

Symfony

*Desarrollo de aplicaciones MVC en el servidor con un
framework PHP*

1. Introducción a los frameworks PHP y a Symfony

Ignacio Iborra Baeza
Julio Martínez Lucas

Índice de contenidos

Symfony.....	1
1.Frameworks PHP.....	3
1.1.Ejemplos de frameworks PHP.....	3
1.2.¿Cuál elegir?.....	4
1.2.1.¿Por qué Symfony?.....	4
2.Recursos previos.....	6
2.1.Instalar Apache, MySQL y PHP.....	6
2.1.1.El manager de XAMPP.....	7
2.1.2.Creando un acceso directo para el manager en el escritorio.....	7
2.2.Instalando Symfony a través de Composer.....	8
2.3.El IDE para desarrollar los proyectos.....	9
3.Primeros pasos con Symfony.....	10
3.1.Nuestro primer proyecto Symfony.....	10
3.1.1.Estructura general de un proyecto Symfony.....	10
3.1.2.Probando nuestro proyecto.....	12
3.2.Distribuir nuestros proyectos Symfony.....	13
3.3.La consola de Symfony.....	14
4.Introducción a YAML.....	15
5.Ejercicios.....	16
5.1.Ejercicio 1.....	16
5.2.Ejercicio 2.....	16

1. Frameworks PHP

Un **framework** es una herramienta que proporciona una serie de módulos que ayudan a organizar y desarrollar un producto software. En el caso concreto de los frameworks PHP, la mayoría de ellos proporcionan una serie de comandos o herramientas para crear proyectos con una estructura determinada (normalmente, siguiendo el patrón MVC que veremos después), de forma que ya dan una base de trabajo hecha, y facilidades para poder crear el modelo de datos, la conexión a la base de datos, las rutas de las diferentes secciones de la aplicación, etc.

1.1. Ejemplos de frameworks PHP

Actualmente existe una gran variedad de frameworks PHP que elegir para desarrollar nuestras aplicaciones. Algunos de los más populares son:

- **Laravel**, un framework relativamente reciente (fue creado en 2011), y que ha ganado bastante popularidad en los últimos años. Su filosofía es el poder desarrollar proyectos de forma elegante y simple. Cuenta con una amplia comunidad de soporte detrás, y se le augura un futuro bastante consolidado.
- **Symfony**, el framework que emplearemos en este curso. Creado en 2005, cuenta con más camino hecho que Laravel, y una estructura más consolidada. En sus primeras versiones se presentaba como un framework más monolítico (se instalaban demasiados módulos que luego no necesitábamos), pero recientemente ha adaptado su estructura para hacerla más modular. Está muy orientado al patrón MVC, y a una estructura rígida y correcta de hacer aplicaciones.
- **CodeIgniter**, un framework más ligero que los anteriores, pero también con un amplio grupo de seguidores y desarrolladores. Fue creado en 2006 y, aunque ha sufrido una etapa de abandono, ha vuelto a coger fuerza en los últimos años, quizá debido a su simplicidad de uso.
- **Phalcon**, otro framework de reciente creación (2012), con una potente capacidad de procesamiento de páginas PHP, y la posibilidad de trabajar como *microframework* (más ligero, para ofrecer funcionalidades muy específicas) o como framework completo. De hecho, muchos frameworks más antiguos también han incorporado recientemente la posibilidad de ejecutarlos como *microframeworks*.
- **CakePHP**, creado en 2005, es otro framework similar a CodeIgniter en cuanto a simplicidad y facilidad de uso. Tiene una amplia comunidad también detrás que le da soporte.
- **Zend**, creado en 2006, es otro framework bastante popular, aunque quizá con menor visibilidad que los anteriores hoy en día.
- ... etc.

Casi todos los frameworks PHP tienen una serie de características comunes, como son el uso del patrón MVC para desarrollar sus proyectos, la inyección de dependencias para gestionar recursos tales como conexiones a bases de datos, o elementos compartidos por

toda la aplicación, la posibilidad de desarrollar tanto webs completas como servicios REST accesibles desde diversos clientes, etc.

1.2. ¿Cuál elegir?

A la hora de decantarnos por uno u otro framework, no nos deberíamos dejar engañar por la popularidad del mismo, en términos de cuota de mercado. En ese terreno, Symfony y Laravel probablemente sean los más beneficiados, pero la curva de aprendizaje en ellos puede que sea más pronunciada que en otros a priori más sencillos, como CodeIgniter o CakePHP.

Cada framework puede estar mejor orientado que otro para determinados tipos de proyectos o necesidades. Si queremos aprender algo rápido para lanzar la aplicación cuanto antes mejor, quizá Symfony no sea la mejor opción. Si, por el contrario, preferimos empaparnos de un framework con una comunidad importante detrás que nos pueda dar soporte y nos garantice un tiempo de vida largo, entonces Symfony o Laravel pueden ser mejores candidatos.

