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:

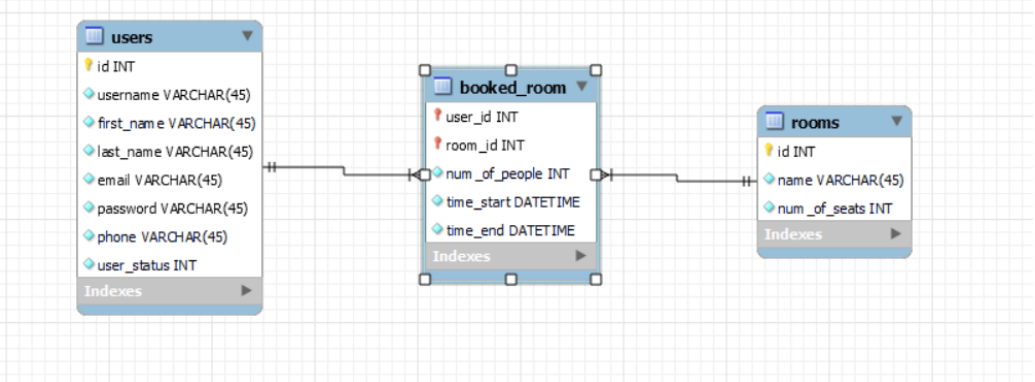


рис.1 ER-діаграма

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:

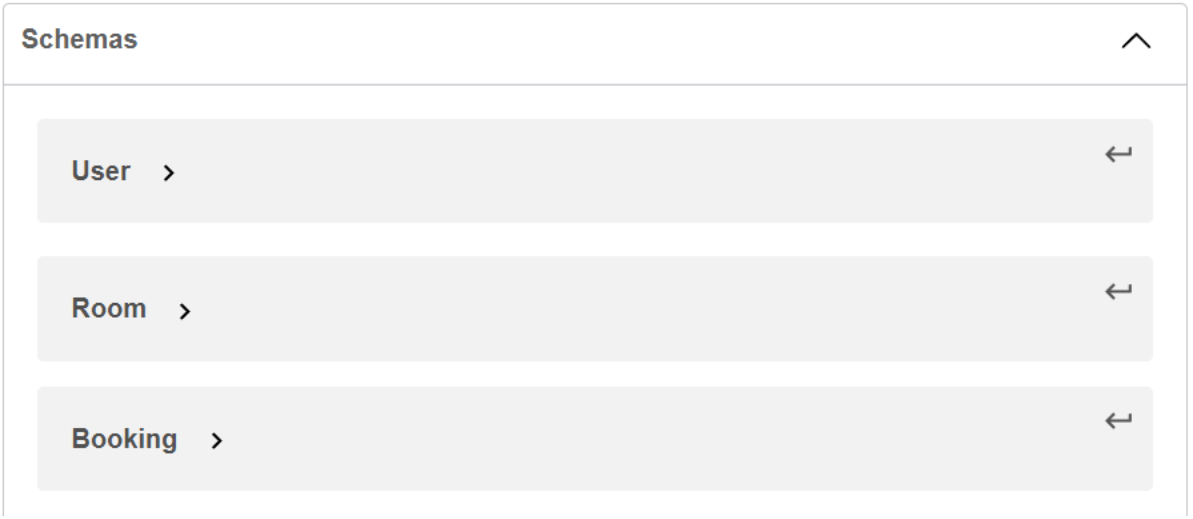
```
import os
from sqlalchemy import *
from sqlalchemy.orm import declarative_base, sessionmaker, relationship


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)
```

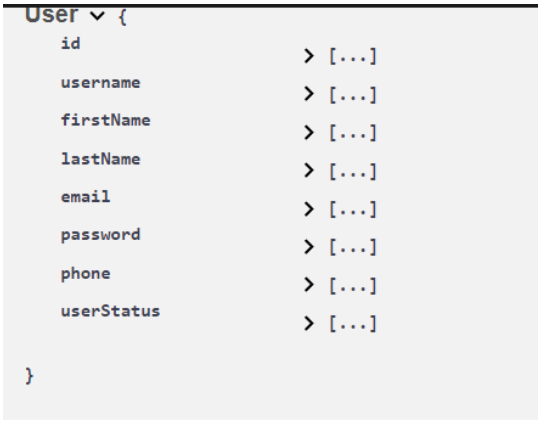
```
class booked_room(BaseModel):

    __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")
```

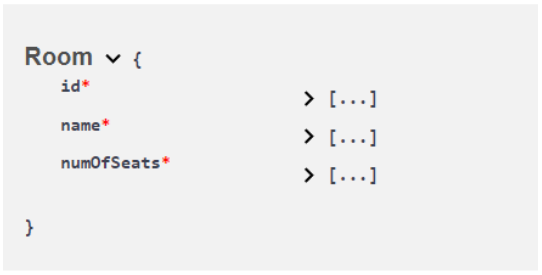
2.2. Опис REST API

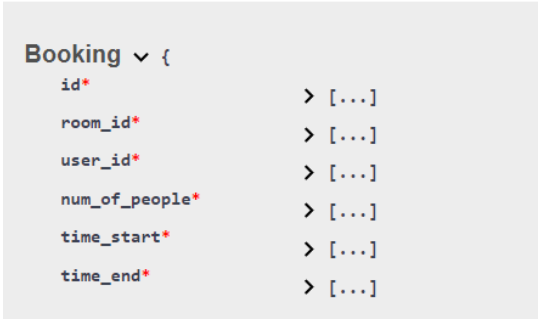
Наша API мати наступні сутності:



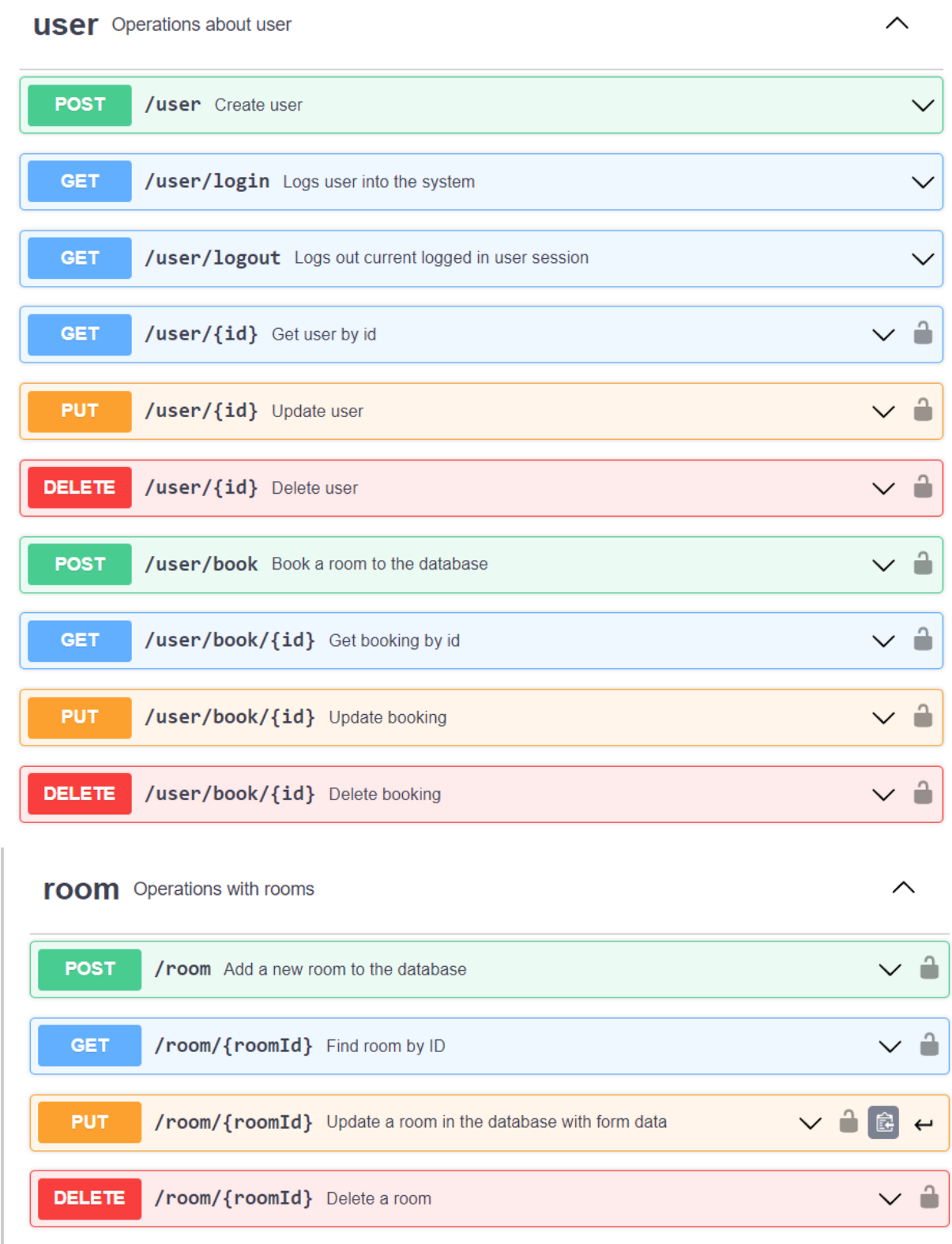
З такими властивостями:







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.

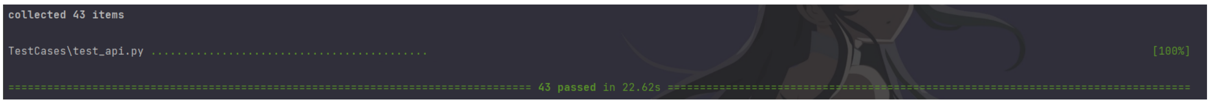
Room методи:

- 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

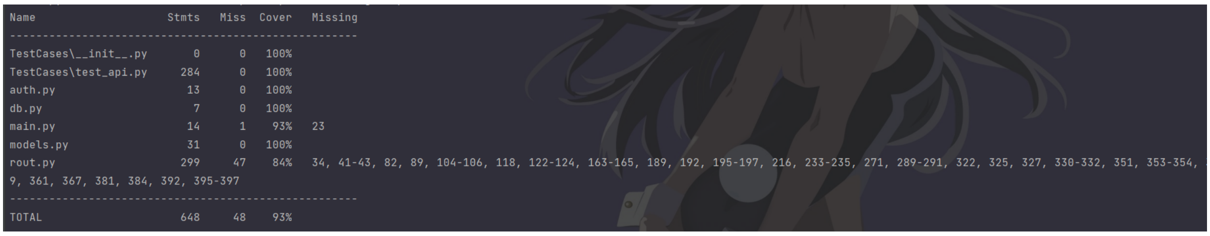
Для Unit тестування я використав пакет pytest, а для показу покриття коду тестів пакет coverage.

