МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №5

Специальность ПО11

Выполнил
П. А. Захарчук
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
12.04.2025 г.

Брест 2025

Цель работы: приобрести практические навыки разработки API и баз данных.

Задание:

Реализовать базу данных из не менее 5 таблиц на заданную тематику.

1. При реализации продумать типизацию полей и внешние ключи в таблицах;
2. Визуализировать разработанную БД с помощью схемы, на которой отображены все таблицы и связи между ними (пример, схема на рис. 1);
3. На языке Python с использованием SQLAlchemy реализовать подключение к БД;
4. Реализовать основные операции с данными (выборку, добавление, удаление, модификацию);
5. Для каждой реализованной операции с использованием FastAPI реализовать отдельный эндпойнт;

Базу данные можно реализовать в любой СУБД (MySQL, PostgreSQL, SQLite и др.) База данных Системный администратор

Код программы:

```
    from fastapi import FastAPI, HTTPException
from sqlalchemy import create_engine, Column, Integer, String, ForeignKey, DateTime
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, relationship
from pydantic import BaseModel
from datetime import datetime
import sqlite3

# Настройка SQLAlchemy
# Создаём подключение к базе данных SQLite с именем sysadmin.db
engine = create_engine('sqlite:///sysadmin.db', echo=True)
Base = declarative_base()
Session = sessionmaker(bind=engine)
session = Session()

# Настройка FastAPI
app = FastAPI()

# Модели SQLAlchemy для таблиц
class User(Base):
    __tablename__ = 'users'  # Таблица "пользователи"
    id = Column(Integer, primary_key=True)  # Уникальный идентификатор
    username = Column(String(50), unique=True, nullable=False)  # Имя пользователя
    email = Column(String(100), unique=True, nullable=False)  # Email
    role = Column(String(50), nullable=False)  # Роль пользователя
    logs = relationship("Log", back_populates="user")  # Связь с таблицей логов
    tasks = relationship("Task", back_populates="user")  # Связь с таблицей задач
    permissions = relationship("Permission", back_populates="user")  # Связь с таблицей прав доступа

class Server(Base):
    __tablename__ = 'servers'  # Таблица "серверы"
    id = Column(Integer, primary_key=True)  # Уникальный идентификатор
    server_name = Column(String(50), unique=True, nullable=False)  # Имя сервера
    ip_address = Column(String(15), unique=True, nullable=False)  # IP-адрес
    status = Column(String(20), nullable=False)  # Статус сервера
    logs = relationship("Log", back_populates="server")  # Связь с таблицей логов
    tasks = relationship("Task", back_populates="server")  # Связь с таблицей задач
    permissions = relationship("Permission", back_populates="server")  # Связь с таблицей прав доступа

class Log(Base):
    __tablename__ = 'logs'  # Таблица "логи"
```

```
    id = Column(Integer, primary_key=True)  # Уникальный идентификатор
    user_id = Column(Integer, ForeignKey('users.id'), nullable=False)  # Внешний ключ на
пользователя
    server_id = Column(Integer, ForeignKey('servers.id'), nullable=False)  # Внешний ключ на сервер
    log_message = Column(String(200), nullable=False)  # Сообщение лога
    timestamp = Column(DateTime, default=datetime.utcnow)  # Временная метка
    user = relationship("User", back_populates="logs")  # Связь с пользователем
    server = relationship("Server", back_populates="logs")  # Связь с сервером

class Task(Base):
    __tablename__ = 'tasks'  # Таблица "задачи"
    id = Column(Integer, primary_key=True)  # Уникальный идентификатор
    user_id = Column(Integer, ForeignKey('users.id'), nullable=False)  # Внешний ключ на
пользователя
    server_id = Column(Integer, ForeignKey('servers.id'), nullable=False)  # Внешний ключ на сервер
    task_description = Column(String(200), nullable=False)  # Описание задачи
    status = Column(String(20), nullable=False)  # Статус задачи
    user = relationship("User", back_populates="tasks")  # Связь с пользователем
    server = relationship("Server", back_populates="tasks")  # Связь с сервером

class Permission(Base):
    __tablename__ = 'permissions'  # Таблица "права доступа"
    id = Column(Integer, primary_key=True)  # Уникальный идентификатор
    user_id = Column(Integer, ForeignKey('users.id'), nullable=False)  # Внешний ключ на
пользователя
    server_id = Column(Integer, ForeignKey('servers.id'), nullable=False)  # Внешний ключ на сервер
    permission_level = Column(String(20), nullable=False)  # Уровень доступа
    user = relationship("User", back_populates="permissions")  # Связь с пользователем
    server = relationship("Server", back_populates="permissions")  # Связь с сервером

# Создание таблиц в базе данных
Base.metadata.create_all(engine)

