**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»**

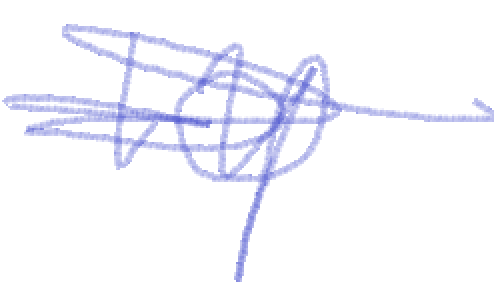
**ЛАБОРАТОРНАЯ РАБОТА № 9**

**Дисциплина: Back-end разработка (Python)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Тема: Работа с базой данных через SQLAlchemy \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Грохотова Е.Д

(Фамилия И.О.) 

**Дата, подпись 06.12.2024 *\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_***

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

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Замечания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва2024**

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.

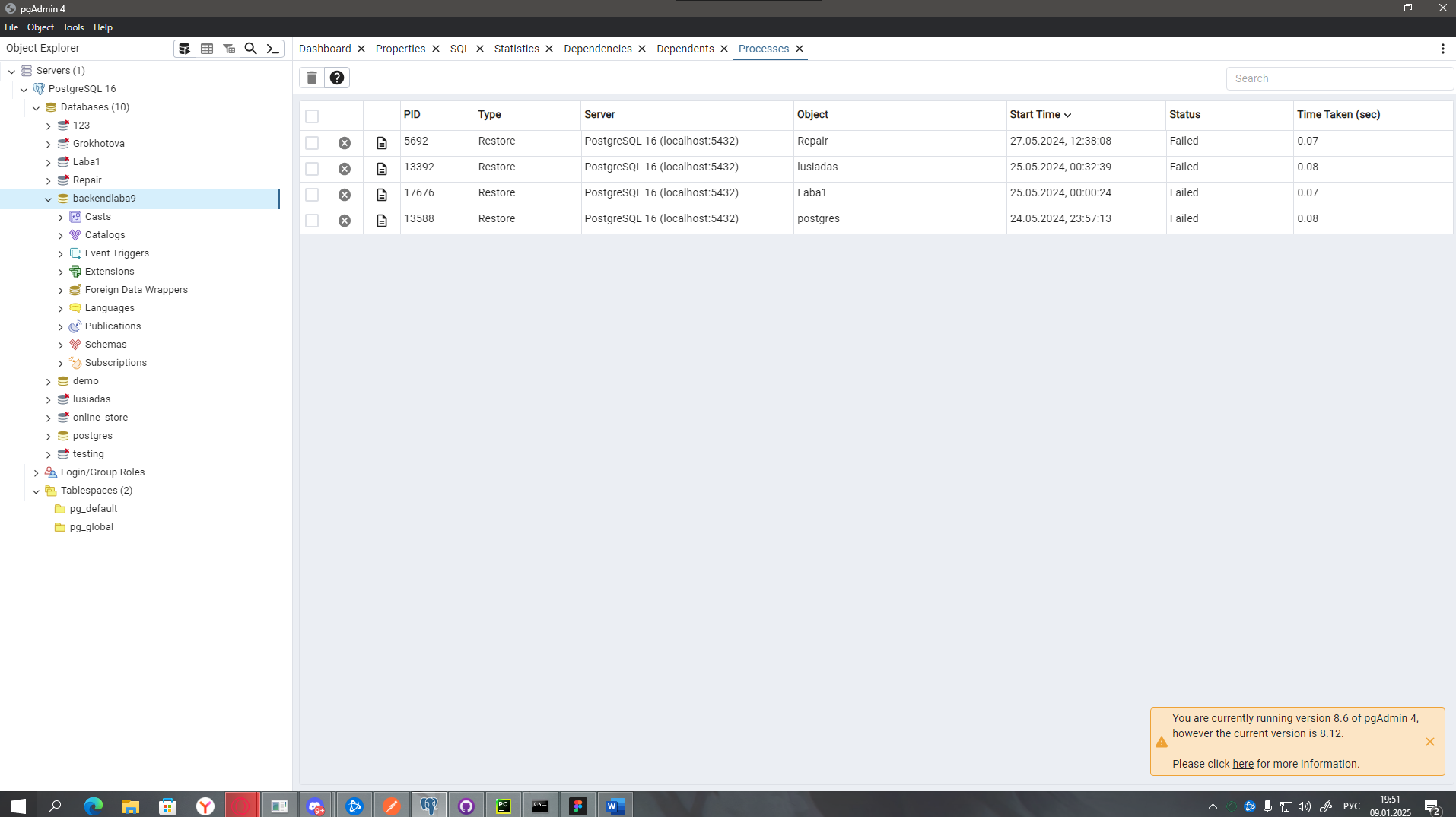
Список всех пользователей/постов с возможностью редактирования и удаления.

