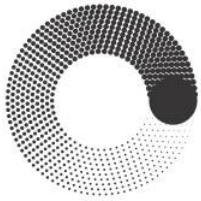
федеральное государственное автономное образовательное учреждение высшего образования



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ)
(Факультет информационных технологий)

(Институт Принтмедиа и информационных технологий) Кафедра Информатики и информационных технологий

направление подготовки 09.03.02
«Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 9

Дисциплина: Backend-разработка (Часть 2 - Python).

Тема: Работа с базой данных через SQLAlchemy.

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

Мироненко Р. Е.

(Фамилия И.О.)

Дата, подпись 09.12.2024 _____

(Дата) (Подпись)

Проверил: _____ _____

(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____

(Дата) (Подпись)

Замечания: _____
_____
_____

Москва 2024

Гит: https://github.com/Mospolytech-IIT/lab9-Roman784.git

```python
'''main.py'''


from contextlib import asynccontextmanager
from typing import Annotated
from fastapi import FastAPI, Depends
import uvicorn
from database import create_tables, delete_tables
from repository import User, Post, UserRepository, PostRepository


@asynccontextmanager
async def lifespan(app: FastAPI):
    '''Жизненный цикл приложения. Обновляет бд'''
    await delete_tables()
    await create_tables()
    yield


app = FastAPI(lifespan=lifespan)


# CRUD для работы с пользователем
@app.post("/create-user")
async def create_user(user: Annotated[User, Depends()]):
    '''Создаёт пользователя'''
    user_id = await UserRepository.add_one(data=user)
    return {"id": user_id}

@app.get("/get-all-users")
async def get_all_users():
    '''Возвращает всех пользователей'''
    users = await UserRepository.get_all()
    return users

@app.put("/update-user-email")
async def update_user_email(user_id: int, new_email: str):
    '''Обновляет данные пользователя'''
    await UserRepository.update_email(user_id=user_id, new_email=new_email)

@app.delete("/delete-user")
async def delete(user_id: int):
    '''Удаляет пользоватлея вместе с его постами'''
    await UserRepository.delete(user_id=user_id)


# CRUD для работы с постами
```

```python
@app.post("/create-post")
async def create_post(post: Annotated[Post, Depends()]):
    '''Создаёт пост'''
    post_id = await PostRepository.add_one(data=post)
    return {"id": post_id}

@app.get("/get-all-posts")
async def get_all_posts():
    '''Возвращает все посты'''
    posts = await PostRepository.get_all()
    return posts

@app.get("/get-posts")
async def get_posts(user_id: int):
    '''Возвращает все посты конкретного пользователя'''
    posts = await PostRepository.get_posts(user_id=user_id)
    return posts

@app.put("/update-post-content")
async def update_post_content(post_id: int, new_content: str):
    '''Обновляет content поста'''
    await PostRepository.update_content(post_id=post_id, new_content=new_content)

@app.delete("/delete-post")
async def delete_post(post_id: int):
    '''Удаляет пост'''
    await PostRepository.delete(post_id=post_id)


if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

```python
'''database.py'''


from sqlalchemy.ext.asyncio import create_async_engine, async_sessionmaker
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship, DeclarativeBase, Mapped, mapped_column


class BaseTable(DeclarativeBase):
    '''База для всех таблиц'''
    pass

class UserTable(BaseTable):
    '''Таблица пользователя'''
    __tablename__ = 'users'
    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    username: Mapped[str]
    email: Mapped[str]
```

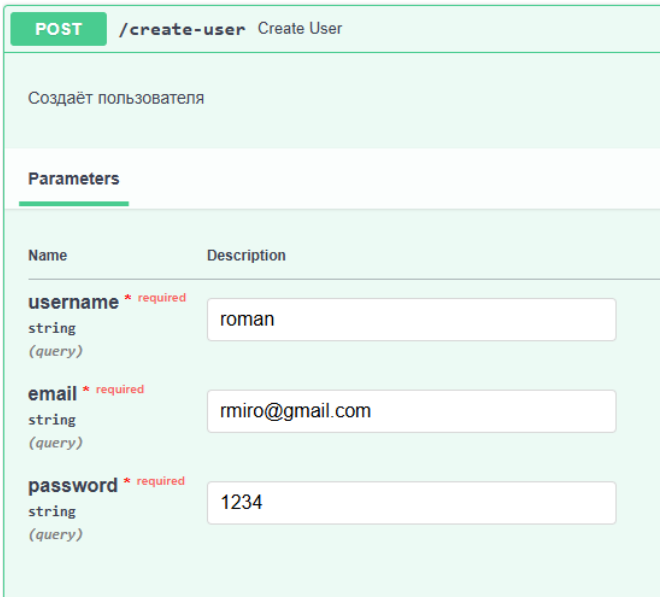
```
    password: Mapped[str]
    posts: Mapped[list['PostTable']] = relationship('PostTable', back_populates='user', cascade="all, delete-orphan")