1.2.1. ¿Por qué Symfony?

Llegados a este punto... ¿qué características tiene Symfony que nos hayan hecho elegirlo para este curso frente a otros frameworks? El principal inconveniente que puede tener este framework es, quizá, su curva de aprendizaje, que es bastante elevada comparada con otros. Sin embargo, las ventajas que ofrece compensan este inconveniente:

- Es un framework con mucho recorrido (creado en 2005)
- Desarrolla versiones LTS (*Long Term Support*) que garantizan un soporte a largo plazo
- Tiene una gran comunidad detrás, lo que permite encontrar fácilmente ayuda para problemas que tengamos
- Tiene una buena documentación. De hecho, en la propia página de Symfony se tienen disponibles algunos libros editados por el propio equipo de desarrollo, y una [web](#) con la documentación oficial de la actual versión
- Utiliza (y anima a utilizar) buenas prácticas de programación en los proyectos
- Se enlaza bien con otros frameworks y librerías de terceros muy útiles, denominadas *bundles*. Algunos ejemplos que utilizaremos en el curso son Twig (un potente motor de plantillas para desarrollar nuestras vistas) o Doctrine (un ORM para poder mapear los datos en objetos).

En realidad, una vez se conoce uno de estos frameworks, es más sencillo asimilar el resto, llegado el momento. Así que Symfony puede ser un buen punto de partida. En concreto, durante el curso utilizaremos la versión 4 del framework, que se apoya en PHP 7 para funcionar, como veremos a continuación.

Symfony is a set of reusable *PHP components...*

The standard foundation on which the best PHP applications are built. Choose any of the 50 stand-alone components available for your own applications.

[Browse components →](#)



... and a *PHP framework* for web projects

Speed up the creation and maintenance of your PHP web applications. End repetitive coding tasks and enjoy the power of controlling your code.

[What is Symfony →](#)

2. Recursos previos

A la hora de trabajar con Symfony (versión 4), necesitamos tener previamente instalados en nuestro sistema una serie de recursos software, como son:

- Un servidor web que soporte PHP 7.1.3 o posterior. En nuestro caso, utilizaremos Apache.
- Un servidor de bases de datos en el que almacenar la información de nuestras aplicaciones. Emplearemos un servidor MariaDB/MySQL.
- PHP (versión 7.1.3 o posterior)
- El propio framework Symfony.
- Un IDE (entorno de desarrollo) con el que editar el código de nuestros proyectos

2.1. Instalar Apache, MySQL y PHP

Para cumplir con los tres primeros requisitos (Apache, MariaDB/MySQL y PHP) vamos a instalar un sistema XAMPP, desde su [web oficial](#). En el caso de la máquina virtual que tenemos disponible para el curso, podemos instalarlo descargando la versión para Linux.



The screenshot shows the XAMPP website with the title "XAMPP Apache + MariaDB + PHP + Perl". Under the heading "¿Qué es XAMPP?", it states that XAMPP is the most popular development environment with PHP, a free distribution of Apache containing MariaDB, PHP, and Perl. At the bottom, there are four download buttons: "Descargar" (with a link to other versions), "XAMPP para Windows 7.2.7 (PHP 7.2.7)", "XAMPP para Linux 7.2.7 (PHP 7.2.7)" (highlighted with a red box), and "XAMPP para OS X XAMPP-VM (PHP 7.2.7)".

Se trata de un archivo `.run`. Primero debemos hacerlo ejecutable, escribiendo este comando desde la carpeta donde lo hayamos descargado (por ejemplo, para la versión 7.2.7 de XAMPP, que es la que está disponible en el momento de escribir estos apuntes):

```
chmod +x xampp-linux-x64-7.2.7-0-installer.run
```

Después, lo ejecutamos con permisos de `root`:

```
sudo ./xampp-linux-x64-7.2.7-0-installer.run
```

