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
Д. М. Андросюк
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
11.04.2025 г.

Брест 2025

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

Ход Работы

Общее задание:

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

Задание 1

1) База данных Деканат

Код программы:

```
from datetime import date
from typing import List
from fastapi import FastAPI, HTTPException
from sqlalchemy import (
    Boolean, Column, Date, Float, ForeignKey, Integer, String, create_engine
)
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker

DATABASE_URL = "sqlite:///./decanat.db"

engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()


class Student(Base):
    __tablename__ = "students"
    id = Column(Integer, primary_key=True, index=True)
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    date_of_birth = Column(Date, nullable=False)
    department_id = Column(Integer, ForeignKey("departments.id"))
    department = relationship("Department", back_populates="students")
    enrollments = relationship("Enrollment", back_populates="student")


class Teacher(Base):
    __tablename__ = "teachers"
    id = Column(Integer, primary_key=True, index=True)
```

```
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    specialization = Column(String(100), nullable=False)
    department_id = Column(Integer, ForeignKey("departments.id"))
    department = relationship("Department", back_populates="teachers")
    courses = relationship("Course", back_populates="teacher")


class Course(Base):
    __tablename__ = "courses"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    teacher_id = Column(Integer, ForeignKey("teachers.id"))
    teacher = relationship("Teacher", back_populates="courses")
    enrollments = relationship("Enrollment", back_populates="course")


class Enrollment(Base):
    __tablename__ = "enrollments"
    id = Column(Integer, primary_key=True, index=True)
    student_id = Column(Integer, ForeignKey("students.id"))
    course_id = Column(Integer, ForeignKey("courses.id"))
    grade = Column(Float, nullable=True)
    student = relationship("Student", back_populates="enrollments")
    course = relationship("Course", back_populates="enrollments")


class Department(Base):
    __tablename__ = "departments"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    students = relationship("Student", back_populates="department")
    teachers = relationship("Teacher", back_populates="department")


Base.metadata.create_all(bind=engine)


app = FastAPI()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()


@app.post("/students/")
def create_student(student: dict):
    db = SessionLocal()
    db_student = Student(**student)
    db.add(db_student)
    db.commit()
    db.refresh(db_student)
```

```
    return db_student

@app.get("/students/")
def get_students():
    db = SessionLocal()
    return db.query(Student).all()

@app.put("/students/{student_id}")
def update_student(student_id: int, student: dict):
    db = SessionLocal()
    db_student = db.query(Student).filter(Student.id == student_id).first()
    if not db_student:
        raise HTTPException(status_code=404, detail="Student not found")
    for key, value in student.items():
        setattr(db_student, key, value)
    db.commit()
    return db_student

@app.delete("/students/{student_id}")
def delete_student(student_id: int):
    db = SessionLocal()
    db_student = db.query(Student).filter(Student.id == student_id).first()
    if not db_student:
        raise HTTPException(status_code=404, detail="Student not found")
    db.delete(db_student)
    db.commit()
    return {"message": "Student deleted"}

@app.post("/teachers/")
def create_teacher(teacher: dict):
    db = SessionLocal()
    db_teacher = Teacher(**teacher)
    db.add(db_teacher)
    db.commit()
    db.refresh(db_teacher)
    return db_teacher

@app.get("/teachers/")
def get_teachers():
    db = SessionLocal()
    return db.query(Teacher).all()

@app.put("/teachers/{teacher_id}")
def update_teacher(teacher_id: int, teacher: dict):
    db = SessionLocal()
    db_teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
    if not db_teacher:
        raise HTTPException(status_code=404, detail="Teacher not found")
    for key, value in teacher.items():
        setattr(db_teacher, key, value)
    db.commit()
    return db_teacher

@app.delete("/teachers/{teacher_id}")
def delete_teacher(teacher_id: int):
```

