

## Práctica 3: Creación de API REST con Bottle

Autores:

Alejandro Serrano Fernández

Pedro Antonio Navas Luque

## Prefacio:

Para que el código funcione correctamente, es necesario la instalación de bottle para Python 3. Dado a que hemos tenido unos cuantos problemas para su instalación, es necesario realizar los siguientes comandos en la terminal de GNU/Linux:

```
sudo apt-get install python3-pip
```

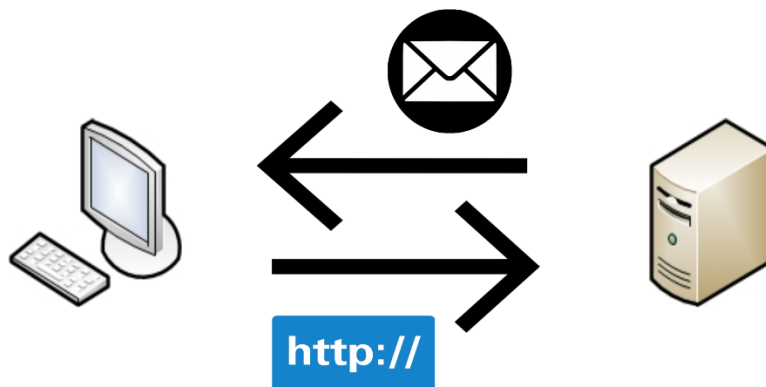
```
pip3 install bottle
```

## Introducción:

Antes de comenzar con la memoria de esta práctica, queremos remarcar que los códigos de esta práctica están realizados bajo la versión 3 de Python. Esta práctica está realizada por los alumnos Alejandro Serrano Fernández y Pedro Antonio Navas Luque para la asignatura de Sistemas Distribuidos. En esta práctica trabajamos con el framework Bottle para ofrecer un servicio web de gestión de habitaciones de un hotel.

## 1. Funcionamiento del sistema

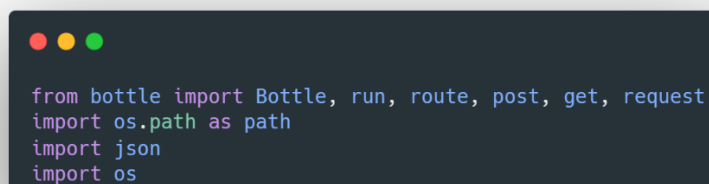
El sistema implementado estaría dispuesto de un cliente y un servidor (en nuestro caso se encuentran en el mismo equipo), en el que ambos interaccionan mediante el protocolo HTTP y el uso de JSON para el envío de datos entre ambos equipos. Cuando el cliente solicite algún dato sobre las habitaciones, hará una petición GET al servidor y éste le ofrecerá el archivo JSON correspondiente. Al contrario, cuando el cliente solicite una petición POST, éste le ofrecerá su JSON (con los datos a enviar) al servidor para hacer las modificaciones que sean pertinentes.



**Figura 0:** Concepto de interacción entre cliente y servidor

## 2. Implementación

Para el correcto funcionamiento del programa, importaremos en el código fuente del servidor, las librerías mostradas en la siguiente imagen:



```
from bottle import Bottle, run, route, post, get, request
import os.path as path
import json
import os
```

**Figura 1:** Librerías utilizadas en el código fuente del servidor

Importaremos `os.path` de la librería `path` para comprobar en el inicio del programa del servidor, que el archivo `data.json` donde se almacenarán nuestros datos existe. En caso contrario creará un archivo nuevo “vacío”. La librería `json` la usaremos para codificar y decodificar los datos JSON del archivo `data.json`.

Finalmente utilizaremos la librería `os` para acceder a los comandos del sistema GNU/Linux, de ellos que utilizaremos el comando `touch` para la creación de ficheros.

