

## Universidad de Las Américas

Facultad de Ingenierías y Ciencias Aplicadas

*Ingeniería de Software*

Informe de Examen de Progreso 2

### 1. DATOS DEL ALUMNO:

- Mateo Nicolas Velásquez Gallardo
- [mateo.velasquez@udla.edu.ec](mailto:mateo.velasquez@udla.edu.ec)

### 2. INSTRUCCIONES:

Examen Progreso 2.

Adjunto encontrarás el caso de estudio a resolver como evaluación de progreso 2

Diseñar e implementar una solución de integración que combine:

1. Un Servicio Web SOAP para manejar consultas de disponibilidad de habitaciones.
2. Una API REST para realizar y cancelar reservas.
3. Un microservicio para gestionar las operaciones de actualización del inventario de habitaciones.

### 3. Servicio Web SOAP

#### Código

```
from flask import Flask, request, Response
import xml.etree.ElementTree as ET
```

```
app = Flask(__name__)
```

```
# Datos en memoria temporal
```

```
data = [
    {"room_id": 1, "room_type": "simple", "available_date": "2024-12-16", "status":
"disponible"},
    {"room_id": 2, "room_type": "doble", "available_date": "2024-12-16", "status":
"disponible"},
    {"room_id": 3, "room_type": "suite", "available_date": "2024-12-16", "status":
"disponible"},
    {"room_id": 4, "room_type": "simple", "available_date": "2024-12-17", "status":
"disponible"},
    {"room_id": 5, "room_type": "doble", "available_date": "2024-12-17", "status":
"reservada"},
```

```
{ "room_id": 6, "room_type": "suite", "available_date": "2024-12-17", "status":  
"disponible"},  
{ "room_id": 7, "room_type": "simple", "available_date": "2024-12-18", "status": "en  
mantenimiento"},  
{ "room_id": 8, "room_type": "doble", "available_date": "2024-12-18", "status":  
"disponible"},  
{ "room_id": 9, "room_type": "suite", "available_date": "2024-12-18", "status":  
"reservada"},  
{ "room_id": 10, "room_type": "simple", "available_date": "2024-12-19", "status":  
"disponible"},  
{ "room_id": 11, "room_type": "doble", "available_date": "2024-12-19", "status":  
"disponible"},  
{ "room_id": 12, "room_type": "suite", "available_date": "2024-12-19", "status": "en  
mantenimiento"},  
{ "room_id": 13, "room_type": "simple", "available_date": "2024-12-20", "status":  
"disponible"},  
{ "room_id": 14, "room_type": "doble", "available_date": "2024-12-20", "status":  
"reservada"},  
{ "room_id": 15, "room_type": "suite", "available_date": "2024-12-20", "status":  
"disponible"},  
]
```

# Helper function para construir la respuesta XML

```
def build_xml_response(rooms):  
    root = ET.Element("Rooms")  
    for room in rooms:  
        room_element = ET.SubElement(root, "Room")  
        for key, value in room.items():  
            ET.SubElement(room_element, key).text = str(value)  
    return ET.tostring(root, encoding="utf-8", method="xml")
```

