes J2EE al igual que a las clases de utilidades del lado del servidor.

1.2.1.3 COMPONENTES DE NEGOCIO

Es toda la parte lógica de una aplicación J2EE, que se encarga de resolver y satisfaces los requerimientos de un negocio en particular como la contabilidad, las colas de espera, la venta online, etc. Dicha lógica es gestionada por los Enterprise JavaBeans que se ejecutan en la capa de negocio.

Hay tres tipos de beans enterprise: beans de sesión, beans de entidad y beans manejados por mensajes.

Un bean de sesión encapsula lógica de negocio que puede ser invocada sobre un cliente local, un cliente remoto o un cliente de servicio web. Para acceder a una aplicación que se despliega en el servidor, el cliente invoca los métodos del bean de sesión, el bean de sesión realiza el trabajo para este cliente protegiéndolo de la complejidad mediante la ejecución de tareas de negocios dentro del servidor.

Un bean de entidad es un objeto remoto que administra datos persistentes, realiza lógica de negocios compleja utilizando varios objetos de java que pueden ser únicamente identificados por una clave primaria.

Un bean manejado por mensajes permite a una aplicación J2EE procesar mensajes de forma asíncrona. Este tipo de bean normalmente actúa como un oyente de Java Message Service (JMS) el cual es similar a un oyente de eventos pero recibe JMS en lugar de eventos. Los mensajes pueden ser enviados por algún componente J2EE o por una aplicación JMS o sistema que no usa tecnología J2EE.

1.2.1.4 CONTENEDORES J2EE

Los componentes basados en una arquitectura J2EE permiten desarrollar aplicaciones fáciles de escribir porque la lógica de negocio está organizada dentro de componentes reusables.

Los servidores J2EE proveen servicios subyacentes dentro de un contenedor para cada tipo de componente, lo cual permite al programador concentrarse en resolver los problemas de la lógica de negocio de una aplicación.

El proceso de implementación de una aplicación J2EE, instala componentes J2EE en los contenedores J2EE tal como se observa en la figura 1.8.

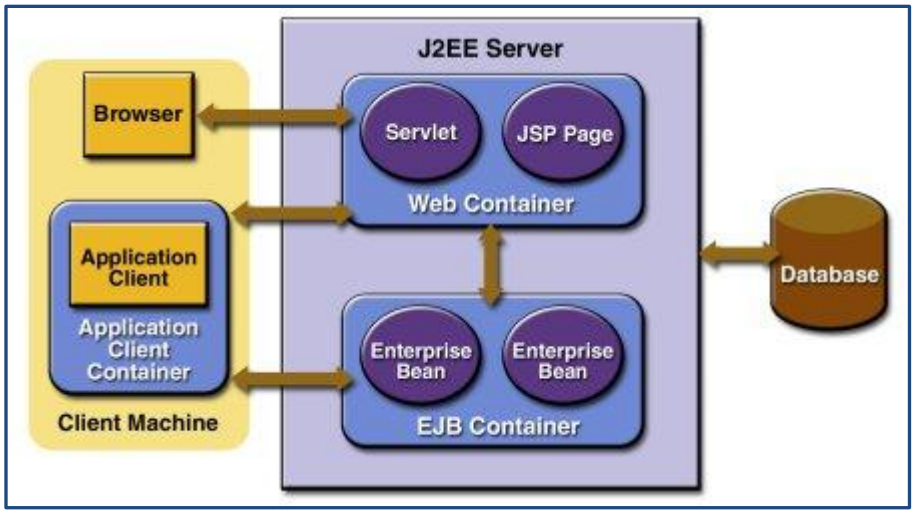

Figura 1.8 Servidores y Contenedores J2EE.
Fuente: Sun Microsystems. 2002. The J2EE Tutorial [Figura 1-5]

- El servidor J2EE corresponde a la parte en la que se ejecuta una aplicación J2EE. Un servidor J2EE provee contenedores web y EJB.

- Un contenedor web que gestiona la ejecución de una página JSP y los componentes servlets en aplicaciones J2EE.

- Un contenedor Enterprise JavaBeans que se encarga de gestionar la ejecución de los beans de empresa para aplicaciones J2EE.

- Un contenedor de aplicaciones cliente que gestiona la ejecución de componentes cliente.

- Un contenedor de applets que gestiona la ejecución de applets.

1.2.1.4.1 EMPAQUETADO

Para ser desplegados dentro de un contenedor, los componentes J2EE deben cumplir con ciertas normas de empaquetamiento.

En una aplicación web los componentes deben estar empaquetados sobre un fichero WAR el cual contiene clases y archivos utilizados en la capa web junto con un descriptor de despliegue de componentes web.

Para un componente EJB, el empaquetado deberá ser sobre un archivo EJB JAR el cual contiene los descriptores de despliegue EJB, los archivos del interface remoto más los archivos de ayuda que requiera el componente EJB.

Para una aplicación cliente el empaquetado deberá realizarse sobre un archivo JAR el cual podrá ser ejecutado sobre un contenedor o aplicación cliente.

1.2.1.5 ROLES DE LA PLATAFORMA J2EE

Durante la construcción, despliegue y uso de un componente EJB, diferentes personas ejecutan diferentes roles.
Los roles comunes involucrados en la construcción, despliegue y uso de un archivo EAR son los siguientes:

- El proveedor de productos J2EE es típicamente un servidor de aplicaciones, un servidor web o un sistema de base de datos quienes proveen una apropiada implementación que cumple con las especificaciones J2EE.

- La persona o empresa que provee componentes de aplicación como puede ser una aplicación EJB o una aplicación web.

- El profesional encargado de armar uno o más componentes J2EE dentro de un archivo de empresa EAR para crear una aplicación J2EE.

- El desarrollador de software se encarga de ensamblar los EJB JAR y los archivos WAR creados en la fase anterior dentro de un archivo de aplicación EAR, especifica el descriptor de despliegue y verifica que el contenido del archivo EAR cumpla con la especificación J2EE.

- El profesional encargado de garantizar el correcto funcionamiento del sistema.

- El proveedor de herramientas es la empresa o persona que crea aplicaciones, ensamblados, herramientas de empaquetado usadas por los proveedores de componentes, ensambladores y desarrolladores.

1.2.2 PATRONES DE DISEÑO

Un patrón provee una solución común a un problema común, por lo tanto dentro de una organización los patrones representan la codificación de una solución común proveniente de la experiencia previa.

Disponer de un buen patrón de diseño es como tener un amplio equipo de expertos que trabajan en conjunto durante el desarrollo de una aplicación.

Aplicar estos patrones en el desarrollo de una aplicación J2EE permite desarrollar aplicaciones de calidad y obtener mejores réditos económicos.

Los patrones de diseño se clasifican en los siguientes:

- Patrones Creacionales.
- Patrones Estructurales.
- Patrones de Comportamiento.

1.2.2.1 PATRONES CREACIONALES

Permiten solucionar los problemas de creación de instancias, encapsulan y abstraen dicha creación.

Fábrica Abstracta (*Abstract Factory*)

Permite crear conjuntos o familias de objetos que dependen mutuamente, sin especificar el objeto en concreto.

Método de Fabricación (Factory Method)

El patrón de diseño Factory se usa cuando en la aplicación se tiene una super clase con múltiples clases y basados en la entrada se requiere retornar una de las sub-clases.

Este patrón toma la responsabilidad de instanciación de una clase desde el programa cliente para la clase factory.

Factory Method hace también que el diseño de clases sea menos complejo para su modificación.

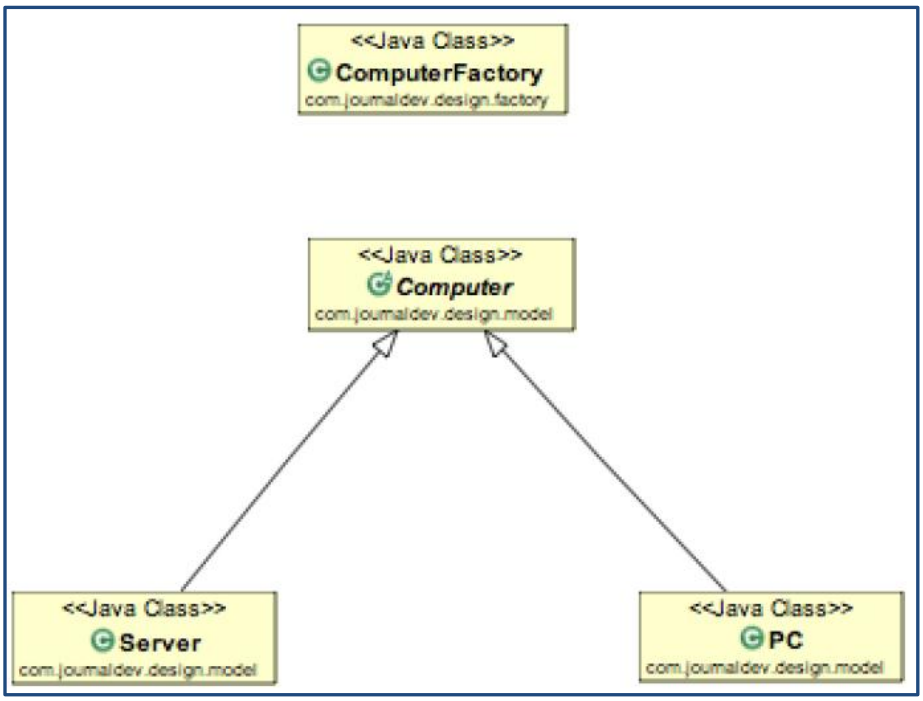

Figura 1.9 Factory Method.

Fuente: (journaldev.com, 2015, pág. 5)

Prototipado (*Prototype*)

Provee un mecanismo de creación de objetos. Es usado cuando al momento de crear un objeto resulta muy costoso y requiere mucho tiempo y recursos y se tiene un objeto similar que ya está creado. Este patrón provee un mecanismo para copiar el objeto original a un objeto nuevo y luego modificar este de acuerdo a lo que se desea.

Singleton

El patrón singleton restringe la instanciación de una clase y asegura que solo una instancia de la clase exista en la máquina virtual de java. La clase singleton debe proveer un punto de acceso global para obtener la instancia de la clase.

Para implementar el patrón singleton tenemos diferentes enfoques pero todos tienen conceptos comunes.

- Un constructor privado restringe la instanciación de la clase desde otras clases.
- Una variable privada estática de la misma clase que es la única instancia de la clase.
- Los métodos públicos estáticos que retornan la instancia de la clase, es el punto de acceso global para que las clases exteriores obtengan la instancia de la clase singleton.

MVC (Model View Controler)

El patrón de diseño MVC está diseñado para la arquitectura de aplicaciones web. Es un patrón ampliamente adoptado, a través de muchos lenguajes y frameworks implementados, cuyo propósito es lograr una separación limpia entre las tres capas de las cuales están conformadas la mayoría de aplicaciones web.

- Modelo que consta de la lógica de negocio y procesamiento de la información.
- Vista que es la capa que implementa la interfaz de usuario.
- Controlador que se encarga de la navegación y entrada de datos en la aplicación.

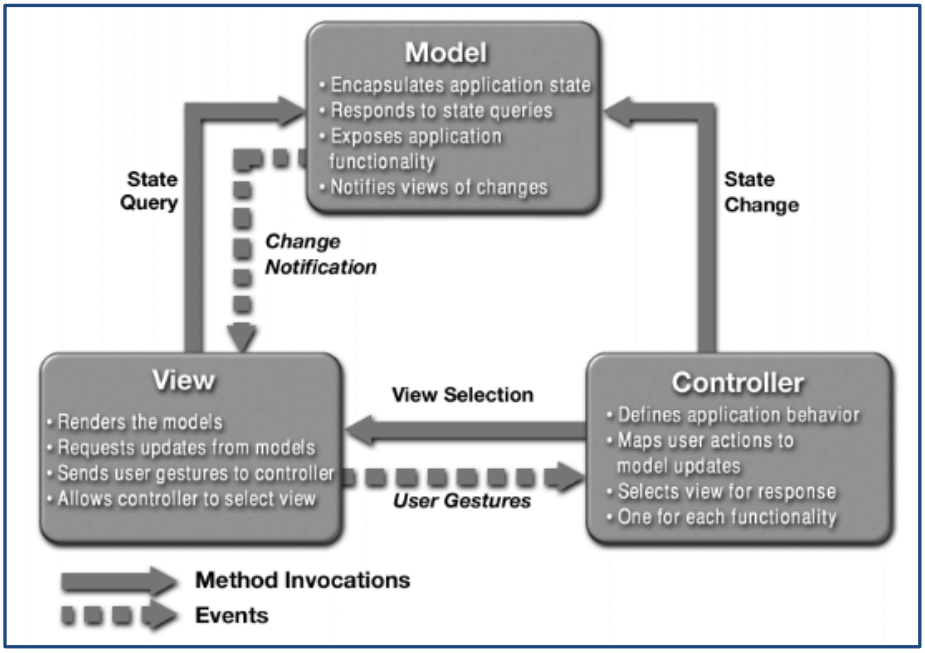

Figura 1.10 Estructura genérica patrón MVC.

Fuente: (wordpress.com, 2015, pág. 1)

1.2.2.2 PATRONES ESTRUCTURALES

Los patrones estructurales proporcionan diversas formas de crear una estructura de clases, por ejemplo utilizando la herencia y composición para crear un objeto grande desde objetos pequeños.

Adaptador (Adapter)

Este patrón es usado con el fin de que dos interfaces no relacionadas puedan trabajar juntas.

El propósito de este patrón es convertir la interfaz de una clase en otra interfaz que el cliente espera.

Puente (Bridge)

Es una técnica usada en programación para desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente sin necesidad de alterar por ello la otra.

Objeto Compuesto (Composite)

Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol.

Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.

Envoltorio (Decorator)

Responde a la necesidad de añadir dinámicamente funcionalidad a un objeto. Esto permite no tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera.

Fachada (Facade)

El propósito de este patrón es proporcionar una interfaz unificada a otro conjunto de interfaces de un subsistema. Facade define una interfaz de más alto nivel que hace que el subsistema sea más fácil de usar.

Peso Ligero (Flyweight)

Sirve para eliminar o reducir la redundancia cuando se tiene gran cantidad de objetos que contienen información idéntica, además de lograr un equilibrio entre flexibilidad y rendimiento (uso de recursos).

Apoderado (Proxy)

Tiene como propósito proporcionar un subrogado o intermediario de un objeto para controlar su acceso.

1.2.2.3 PATRONES DE COMPORTAMIENTO

Los patrones de comportamiento son aquellos que están relacionados con algoritmos y con la asignación de responsabilidades a los objetos. Describen no solamente patrones de objetos o de clases, sino que también engloban patrones de comunicación entre ellos. Al igual que los otros tipos de patrones, se pueden clasificar en función de que trabajen con clases (Template Method, Interpreter) u objetos (Chain of Responsability, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Visitor).

La variación de la encapsulación es la base de muchos patrones de comportamiento. Cuando un aspecto de un programa cambia frecuentemente, estos patrones trabajan con un objeto que encapsula dicho aspecto, teniendo que definir por tanto, una clase abstracta que describe la encapsulación del objeto.

1.2.3 SERVIDORES DE APLICACIONES

Un servidor de aplicaciones es un programa servidor que se encuentra instalado en un computador de una red distribuida, el mismo que proporciona la lógica de negocios que necesita una aplicación alojada en él. Frecuentemente los servidores de aplicaciones son vistos como parte de una aplicación de tres capas que consisten de un servidor de interfaz gráfica de usuario, un servidor de aplicaciones y un servidor de transacciones y base de datos.

Los servidores de aplicación brindan servicios que permiten soportar la ejecución y disponibilidad de las aplicaciones, ejecutan tareas que permiten mantener la seguridad de la información y mantener la persistencia de los datos.

Entre los servidores de aplicaciones más populares tenemos los siguientes:

- JBOOS
- WEBSPHERE
- GLASSFISH
- INTERNET INFORMATION SERVER
- APACHE TOMEE
- APACHE TOMCAT

1.2.4 HERRAMIENTAS DE DESARROLLO

La disponibilidad de Software Libre en Ecuador, ha causado gran interés en todas las instituciones públicas como privadas las cuales han incursionado en su uso de forma más frecuente.

Al ser el software libre una política de gobierno y de estado, ha hecho posible la creación de más empresas de desarrollo y capacitación en software libre por lo cual es más factible incursionar en este tipo de tecnología.

A continuación se hace una breve introducción hacia cada una de las herramientas con las cuales se desarrollará esta tesis.

1.2.4.1 NETBEANS IDE 8 COMO PLATAFORMA DE DESARROLLO

NetBeans IDE 8.0.1 es un entorno de desarrollo visual de código abierto que permite el desarrollo de aplicaciones en lenguaje Java, uno de los lenguajes de programación más poderosos del momento, convirtiéndose en el IDE principal para quienes están interesados en el desarrollo de aplicaciones multiplataforma.

Mediante NetBeans 8.0.1 es posible diseñar aplicaciones con solo arrastrar y soltar los componentes sobre la interfaz gráfica de una aplicación, lo cual permite el rápido desarrollo de aplicaciones no solo de Escritorio sino también aplicaciones Web, aplicaciones móviles sin que cambie la forma de programar. Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos

necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente.

1.2.4.2 JDK 1.8 COMO FRAMEWORK DE DESARROLLO

Java Development Kit o (**JDK**), es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red.

En la unidad de red se pueden tener las herramientas distribuidas en varias computadoras y trabajar como una sola aplicación.

1.2.4.3 APACHE TOMCAT 8 COMO SERVIDOR DE APLICACIONES

Si bien es cierto que Apache Tomcat es más conocido como un contenedor de servlets, las últimas versiones permiten configurarlo como un servidor de aplicaciones J2EE por lo cual este servidor es capaz de soportar todas las especificaciones de la plataforma J2EE.

Apache Tomcat permite la creación de aplicaciones empresariales con componentes web, transaccionales y de persistencia.

1.2.4.4 MICROSOFT SQL SERVER EXPRESS 2008 COMO BASE DE DATOS

Existen muchos servidores de base de datos en el mercado, algunos son de libre utilización como es el caso de PostgreSQL, MYSQL entre los más populares y existen los servidores de base de datos comerciales como Oracle 11g, Microsoft SQL Server 2008 entre los más populares, sin embargo por su

facilidad de uso, su potencia y por ser una de las base de datos más utilizadas en el mundo se ha elegido Microsoft SQL Server 2008.

Microsoft SQL Server 2008 permite el soporte de transacciones, soporta procedimientos almacenados, incluye también un entorno gráfico de administración que permite el uso de comandos DDL y DML gráficamente, permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red sólo acceden a la información, además permite administrar información de otros servidores de datos.

Microsoft SQL Server 2008 dispone de una versión reducida, llamada MSDE con el mismo motor de base de datos pero orientado a proyectos más pequeños, que en su versión 2008 pasa a ser el SQL Express Edition, que se distribuye en forma gratuita.

1.3 REQUERIMIENTOS DE HARDWARE Y SOFTWARE

1.3.1 HARDWARE

El hardware del sistema son todos los equipos con los cuales interactúa el sistema de gestión del flujo de clientes.

El sistema puede utilizar cualquier hardware independientemente de su marca, modelo o tamaño. Esta naturaleza no propietaria permite brindar un mejor mantenimiento del sistema en caso de que algún componente sufra algún desperfecto y sea necesario substituir algún dispositivo en caso de que sea necesario. El sistema puede utilizar cualquier hardware existente, o los

que están disponibles en el mercado, siempre y cuando cumplan con las especificaciones recomendadas y estén en buenas condiciones.

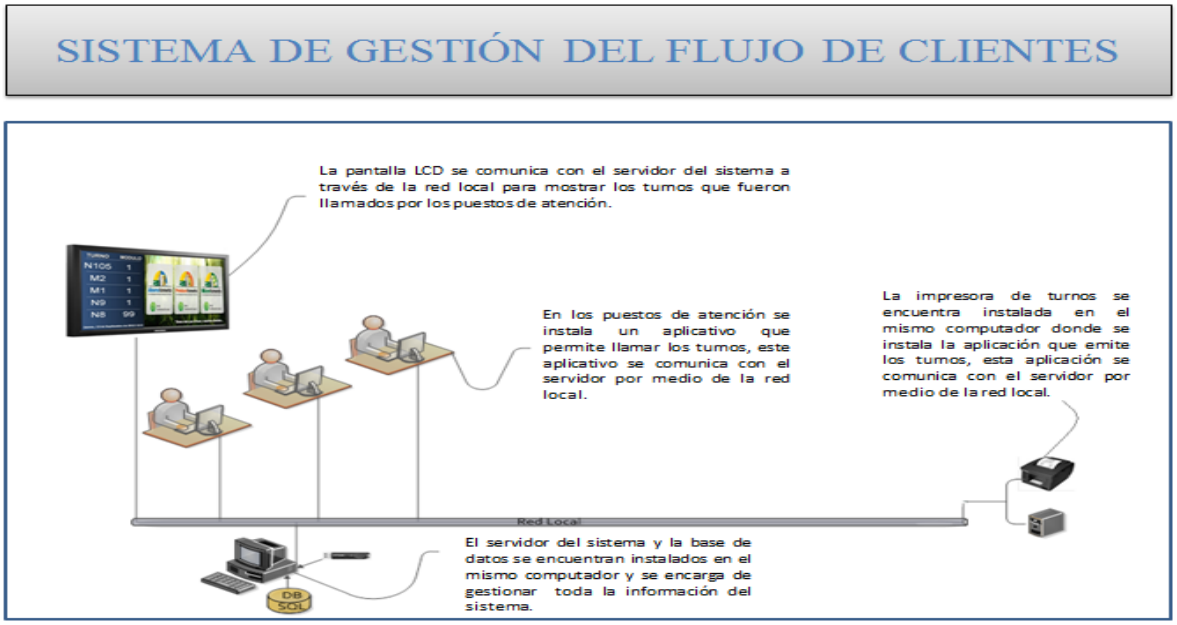

Figura 1.11 Componentes hardware de un sistema de gestión del flujo de clientes.

Fuente: Propia

1.3.1.1 IMPRESORA DE TICKETS

Una impresora de tickets es una impresora térmica la cual se basa en una serie de agujas calientes que van recorriendo un papel especial (termo sensible) que al contacto se vuelve de color negro.

La impresión térmica sólo posibilita la impresión en monocromo color negro, y únicamente en los modelos más recientes mediante un papel especial adicionalmente en rojo o azul. Por otro lado, los costos por copia son muy bajos ya que no consume más que el propio papel. La velocidad de impresión en este caso puede tener en mm/s, refiriéndose a los milímetros de rollo de papel que salen de la impresora. Oscila habitualmente entre 100 y 206 mm/s.

La durabilidad de la impresión es relativamente baja puesto que el desgaste que tiene el papel, en particular las temperaturas altas, hace que se pierda el texto o imagen escrito en el mismo.

Figura 1.12 Impresora de tickets.
Fuente: (epson.es, 2015, pág. 1)

1.3.1.2 IMPRESORA DE TICKETS EPSON TM-T88V

La impresora térmica TM-T88V permite la impresión de tickets de forma eficiente y a un bajo costo económico. Su diseño muy compacto permite reducir el espacio que ocupa en los kioscos o mostradores donde se las ubica.

Figura 1.13 Impresora de tickets Epson TM-T88V.
Fuente: (epson.es, 2012, pág. 3)

Entre las ventajas principales están las siguientes:

- Excelentes indicadores permiten determinar un alto rendimiento de la impresora Epson TM-T88V.

- El tiempo de duración alto demuestra que la impresora es muy amigable con el medio ambiente
- Bajo consumo de energía tanto en modo de funcionamiento como en modo de espera.
- Excelente velocidad de impresión tanto de texto como de gráficos.
- Alta calidad de impresión.
- Permite imprimir las imágenes en escala de grises, por lo cual es posible imprimir el logotipo de la oficina en el ticket.
- Fácil mantenimiento del hardware y del software.
- La instalación de la impresora es muy intuitivo por lo que cualquier persona puede realizar su instalación.

A continuación se detallan las especificaciones de la impresora Epson TM-T88V:

GENERAL	
Dimensiones del producto	145 x 195 x 148 mm (ancho x profundidad x altura)
Peso	1,6 kg
Color	Epson Dark Gray
Nivel de ruido	Operación: 55 dB (A)
Instalación	Horizontal, Vertical, Soporte para pared
Humedad del aire	Operación 10% - 90%, almacenamiento 10% - 90%
Temperatura	Operación 5°C - 45°C, almacenamiento -10°C - 50°C
FUENTES Y ESTILOS	

Velocidad de impresión	300 mm/s
Tamaño de letra	0,99 mm (ancho) x 2,4 mm (alto) / 1,41 mm (ancho) x 3,39 mm (alto)
Registro de caracteres	95 alfanumérico, 16 internacional, 128 × 43 gráfico
Caracteres por pulgada	20 cpp / 15 cpp
Densidad de puntos	180 ppp x 180 ppp
ENCENDIDO	
Fuente de alimentación	PS-180 (opción)
Consumo de energía	Standby: 0,1 A, Medio: 1,8 A
Tensión de servicio	24 V
INTERFACES	
Interfaces	USB 2.0 tipo B, Paralelo bidireccional

Tabla 1.1 Especificaciones de la impresora Epson TM-T88V.
Fuente: (Epson.es, 2015, pág. 2)

1.3.1.3 PANTALLA LCD

"Una pantalla de cristal líquido o LCD (sigla del inglés liquid crystal display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora." (Wikipedia, 2015, 1).

Las características principales de la tecnología LCD son las siguientes:

- Tamaño: Se mide en pulgadas y representa el espacio que hay entre la esquina superior izquierda y la esquina inferior derecha de la pantalla.
- Tecnología: Se la conoce como estática porque el computador es quien envía la señal cuando la pantalla debe cambiar de color.

- Resolución: es la cantidad de píxeles que contiene una pantalla LCD, entre mayor sea la cantidad mejor es la calidad de la imagen.

- Ancho de punto: Es el espacio que existe entre dos puntos del mismo color, entre menos sea el espacio mayor será la calidad de la imagen.

- Tiempo de respuesta: se mide en milisegundos y corresponde al tiempo que demora un píxel en cambiar de un color a otro.

- Tipo de matriz: Activa la cual ofrece una imagen más sensible a una amplia gama de colores.

- Angulo de visión: Corresponde a la dirección desde la cual se puede observar la pantalla sin que se pierda calidad de visión.

- Soporte de color: es la gama de colores que soporta la pantalla.

- Brillo: es la cantidad de luz que mite la pantalla hacia los ojos del observador.

- Contraste: Es la capacidad de la pantalla para mostrar su color más oscuro y su color más claro.

- Aspecto: es la proporción que existe entre el ancho y el alto de la pantalla y depende de la forma de la misma.

- Puertos de entrada: Son los dispositivos que permiten la comunicación entre la pantalla y el mundo exterior como por ejemplo los puertos VGA, HDMI, USB, etc.

Inconvenientes que presenta la tecnología LCD:

- Resolución: Las pantallas LCD muestran imágenes nítidas solo en su resolución máxima, si bajamos de resolución las imágenes se distorsionan.

- Contraste: Las pantallas LCD tienen menor contraste en comparación con otras tecnologías, lo cual reduce la habilidad de observar los detalles en una imagen.

- Tiempo de respuesta: Las pantallas LCD en comparación con otras tecnologías tienen bajos tiempos de respuesta lo cual provoca a veces la aparición de imágenes fantasmas.

- Durabilidad: El tiempo de duración de las pantallas LCD depende de la frecuencia de uso, lo cual provoca que las células de cristal se vayan desgastando.

1.3.1.4 PANTALLA LCD PROFESIONAL SAMSUNG SMT-3223

El monitor profesional SMT-3223 es un Full HD. Este monitor soporta hasta 1366 x 768 de resolución con un alto ratio de contraste de 3500:1. El SMT-3223 tiene un rápido tiempo de respuesta de 8 ms y de movimiento de 120 Hz.

Figura 1.14 Monitor Samsung SMT-3223 32".
Fuente: (samsung-security.com, 2012, pág. 1)

A continuación se detallan las especificaciones técnicas del monitor profesional Samsung SMT-3223:

PANTALLA		
Tamaño de pantalla	32"	
Máxima resolución	1,366 x 768	
Brillo	450cd/m2	
Contraste de radio	3,500 : 1	
Aspecto de radio	16:09	
Angulo de visión	178° / 178°	
Color de pantalla	16.7million	
Tiempo de respuesta	8ms (G to G)	
Sistema de video	8ms (G to G)	
Vida útil del panel	50,000hours	
Tipo de filtro	3D combfilter	
INTERFACE		
Video	Conector	CVBS, HDMI 1, HDMI 2, Component (D-sub 15pin)
RGB / DVI	Conector	Análogo D-sub, DVI-D, Puerto de pantalla
	Señal de entrada	0.7 Vp-p ±5%

	Formatos disponibles	640 x 350@70Hz
		720 x 400@70Hz
		640 x 480@60Hz/72Hz/75Hz (VGA)
		800 x 600@56Hz/60Hz/72Hz/75Hz (SVGA)
		1,024 x 768@60Hz/70Hz/75Hz (XGA)
		1,152 x 864@75Hz
		1,152 x 870@75Hz
		1,280 x 960@60Hz
		1,280 x 1,024@60Hz/75Hz (SXGA)
		1,360 x 768@60Hz
		1,366 x 768@60Hz
HDMI	Conector	HDMI 1, HDMI 2
	Formatos disponibles	640 x 480p@60Hz, 720 x 480i@60Hz, 720 x 480p@60Hz
		720 x 576i@50Hz, 720 x 576p@50Hz, 1,280 x 720p@50Hz/60Hz
		1,920 x 1,080i@50Hz/60Hz, 1,920 x 1,080p@50Hz/60Hz
Audio	Conector	RCA (L/R), Stereo mini jack
	Señal de salida	Loop-through line level (PC only), Speakers : 2 x 5W
Aplicación	Soporte	Control remoto
PRESENTACION EN PANTALLA		
Funciones	VESATM DPM compatible	
Lenguaje	Inglés, Español, Francés, Alemán, Italiano, Portugués, Ruso, etc.	

GENERAL		
Eléctrico	Voltaje de entrada	100 ~ 240V AC (50/60Hz)
	Consumo	Type 150W
Ambiente	Temperatura de operación	0 ~ +40°C (+32°F ~ +104°F)
	Humedad	10% ~ 80% (Non-condensing)
Mecánico	Dimensiones con la base	780.0 x 552.0 x 311.0mm
		(30.71" x 21.73" x 12.24")
	Dimensiones sin la base	780.0 x 482.0 x 109.5mm
		(30.71" x 18.98" x 4.31")
	Peso	13.7Kg (30.2 lb)
	Color de la carcasa	Negro
	Soporte de pared	Opcional

Tabla 1.2 Especificaciones monitor Samsung SMT-3223.
Fuente:https://www.samsungsecurity.com/samsung/upload/product_specifications/smt-3223-specifications.pdf

1.3.1.5 COMPUTADOR SERVIDOR DEL SISTEMA

En este computador se instalará el servidor del sistema para la gestión del flujo de clientes.

Este computador debe cumplir con los siguientes requerimientos de hardware:

Detalle	Procesador	Memoria RAM	Espacio en Disco	Tarjeta de red	Puerto USB
Sistema Operativo Windows	Intel® Core™2 Duo E8400 de 3 Ghz	4096 MB	30 GB	10/100	2.0
Base de Datos SQL Server Express 2008	Intel® Core™2 Duo E8400 de 3 Ghz	4096 MB	30 GB	10/100	2.0

Servidor de Aplicaciones	Intel® Core™2 Duo E8400 de 3 GHz	4096 MB	30 GB	10/100	2.0
JDK 1.8	Intel® Core™2 Duo E8400 de 3 GHz	4096 MB	30 GB	10/100	2.0

Tabla 1.3 Requerimientos de hardware para los componentes del servidor para la gestión del flujo de clientes.

Fuente: Propia

1.3.1.6 COMPUTADOR CLIENTE EMISOR DE TICKETS

En este computador se instalará el aplicativo encargado de emitir los tickets para los clientes, el hardware es el mismo sin importar el sistema operativo en el cual se instale.

Este computador debe cumplir con los siguientes requerimientos de hardware:

Detalle	Procesador	Memoria RAM	Espacio en Disco	Tarjeta de red	Puerto USB
Sistema Operativo Windows o Linux	Intel® Core™2 Duo E8400 de 3 GHz	4096 MB	30 GB	10/100	2.0

Tabla 1.4 Requerimientos de hardware del computador emisor de tickets

Fuente: Propia

1.3.1.7 COMPUTADOR CLIENTE PUESTO DE ATENCIÓN

En este computador se instalará el aplicativo encargado de llamar los turnos asignados a los clientes, el hardware es el mismo sin importar el sistema operativo en el cual se instale.

Este computador debe cumplir con los siguientes requerimientos de hardware:

Detalle	Procesador	Memoria RAM	Espacio en Disco	Tarjeta de red	Puerto USB
Sistema Operativo Windows o Linux	Intel® Core™2 Duo E8400 de 3 GHz	4096 MB	30 GB	10/100	2.0

Tabla 1.5 Requerimientos de hardware del computador puesto de atención
Fuente: Propia

1.3.1.8 COMPUTADOR CLIENTE QUE LEVANTA LA APLICACIÓN DISPLAY DE TURNOS

En este computador se instalará el aplicativo encargado de mostrar los turnos que fueron llamados hacia los puestos de atención en la pantalla LCD, el hardware es el mismo sin importar el sistema operativo en el cual se instale.

Este computador debe cumplir con los siguientes requerimientos de hardware:

Detalle	Procesador	Memoria RAM	Espacio en Disco	Tarjeta de red	Puerto USB
Sistema Operativo Windows o Linux	Intel® Core™2 Duo E8400 de 3 GHz	4096 MB	30 GB	10/100	2.0

Tabla 1.6 Requerimientos de hardware del computador display de turnos
Fuente: Propia

1.3.2 SOFTWARE

El software del sistema son todos los componentes lógicos necesarios para que funcione el sistema de gestión del flujo de clientes, si uno de estos componentes falta el sistema simplemente no funcionará.

El sistema puede instalarse en cualquier sistema operativo que permita instalar el JDK de Java.

1.3.2.1 SERVIDOR DEL SISTEMA

En el servidor del sistema de gestión del flujo de clientes se deben instalar todos los componentes necesarios para levantar la aplicación servidor. El servidor requiere que se instalen los siguientes componentes de software:

1. Sistema Operativo Windows.
2. Base de Datos SQL Server Express 2008.
3. Servidor de Aplicaciones Apache Tomcat 8.
4. JDK 1.8.