Por el contrario, para el código fuente del cliente tan solo importaríamos la librería “`requests`”, con la que haremos uso de las funciones `requests.get()` y `requests.post()`

### 2.1 Alta de una habitación

Definimos una petición POST con el que recibiremos los datos a introducir de la nueva habitación. Para ello primeramente abrimos nuestro archivo JSON y lo cargaremos en una variable local. Posteriormente recibimos los datos introducidos por el cliente con el método `request.json.get()`. Una vez lo recibimos, el servidor comprobará que el identificador de la habitación no está repetido y añadirá sus datos a la variable `habitaciones` para ser posteriormente sobrescrito en el archivo JSON.

```

@post('/send')
def anadir():
    with open('data.json', 'r') as fichero:
        habitacion = json.load(fichero)

    lista_aux = {'Identificador': request.json.get('Identificador'),
'num_plazas':request.json.get('num_plazas'), 'equipamiento':request.json.get('equipamiento'),
'ocupada':request.json.get('ocupada')}

    encontrado = False

    d1 = json.dumps(habitacion) #Lo convierte a un diccionario de json
    d2 = json.loads(d1)
    for key in d2:
        if (key['Identificador'] == lista_aux['Identificador']):
            encontrado = True

    if(encontrado == False):
        habitacion.append(lista_aux)
        print("Añadido correctamente")

    else:
        print("Identificador repetido")

    with open('data.json', 'w') as fichero:
        json.dump(habitacion,fichero)

```

**Figura 2:** Código fuente del servidor para dar de alta una nueva habitación

En cuanto al código fuente del cliente, éste desea conocer si la habitación introducida está registrada o no en el sistema, para ello realiza un `request.get()` a la ruta “/existe” (almacenada en la variable `exists`) con el que obtendrá dicha información (Figura 3).

En el caso de que no lo esté, el cliente introducirá los datos correspondientes y serán enviados al servidor para que los almacene en el archivo JSON.

```

@get('/existe/<id>')
def existir(id):
    with open('data.json', 'r') as fichero:
        habitacion = json.load(fichero)

    encontrado = False
    for i in habitacion:
        if (i['Identificador'] == id):
            encontrado = True

    if(encontrado == False):
        return "NO encontrado"
    else:
        return "SI encontrado"

```

**Figura 3:** Petición GET para conocer si una habitación existe (RUTA: /existe)

```

if (response == '1'):
    num_hab = input("Introduce identificador de la habitacion: ")
    req = requests.get(url = exists+num_hab)
    data = req.text

    if(data == "SI encontrado"):
        print("\n*****")
        print("\n\tLA HABITACION YA EXISTE\n")
        print("*****\n")
    else:
        plazas = input("Introduce plazas de la habitacion: ")
        equip = input("Introduce lista de equipamientos: ")
        ocup = input("¿Ocupada? (si/no): ")

        if(ocup != "si" and ocup != "no"):
            print("\n***Error, respuesta distinta a si o no***")
        else:
            lista = {'Identificador' : num_hab, 'num_plazas' : plazas, 'equipamiento': equip,
'ocupada': ocup}
            requests.post(page, json = lista)

            print("\n*****")
            print("\n\tAÑADIDA CORRECTAMENTE\n")
            print("*****\n")

```

**Figura 4:** Código fuente del cliente para dar de alta una nueva habitación

## 2.2 Modificar una habitación

Cuando el cliente selecciona la opción de modificar una habitación, éste, al igual que en el apartado anterior, desea saber si una habitación existe o no antes de proceder a introducir más datos. Posteriormente, si la habitación se encuentra en el sistema, el cliente le solicitará al servidor los datos introducidos y éste se encargará de recibirlos, eliminar la habitación a modificar, y añadir sus nuevos datos. Una vez añadidos, los nuevos cambios quedarán registrado en el archivo JSON que contiene los datos de las habitaciones.

```

elif (response == '2'):

    num_hab = input("Introduce identificador de la habitacion: ")
    req = requests.get(url = exists+num_hab)
    data = req.text

    if(data == "NO encontrado"):
        print("\n*****")
        print("\n\tNO SE HA ENCONTRADO LA HABITACION\n")
        print("*****\n")
    else:
        plazas = input("Introduce plazas de la habitacion: ")
        equip = input("Introduce lista de equipamientos: ")
        ocup = input("¿Ocupada? (si/no): ")
        lista = {'num_plazas' : plazas, 'equipamiento': equip, 'ocupada': ocup}
        requests.post(modify+num_hab, json = lista)

        print("\n*****")
        print("\n\tSE HA ACTUALIZADO CORRECTAMENTE\n")
        print("*****\n")

