МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА



**КУРСОВА РОБОТА**

з дисципліни “ПРИКЛАДНЕ ПРОГРАМУВАННЯ”

на тему: « Розробка серверної частини

системи для бронювання аудиторій»

Студента групи КН-214

спеціальності 122 “Комп’ютерні науки”

Ляшеник О.А.

Керівник

к.т.н., доц. Шиманський В. М.

Кількість балів: \_\_\_\_\_\_\_\_\_ Оцінка:\_\_\_\_\_\_\_\_\_\_\_

Члени комісії

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Львів – 2022

ЗАВДАННЯ

на курсова роботу з дисципліни “Прикладне програмування”

студента групи КН-214 Ляшеник Остапа

**ТЕМА:**  « **Розробка серверної частини**

**системи для бронювання аудиторій»**

**ЗМІСТ ЗАВДАННЯ**

| **№ з/п** | **Зміст завдання** | **Примітка** |
| --- | --- | --- |
| **1** | Постановка завдання |  |
| **2** | Реалізація завдання |  |
| **2.1** | Модель даних |  |
| **2.2** | Опис REST API |  |
| **2.3** | Опис ORM |  |
| **2.4** | Опис бізнес-логіки |  |
| **2.5** | Опис бізнес-логіки |  |
| **2.6** | Опис тестів |  |
| **3** | Аналіз результатів та інструкція користувача |  |
| **4** | Висновки |  |
| **5** | Список літератури |  |
| **6** | Додатки |  |

Завдання прийнято до виконання: Ляшеник О.А. 25.11.2022 р

Керівник роботи: Виклюк Я.І.

1. Постановка завдання

Завданням курсової роботи було розроблення серверної частини сервісу для бронювання аудиторій, де користувачі мають можливість бронювати від 1 год до 5 днів, адмін створює та видаляє аудиторії,а менеджер керує інформацією про них.

Для виконання цієї курсової я слідкував такому плану:

1. Налаштування сервера;
2. Проектування REST API;
3. Налаштування ORM;
4. Реалізація API;
5. Авторизація;
6. Тестування.

2. Реалізація завдання

2.1. Модель даних

Для реалізації ER-діаграми я використав програму MySQL Workbench через зручний і одночасно простий її функціонал. Наша діаграми буде мати наступний вигляд:

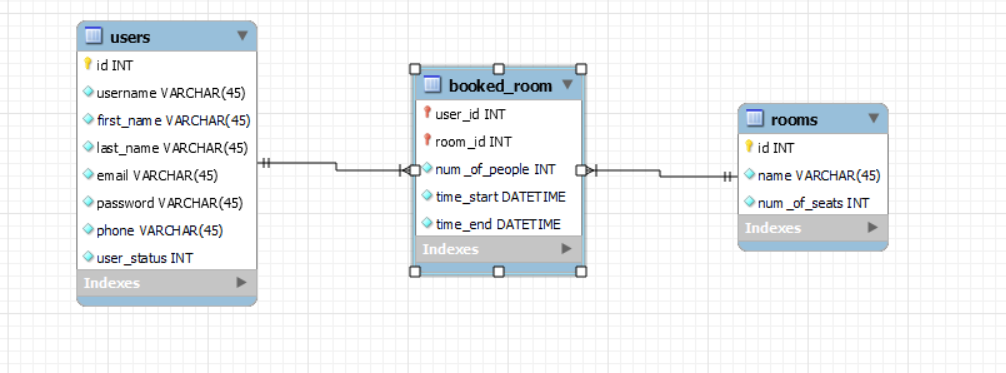


рис.1 ER-діаграма

На рис.1 ми бачимо що наша база даних буде складатись з 3 табличок, пізніше буде 4,але це при наступному етапі. А саме:

1. Табличка “user”
2. Табличка “rooms”
3. Табличка “booked\_room”

Зв’язок між користувачами і аудиторією буде реалізований через 3 табличку.

models.py:

import os

from sqlalchemy import \*

from sqlalchemy.orm import declarative\_base, sessionmaker, relationship

engine = create\_engine("postgresql://postgres:postgres@localhost:5432/booking")

Session = sessionmaker(bind=engine)

BaseModel = declarative\_base()

class Users(BaseModel):

\_\_tablename\_\_ = "users"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

username = Column(String)

first\_name = Column(String)

last\_name = Column(String)

email = Column(String)

password = Column(String)

phone = Column(String)

user\_status = Column(Integer)

class Rooms(BaseModel):

\_\_tablename\_\_ = "rooms"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

name = Column(String)

num\_of\_seats = Column(Integer)

class booked\_room(BaseModel):

\_\_tablename\_\_ = "booked\_room"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

room\_id = Column(Integer, ForeignKey('rooms.id', ondelete="CASCADE"))

user\_id = Column(Integer, ForeignKey('users.id', ondelete="CASCADE"))

num\_of\_people = Column(Integer)

time\_start = Column(TIMESTAMP)

time\_end = Column(TIMESTAMP)

userToBook = relationship(Users, foreign\_keys=[user\_id], backref="user\_id", lazy="joined", cascade="all, delete")

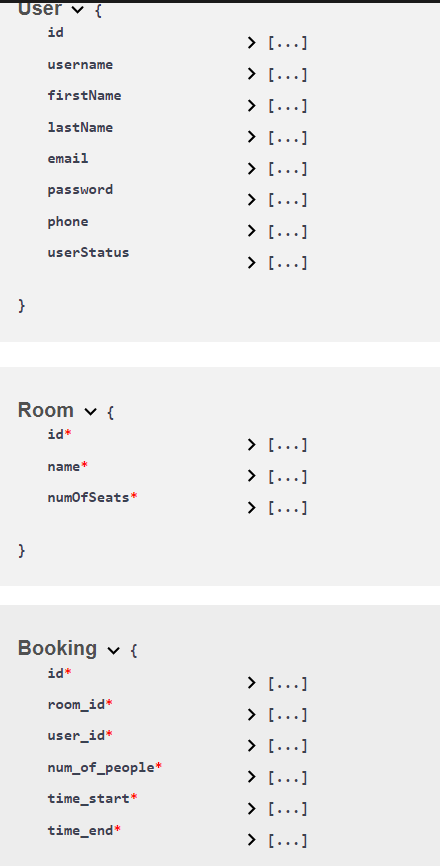
roomToBook = relationship(Rooms, foreign\_keys=[room\_id], backref="room\_id", lazy="joined", cascade="all, delete")

2.2. Опис REST API

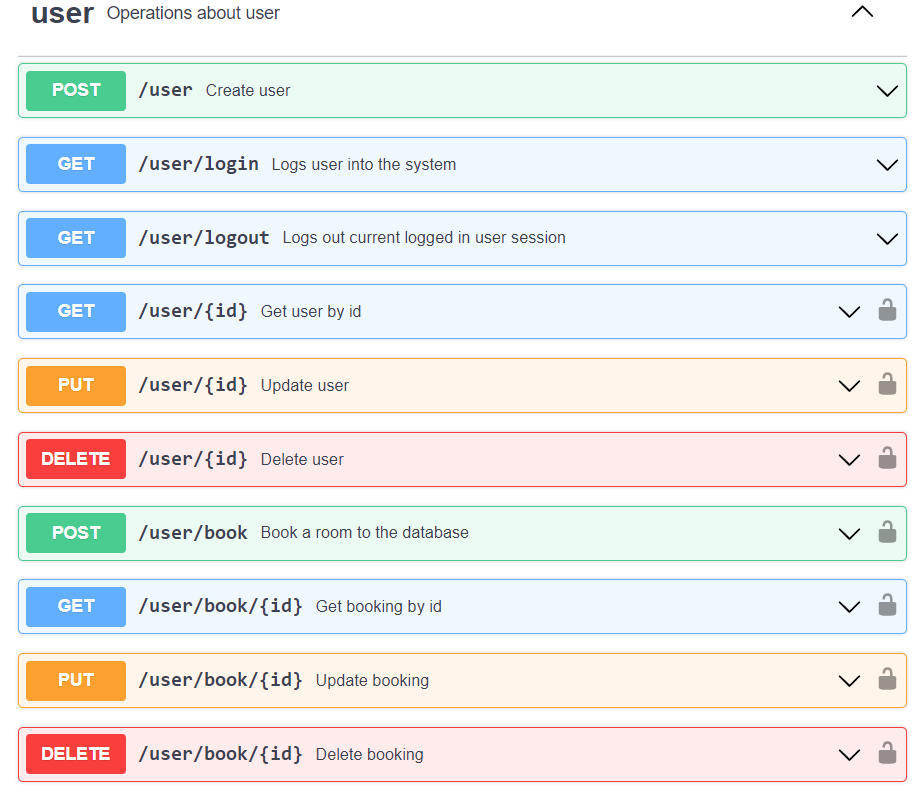
Наша API мати наступні сутності:

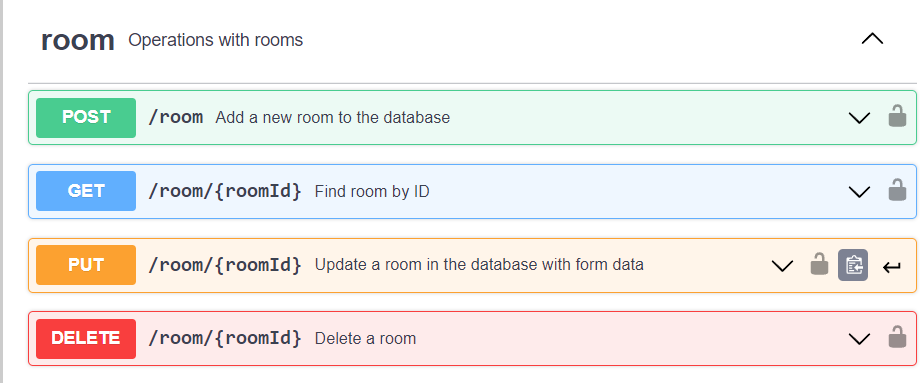


З такими властивостями:



Методи для нашого REST API:





/user:

Метод, який додає до бази нового користувача. Можливі 2 відповіді сервера: 400- коли некоректно вказані дані, а також 200- успішна операція, повертає інформацію про користувача.

/user/login:

Метод для авторизації користувача та повертає 200 при успішній операції, 404 - коли не знайдено користувача, 401 - коли не вийшло верифікувати його та 400 при некоректних даних.

/user/logout:

Метод, який дозволяє вийти користувачу з системи та повертає код 200.

/user/id:

1. Метод, який шукає користувача по id та повертає його з статус кодом 200 або 404 - коли не знайдено користувача .
2. Метод, який видаляє з бази даних користувача за вказаним ідентифікатором. Можливі 2 відповіді сервера: 404 - коли не знайдено користувача , а також 204 - успішна операція.
3. Метод, який оновлює в базі даних користувача за вказаним ідентифікатором. Можливі 2 відповіді сервера: 400- коли некоректно вказані дані, а також 200- успішна операція, повертає інформацію про користувача .

/user/book:

Метод, який робить бронювання аудиторії та повертає 200 при успішній операції та 404, якщо не знайдено кімнат,405 - якщо некоректно введені дані.

/user/book/id:

1. Метод, який шукає бронювання по id та повертає його з статус кодом 200 або 404 - коли не знайдено бронювання.
2. Метод, який видаляє з бази даних бронювання за вказаним ідентифікатором. Можливі 2 відповіді сервера: 404 - коли не знайдено бронювання, а також 204 - успішна операція.
3. Метод, який оновлює в базі даних бронювання за вказаним ідентифікатором. Можливі 2 відповіді сервера: 400- коли некоректно вказані дані, а також 200- успішна операція, повертає інформацію про бронювання.

/room:

Метод, який додає до бази інформацію про аудиторію. Можливі 2 відповіді сервера: 400- коли некоректно вказані дані, а також 200- успішна операція, повертає інформацію про аудиторію.

/room/id:

1. Метод, який шукає зал по id та повертає його з статус кодом 200 або 404 - коли не знайдено аудиторію.
2. Метод, який видаляє з бази даних аудиторію за вказаним ідентифікатором. Можливі 2 відповіді сервера: 404 - коли не знайдено аудиторію, а також 204 - успішна операція.
3. Метод, який оновлює в базі даних аудиторію за вказаним ідентифікатором. Можливі 2 відповіді сервера: 400- коли некоректно вказані дані, а також 200- успішна операція, повертає інформацію про залаудиторію.

2.3. Опис ORM

Для початку створимо файл з моделями з використанням SQLAlchemy.

models.py

import os

from sqlalchemy import \*

from sqlalchemy.orm import declarative\_base, sessionmaker, relationship

engine = create\_engine("postgresql://postgres:postgres@localhost:5432/booking")

Session = sessionmaker(bind=engine)

BaseModel = declarative\_base()

class Users(BaseModel):

\_\_tablename\_\_ = "users"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

username = Column(String)

first\_name = Column(String)

last\_name = Column(String)

email = Column(String)

password = Column(String)

phone = Column(String)

user\_status = Column(Integer)

class Rooms(BaseModel):

\_\_tablename\_\_ = "rooms"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

name = Column(String)

num\_of\_seats = Column(Integer)

class booked\_room(BaseModel):

\_\_tablename\_\_ = "booked\_room"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

room\_id = Column(Integer, ForeignKey('rooms.id', ondelete="CASCADE"))

user\_id = Column(Integer, ForeignKey('users.id', ondelete="CASCADE"))

num\_of\_people = Column(Integer)

time\_start = Column(TIMESTAMP)

time\_end = Column(TIMESTAMP)

userToBook = relationship(Users, foreign\_keys=[user\_id], backref="user\_id", lazy="joined", cascade="all, delete")

roomToBook = relationship(Rooms, foreign\_keys=[room\_id], backref="room\_id", lazy="joined", cascade="all, delete")

Використовуємо команду alembic init migration для підключення alembic.

