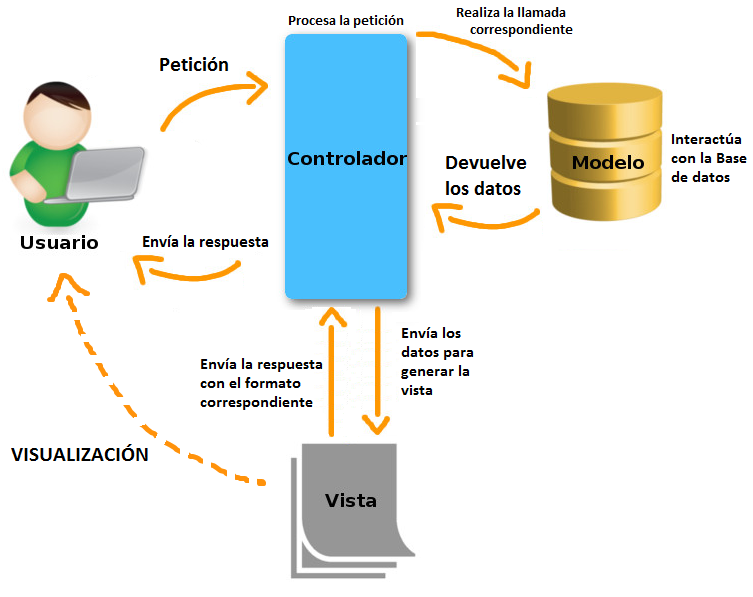
**Synfony**:

Synfony es un framework de PHP, extremadamente modular, para la construcción de aplicaciones web robustas.

Posee un conjunto de componentes PHP reutilizables.

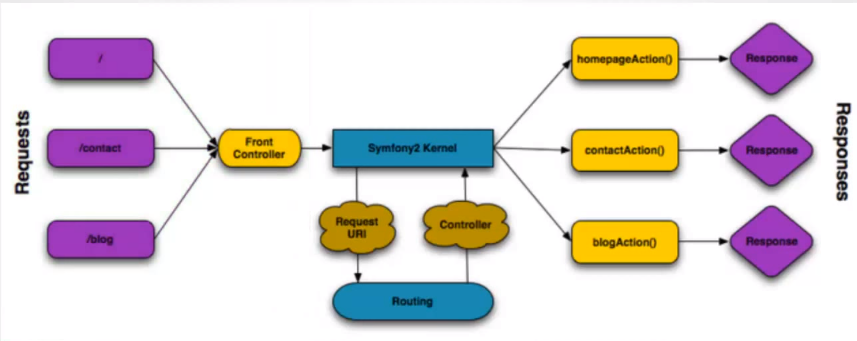
**MVC:**

Está basado en el modelo vista controlador:



**¿Cómo funciona?**

1. Partimos de una ruta con la petición y el “controlador frontal” (front-controller) maneja la solicitud.
2. El núcleo de Synfony (kernel) pide al router que inspeccione la solicitud.
3. El router devuelve la ruta especificadaal controlador.
4. El kernel devuelve un objeto “response”, en última instancia.



**Crear un proyecto en Synfony:**

* Instalamos *Composer*: <https://getcomposer.org/download/>
  1. Abrimos la consola de comandos en modo administrador, y nos situamos en la ruta deseada.
  2. Lanzamos las sentencias que se muestran a continuación:

php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash\_file('SHA384', 'composer-setup.php') === '544e09ee996cdf60ece3804abc52599c22b1f40f4323403c44d44fdfdd586475ca9813a858088ffbc1f233e9b180f061') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP\_EOL;"

php composer-setup.php

php -r "unlink('composer-setup.php');"

* Una vez finalizada dicha instalación automática, procedemos a crear el proyecto:
  1. Lanzamos la siguiente sentencia en la ruta del cmd donde queremos crear nuestro proyecto:
  2. Nos pedirá datos como:

php composer.phar create-project symfony/framework-standard-edition <directorio> 2.1.x-dev

* + - Port: <8000>
    - Database name: <nombre>
    - Etc.
  1. Si deseamos los datos que sugiere por defecto simplemente lo aceptamos con *intro*.
  2. Podemos configurar nuestro editor, y ponernos a trabajar en nuestro proyecto.