```

**Figura 5:** Código fuente del cliente para realizar una modificación en una habitación

En cuanto al servidor, éste recibe a través de la función `request.json.get()` los nuevos datos de la habitación introducida por el cliente, y eliminará sus datos antiguos para posteriormente crear una nueva habitación con mismo identificador y los nuevos datos introducidos.

```
@post('/modify/<id>')
def modificar(id):
    with open('data.json', 'r') as fichero:
        habitacion = json.load(fichero)

    lista_aux = {'Identificador': id, 'num_plazas':request.json.get('num_plazas'),
'equipamiento':request.json.get('equipamiento'), 'ocupada':request.json.get('ocupada')}
    print(lista_aux['Identificador'])
    encontrado = False

    #d1 = json.dumps(habitacion) #Lo convierte a un diccionario de json
    #d2 = json.loads(d1)
    for i in habitacion:
        if (i['Identificador'] == lista_aux['Identificador']):
            habitacion.remove(i)
            encontrado = True

    if(encontrado == True):
        habitacion.append(lista_aux)
        print("Actualizado correctamente")

    else:
        print("NO se ha encontrado la habitacion")

    with open('data.json', 'w') as fichero:
        json.dump(habitacion,fichero)
```

**Figura 6:** Código fuente del servidor para realizar una modificación en una habitación

## 2.3 Consultar la lista completa de habitaciones

```
elif (response == '3'):
    r = requests.get(url = consulta)
    message = r.json()

    print("\n\t~~~~~DATOS DE HABITACIONES~~~~~\n")

    for i in message['dict']:
        print("=====")
        print ("Id. habitación: "+i['Identificador'])
        print ("Num. plazas disponibles: "+i['num_plazas'])
        print ("Lista de equipamientos: "+i['equipamiento'])
        print ("Ocupada: "+i['ocupada'])
        print ("=====\n")
```

**Figura 7:** Código fuente del cliente para mostrar la lista completa de las habitaciones

A través de una petición GET obtiene del servidor la lista de las habitaciones y la convierte en formato JSON, luego imprime la lista mediante un bucle que va iterando por la lista e imprimiendo por pantalla.

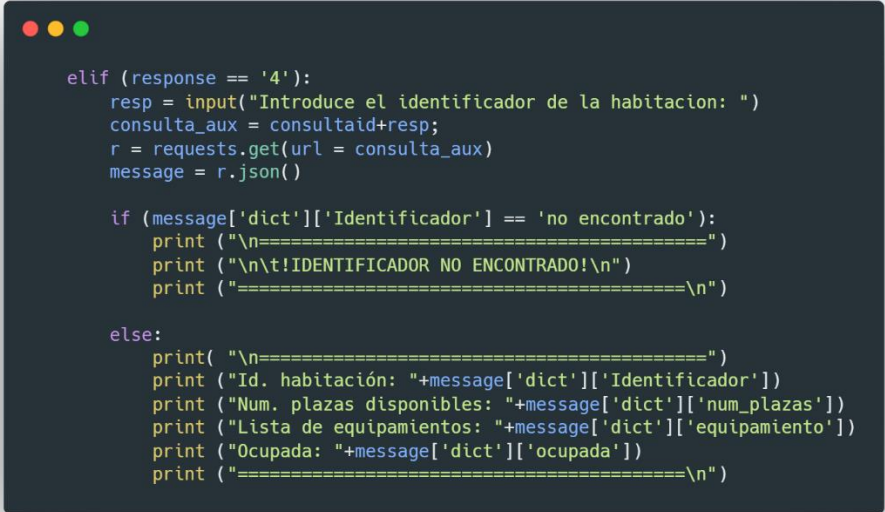


```
@get('/listado')
def mostrar_habitaciones():
    with open('data.json', 'r') as fichero:
        habitacion = json.load(fichero)
    return dict(dict = habitacion)
```