В новоутворених файлах змінюємо підключення до нашої БД та до нашого файлу з моделями.

Створюємо міграцію за допомогою alembic revision --autogenerate. Та

застосовуємо її до бази, ввівши alembic upgrade heads.

2.4. Опис бізнес-логіки

User методи:

* create\_user – додаємо запис у таблицю юзер.
* get\_user – отримуємо з бази користувача з заданим id
* update\_user – переріяємо чи користувач себе оновлює,а тоді оновлюємо в базі інформацію користувача з заданим id, якщо він є авторизованим користувачем.
* delete\_user – переріяємо чи користувач себе видаляє,а тоді видаляємо користувача з БД.
* login – авторизація користувача.
* logout – вихід з системи.
* book - робимо бронювання для авторизованого користувача,якщо це не менеджер чи адмін.
* update\_book - оновлюємо інформацію про бронювання.
* delete\_book - скасовуємо бронювання.

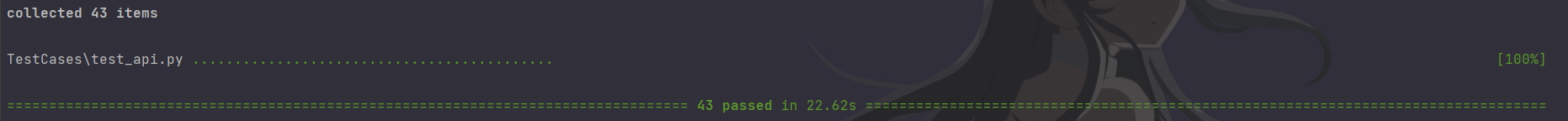
Room методи:

* create\_room - перевіряємо чи запит виконаний адміном, перевіряємо валідність кількості місць, а тоді вже додаємо до бази даних аудиторію.
* get\_room - отримуємо інформацію про аудиторію.
* delete\_room - перевіряємо чи адмін надіслав запит, а тоді видаляємо аудиторію, всі зв’язки видаляються на рівні БД.
* update\_room - перевіряємо чи запит виконаний адміном або менеджером, на валідність даних, а тоді вже оновлюємо значення аудиторії в БД.

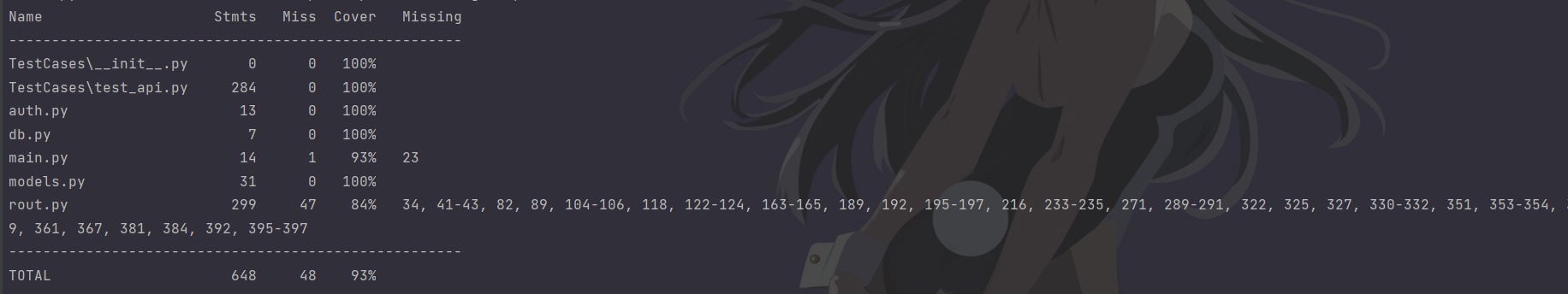
2.5. Опис тестів

Для Unit тестування я використав пакет pytest, а для показу покриття коду тестів пакет coverage.

Результат тестів:



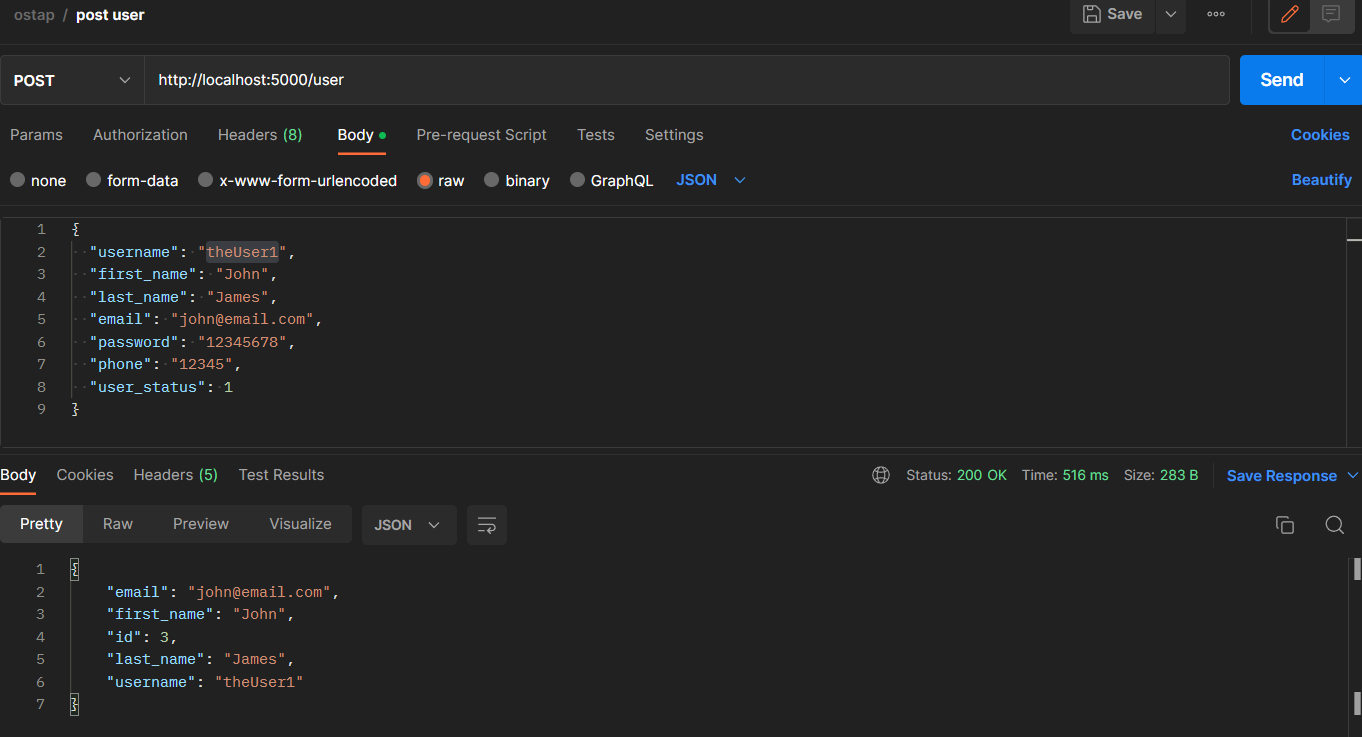
Покриття:

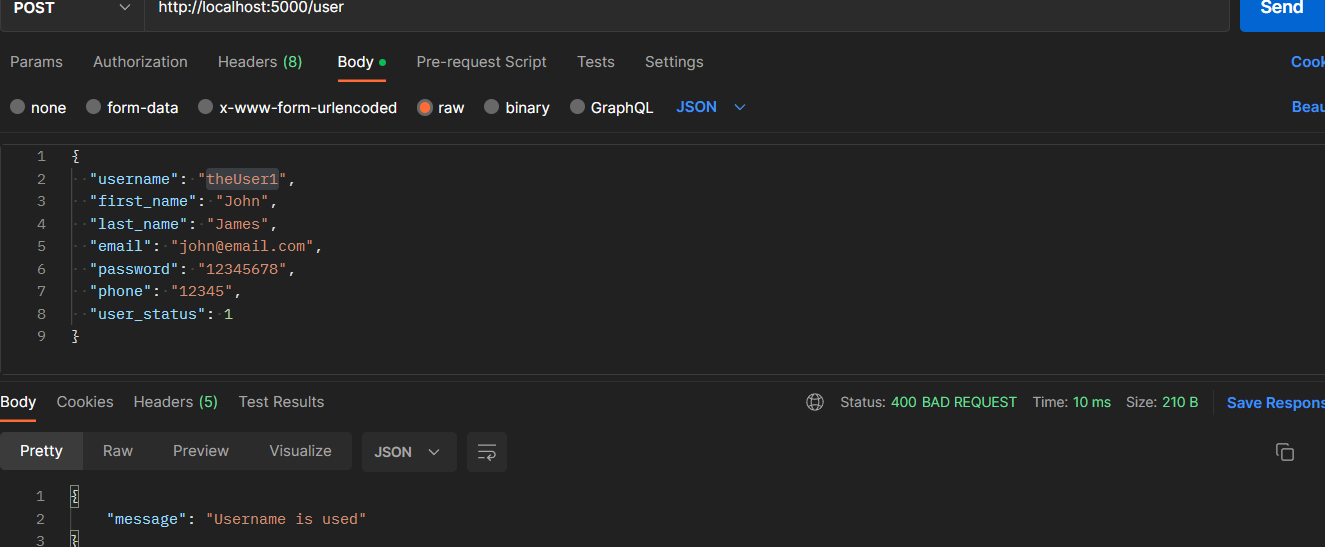


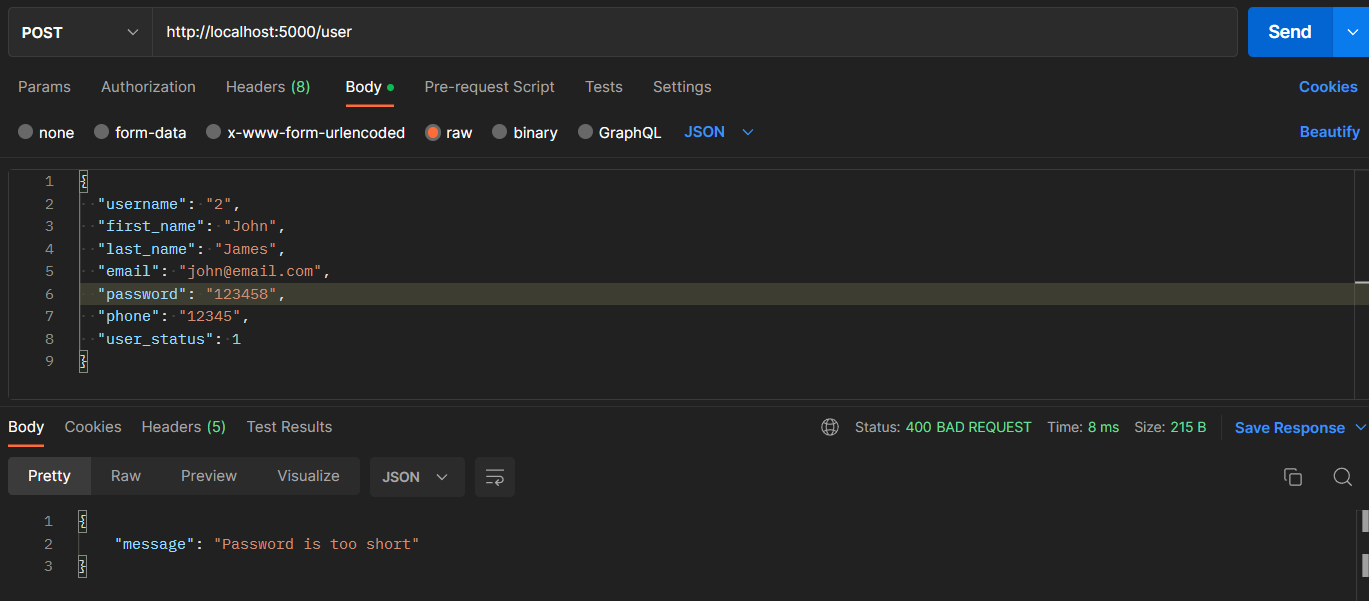
93% - результат нашого покриття.

