MANUAL TÉCNICO Y DE CONFIGURACIÓN

Sistema de Automatización del Tratamiento de Información de Historias Clínicas y Medicamentos en el Subcentro de Salud de San Antonio de Ibarra

Desarrollado por: *Egdo. Jorge Aníbal Ipiales Ipiales*
Telf: *062 933310;* **Cel** *097292992*
Email: *jipiales2006@hotmail.com; j_ipiales2005@yahoo.com*
UTN-2011

ÍNDICE

INTRODUCCIÓN

En la actualidad la mayoría de actividades son automatizadas, pues para conseguir una mayor productividad en el trabajo, para esto necesitamos crear software que se utilice como una herramienta más en la rutina diaria del mismo y guardar los registros en forma segura y ordenada.

El proceso de atención a pacientes en una casa de salud no es la excepción, pues en este caso debemos manejar información básica de pacientes, y de las consultas que se realizan, con estos datos se puede utilizar para representarlos en gráficos estadísticos y realizar un determinado estudio con respecto a epidemias o casos relacionados con estos.

Con la utilización de software libre se pudo realizar un software especializado para realizar el trabajo de automatización de labores más trascendentales en este subcentro de salud, el mismo que agilita en gran parte las labores cotidianas del mismo.

El presente Software, es bastante amigable y fácil de instalar, el mismo que para usarlo se requiere conocimientos básicos en computación y por supuesto conocimientos de la forma de laborar en el Subcentro de Salud con el fin de desenvolverse sin problemas.

LA BASE DE DATOS

Las tablas de las bases de datos que se diseñó para el sistema están orientadas exclusivamente para el Subcentro, con esto se pretende cubrir las necesidades más urgentes de esta casa de salud.

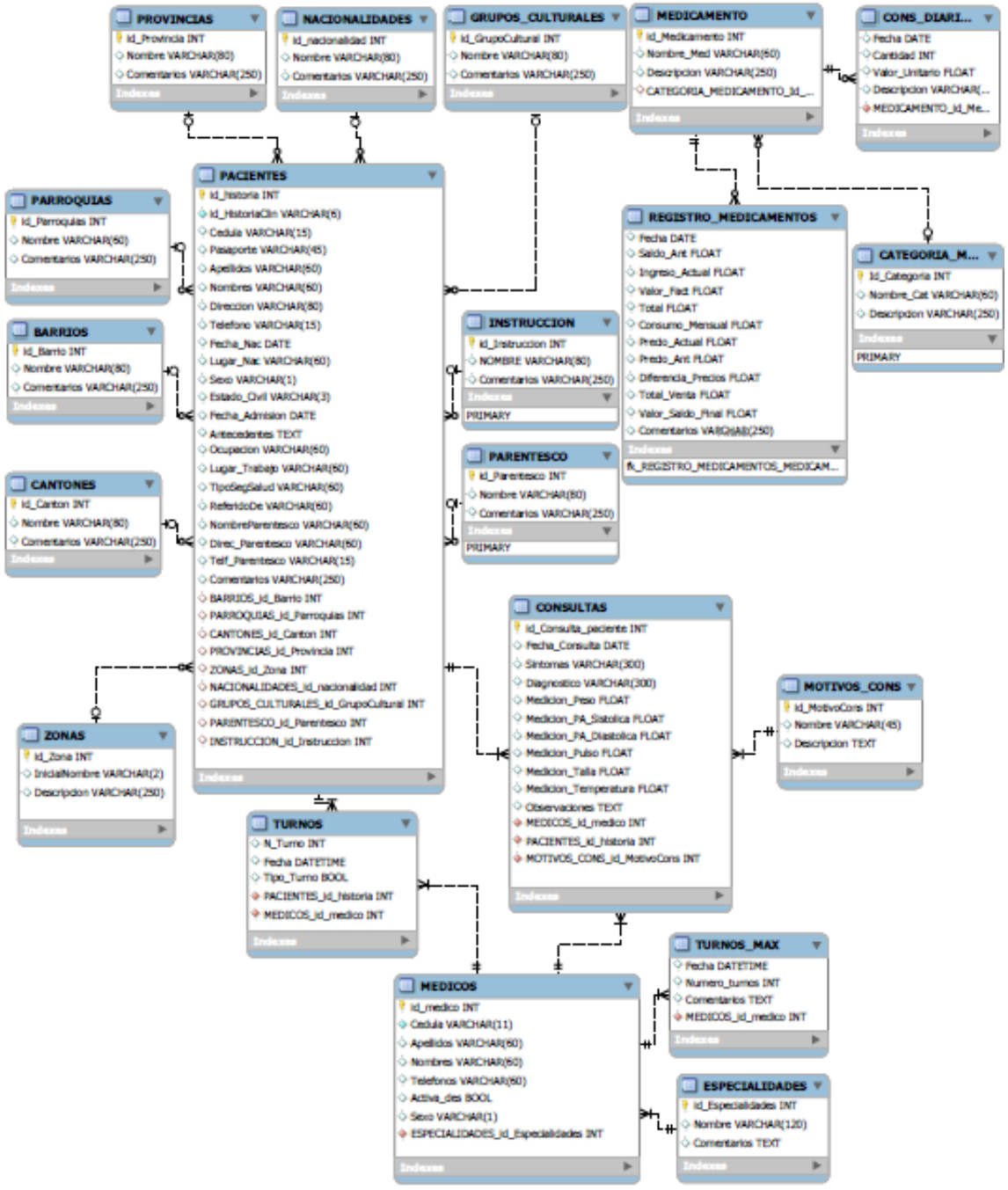


Figura 1.

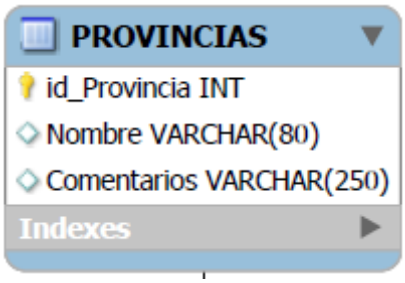
Las tablas que intervienen en la Base de datos del sistema lo presentamos en un Diagrama Entidad-Relación de la ***Figura 1***, estas son:

Tabla PACIENTES: es la más importante y también la más grande del sistema y sirve para registrar los datos básicos de los pacientes, sus campos son:

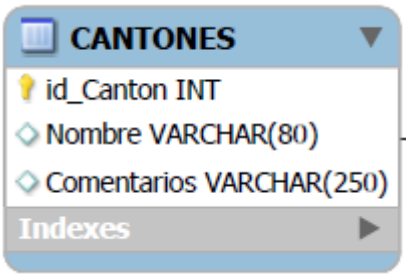
Campo	Tipo	Nulo	Enlaces a
id_historia	int(11)	No	
id_historiaclin	varchar(6)	No	
cedula	varchar(15)	Sí	
pasaporte	varchar(45)	Sí	
apellidos	varchar(60)	Sí	
nombres	varchar(60)	Sí	
direccion	varchar(80)	Sí	
telefono	varchar(15)	Sí	
fecha_nac	date	Sí	
lugar_nac	varchar(60)	Sí	
sexo	varchar(1)	Sí	
estado_civil	varchar(3)	Sí	
fecha_admision	date	Sí	
antecedentes	text	Sí	
ocupacion	varchar(60)	Sí	
lugar_trabajo	varchar(60)	Sí	
tiposegsalud	varchar(60)	Sí	
referidode	varchar(60)	Sí	
nombreparentesco	varchar(60)	Sí	
direc_parentesco	varchar(60)	Sí	
telf_parentesco	varchar(15)	Sí	
comentarios	varchar(250)	Sí	
barrios_id_barrio	int(11)	Sí	barrios -> id_barrio
parroquias_id_parroquias	int(11)	Sí	parroquias -> id_parroquias
cantones_id_canton	int(11)	Sí	cantones -> id_canton
provincias_id_provincia	int(11)	Sí	provincias -> id_provincia
zonas_id_zona	int(11)	Sí	zonas -> id_zona
nacionalidades_id_nacionalidad	int(11)	Sí	nacionalidades -> id_nacionalidad
grupos_culturales_id_grupocultural	int(11)	Sí	grupos_culturales -> id_grupocultural
parentesco_id_parentesco	int(11)	Sí	parentesco -> id_parentesco
instruccion_id_instruccion	int(11)	Sí	instruccion -> id_instruccion

Las siguientes tablas tienen relación con la tabla Pacientes:

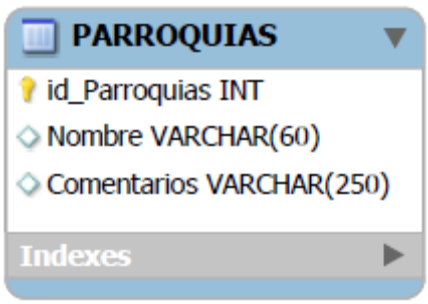
- **Tabla PROVINCIAS:** Esta tabla tiene su clave foránea a la tabla pacientes, y se utiliza para crear nuevas provincias y que sirven para registrar en la tabla pacientes, sus campos son:



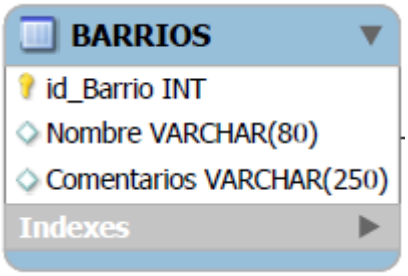
- **Tabla CANTONES:** Esta tabla también tiene su clave foránea a la tabla pacientes y de igual manera sirve para registrar en la tabla pacientes, sus campos son:



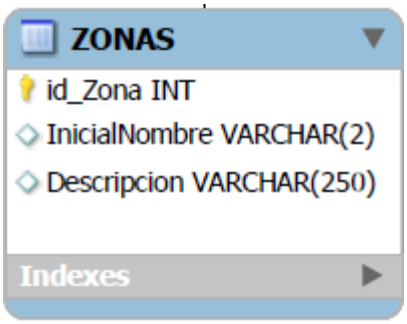
- **Tabla PARROQUIAS:** De igual manera se utiliza esta tabla para crear nuevas parroquias para registrar en la tabla pacientes, sus campos son:



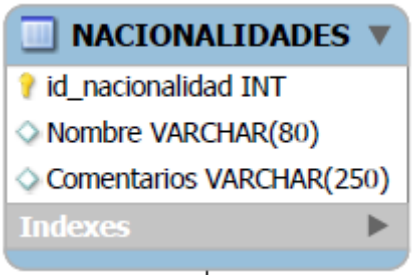
- **Tabla BARRIOS:** Así como las anteriores se utiliza para crear nuevos barrios, que sirven para registrar en los datos de pacientes, sus campos son:



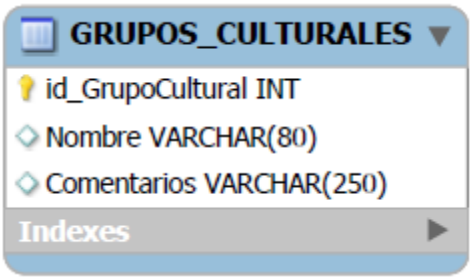
- **Tabla ZONAS:** Para crear nuevos tipos de zonas, y también son datos que sirven para el registro de pacientes; sus campos son:



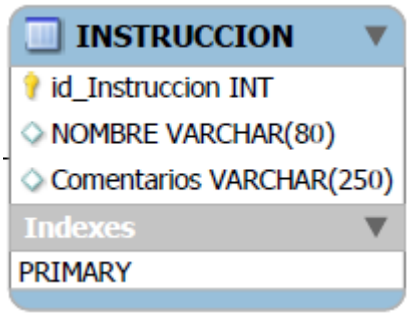
- **Tabla NACIONALIDADES:** Para crear nuevas nacionalidades, y de igual manera se utiliza para el registro de datos del pacientes, sus campos son:



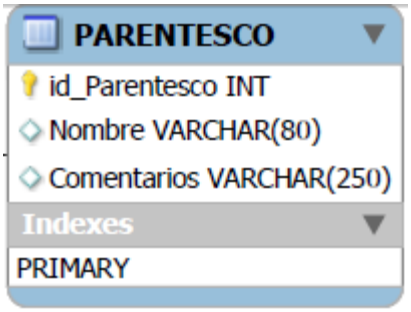
- **Tabla GRUPOS_CULTURALES:** Para crear nuevos tipos de grupos culturales y que se utilizan en el registro de datos de pacientes, su campos son:



- **Tabla INSTRUCCIÓN:** Para registrar nuevos tipos de Instrucción, y que sirve para registrar el nivel de instrucción que tiene el paciente, sus campos son:



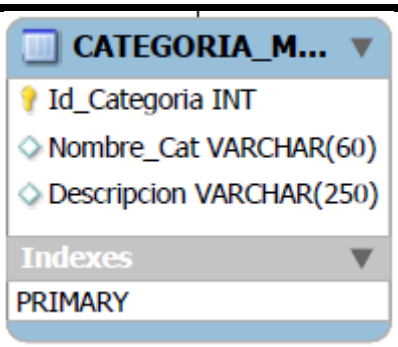
- **Tabla PARENTESCO:** Para crear nuevos tipos de parentescos y que de igual manera se utiliza en el registro de datos personales del paciente, sus campos son:



- **TABLA MEDICAMENTO:** La tabla Medicamento, se utiliza para registrar nuevos datos de medicamentos, los mismos que se expenden en Farmacia del Subcentro, los campos son los siguientes:

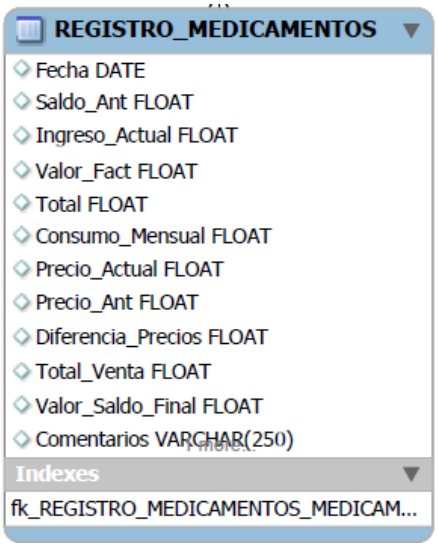
Campo	Tipo	Nulo	Enlaces a
id_medicamento	int(11)	No	
nombre_med	varchar(60)	Sí	
descripcion	varchar(250)	Sí	
categoria_medicamento_id_categoria	int(11)	Sí	categoria_medicamento -> id_categoria

- **Tabla CATEGORIA_MEDICAMENTO:** Esta tabla es utilizada para crear nuevas categorías de medicamentos, pues con el sistema tratamos de categorizar a los medicamentos, para que sea más entendible su orden, los campos son los siguientes.

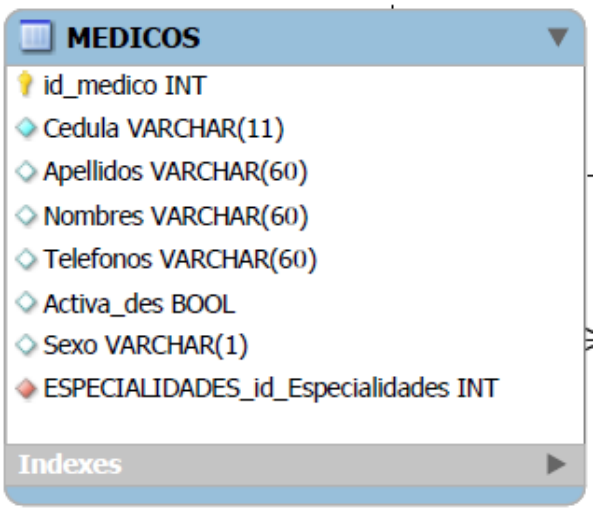


- **Tabla CONS_DIARIO_MEDICAM:** Esta tabla va a contener los registros de las salidas diarias de medicamentos, sus campos son:

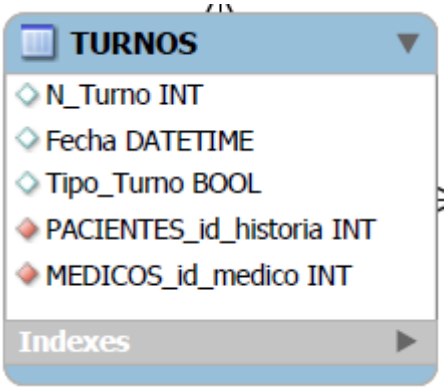
- **Tabla REGISTRO_MEDICAMENTOS:** Va a contener los registros que se realizan cada vez que se recibe medicamentos en el subcentro, el mismo que puede ser mensualmente.



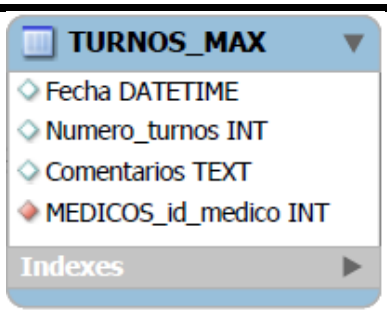
- **TABLA MÉDICOS:** Esta tabla es utilizada para registrar y administrar los datos personales de médicos que prestan servicios en esta institución, sus campos son:



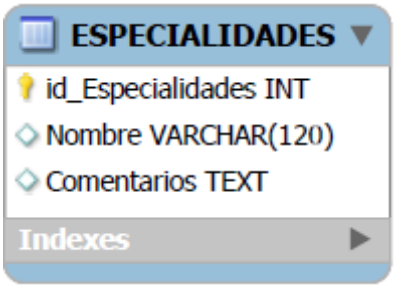
- **Tabla TURNOS:** En esta tabla vamos a registrar los datos de los turnos que se dan a diario en el subcentro, sus campos son:



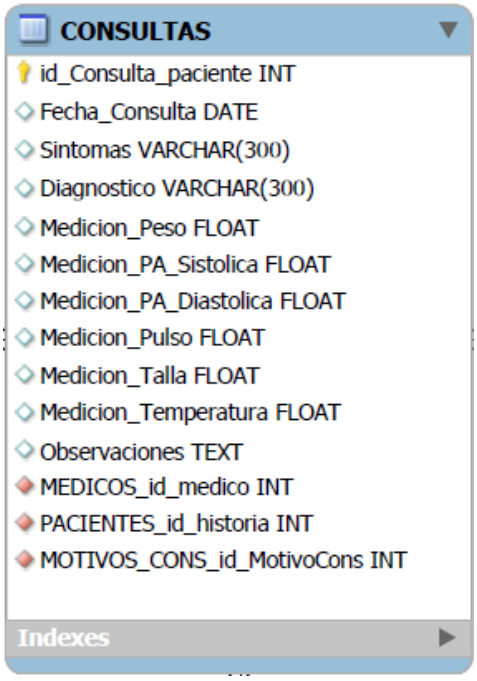
- **Tabla TURNOS_MAX:** En esta tabla se registra el límite de turnos que se asigna a cada médico, pero que por supuesto no es obligatorio ya que en caso contrario va a atender sin límite, sus campos son:



- **Tabla ESPECIALIDADES:** Con esta tabla podemos crear especialidades que se atenderán en el Subcentro, los campos son:



- **Tabla CONSULTAS:** Esta tabla guardará los datos cada vez que el paciente viene a realizarse una consulta, sus campos son:



MANUAL DE CONFIGURACIONES

CONFIGURACIÓN DE SYMFONY EN NETBEANS.

Symfony se basa en el patrón ***MVC (Modelo Vista Controlador)***, es decir, la estructura lógica de los proyectos creados se divide en:

- ***Aplicaciones:*** es decir nuestro proyecto,
- ***Clases:*** las clases los tipos de funcionalidades de las que consta nuestra aplicación (gestión de usuarios, gestión de noticias, gestión de comentarios, etc.) y,
- ***Módulos:*** que es donde se detalla cómo hacer cada funcionalidad de la clase a la que están asignados (alta, borrado y edición de usuarios, etc.).

Ahora nos dedicaremos a configurar Netbeans para trabajar con Symfony, de la siguiente manera:

En primer lugar debemos tener un "LAMP"[1] (Linux-Apache-MySQL-PHP/Python/PERL) instalado en nuestro equipo, en nuestro caso utilizamos WAMP SERVER 2.0.

Luego descargamos la última versión estable de Symfony desde su web, para el inicio de nuestro proyecto la última versión estable fue Symfony 1.4.

Una vez descargado el paquete que normalmente viene empaquetado, lo descomprimimos en el directorio de nuestra web, como nosotros trabajamos de forma local, creamos una carpeta en la unidad C:/, de la siguiente manera: "C:\SubcentroSalud\sistema\", entonces la dirección completa donde lo descomprimimos va a ser algo así: C:\SubcentroSalud\sistema\lib\vendor\symfony

En nuestro siguiente paso abrimos Netbeans, para luego crear un nuevo proyecto PHP, el asistente nos guiará paso a paso, en donde nosotros solo daremos el

[1] Conjunto de aplicaciones para configurar sitios web o servidores dinámicos con un esfuerzo reducido.

nombre a nuestro proyecto y localización de nuestros archivos del proyecto, pero en el paso donde debemos elegir el framework (PHP frameworks), obviamente elegiremos Symphony. Al seleccionarlo nos aparecerán varias opciones de configuración. Tendremos que pulsar sobre Options, a la derecha de la ventana, como se indica en el ***Figura 2*** :

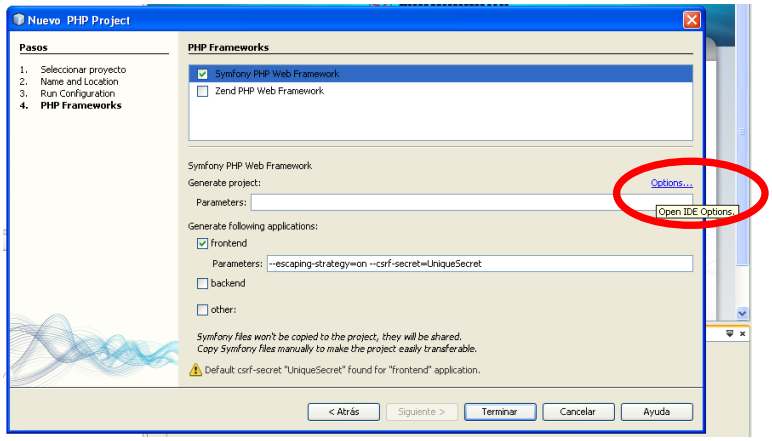


Figura 2.

