Unidad 7 - Introducción a Symfony

Presentación de Symfony

Definición de framework

Un framework, entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. En el caso del software, utilizar un framework permite agilizar los procesos de desarrollo ya que evita tener que escribir código de forma repetitiva, asegura unas buenas prácticas y la consistencia del código.
Un framework es por tanto un conjunto de herramientas y módulos que pueden ser reutilizados para varios proyectos.

Entre las ventajas de utilizar un framework para el desarrollo de software distinguimos:

- El programador ahorra tiempo ya que dispone ya del esqueleto sobre el que desarrollar una aplicación.
- Facilita los desarrollos colaborativos, al dejar definidos unos estándares de programación.
- Al estar ampliamente extendido, es más fácil encontrar herramientas, módulos e información para utilizarlo.
- Proporciona mayor seguridad, al tener gran parte de las potenciales vulnerabilidades resueltas.
- Normalmente existe una comunidad detrás, un conjunto de desarrolladores que pueden ayudar a responder consultas.

Symfony

Symfony es un framework para construir aplicaciones web con PHP. En otras palabras, Symfony es un enorme conjunto de herramientas y utilidades que simplifican el desarrollo de las aplicaciones web.

Symfony emplea el tradicional patrón de diseño MVC (modelo-vista-controlador) para separar las distintas partes que forman una aplicación web. El modelo representa la información con la que trabaja la aplicación y se encarga de acceder a los datos. La vista transforma la información obtenida por el modelo en las páginas web a las que acceden los usuarios. El controlador es el encargado de coordinar todos los demás elementos y transformar las peticiones del usuario en operaciones sobre el modelo y la vista.

El patrón MVC

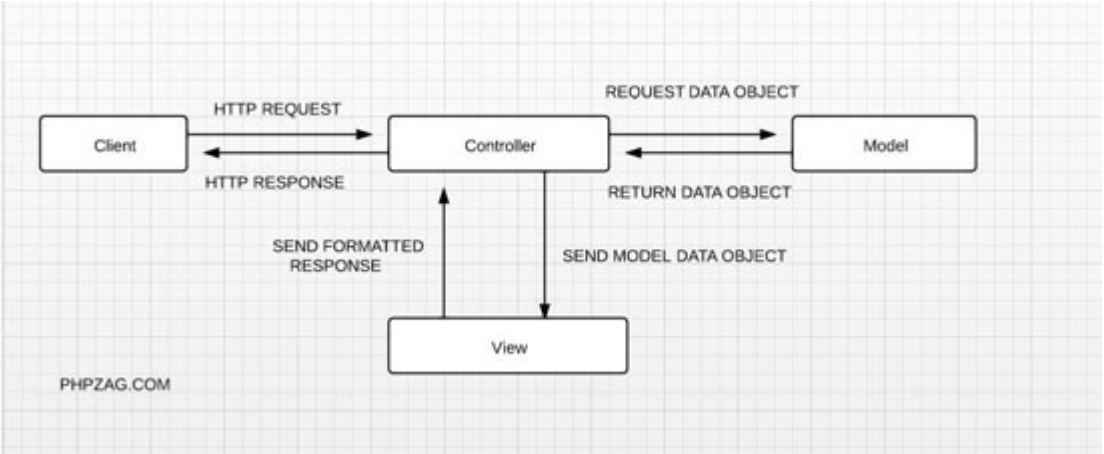
El patrón de diseño (de software) MVC se encarga de separar la lógica de negocio de la interfaz de usuario y es el más utilizado en aplicaciones web, framework, etc, ya que facilita la funcionalidad, mantenibilidad, y escalabilidad del sistema, de forma cómoda y sencilla, a la vez que ayuda no mezclar lenguajes de programación en el mismo código, el conocido “código espagueti”.

MVC divide las aplicaciones en tres niveles de abstracción:

1. Modelo: es la lógica de negocios. Es decir las clases y métodos que se comunican directamente con la base de datos.
2. Vista: es la encargada de mostrar la información al usuario, de forma gráfica y legible.
3. Controlador: el intermediario entre la vista y el modelo, se encarga de controlar las interacciones del usuario en la vista, pide los datos al modelo y los devuelve de nuevo a la vista para que esta los muestre al usuario.