@app.route('/soap/availability', methods=['POST'])

```
def soap_availability():  
    # Parsear la solicitud XML  
    try:  
        request_data = request.data.decode('utf-8')  
        root = ET.fromstring(request_data)  
  
        # Extraer los parámetros de la solicitud  
        start_date = root.findtext("start_date")  
        end_date = root.findtext("end_date")  
        room_type = root.findtext("room_type")
```

```
if not start_date or not end_date or not room_type:
    return Response("<Error>Missing parameters</Error>", status=400,
mimetype='application/xml')

# Filtrar las habitaciones según los parámetros
available_rooms = [
    room for room in data
    if room["room_type"] == room_type and
    room["available_date"] >= start_date and
    room["available_date"] <= end_date and
    room["status"] == "disponible"
]

# Construir y devolver la respuesta XML
response_xml = build_xml_response(available_rooms)
return Response(response_xml, status=200, mimetype='application/xml')

except ET.ParseError:
    return Response("<Error>Invalid XML</Error>", status=400,
mimetype='application/xml')

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

### Explicación del Código

Este código implementa un servicio web simple usando Flask para manejar solicitudes SOAP que consultan la disponibilidad de habitaciones en un hotel.

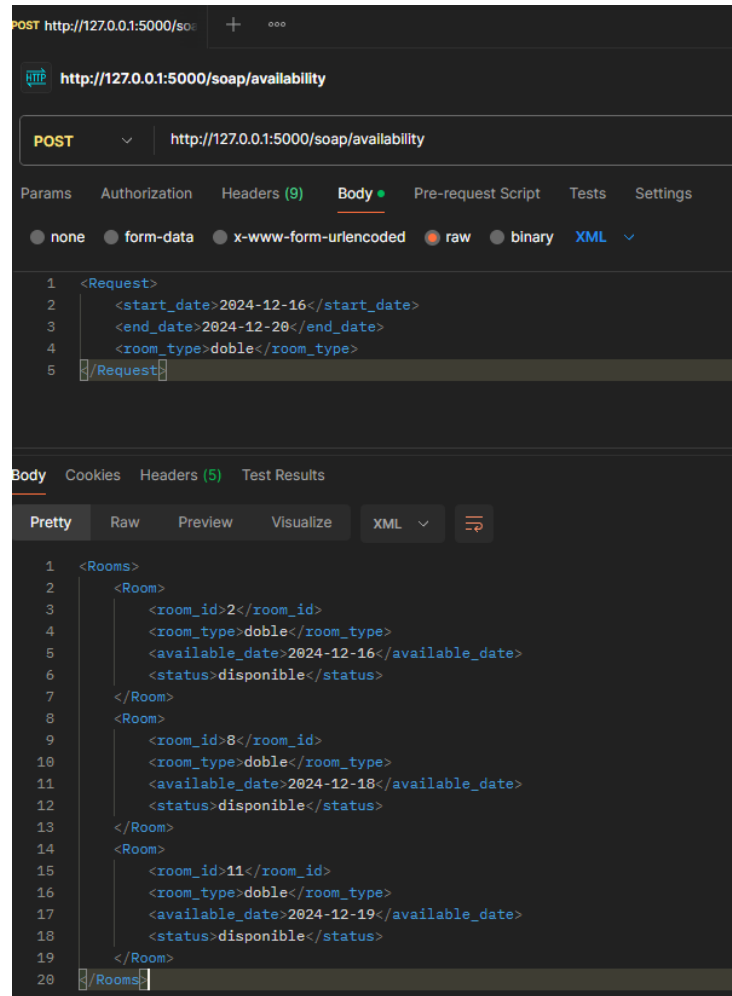
1. Importación de módulos
  - a. Se importa Flask para crear la aplicación web.
  - b. request y Response se usan para manejar las solicitudes y respuestas HTTP.
  - c. xml.etree.ElementTree es utilizado para trabajar con datos XML, que es el formato de entrada y salida del servicio.
2. Datos en memoria
  - a. Se define una lista data con información de las habitaciones del hotel (ID, tipo, fecha de disponibilidad y estado).
  - b. Función build\_xml\_response:

- c. Esta función recibe una lista de habitaciones disponibles y las convierte en un formato XML. Utiliza ElementTree para crear un árbol XML con las habitaciones y sus detalles.
3. Ruta del servicio /soap/availability
  - a. El servicio está disponible en el endpoint /soap/availability y solo acepta solicitudes POST.
  - b. Cuando se recibe una solicitud, se parsea el cuerpo XML para extraer parámetros como start\_date, end\_date y room\_type.
  - c. Si faltan parámetros o el XML es inválido, se responde con un error en formato XML.
4. Filtrado de habitaciones
  - a. Basándose en los parámetros extraídos del XML (fechas y tipo de habitación), se filtran las habitaciones disponibles de la lista data.
  - b. Solo se incluyen aquellas habitaciones que están disponibles (status == "disponible") y que están dentro del rango de fechas solicitado.
5. Respuesta XML
  - a. Después de filtrar las habitaciones, se genera una respuesta en formato XML con los datos de las habitaciones disponibles y se devuelve al cliente con un código de estado 200.
6. Manejo de errores
  - a. Si el XML de entrada es inválido, se devuelve un mensaje de error en XML con código de estado 400.
7. Ejecución de la aplicación
  - a. El servidor Flask se ejecuta en el puerto 5000 en modo de depuración (debug=True).

## Evidencias

```
WebSoap.py X request.xml
WebSoap.py > ...
1 from flask import Flask, request, Response
2 import xml.etree.ElementTree as ET
3
4 app = Flask(__name__)
5
6 # Datos en memoria temporal
7 data = [
8     {"room_id": 1, "room_type": "simple", "available_date": "2024-12-16", "status": "disponible"},
9     {"room_id": 2, "room_type": "doble", "available_date": "2024-12-16", "status": "disponible"},
10    {"room_id": 3, "room_type": "suite", "available_date": "2024-12-16", "status": "disponible"},
11    {"room_id": 4, "room_type": "simple", "available_date": "2024-12-17", "status": "disponible"},
12    {"room_id": 5, "room_type": "doble", "available_date": "2024-12-17", "status": "reservada"},
13    {"room_id": 6, "room_type": "suite", "available_date": "2024-12-17", "status": "disponible"},
14    {"room_id": 7, "room_type": "simple", "available_date": "2024-12-18", "status": "en mantenimiento"},
15    {"room_id": 8, "room_type": "doble", "available_date": "2024-12-18", "status": "disponible"},
16    {"room_id": 9, "room_type": "suite", "available_date": "2024-12-18", "status": "reservada"},
17    {"room_id": 10, "room_type": "simple", "available_date": "2024-12-19", "status": "disponible"},
18    {"room_id": 11, "room_type": "doble", "available_date": "2024-12-19", "status": "disponible"},
19    {"room_id": 12, "room_type": "suite", "available_date": "2024-12-19", "status": "en mantenimiento"},
20    {"room_id": 13, "room_type": "simple", "available_date": "2024-12-20", "status": "disponible"},
21    {"room_id": 14, "room_type": "doble", "available_date": "2024-12-20", "status": "reservada"}
22 ]
23
24 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\maten\OneDrive\Escritorio\Universidad\Integración de Sistemas\Examen P3\Servicio Web SOAP Disponibilidad de Habitaciones> & C:/Users/maten/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/maten/OneDrive/Escritorio/Universidad/Integración de Sistemas/Examen P3/Servicio Web SOAP Disponibilidad de Habitaciones/web Soap.py"
* Serving Flask app "WebSoap"
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 112-897-140
```

*Levantamiento del Servicio Web SOAP*



*Consulta por medio de XML en Postman (Fechas-Tipo)*

## 4. API REST

### Código

