Tercera práctica de SD

José Manuel Heredia Bravo
Jesús Márquez Delgado

Mayo de 2020

Índice

1. Especificación de la API

1.1. Tipos de datos de la API

La API Hotel contiene un conjunto de operaciones útiles para la administración telemática de un hotel a partir de otro programa que sea capaz de hacer solicitudes por medio de URIs.

Una habitación (tipo de dato `Room`) es un tipo de dato que alberga información sobre la misma y se identifica por su identificador. Estos datos categorizados son los siguientes:

- Número de la habitación: identificador de la habitación.
- Número de plazas de la habitación: aforo máximo de la habitación.
- Lista de equipamiento: lista de elementos extras que contiene la habitación (como aire acondicionado, estufa, animales, etc).
- Estado de ocupación: indica si la habitación está o no ocupada.

1.2. Operaciones de la API

1.2.1. Operaciones POST

/altaHabitacion

- Precondición: Recibe un archivo JSON ajustado al siguiente formato: `{"plazas": n_plazas, "equipamiento": lista_equipo, "ocupada": estado}`. Siendo:
  - *n_plazas*: entero que representa las plazas de la habitación. Debe ser al menos uno.
  - *lista_equipo*: lista de prestaciones que tiene la habitación, separadas por coma.
  - *ocupada*: variable lógica que indica si una habitación está (`True`) o no (`False`) ocupada.
- Poscondición: Guarda una nueva habitación y se le crea un id asociado a la misma. Devuelve la habitación recién creada. Si hubiera un error con los parámetros (se incumple la precondición), devolverá la cadena vacía (`''`).

1.2.2. Operaciones PUT

/modificarHabitacion/identificador

- Precondición: Recibe un JSON con lo que se desee modificar de dicha habitación. Según sea, se debe ajustar al formato:
  - *plazas*: `{"plazas": plazas}`

- *equipamiento*: `"equipamiento": lista`. La lista debe estar separada por comas.
- *ocupación*: `{"ocupada": estado}`. Siendo *estado* `True` o `False`.

- *Poscondición*: Modifica el atribute que se mande con un fichero JSON a la habitacion de identificador existente.

1.2.3. Operaciones GET

/buscarHabitacion/identificador

- Poscondición: Devuelve la habitación con el identificador pasado, siguiendo el siguiente formato: `{"id": identificador, "plazas": n_plazas, "equipamiento": lista_equipamiento, "ocupada": estado}`. Siendo:
  - *id*: identificador de la habitación.
  - *plazas*: el número de plazas.
  - *equipamiento*: lista de las prestaciones de la habitación.
  - *estado*: indica si una habitación está o no ocupada.

  Si la habitación no existe, devuelve una cadena vacía (`''`).

/listaHabitaciones

- Poscondición: Devuelve la lista de habitaciones con el siguiente formato: `{id: habitación, ...}`. Siendo:
  - *id*: el identificador de una habitación.
  - *habitación*: los datos de la habitación en un diccionario.

/habitaciones/ocupada

- Poscondición: Si ocupada es `'Ocupadas'`, devuelve la lista de habitaciones ocupadas. Si ocupada es `'Desocupadas'`, devolverá la lista de habitaciones desocupadas. En ambos casos, se devolverá una respuesta ajustada al formato: `{id: habitación, ...}`, siendo:
  - *id*: el identificador de una habitación.
  - *habitación*: los datos de la habitación en un diccionario.

  Si no es ninguno de esos valores, devolverá una cadena vacía (`''`), lo que es un indicativo de un error.

1.2.4. Operaciones DELETE

/eliminarHabitacion/identificador

- Poscondición: Elimina la habitación con el identificador asociado. Devuelve el identificador de la habitación eliminada. Si esta no existiera, devolvería una cadena vacía (`''`).

2. Implementación de la API

Para la implementación de la API, vamos a usar Python versión 3.7.3 por medio del framework Bottle, en su versión 0.12.18.

