1 Slim Framework - PHP

1.1 Sumario

2 Slim framework

Slim framwework es un **micro framework** para PHP que nos permite escribir rápidamente aplicaciones web y APIs.

Para comenzar a utilizar frameworks y aprender MVC (Modelo Vista Controlador) es muy recomendable ya que se pueden hacer aplicaciones muy interesantes con muy poco código. No tiene la potencia de otros frameworks de PHP como Laravel, Code Igniter, etc.. pero hace bastante bien su trabajo.

Página web oficial: http://www.slimframework.com/

Documentación del framework: http://www.slimframework.com/docs/v4

2.1 Características de Slim Framework

- Creador de rutas bastante potente
  - Soporta métodos HTTP standard y personalizados.
  - Parámetros de ruta con comodines y condiciones.
  - Redirecciones de rutas, paros y saltos.
- Renderizado de plantillas y vistas personalizadas.
- Mensajes Flash.
- Encriptación segura de cookies con AES-256.
- Caché HTTP.
- Logging de accesos personalizado.
- Gestión de errores.
- Configuración sencilla.

2.2 Requerimientos

Para la **versión 3** (**a Enero de 2020 ya existe la versión 4**), son necesarios los siguientes requisitos:

- Servidor web con la reescritura de URL activada.
- PHP 5.5 o superior.

2.3 Instalación

2.3.1 Slim Framework 3 - Windows

- Instalar la herramienta **Composer**: **https://getcomposer.org/download/**
- Descargar el ejecutable desde: https://getcomposer.org/Composer-Setup.exe

- Desde la carpeta dónde queramos ejecutar el proyecto ejecutaremos los comandos:

```
composer require slim/slim:"3.*"
```

2.3.2 Slim Framework 3 - Linux

- Instalar la herramienta **Composer**: **https://getcomposer.org/download/**

```
# Instalamos Composer en línea de comandos, en el terminal de Linux:

php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === 'e0012edf3e80b6978849f5eff0d4b4e4c79ff1609dd1e613307e16318854d24ae64f26d17
php composer-setup.php
php -r "unlink('composer-setup.php');"

# Movemos el fichero composer.phar a una ruta global que esté en el PATH:

mv composer.phar /usr/local/bin/composer

# A partir de este momento podemos ejecutar el comando composer desde cualquier directorio.
```

- Desde la carpeta dónde queramos ejecutar el proyecto ejecutaremos los comandos:

```
composer require slim/slim:"3.*"
```

Para su instalación simplemente se descomprime el .zip en la ruta dónde queramos dar servicio con este framework y lo único que nos interesa de ese zip es la **carpeta Slim** y el fichero **.htaccess** incluído en la carpeta principal.

El fichero **index.php** es un ejemplo interesante que estaría bien tenerlo para poder programar nuestra aplicación.

2.4 Ejemplo1 básico para probar la ejecución de Slim Framework

- En la carpeta dónde hemos instalado el framework, crearemos una nueva carpeta llamada /**proyecto1**/**public**
- Y dentro de **public** un fichero **index.php**

- Contenido del fichero /**proyecto1**/**public**/**index.php**:
- Antes de probar el ejemplo vea el siguiente apartado de Configuración

```
<?php
// proyecto1/public/index.php

use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;
use Slim\Factory\AppFactory;

# Ajustar a la carpeta dónde tengamos vendor.
require __DIR__ . '/../../vendor/autoload.php';

// Instanciamos la aplicación.
$app = new \Slim\App;

// Creamos la ruta por defecto
$app->get('/', function (Request $request, Response $response) {
    $response->getBody()->write("El framework Slim funciona !");
    return $response;
});

// Ejecutamos la aplicación.
$app->run();
```

