Práctica 3: Creación de API REST con Bottle

Sistemas Distribuidos

Curso: 2021/2022

Índice

1. Información general de la práctica

- La práctica se puede realizar de forma conjunta con un compañero/a. Ambos integrantes deben pertenecer al mismo grupo de prácticas.
- Se admiten trabajos individuales, pero no de tres estudiantes.
- En el Campus Virtual se encuentra habilitada una tarea para realizar la entrega.
- La fecha de entrega establecida debe consultarse en dicha tarea. Atención, puede ser diferente para cada grupo de prácticas.
- No se admitirán entregas fuera de plazo.
- La entrega debe realizarla solo uno de los integrantes del grupo, en caso de realizar la práctica por parejas.
- La entrega se compone de un archivo comprimido que contendrá los códigos fuentes (**sin crear subdirectorios**). El comprimido deberá nombrarse como `code.zip`.
- Los ejercicios deben realizarse en dos archivos independientes denominados:
  - client.py
  - server.py

2. Servidor (Servicios Web REST con Bottle - API REST)

En esta práctica vamos a trabajar con el framework Bottle visto en el Seminario 4. El objetivo es crear un servicio web de gestión de reserva de espacios para eventos en el `server.py`.

Cada **sala** deberá contar con los siguientes atributos:

- *Identificador de la sala*. El *identificador de la sala* es único y no puede haber dos salas con el mismo *identificador*.
- *Capacidad*, es decir, el total de personas que tienen cabida en la sala.
- *Recursos de la sala (proyector, pizarra, rotuladores, altavoces, micrófono, puntero láser)*. No todas las salas cuentan con los mismos recursos disponibles.

Este servicio de gestión de reservas lo utilizan diferentes **usuarios** de los cuáles debemos gestionar la siguiente información:

- *DNI con letra*. Suponemos que el *DNI* del usuario es único, que no puede haber dos usuarios con el mismo *DNI* y que tiene una longitud de 9 caracteres.
- *Nombre de usuario*. Suponemos que el *nombre de usuario* es único y no puede haber dos usuarios con el mismo *nombre de usuario*.

- *Contraseña*. Suponemos que la contraseña se compone de caracteres alfanuméricos y de una longitud mínima de 8 caracteres.

Las reservas las gestiona un usuario que es la persona responsable de la reserva. De una **reserva** debemos gestionar la siguiente información:

- *Identificador de la reserva*. Se generará de forma consecutiva e incremental.
- *DNI del usuario responsable de la reserva*. Este campo se completará automáticamente, tras haber autenticado al usuario que está haciendo la reserva.
- *Fecha de la reserva*. Día, mes y año.
- *Hora de inicio de la reserva*. Formato de 24 horas (p. ej. 11:30).
- *Hora de fin de la reserva*. Formato de 24 horas (p. ej. 18:45).

Se deberán implementar las siguientes operaciones:

1. Añadir una nueva sala.
   endpoint: `'/addRoom'`
   La información de la sala se recibirá en formato JSON.

2. Mostrar la información de una sala, indicando el identificador de la sala.
   endpoint: `'/showInformationRoom/'roomId'`
   Se devolverá una cadena con la información de la sala en formato JSON, al cliente.

3. Añadir una nueva reserva, indicando *fecha de la reserva*, *hora de inicio* y *duración* de la reserva en minutos (no hora de fin).
   endpoint: `'/addBooking'`
   La información de la reserva se recibirá en formato JSON.
   Si un usuario pretende hacer una reserva y la sala está ocupada, el servidor devolverá una cadena que contendrá la siguiente información en formato JSON:
   - *"La sala que desea reservar está ocupada"* y;
   - la información completa de todas las salas disponibles para la franja horaria indicada de ese día concreto.

4. Mostrar la lista de reservas realizadas por un responsable concreto, identificado por su DNI.
   endpoint: `'/showBookings/userDNI'`
   Se debe mostrar toda la información de la reserva. Se devolverá la información al cliente como una cadena en formato JSON.

5. Eliminar una reserva existente, indicando el identificador de la reserva.
   endpoint: `'/deleteBooking/bookingId'`
   Si la reserva se elimina correctamente se devolverá al cliente el siguiente mensaje como una cadena en formato JSON: *Reserva eliminada"*. Si el identificador de la reserva no existe se devolverá al cliente el siguiente mensaje como una cadena en formato JSON: *"No existe el identificador de la reserva"*.

3. Cliente

Crear un cliente en el `client.py` que permita al usuario interactuar con la API desarrollada.

Las operaciones del menú se corresponderán con lo siguiente:

1. `Añadir sala`
2. `Mostrar información de sala`
3. `Añadir reserva`
4. `Listar reservas`
5. `Eliminar reserva`
6. `Exit`

El flujo de interacción debe coincidir con lo siguiente:

1. Este cliente le mostrará al usuario un menú con las posibles operaciones que se podrán realizar.
2. El usuario indicará la operación que desea realizar. Por ejemplo, introducirá por teclado el número `3`.
3. El cliente le solicitará al usuario los datos que debe introducir por teclado. Continuando con el ejemplo, el cliente debe ir preguntando uno a uno por los datos de la reserva:
   - *Fecha de la reserva.*
   - *Hora de inicio de la reserva.*
   - *Hora de fin de la reserva.*
4. Una vez que el cliente tenga toda la información requerida, enviará la solicitud del servicio al servidor.

Importante:

- Las operaciones que se deben implementar, únicamente las podrá realizar un usuario registrado en el sistema. Por tanto, antes de realizar cada una de las operaciones, siempre debemos comprobar si el usuario está registrado en el sistema. Para ello, el **cliente** le preguntará al usuario su *nombre de usuario* y *contraseña* y adjuntará esta información en cada petición que se realice al servidor. El servidor deberá comprobar que esta información coincide con la almacenada en el sistema, antes de realizar la operación solicitada. Si el usuario no está registrado la operación se cancela y el servidor devolverá al cliente: *"Credenciales no válidas"*.
- La información correspondiente a la gestión de salas, responsables y reservas se debe almacenar de forma persistente en un fichero JSON. Todos los atributos deben contener información.
- Se deben crear *clases* para gestionar la información.
- Las claves de los JSON deben coincidir con las que se encuentran en el archivo `CommunicationJSON.json` de la carpeta *"Templates"*.

4. Evaluación

La nota de la práctica se divide de la siguiente forma:

- Servidor (Servicios Web REST con Bottle - API REST): 80 %
- Cliente: 20 %

Bibliografía

- Python (2022). Disponible online: https://docs.python.org/3/tutorial/classes.html.
- Python For Beginners (2022). Disponible online: https://www.pythonforbeginners.com/files/reading-and-writing-files-in-python.
- Real Python (2022). Disponible online: https://realpython.com/python-json/.
- Stackabuse (2022). Disponible online: https://stackabuse.com/reading-and-writing-json-to-a-file-in-p