```
from flask import Flask, request, jsonify
```

```
app = Flask(__name__)
```

```
# Datos en memoria temporal para reservas
```

```
reservations = []
```

```
next_id = 1 # Variable global para generar IDs incrementales
```

```
# Helper function para encontrar una reserva por ID
```

```
def find_reservation(reservation_id):
```

```
    for reservation in reservations:
```

```
        if reservation["reservation_id"] == reservation_id:
            return reservation
    return None

# Endpoint para crear una nueva reserva
@app.route('/reservations', methods=['POST'])
def create_reservation():
    global next_id
    data = request.get_json()

    # Validar los parámetros de entrada
    required_fields = ["room_number", "customer_name", "start_date", "end_date"]
    for field in required_fields:
        if field not in data:
            return jsonify({"error": f"Missing field: {field}"}), 400

    # Crear la reserva
    reservation = {
        "reservation_id": str(next_id), # Generar un ID incremental
        "room_number": data["room_number"],
        "customer_name": data["customer_name"],
        "start_date": data["start_date"],
        "end_date": data["end_date"],
        "status": "active"
    }
    reservations.append(reservation)
    next_id += 1 # Incrementar el ID para la próxima reserva

    return jsonify(reservation), 201

# Endpoint para consultar una reserva específica
@app.route('/reservations/<reservation_id>', methods=['GET'])
def get_reservation(reservation_id):
    reservation = find_reservation(reservation_id)
    if reservation is None:
        return jsonify({"error": "Reservation not found"}), 404

    return jsonify(reservation), 200

# Endpoint para cancelar una reserva
@app.route('/reservations/<reservation_id>', methods=['DELETE'])
def cancel_reservation(reservation_id):
```

```
reservation = find_reservation(reservation_id)
if reservation is None:
    return jsonify({"error": "Reservation not found"}), 404

reservations.remove(reservation)
return jsonify({"message": "Reservation canceled"}), 200

if __name__ == '__main__':
    app.run(debug=True, port=5001)
```

### Explicación del Código

Este código implementa una API en Flask para manejar reservas de habitaciones de un hotel. Permite crear, consultar y cancelar reservas. A continuación, te explico los componentes clave:

1. Importación de módulos
  - a. Se importa Flask para crear la aplicación web.
  - b. request se usa para obtener datos de las solicitudes HTTP, y jsonify para devolver respuestas en formato JSON.
2. Datos en memoria
  - a. Se utiliza una lista reservations para almacenar las reservas, y una variable global next\_id que lleva un seguimiento del ID incremental para las reservas.
3. Función find\_reservation
  - a. Esta función busca una reserva por su reservation\_id en la lista reservations. Si la encuentra, la devuelve; si no, retorna None.
4. Endpoint para crear una nueva reserva (POST /reservations)
  - a. Este endpoint recibe una solicitud POST con datos en formato JSON.
  - b. Primero, valida que los campos requeridos (room\_number, customer\_name, start\_date, end\_date) estén presentes en los datos.
  - c. Si falta algún campo, responde con un error 400.
  - d. Si los datos son válidos, genera un ID único para la nueva reserva, la agrega a la lista reservations, y responde con la reserva recién creada y un código 201 (creado).
  - e. Después, incrementa el valor de next\_id para la siguiente reserva.

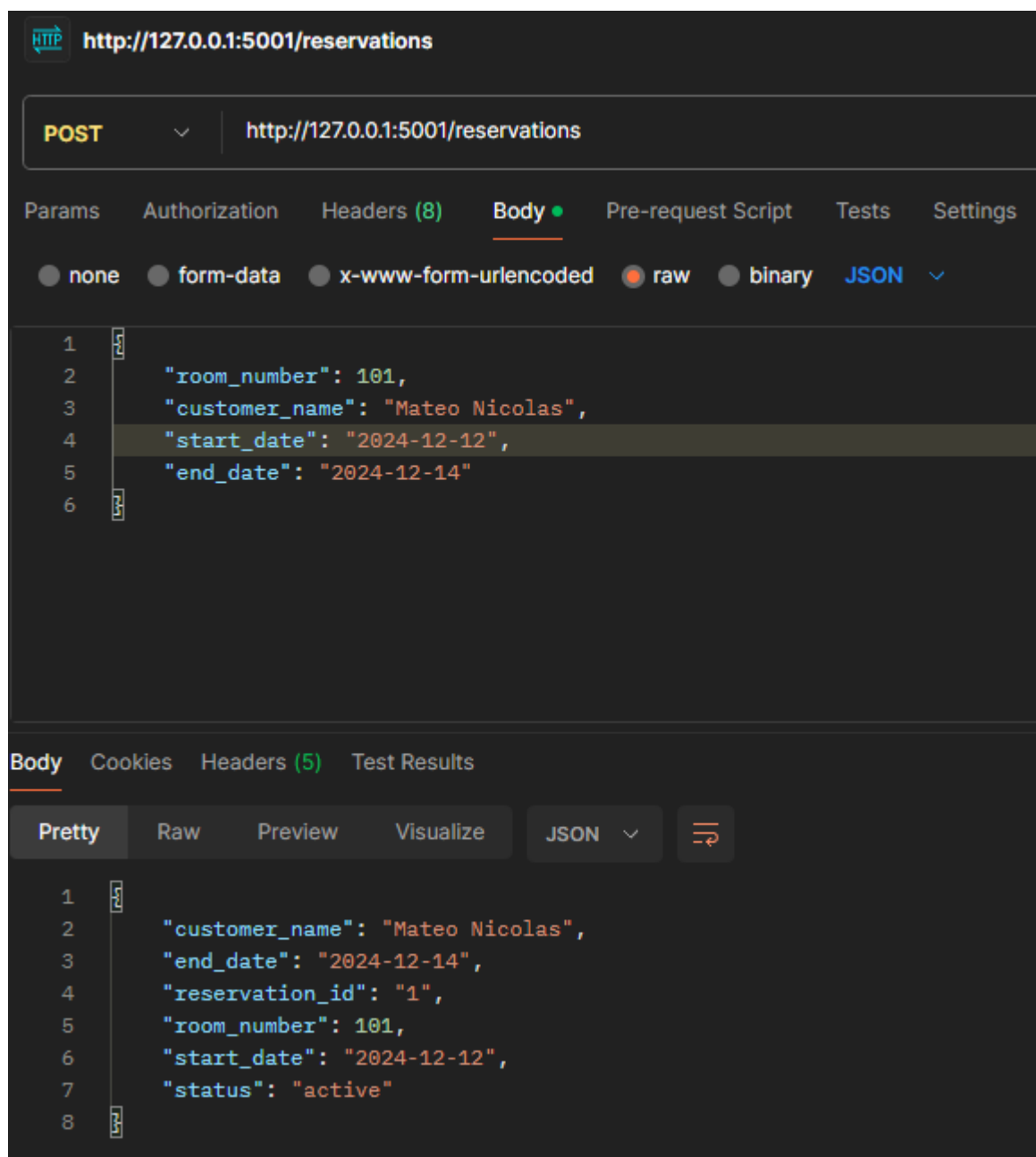


5. Endpoint para consultar una reserva específica (GET /reservations/<reservation\_id>)
  - a. Este endpoint permite consultar una reserva específica usando el reservation\_id como parámetro en la URL.
  - b. Llama a la función find\_reservation para buscar la reserva. Si no la encuentra, responde con un error 404 (no encontrado).
  - c. Si encuentra la reserva, responde con los datos de la reserva y un código 200 (éxito).
6. Endpoint para cancelar una reserva (DELETE /reservations/<reservation\_id>)
  - a. Este endpoint permite cancelar una reserva específica utilizando su reservation\_id en la URL.
  - b. Llama a la función find\_reservation para buscar la reserva. Si no la encuentra, responde con un error 404.
  - c. Si encuentra la reserva, la elimina de la lista reservations y responde con un mensaje de éxito y un código 200 (éxito).
7. Ejecución de la aplicación
  - a. El servidor Flask se ejecuta en el puerto 5001 en modo de depuración (debug=True).

## Evidencias

