Portal de Reservas para Espacios de Trabajo Compartido

Descripción

Objetivo del Proyecto:
El objetivo principal del portal es facilitar la reserva de espacios de trabajo compartido (coworking) para usuarios que necesitan un lugar cómodo y equipado para trabajar temporalmente. El portal permitirá a los propietarios de espacios de trabajo ofrecer sus instalaciones a los usuarios, quienes podrán explorar, reservar y dejar reseñas sobre los espacios utilizados.

Descripción General:
El Portal de Reservas para Espacios de Trabajo Compartido es una plataforma web diseñada para conectar a personas que buscan un lugar para trabajar con propietarios de espacios de coworking. Los usuarios pueden registrarse en el portal, buscar espacios disponibles, realizar reservas en línea, y dejar reseñas sobre su experiencia. Los propietarios de los espacios pueden administrar sus ofertas, controlar la disponibilidad y gestionar las reservas realizadas por los usuarios.

Funcionalidades Principales:

1. **Registro y Autenticación de Usuarios:**
- **Registro de Usuarios**: Los usuarios pueden registrarse en la plataforma proporcionando su nombre, email, contraseña, y otros datos personales.
- **Autenticación:** Los usuarios pueden iniciar sesión para acceder a sus perfiles, hacer reservas y revisar el historial de reservas.

2. **Gestión de Espacios de Trabajo:**
- **Administración de Espacios:** Los propietarios pueden registrar sus espacios de trabajo en la plataforma, proporcionando detalles como nombre, ubicación, servicios disponibles, y tarifas.
- **Gestión de Disponibilidad:** Los propietarios pueden establecer y actualizar la disponibilidad de sus espacios para diferentes fechas y horas.

3. **Sistema de Reservas:**
- **Reserva de Espacios:** Los usuarios pueden buscar espacios de trabajo disponibles según su ubicación y necesidades, visualizar detalles y fotos del espacio, y realizar reservas para fechas y horarios específicos.
- **Confirmación y Estado de Reservas:** Las reservas pueden tener estados como "pendiente", "confirmada", "completada" o "cancelada", según el progreso de la reserva.

4. **Sistema de Reseñas:**
- **Calificación y Comentarios:** Después de utilizar un espacio de trabajo, los usuarios pueden dejar una reseña calificando el espacio y proporcionando comentarios sobre su experiencia.
- **Visualización de Reseñas:** Las reseñas son visibles para otros usuarios interesados en reservar el mismo espacio.

5. **Notificaciones y Alertas:**
- **Alertas de Reservas:** Los usuarios y propietarios reciben notificaciones sobre las reservas realizadas, confirmadas o canceladas.
- **Recordatorios:** Los usuarios pueden recibir recordatorios antes de su reserva para asegurar que no olviden su cita.

Consideraciones Técnicas:
- **Backend**: El sistema estará desarrollado utilizando Node.js con un framework flask para manejar las solicitudes y MongoDB como base de datos.
- **Frontend:** La interfaz de usuario se desarrollará utilizando Angular, proporcionando una experiencia de usuario fluida y dinámica.
- **Base de Datos:** MongoDB será utilizado para almacenar los datos de usuarios, espacios de trabajo, reservas y reseñas, aprovechando su flexibilidad y escalabilidad.

Casos de Uso:
1. Un usuario registrado busca un espacio de coworking en su ciudad, lo reserva para un día completo y recibe una confirmación instantánea de la reserva.
2. Un propietario de un espacio de coworking recibe una solicitud de reserva, la confirma, y posteriormente recibe una reseña positiva del usuario.
3. Un usuario nuevo se registra en la plataforma, explora los espacios disponibles, y se suscribe a notificaciones de disponibilidad para ser informado cuando haya un espacio que cumpla con sus necesidades.

Usuarios Objetivo:
- **Freelancers y Profesionales Remotos:** Personas que necesitan un lugar temporal para trabajar con buenas instalaciones y ambiente profesional.
- **Pequeñas Empresas y Startups:** Equipos que buscan espacios flexibles para reuniones, trabajo en equipo o proyectos cortos.
- **Propietarios de Espacios de Coworking:** Dueños de instalaciones que desean maximizar la utilización de sus espacios ofreciendo a través del portal.

Una vez analizado nuestro proyecto, podremos crear nuestro modelo Entidad Relación para nuestro sistema.

Modelo E-R

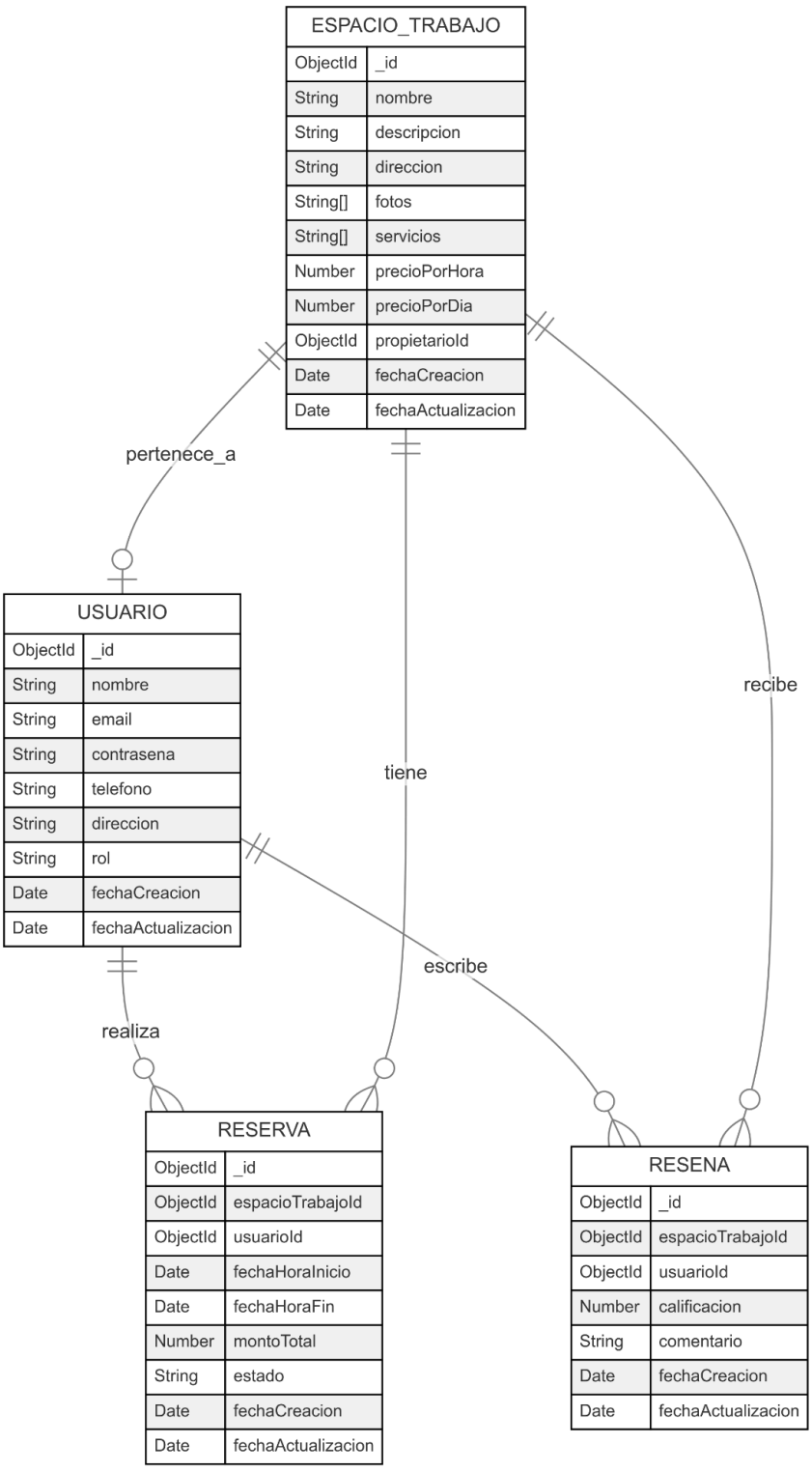


Figura 1: Modelo E-R.