**Figura 8:** Código fuente del servidor para devolver la lista completa de las habitaciones

Al recibir la petición abrirá el archivo JSON “data.json” donde tenemos la información de las habitaciones, leerá y devolverá esta información como diccionario de Python, el cual bottle convierte a JSON

## 2.4 Consultar una habitación mediante identificador



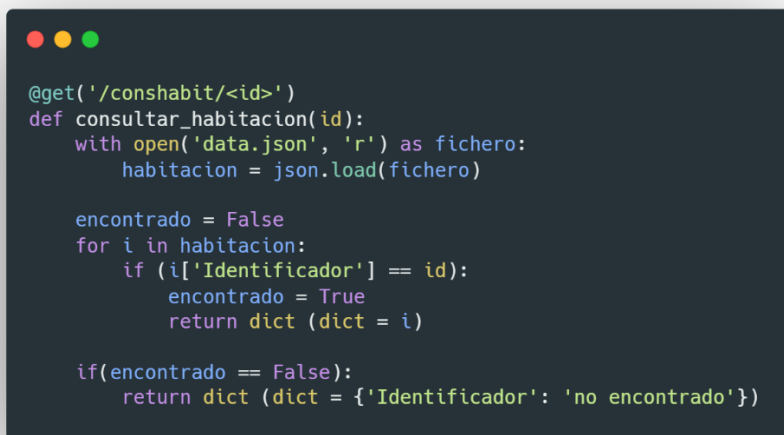
```
elif (response == '4'):
    resp = input("Introduce el identificador de la habitacion: ")
    consulta_aux = consultaaid+resp;
    r = requests.get(url = consulta_aux)
    message = r.json()

    if (message['dict']['Identificador'] == 'no encontrado'):
        print ("\n=====")
        print ("\n\t!IDENTIFICADOR NO ENCONTRADO!\n")
        print ("=====\\n")

    else:
        print( "\\n=====")
        print ("Id. habitación: "+message['dict']['Identificador'])
        print ("Num. plazas disponibles: "+message['dict']['num_plazas'])
        print ("Lista de equipamientos: "+message['dict']['equipamiento'])
        print ("Ocupada: "+message['dict']['ocupada'])
        print ("=====\\n")
```

**Figura 9:** Código fuente del cliente para mostrar la habitación solicitada

Primeramente, se le pedirá al usuario el id de la habitación a buscar, el cual se pasará como parámetro añadiéndose a la ruta utilizada en la petición GET. Si recibe una lista en la cual el campo identificador es igual a “no encontrado” imprime identificador no encontrado ya que no existe una habitación con el id dado, en el caso contrario, ha encontrado la habitación e imprime los diferentes campos de la lista correspondientes a los datos de la habitación.

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function decorated with a Flask route. It opens a JSON file, loads its contents, and iterates through a list of room objects to find one with a matching ID. If found, it returns the object; otherwise, it returns a dictionary with the status 'no encontrado'.

```
@get('/conshabit/<id>')
def consultar_habitacion(id):
    with open('data.json', 'r') as fichero:
        habitacion = json.load(fichero)

    encontrado = False
    for i in habitacion:
        if (i['Identificador'] == id):
            encontrado = True
            return dict(dict = i)

    if(encontrado == False):
        return dict(dict = {'Identificador': 'no encontrado'})
```

**Figura 10:** Código fuente del servidor para devolver la habitación solicitada

Al recibir la petición abrirá el archivo JSON “data.json” donde tenemos la información de las habitaciones, leerá el fichero y comparará el identificador de las habitaciones con el parámetro id buscado si lo encuentra devolverá un diccionario con esa iteración del bucle, es decir, esa habitación que es la buscada. Si no la encuentra devuelve un diccionario con el campo Identificador igual a “no encontrado”.