1.3.2.2 APLICATIVO CLIENTE EMISOR DE TICKETS

Este aplicativo será invocado desde cualquier browser que tenga conexión con el servidor del sistema. El cliente emisor de tickets requiere que se instalen los siguientes componentes de software:

1. Sistema Operativo Windows o Sistema Operativo Linux.
2. Browser Internet Explorer o Mozilla Firefox.

1.3.2.3 APLICATIVO CLIENTE PUESTO DE ATENCIÓN

Este aplicativo será invocado desde cualquier browser que tenga conexión con el servidor del sistema. El cliente Puesto de Atención requiere que se instalen los siguientes componentes de software:

1. Sistema Operativo Windows o Sistema Operativo Linux.
2. Browser Internet Explorer o Mozilla Firefox.

1.3.2.4 APLICATIVO CLIENTE DISPLAY

Este aplicativo será invocado desde cualquier browser que tenga conexión con el servidor del sistema. El cliente Display requiere que se instalen los siguientes componentes de software:

1. Sistema Operativo Windows o Sistema Operativo Linux.
2. Browser Internet Explorer o Mozilla Firefox.

CAPÍTULO II

APLICATIVO

2.1 Modelado

El modelado es una actividad que consiste en la simplificación de la realidad donde se omite lo no esencial y se fija solo en aquello que es más importante para plasmar las necesidades del usuario. Es la base de donde los programadores parten para construir con éxito la aplicación.

Existen muchos lenguajes y herramientas para modelar sistemas, sin embargo, él Lenguaje Unificado de Modelado UML se ha convertido en un estándar que se ha adoptado a nivel internacional por numerosos organismos y empresas para crear esquemas, diagramas y documentación relativa al desarrollo de software.

El entorno de desarrollo integrado Netbeans, él cual utilizaremos para el desarrollo del sistema, cuenta con plug-ins que facilitan la realización de los diagramas UML.

2.1.1 Estereotipos UML

Permiten representar una variación de un elemento existente que posee otra intención, o distinción de uso. La definición de un estereotipo se hace en forma explícita en la vista estática, mediante una relación de generalización con el elemento de UML que es base para su definición. El nombre del estereotipo debe ser distinto de los elementos de UML, y se denota entre

comillas francesas («nombre estereotipo»). También puede considerar una notación gráfica distintiva.

Un **estereotipo** es una subclase de un elemento existente con los mismos atributos y relaciones que ese elemento pero con una intención distinta y, posiblemente, restricciones adicionales, es decir para especializar un elemento.

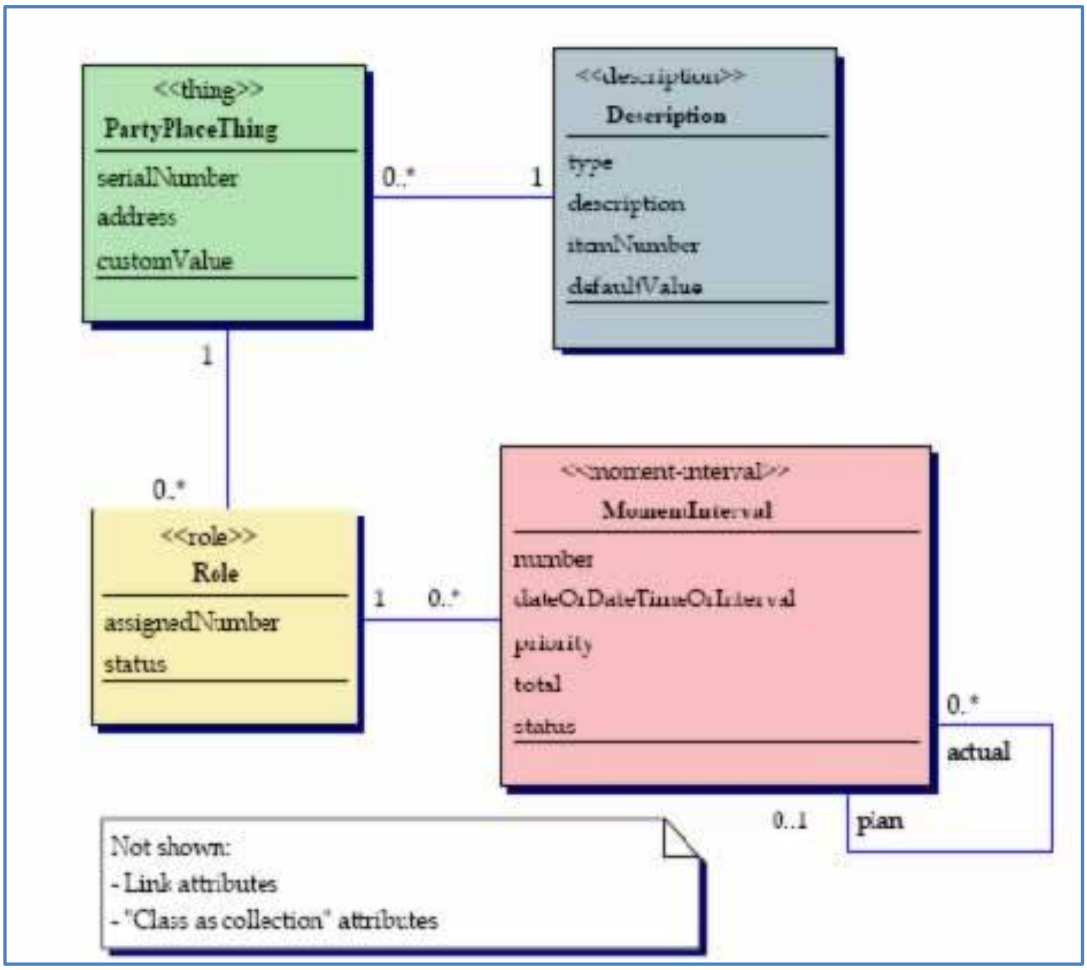

Figura 2.1 Ejemplo de UML con Estereotipos.
Fuente: (scribd.com, 2015, pág. 1)

Cada uno de estos cuatro colores corresponde a las características de un estereotipo, los atributos, enlaces, métodos, puntos de conexión e interacciones que las clases siguen.

Las características de un estereotipo incluyen atributos y enlaces. Una descripción (azul) define su tipo, descripción, número de artículo, y el valor(es) predefinido. Una parte, lugar o cosa (verde) tiene definido su número de serie, dirección, y valor(es) predefinido. Un rol (amarillo) define su número asignado y estado. Un moment-interval rosado sabe su número, fecha (u hora o intervalo de tiempo), su prioridad, su total, y su estado. Estos siguen el siguiente patrón: un azul enlaza a un verde, el verde enlaza a un amarillo, el amarillo enlaza a un rosado. A veces no necesitamos un verde y un amarillo en dicho caso se enlace del azul al rosado.

Un estereotipo representa una distinción de uso. Puede ser aplicado a cualquier elemento de modelado, incluyendo clases, paquetes, relaciones de herencia, etc. Por ejemplo, una clase con estereotipo \'actor\' es una clase usada como un agente externo en el modelado de negocio. Una clase patrón es modelada como una clase con estereotipo parametrizado, lo que significa que puede contener parámetros.

Para el desarrollo del sistema se han utilizado los siguientes estereotipos:

ESTEREOTIPO	SIGNIFICADO
EJB	Representa un componente Enterprise JavaBean asociado con un objeto de negocio.
SessionEJB	Representa un vean de sesión como un todo, sin especificar su interface remota, interface local o su implementación.
EntityEJB	Representa un bean de entidad como un todo, sin especificar su interface remota, interface local o su implementación.
View	Representa una Vista que corresponde a la información que se muestra al cliente.

JSP	Una página JSP implementada en java.
Servlet	Un controlador que es típicamente implementado como un servlet.
Singleton	Una clase que tiene una sola instancia en concordancia con el patrón singleton.
Custom Tag	Las etiquetas en JSP se usan para implementar los objetos Helper, como son los JavaBeans.

Tabla 2.1 Estereotipos UML utilizados para modelar el sistema.
Fuente: (Sun Microsystems. 2003. Core J2EE PATTERSNS [Tabla 5-5])

2.1.2 Arquitectura del Sistema QManagement

La arquitectura del Sistema QManagement es multicapas en la cual se identifican claramente las siguientes capas:

- Capa de acceso a los datos la cual está implementada con el framework Hibernate.

- La capa de presentación implementada en JSF la cual ofrece las APIs que se requieren para la presentación de la información así como la interacción con el usuario.

- La capa de negocio se encarga de la gestión de todos los eventos que se producen en el servidor debido a la interacción del usuario con el servidor del sistema por medio de las páginas JSF.

La capa de persistencia interactúa con la capa de negocio, y maneja la durabilidad de los datos en el tiempo, mediante un motor de base de datos, en nuestro caso SQL Server Express.

La figura 2.2 refleja el diagrama de despliegue, donde se puede observar los componentes del sistema QManagement, graficados como submódulos ya que la funcionalidad les permite operar de manera independiente del módulo principal, sin embargo están incluidos dentro del mismo proyecto.

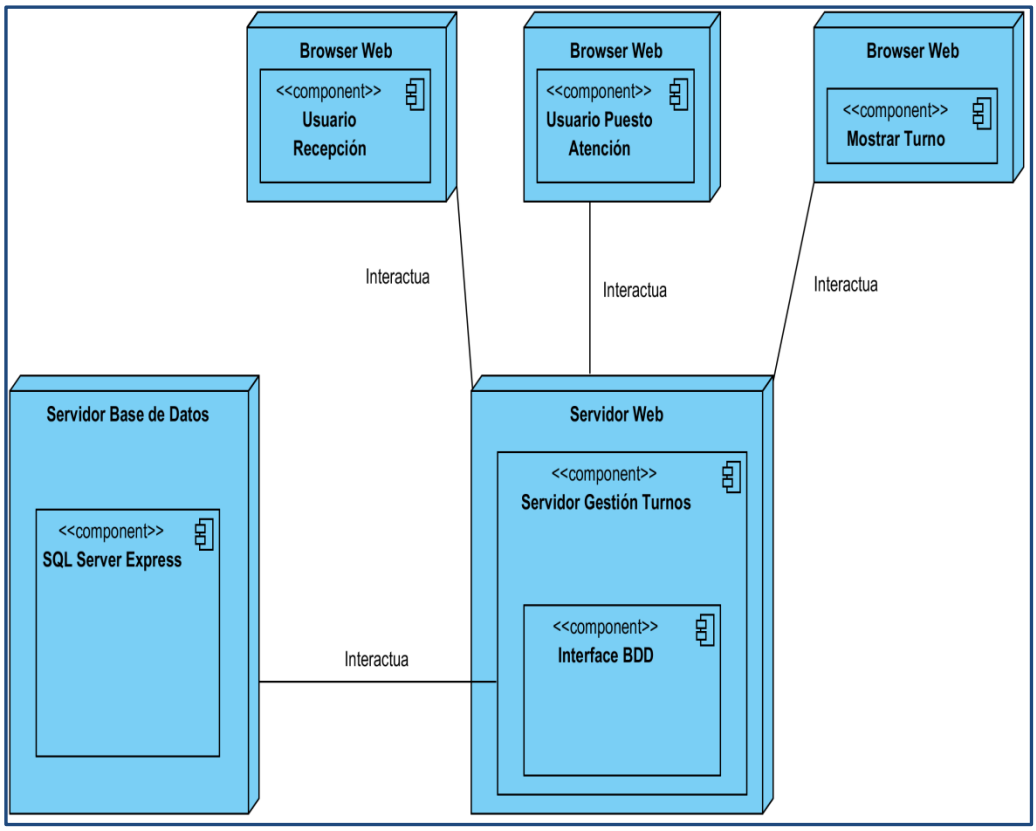

Figura 2.2 Diagrama de Despliegue QManagement
Fuente: Propia

Los submódulos del sistema Usuario Recepción, Usuario Puesto de Atención y Mostrar Turno interactúan con el módulo principal Servidor de Gestión de Turnos que es el que gestiona todo el proceso del turno.

El módulo principal también interactúa todo el tiempo con la base de datos SQL Server Express, de esa manera gestiona toda la información relacionada al flujo de atención del cliente.

2.1.3 Requerimientos del Módulo Servidor de Gestión de Turnos

Este es el principal módulo del sistema porque se encarga de la gestión de todos los servicios necesarios para el proceso de atención a clientes.

El objetivo de este módulo es garantizar la gestión completa del cliente que se presenta a la agencia para ser atendido, desde el momento en que el cliente llega a la agencia hasta el momento que finaliza su atención.

Para cumplir con este objetivo el sistema requiere de parte de los módulos cliente lo siguiente:

- Clave de acceso al sistema. El usuario que desea conectarse al sistema primero debe loguearse en el mismo con la clave asignada por el administrador del sistema.

- Servicio para el que se desea imprimir el turno. El usuario de recepción al momento de imprimir un turno solicita al cliente el servicio para el cual desea obtener el turno, en ese momento elige el servicio en la pantalla del módulo de recepción e imprime el turno.

- El ID del puesto de atención. Cuando el módulo de atención al cliente desea atender un turno, a más de loguearse al sistema debe enviar el ID del puesto de atención asignado por el sistema.

- La prioridad de atención. Cuando el módulo de atención al cliente desea atender un turno, a más de loguearse al sistema debe enviar el ID de la prioridad de espera asignada al puesto de atención por el sistema.

- El ID de la pantalla LCD que va a mostrar los turnos. El módulo que muestra los turnos en la pantalla LCD debe solicitar al servidor del sistema la información de los últimos 10 turnos que fueron llamados, para ello debe enviarle el ID de la pantalla que le fue asignado por el sistema.

- ID de la cola en la que se desea insertar un turno. Cuando el módulo de atención al cliente desea insertar un turno, a más de loguearse al sistema debe enviar el ID de la cola en la que desea insertar el turno.

- ID de la cola a la que desea transferir un turno. Cuando el módulo de atención al cliente desea transferir un turno, a más de loguearse al sistema debe enviar el ID de la cola a la que desea transferir el turno.

2.1.4 Arquitectura del Módulo Servidor de Gestión de Turnos

El servidor del sistema QManagement se ha desarrollado aplicando el patrón de arquitectura de software MVC que separa los datos y la lógica de negocio de la aplicación de la interfaz de usuario.

La capa de presentación está implementada con la tecnología JSF que provee un framework poderoso para el desarrollo de aplicaciones cliente servidor además de permitir una separación limpia entre la capa de presentación y la capa de negocios.

El controlador gestiona las acciones y eventos en la vista de acuerdo a cada interacción del usuario con la aplicación.

La figura 2.3 muestra la arquitectura de JavaServer Faces y como los componentes del sistema se integran en las capas de acuerdo al patrón Modelo-Vista-Controlador.

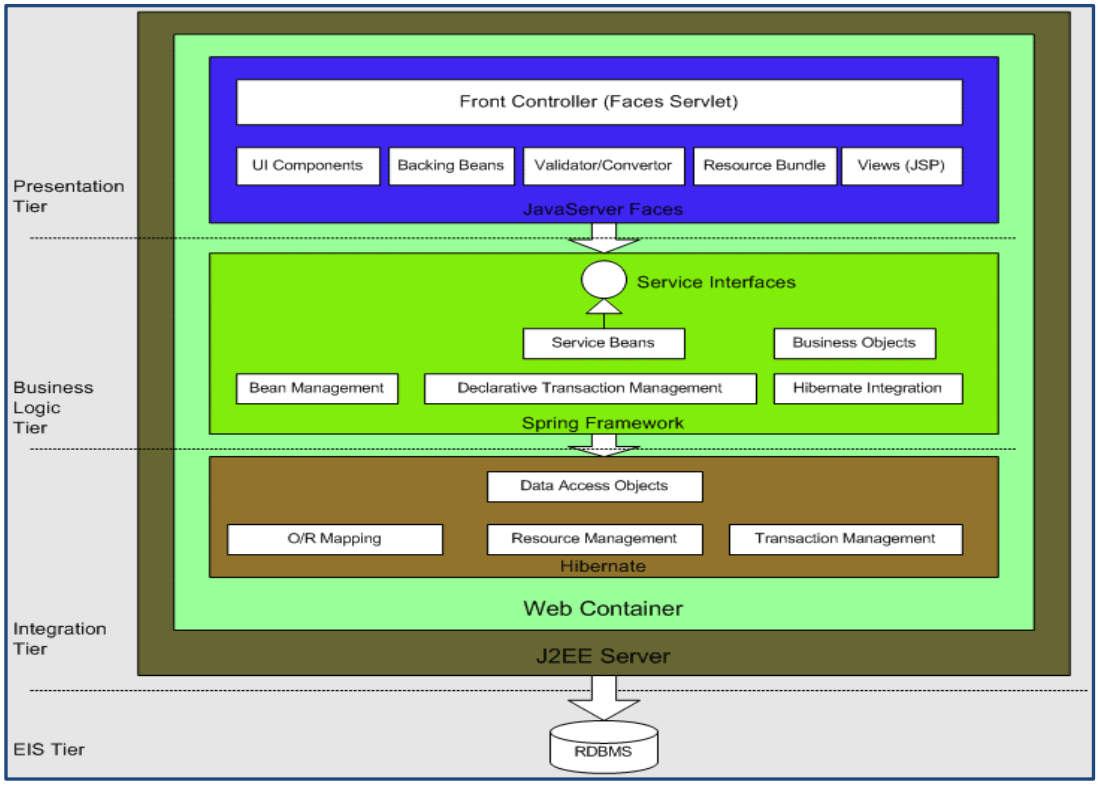

Figura 2.3 Arquitectura MVC de JSF.

Fuente: (programacion.net, 2015, pág. 1)

En la figura anterior podemos observar las tecnologías seleccionadas para cada capa de la arquitectura MVC y su ubicación dentro del contenedor web.

La arquitectura MVC hace que la aplicación sea más simple y escalable.

Se eligió JSF para la capa de presentación por ser una tecnología estándar del modelo MVC la cual nos permite desarrollar interfaces de usuario de manera profesional y de acuerdo a las especificaciones exigidas por la plataforma J2EE.

Para la capa de lógica de negocio hemos elegido POJO con la ayuda del framework spring el permite la integración entre los componentes de la aplicación.

La capa de integración maneja la persistencia de los datos con la base de datos relacional. Se pueden utilizar diferentes aproximaciones para implementar la capa de integración, en nuestro caso se eligió hibernate por ser fácil de implementar y altamente portable.

2.2 Desarrollo

2.2.1 Análisis y Diseño

Los Artefactos del Lenguaje Unificado de Modelado UML permiten explicar de manera clara la arquitectura por la cual se rige el sistema QManagement, donde se definen los requerimientos del sistema en Casos de Uso, se definen las clases que una vez implementadas permiten un funcionamiento óptimo de la aplicación.

2.2.1.1 Casos de Uso

Se utiliza los casos de uso para capturar información de cómo se desea que trabaje el sistema de gestión de clientes. También los casos de uso son necesarios para describir bajo la forma de acciones y reacciones el comportamiento del sistema desde el punto de vista del usuario.

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento del sistema mediante su interacción con los usuarios y los otros módulos del sistema.

2.2.1.1.1 Identificación de los Actores

Un actor es una persona o entidad externa al sistema que puede interactuar con él e interviene en los casos de uso para ejecutar las tareas.

Los actores identificados en el sistema son:

- Usuario Administrador del Sistema
- Usuario de Recepción
- Usuario Puesto de Atención
- Sistema Display Virtual
- Cliente
- Impresora
- Pantalla LCD

2.2.1.1.2 Lista de Casos de Uso

Se enuncia los casos de uso que intervienen en el sistema de acuerdo a cada módulo.

2.2.1.1.2.1 Sistema de Gestión del Cliente

- Gestión del Turno
- Validar Información

2.2.1.1.2.2 Usuario Administrador del Sistema

- Gestión de Prioridades
- Gestión de Servicios
- Gestión de Usuarios
- Gestión de Puestos de Atención

- Monitoreo de Servicios
- Monitoreo de Puestos de Atención
- Generar Alarmas
- Emisión de reports
- Validar Información

2.2.1.1.2.3 Usuario Puesto de Atención

- Login
- Llamar Turno
- Rellamar Turno
- Transferir Turno
- Insertar Turno
- Eliminar Turno
- Enviar Alarma
- Ver Espera
- Logoff
- Validar Información

2.2.1.1.2.4 Usuario de Recepción

- Seleccionar Servicio
- Obtiene Turno
- Imprimir turno
- Validar Información

2.2.1.1.2.5 Sistema Display Virtual

- Solicitar Turno
- Obtiene Turno
- Mostrar Turno

2.2.1.1.3 Descripción de Casos de Uso

CASO DE USO	Gestión del Turno
Nro. CU:	1
Descripción:	Permite la emisión, asignación, visualización de turnos y la finalización del mismo en el sistema de gestión del cliente.
Actores:	Sistema de Gestión del Cliente, Usuario de Recepción, Usuario Puesto de Atención, Sistema Display Virtual, Cliente, Impresora, Pantalla LCD.
Precondiciones:	El Usuario de Recepción debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario de recepción selecciona el servicio para el que el cliente ha solicitado un turno. 2. El sistema verifica que el servicio esté creado y disponible en la base de datos. 3. El sistema asigna automáticamente el turno que le corresponde a ese servicio. 4. El sistema almacena los datos seleccionados. 5. El usuario de recepción obtiene el turno que se asignará a un usuario de un puesto de atención y le entrega al cliente. 6. El sistema asigna el turno según el servicio al usuario del puesto de atención que está en estado de espera.	

7. El sistema display virtual presenta por pantalla el turno y el puesto de atención asignado por el sistema 8. El usuario del puesto de atención cambia al estado "Atendiendo". 9. El usuario del puesto de atención termina de atender al cliente. 10. El sistema finaliza el turno atendido.
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.
Poscondiciones: Se entrega el turno al cliente. El sistema almacena el servicio para el que se emitió el turno El sistema almacena el tiempo de espera del turno. El sistema almacena el tiempo de atención del turno. El sistema almacena el usuario del puesto de atención que atendió el turno.

Tabla 2.2: Caso de Uso: Gestión del Turno

Fuente: **Propia**

CASO DE USO	Gestión de Prioridades
Nro. CU:	2
Descripción:	Permite la creación, asignación, modificación y eliminación de prioridades de atención.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema.
Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de	

configuración de prioridades. 2. El sistema muestra las opciones disponibles en configuración de prioridades. 3. El usuario elige una de las opciones de configuración. 4. El sistema valida la configuración de la prioridad y la almacena. 5. El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.
Poscondiciones: El sistema almacena los datos de gestión de la prioridad. El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.

Tabla 2.3: Caso de Uso: Gestión de Prioridades
Fuente: **Propia**

CASO DE USO	Gestión de Servicios
Nro. CU:	3
Descripción:	Permite la creación, asignación, modificación y eliminación de servicios.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema.
Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de configuración de servicios. 2. El sistema muestra las opciones disponibles en configuración de servicios.	

3. El usuario elige una de las opciones de configuración. 4. El sistema valida la configuración del servicio y lo almacena. 5. El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.
Postcondiciones: El sistema almacena los datos de gestión del servicio El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.

Tabla 2.4: Caso de Uso: Gestión de Servicios
Fuente: **Propia**

CASO DE USO	Gestión de Usuarios
Nro. CU:	4
Descripción:	Permite la creación, asignación, modificación y eliminación de usuarios.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema.
Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de configuración de usuarios. 2. El sistema muestra las opciones disponibles en configuración de usuarios. 3. El usuario elige una de las opciones de configuración. 4. El sistema valida la configuración del usuario y lo almacena. 5. El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.	

Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.
Postcondiciones: El sistema almacena los datos de gestión del usuario. El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.

Tabla 2.5: Caso de Uso: Gestión de Usuarios
Fuente: **Propia**

CASO DE USO	Gestión de Puestos de Atención
Nro. CU:	5
Descripción:	Permite la creación, asignación, modificación y eliminación de puestos de atención.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema.
Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de configuración de puestos de atención. 2. El sistema muestra las opciones disponibles en configuración de puestos de atención. 3. El usuario elige una de las opciones de configuración. 4. El sistema valida la configuración del puesto de atención y lo almacena. 5. El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.	
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.	
Postcondiciones:	

El sistema almacena los datos de gestión del puesto de atención. El usuario administrador del sistema recibe el mensaje de gestión satisfactoria.

Tabla 2.6: Caso de Uso: Gestión de Puestos de Atención
Fuente: **Propia**

CASO DE USO	Monitoreo de Servicios
Nro. CU:	6
Descripción:	Permite la supervisión online de los servicios de atención.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema.
Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de monitoreo de servicios. 2. El usuario visualiza la pantalla de monitoreo de servicios.	
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.	
Postcondiciones: El usuario administrador del sistema visualiza la pantalla de monitoreo de servicios.	

Tabla 2.7: Caso de Uso: Monitoreo de Servicios
Fuente: **Propia**

CASO DE USO	Monitoreo de Puestos de Atención
Nro. CU:	7
Descripción:	Permite la supervisión online de los puestos de atención.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema.

Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de monitoreo de puestos de atención. 2. El usuario visualiza la pantalla de monitoreo de puestos de atención.	
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.	
Postcondiciones: El usuario administrador del sistema visualiza la pantalla de monitoreo de puestos de atención.	

Tabla 2.8: Caso de Uso: Monitoreo de Puestos de Atención
Fuente: **Propia**

CASO DE USO	Generar Alarmas
Nro. CU:	8
Descripción:	Permite él envió de alarmas desde el servidor del sistema hacia los puestos de atención.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema, Usuario de Puesto de Atención.
Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos. El puesto de atención debe estar logueado en el servidor del sistema.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de envío de alarmas. 2. El sistema muestra la pantalla de envío de alarmas	

3. El usuario administrador del sistema elige la prioridad y el puesto de atención al que se debe enviar la alarma. 4. El sistema envía la alarma hacia el puesto de atención elegido. 5. El puesto de atención recibe la alarma. 6. El servidor finaliza el envío de la alarma. 7. El usuario administrador del sistema recibe el mensaje de envío satisfactorio
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.
Postcondiciones: El puesto de atención recibe la alarma. El usuario administrador del sistema recibe el mensaje de envío satisfactorio.

Tabla 2.9: Caso de Uso: Generar Alarmas
Fuente: **Propia**

CASO DE USO	Emisión de Reportes
Nro. CU:	9
Descripción:	Permite la emisión e impresión de reportes estadísticos.
Actores:	Sistema de Gestión del Cliente, Usuario Administrador del Sistema.
Precondiciones:	El Usuario Administrador del Sistema debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos.
Flujo normal: 1. El usuario administrador del sistema ingresa en la opción de reportes. 2. El sistema muestra la pantalla de selección de reportes. 3. El usuario elige el reporte que necesita emitir e ingresa los datos de configuración. 4. El sistema valida la información de configuración del reporte. 5. El sistema emite el reporte elegido.	

6. El usuario visualiza el reporte en pantalla. 7. El usuario elige la opción de impresión del reporte. 8. El sistema envía a imprimir el reporte. 9. El usuario obtiene el reporte impreso.
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.
Postcondiciones: El usuario visualiza el reporte en pantalla. El usuario obtiene el reporte impreso.

Tabla 2.10: Caso de Uso: Emisión de Reportes
Fuente: **Propia**

CASO DE USO	Llamar un turno
Nro. CU:	10
Descripción:	Permite llamar, transferir o eliminar un turno.
Actores:	Sistema de Gestión del Cliente, Usuario Puesto de Atención.
Precondiciones:	El Usuario del puesto de atención debe ser validado por el Sistema de Gestión de Clientes. El usuario debe estar creado y estar activo en la base de datos. Deben existir turnos en la cola de espera.
Flujo normal: 1. El usuario del puesto de atención llama un turno. 2. El sistema verifica si existen turnos disponibles en el servicio asignado al puesto de atención. 3. El sistema asigna automáticamente el turno al puesto de atención. 4. El usuario del puesto de atención recibe el turno y espera la llegada del cliente. 5. El usuario del puesto de atención dependiendo del escenario puede	

transferir o eliminar el turno. 6. El usuario del puesto de atención finaliza la atención del turno. 7. El sistema almacena la información del turno.
Flujo alternativo: El usuario abandona el sistema cerrando la sesión en cualquier momento.
Postcondiciones: El sistema asigna el turno al puesto de atención. El puesto de atención finaliza la atención del turno. El sistema almacena la información del turno.

Tabla 2.11: Caso de Uso: Llamar un turno
Fuente: **Propia**

2.2.1.1.4 Descripción de Escenarios

Se describen los escenarios donde se desarrollan los casos de Uso, es decir, el lugar donde se va a implementar el sistema. Para el sistema existirá un único escenario.

Gestión de Turnos

La gestión del turno consiste en la asignación y direccionamiento automático de un turno de acuerdo a los servicios que presta la agencia de atención al público.

El proceso de atención inicia cuando un cliente solicita un turno de atención para un determinado servicio, el sistema le asigna un número secuencial, el cliente tendrá que esperar que un usuario de un puesto de atención este libre para que el sistema le asigne su turno y pueda ser atendido.

El usuario del puesto de atención puede cambiar su estado al momento de estar atendiendo a ocupado o libre.

2.2.1.1.5 Diagramas de Casos de Uso

Los diagramas de casos de uso permiten documentar el comportamiento de un sistema desde el punto de vista del usuario. Por lo tanto los casos de uso determinan los requisitos funcionales del sistema, es decir, representan las funciones que un sistema puede ejecutar.