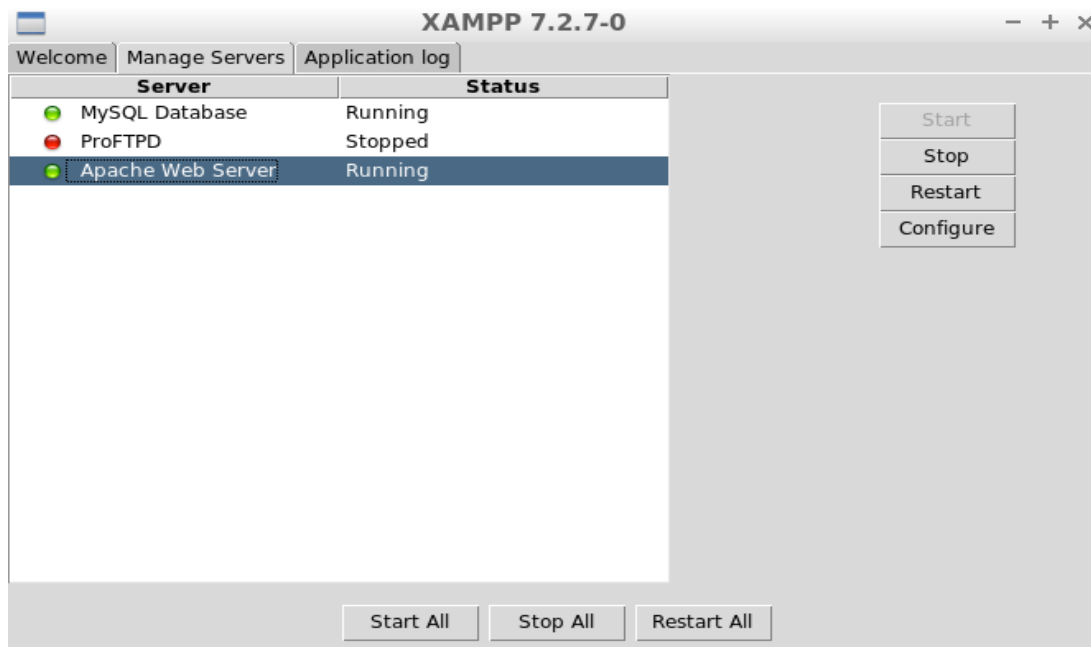
Se iniciará un asistente que nos guiará en la instalación. Podemos dejar todas las opciones marcadas por defecto, salvo la opción sobre Bitnami, que podemos desmarcar ya que no lo utilizaremos (es una especie de asistente que permite instalar otros paquetes como Wordpress, Joomla, Moodle... en nuestros proyectos web).

Recuerda, no obstante, que dispones de una versión de la máquina virtual con todo el software necesario instalado, con lo que puedes omitir este paso si no quieres perder tiempo con él.

2.1.1. El manager de XAMPP

Haremos uso del *manager* que se habrá instalado para poner en marcha los servidores. Debemos buscar el manager en cuestión en la carpeta donde se haya instalado XAMPP. En el caso de Linux, el manager se encuentra en `/opt/lampp/manager-linux-x64.run`, que podemos ejecutar directamente desde un terminal, con permisos de administrador. Si accedemos a la carpeta donde está, el comando quedaría así:

```
sudo ./manager-linux-x64.run
```



2.1.2. Creando un acceso directo para el manager en el escritorio

Como puede ser algo incómodo ir a la carpeta de XAMPP para poner en marcha el *manager* cada vez, podemos crear un acceso directo en el escritorio. Para Linux, existen varias formas de hacerlo, veamos una: creamos un archivo en el escritorio (lo podemos llamar, por ejemplo "xampp.desktop"), y lo editamos con un editor de texto, dejándolo así:

```
[Desktop Entry]
Encoding=UTF-8
Name=XAMPP Manager
Comment=Manager XAMPP
Exec=gksudo /opt/lampp/manager-linux-x64.run
```

```
Icon=/opt/lampp/htdocs/favicon.ico
Categories=Aplicaciones;Programación;Web
Version=7.2.7
Type=Application
Terminal=0
```

Lo que hemos hecho es definir varios parámetros de configuración del acceso directo. Entre ellos, los más destacados son la ruta hacia el ejecutable (parámetro *Exec*) y el icono para el acceso directo (parámetro *Icon*, donde hemos elegido directamente un icono que ya viene con XAMPP).

Después de esto, hacemos que el archivo "xampp.desktop" sea ejecutable (con clic derecho, en sus propiedades, podemos hacerlo), y lo ejecutamos. Puede que nos pida la contraseña de "alumno" para lanzarlo, y después ya podremos ver el *manager*.

2.2. Instalando Symfony a través de Composer

El último requisito para empezar a trabajar con Symfony es, obviamente, disponer del framework Symfony. Para esto, la propia web ofrece algunas alternativas, pero recomienda instalar Symfony a través de la herramienta Composer, que es un gestor de dependencias que permite instalar distintos módulos o librerías en un proyecto. Contiene una base de datos online con muchas librerías disponibles centralizadas, con lo que podemos indicar cuál(es) queremos para cada proyecto concreto, y Composer las descarga e instala por nosotros. Es algo muy similar a otros gestores como NPM (*Node Package Manager*), que se aplica a librerías para Node o Javascript en general.

Composer puede instalarse localmente para cada proyecto web, o de forma global para todo el sistema. Esta última opción es la recomendable en el caso de querer gestionar varios proyectos en nuestro equipo, para no tener que instalarlo en todos ellos. Para hacer esto, escribimos estos comandos desde terminal:

```
curl -sS https://getcomposer.org/installer | /opt/lampp/bin/php
sudo mv composer.phar /usr/local/bin/composer
```

Lo que hemos hecho es descargar Composer con el comando *curl* (puede que necesites instalar el comando *curl* si no te lo reconoce la máquina virtual, mediante `sudo apt-get install curl`). Después, movemos el archivo descargado (*composer.phar*) a la carpeta */usr/local/bin*, con el nombre "composer", para que al ejecutarlo lo encuentre en el PATH del sistema.

