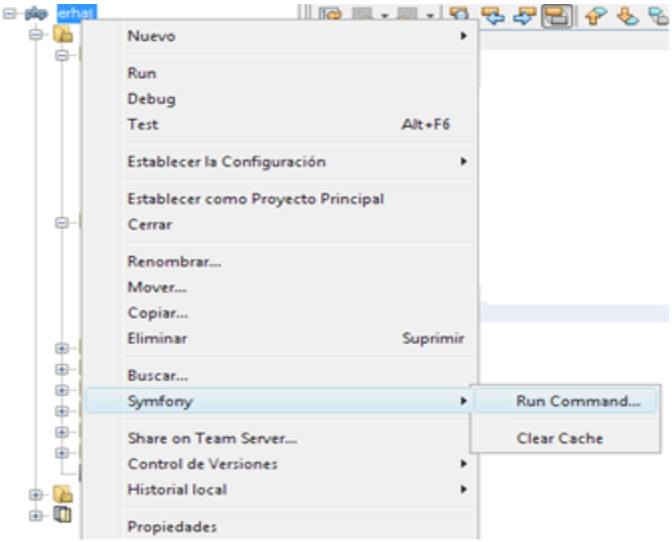
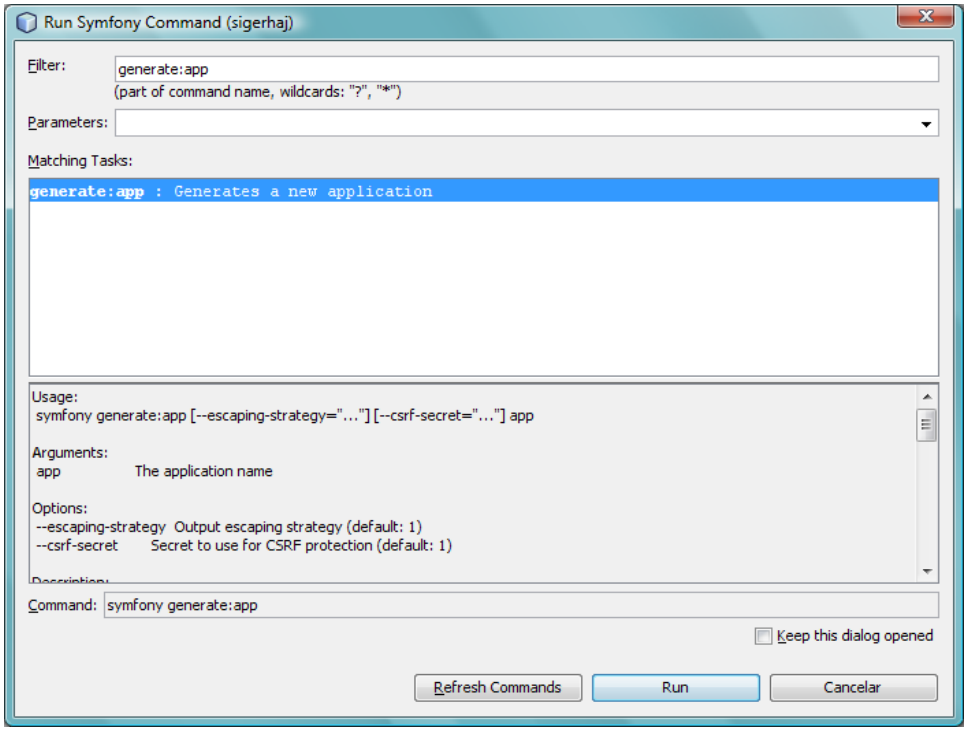
MANUAL TÉCNICO

Creado el proyecto podemos hacer uso de los comandos, para lo cual presionamos clic derecho sobre el proyecto, escogemos Symfony, seguidamente pulsamos sobre Run Command.



Esto abrirá una consola de mando con todo los comandos de Symfony .No tenemos que escribir el todo solamente seleccionamos el comando de la lista y le proporcionamos los parámetros requeridos que se nos indica en la parte de debajo de la ventana.



