МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе No5

Специальность ПО11(о)

Выполнил
И. А. Головач,
студент группы ПО11

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«26» апрель 2025 г.

Брест 2025

Вариант 5

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;

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;

Выполнение:

Код программы:

main.py:

```
from fastapi import FastAPI, HTTPException, Depends
from sqlalchemy.orm import Session
from database import SessionLocal, init_db
from models import Base
from crud import (
    create_component, get_components, get_component,
    update_component, delete_component,
    create_category, get_categories, get_category,
    update_category, delete_category,
    create_manufacturer, get_manufacturers, get_manufacturer,
    update_manufacturer, delete_manufacturer,
    create_build, get_builds, get_build,
    update_build, delete_build,
    add_component_to_build, remove_component_from_build
)
import uvicorn

app = FastAPI()

# Инициализация БД
@app.on_event("startup")
def on_startup():
    init_db()

# Dependency для работы с базой данных
def get_db():
    db = SessionLocal()
    try:
```

```
        yield db
    finally:
        db.close()

# Эндпоинты для Component
@app.post("/components/")
def create_component_endpoint(component: dict, db: Session = Depends(get_db)):
    return create_component(db, component)

@app.get("/components/")
def get_components_endpoint(skip: int = 0, limit: int = 100, db: Session =
Depends(get_db)):
    return get_components(db, skip=skip, limit=limit)

@app.get("/components/{component_id}")
def get_component_endpoint(component_id: int, db: Session = Depends(get_db)):
    component = get_component(db, component_id)
    if not component:
        raise HTTPException(status_code=404, detail="Component not found")
    return component

@app.put("/components/{component_id}")
def update_component_endpoint(component_id: int, component: dict, db: Session =
Depends(get_db)):
    updated_component = update_component(db, component_id, component)
    if not updated_component:
        raise HTTPException(status_code=404, detail="Component not found")
    return updated_component

@app.delete("/components/{component_id}")
def delete_component_endpoint(component_id: int, db: Session = Depends(get_db)):
    component = delete_component(db, component_id)
    if not component:
        raise HTTPException(status_code=404, detail="Component not found")
    return {"message": "Component deleted"}

# Эндпоинты для Category
@app.post("/categories/")
def create_category_endpoint(category: dict, db: Session = Depends(get_db)):
    return create_category(db, category)

@app.get("/categories/")
def get_categories_endpoint(skip: int = 0, limit: int = 100, db: Session =
Depends(get_db)):
    return get_categories(db, skip=skip, limit=limit)

@app.get("/categories/{category_id}")
def get_category_endpoint(category_id: int, db: Session = Depends(get_db)):
    category = get_category(db, category_id)
    if not category:
        raise HTTPException(status_code=404, detail="Category not found")
    return category

@app.put("/categories/{category_id}")
def update_category_endpoint(category_id: int, category: dict, db: Session =
Depends(get_db)):
    updated_category = update_category(db, category_id, category)
    if not updated_category:
        raise HTTPException(status_code=404, detail="Category not found")
```

```
    return updated_category

@app.delete("/categories/{category_id}")
def delete_category_endpoint(category_id: int, db: Session = Depends(get_db)):
    category = delete_category(db, category_id)
    if not category:
        raise HTTPException(status_code=404, detail="Category not found")
    return {"message": "Category deleted"}

# Эндпоинты для Manufacturer
@app.post("/manufacturers/")
def create_manufacturer_endpoint(manufacturer: dict, db: Session = Depends(get_db)):
    return create_manufacturer(db, manufacturer)

@app.get("/manufacturers/")
def get_manufacturers_endpoint(skip: int = 0, limit: int = 100, db: Session =
Depends(get_db)):
    return get_manufacturers(db, skip=skip, limit=limit)

@app.get("/manufacturers/{manufacturer_id}")
def get_manufacturer_endpoint(manufacturer_id: int, db: Session = Depends(get_db)):
    manufacturer = get_manufacturer(db, manufacturer_id)
    if not manufacturer:
        raise HTTPException(status_code=404, detail="Manufacturer not found")
    return manufacturer

@app.put("/manufacturers/{manufacturer_id}")
def update_manufacturer_endpoint(manufacturer_id: int, manufacturer: dict, db: Session
= Depends(get_db)):
    updated_manufacturer = update_manufacturer(db, manufacturer_id, manufacturer)
    if not updated_manufacturer:
        raise HTTPException(status_code=404, detail="Manufacturer not found")
    return updated_manufacturer

@app.delete("/manufacturers/{manufacturer_id}")
def delete_manufacturer_endpoint(manufacturer_id: int, db: Session = Depends(get_db)):
    manufacturer = delete_manufacturer(db, manufacturer_id)
    if not manufacturer:
        raise HTTPException(status_code=404, detail="Manufacturer not found")
    return {"message": "Manufacturer deleted"}

# Эндпоинты для Build
@app.post("/builds/")
def create_build_endpoint(build: dict, db: Session = Depends(get_db)):
    return create_build(db, build)

@app.get("/builds/")
def get_builds_endpoint(skip: int = 0, limit: int = 100, db: Session =
Depends(get_db)):
    return get_builds(db, skip=skip, limit=limit)

@app.get("/builds/{build_id}")
def get_build_endpoint(build_id: int, db: Session = Depends(get_db)):
    build = get_build(db, build_id)
    if not build:
        raise HTTPException(status_code=404, detail="Build not found")
    return build

@app.put("/builds/{build_id}")
```

```
def update_build_endpoint(build_id: int, build: dict, db: Session = Depends(get_db)):
    updated_build = update_build(db, build_id, build)
    if not updated_build:
        raise HTTPException(status_code=404, detail="Build not found")
    return updated_build

@app.delete("/builds/{build_id}")
def delete_build_endpoint(build_id: int, db: Session = Depends(get_db)):
    build = delete_build(db, build_id)
    if not build:
        raise HTTPException(status_code=404, detail="Build not found")
    return {"message": "Build deleted"}

# Эндпоинты для управления компонентами в сборке
@app.post("/builds/{build_id}/components/{component_id}")
def add_component_endpoint(build_id: int, component_id: int, db: Session =
Depends(get_db)):
    build = add_component_to_build(db, build_id, component_id)
    if not build:
        raise HTTPException(status_code=404, detail="Build or Component not found")
    return build

@app.delete("/builds/{build_id}/components/{component_id}")
def remove_component_endpoint(build_id: int, component_id: int, db: Session =
Depends(get_db)):
    build = remove_component_from_build(db, build_id, component_id)
    if not build:
        raise HTTPException(status_code=404, detail="Build or Component not found")
    return build

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

crud.py:

```
from sqlalchemy.orm import Session
from models import Component, Category, Manufacturer, Build

# CRUD для Component
def create_component(db: Session, component_data: dict):
    db_component = Component(**component_data)
    db.add(db_component)
    db.commit()
    db.refresh(db_component)
    return db_component

def get_components(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Component).offset(skip).limit(limit).all()

def get_component(db: Session, component_id: int):
    return db.query(Component).filter(Component.id == component_id).first()

def update_component(db: Session, component_id: int, component_data: dict):
    db_component = db.query(Component).filter(Component.id == component_id).first()
    if db_component:
        for key, value in component_data.items():
            setattr(db_component, key, value)
```

```
        db.commit()
        db.refresh(db_component)
    return db_component

def delete_component(db: Session, component_id: int):
    db_component = db.query(Component).filter(Component.id == component_id).first()
    if db_component:
        db.delete(db_component)
        db.commit()
    return db_component

# CRUD для Category
def create_category(db: Session, category_data: dict):
    db_category = Category(**category_data)
    db.add(db_category)
    db.commit()
    db.refresh(db_category)
    return db_category

def get_categories(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Category).offset(skip).limit(limit).all()

def get_category(db: Session, category_id: int):
    return db.query(Category).filter(Category.id == category_id).first()

def update_category(db: Session, category_id: int, category_data: dict):
    db_category = db.query(Category).filter(Category.id == category_id).first()
    if db_category:
        for key, value in category_data.items():
            setattr(db_category, key, value)
        db.commit()
        db.refresh(db_category)
    return db_category

def delete_category(db: Session, category_id: int):
    db_category = db.query(Category).filter(Category.id == category_id).first()
    if db_category:
        db.delete(db_category)
        db.commit()
    return db_category

# CRUD для Manufacturer
def create_manufacturer(db: Session, manufacturer_data: dict):
    db_manufacturer = Manufacturer(**manufacturer_data)
    db.add(db_manufacturer)
    db.commit()
    db.refresh(db_manufacturer)
    return db_manufacturer

def get_manufacturers(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Manufacturer).offset(skip).limit(limit).all()

def get_manufacturer(db: Session, manufacturer_id: int):
    return db.query(Manufacturer).filter(Manufacturer.id == manufacturer_id).first()

def update_manufacturer(db: Session, manufacturer_id: int, manufacturer_data: dict):
    db_manufacturer = db.query(Manufacturer).filter(Manufacturer.id ==
manufacturer_id).first()
    if db_manufacturer:
```