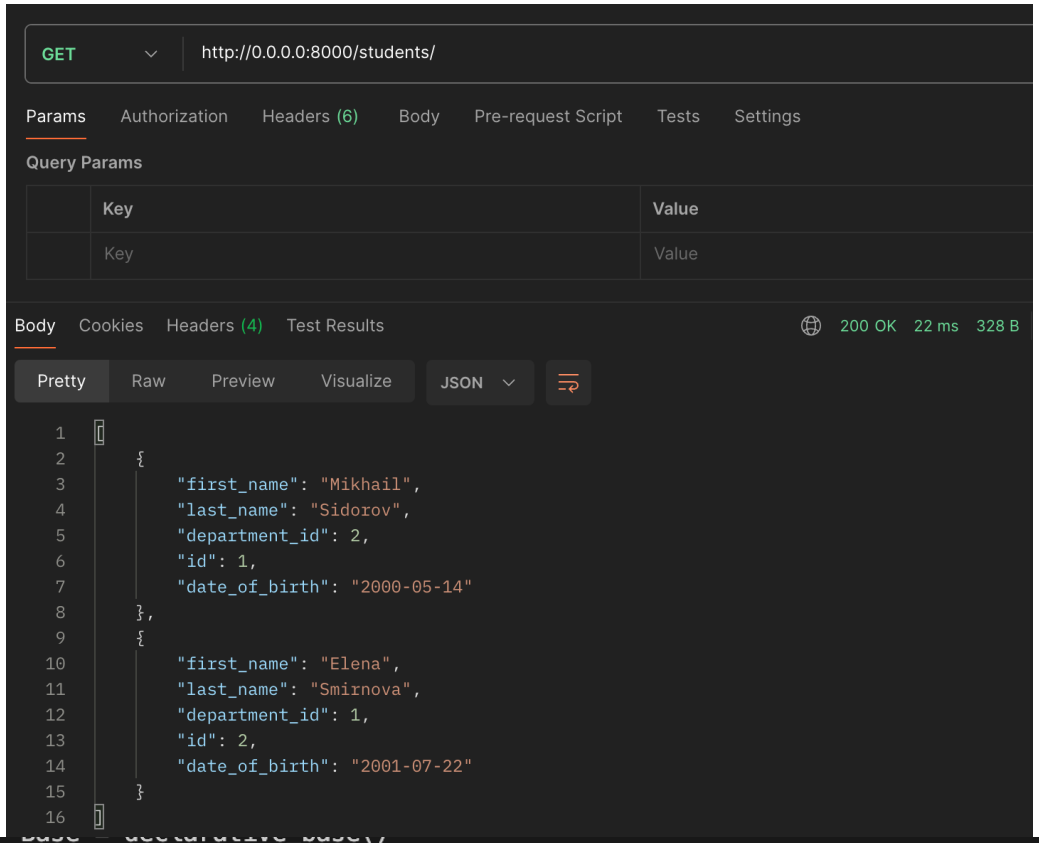
```
    db = SessionLocal()
    db_teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
    if not db_teacher:
        raise HTTPException(status_code=404, detail="Teacher not found")
    db.delete(db_teacher)
    db.commit()
    return {"message": "Teacher deleted"}

@app.post("/courses/")
def create_course(course: dict):
    db = SessionLocal()
    db_course = Course(**course)
    db.add(db_course)
    db.commit()
    db.refresh(db_course)
    return db_course

@app.get("/courses/")
def get_courses():
    db = SessionLocal()
    return db.query(Course).all()

@app.put("/courses/{course_id}")
def update_course(course_id: int, course: dict):
    db = SessionLocal()
    db_course = db.query(Course).filter(Course.id == course_id).first()
    if not db_course:
        raise HTTPException(status_code=404, detail="Course not found")
    for key, value in course.items():
        setattr(db_course, key, value)
    db.commit()
    return db_course

@app.delete("/courses/{course_id}")
def delete_course(course_id: int):
    db = SessionLocal()
    db_course = db.query(Course).filter(Course.id == course_id).first()
    if not db_course:
        raise HTTPException(status_code=404, detail="Course not found")
    db.delete(db_course)
    db.commit()
    return {"message": "Course deleted"}

@app.post("/departments/")
def create_department(department: dict):
    db = SessionLocal()
    db_department = Department(**department)
    db.add(db_department)
    db.commit()
    db.refresh(db_department)
    return db_department

@app.get("/departments/")
def get_departments():
    db = SessionLocal()
```