2.1. Clase Room

Clase muy útil para abstraernos de la realidad de una habitación. Básicamente, va a almacenar los datos de una habitación. He aquí su implementación:

```
class Room:
    def __init__(self, id, plazas, equipamiento, ocupada):
        self.id = id
        self.plazas = plazas
        self.equipamiento = equipamiento
        self.ocupada = ocupada
```

2.2. Operaciones auxiliares de la API

Utilizadas para facilitar el movimiento de la información de la "base de datos".

```
# Guarda los datos (data) de un diccionario de todos las instancias de la clase
# Room en un fichero JSON con ruta filename
def guardar(data,filename):
    f = open(filename,'w')
    json.dump(data,f,indent=4)
    f.close()

# Devuelve un diccionario con todos los datos alojados en el fichero filename
def cargar(filename):
    try:
        f = open(filename)
        datos = json.load(f)
        f.close()
    except:
        datos = {}
        datos['habitaciones'] = {}
        datos['maxId'] = 0
    return datos
```

2.3. Operaciones de la API

Las operaciones públicas a las que tiene acceso un usuario de la API.

```python
# Inserta una nueva habitación en la BD y se le asigna un identificador.
@post('/altaHabitacion')
def alta_habitacion():
    bd = cargar('bd.json')
    habitaciones = bd['habitaciones']
    data = request.json
    plazas = data.get('plazas')

    if not plazas.isnumeric():
        return ""
    if int(plazas) <= 0:
        return ""
    equipamiento = data.get('equipamiento')
    ocupada = data.get('ocupada')

    habitacion = Room(bd['maxId'],plazas,equipamiento,ocupada)
    habitaciones[bd['maxId']] = habitacion.__dict__
    bd['maxId'] += 1

    response.headers['Content-Type'] = 'application/json'

    guardar(bd,'bd.json')
    return json.dumps(habitacion.__dict__)


# Elimina la habitación con el identificador pasado
@delete('/eliminarHabitacion/<identificador>')
def buscar_habitacion(identificador):
    try:
        bd = cargar('bd.json')
        del bd['habitaciones'][identificador]
        guardar(bd,'bd.json')
        return identificador
    except:
        return ""

# Modifica la información de la habitación con el identificador pasado.
@put('/modificarHabitacion/<identificador>')
def modificar_habitacion(identificador):
    bd = cargar('bd.json')

    habitacion = bd['habitaciones'][identificador]

    data = request.json
    plazas = data.get('plazas')
    equipamiento = data.get('equipamiento')
    ocupada = data.get('ocupada')
```

```
48
49      if(plazas != None):
50          if(not plazas.isnumeric()):
51              return ""
52          else:
53              habitacion['plazas'] = plazas
54      if(equipamiento != None):
55          habitacion['equipamiento'] = equipamiento
56      if(ocupada != None):
57          habitacion['ocupada'] = ocupada
58
59      guardar(bd,'bd.json')
60
61      return habitacion
62
63  # Devuelve los datos de la habitación con el identificador pasado.
64  @get('/buscarHabitacion/<identificador>')
65  def buscar_habitacion(identificador):
66      try:
67          bd = cargar('bd.json')
68          return bd['habitaciones'][identificador]
69      except:
70          return ""
71
72  # Devuelve la lista de habitaciones existentes.
73  @get('/listaHabitaciones')
74  def lista_habitaciones():
75      bd = cargar('bd.json')
76      return bd['habitaciones']
77
78  # Devuelve la lista de habitaciones ocupadas o desocupadas.
79  @get('/habitaciones/<ocupada>')
80  def habitaciones_ocupadas(ocupada):
81      bd = cargar('bd.json')
82      res = list()
83      if ocupada == 'Ocupadas':
84          ocupada = True
85      elif ocupada == 'Desocupadas':
86          ocupada = False
87      else:
88          return ""
89      for indice in bd['habitaciones']:
90          if bd['habitaciones'][indice]['ocupada'] == ocupada:
91              res.append(bd['habitaciones'][indice])
92      return '{}' if len(res) == 0 else json.dumps(res, indent=4)
```

2.4. Función principal

Es la que se ejecutará cuando se ejecute el fichero del servidor:

```
1 if __name__ == '__main__':
2     run(host='localhost', port=8080, debug=True)
```

3. Aplicación del cliente

Esta aplicación será la que facilitará la comunicación con la API por parte del cliente. Es una sencilla aplicación de consola que permite utilizar todas las operaciones públicas de la API.