Результат тестів:



```
collected 43 items

TestCases\test_api.py ..........................................                [100%]

=========================== 43 passed in 22.62s ===========================
```

Покриття:

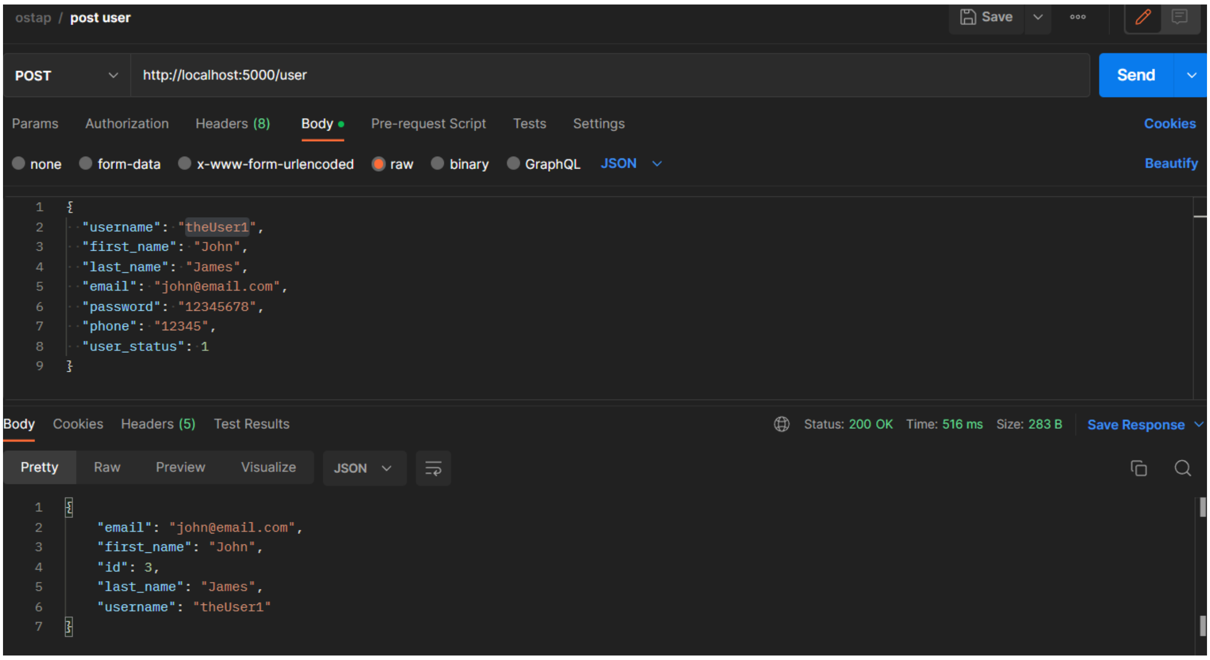


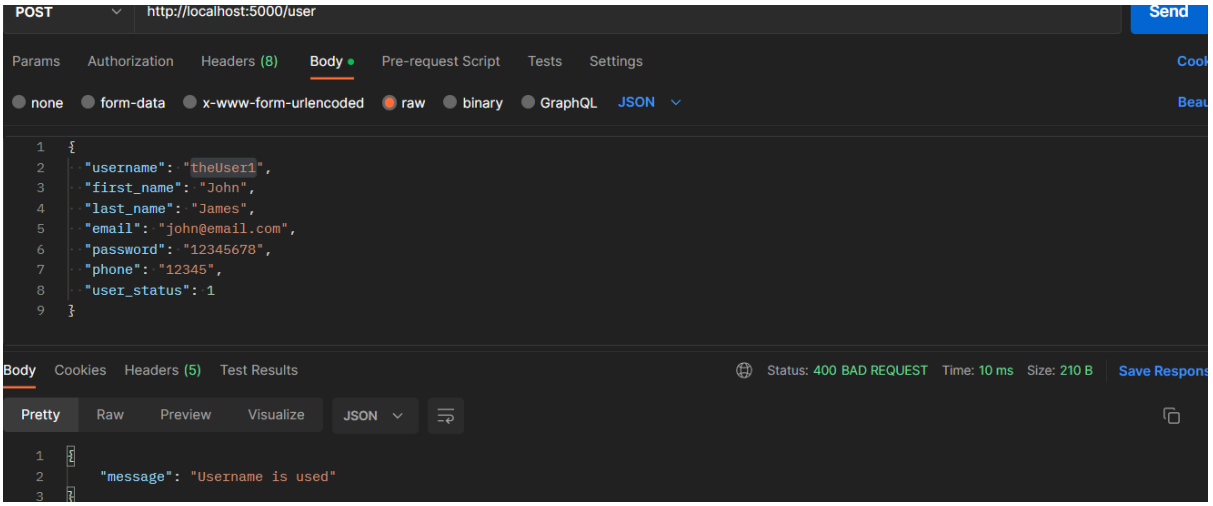
```
Name                    Stmts   Miss  Cover   Missing
-----------------------------------------------------
TestCases\__init__.py       0      0   100%
TestCases\test_api.py     284      0   100%
auth.py                    13      0   100%
db.py                       7      0   100%
main.py                    14      1    93%   23
models.py                  31      0   100%
rout.py                   299     47    84%   34, 41-43, 82, 89, 104-106, 118, 122-124, 163-165, 189, 192, 195-197, 216, 233-235, 271, 289-291, 322, 325, 327, 330-332, 351, 353-354,
9, 361, 367, 381, 384, 392, 395-397
-----------------------------------------------------
TOTAL                     648     48    93%
```

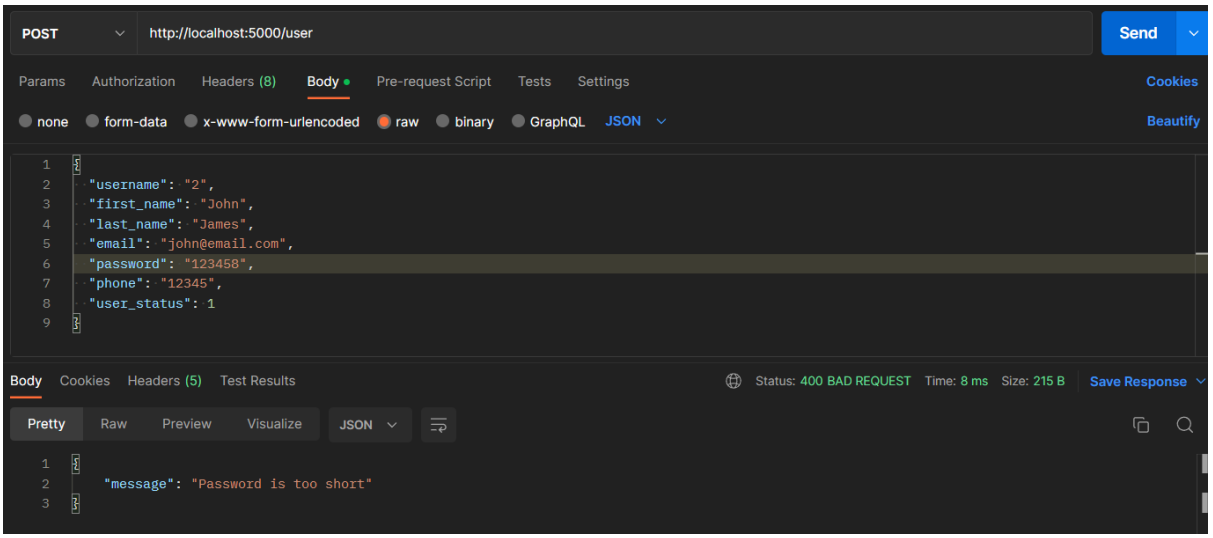
93% - результат нашого покриття.

3. Аналіз результатів та інструкція користувача

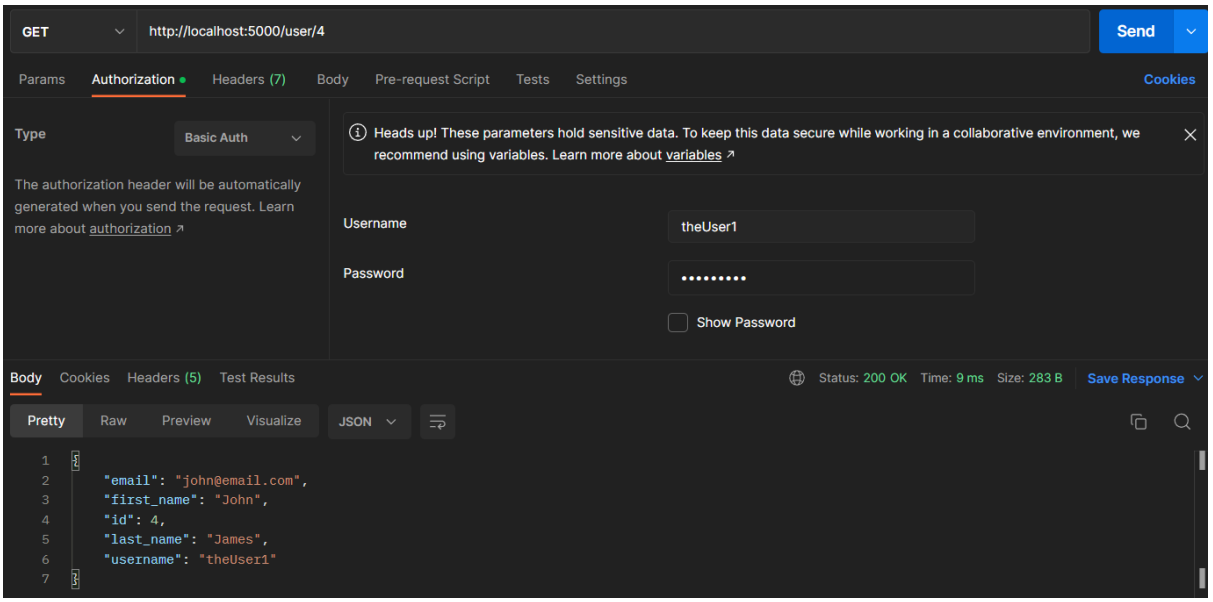
Створюємо користувача:

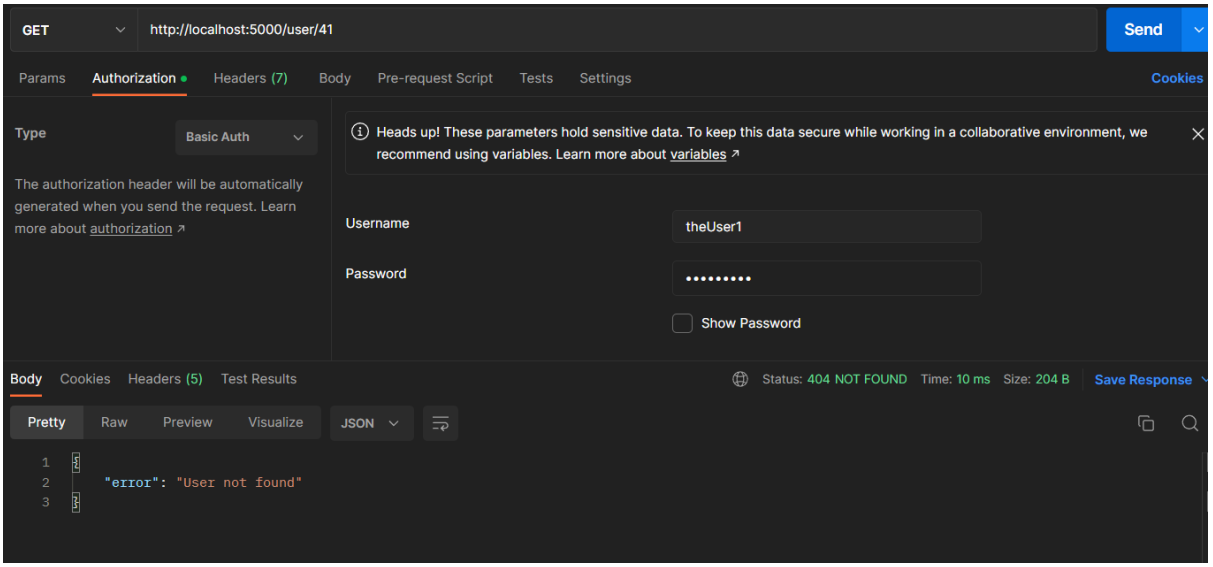




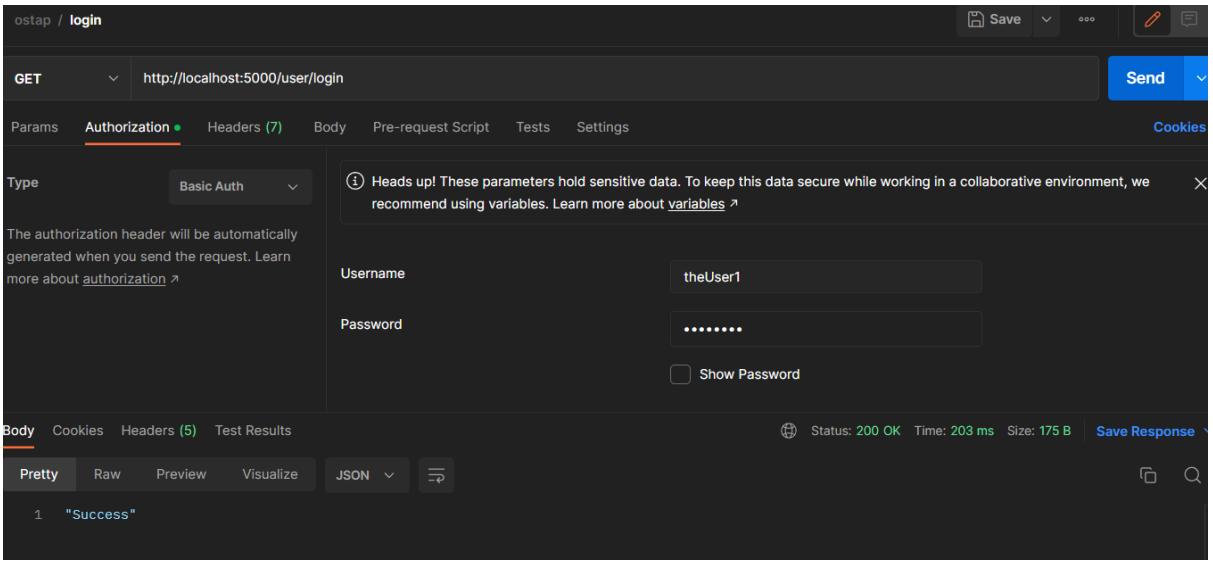


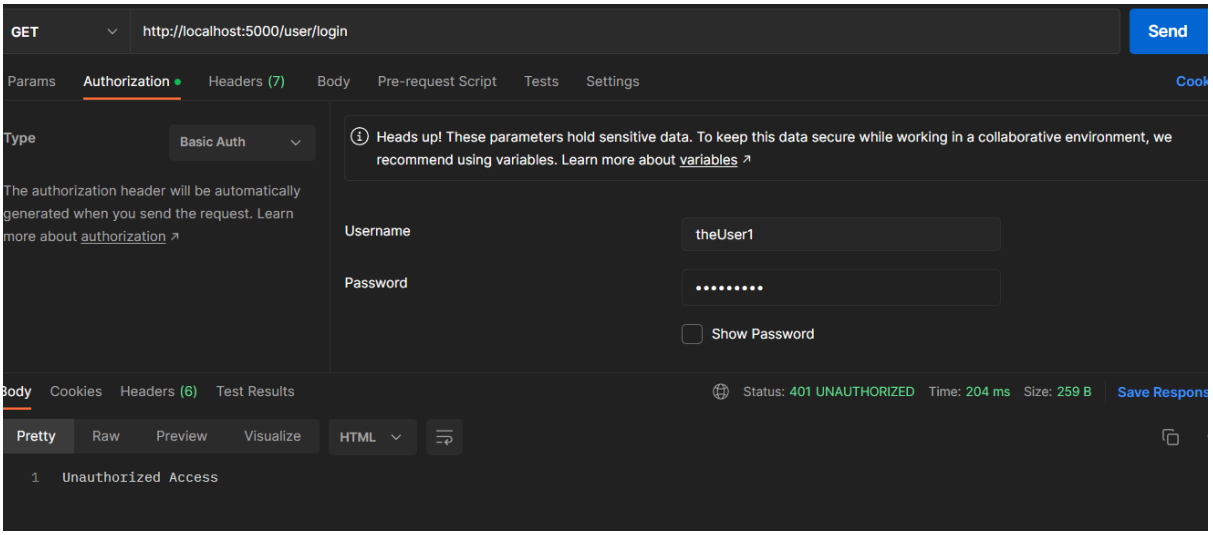
Отримуємо інформацію про користувача:



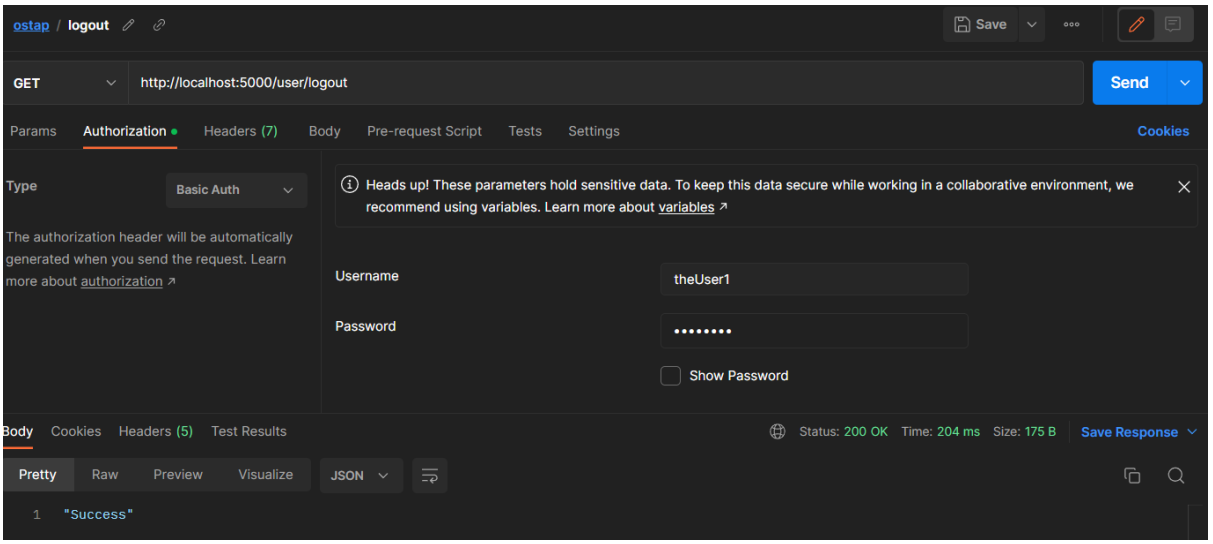


Авторизація:

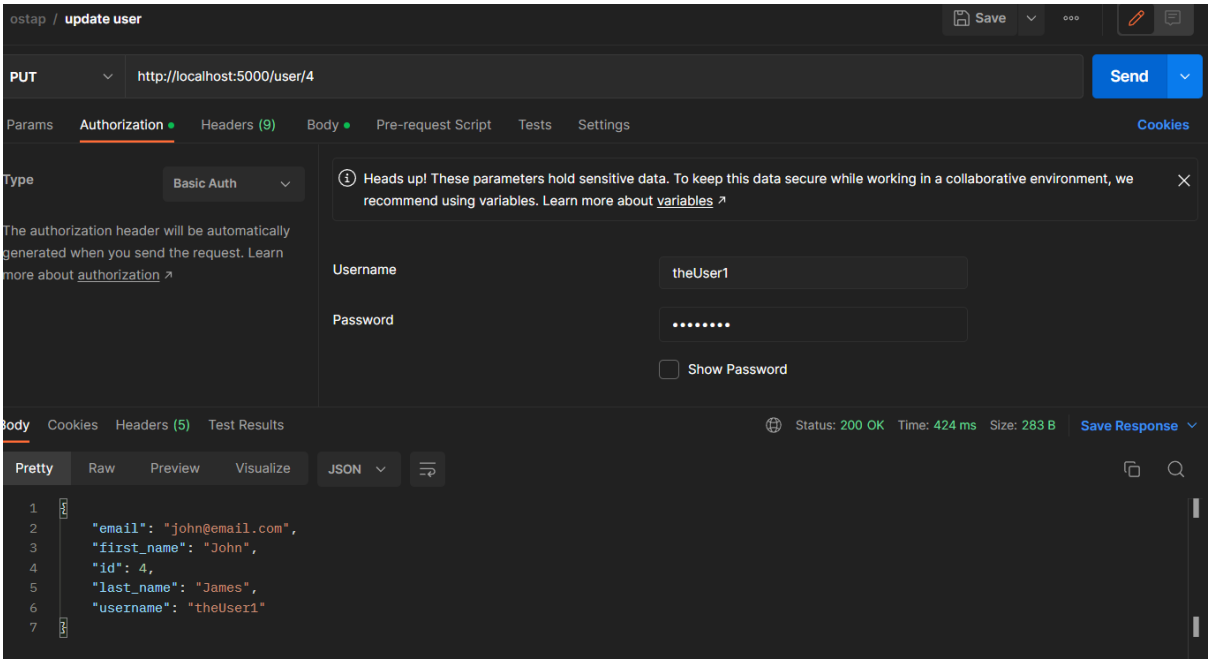


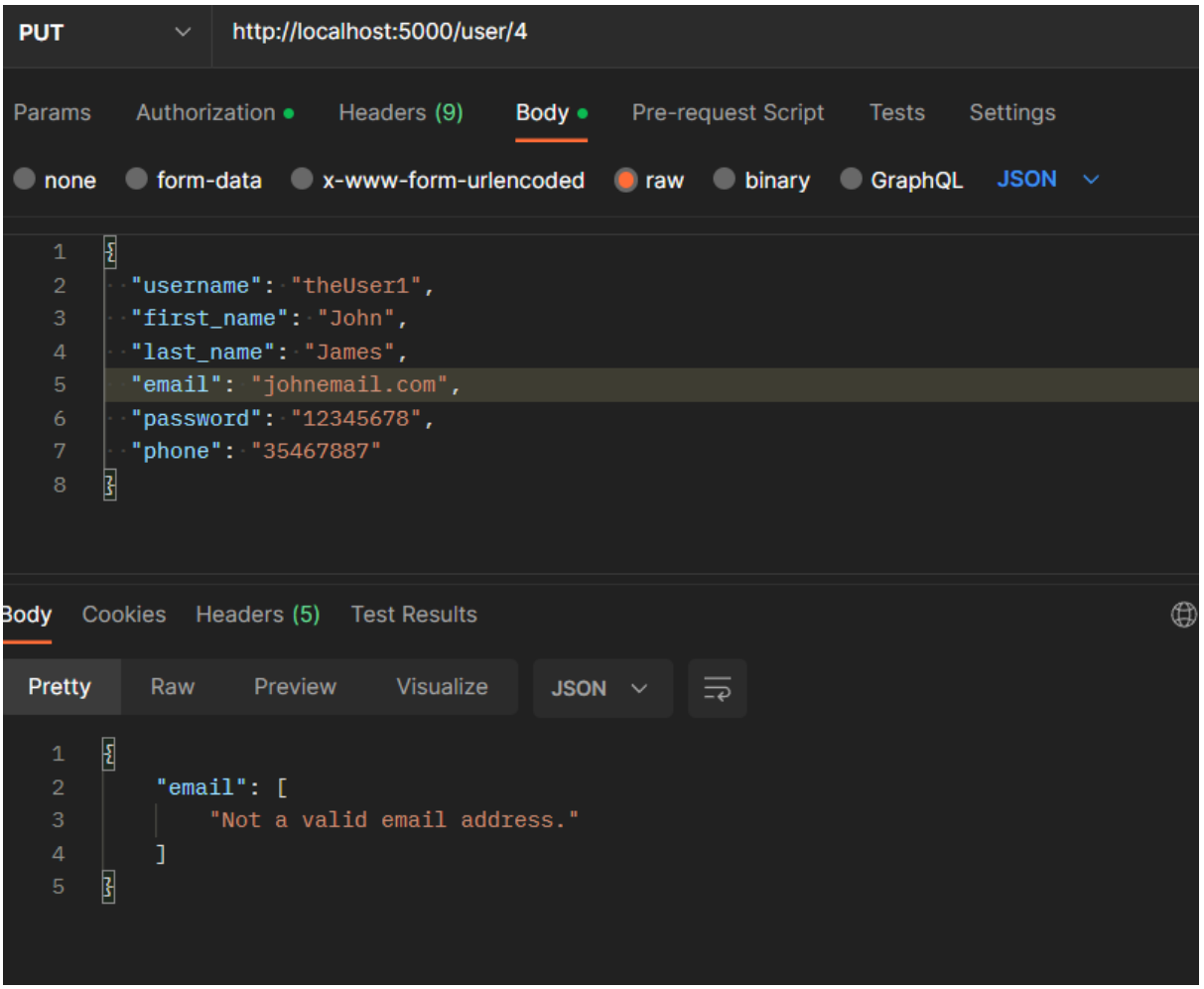


Вихід з системи:

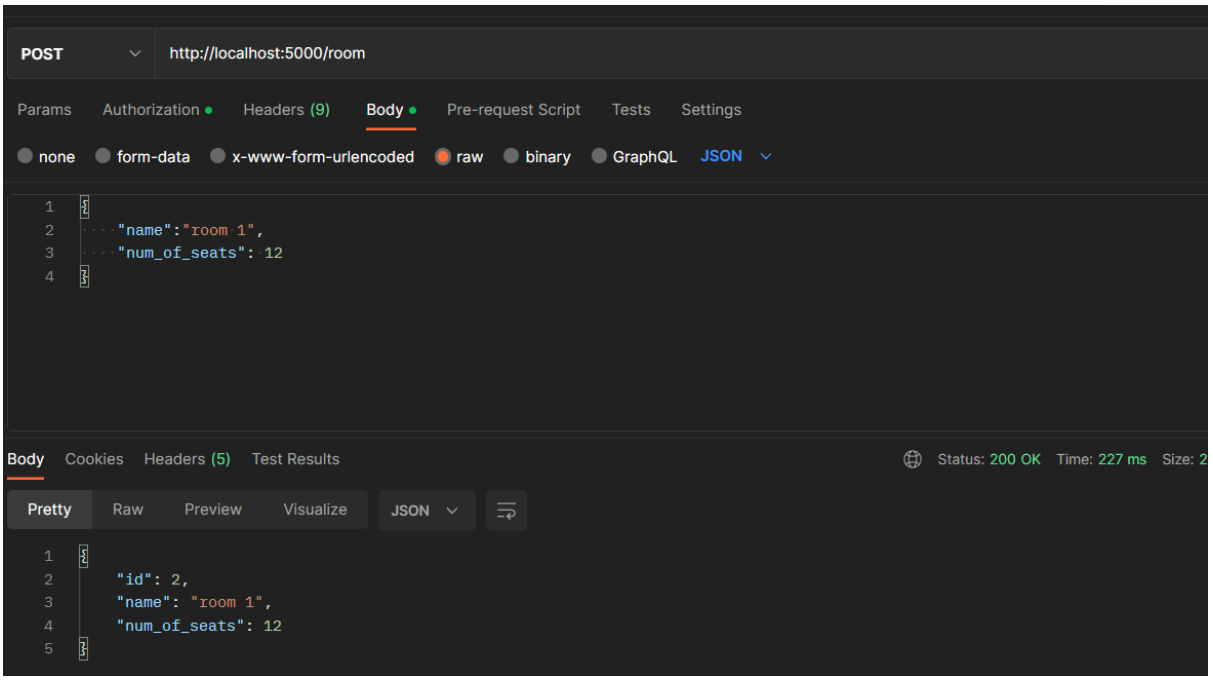


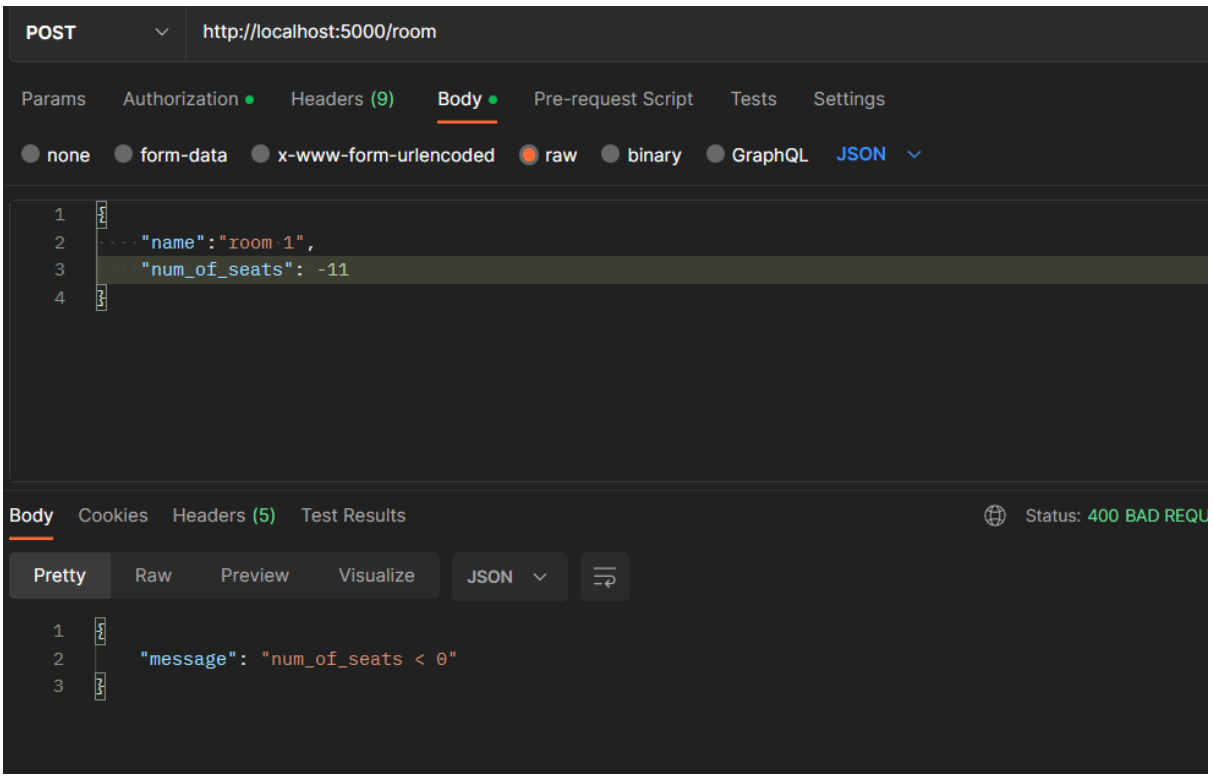
Оновлення користувача:



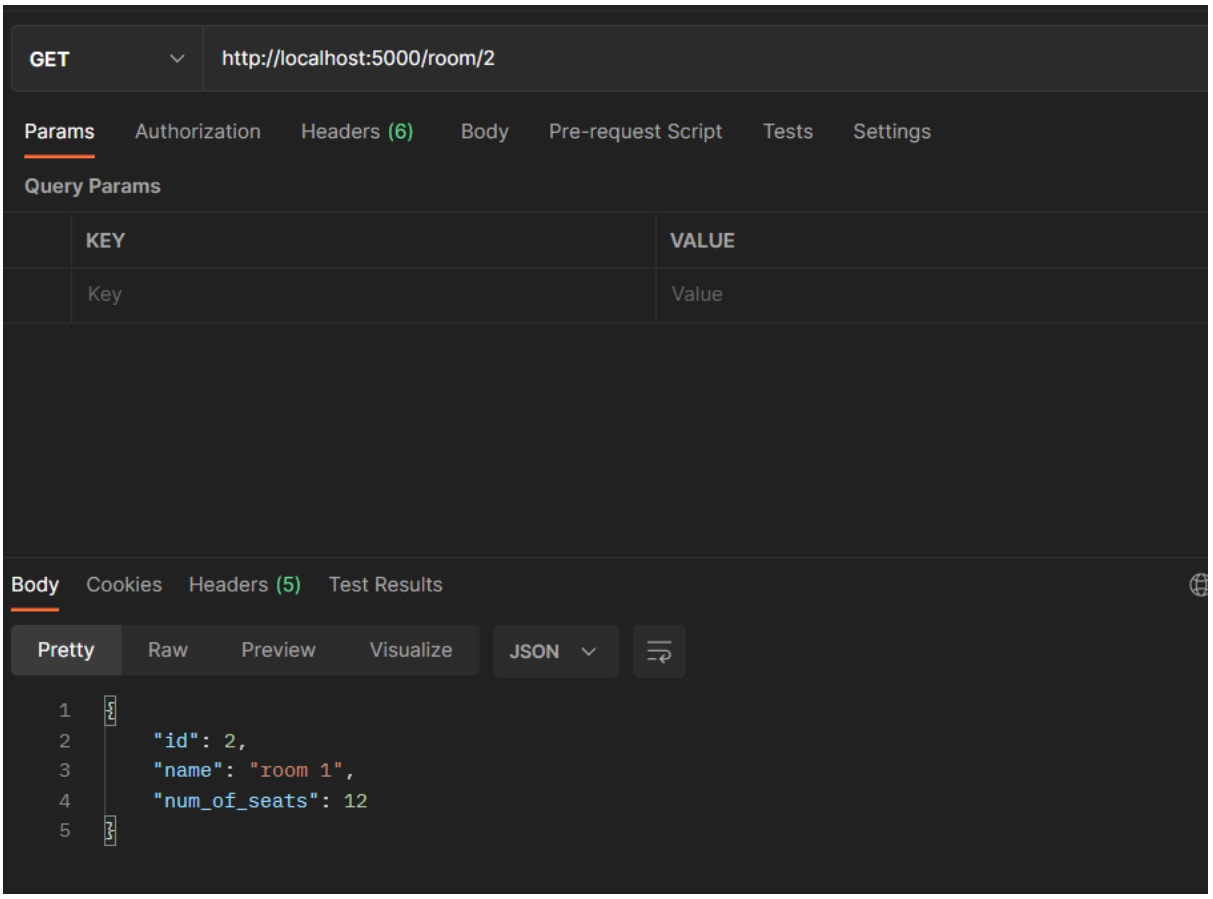


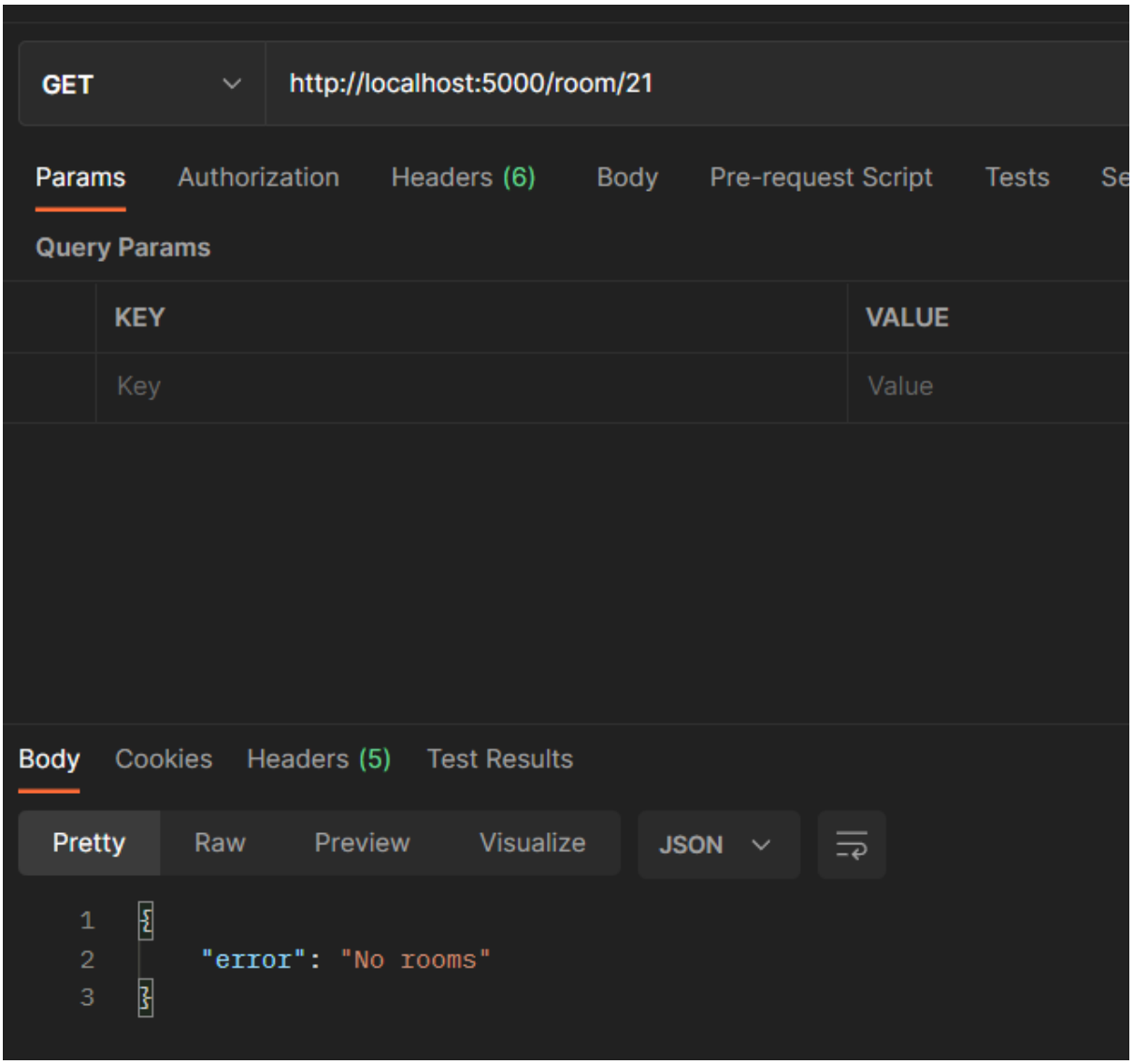
Створення аудиторії:



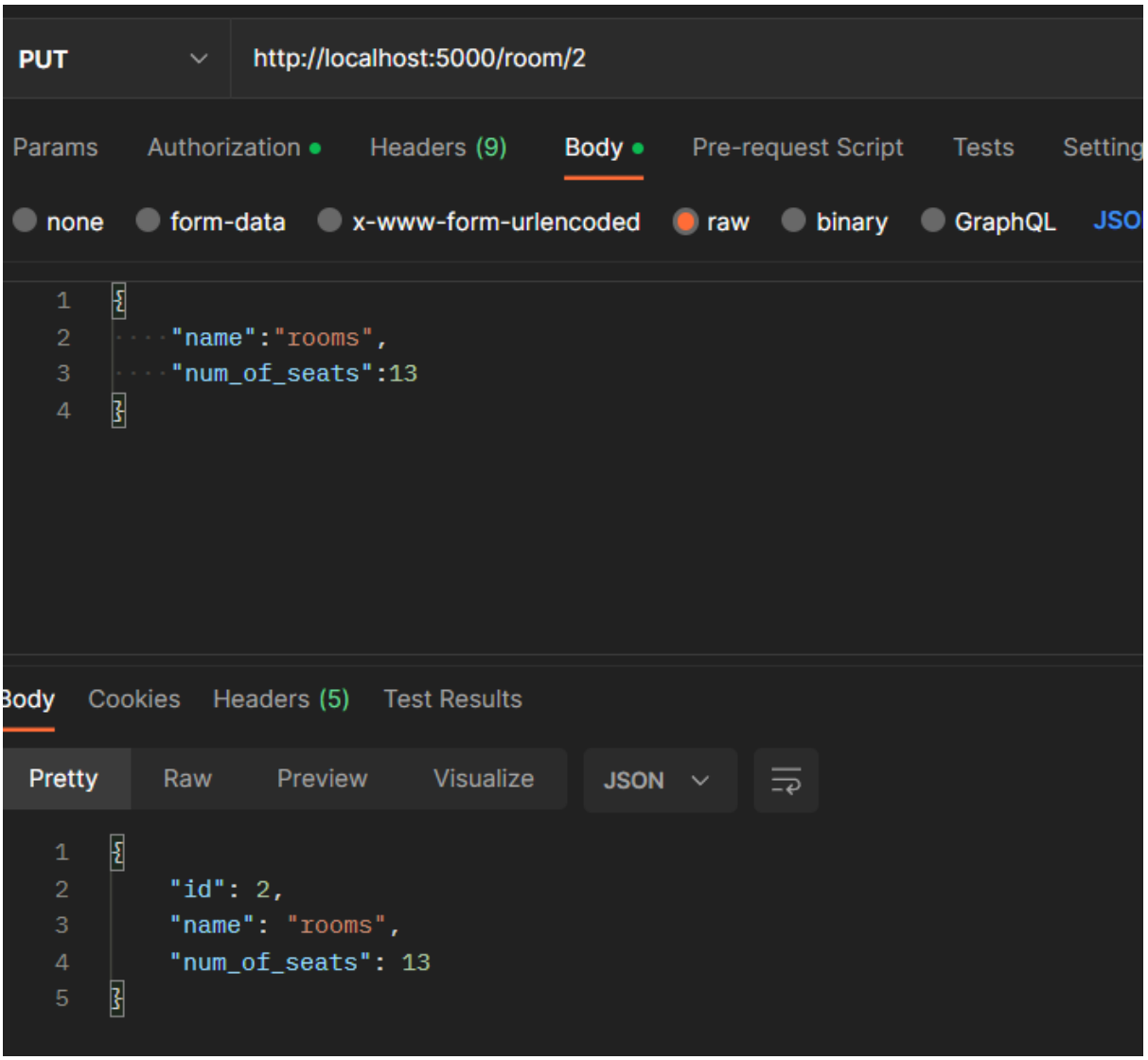


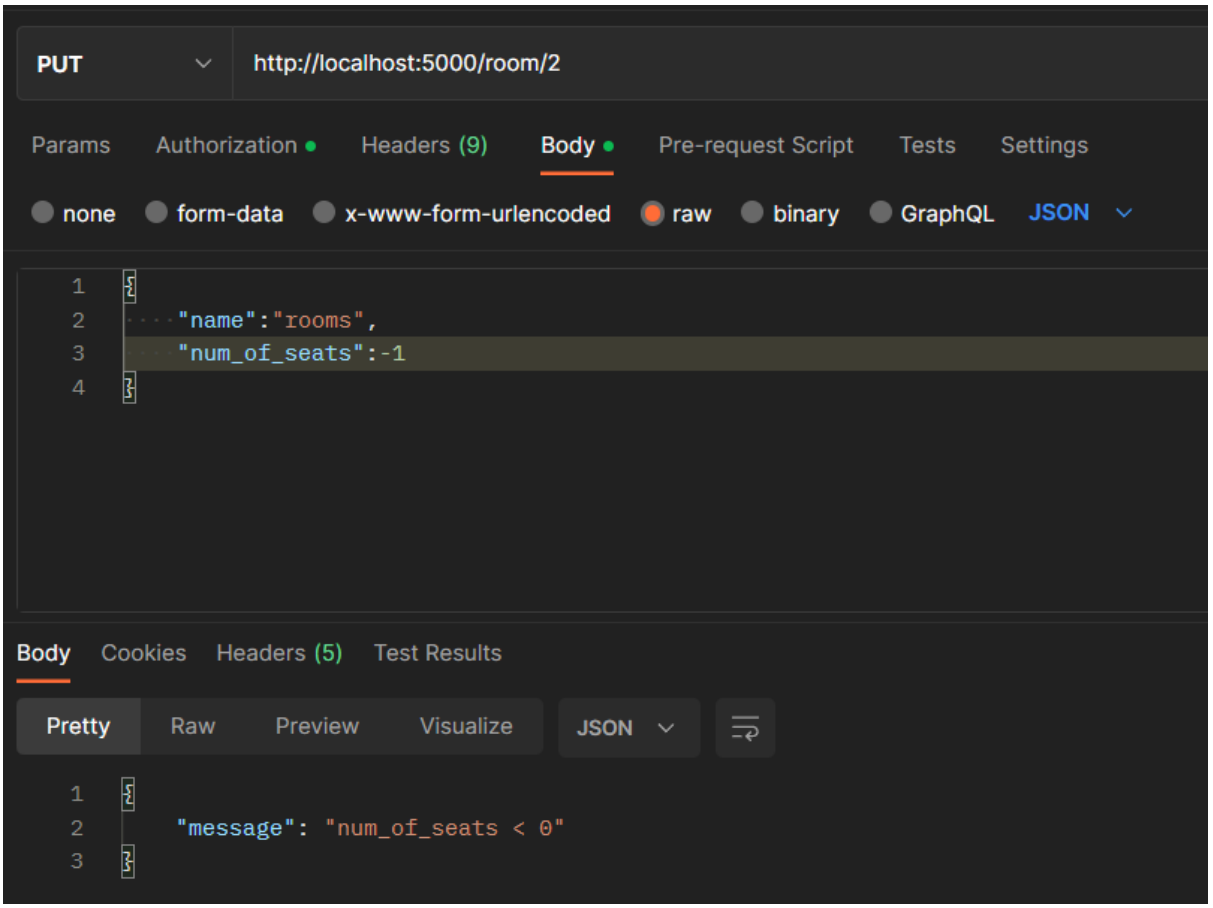
Отримання інформації про аудиторію:


GET
http://localhost:5000/room/2
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Query Params
KEY
VALUE
Key
Value
Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"name": "room 1",
"num_of_seats": 12
}

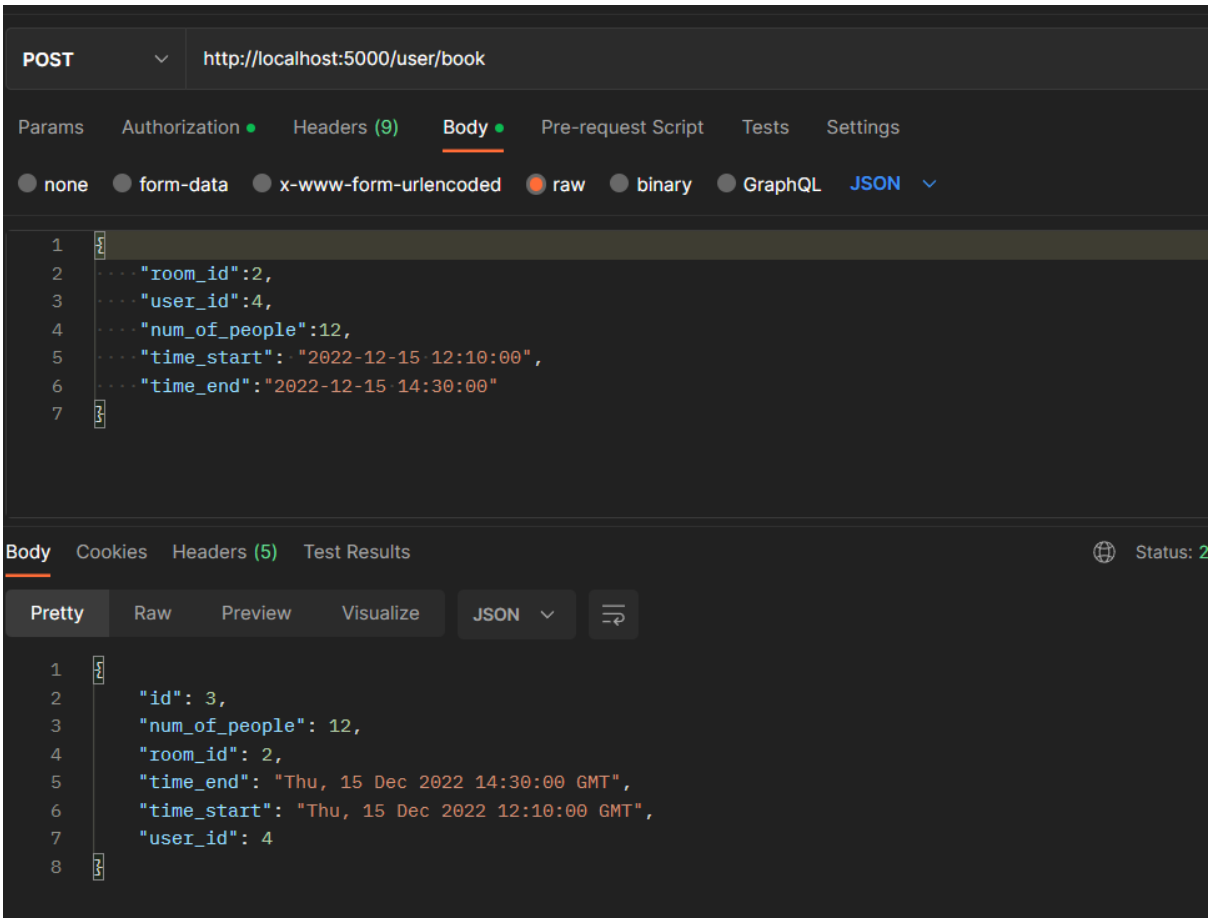


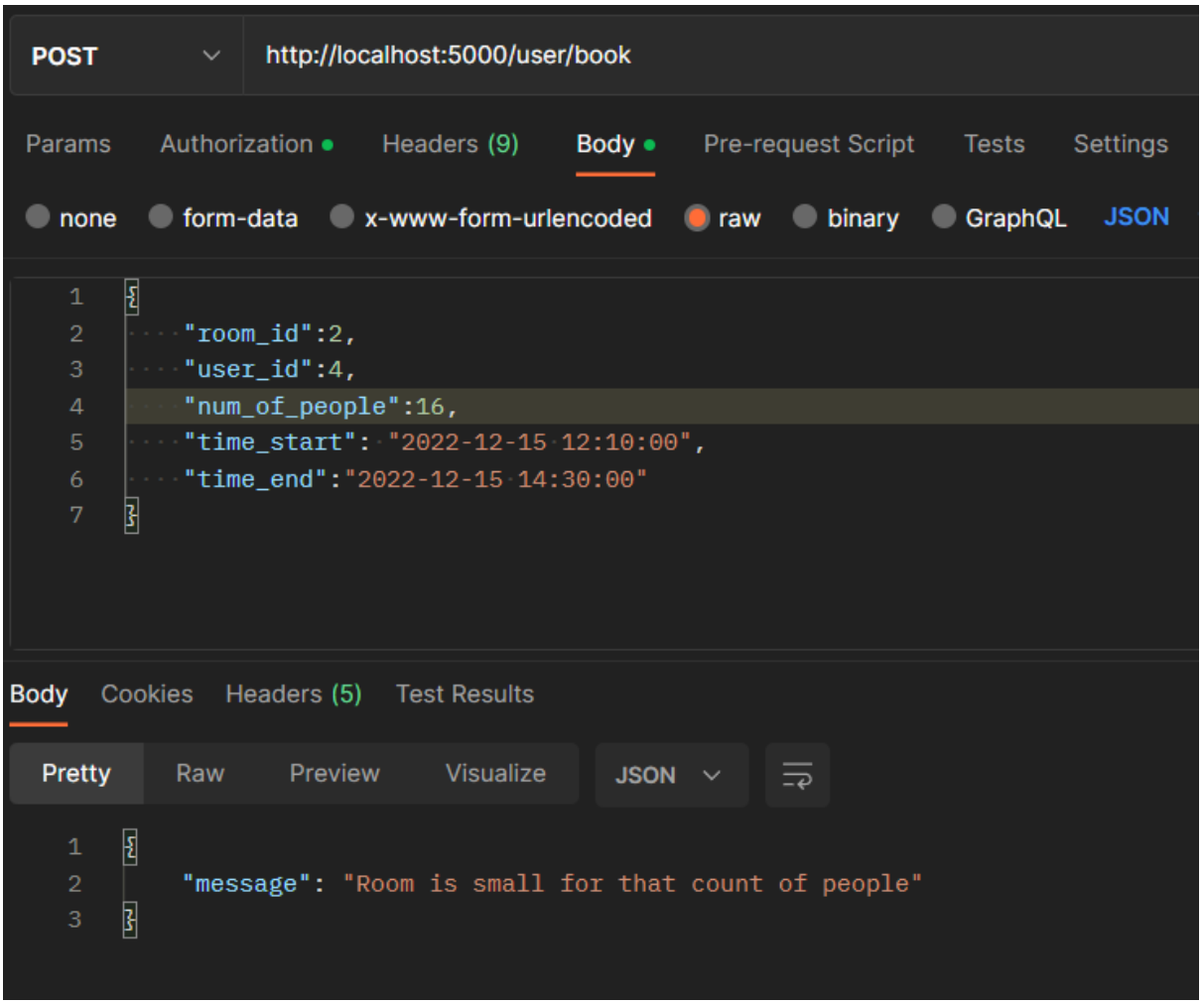
Оновлення аудиторії:

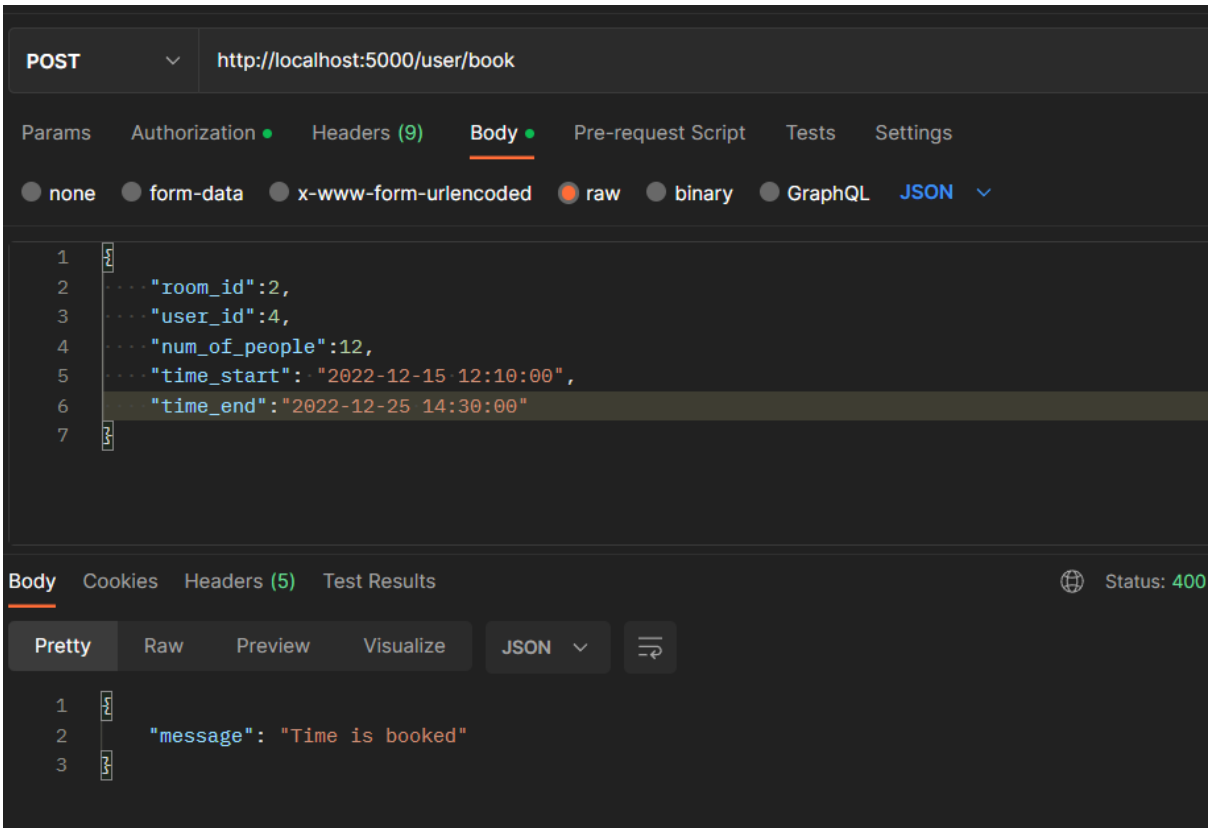

PUT
http://localhost:5000/room/2
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Setting
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSO
1 {
2 "name":"rooms",
3 "num_of_seats":13
4 }
Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "id": 2,
3 "name": "rooms",
4 "num_of_seats": 13
5 }

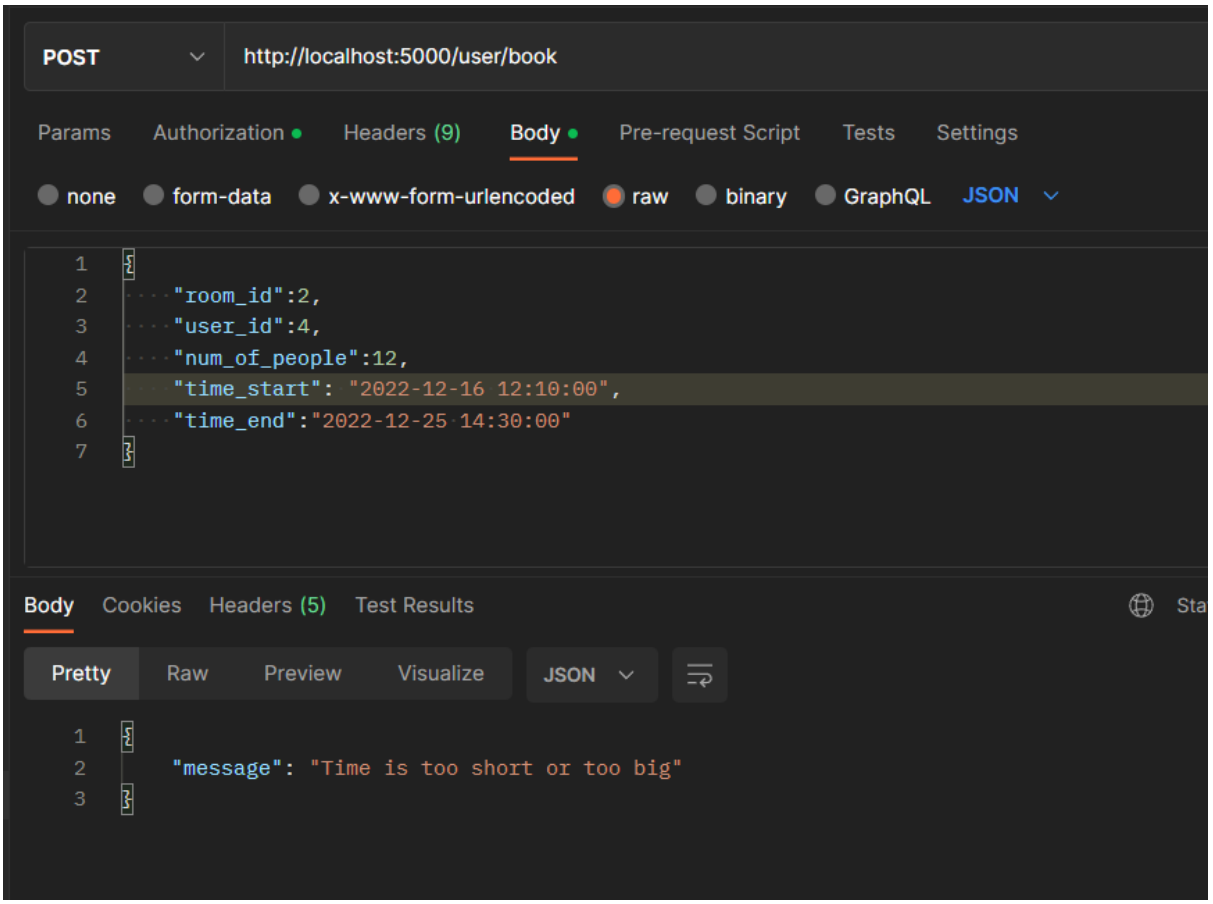


Бронюємо аудиторію:

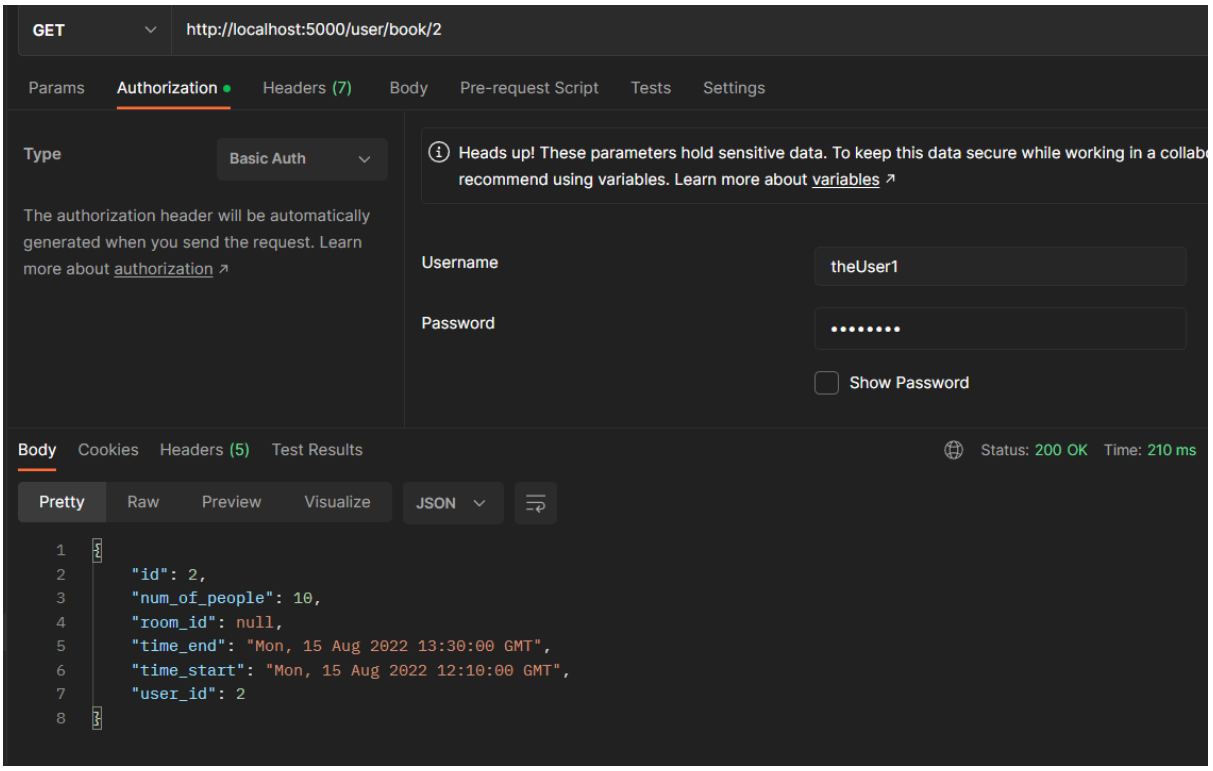

POST
http://localhost:5000/user/book
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"room_id":2,
"user_id":4,
"num_of_people":12,
"time_start": "2022-12-15 12:10:00",
"time_end":"2022-12-15 14:30:00"
}
Body
Cookies
Headers (5)
Test Results
Status: 2
Pretty
Raw
Preview
Visualize
JSON
{
"id": 3,
"num_of_people": 12,
"room_id": 2,
"time_end": "Thu, 15 Dec 2022 14:30:00 GMT",
"time_start": "Thu, 15 Dec 2022 12:10:00 GMT",
"user_id": 4
}