3. Аналіз результатів та інструкція користувача

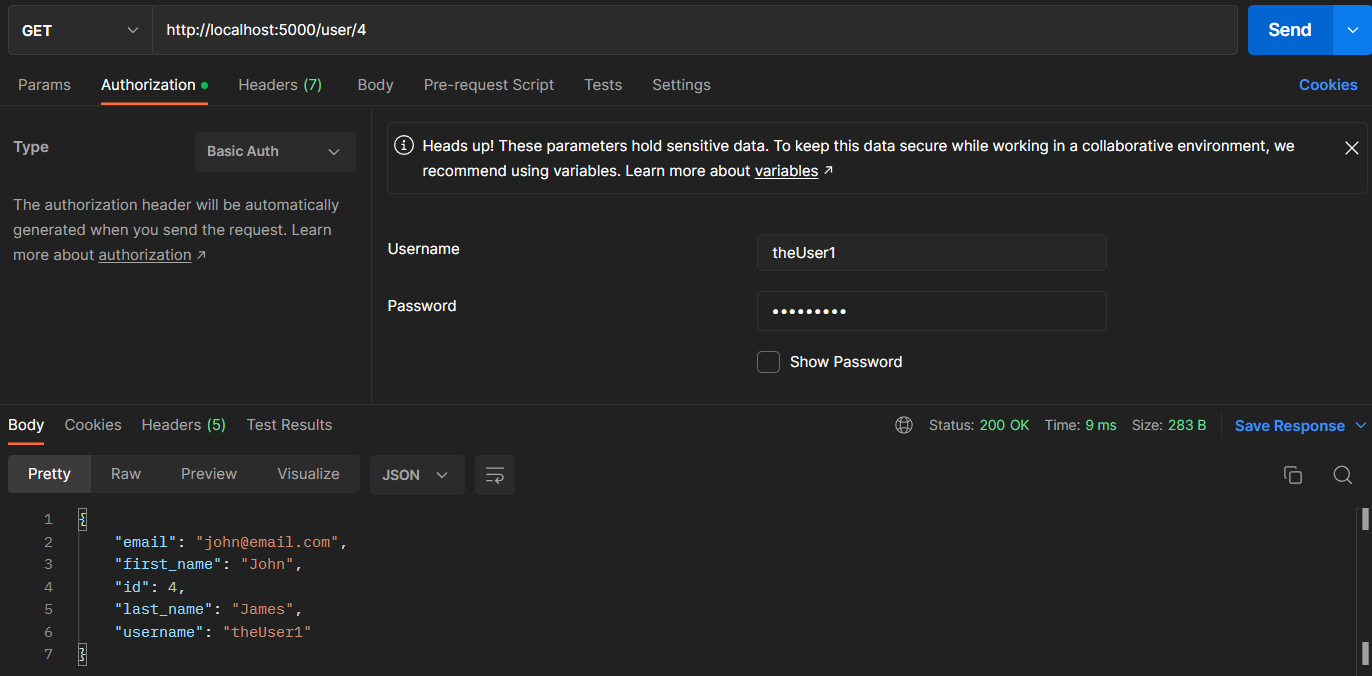
Створюємо користувача:

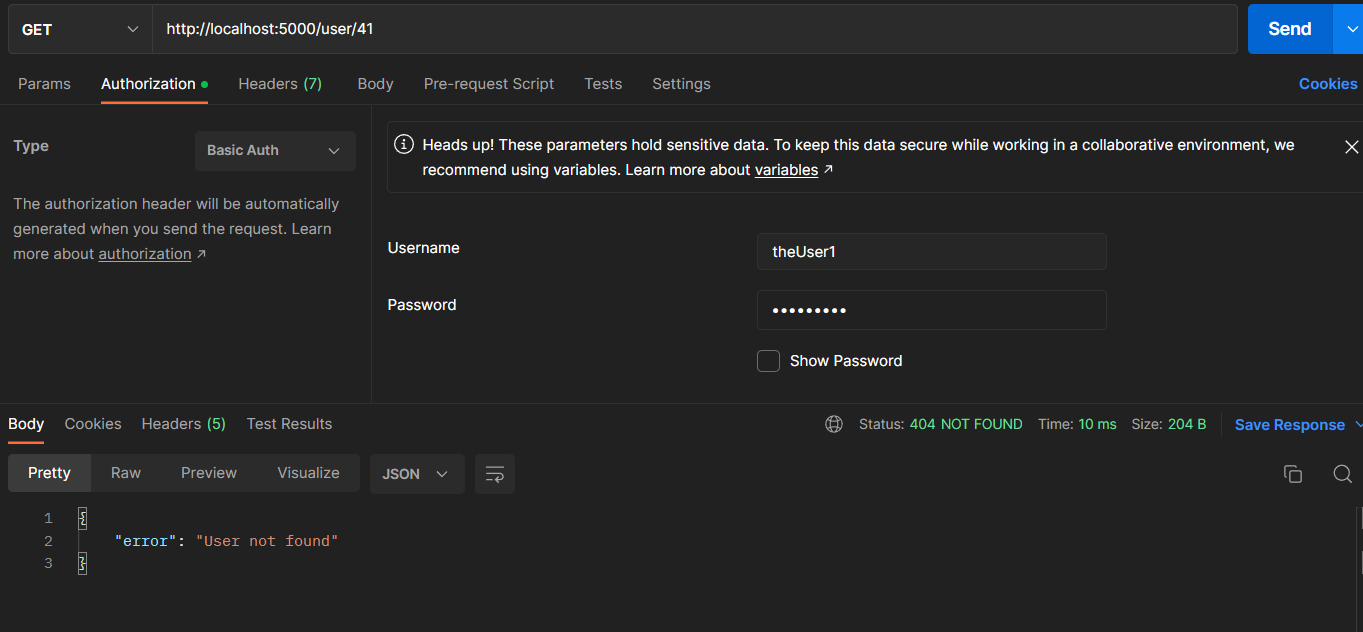




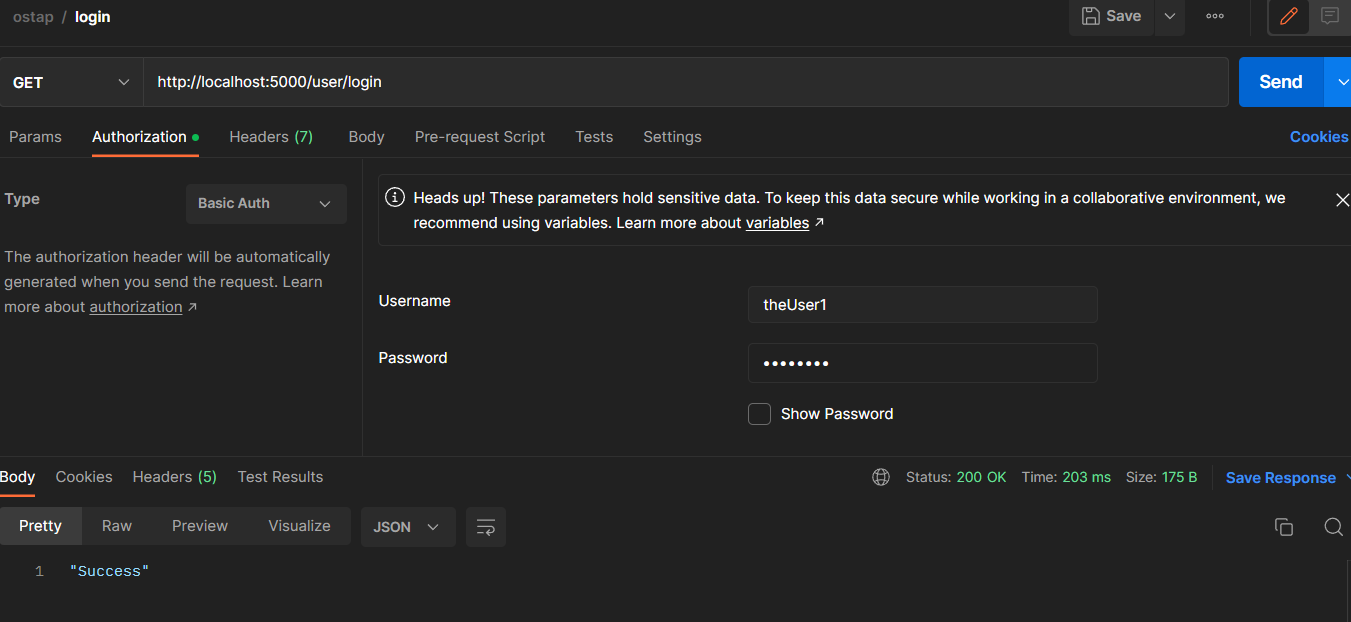


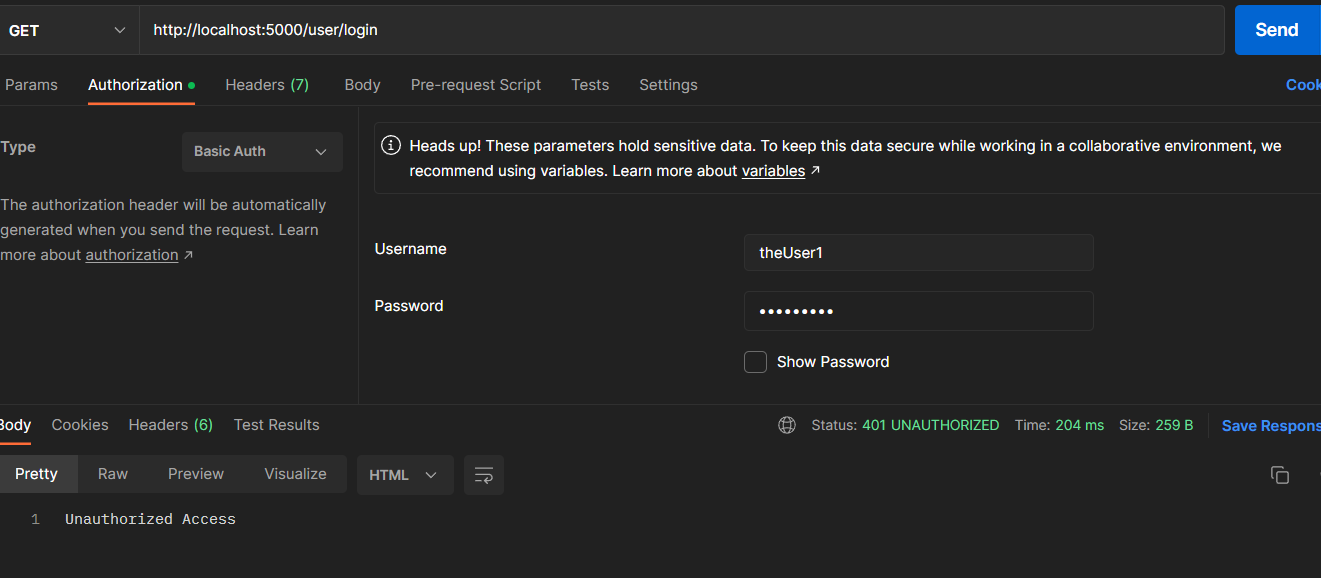
Отримуємо інформацію про користувача:



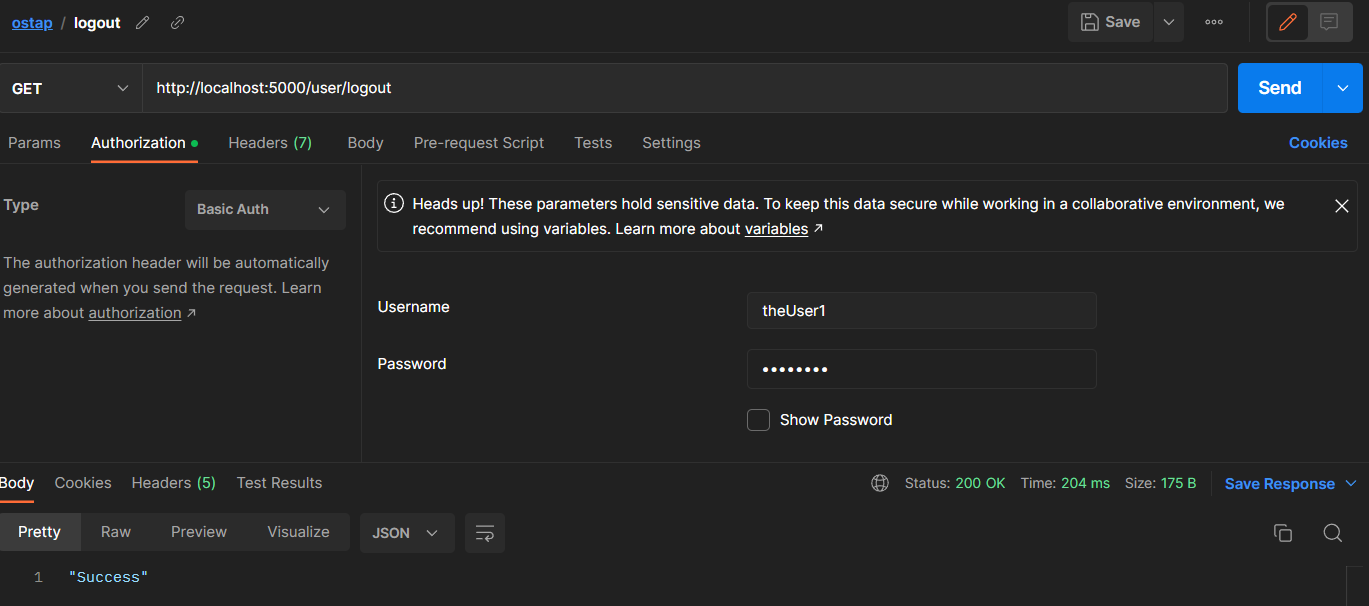


Авторизація:

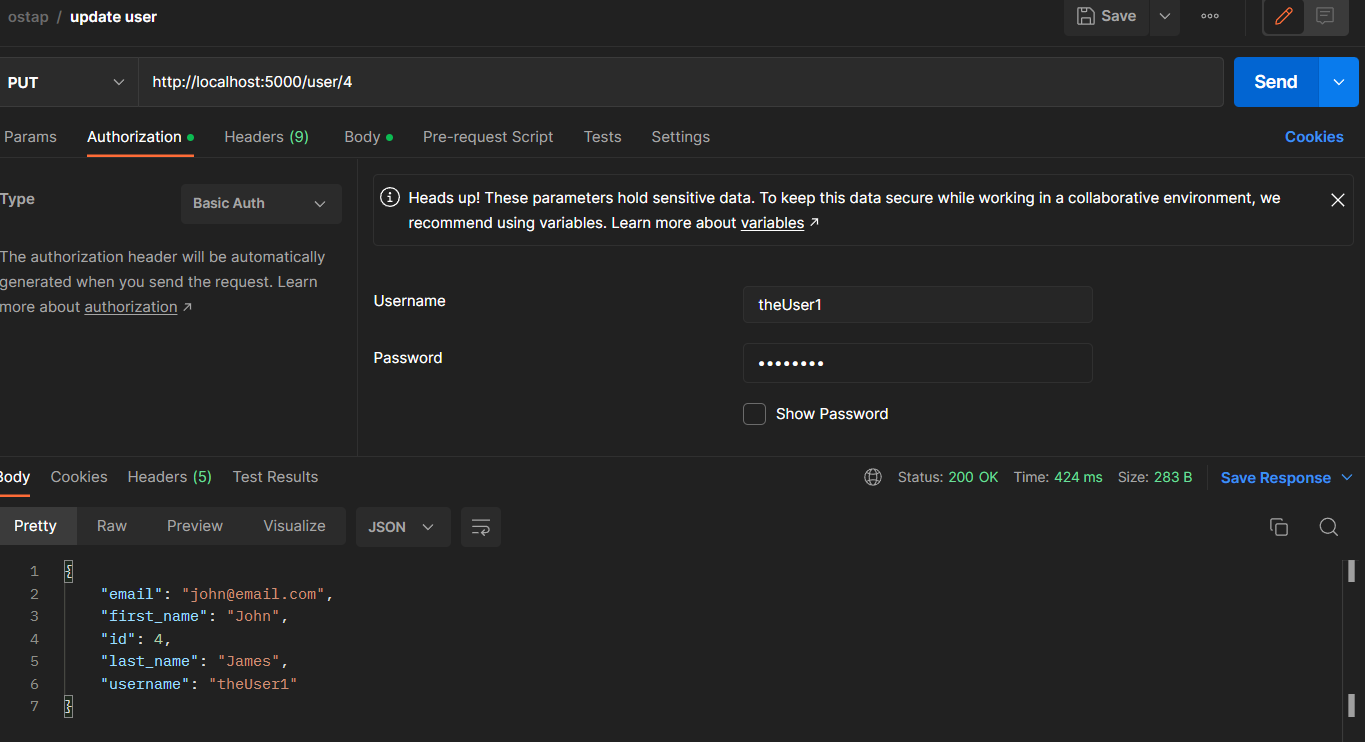


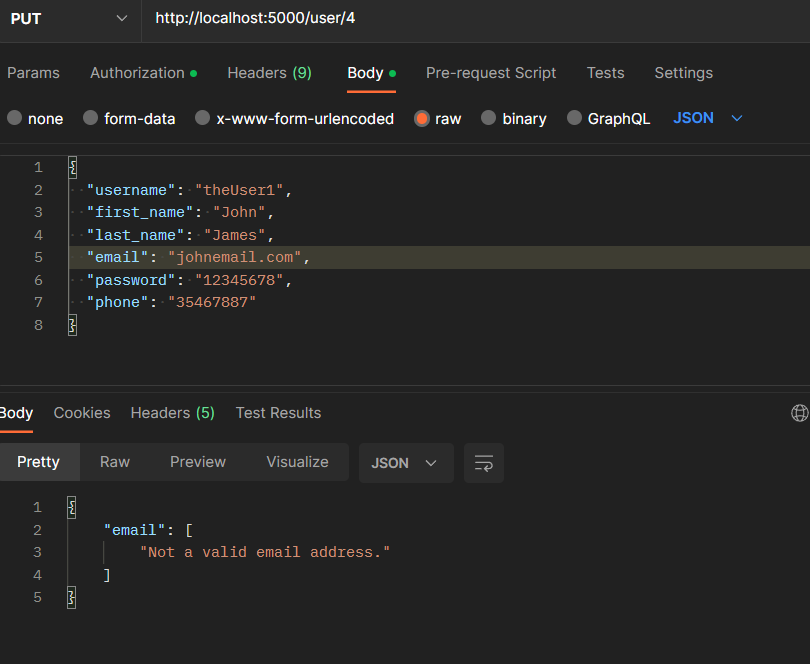


Вихід з системи:

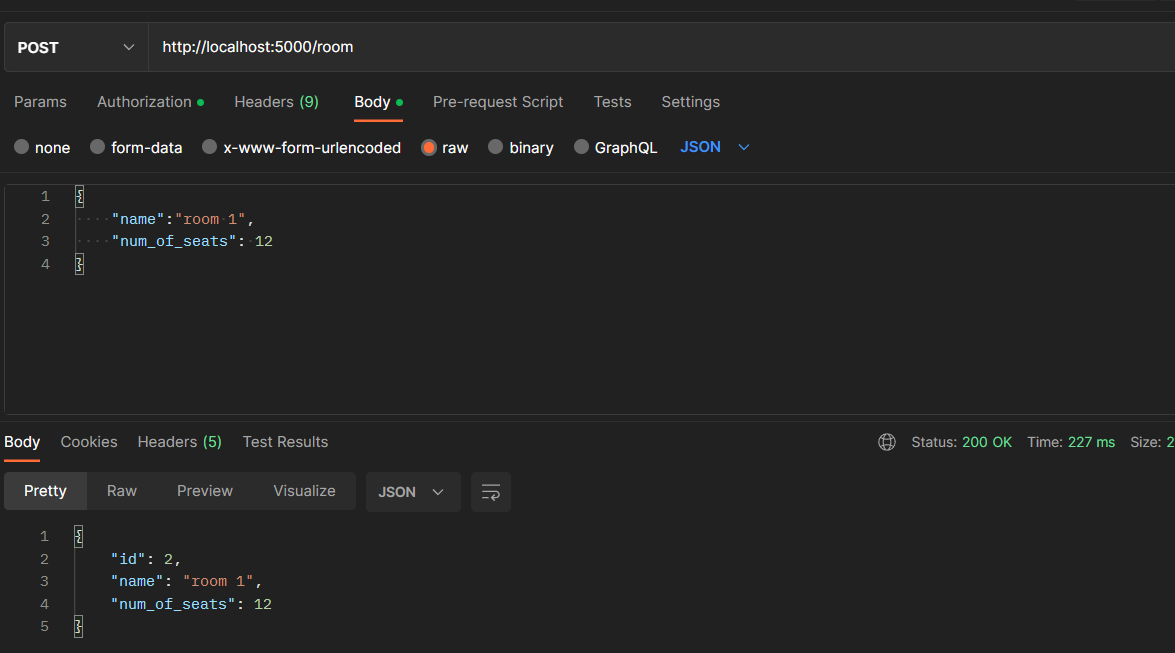


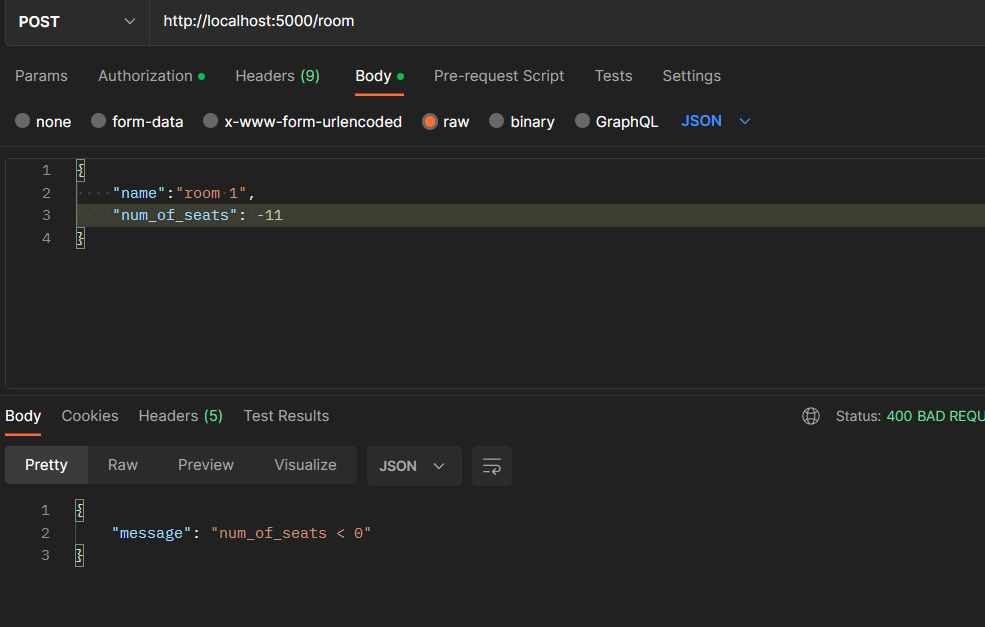
Оновлення користувача:



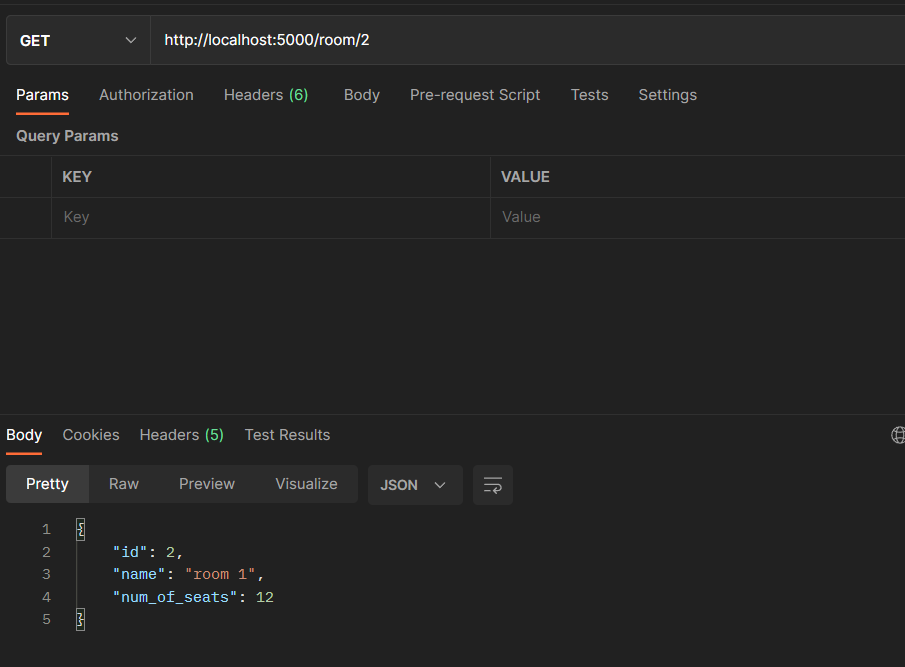


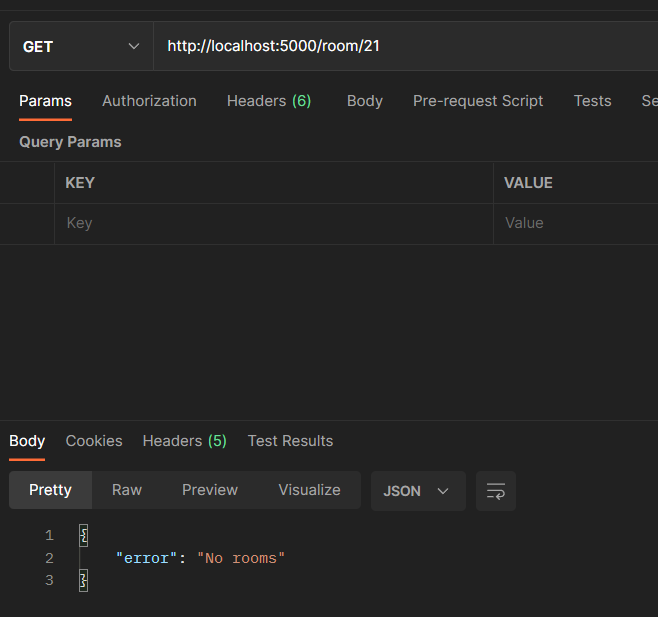
Створення аудиторії:



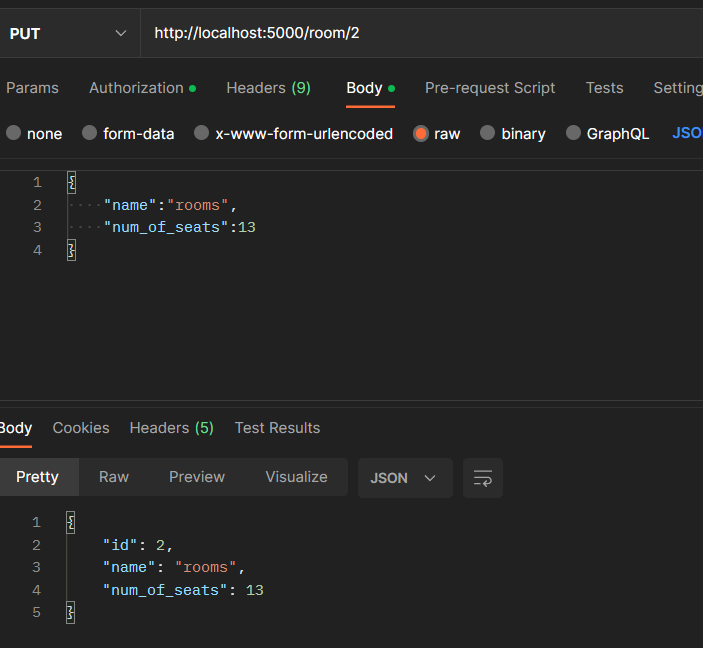


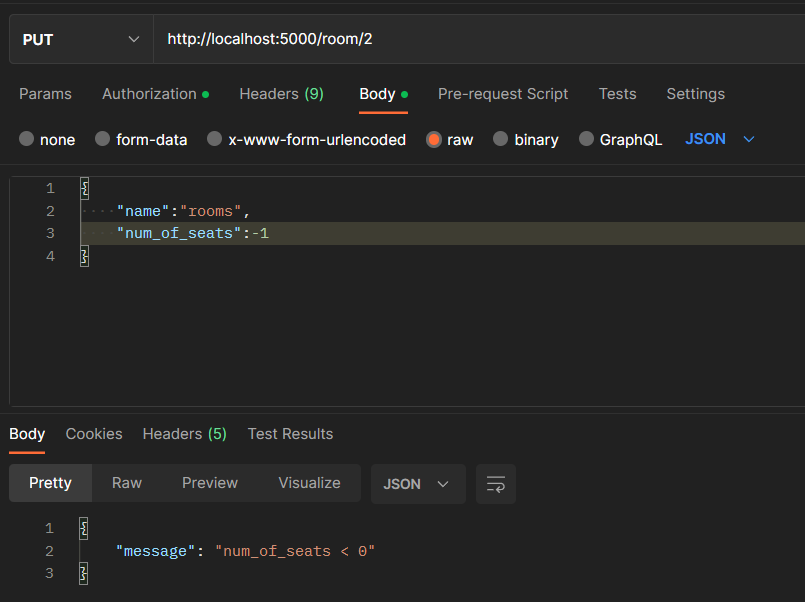
Отримання інформації про аудиторію:



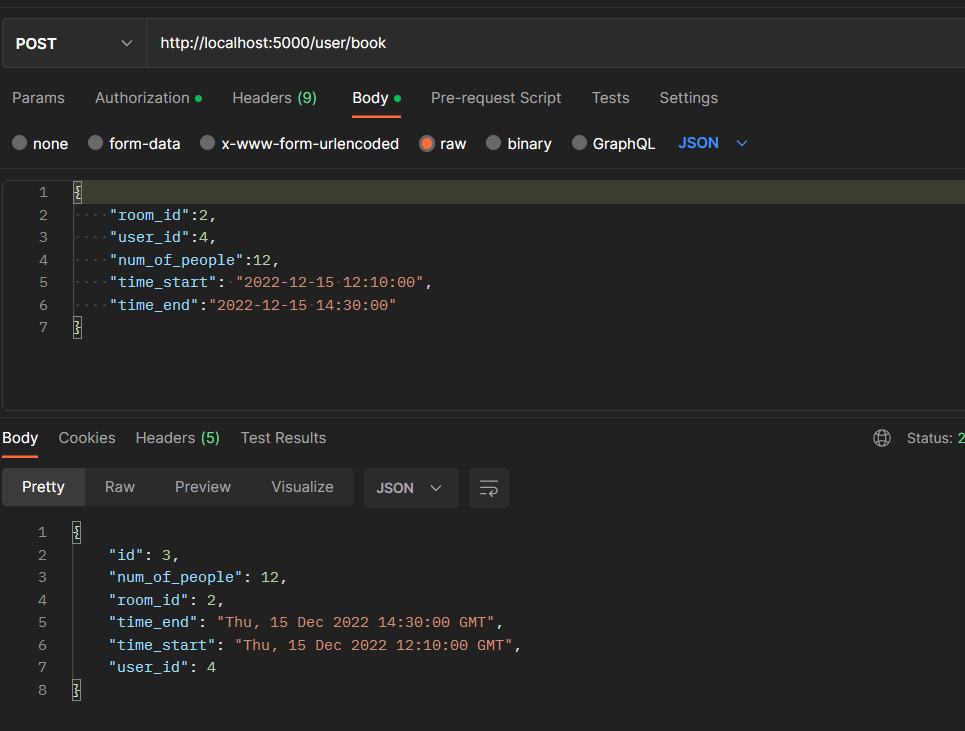


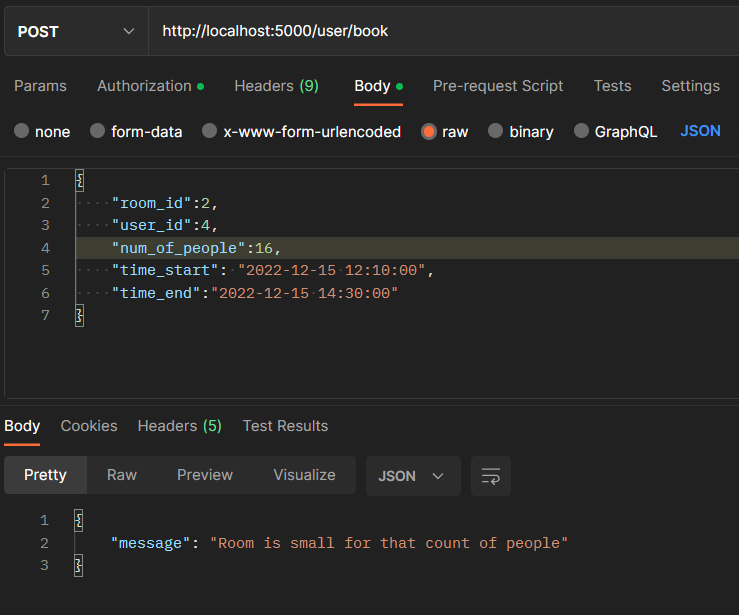
Оновлення аудиторії:

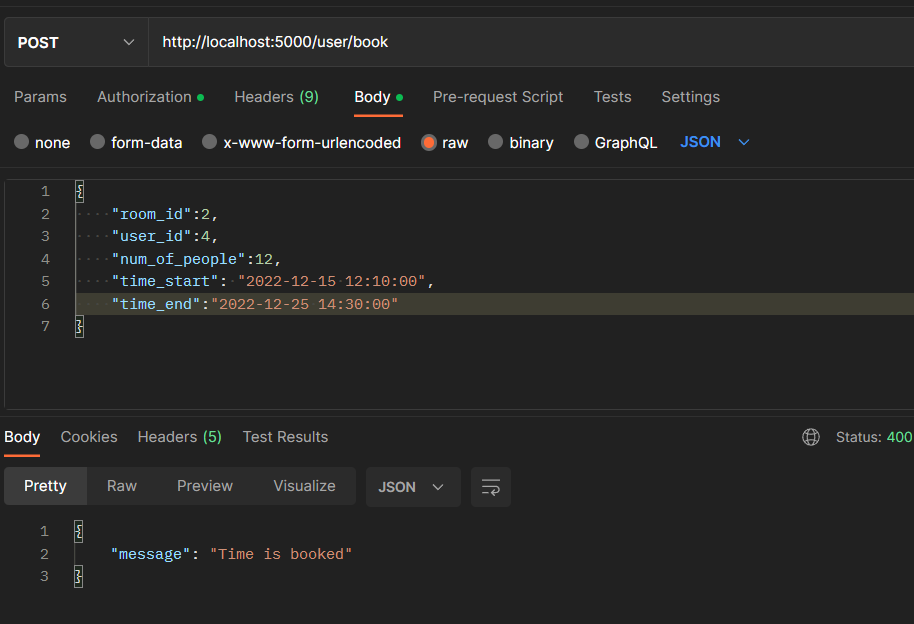


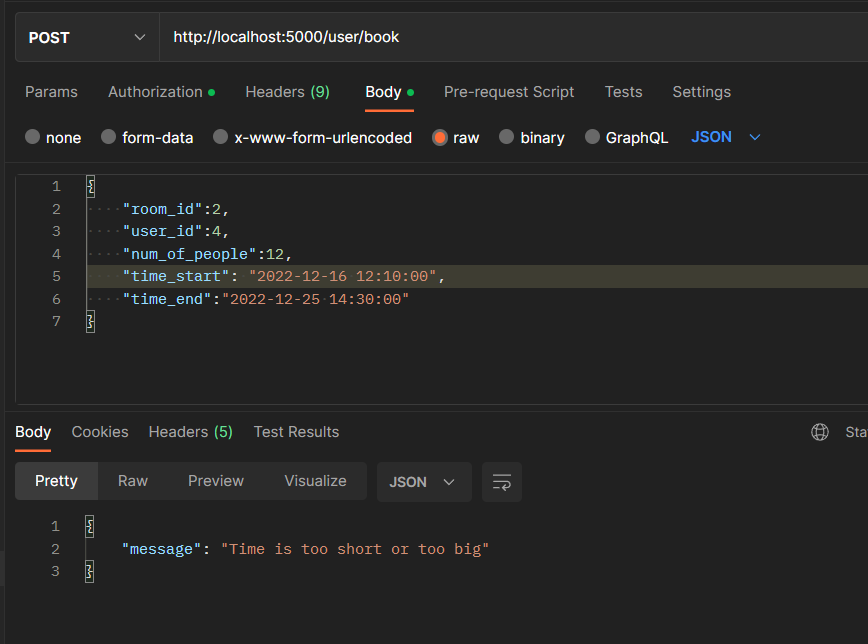


Бронюємо аудиторію:

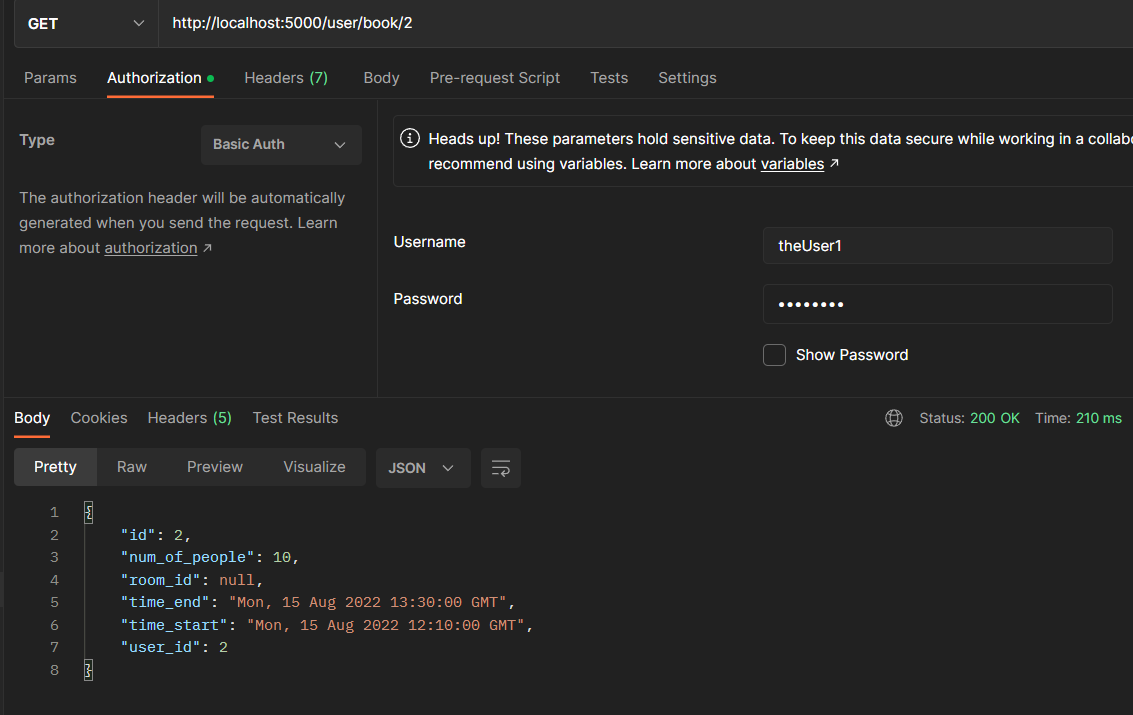


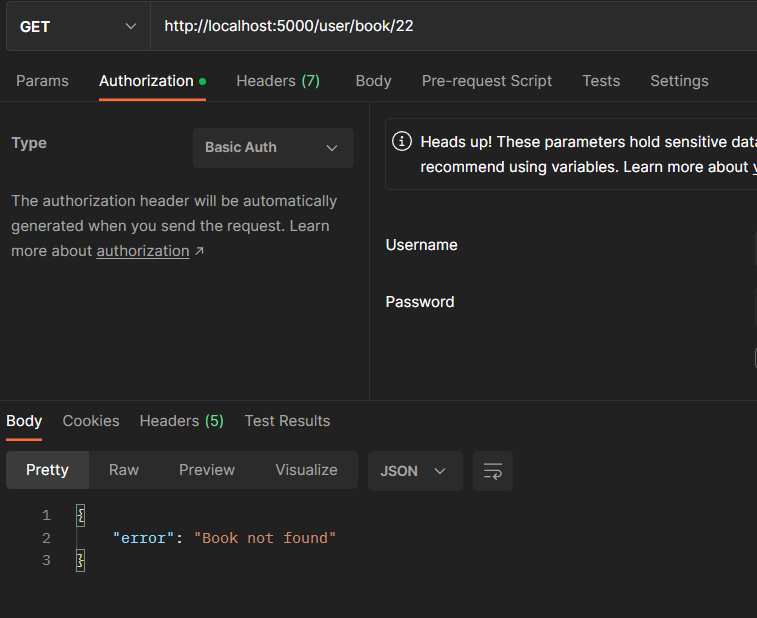




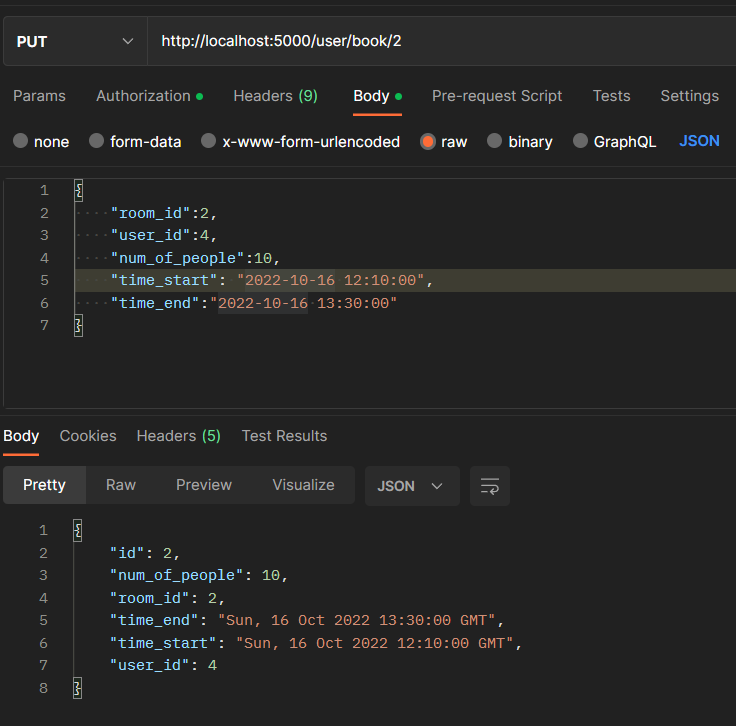


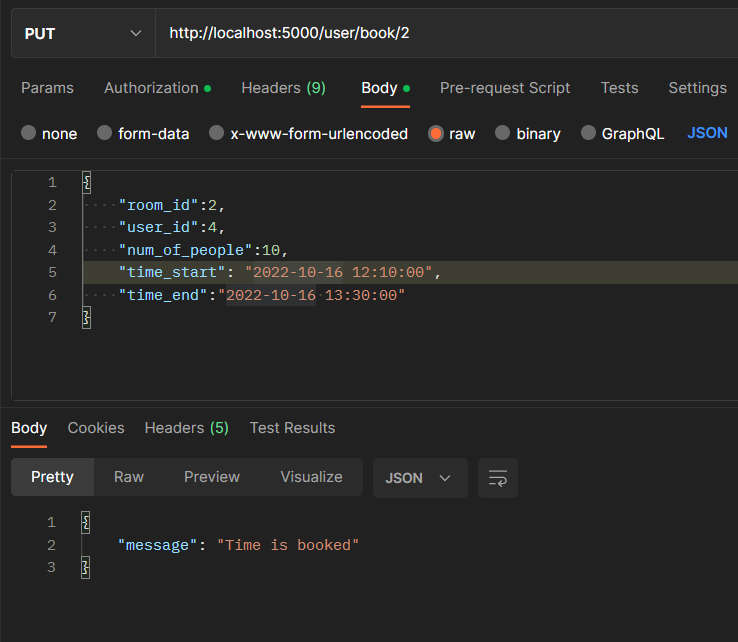
Отримуємо інформацію про бронювання:





Оновлюємо бронювання:





І всі видалення в системі відбуваються без повідомлень,тільки з статус кодом 204.

4. Висновки

Я реалізував серверну частину системи для бронювання аудиторій.

Для виконання курсової я слідкував такому плану:

1. Налаштування сервера;
2. Проектування REST API;
3. Налаштування ORM;
4. Реалізація API;
5. Авторизація;
6. Тестування.

Використав багато різних технологій таких як:

SQLAlchemy, pytest, alembic, flask\_bcrypt, flask\_jwt, Marshmallow, фікстури,SQLAlchemy/orm, pyenv, pipenv . Реалізував OPEN API за допомогою swagger editor. Покрив проект unit тестами за допомогою pytest.

5. Список літератури:

* <https://www.sqlalchemy.org/>
* <https://flask-bcrypt.readthedocs.io/en/1.0.1/>
* <https://editor.swagger.io/>
* <https://alembic.sqlalchemy.org/en/latest/tutorial.html>
* <https://pythonhosted.org/Flask-JWT/>
* https://coverage.readthedocs.io/en/6.5.0/
* <https://docs.pytest.org/en/7.2.x/>
* <https://github.com/pyenv/pyenv>

6. Додатки

models.py

import os

from sqlalchemy import \*

from sqlalchemy.orm import declarative\_base, sessionmaker, relationship

engine = create\_engine("postgresql://postgres:postgres@localhost:5432/booking")

Session = sessionmaker(bind=engine)

BaseModel = declarative\_base()

class Users(BaseModel):

\_\_tablename\_\_ = "users"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

username = Column(String)

first\_name = Column(String)

last\_name = Column(String)

email = Column(String)

password = Column(String)

phone = Column(String)

user\_status = Column(Integer)

class Rooms(BaseModel):

\_\_tablename\_\_ = "rooms"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

name = Column(String)

num\_of\_seats = Column(Integer)

class booked\_room(BaseModel):

\_\_tablename\_\_ = "booked\_room"

id = Column(Integer, Identity(start=1, cycle=False), primary\_key=True)

room\_id = Column(Integer, ForeignKey('rooms.id', ondelete="CASCADE"))

user\_id = Column(Integer, ForeignKey('users.id', ondelete="CASCADE"))

num\_of\_people = Column(Integer)

time\_start = Column(TIMESTAMP)

time\_end = Column(TIMESTAMP)

userToBook = relationship(Users, foreign\_keys=[user\_id], backref="user\_id", lazy="joined", cascade="all, delete")

roomToBook = relationship(Rooms, foreign\_keys=[room\_id], backref="room\_id", lazy="joined", cascade="all, delete")

main.py

from flask import Flask

from rout import user\_blueprint, room\_blueprint

app = Flask(\_\_name\_\_)

app.register\_blueprint(user\_blueprint)

app.register\_blueprint(room\_blueprint)

STUDENT\_ID = 7

@app.route(f'/hello-world')

def hello\_world():

return f"Hello world!", 200

@app.route(f'/hello-world-{STUDENT\_ID}')

def hello\_world\_student():

return f"Hello world, {STUDENT\_ID}!", 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

test\_api.py

import base64

from main import app

import pytest

from models import Session, engine, BaseModel, Users, Rooms, booked\_room

class TestHello:

@pytest.fixture()

def hello\_world(self):

response = app.test\_client().get('/hello-world')

return response.status\_code

def test\_hello\_world(self, hello\_world):

assert hello\_world == 200

class TestHelloStudent:

@pytest.fixture()

def hello\_student(self):

response = app.test\_client().get('/hello-world-7')

return response.status\_code

def test\_hello\_world(self, hello\_student):

assert hello\_student == 200

class TestCreateUser:

@pytest.fixture()

def norm1(self):

user = {

"username": "user1",

"first\_name": "John",

"last\_name": "James",

"email": "john@email.com",

"password": "12345678",

"phone": "12345",

"user\_status": 1

}

return user

@pytest.fixture()

def norm2(self):

user = {

"username": "user2",

"first\_name": "John",

"last\_name": "James",

"email": "john@email.com",

"password": "12345678",

"phone": "12345",

"user\_status": 3

}

return user

@pytest.fixture()

def without(self):

user = {

"username": "user1",

"first\_name": "John",

"last\_name": "James",

"email": "johnemail.com",

"password": "12345678",

"phone": "12345",

"user\_status": 1

}

return user

@pytest.fixture()

def short(self):

user = {

"username": "user1",

"first\_name": "John",

"last\_name": "James",

"email": "johne@mail.com",

"password": "123456",

"phone": "12345",

"user\_status": 1

}

return user

@staticmethod

def create\_tables():

BaseModel.metadata.drop\_all(engine)

BaseModel.metadata.create\_all(engine)

def test\_create\_user(self, norm1):

self.create\_tables()

response = app.test\_client().post('/user', json=norm1)

assert response.status\_code == 200

def test\_create\_user2(self, norm2):

response = app.test\_client().post('/user', json=norm2)

assert response.status\_code == 200

def test\_fail\_create(self, norm1):

response = app.test\_client().post('/user', json=norm1)

assert response.status\_code == 400

def test\_fail\_val(self, without):

response = app.test\_client().post('/user', json=without)

assert response.status\_code == 400

def test\_short\_pass(self, short):

response = app.test\_client().post('/user', json=short)

assert response.status\_code == 400

def test\_get\_none\_user(self):

response = app.test\_client().get('/user/100')

assert response.status\_code == 404

class TestLoginUser:

def test\_login\_user(self):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().get('/user/login', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 200

def test\_fail\_login\_user(self):

valid\_credentials = base64.b64encode(b"user1:4645451").decode("utf-8")

response = app.test\_client().get('/user/login', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 401

class TestLogoutUser:

def test\_logout\_user(self):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().get('/user/logout', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 200

def test\_fail\_logout\_user(self):

valid\_credentials = base64.b64encode(b"user1:4645451").decode("utf-8")

response = app.test\_client().get('/user/logout', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 401

class TestUpdateUser:

@pytest.fixture()

def norm1(self):

user = {

"username": "user1",

"first\_name": "Ostap",

"last\_name": "Ostap",

"email": "pstap@email.com",

"password": "12345678",

"phone": "1111111"

}

return user

@pytest.fixture()

def without(self):

user = {

"username": "user1",

"first\_name": "John",

"last\_name": "James",

"email": "johnemail.com",

"password": "12345678",

"phone": "12345"

}

return user

@pytest.fixture()

def fail(self):

user = {

"username": "user1",

"first\_name": "John",

"last\_name": "James",

"email": "johne@mail.com",

"password": "12345678",

"phone": "12345",

"user\_status": 1

}

return user

def test\_update\_none\_user(self):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/111', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 404

def test\_update\_not\_user(self):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/2', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 403

def test\_update\_user(self, norm1):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm1)

assert response.status\_code == 200

def test\_fail\_user(self, without):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/1', headers={'Authorization': 'Basic ' + valid\_credentials},

json=without)

assert response.status\_code == 400

def test\_none\_json(self):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/1', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 400

def test\_none\_json\_2(self, fail):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=fail)

assert response.status\_code == 400

class TestCreateRoom:

@pytest.fixture()

def norm1(self):

room = {

"name": "room1",

"num\_of\_seats": 10

}

return room

@pytest.fixture()

def fail(self):

room = {

"name": "room2",

"num\_of\_seats": -1

}

return room

@pytest.fixture()

def val(self):

room = {

"name": "room2",

"num\_of\_seats": "a"

}

return room

def test\_create\_room(self, norm1):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/room', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm1)

assert response.status\_code == 200

def test\_fail\_create\_room(self, fail):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/room', headers={'Authorization': 'Basic ' + valid\_credentials}, json=fail)

assert response.status\_code == 400

def test\_without(self, fail):

valid\_credentials = base64.b64encode(b"user2:12345678").decode("utf-8")

response = app.test\_client().post('/room', headers={'Authorization': 'Basic ' + valid\_credentials}, json=fail)

assert response.status\_code == 403

def test\_val(self, val):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/room', headers={'Authorization': 'Basic ' + valid\_credentials}, json=val)

assert response.status\_code == 400

def test\_get\_no\_room(self):

response = app.test\_client().get('/room/111')

assert response.status\_code == 404

class TestUpdateRoom:

@pytest.fixture()

def norm1(self):

room = {

"name": "room1",

"num\_of\_seats": 12

}

return room

@pytest.fixture()

def fail(self):

room = {

"name": "room2",

"num\_of\_seats": -1

}

return room

@pytest.fixture()

def val(self):

room = {

"name": "room2",

"num\_of\_seats": "a"

}

return room

def test\_update\_room(self, norm1):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/room/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm1)

assert response.status\_code == 200

def test\_for\_up(self, norm1):

valid\_credentials = base64.b64encode(b"user2:12345678").decode("utf-8")

response = app.test\_client().put('/room/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm1)

assert response.status\_code == 403

def test\_val(self, val):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/room/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=val)

assert response.status\_code == 400

def test\_fail\_create\_room(self, fail):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/room/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=fail)

assert response.status\_code == 400

def test\_without(self, fail):

valid\_credentials = base64.b64encode(b"user2:12345678").decode("utf-8")

response = app.test\_client().put('/room/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=fail)

assert response.status\_code == 403

def test\_no\_room(self, norm1):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/room/21', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm1)

assert response.status\_code == 404

class TestBook:

@pytest.fixture()

def norm(self):

book = {

"room\_id": 1,

"user\_id": 1,

"num\_of\_people": 10,

"time\_start": "2022-10-15 12:10:00",

"time\_end": "2022-10-15 13:30:00"

}

return book

@pytest.fixture()

def norm2(self):

book = {

"room\_id": 1,

"user\_id": 1,

"num\_of\_people": 10,

"time\_start": "2022-08-15 12:10:00",

"time\_end": "2022-08-15 13:30:00"

}

return book

@pytest.fixture()

def norm\_time(self):

book = {

"room\_id": 1,

"user\_id": 1,

"num\_of\_people": 10,

"time\_start": "2022-10-15 12:10:00",

"time\_end": "2022-10-15 13:30:00"