Luego de hacer clic en Options…, se nos abrirá el panel de configuración de PHP, en esta ventana hacemos clic en la pestaña de PHP en donde indicaremos la ruta donde se encuentra el ejecutable de PHP del LAMP que hayamos instalado, en nuestro caso se encuentra en la siguiente ruta:

C:\wamp\bin\php\php5.3.0\php.exe

Además, en esta misma ventana tendremos que indicar la ruta de instalación de Symfony pulsando el botón ***Add Folder***, que será la carpeta que hemos descomprimido en el directorio de nuestro proyecto

C:\SubcentroSalud\sistema\lib\vendor\symfony

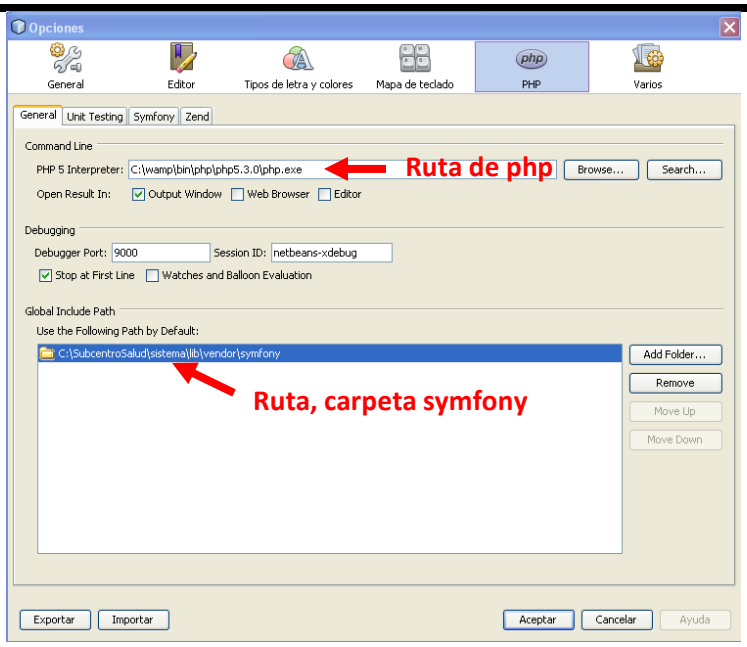


Figura 3.

Ahora, en la pestaña Symfony, de esta misma pantalla tendremos que indicar la ruta del fichero binario del framework, la cual en nuestro caso la encontramos en la siguiente dirección:

C:\SubcentroSalud\sistema\lib\vendor\symfony\data\bin\symfony

Esto lo indicamos en el siguiente gráfico:

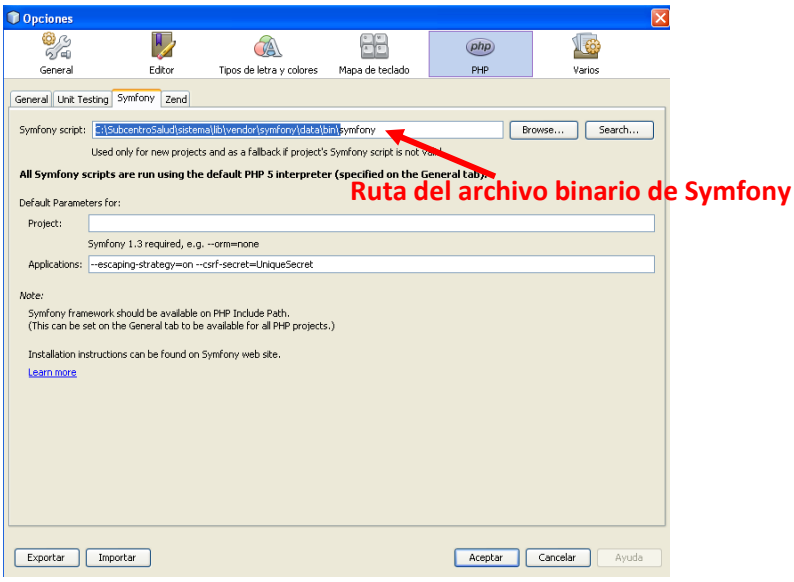


Figura 4.

Y por último, hacemos clic en ***Aceptar,*** pero antes de terminar, seleccionamos el checkbox ***Backend***, en donde debe quedar con un visto como lo indicamos en el siguiente gráfico, esto con el fin de que nuestro proyecto maneje la base de datos, luego de esto si pulsamos en ***Terminar***.

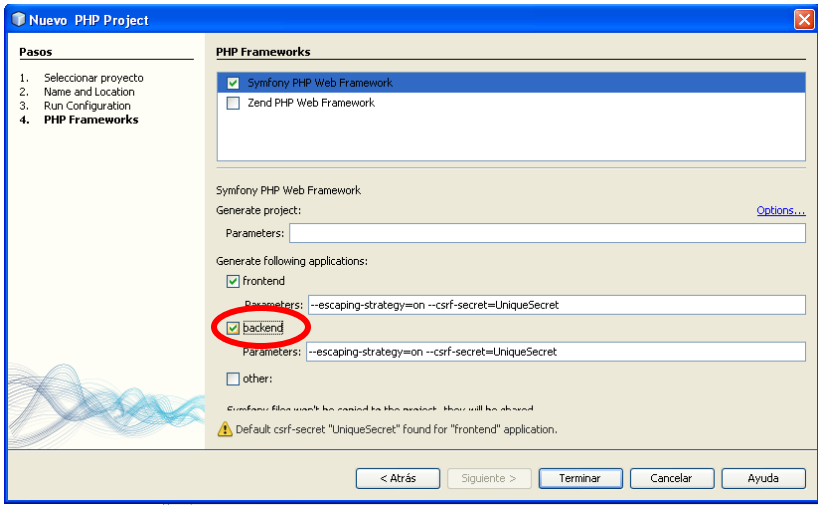


Figura 5.

Si no hay ningún inconveniente en el transcurso de todos los pasos que nos guió el asistente, ya tendremos configurado e instalado nuestro proyecto ***Symfony*** en ***NetBeans***.

CONFIGURACIÓN DE APACHE

Para que funcione la aplicación, debemos ejecutar primeramente nuestro servidor web, que en nuestro caso es el WampServer 2i, por lo que ya se debe tener instalado, en caso contrario no funcionará, por lo que para instalarlo primeramente se debe descargar de internet (http://www.wampserver.com/en/), si lo tenemos instalado debemos ver un icono como el siguiente:

Figura 6

Como ya solucionamos lo anterior, debemos configurar primeramente el host que en este caso nuestro host será local y virtual, de tal manera que se ejecute bajo otro nombre de host y en otro puerto (en este caso el puerto 80).

En internet encontramos varios ejemplos de cómo crear un host virtual para ejecutar el proyecto de forma local, aquí tenemos un ejemplo:

```
NameVirtualHost *:80
<VirtualHost *:80>

ServerName miApp
DocumentRoot "C:\Users\Juanjo\Documents\NetBeansProjects\miApp\ web"
DirectoryIndex index.php
<Directory "C:\Users\Juanjo\Documents\NetBeansProjects\miApp\ web">
AllowOverride All
Allow from All
</Directory>

#solo para la barra de depuración:

Alias /sf "C:\symfony-1.4.11\data\web\sf"
<Directory "C:\symfony-1.4.11\data\web\sf">
AllowOverride All
Allow from All
</Directory>

</VirtualHost>
```

Lo que vamos a hacer nosotros es coger cualquiera de estos códigos y editarlo para acoplarlo a nuestro proyecto, hay que tomar en consideración que nosotros vamos a utilizar Symfony como framework para programar con Php, es por eso que esta configuración hace referencia a esta carpeta en donde tendremos nuestro Symfony, lo que nos quedaría de la siguiente manera:

```
# Esta es la configuración de Sistema para Subcentro de Salud
#Listen 127.0.0.1:80

<VirtualHost 127.0.0.1:80>
  ServerName sistema
  DocumentRoot "C:\SubcentroSalud\sistema\web"
  DirectoryIndex index.php
  Alias /sf "C:\SubcentroSalud\sistema\lib\vendor\symfony\data\web\sf"
  <Directory "C:\SubcentroSalud\sistema\lib\vendor\symfony\data\web\sf">
    AllowOverride All
    Allow from All
```

```
  </Directory>
  <Directory "C:\SubcentroSalud\sistema\web">
    AllowOverride All
    Allow from All
  </Directory>
</VirtualHost>
```

Como ya están establecidas por Symfony las rutas de la carpeta del proyecto, nos queda seguir los siguientes pasos:

1. Copiar el contenido (Ctrl+C) del código anterior (del VirtualHost),
2. Abrir el archivo httpd.conf que se encuentra en la siguiente ruta: C:\wamp\bin\apache\Apache2.2.11\conf,
3. Lo pegamos (Ctrl+V), normalmente en la parte final del contenido de este archivo,
4. Guardamos,
5. Y por último reiniciamos los servicios de Apache.

Si todo ha salido bien, y si hemos configurado correctamente Apache, abrimos el navegador y anotamos la siguiente dirección: http://localhost:80, pero también lo podemos hacer desde Netbeans, digitando la tecla F6 para ejecutar el proyecto. Y de esta manera nos debe salir una pantalla similar a la siguiente:

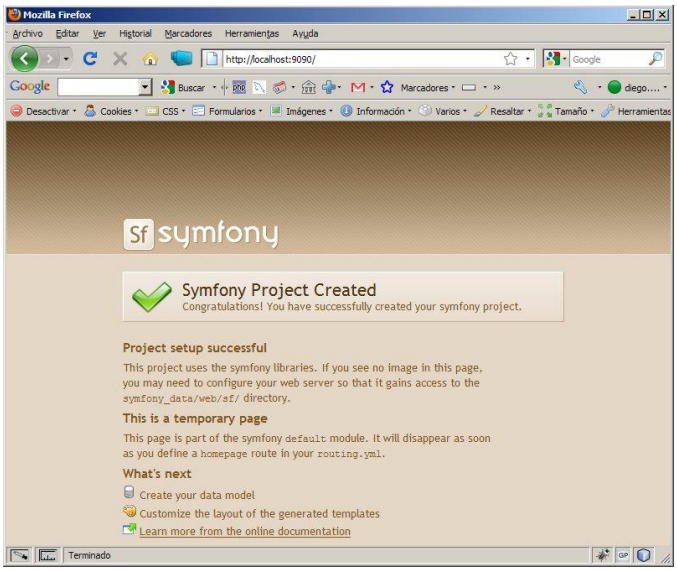


Figura 7.

Todo lo anterior es para correr la aplicación como un servidor virtual, por lo tanto se debe haber agregado el dominio virtual al archivo ***hosts*** de nuestro sistema operativo, para esto voy a la dirección C:\Windows\System32\drivers\etc\hosts y agrego:

127.0.0.1 localhost
127.0.0.1 sistema

Por lo que para entrar a nuestro proyecto solo digitaremos ***http://sistema*** en nuestro navegador (Mozilla Firefox 4 o superior de preferencia).

MANUAL TÉCNICO

Después de haber realizado las configuraciones necesarias para que nuestra aplicación pueda funcionar, vamos a analizar la estructura de nuestro proyecto, el mismo que fue realizado con la ayuda de Netbeans 6.9.1., la última versión estable hasta la fecha de iniciación de nuestro proyecto.

Para empezar a desarrollar proyectos con Symfony, hay que tener un conocimiento por lo menos básico de HTML, CSS, con esta forma de programación aprenderemos a ser un poco más ordenados en relación a lo que tiene que ver al lugar donde van a ir código de programación.

Pues hay veces que utilizaremos solo código HTML, en otros casos deberíamos utilizar solo PHP o una mezcla de ellos, además hacer referencia a archivos CSS para poder dar estilos a nuestras páginas.