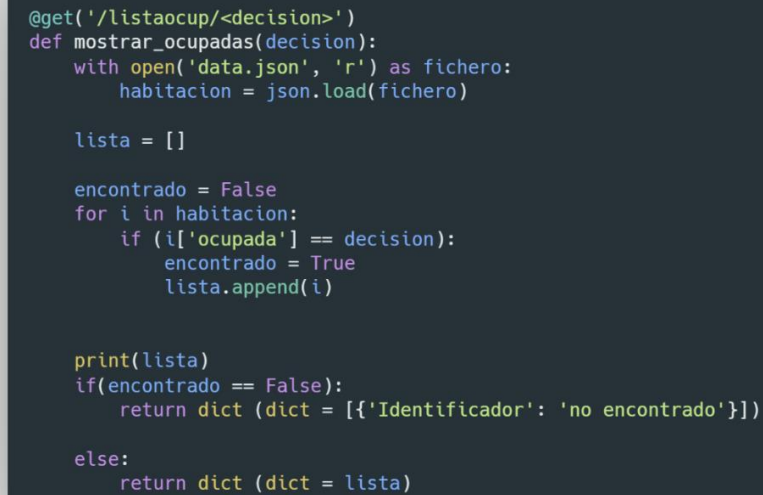


## 2.5 Consultar habitaciones ocupadas o desocupadas



**Figura 11:** Código fuente del cliente para mostrar las habitaciones ocupadas o desocupadas

Primeramente, se le pedirá al usuario si quiere la lista de habitaciones ocupadas o no ocupadas a lo cual deberá responder si o no dependiendo de la que desee, si responde cualquier otra cosa, se le mostrará un error. Seguidamente se añadirá la respuesta como parámetro a la ruta utilizada a la petición GET de la cual recibirá la lista de habitaciones deseadas. A través de un bucle, comprobaremos si el identificador de la habitación es igual a “no encontrado” en cuyo caso significara que no habrá ninguna habitación que cumpla lo pedido por el usuario e imprimirá un mensaje correspondiente. En caso contrario irá imprimiendo todos los campos de las habitaciones obtenidas.

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function decorated with a Flask route. It opens a JSON file, iterates through its contents, filters by a decision parameter, and returns either a list of matching rooms or a 'not found' message.

```
@get('/listaocup/<decision>')
def mostrar_ocupadas(decision):
    with open('data.json', 'r') as fichero:
        habitacion = json.load(fichero)

    lista = []

    encontrado = False
    for i in habitacion:
        if (i['ocupada'] == decision):
            encontrado = True
            lista.append(i)

    print(lista)
    if(encontrado == False):
        return dict (dict = [{'Identificador': 'no encontrado'}])

    else:
        return dict (dict = lista)
```

**Figura 12:** Código fuente del servidor para mostrar las habitaciones ocupadas o desocupadas

Al recibir la petición abrirá el archivo JSON “data.json” donde tenemos la información de las habitaciones y leerá el fichero. Seguidamente crearemos una lista donde iremos guardando las habitaciones deseadas según la decisión del cliente (si/no dependiendo si quiere ocupadas o no ocupadas). Iremos iterando en la lista de habitaciones buscando las que en su campo ocupada concuerde con la decisión del cliente, y las iremos guardando a la lista que creamos antes. Si no encuentra ninguna que concuerde, devolverá un diccionario con el campo identificador igual a “no encontrado”, si ha encontrado devolverá la lista donde hemos guardado las habitaciones que concuerdan.