class PostTable(BaseTable):
    '''Таблица постов'''
    __tablename__ = 'posts'
    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    title: Mapped[str]
    content: Mapped[str]
    user_id: Mapped[int] = mapped_column(ForeignKey('users.id'))
    user: Mapped['UserTable'] = relationship('UserTable', back_populates='posts')


engine = create_async_engine('sqlite+aiosqlite:///database.db')
new_session = async_sessionmaker(bind=engine, expire_on_commit=False)


async def create_tables():
    '''Создаёт таблицы'''
    async with engine.begin() as conn:
        await conn.run_sync(BaseTable.metadata.create_all)
        print("create")

async def delete_tables():
    '''Удаляет таблицы'''
    async with engine.begin() as conn:
        await conn.run_sync(BaseTable.metadata.drop_all)
        print("delete")
```

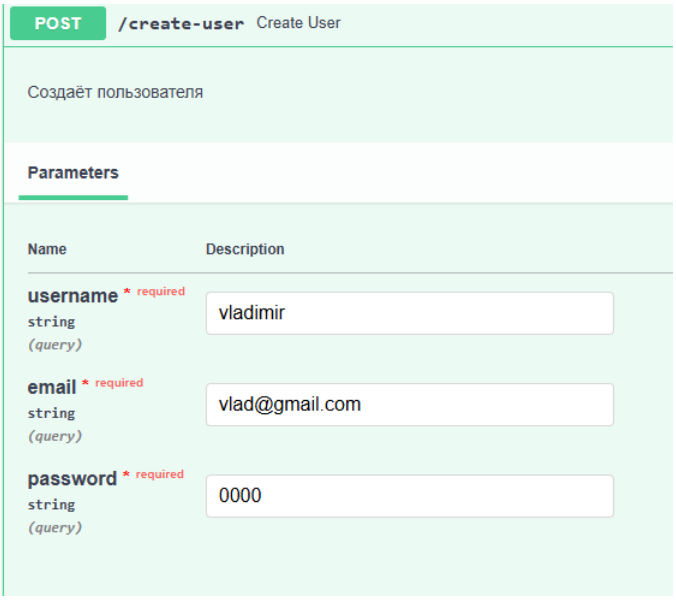
```
'''Функции для работы с бд'''


from sqlalchemy import select, update, delete
from sqlalchemy.orm import joinedload
from pydantic import BaseModel
from database import new_session, UserTable, PostTable


class User(BaseModel):
    '''Модель пользователя'''
    username: str
    email: str
    password: str

class Post(BaseModel):
    '''Модель постов'''
    title: str
    content: str
    user_id: int
```

```python
class UserRepository:
    '''Методы для работы с бд пользователей'''
    @classmethod
    async def add_one(cls, data: User) -> int:
        '''Добавляет пользователя'''
        async with new_session() as session:

            user = UserTable(
                username=data.username,
                email=data.email,
                password=data.password
            )

            session.add(user)
            await session.flush()
            await session.commit()
            return user.id

    @classmethod
    async def get_all(cls):
        '''Возвращает данные обо всех пользователях'''
        async with new_session() as session:
            query = select(UserTable)
            result = await session.execute(query)
            user_models = result.scalars().all()
            return user_models

    @classmethod
    async def update_email(cls, user_id: int, new_email: str):
        '''Обновляет email пользователя'''
        async with new_session() as session:
            query = update(UserTable).where(UserTable.id == user_id).values(email=new_email)
            await session.execute(query)
            await session.commit()

    @classmethod
    async def delete(cls, user_id: int):
        '''Удаляет пользователя и все его посты'''
        async with new_session() as session:
            query = (
                select(UserTable)
                .where(UserTable.id == user_id)
                .options(joinedload(UserTable.posts)))
            user = await session.scalar(query)
            await session.delete(user)
            await session.commit()


class PostRepository:
```

```python
    '''Методы для работы с бд постов'''
    @classmethod
    async def add_one(cls, data: Post) -> int:
        '''Добавляет пост'''
        async with new_session() as session:

            post = PostTable(
                title=data.title,
                content=data.content,
                user_id=data.user_id
            )

            session.add(post)
            await session.flush()
            await session.commit()
            return post.id

    @classmethod
    async def get_all(cls):
        '''Возвращает данные обо всех постах вместе с их пользователями'''
        async with new_session() as session:
            query = select(PostTable).options(joinedload(PostTable.user))
            result = await session.execute(query)
            post_models = result.scalars().all()
            return post_models

    @classmethod
    async def get_posts(cls, user_id: int):
        '''Возвращает все посты конкретного пользователя'''
        async with new_session() as session:
            query = select(PostTable).where(PostTable.user_id == user_id)
            result = await session.execute(query)
            post_models = result.scalars().all()
            return post_models

    @classmethod
    async def update_content(cls, post_id: int, new_content: str):
        '''Обновляет content пользователя'''
        async with new_session() as session:
            query = update(PostTable).where(PostTable.id == post_id).values(content=new_content)
            await session.execute(query)
            await session.commit()