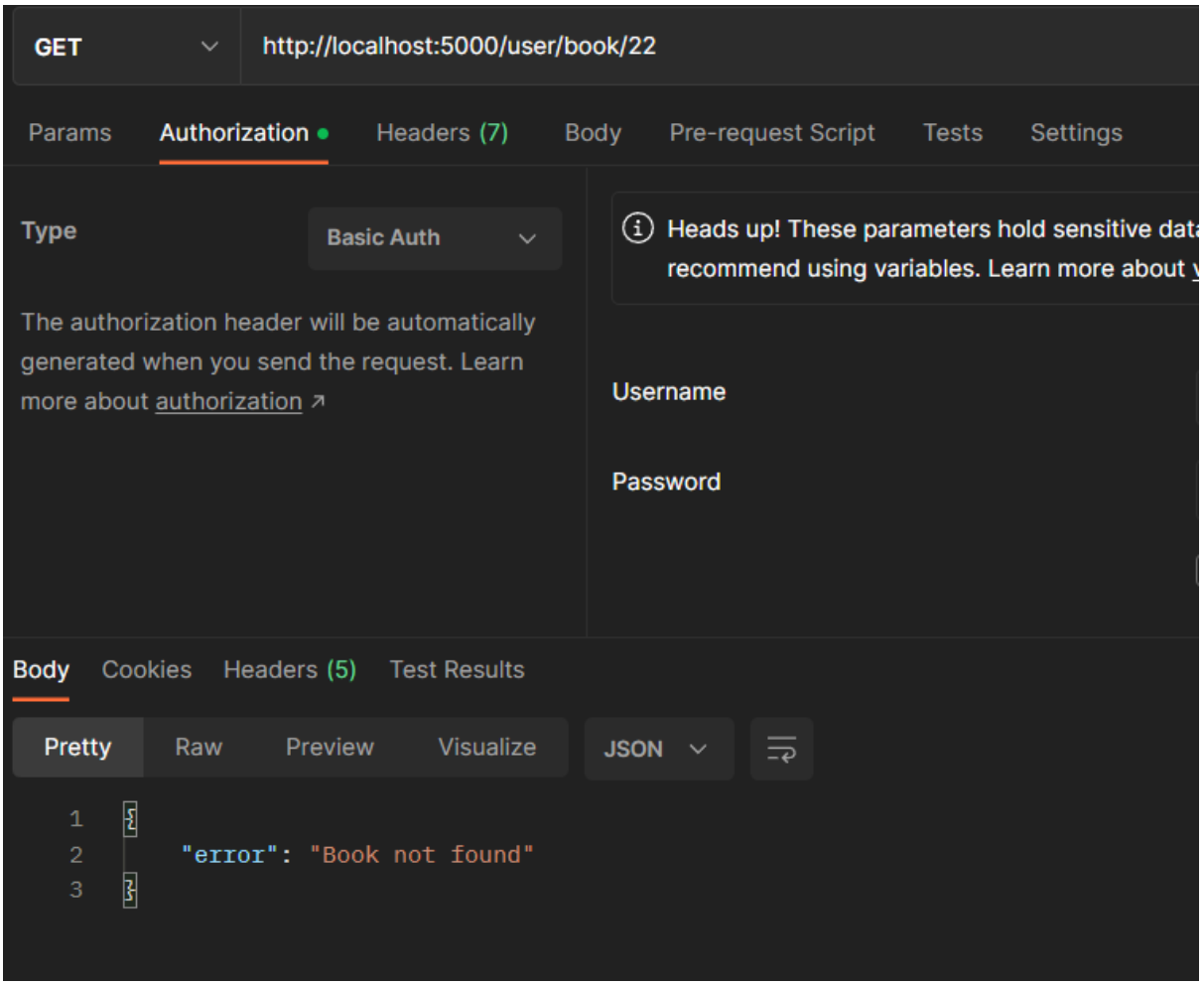

POST
http://localhost:5000/user/book
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"room_id":2,
"user_id":4,
"num_of_people":16,
"time_start": "2022-12-15 12:10:00",
"time_end":"2022-12-15 14:30:00"
}
Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Room is small for that count of people"
}


POST
http://localhost:5000/user/book
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"room_id":2,
"user_id":4,
"num_of_people":12,
"time_start": "2022-12-15 12:10:00",
"time_end":"2022-12-25 14:30:00"
}
Body
Cookies
Headers (5)
Test Results
Status: 400
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Time is booked"
}

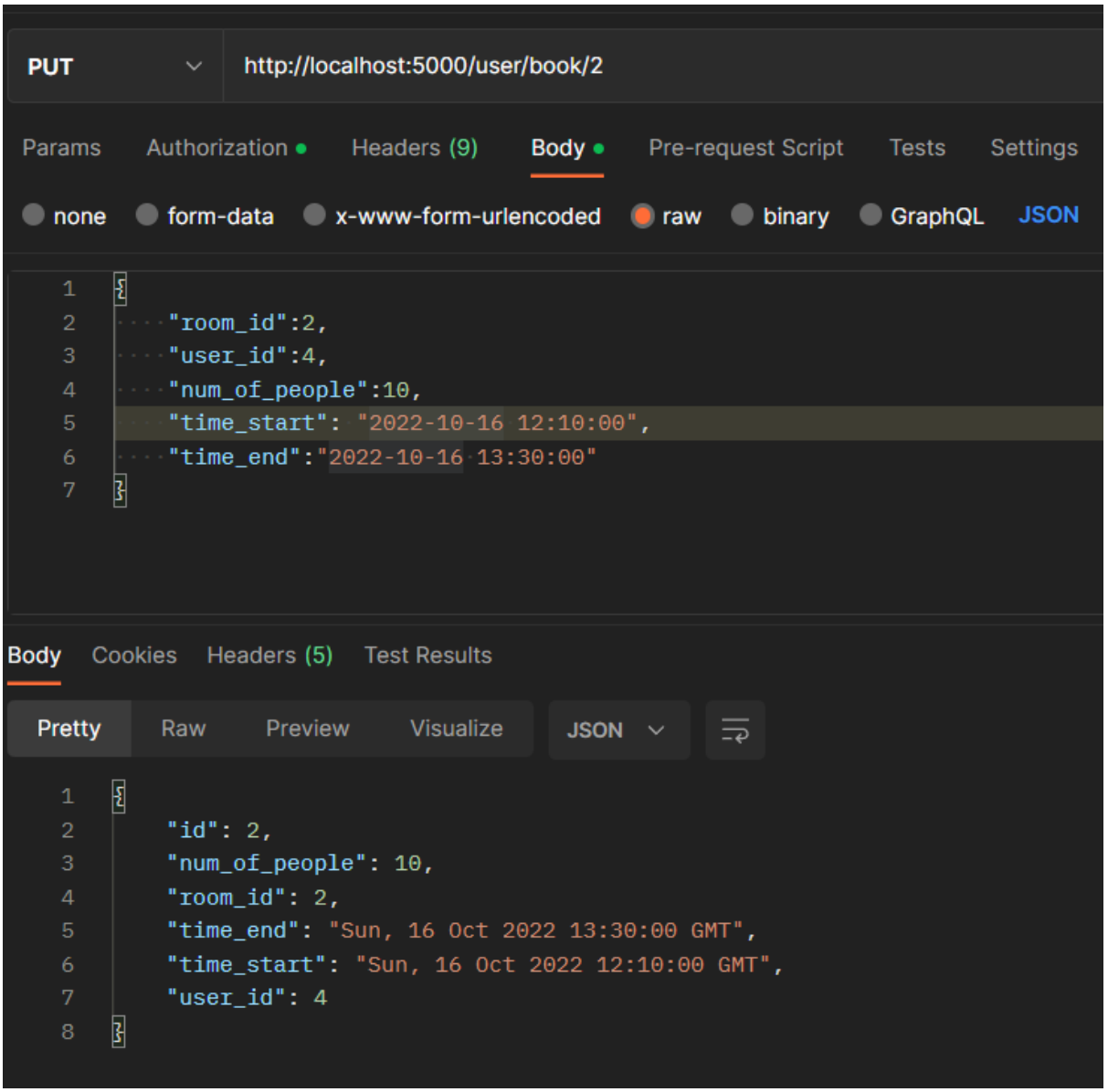


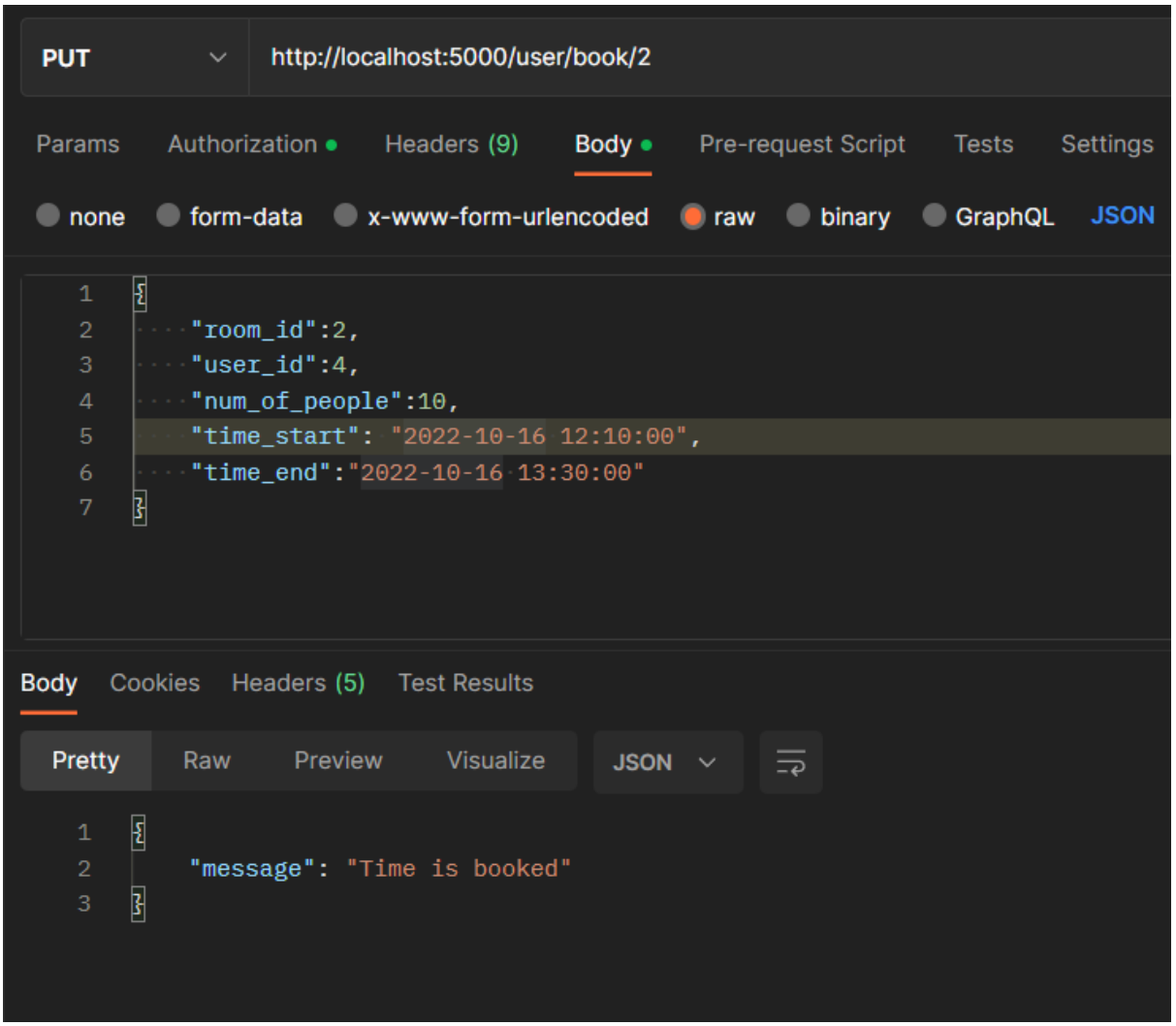
Отримуємо інформацію про бронювання:


GET
http://localhost:5000/user/book/2
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Type
Basic Auth
The authorization header will be automatically generated when you send the request. Learn more about authorization
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collab
recommend using variables. Learn more about variables
Username
theUser1
Password
Show Password
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 210 ms
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"num_of_people": 10,
"room_id": null,
"time_end": "Mon, 15 Aug 2022 13:30:00 GMT",
"time_start": "Mon, 15 Aug 2022 12:10:00 GMT",
"user_id": 2
}



Оновлюємо бронювання:


PUT
http://localhost:5000/user/book/2
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"room_id":2,
"user_id":4,
"num_of_people":10,
"time_start": "2022-10-16 12:10:00",
"time_end":"2022-10-16 13:30:00"
}
Body
Cookies
Headers (5)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"num_of_people": 10,
"room_id": 2,
"time_end": "Sun, 16 Oct 2022 13:30:00 GMT",
"time_start": "Sun, 16 Oct 2022 12:10:00 GMT",
"user_id": 4
}



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
```

```yaml
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:
```

```yaml
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
```