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
Зайченко С.В.
студент группы ПО11

Проверил
Крощенко А. А.
ст. преп. кафедры ИИТ

Брест 2025

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

Задание:

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.)

6. База данных Учет успеваемости

Код программы:

```
import os
from datetime import date
from typing import List, Optional
  import uvicorn
from fastapi import Depends, FastAPI, HTTPException
from pydantic import BaseModel, Field
from sqlalchemy import (
  Boolean,
  Column,
  Date,
  Float,
  ForeignKey,
  Integer,
  String,
  Table,
  create_engine,
)
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker

DATABASE_FILE = "academic_performance.db"
DATABASE_URL = f"sqlite:///./{DATABASE_FILE}"

if os.path.exists(DATABASE_FILE):
  os.remove(DATABASE_FILE)
  print(f"Старый файл базы данных '{DATABASE_FILE}' удален.")

engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

```
teacher_subjects_association = Table(
    "teacher_subjects",
    Base.metadata,
    Column("teacher_id", Integer, ForeignKey("teachers.id"), primary_key=True),
    Column("subject_id", Integer, ForeignKey("subjects.id"), primary_key=True),
)

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=True)
    class_id = Column(Integer, ForeignKey("classes.id"), nullable=True)

    student_class = relationship("Class", back_populates="students")
    grades = relationship("Grade", back_populates="student", cascade="all, delete-orphan")

class Subject(Base):
    __tablename__ = "subjects"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False, unique=True)
    description = Column(String(255), nullable=True)

    grades = relationship("Grade", back_populates="subject")
    teachers = relationship(
        "Teacher",
        secondary=teacher_subjects_association,
        back_populates="subjects",
    )

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)
    hire_date = Column(Date, nullable=True)
    phone_number = Column(String(20), nullable=True)

    subjects = relationship(
        "Subject",
        secondary=teacher_subjects_association,
        back_populates="teachers",
    )
    grades_given = relationship("Grade", back_populates="teacher")
    homeroom_class = relationship("Class", back_populates="homeroom_teacher")
```

```
class Class(Base):
    __tablename__ = "classes"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(10), nullable=False, unique=True)
    year = Column(Integer, nullable=False)
    homeroom_teacher_id = Column(Integer, ForeignKey("teachers.id"), nullable=True)

    students = relationship("Student", back_populates="student_class")
    homeroom_teacher = relationship("Teacher", back_populates="homeroom_class")

class Grade(Base):
    __tablename__ = "grades"

    id = Column(Integer, primary_key=True, index=True)
    student_id = Column(Integer, ForeignKey("students.id"), nullable=False)
    subject_id = Column(Integer, ForeignKey("subjects.id"), nullable=False)
    teacher_id = Column(Integer, ForeignKey("teachers.id"), nullable=False)
    grade_value = Column(Integer, nullable=False)
    grade_date = Column(Date, nullable=False, default=date.today)

    student = relationship("Student", back_populates="grades")
    subject = relationship("Subject", back_populates="grades")
    teacher = relationship("Teacher", back_populates="grades_given")

print("Создание таблиц в базе данных...")
Base.metadata.create_all(bind=engine)
print("Таблицы успешно созданы.")

app = FastAPI(title="API Учета Успеваемости", description="API для управления данными об учениках,
учителях, предметах и оценках.")


class StudentBase(BaseModel):
    first_name: str = Field(..., description="Имя ученика")
    last_name: str = Field(..., description="Фамилия ученика")
    date_of_birth: Optional[date] = Field(None, description="Дата рождения")
    class_id: Optional[int] = Field(None, description="ID класса (необязательно при создании)")

class StudentCreate(StudentBase):
    pass

class StudentResponse(StudentBase):
    id: int

    class Config:
        from_attributes = True

class SubjectBase(BaseModel):
    name: str = Field(..., description="Название предмета")
    description: Optional[str] = Field(None, description="Описание предмета")
```