Como último paso, y ya que Composer utiliza el ejecutable de PHP, necesitamos que dicho ejecutable esté también en el PATH del sistema, para lo que haremos lo siguiente:

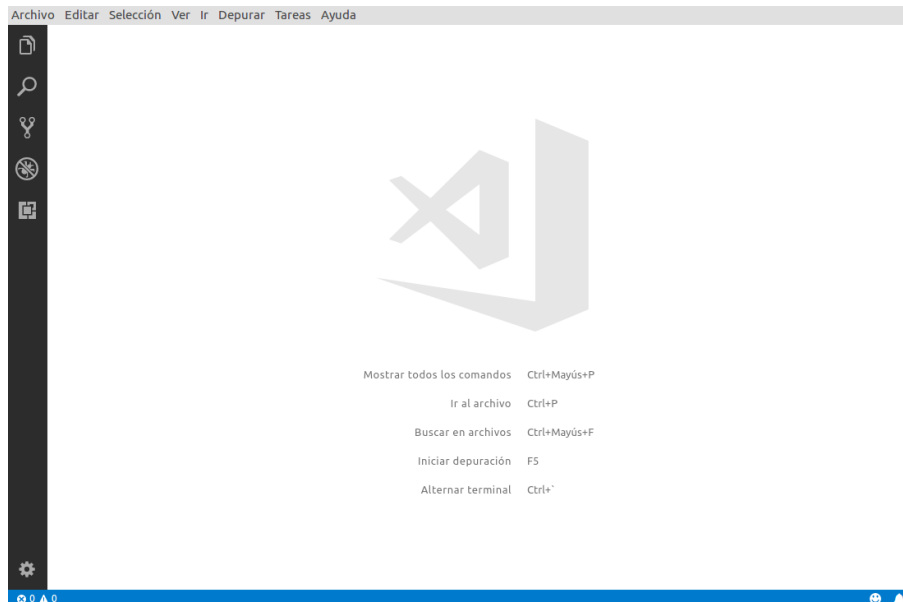
```
echo "export PATH=$PATH:/opt/lampp/bin" >> ~/.bashrc
source ~/.bashrc
```

La primera línea añade la carpeta */opt/lampp/bin* al PATH, con lo que ya se puede localizar el comando *php* a nivel global. La segunda recarga el fichero *.bashrc* para hacer efectivos los cambios.

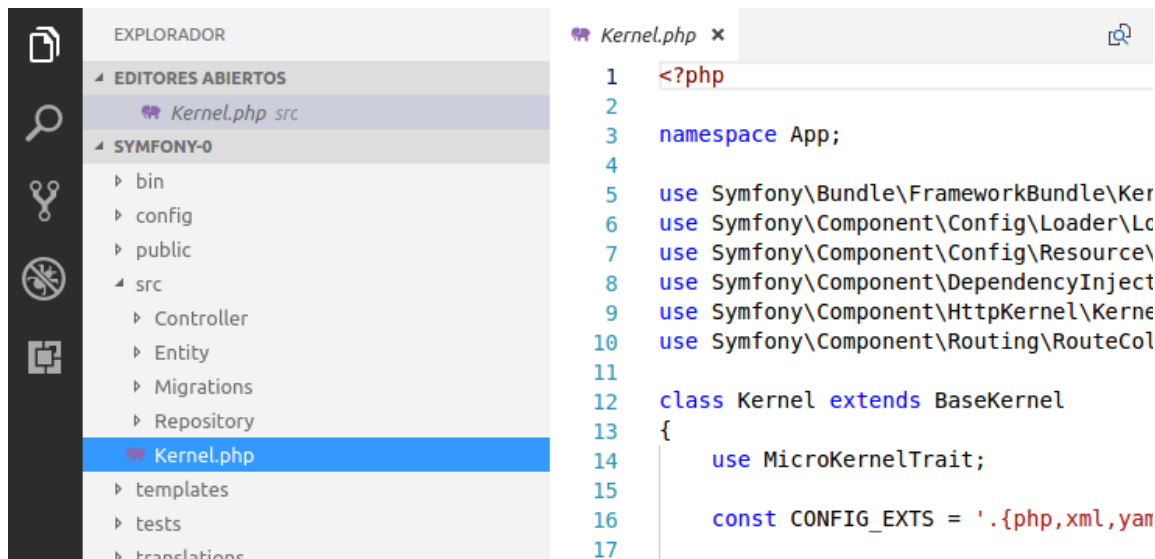
En este punto, ya puedes realizar el [Ejercicio 1](#) de los propuestos al final de la sesión.


