



“Primeros Pasos con Symfony”

© Todos los logos y marcas utilizados en este documento, están registrados y pertenecen a sus respectivos dueños.

Introducción

Uno de los temas que más se ha tratado estos últimos años en los equipos de desarrollo es **cómo mejorar la productividad**, evitar perder tiempo en tareas repetitivas, o en construcción de funcionalidades que deberían ser rápidas de conseguir por lo básico y necesario de su existencia. **La esencia de los frameworks es servirnos como caja de herramientas para construir no solo sistemas, también nuevas herramientas. Symfony** no se ha quedado atrás, ahora no solo debemos estar preparados y actualizados en el desarrollo general usando nuestro lenguaje base, PHP5.6, sino, invertir grandes cantidades de tiempo para seguir el vertiginoso avance de estas herramientas (y que muchas veces no podemos abarcar absolutamente todo lo que ofrecen y debemos concentrarnos en lo que realmente vamos a necesitar en el proyecto que estemos trabajando actualmente).

Y como complemento, iremos aplicando en cada nuevo tema una serie de exigentes **“tareas desafío”** adecuadas a los tiempos que corren, tratando con esto aumentar nuestra profesionalidad, ya que a pesar de contar con una excelente *“caja de herramientas”*, **debemos seguir siendo desarrolladores que deben saber trabajar adecuadamente con objetos (POO)**, más allá de la herramienta que seleccionemos para solucionar nuestros problemas.

Objetivos

Esta clase busca iniciarnos en todo el procedimiento de crear un proyecto Symfony, con sus componentes principales y todos los detalles que hay que tener en cuenta.

Sobre los contenidos de las guías de estudio: si bien la mayoría de la información que vamos a referenciar en estas guías de estudio se extraen de los **manuales oficiales**, las guías nunca deberán ser sustituto de los manuales, ya que esta debe ser siempre nuestra fuente principal de consulta. En las guías iremos agregando **experiencia personal** obtenida del trabajo diario con el framework y ordenando los temas con un criterio que consideramos “**evolutivo**” y que facilita el aprendizaje simulando situaciones cercanas a la realidad de cualquier desarrollo que se inicia de cero.

Requerimientos Mínimos

Symfony 3 al estar escrito en [PHP 5.6](#), un lenguaje multiplataforma, puede ejecutarse en la mayoría de los [S.O](#) tales como [Linux](#), [Mac OS X](#), [Microsoft Windows](#), [Solaris](#), etc.

Para nosotros (aunque pueden existir otras alternativas viables) el "entorno ideal" sería poder contar con un servidor **GNU/Linux** con un [servidor web](#) como [Apache](#), el módulo [mod_rewrite](#) habilitado, [PHP5](#) en su **versión 5.6** o

El "entorno ideal" puede cambiar y/o sustituir algunos componentes con otras alternativas (por ej. en el caso de no contar con mod_rewrite), pero lo "ideal" sería que nuestro servidor o servicio de hosting contara con estos servicios y trabajáramos con todas las herramientas requeridas y con las cuales se seleccionaron para usar con el framework.

superior y asegurarnos que esté instalado [PDO](#) para poder realizar operaciones sobre las bases de datos, generalmente viene activado por defecto, también es necesario tener activadas las siguientes extensiones de PHP5:

- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- MCrypt PHP Extension
- Ctype PHP Extension
- JSON PHP Extension
- Intl PHP Extension
- Iconv PHP Extension



PDO es una extensión que provee una capa de abstracción de acceso a datos para PHP 5, con lo cual se consigue hacer uso de las mismas funciones para hacer consultas y obtener

datos entre distintos manejadores de bases de datos (una capa más de abstracción).



El Mod Rewrite es un módulo para el servidor Apache que nos permite reescribir las apariencias de las URLs. A través de diferentes reglas y condiciones se pueden personalizar esas URLs o Permalinks de nuestras páginas de forma que podamos conseguir enlaces más amigables para los buscadores y para nuestros visitantes. Por ej., en vez de `/index.php?mod=estadisticas&usuario=1` podríamos tener `/estadisticas/usuario/1`

Si trabajamos en GNU/Linux, casi cualquier distribución moderna incluye **PHP5**, **MySQL5** y **Apache2** (este combo es comúnmente llamado **LAMP** por las siglas de sus componentes).



Sin embargo, si estás más acostumbrado a la plataforma **Windows** existe también un combo que se llama **WAMP**. Una de las herramientas que reúne todos los elementos anteriores necesarios de manera muy sencilla se llama **Xampp**. Para instalarlo solo tienen que seguir los pasos indicados según el paquete correspondiente a nuestro [sistema operativo](#).



Si nuestro GNU/Linux no tuviera PHP5, [existen un paquete en Xampp](#) que soluciona este problema.

Primera configuración

Una vez instalado nuestro entorno de trabajo con todos sus elementos, puede que necesitemos hacer un par de ajustes antes de usar **Symfony**.

Una forma de verificar nuestro entorno es ejecutar un `index.php` con un simple `"phpinfo();"`

Si nuestro entorno es Windows

Al instalar **Xampp** no necesitamos configurar nada más ya que viene listo para usar **Symfony**, así que las configuraciones de más abajo no deberían requerirse, pero por las dudas podemos revisarlas y ya conocer cómo y donde es que se configuran.

Configuración Base

Editamos el archivo **httpd.conf** de configuración de Apache ubicado generalmente en **C:\xampp\apache\conf**

Código:

```
#LoadModule rewrite_module modules/mod_rewrite.so
```

Le quitamos el comentario **#** dejándolo así

Código:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

También tenemos que asegurarnos (en caso de no usar XAMPP puesto que viene listo para usar) de que Apache está configurado para admitir el archivo .htaccess. Esto se hace generalmente modificando en el archivo httpd.conf:

AllowOverride None

por

AllowOverride All

Lo encontramos dentro del elemento <Directory "C:/directorio_web_root"> que apunta hacia el directorio web root, por ejemplo en el caso de Xampp sería más o menos así:

```
<Directory "C:/xampp/htdocs">
#
# Possible values for the Options directive are "None", "All",
# or any combination of:
#   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
#
# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you.
#
# The Options directive is both complicated and important. Please see
# http://httpd.apache.org/docs/2.2/mod/core.html#options
# for more information.
#
Options Indexes FollowSymLinks Includes ExecCGI

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
AllowOverride All

#
# Controls who can get stuff from this server.
#
Require all granted

</Directory>
```

Luego editamos el archivo **php.ini** en **c:\xampp\php**

Código:

```
;extension=php_pdo_mysql.dll
```

Quitamos el **punto y coma** dejándolo así

Código:

```
extension=php_pdo_mysql.dll
```

Además se recomiendan las siguientes configuraciones en el php.ini:

- short_open_tag = Off
- magic_quotes_gpc = Off
- register_globals = Off
- session.auto_start = Off

Además tener configurado **date.timezone** según nuestra localización:

- date.timezone=America/Santiago

Luego **debemos reiniciar Apache para que apliquen los cambios** en los archivos de configuración.

Probando nuestro servidor web

Vamos a asegurarnos que todo está corriendo hasta ahora como corresponde. Entonces creamos un archivo **index.php** y añadiremos el siguiente contenido:



```
<?php echo "Hola, Mundo sin Symfony!";
```

Ahora abrimos nuestro navegador favorito (solo **Firefox** ;-)) y escribimos la dirección URL <http://localhost/> para ver una simple página con nuestro texto de burla al mundo de programadores que aún no usa Symfony.