2.5 Ejemplo2 básico para probar la ejecución de Slim Framework

- En la carpeta dónde hemos instalado el framework, crearemos una nueva carpeta /**proyecto2**/**public**
- Y dentro de /**proyecto2**/**public** un fichero **index.php**
- Contenido del fichero /**proyecto2**/**public**/**index.php**:
- Antes de probar el ejemplo vea el siguiente apartado de Configuración

```
<?php
// proyecto2/public/index.php

use Psr\Http\Message\ResponseInterface as Response;
use Psr\Http\Message\ServerRequestInterface as Request;
use Slim\Factory\AppFactory;

# Ajustar a la carpeta dónde tengamos vendor.
require __DIR__ . '/../../vendor/autoload.php';

// Instanciamos la aplicación.
$app = new \Slim\App;

// Creamos la ruta por defecto
$app->get('/', function (Request $request, Response $response) {
    $response->getBody()->write("Prueba a entrar en la siguiente ruta <a href='/hola/Marita'>/hola/Marita</a>");
    return $response;
});

// Otra ruta /hola/ que recibe un parámetro
$app->get('/hola/{parametro1}', function (Request $request, Response $response, array $args) {
    $nombre = $args['parametro1'];
    $response->getBody()->write("Hola <b>$nombre</b>, ¿qué tal estás?");

    return $response;
});

// Ejecutamos la aplicación.
$app->run();
```

2.6 Configuración del servidor web

2.6.1 Trabajando con NGINX

Si estamos trabajando con **Nginx** tendremos que configurar el **root directory** de nuestro servidor para que apunte a la carpeta pública del proyecto (1,2,3,...) que queramos probar.

```
# Como root

nano /etc/nginx/sites-enabled/veiga.dynu.net

# Sustituir la ruta raíz (root) por la carpeta pública de nuestro proyecto. Por ejemplo:

root /var/www/veiga.dynu.net/public/slimframework/proyecto1/public

# Reiniciar el servidor Nginx:
service nginx restart

# A partir de este momento ya nos podemos conectar a nuestro dominio y probar el ejemplo básico de funcionamiento:
http://veiga.dynu.net

El framework Slim funciona !
```

2.6.2 Trabajando directamente con el servidor web integrado de PHP

- Otra forma de trabajar con el framework **si no queremos tocar nuestra configuración de Nginx** sería algo similar a lo que hacemos con NodeJS.
- Consiste en arrancar un servicio web con la carpeta que deseemos como raíz.

```
# Nos situamos en la carpeta slimframework. Por ejemplo:
cd /var/www/veiga.dynu.net/public/slimframework

# Desde allí arrancamos el servidor web integrado con el lenguaje PHP con el siguiente comando:
# php -S 0.0.0.0:puerto [-t carpetapublica]
# ATENCIÓN: si tenemos NGINX funcionando y queremo
s usar el mismo puerto tendríamos que pararlo antes.
# O también podríamos arrancarlo en un puerto diferente, por ejemplo el 8080, para no chocar con el puerto de Nginx.

# Para escuchar en un puerto 8080 de la carpeta proyecto1/public
```

```
php -S 0.0.0.0:8080 -t proyecto1/public

# Para escuchar en el puerto 80 de la carpeta proyecto1/public
php -S 0.0.0.0:80 -t proyecto1/public

# Para escuchar en un puerto 8080 de la carpeta proyecto2/public
php -S 0.0.0.0:8080 -t proyecto2/public

# Si estuviéramos dentro de la carpeta public podríamos arrancarlo directamente con:
php -S 0.0.0.0:80

# Y se muestra la siguiente información:

PHP 7.3.14-1~deb10u1 Development Server started at Thu Feb 20 10:03:20 2020
Listening on http://0.0.0.0:80
Document root is /var/www/veiga.dynu.net/public/slimframework/proyecto1/public
Press Ctrl-C to quit.

# A partir de este momento ya nos podemos conectar a nuestro dominio y probar el ejemplo básico de funcionamiento:
http://veiga.dynu.net

El framework Slim funciona !
```

Veiga (discusión) 10:17 21 feb 2020 (CET)