**Sistema de directorios:**

* **app**: contiene la configuración y la estructura del core del framework.
  + 🗁 **cache**: datos que se almacenan a través de los archivos que se van creando.
  + 🗁 **config**: contiene la configuración global de la aplicación:
    - * archivos **\*.yml**
  + 🗁 **logs**: logs de la aplicación según el entorno.
  + 🗁 **resources**: recursos reutilizables de la aplicación.
    - 🗁 views: vistas y plantillas globales de la aplicación.
  + 🗋 **Appkernel.php**: plugins y bundles de la aplicación.
  + 🗋 **Console**: automatiza las tareas de Synfony.
  + 🗋 **check.php**: para revisar que todo en el proyecto está bien.
* **bin**: donde se instala el urm que usa Synfony.
* **src**: contiene el desarrollo del proyecto. Es desde donde trabajaremos.
* **vendor**: donde se instalan los componentes de terceros.
* **web**: carpeta pública y visible de la aplicación.
  + 🗁 **bundles**: donde se almacenan los recursos de los bundles (paquetes) creados en la aplicación que van a ser públicos.
  + 🗋 **app.php** 🗋 **app-dev.php**: archivos relacionados con el front-controler, el cual consiste en un archive que se encarga de recivir las peticiones del cliente a la aplicación.

**Bundles:**

Son pequeños paquetes o pluguins que van a tener una única funcionalidad, lo que hace a Synfony completamente modular.

Cada uno va a tener su .php, su plantilla, su hoja de estilo, su JavaScript, sus archivos de prueba, etc.

Según creemos nuestros bundles, se registrarán automáticamente en “AppKernel.php” en la función de “registerBundles()” por lo que es importante definir y guardarlos en su lugar dentro de appKernel.

**Crear un Bundle:**

Debemos situarnos en el directorio raíz del proyecto mediante la consola ⇨ se deben usar comandos para crearlo, por lo que se requiere un terminal para lanzarlos:

php bin/console generate:bundle

* Se abrirá el generados de bundle, donde habrá unas especificaciones de cómo ir creando el main space de nuestro bundle.
* Indicaremos el nombre en formato “CamelCased” solo con caracteres alfanuméricos (para el namespace).
  + Siempre terminados con la terminación “Bundle” ⇨ <nombreBumdle>
* Cuando pida:
  + **name**: aceptaremos la sugerencia por defecto.
  + **directorio:** también aceptaremos la sugerencia por defecto.
  + **formato [yml, xml, php, anotation]:** indicaremos el que prefiramos ⇨ <yml>
* Generará automáticamente los archivos necesarios para el bundle, y finalizará.
* Si al finalizar muestra un mensaje tal que:

-The command was not able to configure everything atomatically.

You’ll need to make the following changes manually.

-Edit the composer.json file and register the bundle namespace in the “autoload” section:

Debemos actualizar dicha sección en el archivo “*<nombre proyecto>/composer.json*” :

(...)

“autoload”:{

“psr-4” {

**“AppBundle\\”:“src/AppBundle”,**

},

(...)

(...)

“autoload”:{

“psr-4” {

**“”: “src/”,**

},

(...)

Y desde la consola de comandos, situados en la carpeta del proyecto, lanzar la siguiente consulta para reiniciar el composer y que se detecten correctamente los cambios:

composer dump-autoload

**Rutas:**

Para poder configurar rutas debamos hacerlo desde el archivo *routing.yml* del directorio *<proyecto>/Resources/config/…* donde las almacenaremos.

Hay varias formas de definir una ruta, por anotaciones, formatos yml, xml o php, en nuestro caso usaremos el formato *yml*.

El archivo “*DefaultController.php*” del directorio *…/Controler/* de nuestro bundle es el que nos crea el framework por defecto. Además, tenemos una acción [*…Action()*] llamada *indexAction()* la cual está siendo ruteada desde nuestro archivo *routing.yml* del bundle.

**¿Cómo se configura una ruta?**

nombre\_ruta:

path: /controladorAlQueApuntamos/acciónALaQueApuntamos/{parámetro}

defaults: {\_controler:NombreBundle:Controlador<sinSufijos>:opción }

**Crear nueva ruta:**

1. Debemos crear otro archivo similar al de *DefaultControler.php* con su propio y único nombre, y en *routing.yml* definimos la ruta.
2. Primero debemos definir la acción ya que la ruta la empleará al activarse la ruta.
3. Debemos importar: *use Synfony\Component\HttpFoundation<componente>\Response<clase>;* dentro del controlador.

Si en la función de la acción ponemos:

⇨ *return new response(‘<respuesta>’);* ⇦

Devolverá dicha respuesta, siempre un String.

1. Si en *defaults* ponemos *{\_controler:NombreBundle:Controlador:accion, parámetro:valor}* el valor por defecto de dicho *parámetro* será el indicado salvo que en la ruta se indique lo contrario.
2. A continuación de

*defaults:…*

*[methods: POST … ]* método que va a usar nuestra ruta.