2.2.1.1.5.1 Diagrama de Caso de Uso Gestión del Turno

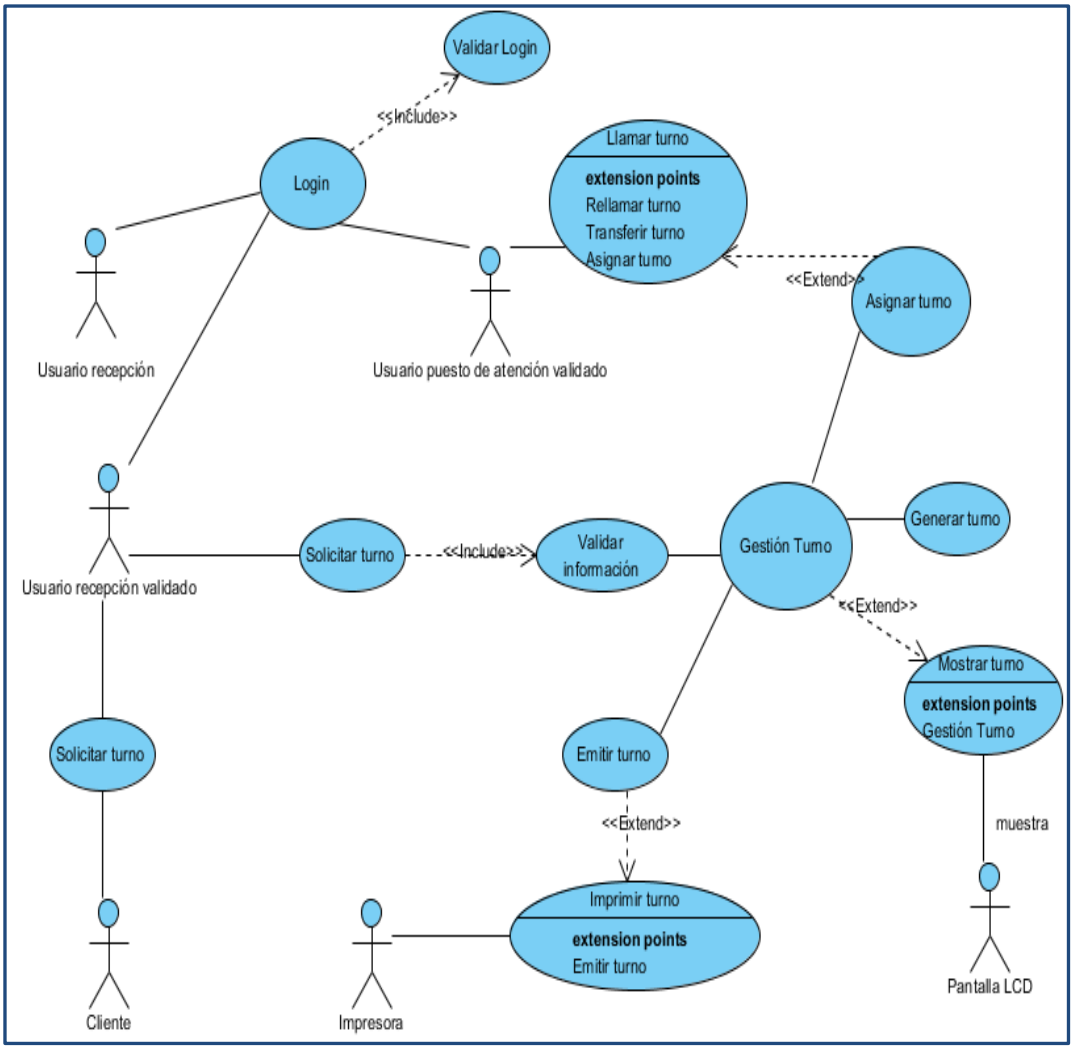

Figura 2.4 Caso de uso: Gestión del Turno
Fuente: Propia

2.2.1.1.5.2 Diagrama de Caso de Uso Gestión del Sistema

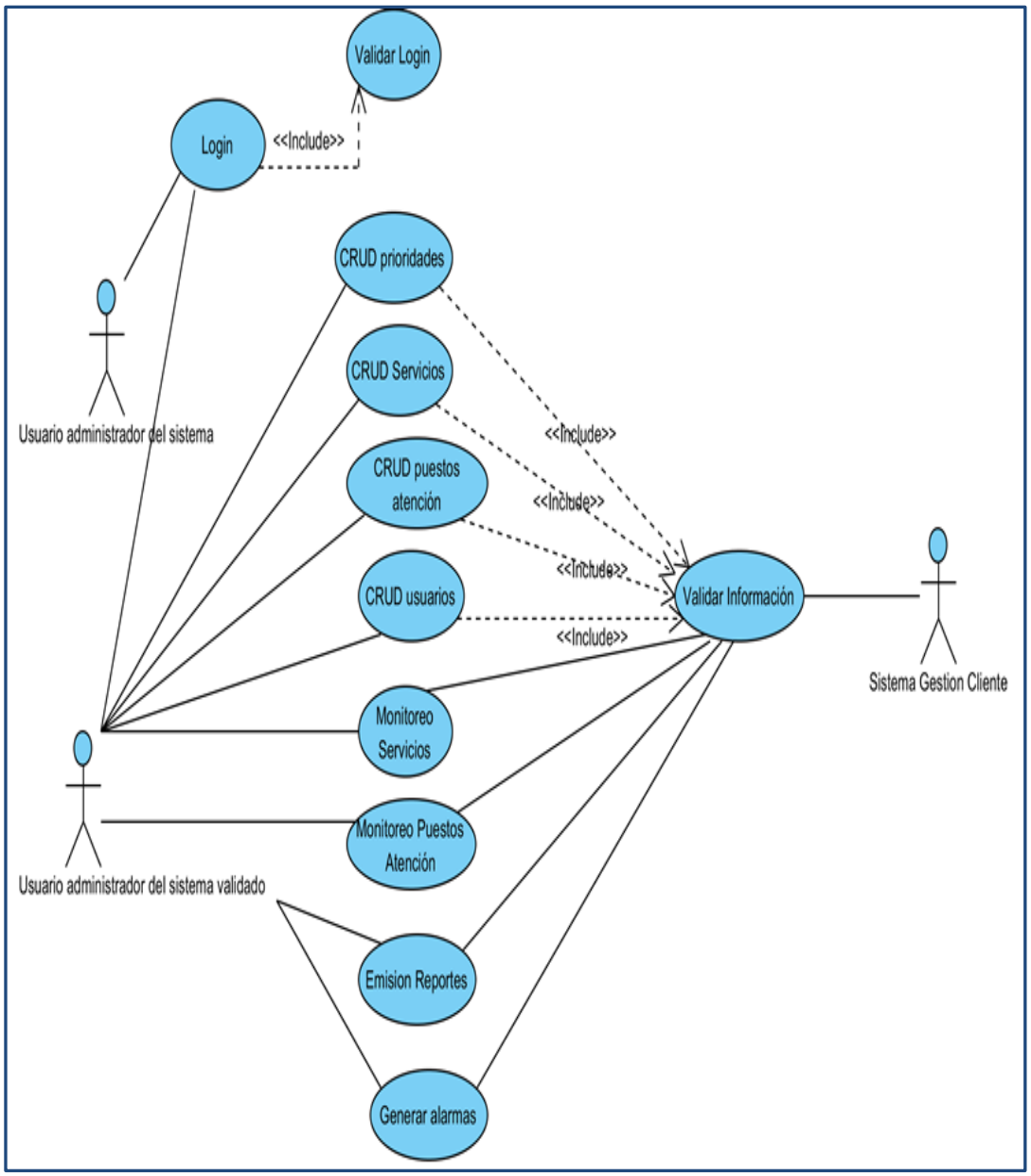

Figura 2.5 Caso de uso: Gestión del Sistema

Fuente: Propia

2.2.1.1.5.3 Diagrama de Caso de Uso Emitir Turno

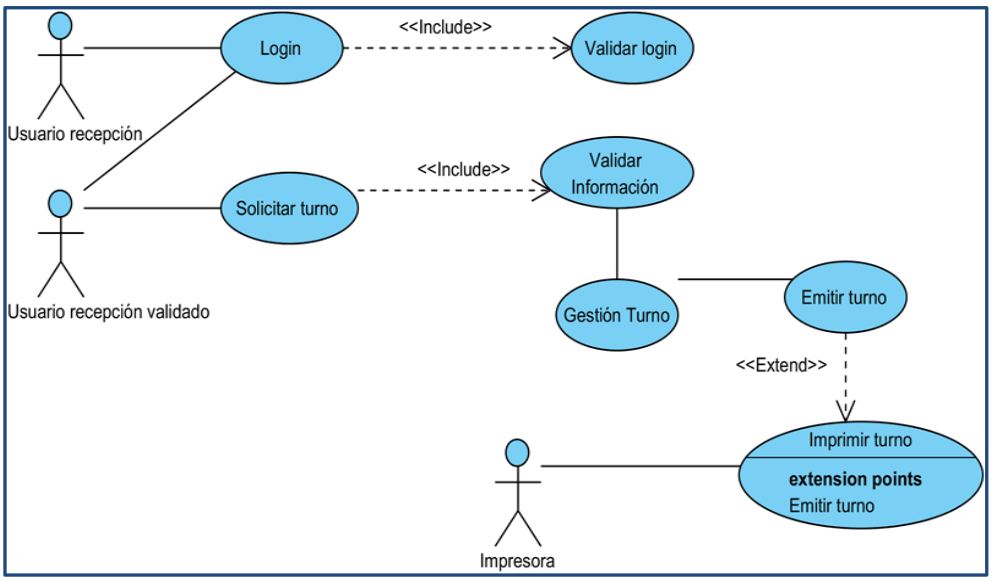

Figura 2.6 Caso de uso: Emitir Turno
Fuente: Propia

2.2.1.1.5.4 Diagrama de Caso de Uso Llamar Turno

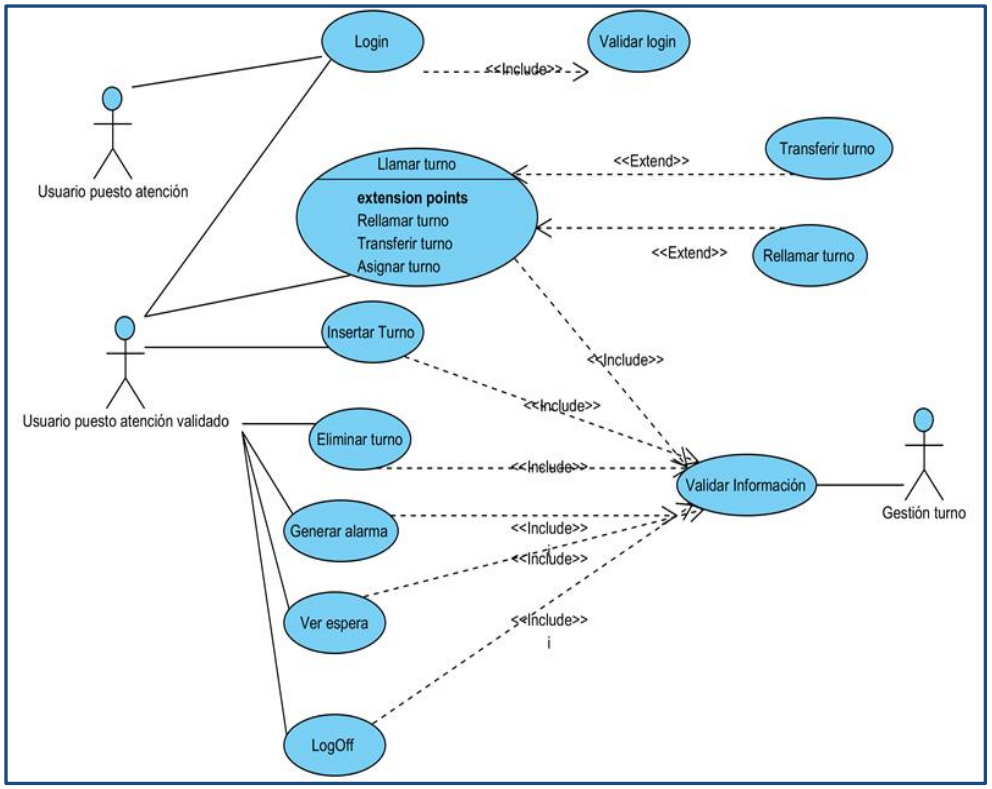

Figura 2.7 Caso de uso: Llamar Turno
Fuente: Propia

2.2.1.1.5.5 Diagrama de Caso de Uso Mostrar Turno

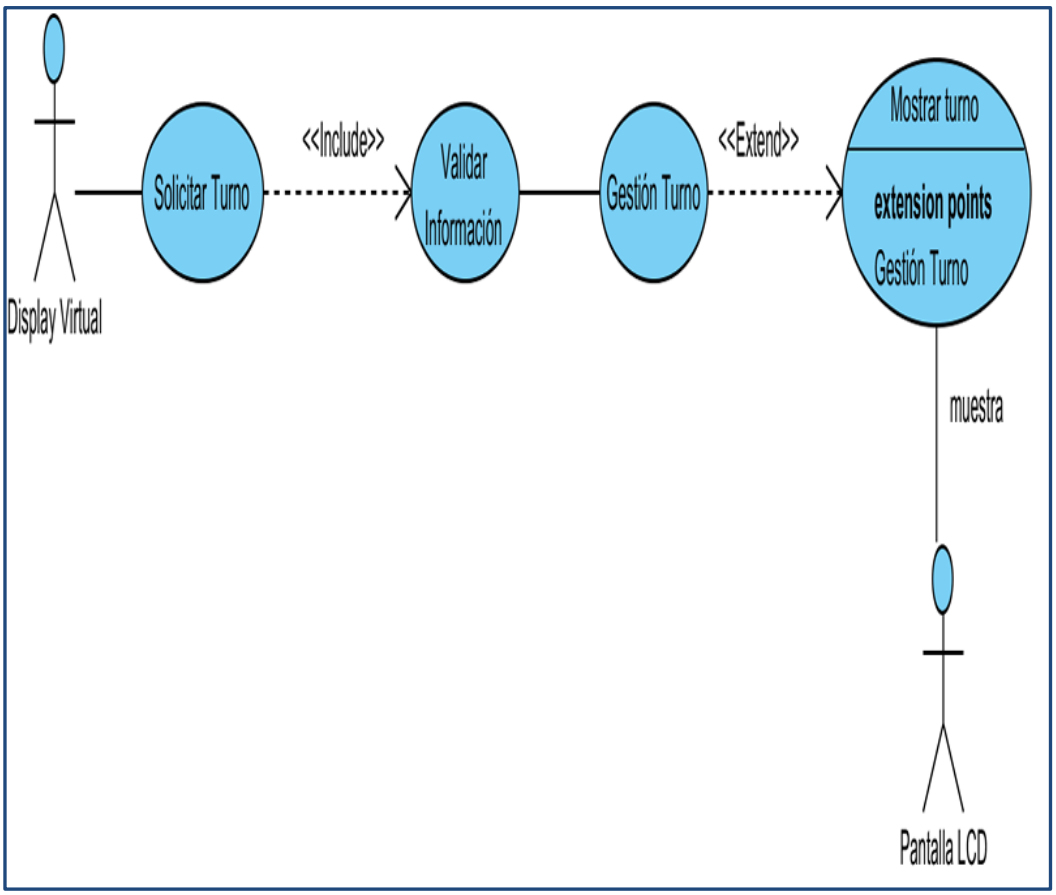

Figura 2.8 Caso de uso: Mostrar Turno

Fuente: Propia

2.2.1.1.6 Diagramas de Secuencia

En estos diagramas mostramos en forma secuencial los mensajes que intercambian los objetos de la aplicación de una forma ordenada.

En estos diagramas el tiempo fluye hacia abajo y muestra el flujo de control de un actor a otro.

En la siguiente figura se muestra el diagrama de secuencia genérico que se aplica para todo el sistema:

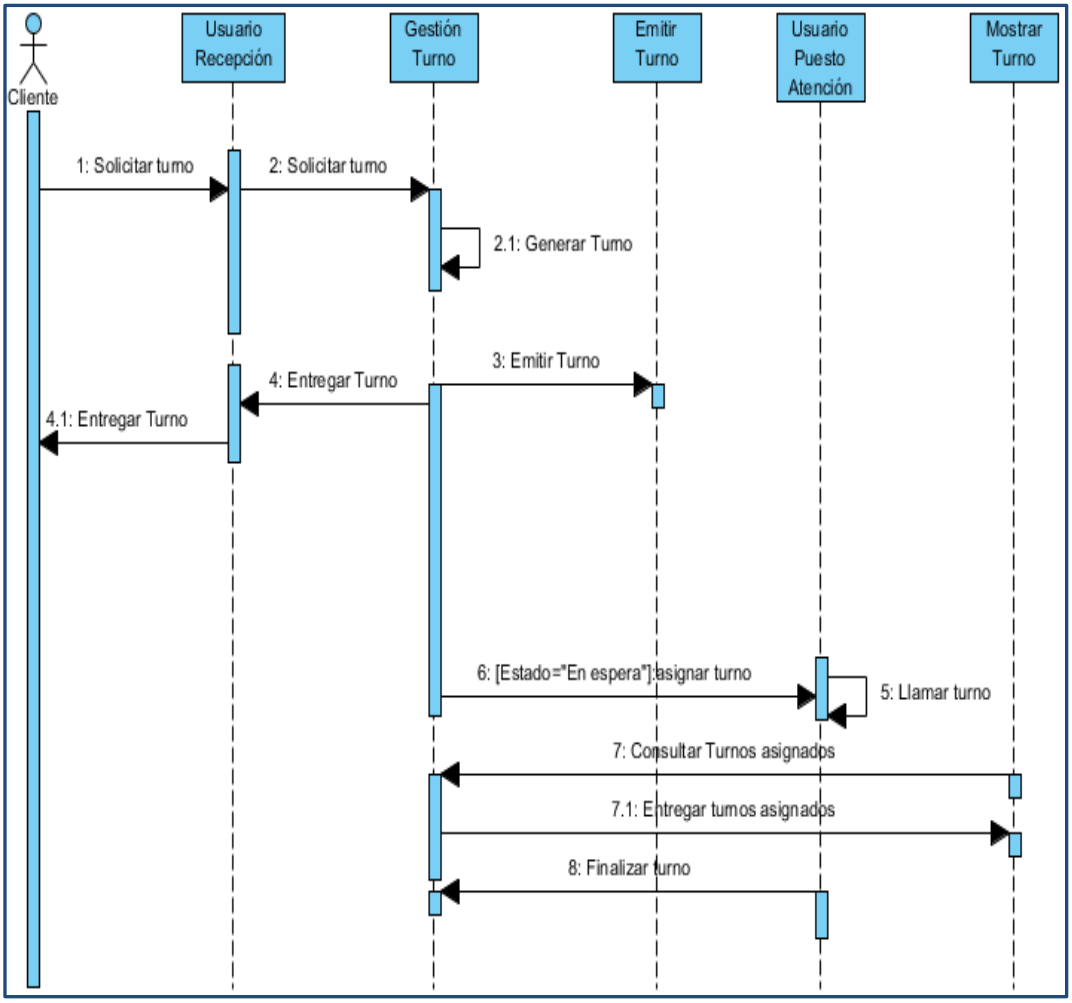

Figura 2.9 Diagrama de Secuencia: Gestión Turno

Fuente: Propia

2.2.1.1.7 Diagramas de Comunicación

En estos diagramas se muestra las interacciones organizadas alrededor de los objetos. Estos diagramas permiten mostrar explícitamente las relaciones de un objeto con respecto de los demás.

No muestran el tiempo como una dimensión aparte, por lo que resulta necesario etiquetar con números de secuencia tanto la secuencia de mensajes como los hilos concurrentes.

En la siguiente figura se muestra el diagrama de comunicación genérico que se aplica para todo el sistema:

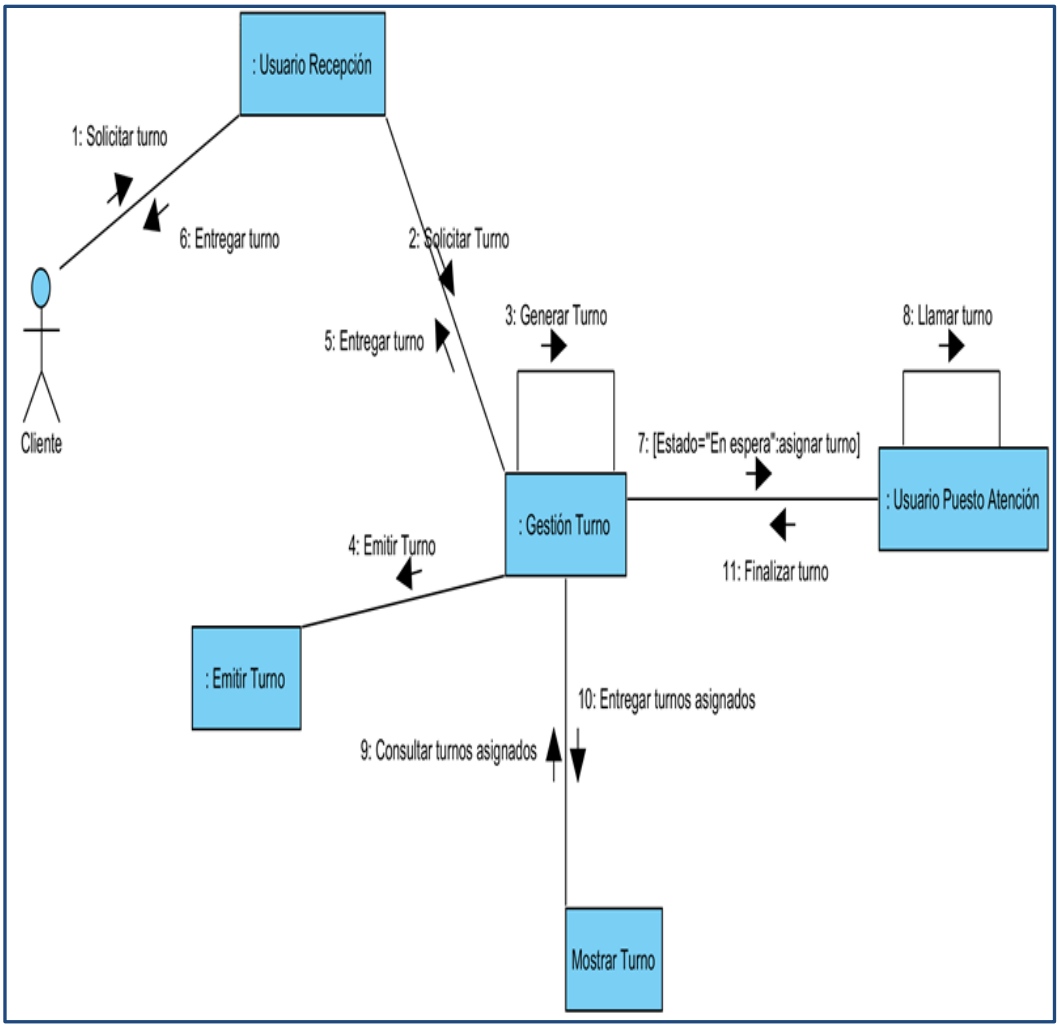

Figura 2.10 Diagrama de Comunicación: Gestión Turno

Fuente: Propia

2.2.1.2 Diagrama Entidad Relación

El diseño del Sistema de Gestión de Clientes en la base de datos se lo realiza mediante el diagrama Entidad Relación, el cual permite representar de manera gráfica las relaciones que existen entre los componentes del sistema.

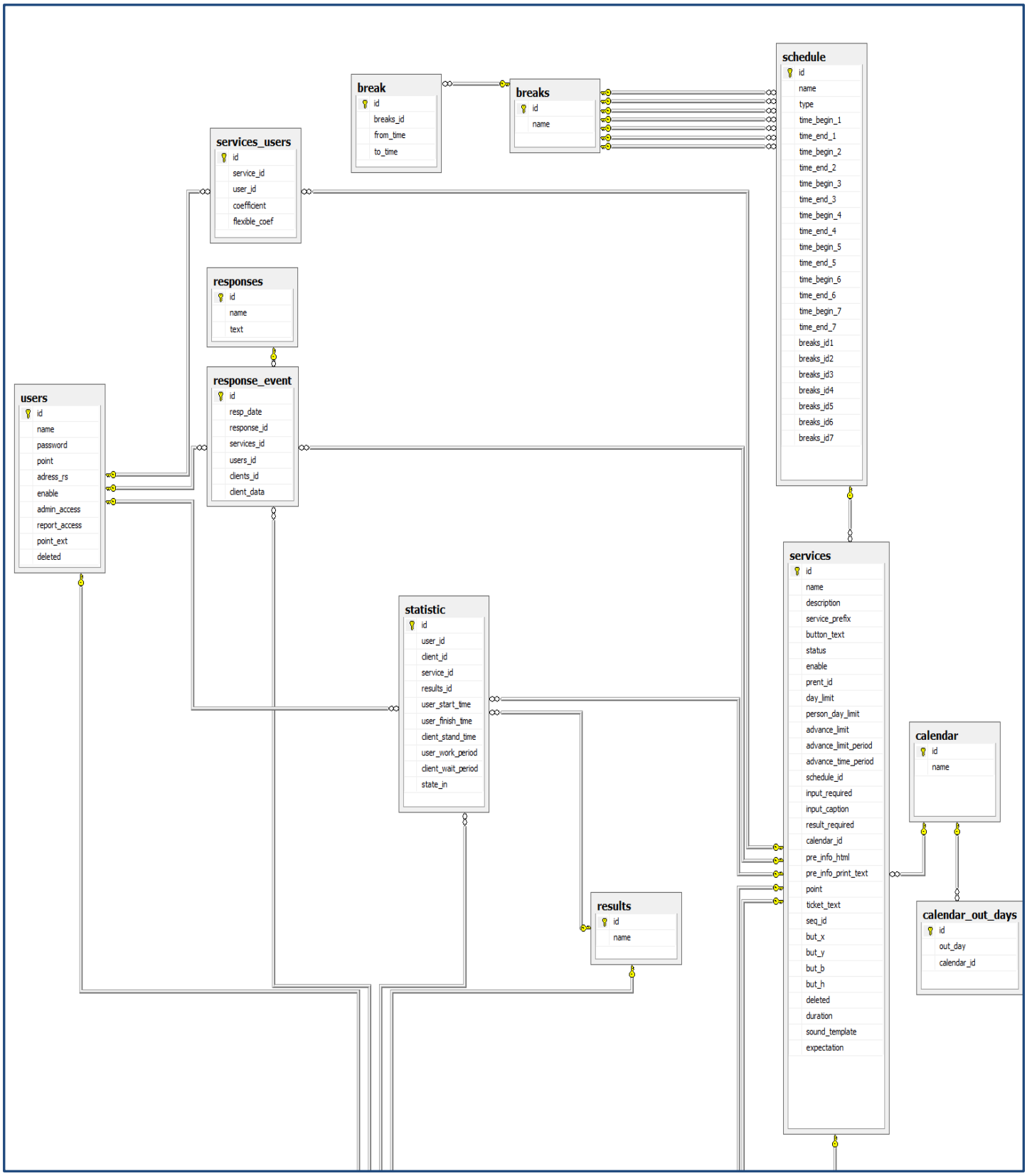

Figura 2.11 Diagrama Entidad-Relación: Parte 1

Fuente: Propia

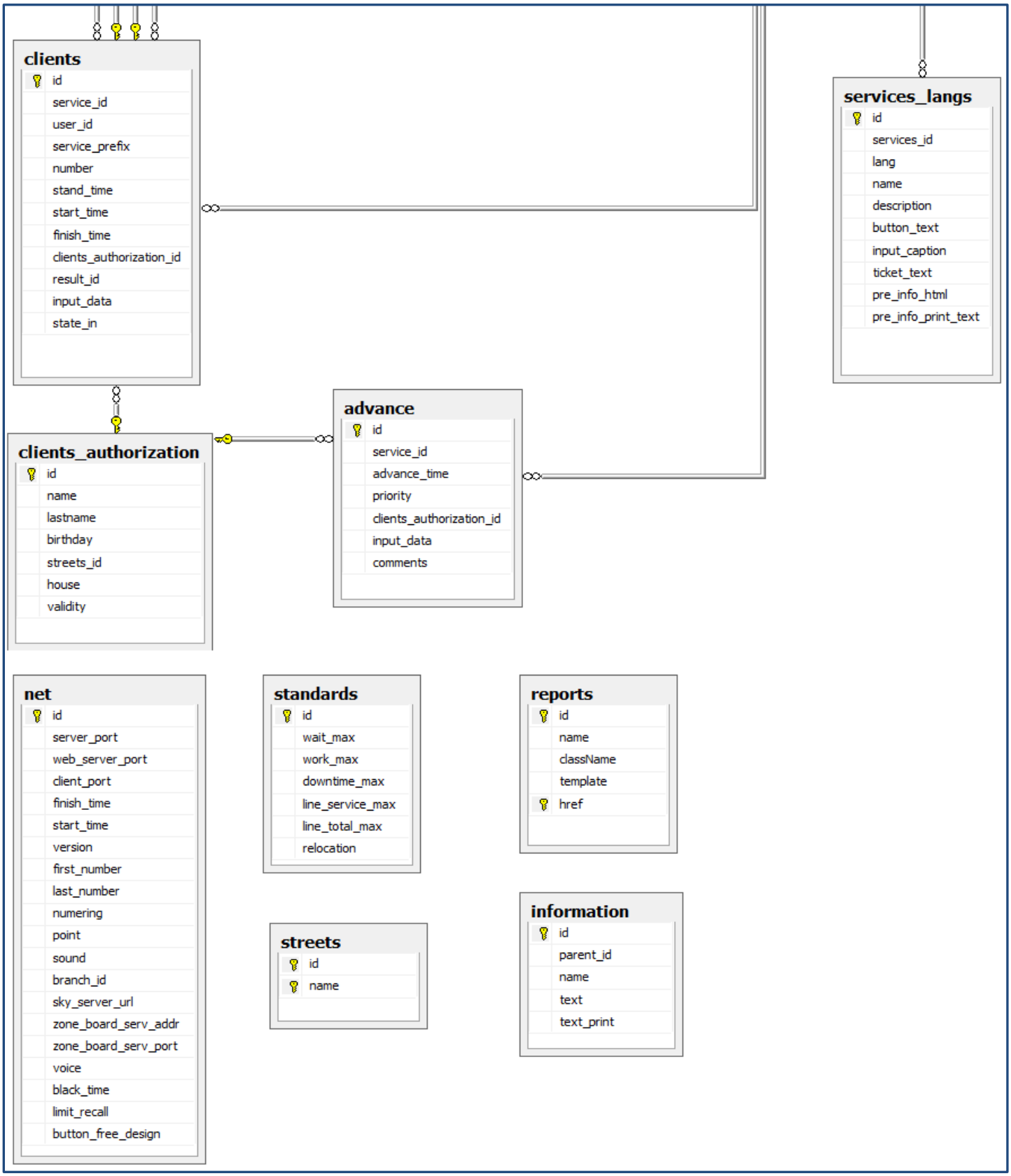

Figura 2.12 Diagrama Entidad-Relación: Parte 2

Fuente: Propia

2.2.1.3 Diagrama de Clases

El diagrama de clases permite mostrar las relaciones entre las clases involucradas en el sistema QManagement.

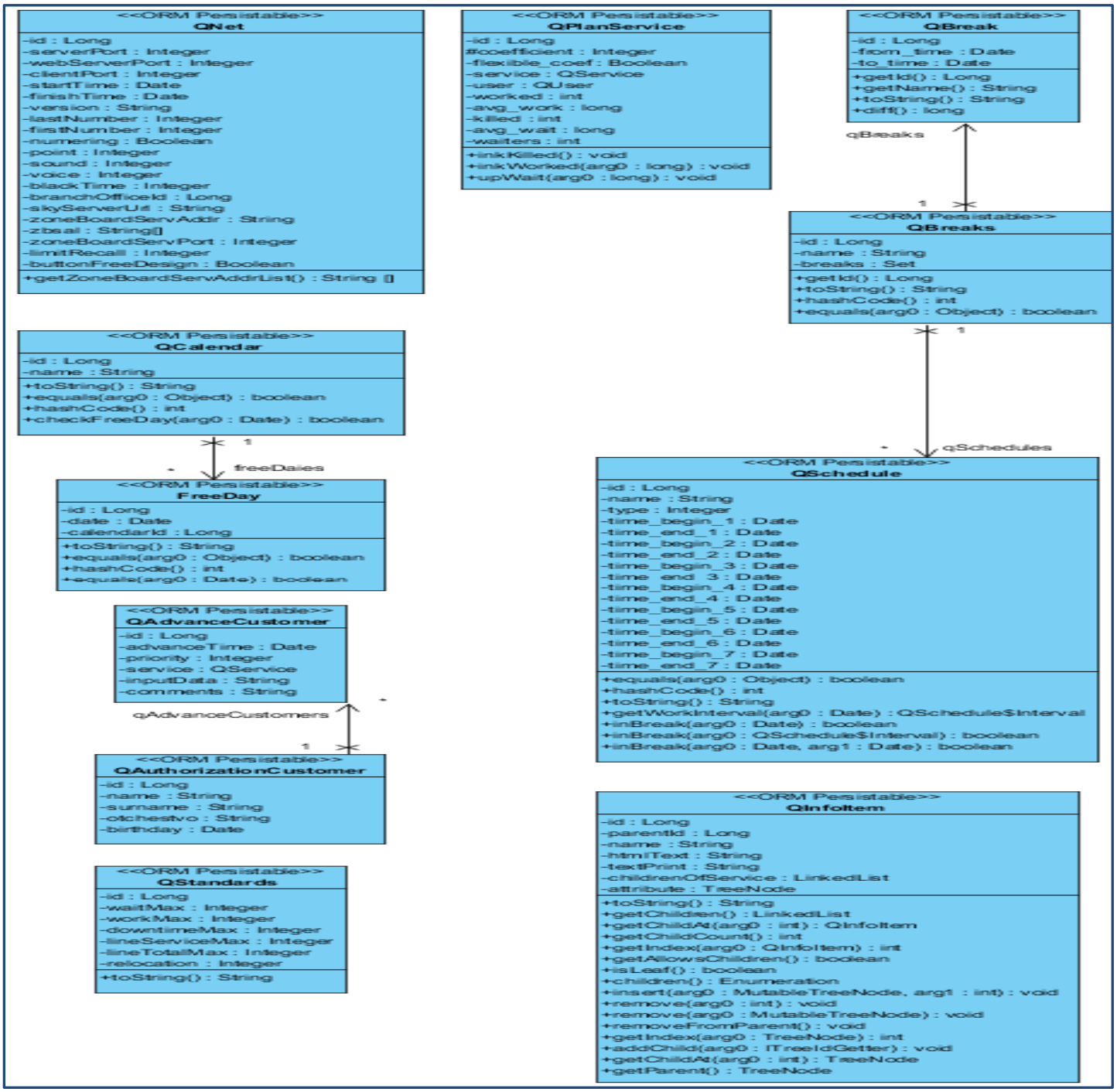

Figura 2.13 Diagrama de Clases: Parte 1

Fuente: Propia

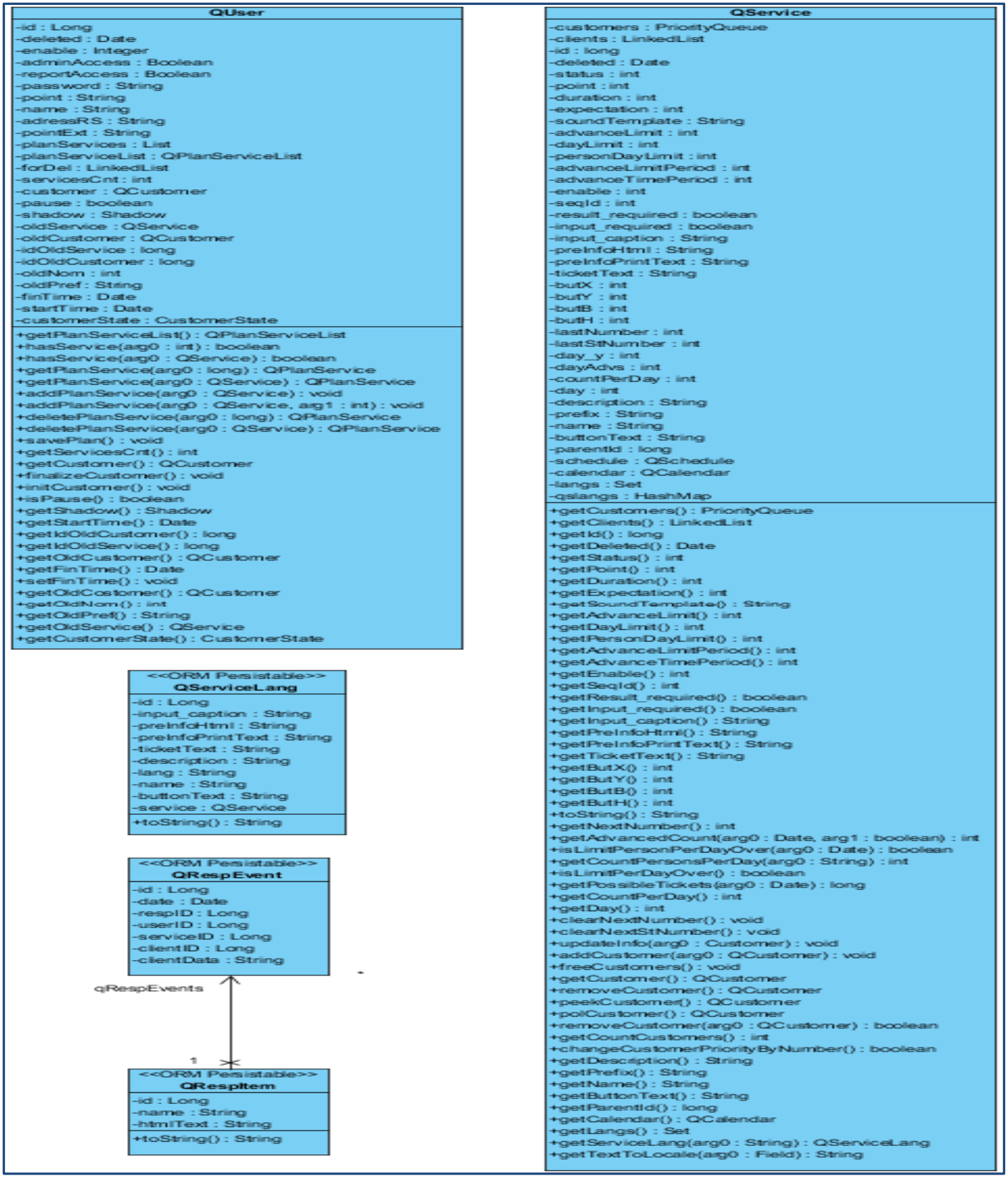

Figura 2.14 Diagrama de Clases: Parte 2

Fuente: Propia

2.2.1.4 Diagrama de Flujo o Actividades

El diagrama de flujo o también conocido como diagrama de actividades permite representar graficamente el flujo de trabajo que se involucra en la gestion de un turno paso a paso.