El comando ejecutado anteriormente, inicializa una aplicación. En el directorio web raíz del proyecto, se crean algunos archivos correspondientes a los controladores frontales de cada uno de los entornos por defecto: index.php, test.php (producción), test_dev.php(desarrollo).

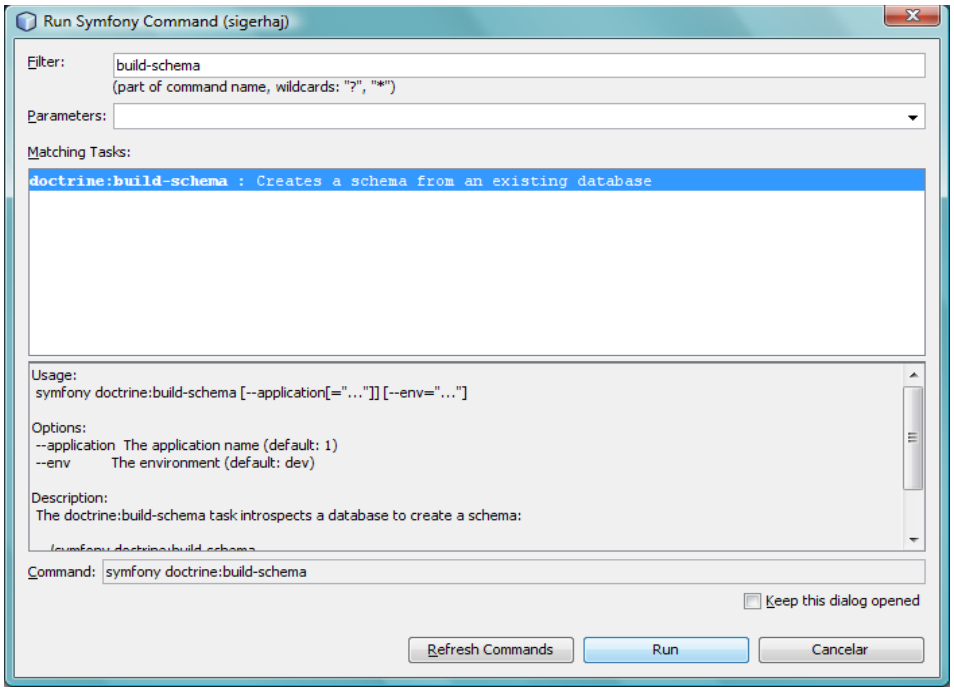
Generar módulos por default, es decir sin hacer referencia al modelo se realiza como se muestra en la imagen, para lo cual se utiliza el comando **generate:module** el cual recibe como parámetros <nombre de la aplicación > <nombre del módulo>

Para conectar bdd es necesario configurar manualmente el archivo databases.yml, en donde se coloca el nombre de la base de datos en este caso bdd_erhaj con su respectivo username y password.

```
all:
  doctrine:
    class: sfDoctrineDatabase
    param:
      dsn:      mysql:host=localhost;dbname= bdd_erhaj
      username: erhaj
      password: admin
```

Seguidamente se ejecuta el comando

.



Quién genera el archivo schema.yml para la representación de una base de datos existente.

Generada la base de datos es necesario representar las clases PHP del modelo, mediante el comando **build-model**, según el modelo de datos descrito en el archivo schema.yml, las clases base del modelo se crean en el directorio sigerhaja/lib/model/doctrine/base/ ejemplo: BaseRhPersonal.class.php

También se crean las clases de acceso a los datos en el directorio sigerhaja/lib/model/doctrine/
ejemplo RhPersonal.class.php y RhPersonalTable.class.php

Para generar los formularios relacionados con un objeto del modelo de datos, se emplea el comando **build-forms,** esta tarea crea una clase por cada tabla y le añade los validadores y widgets necesarios para cada columna teniendo en cuenta la información disponible en el modelo y la relación entre las tablas.