```
ApiRest.py U X
API REST Gestión de Reserva > ApiRest.py > create_reservation
6 reservations = []
7 next_id = 1 # Variable global para generar IDs incrementales
8
9 # Helper function para encontrar una reserva por ID
10 def find_reservation(reservation_id):
11     for reservation in reservations:
12         if reservation["reservation_id"] == reservation_id:
13             return reservation
14     return None
15
16 # Endpoint para crear una nueva reserva
17 @app.route('/reservations', methods=['POST'])
18 def create_reservation():
19     global next_id
20     data = request.get_json()
21
22     # Validar los parámetros de entrada
23     required_fields = ["room_number", "customer_name", "start_date", "end_date"]
24     for field in required_fields:
25         if field not in data:
26             return jsonify({"error": f"Missing field: {field}"}), 400
27
28     # Crear la reserva
29     reservation = {
30         "reservation_id": str(next_id), # Generar un ID incremental
31
32 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\maten\OneDrive\Escritorio\Universidad\Integración de Sistemas\Examen P3> & C:/Users/maten/AppData/Local/P
e Sistemas/Examen P3/API REST Gestión de Reserva/ApiRest.py"
* Serving Flask app 'ApiRest'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
* Running on http://127.0.0.1:5001
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 112-897-140
127.0.0.1 - - [15/Dec/2024 16:18:39] "DELETE /reservations/1 HTTP/1.1" 404 -
127.0.0.1 - - [15/Dec/2024 16:18:42] "DELETE /reservations/2 HTTP/1.1" 404 -
127.0.0.1 - - [15/Dec/2024 16:18:45] "DELETE /reservations/3 HTTP/1.1" 404 -
127.0.0.1 - - [15/Dec/2024 16:18:58] "POST /reservations HTTP/1.1" 201 -
```

*Levantamiento del Servicio API REST*



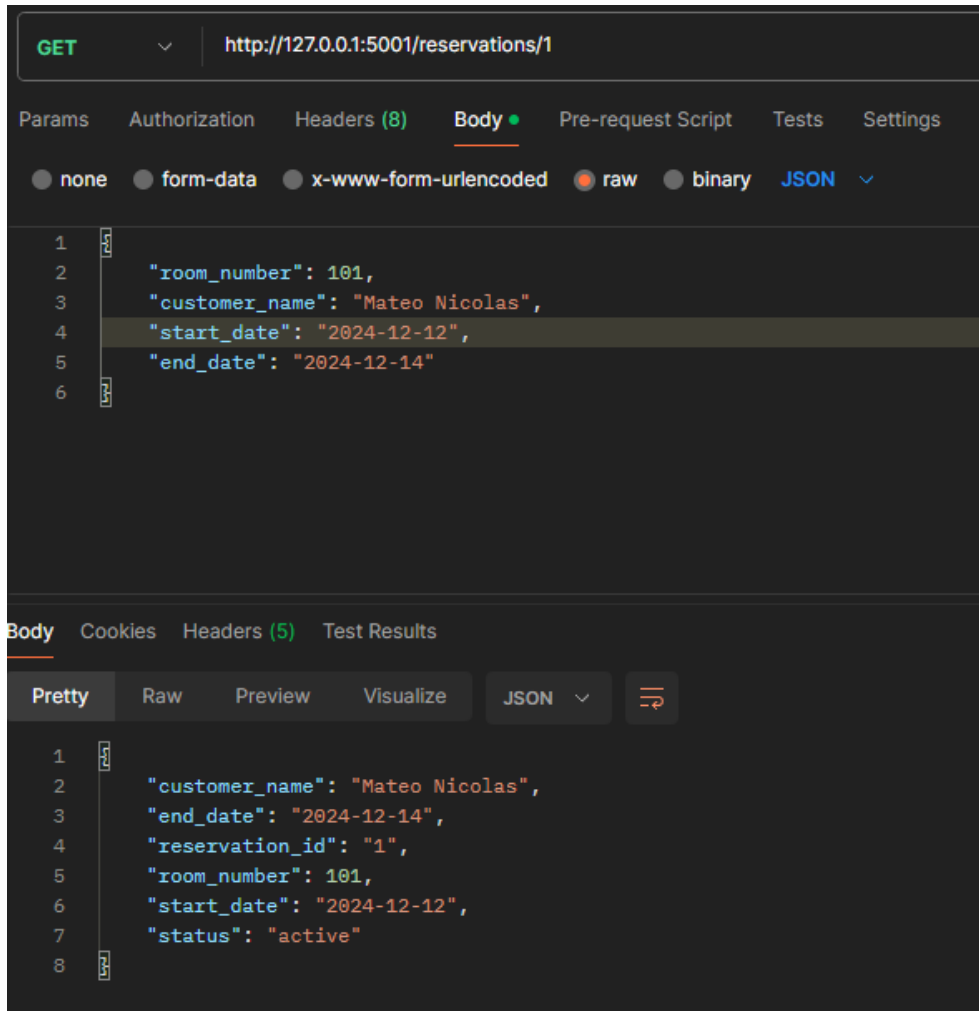
The screenshot displays a REST client interface with the following details:

- URL:** `http://127.0.0.1:5001/reservations`
- Method:** `POST`
- Body Type:** `JSON` (selected from a dropdown menu that also includes `none`, `form-data`, `x-www-form-urlencoded`, `raw`, and `binary`).
- Body Content:**