Como siguiente, crearemos nuestros diagramas de secuencia.

Registro de Usuario

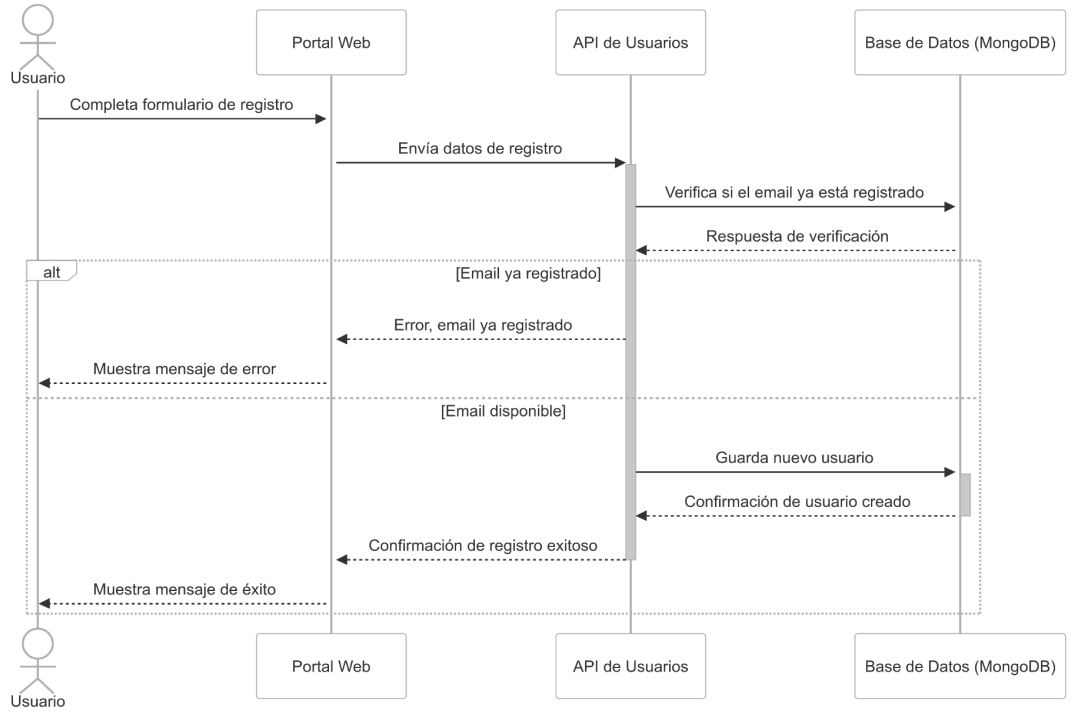


Figura 2: Diagrama de secuencia de Registro.

Login

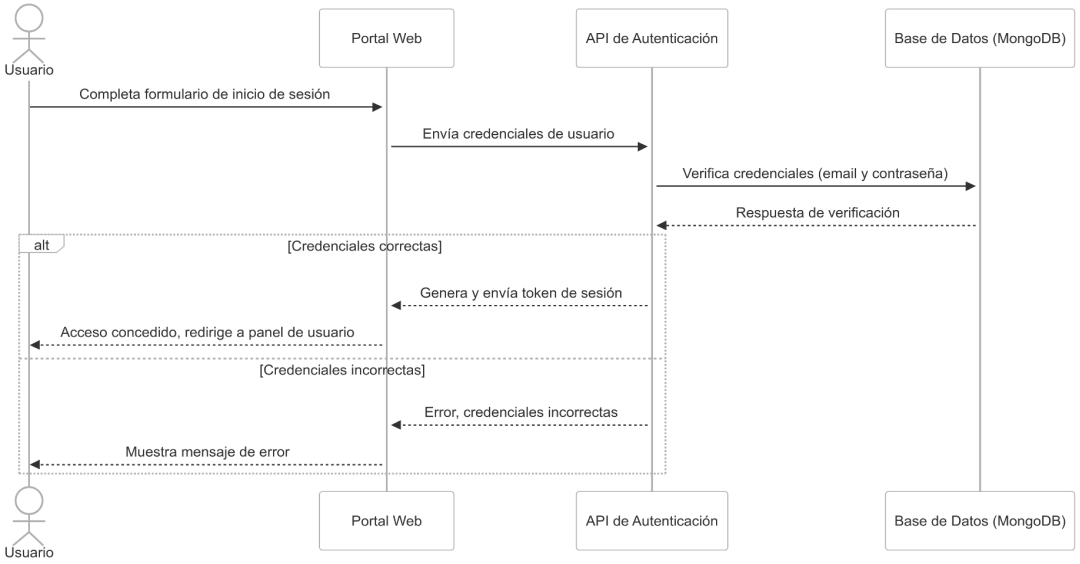


Figura 3: Diagrama de secuencia de Login.

Reserva

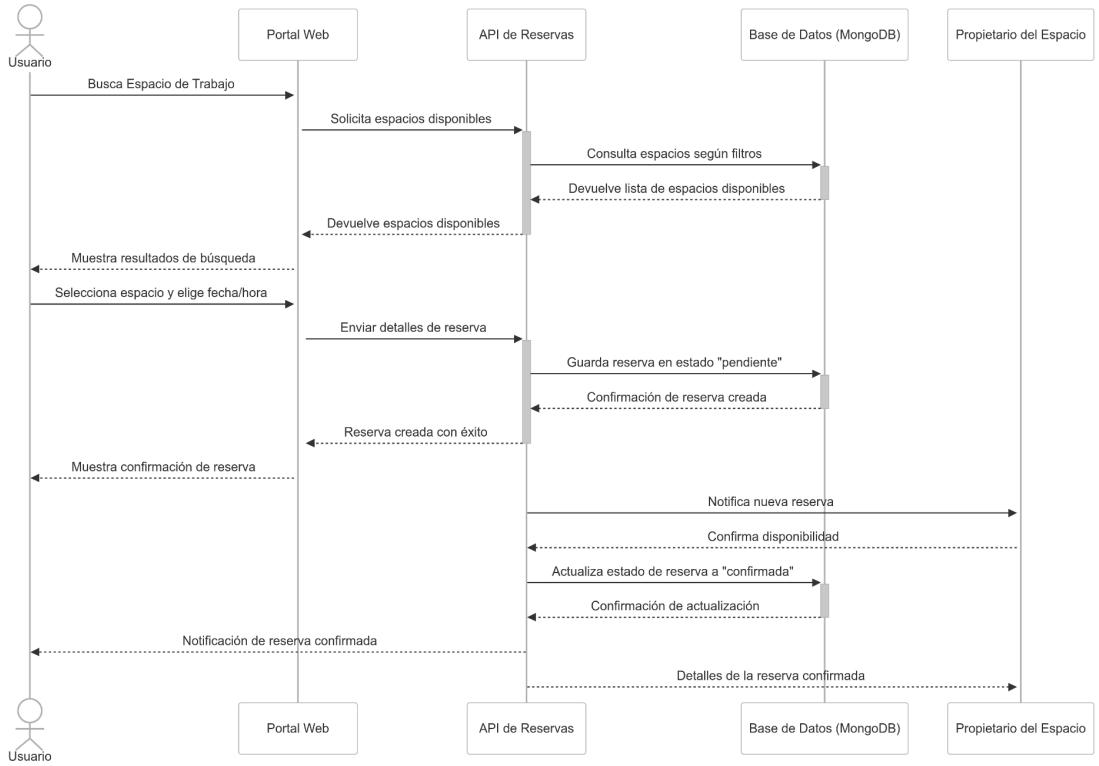


Figura 4: Diagrama de secuencia de Reserva.