2.3. El IDE para desarrollar los proyectos

Finalmente, para poder desarrollar los proyectos, necesitamos un entorno de desarrollo que nos permita editar el código de los archivos que necesitemos. En la máquina virtual que se os proporciona tenéis preinstalado Visual Studio Code, que será el IDE que emplearemos por su facilidad de uso y versatilidad. Al iniciarlo, aparece la página principal:



Desde el menú *Archivo > Abrir* podemos elegir una carpeta que abrir, y se mostrará en un panel izquierdo todo el contenido de dicha carpeta:



También podemos arrastrar la carpeta desde el explorador de archivos a una ventana abierta de Visual Studio Code, y se abrirá igualmente. A partir de ahí, podemos elegir el archivo a editar desde el panel izquierdo, y editarlo en el visor de código principal (ver imagen anterior). El propio panel izquierdo de archivos se puede mostrar u ocultar con el icono del explorador  de la barra de herramientas izquierda.

3. Primeros pasos con Symfony

Ahora que ya tenemos todo lo necesario para empezar a trabajar con Symfony, es hora de crear nuestro primer proyecto de prueba. Veremos qué pasos seguir para crear los proyectos, qué estructura tienen y cómo probarlos.

3.1. Nuestro primer proyecto Symfony

Para crear un proyecto Symfony, podemos descargar directamente el framework e instalarlo en nuestra carpeta de aplicación web, o bien utilizar Composer (opción recomendada). En este último caso, existen dos alternativas para crear proyectos Symfony:

```
composer create-project symfony/skeleton nombre-proyecto
```

Que creará un proyecto con el nombre indicado en la carpeta actual, conteniendo la estructura mínima, sin librerías de terceros. Será nuestra responsabilidad añadirlas más tarde. Esta funcionalidad ha sido añadida en la versión 4 de Symfony, para permitir que se instale como *microframework* y no dejar un proyecto demasiado pesado para nuestras necesidades.

```
composer create-project symfony/website-skeleton nombre-proyecto
```

Que hace lo mismo que la opción anterior, pero rellena el proyecto con una serie de dependencias ya instaladas de serie. Esta opción se deja por comodidad, y por motivos de "tradición", para que los desarrolladores que venían utilizando Symfony en sus versiones anteriores no aprecien cambios significativos al crear proyectos con las dependencias habituales ya instaladas. En versiones anteriores, se instalaba lo que se conocía como *Symfony Standard Edition*, que era una versión mucho más extensa, con varias dependencias preinstaladas.

Suele ser bastante habitual emplear esta segunda opción para crear proyectos, ya que, aunque nos instala dependencias que puede que no lleguemos a utilizar, sí nos instala automáticamente otras muy requeridas, como el motor de plantillas Twig, el ORM Doctrine, o el gestor de logs Monolog.

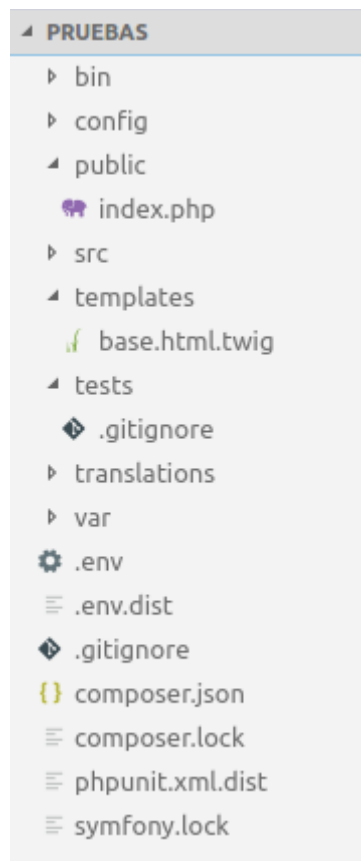
Para empezar, vamos a crear un proyecto llamado "pruebas" con la segunda opción. Accedemos a la carpeta de trabajo (podemos crear una en `/home/alumno/symfony`, por ejemplo), y escribimos este comando desde dentro de esa carpeta:

```
composer create-project symfony/website-skeleton pruebas
```

3.1.1. Estructura general de un proyecto Symfony

Tras completar el comando anterior, se habrá creado una estructura con varias carpetas y archivos dentro de `/home/alumno/symfony/pruebas`. Si abrimos esta carpeta desde Visual Studio Code, podremos ver en el panel izquierdo (Explorador) la estructura del proyecto. Esta estructura ha variado ligeramente respecto a la que se obtenía en la versión anterior, la 3, que a su vez también varió la de su predecesora para asemejarse más a una estructura Unix.