*requirements: …*

*parameter: \expresión regular para definir los valores del parámetro*

**Ver como se gestionan las rutas.**

Mediante comandos desde la consola

* php app/console debug route

Muestra el listado de las rutas con su nombre, método, scheme (esquema), host, path (ruta).

* php app/console debug route <nombre\_ruta>

Muestra los detalles de la ruta concreta.

**ORM de Symfony ⇨ Doctrine.**

Doctrine es el ORM (Object Relation Maper) que usa Symfony. Nos permite trabajar a través de una capa de abstracción con cada tabla de nuestra BBDD como objetos mediante sus entidades asociadas.

1. **Verificación de conexión a la BBDD:**

En el archivo *parameters.yml* del directorio */app/config* podemos observar los parámetros de la conexión a la base de datos que previamente hemos configurado.

1. **Crear una entidad (asociada a una tabla):**

Las entidades (objetos que representan a cada tabla de la base de datos) las podemos crear con comandos:

*php app/console doctrine:generate:entity*

1. Inicialmente pregunta por el bundle donde deseamos crear nuestra entidad.
2. Posteriormente nos pide que formato deseamos que use para su configuración, el sistema de anotaciones es muy cómodo.
3. A continuación nos pregunta por cada campo (o atributo) de nuestra entidad:
   1. Nombre.
   2. Tipo de dato.
   3. Longitud / valores / etc. Según cada tipo de dato.
4. Al terminar de indicar un atributo, nos pregunta si deseamos crear uno nuevo o si no queremos más campos, si no queremos más campos simplemente aceptamos con enter.
5. Pregunta si queremos generar un repositorio vacío, por lo general *yes* para facilitar el trabajo posterior.
6. Muestra un resumen y nos pide confirmación.

Ya podemos ver un nievo directorio dentro de nuestro Bundle llamado *Entity* donde se encuentran las entidades que hemos generado y los repositorios vacíos si los hemos pedido.

Hay que tener presente que con esto solo se crean las entidades, no se crea ninguna tabla. Éstas deberemos crearlas y asociarlas con otro comando.

1. **Editar la entidad:**

Desde el directorio del Bundle en el apartado de *Entity* podemos acceder al archivo php donde se enuentra la configuración de nuestra entidad.

Oservamos que en cada atributo están definida (mediante anotaciones en nuestro caso) la configuración de la misma tal y como hemos indicado por sonsola.

En la zona superior tendremos la definición general de la entidad:

*/\*\**

*\* <nombre\_entidad>*

*\* @ORM\Table(name=”<nombre\_tabla”)*

*…*

*\*\*/*

Hay algunos campos que la consola no detecta, por lo que deberemos modificarlos a mano, por ejemplo: ENUM

*/\*\**

*\*@var string*

*\*@ORM\Colum (name=”name\_colum”, type=”String, columDefinition=”ENUM(‘v1’, ’v2’)”, length=5)*

*\*\*/*

1. **Crear la tabla:**

También se hará mediante comandos:

*php app/console doctrine:schema:update* ***--force***

Fuerza la actualización del schema, el cual ejecuta una query que crea en la BBDD la tabla que se asociará a la entidad anteriormente creada.

* **Migraciones:** consultar la documentación oficial ya que hay varias formas de hacerlo según se desee migrar desde un sistema u otro. → <http://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html>

1. **Operaciones con BBDD:**

Las operaciones más comunes de Symfonypara bases de datos son:

*php bin/console doctrine:create*

*php bin/console doctrine:drop --force*

*php bin/console doctrine:create*

*php bin/console doctrine:update –force*

Las cuales crean una BBDD, la borran, crean una tabla y la actualizan respectivamente.

Doctrine emplea métodos que se encargan de la renderización de las consultas más habituales, éstos métodos se pueden consultar en detalle en la web oficial.

**Motor de plantillas ⇨ Twig:**

**Plantilla:** Se trata de un archivo de texto (html, xml, css, …). En nuestro caso se trata de un archivo de texto y código que analiza php, mediante los cuales definimos las vistas que mostraremos en la web.

Symfony tiene un motor de plantillas integrado llamado Twig, gracias al cual podemos dividir la lógica de la capa de presentación de la aplicación.

**Vistas:**

En el directorio del Bundle, en *Resources* encontramos el subdirectorio *views* ¿?¿?dentro del cual deberemos crear el subdirectorio donde almacenaremos nuestras vistas¿?¿?

**TWIG:**

Twig tiene 3 tipos de sintaxis:

* **{{ … }} :** sirve para devolver un valor o resultado (análogo al *echo* de php).
* **{% … %} :** sirve para controlar la lógica de la plantilla, y para trabajar con bucles (como con el *for*).
* **{# … #} :** sirve para incluir comentarios.