МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №5
По дисциплине “**Современные платформы программирования**”

Выполнил:
студент группы ПО-11
Турабов А. В.
Проверил:
Козик И. Д.

Брест 2025

Цель работы: приобрести практические навыки разработки API и баз данных

Общее задание

1. Реализовать базу данных из не менее 5 таблиц на заданную тематику. При реализации продумать типизацию полей и внешние ключи в таблицах;

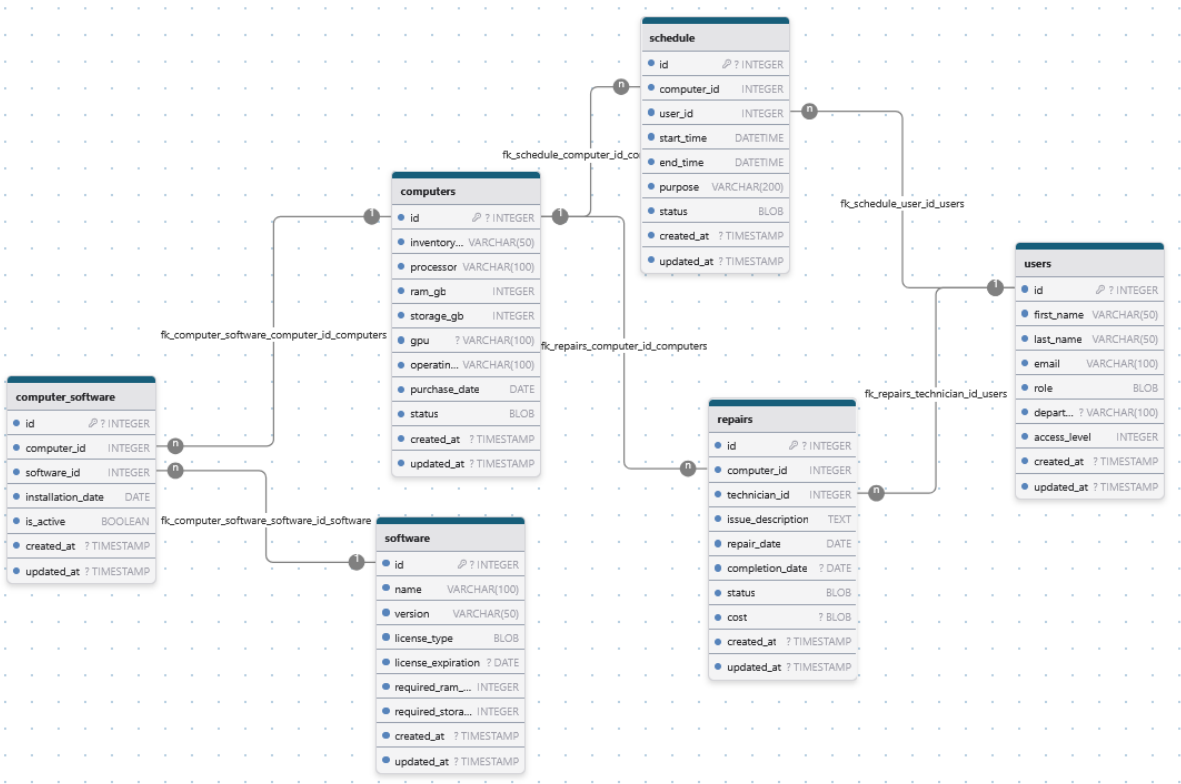
2. Визуализировать разработанную БД с помощью схемы, на которой отображены все таблицы и связи между ними (пример, схема на рис. 1);

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;

