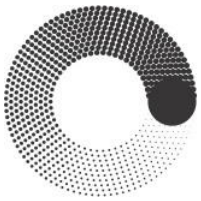
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

Лабораторная №9

Дисциплина: Backend-разработка

Тема: SQLAlchemy

Выполнил(а): студент(ка) группы 221-3711

Морозов К.А.
(Фамилия И.О.)

Дата, подпись _____
(Дата) (Подпись)

Проверил: _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания: _____

Москва

2024

Лабораторная №9
«SQLAlchemy»

Ссылка на git:

Цель работы:

Освоить основные принципы работы с базой данных через SQLAlchemy: подключение к базе данных, создание таблиц, выполнение запросов и интеграция с веб-приложением.

Задание:

Часть 1: Подключение к базе данных и создание таблиц

Выбор базы данных:
Выберите одну из баз данных: MSSQL, SQLite, PostgreSQL, MySQL.
Установите необходимые библиотеки для работы с выбранной базой данных и SQLAlchemy.
Создание модели данных:
Опишите модель данных, состоящую из двух таблиц: Users и Posts.
Таблица Users должна содержать следующие поля:
id (целое число, первичный ключ, автоинкремент)
username (строка, уникальное значение)
email (строка, уникальное значение)
password (строка)
Таблица Posts должна содержать следующие поля:
id (целое число, первичный ключ, автоинкремент)
title (строка)
content (текст)
user_id (целое число, внешний ключ, ссылающийся на поле id таблицы Users)
Создание таблиц:
Напишите программу на Python, которая подключается к выбранной базе данных и создает таблицы Users и Posts на основе описанной модели данных.

Часть 2: Взаимодействие с базой данных

Добавление данных:
Напишите программу, которая добавляет в таблицу Users несколько записей с разными значениями полей username, email и password.
Напишите программу, которая добавляет в таблицу Posts несколько записей, связанных с пользователями из таблицы Users.
Извлечение данных:
Напишите программу, которая извлекает все записи из таблицы Users.
Напишите программу, которая извлекает все записи из таблицы Posts, включая информацию о пользователях, которые их создали.

Напишите программу, которая извлекает записи из таблицы Posts, созданные конкретным пользователем.
Обновление данных:
Напишите программу, которая обновляет поле email у одного из пользователей.
Напишите программу, которая обновляет поле content у одного из постов.
Удаление данных:
Напишите программу, которая удаляет один из постов.
Напишите программу, которая удаляет пользователя и все его посты.