```
class SubjectCreate(SubjectBase):
  pass

class SubjectResponse(SubjectBase):
  id: int

  class Config:
    from_attributes = True

class TeacherBase(BaseModel):
  first_name: str = Field(..., description="Имя учителя")
  last_name: str = Field(..., description="Фамилия учителя")
  hire_date: Optional[date] = Field(None, description="Дата приема на работу")
  phone_number: Optional[str] = Field(None, description="Контактный телефон")

class TeacherCreate(TeacherBase):
  pass

class TeacherResponse(TeacherBase):
  id: int

  class Config:
    from_attributes = True

class ClassBase(BaseModel):
  name: str = Field(..., description="Название класса (e.g., '9B')")
  year: int = Field(..., description="Учебный год")
  homeroom_teacher_id: Optional[int] = Field(None, description="ID классного руководителя")

class ClassCreate(ClassBase):
  pass

class ClassResponse(ClassBase):
  id: int

  class Config:
    from_attributes = True

class GradeBase(BaseModel):
  student_id: int = Field(..., description="ID ученика")
  subject_id: int = Field(..., description="ID предмета")
  teacher_id: int = Field(..., description="ID учителя")
  grade_value: int = Field(..., ge=1, le=5, description="Оценка (например, от 1 до 5)")
  grade_date: date = Field(default_factory=date.today, description="Дата оценки")

class GradeCreate(GradeBase):
  pass

class GradeResponse(GradeBase):
  id: int
```

```
    class Config:
        from_attributes = True


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

@app.post("/students/", response_model=StudentResponse, tags=["Students"], summary="Создать нового ученика")
def create_student(student: StudentCreate, db: SessionLocal = Depends(get_db)):
    db_student = Student(**student.model_dump())
    if student.class_id:
        db_class = db.query(Class).filter(Class.id == student.class_id).first()
        if not db_class:
            raise HTTPException(status_code=404, detail=f"Класс с ID {student.class_id} не найден.")
    db.add(db_student)
    db.commit()
    db.refresh(db_student)
    return db_student

@app.get("/students/", response_model=List[StudentResponse], tags=["Students"], summary="Получить список всех учеников")
def get_students(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    students = db.query(Student).offset(skip).limit(limit).all()
    return students

@app.get("/students/{student_id}", response_model=StudentResponse, tags=["Students"], summary="Получить ученика по ID")
def get_student(student_id: int, db: SessionLocal = Depends(get_db)):
    student = db.query(Student).filter(Student.id == student_id).first()
    if student is None:
        raise HTTPException(status_code=404, detail="Ученик не найден")
    return student

@app.put("/students/{student_id}", response_model=StudentResponse, tags=["Students"], summary="Обновить информацию об ученике")
def update_student(student_id: int, student: StudentCreate, db: SessionLocal = Depends(get_db)):
    db_student = db.query(Student).filter(Student.id == student_id).first()
    if db_student is None:
        raise HTTPException(status_code=404, detail="Ученик не найден")

    if student.class_id and student.class_id != db_student.class_id:
        db_class = db.query(Class).filter(Class.id == student.class_id).first()
        if not db_class:
            raise HTTPException(status_code=404, detail=f"Класс с ID {student.class_id} не найден.")
```

```
    update_data = student.model_dump(exclude_unset=True)
    for key, value in update_data.items():
        setattr(db_student, key, value)

    db.commit()
    db.refresh(db_student)
    return db_student

@app.delete("/students/{student_id}", tags=["Students"], summary="Удалить ученика")
def delete_student(student_id: int, db: SessionLocal = Depends(get_db)):
    db_student = db.query(Student).filter(Student.id == student_id).first()
    if db_student is None:
        raise HTTPException(status_code=404, detail="Ученик не найден")
    db.delete(db_student)
    db.commit()
    return {"message": f"Ученик с ID {student_id} успешно удален"}

@app.post("/subjects/", response_model=SubjectResponse, tags=["Subjects"], summary="Создать новый
предмет")
def create_subject(subject: SubjectCreate, db: SessionLocal = Depends(get_db)):
    existing_subject = db.query(Subject).filter(Subject.name == subject.name).first()
    if existing_subject:
        raise HTTPException(status_code=400, detail=f"Предмет с названием '{subject.name}' уже
существует.")
    db_subject = Subject(**subject.model_dump())
    db.add(db_subject)
    db.commit()
    db.refresh(db_subject)
    return db_subject

@app.get("/subjects/", response_model=List[SubjectResponse], tags=["Subjects"], summary="Получить
список всех предметов")
def get_subjects(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    subjects = db.query(Subject).offset(skip).limit(limit).all()
    return subjects

@app.get("/subjects/{subject_id}", response_model=SubjectResponse, tags=["Subjects"],
summary="Получить предмет по ID")
def get_subject(subject_id: int, db: SessionLocal = Depends(get_db)):
    subject = db.query(Subject).filter(Subject.id == subject_id).first()
    if subject is None:
        raise HTTPException(status_code=404, detail="Предмет не найден")
    return subject