Gestión de Espacios

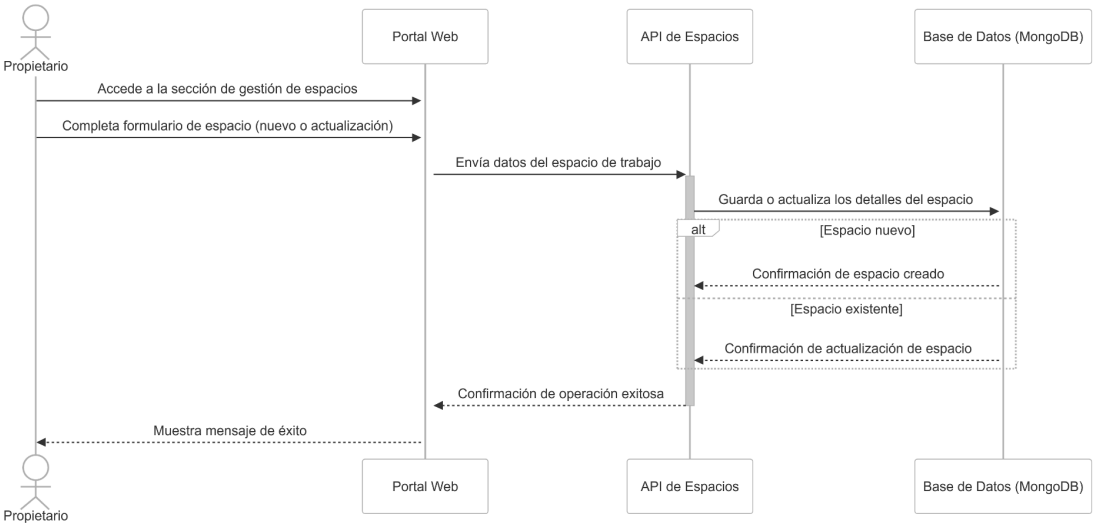


Figura 5: Diagrama de secuencia Gestión de Espacios.

Gestión de Espacios con Notificaciones

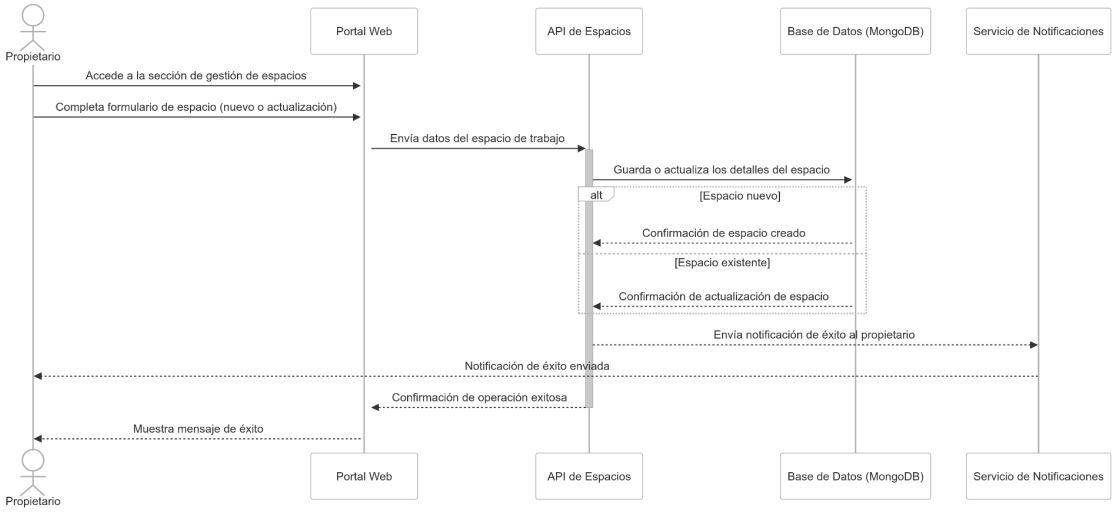


Figura 6: Diagrama de secuencia Gestión de Espacios con Notificaciones.

Reseña

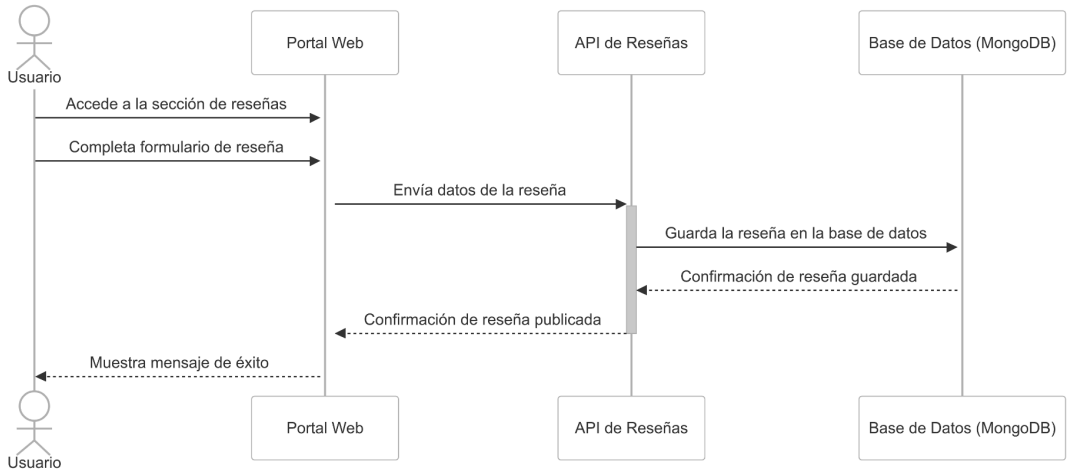


Figura 7: Diagrama de secuencia de Reseña.

Reseñas con Notificaciones

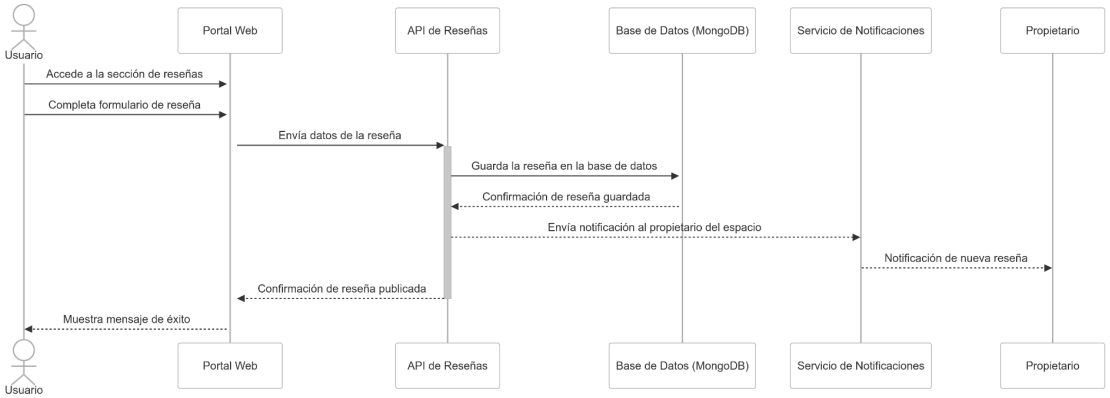


Figura 8: Diagrama de secuencia de Reseña con Notificaciones.