Lo primero que debemos hacer después de descargar el paquete symfony desde su sitio oficial que está empaquetado, es desempaquetar, una vez que tengamos en nuestro computador debemos grabar esta carpeta en la carpeta de nuestro proyecto, o puede ser en la carpeta www de la carpeta wamp en el caso de Windows c:\wamp\www\, es recomendable grabar en una carpeta distinta si vamos a trabajar en forma local, es decir sin subir al sitio en Internet, en el caso que se va a subir al ciberespacio, pero esto hay que hacerlo después de haber creado el proyecto con symfony.

Nos parece importante entender como es la estructura del código Symfony. Todo proyecto creado por Symfony consta de una serie de carpetas y archivos que contienen el código base del framework. El árbol de carpetas de un proyecto es:

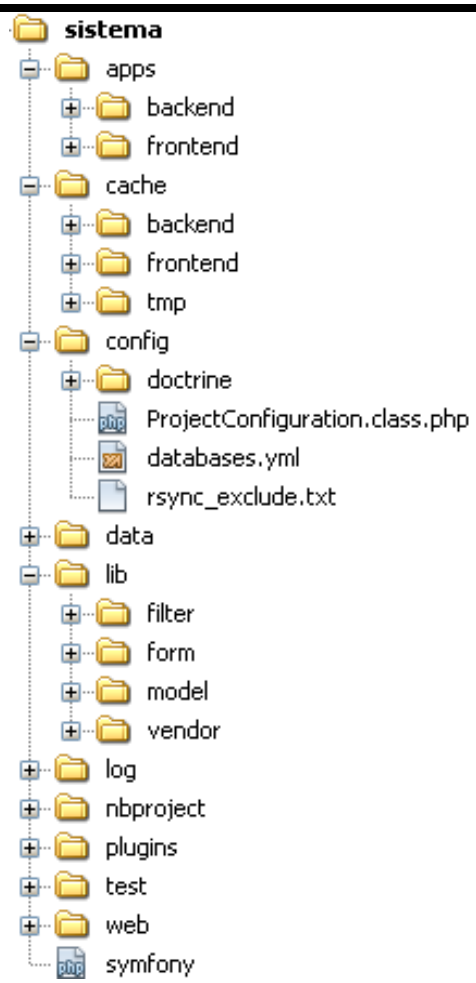


Figura 8.

Además, un proyecto creado con symfony están divididos en:

- **Aplicaciones:** los mismos que pueden tener varios módulos, normalmente pueden haber dos aplicaciones Backend y Frontend.
- **Módulos:** Estos a su vez suelen coincidir con las tablas de la base de datos creada según sea el propósito del proyecto.

Después que tengamos la base de datos y tengamos claro como es la estructura de un proyecto symfony, vamos a crear nuestro código.

Para esto tenemos que crear una carpeta en donde vamos a crear todas las carpetas y también copiar la carpeta Synfony descargada desde su sitio oficial, en

nuestro caso vamos a ayudarnos de una herramienta “Netbeans”, la misma que me permite crear en forma visual nuestro proyecto, de la siguiente manera:

Figura 9.

Esta forma de crear el proyecto reemplaza al siguiente código:

symfony generate:project sistema

Cualquiera sea la forma en la que creamos nuestro proyecto, nos habrá creado todo el árbol de directorios y archivos necesarios para poner en marcha nuestra web con este framework.

Luego creamos nuestros módulos, pero si utilizamos Netbeans como es en nuestro caso al crear nuestro proyecto también ya se crea estas dos aplicaciones por defecto.

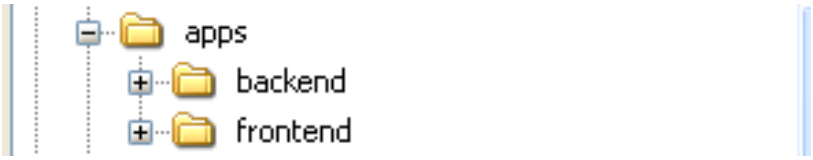


Figura 10.

En cambio si no utiliza Netbeans, tendrá que crearlos manualmente, aplicando un código similar al siguiente:

- *symfony generate:app --escaping-strategy=on --csrf-secret=UniqueSecret frontend*
- *symfony generate:app --escaping-strategy=on --csrf-secret=UniqueSecret backend*

Al llamar a la tarea generate:app, también hemos pasado dos opciones relacionadas con la seguridad:

- *–escaping-strategy: Permite escapar la salida para evitar ataques XSS*
- *–csrf-secret: Permite tokens de sesión en los formularios para prevenir los ataques CSRF*

El siguiente paso, será conectar nuestra aplicación con la base de datos, para ello, Symfony dispone de un archivo de configuración **databases.yml** en el que le indicaremos donde está localizada, a que base de datos y con qué usuario y contraseña lo deberá hacer.

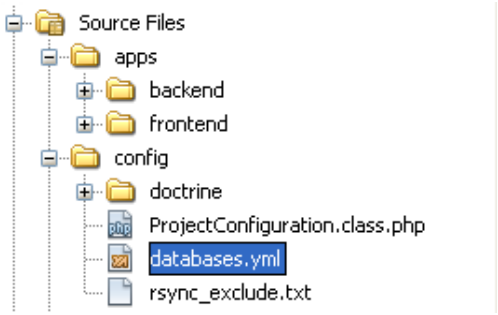


Figura 11.

Este archivo normalmente contiene el siguiente un código similar al siguiente:

```
all:
  doctrine:
    class: sfDoctrineDatabase
    param:
      dsn: 'mysql:host=localhost;dbname=bdblog'
      username: root
      password: null
```

Con esto, ya tenemos configurado nuestro proyecto con la base de datos.

Ahora le decimos a Doctrine que genere el archivo *schema.yml* a partir de la base de datos:

symfony doctrine:build-schema

Luego creamos el modelo, para que se genere clases para manejar la base de datos mediante objetos, de la siguiente manera:

symfony doctrine:build-model

De una manera similar crearemos con doctrine, los filtros, los formularios, etc. Para no realizar de manera separada todas estas acciones, utilizaremos un solo comando, de la siguiente manera:

Symfony doctrine:build –all

Solo que con el comando anterior hay que tener cuidado si es que ya tiene creado la base de datos anteriormente, ya que esta es una forma de resetear la base de datos y dejarla en blanco, o los vuelve a reconstruir.

Como hicimos notar anteriormente, nosotros utilizamos Netbeans, por lo que estas líneas de código lo podemos ejecutar de forma visual.

Si todo salió bien, ya podemos irnos con nuestro navegador a ver la pantalla de Symfony en donde nos dice que el proyecto ha sido creado, para eso visitamos la dirección **http://localhost/sistema/**, de esta forma:

Figura 12.

De aquí en adelante no es más que crear módulos en nuestras aplicaciones como es el caso de Backend y Frontend, además de insertar código para manipular y controlar nuestro objetivo en el sistema, pero estas líneas de código PHP van a ser mínimos.

ESTRUCTURA DE ARCHIVOS NUESTRO PROYECTO SYMFONY

En el presente proyecto tenemos dos aplicaciones, Backend y Frontend:

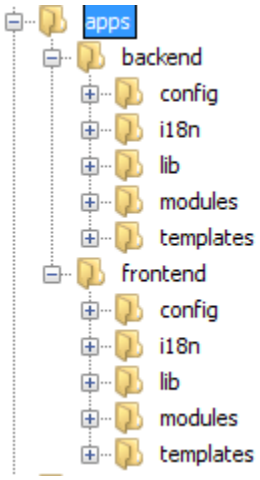


Figura 13

FRONTEND: En la **Figura 13** podemos observar visualmente que esta aplicación está formada por otros directorios que cuando se crea la aplicación Symfony con Netbeans las crea.

BACKEND: Esta aplicación tiene una estructura similar a la anterior como lo podemos observar en la **Figura 13**, solo que más adelante podremos ver que en esta aplicación es donde se invirtió mayor esfuerzo en programación y dedicación de tiempo.

ESTRUCTURA DE LA APLICACIÓN

La aplicación contiene varios subdirectorios como podemos visualizar en la f***igura 13***, estas son:

SUBDIRECTORIO config:

La carpeta que contiene archivos yaml para configurar la aplicación, estas la podemos observar en la siguiente figura:

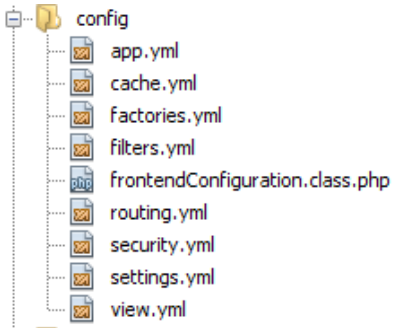


Figura 14.

Como podemos observar, existen archivos yaml, para configurar la aplicación (app.yml), para configurar la cache (cache.yml), los objetos básicos que necesita el marco de trabajo durante la vida de cualquier solicitud (factories.yml), para configurar los filtros (filters.yml), para configurar las formas de ingresar a las diferentes páginas llamadas rutas (routing.yml), para asegurar o no la aplicación y que solo personas autorizadas puedan ingresar (security.yml), para configurar los diferentes entornos de ejecución del sistema (settings.yml) y por supuesto para configurar las vistas (view.yml). Todos estos archivos ya vienen con su propio código que fue generado en la creación de la aplicación.

SUBDIRECTORIO I18n:

En esta carpeta guardaremos archivos de configuración de idiomas para las aplicaciones, en el caso de la aplicación frontend, no tiene ningún archivo de configuración para esta situación, pero para el caso de la aplicación Backend si la tenemos, este archivo se localiza en la dirección que se muestra en la ***Figura 15***.

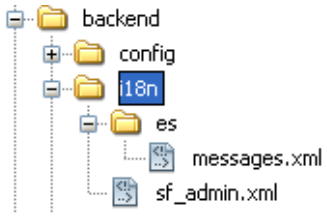


Figura 15.

Como podemos observar, son archivos xml, en donde si lo abrimos existen traducciones al español para los mensajes más comunes que normalmente se muestran en inglés,

SUBDIRECTORIO lib:

Esta carpeta contiene las librerías útiles para la aplicación, que pueden ser creadas por el mismo framework o por el programador para crear o utilizar sus propias librerías, normalmente viene con un solo archivo que se crea por defecto al crear nuestra aplicación, como se muestra en la siguiente figura:

Figura 16.

SUBDIRECTORIO modules:

Este directorio es el que contiene los módulos que se van a crear para el sistema, los mismos que pueden ser creados visualmente ayudado de netbeans, como se muestra en la siguiente figura:

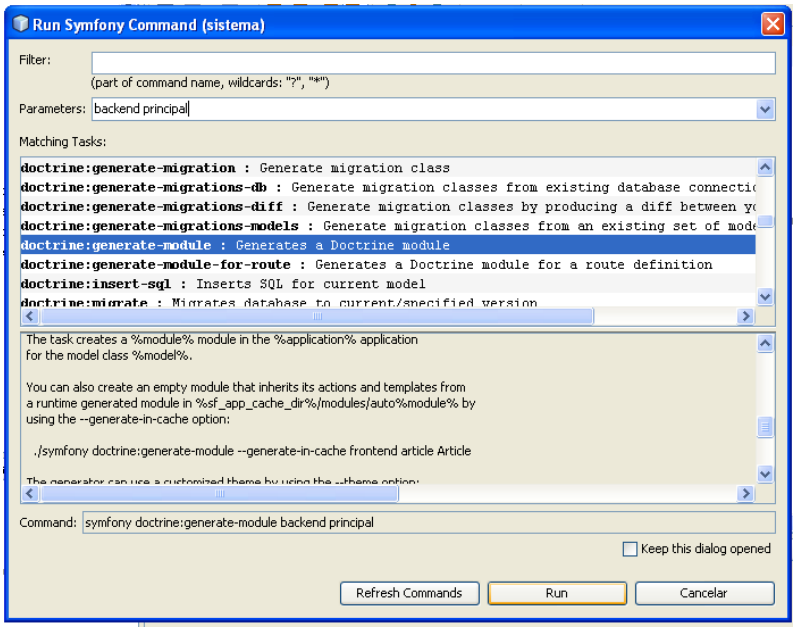


Figura 15.