Часть 3: Базовые операции с базой данных в веб-приложении

Создание веб-приложения:
Создайте простое веб-приложение на FastAPI.
Интегрируйте SQLAlchemy в ваше веб-приложение.
Реализация CRUD-операций:
Реализуйте веб-страницы для выполнения CRUD-операций (создание, чтение, обновление, удаление) с записями в таблицах Users и Posts.
Страницы должны включать:
Форму для создания нового пользователя/поста.
Список всех пользователей/постов с возможностью редактирования и удаления.
Страницу для редактирования информации о пользователе/посте.

```python
from sqlalchemy import create_engine, Column, Integer, String, Text, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from sqlalchemy.orm import Session, relationship

from pydantic import BaseModel

from fastapi import FastAPI, Depends, HTTPException

from typing import List
from typing import Optional

app = FastAPI()

# Database SQLite
DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

#Модель пользователя
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    email = Column(String, unique=True, index=True)
    password = Column(String, nullable=False)
    posts = relationship("Post", back_populates="user", cascade="all, delete")

#Модель поста
class Post(Base):
    __tablename__ = "posts"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, nullable=False)
    content = Column(Text, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    user = relationship("User", back_populates="posts")


#Создание таблицы в базе данных
Base.metadata.create_all(bind=engine)


#Получение сессии базы данных
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
```

```
        db.close()


class UserCreate(BaseModel):
    name: str
    email: str
    password: str

class UserResponse(BaseModel):
    id: int
    name: str
    email: str
    password: str

class PostCreate(BaseModel):
    title: str
    content: str
    user_id: int

class PostResponse(BaseModel):
    id: int
    title: str
    content: str
    user_id: int
    user: UserResponse

    class Config:
        orm_mode = True


@app.post("/users/", response_model=UserResponse)
def create_user(user: UserCreate, db: Session = Depends(get_db)):
    db_user = User(name=user.name, email=user.email, password=user.password)
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

@app.get("/users/", response_model=List[UserResponse])
def read_users(skip: int = 0, Limit: int = 10, db: Session = Depends(get_db)):
    users = db.query(User).offset(skip).limit(Limit).all()
    return users

@app.post("/posts/", response_model=PostResponse)
def create_post(post: PostCreate, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.id == post.user_id).first()
    if not db_user:
        raise HTTPException(status_code=404, detail="User not found")
    db_post = Post(title=post.title, content=post.content,
user_id=post.user_id)
    db.add(db_post)
```

```
    db.commit()
    db.refresh(db_post)
    return db_post

@app.get("/posts/", response_model=List[PostResponse])
def read_posts(skip: int = 0, limit: int = 10, db: Session = Depends(get_db)):
    posts = db.query(Post).offset(skip).limit(limit).all()
    return posts

@app.get("/posts/user/{user_id}", response_model=List[PostResponse])
def read_posts_by_user(user_id: int, db: Session = Depends(get_db)):
    posts = db.query(Post).filter(Post.user_id == user_id).all()
    return posts

@app.put("/posts/{post_id}", response_model=PostResponse)
def update_post(post_id: int, post: PostCreate, db: Session = Depends(get_db)):
    db_post = db.query(Post).filter(Post.id == post_id).first()
    if not db_post:
        raise HTTPException(status_code=404, detail="Post not found")
    db_post.title = post.title
    db_post.content = post.content
    db.commit()
    db.refresh(db_post)
    return db_post

@app.delete("/posts/{post_id}", response_model=PostResponse)
def delete_post(post_id: int, db: Session = Depends(get_db)):
    db_post = db.query(Post).filter(Post.id == post_id).first()
    if not db_post:
        raise HTTPException(status_code=404, detail="Post not found")
    db.delete(db_post)
    db.commit()
    return db_post


@app.get("/users/{user_id}", response_model=UserResponse)
def read_user(user_id: int, db: Session = Depends(get_db)):
    user = db.query(User).filter(User.id == user_id).first()
    if user is None:
        raise HTTPException(status_code=404, detail="User not found")
    return user


class UserUpdate(BaseModel):
    name: Optional[str] = None
    email: Optional[str] = None


@app.put("/users/{user_id}", response_model=UserResponse)
def update_user(user_id: int, user: UserUpdate, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.id == user_id).first()
```

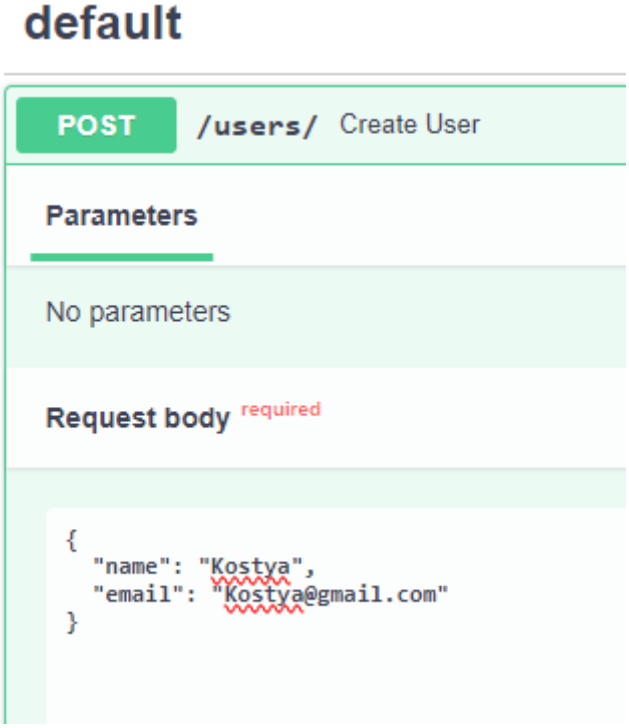
```python
    if db_user is None:
        raise HTTPException(status_code=404, detail= "User not found")

    db_user.name = user.name if user.name is not None else db_user.name
    db_user.email = user.email if user.email is not None else db_user.email
    db.commit()
    db.refresh(db_user)
    return db_user

@app.delete("/users/{user_id}", response_model=UserResponse)
def delete_user(user_id: int, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.id == user_id).first()

    if db_user is None:
        raise HTTPException(status_code=404, detail= "User not found")

    db.delete(db_user)
    db.commit()
    return db_user
```