En el caso concreto de un proyecto Symfony 4, la estructura queda como sigue:



- La carpeta **bin** contiene algunos ejecutables de nuestro proyecto, como *console* para escribir comandos de consola, o *phpunit* para lanzar las pruebas unitarias. Analizaremos el primero de ellos más adelante en esta misma sesión.
- La carpeta **config** contiene los archivos de configuración para los diferentes ámbitos en que se desarrolle el proyecto. Por defecto se definen 3 ámbitos: *dev* (para desarrollo), *prod* (para puesta en producción) y *test* (para pruebas), aunque podemos añadir los que queramos. Cada ámbito contiene unos archivos de configuración YAML (veremos más adelante en qué consiste este formato), para poder especificar opciones concretas. Por ejemplo, en el ámbito *dev* nos puede interesar que se muestre toda la información necesaria por pantalla, o a un archivo, mientras que en el ámbito *prod* primará más la eficiencia. En cualquier caso, fundamentalmente emplearemos esta carpeta para configurar rutas y servicios, como veremos en sesiones posteriores.
- La carpeta **public** (llamada *web* en versiones anteriores) contendrá la parte pública o estática de la web. Aquí colocaremos hojas de estilos CSS, archivos Javascript para la parte del cliente, páginas estáticas HTML o PHP...
- La carpeta **src** contiene el código fuente PHP propiamente dicho, es decir, las clases que conformarán nuestro modelo de datos, y los controladores de la aplicación, fundamentalmente. También hay elementos importantes, como el archivo **Kernel.php**, que de alguna forma controla el resto de la configuración: establece qué *bundles* deben activarse, y para qué ámbitos, entre otras cosas.

- La carpeta **templates** contendrá las plantillas para las vistas, es decir páginas que se preprocesarán por el motor de plantillas (Twig, en nuestro caso) para generar las vistas de la aplicación, como veremos más adelante.
- La carpeta **tests** se destina al desarrollo de pruebas. En versiones anteriores, esta carpeta no era de primer nivel, y para la versión 4 se ha establecido que sí lo sea.
- La carpeta **translations** se utiliza para opciones de internacionalización
- La carpeta **var** se emplea para archivos temporales, como por ejemplo archivos de caché o de *log*.
- La carpeta **vendor** se emplea para instalar los *bundles* de terceros que nos descarguemos, como también veremos en sesiones posteriores.

3.1.2. Probando nuestro proyecto

Para poder probar el proyecto, y teniendo en cuenta que lo hemos creado fuera de la carpeta de trabajo por defecto de XAMPP (*/opt/lampp/htdocs*), debemos crear un *virtual host* que apunte a nuestra carpeta. Para ello, seguimos estos pasos:

1. Edita el archivo */etc/hosts* (con permisos de root) para añadir un nuevo dominio local para nuestra aplicación. Llamaremos al dominio *symfony.pruebas*:

```
127.0.0.1 symfony.pruebas
```

2. Debemos editar un archivo de configuración secundario de XAMPP para dar permisos de acceso a nuestra carpeta de trabajo. El archivo en cuestión es */opt/lampp/apache2/conf/httpd.conf*, y debemos añadir al final la siguiente entrada:

```
<Directory "/home/alumno/symfony">
    Options Indexes FollowSymLinks
    AllowOverride All
    Order allow,deny
    Allow from all
    Require all granted
</Directory>
```

3. Habilita la creación de hosts virtuales en Apache, editando el archivo principal de configuración */opt/lampp/etc/httpd.conf* y descomentando la línea que aparece en negrita, que está casi al final del archivo:

```
# Virtual hosts
```

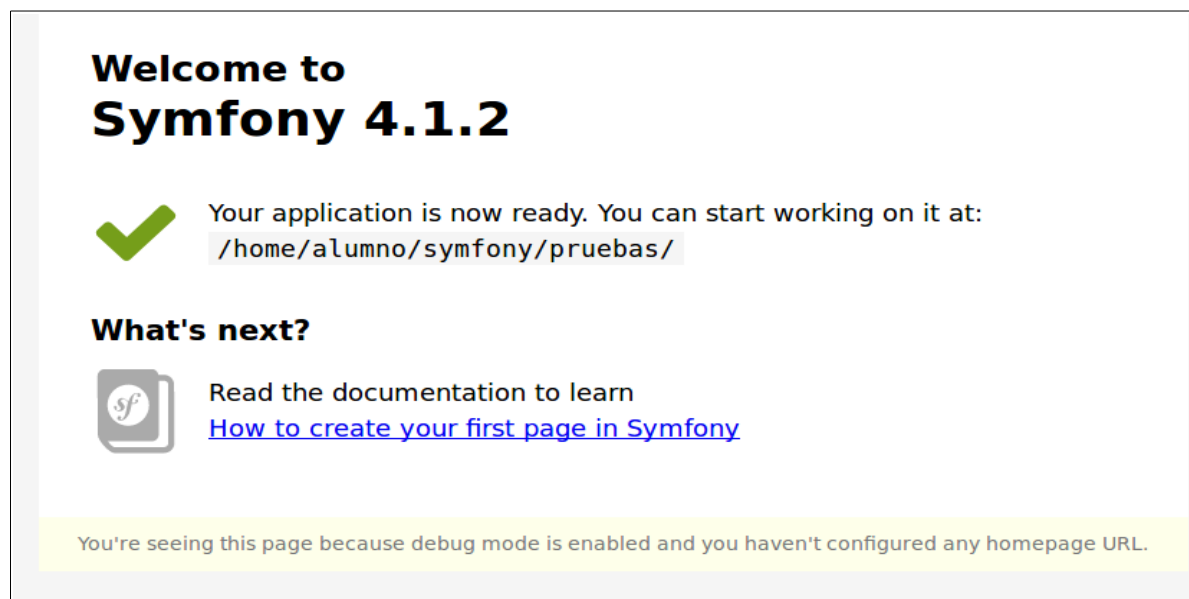
```
Include etc/extra/httpd-vhosts.conf
```