```
        for key, value in manufacturer_data.items():
            setattr(db_manufacturer, key, value)
        db.commit()
        db.refresh(db_manufacturer)
    return db_manufacturer

def delete_manufacturer(db: Session, manufacturer_id: int):
    db_manufacturer = db.query(Manufacturer).filter(Manufacturer.id ==
manufacturer_id).first()
    if db_manufacturer:
        db.delete(db_manufacturer)
        db.commit()
    return db_manufacturer

# CRUD для Build
def create_build(db: Session, build_data: dict):
    db_build = Build(**build_data)
    db.add(db_build)
    db.commit()
    db.refresh(db_build)
    return db_build

def get_builds(db: Session, skip: int = 0, limit: int = 100):
    return db.query(Build).offset(skip).limit(limit).all()

def get_build(db: Session, build_id: int):
    return db.query(Build).filter(Build.id == build_id).first()

def update_build(db: Session, build_id: int, build_data: dict):
    db_build = db.query(Build).filter(Build.id == build_id).first()
    if db_build:
        for key, value in build_data.items():
            setattr(db_build, key, value)
        db.commit()
        db.refresh(db_build)
    return db_build

def delete_build(db: Session, build_id: int):
    db_build = db.query(Build).filter(Build.id == build_id).first()
    if db_build:
        db.delete(db_build)
        db.commit()
    return db_build

def add_component_to_build(db: Session, build_id: int, component_id: int):
    build = db.query(Build).filter(Build.id == build_id).first()
    component = db.query(Component).filter(Component.id == component_id).first()
    if build and component:
        build.components.append(component)
        db.commit()
        db.refresh(build)
    return build

def remove_component_from_build(db: Session, build_id: int, component_id: int):
    build = db.query(Build).filter(Build.id == build_id).first()
    component = db.query(Component).filter(Component.id == component_id).first()
    if build and component:
        build.components.remove(component)
        db.commit()
```

```
        db.refresh(build)
    return build
```

const.py:

```
from dataclasses import dataclass

@dataclass
class Const:
    DATABASE_URL: str = "sqlite:///./computer_build.db"
```

database.py:

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from const import Const

engine = create_engine(Const.DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

def init_db():
    Base.metadata.create_all(bind=engine)
```

models.py:

```
from sqlalchemy import Column, Integer, String, Float, ForeignKey, Table
from sqlalchemy.orm import relationship
from database import Base

# Связь многие-ко-многим между Build и Component
build_component = Table(
    'build_component', Base.metadata,
    Column('build_id', Integer, ForeignKey('builds.id')),
    Column('component_id', Integer, ForeignKey('components.id'))
)

class Category(Base):
    __tablename__ = "categories"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(50), nullable=False, unique=True)
    components = relationship("Component", back_populates="category")

class Manufacturer(Base):
    __tablename__ = "manufacturers"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False, unique=True)
    website = Column(String(200))
    components = relationship("Component", back_populates="manufacturer")

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

```
    name = Column(String(100), nullable=False)
    description = Column(String(250))
    price = Column(Float, nullable=False)
    category_id = Column(Integer, ForeignKey("categories.id"))
    manufacturer_id = Column(Integer, ForeignKey("manufacturers.id"))
    category = relationship("Category", back_populates="components")
    manufacturer = relationship("Manufacturer", back_populates="components")
    builds = relationship("Build", secondary=build_component,
back_populates="components")