Server response

Code	Details
200	Response body

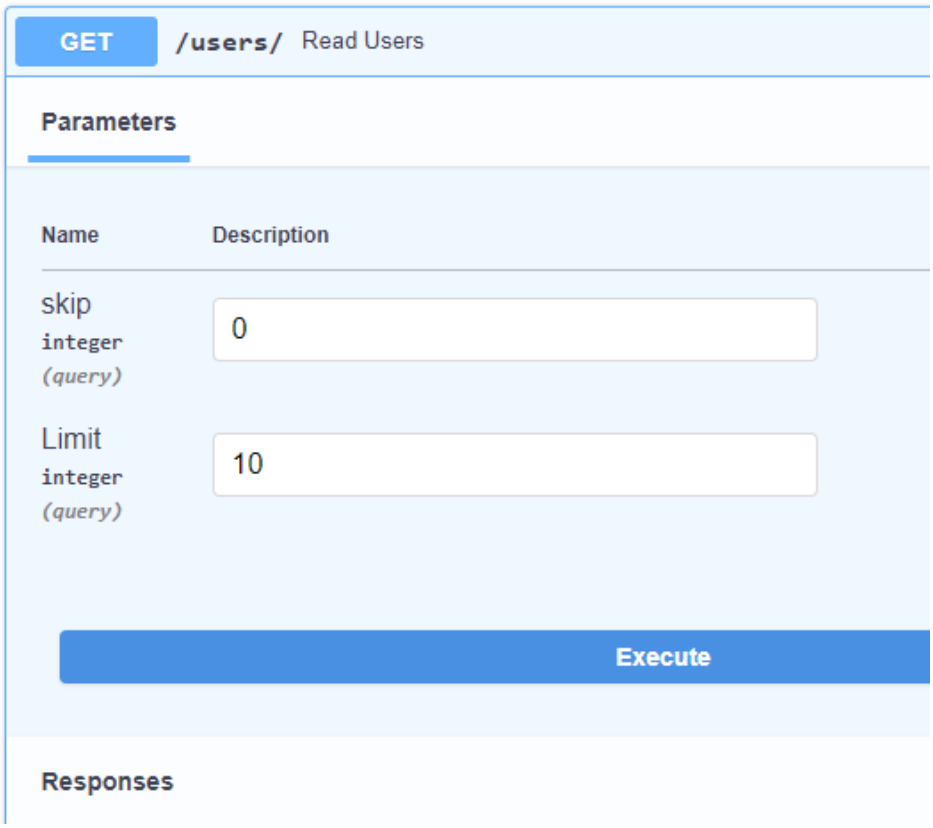
```
{
  "id": 1,
  "name": "Kostya",
  "email": "Kostya@gmail.com"
}
```

Response headers

```
content-length: 51
content-type: application/json
date: Sat,28 Dec 2024 06:52:24 GMT
server: uvicorn
```