Para instalar un plugins se lo hace con el comando siguiente: **plugin:install**, que recibe como parámetro el nombre del plugin y para publicarlos se lo hace mediante la tarea: **plugin:publish-assets**
En la clase que se indica a continuación se coloca los plugins del proyecto.

```
class ProjectConfiguration extends sfProjectConfiguration
{
  public function setup()
        {
          $this->enablePlugins('sfDoctrinePlugin');
          $this->enablePlugins('sfTCPDFPlugin');
          $this->enablePlugins('sfProtoculousPlugin');
          $this->enablePlugins('sfJQueryUIPlugin');
        }
}
```

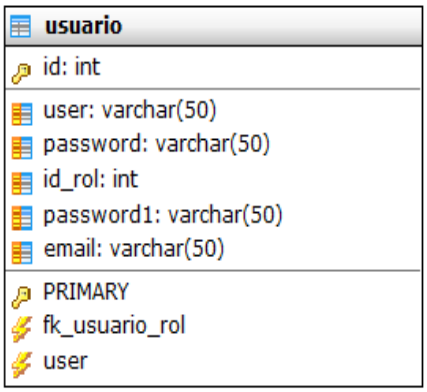
En symfony cada modulo creado tiene acciones y se le puede asignar un template. Para nombrar las acciones se antepone la palabra **execute** seguido del nombre de la acción en mayúsculas ejemplo: executeIndex.
Para asignar un template a la acción se coloca el nombre de la acción en minúsculas seguido de la palabra **Success** en la primera letra mayúscula como se indica en el ejemplo siguiente: listarSuccess.php

En la carpeta Config de la aplicación **test**, hay que configurar **routing.yml** con el fin de indicar a que página se direcciona al momento de ingresar a la aplicación

Creación Usuarios

La creación de usuarios y la asignación de roles es por parte del Administrador del Sistema.

Usuario: donde se encuentran todos los usuarios que se han creado por parte del Administrador del sistema al que le corresponde la tabla usuario que se compone de los campos que se muestran a continuación.



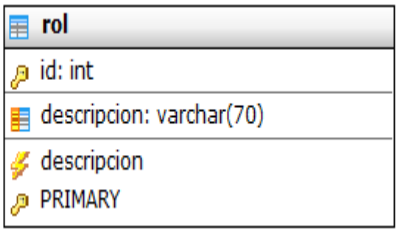
Id.- representa al identificador de la tabla

user.- Campo que permite almacenar el nombre de usuario (varchar).

password.- campo que almacena la clave o password del usuario que debe ser igual al valor almacenado en **Password1**. Cabe indicar que el valor password será encriptado al momento de ser almacenado para mayor seguridad,

email.- correo electrónico del usuario a registrarse, el cual debe ser único para cada usuario registrado.

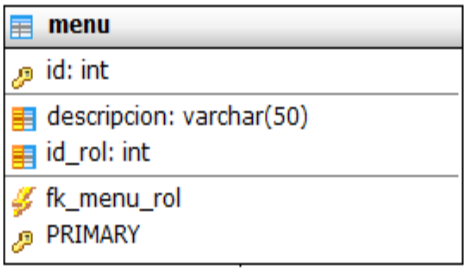
id_rol.- clave foránea (tabla rol)



Id.- representa al identificador de la tabla rol

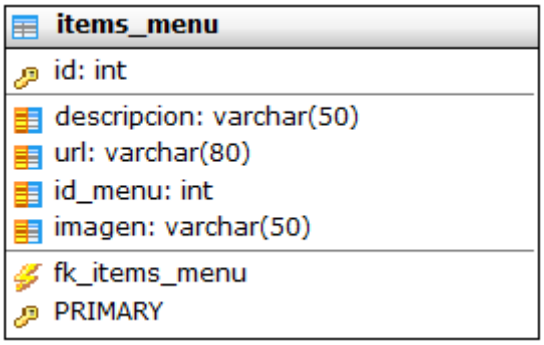
descripcion.- Campo que permite almacenar un varchar en el cual se colocará el nombre del rol que puede tener el usuario como puede ser: secretaria, administrador, analista, etc

Previamente al registro de un usuario ya se debe tener almacenado la información necesaria en la tabla rh_ítems menús, los cuales se desplegaran según el valor que se almacena en el campo id_rol de la tabla rh_menú.