@app.put("/subjects/{subject_id}", response_model=SubjectResponse, tags=["Subjects"],
summary="Обновить информацию о предмете")
def update_subject(subject_id: int, subject: SubjectCreate, db: SessionLocal = Depends(get_db)):
    db_subject = db.query(Subject).filter(Subject.id == subject_id).first()
    if db_subject is None:
        raise HTTPException(status_code=404, detail="Предмет не найден")
```

```
    if subject.name != db_subject.name:
        existing_subject = db.query(Subject).filter(Subject.name == subject.name).first()
        if existing_subject:
            raise HTTPException(status_code=400, detail=f"Предмет с названием '{subject.name}' уже существует.")

    update_data = subject.model_dump(exclude_unset=True)
    for key, value in update_data.items():
        setattr(db_subject, key, value)

    db.commit()
    db.refresh(db_subject)
    return db_subject

@app.delete("/subjects/{subject_id}", tags=["Subjects"], summary="Удалить предмет")
def delete_subject(subject_id: int, db: SessionLocal = Depends(get_db)):
    db_subject = db.query(Subject).filter(Subject.id == subject_id).first()
    if db_subject is None:
        raise HTTPException(status_code=404, detail="Предмет не найден")

    related_grades = db.query(Grade).filter(Grade.subject_id == subject_id).count()
    if related_grades > 0:
        raise HTTPException(status_code=400, detail=f"Невозможно удалить предмет '{db_subject.name}', так как есть связанные с ним оценки ({related_grades} шт.). Сначала удалите или измените оценки.")
    db.delete(db_subject)
    db.commit()
    return {"message": f"Предмет с ID {subject_id} успешно удален"}


@app.post("/teachers/", response_model=TeacherResponse, tags=["Teachers"], summary="Создать нового учителя")
def create_teacher(teacher: TeacherCreate, db: SessionLocal = Depends(get_db)):
    db_teacher = Teacher(**teacher.model_dump())
    db.add(db_teacher)
    db.commit()
    db.refresh(db_teacher)
    return db_teacher

@app.get("/teachers/", response_model=List[TeacherResponse], tags=["Teachers"], summary="Получить список всех учителей")
def get_teachers(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    teachers = db.query(Teacher).offset(skip).limit(limit).all()
    return teachers

@app.get("/teachers/{teacher_id}", response_model=TeacherResponse, tags=["Teachers"], summary="Получить учителя по ID")
def get_teacher(teacher_id: int, db: SessionLocal = Depends(get_db)):
    teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
    if teacher is None:
        raise HTTPException(status_code=404, detail="Учитель не найден")
    return teacher
```

```
@app.put("/teachers/{teacher_id}", response_model=TeacherResponse, tags=["Teachers"],
summary="Обновить информацию об учителе")
def update_teacher(teacher_id: int, teacher: TeacherCreate, db: SessionLocal = Depends(get_db)):
  db_teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
  if db_teacher is None:
    raise HTTPException(status_code=404, detail="Учитель не найден")

  update_data = teacher.model_dump(exclude_unset=True)
  for key, value in update_data.items():
    setattr(db_teacher, key, value)

  db.commit()
  db.refresh(db_teacher)
  return db_teacher

@app.delete("/teachers/{teacher_id}", tags=["Teachers"], summary="Удалить учителя")
def delete_teacher(teacher_id: int, db: SessionLocal = Depends(get_db)):
  db_teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
  if db_teacher is None:
    raise HTTPException(status_code=404, detail="Учитель не найден")

  if db.query(Grade).filter(Grade.teacher_id == teacher_id).count() > 0:
    raise HTTPException(status_code=400, detail="Невозможно удалить учителя, так как он выставил
оценки.")
  if db.query(Class).filter(Class.homeroom_teacher_id == teacher_id).count() > 0:
    raise HTTPException(status_code=400, detail="Невозможно удалить учителя, так как он является
классным руководителем.")
  if db_teacher.subjects:
     db_teacher.subjects.clear()

  db.delete(db_teacher)
  db.commit()
  return {"message": f"Учитель с ID {teacher_id} успешно удален"}

class TeacherSubjectLink(BaseModel):
  subject_id: int

@app.post("/teachers/{teacher_id}/subjects/", status_code=201, tags=["Teachers", "Subjects"],
summary="Назначить предмет учителю")
def link_teacher_to_subject(teacher_id: int, link: TeacherSubjectLink, db: SessionLocal = Depends(get_db)):
  db_teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
  if not db_teacher:
    raise HTTPException(status_code=404, detail="Учитель не найден")
  db_subject = db.query(Subject).filter(Subject.id == link.subject_id).first()
  if not db_subject:
    raise HTTPException(status_code=404, detail="Предмет не найден")
  if db_subject in db_teacher.subjects:
     raise HTTPException(status_code=400, detail="Учитель уже преподает этот предмет")

  db_teacher.subjects.append(db_subject)
```

```
    db.commit()
    return {"message": f"Предмет '{db_subject.name}' назначен учителю '{db_teacher.first_name} {db_teacher.last_name}'"}