}

return book

@pytest.fixture()

def val(self):

book = {

"room\_id": "a",

"user\_id": 1,

"num\_of\_people": 10,

"time\_start": "2022-12-15 12:10:00",

"time\_end": "2022-12-15 13:30:00"

}

return book

@pytest.fixture()

def no\_rooms(self):

book = {

"room\_id": 100,

"user\_id": 1,

"num\_of\_people": 10,

"time\_start": "2022-10-15 12:10:00",

"time\_end": "2022-10-15 13:30:00"

}

return book

@pytest.fixture()

def seats(self):

book = {

"room\_id": 1,

"user\_id": 1,

"num\_of\_people": -1,

"time\_start": "2022-11-15 12:10:00",

"time\_end": "2022-11-15 13:30:00"

}

return book

@pytest.fixture()

def time(self):

book = {

"room\_id": 1,

"user\_id": 1,

"num\_of\_people": 10,

"time\_start": "2022-11-15 12:10:00",

"time\_end": "2022-12-15 13:30:00"

}

return book

def test\_book(self, norm):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/user/book', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm)

assert response.status\_code == 200

def test\_book\_time(self, norm\_time):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/user/book', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm\_time)

assert response.status\_code == 400

def test\_val(self, val):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/user/book', headers={'Authorization': 'Basic ' + valid\_credentials}, json=val)

assert response.status\_code == 400

def test\_no\_rooms(self, no\_rooms):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/user/book', headers={'Authorization': 'Basic ' + valid\_credentials}, json=no\_rooms)

assert response.status\_code == 404

def test\_seats(self, seats):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/user/book', headers={'Authorization': 'Basic ' + valid\_credentials}, json=seats)

assert response.status\_code == 400

def test\_time(self, time):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().post('/user/book', headers={'Authorization': 'Basic ' + valid\_credentials}, json=time)

assert response.status\_code == 400

def test\_other\_user(self, norm2):

valid\_credentials = base64.b64encode(b"user2:12345678").decode("utf-8")

response = app.test\_client().post('/user/book', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm2)

assert response.status\_code == 200

def test\_get\_no(self):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().get('/user/book/100', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 404

def test\_forbidden(self):

valid\_credentials = base64.b64encode(b"user2:12345678").decode("utf-8")

response = app.test\_client().get('/user/book/1', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 403

class TestUpdateBook:

@pytest.fixture()

def norm(self):

book = {

"room\_id": 1,

"user\_id": 1,

"num\_of\_people": 11,

"time\_start": "2022-10-11 12:10:00",

"time\_end": "2022-10-11 13:30:00"

}

return book

@pytest.fixture()

def seats(self):

book = {

"room\_id": 1,

"user\_id": 1,

"num\_of\_people": -1,

"time\_start": "2022-11-15 12:10:00",

"time\_end": "2022-11-15 13:30:00"

}

return book

def test\_update\_book(self, norm):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/book/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=norm)

assert response.status\_code == 200

def test\_update\_seats(self, seats):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().put('/user/book/1', headers={'Authorization': 'Basic ' + valid\_credentials}, json=seats)

assert response.status\_code == 400

class TestDeleteRoom:

@pytest.fixture()

def norm(self):

user = {

"username": "userroom",

"first\_name": "John",

"last\_name": "James",

"email": "john@email.com",

"password": "12345678",

"phone": "12345",

"user\_status": 1

}

return user

def test\_room\_delete(self, norm):

app.test\_client().post('/user', json=norm)

valid\_credentials = base64.b64encode(b"userroom:12345678").decode("utf-8")

response = app.test\_client().delete('/room/1', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 204

class TestDeleteUser:

def test\_user\_delete(self):

valid\_credentials = base64.b64encode(b"userroom:12345678").decode("utf-8")

response = app.test\_client().delete('/user/3', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 204

class TestDeleteBook:

def test\_book\_delete(self):

valid\_credentials = base64.b64encode(b"user1:12345678").decode("utf-8")

response = app.test\_client().delete('/user/book/1', headers={'Authorization': 'Basic ' + valid\_credentials})

assert response.status\_code == 204

auth.py

from flask import Flask

from flask\_httpauth import HTTPBasicAuth

import db

from flask\_bcrypt import Bcrypt

import models

app = Flask(\_\_name\_\_)

auth = HTTPBasicAuth()

@auth.verify\_password

def verify\_password(username, password):

user = db.session.query(models.Users).filter(models.Users.username == username).first()

if user is not None and Bcrypt().check\_password\_hash(user.password, password):

return user

return None

rout.py

import datetime

from flask import Blueprint, jsonify, request

from marshmallow import Schema, fields, ValidationError, validate

from flask\_bcrypt import Bcrypt

from models import Users, Rooms, booked\_room

import db

from auth import auth

user\_blueprint = Blueprint('user', \_\_name\_\_, url\_prefix='/user')

room\_blueprint = Blueprint('room', \_\_name\_\_, url\_prefix='/room')

bcrypt = Bcrypt()

@user\_blueprint.route('', methods=['POST'])

def create\_user():

try:

class UserToCreate(Schema):

username = fields.String(required=True)

first\_name = fields.String(required=True)

last\_name = fields.String(required=True)

email = fields.Email(required=True)

password = fields.String(required=True)

phone = fields.Integer(required=True)

user\_status = fields.Integer(required=True)

UserToCreate().load(request.json)

except ValidationError as err:

return jsonify(err.messages), 400

users = db.session.query(Users).filter\_by(username=request.json['username']).all()

if len(users) > 0:

return jsonify({"message": "Username is used"}), 400

if len(request.json['password']) < 8:

return jsonify({"message": "Password is too short"}), 400

user = Users(username=request.json['username'], first\_name=request.json['first\_name'],

last\_name=request.json['last\_name'], email=request.json['email'],

password=bcrypt.generate\_password\_hash(request.json['password']).decode('utf-8'),

phone=request.json['phone'], user\_status=request.json['user\_status'])

try:

db.session.add(user)

except:

db.session.rollback()

return jsonify({"message": "Error user create"}), 500

db.session.commit()

return get\_user(user.id)

@user\_blueprint.route('/<int:user\_id>', methods=['GET'])

def get\_user(user\_id):

user = db.session.query(Users).filter\_by(id=user\_id).first()

if user is None:

return jsonify({'error': 'User not found'}), 404

res\_json = {'id': user.id,

'email': user.email,

'username': user.username,

'first\_name': user.first\_name,

'last\_name': user.last\_name

}

return jsonify(res\_json), 200

@user\_blueprint.route('/<int:user\_id>', methods=['PUT'])

@auth.login\_required

def update\_user(user\_id):

user = db.session.query(Users).filter\_by(id=user\_id).first()

if user is None:

return jsonify({'error': 'No users'}), 404

if user != auth.current\_user():

return jsonify({'error': 'Forbidden'}), 403

try:

class UserToUpdate(Schema):

username = fields.String()

first\_name = fields.String()

last\_name = fields.String()

email = fields.Email()

password = fields.String()

phone = fields.Integer()

if not request.json:

raise ValidationError('No input data provided')

UserToUpdate().load(request.json)

except ValidationError as err:

return jsonify(err.messages), 400

if 'username' in request.json:

users = db.session.query(Users).filter\_by(username=request.json['username']).all()

if len(users) > 0 and users[0].id != user\_id:

return jsonify({"message": "Username is used"}), 400

try:

if 'username' in request.json:

user.username = request.json['username']

if 'first\_name' in request.json:

user.first\_name = request.json['first\_name']

if 'last\_name' in request.json:

user.last\_name = request.json['last\_name']

if 'email' in request.json:

user.email = request.json['email']

if 'password' in request.json:

user.password = bcrypt.generate\_password\_hash(request.json['password']).decode('utf-8')

if 'phone' in request.json:

user.phone = request.json['phone']

except:

db.session.rollback()

return jsonify({"User Data is not valid"}), 400

db.session.commit()

return get\_user(user\_id)

@user\_blueprint.route('/<int:user\_id>', methods=['DELETE'])

@auth.login\_required

def delete\_user(user\_id):

user = db.session.query(Users).filter\_by(id=user\_id).first()

if user != auth.current\_user():

return jsonify({'error': 'Forbidden'}), 403

try:

db.session.delete(user)

except:

db.session.rollback()

return jsonify({"User data is not valid"}), 400

db.session.commit()

return "", 204

@user\_blueprint.route('/login', methods=['GET'])

@auth.login\_required

def login():

return jsonify("Success")

@user\_blueprint.route('/logout', methods=['GET'])

@auth.login\_required

def logout():

return jsonify("Success")

@room\_blueprint.route('', methods=['POST'])

@auth.login\_required

def create\_room():

user = db.session.query(Users).filter\_by(username=auth.username()).first()

if user.user\_status != 1:

return jsonify({'error': 'Forbidden'}), 403

try:

class RoomToCreate(Schema):

name = fields.String(required=True)

num\_of\_seats = fields.Integer(required=True)

RoomToCreate().load(request.json)

except ValidationError as err:

return jsonify(err.messages), 400

if request.json['num\_of\_seats'] < 0:

return ({"message": "num\_of\_seats < 0"}), 400

room = Rooms(name=request.json['name'], num\_of\_seats=request.json['num\_of\_seats'])

try:

db.session.add(room)

except:

db.session.rollback()

return jsonify({"message": "Error room create"}), 500

db.session.commit()

return get\_room(room.id)

@room\_blueprint.route('/<int:room\_id>', methods=['GET'])

def get\_room(room\_id):

room = db.session.query(Rooms).filter\_by(id=room\_id).first()

if room is None:

return jsonify({'error': 'No rooms'}), 404

res\_json = {'id': room.id,

'name': room.name,

'num\_of\_seats': room.num\_of\_seats

}

return jsonify(res\_json), 200

@room\_blueprint.route('/<int:room\_id>', methods=['DELETE'])

@auth.login\_required

def delete\_room(room\_id):

user = db.session.query(Users).filter\_by(username=auth.username()).first()

if user.user\_status != 1:

return jsonify({'error': 'Forbidden'}), 403

room = db.session.query(Rooms).filter\_by(id=room\_id).first()

if room is None:

return jsonify({'error': 'No rooms'}), 404

try:

db.session.delete(room)

except:

db.session.rollback()

return jsonify({"Room data is not valid"}), 400

db.session.commit()

return "", 204

@room\_blueprint.route('/<int:room\_id>', methods=['PUT'])

@auth.login\_required

def update\_room(room\_id):

user = db.session.query(Users).filter\_by(username=auth.username()).first()

if user.user\_status != 1 and user.user\_status != 2:

return jsonify({'error': 'Forbidden'}), 403

try:

class RoomToUpdate(Schema):

name = fields.String()

num\_of\_seats = fields.Integer()

if not request.json:

raise ValidationError('No input data provided')

RoomToUpdate().load(request.json)

except ValidationError as err:

return jsonify(err.messages), 400

if request.json['num\_of\_seats'] < 0:

return ({"message": "num\_of\_seats < 0"}), 400

room = db.session.query(Rooms).filter(Rooms.id == room\_id).first()

if room is None:

return jsonify({'error': 'No rooms'}), 404

try:

if 'name' in request.json:

room.name = request.json['name']

if 'num\_of\_seats' in request.json:

room.num\_of\_seats = request.json['num\_of\_seats']

except:

db.session.rollback()

return jsonify({"Room Data is not valid"}), 400

db.session.commit()

return get\_room(room\_id)

@user\_blueprint.route('/book', methods=['POST'])

@auth.login\_required

def create\_book():

user = db.session.query(Users).filter\_by(username=auth.username()).first()

try:

class RoomToBook(Schema):

room\_id = fields.Integer(required=True)

user\_id = fields.Integer(required=True)

num\_of\_people = fields.Integer(required=True)

time\_start = fields.DateTime(required=True)

time\_end = fields.DateTime(required=True)

RoomToBook().load(request.json)

except ValidationError as err:

return jsonify(err.messages), 400

seats = db.session.query(Rooms).filter\_by(id=request.json['room\_id']).first()

if seats is None:

return ({"message": "No rooms"}), 404

if request.json['num\_of\_people'] < 0 or request.json['num\_of\_people'] > seats.num\_of\_seats:

return ({"message": "Room is small for that count of people"}), 400

books = db.session.query(booked\_room).filter\_by(room\_id=request.json['room\_id']).all()

for check in books:

if check.time\_start <= datetime.datetime.strptime(request.json['time\_start'],

'%Y-%m-%d %H:%M:%S') < check.time\_end:

return ({"message": "Time is booked"}), 400

if check.time\_start < datetime.datetime.strptime(request.json['time\_end'],

'%Y-%m-%d %H:%M:%S') <= check.time\_end:

return ({"message": "Time is booked"}), 400

if datetime.datetime.strptime(request.json['time\_end'], '%Y-%m-%d %H:%M:%S') - datetime.datetime.strptime(

request.json['time\_start'], '%Y-%m-%d %H:%M:%S') > datetime.timedelta(days=5) or datetime.datetime.strptime(

request.json['time\_end'], '%Y-%m-%d %H:%M:%S') - datetime.datetime.strptime(request.json['time\_start'],