2.2.1.4.1 Gestión de Turno

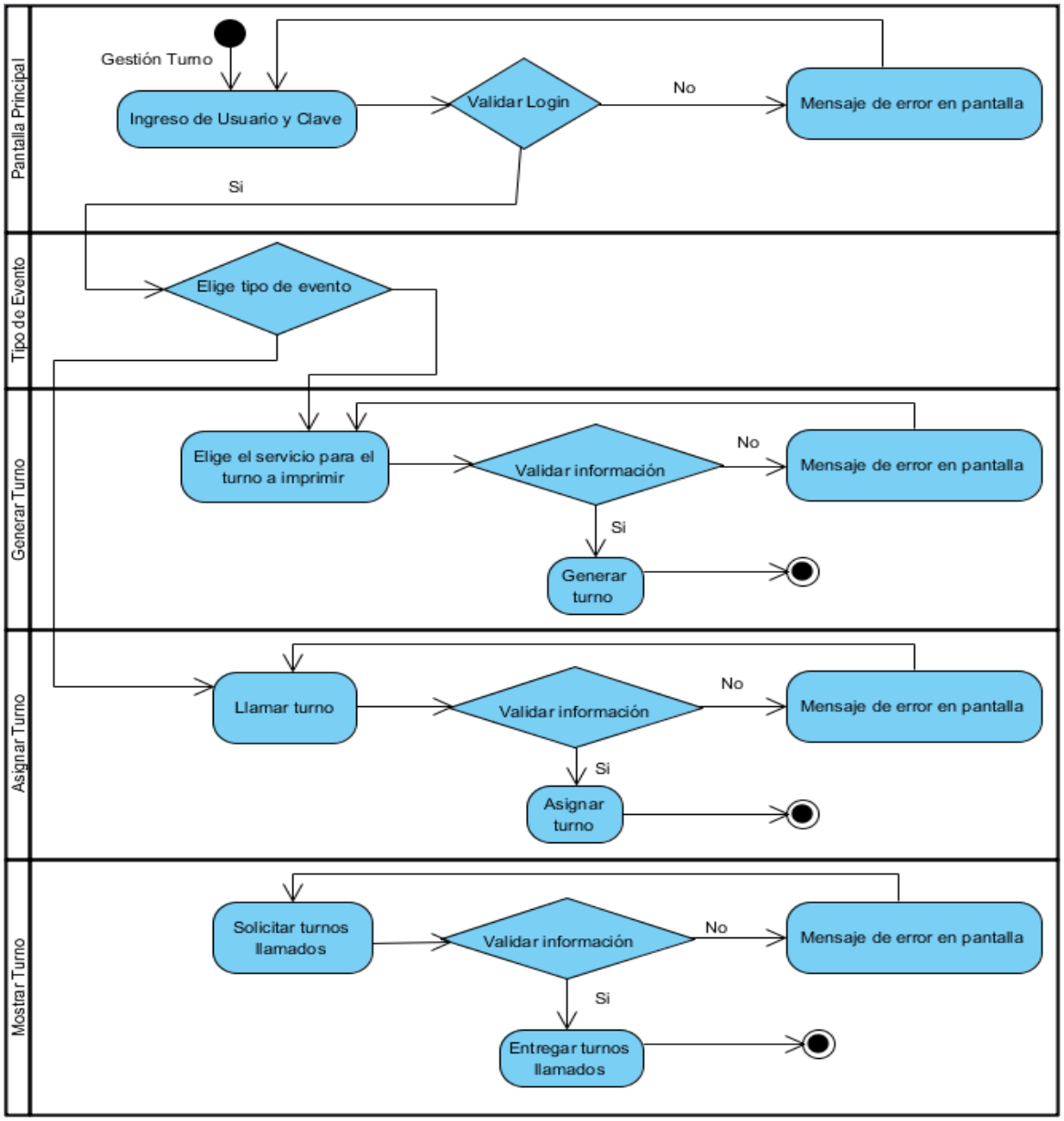

Figura 2.15 Diagrama de Actividad: Gestión Turno

Fuente: Propia

2.2.1.4.2 Gestión de Prioridades

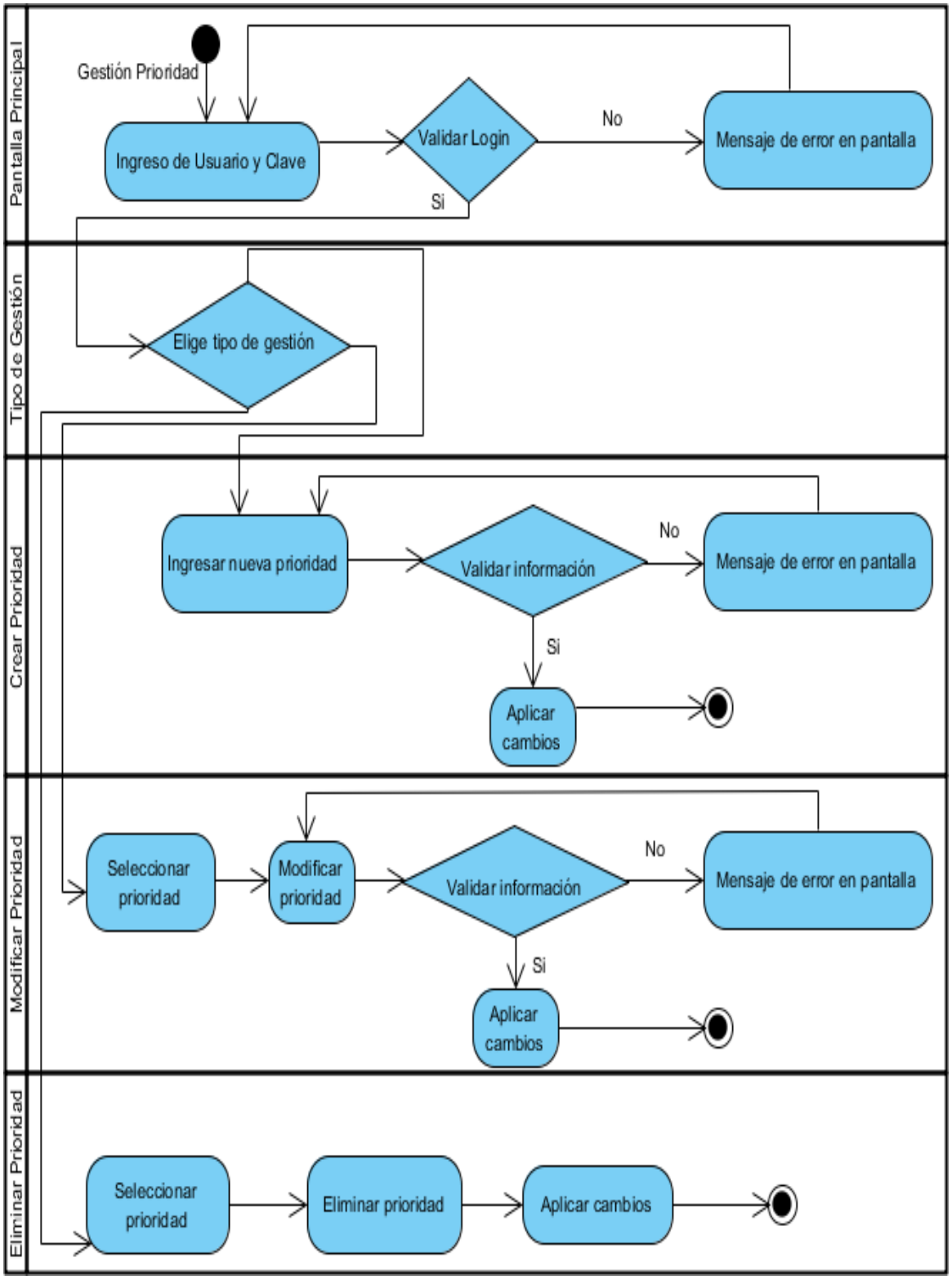

Figura 2.16 Diagrama de Actividad: Gestión de Prioridades

Fuente: Propia

2.2.1.4.3 Gestión de Usuarios

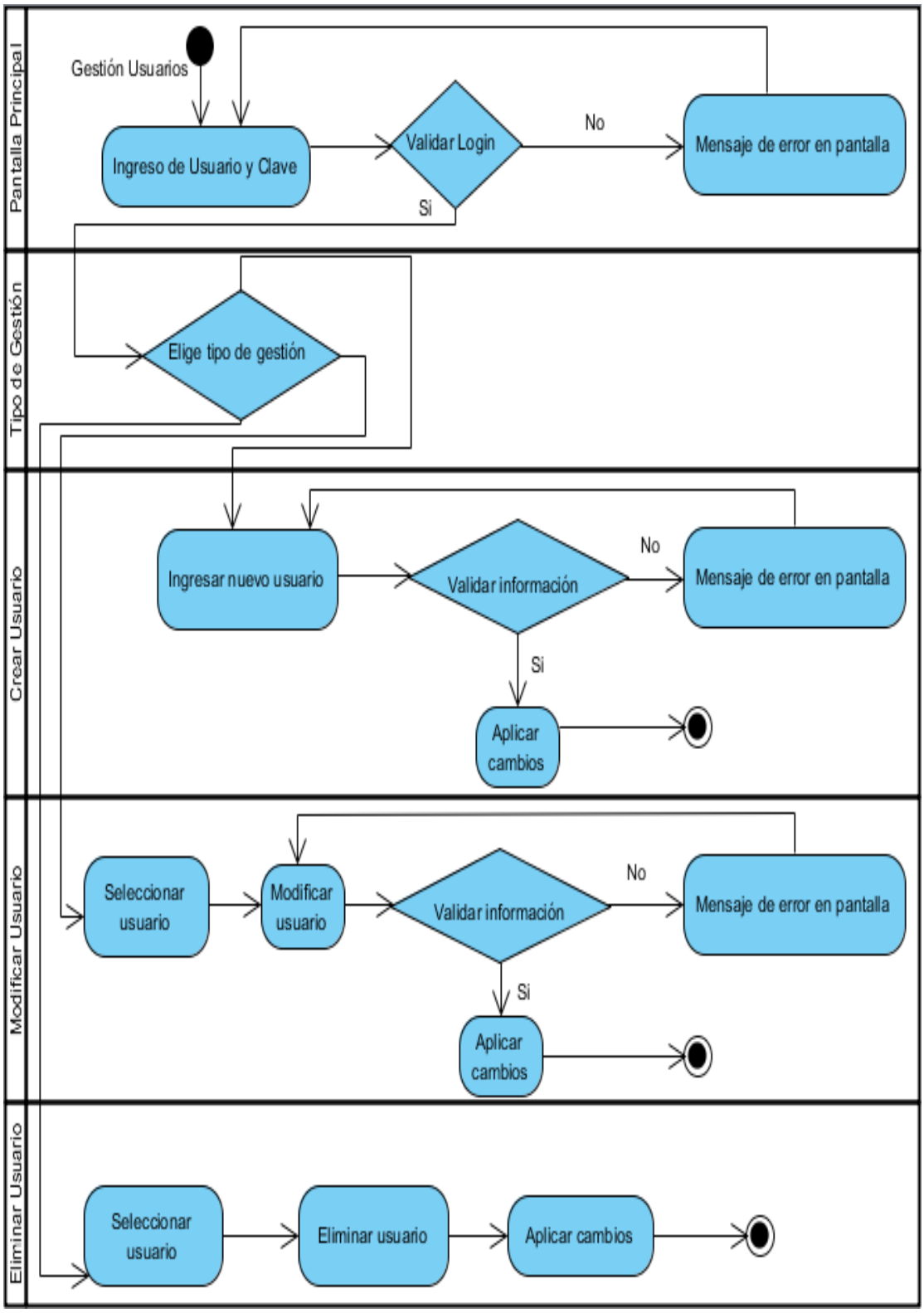

Figura 2.17 Diagrama de Actividad: Gestión de Usuarios

Fuente: Propia

2.2.1.4.4 Gestión de Puestos de Atención

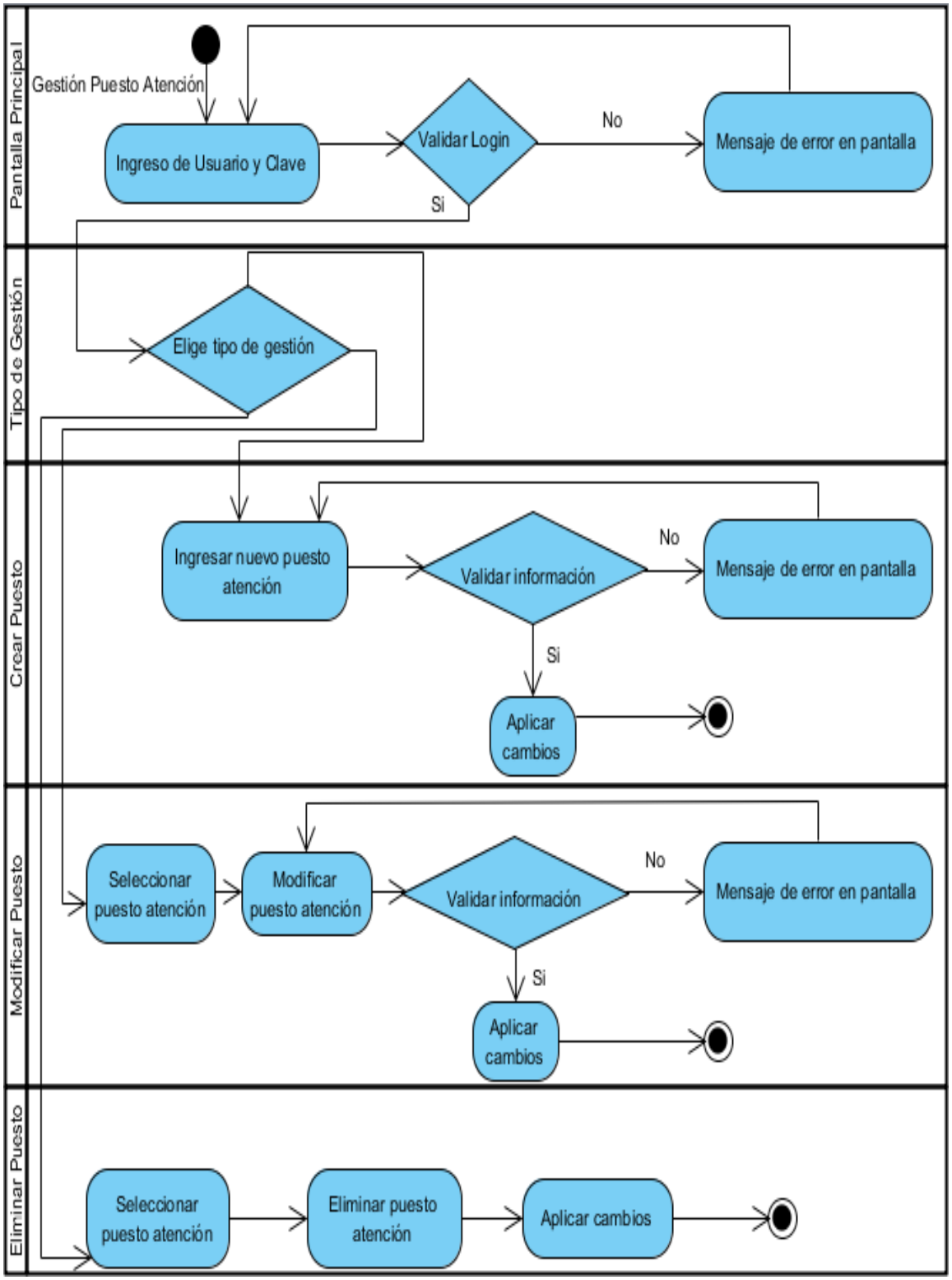

Figura 2.18 Diagrama de Actividad: Gestión de Puestos de Atención

Fuente: Propia

2.2.1.4.5 Gestión de Servicios

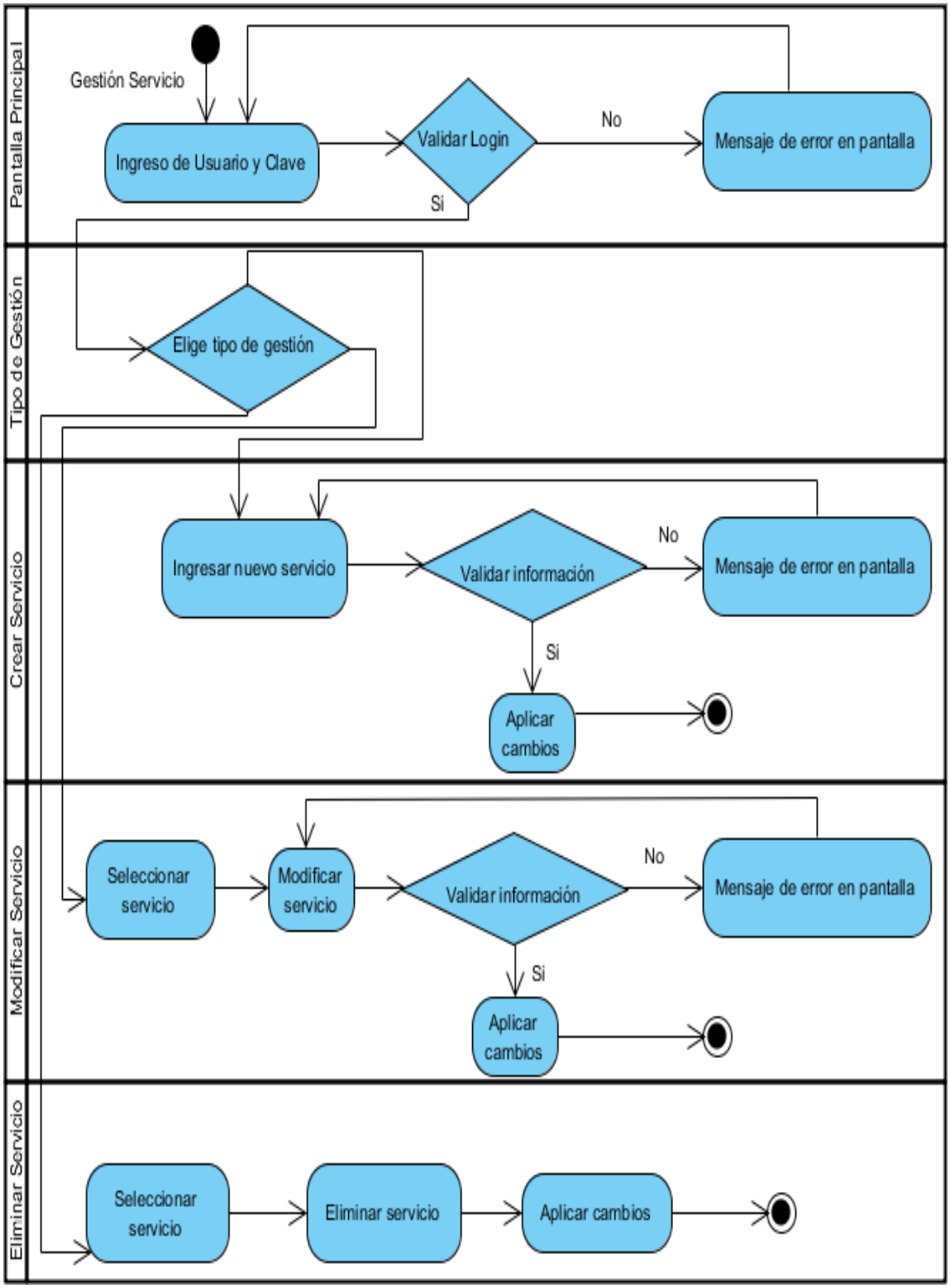

Figura 2.19 Diagrama de Actividad: Gestión de Servicios

Fuente: Propia

2.2.1.4.6 Imprimir Turno

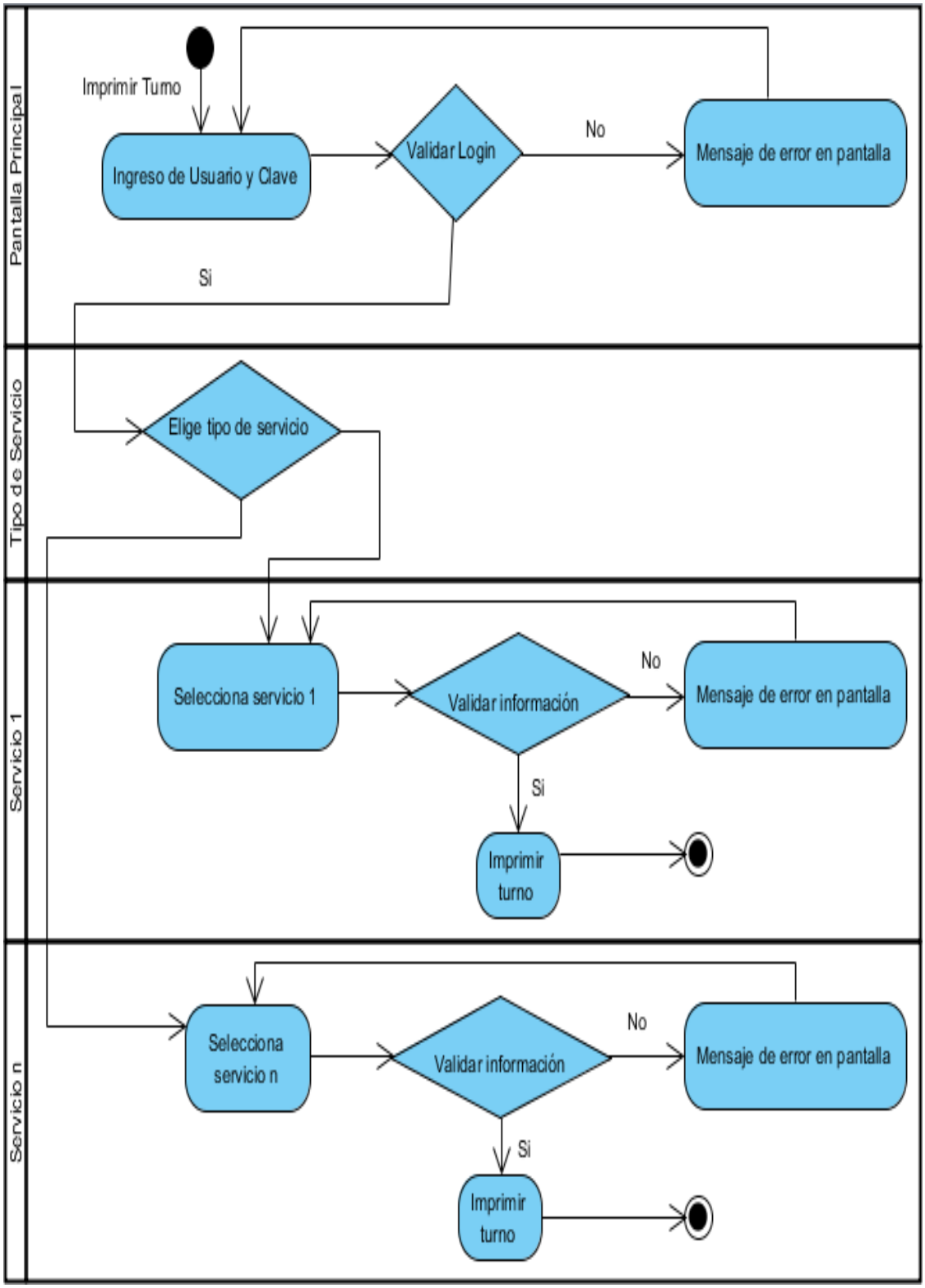

Figura 2.20 Diagrama de Actividad: Imprimir Turno

Fuente: Propia

2.2.1.4.7 Llamar Turno

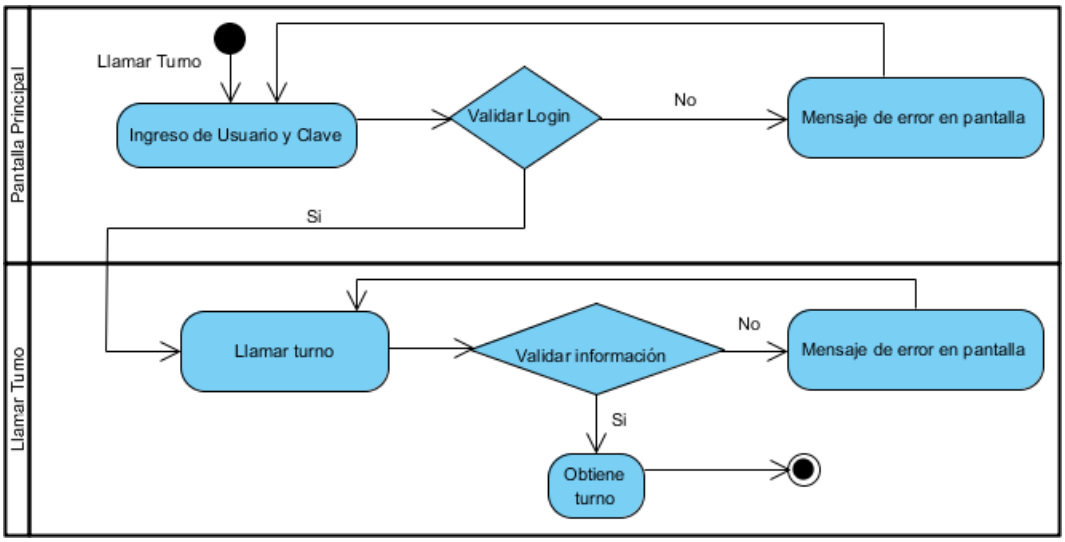

Figura 2.21 Diagrama de Actividad: Llamar Turno

Fuente: Propia

2.2.1.4.8 Mostrar Turno

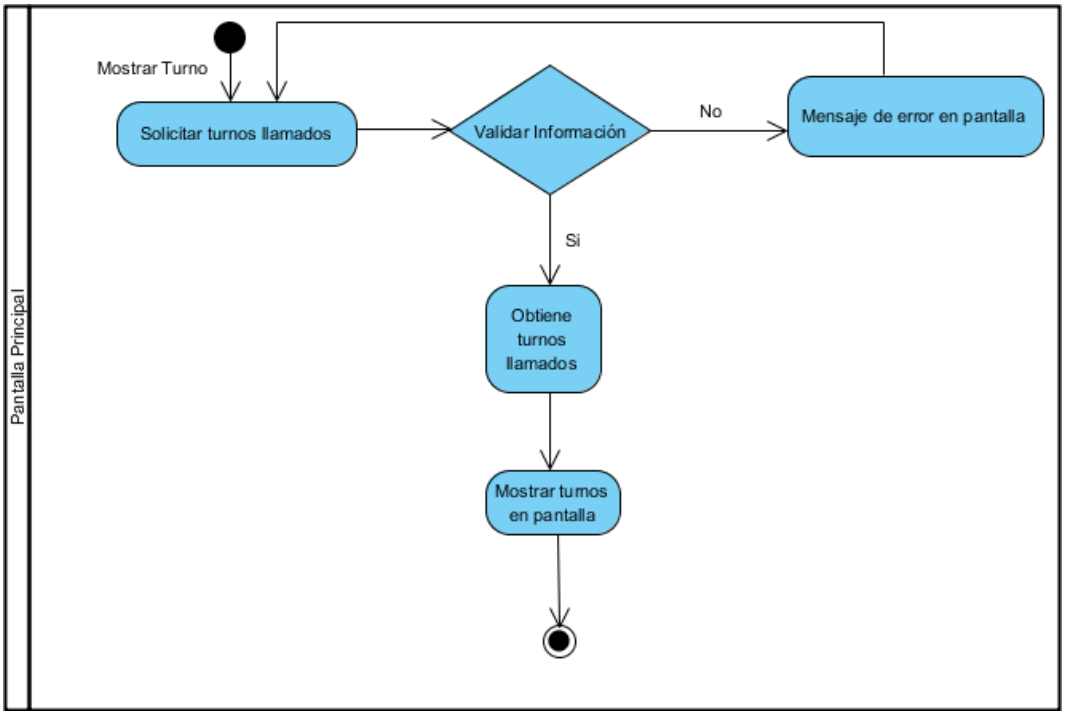

Figura 2.22 Diagrama de Actividad: Mostrar Turno

Fuente: Propia

2.2.1.5 Diccionario de datos

A continuación, se presenta el diccionario de datos para el sistema QManagement.

Nombre de tabla: ADVANCE **Descripción:** permite el ingreso de información de pre registro de clientes.			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del cliente pre registrado	PK
service_id	bigint	Identifica el servicio	FK
advance_time	datetime2(0)	Hola de pre registro	
priority	int	Prioridad del cliente pre registrado	
clients_authorization_id	bigint	Determinar si el cliente está registrado	FK
input_data	nvarchar(150)	Ingresar los datos del cliente	
comments	nvarchar(345)	Ingresar algún comentario por parte del usuario operador	

Tabla 2.12 Diccionario de Datos: Tabla ADVANCE.

Fuente: Propia

Nombre de tabla: BREAK **Descripción:** permite el manejo de información sobre los recesos de los usuarios			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del receso	PK
breaks_id	bigint	Registra el id del receso que contiene el nombre	FK
from_time	time(7)	Hora de inicio del receso	
to_time	time(7)	Hora final del receso	

Tabla 2.13 Diccionario de Datos: Tabla BREAK.

Fuente: Propia

Nombre de tabla: BREAKS **Descripción:** permite almacenar los nombres de los recesos			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del nombre del receso	PK
name	nvarchar(245)	Nombre del receso	

Tabla 2.14 Diccionario de Datos: Tabla BREAKS.

Fuente: Propia

Nombre de tabla: CALENDAR **Descripción:** permite almacenar los nombres del calendario en el cual se trabaja			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del nombre del calendario	PK
name	nvarchar(45)	Nombre del calendario	

Tabla 2.15 Diccionario de Datos: Tabla CALENDAR.

Fuente: Propia

Nombre de tabla: CALENDAR_OUT_DAYS **Descripción:** permite almacenar las fechas del calendario en las cuales no se trabaja			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del calendario	PK
out_day	date	Fecha en la que no se trabaja	
calendar_id	bigint	Identificador del nombre del calendario	FK

Tabla 2.16 Diccionario de Datos: Tabla CALENDAR_OUT_DAYS.

Fuente: Propia

Nombre de tabla: CLIENTS

Descripción: permite almacenar toda la información relacionada a la atención de los clientes, es una de las tablas más importantes del sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del registro de atención del cliente	PK
service_id	bigint	Identifica el id del servicio para el que se atendió al cliente	FK
user_id	bigint	Identifica el id del usuario que atendió al cliente	FK
service_prefix	nvarchar(45)	Prefijo o letra del turno atendido	
number	int	Número del turno atendido	
stand_time	datetime2(0)	Tiempo de emisión del turno	
start_time	datetime2(0)	Tiempo de inicio de atención del turno	
finish_time	datetime2(0)	Tiempo de finalización de atención del turno	
clients_authorization_id	bigint	Identifica el id del cliente atendido	FK
result_id	bigint	Identifica la salida generada	FK
input_data	nvarchar(150)	Muestra los datos ingresados por el usuario	
state_in	int	Estado del cliente	

Tabla 2.17 Diccionario de Datos: Tabla CLIENTS.

Fuente: Propia

Nombre de tabla: NET

Descripción: permite almacenar toda la información relacionada con la comunicación entre el sistema Servidor del Sistema y los módulos clientes

Columnas	Tipo de Datos	Descripción	Referencia
id	int	Identificador del registro de información de red	PK
server_port	int	Puerto del servidor	
web_server_port	int	Puerto del servidor web	
client_port	int	Puerto de comunicación del cliente	
finish_time	time	Hora en la que se detiene el servidor	
start_time	time	Hora en la que inicia el funcionamiento del servidor	
version	nvarchar(25)	Versión de la BDD	
first_number	int	Primer número de la secuencia del turno	
last_number	int	Ultimo número de la secuencia del turno	
numering	smallint	Se asigna 0 si todos los servicios van a tener la misma numeración o 1 si cada servicio tiene su propia numeración	
point	int	Se define la descripción del puesto de atención como se va a mostrar	
sound	int	Se define el tipo de configuración para la voz, 0 para no reproducir ningún audio, 1 para mostrar solo las señales y 2 para mostrar señales más la voz	
branch_id	bigint	Identifica el id de la agencia	
sky_server_url	nvarchar(145)	Url del servidor	

zone_board_serv_addr	nvarchar(145)	Dirección del panel del panel del servidor donde se reciben los datos	
zone_board_serv_port	bigint	Puerto del panel del servidor donde se reciben los datos	
voice	int	Reproducir la Voz al mostrar turno	
black_time	int	Tiempo transcurrido de los turnos en la lista negra	
limit_recall	int	Límite de rellamadas	
button_free_design	smallint	Libre diseño de los botones en el punto de entrada	

Tabla 2.18 Diccionario de Datos: Tabla NET.

Fuente: Propia

Nombre de tabla: REPORTS

Descripción: permite almacenar la información relacionada a la generación de reportes

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del reporte	PK
name	nvarchar(255)	Nombre del reporte	
className	nvarchar(150)	Clase que permite generar el reporte	
template	nvarchar(150)	Plantilla del reporte	
href	nvarchar(150)	Url del reporte	FK

Tabla 2.19 Diccionario de Datos: Tabla REPORTS.

Fuente: Propia

Nombre de tabla: RESPONSE

Descripción: permite almacenar la información relacionada a las respuestas de satisfacción de los clientes

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador de la respuesta	PK
name	nvarchar(100)	Descripción de la respuesta	
text	nvarchar(500)	Texto de la respuesta	

Tabla 2.20 Diccionario de Datos: Tabla RESPONSE.

Fuente: Propia

Nombre de tabla: RESPONSE_EVENT

Descripción: permite almacenar la información relacionada a las respuestas emitidas por los clientes

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador de la opinión	PK
resp_date	datetime2(0)	Fecha de la opinión	
response_id	bigint	Identificador del id de la opinión	FK
services_id	bigint	Identificador del servicio que recibió la opinión	FK
users_id	bigint	Identificador del usuario que recibió la opinión	FK
clients_id	bigint	Identificador del id del cliente que emitió la opinión	FK
client_data	nvarchar(245)	Comentario emitido por el cliente	

Tabla 2.21 Diccionario de Datos: Tabla RESPONSE_EVENT.

Fuente: Propia

Nombre de tabla: RESULTS **Descripción:** permite almacenar el estado de procesamiento de las tareas del sistema			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del resultado	PK
name	nvarchar(150)	Descripción del estado	

Tabla 2.22 Diccionario de Datos: Tabla RESULTS.

Fuente: Propia

Nombre de tabla: SCHEDULE **Descripción:** permite almacenar el horario de trabajo en el que se encuentra habilitado el sistema			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del resultado	PK
name	nvarchar(150)	Descripción del estado	
type	int	Tipo de planificación, puede ser semanal o por días	
time_begin_1	time(7)	Hora de inicio	
time_end_1	time(7)	Hora de finalización	
time_begin_2	time(7)	Hora de inicio	
time_end_2	time(7)	Hora de finalización	
time_begin_3	time(7)	Hora de inicio	
time_end_3	time(7)	Hora de finalización	
time_begin_4	time(7)	Hora de inicio	
time_end_4	time(7)	Hora de finalización	
time_begin_5	time(7)	Hora de inicio	
time_end_5	time(7)	Hora de finalización	
time_begin_6	time(7)	Hora de inicio	
time_end_6	time(7)	Hora de finalización	
time_begin_7	time(7)	Hora de inicio	
time_end_7	time(7)	Hora de finalización	
breaks_id1	bigint	Identifica la hora de receso	FK
breaks_id2	bigint	Identifica la hora de receso	FK

breaks_id3	bigint	Identifica la hora de receso	FK
breaks_id4	bigint	Identifica la hora de receso	FK
breaks_id5	bigint	Identifica la hora de receso	FK
breaks_id6	bigint	Identifica la hora de receso	FK
breaks_id7	bigint	Identifica la hora de receso	FK

Tabla 2.23 Diccionario de Datos: Tabla SCHEDULE.