¡Ey! Se olvidaron de colocar el ?>! (jejeje, los pillamos volando bajo!)

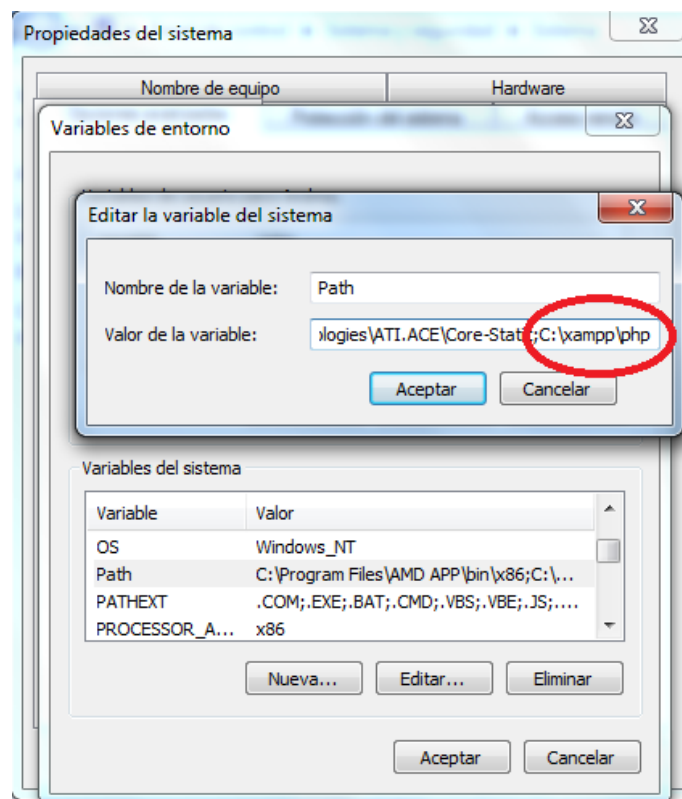
*¡No nos olvidamos de la etiqueta de cierre ?> de PHP! Se debe omitir intencionalmente **cuando se trata de archivos PHP que contengan sólo código PHP** como por ejemplo las clases. Estamos omitiendo intencionalmente para evitar la salida involuntaria caracteres o espacios en blanco que se puedan producirse. De hecho, esta es una de las "mejores prácticas" recomendadas por el estándar de PHP5.*

Configurar acceso global a los comandos PHP desde consola

En el caso de Windows, **necesitamos también definir la ruta hacia PHP C:\xampp\php** para que nos reconozca los comandos de php, y para ello podemos agregar la ruta a través de **variables del sistema**.

Por lo general, en la mayoría de las versiones de Windows recientes debemos ir a:

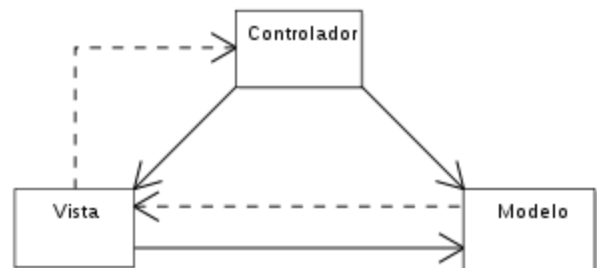
Panel de Control > Sistema > Configuración Avanzada y luego **modificar las variables de ambiente** y agregar las rutas en el path: **C:\xampp\php** (Luego de cambiarlo, deberás reiniciar tu sistema para que tome las nuevas variables de ambiente).



Antes de empezar, cómo funciona rápidamente un MVC

El patrón MVC podría considerarse una "especialización" de una típica "arquitectura de 3 capas" pero con una lógica de funcionamiento más ajustada a entornos web, donde se intenta -por lo general- concentrar todas las peticiones del exterior a través de un solo punto de entrada (clásicos sistemas "modulares")

En este nuevo diseño tendremos que **todas las peticiones del usuario se dirigirán hacia un Controller ("Controlador")** que se encargará de decidir si solicita información de la "lógica de negocio" al **Model ("Modelo")** y/o pasa a la **View ("Vista")** para que se encargue de generar la interfaz de usuario.



Nota: se recomienda [la lectura de la explicación](#) escrita en Wikipedia

Aquí se puede visualizar un **diagrama que muestra la relación con sus componentes** (más adelante entraremos en detalles, por ahora solo veremos lo básico y esencial para entender el funcionamiento del patrón).

Aclaración importante: los patrones de diseño son *"la mejor solución encontrada para resolver un problema genérico y recurrente"*, pero, la solución es una "guía" y no un "manual" para resolver un problema específico, ya que cada contexto tendrá sus particularidades. **Symfony "implementa su propia versión" del patrón de diseño**, por lo tanto no significará nunca que lo que veamos de Symfony es sinónimo puro de MVC, en realidad veremos que existen muchos frameworks que implementan su propia versión del patrón, y todas estarán correctas, unas mejores que otras.

Un patrón de diseño es solo eso, una guía.

Como obtener y buscar información Oficial de Symfony

Unos de los principios que seguiremos en el del taller es el de leer y buscar información en la documentación oficial de Symfony en su versión 3, cosa que la mayoría es lo que menos hace, o lo deja como última instancia, cuando debiera ser todo lo contrario.

Este principio es conocido como [RTFM](#) es un acrónimo que en inglés significa "Read The Fucking Manual" (Lee el Maldito Manual" en español), otra traducción un poco más educadas sería "Read The Fine Manual" o Lee el Buen Manual 😊

La documentación se organiza en dos grandes puntos: el **APIs** y la **Guía de Referencia**:

1. [APIs](#): Si estamos buscando información sobre un método particular, un atributo de alguna clase o bien no recordamos el nombre de una constante. Aquí podemos obtener toda la información y documentación de las clases y componentes de Symfony a unos pocos clic de distancia.
2. [Guía de Referencia o Manual](#): Con más de 1000 páginas y sobre 500 ejemplos, tiene un sistema de búsqueda dentro del manual, contiene en detalle todos los Componentes de Symfony con una notable calidad en sus explicaciones y códigos de ejemplo.
3. Además de contar con la **documentación de las APIs** y de la **Guía de Referencia de Symfony** en el sitio oficial, tenemos disponible una completísima sección de tutoriales llamados libro de cocina o Cookbook la URL es el siguiente: <http://symfony.com/doc/current/cookbook/index.html>

Estructura de una Aplicación bajo Symfony Framework

Primero necesitaremos tener un directorio que representará nuestra aplicación. Para este ejemplo **usaremos el nombre “symfony-project”**, donde posteriormente deberá tener una estructura estándar para alojar todos los elementos que vayamos necesitando agregar de acuerdo a los componentes de MVC que requiere nuestra aplicación.