4. Edita el archivo */opt/lampp/etc/extra/httpd-vhosts.conf* y añade un nuevo host virtual para nuestra aplicación. En nuestro caso, deberá quedarte así, si has seguido los pasos anteriores para crear el proyecto en la carpeta correspondiente:

```
<VirtualHost *:80>
    DocumentRoot "/home/alumno/symfony/pruebas/public"
    ServerName symfony.pruebas
```

</VirtualHost>

5. Reinicia el servidor Apache, e intenta acceder a `http://symfony.pruebas`, para ver la página de inicio:



NOTA: para las siguientes aplicaciones que hagas, no será necesario realizar los pasos 2 y 3, ya que con esto ya habrás dado permisos de acceso a tu carpeta de trabajo, y dejado habilitados los hosts virtuales para el futuro.

En este punto, puedes realizar el [Ejercicio 2](#) del final de la sesión.

3.2. Distribuir nuestros proyectos Symfony

Si echas un vistazo al tamaño de la carpeta `/home/alumno/symfony/pruebas` del proyecto que hemos creado, comprobarás que ocupa bastantes megas. Gran parte de ese tamaño se debe a las dependencias que por defecto se instalan al crear el proyecto con la opción `website-skeleton`, y el tamaño crecerá aún más si necesitamos instalar alguna otra dependencia adicional a las que ya vienen.

Obviamente, distribuir una carpeta de ese tamaño no es algo cómodo, ni recomendable. En lugar de eso, lo que se puede/suele hacer es omitir la subcarpeta `vendor`, que es donde se instalan todas las dependencias externas. Este hecho no supone ningún problema, ya que, una vez obtenido el proyecto Symfony, se puede regenerar la carpeta con este comando (desde la carpeta principal del proyecto Symfony en cuestión):

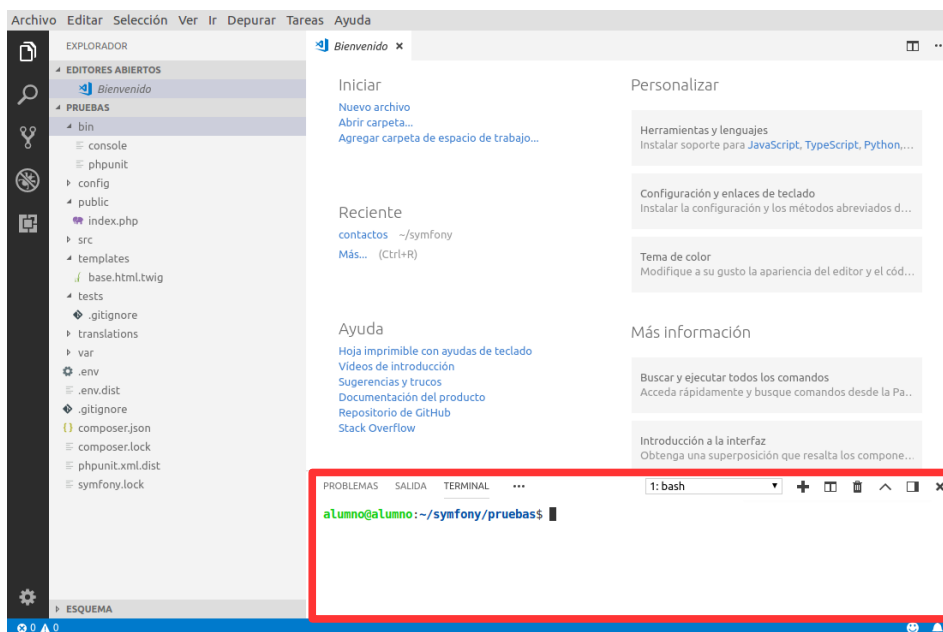
```
composer install
```

que simplemente vuelve a crear la carpeta `vendor` y reinstala dentro las dependencias que figuran en el archivo `composer.json`, que por defecto contiene todas las dependencias necesarias para el `website-skeleton`. Más adelante veremos cómo añadir más dependencias a los proyectos Symfony.