3.1. Especificación de las operaciones

imprimirHabitacion(habitacion)* → *None

- Precondición: *habitacion* es un diccionario con los siguientes elementos:
  - *id*: identificador de la habitación.
  - *plazas*: número de plazas de la habitación.
  - *equipamiento*: lista de las prestaciones de la habitación.
  - *ocupada*: indica si la habitación está ocupada.
- Poscondición: Muestra por pantalla los datos de la habitación.

altaHabitacion()* → *None

- Poscondición: Se solicita insertar los datos pedidos por pantalla en la API.

modificarHabitacion()* → *None

- Poscondición: Pide el identificador de una habitación y pide los cambios que se deseen realizar. Hace una solicitud a la API para modificarlos.

listaHabitaciones()* → *None

- Poscondición: Solicita a la API la lista con todas las habitaciones y las muestra por pantalla.

habitacionesOcupadas()* → *None

- Poscondición: Se solicita a la API la lista con todas las habitaciones ocupadas y las muestra por pantalla.

habitacionesDesocupadas()* → *None

- Poscondición: Se solicita a la API la lista con todas las habitaciones desocupadas y las muestra por pantalla.

buscarHabitacion()* → *None

- Poscondición: Pide al usuario el identificador de una habitación y se solicita a la API los datos de dicha habitación.

eliminarHabitacion()* → *None

- Poscondición: Pide al usuario el identificador de una habitación y se solicita a la API eliminar la habitación.

3.2. Implementación de las operaciones

```
# Muestra una habitación por pantalla
def imprimirHabitacion(habitacion):
    id = habitacion['id']
    plazas = habitacion['plazas']
    equipamiento = habitacion['equipamiento']
    n = len(equipamiento)
    cadequip = equipamiento[0]
    if(n > 1):
        i = 1
        while(i < n):
            if(i < n-1):
                cadequip += ", "+equipamiento[i]
            else:
                cadequip += " y "+equipamiento[i]
            i += 1
    ocupada = habitacion['ocupada']

    print("Habitación "+str(id))
    print("\t- "+plazas+" plazas")
    print("\t- Equipamiento: "+cadequip)
    if(ocupada):
        print("\t- Ocupada")
    else:
        print("\t- Libre")

# Solicita insertar una habitación
def altaHabitacion():
    plazas = input("Introduce el numero de plazas: ")
    equipamiento = input("Introduce el equipamiento separado por (,): ")
    ocupada = input("Ocupada(s/n): ")
    if(ocupada == "s"):
        bOcupada = True
    else:
        bOcupada = False
    res = requests.post("http://localhost:8080/altaHabitacion",
        json={"plazas": plazas, "equipamiento": equipamiento.split(","),
```

```
                "ocupada": bOcupada})
    if(res.text == ""):
        print("Los datos no se introdujeron correctamente")
    else:
        print("Creada la habitación: ")
        imprimirHabitacion(json.loads(res.text))

# Solicita elimina una habitación
def eliminarHabitacion():
    id = input("Identificador de la habitacion: ")
    res = requests.delete("http://localhost:8080/eliminarHabitacion/"+id)
    if(res.text == ""):
        print("La habitación a eliminar no existe")
    else:
        print("La habitacion "+res.text+" ha sido eliminada")

# Solicita modificar una habitación
def modificarHabitacion():
    id = input("Identificador de la habitación: ")
    op = -1
    res = requests.get("http://localhost:8080/buscarHabitacion/"+id)
    if(res.text == ""):
        print("La habitación buscada no existe")
    else:
        imprimirHabitacion(res.json())
        while(op != "0"):
            print("\n|MODIFICAR HABITACIÓN "+id+"|")
            print("1.Modificar plazas")
            print("2.Modificar equipamiento")
            print("3.Modificar estado de ocupación")
            print("0.Salir")
            op = input("\nOpción: ")
            if(op == "1"):
                plazas = input("Introduce el numero de plazas: ")
                res = requests.put("http://localhost:8080/modificarHabitacion/"+
                    id,json={"plazas": plazas})
            elif(op == "2"):
                equipamiento = input("Introduce el equipamiento separado por (,): ")
                res = requests.put("http://localhost:8080/modificarHabitacion/"+
                    id,json={"equipamiento": equipamiento.split(",")})
            elif(op == "3"):
                ocupada = input("Ocupada(s/n): ")
                if(ocupada == "s"):
                    bOcupada = True
                else:
                    bOcupada = False
                res = requests.put("http://localhost:8080/modificarHabitacion/"+
                    id,json={"ocupada": bOcupada})
```

```
85              if(op == "1" or op == "2" or op == "3"):
86                  if(res.text == ""):
87                      print("La modificacion introducida es incorrecta")
88                  else:
89                      print("Habitación modificada:")
90                      imprimirHabitacion(json.loads(res.text))
91              elif op == not '0':
92                  print('Operación no contemplada.\n')
93
94  # Muestra la lista de todas las habitaciones.
95  def listaHabitaciones():
96      res = requests.get("http://localhost:8080/listaHabitaciones")
97      if(res.text == "{}"):
98          print("No existen habitaciones")
99      else:
100         print("|LISTA DE HABITACIONES|")
101         print(res.text)
102         for valor in res.json():
103             imprimirHabitacion(res.json()[valor])
104
105 # Busca una habitación y la muestra
106 def buscarHabitacion():
107     id = input("Identificador de la habitacion: ")
108     res = requests.get("http://localhost:8080/buscarHabitacion/"+id)
109     if(res.text == ""):
110         print("La habitación buscada no existe")
111     else:
112         print("|HABITACIÓN "+id+"|")
113         imprimirHabitacion(res.json())
114
115 # Busca una habitación y la elimina
116 def eliminarHabitacion():
117     id = input("Identificador de la habitacion: ")
118     res = requests.post("http://localhost:8080/eliminarHabitacion/"+id)
119     if(res.text == ""):
120         print("La habitación a eliminar no existe")
121     else:
122         print("La habitacion "+res.text+" ha sido eliminada")
123
124 # Muestra las habitaciones desocupadas
125 def habitacionesDesocupadas():
126     res = requests.get('http://localhost:8080/habitaciones/Desocupadas')
127     if(res.text == "{}"):
128         print("No hay habitaciones vacías.")
129     else:
130         print("|LISTA DE HABITACIONES DESOCUPADAS|")
131         for valor in res.json():
132             imprimirHabitacion(valor)
```

```
133
134 # Muestra las habitaciones ocupadas
135 def habitacionesOcupadas():
136     res = requests.get('http://localhost:8080/habitaciones/Ocupadas')
137     if(res.text == "{}"):
138         print("No hay habitaciones ocupadas.")
139     else:
140         print("|LISTA DE HABITACIONES OCUPADAS|")
141         for valor in res.json():
142             imprimirHabitacion(valor)
```