La estructura resumida de directorio debería una típica aplicación **Symfony** sería la siguiente:

```
symfony-project/  
  app/  
    config/  
    Resources/  
      /views/  
        <*.php archivos de vistas>  
      // etc.  
  src/  
    AppBundle/  
      /Controller/  
        <*.php clases controladoras>  
      // etc.  
  tests/  
    AppBundle/  
  var/  
    cache/  
    logs/  
    sessions/  
  web/  
    css/  
    images/  
    js/  
    .htaccess  
    app.php  
  vendor/  
    bin/  
    composer/  
    doctrine/  
    swiftmailer/  
    symfony/  
    twig/  
    // etc.
```

De la estructura de directorio anterior podemos observar lo siguiente:

Primero que todo, tenemos el directorio de la aplicación "**app**". Este es considerado como uno de los más importante ya que es donde encontramos nuestras vistas (script o plantillas) se ubican dentro del directorio **app/Resources/views**. Observamos también el directorio **app/config** contiene toda la configuración utilizada por Symfony para cargar los settings de las sesiones http y otras configuraciones (por ejemplo, autenticación o seguridad, mail, la configuración de base de datos y credenciales de acceso), rutas url, que detallaremos más adelante.

Tercero, tenemos al directorio **src/**. Donde creamos e implementamos nuestra estructura MVC, las clases controladoras/controllers, modelos/models etc.

Luego tenemos al directorio **tests/**, lugar donde llevaremos a cabo nuestras "pruebas unitarias" (testing). Esto puede ser usado generalmente para probar los modelos que deben contener la lógica de negocio y nuestros controladores.

Después tenemos el directorio **vendor**, lugar donde guardamos nuestros propios componentes y librerías de terceros como Doctrine, Swiftmailer, Composer y las mismas de Symfony.

Finalmente, tenemos al directorio **web/**. Este no solo guarda nuestro **app.php** quien se encarga de arrancar nuestra aplicación, sino que además podría **contener archivos de imágenes, hojas de estilos css, javascript etc.**

Podemos ver una estructura que probablemente se repetirá en la mayoría de los framework: los directorios controllers, models y views que veremos más adelante que se encuentran dentro de nuestra aplicación.

Aquí podemos empezar a entender las ventajas de un framework a la hora de organizar la estructura de nuestro sistema: nunca más discutiremos donde deberíamos ubicar el archivo de configuración o donde deberían ir los componentes de nuestro sistema. Ya está resuelto y nosotros debemos seguir la forma de trabajo definida.

El directorio "web"

El directorio *web* deberá tener todos los archivos que pueden ser directamente accesibles desde nuestro servidor (archivos públicos, que no nos debería afectar en nada si son bajados desde el exterior de forma anónima), tales como el "**arranque**" (app.php), imágenes, css, js etc... Y **deberá incluir su respectivo archivo .htaccess con las reglas de acceso**. En esencia se busca que cualquier acceso a nuestra estructura sea redirigida a nuestro app.php para tener el control por un único punto de acceso, a menos que digamos lo contrario, como sucede con las imágenes, css y js, que deben poder accederse de forma pública o no funcionarían. Vamos a asumir que estamos corriendo sobre nuestra máquina local y que nuestro directorio *web* es accesible mediante la url: <http://localhost/symfony-project/web>

TIPS

*Si tenemos la ruta a nuestro servidor web en **c:\xampp\htdocs** podemos decirle a nuestro IDE que cree en este directorio todos los proyectos, lo que nos permitirá de forma simple poder trabajar en varios sistemas a la vez accediendo a **http://localhost/nombreproyecto***

Instalación

La instalación de Symfony cuenta con diferentes formas de instalar, para efecto del curso veremos sólo dos de ellas, instalando **vía Symfony Intaller** y la segunda **vía Composer con el comando create-project**.



Para más detalles sobre la Instalación Symfony: favor visitar la siguiente página: <https://symfony.com/doc/current/book/installation.html>



Para más detalles sobre Composer: favor visitar la siguiente página: <https://getcomposer.org/doc/00-intro.md>



A pesar de que podemos instalar **Symfony** con diferentes métodos, por ejemplo "**vía Symfony Installer o bien Composer**" (desde el sitio oficial) **o vía GitHub:** instalación mediante la descarga del Skeleton symfony-master, sin embargo el proceso de instalación más recomendable es utilizar **Symfony Installer**

Primera forma, instalando vía Symfony Installer (Recomendada)

La primera forma, la recomendada, con el fin de construir nuestra aplicación **Symfony** debemos descargar el instalador **Symfony Installer**. El instalador es una aplicación PHP que debe ser instalada en el sistema una sola vez y luego se pueden crear varios proyectos Symfony.

Dependiendo del sistema operativo, el instalador debe instalarse de diferentes formas:

Si utilizamos sistema operativo Unix, Linux o Mac, simplemente en el terminal o consola escribimos:

```
$ sudo curl -Ls https://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

Esto creará un comando global de Symfony en nuestro sistema **(Linux, Mac OS X)**. Una vez que el instalador Symfony está disponible globalmente, podemos crear nuestra primera aplicación Symfony con el comando:

```
$ symfony new symfony-project
```

El comando anterior se ejecuta sobre el directorio httdocs o webroot donde se encuentran nuestros proyectos en apache, por ejemplo httdocs de /opt/lampp/ (xampp)

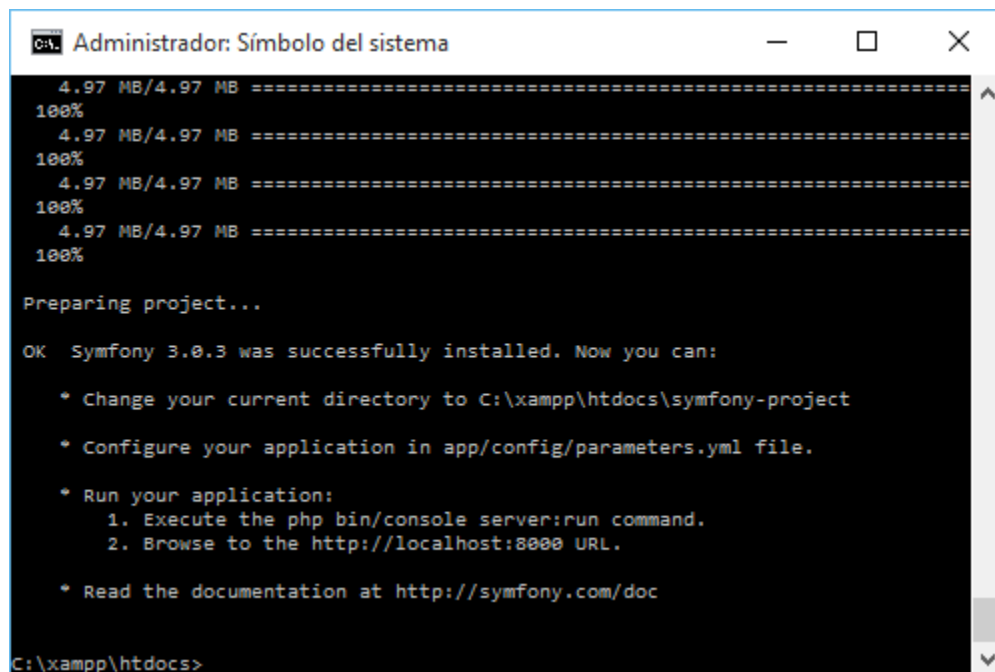
Si estamos utilizando el sistema operativo Windows simplemente con el comando CMD:

```
c:\> cd C:\xampp\htdocs  
c:\> php -r "readfile('https://symfony.com/installer');" > symfony
```

Este comando descarga el archivo symfony en el directorio htdocs o webroot donde se encuentran nuestros proyectos, en el ejemplo el htdocs de XAMPP.

Una vez que el instalador Symfony está disponible, podemos crear nuestra primera aplicación Symfony con el siguiente comando:

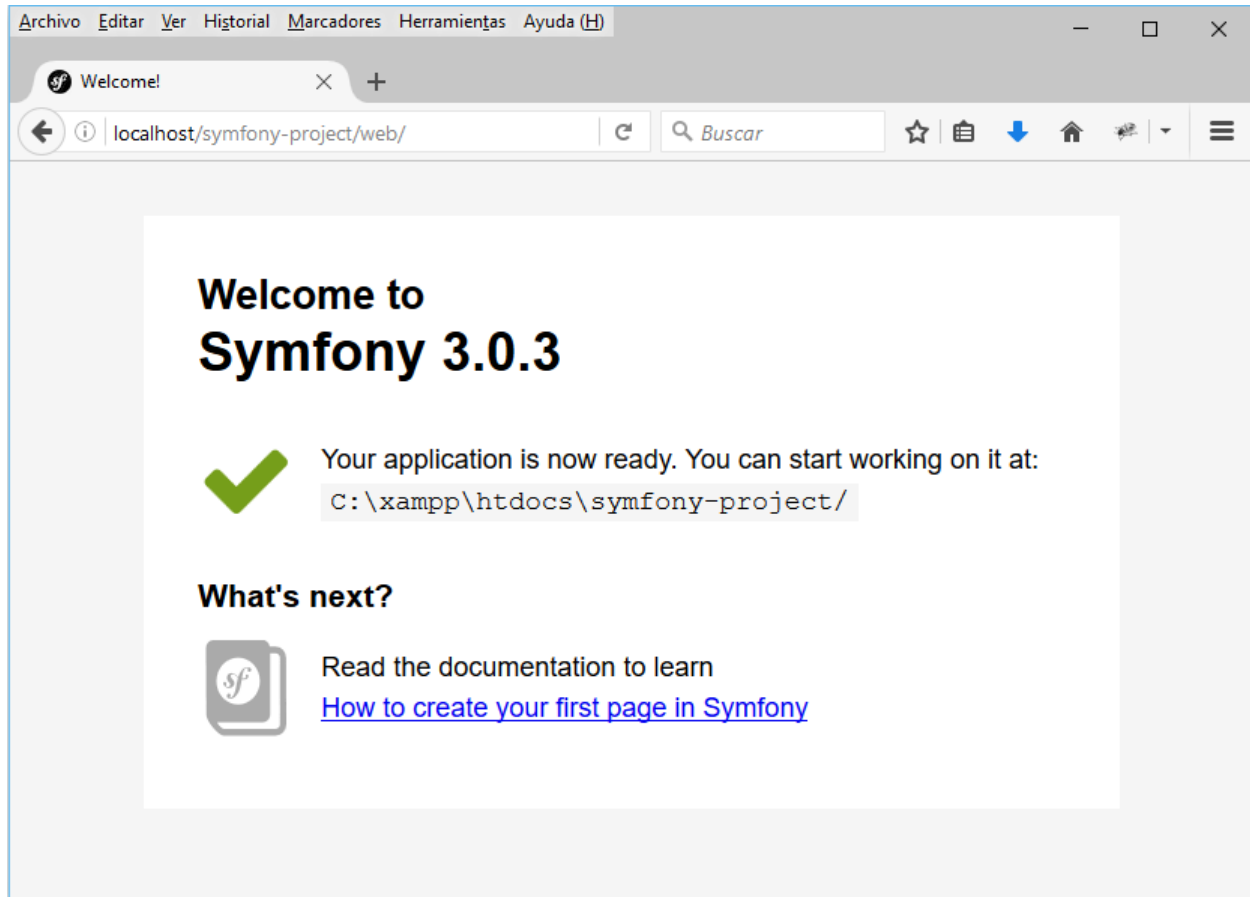
```
C:\xampp\htdocs> php symfony new symfony-project
```



```
Administrador: Símbolo del sistema  
4.97 MB/4.97 MB =====  
100%  
4.97 MB/4.97 MB =====  
100%  
4.97 MB/4.97 MB =====  
100%  
4.97 MB/4.97 MB =====  
100%  
Preparing project...  
OK  Symfony 3.0.3 was successfully installed. Now you can:  
  
* Change your current directory to C:\xampp\htdocs\symfony-project  
* Configure your application in app/config/parameters.yml file.  
* Run your application:  
  1. Execute the php bin/console server:run command.  
  2. Browse to the http://localhost:8000 URL.  
* Read the documentation at http://symfony.com/doc  
C:\xampp\htdocs>
```

Este comando crea un nuevo directorio llamado **symfony-project** que contiene un nuevo proyecto Symfony, basado en la versión más estable y reciente disponible de Symfony. Además, el instalador comprueba si el sistema cumple con los requisitos mínimos para ejecutar nuestra aplicación Symfony. Si no es así, mostrará una lista con todos los cambios necesarios para cumplir con estos requisitos.

Si todo sale bien, entonces nos vamos al browser, escribimos la dirección:
<http://localhost/symfony-project/web/> y deberíamos ver algo como esto:



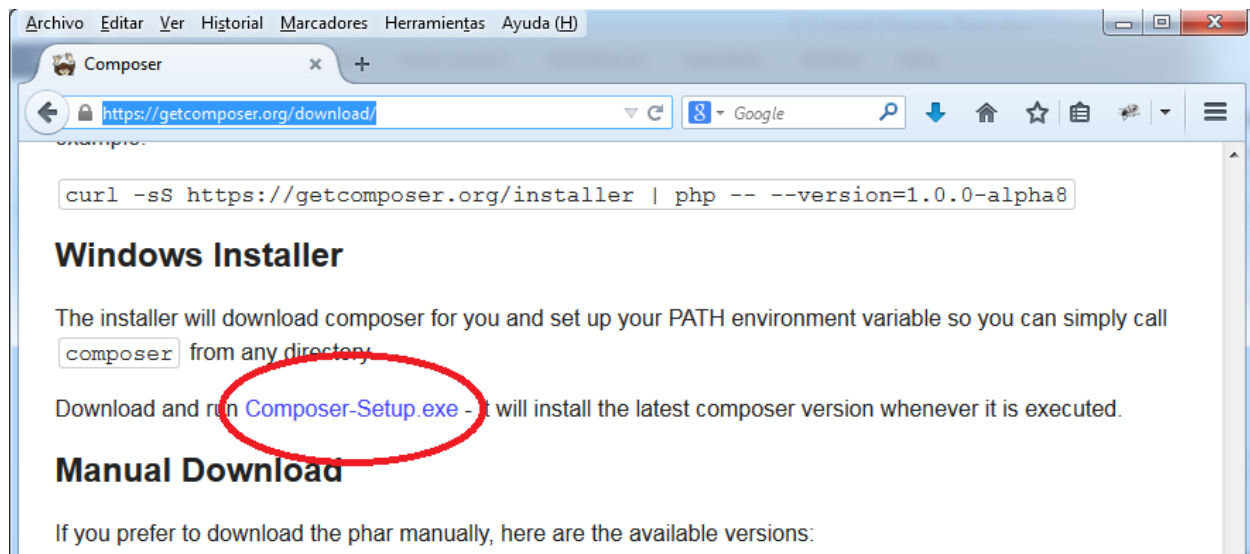
Segunda forma, instalando vía Composer

Antes de realizar la instalación de **Symfony** vía composer, primero necesitamos instalar la herramienta [Composer](http://getcomposer.org) (<http://getcomposer.org>). **Symfony utiliza Composer para administrar las dependencias y librerías requeridas.**

Primero, necesitamos descargar una copia del archivo **composer.phar**. Una vez que tengamos nuestro archivo PHAR, debemos mantener una copia dentro del directorio de nuestro proyecto **Symfony**. Si estamos usando sistema operativo **Linux**, como recomendación, podemos mover el archivo **composer.phar** al directorio **usr/local/bin** para usarlo de forma global en nuestro sistema. Sobre sistemas **Windows**, podemos usar el instalador [Composer Windows installer](#), que se encarga de configurar todo de forma automática.

Instalación de Composer:

Si estamos utilizando el Sistema Operativo Windows simplemente podemos descargar el ejecutable de composer ([Composer-Setup.exe](#)) e instalarlo con doble clic, para descargarlo nos vamos a la siguiente página: <https://getcomposer.org/download/> y clic en [Composer-Setup.exe](#)



Si utilizamos sistema operativo Unix, Linux o Mac, simplemente en el terminal, consola o CMD escribimos:

```
php -r "readfile('https://getcomposer.org/installer');" | php
mv composer.phar /usr/local/bin/composer
```

Este script instalador simplemente comprueba algunos ajustes php.ini, nos da una advertencia si algo anda mal, pero no debería, y luego descargar la última versión de **composer.phar** en el directorio actual.

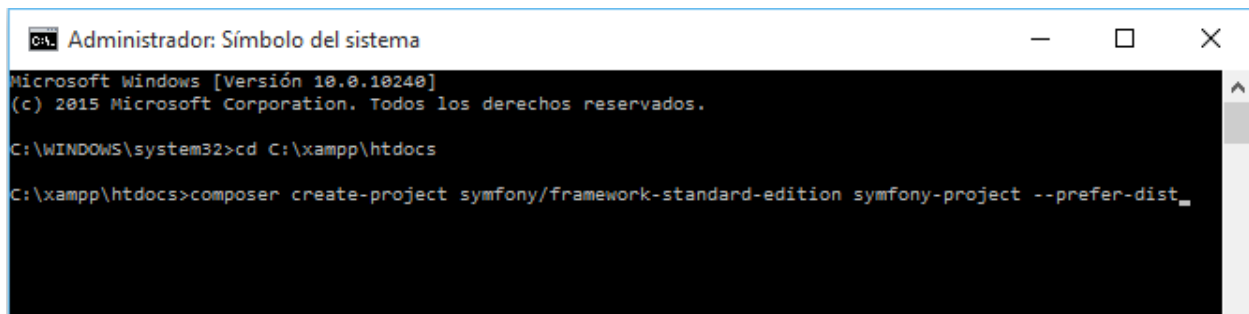
Entonces **resumiendo**, se descarga una copia de **composer.phar**. Una vez que tengamos el archivo Phar, debemos moverlo a "**usr/local/bin**" para ser utilizado a nivel global en nuestro sistema.

Para esto es necesario tener el ejecutable de PHP en la variable de entorno del sistema operativo PATH, configurado previamente.

Bien, continuamos con la instalación de Symfony vía Composer:

La segunda forma es bastante sencilla, podemos instalar Symfony en un solo paso con el comando **composer create-project** en el terminal o consola, posicionado sobre el web-root, en nuestro caso en htdocs:

```
composer create-project symfony/framework-standard-edition symfony-project --prefer-dist
```



```
Administrador: Símbolo del sistema - composer create-project symfony/framework-standa...
Microsoft Windows [Versión 10.0.10240]
(c) 2015 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>cd C:\xampp\htdocs

C:\xampp\htdocs>composer create-project symfony/framework-standard-edition symfony-project --prefer-dist
Warning: This development build of composer is over 60 days old. It is recommended to update it by running
"C:\ProgramData\ComposerSetup\bin\composer.phar self-update" to get the latest version.
Installing symfony/framework-standard-edition (v3.0.3)
- Installing symfony/framework-standard-edition (v3.0.3)
  Loading from cache

Created project in symfony-project
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
- Installing doctrine/lexer (v1.0.1)
  Loading from cache

- Installing doctrine/annotations (v1.2.7)
  Loading from cache

- Installing twig/twig (v1.24.0)
  Loading from cache

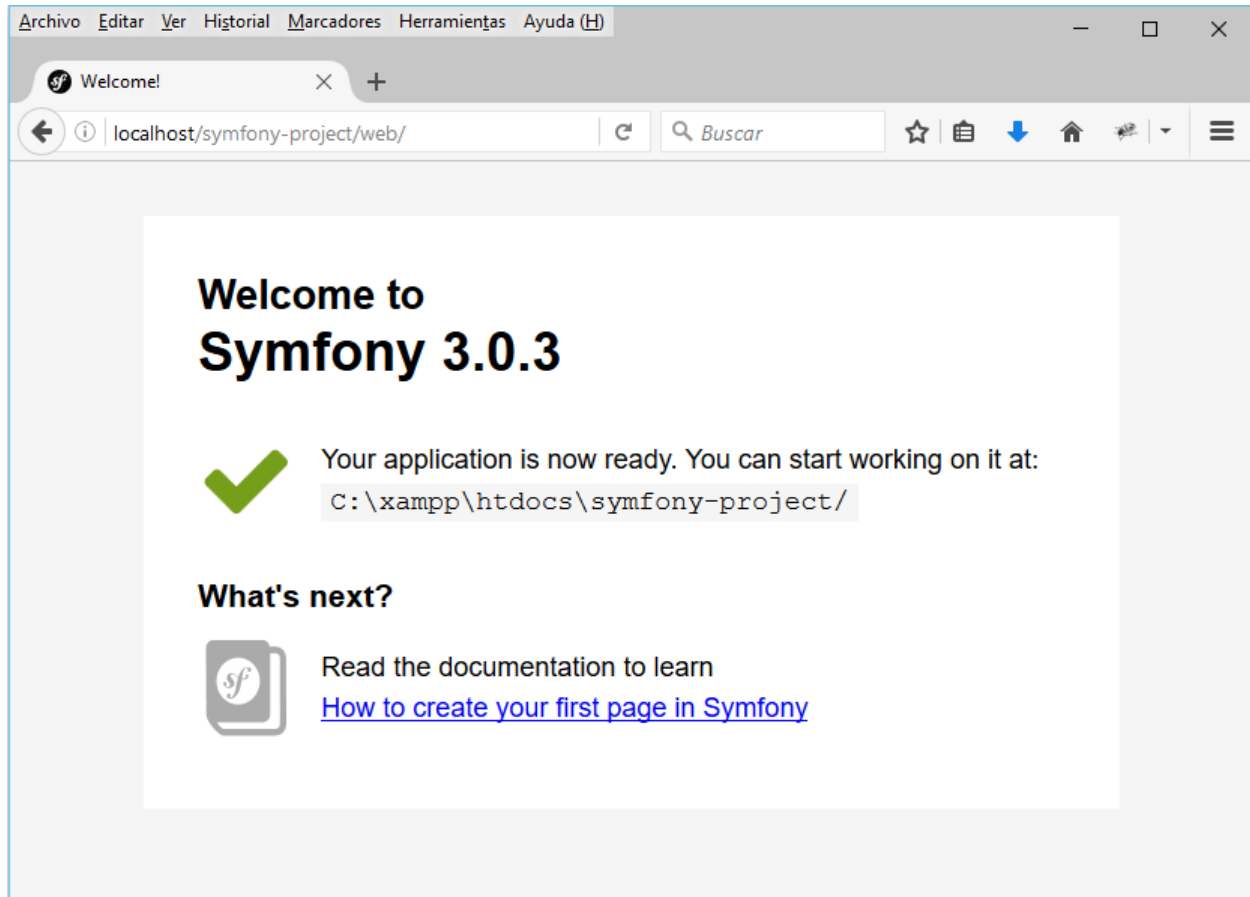
- Installing symfony/polyfill-util (v1.1.0)
```

Casi al final de la instalación nos pide algunos parámetros de instalación, por ejemplos parámetros de conexión a la base de datos, los cuales tenemos que ir llenando según el valor por defecto que muestra entre paréntesis:

```
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_host (127.0.0.1): 127.0.0.1
database_port (null): null
database_name (symfony): symfony
database_user (root): root
database_password (null): null
mailer_transport (smtp): smtp
mailer_host (127.0.0.1): 127.0.0.1
mailer_user (null): null
mailer_password (null): null
secret (ThisTokenIsNotSoSecretChangeIt): c06452e32f65bd0cb92ddaec85fb8702927dc825_
```

database_host: 127.0.0.1
database_port: null
database_name: symfony
database_user: root
database_password: null
mailer_transport: smtp
mailer_host: 127.0.0.1
mailer_user: null
mailer_password: null
secret: c06452e32f65bd0cb92ddaec85fb8702927dc825

Si todo sale bien, entonces nos vamos al browser, escribimos la dirección:
<http://localhost/symfony-project/web/> y deberíamos ver algo como esto:



Nuestro archivo .htaccess

Symfony está diseñado implementando el patrón [Front Controller](#): éste patrón se encarga de recibir y controlar todas las peticiones que vienen desde una URL (exterior), y según estos parámetros de petición direcciona hacia un módulo y a un controlador específico (de acuerdo a la información que definimos en la url), posteriormente llama a la acción deseada de nuestro controlador.

Recordar: “configuración versus convención”, si respetamos las reglas de nombres el framework logra encontrar los componentes que tiene que ejecutar.

Esto significa que se debe reescribir todas las peticiones a un único script PHP que inicializará el FrontController, de ahí la razón de por qué si queremos acceder a un directorio particular el sistema (a través de las reglas de .htaccess) redireccionará todas las peticiones a nuestro "bootstrap" (app.php) para luego a través del FrontController decidir a qué controller derivar el control del sistema según la ruta url.

Para el Servidor Web Apache se debe tener el archivo **symfony-project/web/.htaccess** que viene ya implementado en nuestra instalación de Symfony:

Código:

DirectoryIndex app.php

```
<IfModule mod_negotiation.c>
    Options -MultiViews
</IfModule>

<IfModule mod_rewrite.c>
    RewriteEngine On

    RewriteCond %{REQUEST_URI}::$1 ^(/.+)/(.*):~2$
    RewriteRule ^(.*) - [E=BASE:%1]

    RewriteCond %{HTTP:Authorization} .
    RewriteRule ^ - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

    RewriteCond %{ENV:REDIRECT_STATUS} ^$
    RewriteRule ^app\.php(?:/(.*)|)$ %{ENV:BASE}/$1 [R=301,L]

    RewriteCond %{REQUEST_FILENAME} -f
    RewriteRule ^ - [L]

    RewriteRule ^ %{ENV:BASE}/app.php [L]
</IfModule>

<IfModule !mod_rewrite.c>
    <IfModule mod_alias.c>
        RedirectMatch 302 ^/$ /app.php/
    </IfModule>
</IfModule>
```

Esta regla de sobre escritura se entregará a todas las peticiones de URLs que no terminen en .js, .ico, .gif, .jpg, .png, or .css a través de app.php. El archivo app.php es la puerta de entrada a nuestra aplicación, equivaldría al método "main" de Java o C, por eso se lo conoce comúnmente con el nombre **bootstrap**, que quiere decir el "arranque", o proceso de inicio de la aplicación.

Casi sin darte cuenta, eso es todo! **Symfony** está instalado y listo para usar.

Repasemos

1. Creamos nuestro proyecto a partir de **Symfony Intaller**
2. Con la estructura de directorios determinada por Symfony
3. Estudiamos nuestro Bootstrap app.php con su .htaccess
4. Instalamos Symfony en el directorio vendor vía Symfony Intaller

¡Estamos listos!



Mi primer Bootstrap

Lo primero que tenemos que tener claro es cuál es nuestro bootstrap y analizar nuestro app.php (que sería la puerta de entrada o Bootstrapping) que ya viene creado y listo para arrancar nuestra aplicación.

Antes de empezar necesitamos tener un archivo “.htaccess” el cuál se encuentra dentro de la carpeta “web” del proyecto. Debe contener el siguiente código:

```
# Use the front controller as index file. It serves as a fallback solution when
# every other rewrite/redirect fails (e.g. in an aliased environment without
# mod_rewrite). Additionally, this reduces the matching process for the
# start page (path "/") because otherwise Apache will apply the rewriting rules
# to each configured DirectoryIndex file (e.g. index.php, index.html, index.pl).
DirectoryIndex app.php

# By default, Apache does not evaluate symbolic links if you did not enable this
# feature in your server configuration. Uncomment the following line if you
# install assets as symlinks or if you experience problems related to symlinks
# when compiling LESS/Sass/CoffeeScript assets.
# Options FollowSymlinks

# Disabling MultiViews prevents unwanted negotiation, e.g. "/app" should not resolve
# to the front controller "/app.php" but be rewritten to "/app.php/app".
<IfModule mod_negotiation.c>
    Options -MultiViews
</IfModule>

<IfModule mod_rewrite.c>
    RewriteEngine On

    # Determine the RewriteBase automatically and set it as environment variable.
    # If you are using Apache aliases to do mass virtual hosting or installed the
    # project in a subdirectory, the base path will be prepended to allow proper
    # resolution of the app.php file and to redirect to the correct URI. It will
    # work in environments without path prefix as well, providing a safe, one-size
    # fits all solution. But as you do not need it in this case, you can comment
    # the following 2 lines to eliminate the overhead.
    RewriteCond %{REQUEST_URI}::$1 ^(/.+)/(.*):~2$
    RewriteRule ^(.*) - [E=BASE:%1]

    # Sets the HTTP_AUTHORIZATION header removed by Apache
    RewriteCond %{HTTP:Authorization} .
    RewriteRule ^ - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

    # Redirect to URI without front controller to prevent duplicate content
    # (with and without `/app.php`). Only do this redirect on the initial
    # rewrite by Apache and not on subsequent cycles. Otherwise we would get an
```

