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 РАБОТА № _9_

Дисциплина: Backend-разработка

Тема: Работа с базой данных через SQLAlchemy

Выполнил(а): студент(ка) группы __221-379__

_____Кодиров Жамшид Мурод угли_____
(Фамилия И.О.)

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

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания:_____

_____

_____

Москва

2025

Задание

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

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.**Ход работы**

1. **Создание проекта FastAPI**
   Для начала было создано приложение с использованием фреймворка FastAPI. В проекте был инициализирован файл `main.py`, который является основным файлом для обработки запросов. Для рендеринга HTML-шаблонов был использован Jinja2, а для работы с базой данных — SQLAlchemy. В проекте используется SQLite в качестве базы данных.

   Установлены необходимые зависимости:

   ```
   pip install fastapi uvicorn jinja2 sqlalchemy
   ```

2. **Создание моделей с использованием SQLAlchemy**
   В проекте были созданы две основные модели:
   - **User (Пользователь)**: модель, которая описывает пользователя системы с полями: `id`, `username`, `email`, `password_hash`.
   - **Post (Пост)**: модель, которая описывает посты пользователей с полями: `id`, `title`, `content` и ссылкой на пользователя через поле `author_id`.

   Модель **Post** имеет связь "один ко многим" с моделью **User**, что означает, что каждый пользователь может иметь несколько постов. Эти связи были установлены через внешние ключи в SQLAlchemy.

   Пример модели:

   ```
   class User(Base):
       __tablename__ = 'users'

       id = Column(Integer, primary_key=True,
   index=True)
       username = Column(String, unique=True,
   index=True)
       email = Column(String, unique=True, index=True)
       password_hash = Column(String)

   class Post(Base):
       __tablename__ = 'posts'

       id = Column(Integer, primary_key=True,
   index=True)
   ```

```
    title = Column(String, index=True)
    content = Column(Text)
    author_id = Column(Integer,
ForeignKey('users.id'))
    author = relationship('User',
back_populates='posts')
```

3. **Создание шаблона index.html**
   В файле `index.html`, используя Jinja2, был создан шаблон для отображения формы создания пользователей и постов, а также для отображения списка пользователей и постов. В шаблоне использовались условные операторы и циклы для динамического отображения данных.

   Шаблон включает:

   - Формы для создания пользователя и поста.
   - Списки всех пользователей и постов с возможностью их редактирования и удаления.

   Пример шаблона для отображения списка пользователей:

```
html
КопироватьРедактировать
<h2>Все пользователи</h2>
<ul>
    {% for user in users %}
    <li>
        {{ user.username }} ({{ user.email }})
        <a href="/users/edit/{{ user.id
}}">Редактировать</a>
        <form method="post"
action="/users/delete/{{ user.id }}"
style="display:inline;">
            <button type="submit">Удалить</button>
        </form>
    </li>
    {% endfor %}
</ul>
```

4. **Обработка маршрутов в FastAPI**
   В файле `main.py` были реализованы следующие маршруты:
   - **GET /**: отобразить главную страницу с формами для создания пользователей и постов, а также с перечнем всех пользователей и постов.
   - **POST /users/create**: обработка создания нового пользователя.

- **POST /posts/create**: обработка создания нового поста.
- **POST /users/delete/{id}**: обработка удаления пользователя по его `id`.
- **POST /posts/delete/{id}**: обработка удаления поста по его `id`.
- **GET /users/edit/{id}, POST /users/edit/{id}**: маршруты для редактирования информации о пользователе.
- **GET /posts/edit/{id}, POST /posts/edit/{id}**: маршруты для редактирования поста.

Пример маршрута для создания пользователя:

```
КопироватьРедактировать
@app.post("/users/create")
async def create_user(username: str = Form(...),
email: str = Form(...), password: str = Form(...)):
    hashed_password = get_password_hash(password)
    db_user = User(username=username, email=email,
password_hash=hashed_password)
    db.add(db_user)
    db.commit()
    return RedirectResponse(url='/',
status_code=303)
```

5. **Обработка ошибок**
   В процессе работы с базой данных были добавлены проверки на существование пользователя при создании поста. Также было предусмотрено исключение ошибок при редактировании и удалении записей. Например, при удалении поста или пользователя проверяется, существует ли объект в базе данных.