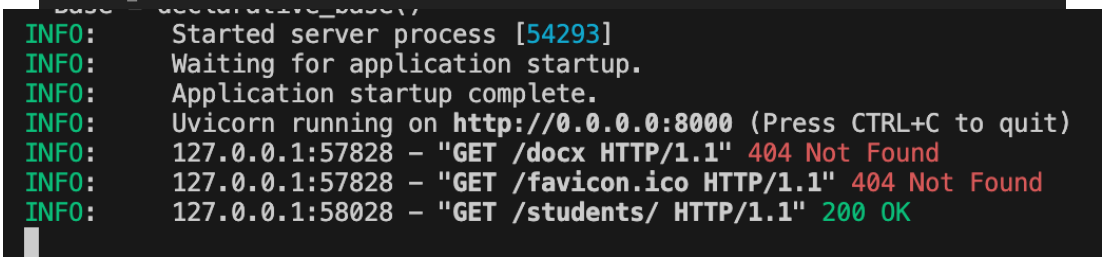
```
    return db.query(Department).all()

@app.put("/departments/{department_id}")
def update_department(department_id: int, department: dict):
    db = SessionLocal()
    db_department = db.query(Department).filter(Department.id == department_id).first()
    if not db_department:
        raise HTTPException(status_code=404, detail="Department not found")
    for key, value in department.items():
        setattr(db_department, key, value)
    db.commit()
    return db_department

@app.delete("/departments/{department_id}")
def delete_department(department_id: int):
    db = SessionLocal()
    db_department = db.query(Department).filter(Department.id == department_id).first()
    if not db_department:
        raise HTTPException(status_code=404, detail="Department not found")
    db.delete(db_department)
    db.commit()
    return {"message": "Department deleted"}

@app.post("/enrollments/")
def create_enrollment(enrollment: dict):
    db = SessionLocal()
    db_enrollment = Enrollment(**enrollment)
    db.add(db_enrollment)
    db.commit()
    db.refresh(db_enrollment)
    return db_enrollment

@app.get("/enrollments/")
def get_enrollments():
    db = SessionLocal()
    return db.query(Enrollment).all()

@app.put("/enrollments/{enrollment_id}")
def update_enrollment(enrollment_id: int, enrollment: dict):
    db = SessionLocal()
    db_enrollment = db.query(Enrollment).filter(Enrollment.id == enrollment_id).first()
    if not db_enrollment:
        raise HTTPException(status_code=404, detail="Enrollment not found")
    for key, value in enrollment.items():
        setattr(db_enrollment, key, value)
    db.commit()
    return db_enrollment

@app.delete("/enrollments/{enrollment_id}")
def delete_enrollment(enrollment_id: int):
    db = SessionLocal()
    db_enrollment = db.query(Enrollment).filter(Enrollment.id == enrollment_id).first()
    if not db_enrollment:
        raise HTTPException(status_code=404, detail="Enrollment not found")
    db.delete(db_enrollment)
```