База данных Компьютерная лаборатория



Код программы

main.py

```
from fastapi import FastAPI, HTTPException, Depends
from pydantic import BaseModel
from typing import List, Optional
from datetime import date, datetime
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, Session
from models import Base, Computer, Software, User, Repair, Schedule, ComputerSoftware

app = FastAPI()

DATABASE_URL = "sqlite:///computer_lab.db"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base.metadata.create_all(bind=engine)

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


class ComputerBase(BaseModel):
    inventory_number: str
    processor: str
    ram_gb: int
    storage_gb: int
    gpu: str
    operating_system: str
    purchase_date: date
    status: str

class ComputerCreate(ComputerBase):
    pass

class ComputerResponse(ComputerBase):
    id: int

    class Config:
        orm_mode = True

class SoftwareBase(BaseModel):
    name: str
    version: str
    license_type: str
    license_expiration: Optional[date] = None
    required_ram_gb: int
    required_storage_gb: int

class SoftwareCreate(SoftwareBase):
    pass

class SoftwareResponse(SoftwareBase):
    id: int

    class Config:
        orm_mode = True

class UserBase(BaseModel):
    first_name: str
    last_name: str
    email: str
    role: str
    department: str
    access_level: int
```

```
class UserCreate(UserBase):
    pass

class UserResponse(UserBase):
    id: int

    class Config:
        orm_mode = True

class RepairBase(BaseModel):
    computer_id: int
    technician_id: int
    issue_description: str
    repair_date: date
    completion_date: Optional[date] = None
    status: str
    cost: float

class RepairCreate(RepairBase):
    pass

class RepairResponse(RepairBase):
    id: int

    class Config:
        orm_mode = True

class ScheduleBase(BaseModel):
    computer_id: int
    user_id: int
    start_time: datetime
    end_time: datetime
    purpose: str
    status: str

class ScheduleCreate(ScheduleBase):
    pass

class ScheduleResponse(ScheduleBase):
    id: int

    class Config:
        orm_mode = True

class ComputerSoftwareBase(BaseModel):
    computer_id: int
    software_id: int
    installation_date: date
    is_active: bool

class ComputerSoftwareCreate(ComputerSoftwareBase):
    pass

class ComputerSoftwareResponse(ComputerSoftwareBase):
    id: int

    class Config:
        orm_mode = True


@app.post("/computers/", response_model=ComputerResponse)
def create_computer(computer: ComputerCreate, db: Session = Depends(get_db)):
    db_computer = Computer(**computer.dict())
    db.add(db_computer)
    db.commit()
    db.refresh(db_computer)
    return db_computer

@app.get("/computers/", response_model=List[ComputerResponse])
def read_computers(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    computers = db.query(Computer).offset(skip).limit(limit).all()
    return computers

@app.get("/computers/{computer_id}", response_model=ComputerResponse)
def read_computer(computer_id: int, db: Session = Depends(get_db)):
    computer = db.query(Computer).filter(Computer.id == computer_id).first()
    if computer is None:
```

```
        raise HTTPException(status_code=404, detail="Computer not found")
    return computer

@app.put("/computers/{computer_id}", response_model=ComputerResponse)
def update_computer(computer_id: int, computer: ComputerCreate, db: Session =
Depends(get_db)):
    db_computer = db.query(Computer).filter(Computer.id == computer_id).first()
    if db_computer is None:
        raise HTTPException(status_code=404, detail="Computer not found")
    for key, value in computer.dict().items():
        setattr(db_computer, key, value)
    db.commit()
    db.refresh(db_computer)
    return db_computer

@app.delete("/computers/{computer_id}")
def delete_computer(computer_id: int, db: Session = Depends(get_db)):
    db_computer = db.query(Computer).filter(Computer.id == computer_id).first()
    if db_computer is None:
        raise HTTPException(status_code=404, detail="Computer not found")
    db.delete(db_computer)
    db.commit()
    return {"message": "Computer deleted successfully"}


@app.post("/software/", response_model=SoftwareResponse)
def create_software(software: SoftwareCreate, db: Session = Depends(get_db)):
    db_software = Software(**software.dict())
    db.add(db_software)
    db.commit()
    db.refresh(db_software)
    return db_software

@app.get("/software/", response_model=List[SoftwareResponse])
def read_software(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    software = db.query(Software).offset(skip).limit(limit).all()
    return software

@app.get("/software/{software_id}", response_model=SoftwareResponse)