Diagrama de Clases

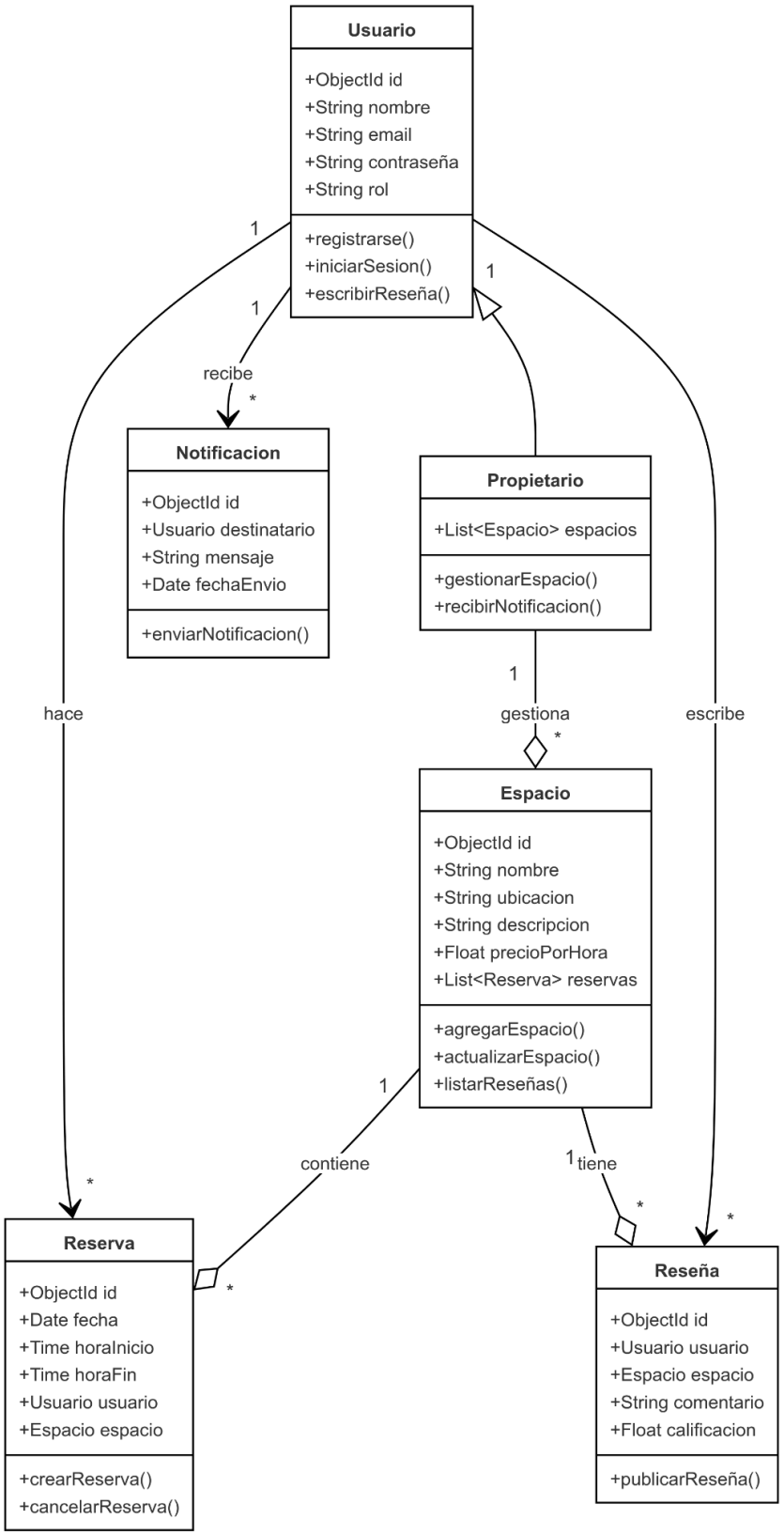


Figura 9: Diagrama de secuencia de Clases.

Una vez obtenido nuestro proyecto, crearemos nuestro Backend en el microservicio de Flask API.
Para conseguir que nuestro proyecto Flask funcione correctamente, deberemos crear un entorno virtual para su funcionamiento y ejecutar sus scripts activate.bat para instalar las dependencias que necesitaremos para su funcionamiento.

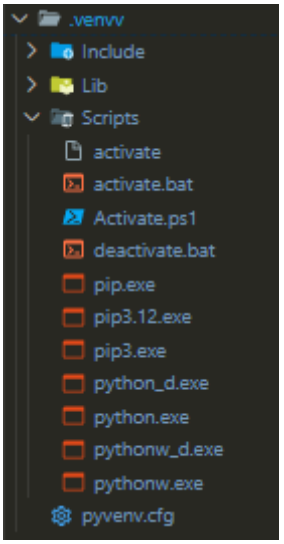


Figura 10: Estructura virtual python.

Ingresamos al entorno de Flask e instalamos flask y pymongo. Para verificar nuestras dependencias usaremos pip freeze para ver las librerías que ocuparemos.

```
(.venvv) C:\proyectos\PortalReservasEspacioTrab
ajoCompartido\Backend>pip freeze
blinker==1.8.2
click==8.1.7
colorama==0.4.6
dnspython==2.6.1
Flask==3.0.3
Flask-PyMongo==2.3.0
itsdangerous==2.2.0
Jinja2==3.1.4
MarkupSafe==2.1.5
pymongo==4.8.0
Werkzeug==3.0.4
```

Figura 11: Dependencias de librerías.

Configuraremos en MongoDB Compass la conexión por defecto, claro, ejecutamos mondongo primero para el funcionamiento.

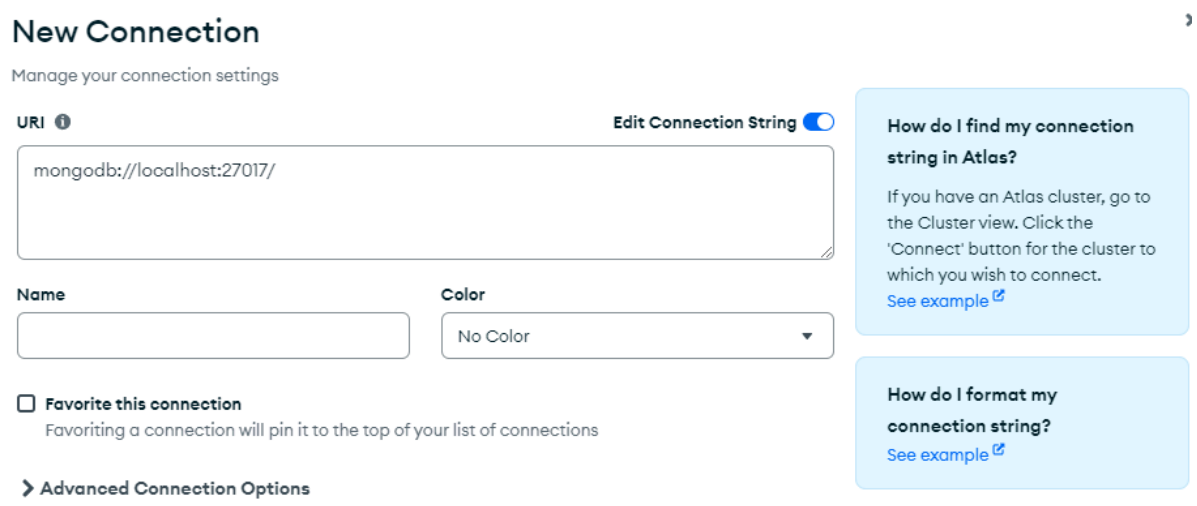


Figura 12: Comunicación Local MongoDB Compass.

Una vez conectados crearemos las colecciones de nuestra base de datos

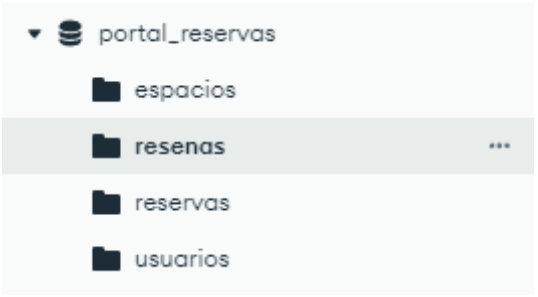


Figura 13: Colecciones de la base de datos.

Aprovechamos que el mongodb genera una base de datos dinámica según las peticiones y seguiremos con Flask y su código.

Nuestro proyecto, se divide en varios archivos, como:
config: almacenará las configuraciones de la aplicación, las que incluimos claves secretas y comunicación a MongoDB.
init: Aquí configuramos nuestra aplicación Flask.
routes: En está sección configuramos las rutas o Endpoints de la aplicación y sus funcionalidades. Además de las respuestas y peticiones.
model: Definimos las entidades mediante el uso de clases
run: este archivo se centrará en correr nuestra aplicación flask
requirements: Este archivo solo contiene datos generales sobre las dependencias que ocupamos en este proyecto.

