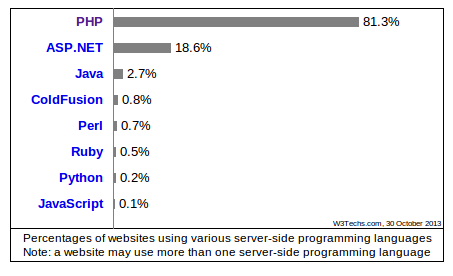
La evolución de PHP y los frameworks MVC

PHP nace en 1995 de la mano de Rasmus Lerdorf, en una época donde emergen otros lenguajes de tipado dinámico como Python o Ruby. Fue concebido para el *desarrollo de webs dinámicas* y a día de hoy es el lenguaje más extendido entre los servidores web, según [w3techs.com](http://w3techs.com/).



Lenguajes más usados por servidores web

Entre los motivos que explican la expansión de PHP podemos encontrar su suave curva de aprendizaje, la facilidad de instalación y configuración en cualquier entorno y la actual presencia de programadores en el mercado. Al ser un lenguaje interpretado, el despliegue de actualizaciones es tan sencillo como sobreescribir los ficheros existentes, por lo que el proceso de desarrollo se acelera con respecto a los lenguajes compilados como Java.

Pese a todos estos logros, PHP sigue estando muy mal considerado por amplios sectores de la comunidad de desarrolladores. ¿Por qué? Para responder a esta pregunta podemos partir de las decisiones en el diseño del lenguaje (un ejemplo, la cantidad y desorden de funciones para manejar arrays), pero también por su comunidad. Las mismas ventajas del lenguaje han atraído a multitud de programadores semi-profesionales o amateurs, y esto ha ayudado a generalizar las malas prácticas entre los proyectos PHP. Plataformas como Drupal o Wordpress [han extendido en gran medida esta imagen](https://api.drupal.org/api/drupal/modules!user!user.module/function/user_save/7).

Podemos decir, sin embargo, que la comunidad PHP ya ha superado el punto de inflexión y empieza a remontar. Las versiones 5.0 y posteriores han intentado enmendar los errores del pasado con algunos cambios importantes, como el soporte para orientación a objetos y closures, entre otros.

Uno de las mayores transformaciones ha sido la aparición de diversos frameworks de desarrollo que ha impulsado la estandarización y difusión de buenas prácticas entre la comunidad de desarrolladores. Entre esos frameworks, Symfony en su segunda versión es tal vez el más influyente en la actualidad.

|  |  |  |  |
| --- | --- | --- | --- |
| Framework | Lenguaje | Año de lanzamiento | Versión actual |
| CodeIgniter | PHP | 2002 | 2.1.3 |
| Ruby on Rails | Ruby | 2004 | 4.0.0 |
| CakePHP | PHP | 2005 | 2.4.1 |
| Symfony | PHP | 2005 | 2.3.3 |
| Django | Python | 2005 | 1.5.5 |
| Turbogears | Python | 2005 | 1.0 |
| Zend | PHP | 2006 | 2.2.4 |
| web2py | Python | 2007 | 2.6.3 |
| Sinatra | Ruby | 2007 | 1.4.2 |
| Yii | PHP | 2008 | 1.1.14 |
| Tornado | Python | 2009 | 3.0 |
| Padrino | Ruby | 2010 | 0.11.10 |
| Laravel | PHP | 2013 | 4.0.7 |
| Silex | PHP | 2010 | 1.1 |

Fuente: [Comparison of web application frameworks, Wikipedia](http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks)

De entre los freworks, podemos distinguir los monolíticos y los basados en componentes. Symfony 2 pertenece al segundo grupo.

Ventajas e inconvenientes de los frameworks

Aunque los frameworks de desarrollo suponen por lo general una gran ventaja en la mayoría de proyectos, también tienen algunos inconvenientes.

Ventajas

Productividad

Los frameworks proporcionan soluciones prefabricadas para los problemas más comunes. Atención de peticiones, formularios e interacción con base de datos son ejemplos de soluciones que casi todos los frameworks ofrecen. Esto permite a los desarrolladores centrarse en las necesidades de negocio sin tener que resolver los detalles técnicos de más bajo nivel.

Organización

Un framework, normalmente, ofrece una estructura clara y organizada a varios niveles, como la estructura de directorios y la separación por capas. En consecuencia es más fácil saber dónde encontrar cualquier recurso cuando sea necesario cambiarlo.

Convención

Relacionado con el punto anterior, alrededor de un framework se construye una *manera de resolver las cosas*, un estilo. Empezando por los coding styles hasta patrones de diseño concretos, los desarrolladores pueden y deben acogerse a esas convenciones. Así, cualquier desarrollador habituado al framework pueda integrarse en cualquier proyecto con mayor facilidad.

