Introducción a los servicios REST

Los servicios REST se han convertido en una arquitectura de aplicaciones muy popular a la hora de intercambiar información entre el lado cliente y servidor, por su versatilidad, ya que permiten que diferentes tipos de clientes (aplicaciones móviles, de escritorio o web) puedan solicitar información a un mismo servidor, y utilizarla de forma adaptada a cada aplicación. Veremos en este documento las bases sobre las que se sustenta.

1. El protocolo HTTP y las URL

Para lo que vamos a ver a partir de esta sesión, conviene tener claros algunos conceptos de base. Para empezar, cualquier aplicación web se basa en una arquitectura cliente-servidor, donde un servidor queda a la espera de conexiones de clientes, y los clientes se conectan a los servidores para solicitar ciertos recursos.

Estas comunicaciones se realizan mediante un protocolo llamado **HTTP** (o HTTPS, en el caso de comunicaciones seguras). En ambos casos, cliente y servidor se envían cierta información estándar, en cada mensaje:

- En cuanto a los **clientes**, envían al servidor los datos del recurso que solicitan, junto con cierta información adicional, como por ejemplo las cabeceras de petición (información relativa al tipo de cliente o navegador, contenido que acepta, etc), y parámetros adicionales llamados normalmente *datos del formulario*.
- Por lo que respecta a los **servidores**, aceptan estas peticiones, las procesan y envían de vuelta algunos datos relevantes, como un código de estado (indicando si la petición pudo ser atendida satisfactoriamente o no), cabeceras de respuesta (indicando el tipo de contenido enviado, tamaño, idioma, etc), y el recurso solicitado propiamente dicho, si todo ha ido correctamente.

Para solicitar los recursos, los clientes se conectan o solicitan una determinada **URL** (siglas en inglés de "localización uniforme de recursos", *Uniform Resource Location*). Esta URL consiste en una línea de texto con tres secciones diferenciadas:

- El **protocolo** empleado (HTTP o HTTPS)
- El **nombre de dominio**, que identifica al servidor y lo localiza en la red.
- La **ruta** hacia el recurso solicitado, dentro del propio servidor. Esta última parte también suele denominarse **URI** (identificador uniforme de recurso, o en inglés, *Uniform Resource Identifier*). Esta URI identifica unívocamente el recurso buscado entre todos los demás recursos que pueda albergar el servidor.

Por ejemplo, la siguiente podría ser una URL válida:

http://miservidor.com/libros?id=123

El protocolo empleado es *http*, y el nombre de dominio es *miservidor.com*. Finalmente, la ruta o URI es *libros?id=123*, y el texto tras el interrogante '?' es la información adicional llamada *query string* o *datos del formulario*. Esta información permite aportar algo más de información sobre el recurso solicitado. En este caso, podría hacer referencia al código del libro que estamos buscando. Dependiendo de cómo se haya implementado el servidor, también podríamos reescribir esta URL de este otro modo, con el mismo significado:

http://miservidor.com/libros/123

2. Los servicios REST

En esta sección del curso veremos cómo aplicar lo aprendido hasta ahora para desarrollar un servidor sencillo que proporcione una API REST a los clientes que se conecten. **REST** son las siglas de *REpresentational State Transfer*, y designa un estilo de arquitectura de aplicaciones distribuidas, como las aplicaciones web. En un sistema REST, identificamos cada recurso a solicitar con una URI (identificador uniforme de recurso), y definimos un conjunto delimitado de comandos o métodos a realizar, que típicamente son:

- **GET**: para obtener resultados de algún tipo (listados completos o filtrados por alguna condición)
- **POST**: para realizar inserciones o añadir elementos en un conjunto de datos
- **PUT**: para realizar modificaciones o actualizaciones del conjunto de datos
- **DELETE**: para realizar borrados del conjunto de datos

Existen otros tipos de comandos o métodos, como por ejemplo PATCH (similar a PUT, pero para cambios parciales), HEAD (para consultar sólo el encabezado de la respuesta obtenida), etc. Nos centraremos, no obstante, en los cuatro métodos principales anteriores

Por lo tanto, identificando el recurso a solicitar (URI) y el comando a aplicarle, el servidor que ofrece esta API REST proporciona una respuesta a esa petición. Esta respuesta típicamente viene dada por un mensaje en formato JSON o XML (aunque éste último cada vez está más en desuso).