Страницу для редактирования информации о пользователе/посте.

Ход работы:

Выбранная СУБД – PostgreSQL

Создадим БД Backendlaba9

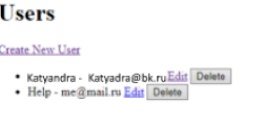


Опишем модель данных Users, Posts:



Взаимодействия с Users

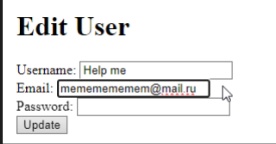
Текущие пользователи



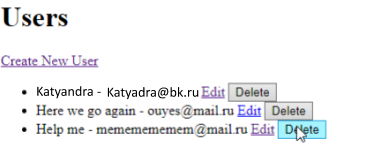
Создание нового пользователя

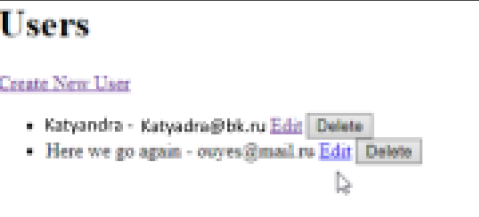


Изменение пользователя:



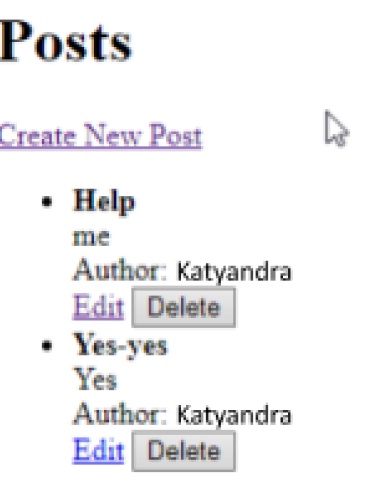
Удаление пользователя:





Взаимодействия с posts

Текущие посты:



Создадим новый пост:



Листинг кода:

Main.py:

from fastapi import FastAPI, Request, Depends, Form, HTTPException  
from fastapi.responses import HTMLResponse, RedirectResponse  
from fastapi.templating import Jinja2Templates  
from sqlalchemy.orm import Session  
from models import SessionLocal, User, Post  
  
app = FastAPI()  
templates = Jinja2Templates(directory="templates")  
  
  
def get\_db():  
 db = SessionLocal()  
 try:  
 yield db  
 finally:  
 db.close()  
  
  
@app.get("/users", response\_class=HTMLResponse)  
def get\_users\_page(request: Request, db: Session = Depends(get\_db)):  
 users = db.query(User).all()  
 return templates.TemplateResponse("users.html", {"request": request, "users":  
 users})  
  
  
@app.get("/users/new", response\_class=HTMLResponse)  
def new\_user\_page(request: Request):  
 return templates.TemplateResponse("new\_user.html", {"request": request})  
  
  
@app.post("/users/new")  
def create\_user(  
 username: str = Form(...), email: str = Form(...), password: str = Form(...), db:  
 Session = Depends(get\_db)  
):  
 user\_exists = db.query(User).filter(User.email == email).first()  
 if user\_exists:  
 raise HTTPException(status\_code=400, detail="Email already registered")  
 new\_user = User(username=username, email=email, password=password)  
 db.add(new\_user)  
 db.commit()  
 return RedirectResponse(url="/users", status\_code=303)  
  
  
@app.get("/users/edit/{user\_id}", response\_class=HTMLResponse)  
def get\_user(user\_id: int, request: Request, db: Session = Depends(get\_db)):  
 user = db.query(User).filter(User.id == user\_id).first()  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 return templates.TemplateResponse("edit\_user.html", {"request": request, "user":  
 user})  
  
  
@app.put("/users/edit/{user\_id}")  
def update\_user(  
 user\_id: int, username: str = Form(...), email: str = Form(...), password: str =  
 Form(...), db: Session = Depends(get\_db)  
):  
 user = db.query(User).filter(User.id == user\_id).first()  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 user.username = username  
 user.email = email  
 user.password = password  
 db.commit()  
 return RedirectResponse(url="/users", status\_code=303)  
  
  
@app.delete("/users/delete/{user\_id}")  
def delete\_user(user\_id: int, db: Session = Depends(get\_db)):  
 user = db.query(User).filter(User.id == user\_id).first()  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 db.delete(user)  
 db.commit()  
 return RedirectResponse(url="/users", status\_code=303)  
  
  
@app.get("/posts", response\_class=HTMLResponse)  
def get\_posts\_page(request: Request, db: Session = Depends(get\_db)):  
 posts = db.query(Post).all()  
 return templates.TemplateResponse("posts.html", {"request": request, "posts":  
 posts})  
  