Documentación

Todos los frameworks disponen de un sitio web con documentación más o menos completa, bloggers que comparten sus soluciones, vídeos, charlas y conferencias.

Seguridad

Los problemas de seguridad suelen estar resueltos por el mismo framework. En el caso de descubrir amenazas que puedan comprometer la seguridad, es la misma comunidad la que los soluciona de manera más rápida y eficaz que si lo hicieras tú o tu departamento.

Rendimiento

Cuantos más desarrolladores utilicen el framework, más ojos hay pendientes del rendimiento que les ofrece. Por ello, los frameworks suelen ofrecer implementaciones más rápidas de las que pueda implementar un desarrollador individual.

Comunidad

Muchas de las ventajas anteriores serían imposibles sin la existencia de las comunidades. Éstas proveen, además, multitud de módulos, plugins, bundles o gemas de manera libre y gratuíta. Antes de enfrentarse a cualquier problema conviene realizar una búsqueda en internet. Seguramente alguien ya lo haya resuelto.

Por otra parte, las comunidades suelen organizarse en grupos locales y reunirse en eventos nacionales e internacionales. ¡Son una buena oportunidad de conocer gente interesante y apasionada!

Desventajas

Rendimiento (recursos de proceso y memoria)

Los frameworks consumen, en general, más recursos que una aplicación ad-hoc orientada al rendimiento. En aplicaciones muy exigentes, un framework puede resultar poco apropiado.

Curva inicial de aprendizaje

Cada framework tiene su ecosistema de componentes que el desarrollador debe aprender, no basta con conocer el lenguaje sobre el que está escrito. Por ello, los frameworks son islas de conocimiento.

Convención

Aunque normalmente las convenciones constituyen una ventaja, en ocasiones también pueden resultar un impedimento. Algunas veces, ante problemas muy concretos, el establecimiento de convenios obliga a los desarrolladores a *esquivar* al framework. Algunos desarrolladores sienten también cierta falta de libertad y creatividad al utilizar frameworks muy orientados a los convenios, como Ruby on Rails.

Sensación de bala de plata

A medida que un desarrollador conoce el framework, se introduce en una *zona de confort*. A la larga es posible que el desarrollador piense que su framework es la mejor solución para todo, sin estudiar otras alternativas. Por ello es muy recomendable actualizarse constantemente y conocer otros frameworks y plataformas que enriquezcan nuestra *caja de herramientas*.

¿De verdad necesito un framework?

Ante el impulso inicial de los grandes frameworks monolíticos han surgido alternativas que proporcionan capas más finas de funcionalidad y ofrecen una mayor flexibilidad al desarrollador; son los llamados *microframeworks*. Entre ellos tenemos Sinatra (Ruby), Flask (Python) o Silex (PHP).

Gracias a los gestores de componentes, como Composer en PHP, es sencillo construirse aplicaciones a medida. Por ejemplo, una aplicación PHP podría tomar el componente de inyección de dependencias Pimple, el Event Dispatcher de Symfony 2 y cualquier otro componente que se proporcione aislado. Esta solución puede ser más apropiada para necesidades de negocio complejas o desarrolladores exigentes.

Como ejemplo de esta segregación en paquetes, es interesante estudiar el caso de [The Aura Project](http://auraphp.com/).

Otros recursos sobre ventajas y desventajas de los frameworks: - [Software Framework Advantages and Disadvantages](http://nagbhushan.wordpress.com/2010/10/03/framework-advantages-and-disadvantages/) - [Pros and cons of using frameworks](http://www.1stwebdesigner.com/design/pros-cons-frameworks/)

¿Qué es Symfony?

Qué es Symfony

De acuerdo a la [definición de Symfony en su propia web](http://symfony.com/what-is-symfony), Symfony es *un framework PHP, una filosofía y una comunidad*. En su artículo [What is Symfony2?](http://fabien.potencier.org/article/49/what-is-symfony2), Fabien Potencier se extiende un poco más en la definición de Symfony. Según Fabien, leemos que, por una parte...

Symfony2 is a reusable set of standalone, decoupled, and cohesive PHP components that solve common web development problems.