En la aplicación **FrontEnd**, solo tenemos un solo módulo en donde ingresamos información básica del Subcentro (Nosotros) y de la aplicación (Acerca de..), el mismo que se llama **principal.**

ESTRUCTURA DE LOS MÓDULOS

En cada módulo symfony crea por defecto subdirectorios al ejecutar el comando para crear módulos que en forma visual ayudado con netbeans se lo puede localizar ayudado de la siguiente figura:

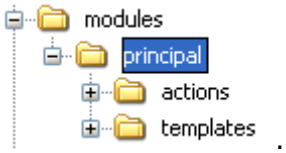


Figura 15.

SUBDIRECTORIO actions:

En donde van a tener las funciones necesarias para ejecutar alguna acción, al ejecutar estas acciones devuelven el resultado al layout respectivo, por lo que para cada acción tenemos un archivo de visualización (layout) en el subdirectorio templates del subdirectorio del módulo como podemos observar en la figura 15, la sintaxis del nombre de la acción es como sigue:

public o protected ***execute*** *+ [nombre_acción]+([argumentos])*

Por ejemplo *public executeIndex(sfWebRequest $request),* todas estas funciones están en el archivo *actions.class.php* las mismas que pueden ser públicas (public) o privadas (protected) dentro del directorio del mòdulo que estemos viendo.

Para el caso del sistema del Subcentro de Salud, generamos los módulos con el comando de symfony admin-generator, por lo que se crea automáticamente otros subdirectorios a más de los que visualizamos en la figura 15, esto lo vamos a mostrar en la figura 16.

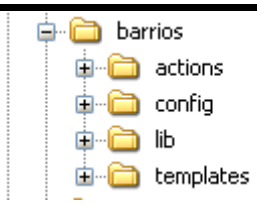


Figura 16.

Como podemos observar en la figura 16, a màs de las dos carpetas que se pudo visualizar en la figura 15, tenemos una carpeta en donde contendrà archivos de configuración (config) y una carpeta en donde se pueda guardar las librerìas propias del módulo (lib).

Pero cada carpeta contiene archivos, que pueden ser con extensión .yml para configuraciones normalmente, o .class.php en donde podemos introducir nuestras propias funciones o manipular las funciones que por defecto crea symfony y archivos .php, para parciales o para las vistas del módulo, algunas de estas las podemos observar en la siguiente figura:

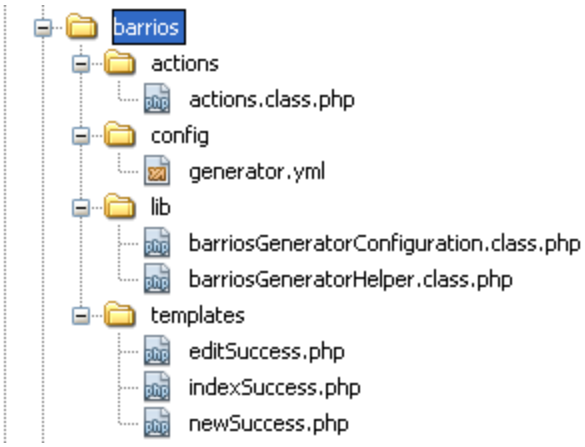


Figura 17.

SUBDIRECTORIO config:

El subdirectorio config de un módulo, inicialmente los módulos no tienen ninguna configuración específica, en cualquier caso, es posible modificar las opciones de la aplicación en cualquier módulo que así lo requiera. Algunos de los usos típicos son los de cambiar la descripción HTML en todas las acciones de un módulo,

también se pueden añadir nuevos parámetros exclusivamente para un módulo concreto.

El archivo que más se ha manipulado en nuestro caso se llama el generator.yml, que podemos observar visualmente en la **Figura 17**.

El archivo ***generator.yml*** se utiliza para los módulos generados automáticamente a partir de una tabla de la base de datos, es decir, para los módulos utilizados en el *scaffolding* y para las partes de administración creadas de forma automática, contiene las opciones que definen como se muestran las filas y los registros en las páginas generadas y también define las interacciones con el usuario: filtros, ordenación, botones, etc.

Además podemos crear manualmente otros archivos para configurar nuestro módulo entre ellas pueden ser: para configurar la seguridad ***security.yml.*** para configurar la vista del módulo ***view.yml,*** etc.

SUBDIRECTORIO lib:

El directorio /lib debería contener sólo las bibliotecas (libraries) necesarias para ejecutar llamadas a funciones creadas por defecto del framework o creadas por el programador.

En el presente caso solo utilizamos las creadas por el framework, estas librerías se encuentran en dos archivos ***nombreGeneratorConfiguration.class.php,*** en donde nombre se refiere al nombre del módulo, y el ***nombreGeneratorHelp.class.php,*** de la misma manera, nombre se refiere al nombre del módulo.

Estos dos archivos contienen funciones públicas que son llamadas por el sistema, para que realice alguna actividad, pero estas solo se encuentran en el directorio cache en forma temporal, mientras este módulo se lo utilice, ya que si lo mandamos a limpiar el cache, todas estas desaparecen, aquí vamos a ver el contenido de la caché.

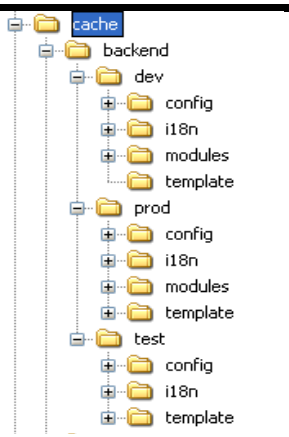


Figura 18.

Como hemos podido observar en la ***figura 18***, tres subdirectorios dev, prod y test, estos son los llamados entornos: El **entorno de desarrollo (dev)**: Este es el ambiente utilizado por desarrolladores web para añadir nuevas funciones, corregir los errores, etc., el **entorno de producción (prod):** Este es el entorno donde un usuario final interactúa y el **entorno de prueba (test):** Este entorno se utiliza para probar automáticamente la aplicación. Depende cómo se esté trabajando para que se muestren los archivos temporales en la caché, es decir que si estamos trabajando en el entorno de desarrollo los directorios *prod y test* van a estar vacios, y así en los otros entornos solo mostraran el contenido de estos directorios si están trabajando en ellos.

NOTA: Como se muestra en la **figura 18**, el contenido de la caché en cada uno de los entornos, también contienen los mismos directorios que mostramos en el directorio apps de la aplicación y de los respectivos módulos, es decir que en este directorio se crea como una especie de espejo en la caché, solo que en esta se muestra el código de las funciones creadas automáticamente por el framework, en las cuales podemos introducir nuestro propio código para personalizarlos o nuestras propias funciones.

SUBDIRECTORIO templates:

Este directorio contiene todas las plantillas globales para una aplicación cuando es el caso de la aplicación y para el módulo si este es el caso, es decir que este directorio se encarga de la vista en Symfony, la vista está formada por dos partes:

- La presentación HTML del resultado de la acción (que se guarda en la plantilla, en el layout y en los fragmentos de plantilla)
- El resto, que incluye entre otros los siguientes elementos:
    - Declaraciones <meta>: palabras clave (keywords), descripción (description), duración de la cache, etc.
    - El título de la página: no solo es útil para los usuarios que tienen abiertas varias ventanas del navegador, sino que también es muy importante para que los buscadores indexen bien la página.
    - Inclusión de archivos: de JavaScript y de hojas de estilos.
    - Layout: algunas acciones necesitan un layout personalizado (ventanas emergentes, anuncios, etc.) o puede que no necesiten cargar ningún layout (por ejemplo en las acciones relacionadas con Ajax).

En la vista, todo lo que no es HTML se considera configuración de la propia vista y Symfony permite 2 formas de manipular esa configuración. La forma habitual es mediante el archivo de configuración view.yml. Se utiliza cuando los valores de configuración no dependen del contexto o de alguna consulta a la base de datos. Cuando se trabaja con valores dinámicos que cambian con cada acción, se recurre al segundo método para establecer la configuración de la vista: añadir los atributos directamente en el objeto sfResponse durante la acción.

El layout de una aplicación se llama layout.php y se puede encontrar en el directorio apps/frontend/templates/, en el caso de la aplicación, como se muestra en la siguiente figura:

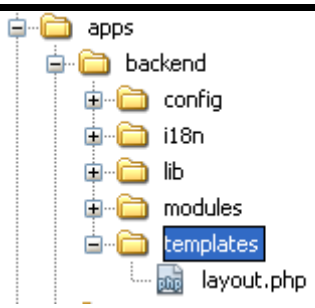


Figura 19.

Y archivos .php que pueden ser fragmentos de código que se les conoce como parciales y su nombre empiezan con dos rayas bajas seguidas y los archivos de vistas que siempre terminan con Success, como por ejemplo indexSuccess.php todos los mismos sirven como vistas en cada módulo, estos se muestran en la siguiente figura:

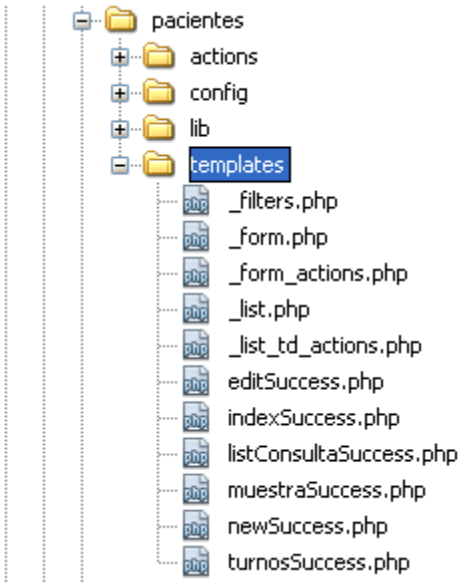


Figura 20

En resumen, la **figura 21**, muestra un esquema de cómo se organizan los archivos y subdirectorios de nuestro proyecto, siguiendo la estructura de proyecto / aplicación / módulo / acción.