# Модели Pydantic для FastAPI (для валидации входных данных)
class UserCreate(BaseModel):
    username: str  # Имя пользователя
    email: str  # Email
    role: str  # Роль

class ServerCreate(BaseModel):
    server_name: str  # Имя сервера
    ip_address: str  # IP-адрес
    status: str  # Статус

class LogCreate(BaseModel):
    user_id: int  # ID пользователя
    server_id: int  # ID сервера
    log_message: str  # Сообщение лога

class TaskCreate(BaseModel):
    user_id: int  # ID пользователя
    server_id: int  # ID сервера
    task_description: str  # Описание задачи
    status: str  # Статус

class PermissionCreate(BaseModel):
    user_id: int  # ID пользователя
    server_id: int  # ID сервера
    permission_level: str  # Уровень доступа

# Эндпойнты FastAPI
# CRUD для пользователей
@app.post("/users/")
```

```python
def create_user(user: UserCreate):
    # Создание нового пользователя
    db_user = User(**user.dict())
    session.add(db_user)
    session.commit()
    session.refresh(db_user)
    return db_user

@app.get("/users/{user_id}")
def read_user(user_id: int):
    # Получение пользователя по ID
    user = session.query(User).filter(User.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="Пользователь не найден")
    return user

@app.put("/users/{user_id}")
def update_user(user_id: int, user: UserCreate):
    # Обновление данных пользователя
    db_user = session.query(User).filter(User.id == user_id).first()
    if not db_user:
        raise HTTPException(status_code=404, detail="Пользователь не найден")
    for key, value in user.dict().items():
        setattr(db_user, key, value)
    session.commit()
    session.refresh(db_user)
    return db_user

@app.delete("/users/{user_id}")
def delete_user(user_id: int):
    # Удаление пользователя
    user = session.query(User).filter(User.id == user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="Пользователь не найден")
    session.delete(user)
    session.commit()
    return {"message": "Пользователь удалён"}

# CRUD для серверов
@app.post("/servers/")
def create_server(server: ServerCreate):
    # Создание нового сервера
    db_server = Server(**server.dict())
    session.add(db_server)
    session.commit()
    session.refresh(db_server)
    return db_server

@app.get("/servers/{server_id}")
def read_server(server_id: int):
    # Получение сервера по ID
    server = session.query(Server).filter(Server.id == server_id).first()
    if not server:
        raise HTTPException(status_code=404, detail="Сервер не найден")
    return server

@app.put("/servers/{server_id}")
def update_server(server_id: int, server: ServerCreate):
    # Обновление данных сервера
    db_server = session.query(Server).filter(Server.id == server_id).first()
    if not db_server:
        raise HTTPException(status_code=404, detail="Сервер не найден")
    for key, value in server.dict().items():
```

```
        setattr(db_server, key, value)
    session.commit()
    session.refresh(db_server)
    return db_server

@app.delete("/servers/{server_id}")
def delete_server(server_id: int):
    # Удаление сервера
    server = session.query(Server).filter(Server.id == server_id).first()
    if not server:
        raise HTTPException(status_code=404, detail="Сервер не найден")
    session.delete(server)
    session.commit()
    return {"message": "Сервер удалён"}

# CRUD для логов
@app.post("/logs/")
def create_log(log: LogCreate):
    # Создание нового лога
    db_log = Log(**log.dict(), timestamp=datetime.utcnow())
    session.add(db_log)
    session.commit()
    session.refresh(db_log)
    return db_log

@app.get("/logs/{log_id}")
def read_log(log_id: int):
    # Получение лога по ID
    log = session.query(Log).filter(Log.id == log_id).first()
    if not log:
        raise HTTPException(status_code=404, detail="Лог не найден")
    return log

@app.put("/logs/{log_id}")
def update_log(log_id: int, log: LogCreate):
    # Обновление данных лога
    db_log = session.query(Log).filter(Log.id == log_id).first()
    if not db_log:
        raise HTTPException(status_code=404, detail="Лог не найден")
    for key, value in log.dict().items():
        setattr(db_log, key, value)
    session.commit()
    session.refresh(db_log)
    return db_log

@app.delete("/logs/{log_id}")
def delete_log(log_id: int):
    # Удаление лога
    log = session.query(Log).filter(Log.id == log_id).first()
    if not log:
        raise HTTPException(status_code=404, detail="Лог не найден")
    session.delete(log)
    session.commit()
    return {"message": "Лог удалён"}

# CRUD для задач
@app.post("/tasks/")
def create_task(task: TaskCreate):
    # Создание новой задачи
    db_task = Task(**task.dict())
    session.add(db_task)
    session.commit()
    session.refresh(db_task)
```

