Grado en Ingeniería Informática

Departamento de Ingeniería Informática

Sistemas Distribuidos

Práctica 3: Creación de API REST con Bottle

Autores:
Borja Romero Fernández
Carmen del Mar Ruiz de Celis

Fecha:
24 de mayo de 2020

Índice

1. Parte 1: Servicio Web REST con Bottle **2**

2. Parte 2: Cliente **4**

1. Parte 1: Servicio Web REST con Bottle

En esta práctica nos hemos centrado en crear una API REST simple haciendo uso de Bottle. Los objetos con los que trabajamos son habitaciones, cada una de ellas con los siguientes campos:

- **Campo *id***: Número identificador de la habitación, representado por un dígito.
- **Campo *cap***: Capacidad de personas que permite la habitación, también representado por un dígito.
- **Campo *equip***: Equipamiento que tiene la sala, representado por una cadena de caracteres.
- **Campo *occ***: Booleano que representa si una habitación esta ocupada (*occupied*) o desocupada (*free*)

En cuanto a nuestra API, consta de varios **endpoint** que vamos a comenzar a explicar:

- **Creación de una nueva habitación**: En este *endpoint* hacemos uso del método POST, ya que vamos a crear una nueva entrada en la misma. La API comprobará que el servidor le ha mandado un dato correcto y guardará el mismo en una variable temporal. Finalmente devolverá un mensaje para que el servidor confirme que la operación se ha realizado correctamente y así poder comunicarle lo mismo al cliente. La URL asignada para esta operativa será *http://localhost:8080/new*.
- **Modificación de una habitación existente**: En este *endpoint* hacemos uso del método POST ya que el método PUT nos ha causado muchos problemas que no sabíamos resolver tras indagar sobre ellos. En esta operativa la API comprueba si existe una habitación con el id que ha indicado el usuario y en caso de encontrarla, modifica todos sus datos con los que ha introducido el usuario. Finalmente se manda un mensaje para que el servidor comunique que se ha realizado correctamente la función. Este *endpoint* hace uso de las variables por URL, utilizando el id de la habitación como un atributo. La URL asignada para esta operativa será *http://localhost:8080/modify/<id>*.
- **Lista completa de habitaciones existentes**: Este *endpoint* simplemente lista todos los datos que se encuentren actualmente en la API guardados. EL método usado es GET, ya que sólo necesitamos obtener datos de la API. Devuelve además una lista en forma de cadena de caracteres para que el cliente pueda imprimarla por pantalla al usuario. La URL asignada para esta operativa será *http://localhost:8080/rooms*.

- **Datos de una habitación dado su número identificador**: Este *endpoint* también hace uso de atributos pasados por URL para recibir el identificador de una habitación y así devolver sus datos. Hacemos uso del método GET por las mismas razones que el anterior. La API comprueba si existe el id en las habitaciones existentes y en caso de encontrarla envía sus datos al servidor para que este se los comunique al cliente y pueda así imprimirlos por pantalla. La URL asignada para esta operativa será *http://localhost:8080/room/<id>*.
- **Lista de habitaciones ocupadas o desocupadas**: Para este *endpoint* también hacemos uso del método GET y de las variables de URL. Para su comprobación añadiremos ya sea *free* o *occuppied* dependiendo de qué lista queramos ver (habitaciones libres y habitaciones ocupadas respectivamente). La API comprobará el campo *occ* de cada habitación existente y guardará en una lista temporal aquellas que cumplan la condición pedida. Finalmente la API devolverá la lista temporal al servidor. La URL asignada para esta operativa será *http://localhost:8080/rooms/available/<occu>*, siendo *occu* el valor *free* o el valor *occupied*.
- **Eliminación de habitación existente**: Este último *endpoint* es el último que hace uso de variables por URL. Este último también hace uso del método POST por complicaciones. Le pasaremos al mismo un identificador de habitación. Comprobará entonces la API que exista esa habitación con ese identificador y en ese caso elimina la habitación de las guardadas. Mostrará un mensaje en caso de realizar la operación correctamente. La URL asignada para esta operativa será *http://localhost:8080/delete/<id>*.