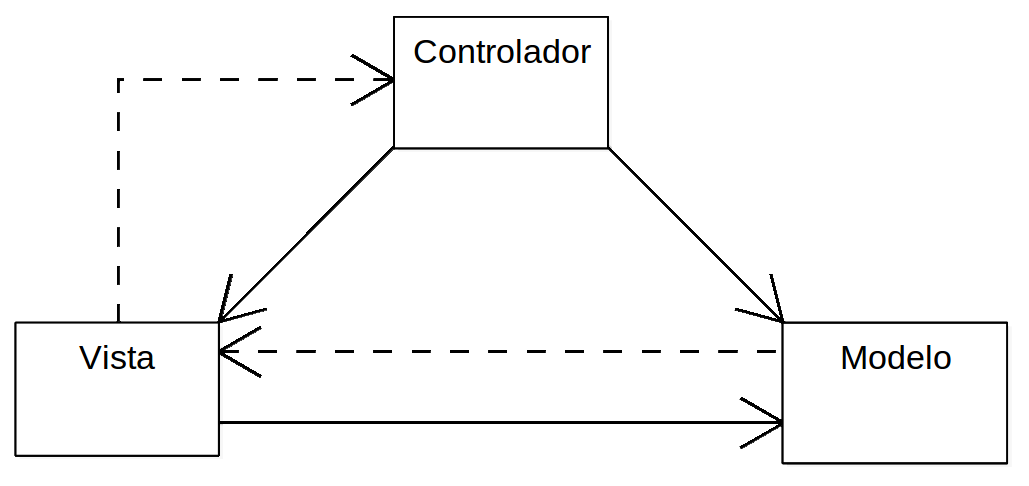
... y por otra ...

Based on these components, Symfony2 is also a full-stack web framework.

Es decir, hay varias maneras de utilizar Symfony en los proyectos PHP. La más obvia consiste en construir nuestra aplicacion sobre el framework Symfony 2 al completo, pero si lo deseamos también podemos utilizar únicamente algunos de sus componentes.

HTTP Framework

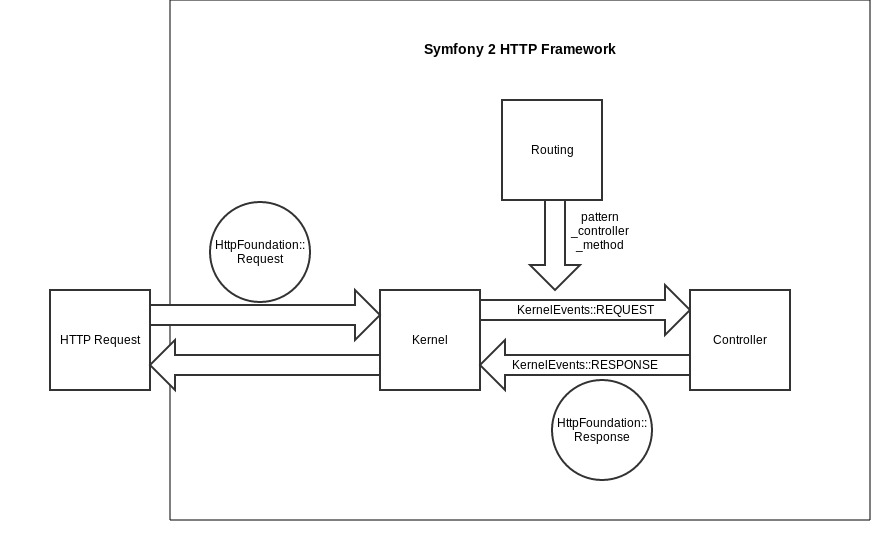
A menudo, Symfony 2 es definido como un *framework MVC*. El patrón MVC consiste en separar en capas distintas los componentes encargados de manejar la vista, el modelo y el controlador.



Arquitectura MVC

Aunque Symfony 2 comparte algunos de los conceptos del patrón MVC (separación por capas), su objetivo es otro; atender peticiones HTTP de una manera organizada y eficaz. Por ello, Symfony 2 se define como un *framework HTTP*.

Symfony abstrae la petición HTTP en un objeto Request que es procesado por el framework. Para ello intervienen varios componentes; el enrutado, el controlador responsable de dicha petición y el Event Dispatcher. La forma en que esté organizado el modelo depende completamente de nosotros. Podemos devolver contenido HTML o respuestas en JSON, XML, o cualquier otro formato. Por lo tanto, ni el modelo ni la vista dependen en absoluto del framework.

Symfony2 HTTP Framework

Las ventajas de esta arquitectura son innumerables. Al abstraer la petición PHP en un objeto response, el framework ya no depende de las históricas variables PHP como $\_SESSION, $\_SERVER, $\_POST o $\_GET. Esto permite crear peticiones programáticamente y pasárselas al kernel sin necesidad de emplear peticiones *reales*. De esta manera es posible utilizar aplicaciones Symfony desde distintos entornos, como programas externos o tests automáticos.

Por otra parte y gracias a su sistema de eventos, el framework Symfony permite a los desarrolladores intervenir en cualquier punto de la petición para transformar los datos o realizar operaciones en paralelo.

Comunidad

Ninguna plataforma open-source sería nada sin su comunidad. Las comunidades de software libre son lugares excelentes donde aprender de los demás, recibir y aportar nuevos puntos de vista y, en definitiva, pasar un buen rato.

Además de los eventos internacionales, en España se celebra anualmente la conferencia [deSymfony](http://desymfony.com/) donde se citan los mejores desarrolladores del framework. También se han organizado otros grupos a nivel local, entre los que tenemos el grupo local de Valencia, [SymfonyVLC](http://www.symfony-valencia.es/).