    @classmethod
    async def delete(cls, post_id: int):
        '''Удаляет пост'''
        async with new_session() as session:
            query = delete(PostTable).where(PostTable.id == post_id)
            await session.execute(query)
            await session.commit()
```

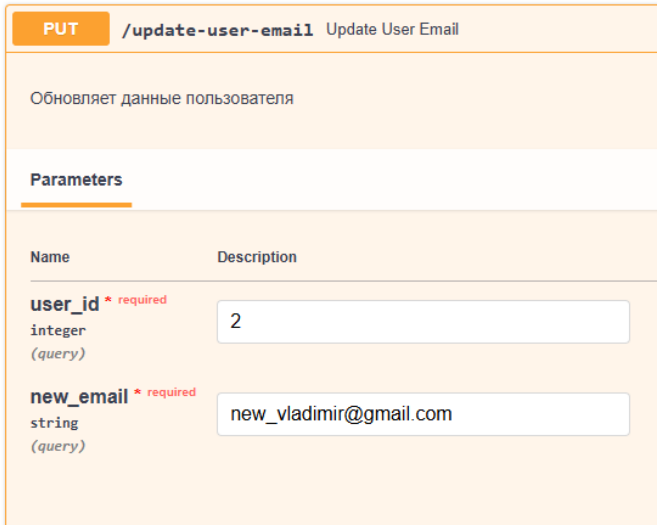
Создание пользователей.

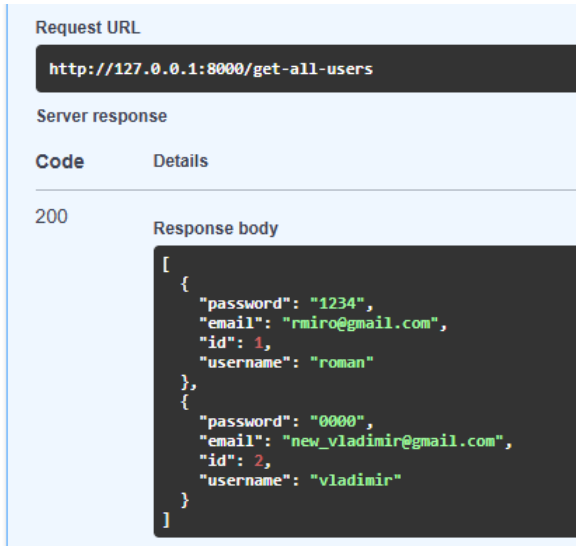




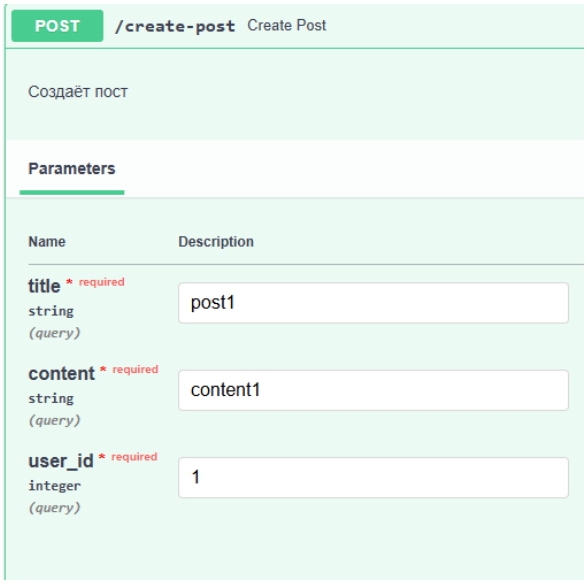
Получение всех пользователей.

Изменение почты.





Создание постов.



....

Получение всех постов с их пользователями.

Получение всех постов 1го пользователя.

Request URL

```
http://127.0.0.1:8000/get-posts?user_id=1
```

Server response

Code	Details
200	Response body


```
[
  {
    "user_id": 1,
    "content": "content1",
    "title": "post1",
    "id": 1
  },
  {
    "user_id": 1,
    "content": "content2",
    "title": "post2",
    "id": 2
  },
  {
    "user_id": 1,
    "content": "content3",
    "title": "post3",
    "id": 3
  }
]
```


Download

Обновление контента у поста.

PUT /update-post-content Update Post Content

Обновляет content поста

Parameters

Name	Description
post_id * required integer (query)	1
new_content * required string (query)	new post content


```
{
  "user_id": 1,
  "content": "new post content",
  "title": "post1",
  "id": 1
},
```


Удаление поста.

DELETE /delete-post Delete Post

Удаляет пост

Parameters

Name	Description
post_id * required integer (query)	1

Request URL

```
http://127.0.0.1:8000/get-all-posts
```

Server response

Code	Details
200	Response body


```
[
  {
    "user_id": 1,
    "content": "content2",
    "title": "post2",
    "id": 2,
    "user": {
      "password": "1234",
      "email": "rmiro@gmail.com",
      "id": 1,
      "username": "roman"
    }
  },
  {
    "user_id": 1,
    "content": "content3",
    "title": "post3",
    "id": 3,
    "user": {
      "password": "1234",
      "email": "rmiro@gmail.com",
      "id": 1,
      "username": "roman"
    }
  },
  {
    "user_id": 2,
    "content": "content4",
```

Удаление пользователя с его постами.