Responses

Code	Description
200	Successful Response



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/users/?skip=0&Limit=10' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/users/?skip=0&Limit=10
```

Server response

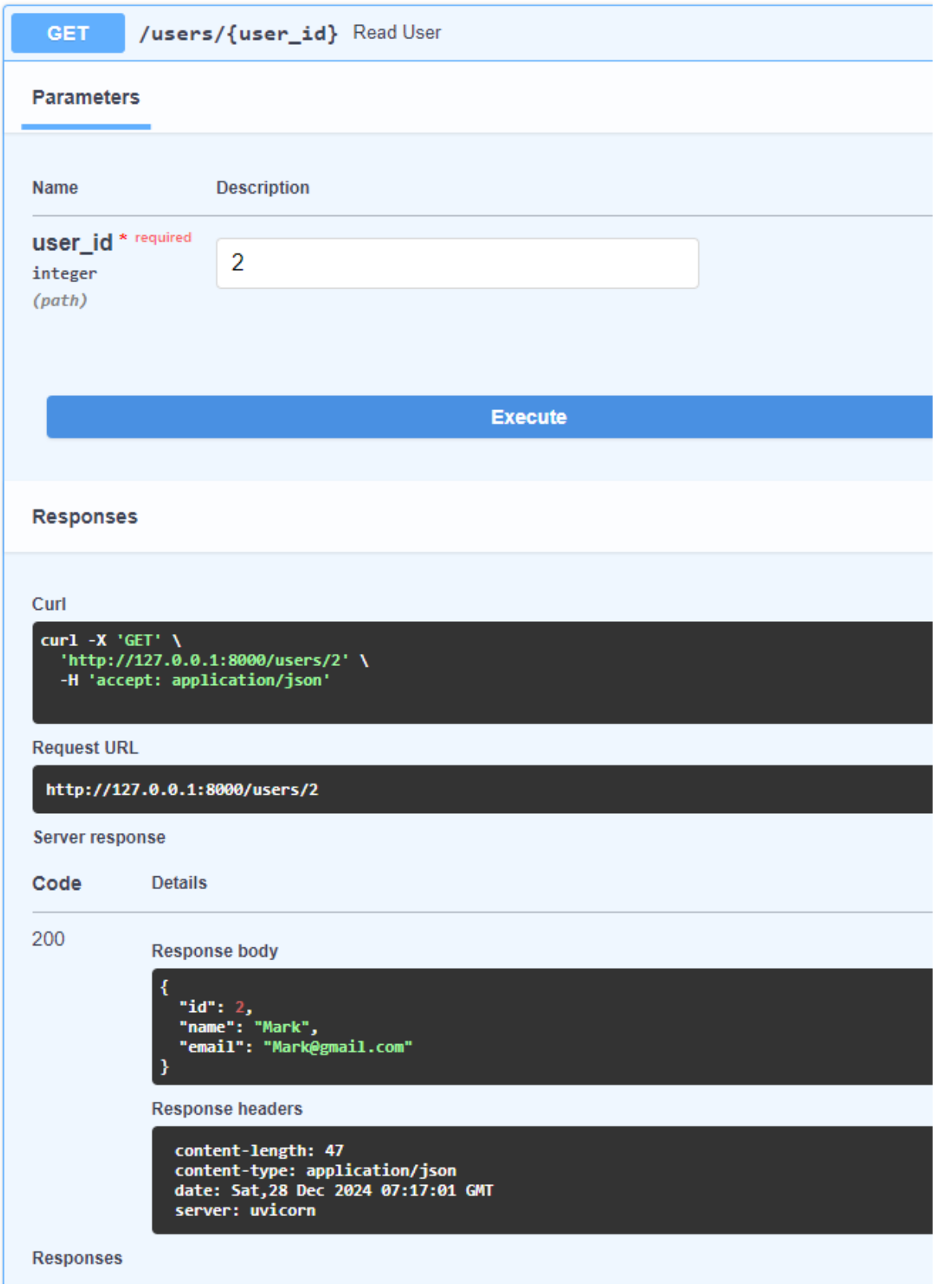
Code	Details
200	Response body