Fuente: Propia

Nombre de tabla: SERVICE **Descripción:** permite almacenar la información de los servicios que presta la agencia			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador del servicio	PK
name	nvarchar(2000)	Nombre del servicio	
description	nvarchar(2000)	Descripción del servicio	
service_prefix	nvarchar(10)	Prefijo o letra del servicio	
button_text	nvarchar(2500)	El texto que se imprimirá al elegir el servicio	
status	int	Estado del servicio, 1 disponible, 0 no disponible	
enable	int	Si está habilitado o no	
prent_id	bigint	Grupo al que pertenece	
day_limit	int	Restricción de turnos emitidos por día. Si no hay restricción su valor es 0	
person_day_limit	int	Restricción de turnos emitidos por día para un cliente. Si no hay restricción su valor es 0	
advance_limit	int	Limitar el número de clientes pre registrados por hora	
advance_limit_period	int	Limite en días para el pre registro por anticipado. Si no	

		hay restricción su valor es 0	
advance_time_period	int	Periodos en los que se ha divide un día para el pre registro	
schedule_id	bigint	Identifica el horario de trabajo del servicio	FK
input_required	smallint	Ingreso obligado de información del cliente antes de entrar a la cola	
input_caption	nvarchar(2000)	Ingresar el número de documento al momento de ingresar la información del cliente	
result_required	smallint	El usuario debe ingresar el estado del proceso al atender al cliente	
calendar_id	bigint	Identifica el id del calendario	FK
pre_info_html	nvarchar(MAX)	Texto de información que se muestra en la pantalla	
pre_info_print_text	nvarchar(MAX)	Texto que se imprimirá con el servicio antes del encolamiento	
point	int	Punto de registro del cliente	
ticket_text	nvarchar(1500)	Texto que se imprimirá en el turno	
seq_id	int	Orden de los botones en el punto de registro	
but_x	int	Posición x del botón	
but_y	int	Posición y del botón	
but_b	int	Ancho del botón	
but_h	int	Alto del botón	
deleted	date	Confirmar la fecha de eliminación del servicio	

duration	int	Tiempo promedio para atender el servicio	
sound_template	nvarchar(45)	Plantilla para emitir un sonido al llamar el servicio	
expectation	int	Tiempo de espera obligatorio	

Tabla 2.24 Diccionario de Datos: Tabla SERVICES.

Fuente: Propia

Nombre de tabla: SERVICES_LANGS

Descripción: permite almacenar la información de los servicios en diferentes idiomas

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del idioma del servicio	PK
services_id	bigint	Identifica el servicio	FK
lang	nvarchar(45)	Lenguaje	
name	nvarchar(2000)	Nombre del lenguaje	
description	nvarchar(2000)	Descripción del lenguaje	
button_text	nvarchar(2500)	Identificador del texto del botón	
input_caption	nvarchar(2000)	Texto de entrada del servicio	
ticket_text	nvarchar(1500)	Texto a imprimir en el turno	
pre_info_html	nvarchar(MAX)	Información a mostrarse en la pantalla	
pre_info_print_text	nvarchar(MAX)	Información a imprimirse antes del encolamiento	

Tabla 2.25 Diccionario de Datos: Tabla SERVICES_LANGS.

Fuente: Propia

Nombre de tabla: SERVICES_USERS

Descripción: permite almacenar la información de los servicios asignados a los usuarios del sistema

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del servicio de usuario	PK
services_id	bigint	Identifica el id del servicio	FK
user_id	bigint	Identifica el id del usuario	FK
coefficient	int	Porcentaje de atención	
flexible_coef	smallint	Es posible cambiar el coeficiente de atención	

Tabla 2.26 Diccionario de Datos: Tabla SERVICES_USERS.

Fuente: Propia

Nombre de tabla: STANDARS

Descripción: permite almacenar la información de las reglas de negocio establecidas para la atención de los clientes por parte de los usuarios del sistema

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador de la regla de negocio	PK
wait_max	int	Tiempo de espera máximo	
work_max	int	Tiempo de atención máximo	
downtime_max	int	Tiempo máximo de inactividad en la cola	
line_service_max	int	Longitud máxima de la cola para un servicio	
line_total_max	int	El número máximo de clientes para todos los servicios	
relocation	int	Tiempo de traslado de un servicio	

Tabla 2.27 Diccionario de Datos: Tabla STANDARS.

Fuente: Propia

Nombre de tabla: STATISTIC **Descripción:** permite almacenar la información sobre las estadísticas del sistema			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador de la estadística	PK
user_id	bigint	Identifica el id del usuario	FK
client_id	bigint	Identifica el id del cliente	FK
service_id	bigint	Identifica el id del servicio	FK
results_id	bigint	Identifica el estado de la tarea	FK
user_start_time	datetime2(0)	Tiempo de inicio de la atención del usuario con los clientes	
user_finish_time	datetime2(0)	Tiempo de finalización de la atención del usuario con los clientes	
client_stand_time	datetime2(0)	Tiempo en reposo del cliente	
user_work_period	int	Tiempo de trabajo del usuario con el cliente	
client_wait_period	int	Tiempo de espera del cliente	
state_in	int	Estado del cliente	

Tabla 2.28 Diccionario de Datos: Tabla STATISTIC.

Fuente: Propia

Nombre de tabla: STREETS **Descripción:** permite almacenar las direcciones de los usuarios del sistema			
Columnas	**Tipo de Datos**	**Descripción**	**Referencia**
id	bigint	Identificador de la dirección	PK
name	nvarchar(100)	Descripción	PK

Tabla 2.29 Diccionario de Datos: Tabla STREETS.

Fuente: Propia

Nombre de tabla: USERS

Descripción: permite almacenar toda la información de los usuarios del sistema

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del usuario	PK
name	nvarchar(150)	Nombre del usuario	
password	nvarchar(45)	Clave del usuario	
point	nvarchar(45)	Lugar de trabajo	
adress_rs	smallint	Dirección	
enable	int	Estado del usuario	
admin_access	smallint	Si tiene permiso para ingresar al perfil de administrador del sistema	
report_access	smallint	Si tiene permiso para ingresar al perfil de reportería del sistema	
point_ext	nvarchar(1045)	Mostrar puesto en el panel principal de llamadas	
deleted	date	Fecha de confirmación de eliminación	

Tabla 2.30 Diccionario de Datos: Tabla USERS.

Fuente: Propia

2.3 IMPLEMENTACIÓN

El sistema QManagement se desarrolló en Java versión 1.8 y se utilizó los siguientes componentes que permiten cumplir con el patrón de desarrollo Modelo-Vista-Controlador (MVC).

- Framework de desarrollo Spring Web MVC 4.0.1 el cual permite la integración de las capas de persistencia, gestión y servicios, además del manejo de transaccionalidad con la base de datos SQL Server.

- Hibernate 4.3 que permite el mapeo objeto-relacional de los objetos de negocio.

- JavaServer Faces 2.1 para la capa de presentación.

Para la distribución de paquetes del sistema se utiliza la nomenclatura **ec.edu.utn.fica.eisic.qmanagement** y dependiendo de la funcionalidad de cada paquete se crean las clases correspondientes.

Al momento de crear la aplicación web en Netbeans se crean los siguientes archivos XML necesarios para su funcionamiento:

- Archivo **applicationContext.xml** de Spring en el cual se debe configurar todos los componentes necesarios para la integración de las capas del modelo MVC.

Como se observa en el siguiente fragmento del archivo **applicationContext** QNetDAO se conecta con SessionFactory mediante Spring.

```
<bean id="QNetDAO"
class="ec.edu.utn.fica.eisic.qmanagement.servidor.dao.QNetDAO">
<property name="sessionFactory" ref="SessionFactory" />
</bean>
```

Para configurar la fábrica de sesiones de hibernate se conecta el bean SessionFactory con la propiedad c3p0DataSource y se anotan las clases para hibernate.

```
<bean id="SessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean"
lazy-init="false" autowire="default">
 <property name="dataSource" ref="c3p0DataSource"/>
 <property name="hibernateProperties">
    <props merge="default">
<prop
key="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</prop>
<prop key="hibernate.show_sql">true</prop>
<prop key="hibernate.format_sql">true</prop>
</props>
</property>
<property name="annotatedClasses">
<list>
<value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet</value>
</list>
</property>
</bean>
```

Se crea el bean c3p0DataSource para asignar los parámetros para el acceso a la fuente de datos.

```
<bean id="c3p0DataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-
method="close">
       <property name="driverClass" value="#{conf.driver}"/>
       <property name="jdbcUrl" value="#{conf.url}"/>
       <property name="user" value="#{conf.user}"/>
       <property name="password" value="#{conf.password}"/>
       <property name="checkoutTimeout" value="10000"/>
       <property name="idleConnectionTestPeriod">
          <value>3600</value>
       </property>
</bean>
```

Se crea el bean conf que permite obtener los valores de los parámetros de conexión a la base de datos desde el archivo configbdd.dat

```
<bean id="conf"
class="ec.edu.utn.fica.eisic.qmanagement.hibernate.AnnotationSessionFact
oryBean"/>
```

Se crea el bean transactionManager para administrar todas las transacciones que realiza hibernate con la base de datos.

```
<bean id="transactionManager"
class="org.springframework.orm.hibernate4.HibernateTransactionManager"
>
<property name="sessionFactory" ref="SessionFactory" />
</bean>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xs
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

    <!-- Declaración de Beans -->
    <bean id="conf" class="ec.edu.utn.fica.eisic.qmanagement.hibernate.AnnotationSessionFactoryBean"/>
    <bean id="QNet" class="ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet"/>

        <!-- Declaración de los beans de servicio -->
        <bean id="QNetService" class="ec.edu.utn.fica.eisic.qmanagement.servidor.servicio.QNetService">
                <property name="qnetDAO" ref="QNetDAO" />
        </bean>
        <!-- Declaración de los beans DAO -->
        <bean id="QNetDAO" class="ec.edu.utn.fica.eisic.qmanagement.servidor.dao.QNetDAO">
                <property name="sessionFactory" ref="SessionFactory" />
        </bean>
    <!-- Declaración de los beans para el acceso a los datos -->
    <bean id="c3p0DataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
        <property name="driverClass" value="#{conf.driver}"/>
        <property name="jdbcUrl" value="#{conf.url}"/>
        <property name="user" value="#{conf.user}"/>
        <property name="password" value="#{conf.password}"/>
        <property name="checkoutTimeout" value="10000"/>
        <property name="idleConnectionTestPeriod">
            <value>3600</value>
        </property>
    </bean>
    <!-- Fabrica de produccion de sesiones con la BDD. Listado de clases anotadas para hibernate. -->
    <bean id="SessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" lazy-init="false" autowire="default">
        <property name="dataSource" ref="c3p0DataSource"/>
        <property name="hibernateProperties">
            <props merge="default">
                <prop key="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
            </props>
        </property>

        <property name="annotatedClasses">
            <list>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QStandards</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QService</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QServiceLang</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QUser</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.comun.modelo.QCustomer</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QPlanService</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QAdvanceCustomer</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QAuthorizationCustomer</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.infosistema.QInfoItem</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.respuesta.QRespItem</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.respuesta.QRespEvent</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.horario.QSchedule</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.horario.QBreaks</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.horario.QBreak</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.resultado.QResult</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.calendario.QCalendar</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.calendario.FreeDay</value>
            </list>
        </property>
    </bean>
    <!-- Habilitar la configuración del funcionamiento transaccional basado en anotaciones-->
    <tx:annotation-driven transaction-manager="transactionManager
    <bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="SessionFactory" />
    </bean>
</beans>
```

Figura 2.23 Contenido archivo **applicationContext.xml**

Fuente: Propia

- Archivo web.xml que describe diversas características del archivo WAR.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
    <param-name>org.richfaces.skin</param-name>
    <param-value>classic</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <!--<param-value>classpath*:applicationContext.xml</param-value>-->
        <param-value>/WEB-INF/classes/applicationContext.xml</param-value>
    </context-param>
    <context-param>
    <param-name>javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE</param-name>
    <param-value>true</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <listener>
        <listener-class>ec.edu.utn.fica.eisic.qmanagement.servidor.ContextListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            300
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index_admin.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Figura 2.24 Contenido archivo **web.xml**

Fuente: Propia

A continuación se describen los elementos del archivo web.xml

- El elemento <servlet> define las características de un Servlet y a su vez está compuesto por los elementos <servlet-name> y <servlet-class> que indican un nombre corto para el Servlet así como el nombre de la Clase Java que contiene el Servlet respectivamente. En este caso se indica que la Clase llamada javax.faces.webapp.FacesServlet será

denominada con el nombre Faces Servlet que habilita el motor de JSF en el sistema.

```
<servlet>
	<servlet-name>Faces Servlet</servlet-name>
	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
	<load-on-startup>1</load-on-startup>
</servlet>
<servlet>
	<servlet-name>dispatcher</servlet-name>
	<servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
	<load-on-startup>2</load-on-startup>
</servlet>
```

- El elemento <servlet-mapping> define la ubicación en términos de directorios de un sitio (URL), esto es, el elemento <servlet-name>Faces Servlet</servlet-name> está indicando que el Servlet llamado Faces Servlet será accesado cada vez que se accede al directorio base indicado dentro del elemento <url-pattern>.

```
<servlet-mapping>
	<servlet-name>Faces Servlet</servlet-name>
	<url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

- El elemento <welcome-file-list> indica que cuando se solicite cualquier directorio sin indicar un archivo en específico se envíe el archivo llamado index_admin.xhtml.

```
<welcome-file-list>
	<welcome-file>index_admin.xhtml</welcome-file>
</welcome-file-list>
```

- El elemento <session-config> indica la duración en minutos de la sesión del usuario cuando navegue en la aplicación.

```
<session-config>
      <session-timeout>
         300
      </session-timeout>
</session-config>
```

- El elemento <listener> define las características de un evento ServletContextEvent que ocurre dentro de la clase ContextListener.

Definimos el listener ContextListener dentro del archivo web.xml para que sea la primera clase que se ejecute al momento de levantar la aplicación.

```
<listener>
<listener-
class>ec.edu.utn.fica.eisic.qmanagement.servidor.ContextListener
</listener-class>
</listener>
```

- El elemento <context-param> permite activar la biblioteca de componentes Richfaces que está dentro del paquete org.richfaces.skin donde el valor asignado es el classic que corresponde al tipo de presentación de la interfaz de usuario,

```
<context-param>
   <param-name>org.richfaces.skin</param-name>
   <param-value>classic</param-value>
</context-param>
```

Para indicar la ruta del archivo que carga el contexto de la aplicación se debe ingresar en el elemento <param-value>.

```
<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/classes/applicationContext.xml</param-
value>
</context-param>
```

- Archivo **faces-config.xml** que es el archivo de configuración de JSF.

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.1"
              xmlns="http://java.sun.com/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
              http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd">
        <!-- Integración JSF y Spring -->
        <application>
        <el-resolver>
                org.springframework.web.jsf.el.SpringBeanFacesELResolver
        </el-resolver>
        </application>

        <!-- Configuración de las reglas de navegación -->
        <navigation-rule>
            <from-view-id>/paginas/index.xhtml</from-view-id>
            <navigation-case>
                <from-outcome>success</from-outcome>
            <to-view-id>/paginas/success.xhtml</to-view-id>
        </navigation-case>
         <navigation-case>
                <from-outcome>error</from-outcome>
            <to-view-id>/paginas/error.xhtml</to-view-id>
        </navigation-case>
        </navigation-rule>
</faces-config>
```

Figura 2.25 Contenido archivo **faces-config.xml**

Fuente: Propia

A continuación se describen los elementos del archivo faces-config.xml

- El elemento <navigation-rule> permite definir las reglas de navegación de la aplicación.

- El elemento <el-resolver>

org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver> permite activar el soporte de Spring para la inyección de dependencias de tal forma que JSF sabrá que si no encuentra un bean bajo su contexto debe ir a buscarlo al contexto de Spring.

```
<application>
      <el-resolver>
            org.springframework.web.jsf.el.SpringBeanFacesELResolver
      </el-resolver>
</application>
```

2.3.1 PAQUETES DEL SISTEMA QMANAGEMENT

- Paquete predeterminado que contiene el archivo log4j.xml que permite habilitar la funcionalidad de control de mensajes de error, de información, de advertencia y de depuración de la aplicación.

- Paquete **ec.edu.utn.fica.eisic.qmanagement** que contiene solamente la clase About que permite obtener la información del archivo de propiedades version.properties tal como la versión del sistema, versión de la base de datos, fecha de publicación, etc.

```
package ec.edu.utn.fica.eisic.qmanagement;

import java.util.Properties;
import ec.edu.utn.fica.eisic.qmanagement.servidor.ContextListener;
import java.io.File;
import java.io.FileInputStream;

public class About {

    public static String ver = "";
    public static String date = "";
    public static String db = "";
    public static String name = "";


    public static void load() {

        try {
                        File archivo = new File(ContextListener.pathversion);
                        if (!archivo.exists()) {
                                System.out.println("Archivo de Propiedades no existe");

                        }

                        Properties propiedades = new Properties();

                        FileInputStream f = new FileInputStream(archivo);
                        propiedades.load(f);
                        f.close();

                        ver = propiedades.getProperty("version");
                        date = propiedades.getProperty("date");
                        db = propiedades.getProperty("version_db");
                        name=propiedades.getProperty("name");


                } catch (Exception e) {
                        e.printStackTrace(System.err);

                }
    }
}
```

Figura 2.26 Contenido de la clase About.java

Fuente: Propia

- Paquete **ec.edu.utn.fica.eisic.qmanagement.cliente** que contiene la clase Locales la cual permite configurar el idioma en el que se desea mostrar la aplicación en caso de que se desee habilitar esa funcionalidad.

```
package ec.edu.utn.fica.eisic.qmanagement.cliente;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Locale;
import org.apache.commons.configuration.ConfigurationException;
import org.apache.commons.configuration.PropertiesConfiguration;
import ec.edu.utn.fica.eisic.qmanagement.comun.QLog;
import ec.edu.utn.fica.eisic.qmanagement.servidor.ContextListener;

public final class Locales {
    private final PropertiesConfiguration config;
    private Locales() {
        String configFileName = ContextListener.pathlangs ;
        config = new PropertiesConfiguration();
        config.setEncoding("utf8");
        File f = new File(configFileName);
        if (f.exists()) {
            config.setFileName(configFileName);
        } else {
            f = new File(configFileName);
            if (f.exists()) {
                config.setFileName(configFileName);
                config.setEncoding("utf8");
            } else {
                final Exception ex = new FileNotFoundException(configFileName);
                QLog.l().logger().error(ex);
                throw new RuntimeException(ex);
            }
        }
        try {
            config.load();
        } catch (ConfigurationException ex) {
            QLog.l().logger().error(ex);
            throw new RuntimeException(ex);
        }
        config.setAutoSave(true);
        for (Iterator<String> itr = config.getKeys(); itr.hasNext();) {
            String s = itr.next();
            if (s.startsWith("locale")) {
                s = s.substring(s.indexOf(".") + 1);
                if (s.contains(".")) {
                    s = s.substring(0, s.indexOf("."));
                    if (locales.get(s) == null) {
                        final Locale locale = new Locale(config.getString("locale." + s + ".lng"), config.getString("locale." + s + ".country"));
                        locales.put(s, locale);
                        locales_name.put(locale, s);
                        lngs.put(config.getString("locale." + s + ".name"), s);
                        lngs_names.put(s, config.getString("locale." + s + ".name"));
                        lngs_buttontext.put(s, config.getString("locale." + s + ".buttontext"));
                    }
                }
            }
        }

    }
    private final LinkedHashMap<String, Locale> locales = new LinkedHashMap<>();
    private final LinkedHashMap<Locale, String> locales_name = new LinkedHashMap<>();
    private final LinkedHashMap<String, String> lngs = new LinkedHashMap<>();
    private final LinkedHashMap<String, String> lngs_names = new LinkedHashMap<>();
    private final LinkedHashMap<String, String> lngs_buttontext = new LinkedHashMap<>();
    public static Locales getInstance() {
        return LocalesHolder.INSTANCE;
    }
    private static class LocalesHolder {
        private static final Locales INSTANCE = new Locales();
    }
    private final String LANG_CURRENT = "locale.current";
    private final String WELCOME_LNG = "welcome.multylangs";
    public boolean isWelcomeMultylangs() {
        return config.getString(WELCOME_LNG) == null ? false : "1".equals(config.getString(WELCOME_LNG)) || config.getString(WELCOME_LNG).startsWith("$");
    }
    public Locale getLangCurrent() {
        return locales.get(config.getString(LANG_CURRENT)) == null ? Locale.getDefault() : locales.get(config.getString(LANG_CURRENT));
    }
    public Locale getLocaleByName(String name) {
        return locales.get(name) == null ? Locale.getDefault() : locales.get(name);
    }
    public String getLangCurrName() {
        return "".equals(config.getString(LANG_CURRENT)) ? lngs_names.get("eng") : lngs_names.get(config.getString(LANG_CURRENT));
    }
    public String getLangButtonText(String lng) {
        return lngs_buttontext.get(lng);
    }
    public String getNameOfPresentLocale() {
        return locales_name.get(Locale.getDefault());
    }
    public void setLangCurrent(String name) {
        config.setProperty(LANG_CURRENT, lngs.get(name));
    }
    public ArrayList<String> getAvailableLocales() {
        final ArrayList<String> res = new ArrayList<>(lngs.keySet());
        return res;
    }
    public ArrayList<String> getAvailableLangs() {
        final ArrayList<String> res = new ArrayList<>(lngs_names.keySet());
        return res;
    }
}
```

Figura 2.27 Contenido de la clase Locales.java

Fuente: Propia

- Paquete **ec.edu.utn.fica.eisic.qmanagement.cliente.comun** que contiene la clase ClientNetProperty que permite la gestión de los comandos que envían los clientes hacia el servidor.

```
package ec.edu.utn.fica.eisic.qmanagement.cliente.comun;

import java.net.InetAddress;
import java.net.UnknownHostException;
import ec.edu.utn.fica.eisic.qmanagement.comun.QLog;
import ec.edu.utn.fica.eisic.qmanagement.comun.modelo.IClientNetProperty;

public class ClientNetProperty implements IClientNetProperty {

    private Integer portServer = -1; // Puerto del servidor
    private Integer portClient = -1; // Puerto del cliente
    private String adress; // Direccion del servidor

    public ClientNetProperty(String[] args) {
        for (Integer i = 0; i < args.length; i++) {
            if ("-sport".equalsIgnoreCase(args[i])) {
                portServer = Integer.parseInt(args[i + 1]);
            }
            if ("-cport".equalsIgnoreCase(args[i])) {
                portClient = Integer.parseInt(args[i + 1]);
            }
            if ("-s".equalsIgnoreCase(args[i])) {
                adress = args[i + 1];
            }
        }
    }

    @Override
    public Integer getPort() {
        return portServer;
    }

    @Override
    public Integer getClientPort() {
        return portClient;
    }

    @Override
    public InetAddress getAddress() {
        InetAddress adr = null;
        try {
            adr = InetAddress.getByName(adress);
        } catch (UnknownHostException ex) {
            QLog.l().logger().error("Error!", ex);
        }
        return adr;
    }
}
```

Figura 2.28 Contenido de la clase ClientNetProperty.java

Fuente: Propia

- Paquete **ec.edu.utn.fica.eisic.qmanagement.comun** contiene las clases comunes tanto al servidor como a los clientes que se conectan al servidor de gestión de colas.

  Clase AUDPServer permite la comunicación UDP entre el servidor y los clientes que se conectan al sistema.

  Clase Base64Coder permite la codificación de las tramas que se envían hacia los clientes desde el servidor.

Clase CustomerState permite obtener los diferentes estados en los que puede estar un cliente que está inmerso en el proceso de atención del sistema.

Clase GsonPool permite el intercambio de información de los turnos entre el servidor y los clientes en formato JSON (JavaScript Object Notation).

Clase NetCommander contiene los métodos para enviar y recibir información de las tareas que se ejecutan en el servidor. Los métodos devuelven la respuesta del servidor en formato XML.

Clase QLog permite escribir mensajes de registro en archivos .log con el objetivo de dejar constancia de una determinada transacción en tiempo de ejecución.

Clase Uses permite configurar todas las tareas que ha de ejecutar el servidor del sistema. Las aplicaciones cliente que se conectan al servidor deben enviar el nombre de la tarea que desean que ejecute el servidor e inmediatamente obtendrán la respuesta correspondiente.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.comun.comandos** contiene las clases que permiten ejecutar cada una de las tareas que se crearon en la clase Uses del paquete ec.edu.utn.fica.eisic.qmanagement.comun.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.comun.excepciones** contiene las clases que implementan la funcionalidad de control de excepciones que se producen tanto en el servidor como en los clientes que se conectan al servidor del sistema.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.comun.modelo** contiene las clases comunes a todos los componentes del sistema las cuales persisten con la base de datos.

  Clase Priority permite gestionar la prioridad de atención de los turnos insertados en la cola de espera.

```
package ec.edu.utn.fica.eisic.qmanagement.comun.modelo;

import java.util.Arrays;
import ec.edu.utn.fica.eisic.qmanagement.comun.Uses;
/**
 * Implementacion de la lista de prioridadesde espera.
 * Prioridad - entero.
 * Entre mas alto es el numero la prioridad es más alta.
 * Restricciones en las posibles prioridades de uso.
 * La prioridad por defaul es la que se detalla en la clase Uses.PRIORITY_NORMAL;
 *
 */
public final class Priority implements IPriority {

    private int priority;

    public Priority(int priority) {
        set(priority);
    }

    public Priority() {
        // La prioridad por default
        priority = Uses.PRIORITY_NORMAL;
    }

    @Override
    public void set(int priority) {
        if (Arrays.binarySearch(Uses.PRIORITYS, priority) == -1) {
            throw new IllegalArgumentException("No es posible establecer la prioridad." +
                    " Valor " + priority +
                    " No pertenece a los valores permitidos: " + Arrays.toString(Uses.PRIORITYS));
        }
        this.priority = priority;
    }

    @Override
    public int get() {
        return priority;
    }

    /**
     * Comparacion de dos prioridades.
     * Prioridad - Entero, Cuanto mayor sea el numero mayor sera la prioridad
     * @parametro priority
     * @return 0 - Las prioridades son
     * 1 - anterior a la prioridad del parametro
     * -1 - abajo de la prioridad del parametro
     */
    @Override
    public int compareTo(IPriority priority) {
        int res = 0;
        if (this.get() > priority.get()) {
            res = 1;
        } else if (this.get() < priority.get()) {
            res = -1;
        }
        return res;
    }
}
```

Figura 2.29 Contenido de la clase Priority.java

Fuente: Propia

Clase QCustomer es la encargada de implementar la funcionalidad de emisión y gestión de un turno. También persiste con la base de datos para almacenar la información del turno en la tabla clients.

- Paquete ec.edu.utn.fica.eisic.qmanagement.extra contiene las interfaces

que se encargan de declarar los métodos que implementan funcionalidades adicionales en los turnos como cambiar de posición en la cola, cambiar el estado de un turno y seleccionar el siguiente servicio.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.hibernate** contiene las clases que permiten cargar los parámetros de conexión con la base de datos para su posterior integración con el framework hibernate.

  Clase SqlServers contiene los métodos y propiedades necesarias para la configuración de los parámetros de conexión con la base de datos.

```
package ec.edu.utn.fica.eisic.qmanagement.hibernate;

import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
import java.util.LinkedList;

public class SqlServers {
    @Expose
    @SerializedName("servers")
    private LinkedList<SqlServer> servers = new LinkedList<>();

    public LinkedList<SqlServer> getServers() {
        return servers;
    }
    public void setServers(LinkedList<SqlServer> servers) {
        this.servers = servers;
    }
    public SqlServers() {
    }
    public SqlServers(LinkedList<SqlServer> servers) {
        this.servers = servers;
    }
    public static class SqlServer {
        @Expose
        @SerializedName("driver")
        private String driver = "com.microsoft.sqlserver.jdbc.SQLServerDriver";
        @Expose
        @SerializedName("url")
        private String url = "jdbc:sqlserver://PCSERVIDORBDD\\TESIS;databaseName=qmanagement?autoReconnect\\u003dtrue\\u0026amp;characterEncoding\\u003dUTF-8";
        @Expose
        @SerializedName("user")
        private String user = "sa";
        @Expose
        @SerializedName("password")
        private String password = "password123";
        @Expose
        @SerializedName("main")
        private Boolean main = false;
        @Expose
        @SerializedName("current")
        private Boolean current = false;
        @Expose
        @SerializedName("name")
        private String name = "";

        public SqlServer() {
        }
        public SqlServer(String name, String user, String password, String url, boolean main, boolean current) {
            this.current = current;
            this.main = main;
            this.password = password;
            this.url = url;
            this.user = user;
            this.name = name;
        }
        public String getDriver() {
            return driver;
        }
        public void setDriver(String driver) {
            this.driver = driver;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public Boolean isCurrent() {
            return current;
        }
        public void setCurrent(Boolean current) {
            this.current = current;
        }
        public String getUrl() {
            return url;
        }
        public void setUrl(String url) {
            this.url = url;
        }
        public String getUser() {
            return user;
        }
        public void setUser(String user) {
            this.user = user;
        }
        public String getPassword() {
            return password;
        }
        public void setPassword(String password) {
            this.password = password;
        }
        public Boolean isMain() {
            return main;
        }
        public void setMain(Boolean main) {
            this.main = main;
        }
        @Override
        public String toString() {
            return name + "    " + (isCurrent() ? "C" : "_") + "    " + (isMain() ? "M" : "_") + "    " + getUrl();
        }
    }
}
```

Figura 2.30 Contenido de la clase SqlServers.java

Fuente: Propia

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor** es el paquete más importante del sistema porque contiene la clase ContextListener que es la primera clase que se ejecuta al momento de levantar el sistema.

  La clase ContextListener se encarga de abrir los puertos de comunicación en el servidor del sistema, cargar el contexto de la aplicación y todas las funcionalidades implementadas en el sistema de gestión del flujo clientes.

  Para que esta clase se ejecute primero se debe indicar en el archivo de configuración web.xml en la etiqueta <listener-class>.

  ```
  <listener>
  <listener-class>
  ec.edu.utn.fica.eisic.qmanagement.servidor.ContextListener
  </listener-class>
  </listener>
  ```

```
package ec.edu.utn.fica.eisic.qmanagement.servidor;
import com.google.gson.Gson;
import com.google.gson.JsonParseException;
import com.google.gson.JsonSyntaxException;
import com.google.gson.annotations.Expose;
import com.google.gson.annotations.SerializedName;
import java.io.*;
import java.net.*;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Locale;
import java.util.Scanner;
import java.util.ServiceLoader;
import ec.edu.utn.fica.eisic.qmanagement.About;
import ec.edu.utn.fica.eisic.qmanagement.cliente.Locales;
import ec.edu.utn.fica.eisic.qmanagement.comun.GsonPool;
import ec.edu.utn.fica.eisic.qmanagement.comun.Uses;
import ec.edu.utn.fica.eisic.qmanagement.comun.QLog;
import ec.edu.utn.fica.eisic.qmanagement.comun.comandos.JsonRPC20;
import ec.edu.utn.fica.eisic.qmanagement.comun.comandos.RpcGetAdvanceCustomer;
import ec.edu.utn.fica.eisic.qmanagement.comun.excepciones.ServerException;
import ec.edu.utn.fica.eisic.qmanagement.comun.modelo.ATalkingClock;
import ec.edu.utn.fica.eisic.qmanagement.comun.modelo.QCustomer;
import ec.edu.utn.fica.eisic.qmanagement.hibernate.AnnotationSessionFactoryBean;
import ec.edu.utn.fica.eisic.qmanagement.servidor.controlador.Executer;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QPlanService;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QService;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QServiceTree;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QUser;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QUserList;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.parking.QPostponedList;
import static java.lang.Thread.NORM_PRIORITY;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import java.net.SocketTimeoutException;
import java.util.Properties;

public class ContextListener extends Thread implements ServletContextListener {
    private Socket socket=null;
    private static volatile boolean globalExit = false;
    private static final String KEY_HTML = "-HTTP";
    public static String pathlangs = "";
    public static String pathversion = "";
    public static String pathtemp = "";
    public static String pathtemp_filetemp="";
    public static String pathtemp_filecomplex="";
    public static String pathtemp_filestatistic="";
    public static String path_applicationcontextXML="";
    public static String path_serverfile="";

    public ContextListener() {
            super();
        }

        public ContextListener(Socket socket) {
        this.socket = socket;
         //Lanzamiento de un nuevo hilo
        setDaemon(true);
        setPriority(NORM_PRIORITY);
    }

    @Override
    public void contextInitialized(ServletContextEvent event) {
        pathlangs = event.getServletContext().getRealPath("/config/langs.properties");
        pathversion=event.getServletContext().getRealPath("/config/version.properties");
        pathtemp=event.getServletContext().getRealPath("/temp");
        pathtemp_filetemp=event.getServletContext().getRealPath("/temp/temp.json");
        pathtemp_filecomplex=event.getServletContext().getRealPath("/temp/complex.json");
        pathtemp_filestatistic=event.getServletContext().getRealPath("/temp/temp_statistic.xml");
        path_applicationcontextXML=event.getServletContext().getServletContextName();
        path_serverfile=event.getServletContext().getRealPath("/files/server.file");


        Locale.setDefault(Locales.getInstance().getLangCurrent());
        final long start = System.currentTimeMillis();
        About.load();
        Thread thread = new Thread(){
                        public void run(){

                                loadPool();
                                if (!(Uses.format_HH_mm.format(ServerProps.getInstance().getProps().getStartTime()).equals(Uses.format_HH_mm.format(ServerProps.getInstance().getPro
                                {
                                        // Timer que limpia todos los servicios
                                        ATalkingClock clearServices = new ATalkingClock(Uses.DELAY_CHECK_TO_LOCK, 0) {

                                                @Override
                                                public void run() {
                                                        // Es cero
                                                        if (!QLog.isRETAIN && Uses.format_HH_mm.format(new Date(new Date().getTime() + 10 * 60 * 1000)).equals(Uses.format_HH_mm.for
                                                                QLog.l().logger().info("Limpiar todos los servicios.");
                                                                // Limpiar todos los servicios del ultimo dia
                                                                ContextListener.clearAllQueue();
                                                        }
                                                }
                                        };
                                        clearServices.start();
                                }
}
                                // Socket que debe estar abierto en el puerto local 3128
                                final ServerSocket server;
                                try {
                                        QLog.l().logger().info("Servidor del sistema escucha el puerto \"" + ServerProps.getInstance().getProps().getServerPort() + "\".");
                                        server = new ServerSocket(ServerProps.getInstance().getProps().getServerPort());
                                } catch (IOException e) {
                                        throw new ServerException("Error al crear un socket de servidor: " + e);
                                } catch (Exception e) {
                                        throw new ServerException("Error de red: " + e);
                                }
```

Figura 2.31 Contenido de la clase ContextListener.java

Fuente: Propia

Clase QSession permite recuperar la sesión de los usuarios conectados al sistema.