Conexión MongoDB
Lo primero para conectarnos a mongodb en nuestro entorno visual studio code, es el usar de MONGO_URI, donde almacenará la conexión a la base de datos que usaremos y las colecciones que ésta posee.

```python
import os

class Config:
    SECRET_KEY=os.environ.get('SECRET_KEY') or 'clave'
    MONGO_URI = 'mongodb://localhost:27017/portal_reservas'
```

Figura 14: Comunicación Flask con MongoDB.

Para garantizar esta clase, modificamos el iniciador de Flask.

```python
from flask import Flask
from flask_pymongo import PyMongo
from config import Config

mongo = PyMongo()

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)

    mongo.init_app(app)

    from app import routes
    app.register_blueprint(routes.bp)

    return app
```

Figura 15: Configuración __init__.

Luego modificamos nuestros router, que manejara las rutas y lógica del negocio.

```python
from flask import Blueprint, request, jsonify
from app import mongo
from bson import ObjectId
from werkzeug.security import generate_password_hash, check_password_hash

bp = Blueprint('routes', __name__)

# Rutas de Usuario
@bp.route('/usuarios', methods=['POST'])
def create_usuario():
    data = request.json
    if mongo.db.usuarios.find_one({'email': data['email']}):
        return jsonify({"msg": "El usuario ya existe"}), 400

    hashed_password = generate_password_hash(data['contraseña'], method='sha256')
    data['contraseña'] = hashed_password
    mongo.db.usuarios.insert_one(data)
    return jsonify({"msg": "Usuario creado exitosamente"}), 201

@bp.route('/login', methods=['POST'])
def login():
    data = request.json
    usuario = mongo.db.usuarios.find_one({'email': data['email']})
    if not usuario or not check_password_hash(usuario['contraseña'], data['contraseña']):
        return jsonify({"msg": "Credenciales incorrectas"}), 401

    return jsonify({"msg": "Inicio de sesión exitoso", "usuario_id": str(usuario['_id'])})

@bp.route('/perfil/<usuario_id>', methods=['GET'])
def get_perfil(usuario_id):
    usuario = mongo.db.usuarios.find_one({'_id': ObjectId(usuario_id)}, {'contraseña': 0})
    if not usuario:
        return jsonify({"msg": "Usuario no encontrado"}), 404

    return jsonify(usuario)
```

Figura 16: Configuración endpoints.

Luego, de configurar lo necesario de nuestro código, será necesario validar el backend los endpoint y su base de datos.
Para conseguir esto, evaluaremos mediante el uso de postman las peticiones de usuario.

Creación de usuario

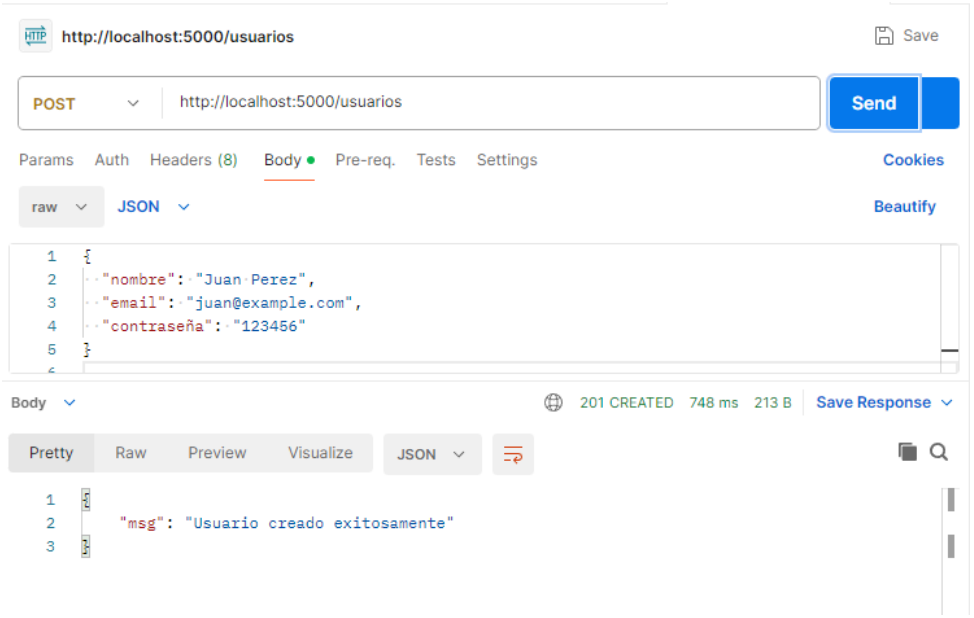


Figura 17: Creación de usuario.

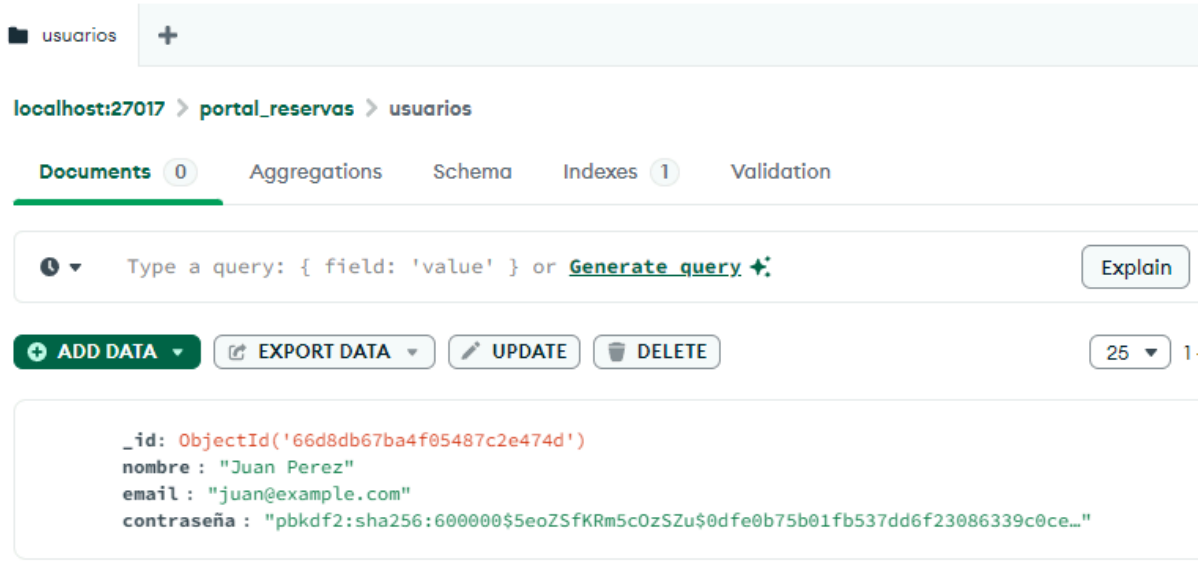


Figura 18: Generación de usuario.

