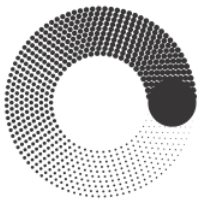
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



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 _____

Оксак Г.А.
(Фамилия И.О.)

Дата, подпись _____ _____
(Дата) (Подпись)

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

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания: _____

_____

_____

Москва

2025

Добавить нового пользователя

Имя пользователя: Maksim

Email: megabebra@gmail.com

Пароль: mega

Подтвердить

Пользователи

Создать нового пользователя

Имя пользователя		Информация	
Gleb	Публикации	Email:bebra@gmail.com Пароль:bebra	Изменить данные Удалить пользователя
Maksim	Публикации	Email:megabebra@gmail.com Пароль:mega	Изменить данные Удалить пользователя

Новая публикация

Название: publication

Содержание: aaaa ooga booga politeh.

Подтвердить

Публикации пользователя Gleb

<

Создать публикацию

Имя пользователя		Информация
publication	aaaa ooga booga politeh.	Изменить данные Удалить публикацию

main.py

```python
import asyncio
import uvicorn

def shutdown_rest_of_app(_, __):
    raise KeyboardInterrupt


def main():
    try:
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)

        web_config = uvicorn.Config(
            "app:app", #Сюда пишете ".py файл:app
            host="127.0.0.1",
            port=1404,
            loop="asyncio",
            workers=4,
        )

        web_server = uvicorn.Server(config=web_config)
        loop.create_task(web_server.serve())
        loop.run_forever()
    except KeyboardInterrupt:
        print("Caught Ctrl+C. Exiting gracefully.")

if __name__ == "__main__":
    main()
```

app.py

```
from fastapi import FastAPI, Header, Cookie, Request, HTTPException, status,
Form
from fastapi.responses import FileResponse, RedirectResponse, JSONResponse,
HTMLResponse
from fastapi.templating import Jinja2Templates

from modules.models import session, User, Post, engine

app = FastAPI()
templates = Jinja2Templates(directory="templates")
engine.connect()


# Главная страница
@app.get("/")
async def home(request: Request):
    return RedirectResponse("/users")


# === CRUD для Users ===

@app.get("/users")
async def users(request: Request):
    users = session.query(User).all()
    return templates.TemplateResponse("users.html", {"request": request,
"users": users})


@app.get("/users/new")
async def newUser(request: Request):
    return templates.TemplateResponse("UserForm.html", {"request": request,
"message": ""})


@app.post("/users/new")
async def createUser(request : Request, username: str = Form(...), email: str =
Form(...), password: str = Form(...)):
    try:
        new_user = User(username=username, email=email, password=password)
        session.add(new_user)
        session.commit()
        return RedirectResponse("/users/", status_code=303)
    except:
        session.rollback()
        return templates.TemplateResponse("UserForm.html", {"request": request,
"message": "Ошибка при добавлении, попробуйте ввести корректные данные"})



@app.get("/users/edit")
async def changeUser(request: Request, user_id: int):
    user = session.query(User).filter(User.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    return templates.TemplateResponse("UserForm.html", {"request": request,
"user": user})

@app.post("/users/edit")
async def updateUser(request:Request, user_id: int, username: str = Form(...),
email: str = Form(...), password: str = Form(...)):
    a = await request.form()
    user = a.get("user")
    try:
        user = session.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
        session.commit()
        return RedirectResponse("/users", status_code=303)
    except:
        session.rollback()
        return templates.TemplateResponse("UserForm.html", {"request": request,
"user": user})

@app.get("/users/delete")
async def deleteUser(user_id: int):
    print(user_id)
    user = session.query(User).filter(User.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    posts = session.query(Post).filter(Post.user_id == user_id).all()
    for i in posts:
        session.delete(i)
        session.commit()
    session.delete(user)
    session.commit()
    return RedirectResponse("/users", status_code=303)


# === CRUD для Posts ===

@app.get("/posts")
async def posts(user_id:int, request: Request):
    posts = session.query(Post).filter(Post.user_id == user_id).all()
    user = session.query(User).filter(User.id == user_id).first()
    return templates.TemplateResponse("posts.html", {"request": request,
"posts": posts, "user": user})

@app.get("/posts/new")
async def newPost(user_id: int, request: Request):
    user = session.query(User).filter(User.id == user_id).first()
    return templates.TemplateResponse("PostForm.html", {"request": request,
"user": user})


@app.post("/posts/new")
async def createPost(request: Request, user_id: int, title: str = Form(...),
content: str = Form(...)):
    a = await request.form()
    user = session.query(User).filter(User.id == user_id).first()
    try:
        new_post = Post(title=title, content=content, user_id=user.id)
        session.add(new_post)
        session.commit()
        return RedirectResponse("/posts?user_id=" + str(user_id),
status_code=303)
    except:
        session.rollback()
        return templates.TemplateResponse("PostForm.html", {"request": request,
"user": user})



@app.get("/posts/edit")
async def changePost(request: Request, post_id: int, user_id:int):
    post = session.query(Post).filter(Post.id == post_id).first()
    if not post:
        raise HTTPException(status_code=404, detail="Post not found")
```

```
    return templates.TemplateResponse("PostForm.html", {"request": request,
"post": post})


@app.post("/posts/edit")
async def updatePost(request: Request, post_id: int, user_id: int, title: str =
Form(...), content: str = Form(...)):
    a = await request.form()
    SavedPost = a.get("post")
    user = session.query(User).filter(User.id == user_id).first()
    try:
        post = session.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
        post.user_id = user.id
        session.commit()
        return RedirectResponse("/posts?user_id="+str(user.id), status_code=303)
    except Exception as ex:
        session.rollback()
        print(ex.args)
        return templates.TemplateResponse("PostForm.html", {"request": request,
"post": SavedPost})

@app.get("/posts/delete")
async def deletePost(post_id: int):
    post = session.query(Post).filter(Post.id == post_id).first()
    user_id = post.user_id
    if not post:
        raise HTTPException(status_code=404, detail="Post not found")
    session.delete(post)
    session.commit()
    return RedirectResponse("/posts?user_id=" +str(user_id), status_code=303)
```