Cabe destacar que todos los *endpoints* contienen algún tipo de verificación que añade seguridad a nuestra API. En caso de realizarse alguna operación incorrectamente, contamos con Event Handlers que lanzarán mensajes y advertencias para la correcta ejecución. Aún así también aseguramos los datos por parte del cliente, para que introduzca datos correctos y que no afecten al mal funcionamiento de la API o el servidor.

También contamos con una opción para poder salir de programa. Esta opción se encarga simplemente de comunicar al cliente y servidor que no tienen que realizar mas iteraciones, cerrando la conexión en ambos casos.

Todo esto funciona en cualquier momento, incluso si terminamos la ejecución del servicio. Esto se debe a que hemos implementado además un almacenamiento de información mediante ficheros. El almacenamiento se basa en un fichero *.json* local que es comprobado cada vez que se ejecuta la API. En caso de encontrarse vacío, la variable local _rooms (que almacena temporalmente las habitaciones en la API), se inicializa vacía. En caso de que este fichero contenga algo, se vuelcan en formato

JSON todos los datos encontrados. Además en cada operación modificadora de la API, se llama a la función *write_file()* que recoge los datos almacenados en la variable _rooms y vuelca esos datos en el fichero sobrescribiéndolos.

2. Parte 2: Cliente

En cuanto al cliente, hemos creado un menú para que el usuario pueda elegir aquella opción que desea realizar. Para ello hemos implementado un "switch" muy rudimentario, basado en condiciones *if* y *else*. En cada bloque *if* se realiza una de las operativas que ofrece la API. Comprobemos el funcionamiento de cada uno:

1. **Crear habitación**: Inicialmente pedimos al usuario los datos de la nueva habitación que quiere añadir, asegurando que todos los datos sean del tipo correcto que se piden (id como dígito, capacidad como dígito, ...). Una vez introducidos, creamos un diccionario con estos datos y convertimos el JSON el resultado. Este JSON será el que le pasamos al servidor para que trate con el mismo. Finalmente esperamos a que el servidor nos devuelva una respuesta, en este caso un mensaje que define el correcto funcionamiento.

2. **Modificar habitación**: Esta opción tiene la misma implementación que la anterior, ya que realmente se trata con los mismos datos. Es el servidor el encargado de tratarlos de distinta manera esta vez.

3. **Listar todas las habitaciones**: El cliente simplemente espera a que el servidor devuelva la lista de habitaciones para mostrarla por pantalla.

4. **Mostrar datos de una habitación**: Se le pide al usuario un identificador de habitación, que será pasado al servidor para pedirle a la API los datos correspondientes. El cliente a la respuesta del servidor.

5. **Mostrar lista de habitaciones ocupadas/desocupadas**: En esta opción el usuario cuenta con otro menú, donde pulsará 1 en caso de querer ver las habitaciones libres y 2 en caso de querer ver las habitaciones ocupadas. Dependiendo de la opción que elija el cliente controlará lo que envía al servidor, para después esperar a la lista de habitaciones.

6. **Eliminar habitación**: Al igual que en la opción de mostrar los datos de una habitación, el cliente pide al usuario un identificador de habitación para poder decirle al servidor qué habitación eliminar. Finalmente el cliente espera un mensaje para comprobar su correcto funcionamiento o no.

7. **Salir de la aplicación**: El cliente cambia el estado de nuestro booleano *status* a false, ya que controla la repetición del servicio hasta que no cambia. Si el

status pasa a false tanto el cliente como el servidor cierran la conexión.

Hay que decir que en cada opción lo primero que se hace es enviar al servidor un mensaje con la opción que ha elegido el usuario. Esta es nuestra manera de controlar el comportamiento del servidor ante la elección del cliente.