class Build(Base):
    __tablename__ = "builds"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    description = Column(String(250))
    total_price = Column(Float)
    components = relationship("Component", secondary=build_component,
back_populates="builds")
```

test.http:

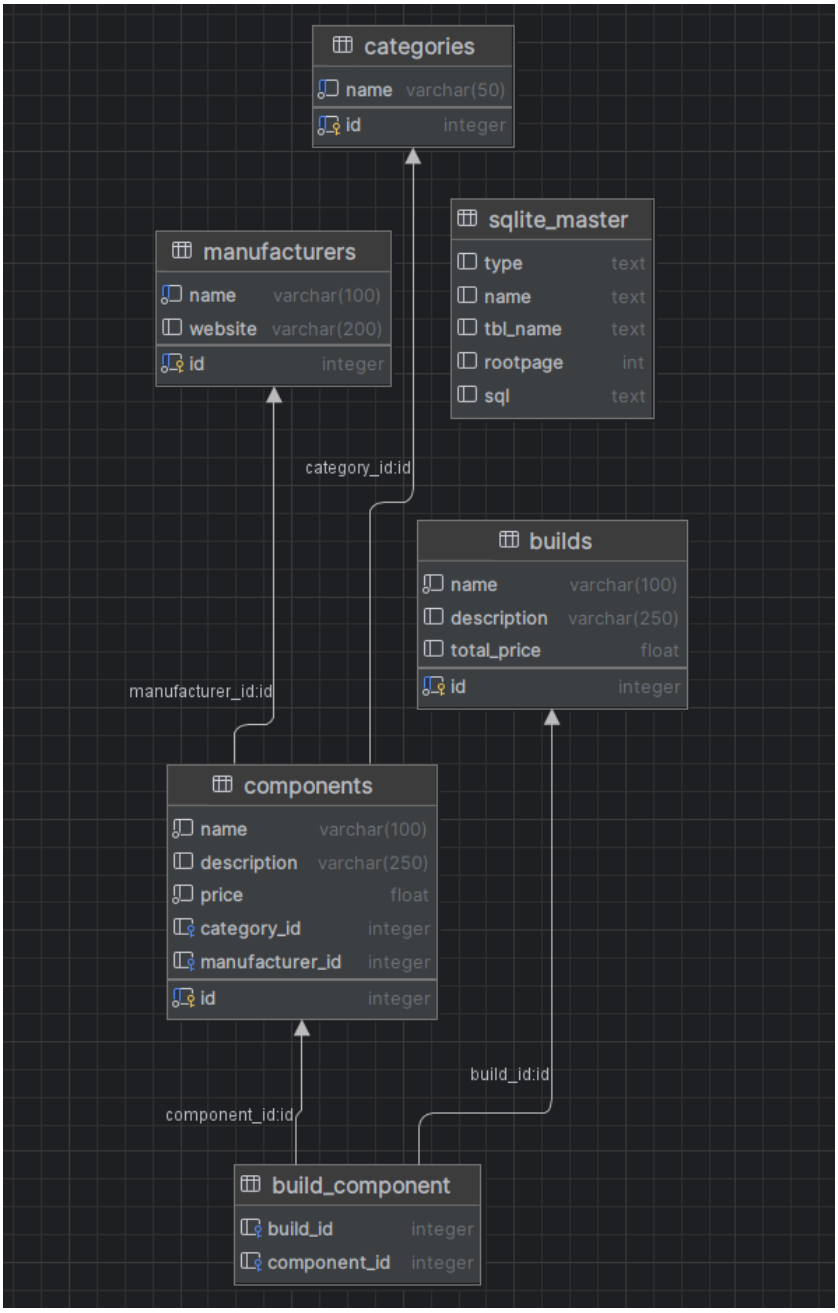
```
# Test your FastAPI endpoints for Computer Build System

# Components
POST http://127.0.0.1:8000/components/
Accept: application/json
Content-Type: application/json

{
    "name": "Intel Core i9-13900K",
    "description": "24-ядерный процессор",
    "price": 600.0,
    "category_id": 1,
    "manufacturer_id": 1
}

###

GET http://127.0.0.1:8000/components/
Accept: application/json

###

GET http://127.0.0.1:8000/components/1
Accept: application/json

###

PUT http://127.0.0.1:8000/components/1
Accept: application/json
Content-Type: application/json

{
    "name": "Intel Core i9-13900KS",
    "price": 650.0
}
```

```
###

DELETE http://127.0.0.1:8000/components/1
Accept: application/json

###

# Categories
POST http://127.0.0.1:8000/categories/
Accept: application/json
Content-Type: application/json

{
    "name": "Блоки питания"
}

###

GET http://127.0.0.1:8000/categories/
Accept: application/json

###

GET http://127.0.0.1:8000/categories/1
Accept: application/json

###

PUT http://127.0.0.1:8000/categories/1
Accept: application/json
Content-Type: application/json