DBCreate.py

```
from modules.models import session, User, Post

def DropAll():
    for i in session.query(User).all():
        DeleteUserPosts(i.id)
        session.query(User).filter(User.id==i.id).delete()
        session.commit()
#Напишите программу, которая добавляет в таблицу Users несколько записей с
разными значениями полей username, email и password.
def AddUsers():
    users = [
        User(username="111", email="111@222.com", password=111),
        User(username="222", email="222@222.com", password=111),
        User(username="333", email="333@222.com", password=111),
    ]
    session.add_all(users)
    session.commit()

#Напишите программу, которая добавляет в таблицу Posts несколько записей,
связанных с пользователями из таблицы Users.
def AddPosts():
    posts = [
        Post(title="postic1", content="lalalalalalal", user_id=1),
        Post(title="postic2", content="lalalalalalal", user_id=1),
        Post(title="postic3", content="lalalalalalal", user_id=2),
        Post(title="postic4", content="lalalalalalal", user_id=2),
        Post(title="postic5", content="lalalalalalal", user_id=3),
    ]
    session.add_all(posts)
    session.commit()

#Напишите программу, которая извлекает все записи из таблицы Users.
def GetUsers():
    users = session.query(User).all()
    for user in users:
        return f"ID: {user.id}, Username: {user.username}, Email: {user.email}"

#Напишите программу, которая извлекает все записи из таблицы Posts, включая
информацию о пользователях, которые их создали.
def GetPosts():
    posts = session.query(Post).all()
    for post in posts:
        return f"Post ID: {post.id}, Title: {post.title}, User:
{post.user.username}"

#Напишите программу, которая извлекает записи из таблицы Posts, созданные
конкретным пользователем.
def GetUserPosts(user_id):
    posts = session.query(Post).filter(Post.user_id == user_id).all()
    for post in posts:
        return f"Post ID: {post.id}, Title: {post.title}, Content:
{post.content}"

#Напишите программу, которая обновляет поле email у одного из пользователей.
def UpdateEmail(user_id, new_email):
    user = session.query(User).filter(User.id == user_id).first()
    if user:
        user.email = new_email
        session.commit()
        return f"Email пользователя с ID {user_id} обновлен."
    else:
        return "Пользователь не найден."

#Напишите программу, которая обновляет поле content у одного из постов.
```

```python
def UpdateContent(post_id, new_content):
    post = session.query(Post).filter(Post.id == post_id).first()
    if post:
        post.content = new_content
        session.commit()
        return f"Контент поста с ID {post_id} обновлен."
    else:
        return "Пост не найден."

#Напишите программу, которая удаляет один из постов.
def DeletePost(post_id):
    post = session.query(Post).filter(Post.id == post_id).first()
    if post:
        session.delete(post)
        session.commit()
        return f"Пост с ID {post_id} удален."
    else:
        return "Пост не найден."

#Напишите программу, которая удаляет пользователя и все его посты.
def DeleteUserPosts(user_id):
    user = session.query(User).filter(User.id == user_id).first()
    if user:
        session.query(Post).filter(Post.user_id == user_id).delete()
        session.delete(user)
        session.commit()
        return f"Пользователь с ID {user_id} и его посты удалены."
    else:
        return "Пользователь не найден."

def generate():
    DropAll()
    AddUsers()
    AddPosts()

    print(f"Все пользователи: {GetUsers()}")
    print(f"Все посты:{GetPosts()}")
    print(f"Посты пользователя с ID 1:{GetUserPosts(1)}")
    print(f"Обновление email пользователя 1:{UpdateEmail(1,
"email@email.com")}")
    print(f"Обновление контента поста:{UpdateContent(1, "Updated content for
post 1")}")
    print(f"Удаление поста:{DeletePost(2)}")
    print(f"Удаление пользователя и его постов:{DeleteUserPosts(3)}")

generate()
```

models.py

```
from modules.Global import Database_Url
from sqlalchemy import create_engine, Column, Integer, String, Text, ForeignKey
from sqlalchemy.orm import declarative_base, relationship, sessionmaker

DATABASE_URL = Database_Url

engine = create_engine(DATABASE_URL, echo=True)
Base = declarative_base()

class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True, autoincrement=True)
    username = Column(String(50), unique=True, nullable=False)
    email = Column(String(100), unique=True, nullable=False)
    password = Column(String(100), nullable=False)

    posts = relationship("Post", back_populates="user")

class Post(Base):
    __tablename__ = 'posts'

    id = Column(Integer, primary_key=True, autoincrement=True)
    title = Column(String(100), nullable=False)
    content = Column(Text, nullable=False)
    user_id = Column(Integer, ForeignKey('users.id'), nullable=False)

    user = relationship("User", back_populates="posts")

Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)
session = Session()
```

Global.py

```
Database_Url = "mysql+pymysql://root:root@127.0.0.1/test"
```