  
@app.get("/posts/new", response\_class=HTMLResponse)  
def new\_post\_page(request: Request):  
 return templates.TemplateResponse("new\_post.html", {"request": request})  
  
  
@app.post("/posts/new")  
def create\_post(  
 title: str = Form(...), content: str = Form(...), user\_id: int = Form(...), db:  
 Session = Depends(get\_db)  
):  
 user = db.query(User).filter(User.id == user\_id).first()  
 if not user:  
 raise HTTPException(status\_code=400, detail="User not found")  
 new\_post = Post(title=title, content=content, user\_id=user\_id)  
 db.add(new\_post)  
 db.commit()  
 return RedirectResponse(url="/posts", status\_code=303)  
  
  
@app.get("/posts/edit/{post\_id}", response\_class=HTMLResponse)  
def get\_post(post\_id: int, request: Request, db: Session = Depends(get\_db)):  
 post = db.query(Post).filter(Post.id == post\_id).first()  
 if not post:  
 raise HTTPException(status\_code=404, detail="Post not found")  
 return templates.TemplateResponse("edit\_post.html", {"request": request, "post":  
 post})  
  
  
@app.put("/posts/edit/{post\_id}")  
def update\_post(  
 post\_id: int, title: str = Form(...), content: str = Form(...), db: Session =  
 Depends(get\_db)  
):  
 post = db.query(Post).filter(Post.id == post\_id).first()  
 if not post:  
 raise HTTPException(status\_code=404, detail="Post not found")  
 post.title = title  
 post.content = content  
 db.commit()  
 return RedirectResponse(url="/posts", status\_code=303)  
  
  
@app.delete("/posts/delete/{post\_id}")  
def delete\_post(post\_id: int, db: Session = Depends(get\_db)):  
 post = db.query(Post).filter(Post.id == post\_id).first()  
 if not post:  
 raise HTTPException(status\_code=404, detail="Post not found")  
 db.delete(post)  
 db.commit()  
 return RedirectResponse(url="/posts", status\_code=303)

Models.py:

*"""Модели данных для базы данных"""*from sqlalchemy import create\_engine, Column, Integer, String, ForeignKey, Text  
from sqlalchemy.orm import declarative\_base  
from sqlalchemy.orm import relationship, sessionmaker  
  
Base = declarative\_base()  
  
  
class User(Base):  
 \_\_tablename\_\_ = 'users'  
 id = Column(Integer, primary\_key=True, autoincrement=True)  
 username = Column(String, unique=True, nullable=False)  
 email = Column(String, unique=True, nullable=False)  
 password = Column(String, nullable=False)  
 posts = relationship('Post', back\_populates='user')  
  
  
class Post(Base):  
 \_\_tablename\_\_ = 'posts'  
 id = Column(Integer, primary\_key=True, autoincrement=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')  
  
  
DATABASE\_URL = "postgresql://postgres:123125qQW@localhost/backendlaba9"  
engine = create\_engine(DATABASE\_URL)  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Fetch\_data.py:

*"""Скрипт для извлечения данных из базы данных"""*from models import SessionLocal, User, Post  
  
  
def fetch\_all\_users():  
 *"""Выводит всех пользователей"""* db = SessionLocal()  
 try:  
 users = db.query(User).all()  
 for user in users:  
 print(f"User: {user.username}, Email: {user.email}")  
 finally:  
 db.close()  
  
  
def fetch\_posts\_with\_users():  
 *"""Выводит все посты вместе с пользователями"""* db = SessionLocal()  
 try:  
 posts = db.query(Post).all()  
 for post in posts:  
 print(f"Post: {post.title}, Author: {post.user.username}")  
 finally:  
 db.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 fetch\_all\_users()  
 fetch\_posts\_with\_users()

Add\_data.py:

*"""Скрипт для добавления данных в таблицы"""*from models import SessionLocal, User, Post  
  
  
def add\_users\_and\_posts():  
 *"""Добавляет пользователей и посты в базу данных"""* db = SessionLocal()  
 try:  
 user1 = User(username="user1", email="user1@example.com",  
 password="password1")  
 user2 = User(username="user2", email="user2@example.com",  
 password="password2")  
 db.add\_all([user1, user2])  
 db.commit()  
  
 post1 = Post(title="Post 1", content="Content of Post 1", user\_id=user1.id)  
 post2 = Post(title="Post 2", content="Content of Post 2", user\_id=user2.id)  
 db.add\_all([post1, post2])  
 db.commit()  
 finally:  
 db.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 add\_users\_and\_posts()

Ссылка на гит: https://github.com/Mospolytech-IIT/lab9-Katyadra