Id.- representa al identificador de la tabla menú

descripcion.- campo de tipo varchar donde se almacena el nombre del menú el cual va ser asignado al campo **id_rol** que será escogido de la tabla rol.



Id.- representa al identificador de la tabla ítems menú

descripcion.- campo de tipo varchar donde se almacena el nombre del ítems menú el cual va ser visualizado por el usuario mediante la plantilla a la que es dirigido después de haberse autentificado.

url.- campo de tipo varchar en donde se almacén a la url o dirección que hace referencia al módulo

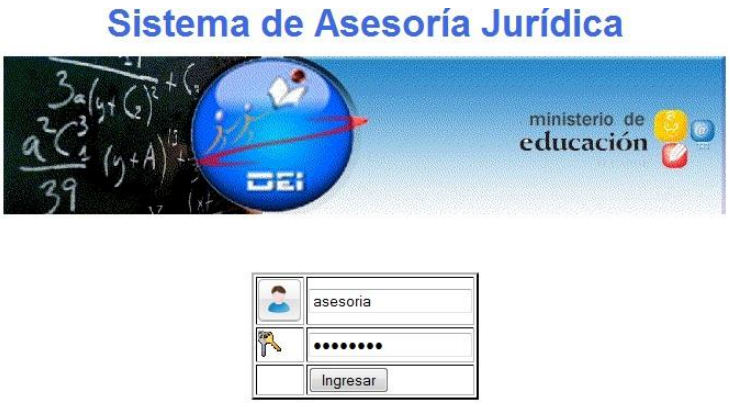
id_menu.- campo que almacena el numero de referencia de la tabla menú para el ítems menú correspondiente.

Imagen.- campo en el cual se almacena el nombre de la imagen que distingue al ítems menú de otro con su respectivo formato ejemplo: guardar.jpg

Módulo <login>

Creado con el fin de Autentificar usuarios, es decir encargado de verificar que tanto el usuario y password ya se encuentren registrados en la base de datos para lo cual del template **indexSuccess.php** se captura los valores de usuario y password ingresados por el usuario, seguidamente el controlador hace un llamado al modelo, haciendo uso de una consulta **verificarUsuario**($user, $password**)** en la que se envía los parámetros de ingreso pata ser comprobados con los existentes en los registros de usuarios.

Hay que tomar en cuenta que al momento de guardar un registro de usuario el password es encriptado, por tanto al momento de hacer la comparación del password ingresado con el existente en el registro, al ingresado por el usuario también hay que encriptar antes de dicha comparación.



Una vez autentificado el usuario se despliega las actividades que puede cumplir para lo cual según el rol se le asigna los privilegios previamente registrados.

Módulo <intro>

El cual se encarga de mostrar el menú que le corresponde al usuario que ha sido autentificado, haciendo uso del layout auxiliar.

Mediante esta línea de código **$usuario=$this->getUser()->getUsuario()**; conocemos el usuario que se encuentra autentificado, al cual se procede identificar el rol que le ha sido asignado con el fin de conocer el menú registrado para él, con

sus correspondientes ítems_menus, que mediante el uso de css, se presentan al usuario los cuales puede hacer uso con solo presionar sobre cada uno de los iconos que se presentan.

Módulo <usuario>

Acción New Para el registro de usuarios en la tabla Usuarios, módulo que verifica que el usuario no se encuentre en uso y que el password sea igual a password1 y el email no haya sido registrado por lo que se hizo uso del siguiente validador symfony:

```
$this->validatorSchema->setPostValidator(new sfValidatorAnd(array( new sfValidatorSchemaCompare('password1', sfValidatorSchemaCompare::EQUAL, 'password', array('throw_global_error' => true), array('invalid' => "Las dos contraseñas no coinciden")),new sfValidatorDoctrineUnique(array('model' => 'Usuario', 'column' => array('email')), array('invalid'=> "Este email ya está en uso")), new sfValidatorDoctrineUnique(array('model' => 'Usuario', 'column' => array('user')), array('invalid'=> "Este usuario ya está en uso")) ));
```

que se lo encuentra en la clase **UsuarioForm.class**.