```
[
  {
    "id": 1,
    "name": "Kostya",
    "email": "Kostya@gmail.com"
  },
  {
    "id": 2,
    "name": "Mark",
    "email": "Mark@gmail.com"
  }
]
```

Response headers

```
content-length: 101
content-type: application/json
date: Sat,28 Dec 2024 07:14:24 GMT
server: uvicorn
```

Responses


GET
/users/{user_id} Read User
Parameters
Name
Description
user_id * required
integer
(path)
2
Execute
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/users/2' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/users/2
Server response
Code
Details
200
Response body
{
"id": 2,
"name": "Mark",
"email": "Mark@gmail.com"
}
Response headers
content-length: 47
content-type: application/json
date: Sat,28 Dec 2024 07:17:01 GMT
server: uvicorn
Responses

PUT /users/{user_id} Update User

Parameters

Name	Description
user_id * required integer *(path)*	2

Request body required

```
{
  "name": "Mark",
  "email": "Mark777@gmail.com"
}
```

Responses

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/users/2' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Mark",
  "email": "Mark777@gmail.com"
}'
```

Request URL

```
http://127.0.0.1:8000/users/2
```

Server response

Code	Details
200	Response body

```
{
  "id": 2,
  "name": "Mark",
  "email": "Mark777@gmail.com"
}
```

Request URL

```
http://127.0.0.1:8000/users/?skip=0&Limit=10
```

Server response

Code	Details
200	Response body

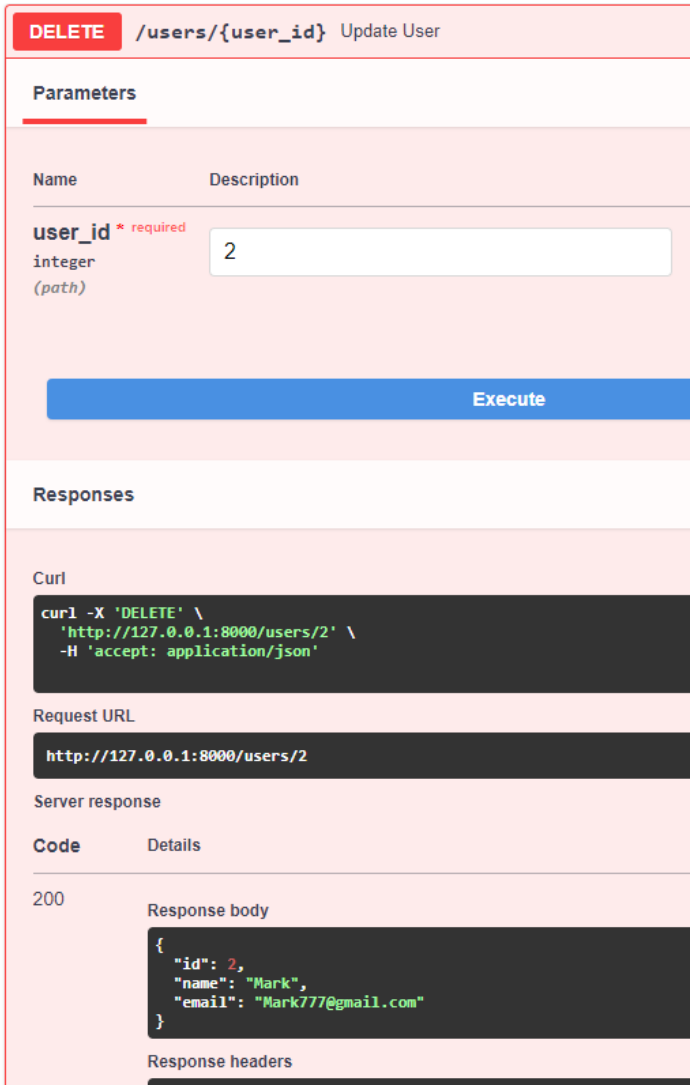
```
[
  {
    "id": 1,
    "name": "Kostya",
    "email": "Kostya@gmail.com"
  },
  {
    "id": 2,
    "name": "Mark",
    "email": "Mark777@gmail.com"
  }
]
```