def read_software_item(software_id: int, db: Session = Depends(get_db)):
    software = db.query(Software).filter(Software.id == software_id).first()
    if software is None:
        raise HTTPException(status_code=404, detail="Software not found")
    return software

@app.put("/software/{software_id}", response_model=SoftwareResponse)
def update_software(software_id: int, software: SoftwareCreate, db: Session =
Depends(get_db)):
    db_software = db.query(Software).filter(Software.id == software_id).first()
    if db_software is None:
        raise HTTPException(status_code=404, detail="Software not found")
    for key, value in software.dict().items():
        setattr(db_software, key, value)
    db.commit()
    db.refresh(db_software)
    return db_software

@app.delete("/software/{software_id}")
def delete_software(software_id: int, db: Session = Depends(get_db)):
    db_software = db.query(Software).filter(Software.id == software_id).first()
    if db_software is None:
        raise HTTPException(status_code=404, detail="Software not found")
    db.delete(db_software)
    db.commit()
    return {"message": "Software deleted successfully"}


@app.post("/users/", response_model=UserResponse)
def create_user(user: UserCreate, db: Session = Depends(get_db)):
    db_user = User(**user.dict())
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

@app.get("/users/", response_model=List[UserResponse])
def read_users(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
```

```
    users = db.query(User).offset(skip).limit(limit).all()
    return users

@app.get("/users/{user_id}", response_model=UserResponse)
def read_user(user_id: int, db: Session = Depends(get_db)):
    user = db.query(User).filter(User.id == user_id).first()
    if user is None:
        raise HTTPException(status_code=404, detail="User not found")
    return user

@app.put("/users/{user_id}", response_model=UserResponse)
def update_user(user_id: int, user: UserCreate, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.id == user_id).first()
    if db_user is None:
        raise HTTPException(status_code=404, detail="User not found")
    for key, value in user.dict().items():
        setattr(db_user, key, value)
    db.commit()
    db.refresh(db_user)
    return db_user

@app.delete("/users/{user_id}")
def delete_user(user_id: int, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.id == user_id).first()
    if db_user is None:
        raise HTTPException(status_code=404, detail="User not found")
    db.delete(db_user)
    db.commit()
    return {"message": "User deleted successfully"}


@app.post("/repairs/", response_model=RepairResponse)
def create_repair(repair: RepairCreate, db: Session = Depends(get_db)):
    db_repair = Repair(**repair.dict())
    db.add(db_repair)
    db.commit()
    db.refresh(db_repair)
    return db_repair

@app.get("/repairs/", response_model=List[RepairResponse])
def read_repairs(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    repairs = db.query(Repair).offset(skip).limit(limit).all()
    return repairs

@app.get("/repairs/{repair_id}", response_model=RepairResponse)
def read_repair(repair_id: int, db: Session = Depends(get_db)):
    repair = db.query(Repair).filter(Repair.id == repair_id).first()
    if repair is None:
        raise HTTPException(status_code=404, detail="Repair not found")
    return repair

@app.put("/repairs/{repair_id}", response_model=RepairResponse)
def update_repair(repair_id: int, repair: RepairCreate, db: Session = Depends(get_db)):
    db_repair = db.query(Repair).filter(Repair.id == repair_id).first()
    if db_repair is None:
        raise HTTPException(status_code=404, detail="Repair not found")
    for key, value in repair.dict().items():
        setattr(db_repair, key, value)
    db.commit()
    db.refresh(db_repair)
    return db_repair

@app.delete("/repairs/{repair_id}")
def delete_repair(repair_id: int, db: Session = Depends(get_db)):
    db_repair = db.query(Repair).filter(Repair.id == repair_id).first()
    if db_repair is None:
        raise HTTPException(status_code=404, detail="Repair not found")
    db.delete(db_repair)
    db.commit()
    return {"message": "Repair deleted successfully"}


@app.post("/schedules/", response_model=ScheduleResponse)
def create_schedule(schedule: ScheduleCreate, db: Session = Depends(get_db)):
    db_schedule = Schedule(**schedule.dict())
    db.add(db_schedule)
    db.commit()
```

```
    db.refresh(db_schedule)
    return db_schedule

@app.get("/schedules/", response_model=List[ScheduleResponse])
def read_schedules(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    schedules = db.query(Schedule).offset(skip).limit(limit).all()
    return schedules

@app.get("/schedules/{schedule_id}", response_model=ScheduleResponse)