Instalación

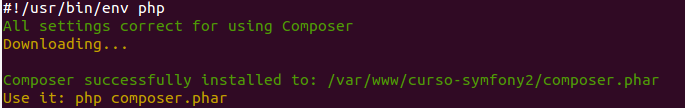
Aunque Symfony puede ser descargado y descomprimido directamente allí donde lo queramos dejar, en este curso vamos a utilizar desde el principio el gestor de dependencias *composer*. Esta guía describe el proceso de instalación en una máquina basada en Debian. Para otros sistemas operativos, consultad la [guía de instalación](http://symfony.com/doc/current/book/installation.html) en la web del framework.

Cómo descargar composer

Situaos en el directorio desde el que pendan los distintos sites del servidor. Por ejemplo, /var/www.

$ sudo apt-get install curl

$ curl -sS https://getcomposer.org/installer | php



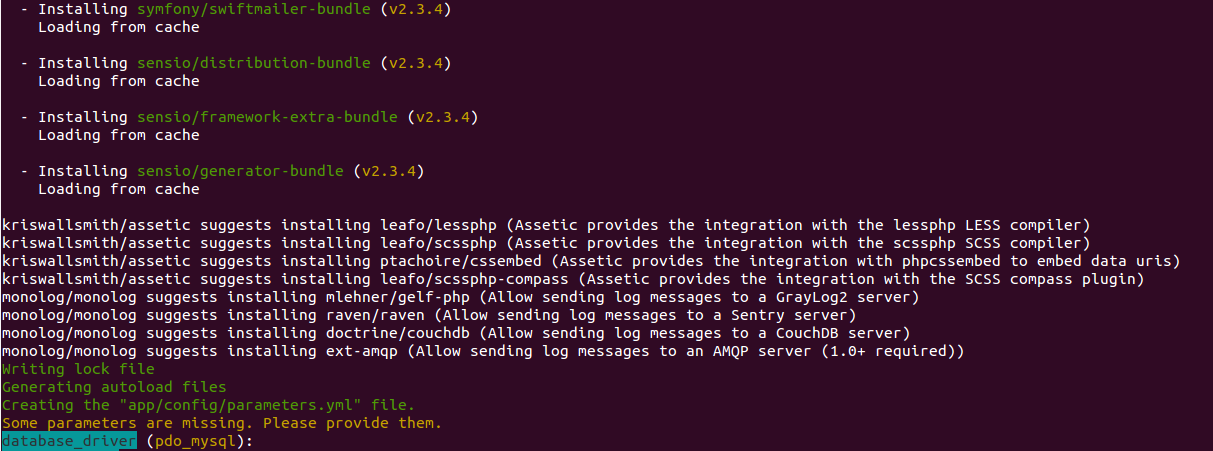
Instalación de Composer

Cómo instalar Symfony 2

En el mismo directorio donde hayáis descargado composer, ejecutad la siguiente instrucción:

$ php composer.phar create-project symfony/framework-standard-edition nombre-de-mi-proyecto/

Tras descargar los componentes necesarios, el terminal nos pedirá interactivamente que le proporcionemos cierta información:

Configuración de Symfony

* *database\_driver*: Configura el motor de base de datos a utilizar para la instalación. Algunas opciones son pdo\_mysql o pdo\_sqlite. Recuerda que necesitarás tener instaladas en tu equipo las extensiones correspondientes.
* *database\_host*: La máquina donde se aloja la base de datos. Por defecto la máquina local (127.0.0.1).
* *database\_port*: El puerto mediante el cual se accede a la base de datos. Dejándolo a null se utilizará el puerto por defecto del motor elegido.
* *database\_name*: Nombre de la base de datos de la instalación.
* *database\_user*: Usuario con el que se accederá a la base de datos.
* *database\_password*: Contraseña del usuario de la base de datos.
* *mailer\_transport*: Protocolo de transporte a utilizar en el envío de emails. Algunas opciones son sendmail o smtp.
* *mailer\_user*: Si aplica, usuario a utilizar para el envío de emails.
* *mailer\_password*: Si aplica, contraseña del usuario para el envío de emails.
* *local*: Localización idiomática del sitio. Afecta, por ejemplo, al modo en el que se formatean las fechas y a los ficheros de traducción cargados.
* *secret*: Esta clave es utilizada por Symfony en sus mecanismos de encriptación. ¡No te olvides de cambiarla!.

Una vez proporcionados los parámetros necesarios daremos permisos de escritura a los directorios app/cache y app/logs [tal y como se describe en la web oficial](http://symfony.com/doc/current/book/installation.html" \l "configuration-and-setup).