Clase Spring implementa la funcionalidad que permite gestionar los diferentes componentes del sistema para lo cual lee el archivo applicationContext.xml donde se definen todos los beans de la aplicación.

```
package ec.edu.utn.fica.eisic.qmanagement.servidor;

import com.mchange.v2.c3p0.ComboPooledDataSource;
import java.io.Serializable;
import java.util.Collection;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.criterion.DetachedCriteria;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.BeanCreationException;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.orm.hibernate4.HibernateTransactionManager;
import org.springframework.transaction.support.TransactionTemplate;
import ec.edu.utn.fica.eisic.qmanagement.comun.excepciones.ServerException;

public class Spring {

    private final BeanFactory factory;
    private final String driverClassName;
    private final String url;
    private final String username;
    private final String password;

    public String getDriverClassName() {
        return driverClassName;
    }
    public BeanFactory getFactory() {
        return factory;
    }
    public String getPassword() {
        return password;
    }
    public String getUrl() {
        return url;
    }
    public String getusername() {
        return username;
    }
    public TransactionTemplate getTt() {
        return new TransactionTemplate(getTxManager());
    }
    public HibernateTransactionManager getTxManager() {
        return (HibernateTransactionManager) factory.getBean("transactionManager");
    }
    private Spring() {
        System.out.println("SOLSITO");
        try {
            factory = new ClassPathXmlApplicationContext("applicationContext.xml");
        } catch (BeanCreationException ex) {
            throw new ServerException("Error creando la clase bean del contexto de la aplicacion: " + ex.getCause().getMessage() );
        } catch (BeansException ex) {
            throw new ServerException("Error en la clase bean del contexto de la aplicacion: " + ex.getCause().getMessage()  );
        } catch (Exception ex) {
            throw new ServerException("Fallo al crear el contexto de la aplicacion: " + ex);
        }
       final ComboPooledDataSource bds = (ComboPooledDataSource) factory.getBean("c3poDataSource");
       driverClassName = bds.getDriverClass();
       url = bds.getJdbcUrl();
       username = bds.getUser();
       password = bds.getPassword();
    }

    public static Spring getInstance() {
        return SpringHolder.INSTANCE;
    }
    private static class SpringHolder {
        private static final Spring INSTANCE = new Spring();
    }
    public Spring getHt() {
        return this;
    }
    public void saveAll(Collection list) {
        final Session ses = getTxManager().getSessionFactory().getCurrentSession();
        list.stream().forEach((object) -> {
            ses.save(object);
        });
        ses.flush();
    }
    public void saveOrUpdateAll(Collection list) {
        final Session ses = getTxManager().getSessionFactory().getCurrentSession();
        list.stream().forEach((object) -> {
            ses.saveOrUpdate(object);
        });
        ses.flush();
    }
    public void saveOrUpdate(Object obj) {
        final Session ses = getTxManager().getSessionFactory().getCurrentSession();
        ses.saveOrUpdate(obj);
        ses.flush();
    }
    public void deleteAll(Collection list) {
        final Session ses = getTxManager().getSessionFactory().getCurrentSession();
        list.stream().forEach((object) -> {
            ses.delete(object);
        });
        ses.flush();
    }
    public void delete(Object obj) {
        final Session ses = getTxManager().getSessionFactory().getCurrentSession();
        ses.delete(obj);
        ses.flush();
    }
    public List loadAll(Class clazz) {
        final Session ses = getTxManager().getSessionFactory().openSession();
        try {
            return ses.createCriteria(clazz).list();
        } finally {
            ses.close();
        }
    }
    public void load(Object obj, Serializable srlzbl) {
        final Session ses = getTxManager().getSessionFactory().openSession();
        try {
            ses.load(obj, srlzbl);
        }
        catch (Exception ex) {
            throw new ServerException("ERROR AL LEER TABLAS DE BDD: " + ex);
        }
        finally {
            ses.close();
        }
    }
    public List findByCriteria(DetachedCriteria dCriteria) {
        List list;
        final Session ses = getTxManager().getSessionFactory().openSession();
        try {
            list = dCriteria.getExecutableCriteria(ses).list();
        } finally {
            ses.close();
        }
        return list;
    }
    public <T> T get(Class<T> clazz, Serializable srlzbl) {
        final Session ses = getTxManager().getSessionFactory().openSession();
        try {
            return (T) ses.get(clazz, srlzbl);
        } finally {
            ses.close();
        }
    }
    public List find(String hql) {
        final Session ses = getTxManager().getSessionFactory().openSession();
        try {
            return ses.createQuery(hql).list();
        } finally {
            ses.close();
        }
    }
    public SessionFactory getSessionFactory() {
        return getTxManager().getSessionFactory();
    }
}
```

Figura 2.32 Contenido de la clase Spring.java

Fuente: Propia

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.controlador** contiene las clases que permiten ejecutar las peticiones de los clientes que se conectan al servidor del sistema.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.dao** contiene las interfaces que permiten abstraer y encapsular todos los accesos a la base de datos. También contiene las clases que implementan las interfaces DAO las cuales implementan la lógica específica de hibernate para manejar los datos persistentes.

```
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet;
import java.util.List;

public interface IQNetDAO {

        public void addQNet(QNet net);
        public void updateQNet(QNet net);
        public void deleteQNet(QNet net);
        public QNet getQNetById(int id);
        public List<QNet> getQNets();
}
```

Figura 2.33 Contenido de la interface IQNetDAO.java

Fuente: Propia

En la interface IQNetDAO se declaran los métodos necesarios para el desarrollo de las operaciones CRUD en la tabla net de la base de datos del sistema.

```
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import java.util.List;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet;
import java.io.Serializable;
import org.hibernate.SessionFactory;

public class QNetDAO implements IQNetDAO,Serializable {
    private SessionFactory sessionFactory;
        public SessionFactory getSessionFactory() {
                return sessionFactory;
        }
        public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
        }
        @Override
        public void addQNet(QNet net) {
                getSessionFactory().getCurrentSession().save(net);
        }
        @Override
        public void deleteQNet(QNet net) {
                getSessionFactory().getCurrentSession().delete(net);
        }
        @Override
        public void updateQNet(QNet net) {
                getSessionFactory().getCurrentSession().merge(net);
        }
        @Override
        public QNet getQNetById(int id) {
                List list = getSessionFactory().getCurrentSession()
                .createQuery("from QNet where id=?")
                .setParameter(0, id).list();
                return (QNet)list.get(0);
        }
        @Override
        public List<QNet> getQNets() {
                List list = getSessionFactory().getCurrentSession().createQuery("from QNet").list();
                return list;
        }
}
```

Figura 2.34 Contenido de la clase QNetDAO.java

Fuente: Propia

La clase QNetDAO implementa los métodos declarados en la interface IQNetDAO.

```
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.comun.modelo.QCustomer;
import java.util.Date;
import java.util.List;

public interface IQClientDAO {
        public void addQClient(QCustomer user);
        public void updateQClient(QCustomer user);
        public void deleteQClient(QCustomer user);
        public QCustomer getQClientById(int id);
        public List<QCustomer> getQClients();
        public List<QCustomer> getQClientsAtendidos(Date desde, Date hasta, String servicio,String usuario);
}
```

Figura 2.35 Contenido de la interface IQClientDAO.java

Fuente: Propia

En la interface IQClientDAO se declaran los métodos necesarios para el desarrollo de las operaciones CRUD en la tabla clients de la base de datos del sistema.

```java
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.comun.modelo.QCustomer;
import static ec.edu.utn.fica.eisic.qmanagement.servidor.utilidades.Fechas.DiaPiso;
import static ec.edu.utn.fica.eisic.qmanagement.servidor.utilidades.Fechas.DiaTecho;
import java.io.Serializable;
import java.util.Date;
import java.util.List;
import org.hibernate.SessionFactory;

public class QClientDAO implements IQClientDAO,Serializable {
        private SessionFactory sessionFactory;
      public SessionFactory getSessionFactory() {
            return sessionFactory;
      }
      public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
        }
      @Override
      public void addQClient(QCustomer client) {
            getSessionFactory().getCurrentSession().save(client);
      }
      @Override
      public void deleteQClient(QCustomer client) {
            getSessionFactory().getCurrentSession().delete(client);
      }
      @Override
      public void updateQClient(QCustomer client) {
            getSessionFactory().getCurrentSession().merge(client);
      }
      @Override
      public QCustomer getQClientById(int id) {
            List list = getSessionFactory().getCurrentSession()
            .createQuery("from QCustomer where id=?")
            .setParameter(0, id).list();
            return (QCustomer)list.get(0);
      }
      @Override
      public List<QCustomer> getQClients() {
            List list = getSessionFactory().getCurrentSession().createQuery("from QCustomer").list();
                //Iterator iter = list.iterator();
                //while (iter.hasNext())
                //System.out.println(iter.next());

            return list;
      }
        @Override
      public List<QCustomer> getQClientsAtendidos(Date desde, Date hasta, String servicio,String usuario) {
            String c_servicio = "";
            String c_usuario="";
            if (!servicio.isEmpty()) {
                  c_servicio = " and q.service like '" + servicio + "'";
            } else {
                  c_servicio = " and q.service IS NOT NULL";
            }
            if (!usuario.isEmpty()) {
                  c_usuario = " and q.user like '" + usuario + "'";
            } else {
                  c_usuario = " and q.user IS NOT NULL";
            }
            List list = getSessionFactory().getCurrentSession().createQuery("from QCustomer q where q.standTime>=? "
                  + "and q.standTime<=? "+c_servicio+" "+c_usuario+" order by q.standTime asc ")
            .setParameter(0, DiaPiso(desde)).setParameter(1,DiaTecho(hasta)).list();

            return list;
      }
}
```

Figura 2.36 Contenido de la clase QClientDAO.java

Fuente: Propia

La clase QClientDAO implementa los métodos declarados en la interface IQClientDAO.

```java
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QUser;
import java.util.List;

public interface IQUserDAO {
        public void addQUser(QUser user);
      public void updateQUser(QUser user);
      public void deleteQUser(QUser user);
      public QUser getQUserById(Long id);
        public QUser getQUserLogin(String usuario, String clave);
      public List<QUser> getQUsers();
}
```

Figura 2.37 Contenido de la interface IQUserDAO.java

Fuente: Propia

En la interface IQUserDAO se declaran los métodos necesarios para el desarrollo de las operaciones CRUD en la tabla users de la base de datos del sistema.

```java
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QUser;
import java.io.Serializable;
import java.util.List;
import org.hibernate.SessionFactory;

public class QUserDAO implements IQUserDAO,Serializable {
      private SessionFactory sessionFactory;
      public SessionFactory getSessionFactory() {
            return sessionFactory;
      }
      public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
        }
      @Override
      public void addQUser(QUser user) {
            getSessionFactory().getCurrentSession().save(user);
      }
      @Override
      public void deleteQUser(QUser user) {
            getSessionFactory().getCurrentSession().delete(user);
      }
      @Override
      public void updateQUser(QUser user) {
            getSessionFactory().getCurrentSession().merge(user);
      }
      @Override
      public QUser getQUserById(Long id) {
            List list = getSessionFactory().getCurrentSession()
            .createQuery("from QUser where id=?")
            .setParameter(0, id).list();
            return (QUser)list.get(0);
      }
      @Override
      public QUser getQUserLogin(String usuario, String clave) {
            System.out.println("Usuario"+usuario+"    Clave:"+clave);
            List list = getSessionFactory().getCurrentSession()
            .createQuery("from QUser u where u.name=? and u.password=?")
            .setParameter(0, usuario).setParameter(1,clave).list();
            return (QUser)list.get(0);
      }
      @Override
      public List<QUser> getQUsers() {
            List list = getSessionFactory().getCurrentSession().createQuery("from QUser").list();
            return list;
      }
}
```

Figura 2.38 Contenido de la clase QUserDAO.java

Fuente: Propia

La clase QUserDAO implementa los métodos declarados en la interface IQUserDAO.

```java
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QService;
import java.util.List;

public interface IQServiceDAO {
        public void addQService(QService user);
      public void updateQService(QService user);
      public void deleteQService(QService user);
      public QService getQServiceById(Long id);
      public List<QService> getQServices();
}
```

Figura 2.39 Contenido de la interface IQServiceDAO.java

Fuente: Propia

En la interface IQServiceDAO se declaran los métodos necesarios para el desarrollo de las operaciones CRUD en la tabla services de la base de datos del sistema.

```java
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QService;
import java.io.Serializable;
import java.util.List;
import org.hibernate.SessionFactory;

public class QServiceDAO implements IQServiceDAO,Serializable {
    private SessionFactory sessionFactory;
      public SessionFactory getSessionFactory() {
            return sessionFactory;
      }
      public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
        }
      @Override
      public void addQService(QService service) {
            getSessionFactory().getCurrentSession().save(service);
      }
      @Override
      public void deleteQService(QService service) {
            getSessionFactory().getCurrentSession().delete(service);
      }
      @Override
      public void updateQService(QService service) {
            getSessionFactory().getCurrentSession().merge(service);
      }
      @Override
      public QService getQServiceById(Long id) {
            List list = getSessionFactory().getCurrentSession()
            .createQuery("from QService where id=?")
            .setParameter(0, id).list();
            return (QService)list.get(0);
      }
      @Override
      public List<QService> getQServices() {
            List list = getSessionFactory().getCurrentSession().createQuery("from QService").list();
            return list;
      }
}
```

Figura 2.40 Contenido de la clase QServiceDAO.java

Fuente: Propia

La clase QServiceDAO implementa los métodos declarados en la interface IQServiceDAO.

```java
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QPlanService;
import java.util.List;
public interface IQPlanDAO {
	public void addQPlan(int idservicio,int idusuario, int coeficiente,int flexcoef);
	public void updateQPlan(QPlanService plan);
	public void deleteQPlan(QPlanService plan);
	public QPlanService getQPlanById(int id);
	public List<QPlanService> getQPlanes();
}
```

Figura 2.41 Contenido de la interface IQPlanDAO.java

Fuente: Propia

En la interface IQPlanDAO se declaran los métodos necesarios para el desarrollo de las operaciones CRUD en la tabla services_users de la base de datos del sistema.

```java
package ec.edu.utn.fica.eisic.qmanagement.servidor.dao;

import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QPlanService;
import java.io.Serializable;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.SessionFactory;
public class QPlanDAO implements IQPlanDAO,Serializable {
    private SessionFactory sessionFactory;
      public SessionFactory getSessionFactory() {
            return sessionFactory;
      }
      public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
        }
      @Override
      public void addQPlan(int idservicio,int idusuario, int coeficiente,int flexcoef) {
        Query query=getSessionFactory().getCurrentSession().createSQLQuery("insert into services_users "
          + "(service_id,user_id,coefficient,flexible_coef) Values (:valor1,:valor2,:valor3,:valor4)");
        query.setParameter("valor1", idservicio);
        query.setParameter("valor2", idusuario);
        query.setParameter("valor3", coeficiente);
        query.setParameter("valor4", flexcoef);
        query.executeUpdate();
      }
      @Override
      public void deleteQPlan(QPlanService plan) {
            getSessionFactory().getCurrentSession().delete(plan);
      }
      @Override
      public void updateQPlan(QPlanService plan) {
            getSessionFactory().getCurrentSession().merge(plan);
      }
      @Override
      public QPlanService getQPlanById(int id) {
            List list = getSessionFactory().getCurrentSession()
            .createQuery("from QPlanService where id=?")
            .setParameter(0, id).list();
            return (QPlanService)list.get(0);
      }
      @Override
      public List<QPlanService> getQPlanes() {
            List list = getSessionFactory().getCurrentSession().createQuery("from QPlanService").list();
            return list;
      }
}
```

Figura 2.42 Contenido de la clase QPlanDAO.java

Fuente: Propia

La clase QPlanDAO implementa los métodos declarados en la interface IQPlanDAO.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.managedbean** contiene las clases de gestión que permiten conectar la capa de presentación con la capa de negocio.

  La clase QNetManagedBean permite conectar la vista CRUDRed.xhtml de JSF con la capa de negocio y de datos.

  En la vista CRUDRed.xhtml se ingresan los datos de comunicación con del servidor del sistema de gestión de clientes.

```
package ec.edu.utn.fica.eisic.qmanagement.servidor.managedbean;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.ViewScoped;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet;
import ec.edu.utn.fica.eisic.qmanagement.servidor.servicio.IQNetService;
import ec.edu.utn.fica.eisic.qmanagement.servidor.utilidades.JSFUtil;
import java.util.Date;
import javax.faces.application.FacesMessage;
import javax.faces.event.ActionEvent;
import javax.faces.event.ValueChangeEvent;
import org.richfaces.component.UIDataTable;

@ManagedBean(name="qnetMB")
@ViewScoped
public class QNetManagedBean implements Serializable {
        private static final long serialVersionUID = 1L;
        private static final String SUCCESS = "success";
        private static final String ERROR   = "error";
        private static final int CLIENT_ROWS_IN_AJAX_MODE = 15;
        @ManagedProperty(value="#{QNetService}")
        IQNetService qnetService;
        List<QNet> qnetList;
        private QNet net;
        private Integer id;
        private Integer serverPort;
        private Integer webServerPort;
        private Integer clientPort;
        private Date finishTime;
        private int indiceActual;
        private int page = 1;
        private int clientRows;

        public void switchAjaxLoading(ValueChangeEvent event) {
        this.clientRows = (Boolean) event.getNewValue() ? CLIENT_ROWS_IN_AJAX_MODE : 0;
        }
        public int getIndiceActual() {
        return indiceActual;
        }
        public void setIndiceActual(int indiceActual) {
        this.indiceActual = indiceActual;
        }
        public int getPage() {
        return page;
        }
        public void setPage(int page) {
        this.page = page;
        }
        public int getClientRows() {
        return clientRows;
        }
        public void setClientRows(int clientRows) {
        this.clientRows = clientRows;
        }
        public IQNetService getQnetService() {
                return qnetService;
        }
        public void setQnetService(IQNetService qnetService) {
                this.qnetService = qnetService;
        }
        public List<QNet> getQnetList() {
                qnetList = new ArrayList<QNet>();
                qnetList.addAll(getQnetService().getQNets());
                return qnetList;
        }
        public void setQnetList(List<QNet> qnetList) {
                this.qnetList = qnetList;
        }
        public void deleteAction(ActionEvent e) {
         UIDataTable dataTab= (UIDataTable)e.getComponent().getParent().getParent();
        if (dataTab.getRowData() != null)
            {
            try
            {
            QNet net1 = (QNet) dataTab.getRowData();
            getQnetService().deleteQNet(net1);
            JSFUtil.writeMessage(FacesMessage.SEVERITY_INFO, "Operacion exitosa", "El usuario se elimino correctamente");

            } catch (Exception ex) {
            ex.printStackTrace(System.err);
            }

            }
        }
        public void updateNet(ActionEvent e){
            try{
            QNet net1= net;
            net1.setServerPort(getServerPort());
            net1.setWebServerPort(getWebServerPort());
            net1.setClientPort(getClientPort());
            net1.setFinishTime(getFinishTime());
            getQnetService().updateQNet(net1);
            } catch(Exception ex){
            ex.printStackTrace(System.err);
            }
        }
        public void setQNet(QNet net) {
                this.net = net;
        }
        public QNet getQNet() {
                return net;
        }
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public Integer getServerPort() {
                return serverPort;
        }
        public void setServerPort(Integer serverPort) {
                this.serverPort = serverPort;
        }
        public Integer getWebServerPort() {
                return webServerPort;
        }
        public void setWebServerPort(Integer webServerPort) {
                this.webServerPort = webServerPort;
        }
        public Integer getClientPort() {
                return clientPort;
        }
        public void setClientPort(Integer clientPort) {
                this.clientPort = clientPort;
        }
        public Date getFinishTime() {
                return finishTime;
        }
        public void setFinishTime(Date finishTime) {
                this.finishTime = finishTime;
        }
    public void addQNet() {
                try {
                        QNet qnet = new QNet();
                        qnet.setId(getId());
                        qnet.setServerPort(getServerPort());
                        qnet.setWebServerPort(getWebServerPort());
                        qnet.setClientPort(getClientPort());
                        qnet.setFinishTime(getFinishTime());
                        getQnetService().addQNet(qnet);
                        this.LimpiarRegistro();
                } catch (Exception e) {
                                e.printStackTrace(System.err);
                }finally{
                this.LimpiarRegistro();
                }
        }
    void LimpiarRegistro()
    {
    this.setServerPort(null);
    this.setWebServerPort(null);
    this.setClientPort(null);
    this.setFinishTime(null);
    this.net=new QNet();
    }
}
```

Figura 2.43 Contenido de la clase QNetManagedBean.java

Fuente: Propia

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.modelo** contiene los objetos de negocio POJOs necesarios para el manejo de la persistencia mediante hibernate.

```
package ec.edu.utn.fica.eisic.qmanagement.servidor.modelo;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.Transient;

@Entity
@Table(name = "net")
public class QNet implements Serializable {
    public QNet() {
    }
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    public void setId(Integer id) {
        this.id = id;
    }
    public Integer getId() {
        return id;
    }
    @Column(name = "server_port")
    private Integer serverPort;
    public void setServerPort(Integer serverPort) {
        this.serverPort = serverPort;
    }
    public Integer getServerPort() {
        return serverPort;
    }
    @Column(name = "web_server_port")
    private Integer webServerPort;
    public void setWebServerPort(Integer webServerPort) {
        this.webServerPort = webServerPort;
    }
    public Integer getWebServerPort() {
        return webServerPort;
    }
    @Column(name = "client_port")
    private Integer clientPort;
    public void setClientPort(Integer clientPort) {
        this.clientPort = clientPort;
    }
    public Integer getClientPort() {
        return clientPort;
    }
    @Column(name = "start_time")
    @Temporal(javax.persistence.TemporalType.TIME)
    private Date startTime;
    public void setStartTime(Date startTime) {
        this.startTime = startTime;
    }
    public Date getStartTime() {
        return startTime;
    }
    @Column(name = "finish_time")
    @Temporal(javax.persistence.TemporalType.TIME)
    private Date finishTime;
    public void setFinishTime(Date finishTime) {
        this.finishTime = finishTime;
    }
    private Date finishTime;
    public void setFinishTime(Date finishTime) {
        this.finishTime = finishTime;
    }
    public Date getFinishTime() {
        return finishTime;
    }
    @Column(name = "version")
    private String version = "No asignado";
    public String getVersion() {
        return version;
    }
    public void setVersion(String version) {
        this.version = version;
    }
    @Column(name = "last_number")
    private Integer lastNumber;
    public Integer getLastNumber() {
        return lastNumber;
    }
    public void setLastNumber(Integer lastNumber) {
        this.lastNumber = lastNumber;
    }
    @Column(name = "first_number")
    private Integer firstNumber;
    public Integer getFirstNumber() {
        return firstNumber;
    }
    public void setFirstNumber(Integer firstNumber) {
        this.firstNumber = firstNumber;
    }
    @Column(name = "numering")
    private Boolean numering;
    public Boolean getNumering() {
        return numering;
    }
    public void setNumering(Boolean numering) {
        this.numering = numering;
    }
    @Column(name = "point")
    private Integer point;
    public Integer getPoint() {
        return point;
    }
    public void setPoint(Integer point) {
        this.point = point;
    }
    @Column(name = "branch_id")
    private Long branchOfficeId;
    public Long getBranchOfficeId() {
        return branchOfficeId;
    }
    public void setBranchOfficeId(Long branchOfficeId) {
        this.branchOfficeId = branchOfficeId;
    }
    @Column(name = "limit_recall")
    private Integer limitRecall;

    public Integer getLimitRecall() {
        return limitRecall;
    }
    public void setLimitRecall(Integer limitRecall) {
        this.limitRecall = limitRecall;
    }
}
```

Figura 2.44 Contenido de la clase QNet.java

Fuente: Propia

La clase QNet declara todos los atributos de la tabla net con sus respectivos set y get.

Esta clase implementa la lógica de negocio que permite configurar los parámetros para el funcionamiento del servidor tales como hora de inicio de atención, hora de finalización, puerto de comunicación, etc.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.calendario** contiene las clases que permiten controlar las fechas en las cuales estarán habilitados los servicios para los cuales se toman los turnos.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.horario** contiene las clases que permiten controlar el horario de trabajo en el cual los puestos de atención estarán disponibles para atender a los clientes que están en la cola de espera.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.parking** contiene la clase QPostponedList que implementa la funcionalidad para transferir a un cliente que está siendo atendido a una cola de espera.

- Paquete **ec.edu.utn.fica.eisic.qmanagement.servidor.servicio** contiene las clases que permiten exponer los servicios de negocio que interactúan con los objetos de negocio definidos en el paquete ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.

  La clase QNetService contiene la lógica de negocio relacionada con la configuración de red del sistema.

```
package ec.edu.utn.fica.eisic.qmanagement.servidor.servicio;
import java.util.List;
import ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet;
import ec.edu.utn.fica.eisic.qmanagement.servidor.dao.IQNetDAO;
import java.io.Serializable;
import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

@Transactional(readOnly = true)
public class QNetService implements IQNetService,Serializable {

	//  QNetDAO es inyectado...
	IQNetDAO qnetDAO;

	@Transactional(readOnly = false)
	@Override
	public void addQNet(QNet net) {
		getQnetDAO().addQNet(net);
	}
	@Transactional(readOnly = false)
	@Override
	public void deleteQNet(QNet net) {
		getQnetDAO().deleteQNet(net);
	}

	@Transactional(readOnly = false)
	@Override
	public void updateQNet(QNet net) {
		getQnetDAO().updateQNet(net);
	}
	@Override
	public QNet getQNetById(int id) {
		return getQnetDAO().getQNetById(id);
	}
	@Override
	public List<QNet> getQNets() {
		return getQnetDAO().getQNets();
	}
	public IQNetDAO getQnetDAO() {
	    return qnetDAO;
	}
	public void setQnetDAO(IQNetDAO netDAO) {
		this.qnetDAO = netDAO;
	}

}
```

Figura 2.45 Contenido de la clase QNetService.java

Fuente: Propia

- Paquete ec.edu.utn.fica.eisic.qmanagement.cliente.printer contiene las clases que implementan la lógica de negocio de la aplicación que emite los turnos.

- Paquete ec.edu.utn.fica.eisic.qmanagement.cliente.display contiene las clases que implementan la lógica de negocio de la aplicación que muestra los turnos en la pantalla LCD.

- Paquete ec.edu.utn.fica.eisic.qmanagement.cliente.teclado contiene las clases que implementan la lógica de negocio de la aplicación que llama los turnos desde los puestos de atención.

- Paquete ec.edu.utn.fica.eisic.qmanagement.servidor.utilidades contiene clases que permiten agregar funcionalidades adicionales al sistema en caso de que sea necesario.

CAPÍTULO III

CONCLUSIONES Y RECOMENDACIONES

3.1 CONCLUSIONES

- El desarrollo de esta aplicación permitirá a la empresa Q-Matic Ecuador disponer de una herramienta multiplataforma para la gestión de colas de espera.

- La utilización de herramientas de desarrollo así como frameworks de terceros, y su posterior integración se logró gracias al patrón de diseño MVC el cual permite identificar de manera clara los componentes que se utilizan en cada capa de la aplicación.

- Se comprobó que siguiendo las especificaciones de la plataforma J2EE es posible obtener un sistema robusto, fiable, escalable y seguro.

- Se desarrolló una aplicación muy intuitiva para el usuario, el cual puede gestionar de manera fácil el sistema de gestión de clientes.

- Al realizar el diseño del aplicativo siguiendo las especificaciones del Proceso de Desarrollo de Software RUP se abre la posibilidad de realizar actualizaciones, mejoras o crear nuevos módulos del sistema en el futuro.

3.2 RECOMENDACIONES

- Al elegir las herramientas de desarrollo se debe tomar en cuenta el nivel de conocimiento que se posee, ya que de eso depende el cumplimiento de los plazos establecidos para el desarrollo de un sistema.

- En el caso de que un sistema requiera de dispositivos de harware adicionales a los computadores normales, se debe evaluar su disponibilidad en el mercado así como su costo.

- Antes de elegir el tipo de sistema que se va a desarrollar se debe evaluar la cantidad de información que existe y el nivel de conocimiento que se posee sobre el área en el que se implementará el sistema.

- Antes de instalar el sistema, se debe verificar que la institución o empresa donde se instale disponga de la infraestructura de red necesaria para la comunicación entre los módulos del sistema.

BIBLIOGRAFIA

Referencias Bibliográficas

Carro, R. (2009). Investigación de Operaciones en Administración. (Edición 2009). En Roberto Carro, Investigación de Operaciones en Adminsitración (pág 421). Mar del Plata, PINCU.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (págs. 147-184). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (pág. 273). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (pág. 365). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (pág. 439). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (págs. 537-578). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (págs. 669-706). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (págs. 731-776). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (págs. 783-816). España: RA-MA Editorial.

Ceballos, F.J. (2010). Java 2 Curso de Programación. (4ª ed.). En Fco. Javier Ceballos, Java 2 Curso de Programación (págs. 819-822). España: RA-MA Editorial.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 72-85). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (pág. 173). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 241-289). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 470-506). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 830-867). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 905-930). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 1046-1093). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 1128-1150). Boston, Massachusetts: Prentice Hall.

Deitel, P., Deitel, H. (2012). Java How To Program. (9ª ed.). En Paul Deitel, Harvey Deitel, Java How To Program (págs. 1236-1295). Boston, Massachusetts: Prentice Hall.

Render, B., Stair, R., Hanna, M. (2012). Métodos cuantitativos para los negocios. (11ª ed.). En Barry Render, Ralph Stair, Michael Hanna (págs. 499-532). México, PEARSON.

Ritzman, L., Krajewski, L., Malhotra, M. (2008). Administración de Operaciones, Procesos y cadenas de valor. (8ª ed.). En Ritzman, L., Krajewski, L., Malhotra, M., Administración de Operaciones, Procesos y cadenas de valor (pág. 292-308). México, Prentice Hall.

Schroeder, R. (1992). Administración de Operaciones. (3ª ed.). En Roger Schroeder, Administración de Operaciones (pág. 164). México, MCGRAW-HILL.

Rerefencias de Internet

Epson Ibérica S.A. (15 de Febrero del 2015). Epson TM-T88V Serie. Obtenido de http://www.epson.es/es/es/viewcon/corporatesite/products/mainunits/overview/8396?searchKey=TM-T88V

Google. (15 de Enero del 2015). The J2EE Tutorial. Obtenido de https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxwcmFuZWV0aGVib29rc3xneDoxOTBjMDNiYzcxOGViOGVk

Hopkins, (2 de Enero del 2015). JSF 2 Tutorial Series. Obtenido de http://www.coreservlets.com/JSF-Tutorial/jsf27

MarkMonitor, Inc. (10 de Abril del 2015). Manual de Instalación de SQL Server 2008. Obtenido de http://es.slideshare.net/SonGoku10/manual-de-instalacin-sql-server-2008

MarkMonitor, Inc. (10 de Mayo del 2015). Manual de Instalación de Apache Tomcat 8. Obtenido de http://es.slideshare.net/pablozacrosuarez/instalacin-de-apache-tomcat-8

Portilla, L. M., Arias Montoya, L. & Fernández Henao, S. A. (5 de Junio del 2015). Análisis de Líneas de Espera a través de Teoría de Colas y Simulación. Obtenido de http://www.redalyc.org/articulo.oa?id=84920977012

Q-Matic. (17 de Enero del 2011). Sistema de gestión de colas de espera Q-Matic. Obtenido de http://www.q-matic.com/

SAMSUNG SDS CO., LTD. (5 de Enero del 2011). Monitor Samsung SMT-3223. Obtenido de https://www.samsungsecurity.com/samsung/upload/product_specifications/smt-3223-specifications.pdf

TV Media. (18 de Enero del 2011). Sistema administrador de colas TV Media. Obtenido de http://www.cejamericas.org/doc/documentos/sistemaadministradordecolas.pdf

ANEXOS

ANEXO A: MANUAL DE ADMINISTRACION DEL SISTEMA

A.1 Introducción

Este manual está dirigido al tipo de usuario administrador que será el encargado de la configuración general del sistema de gestión del flujo de clientes.

A.2 Objetivo

Proporcionar una guía de operación y manejo de las funciones del Sistema de Gestión del Flujo de Clientes QManagement.

1. Ingreso al Portal de Gestión del Flujo de Clientes "QManagement" como administrador

Para ingresar al sistema bien sea como administrador o como operador del sistema invocamos la siguiente URL que nos muestra la pantalla de Login en la cual ingresamos el usuario y la clave de acceso al sistema.

http://localhost:8080/QManagement/

Figura 0.1: Interfaz de ingreso al sistema

Fuente: Propia

Una vez ingresados el usuario y la clave damos clic en el botón Ingresar luego de lo cual accedemos a la interfaz de administración del sistema en la cual podemos ver todas las opciones habilitadas para el perfil administrador.