'%Y-%m-%d %H:%M:%S') < datetime.timedelta(

hours=1):

return ({"message": "Time is too short or too big"}), 400

if user.user\_status == 1 or user.user\_status == 2:

book = booked\_room(room\_id=request.json['room\_id'], user\_id=request.json['user\_id'],

num\_of\_people=request.json['num\_of\_people'], time\_start=request.json['time\_start'],

time\_end=request.json['time\_end'])

else:

book = booked\_room(room\_id=request.json['room\_id'], user\_id=user.id,

num\_of\_people=request.json['num\_of\_people'], time\_start=request.json['time\_start'],

time\_end=request.json['time\_end'])

try:

db.session.add(book)

except:

db.session.rollback()

return jsonify({"message": "Error book create"}), 500

db.session.commit()

return get\_book(book.id)

@user\_blueprint.route('/book/<int:book\_id>', methods=['GET'])

@auth.login\_required

def get\_book(book\_id):

user = db.session.query(Users).filter\_by(username=auth.username()).first()

book = db.session.query(booked\_room).filter\_by(id=book\_id).first()

if book is None:

return jsonify({'error': 'Book not found'}), 404

if user.user\_status != 1 and user.user\_status != 2 and user.id != book.user\_id:

return jsonify({'error': 'Forbidden'}), 403

res\_json = {'id': book.id,

'room\_id': book.room\_id,

'user\_id': book.user\_id,

'num\_of\_people': book.num\_of\_people,

'time\_start': book.time\_start,

'time\_end': book.time\_end

}

return jsonify(res\_json), 200

@user\_blueprint.route('/book/<int:book\_id>', methods=['DELETE'])

@auth.login\_required

def delete\_book(book\_id):

user = db.session.query(Users).filter\_by(username=auth.username()).first()

if user is None:

return jsonify({'error': 'No users'}), 404

book = db.session.query(booked\_room).filter\_by(id=book\_id).first()

if book is None:

return jsonify({'error': 'No books'}), 404

if user.user\_status != 1 and user.user\_status != 2 and user.id != book.user\_id:

return jsonify({'error': 'Forbidden'}), 403

try:

db.session.delete(book)

except:

db.session.rollback()

return jsonify({"Book data is not valid"}), 400

db.session.commit()

return "", 204

@user\_blueprint.route('/book/<int:book\_id>', methods=['PUT'])

@auth.login\_required

def update\_book(book\_id):

try:

class RoomToBook(Schema):

room\_id = fields.Integer()

user\_id = fields.Integer()

num\_of\_people = fields.Integer()

time\_start = fields.DateTime()

time\_end = fields.DateTime()

if not request.json:

raise ValidationError('No input data provided')

RoomToBook().load(request.json)

except ValidationError as err:

return jsonify(err.messages), 400

user = db.session.query(Users).filter\_by(username=auth.username()).first()

book = db.session.query(booked\_room).filter\_by(id=book\_id).first()

if book is None:

return jsonify({'error': 'No books'}), 404

if user.user\_status != 1 and user.user\_status != 2 and user.id != book.user\_id:

return jsonify({'error': 'Forbidden'}), 403

try:

if 'room\_id' in request.json:

seats = db.session.query(Rooms).filter\_by(id=request.json['room\_id']).first()

if seats is None:

return ({"message": "No rooms"}), 404

book.room\_id = request.json['room\_id']

if 'user\_id' in request.json:

book.user\_id = request.json['user\_id']

if 'num\_of\_people' in request.json:

if request.json['num\_of\_people'] < 0 or request.json['num\_of\_people'] > seats.num\_of\_seats:

return ({"message": "Room is small for that count of people"}), 400

book.num\_of\_people = request.json['num\_of\_people']

if 'time\_start' in request.json and 'time\_end' in request.json:

books = db.session.query(booked\_room).filter\_by(room\_id=request.json['room\_id']).all()

for check in books:

if check.time\_start <= datetime.datetime.strptime(request.json['time\_start'],

'%Y-%m-%d %H:%M:%S') < check.time\_end:

return ({"message": "Time is booked"}), 400

if check.time\_start < datetime.datetime.strptime(request.json['time\_end'],

'%Y-%m-%d %H:%M:%S') <= check.time\_end:

return ({"message": "Time is booked"}), 400

if datetime.datetime.strptime(request.json['time\_end'], '%Y-%m-%d %H:%M:%S') - datetime.datetime.strptime(

request.json['time\_start'], '%Y-%m-%d %H:%M:%S') > datetime.timedelta(

days=5) or datetime.datetime.strptime(

request.json['time\_end'], '%Y-%m-%d %H:%M:%S') - datetime.datetime.strptime(request.json['time\_start'],

'%Y-%m-%d %H:%M:%S') < datetime.timedelta(

hours=1):

return ({"message": "Time is too short or too big"}), 400

book.time\_start = request.json['time\_start']

book.time\_end = request.json['time\_end']

except:

db.session.rollback()

return jsonify({"Book Data is not valid"}), 400

db.session.commit()

return get\_book(book\_id)

swagger.yaml

openapi: 3.0.3

info:

title: Swagger Booking - OpenAPI 3.0

description: |-

This is an API for Booking application.

version: 1.0.0

tags:

- name: user

description: Operations about user

- name: room

description: Operations with rooms

paths:

/user:

post:

tags:

- user

summary: Create user

description: This can only be done by the logged in user.

operationId: createUser

requestBody:

$ref: '#/components/requestBodies/UserToCreate'

responses:

default:

description: successful operation

content:

application/json:

schema:

$ref: '#/components/schemas/User'

application/xml:

schema:

$ref: '#/components/schemas/User'

/user/login:

get:

tags:

- user

summary: Logs user into the system

description: ''

operationId: loginUser

parameters:

- name: username

in: query

description: The user name for login

required: false

schema:

type: string

- name: password

in: query

description: The password for login in clear text

required: false

schema:

type: string

responses:

'200':

description: successful operation

'400':

description: Invalid username/password supplied

'404':

description: User not found

/user/logout:

get:

tags:

- user

summary: Logs out current logged in user session

description: ''

operationId: logoutUser

parameters: []

responses:

default:

description: successful operation

/user/{id}:

get:

tags:

- user

summary: Get user by id

description: ''

operationId: getUserById

parameters:

- name: id

in: path

description: 'The id that needs to be fetched.'

required: true

schema:

type: string

responses:

'200':

description: successful operation

content:

application/json:

schema:

$ref: '#/components/schemas/User'

application/xml:

schema:

$ref: '#/components/schemas/User'

'400':

description: Invalid id supplied

'404':

description: User not found

security:

- booking\_auth:

- admin

- manager

- customer

put:

tags:

- user

summary: Update user

description: This can only be done by the logged in user.

operationId: updateUser

parameters:

- name: id

in: path

description: id that need to be deleted

required: true

schema:

type: string

requestBody:

description: Update an existent user in the database

content:

application/json:

schema:

type: object

properties:

firstName:

type: string

example: John

lastName:

type: string

example: James

email:

type: string

example: john@email.com

password:

type: string

example: '12345'

phone:

type: string

example: '12345'

userStatus:

type: integer

description: User Status

format: int32

example: 1

xml:

name: user

responses:

default:

description: successful operation

security:

- booking\_auth:

- admin

- manager

- customer

delete:

tags:

- user

summary: Delete user

description: This can only be done by the logged in user.

operationId: deleteUser

parameters:

- name: id

in: path

description: The name that needs to be deleted

required: true

schema:

type: string

responses:

'200':

description: successful operation

'400':

description: Invalid username supplied

'404':

description: User not found

security:

- booking\_auth:

- admin

- manager

- customer

/user/book:

post:

tags:

- user

summary: Book a room to the database

description: Book a room to the database

operationId: bookRoom

requestBody:

$ref: '#/components/requestBodies/RoomToBook'

responses:

'200':

description: Successful operation

content:

application/json:

schema:

$ref: '#/components/schemas/Room'

application/xml:

schema:

$ref: '#/components/schemas/Room'

'405':

description: Invalid input

security:

- booking\_auth:

- admin

/user/book/{id}:

get:

tags:

- user

summary: Get booking by id

description: ''

operationId: getBookingById

parameters:

- name: id

in: path

description: 'The id that needs to be fetched.'

required: true

schema:

type: string

responses:

'200':

description: successful operation

content:

application/json:

schema:

$ref: '#/components/schemas/Booking'

application/xml:

schema:

$ref: '#/components/schemas/Booking'

'400':

description: Invalid id supplied

'404':

description: User not found

security:

- booking\_auth:

- admin

- manager

- customer

put:

tags:

- user

summary: Update booking

description: This can only be done by the logged in user.

operationId: updateBooking

parameters:

- name: id

in: path

description: id that need to be deleted

required: true

schema:

type: string

requestBody:

description: Update an existent booking in the database

content:

application/json:

schema:

type: object

properties:

room\_id:

type: integer

format: int64

example: 2

user\_id:

type: integer

format: int64

example: 2

num\_of\_people:

type: integer

format: int64

example: 22

time\_start:

type: string

example: '2022-10-10 13:10:00'

time\_end:

type: string

example: '2022-10-10 14:10:00'

xml:

name: booking

responses:

default:

description: successful operation

security:

- booking\_auth:

- admin

- manager

- customer

delete:

tags:

- user

summary: Delete booking

description: This can only be done by the logged in user.

operationId: deleteBooking

parameters:

- name: id

in: path

description: The name that needs to be deleted

required: true

schema:

type: string

responses:

'200':

description: successful operation

'400':

description: Invalid id supplied

'404':

description: Booking not found

security:

- booking\_auth:

- admin

- manager

- customer

/room:

post:

tags:

- room

summary: Add a new room to the database

description: Add a new room to the database

operationId: addRoom

requestBody:

$ref: '#/components/requestBodies/RoomToCreate'

responses:

'200':

description: Successful operation

content:

application/json:

schema:

$ref: '#/components/schemas/Room'

application/xml:

schema:

$ref: '#/components/schemas/Room'

'405':

description: Invalid input

security:

- booking\_auth:

- admin

/room/{roomId}:

get:

tags:

- room

summary: Find room by ID

description: Returns a single room

operationId: getRoomById

parameters:

- name: roomId

in: path

description: ID of room to return

required: true

schema:

type: integer

format: int64

responses:

'200':

description: successful operation

content:

application/json:

schema:

$ref: '#/components/schemas/Room'

application/xml:

schema:

$ref: '#/components/schemas/Room'

'400':

description: Invalid ID supplied

'404':

description: Room not found

security:

- api\_key: []

- booking\_auth:

- admin

- manager

put:

tags:

- room

summary: Update a room in the database with form data

description: ''

operationId: updateRoomWithForm

parameters:

- name: roomId

in: path

description: ID of room that needs to be updated

required: true

schema:

type: integer

format: int64

- name: name

in: query

description: Name of room that needs to be updated

schema:

type: string

- name: numOfSeats

in: query

description: Number of seats in the room that needs to be updated

schema:

type: string

responses:

'405':

description: Invalid input

security:

- booking\_auth:

- admin

delete:

tags:

- room

summary: Delete a room

description: delete a room

operationId: deleteRoom

parameters:

- name: api\_key

in: header

description: ''

required: false

schema:

type: string

- name: roomId

in: path

description: Room id to delete

required: true

schema:

type: integer

format: int64

responses:

'400':

description: Invalid room id

security:

- booking\_auth:

- admin

components:

schemas:

User:

type: object

properties:

id:

type: integer

format: int64

example: 10

username:

type: string

example: theUser

firstName:

type: string

example: John

lastName:

type: string

example: James

email:

type: string

example: john@email.com

password:

type: string

example: '12345'

phone:

type: string

example: '12345'

userStatus:

type: integer

description: User Status

format: int32

example: 1

xml:

name: user

Room:

required:

- id

- name

- numOfSeats

type: object

properties:

id:

type: integer

format: int64

example: 2

name:

type: string

example: 'Room #5'

numOfSeats:

type: integer

format: int64

example: 22

xml:

name: room

Booking:

required:

- id

- room\_id

- user\_id

- time\_start

- time\_end

- num\_of\_people

type: object

properties:

id:

type: integer

format: int64

example: 2

room\_id:

type: integer

format: int64

example: 2

user\_id:

type: integer

format: int64

example: 2

num\_of\_people:

type: integer

format: int64

example: 22

time\_start:

type: string

example: '2022-10-10 13:10:00'

time\_end:

type: string

example: '2022-10-10 14:10:00'

xml:

name: room

requestBodies:

UserToCreate:

description: Created user object

required: true

content:

application/json:

schema:

type: object

properties:

username:

type: string

example: theUser

firstName:

type: string

example: John

lastName:

type: string

example: James

email:

type: string

example: john@email.com

password:

type: string

example: '12345'

phone:

type: string

example: '12345'

userStatus:

type: integer

description: User Status

format: int32

example: 1

xml:

name: user

RoomToCreate:

description: Create a new room

required: true

content:

application/json:

schema:

required:

- name

- numOfSeats

type: object

properties:

name:

type: string

example: 'Room #5'

numOfSeats:

type: integer

format: int64

example: 22

xml:

name: room

RoomToBook:

description: Book a room

required: true

content:

application/json:

schema:

required:

- name

- numOfSeats

- time\_start

- time\_end

type: object

properties:

name:

type: string

example: 'Room #5'

numOfSeats:

type: integer

format: int64

example: 22

time\_start:

type: string

example: '2022-10-10 13:10'

time\_end:

type: string

example: '2022-10-10 14:10'

xml:

name: room

securitySchemes:

booking\_auth:

type: oauth2

flows:

implicit:

authorizationUrl: url

scopes:

admin : can do all actions

manager : can update infos about room

user : can book room and change infos about booking

api\_key:

type: apiKey

name: api\_key

in: header