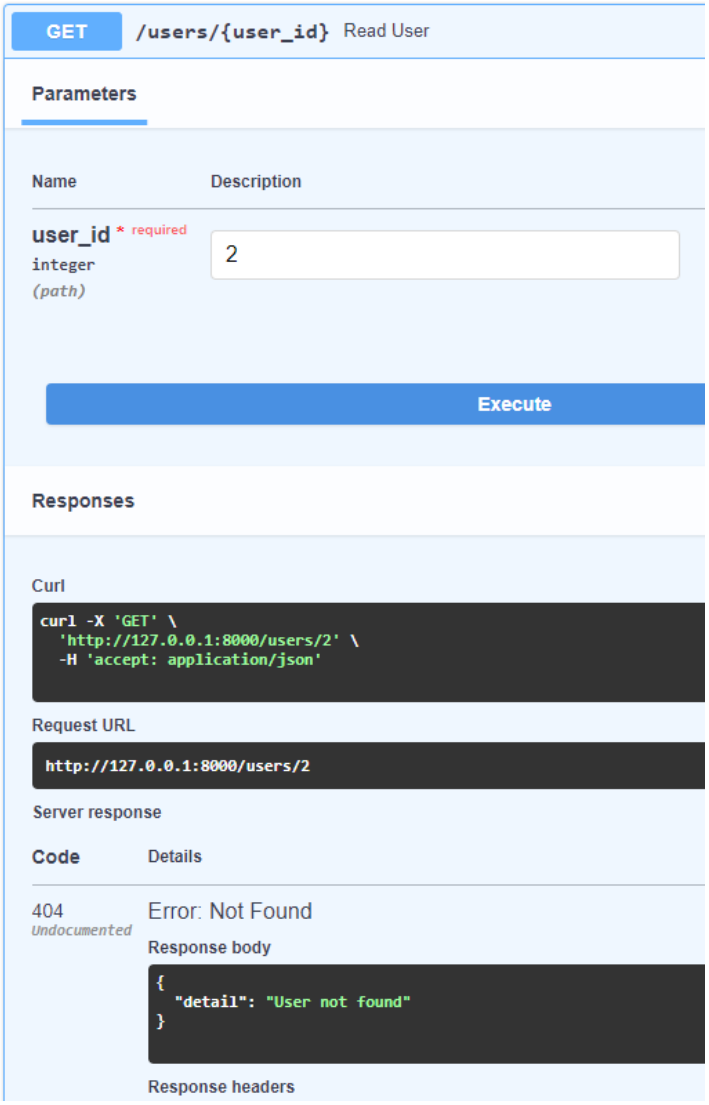
Response headers

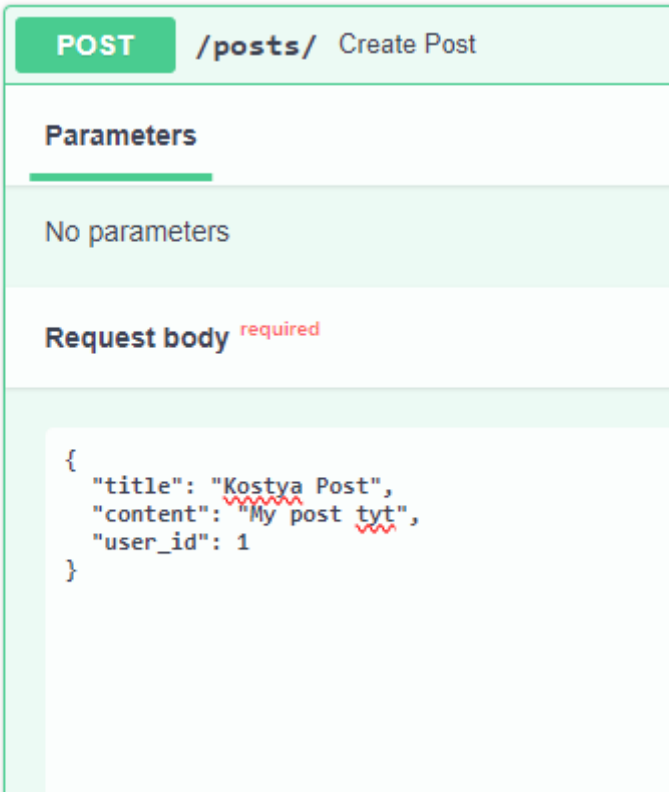
```
content-length: 104
content-type: application/json
date: Sat,28 Dec 2024 07:54:42 GMT
server: uvicorn
```

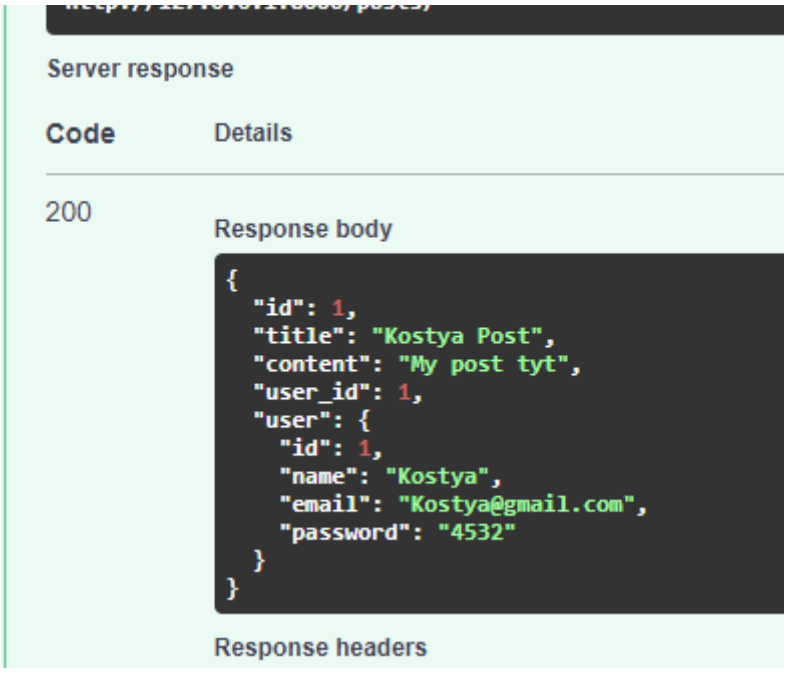
Responses

Code Description


DELETE
/users/{user_id}
Update User
Parameters
Name
Description
user_id * required
integer
(path)
2
Execute
Responses
Curl
curl -X 'DELETE' \
'http://127.0.0.1:8000/users/2' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/users/2
Server response
Code
Details
200
Response body
{
"id": 2,
"name": "Mark",
"email": "Mark777@gmail.com"
}
Response headers


GET
/users/{user_id} Read User
Parameters
Name
Description
user_id * required
integer
(path)
2
Execute
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/users/2' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/users/2
Server response
Code
Details
404
Undocumented
Error: Not Found
Response body
{
"detail": "User not found"
}
Response headers


POST
/posts/ Create Post
Parameters
No parameters
Request body required
{
"title": "Kostya Post",
"content": "My post tyt",
"user_id": 1
}


Server response
Code
Details
200
Response body
{
"id": 1,
"title": "Kostya Post",
"content": "My post tyt",
"user_id": 1,
"user": {
"id": 1,
"name": "Kostya",
"email": "Kostya@gmail.com",
"password": "4532"
}
}
Response headers