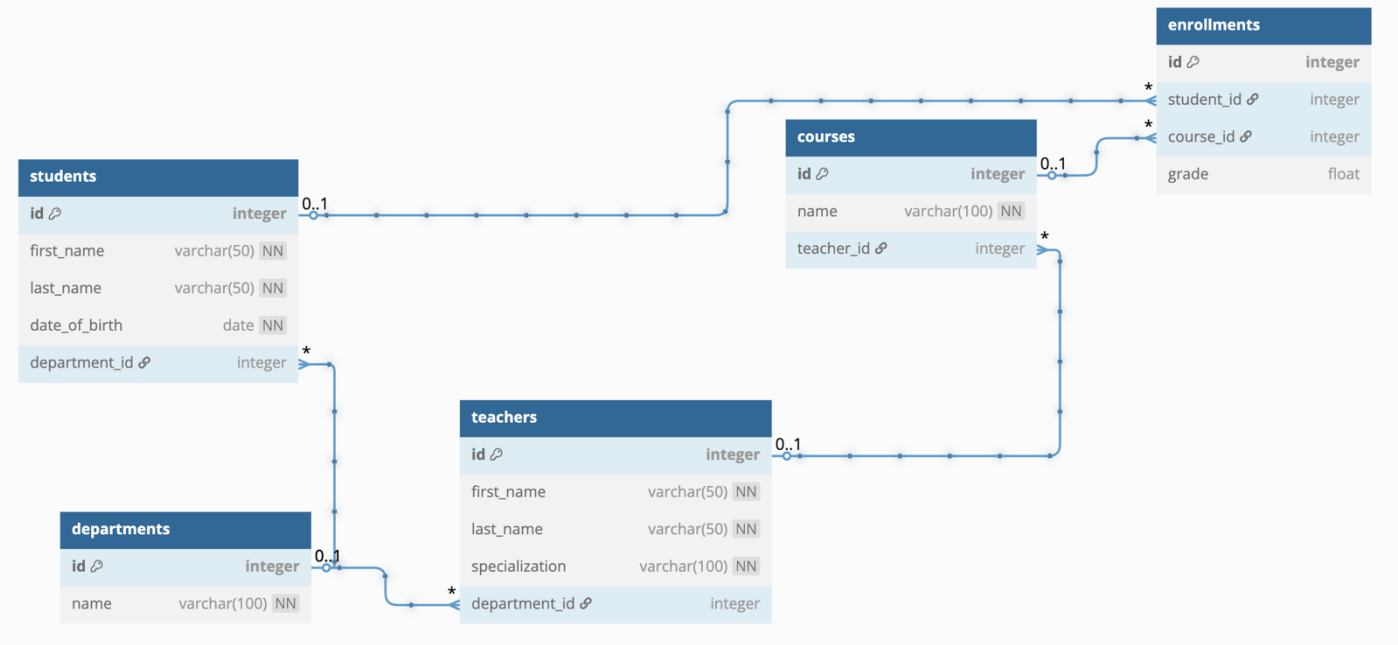
```
    db.commit()
    return {"message": "Enrollment deleted"}

@app.post("/fill_test_data/")
def fill_test_data():
    db = SessionLocal()


    physics_department = Department(name="Physics")
    math_department = Department(name="Mathematics")
    db.add_all([physics_department, math_department])

    teacher1 = Teacher(
        first_name="Ivan",
        last_name="Petrov",
        specialization="Mathematics",
        department=math_department
    )
    teacher2 = Teacher(
        first_name="Anna",
        last_name="Ivanova",
        specialization="Physics",
        department=physics_department
    )
    db.add_all([teacher1, teacher2])

    course1 = Course(name="Linear Algebra", teacher=teacher1)
    course2 = Course(name="Quantum Mechanics", teacher=teacher2)
    db.add_all([course1, course2])


    student1 = Student(
        first_name="Mikhail",
        last_name="Si   rov",
        date_of_birth=date(2000, 5, 14),
        department=math_department
    )
    student2 = Student(
        first_name="Elena",
        last_name="Smirnova",
        date_of_birth=date(2001, 7, 22),
        department=physics_department
    )
    db.add_all([student1, student2])


    enrollment1 = Enrollment(student=student1, course=course1, grade=4.5)
    enrollment2 = Enrollment(student=student2, course=course2, grade=3.8)
    db.add_all([enrollment1, enrollment2])

    db.commit()
    return {"message": "Test data added successfully"}

if __name__ == "__main__":
    import uvicorn
```

uvicorn.run(app, host="0.0.0.0", port=8000)

Рисунки с результатами работы программы:







Вывод: Приобрел практические навыки разработки API и баз данных.