{
    "name": "Процессоры (CPU)"
}

###

DELETE http://127.0.0.1:8000/categories/1
Accept: application/json

###

# Manufacturers
POST http://127.0.0.1:8000/manufacturers/
Accept: application/json
Content-Type: application/json

{
    "name": "ASUS",
    "website": "https://www.asus.com"
}

###

GET http://127.0.0.1:8000/manufacturers/
Accept: application/json

###
```

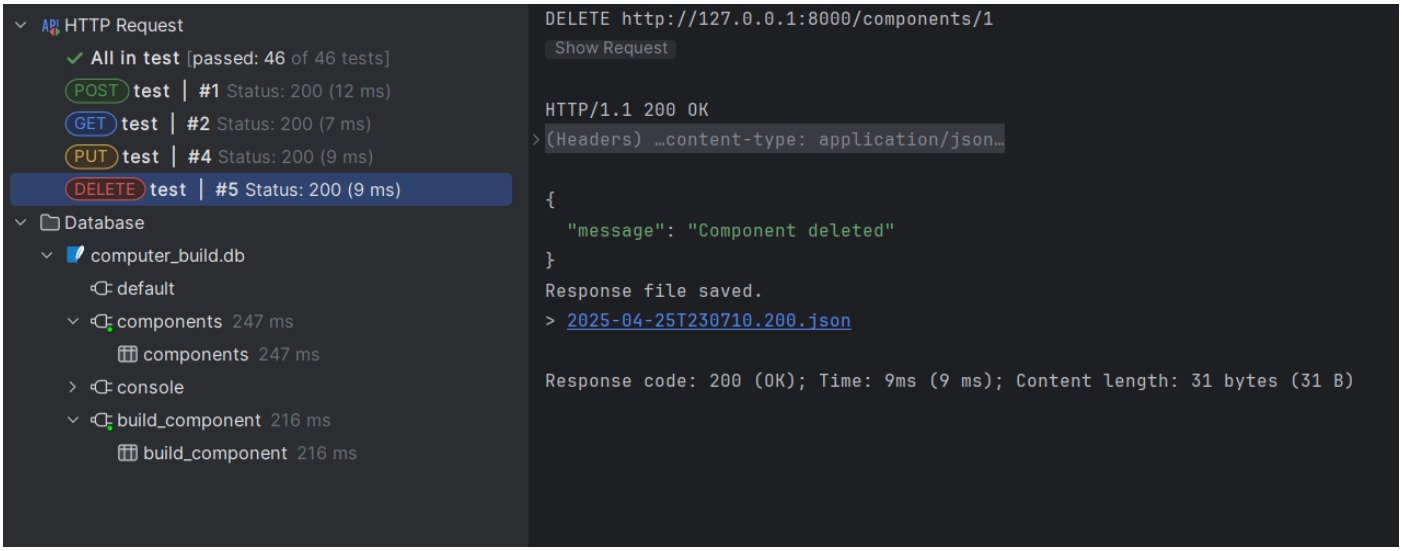
```
GET http://127.0.0.1:8000/manufacturers/1
Accept: application/json

###

PUT http://127.0.0.1:8000/manufacturers/1
Accept: application/json
Content-Type: application/json

{
    "name": "ASUS ROG",
    "website": "https://rog.asus.com"
}

###

DELETE http://127.0.0.1:8000/manufacturers/1
Accept: application/json

###

# Builds
POST http://127.0.0.1:8000/builds/
Accept: application/json
Content-Type: application/json

{
    "name": "Стримерский ПК",
    "description": "Мощный ПК для стриминга",
    "total_price": 2500.0
}

###

GET http://127.0.0.1:8000/builds/
Accept: application/json

###

GET http://127.0.0.1:8000/builds/1
Accept: application/json

###

PUT http://127.0.0.1:8000/builds/1
Accept: application/json
Content-Type: application/json

{
    "name": "Профессиональный стримерский ПК",
    "total_price": 2700.0
}

###

DELETE http://127.0.0.1:8000/builds/1
Accept: application/json
```

```
###

# Build Components Management
POST http://127.0.0.1:8000/builds/1/components/2
Accept: application/json
Content-Type: application/json

###

DELETE http://127.0.0.1:8000/builds/1/components/2
Accept: application/json

###

# Get Build with Components
GET http://127.0.0.1:8000/builds/1
Accept: application/json
```

Результаты работы программы:



```
  @app.on_event("startup")
INFO:     Started server process [22092]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```



Использовал SQLite для данной лабораторной работы.

Вывод: приобрёл практические навыки разработки API и базы данных.