El funcionamiento básico del patrón MVC, puede resumirse en:

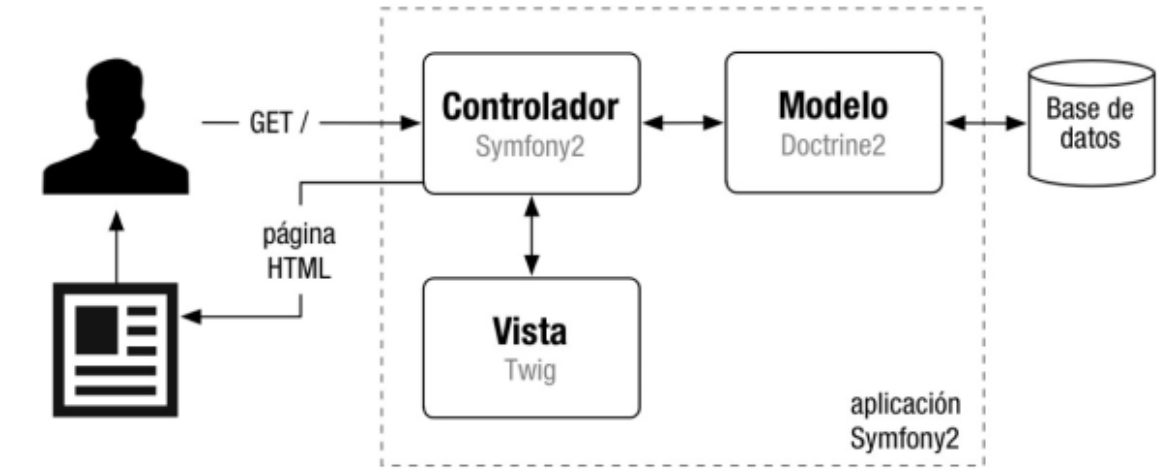
1. El usuario realiza una petición.
2. El controlador captura la petición.
3. Hace la llamada al modelo correspondiente.
4. El modelo será el encargado de interactuar con la base de datos.
5. El controlador recibe la información y la envía a la vista.
6. La vista muestra la información.



El patrón MVC en Symfony

Cuando el usuario solicita ver una determinada página, internamente sucede lo siguiente:

1. El sistema de enrutamiento determina qué Controlador está asociado con la página solicitada.
2. Symfony ejecuta el Controlador asociado a dicha página. Un controlador no es más que una clase PHP en la que podemos ejecutar cualquier código que queramos.
3. El Controlador solicita al Modelo los datos necesarios. El modelo no es más que una clase PHP especializada en obtener información, normalmente de una base de datos.
4. Con los datos devueltos por el Modelo, el Controlador solicita a la Vista que cree una página mediante una plantilla y que inserte los datos del Modelo.
5. El Controlador entrega al servidor la página creada por la Vista.



A pesar de que podemos llegar a hacer cosas muy complejas con Symfony, el funcionamiento interno siempre es el mismo:

1. El Controlador manda y ordena.
2. El Modelo busca la información que se le pide.
3. La Vista crea páginas con plantillas y datos.

Instalación de Symfony

Symfony se puede instalar de varias formas. Nosotros vamos a instalar Symfony via composer. Para ello desde la consola debemos ejecutar los siguientes comandos:

```
composer create-project symfony/skeleton:"^5.4" myapp
cd myapp
composer require symfony/webapp-pack
composer require symfony/apache-pack
composer require annotations
```

Con el primer comando le decimos a composer que queremos crear un proyecto symfony. Le indicamos que nos instale el “esqueleto de symfony” (o sea las librerías básicas), la versión de symfony que queremos usar (en este caso 5.4) y el nombre del directorio donde estará alojado el proyecto (en el común de los casos, el nombre del directorio coincide con el nombre del proyecto).

Una vez que composer nos instaló el esqueleto de symfony nos pasamos al directorio del proyecto (eso se hace con el segundo comando a ejecutar).

Con el tercer comando le estamos pidiendo a composer que nos instale todas las librerías necesarias para que nuestro proyecto sea una aplicación web.

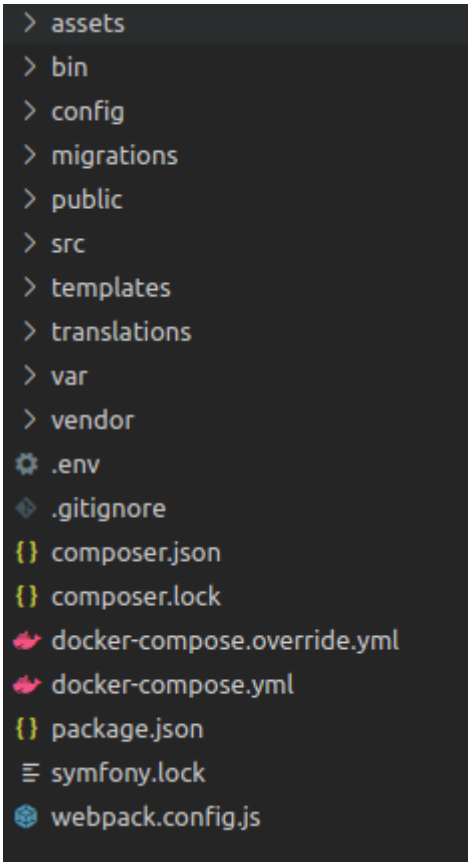
Con el último comando le pedimos a composer que nos instale los paquetes para que symfony funcione con el servidor web Apache.