```
1  {
2    "room_number": 101,
3    "customer_name": "Mateo Nicolas",
4    "start_date": "2024-12-12",
5    "end_date": "2024-12-14"
6  }
```
- Response Section:** Includes tabs for `Body`, `Cookies`, `Headers (5)`, and `Test Results`. The `Body` tab is active, showing a `Pretty` view of the response JSON:

```
1  {
2    "customer_name": "Mateo Nicolas",
3    "end_date": "2024-12-14",
4    "reservation_id": "1",
5    "room_number": 101,
6    "start_date": "2024-12-12",
7    "status": "active"
8  }
```

*Método POST Reserva*



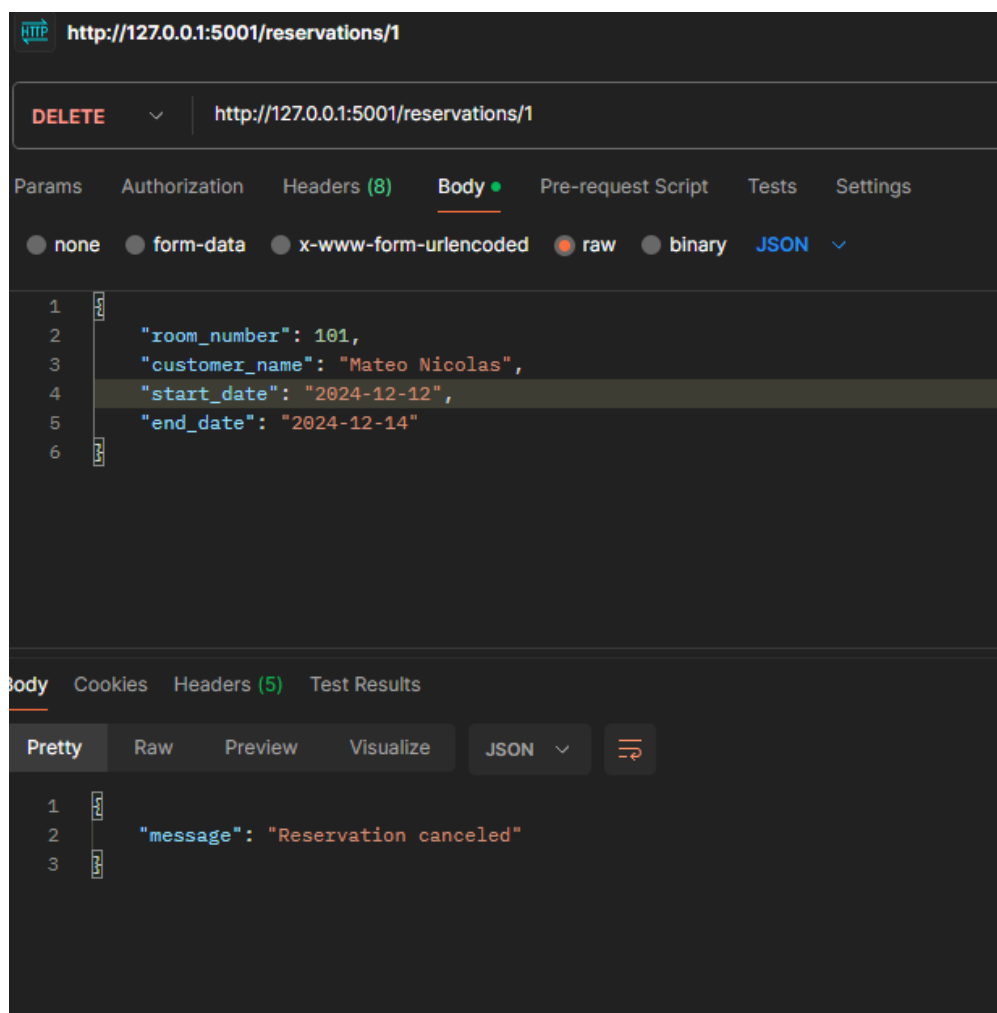
The screenshot displays a REST client interface. At the top, a GET request is configured for the URL `http://127.0.0.1:5001/reservations/1`. The 'Body' tab is selected, showing a JSON payload with the following structure:

```
1 {
2   "room_number": 101,
3   "customer_name": "Mateo Nicolas",
4   "start_date": "2024-12-12",
5   "end_date": "2024-12-14"
6 }
```

Below the request, the 'Body' tab of the response is shown, displaying a JSON object with the following structure:

```
1 {
2   "customer_name": "Mateo Nicolas",
3   "end_date": "2024-12-14",
4   "reservation_id": "1",
5   "room_number": 101,
6   "start_date": "2024-12-12",
7   "status": "active"
8 }
```

*Método GET*



*Método DELETE*

## 5. Microservicio

### Código

```
from flask import Flask, request, jsonify

app = Flask(__name__)

# Datos en memoria temporal para habitaciones
rooms = []
next_room_id = 1 # Variable global para generar IDs incrementales

# Helper function para encontrar una habitación por ID
def find_room(room_id):
    for room in rooms:
        if room["room_id"] == room_id:
            return room
    return None

# Endpoint para registrar una nueva habitación
@app.route('/rooms', methods=['POST'])
def create_room():
    global next_room_id
    data = request.get_json()

    # Validar los parámetros de entrada
    required_fields = ["room_number", "room_type", "status"]
    for field in required_fields:
        if field not in data:
            return jsonify({"error": f"Missing field: {field}"}), 400

    # Crear la habitación
    room = {
        "room_id": str(next_room_id), # Generar un ID incremental
        "room_number": data["room_number"],
        "room_type": data["room_type"],
        "status": data["status"]
    }
    rooms.append(room)
    next_room_id += 1 # Incrementar el ID para la próxima habitación

    return jsonify(room), 201
```

```
# Endpoint para actualizar el estado de una habitación
@app.route('/rooms/<room_id>', methods=['PATCH'])
def update_room_status(room_id):
    room = find_room(room_id)
    if room is None:
        return jsonify({"error": "Habitacion no encontrada"}), 404

    data = request.get_json()
    if "status" not in data:
        return jsonify({"error": "Missing field: status"}), 400