## 2.6 Consultar números de habitaciones con las plazas deseadas

Cuando se selecciona esta operación, se le solicita al cliente el número inferior y superior de plazas de las habitaciones a buscar, y se añadirán estos parámetros a la ruta, para que el servidor pueda recibirlos como parámetros. Una vez solicitado, el servidor nos devolverá las habitaciones que se encuentren en el rango, en caso contrario nos devolverá “no encontrado” en el valor de la variable Identificador.

```

elif (response == '6'):

    print("Introduzca el intervalo de plazas")
    resp1 = input("Limite inferior de plazas: ")
    resp2 = input("Limite superior de plazas: ")

    if(int(resp1) > int(resp2)):
        print("Error, intervalo mal introducido")
    else:
        r = requests.get(url = plaza+resp1+"/"+resp2)
        message = r.json()

        print("\n")
        for i in message['dict']:
            if (i['Identificador'] == 'no encontrado'):
                print ("\n=====")
                print ("\n\ti NO SE HAN ENCONTRADO HABITACIONES !\n")
                print ("=====")
            else:
                print( "=====")
                print ("Id. habitación: "+i['Identificador'])
                print ("Num. plazas disponibles: "+i['num_plazas'])
                print ("Lista de equipamientos: "+i['equipamiento'])
                print ("Ocupada: "+i['ocupada'])
                print ("=====")

```

**Figura 13:** Código fuente del cliente para mostrar las habitaciones con el número de plazas solicitadas

En cuanto al servidor, éste recibirá en la ruta dos parámetros, num (límite inferior) y num2 (límite superior). Una vez recibido, entrará en un bucle en el que almacenará en una lista auxiliar todas las habitaciones que se encuentren en el rango. Una vez almacenadas todas, se devolverá tal lista al cliente.

```

@get('/plazas/<num>/<num2>')
def plazas_habitacion(num,num2):
    with open('data.json', 'r') as fichero:
        habitacion = json.load(fichero)

    lista = []

    encontrado = False
    for i in habitacion:
        if (int(i['num_plazas']) >= int(num) and int(i['num_plazas']) <= int(num2)):
            encontrado = True
            lista.append(i)

    print(lista)
    if(encontrado == False):
        return dict (dict = [{'Identificador': 'no encontrado'}])

    else:
        return dict (dict = lista)

```

**Figura 14:** Código fuente del servidor para devolver las habitaciones con el número de plazas solicitadas

## 2.7 Borrar una habitación

Cuando el cliente selecciona esta operación, solicitará al servidor la existencia de la habitación. Si ésta existe, solicitará al servidor el borrado de la habitación. El identificador introducido se añade a la ruta a consultar.

```
elif (response == '7'):  
    num_hab = input("Introduce identificador de la habitacion: ")  
    req = requests.get(url = exists+num_hab)  
    data = req.text  
  
    if(data == "NO encontrado"):  
        print("\n*****")  
        print("\n\tNO SE HA ENCONTRADO LA HABITACION\n")  
        print("*****\n")  
    else:  
        requests.post(remover+num_hab)  
        print("\n*****")  
        print("\n\tSE HA BORRADO CORRECTAMENTE\n")  
        print("*****\n")
```

Figura 15: Código fuente del cliente para borrar una habitación

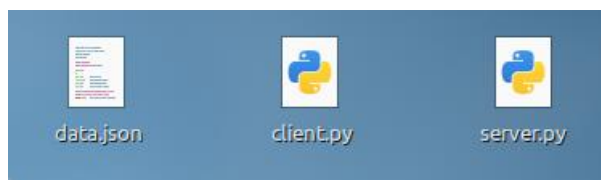
En cuanto al código fuente del servidor, éste recibirá como parámetro el id de la habitación a borrar. A través de un bucle se buscará la posición en la que se encuentra la habitación y procederá a su borrado.

```
@post('/remove/<id>')  
def modificar(id):  
  
    with open('data.json', 'r') as fichero:  
        habitacion = json.load(fichero)  
  
    encontrado = False  
  
    for i in habitacion:  
        if (i['Identificador'] == id):  
            habitacion.remove(i)  
            encontrado = True  
  
    if(encontrado == True):  
        print("Borrado correctamente")  
  
    else:  
        print("NO se ha encontrado la habitacion")  
  
    with open('data.json', 'w') as fichero:  
        json.dump(habitacion,fichero)
```