def read_schedule(schedule_id: int, db: Session = Depends(get_db)):
    schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if schedule is None:
        raise HTTPException(status_code=404, detail="Schedule not found")
    return schedule

@app.put("/schedules/{schedule_id}", response_model=ScheduleResponse)
def update_schedule(schedule_id: int, schedule: ScheduleCreate, db: Session =
Depends(get_db)):
    db_schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if db_schedule is None:
        raise HTTPException(status_code=404, detail="Schedule not found")
    for key, value in schedule.dict().items():
        setattr(db_schedule, key, value)
    db.commit()
    db.refresh(db_schedule)
    return db_schedule

@app.delete("/schedules/{schedule_id}")
def delete_schedule(schedule_id: int, db: Session = Depends(get_db)):
    db_schedule = db.query(Schedule).filter(Schedule.id == schedule_id).first()
    if db_schedule is None:
        raise HTTPException(status_code=404, detail="Schedule not found")
    db.delete(db_schedule)
    db.commit()
    return {"message": "Schedule deleted successfully"}


@app.post("/computer-software/", response_model=ComputerSoftwareResponse)
def create_computer_software(cs: ComputerSoftwareCreate, db: Session = Depends(get_db)):
    db_cs = ComputerSoftware(**cs.dict())
    db.add(db_cs)
    db.commit()
    db.refresh(db_cs)
    return db_cs

@app.get("/computer-software/", response_model=List[ComputerSoftwareResponse])
def read_computer_software(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    cs = db.query(ComputerSoftware).offset(skip).limit(limit).all()
    return cs

@app.get("/computer-software/{cs_id}", response_model=ComputerSoftwareResponse)
def read_computer_software_item(cs_id: int, db: Session = Depends(get_db)):
    cs = db.query(ComputerSoftware).filter(ComputerSoftware.id == cs_id).first()
    if cs is None:
        raise HTTPException(status_code=404, detail="Computer-Software link not found")
    return cs

@app.put("/computer-software/{cs_id}", response_model=ComputerSoftwareResponse)
def update_computer_software(cs_id: int, cs: ComputerSoftwareCreate, db: Session =
Depends(get_db)):
    db_cs = db.query(ComputerSoftware).filter(ComputerSoftware.id == cs_id).first()
    if db_cs is None:
        raise HTTPException(status_code=404, detail="Computer-Software link not found")
    for key, value in cs.dict().items():
        setattr(db_cs, key, value)
    db.commit()
    db.refresh(db_cs)
    return db_cs

@app.delete("/computer-software/{cs_id}")
def delete_computer_software(cs_id: int, db: Session = Depends(get_db)):
    db_cs = db.query(ComputerSoftware).filter(ComputerSoftware.id == cs_id).first()
    if db_cs is None:
        raise HTTPException(status_code=404, detail="Computer-Software link not found")
    db.delete(db_cs)
    db.commit()
    return {"message": "Computer-Software link deleted successfully"}
```

```
@app.get("/computers/{computer_id}/software", response_model=List[SoftwareResponse])
def get_software_for_computer(computer_id: int, db: Session = Depends(get_db)):
    computer = db.query(Computer).filter(Computer.id == computer_id).first()
    if computer is None:
        raise HTTPException(status_code=404, detail="Computer not found")

    software = db.query(Software).join(ComputerSoftware).filter(ComputerSoftware.computer_id
== computer_id).all()
    return software

@app.get("/software/{software_id}/computers", response_model=List[ComputerResponse])
def get_computers_with_software(software_id: int, db: Session = Depends(get_db)):
    software = db.query(Software).filter(Software.id == software_id).first()
    if software is None:
        raise HTTPException(status_code=404, detail="Software not found")

    computers = db.query(Computer).join(ComputerSoftware).filter(ComputerSoftware.software_id
== software_id).all()
    return computers

@app.get("/users/{user_id}/schedules", response_model=List[ScheduleResponse])
def get_user_schedules(user_id: int, db: Session = Depends(get_db)):
    user = db.query(User).filter(User.id == user_id).first()
    if user is None:
        raise HTTPException(status_code=404, detail="User not found")

    schedules = db.query(Schedule).filter(Schedule.user_id == user_id).all()
    return schedules

@app.get("/computers/{computer_id}/schedules", response_model=List[ScheduleResponse])
def get_computer_schedules(computer_id: int, db: Session = Depends(get_db)):
    computer = db.query(Computer).filter(Computer.id == computer_id).first()
    if computer is None:
        raise HTTPException(status_code=404, detail="Computer not found")

    schedules = db.query(Schedule).filter(Schedule.computer_id == computer_id).all()
    return schedules
```