3.3. La consola de Symfony

Al analizar la estructura de un proyecto Symfony, hemos visto que existe una carpeta *bin* con un par de archivos ejecutables a través de PHP. Uno de ellos se emplea para pruebas unitarias (*phpunit*), y no lo veremos en el curso por motivos de tiempo, y el otro (**console**) sí es más ampliamente utilizado para ayudarnos a ciertas tareas, como por ejemplo crear entidades, poner en marcha un servidor de pruebas, examinar servicios, etc.

Para empezar, prueba a abrir un terminal y situarte en la carpeta del proyecto “pruebas” de Symfony que hemos creado esta sesión. Puedes hacer esto directamente si abres el terminal integrado que incorpora Visual Studio Code, desde el menú *Ver > Terminal integrado*.



Prueba a ejecutar este comando:

```
php bin/console debug:autowiring
```

Obtendrás un listado de los servicios que hay actualmente predefinidos en Symfony (más adelante veremos qué es eso de los servicios).

Existen otros comandos útiles, como un pequeño servidor PHP para poder probar rápidamente el proyecto sin necesidad de Apache, u otros comandos para crear ciertos elementos de la aplicación o examinarlos. Los iremos viendo más adelante.

4. Introducción a YAML

La configuración de los proyectos Symfony, por defecto, se define mediante archivos YAML. Estos archivos son una alternativa al formato XML, más largos y engorrosos normalmente, y al propio formato PHP, más complicado de escribir. Podríamos decir que supone un término medio en cuanto a facilidad de uso y concreción, y son archivos que se pueden ejecutar sobre la marcha. Esto tiene el inconveniente de que los posibles errores (de sintaxis, por ejemplo) no se mostrarán hasta que la aplicación no se ejecute.

El estándar YAML es muy amplio (se puede consultar en su [web oficial](#)), pero para Symfony sólo es necesario emplear un subconjunto reducido. Como podemos ver si editamos cualquier archivo YAML de un proyecto Symfony, la información se estructura en parejas *clave: valor*. Si la clave o el valor están compuestos por más de una palabra separadas por espacios, se encierran entre comillas dobles (aunque no es habitual).

```
driver: 'pdo_mysql'
```

La jerarquía de la información se establece mediante indentaciones, pero éstas no deben hacerse con el tabulador, sino con la barra espaciadora (cuatro espacios por nivel).

```
default_table_options:
    charset: utf8mb4
    collate: utf8mb4_unicode_ci
```

Puede haber grupos de elementos *clave: valor*, formando arrays. Estos arrays van entre corchetes, en el caso de arrays normales, o entre llaves, en el caso de arrays asociativos:

```
roles: [ROLE_USER, ROLE_ADMIN]
users:
    admin: { password: adminpass, roles: [ROLE_ADMIN] }
    user: { password: userpass, roles: [ROLE_USER] }
```

Los comentarios se indican con una almohadilla al principio de cada línea del comentario.

```
# In-memory users
```

```
users:
    admin: ...
```

La primera clave de cada archivo YAML es su clave principal, que contiene al resto de claves y valores (el resto van indentadas uno o varios niveles respecto a ésta). Cuando se ejecuta la aplicación, Symfony convierte estos archivos YAML en archivos PHP con un código equivalente.

5. Ejercicios

5.1. Ejercicio 1

Tras los tres primeros apartados de la sesión, ya deberías tener operativa la máquina virtual con XAMPP y Composer instalados, bien porque lo hayas seguido todo paso a paso, o bien porque hayas decidido emplear la versión de máquina virtual con todo el software ya instalado.

En cualquiera de los dos casos, y para verificar que tu instalación es correcta antes de continuar, se pide que realices dos capturas de pantalla:

1. Pon en marcha Apache y MySQL desde el manager de XAMPP, accede con Firefox a la URL `http://localhost` y captura la pantalla que muestra el navegador. Guárdala en un archivo llamado "xampp.png" (o JPG, o el formato de imagen que prefieras).
2. Desde un terminal, ejecuta el comando:

```
composer --version
```

Y captura un pantallazo donde se vea la versión de Composer que tienes instalada. Guárdalo en "composer.png" (o el formato de imagen que prefieras)

5.2. Ejercicio 2

En este ejercicio vamos a crear un proyecto Symfony llamado *contactos*, que iremos completando en sesiones posteriores, en la carpeta de trabajo `/home/alumno/symfony`, con la estructura básica de *website-skeleton*. Después de crearlo, añade un *virtual host* para poder acceder al proyecto con la URL `http://symfony.contactos`

Repite los mismos pasos para otro proyecto llamado *libros*, también alojado en tu carpeta de trabajo `/home/alumno/symfony`. Añade el correspondiente *virtual host* para acceder a él con la URL `http://symfony.libros`

Como resultado de este ejercicio, deberás adjuntar una captura de pantalla para cada proyecto, donde se vea en el navegador la URL correspondiente (`http://symfony.contactos` o `http://symfony.libros`, respectivamente), y la página de bienvenida por defecto de Symfony. Guarda las capturas como "contactos.png" y "libros.png" (o el formato de imagen que prefieras).