$ APACHEUSER=`ps aux | grep -E '[a]pache|[h]ttpd' | grep -v root | head -1 | cut -d\ -f1`

$ sudo setfacl -R -m u:$APACHEUSER:rwX -m u:`whoami`:rwX app/cache/ app/logs/

$ sudo setfacl -dR -m u:$APACHEUSER:rwX -m u:`whoami`:rwX app/cache/ app/logs/

Ejemplo de configuración en Apache web server

<VirtualHost \*:80>

ServerAdmin mi@mail.es

ServerName local.symfony.com

DocumentRoot /var/www/vhosts/symfony/web

<Directory /var/www/vhosts/symfony/web>

AllowOverride None

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^(.\*) app.php [QSA,L]

</Directory>

ErrorLog ${APACHE\_LOG\_DIR}/symfony-error.log

# Possible values include: debug, info, notice, warn, error, crit,

# alert, emerg.

LogLevel warn

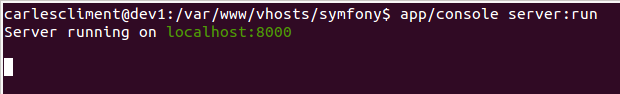
CustomLog ${APACHE\_LOG\_DIR}/access.log combined

</VirtualHost>

Servidor embebido

Si no deseamos perder tiempo con un servidor de desarrollo y disponemos de una versión de PHP 5.4 o superior podemos utilizar el [servidor embebido de PHP](http://www.php.net/manual/en/features.commandline.webserver.php). Este servidor está pensado para entornos de desarrollo y nunca debería usarse en entornos de producción.

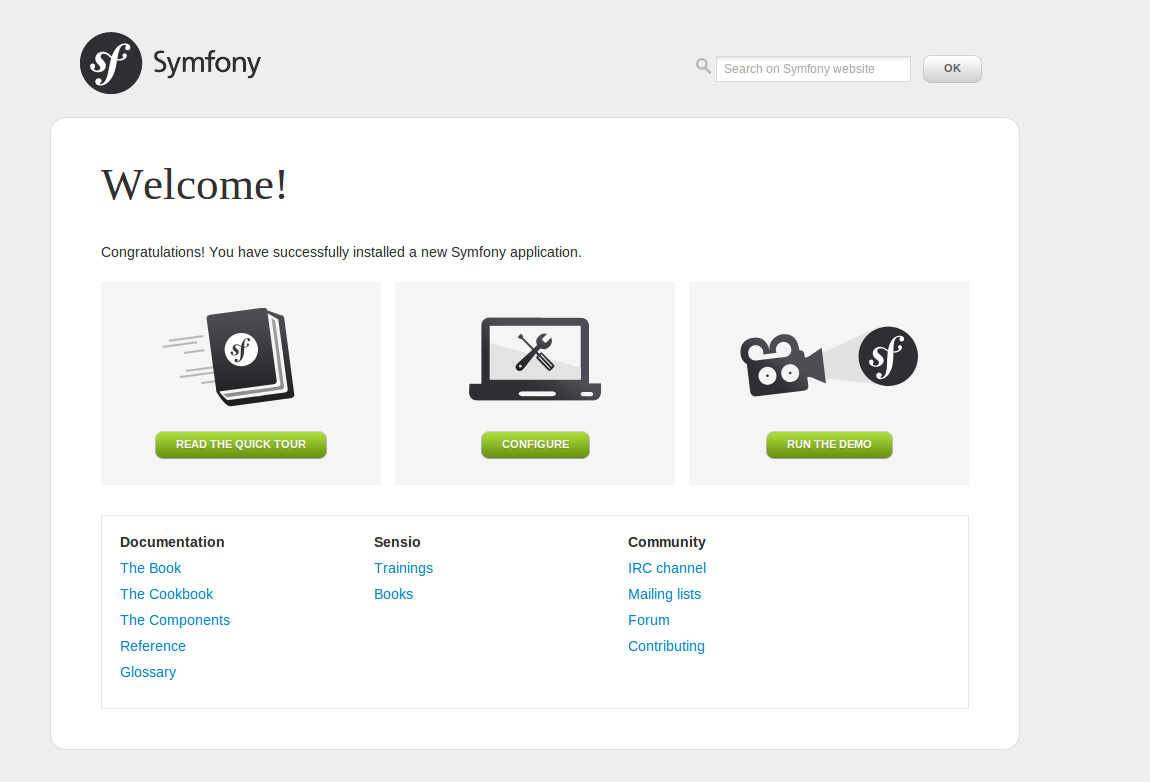
La consola de Symfony 2 proporciona un comando para arrancar nuestra aplicación Symfony; app/console server:run.



Builtin server

¡Bienvenido!

Tras configurar el servidor web para que dirija correctamente las peticiones al sitio podremos acceder a él desde nuestro navegador en http://tu-site-symfony.com/app\_dev.php/ o localhost:8000 si hemos arrancado el servidor embebido.



Pantalla de bienvenida

Tómate tu tiempo curioseando tu instalación. ¡Enhorabuena!.