@app.delete("/teachers/{teacher_id}/subjects/{subject_id}", tags=["Teachers", "Subjects"], summary="Убрать предмет у учителя")
def unlink_teacher_from_subject(teacher_id: int, subject_id: int, db: SessionLocal = Depends(get_db)):
    db_teacher = db.query(Teacher).filter(Teacher.id == teacher_id).first()
    if not db_teacher:
        raise HTTPException(status_code=404, detail="Учитель не найден")
    db_subject = db.query(Subject).filter(Subject.id == subject_id).first()
    if not db_subject:
        raise HTTPException(status_code=404, detail="Предмет не найден")

    if db_subject not in db_teacher.subjects:
        raise HTTPException(status_code=404, detail="Учитель не преподает данный предмет")

    db_teacher.subjects.remove(db_subject)
    db.commit()
    return {"message": f"Предмет '{db_subject.name}' убран у учителя '{db_teacher.first_name} {db_teacher.last_name}'"}


@app.post("/classes/", response_model=ClassResponse, tags=["Classes"], summary="Создать новый класс")
def create_class(class_in: ClassCreate, db: SessionLocal = Depends(get_db)):
    existing_class = db.query(Class).filter(Class.name == class_in.name).first()
    if existing_class:
        raise HTTPException(status_code=400, detail=f"Класс с названием '{class_in.name}' уже существует.")
    if class_in.homeroom_teacher_id:
        db_teacher = db.query(Teacher).filter(Teacher.id == class_in.homeroom_teacher_id).first()
        if not db_teacher:
            raise HTTPException(status_code=404, detail=f"Учитель с ID {class_in.homeroom_teacher_id} не найден.")
    db_class = Class(**class_in.model_dump())
    db.add(db_class)
    db.commit()
    db.refresh(db_class)
    return db_class

@app.get("/classes/", response_model=List[ClassResponse], tags=["Classes"], summary="Получить список всех классов")
def get_classes(skip: int = 0, limit: int = 100, db: SessionLocal = Depends(get_db)):
    classes = db.query(Class).offset(skip).limit(limit).all()
    return classes

@app.get("/classes/{class_id}", response_model=ClassResponse, tags=["Classes"], summary="Получить класс по ID")
def get_class(class_id: int, db: SessionLocal = Depends(get_db)):
    db_class = db.query(Class).filter(Class.id == class_id).first()
    if db_class is None:
```

```
    raise HTTPException(status_code=404, detail="Класс не найден")
  return db_class

@app.put("/classes/{class_id}", response_model=ClassResponse, tags=["Classes"], summary="Обновить информацию о классе")
def update_class(class_id: int, class_in: ClassCreate, db: SessionLocal = Depends(get_db)):
  db_class = db.query(Class).filter(Class.id == class_id).first()
  if db_class is None:
    raise HTTPException(status_code=404, detail="Класс не найден")

  if class_in.name != db_class.name:
    existing_class = db.query(Class).filter(Class.name == class_in.name).first()
    if existing_class:
      raise HTTPException(status_code=400, detail=f"Класс с названием '{class_in.name}' уже существует.")

  if class_in.homeroom_teacher_id and class_in.homeroom_teacher_id != db_class.homeroom_teacher_id:
    db_teacher = db.query(Teacher).filter(Teacher.id == class_in.homeroom_teacher_id).first()
    if not db_teacher:
      raise HTTPException(status_code=404, detail=f"Учитель с ID {class_in.homeroom_teacher_id} не найден.")

  update_data = class_in.model_dump(exclude_unset=True)
  for key, value in update_data.items():
    setattr(db_class, key, value)

  db.commit()
  db.refresh(db_class)
  return db_class

@app.delete("/classes/{class_id}", tags=["Classes"], summary="Удалить класс")
def delete_class(class_id: int, db: SessionLocal = Depends(get_db)):
  db_class = db.query(Class).filter(Class.id == class_id).first()
  if db_class is None:
    raise HTTPException(status_code=404, detail="Класс не найден")
  if db.query(Student).filter(Student.class_id == class_id).count() > 0:
    db.query(Student).filter(Student.class_id == class_id).update({Student.class_id: None})
    print(f"Ученики класса ID {class_id} откреплены от класса.")