Figura 16: Código fuente del servidor para borrar una habitación

### 3. Ejecución del programa

Para la ejecución del programa es necesario disponer de la librería Bottle, tal y como se indica en el prefacio de este documento. Disponemos de tres archivos, `server.py` (encargado de recibir y de enviar datos de las habitaciones al cliente), `client.py` (Dispone de una interfaz con la que interactuar o manipular los datos de las habitaciones) y de un archivo JSON, llamado `data.json` (donde se almacenarán los datos de las habitaciones). Este último, si no existe, se creará en el mismo directorio que el archivo `server.py`. Para el correcto funcionamiento del programa es necesario que no se modifique ni se borre el archivo JSON durante la ejecución del servicio.



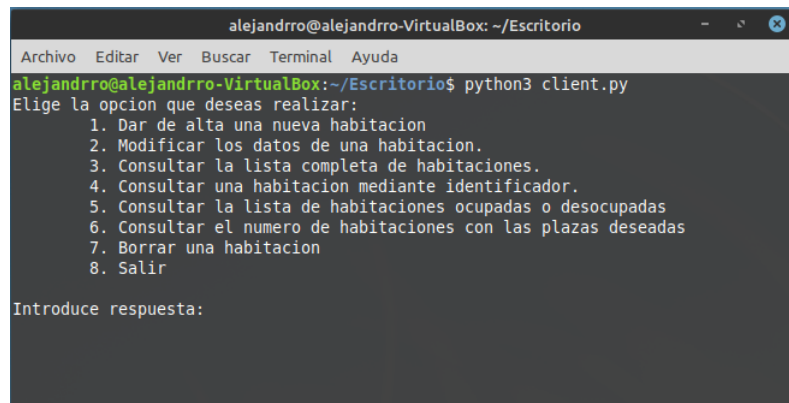
**Figura 17:** Muestra de los ficheros necesarios para ejecutar el servicio

Para comenzar, primero ejecutaremos el archivo `server.py` con Python 3. Una vez ejecutado el servidor, solicitará al administrador/usuario si desea crear una copia de seguridad antes de comenzar a ejecutar el servidor. Posteriormente, este estará preparado para recibir las distintas peticiones del cliente. El servidor quedaría abierto hasta que no se le indique lo contrario, con el comando `Ctrl + c`.

```
alejandro@alejandro-VirtualBox: ~/Escritorio
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
alejandro@alejandro-VirtualBox:~/Escritorio$ python3 server.py
¿Desea hacer un backup?: (si/no) si
Bottle v0.12.18 server starting up (using WSGIRefServer())...
Listening on http://localhost:8081/
Hit Ctrl-C to quit.
```

**Figura 18:** Muestra de la terminal del archivo `server.py`

Posteriormente, podemos iniciar el archivo `client.py` con el que interactuaremos con el servidor. Una vez iniciado se nos mostraría de la siguiente manera:



```
alejandro@alejandro-VirtualBox: ~/Escritorio
Archivo Editar Ver Buscar Terminal Ayuda
alejandro@alejandro-VirtualBox:~/Escritorio$ python3 client.py
Elige la opcion que deseas realizar:
  1. Dar de alta una nueva habitacion
  2. Modificar los datos de una habitacion.
  3. Consultar la lista completa de habitaciones.
  4. Consultar una habitacion mediante identificador.
  5. Consultar la lista de habitaciones ocupadas o desocupadas
  6. Consultar el numero de habitaciones con las plazas deseadas
  7. Borrar una habitacion
  8. Salir

Introduce respuesta:
```

**Figura 19:** Muestra de la terminal del archivo client.py

Finalmente, quedaría a disposición del usuario la elección de cualquiera de las opciones que se proponen.

Hay que destacar, que el programa del cliente está totalmente preparado por si en algún momento el usuario introduce una opción incorrecta, habitación incorrecta...