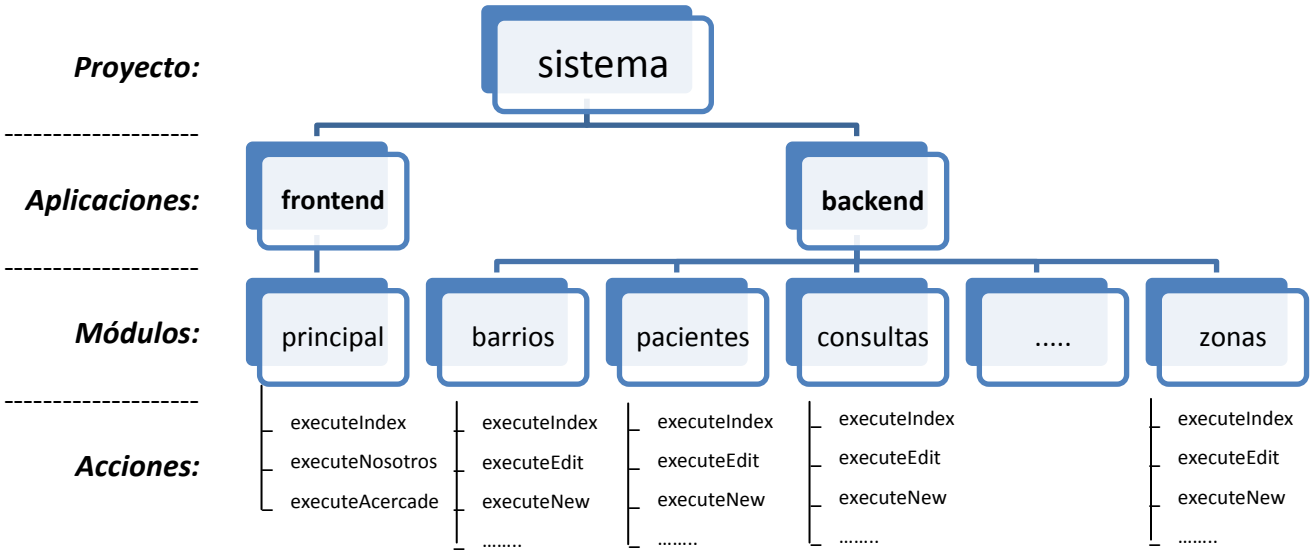


Figura 21

ESTRUCTURA DEL DIRECTORIO config DE LA APLICACIÓN

Nuestro directorio config, a nivel de aplicación contiene archivos de configuración de la aplicación a su vez otros directorios y archivos como se muestra en la siguiente pantalla:

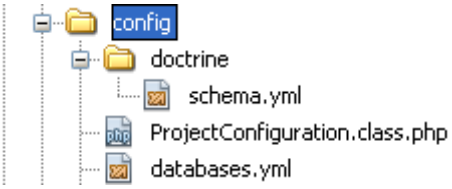


Figura 22

SUBDIRECTORIO doctrine:

Este subdirectorio contiene a su vez el archivo shema.yml, en donde contiene código de la estructura de la base de datos como son las tablas y relaciones entre ellas, pero escritas en formato yml, esto con el fin de estandarizar y poder realzar su conexión cualquier otro motor de base de datos (Mysql, Postgressql, etc.), este código es similar al siguiente:

```
Barrios:
  connection: doctrine
  tableName: barrios
  columns:
    id_barrio:
      type: integer(4)
      fixed: false
      unsigned: false
      primary: true
      autoincrement: true
    nombre:
      type: string(80)
      fixed: false
      unsigned: false
      primary: false
      notnull: false
      autoincrement: false
    comentarios:
      type: string(250)
      fixed: false
      unsigned: false
      primary: false
      notnull: false
      autoincrement: false
  relations:
    Pacientes:
      local: id_barrio
      foreign: barrios_id_barrio
      type: many
```

Este código se consigue ejecutando el comando doctrine:build-shema, mostraremos su forma de ejecutarlo ayudado con netbeans:

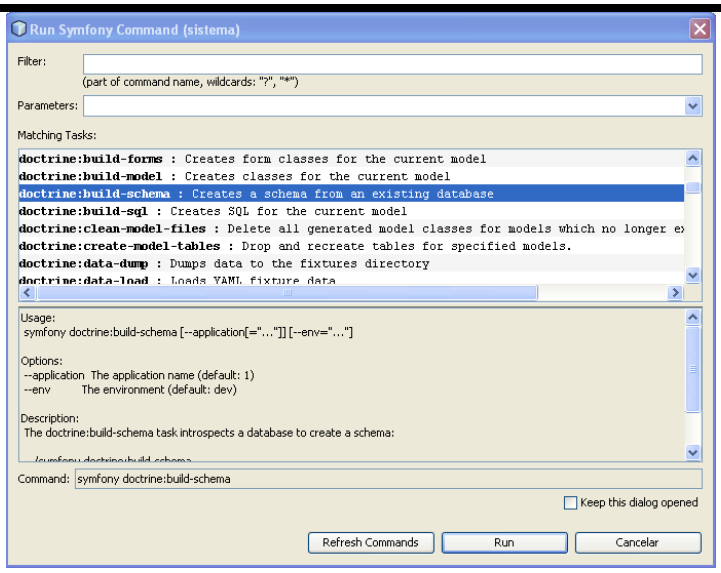


Figura 23

ARCHIVO ProjectConfiguration.class.php:

Este es un fichero de configuración del proyecto, en donde configuramos la dirección del proyecto y desde donde podemos habilitar o deshabilitar los plugins que utiliza el proyecto, ¿Qué conseguimos con esto?, pues simplemente quitarle la rigidez al buscar el archivo sfCoreAutoload.php predefinido con la ruta absoluta y decirle que lo busque desde donde esté el archivo ProjectConfiguration.class.php. Ahora se podrá llevar la carpeta del proyecto a cualquier ubicación y el proyecto seguirá funcionando perfectamente, el código que encontramos en este fichero es:

```
<?php
require_once
dirname(__FILE__).'/..\lib\vendor\symfony\lib/autoload/sfCoreAutoload.class.php';
sfCoreAutoload::register();
class ProjectConfiguration extends sfProjectConfiguration
{
  public function setup()
  {
    $this->enablePlugins('sfDoctrinePlugin');
    $this->enablePlugins('sfDoctrineGuardPlugin');
    $this->enablePlugins('sfTCPDFPlugin');
    $this->enablePlugins('sfFormExtraPlugin');
    $this->enablePlugins('sfJqueryReloadedPlugin');
  }
}
```

ARCHIVO databases.yml:

Es un archivo bastante importante para configurar la conexión con la base de datos, pues aquí es donde especificamos los datos de usuario y la base de datos con la que queremos conectarnos, aquí encontramos el siguiente caso:

```
all:
  doctrine:
    class: sfDoctrineDatabase
    param:
      dsn: 'mysql:host=localhost;dbname=bdsubcentro'
      username: root
      password: null
```

ESTRUCTURA DEL DIRECTORIO data DE LA APLICACIÓN

En nuestro directorio data que encontramos dentro del directorio de nuestro proyecto se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas, este directorio lo vamos a visualizar en la **figura 24**:

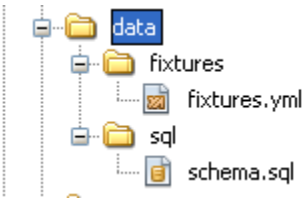


Figura 24

El framework Symfony soporta todas las Base De Datos soportadas por "PDO"[2] (MySQL, PostgreSQL, SQLite, Oracle, MSSQL). PDO es la capa de abstracción de base de datos que viene con PHP.

En nuestro caso vamos a usar MySQL, este puede ser cualquier otro, pues no será difícil adaptar el código que vamos a escribir ya que vamos a utilizar el ORM quien será quien escriba el SQL por nosotros.

[2] *PHP Data Objects (PDO) define un interfaz ligera, para tener acceso a bases de datos en PHP*

SUBDIRECTORIO fixtures:

En el directorio fixtures, encontramos el archivo fixtures.yml, los mismos que son una configuración de datos para nuestro proyecto Symfon, una buena forma de precargar información en nuestra base de datos de forma semi-automática. Es habitual que en el transcurso del desarrollo de una aplicación se restablezcan una y otra vez la información contenida en nuestra base de datos, la utilidad del mecanismo de los fixtures es la reinserción bajo demanda de los datos necesarios para la correcta ejecución de esta (pej: usuarios), asi:

User:
fabien:
username: fabien
password: changeme
name: Fabien Potencier
email: fabien.potencier@symfony-project.com

Para realizar la acción de cargar a este usuario tenemos que ejecutar el siguiente comando ayudado por netbeans, como muestra la siguiente pantalla:

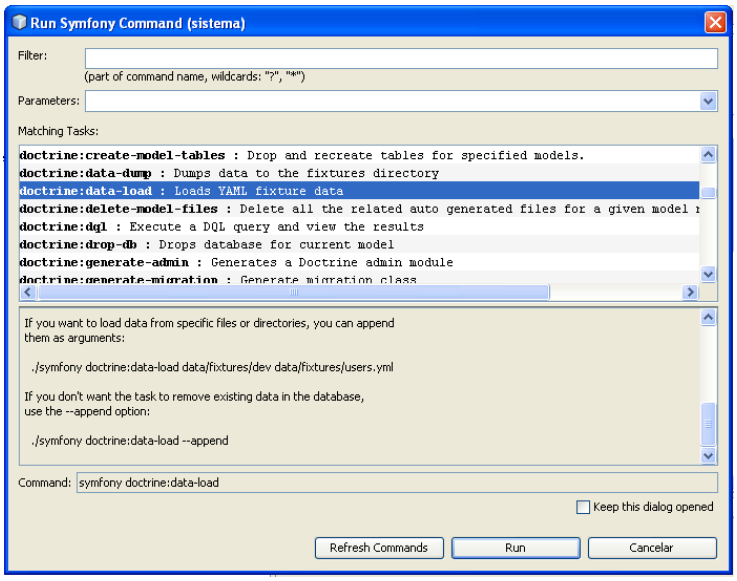


Figura 25.

El comando que se ejecuta, como se indica en la figura 25, es *doctrine:data-load,* el mismo que carga los datos del contenido de los archivos de este directorio.

SUBDIRECTORIO sql:

Este directorio contiene el archivo shema.sql, el mismo que se crea ejecutando el comando *doctrine:build –sql,* el mismo que ayudado con netbeans se ejecuta de la siguiente manera:

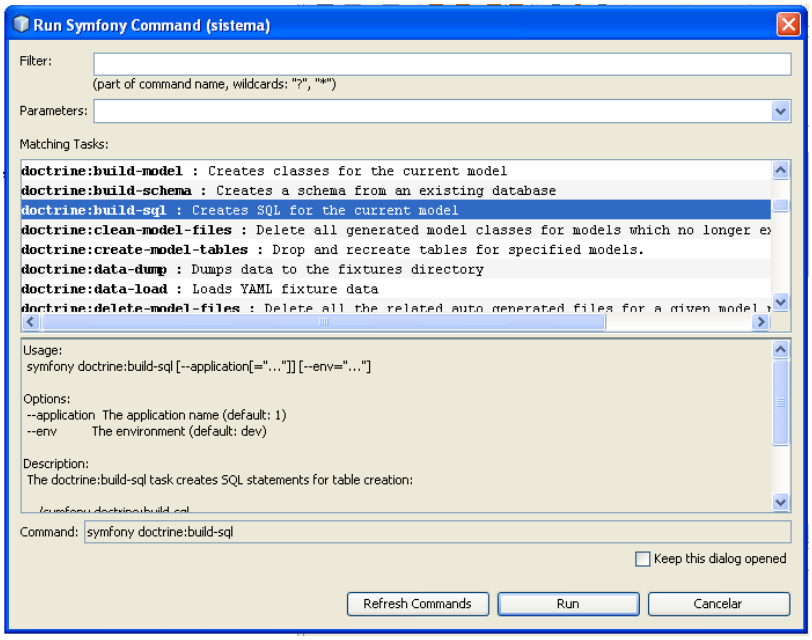


Figura 26.