    # Actualizar el estado de la habitación
    room["status"] = data["status"]
    return jsonify(room), 200

if __name__ == '__main__':
    app.run(debug=True, port=5002)
```

### Explicación del Código

Este código implementa una API en Flask para manejar la creación y actualización del estado de habitaciones en un sistema de reservas de un hotel. A continuación, te explico los componentes clave:

1. Importación de módulos a. Flask
  - a. Se importa Flask para crear la aplicación web que gestionará las solicitudes HTTP.
  - b. request: Se utiliza para obtener datos de las solicitudes HTTP, como el cuerpo de la solicitud en formato JSON.
  - c. jsonify: Se usa para devolver respuestas en formato JSON.
2. Datos en memoria
  - a. Se utiliza una lista rooms para almacenar las habitaciones del hotel.
  - b. Una variable global next\_room\_id se usa para llevar un seguimiento del ID incremental de las habitaciones.
3. Función find\_room
  - a. Esta función busca una habitación por su room\_id en la lista rooms. Si encuentra la habitación, la devuelve; si no, retorna None.
4. Endpoint para crear una nueva habitación (POST /rooms)
  - a. Este endpoint permite crear una nueva habitación. Recibe una solicitud POST con los datos de la habitación en formato JSON.

- b. Primero, valida que los campos requeridos (como room\_number, room\_type y status) estén presentes en los datos.
  - c. Si falta alguno de estos campos, responde con un error 400 indicando el campo faltante.
  - d. Si los datos son válidos, genera un ID único para la nueva habitación, la agrega a la lista rooms, y responde con los datos de la habitación recién creada junto con un código 201 (creado).
  - e. Después, incrementa el valor de next\_room\_id para que el próximo ID de habitación sea único.
- 5. Endpoint para actualizar el estado de una habitación (PATCH /rooms/<room\_id>)
  - a. Este endpoint permite actualizar el estado de una habitación específica utilizando su room\_id como parámetro en la URL.
  - b. Llama a la función find\_room para buscar la habitación. Si no la encuentra, responde con un error 404 (no encontrada).
  - c. Si encuentra la habitación, valida que el campo status esté presente en los datos enviados.
  - d. Si falta el campo status, responde con un error 400 indicando que el campo está ausente.
  - e. Si el campo status está presente, actualiza el estado de la habitación y responde con los datos de la habitación actualizada y un código 200 (éxito).
- 6. Ejecución de la aplicación
  - a. La aplicación se ejecuta en el puerto 5002 y en modo de depuración (debug=True), lo que permite ver detalles adicionales en caso de error.



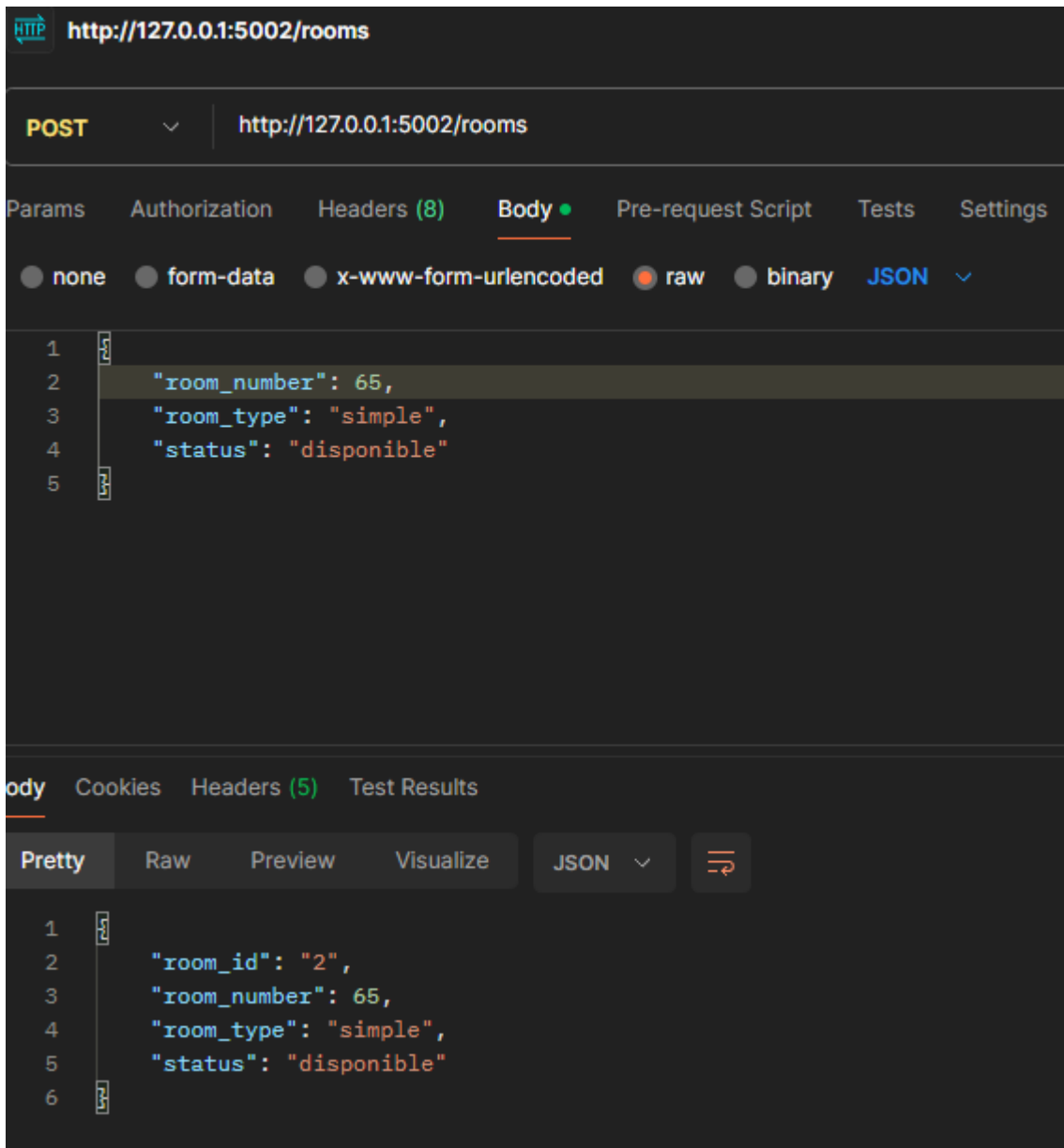
## Evidencias