Organización de directorios

Una instalación de Symfony tiene una estructura similar a la siguiente:

Directorios

En /app se encuentran los archivos correspondientes a la aplicación:

* AppKernel.php define qué *bundles* hay instalados en nuestra instalación. Cada vez que queramos instalar un nuevo bundle deberemos incluirlo en el método registerBundles() de esta clase.
* config almacena los distintos archivos de configuración de la aplicación. Esto incluye los *parámetros* de la aplicación según entorno, los *servicios* incluídos y los *enrutadores* y *firewalls* instalados.
* cache es el directorio por defecto en el que Symfony almacena algunos datos para optimizar el rendimiento de la caché.
* logs contiene los registros de actividad para cada entorno.
* console es un binario que contiene la consola de Symfony, útil para realizar algunas operaciones. La veremos en próximos capítulos.
* Resources almacena recursos de distinta índole, ya sean *plantillas*, *fixtures* o librerías de diversa índole.

En /bin se almacenan ejecutables destinados a ser invocados desde terminal.

/src contiene nuestros propios bundles. Es decir, los componentes (controladores, rutas, entidades, modelos, vistas...) escritos por nosotros.

Por último, en /web se deposita la parte pública de la aplicación web. Hojas de estilo, Javascripts y elementos estáticos como imágenes o vídeos.

Casi todos los elementos de esta estructura de directorios pueden configurarse, tal y como se explica en la [documentación oficial](http://symfony.com/doc/current/cookbook/configuration/override_dir_structure.html).

Gestión de paquetes con Composer

Qué es Composer

[Composer](http://getcomposer.org/) es una herramienta para la gestión de dependencias en PHP. A diferencia de un gestor de paquetes, la función de Composer no es instalar un paquete en un sistema operativo, sino gestionarlas dentro de una aplicación concreta.

Liberado en 2012, Composer ha supuesto una auténtica revolución en el mundo PHP, muy necesitado de esta herramienta. Supone la estandarización en la definición de las dependencias tal y como ya hicieron en NodeJS [npm](https://npmjs.org/) o Ruby [bundler](http://bundler.io/). Esta estandarización ha permitido el florecimiento de todo un ecosistema de bibliotecas base y para distintos frameworks que la han adoptado.

En Symfony, Composer fue introducido en su versión 2.1. En la versión anterior se utilizaba un gestor de dependencias propio.

Cómo definir un proyecto con Composer

Toda aplicación que utilice Composer debe contener en su raíz un archivo composer.json que defina sus atributos básicos. Por ejemplo:

{

"name": "carlescliment/html2pdf-service",

"description": "A REST microservice that converts html input into pdf files. Written in Silex.",

"version": "0.0.2",

"type": "project",

"keywords": ["printing", "pdf"],

"license": "GPL-2.0",

"authors": [

{

"name": "Carles Climent Granell",

"email": "carlescliment@gmail.com",

"homepage": "http://www.carlescliment.com",

"role": "Developer"

}

],

"require": {

"php": ">=5.3.2",

# ...

},

"require-dev": {

"symfony/browser-kit": ">=2.3,<2.4-dev",

"phpunit/phpunit": "3.7.\*"

},

"autoload": {

"psr-0": { "carlescliment\\Html2Pdf": "src" }

},

"minimum-stability" : "stable"

}

De este ejemplo podemos extraer que: \* El nombre del paquete es html2pdf-service, y el vendor (autor) es carlescliment. \* Se encuentra en la versión 0.0.2. \* Es de tipo "project". Los tipos válidos estándar son; library (por defecto), project, metapackage y composer-plugin. Otros tipos personalizados también son admitidos, como wordpress-plugin. \* Se han añadido dos palabras clave printing y pdf para facilitar su búsqueda. \* Está liberado bajo licencia GPL 2. \* Su autor es Carles Climent Granell. \* Tiene algunas dependencias necesarias para su puesta en producción (require) y otras para entornos de desarrollo (require-dev). \* Utiliza una estrategia de autocarga definida por el estándar PSR-0. \* Sólo admite dependencias estables.

Veamos cómo funciona con mayor detenimiento.

Cómo establecer dependencias

Para establecer una dependencia introduciremos el nombre del paquete o biblioteca en una clave *"require"*.

{

"require": {

"php": ">=5.3.2",

"monolog/monolog": "1.2.\*"

}

}

En la parte izquierda especificaremos el *nombre del paquete*, mientras que en la derecha especificamos la versión.