Al ejecutar este comando como se indica en la ***figura 26*** genera comandos SQL en el directorio data/sql/, optimizado para el motor de base de datos que hemos configurado, en donde abriendo el archivo shema.sql, podemos observar código sql similar al siguiente:

CREATE TABLE barrios (id_barrio INT AUTO_INCREMENT, nombre VARCHAR(80), comentarios VARCHAR(250), PRIMARY KEY(id_barrio)) ENGINE = INNODB;

CREATE TABLE cantones (id_canton INT AUTO_INCREMENT, nombre VARCHAR(80), comentarios VARCHAR(250), PRIMARY KEY(id_canton)) ENGINE = INNODB;

ESTRUCTURA DEL DIRECTORIO lib DE LA APLICACIÓN

En este directorio se guardan las librerías y clases del proyecto, algunas son creadas automáticamente ejecutando los comando symfony, su contenido se lo muestra en la siguiente figura:

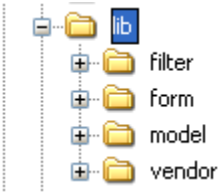


Figura 27

SUBDIRECTORIO filter:

En donde se van a crear los formularios que tienen relación con los filtros para búsquedas en la base de datos, estos directorios se crean en forma automática ejecutando el comando symfony doctrine:build-filters, como se indican en la figura 28a y su contenido creado en la figura 28b**.**

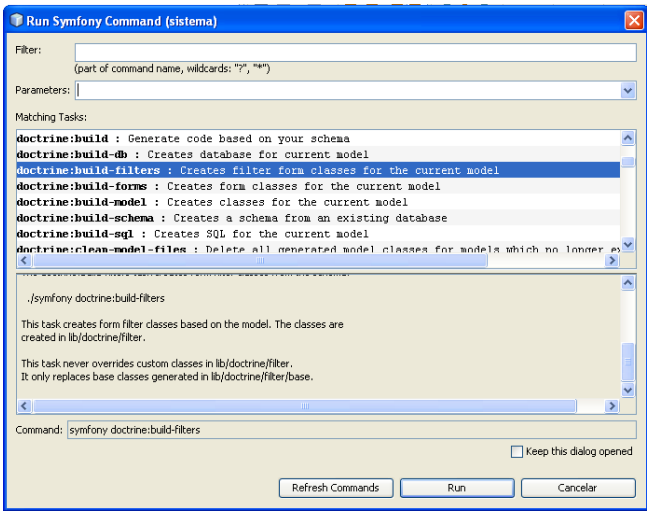


Figura 28a

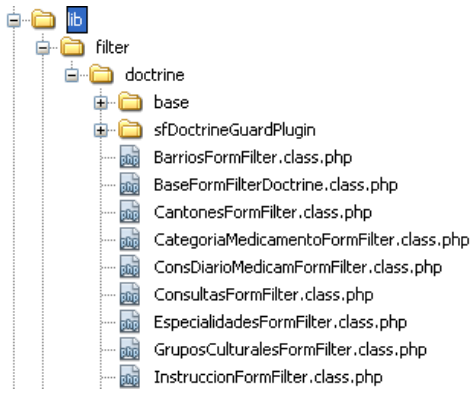


Figura 28b

SUBDIRECTORIO form:

Aquí es en donde se van a crear los formularios para ingreso y modificación de datos para el modelo actual de base de datos, estos directorios se crean en forma

automática ejecutando el comando symfony doctrine:build-forms, como se muestran en la figura 29a y su contenido creado en la figura 29b.

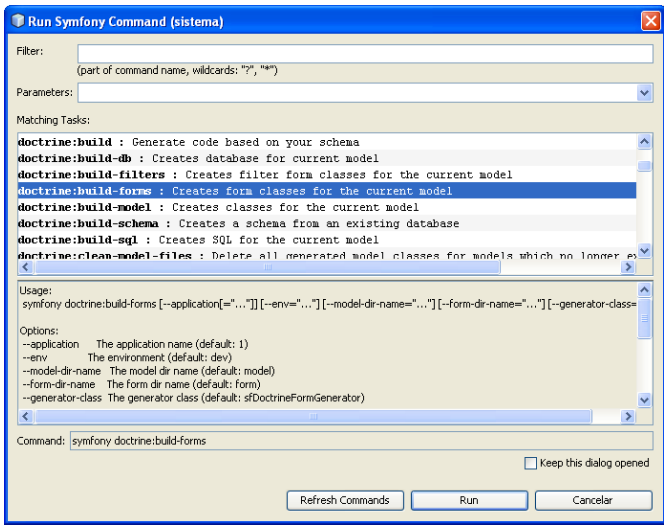


Figura 29a

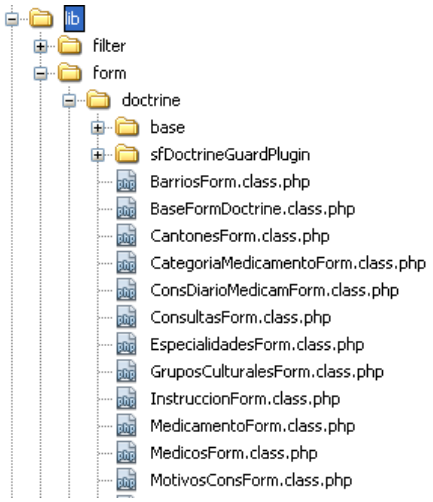


Figura 29b

SUBDIRECTORIO model:

Aquí es en donde se van a crear las clases para el modelo actual de base de datos, este directorio se crea en forma automática ejecutando el comando symfony doctrine:build-model, como se muestran en la figura 30a y su contenido creado en la figura 30b.

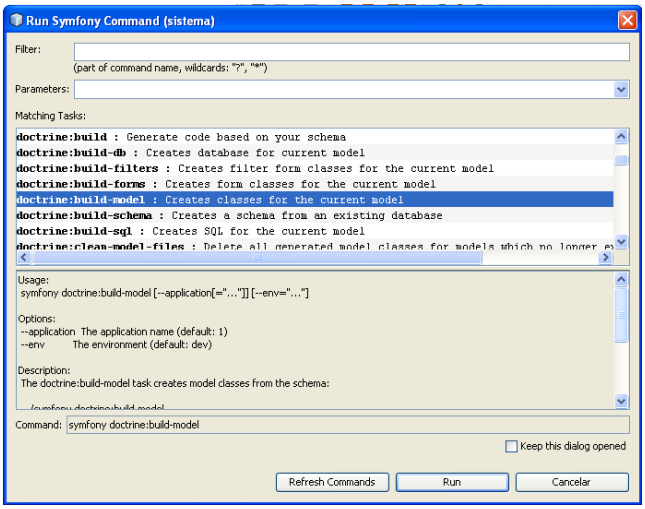


Figura 30a

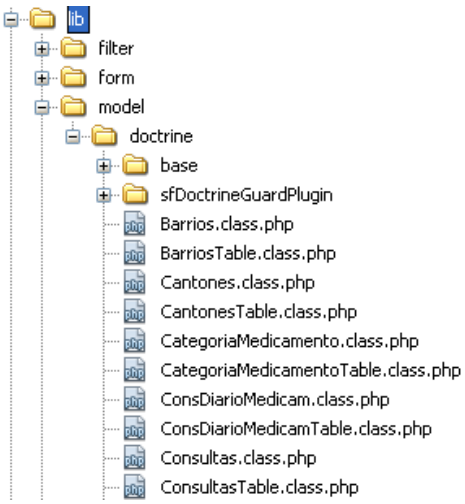


Figura 30b

SUBDIRECTORIO vendor:

Este directorio se crea manualmente, con el fin de guardar la carpeta núcleo de symfony (*www.symfony.es/*), el mismo que se baja desde internet, se desempaqueta y lo guardaremos en esta dirección, como se muestra en la figura 31.

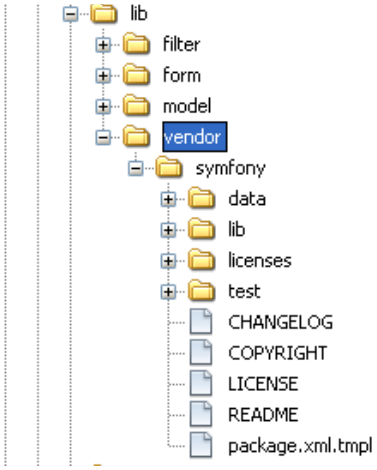


Figura 31

ESTRUCTURA DEL DIRECTORIO log DE LA APLICACIÓN

Además de los archivos de log creados por PHP, Symfony también guarda mucha información de sus propios eventos en otros archivos de log. Los archivos de log creados por Symfony se encuentran en el directorio miproyecto/log/. Symfony crea un archivo por cada aplicación y cada entorno, esto lo podemos visualizar en la **figura 32:**

Figura 32

Como vemos en la figura 32, se ha creado tres archivos .log backend_dev, frontend_dev e inicio_dev, pues estamos trabajando con el entorno de desarrollo,

si es que trabajaríamos en el entorno de producción sería, backend_prod, frontend_prod e inicio_prod.

Si se dispone de una aplicación Symfony ejecutándose, se puede observar que la sintaxis de los archivos de log generados es muy sencilla. Cada evento resulta en una nueva línea en el archivo de log de la aplicación. Cada línea incluye la fecha y hora a la que se ha producido, el tipo de evento, el objeto que ha sido procesado y otros detalles relevantes que dependen de cada tipo de evento y/o objeto procesado. En el siguiente listado mostramos el contenido de uno de este tipo de archivos (frontend_dev.log):

```
Jan 11 06:40:39 symfony [info] {sfPatternRouting} Match route "default_index" (/:module) for /nosotros with parameters array (  'module' => 'nosotros',  'action' => 'index',)
Jan 11 06:40:39 symfony [info] {sfFilterChain} Executing filter "sfRenderingFilter"
Jan 11 06:40:39 symfony [info] {sfFilterChain} Executing filter "sfExecutionFilter"
Jan 11 06:40:39 symfony [info] {NosotrosActions} Call "NosotrosActions->executeIndex()"
Jan 11 06:40:39 symfony [info] {sfPHPView} Render "C:\SubcentroSalud\sistema\apps\frontend\modules/nosotros/templates/indexSuccess.php"
Jan 11 06:40:39 symfony [info] {sfPHPView} Decorate content with "C:\SubcentroSalud\sistema\apps\frontend\templates/layout.php"
Jan 11 06:40:39 symfony [info] {sfPHPView} Render "C:\SubcentroSalud\sistema\apps\frontend\templates/layout.php"
Jan 11 06:40:39 symfony [info] {sfWebResponse} Send status "HTTP/1.1 200 OK"
Jan 11 06:40:39 symfony [info] {sfWebResponse} Send header "Content-Type: text/html; charset=utf-8"
Jan 11 06:40:39 symfony [info] {sfWebDebugLogger} Configuration 993.71 ms (9)
Jan 11 06:40:39 symfony [info] {sfWebDebugLogger} Factories 268.28 ms (1)
Jan 11 06:40:39 symfony [info] {sfWebDebugLogger} Action "nosotros/index" 0.14 ms (1)
Jan 11 06:40:39 symfony [info] {sfWebDebugLogger} View "Success" for "nosotros/index" 37.75 ms (1)
Jan 11 06:40:39 symfony [info] {sfWebResponse} Send content (30107 o)
Jan 11 06:40:39 symfony [err] {sfError404Exception} Empty module and/or action after parsing the URL "/engine/style.css" (/).
Jan 11 06:40:39 symfony [info] {sfWebResponse} Send status "HTTP/1.1 404 Not Found"
Jan 11 06:40:39 symfony [info] {sfWebResponse} Send header "Content-Type: text/html; charset=utf-8"
```