```
Microservicio.py U X {} MetodoPOST.json U
Microservicio Gestión del Inventario > Microservicio.py > create_room
1 from flask import Flask, request, jsonify
2
3 app = Flask(__name__)
4
5 # Datos en memoria temporal para habitaciones
6 rooms = []
7 next_room_id = 1 # Variable global para generar IDs incrementales
8
9 # Helper function para encontrar una habitación por ID
10 def find_room(room_id):
11     for room in rooms:
12         if room["room_id"] == room_id:
13             return room
14     return None
15
16 # Endpoint para registrar una nueva habitación
17 @app.route('/rooms', methods=['POST'])
18 def create_room():
19     global next_room_id
20     data = request.get_json()
21
22     # Validar los parámetros de entrada
23     required_fields = ["room_number", "room_type", "status"]
24     for field in required_fields:
25         if field not in data:
26             return jsonify({"error": "Missing required field: " + field}), 400
27
28     # Crear nueva habitación
29     new_room = {
30         "room_id": next_room_id,
31         "room_number": data["room_number"],
32         "room_type": data["room_type"],
33         "status": data["status"]
34     }
35     rooms.append(new_room)
36     next_room_id += 1
37
38     return jsonify(new_room), 201
39
40 if __name__ == '__main__':
41     app.run(debug=True)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
PS C:\Users\maten\OneDrive\Escritorio\Universidad\Integración de Sistemas\Examen P3> & C:/Users/maten/AppData/Local/Programs/Python/Python312/python.exe e
e Sistemas/Examen P3/Microservicio Gestión del Inventario/Microservicio.py"
* Serving Flask app 'Microservicio'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5082
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 112-897-140
127.0.0.1 - - [15/Dec/2024 16:46:40] "POST /rooms/1 HTTP/1.1" 405 -
127.0.0.1 - - [15/Dec/2024 16:46:45] "POST /rooms/2 HTTP/1.1" 405 -
127.0.0.1 - - [15/Dec/2024 16:46:47] "POST /rooms/1 HTTP/1.1" 405 -
127.0.0.1 - - [15/Dec/2024 16:46:56] "GET /rooms/1 HTTP/1.1" 405 -
127.0.0.1 - - [15/Dec/2024 16:47:03] "POST /rooms HTTP/1.1" 201 -
127.0.0.1 - - [15/Dec/2024 16:47:08] "GET /rooms/1 HTTP/1.1" 405 -
127.0.0.1 - - [15/Dec/2024 16:47:37] "PATCH /rooms/1 HTTP/1.1" 200 -
```

*Levantamiento del Microservicio*



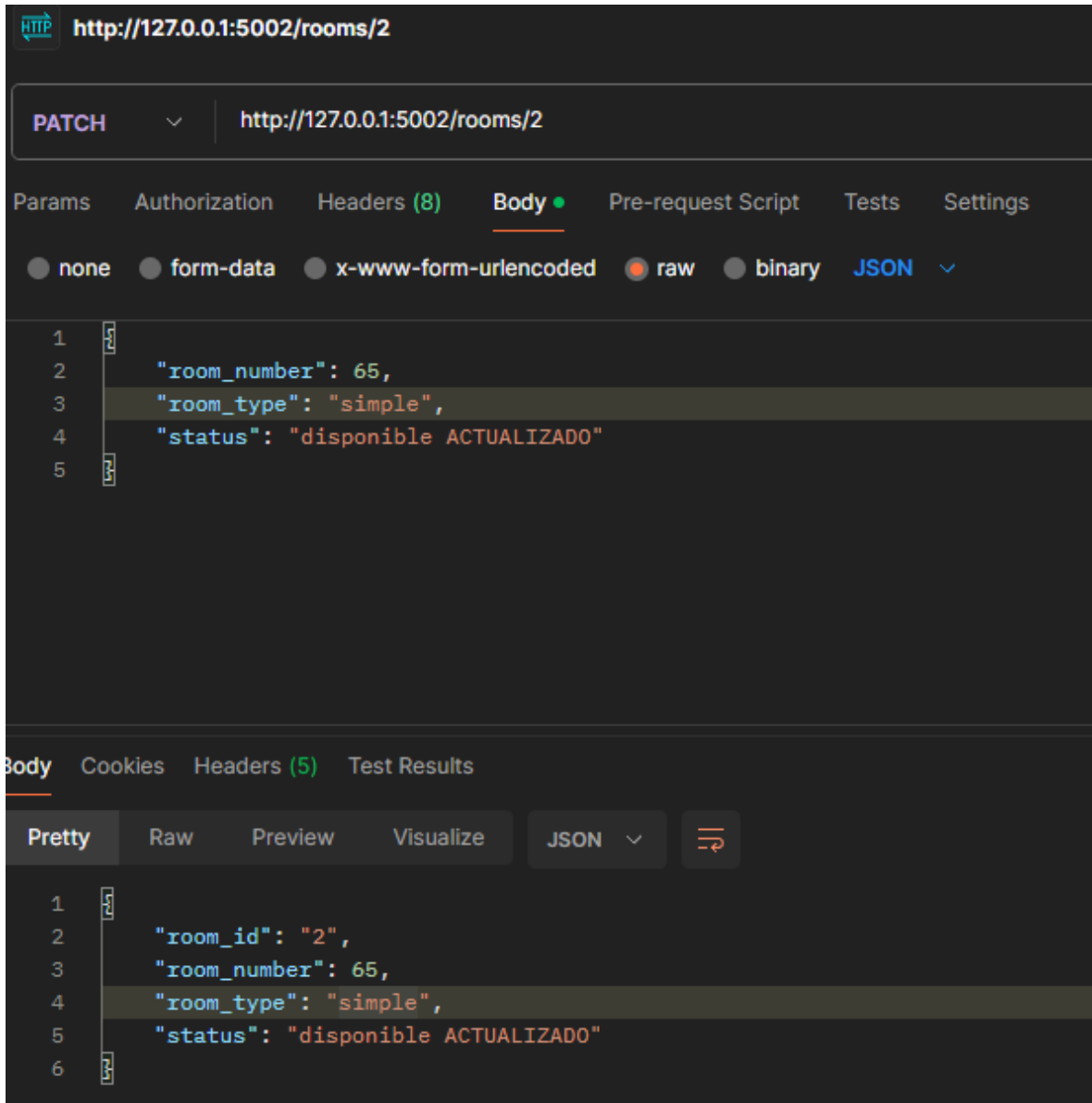
The screenshot displays a REST client interface. At the top, the URL bar shows `http://127.0.0.1:5002/rooms`. Below it, the method is set to **POST**. The request body is configured as **JSON** and contains the following data:

```
1 {
2   "room_number": 65,
3   "room_type": "simple",
4   "status": "disponible"
5 }
```

The bottom section shows the response body, which is also in **JSON** format and includes an additional `room_id` field:

```
1 {
2   "room_id": "2",
3   "room_number": 65,
4   "room_type": "simple",
5   "status": "disponible"
6 }
```

*Método POST*



*Método PATCH*

#### Link del repositorio GitHub

<https://github.com/MateVelasquez/Examen-P3.git>

#### Link del Video Demostrativo

<https://youtu.be/6ibVEG4Q6eA>