   Пример:

```
user = db.query(User).filter(User.id ==
user_id).first()
if user is None:
    raise HTTPException(status_code=404,
detail="User not found")
```

6. **Использование Bootstrap для улучшения внешнего вида**
   Для улучшения визуального восприятия и улучшения интерфейса был использован фреймворк Bootstrap 5. Это позволило сделать формы и списки более структурированными и удобными для пользователя. Все формы были оформлены с использованием классов Bootstrap, таких как:
   - `.form-control` для полей ввода.

- `.btn` для кнопок.
- `.card` для контейнеров с контентом.

Пример оформления формы:

```
<form action="/users/create" method="post"
class="card p-4">
    <input name="username" placeholder="Username"
required class="form-control mb-2">
    <input name="email" placeholder="Email"
required class="form-control mb-2">
    <input name="password" placeholder="Password"
required class="form-control mb-2">
    <button type="submit" class="btn btn-
primary">Создать</button>
</form>
```

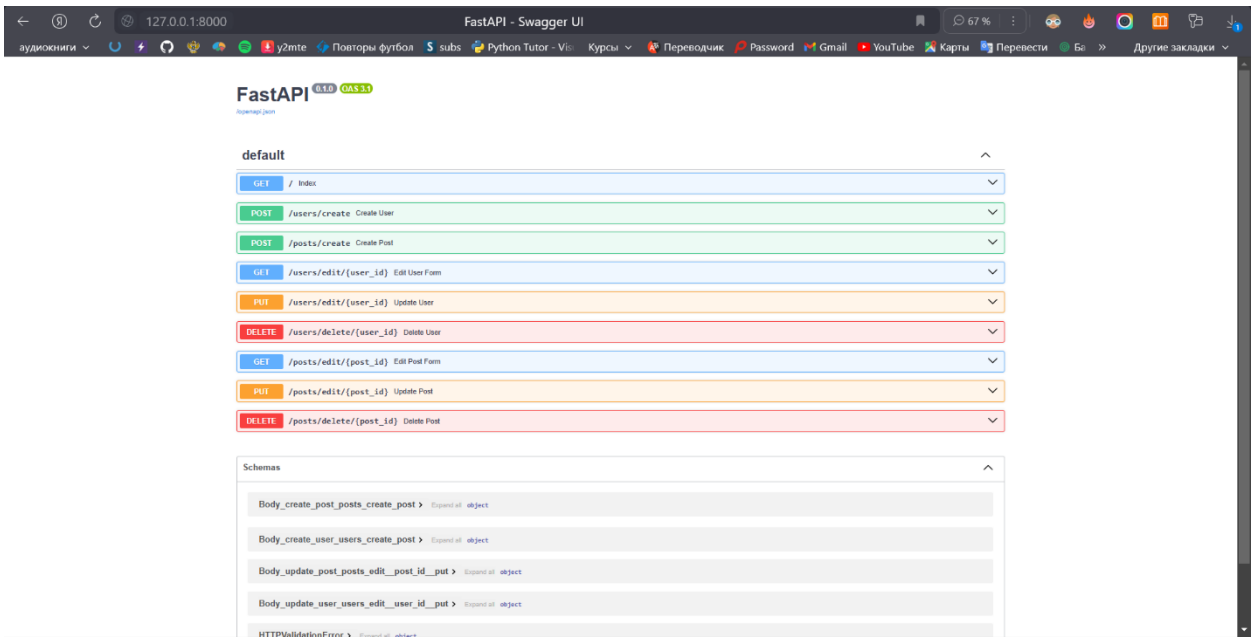
7. **Запуск приложения**
   После завершения разработки приложение было запущено с использованием сервера Uvicorn. Приложение стало доступно по адресу http://127.0.0.1:8000. При запуске приложение автоматически подключается к базе данных и запускает сервер для обработки запросов.

   Для запуска приложения была использована команда:

```
uvicorn main:app --reload
```

Вывод

В ходе выполнения лабораторной работы было разработано веб-приложение с использованием FastAPI, которое реализует все операции CRUD для пользователей и постов. Приложение поддерживает создание, редактирование и удаление пользователей и постов, а также выводит их в удобном для пользователя виде с помощью шаблонов Jinja2 и стилей Bootstrap.



Ссылка на Github: https://github.com/QodirovJM/BackendPython