Verificación de Login

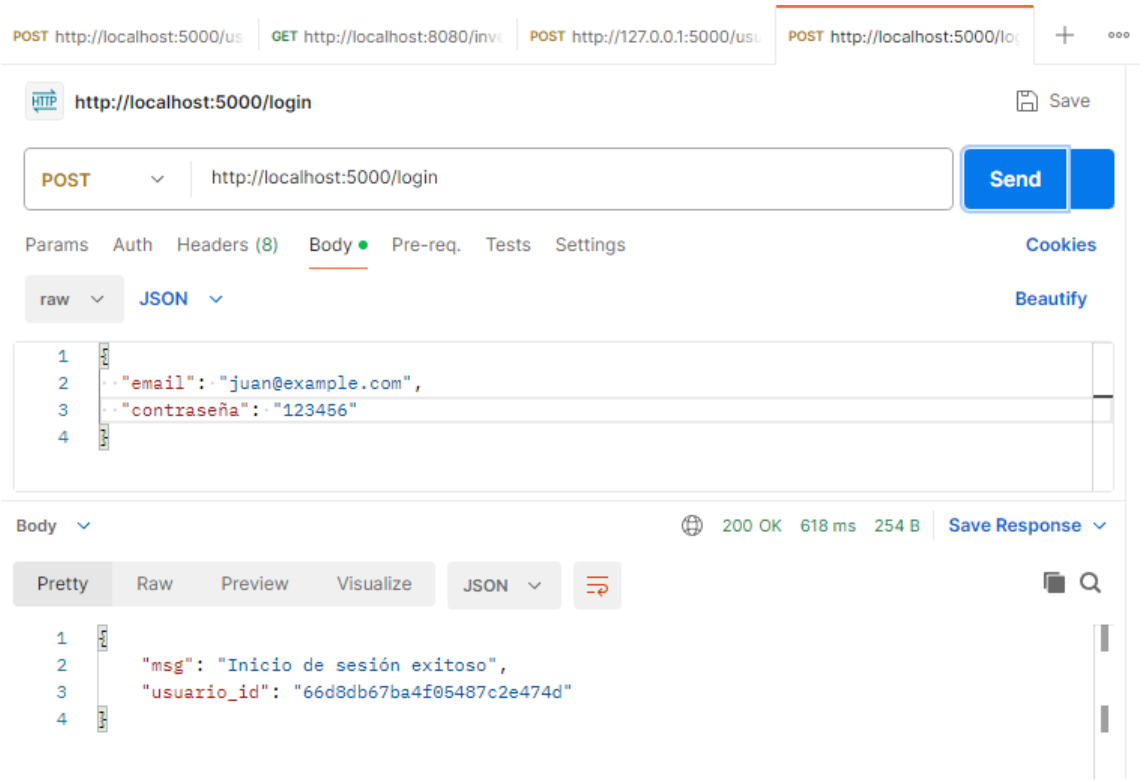


Figura 19: Verificador de usuarios.

Obtención de usuario por id

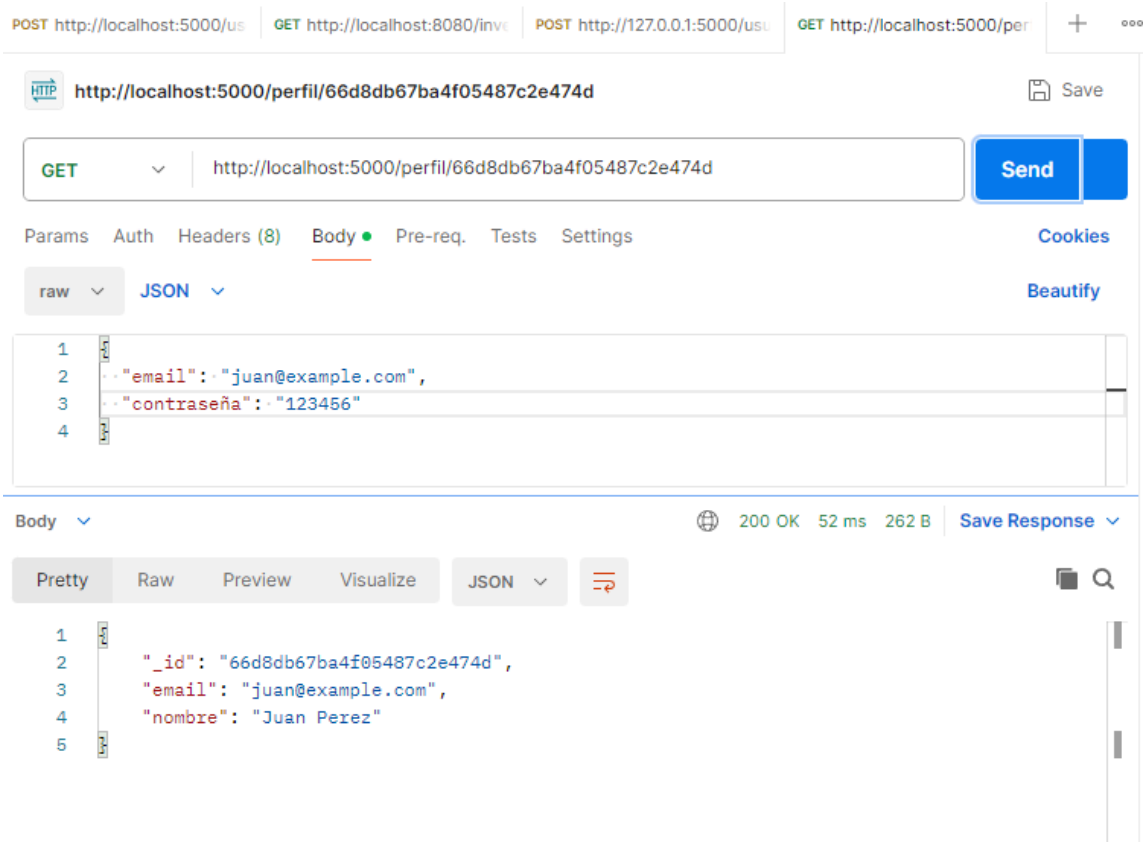


Figura 20: Obtener usuario por id.

Creación de Espacio

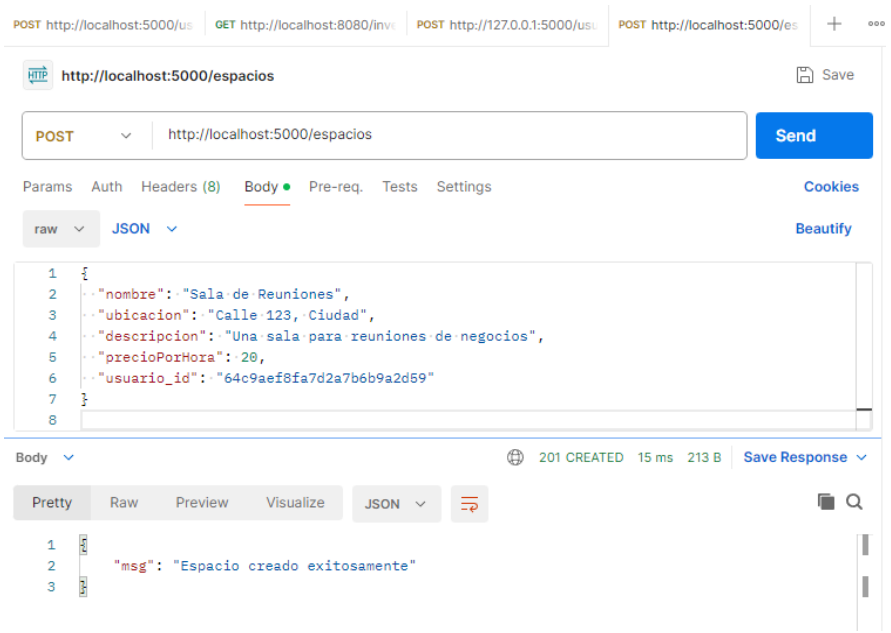


Figura 21: Creación de espacios.

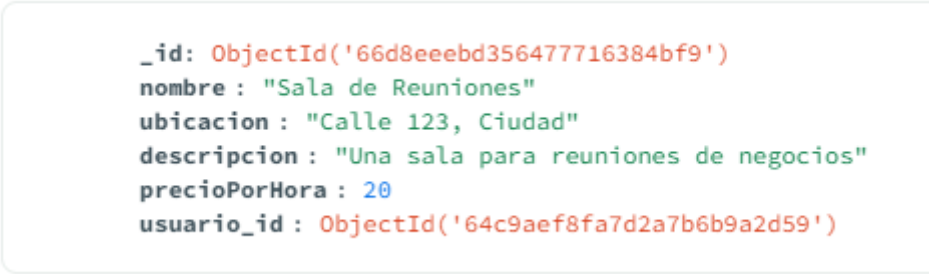


Figura 22: Creación del registro espacio.