A continuación podemos probar si la instalación se realizó en forma correcta. Abrimos el navegador y escribimos en la url:

```
http://localhost/myapp/public
```

Estructura de un proyecto Symfony

Después de instalar Symfony, vamos a dar un repaso a las distintas carpetas y ficheros que forman nuestro proyecto. Con el tiempo y conforme vayamos avanzando, aparecerán nuevo documentos y carpetas con los que iremos aumentando las funcionalidades del proyecto.



- **bin:** Esta carpeta contiene los ejecutables, entre ellos phpunit y el comando utilitarios que usaremos durante el proyecto.
- **config:** En esta carpeta encontraremos todos los ficheros de configuración.
- **migrations:** En esta carpeta se encuentran las migraciones. Son unos ficheros que nos ayudarán a realizar los cambios necesarios en las bases de datos, conforme vayamos evolucionando nuestro proyecto.
- **public:** Esta será la parte visible de nuestro proyecto y donde podrás encontrar css, js o imágenes.
- **src:** En esta carpeta estarán los ficheros que darán la funcionalidad a nuestro proyecto.
- **templates:** Sera utilizada para las plantillas twig de nuestro proyecto

- **test:** Aquí tenemos los test que realizaremos sobre nuestro Proyecto.
- **translations:** Esté es el lugar donde están los ficheros de traducción, que nos permiten la traducción de nuestra aplicación.
- **var:** En este lugar estarán cache y logs que nos permitirán agilizar y analizar nuestra aplicación.
- **vendor:** En esta carpeta se encuentran las librerías de terceros utilizadas por nuestra aplicación.

Creando la primera página

Creando el controlador

Crear una nueva página en Symfony es un sencillo proceso de dos pasos:

- Crea una ruta: Una ruta define la URL de tu página (por ejemplo /lucky) y especifica un controlador (el cual es una función PHP) que Symfony debe ejecutar cuando la URL de una petición entrante coincida con el patrón de la ruta;
- Crea un controlador: Un controlador es una función PHP que toma la petición entrante y la transforma en el objeto Response (Respuesta) de Symfony que es devuelto al usuario.

```
# src/Controller/LuckyController.php

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class LuckyController extends AbstractController
{
    /**
     * @Route("/lucky", name="app_lucky")
     */
    public function lucky(): Response
    {
        $number = random_int(0, 100);
        $html = '<html><body>El nro. de la suerte:'. $number .'</body></html>';
        return new Response($html);
    }
}
```

Se pueden crear varias rutas para un mismo controlador y cada ruta ejecuta la función correspondiente.

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class LuckyController extends AbstractController
{
    /**
    * @Route("/lucky", name="app_lucky")
    */
    public function lucky(): Response
    {
      $number = random_int(0, 100);
      $html = '<html><body>El nro. de la suerte: '. $number . '</body></html>';
      return new Response($html);
    }

    /**
    * @Route("/nolucky", name="app_no_lucky")
    */
    public function noLucky(): Response
    {
      return new Response('<html><body>No has tenido suerte</body></html>');
    }
}
```

Creando la vista

Podemos renderizar la salida de la página gracias al motor de plantillas Twig, incluído en el framework. Como primera medida creamos la plantilla twig

templates/lucky/lucky.html.twig

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>Bienvenido a Symfony</title>
    </head>
    <body>
        <h1>La suerte desde una plantilla </h1>
        <p>El nro. de la suerte: {{ numero }}</p>
    </body>
</html>
```

Luego modificamos el controlador para que devuelva el html generado con la plantilla:

```
# src/Controller/LuckyController.php
```

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class LuckyController extends AbstractController
{
    /**
     * @Route("/lucky", name="app_lucky")
     */
    public function lucky(): Response
    {
        $number = random_int(0, 100);

        return $this->render('lucky/lucky.html.twig', ['numero' => $number]);
    }
}
```

Resolviendo las rutas relativas

Las templates normalmente hacen referencia a imágenes, JavaScript, hojas de estilo y otros assets. Symfony proporciona una forma dinámica y flexible de cargarlos a través de la función asset de Twig:

```
<img src="{{ asset('images/logo.png') }}" alt="Symfony!" />

<link href="{{ asset('css/blog.css') }}" rel="stylesheet" />
```

El principal objetivo de la función asset es hacer la aplicación más portable. Si la aplicación está en el root del host (http://ejemplo.com), entonces los directorios deberían ser /images/logo.png. Pero si la aplicación está en un subdirectorio (http://example.com/my_app), cada directorio de los assets deberá renderizarse en el subdirectorio (/myapp/images/logo.png). La función asset se hace cargo de esto determinando cómo está construida y generando los directorios correctos.

```
# ./templates/base.html.twig
<!DOCTYPE html>
<html>
      <head>
            <meta charset="UTF-8">
            <title>My app</title>
            <link rel="stylesheet" href="{{ asset('css/estilos.css') }}">
      </head>
      <body>
            {% block body %}{% endblock %}
      </body>
</html>
```