  db.delete(db_class)
  db.commit()
  return {"message": f"Класс с ID {class_id} успешно удален"}


@app.post("/grades/", response_model=GradeResponse, tags=["Grades"], summary="Добавить оценку")
def create_grade(grade: GradeCreate, db: SessionLocal = Depends(get_db)):
  if not db.query(Student).filter(Student.id == grade.student_id).first():
    raise HTTPException(status_code=404, detail=f"Ученик с ID {grade.student_id} не найден.")
  if not db.query(Subject).filter(Subject.id == grade.subject_id).first():
    raise HTTPException(status_code=404, detail=f"Предмет с ID {grade.subject_id} не найден.")
```

```
    if not db.query(Teacher).filter(Teacher.id == grade.teacher_id).first():
        raise HTTPException(status_code=404, detail=f"Учитель с ID {grade.teacher_id} не найден.")

    db_grade = Grade(**grade.model_dump())
    db.add(db_grade)
    db.commit()
    db.refresh(db_grade)
    return db_grade

@app.get("/grades/", response_model=List[GradeResponse], tags=["Grades"], summary="Получить список всех оценок")
def get_grades(
    student_id: Optional[int] = None,
    subject_id: Optional[int] = None,
    teacher_id: Optional[int] = None,
    skip: int = 0,
    limit: int = 100,
    db: SessionLocal = Depends(get_db)
):
    query = db.query(Grade)
    if student_id:
        query = query.filter(Grade.student_id == student_id)
    if subject_id:
        query = query.filter(Grade.subject_id == subject_id)
    if teacher_id:
        query = query.filter(Grade.teacher_id == teacher_id)

    grades = query.offset(skip).limit(limit).all()
    return grades

@app.get("/grades/{grade_id}", response_model=GradeResponse, tags=["Grades"], summary="Получить оценку по ID")
def get_grade(grade_id: int, db: SessionLocal = Depends(get_db)):
    grade = db.query(Grade).filter(Grade.id == grade_id).first()
    if grade is None:
        raise HTTPException(status_code=404, detail="Оценка не найдена")
    return grade

@app.put("/grades/{grade_id}", response_model=GradeResponse, tags=["Grades"], summary="Обновить оценку")
def update_grade(grade_id: int, grade: GradeCreate, db: SessionLocal = Depends(get_db)):
    db_grade = db.query(Grade).filter(Grade.id == grade_id).first()
    if db_grade is None:
        raise HTTPException(status_code=404, detail="Оценка не найдена")
    if grade.student_id != db_grade.student_id and not db.query(Student).filter(Student.id == grade.student_id).first():
        raise HTTPException(status_code=404, detail=f"Ученик с ID {grade.student_id} не найден.")
    if grade.subject_id != db_grade.subject_id and not db.query(Subject).filter(Subject.id == grade.subject_id).first():
        raise HTTPException(status_code=404, detail=f"Предмет с ID {grade.subject_id} не найден.")
    if grade.teacher_id != db_grade.teacher_id and not db.query(Teacher).filter(Teacher.id ==
```

```
grade.teacher_id).first():
        raise HTTPException(status_code=404, detail=f"Учитель с ID {grade.teacher_id} не найден.")


    update_data = grade.model_dump(exclude_unset=True)
    for key, value in update_data.items():
        setattr(db_grade, key, value)

    db.commit()
    db.refresh(db_grade)
    return db_grade

@app.delete("/grades/{grade_id}", tags=["Grades"], summary="Удалить оценку")
def delete_grade(grade_id: int, db: SessionLocal = Depends(get_db)):
    db_grade = db.query(Grade).filter(Grade.id == grade_id).first()
    if db_grade is None:
        raise HTTPException(status_code=404, detail="Оценка не найдена")
    db.delete(db_grade)
    db.commit()
    return {"message": f"Оценка с ID {grade_id} успешно удалена"}


if __name__ == "__main__":
    print("Запуск FastAPI приложения...")
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

```
Создание таблиц в базе данных...
Таблицы успешно созданы.
Запуск FastAPI приложения...
INFO:     Started server process [11952]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

Вывод: приобрел практические навыки разработки API и баз данных.