```

# endless redirect loop (request -> rewrite to front controller ->
# redirect -> request -> ...).
# So in case you get a "too many redirects" error or you always get redirected
# to the start page because your Apache does not expose the REDIRECT_STATUS
# environment variable, you have 2 choices:
# - disable this feature by commenting the following 2 lines or
# - use Apache >= 2.3.9 and replace all L flags by END flags and remove the
# following RewriteCond (best solution)
RewriteCond %{ENV:REDIRECT_STATUS} ^$
RewriteRule ^app\.php(?:/(.*)|)$ %{ENV:BASE}/$1 [R=301,L]

# If the requested filename exists, simply serve it.
# We only want to let Apache serve files and not directories.
RewriteCond %{REQUEST_FILENAME} -f
RewriteRule ^ - [L]

# Rewrite all other queries to the front controller.
RewriteRule ^ %{ENV:BASE}/app.php [L]
</IfModule>

<IfModule !mod_rewrite.c>
    <IfModule mod_alias.c>
        # When mod_rewrite is not available, we instruct a temporary redirect of
        # the start page to the front controller explicitly so that the website
        # and the generated links can still be used.
        RedirectMatch 302 ^/$ /app.php/
        # RedirectTemp cannot be used instead
    </IfModule>
</IfModule>

```

A continuación nuestro “*app.php*” (bootstrap) que lo encontraremos dentro de la carpeta “*web*” junto a nuestro *.htaccess*. Deberá contener más menos el siguiente código:

```

<?php
use Symfony\Component\HttpFoundation\Request;

/**
 * @var Composer\Autoload\ClassLoader
 */
$loader = require __DIR__.'/../app/autoload.php';
include_once __DIR__.'/../var/bootstrap.php.cache';

$kernel = new AppKernel('prod', false);
$kernel->loadClassCache();

$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
$kernel->terminate($request, $response);

```

El archivo **app.php** se encarga de recibir todas las peticiones de los usuarios a nuestro sitio, es la única puerta de acceso a nuestra aplicación, el objeto kernel (application) invoca el método **terminate(\$request, \$response)**, donde se procesa el request o petición y retorna una respuesta al usuario (response).

Listo, esto es lo mínimo para funcionar:

1. Por defecto queda definida donde está posicionado el sistema.
2. Por defecto queda definida la ruta para encontrar la librería del Symfony.
3. Luego definimos cual es la ruta donde se deben buscar los controllers, por defecto buscará un archivo con el nombre **DefaultController.php** en el **directorio src\AppBundle\Controller** e invocará el método acción **indexAction()**, como veremos luego esto se configura vía anotación sobre el método de acción:

```
/**
 * @Route("/", name="homepage")
 */
public function indexAction(Request $request)
{
    // replace this example code with whatever you need
}
```

Donde el primer parámetro de la anotación **@Route** es la ruta url (o mapeo) y el segundo en nombre de la ruta.

Algunas configuraciones previas de Symfony

Symfony no necesita casi ninguna configuración extra. Dado lo anterior somos libre de comenzar con nuestro desarrollo! Sin embargo, es posible que necesitemos revisar el archivo de configuración **"/app/config/config.yml"**. Que contiene varias opciones como la zona horaria y la configuración regional que es posible que deseemos cambiar de acuerdo a nuestros requerimientos y localización.

Otras configuraciones...

También podemos configurar nuestro locale dentro del archivo por ejemplo:

```
parameters:
    locale: en
```

De todas formas, para efectos del curso NO necesitaremos cambiar nada de esto, sólo lo mencionamos para que lo tengan presente a futuro.

Una vez que **Symfony** está instalado, también podemos acceder al **entorno de desarrollo** ([environment](#)). Esto nos permitirá recibir los mensajes de errores, es decir que se muestren en pantalla los mensajes detallados de errores cuando éstos ocurran en nuestro desarrollo. Simplemente accediendo a la URL:

http://localhost/symfony-project/web/app_dev.php

Más detalle en la documentación oficial de symfony:

<http://symfony.com/doc/current/book/configuration.html>

Nuestro “Controlador por Defecto”

Debemos tener la clase "*DefaultController.php*" que se encuentra dentro del directorio "**src\AppBundle\Controller**" del proyecto. Que contiene lo siguiente:

```
<?php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(Request $request)
    {
        // replace this example code with whatever you need
        return $this->render('default/index.html.twig', [
            'base_dir' => realpath($this->getParameter('kernel.root_dir').'/..'),
        ]);
    }
}
```

Lo que tenemos aquí es el controller por defecto, llamado `DefaultController` y extiende de la clase `Controller` toda la estructura y funcionalidad base que debe contar un `Controller` de Symfony.

Continuando con el ejemplo, tenemos un método de acción o “action” que maneja una solicitud HTTP o `Request`:

```
public function indexAction(Request $request)
{
    return $this->render('default/index.html.twig',
        ['base_dir' => realpath($this->getParameter('kernel.root_dir').'/..')]);
}
```

...que contiene una sentencia que retorna un objeto que representa a la vista **View**, en otras palabras es el objeto que comunica el controlador con la vista, en ella podemos pasar los datos o atributos a la vista y definir el nombre de la vista a cargar, los datos

que asignemos posteriormente a la vista, los veremos por ejemplo cómo `{{ base_dir }}`, es decir una variable local del motor de template o **vista Twig**.

Internamente en el método `$this->render('default/index.html.twig')`, el parámetro `'default/index.html.twig'` representa el nombre la vista **Twig** a cargar y se encarga de dibujar la plantilla en el navegador (Rendering Templates), ese decir poner el contenido de la vista en el objeto Response, lo explicaremos más abajo.



*Es necesario importar la clase con la cláusula **use**, la capacidad de referirse a un nombre completamente cualificado externo con un alias, o importar, es una característica importante de los espacios de nombres. Esto es similar a la capacidad de los sistemas de ficheros basados en Unix de crear enlaces simbólicos a un fichero o directorio, más [detalles aquí](#).*

Para pasar atributos o datos a la vista, simplemente en el método render pasamos como segundo argumento los datos como un arreglo asociativo, en el siguiente ejemplo pasando un título:

```
public function indexAction()
{
    // renders app/Resources/views/hola/index.html.twig
    return $this->render('hola/index.html.twig', array('titulo' => $titulo));
}
```

Por defecto, el nombre de la vista se tiene que pasar como primer parámetro en el método render, un ejemplo sería `render('hola-mundo.html.twig')` por lo tanto buscará la vista **hola-mundo.html.twig** en el directorio **app/Resources/views**:

```
public function indexAction()
{
    // buscará y cargará la vista hola-mundo.html.twig
    // ubicada dentro del directorio views en app (app/Resources/views)
    return $this->render('hola-mundo.html.twig');
}
```


Cuando retornamos e invocamos el método **render**, parte de la capa de la vista o presentación (a través del motor de plantilla **Symfony\Bundle\TwigBundle**), Symfony usará este objeto para dibujar e invocar el **render del motor de plantilla (o templating Engine)** asignando las variables al template o plantilla y finalmente pone todo el contenido resultante en el objeto Response.

Nuestra vista default/index.html.twig

Ahora, solo nos queda estudiar **el archivo de vista con extensión .html.twig** (extensión del motor de plantilla [Twig de Symfony](#)). Observamos un archivo de nombre **"index.html.twig"** que lo encontramos dentro de la carpeta **"app\Resources\views\default"**.

Dentro del archivo de vista **index.html.twig** observamos el siguiente código por defecto:

```
<html>
{% extends 'base.html.twig' %}

{% block body %}
    <div id="wrapper">
        <div id="container">
            <div id="welcome">
                <h1><span>Welcome to</span> Symfony {{
constant('Symfony\Component\HttpKernel\Kernel::VERSION') }}</h1>
            </div>
            <div id="status">
                <p>
                    ETC...
                    Your application is now ready. You can start working on it at:
                    <code>{{ base_dir }}</code>
                </p>
            </div>
            ETC...
        </div>
    </div>
{% endblock %}
{% block stylesheets %}
<style>
    body { background: #F5F5F5; font: 18px/1.5 sans-serif; }
    ETC...
{% endblock %}
```

Todas las plantillas Twing deben utilizar la extensión. **html.twig**

Ahora, como ejercicio vamos a modificar el método **indexAction()** del controlador **DefaultController** para pasar el atributo título a la vista:

```
public function indexAction(Request $request)
{
    return $this->render('default/index.html.twig', [
        'base_dir' => realpath($this->getParameter('kernel.root_dir').'/..'),
        'titulo' => 'Hola mundo con Symfony',
    ]);
}
```

Nada nuevo! luego modificamos nuestra vista por defecto **default/index.html.twig** agregando el siguiente contenido:

ETC...

```
<div id="wrapper">
    <div id="container">
        <div id="welcome">
            <h1><span>Welcome to</span> Symfony {{
constant('Symfony\\Component\\HttpKernel\\Kernel::VERSION') }}</h1>
            <h3>{{ titulo }}</h3>
```

ETC...

```
{% extends 'base.html.twig' %}

{% block body %}
    <div id="wrapper">
        <div id="container">
            <div id="welcome">
                <h1><span>Welcome to</span> Symfony {{ constant('Symfony\\Component\\HttpKernel\\Kernel::VERSION') }}</h1>
                <h3>{{ titulo }}</h3>
            </div>
            <div id="status">
                <p>
                    <svg id="icon-status" width="1792" height="1792" viewBox="0 0 1792 1792" xmlns="http://www.w3.org/2000/
                    Your application is now ready. You can start working on it at:
                    <code>{{ base_dir }}</code>
                </p>
            </div>
            <div id="next">
                <h2>What's next?</h2>
                <p>
```

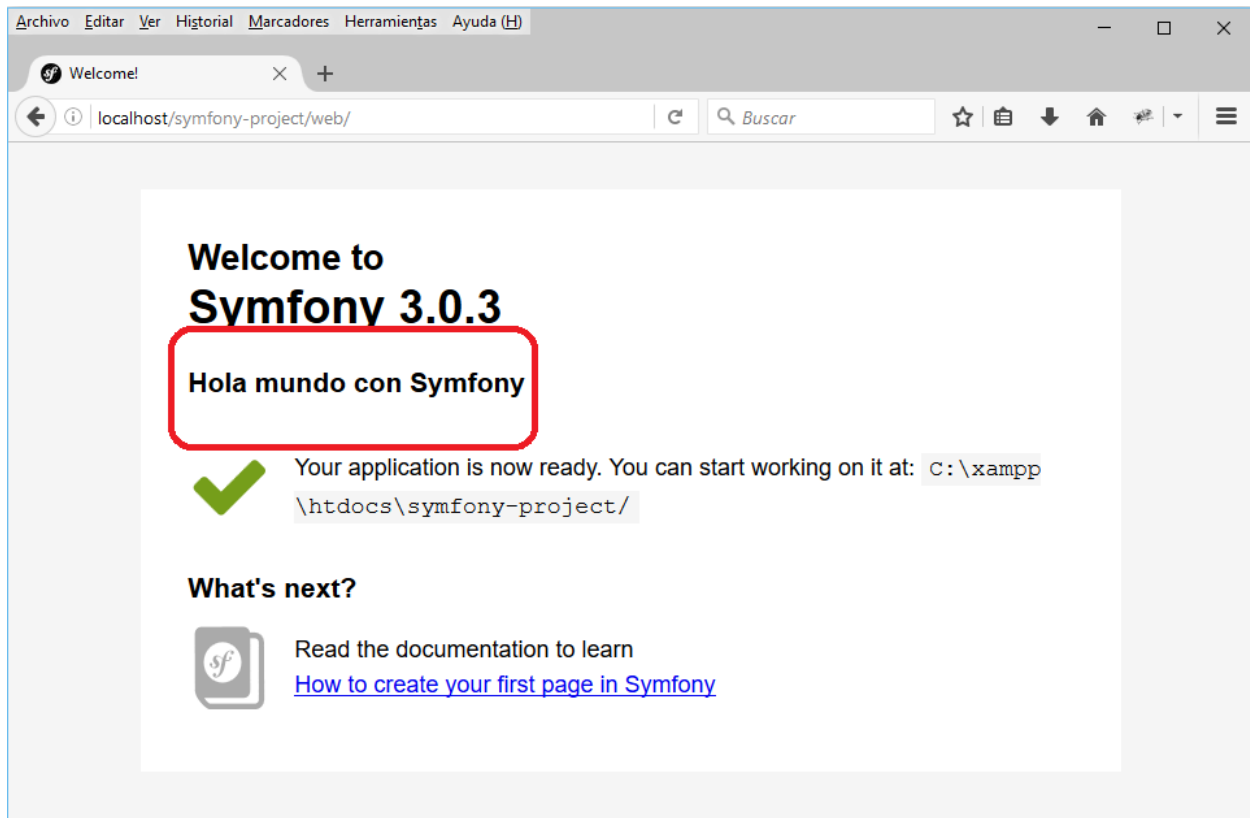


Es necesario e importante borrar el cache (ruta `\var\cache`) ante cualquier cambio o modificación de las vistas twig o plantillas, ya que éstas son compiladas en su primera ejecución y luego el resultado se almacena en el cache, lo mismo sucede con los archivos de configuraciones en `app\config`.

Si hicimos todo correctamente, lo probamos escribiendo la siguiente dirección en el navegador:

<http://localhost/symfony-project/web/>

¡Felicitaciones!, ¡hemos terminado nuestro primer ejemplo con Symfony “Hola Mundo”!.



Más adelante estudiaremos las plantillas propias de Symfony que utilizan el motor de plantillas [Twig](#), propio del framework, además veremos que utilizan un **Sistema de Layout con `{% extends 'base.html.twig' %}`**, todos estos detalles los veremos más adelante en sus respectivos módulos del curso, por ahora no hay que adelantarse!

Resumiendo, para que todo lo anterior funcione correctamente, debemos tener configurado las rutas usando la anotación **@Route** (sobre cada método de acción en el controller) los mapeos correspondientes de nuestro(s) controlador(es) (que veremos en detalle más adelante), es decir primero configurar el Controlador y luego mapear éste a una determinada URL o Routing.

En resumen

En este ejemplo logramos recibir una petición del exterior a través de la url **/symfony-project/web/**, que gracias a las rutas configuradas **mediante la anotación @Route** redirigió el control al controller por defecto **DefaultController**, y este le pasó los datos necesarios a la vista, y esta, se encargó de visualizar el valor recibido como si fuera una variable propia (creada dinámicamente).

Tarea: Intenta seguir todos estos pasos y cumplir con este ejemplo, si tienes dudas o problemas, envía una consulta al foro.

Fin.

Envía tus consultas a los foros!

Aquí es cuando debes sacarte todas las dudas haciendo consultas en los foros correspondientes