models.py

```
from sqlalchemy import create_engine, Column, Integer, String, Float, Text, Date, DateTime,
Boolean, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship

Base = declarative_base()

class Computer(Base):
    __tablename__ = 'computers'

    id = Column(Integer, primary_key=True, index=True)
    inventory_number = Column(String(50), unique=True, nullable=False)
    processor = Column(String(100))
    ram_gb = Column(Integer)
    storage_gb = Column(Integer)
    gpu = Column(String(100))
    operating_system = Column(String(100))
    purchase_date = Column(Date)
    status = Column(String(20))

    repairs = relationship("Repair", back_populates="computer")
    schedules = relationship("Schedule", back_populates="computer")
    software_installed = relationship("ComputerSoftware", back_populates="computer")

class Software(Base):
    __tablename__ = 'software'

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    version = Column(String(50))
    license_type = Column(String(50))
    license_expiration = Column(Date)
```

```
    required_ram_gb = Column(Integer)
    required_storage_gb = Column(Integer)

    computers = relationship("ComputerSoftware", back_populates="software")

class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True, index=True)
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    email = Column(String(100), unique=True)
    role = Column(String(20))
    department = Column(String(100))
    access_level = Column(Integer)

    repairs = relationship("Repair", back_populates="technician")
    schedules = relationship("Schedule", back_populates="user")

class Repair(Base):
    __tablename__ = 'repairs'

    id = Column(Integer, primary_key=True, index=True)
    computer_id = Column(Integer, ForeignKey('computers.id'))
    technician_id = Column(Integer, ForeignKey('users.id'))
    issue_description = Column(Text)
    repair_date = Column(Date)
    completion_date = Column(Date)
    status = Column(String(20))
    cost = Column(Float)

    computer = relationship("Computer", back_populates="repairs")
    technician = relationship("User", back_populates="repairs")

class Schedule(Base):
    __tablename__ = 'schedule'

    id = Column(Integer, primary_key=True, index=True)
    computer_id = Column(Integer, ForeignKey('computers.id'))
    user_id = Column(Integer, ForeignKey('users.id'))
    start_time = Column(DateTime)
    end_time = Column(DateTime)
    purpose = Column(String(200))
    status = Column(String(20))

    computer = relationship("Computer", back_populates="schedules")
    user = relationship("User", back_populates="schedules")

class ComputerSoftware(Base):
    __tablename__ = 'computer_software'

    id = Column(Integer, primary_key=True, index=True)
    computer_id = Column(Integer, ForeignKey('computers.id'))
    software_id = Column(Integer, ForeignKey('software.id'))
    installation_date = Column(Date)
    is_active = Column(Boolean)

    computer = relationship("Computer", back_populates="software_installed")
    software = relationship("Software", back_populates="computers")
```

Вывод: приобрёл практические навыки разработки API и баз данных