Veremos cómo podemos identificar los diferentes tipos de comandos de nuestra API, y las URIs de los recursos a solicitar, para luego dar una respuesta en formato JSON ante cada petición.

2.1. Los *endpoints*

Un *endpoint*, aplicado a servicios REST, consiste en la URI y el comando que se debe solicitar al servidor para un determinado servicio. Por ejemplo, si debemos acceder a la URI `/libros` por GET para obtener un listado de libros, el *endpoint* correspondiente sería:

```
GET /libros
```

Es conveniente recordar esta nomenclatura, pues es bastante habitual encontrarla cuando trabajamos con servicios REST.

3. El formato JSON

JSON son las siglas de *JavaScript Object Notation*, una sintaxis propia de Javascript para poder representar objetos como cadenas de texto, y poder así serializar y enviar información de objetos a través de flujos de datos (archivos de texto, comunicaciones cliente-servidor, etc).

Un objeto JavaScript se define mediante una serie de propiedades y valores. Por ejemplo, los datos de una persona (como nombre y edad) podríamos almacenarlos así:

```
let persona = {
    nombre: "Nacho",
    edad: 39
};
```

Este mismo objeto, convertido a JSON, formaría una cadena de texto con este contenido:

```
{"nombre":"Nacho","edad":39}
```

Del mismo modo, si tenemos una colección (vector) de objetos como ésta:

```
let personas = [
    { nombre: "Nacho", edad: 39},
    { nombre: "Mario", edad: 4},
    { nombre: "Laura", edad: 2},
    { nombre: "Nora", edad: 10}
];
```

Transformada a JSON sigue la misma sintaxis, pero entre corchetes:

```
[{"nombre":"Nacho","edad":39}, {"nombre":"Mario","edad":4},
 {"nombre":"Laura","edad":2}, {"nombre":"Nora","edad":10}]
```

JavaScript ofrece un par de métodos útiles para convertir datos a formato JSON y viceversa. Estos métodos son `JSON.stringify` (para convertir un objeto o array JavaScript a JSON) y `JSON.parse` (para el proceso inverso, es decir, convertir una cadena JSON en un objeto JavaScript). Aquí vemos un ejemplo de cada uno:

```
let personas = [
    { nombre: "Nacho", edad: 39},
    { nombre: "Mario", edad: 4},
    { nombre: "Laura", edad: 2},
    { nombre: "Nora", edad: 10}
];

// Convertir array a JSON
let personasJSON = JSON.stringify(personas);
console.log(personasJSON);

// Convertir JSON a array
let personas2 = JSON.parse(personasJSON);
console.log(personas2);
```

En los siguientes ejemplos vamos a realizar comunicaciones cliente-servidor donde el cliente va a solicitar al servidor una serie de servicios, y éste responderá devolviendo un contenido en formato JSON. Sin embargo, gracias al framework Express que utilizaremos, la conversión desde un formato a otro será automática, y no tendremos que preocuparnos de utilizar estos métodos de conversión.

3.1. JSON y servicios REST

Como comentábamos antes, JSON es hoy en día el formato más utilizado para dar respuesta a peticiones de servicios REST. Su otro "competidor", el formato XML, está cada vez más en desuso para estas tareas.

A la hora de emitir una respuesta a un servicio utilizando formato JSON, es habitual que ésta tenga un formato determinado. En general, en las respuestas que emitamos a partir de ahora para servicios REST en el curso, utilizaremos una estructura general basada en:

- Un dato booleano (podemos llamarlo `ok`, por ejemplo), que indique si la petición se ha podido atender satisfactoriamente o no.
- Un mensaje de error (podemos llamarlo `error`, por ejemplo), que estará presente únicamente si el anterior dato booleano es falso, lo que indicaría que la petición no se ha podido resolver.
- Los datos de respuesta, que estarán presentes sólo si el dato booleano es verdadero, lo que indica que la petición se ha podido atender satisfactoriamente. Notar que estos datos de respuesta pueden ser un texto, un objeto simple JavaScript, o un array de objetos.

Adicionalmente, como veremos en los ejemplos a continuación, también es recomendable añadir a la respuesta un código de estado HTTP, que indique si se ha podido servir satisfactoriamente o ha habido algún error.