Para limpiar esta clase de archivos, lo podemos realizar con un comando symfony log:clear, que ayudado con symfony se vería como en la siguiente figura:

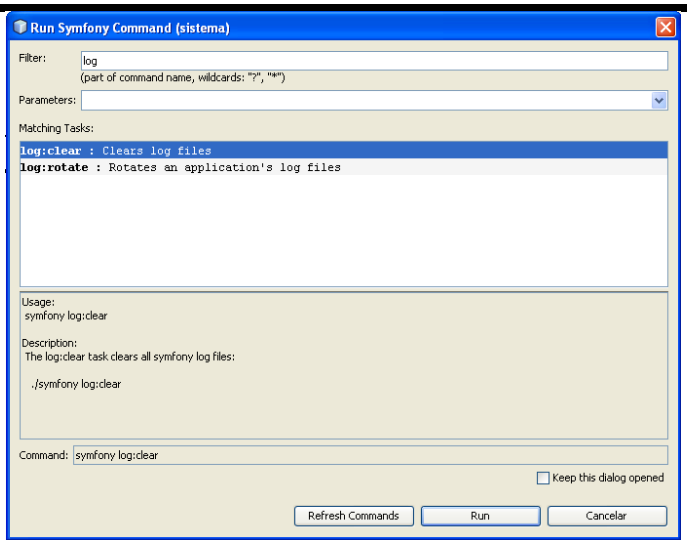


Figura 33

ESTRUCTURA DEL DIRECTORIO plugins DE LA APLICACIÓN

En ocasiones, es necesario reutilizar una porción de código desarrollada para alguna aplicación Symfony. Los plugins permiten agrupar todo el código diseminado por diferentes archivos y reutilizar este código en otros proyectos, estos plugins se encuentra en este directorio, como lo podemos observar en la siguiente figura:

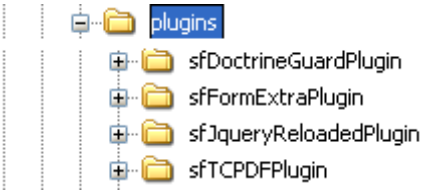


Figura 34

Los plugins son fáciles de instalar, de actualizar y de desinstalar. La forma en la que Symfony carga los plugins permite que los proyectos puedan utilizarlos como si fueran parte del propio framework.

Como podemos observar en la figura 34, aquí vemos los plugins que utiliza nuestro sistema:

sfDoctrineGuardPlugin: Es un plugin bajado de Internet, que lo utilizamos para asegurar nuestro proyecto, con el que podemos administrar los usuarios y permisos que intervienen en el sistema.

sfFormExtraPlugin: Utilizado para dar un aspecto más ordenado en la administración de nuestros módulos, es decir para las vistas.

sfJqueryReloadedPlugin: Este plugin se lo utilizó en nuestro proyecto para poder utilizar las ventajas de Jquery, entre ellas los calendarios para ingresar fechas (datepicker).

sfTCPDFPlugin: Utilizado para proveer herramientas con las que podemos crear informes en formato PDF.

ESTRUCTURA DEL DIRECTORIO test DE LA APLICACIÓN

Existen dos tipos de pruebas automáticas en symfony: **Pruebas Unitarias** y **Pruebas Funcionales**.

Las Pruebas Unitarias verificar que cada método y función está trabajando correctamente. Cada prueba deberá ser lo más independiente posible de las demás.

Por otro lado, Pruebas Funcionales verifican que la aplicación resultante se comporta correctamente en todo su conjunto.

Todas las pruebas en symfony están ubicadas bajo el directorio test/ del proyecto. Este tiene a su vez dos sub-directorios, uno para pruebas unitarias (test/unit/) y otro para las pruebas funtionales (test/functional/).

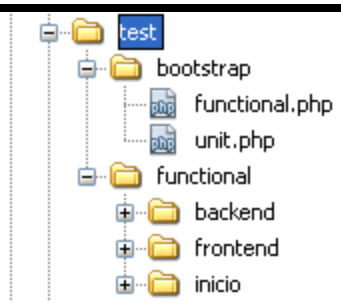


Figura 35

Las pruebas unitarias aseguran que un único componente de la aplicación produce una salida correcta para una determinada entrada. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular. Las pruebas unitarias se encargan de un único caso cada vez, lo que significa que un único método puede necesitar varias pruebas unitarias si su funcionamiento varía en función del contexto.

Las pruebas funcionales no solo validan la transformación de una entrada en una salida, sino que validan una característica completa. Un sistema de cache por ejemplo solamente puede ser validado por una prueba funcional, ya que comprende más de 1 solo paso: la primera vez que se solicita una página, se produce su código; la segunda vez, se obtiene directamente de la cache. De modo que las pruebas funcionales validan procesos y requieren de un escenario. En Symfony, se deberían crear pruebas funcionales para todas las acciones.

ESTRUCTURA DEL DIRECTORIO web DE LA APLICACIÒN

Existen pocas restricciones sobre la estructura del directorio web, que es el directorio que contiene los archivos que se pueden acceder de forma pública. Si se utilizan algunas convenciones básicas en los nombres de los subdirectorios, se pueden simplificar las plantillas. La siguiente es una estructura típica del directorio web:

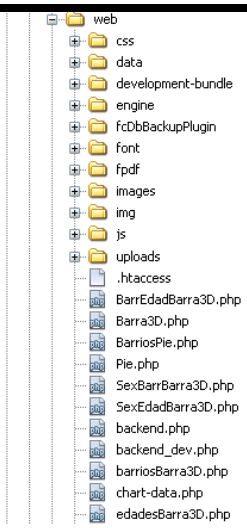


Figura 35

Normalmente, los archivos estáticos se organizan según los directorios de la siguiente tabla.

SUBDIRECTORIOS HABITUALES EN LA CARPETA WEB

Directorio	**Descripción**
css/	Contiene los archivos de hojas de estilo creados con CSS (archivos con extensión .css
images/	Contiene las imágenes del sitio con formato .jpg, .png o .gif
js/	Contiene los archivos de JavaScript con extensión .js
uploads/	Se almacenan los archivos subidos por los usuarios. Aunque normalmente este directorio contiene imágenes, no se debe confundir con el directorio que almacena las imágenes del sitio (images/). Esta distinción permite sincronizar los servidores de desarrollo y de producción sin afectar a las imágenes subidas por los usuarios

NOTA: Aunque es muy recomendable mantener la estructura definida por defecto, es posible modificarla para adaptarse a las necesidades específicas de cada proyecto, como por ejemplo los proyectos que se ejecutan en servidores con sus

propias estructuras de directorios definidas y con otras políticas para el desarrollo de las aplicaciones.

Como observamos en la **figura 35**, a más de los directorios explicados en nuestra tabla anterior, utilizamos otras, las cuales guardan archivos de plugins descargados y es aquí donde encontramos los archivos de arranque como es el Index.php o frontend.php, backend_.php, frontend_dev.php que sirven para ingresar al sistema sea en entorno de producción o de desarrollo.

El uso de un framework que utiliza MVC obliga a dividir y organizar el código de acuerdo a las convenciones establecidas por el framework. El código de la presentación se guarda en la vista, el código de manipulación de datos se guarda en el modelo y la lógica de procesamiento de las peticiones constituye el controlador. Aplicar el patrón MVC a una aplicación resulta bastante útil además de restrictivo.

Symfony es un framework de tipo MVC escrito en PHP 5. Su estructura interna se ha diseñado para obtener lo mejor del patrón MVC y la mayor facilidad de uso. Gracias a su versatilidad y sus posibilidades de configuración, Symfony es un framework adecuado para cualquier proyecto de aplicación web.

INDICACIONES ADICIONALES ACERCA DEL SISTEMA

1. Una de las muchas ventajas de Symfony es que utiliza "Mapeo de Objetos a Base de datos" (ORM). Los dos ORM para PHP más utilizados son sin lugar a dudas Doctrine y Propel.

   En nuestro sistema se ha utilizado Doctrine, aunque ambos ORMs tienen bastantes características básicas similares, ya que soportan cualquier operación usual en un CRUD (Create, Retrieve, Update and Delete), ya sea desde crear un nuevo registro o actualizar los registros existentes, pero symfony en la actualidad se está cambiando a doctrine como ORM por defecto.

2. Para crear nuestros módulos de administración de datos en su mayoría, lo creamos con el comando *doctrine:generate-admin*, que ayudado con symfony se lo puede crear con facilidad, solo hay que indicar el nombre de la aplicación y la tabla a la que quiere administrar (esto en la sección *parameters*), solo que esto genera un módulo de administración controlados por defecto, a los que podemos configurarlos y adaptarlos a nuestras necesidades.
3. Para el Generador de Admin, el módulo generado se puede configurar editando el archivo config/generator.yml que se encuentra en cada módulo creado con el comando anteriormente especificado.
4. La forma de ejecutar un comando de symfony es sencillo, solo hacemos clic derecho sobre el nombre de la aplicación en la que estamos trabajando y nos aparecerá una ventana como se indica en la **figura 36:**

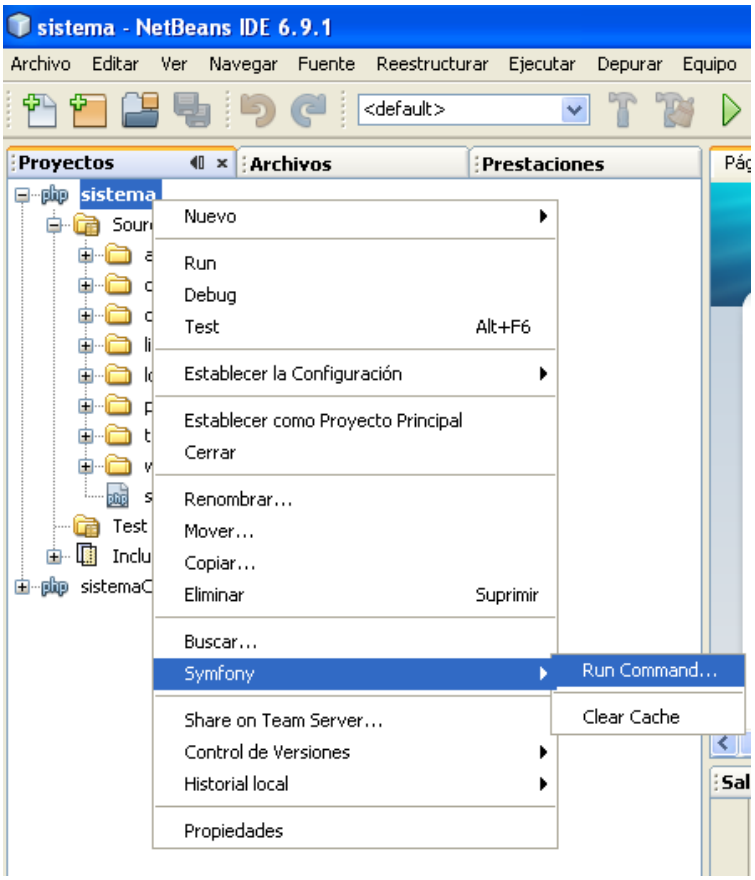


Figura 36

Por lo que hacemos clic izquierdo en la opción *Run Command…* del menú desplegable como se indica en la **figura 36**.

Para encontrar mayor información acerca del framework utilizado, y específicamente de admin generator de Doctrine, lo pueden encontrar en el capítulo 12 del libro “Practical Symfony” en español (Día 12), en la dirección web http://www.symfony-project.org/jobeet/1_2/Doctrine/es/.