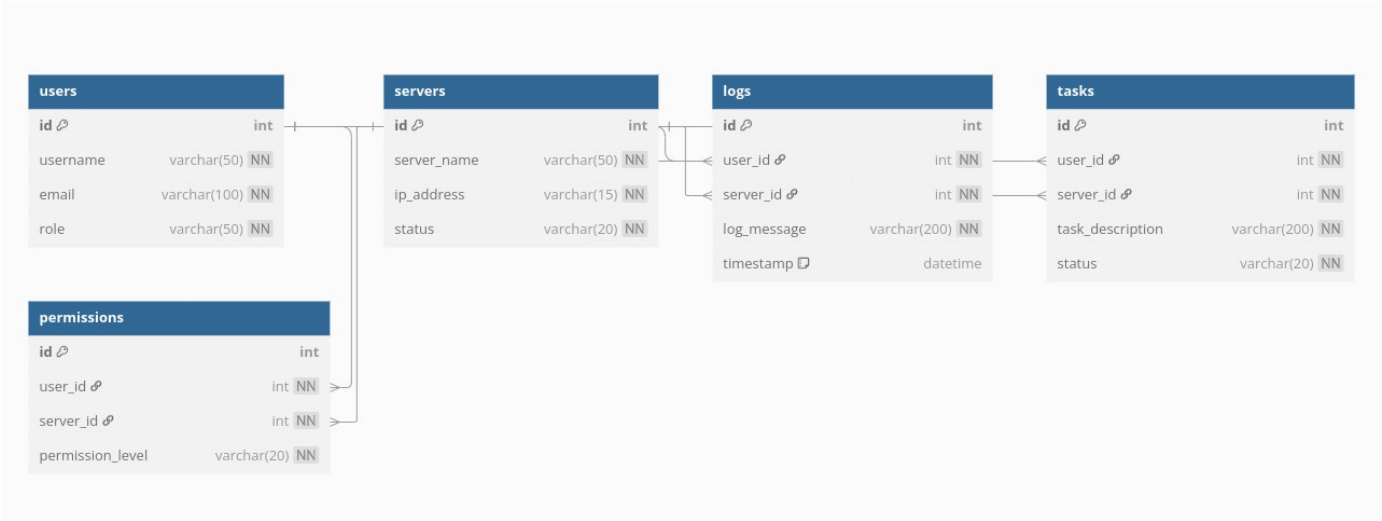
```python
    return db_task

@app.get("/tasks/{task_id}")
def read_task(task_id: int):
    # Получение задачи по ID
    task = session.query(Task).filter(Task.id == task_id).first()
    if not task:
        raise HTTPException(status_code=404, detail="Задача не найдена")
    return task

@app.put("/tasks/{task_id}")
def update_task(task_id: int, task: TaskCreate):
    # Обновление данных задачи
    db_task = session.query(Task).filter(Task.id == task_id).first()
    if not db_task:
        raise HTTPException(status_code=404, detail="Задача не найдена")
    for key, value in task.dict().items():
        setattr(db_task, key, value)
    session.commit()
    session.refresh(db_task)
    return db_task

@app.delete("/tasks/{task_id}")
def delete_task(task_id: int):
    # Удаление задачи
    task = session.query(Task).filter(Task.id == task_id).first()
    if not task:
        raise HTTPException(status_code=404, detail="Задача не найдена")
    session.delete(task)
    session.commit()
    return {"message": "Задача удалена"}

# CRUD для прав доступа
@app.post("/permissions/")
def create_permission(permission: PermissionCreate):
    # Создание нового права доступа
    db_permission = Permission(**permission.dict())
    session.add(db_permission)
    session.commit()
    session.refresh(db_permission)
    return db_permission

@app.get("/permissions/{permission_id}")
def read_permission(permission_id: int):
    # Получение права доступа по ID
    permission = session.query(Permission).filter(Permission.id == permission_id).first()
    if not permission:
        raise HTTPException(status_code=404, detail="Право доступа не найдено")
    return permission

@app.put("/permissions/{permission_id}")
def update_permission(permission_id: int, permission: PermissionCreate):
    # Обновление данных права доступа
    db_permission = session.query(Permission).filter(Permission.id == permission_id).first()
    if not db_permission:
        raise HTTPException(status_code=404, detail="Право доступа не найдено")
    for key, value in permission.dict().items():
        setattr(db_permission, key, value)
    session.commit()
    session.refresh(db_permission)
    return db_permission

@app.delete("/permissions/{permission_id}")
```

```python
def delete_permission(permission_id: int):
    # Удаление права доступа
    permission = session.query(Permission).filter(Permission.id == permission_id).first()
    if not permission:
        raise HTTPException(status_code=404, detail="Право доступа не найдено")
    session.delete(permission)
    session.commit()
    return {"message": "Право доступа удалено"}
```

Рисунки с результатами работы программы:

```
INFO:     Will watch for changes in these directories: ['/home/twinkle/PycharmProjects/pythonProject']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [25715] using StatReload
```

```
INFO:     Started server process [25717]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:48330 - "GET / HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:48330 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:32804 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:32804 - "GET /openapi.json HTTP/1.1" 200 OK
```



Вывод: приобрел практические навыки разработки API и баз данных.