1 API REST Creación con Slim Framework

1.1 Sumario

2 Introducción a REST

REST al igual que otras tecnologías, como Git y CSS, requiere un poco de tiempo para su comprensión. En este breve manual nos centraremos en la filosofía que está detrás de REST(así como sus diferencias con SOAP), y en los aspectos prácticos. ¿Cómo podemos implementar REST hoy?

2.1 ¿Que es es REST?

REST son las siglas de **Representational State Transfer**. Fue definido hace una década por Roy Fielding en su tesis doctoral, y proporciona una forma sencilla de interacción entre sistemas, la mayor parte de las veces a través de un navegador web y HTTP. Esta cohesión con HTTP viene también de que Roy es uno de los principales autores de HTTP.

REST es un estilo arquitectónico, un conjunto de convenciones para aplicaciones web y servicios web, que se centra principalmente en la manipulación de recursos a través de especificaciones HTTP. Podemos decir que REST es una interfaz web estándar y simple que nos permite interactuar con servicios web de una manera muy cómoda.

Gracias a REST la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- Un **protocolo cliente/servidor sin estado**: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST).
- Un **conjunto de operaciones bien definidas** que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son **POST**, **GET**, **PUT** y **DELETE**. Con frecuencia estas operaciones se equiparan a las operaciones CRUD que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.

- Una **sintaxis universal para identificar los recursos**. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

- El **uso de hipermedios**, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

Vamos a ver primero lo que son las URIs. Una URI es esencialmente un identificador de un recurso. Veamos el siguiente ejemplo:

```
GET /amigos
```

Podríamos llamar a esto un recurso. Cuando esta ruta es llamada, siguiendo los patrones REST, se obtendrán todos los amigos (generalmente de una base de datos), y se mostrarán en pantalla o se devolverán en un formato determinado a quien lo solicite.

Pero, cómo haremos si queremos especificar un amigo en particular?.

```
GET /amigos/marta
```

Como se puede ver es fácilmente comprensible. Esa es una de las claves de la arquitectura RESTful. Permite el uso de URIs que son fácilmente comprensibles por los humanos y las máquinas.

Piensa en un recurso como un nombre en plural. Contactos, estados, usuarios, fotos --- todos éstos serían elecciones perfectas.

Hasta ahora, hemos visto como identificar a una colección y a elementos individuales en esa colección:

```
GET /amigos
GET /amigos/marta
```

De hecho, encontrarás que estos dos segmentos son todo lo que tendrías que haber necesitado siempre. Pero podemos profundizar un poco más en la potencia de HTTP para indicar cómo queremos que el servidor responda a esas peticiones. Veamos:

Cada petición HTTP especifica un método, o un verbo, en sus encabezados. Generalmente te sonarán un par de ellos como GET y POST.

Por defecto el verbo utilizado cuando accedemos o vemos una página web es **GET**.

Para cualquier URI dada, podemos referenciar hasta 4 tipos diferentes de métodos: **GET, POST, PUT y DELETE**.

```
GET /amigos
POST /amigos
PUT /amigos
DELETE /amigos
```

Esencialmente, estos verbos HTTP indican al servidor que hacer con los datos especificados en la URI. Una forma fácil de asociar estos verbos a las acciones realizadas, es comparándolo con **CRUD** (Create-Read-Update-Delete).

```
GET => READ
POST => CREATE
PUT => UPDATE
DELETE => DELETE
```

Anteriormente hemos dicho que GET es el método utilizado por defecto, pero también te debería sonar POST. Cuando enviamos datos desde un formulario al servidor, solemos utilizar el método POST. Por ejemplo si quisiéramos añadir nuevos Tweets a nuestra base de datos, el formulario debería hacer un POST de los tweets POST /tweets, en lugar de hacer /tweets/añadirNuevoTweet.php.

Ejemplos de URIs que son no RESTful y que no se recomienda utilizar:

```
/tweets/añadirNuevoTweet.php
/amigos/borrarAmigoPorNombre.php
/contactos/actualizarContacto.php
```

¿Pero entonces, cuáles serían las URIs correctas para presentar un formulario al usuario, con el objetivo de añadir o editar un recurso?

En situaciones como esta, tiene más sentido añadir URIs como:

```
GET /amigos/nuevo
GET /amigos/marta/editar
```

La primera parte de la trayectoria /amigos/nuevo, debería presentar un formulario al usuario para añadir un amigo nuevo. Inmediatamente después de enviar el formulario, debería usarse una solicitud POST, ya que estamos añadiendo un nuevo amigo.

Para el segundo caso /amigos/marta/editar, este formulario debería editar un usuario existente en la base de datos. Cuando actualizamos los datos de un recurso, se debería utilizar una solicitud PUT.

Más información de cómo nombrar las URI en una API REST: http://www.restapitutorial.com/lessons/restfulresourcenaming.html

2.2 Verbos

Antes de seguir adelante con ejemplos concretos, vamos a revisar un poco más los verbos utilizados en las peticiones a una API REST.

2.2.1 GET

GET es el método HTTP utilizado por defecto en las peticiones web. Una advertencia a tener en cuenta es que deberíamos utilizar GET, para hacer peticiones sólo de lectura, y deberíamos obtener siempre el mismo tipo de resultado, independientemente de las veces que sea llamado ese método.

Como programador puedes hacer lo que quieras cuando se hace una llamada a las rutas en la URI, pero una buena práctica es seguir las reglas generales para diseñar una API REST correctamente.

2.2.2 POST

El segundo método que te resultará familiar es POST. Se utilizará para indicar que vamos a añadir nuevos datos a un recurso. Por ejemplo, si queremos añadir nuevos amigos, el método POST sería la opción perfecta para hacerlo:

```
POST /amigos
```

2.2.3 PUT

Tradicionalmente, una solicitud PUT debería utilizarse cuando queremos actualizar un recurso. Imaginemos que queremos actualizar la edad de un amigo Martin. Una vez actualizados los datos en el formulario de edición, estos datos deberían ser enviados a la URI utilizando una petición PUT.

2.2.4 DELETE

Por último DELETE debería se usado cuando queremos borrar el recurso especificado en la URI. Por ejemplo si ya no somos más amigos de macarena, siguiendo los principios de REST, podríamos borrarla usando una petición delete a la URI:

```
DELETE /amigos/macarena
```

2.3 RESTFUL API o API REST en PHP

Para crear una API REST o RESTFUL API (en inglés) en php podremos hacerlo utilizando un fichero .htaccess dónde programamos todos los tipos de URI's que gestionaremos en la API o bien utilizando un framework que nos facilite dicha programación.

Veremos para ello el Slim Framework, que es bastante sencillo y nos facilita muchísimo este tipo de creación.

2.4 Slim framework

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

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

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

2.4.1 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.

2.4.2 Requerimientos

Es necesario tener instalado PHP 5.3.0 o superior.

2.4.3 Descarga de Slim framework

El framework Slim se puede descargar desde Github en: https://github.com/codeguy/Slim/zipball/master

2.4.4 Instalación

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.

--Veiga (discusión)