Las versiones pueden especificarse de distintas maneras:

|  |  |  |
| --- | --- | --- |
| Match | Ejemplo | Descripción |
| Exacto | 1.0.2 | Descarga la versión especificada y solo esa. |
| Rango | >=1.0 | Versión igual o mayor que 1.0 |
| Rango | >=1.0,!=1.4 | Versión igual o mayor que 1.0, excepto la 1.4. La coma se interpreta como un AND. |
| Rango | >=1.0,<2.0 | Versiones entre 1.0 y 2.0 |
| Rango | >=1.0,<1.1 | >=1.2 | Versiones entre 1.0 y 1.1, o mayor que la 1.2. La tubería se interpreta como un OR |
| Comodín | 1.0.\* | Equivale a >=1.0,<1.1 |
| Tilde | ~1.2 | Equivale a >=1.2,<2.0 |

Por defecto, solo se tendrán en cuenta las versiones estables de cada paquete. Este comportamiento puede ser modificado mediante la clave "minimum-stability". Las opciones son dev, apha, beta, RC y stable.

{

# ...

"minimum-stability": "dev"

}

Una vez especificadas las dependencias, basta con instalarlas ejecutando php composer.phar update. Tras ejecutar el comando se iniciará la descarga de las dependencias, se generará un autoloader y se creará un archivo composer.lock.

Autoloading

Composer crea un archivo vendor/autoload.php que contiene la carga automática de clases y espacios de nombres de aquellos paquetes que lo necesiten. Esto te permitirá incluir fácilmente estos paquetes en tu proyecto añadiendo simplemente un *require* en tu aplicación:

require 'vendor/autoload.php';

Para cargar tu propio código en el autoloader puedes añadir la etiqueta autoload en composer.json.

{

# ...

"autoload": {

"psr-0": { "carlescliment\\Html2Pdf": "src" }

}

}

El Lock File

El archivo composer.lock contiene las versiones instaladas actualmente en la aplicación. Si composer encuentra este archivo, descargará exactamente las versiones definidas en el archivo. Es recomendable añadir el archivos composer.lock al repositorio de versiones. De esta manera, cualquiera que se descargue el código instalará las mismas versiones que quien añadió el archivo. ¡Pensad por ejemplo en los sistemas de integración continua!.

En el caso de que Composer no encuentre un composer.lock, generará uno nuevo a partir del composer.json.

Packagist

Si queremos publicar un proyecto para que otros puedan descargarlo via Composer, debemos dotarlo de un archivo composer.json válido y darlo de alta en la base de datos de la web [Packagist](https://packagist.org/). La mayoría de los gestores de versiones del mercado permiten disparar de manera automática acciones (hooks) que informarán a Packagist de nuevas versiones.

Es posible añadir otras fuentes, además de Packagist. [Satis](https://github.com/composer/satis), por ejemplo, es una herramienta que permite crearnos un \*Packagist privado". Resulta muy útil cuando tenemos dependencias de código que no deseamos publicar. Para más información sobre cómo gestionar repositorios privados con composer ver el siguiente enlace:

[Handling private packages with Satis](https://github.com/composer/composer/blob/master/doc/articles/handling-private-packages-with-satis.md)

El profiler y la consola

La instalación Symfony proporciona dos útiles herramientas que todo desarrollador utiliza tarde o temprano. El profiler servirá para proporcionar diversa información sobre lo que ocurre en cualquier petición. La consola, además de para extraer alguna información, permitirá realizar acciones desde el terminal, facilitando la automatización de procesos.

Profiler

El profiler es un componente de Symfony que recoge información de cada petición que recibe la aplicación y la almacena para su análisis posterior. En la edición estándar de Symfony2, el profiler y las herramientas incorporadas Web Debug Toolbar y Web Profiler están activadas para el entorno de desarrollo.

Podemos (des)activar y personalizar el profiler a través del archivo de configuración config.yml.

web\_profiler:

toolbar: true

position: bottom

# gives you the opportunity to look at the collected data before following the redirect

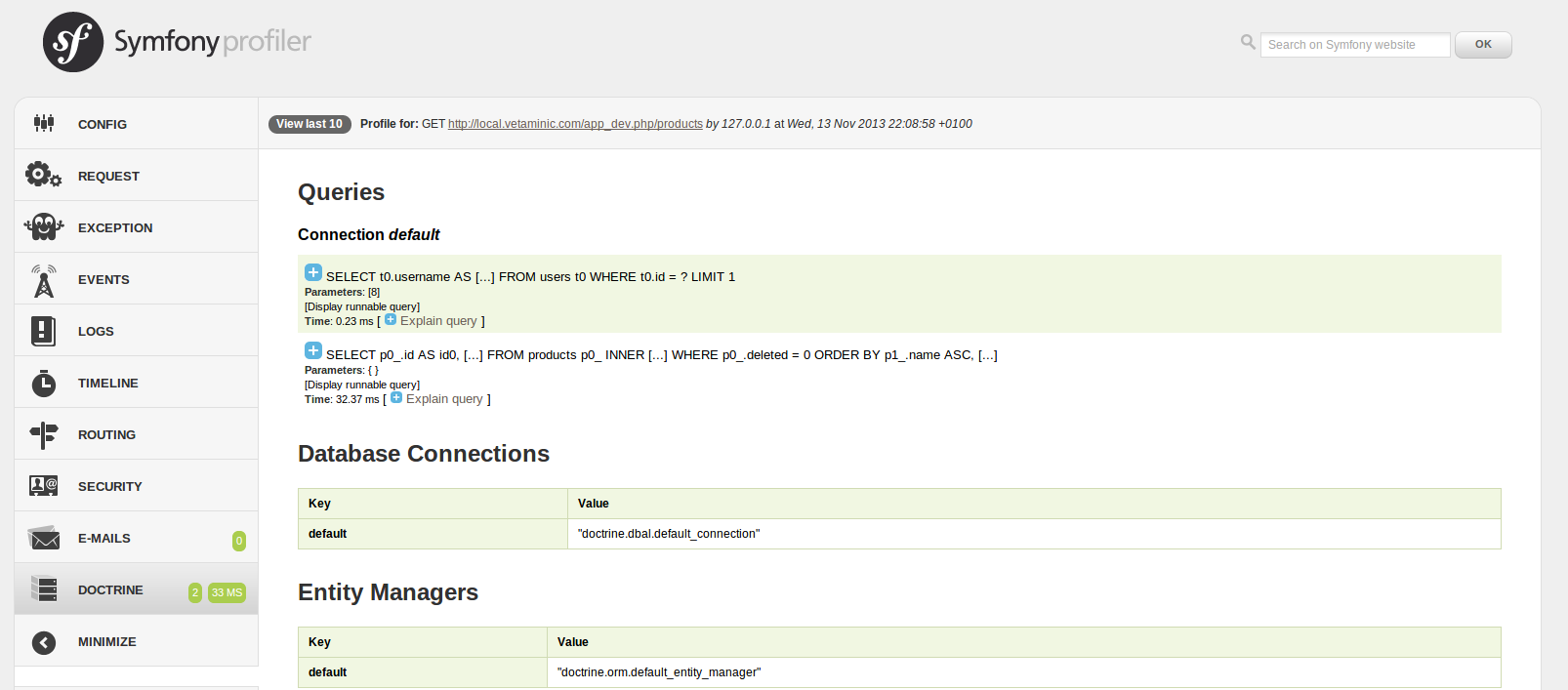
intercept\_redirects: false

Una vez activado en un entorno, ante cualquier petición se nos mostrará una útil barra de herramientas con información diversa.

Web Debug Toolbar

Web Debug Toolbar

Los distintos iconos se expandirán tras seleccionarlos y mostrarán información más detallada. A continuación se muestra el detalle de consultas realizadas a la base de datos en una petición.



Web Profiler

El profiler en tests funcionales

El profiler de Symfony puede utilizarse para automatizar pruebas de rendimiento. Por ejemplo, se podría implementar una batería de tests que recorriese una aplicación y se asegurase de que no se sobrepasa un número de consultas determinado. Para obtener más información sobre cómo implementar estas pruebas, consultad la receta [How to use the Profiler in a Functional Test](http://symfony.com/doc/current/cookbook/testing/profiling.html) en la documentación oficial.

El profiler y el rendimiento

La activación del profiler genera un impacto profundo en el rendimiento de la aplicación. La recolección de los datos, procesamiento y posterior almacenamiento en ficheros temporales hacen del profiler una herramienta peligrosa en entornos de producción. Lo mejor, por ello, es activarlo únicamente en el entorno de desarrollo.

Del mismo modo, conviene desactivar el profiler en todos los tests funcionales en los que no sea estrictamente necesario, permitiendo baterías de tests más rápidas que agilicen los procesos de integración.

Consola

La consola de Symfony 2 proporciona una interfaz de terminal para operar con nuestra aplicación Symfony. Para comprobar los comandos disponibles debemos ejecutar app/console sin parámetros. El número de comandos disponibles variará en función de los bundles que hayamos instalados, puesto que estos pueden registrar acciones en la consola.

Los comandos de la sección generate permiten automatizar la creación de bundles, controladores o entidades de doctrine. Es el llamado scaffolding o andamiaje.

Otros comandos imprescindibles de la consola son aquellos que permiten gestionar la caché:

app/console cache:clear eliminará los archivos de caché. app/console cache:warmup efectuará el calentamiento de una caché vacía, mejorando el rendimiento de la web antes de que lleguen peticiones.

Casi todos los comandos permiten especificar el entorno en el que van a ser utilizados con el parámetro -e. El siguiente comando se ejecutará únicamente en el entorno de test:

app/console doctrine:schema:create -e test

En una instalación estándar de Symfony 2 hay demasiados comandos para ser tratados aquí individualmente. La mejor referencia de cada uno de ellos se encuentra en la propia interfaz de ayuda de la consola. Para una información más extensa y actualizada [consulta la documentación oficial](http://symfony.com/doc/current/components/console/introduction.html).