
# Desarrollo de aplicaciones con Symfony 4

## Introducción

# ¿Por qué usar Symfony?

- Porque está basado buenas prácticas.
- Por sus diferentes entornos.
- Por su velocidad, gracias entre otras cosas a su sistema de cache.
- Por su sistema de enrutado.
- Por su documentación y comunidad.
- Porque cada vez se está extendiendo más a otros proyectos: Drupal 8, Joomla, phpBB, Prestashop, Magengo, ...
- Por la cantidad de bundles disponibles.

# ¿Quien usa Symfony?

Dos ejemplos con webs que alguna que otra visita

**YouPorn.com**

Uno de los 100 mayores sitios web del mundo, con más de 200 millones de páginas cada día

**socialpoint.es**

Utilizan Symfony en el backend. Algunos de sus juegos cuentan con más de 7 millones de jugadores

AvaiBook BOOKING

Symfony

# Conceptos generales

- Creado por Fabien Potencier.
- Symfony es patrocinado por Sensio Labs, una compañía francesa que provee consultoría, servicios, formación sobre tecnologías open source
- Basado en conceptos de otros frameworks.
- Uso de otros proyectos PHP como Doctrine, PHPUnit,…
  - Reutilización... ¿para qué reinventar la rueda?
- Versión mínima de PHP: 7.1.3
- La última versión a día de hoy es la 4.2 (la versión 4.3  sale este mismo mes)
- Diseñado usando buenas prácticas, y para que utilices buenas prácticas
  - Don’t be STUPID, grasp SOLID http://nikic.github.io/2011/12/27/Dont-be-STUPID-GRASP-SOLID.html

# Antes de empezar

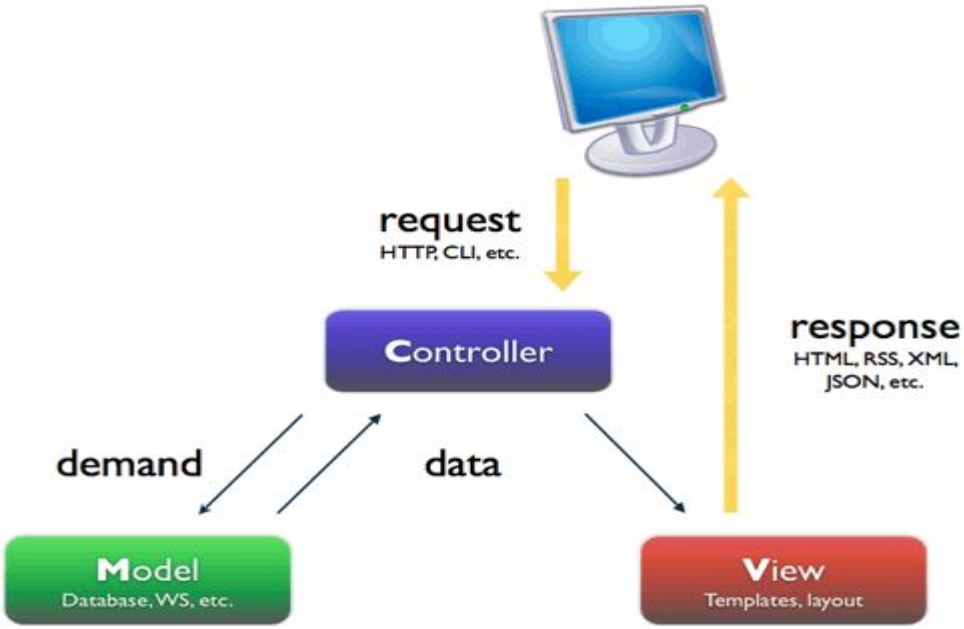
- MVC
- Namespaces
- Anotaciones
- YAML
- DOCTRINE
- TWIG
- Composer
- Git

AvaiBook BOOKING

Symfony

# MVC

- La capa del **modelo** define la lógica de negocio.
- La **vista** presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar.
- El **controlador** hace de intermediario entre la 'vista' y el 'modelo'.




AvaiBook BOOKING

Symfony

# Namespaces

Según la Wikipedia, un *namespace* es *"un contenedor abstracto que agrupa de forma lógica varios símbolos e identificadores*".

Ejemplo en Symfony:

```
<?php
namespace Avaiboook\PruebaBundle\Entity;
```

De esta forma, podemos tener dos clases Entity sin problema de colisión. Además se les puede asignar un alias:

```
<?php
use Avaiboook\PruebaBundle\Entity as EntityPrueba;
use Avaiboook\EntityBundle\Entity;
```

# Anotaciones

Las anotaciones son un mecanismo muy utilizado en lenguajes de programación como Java. Aunque PHP todavía no soporta anotaciones, las aplicaciones Symfony pueden hacer uso de ellas gracias a una librería desarrollada por el proyecto Doctrine2.