Modificar Espacio

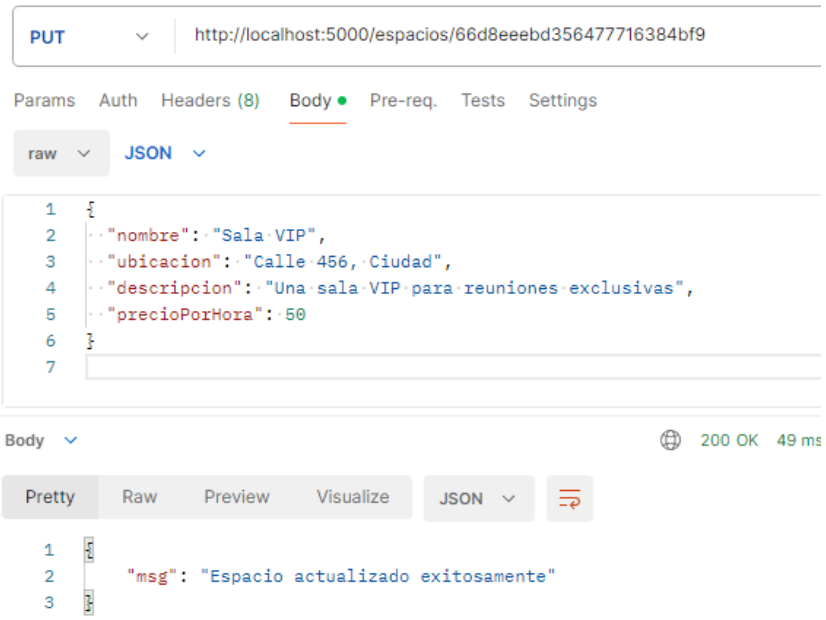
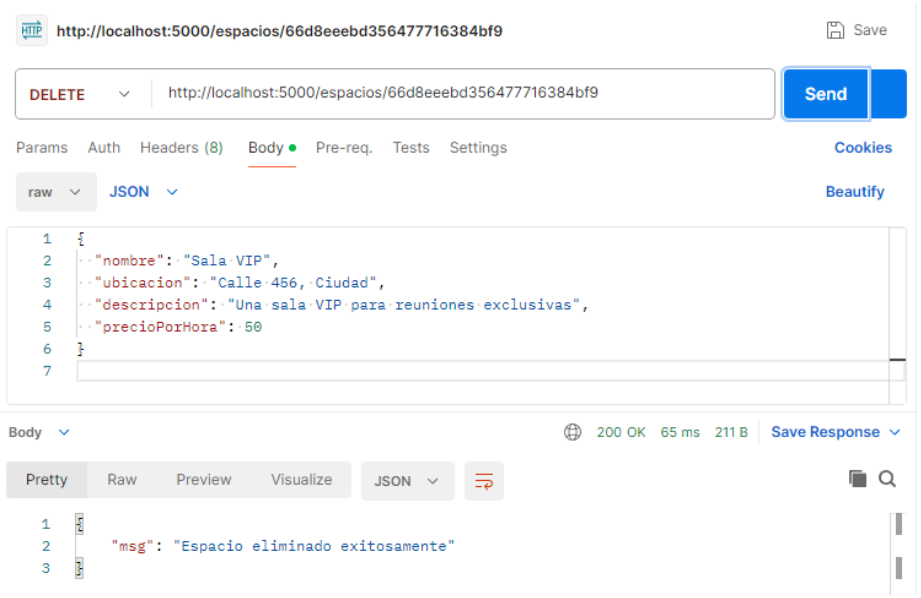


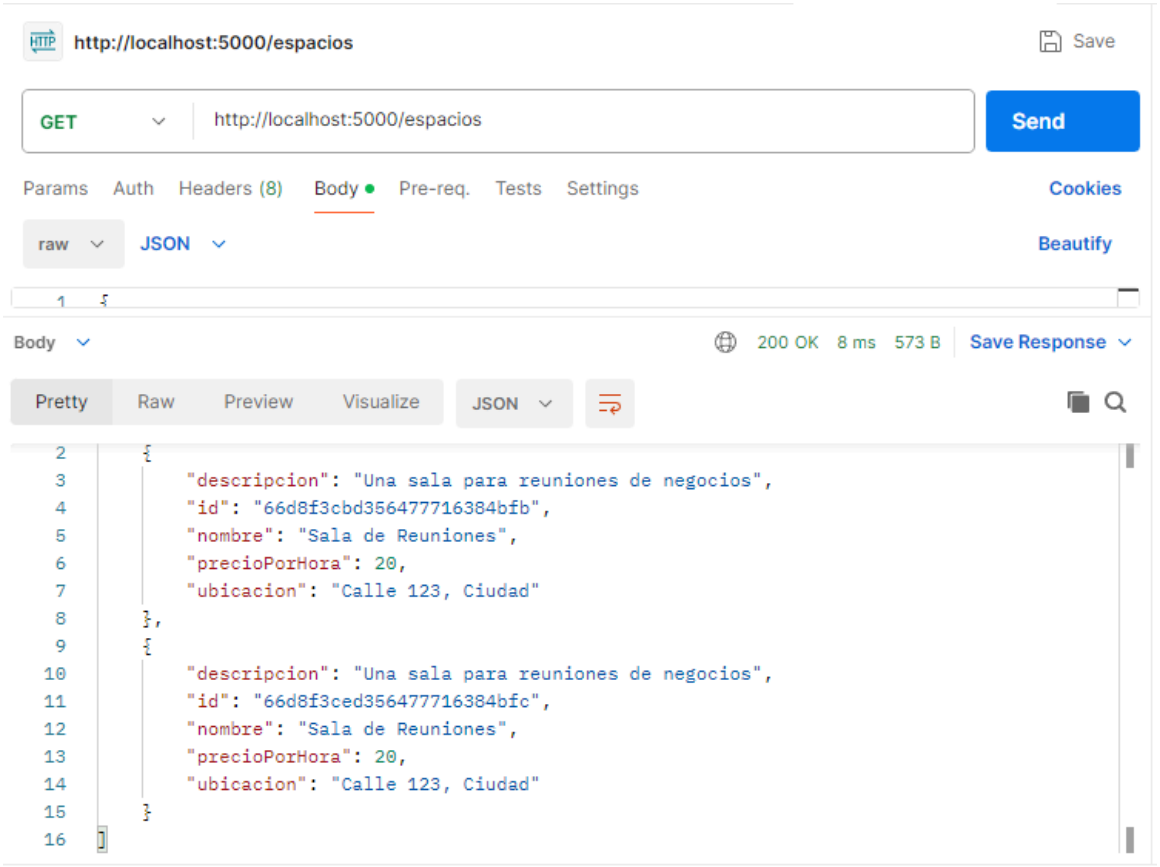
Figura 23: Actualización de espacio.

```
_id: ObjectId('66d8eeebd356477716384bf9')
nombre : "Sala VIP"
ubicacion : "Calle 456, Ciudad"
descripcion : "Una sala VIP para reuniones exclusivas"
precioPorHora : 50
usuario_id : ObjectId('64c9aef8fa7d2a7b6b9a2d59')
```