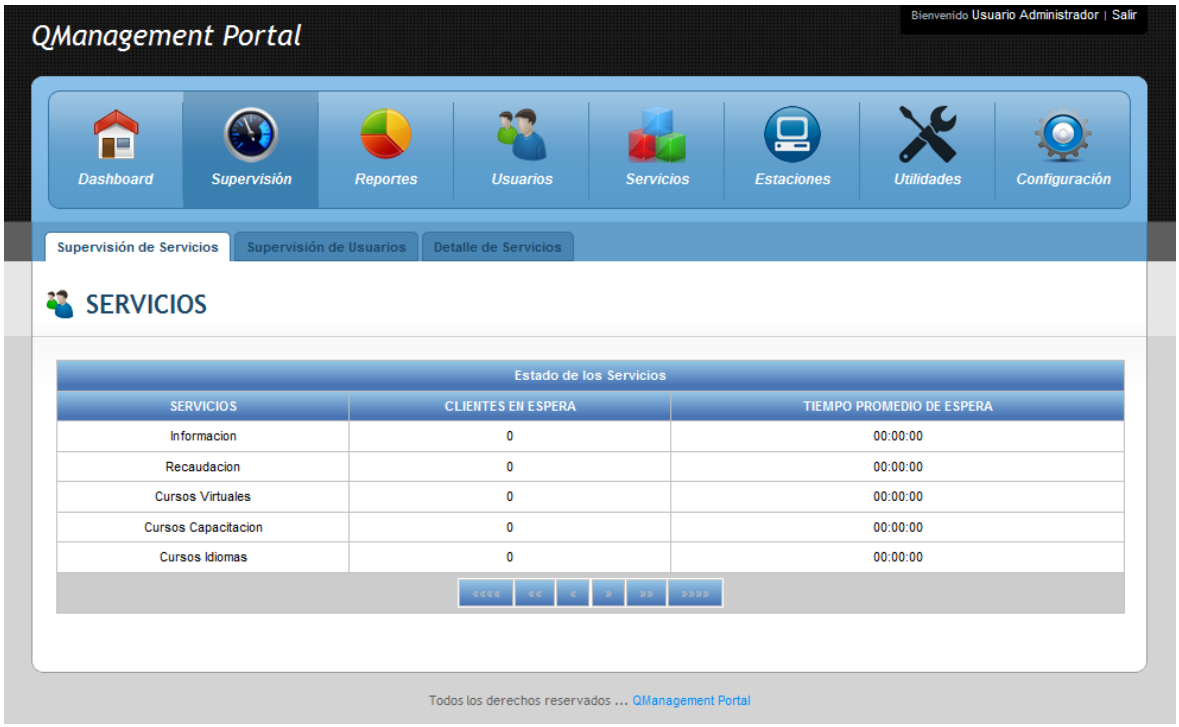

Figura 0.2: Interfaz de administración del sistema

Fuente: Propia

A continuación se describe cada opción del menú de administración y su respectiva configuración.

1.1 Menú Principal

Son todas las opciones disponibles para el usuario administrador desde donde se puede gestionar los usuarios, servicios, estaciones, configurar la conexión del servidor, utilidades del sistema y opciones de supervisión.

Figura 0.3: Opciones usuario administrador

Fuente: Propia

Opción Supervisión

Sub Menú:

Supervisión de Servicios.- permite el monitoreo en línea del estado de los servicios, de los cuales muestra los clientes en espera por cada servicio y el tiempo promedio de espera.

Figura 0.4: Interfaz Opción Supervisión, Sub Menú Supervisión de Servicios

Fuente: Propia

Supervisión de Usuarios.- permite el monitoreo en línea del estado de los usuarios, de los cuales muestra el nombre, el estado de conexióny el tiempo de actividad.

Figura 0.5: Interfaz Opción Supervisión, Sub Menú Supervisión de Usuarios

Fuente: Propia

Detalle de servicios.- permite monitorear en línea el estado de cada turno que está en espera. Muestra el nombre del servicio al que pertenece el turno, el turno, la prioridad asignada al turno, la hora de emisión y el tiempo de espera.

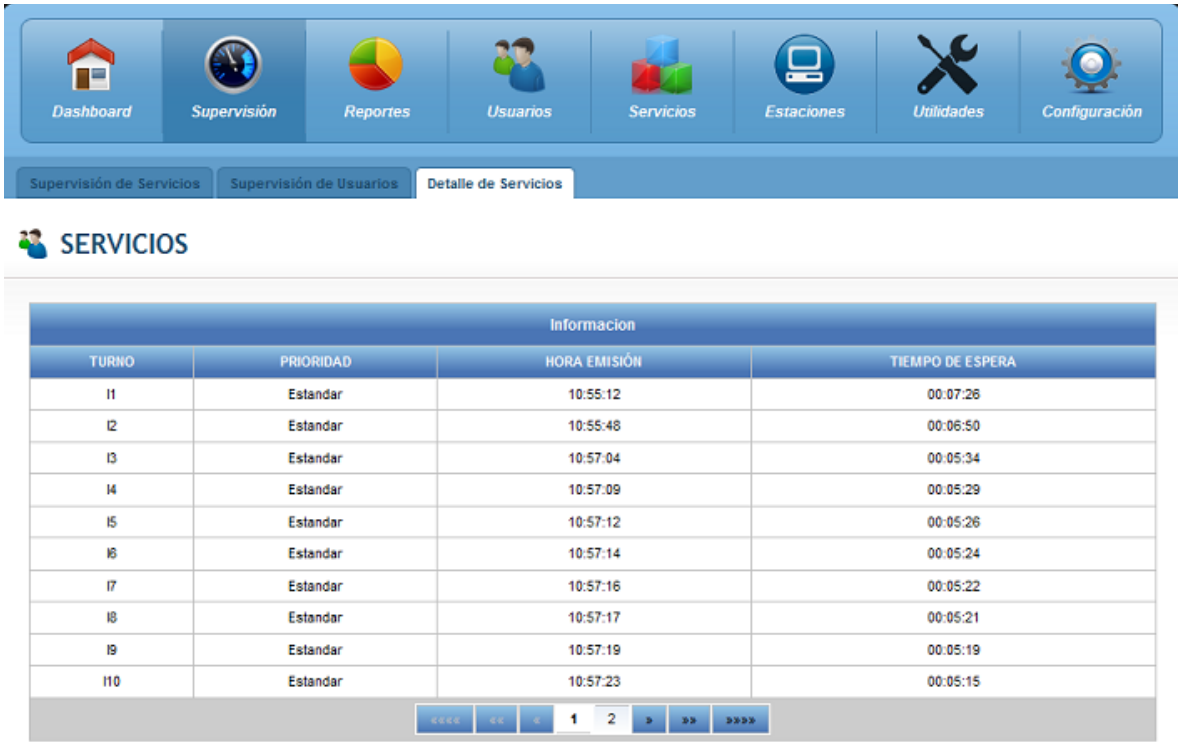

Figura 0.6: Interfaz Opción Supervisión, Sub Menú Detalle de Servicios

Fuente: Propia

Opción Usuarios

Sub Menú:

Gestión de Usuarios.- permite crear, modificar y eliminar un usuario. Las propiedades del usuario son las siguientes:

- Id del usuario en la base de datos.
- Nombre del usuario.
- Clave de ingreso.
- Puesto asignado.
- Estado.
- Tipo de acceso.

Figura 0.7: Interfaz Opción Usuarios, Sub Menú Gestión de Usuarios

Fuente: Propia

Asignación de Servicios.- permite asignar o quitar los servicios que atenderá un usuario en particular.

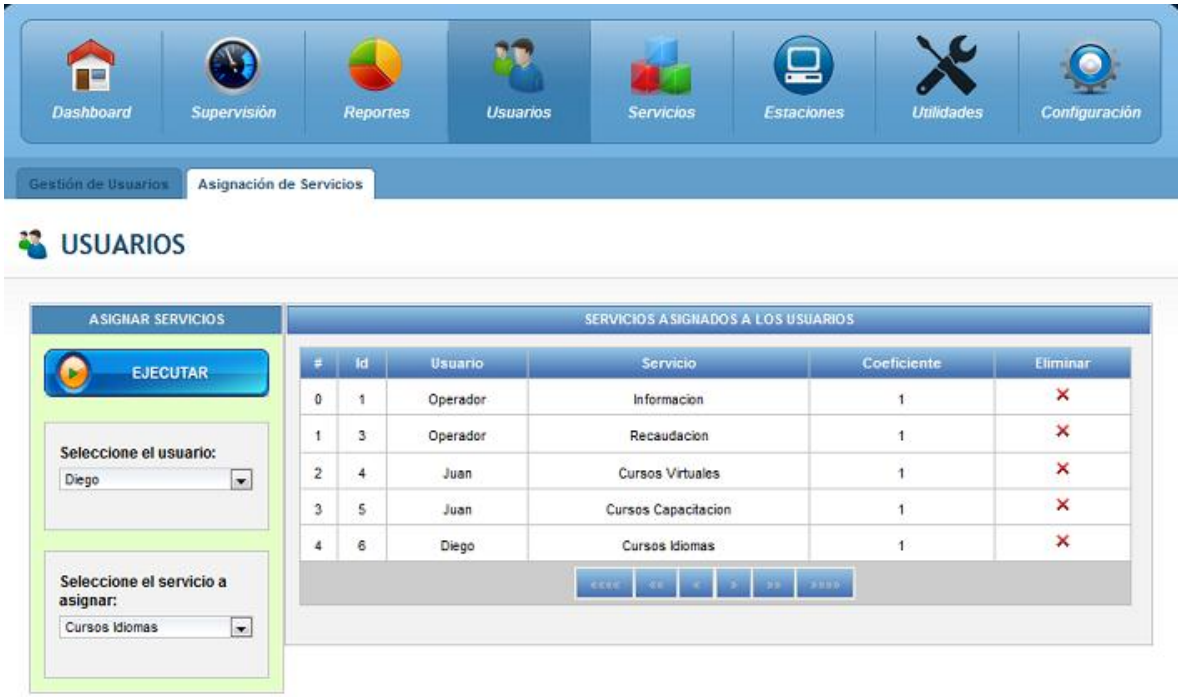

Figura 0.8: Interfaz Opción Usuarios, Sub Menú Asignación de Servicios

Fuente: Propia

Opción Servicios

Sub Menú:

Gestión de Servicios.- permite crear, modificar y eliminar servicios. Las propiedades del servicio que se deben ingresar son las siguientes:

- Id del servicio en la base de datos.
- Nombre del servicio.
- Descripción del texto que se imprimirá en el ticket.
- Prefijo asignado al servicio.

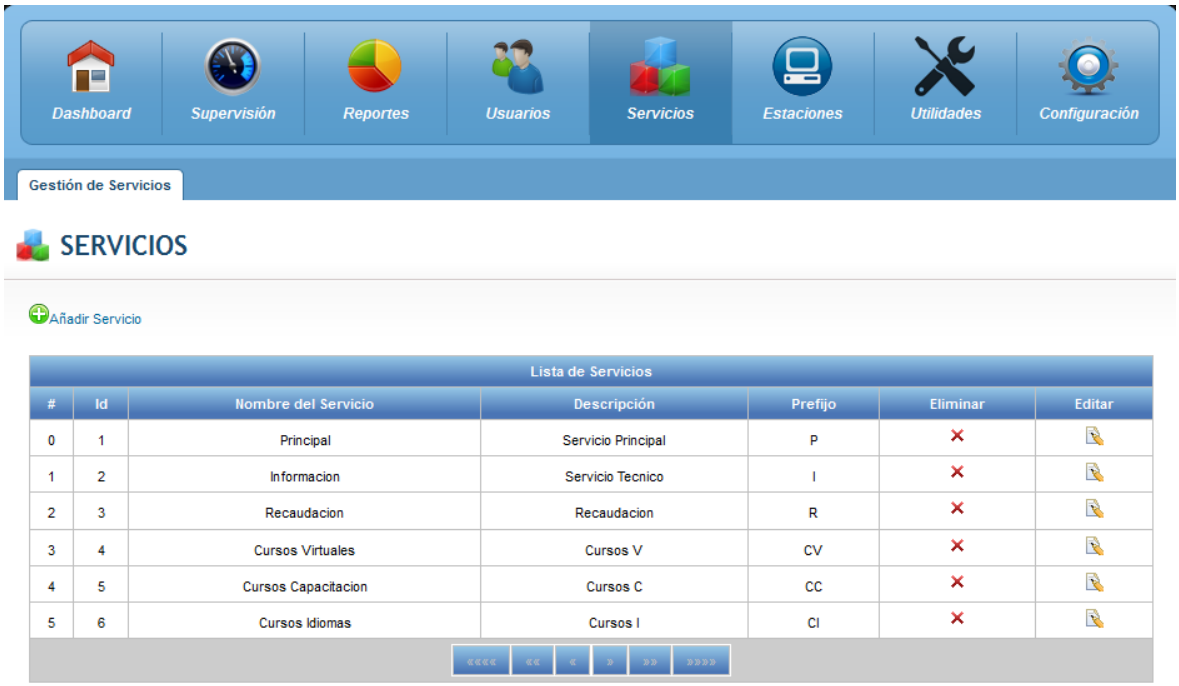

Figura 0.9: Interfaz Opción Servicios, Sub Menú Gestión de Servicios

Fuente: Propia

Opción Reportes

Sub Menú:

Clientes Atendidos.- permite emitir y exportar un reporte de clientes atendidos de acuerdo a los parámetros ingresados en la configuración del reporte.

Los parámetros necesarios para emitir el reporte son los siguientes:

- Fecha de Inicio.
- Fecha Final.
- El servicio del cual se va a emitir el reporte.

- El usuario del que se desea emitir el reporte.

Las propiedades que se mostraran en el reporte generado son las siguientes:

- Fecha en la que se generó el turno.
- Turno atendido.
- Servicio al que pertenece el turno.
- Hora de emisión del turno.
- Hora de inicio de la atención del turno.
- Hora de finalización de la atención del turno.
- Usuario que atendió el turno.

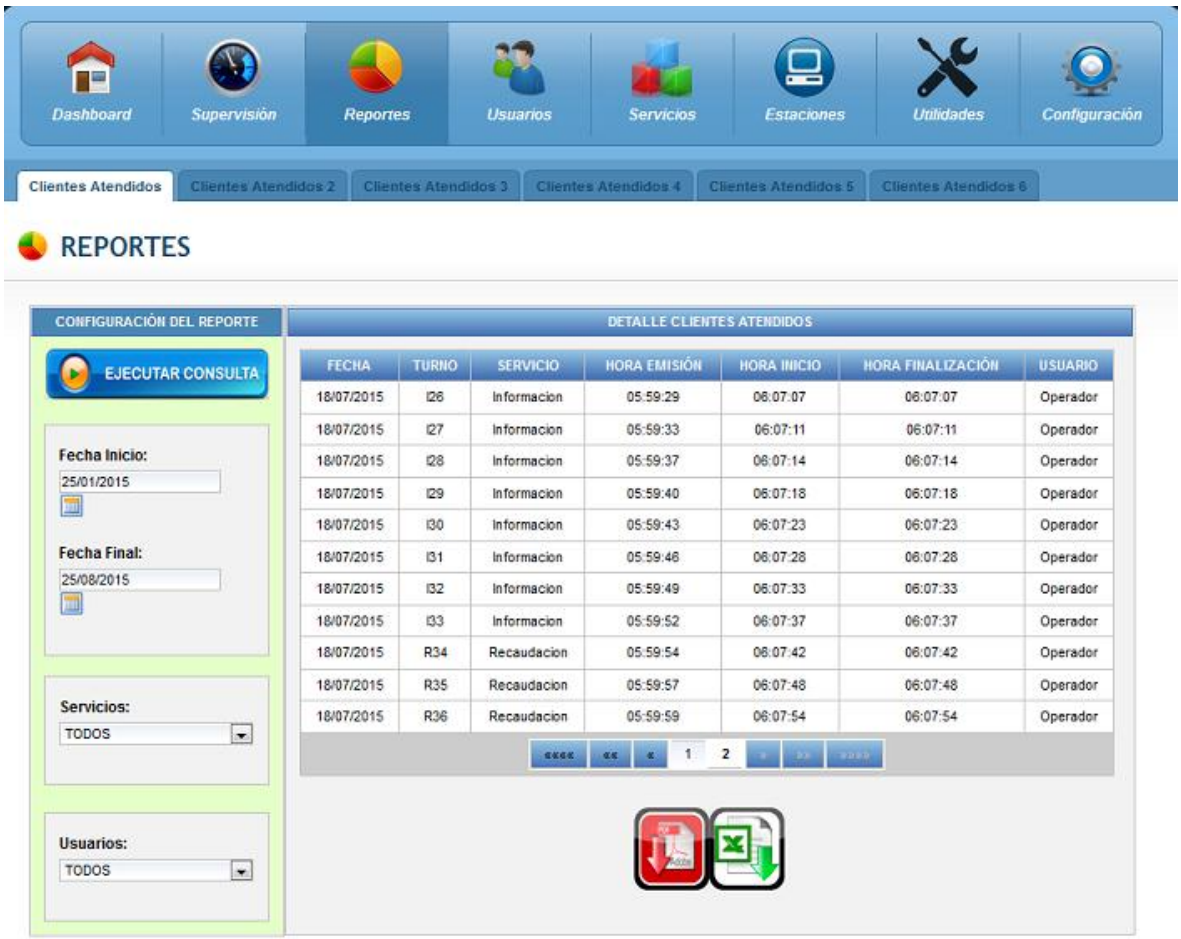

Figura 0.9: Interfaz Opción Reportes, Sub Menú Clientes Atendidos

Fuente: Propia

Opción Utilidades

Sub Menú:

Reiniciar Servidor.- permite reiniciar todos los procesos que se encuentran levantados en el servidor del sistema QManagement por lo cual todos los turnos en espera se eliminan de la cola y los usuarios conectados al sistema se desconentan automáticamente.

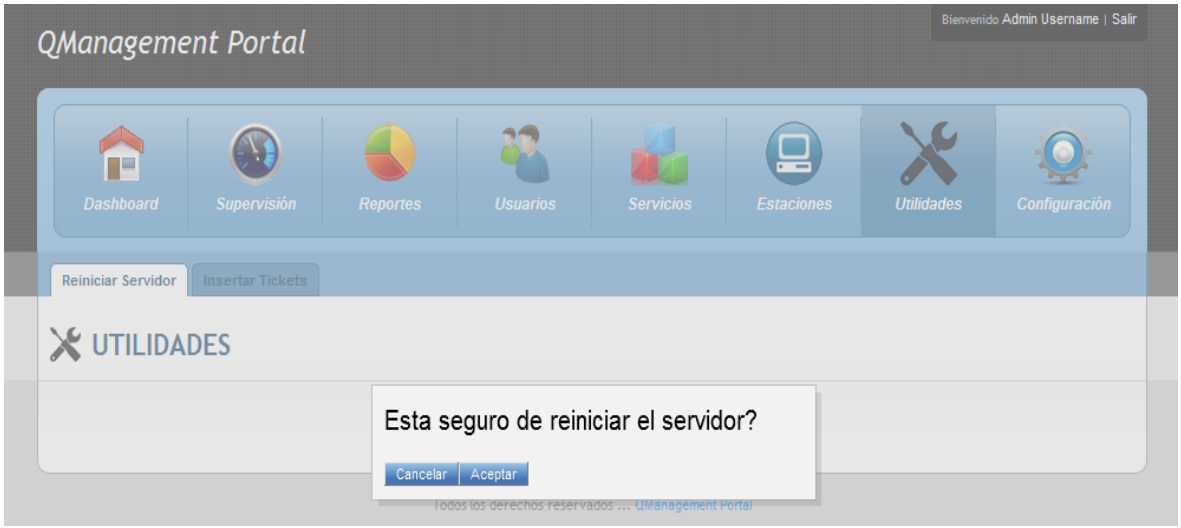

Figura 0.10: Interfaz Opción Utilidades, Sub Menú Reiniciar Servidor

Fuente: Propia

Opción Utilidades

Sub Menú:

Insertar Tickets.- permite crear turnos virtuales en cualesquiera de las colas de espera activas.

Los parámetros necesarios para crear turnos virtuales son los siguientes:

- Servicio en el que se desea insertar los turnos virtuales.
- Cantidad de turnos a insertar.

Para insertar un turno virtual seleccionamos el servicio en el cual se desea insertar los turnos e ingresamos la cantidad de turnos a insertar tal como se observa en la siguiente figura.

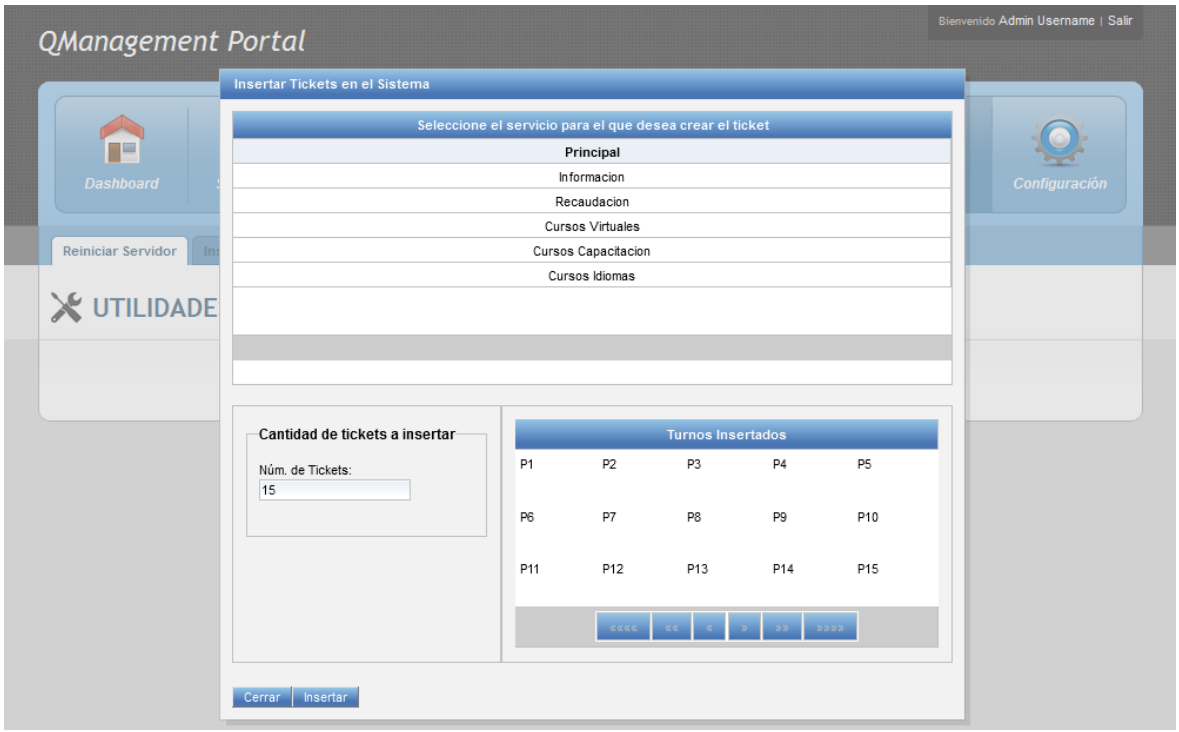

Figura 0.11: Interfaz Opción Utilidades, Sub Menú Insertar Tickets

Fuente: Propia

Opción Configuración

Sub Menú:

Gestión de Red.- permite editar los parámetros de conexión de red del servidor de gestión de colas. Las propiedades de conexión de red son las siguientes:

- Id de agencia asignada al servidor.
- Puerto de comunicación TCP por el cual el servidor escucha las peticiones de las aplicaciones cliente.
- Hora de inicio de la atención de la agencia.
- Hora de finalización de la atención en la agencia.
- Primer número de la secuencia de los turnos en las colas de espera.
- Ultimo número de la secuencia de los turnos en las colas de espera.
- Tipo de numeración permite definir si la numeración será individual por cada servicio o se aplicará una sola secuencia para todos los servicios.
- Limite de rellamadas define la cantidad de veces que un turno puede ser llamado antes de llamar al siguiente turno de la cola.

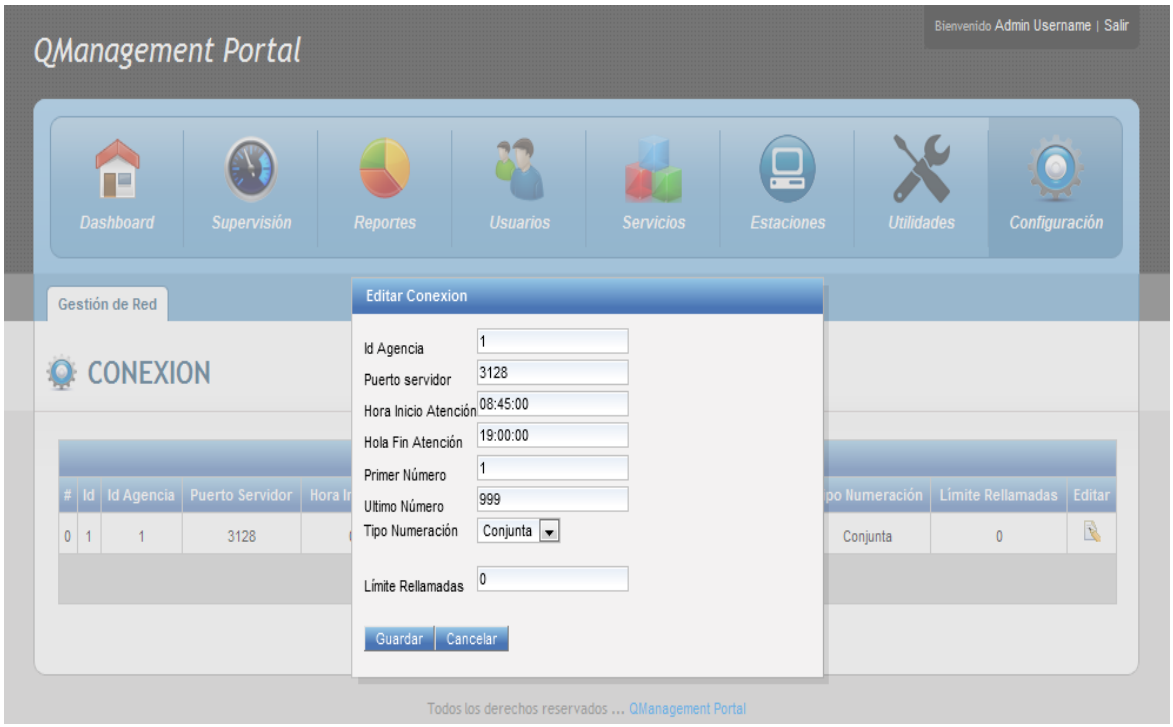

Figura 0.12: Interfaz Opción Configuración, Sub Menú Gestión de Red

Fuente: Propia

2. Ingreso al Portal de Gestión del Flujo de Clientes "QManagement" como operador

Una vez ingresados el usuario y la clave de un usuario operador accedemos a la interfaz del módulo Teclado Virtual en la cual podemos ver todas las opciones habilitadas para el puesto de atención.

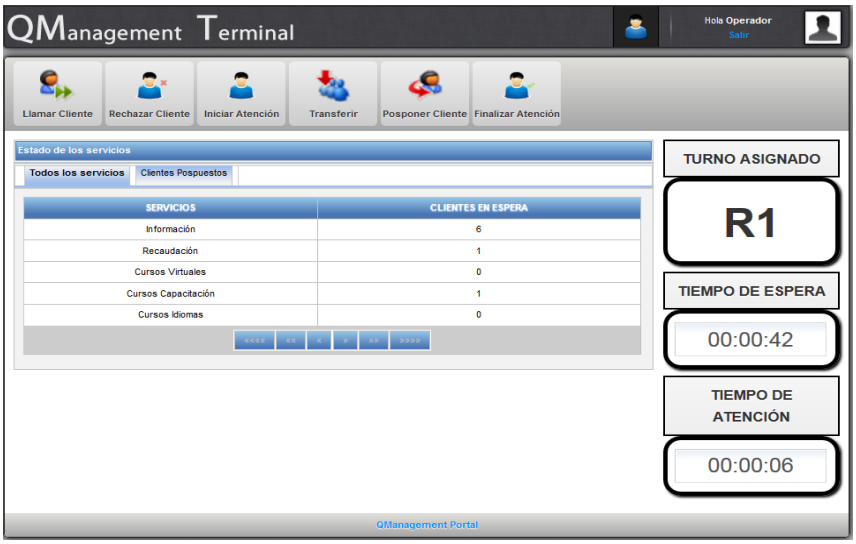

Figura 0.13: Interfaz del módulo Teclado Virtual

Fuente: Propia

Opciones Disponibles

- **Llamar Cliente.-** permite llamar un turno de la cola de espera de acuerdo a la prioridad de atención asignada a un servicio. Una vez llamado un turno muestra las siguientes propiedades del turno:
  - Turno asignado.
  - Tiempo de espera del turno.
  - Tiempo de atención.

Figura 0.14: Propiedades mostradas al llamar un turno
Fuente: Propia

- **Rechazar Cliente.-** permite rechazar el turno que fue llamado por el puesto de atención luego de que el cliente no se presentó para ser atendido.
- **Iniciar Atención.-** permite iniciar el tiempo de atención y finalizar el tiempo de espera de un turno luego de que el cliente se presentó al puesto de atención para ser atendido.
- **Transferir.-** permite transferir a un cliente que está siendo atendido en el puesto de atención hacia otro servicio. Para transferirlo se requiere ingresar el parámetro servicio al cual se transfiere el turno y el parámetro comentario que permite ingresar alguna observación o comentario sobre el turno.

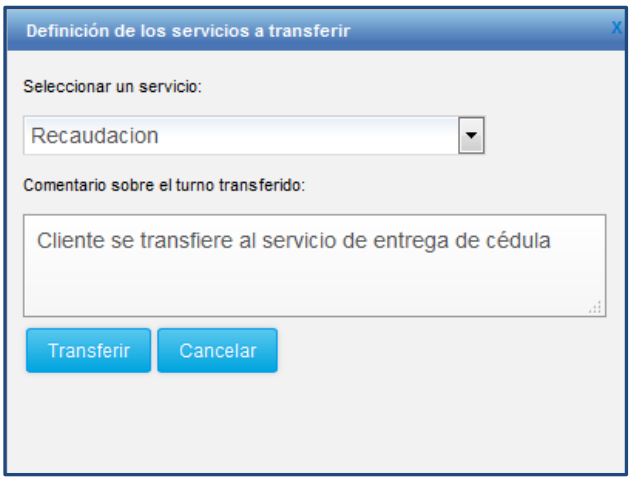

Figura 0.15: Interfaz para transferir un turno

Fuente: Propia

- **Posponer Cliente.-** permite posponer un turno durante un tiempo definido luego del cual se continuará con la atención del mismo. Para posponerlo se requiere ingresar el parámetro motivo por el cual se pospone el turno y el parámetro periodo de estancia que corresponde al tiempo que el turno permanecerá en la lista de espera antes de ser llamado nuevamente.

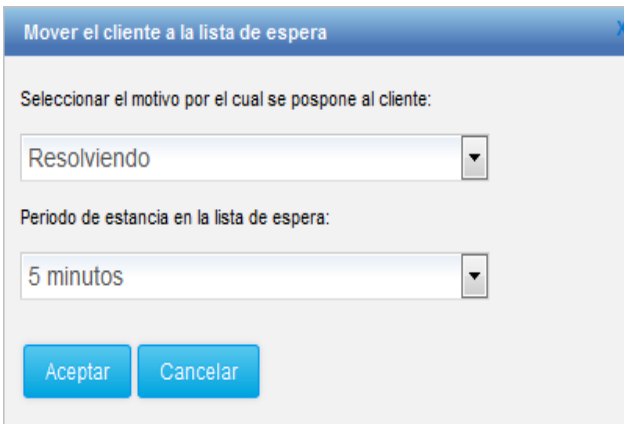

Figura 0.16: Interfaz para posponer un turno

Fuente: Propia

- **Finalizar Atención.-** permite finalizar el tiempo de atención del turno luego de que terminó la atención al cliente en el puesto de atención.
- **Estado de los servicios.-** permite mostrar la cantidad de clientes en espera en cada servicio así como la cantidad de clientes pospuestos.

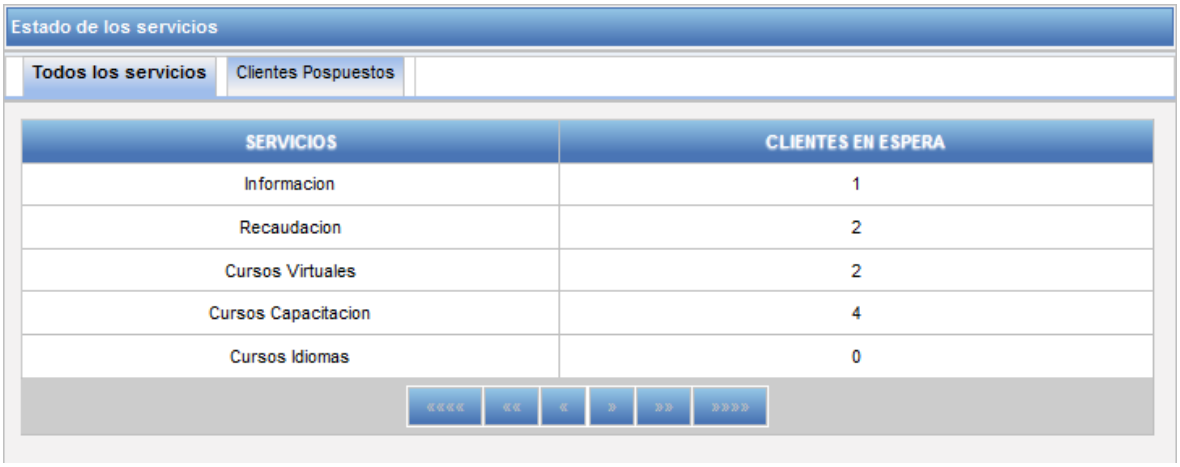

Figura 0.17: Interfaz de estado de los servicios, módulo Teclado Virtual

Fuente: Propia

3. Ingreso al módulo de Emisión de turnos

Para ingresar al módulo de emisión de turnos no es necesario loguearse en el sistema, solo debemos invocar el link correspondiente y obtenemos la interfaz de emisión de turnos.

http://localhost:8084/QManagement/imprimirticket.xhtml

Figura 0.18: Interfaz de usuario, módulo de emisión de turnos

Fuente: Propia

En la pantalla de emisión de turnos se debe elegir el servicio deseado y la aplicación muestra un POPUP con el turno asignado al cliente.

Figura 0.19: POPUP que muestra el turno asignado

Fuente: Propia

4. Ingreso al módulo de Presentación de turnos Display

Para ingresar al módulo de presentación de turnos en la pantalla LCD, no es necesario loguearse en el sistema, solo debemos invocar el link correspondiente y obtenemos la interfaz de emisión de turnos.

http://localhost:8084/QManagement/display.xhtml

Figura 0.20: Interfaz de usuario, módulo de presentación de turnos

Fuente: Propia

En la pantalla de presentación de turnos la aplicación muestra los cinco últimos turnos que el servidor de gestión de colas asignó a los puestos de atención.

Cuando un puesto de atención llama un turno, la aplicación display muestra un POPUP con el turno y el módulo al que debe dirigirse el cliente para ser atendido.

Figura 0.21: POPUP que muestra último turno llamado

Fuente: Propia

ANEXO B: MANUAL TECNICO DEL SISTEMA

B.1 Objetivo

El objetivo del manual técnico es dar a conocer los aspectos técnicos del sistema QManagement, de tal manera que los programadores que lean el documento puedan entenderlo con facilidad y puedan realizar actualizaciones o mantenimiento del mismo en caso de que lo requieran.