Técnicamente las anotaciones no son más que comentarios incluidos en el propio código fuente de la aplicación. A diferencia de los comentarios normales, las anotaciones no sólo no se ignoran, sino que se tienen en cuenta y pueden influir en la ejecución del código.

```
/**
 * @ORM\Column(type="string")
 * @Assert\NotBlank()
 */
protected $nombre;
```

# YAML

Los archivos de configuración de Symfony se pueden escribir en PHP, en XML o en YAML. Desde el punto de vista del rendimiento no hay ninguna diferencia entre los tres, ya que todos ellos se transforman a PHP antes de ejecutar la aplicación.

## Pero, ¿por qué utilizar YAML?

Muy simple, de los tres, YAML es probablemente el formato más equilibrado, conciso y fácil de entender o modificar.

# YAML

## ¡Veamos que es YAML!

Según el sitio web oficial de YAML (http://www.yaml.org/), YAML es "*un formato para serializar datos que es fácil de procesar por las máquinas, fácil de leer para las personas y fácil de interactuar con los lenguajes de script*".

Dicho de otra forma, YAML es un lenguaje muy sencillo que permite describir los datos como en XML, pero con una sintaxis mucho más sencilla. YAML es un formato especialmente útil para describir datos que pueden ser transformados en arrays simples y asociativos.

Veamos un ejemplo:

# YAML

```
$casa = array(
    'familia' => array(
        'apellido' => 'García',
        'padres' => array('Antonio', 'María'),
        'hijos' => array('Jose', 'Manuel', 'Carmen')
    ),
    'direccion' => array(
        'numero' => 34,
        'calle' => 'Gran Vía',
        'ciudad' => 'Cualquiera',
        'codigopostal' => '12345' )
    );
```

# YAML

```
casa:
   familia:
            apellido: García
      padres:
            - Antonio
            - María
      hijos:
            - Jose
            - Manuel
            - Carmen
      direccion:
            numero: 34
            calle: “Gran Vía”
            ciudad: Cualquiera
            codigopostal: “12345”
```

# YAML - Características

YAML utiliza espacios en blanco para indicar su estructura.

Los elementos que forman una secuencia utilizan un guión medio y los pares clave/valor de los array asociativos se separan con dos puntos.

YAML también dispone de una notación resumida para describir la misma estructura con menos líneas: los arrays simples se definen con[] y los arrays asociativos se definen con {}. Por tanto, los datos YAML anteriores se pueden escribir de forma abreviada de la siguiente manera:

```
casa:
  familia: { apellido: García, padres: [Antonio, María], hijos: [Jose, Manuel, Carmen] }
  direccion: { numero: 34, direccion: “Gran Vía”, ciudad: Cualquiera, codigopostal: "12345" }
```

# Doctrine

Según la Wikipedia, Doctrine es un mapeador de objetos-relacional (ORM) escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un SGBD.

Doctrine tiene influencias de docenas de proyectos de personas muy diferentes. Las mayores influencias son de Hibernate (el ORM de Java) y de ActiveRecord (de Ruby on Rails). Ambos tienen una implementación completa tanto en Java como en Ruby. El propósito de Doctrine es construir una solución igual de potente para PHP.

# TWIG

Twig es un motor y lenguaje de plantillas para PHP muy rápido y eficiente. Symfony recomienda utilizar Twig para crear todas las plantillas de la aplicación. No obstante, si lo prefieres puedes seguir escribiendo las plantillas con código PHP normal y corriente.

La sintaxis de Twig se ha diseñado para que las plantillas sean concisas y muy fáciles de leer y de escribir.

Esto en Twig:

```
{% if usuario is defined %}
    Hola {{ usuario.nombre }} hoy es {{ 'now' | date('d/m/Y') }}
{% endif %}
```

Veamos lo mismo en PHP:

# TWIG

```
<?php if (isset($usuario)): ?>
Hola <?php echo htmlspecialchars($usuario->getNombre(), ENT_QUOTES, 'UTF-8') ?>
hoy es <?php $hoy = new \DateTime(); echo $hoy->format('d/m/Y'); ?>
<?php endif; ?>
```

AvaiBook BOOKING

Symfony

# TWIG – Sintaxis básica

- {{ y }} para mostrar el valor de una variable.
- {% y %} para añadir lógica en la plantilla.
- {# y #} para incluir un comentario.
- Filtros: {{ articulo.titular | upper }}
- Anidar filtros: {{ articulo.titular | striptags | upper }}
- Variables: {% set variable = valor %}

AvaiBook BOOKING

Symfony

# Composer

Los proyectos PHP grandes, como por ejemplo las aplicaciones Symfony, dependen a su vez de muchos otros proyectos. Cuando envías por ejemplo un email, Symfony utiliza una librería externa llamada SwiftMailer. Así que para que tu aplicación funcione bien, Symfony necesita que todas esas librerías externas (llamadas *dependencias*) se instalen correctamente.

De la misma forma, si quieres actualizar Symfony a una nueva versión, es necesario comprobar todas y cada una de sus dependencias, por si también han sido actualizadas. Además, para cada una hay que comprobar si la nueva versión es compatible con las nuevas versiones del resto de dependencias.