La función principal, que hará uso de las anteriores operaciones es la siguiente:

```
1  import json
2  import requests
3
4  op = -1
5
6  while(op != "0"):
7      print("|API HOTEL|")
8      print("1. Dar de alta habitacion")
9      print("2. Modificar habitacion")
10     print("3. Lista de habitaciones")
11     print("4. Buscar habitacion")
12     print('5. Habitaciones desocupadas')
13     print('6. Habitaciones ocupadas')
14     print("0. Salir")
15     op = input("\nOpcion: ")
16     if(op == "1"):
17         altaHabitacion()
18     elif(op == "2"):
19         modificarHabitacion()
20     elif(op == "3"):
21         listaHabitaciones()
22     elif(op == "4"):
23         buscarHabitacion()
24     elif(op == '5'):
25         habitacionesDesocupadas()
26     elif(op == '6'):
27         habitacionesOcupadas()
```

4. Mejoras implementadas

Las mejoras que le hemos aplicado al programa son las siguientes:

- **Clase Room.** Contiene los datos de una habitación. La utilizaremos en la API para tener un manejo más sencillo de la base de datos.
- **Operación eliminar habitación.** Consiste en una solicitud DELETE para eliminar una habitación de la base de datos.
- **Consistencia de la información.** Cada vez que se realice una operación de modificación, eliminación y creación, todos los datos se cargarán y guardarán al terminar el bloque de instrucciones. En el caso de las operaciones consultoras, simplemente los datos se cargarán pero no se guardarán. Esto, a pesar de tener un mayor coste en eficiencia, nos dará la seguridad de la consistencia de la información.
- **Robustez ante errores.** Las precondiciones de las operaciones de la API no detendrán el programa abruptamente ni tendrán un comportamiento desconocido. En dichos casos, devolverá una cadena nula " y si se hiciera una consulta que devuelve un conjunto vacío devolvería la cadena ''.