B.1.1 Diccionario de Datos

B.1.1.1 Nombre de la Tabla: ADVANCE

Descripción: Almacena información de pre registro de clientes.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del cliente pre registrado	PK
service_id	bigint	Identifica el servicio	FK
advance_time	datetime2(0)	Hola de pre registro	
priority	int	Prioridad del cliente pre registrado	
clients_authorization_id	bigint	Determinar si el cliente está registrado	FK
input_data	nvarchar(150)	Ingresar los datos del cliente	
comments	nvarchar(345)	Ingresar algún comentario por parte del usuario operador	

Tabla B.1.1.1: Descripción de los campos de la tabla Advance
Fuente: Propia

B.1.1.2 Nombre de la Tabla: BREAK

Descripción: Almacena información sobre los recesos de los usuarios.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del receso	PK
breaks_id	bigint	Registra el id del receso que contiene el nombre	FK
from_time	time(7)	Hora de inicio del receso	
to_time	time(7)	Hora final del receso	

Tabla B.1.1.2: Descripción de los campos de la tabla Break
Fuente: Propia

B.1.1.3 Nombre de la Tabla: BREAKS

Descripción: Almacena la información que describe los recesos de los usuarios.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del nombre del receso	PK
name	nvarchar(245)	Nombre del receso	

Tabla B.1.1.3: Descripción de los campos de la tabla Breaks
Fuente: Propia

B.1.1.4 Nombre de la Tabla: CALENDAR

Descripción: Almacena la información de los nombres asignados al calendario de trabajo.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del nombre del calendario	PK
name	nvarchar(45)	Nombre del calendario	

Tabla B.1.1.4: Descripción de los campos de la tabla Calendar
Fuente: Propia

B.1.1.5 Nombre de la Tabla: CALENDAR_OUT_DAYS

Descripción: Almacena la información de las fechas del calendario en las cuales no trabaja el sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del	PK

		calendario	
out_day	date	Fecha en la que no se trabaja	
calendar_id	bigint	Identificador del nombre del calendario	FK

Tabla B.1.1.5: Descripción de los campos de la tabla Calendar_Out_Days
Fuente: Propia

B.1.1.6 Nombre de la Tabla: CLIENTS

Descripción: Almacena la información detallada de los turnos atendidos por el sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del registro de atención del cliente	PK
service_id	bigint	Identifica el id del servicio para el que se atendió al cliente	FK
user_id	bigint	Identifica el id del usuario que atendió al cliente	FK
service_prefix	nvarchar(45)	Prefijo o letra del turno atendido	
number	int	Número del turno atendido	
stand_time	datetime2(0)	Tiempo de emisión del turno	
start_time	datetime2(0)	Tiempo de inicio de atención del turno	
finish_time	datetime2(0)	Tiempo de finalización de atención del turno	
clients_authorization_id	bigint	Identifica el id del cliente atendido	FK
result_id	bigint	Identifica la salida generada	FK
input_data	nvarchar(150)	Muestra los datos ingresados por el usuario	
state_in	int	Estado del cliente	

Tabla B.1.1.6: Descripción de los campos de la tabla Clients
Fuente: Propia

B.1.1.7 Nombre de la Tabla: NET

Descripción: Almacena la información sobre los parámetros de conexión del servidor del sistema con los módulos clientes.

Columnas	Tipo de Datos	Descripción	Referencia
id	int	Identificador del registro de información de red	PK
server_port	int	Puerto del servidor	
web_server_port	int	Puerto del servidor web	
client_port	int	Puerto de comunicación del cliente	
finish_time	time	Hora en la que se detiene el servidor	
start_time	time	Hora en la que inicia el funcionamiento del servidor	
version	nvarchar(25)	Versión de la BDD	
first_number	int	Primer número de la secuencia del turno	
last_number	int	Ultimo número de la secuencia del turno	
numering	smallint	Se asigna 0 si todos los servicios van a tener la misma numeración o 1 si cada servicio tiene su propia numeración	
point	int	Se define la descripción del puesto de atención como se va a mostrar	
sound	int	Se define el tipo de configuración para la voz, 0 para no reproducir ningún audio, 1 para mostrar solo las señales y 2 para mostrar señales más la voz	
branch_id	bigint	Identifica el id de la agencia	
sky_server_url	nvarchar(145)	Url del servidor	
zone_board_serv_addr	nvarchar(145)	Dirección del panel del panel del servidor donde se reciben los datos	

zone_board_serv_port	bigint	Puerto del panel del servidor donde se reciben los datos	
voice	int	Reproducir la Voz al mostrar turno	
black_time	int	Tiempo transcurrido de los turnos en la lista negra	
limit_recall	int	Límite de rellamadas	
button_free_design	smallint	Libre diseño de los botones en el punto de entrada	

Tabla B.1.1.7: Descripción de los campos de la tabla Net
Fuente: Propia

B.1.1.8 Nombre de la Tabla: REPORTS

Descripción: Almacena la información de los reportes que emite el sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del reporte	PK
name	nvarchar(255)	Nombre del reporte	
className	nvarchar(150)	Clase que permite generar el reporte	
template	nvarchar(150)	Plantilla del reporte	
href	nvarchar(150)	Url del reporte	FK

Tabla B.1.1.8: Descripción de los campos de la tabla Reports
Fuente: Propia

B.1.1.9 Nombre de la Tabla: RESPONSE

Descripción: Almacena la información sobre los tipos de respuesta que puede elegir un cliente en la consulta de satisfacción de atención.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador de la respuesta	PK
name	nvarchar(100)	Descripción de la respuesta	
text	nvarchar(500)	Texto de la respuesta	

Tabla B.1.1.9: Descripción de los campos de la tabla Response
Fuente: Propia

B.1.1.10 Nombre de la Tabla: RESPONSE_EVENT

Descripción: Almacena la información receptada en la consulta de satisfacción del cliente.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador de la opinión	PK
resp_date	datetime2(0)	Fecha de la opinión	
response_id	bigint	Identificador del id de la opinión	FK
services_id	bigint	Identificador del servicio que recibió la opinión	FK
users_id	bigint	Identificador del usuario que recibió la opinión	FK
clients_id	bigint	Identificador del id del cliente que emitió la opinión	FK
client_data	nvarchar(245)	Comentario emitido por el cliente	

Tabla B.1.1.10: Descripción de los campos de la tabla Response_Event
Fuente: Propia

B.1.1.11 Nombre de la Tabla: RESULTS

Descripción: Almacena la información sobre el estado de procesamiento de tareas que ejecuta el sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del resultado	PK
name	nvarchar(150)	Descripción del estado	

Tabla B.1.1.11: Descripción de los campos de la tabla Results
Fuente: Propia

B.1.1.12 Nombre de la Tabla: SCHEDULE

Descripción: Almacena la información sobre el horario de trabajo en el que se encuentra habilidado el sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del resultado	PK
name	nvarchar(150)	Descripción del estado	
type	int	Tipo de planificación, puede ser semanal o	

		por días	
time_begin_1	time(7)	Hora de inicio	
time_end_1	time(7)	Hora de finalización	
time_begin_2	time(7)	Hora de inicio	
time_end_2	time(7)	Hora de finalización	
time_begin_3	time(7)	Hora de inicio	
time_end_3	time(7)	Hora de finalización	
time_begin_4	time(7)	Hora de inicio	
time_end_4	time(7)	Hora de finalización	
time_begin_5	time(7)	Hora de inicio	
time_end_5	time(7)	Hora de finalización	
time_begin_6	time(7)	Hora de inicio	
time_end_6	time(7)	Hora de finalización	
time_begin_7	time(7)	Hora de inicio	
time_end_7	time(7)	Hora de finalización	
breaks_id1	bigint	Identifica la hora de receso	FK
breaks_id2	bigint	Identifica la hora de receso	FK
breaks_id3	bigint	Identifica la hora de receso	FK
breaks_id4	bigint	Identifica la hora de receso	FK
breaks_id5	bigint	Identifica la hora de receso	FK
breaks_id6	bigint	Identifica la hora de receso	FK
breaks_id7	bigint	Identifica la hora de receso	FK

Tabla B.1.1.12: Descripción de los campos de la tabla Schedule
Fuente: Propia

B.1.1.13 Nombre de la Tabla: SERVICES

Descripción: Almacena la información de los servicios disponibles en la agencia.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del servicio	PK

name	nvarchar(2000)	Nombre del servicio	
description	nvarchar(2000)	Descripción del servicio	
service_prefix	nvarchar(10)	Prefijo o letra del servicio	
button_text	nvarchar(2500)	El texto que se imprimirá al elegir el servicio	
status	int	Estado del servicio, 1 disponible, 0 no disponible	
enable	int	Si está habilitado o no	
prent_id	bigint	Grupo al que pertenece	
day_limit	int	Restricción de turnos emitidos por día. Si no hay restricción su valor es 0	
person_day_limit	int	Restricción de turnos emitidos por día para un cliente. Si no hay restricción su valor es 0	
advance_limit	int	Limitar el número de clientes pre registrados por hora	
advance_limit_period	int	Limite en días para el pre registro por anticipado. Si no hay restricción su valor es 0	
advance_time_period	int	Periodos en los que se ha divide un día para el pre registro	
schedule_id	bigint	Identifica el horario de trabajo del servicio	FK
input_required	smallint	Ingreso obligado de información del cliente antes de entrar a la cola	
input_caption	nvarchar(2000)	Ingresar el número de documento al	

		momento de ingresar la información del cliente	
result_required	smallint	El usuario debe ingresar el estado del proceso al atender al cliente	
calendar_id	bigint	Identifica el id del calendario	FK
pre_info_html	nvarchar(MAX)	Texto de información que se muestra en la pantalla	
pre_info_print_text	nvarchar(MAX)	Texto que se imprimirá con el servicio antes del encolamiento	
point	int	Punto de registro del cliente	
ticket_text	nvarchar(1500)	Texto que se imprimirá en el turno	
seq_id	int	Orden de los botones en el punto de registro	
but_x	int	Posición x del botón	
but_y	int	Posición y del botón	
but_b	int	Ancho del botón	
but_h	int	Alto del botón	
deleted	date	Confirmar la fecha de eliminación del servicio	
duration	int	Tiempo promedio para atender el servicio	
sound_template	nvarchar(45)	Plantilla para emitir un sonido al llamar el servicio	
expectation	int	Tiempo de espera obligatorio	

Tabla B.1.1.13: Descripción de los campos de la tabla Services
Fuente: Propia

B.1.1.14 Nombre de la Tabla: SERVICES_USERS

Descripción: Almacena la información de los servicios asignados a los usuarios del sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del servicio de usuario	PK
services_id	bigint	Identifica el id del servicio	FK
user_id	bigint	Identifica el id del usuario	FK
coefficient	int	Porcentaje de atención	
flexible_coef	smallint	Es posible cambiar el coeficiente de atención	

Tabla B.1.1.14: Descripción de los campos de la tabla Services_Users
Fuente: Propia

B.1.1.15 Nombre de la Tabla: STANDARS

Descripción: Almacena la información de las reglas de negocio establecidas para la atención de los clientes por parte de los asesores de la agencia.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador de la regla de negocio	PK
wait_max	int	Tiempo de espera máximo	
work_max	int	Tiempo de atención máximo	
downtime_max	int	Tiempo máximo de inactividad en la cola	
line_service_max	int	Longitud máxima de la cola para un servicio	
line_total_max	int	El número máximo de clientes para todos los servicios	
relocation	int	Tiempo de traslado de un servicio	

Tabla B.1.1.15: Descripción de los campos de la tabla Standars
Fuente: Propia

B.1.1.16 Nombre de la Tabla: STATISTICS

Descripción: Almacena la información sobre las estadísticas que genera el sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador de la estadística	PK
user_id	bigint	Identifica el id del usuario	FK
client_id	bigint	Identifica el id del cliente	FK
service_id	bigint	Identifica el id del servicio	FK
results_id	bigint	Identifica el estado de la tarea	FK
user_start_time	datetime2(0)	Tiempo de inicio de la atención del usuario con los clientes	
user_finish_time	datetime2(0)	Tiempo de finalización de la atención del usuario con los clientes	
client_stand_time	datetime2(0)	Tiempo en reposo del cliente	
user_work_period	int	Tiempo de trabajo del usuario con el cliente	
client_wait_period	int	Tiempo de espera del cliente	
state_in	int	Estado del cliente	

Tabla B.1.1.16: Descripción de los campos de la tabla Statistics
Fuente: Propia

B.1.1.17 Nombre de la Tabla: USERS

Descripción: Almacena toda la información de los usuarios del sistema.

Columnas	Tipo de Datos	Descripción	Referencia
id	bigint	Identificador del usuario	PK
name	nvarchar(150)	Nombre del usuario	
password	nvarchar(45)	Clave del usuario	

point	nvarchar(45)	Lugar de trabajo	
adress_rs	smallint	Dirección	
enable	int	Estado del usuario	
admin_access	smallint	Si tiene permiso para ingresar al perfil de administrador del sistema	
report_access	smallint	Si tiene permiso para ingresar al perfil de reportería del sistema	
point_ext	nvarchar(1045)	Mostrar puesto en el panel principal de llamadas	
deleted	date	Fecha de confirmación de eliminación	

Tabla B.1.1.17: Descripción de los campos de la tabla Users
Fuente: Propia

B.1.2 Prototipo de Interfaz de Usuario

Una interfaz de usuario debe ser sencilla y de fácil manejo para los usuarios de la aplicación. Debe permitir la comunicación entre el usuario y el computador de forma rápida, intuitiva y transparente.

B.1.2.1 Diseño de la Plantilla Principal

La plantilla principal del Sistema de Gestión del Flujo de Clientes, está estructurada de la siguiente manera:

- **Barra de Títulos:** muestra el nombre del aplicativo y el nombre del usuario logueado en el sistema.
- **Barra de Menú de Opciones:** presenta diferentes opciones que muestran pestañas que permiten ejecutar las diferentes funcionalidades del sistema.
- **Seccion de Contenidos:** abarca la zona principal del sistema, se encuentra ubicada en la parte central de la pantalla.
- **Pie de Página:** contiene un enlace a la página de inicio de la aplicación y

se encuentra ubicada en la parte inferior de la pantalla.

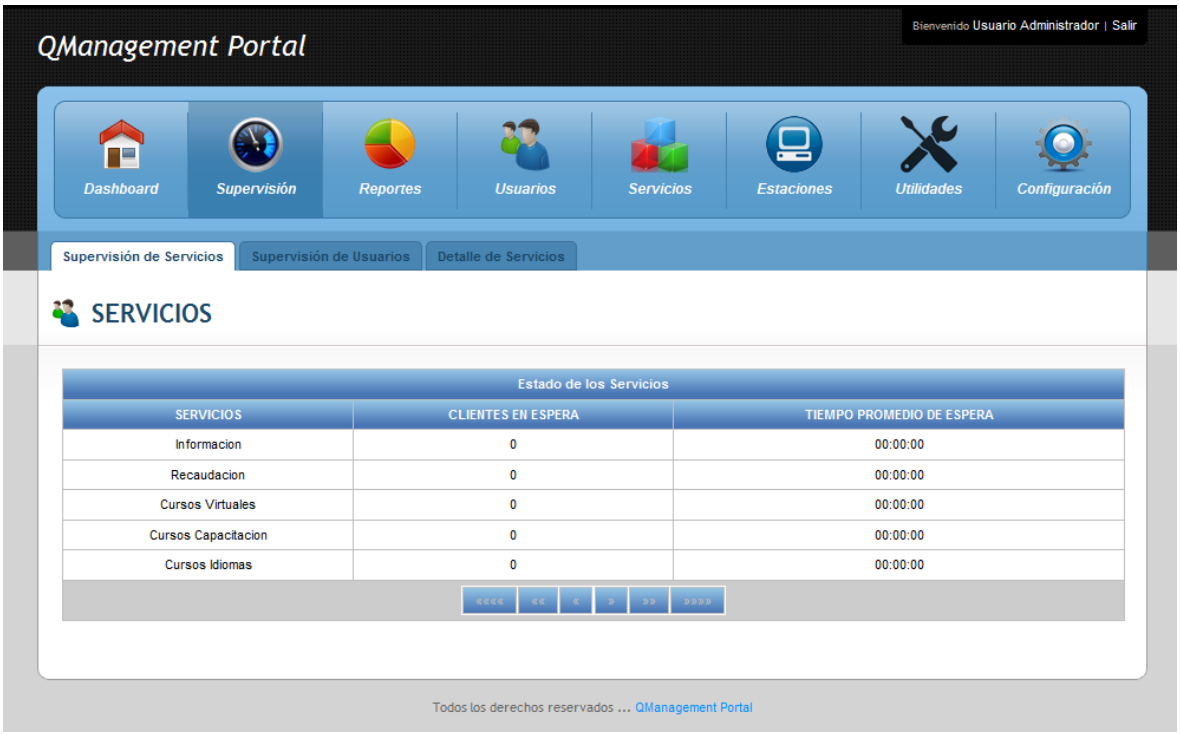

Figura B.1.2.1: Plantilla Principal
Fuente: Propia

B.1.3 Tecnologías utilizadas en el desarrollo del sistema

Todas las tecnologías utilizadas en el desarrollo del sistema son de libre utilización ya que están regidas bajo los términos de la licencia GPL.
Las tecnologías utilizadas son las siguientes:

- Entorno de Desarrollo Integrado NetBeans.
- Servidor de Aplicaciones Apache Tomcat.
- Plataforma de Desarrollo J2EE.
- Framework Hibernate para la persistencia de datos.
- Framework Spring para la integración de los componentes de la aplicación.
- Framework JavaServer Faces para el diseño de la interfaz de usuario.
- Framework RichFaces para la integración de funcionalidades AJAX en JSF.

- Estándares de código abierto HTML, JavaScript, CSS, Ajax.

- Servidor de Base de Datos SQL Server Express 2008.

B.1.4 Configuración del Proyecto

El proyecto utiliza las tecnologías mencionadas anteriormente y se deben configurar de la siguiente manera:

B.1.4.1 Librerias

- Spring framework 4.0.1 que se debe activar directamente desde las propiedades del proyecto.

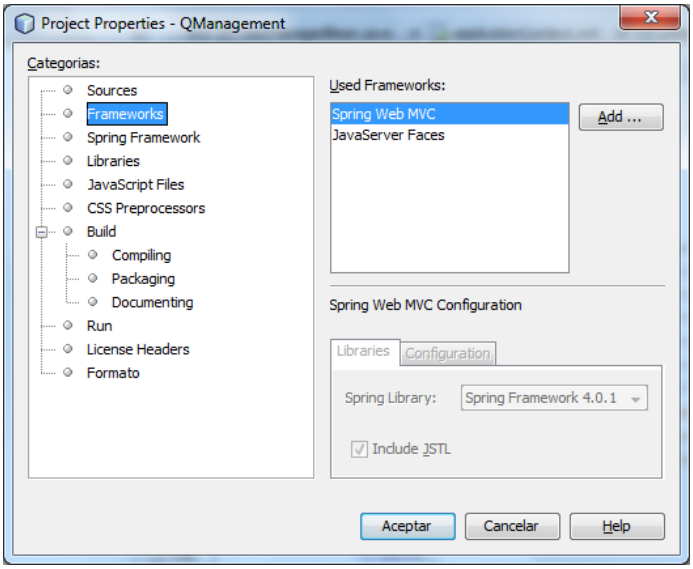

Figura B.1.4.1: Pantalla para agregar el framework Spring
Fuente: Propia

- Framework JavaServer faces 2.2 que se debe activar directamente desde las propiedades del proyecto.

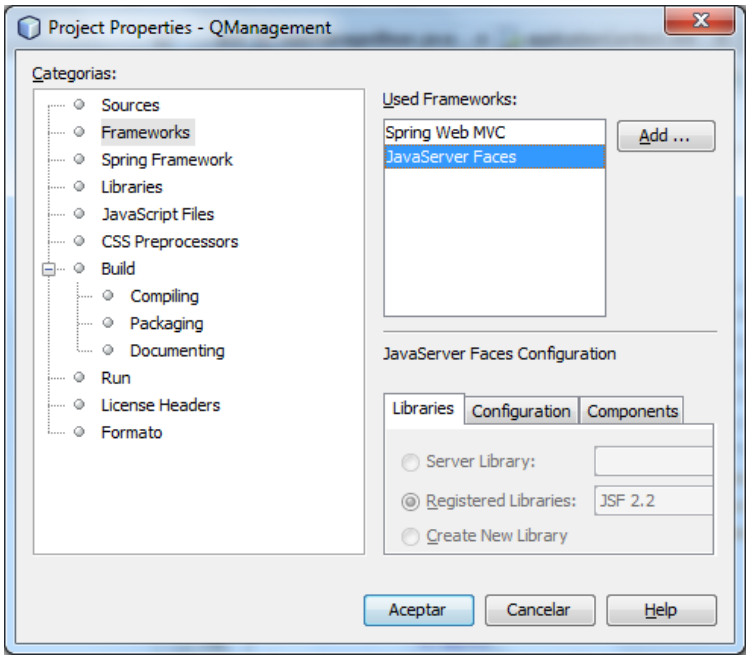

Figura B.1.4.2: Pantalla para agregar el framework JavaServer Faces
Fuente: Propia

- Framework RichFaces 4.2.0, se deben agregar las librerías desde el botón **Add JAR/Folder** y se deben agregar al proyecto.

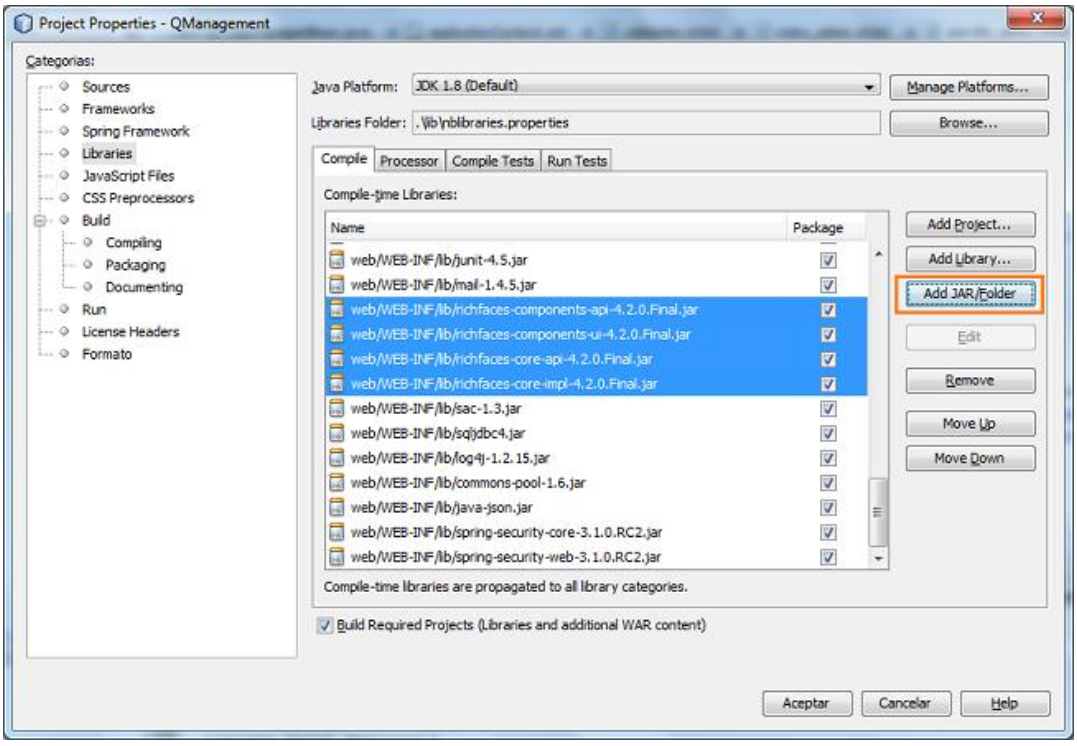

Figura B.1.4.3: Pantalla para agregar el framework RichFaces
Fuente: Propia

- Framework Hibernate 4.3.1, se debe activar directamente desde las propiedades del proyecto.

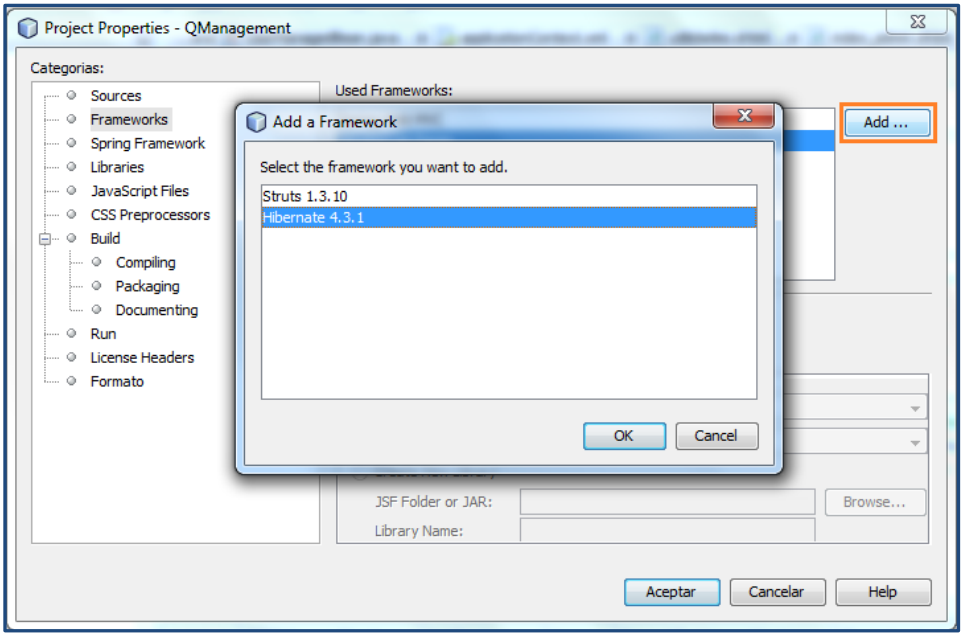

Figura B.1.4.4: Pantalla para agregar el framework Hibernate
Fuente: Propia

B.1.4.2 Archivos de Configuración

- **applicationContext.xml** de Spring en el cual se debe configurar todos los componentes necesarios para la integración de las capas del modelo MVC.

Como se observa en el siguiente fragmento del archivo **applicationContext** QNetDAO se conecta con SessionFactory mediante Spring.

```
<bean id="QNetDAO"
class="ec.edu.utn.fica.eisic.qmanagement.servidor.dao.QNetDAO">
<property name="sessionFactory" ref="SessionFactory" />
</bean>
```

Para configurar la fábrica de sesiones de hibernate se conecta el bean SessionFactory con la propiedad c3p0DataSource y se anotan las clases para hibernate.

```
<bean id="SessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean"
lazy-init="false" autowire="default">
 <property name="dataSource" ref="c3p0DataSource"/>
 <property name="hibernateProperties">
    <props merge="default">
<prop
key="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</prop>
<prop key="hibernate.show_sql">true</prop>
<prop key="hibernate.format_sql">true</prop>
</props>
</property>
<property name="annotatedClasses">
<list>
<value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet</value>
</list>
</property>
</bean>
```

Se crea el bean c3p0DataSource para asignar los parámetros para el acceso a la fuente de datos.

```
<bean id="c3p0DataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
       <property name="driverClass" value="#{conf.driver}"/>
       <property name="jdbcUrl" value="#{conf.url}"/>
       <property name="user" value="#{conf.user}"/>
       <property name="password" value="#{conf.password}"/>
       <property name="checkoutTimeout" value="10000"/>
       <property name="idleConnectionTestPeriod">
          <value>3600</value>
       </property>
</bean>
```

Se crea el bean conf que permite obtener los valores de los parámetros de conexión a la base de datos desde el archivo configbdd.dat

```
<bean id="conf"
class="ec.edu.utn.fica.eisic.qmanagement.hibernate.AnnotationSessionFact
oryBean"/>
```

Se crea el bean transactionManager para administrar todas las transacciones que realiza hibernate con la base de datos.

```
<bean id="transactionManager"
class="org.springframework.orm.hibernate4.HibernateTransactionManager">
<property name="sessionFactory" ref="SessionFactory" />
</bean>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xs
       http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
       http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

    <!-- Declaración de Beans -->
    <bean id="conf" class="ec.edu.utn.fica.eisic.qmanagement.hibernate.AnnotationSessionFactoryBean"/>
    <bean id="QNet" class="ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet"/>

        <!-- Declaración de los beans de servicio -->
        <bean id="QNetService" class="ec.edu.utn.fica.eisic.qmanagement.servidor.servicio.QNetService">
                <property name="qnetDAO" ref="QNetDAO" />
        </bean>
        <!-- Declaración de los beans DAO -->
        <bean id="QNetDAO" class="ec.edu.utn.fica.eisic.qmanagement.servidor.dao.QNetDAO">
                <property name="sessionFactory" ref="SessionFactory" />
        </bean>
    <!-- Declaración de los beans para el acceso a los datos -->
    <bean id="c3p0DataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
        <property name="driverClass" value="#{conf.driver}"/>
        <property name="jdbcUrl" value="#{conf.url}"/>
        <property name="user" value="#{conf.user}"/>
        <property name="password" value="#{conf.password}"/>
        <property name="checkoutTimeout" value="10000"/>
        <property name="idleConnectionTestPeriod">
            <value>3600</value>
        </property>
    </bean>
    <!-- Fabrica de produccion de sesiones con la BDD. Listado de clases anotadas para hibernate. -->
    <bean id="SessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" lazy-init="false" autowire="default">
        <property name="dataSource" ref="c3p0DataSource"/>
        <property name="hibernateProperties">
            <props merge="default">
                <prop key="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.format_sql">true</prop>
            </props>
        </property>

        <property name="annotatedClasses">
            <list>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QNet</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QStandards</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QService</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QServiceLang</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QUser</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.comun.modelo.QCustomer</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QPlanService</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QAdvanceCustomer</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.QAuthorizationCustomer</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.infosistema.QInfoItem</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.respuesta.QRespItem</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.respuesta.QRespEvent</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.horario.QSchedule</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.horario.QBreaks</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.horario.QBreak</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.resultado.QResult</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.calendario.QCalendar</value>
                <value>ec.edu.utn.fica.eisic.qmanagement.servidor.modelo.calendario.FreeDay</value>
            </list>
        </property>
    </bean>
    <!-- Habilitar la configuración del funcionamiento transaccional basado en anotaciones-->
    <tx:annotation-driven transaction-manager="transactionManager
    <bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="SessionFactory" />
    </bean>
</beans>
```

Figura B.1.4.5: Contenido del archivo **applicationContext.xml**
Fuente: Propia

- **web.xml** que describe diversas características del archivo WAR.

El elemento <servlet> define las características de un Servlet y a su vez está compuesto por los elementos <servlet-name> y <servlet-class> que indican un nombre corto para el Servlet así como el nombre de la Clase Java que contiene el Servlet respectivamente. En este caso se indica que la Clase llamada javax.faces.webapp.FacesServlet será denominada con el nombre Faces Servlet que habilita el motor de JSF en el sistema.

```
<servlet>
	<servlet-name>Faces Servlet</servlet-name>
	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
	<load-on-startup>1</load-on-startup>
</servlet>
<servlet>
	<servlet-name>dispatcher</servlet-name>
	<servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
	<load-on-startup>2</load-on-startup>
</servlet>
```

El elemento <servlet-mapping> define la ubicación en términos de directorios de un sitio (URL), esto es, el elemento <servlet-name>Faces Servlet</servlet-name> está indicando que el Servlet llamado Faces Servlet será accesado cada vez que se accede al directorio base indicado dentro del elemento <url-pattern>.

```
<servlet-mapping>
	<servlet-name>Faces Servlet</servlet-name>
	<url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

El elemento <welcome-file-list> indica que cuando se solicite cualquier directorio sin indicar un archivo en específico se envíe el archivo llamado index_admin.xhtml.

```
<welcome-file-list>
	<welcome-file>index_admin.xhtml</welcome-file>
</welcome-file-list>
```

El elemento <session-config> indica la duración en minutos de la sesión del usuario cuando navegue en la aplicación.

```
<session-config>
      <session-timeout>
         300
      </session-timeout>
</session-config>
```

El elemento <listener> define las características de un evento ServletContextEvent que ocurre dentro de la clase ContextListener.
Definimos el listener ContextListener dentro del archivo web.xml para que sea la primera clase que se ejecute al momento de levantar la aplicación.

```
<listener>
<listener-
class>ec.edu.utn.fica.eisic.qmanagement.servidor.ContextListener
</listener-class>
</listener>
```

El elemento <context-param> permite activar la biblioteca de componentes Richfaces que está dentro del paquete org.richfaces.skin donde el valor asignado es el classic que corresponde al tipo de presentación de la interfaz de usuario.

```
<context-param>
   <param-name>org.richfaces.skin</param-name>
   <param-value>classic</param-value>
</context-param>
```

Para indicar la ruta del archivo que carga el contexto de la aplicación se debe ingresar en el elemento <param-value>.

```
<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/classes/applicationContext.xml</param-
value>
</context-param>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
    <param-name>org.richfaces.skin</param-name>
    <param-value>classic</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <!--<param-value>classpath*:applicationContext.xml</param-value>-->
        <param-value>/WEB-INF/classes/applicationContext.xml</param-value>
    </context-param>
    <context-param>
    <param-name>javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE</param-name>
    <param-value>true</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <listener>
        <listener-class>ec.edu.utn.fica.eisic.qmanagement.servidor.ContextListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            300
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index_admin.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Figura B.1.4.6: Contenido del archivo **web.xml**
Fuente: Propia

- **faces-config.xml** que es el archivo de configuración de JavaServer Faces.

El elemento <navigation-rule> permite definir las reglas de navegación de la aplicación.

El elemento <el-resolver>
org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver>
permite activar el soporte de Spring para la inyección de dependencias de tal forma que JSF sabrá que si no encuentra un bean bajo su contexto debe ir a

buscarlo al contexto de Spring.

```
<application>
      <el-resolver>
            org.springframework.web.jsf.el.SpringBeanFacesELResolver
      </el-resolver>
</application>
```

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.1"
              xmlns="http://java.sun.com/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
              http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd">
        <!-- Integración JSF y Spring -->
        <application>
        <el-resolver>
                org.springframework.web.jsf.el.SpringBeanFacesELResolver
        </el-resolver>
        </application>

        <!-- Configuración de las reglas de navegación -->
        <navigation-rule>
            <from-view-id>/paginas/index.xhtml</from-view-id>
            <navigation-case>
                <from-outcome>success</from-outcome>
            <to-view-id>/paginas/success.xhtml</to-view-id>
        </navigation-case>
         <navigation-case>
                <from-outcome>error</from-outcome>
            <to-view-id>/paginas/error.xhtml</to-view-id>
        </navigation-case>
        </navigation-rule>
</faces-config>
```

Figura B.1.4.7: Contenido del archivo **faces-config.xml**
Fuente: Propia