Composer es la solución a todos estos problemas. **Composer comprueba la lista de dependencias de cada proyecto y decide qué librerías hay que instalar, qué versiones concretas se instalan y el orden correcto de instalación**.

# Symfony Flex

- No es una versión de symfony
- Reemplaza al instalador de symfony
- Es opcional

**¿Cómo funciona?**
Técnicamente es un plugin de composer que modifica el comportamiento de este, en concreto el comando de instalar y modificar paquetes.
Lo que hace es ponerse por delante para indicar si hay que hacer alguna acción adicional.

# Symfony Flex

Cuando ejecutamos, por ejemplo:

**$ composer require doctrine**

# Symfony Flex


AvaiBook BOOKING

Symfony

# Symfony Flex


AvaiBook BOOKING

Symfony

# Symfony

¿Empezamos
a crear
un proyecto?

AvaiBook
BOOKING

Symfony

# Crear el proyecto Symfony 4

En la actual versión cuatro de Symfony podemos instalarlo de dos maneras diferentes dependiendo del tipo de arquitectura que se necesite.

Opción 1: Si nuestro proyecto será una API REST o un Micro servicio:

$ **composer create-project symfony/skeleton mi-proyecto**

Opción 2: Si nuestro Proyecto será una aplicación con arquitectura monolítica debemos ejecutar.

$ c**omposer create-project symfony/website-skeleton mi-proyecto**

La diferencia de ambas instalaciones se encuentra en el número de librerías que se instalarán, mientras la primera opción instala las mínimas dependencias para que funcione Symfony la segunda opción instala todas las librerías más importantes como twig o doctrine.

Ver: https://symfony.com/doc/current/setup.html

AvaiBook BOOKING

Symfony

# El server de Symfony 4

Puedes ejecutar aplicaciones Symfony con cualquier servidor web (Apache, nginx, el servidor web interno de PHP, etc.). Sin embargo, Symfony proporciona su propio servidor web para que sea más productivo mientras desarrolla sus aplicaciones.

Para ello simplemente ejecuta dentro de tu proyecto el comando

```
$ symfony server:start
```

Para saber más: https://symfony.com/doc/current/setup/symfony_server.html

# Estructura de directorios

bin

Contiene los archivos ejecutables como console o phpunit.

config/

Contiene todos los archivos de configuración de la aplicación.

public/

Contiene el controlador frontal y los assets de nuestra aplicación (librerias js, css, imágenes)

src/

Contiene nuestro código.

# Estructura de directorios

templates/

Contiene las plantillas de nuestra aplicación.

tests/

Contiene los test de nuestra aplicación.

translations/

Contiene los archivos con las traducciones de nuestra aplicación

var/

Contiene la cache y los logs.

# Estructura de directorios

vendor/

Contiene las libreréas de terceros.

¡¡¡Veamos en phpStorm más detalladamente la estructura y que contiene cada una.!!!

# Symfony maker

En symfony.com lo definen así:
Symfony Maker te ayuda a crear comandos, controladores, clases de formularios, tests y más para que puedas olvidarte de escribir el código repetitivo.

Podemos saber que nos ofrece ejecutando:

**$ php bin/console list make**

y si necesitamos información adicional de algúncomando:

**$ php bin/console make:controller --help**

y también podemos crear nuestros propios makers, simplemente crear una clase que extienda de AbstractMaker y ponerla en el  directorio src/Maker/

# Entornos de trabajo

Una aplicación symfony puede funcionar en diversos entornos. Los diferentes entornos comparten el mismo código PHP, pero usan una configuración diferente.

Normalmente trabajaremos con 3 entornos (prod, dev y test) aunque podemos crear tantos entornos como queramos.

# Entornos de trabajo

**¿Como se definen?**

En el raiz de nuestra aplicación veremos un archivo .env, es aquí donde se definen ciertos parámetros de configuración como los datos de acceso a la base de datos, la configuración del SMTP, etc…

Si entramos, veremos el parámetro:

APP_ENV=dev

Que indica que estamos trabajando en el entorno de desarrollo, lo que supone que tomará la configuración de config/packages/dev, config/routing/dev, etc…

Para cambiar a producción, simplemente modificamos el valor del parámetro:

APP_ENV=prod

# Entornos de trabajo

**¿Y como hacemos para tener diferentes parámetros de .env según el entorno?**

.env  # Siempre se cargará

.env.local  # sobreescribirá los parámetros de .env y esta fuera del control de versiones

.env.{enviroment} # Se cargará según el valor de APP_ENV en .env

.env.{enviroment} # Se cargará según el valor de APP_ENV en .env y está fuera del conctol de versiones.

# Bibliografía

- ¡¡¡¡¡¡ RTFM !!!!!! http://symfony.com/
- https://www.youtube.com/watch?v=wMpqXF2uGGA

AvaiBook BOOKING

Symfony

¿Preguntas?