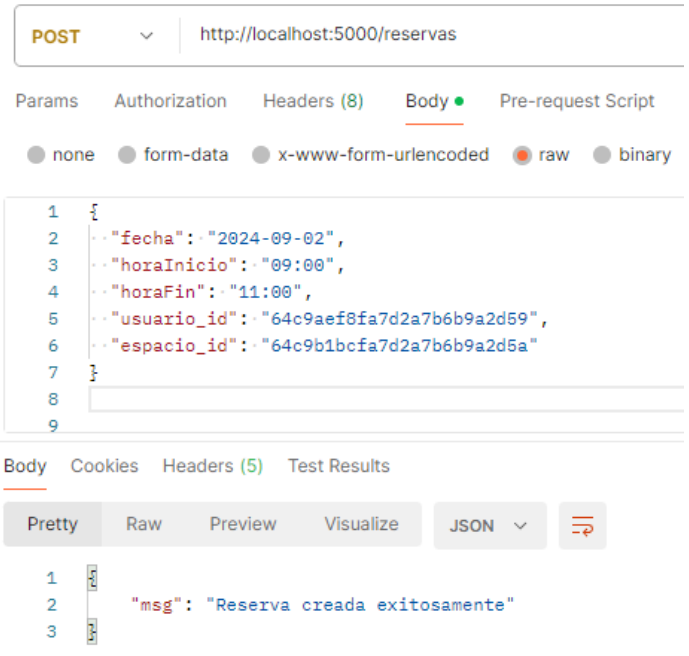
Eliminar Espacio



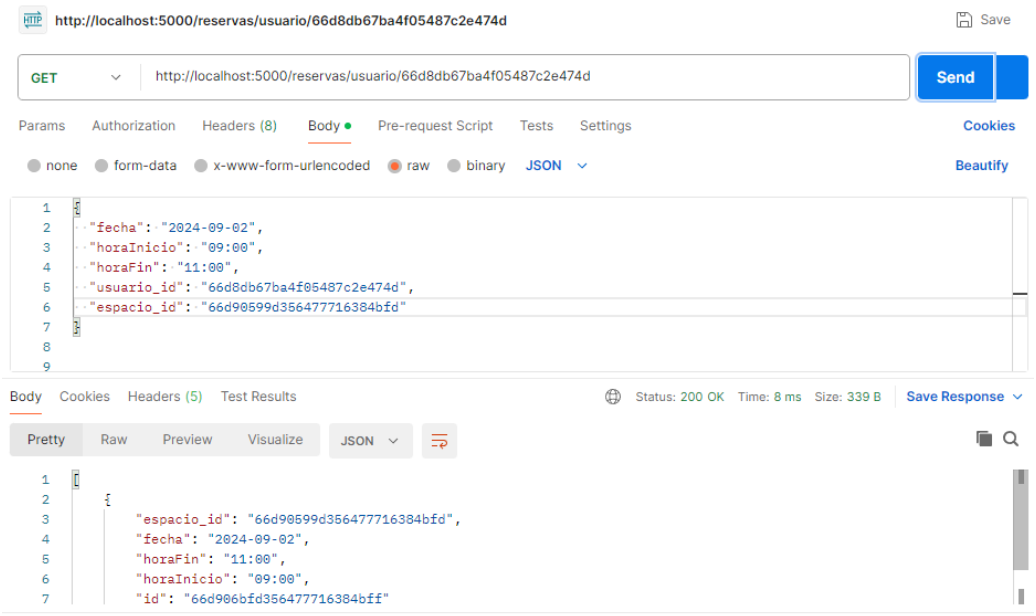
Despliegue de los espacios adquiridos



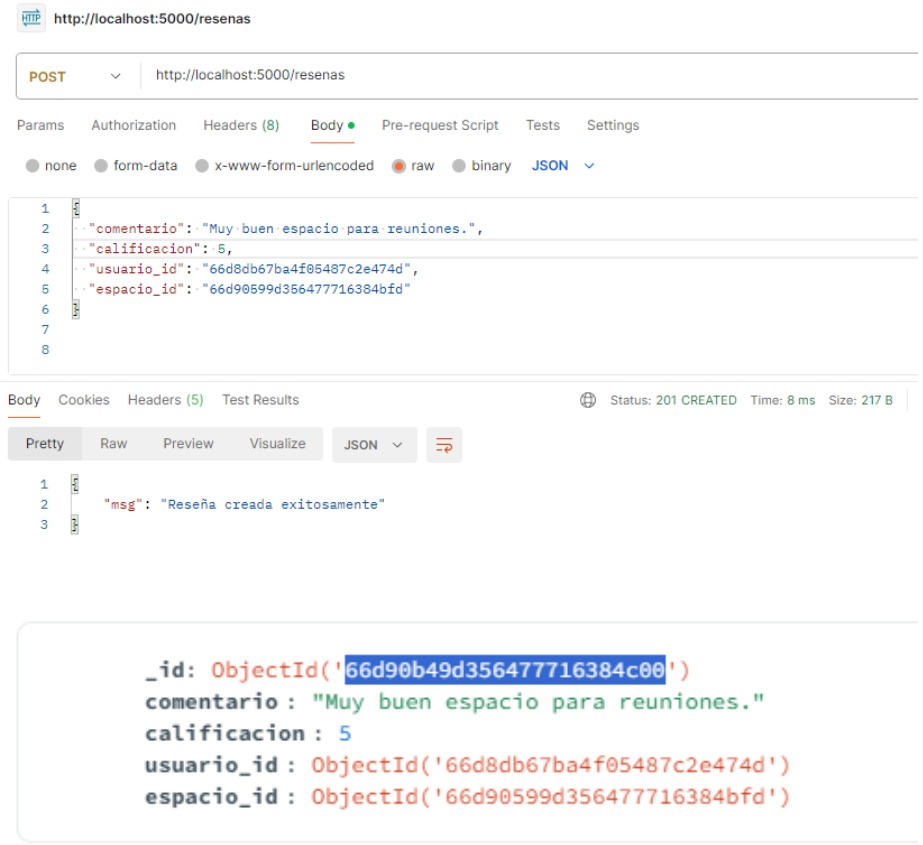
Creación de reservas



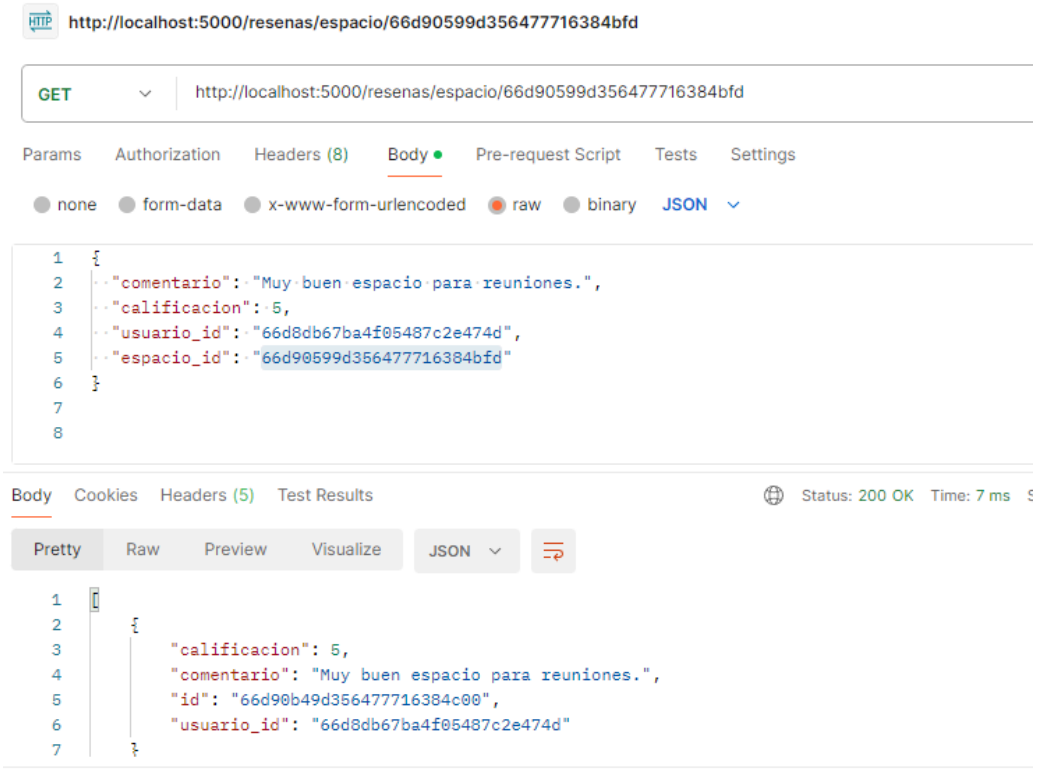
Reservación por usuario



Creación de Reseñas



Obtener Reseña del Espacio



Así comprobamos el funcionamiento de nuestros Endpoints y APIs. Solo nos faltaría validarlos mediante una interfaces para que el usuario lo consume.

Mejoras Futuras

1. Para mejorar el proyecto, en el futuro se propone subir las APIs a un servidor en nube para mejorar el rendimiento y el consumo sea más sencillo.
2. Se propone que el proyecto separe los routers a futuro, para trabajarlos por servicio y solo se ocupen los endpoints de momento.
3. Se propone desarrollar microservicios para trabajar mejor los servicios con el proyecto principal backend.