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
К. А. Головач,
студент группы ПО11

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

Брест 2025

Вариант 6

Цель работы: приобрести практические навыки разработки API и баз данных

Общее задание:

1. Реализовать базу данных из не менее 5 таблиц на заданную тематику 6) База данных Библиотека. 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 crud import (
    create_book, get_books, get_book, update_book, delete_book,
    create_genre, get_genres, get_genre, update_genre, delete_genre,
    create_author, get_authors, get_author, update_author, delete_author,
    create_collection, get_collections, get_collection, update_collection, delete_collection,
    add_book_to_collection, remove_book_from_collection
)
import uvicorn

app = FastAPI()

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

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

# Эндпоинты для Book
@app.post("/books/")
def create_book_endpoint(book: dict, db: Session = Depends(get_db)):
    return create_book(db, book)

@app.get("/books/")
def get_books_endpoint(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    return get_books(db, skip=skip, limit=limit)

@app.get("/books/{book_id}")
def get_book_endpoint(book_id: int, db: Session = Depends(get_db)):
    book = get_book(db, book_id)
    if not book:
        raise HTTPException(status_code=404, detail="Book not found")
    return book

@app.put("/books/{book_id}")
def update_book_endpoint(book_id: int, book: dict, db: Session = Depends(get_db)):
    updated_book = update_book(db, book_id, book)
    if not updated_book:
        raise HTTPException(status_code=404, detail="Book not found")
    return updated_book

@app.delete("/books/{book_id}")
def delete_book_endpoint(book_id: int, db: Session = Depends(get_db)):
    book = delete_book(db, book_id)
    if not book:
        raise HTTPException(status_code=404, detail="Book not found")
    return {"message": "Book deleted"}

# Аналогично создаем эндпоинты для Genre, Author, Collection и операций с книгами в подборках.
```

crud.py:

```
from sqlalchemy.orm import Session
from models import Book, Genre, Author, Collection

# CRUD для Book
def create_book(db: Session, book_data: dict):
    db_book = Book(**book_data)
    db.add(db_book)
    db.commit()
    db.refresh(db_book)
    return db_book

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

def get_book(db: Session, book_id: int):
    return db.query(Book).filter(Book.id == book_id).first()

def update_book(db: Session, book_id: int, book_data: dict):
    db_book = db.query(Book).filter(Book.id == book_id).first()
    if db_book:
        for key, value in book_data.items():
            setattr(db_book, key, value)
        db.commit()
        db.refresh(db_book)
    return db_book

def delete_book(db: Session, book_id: int):
    db_book = db.query(Book).filter(Book.id == book_id).first()
    if db_book:
        db.delete(db_book)
        db.commit()
    return db_book

# CRUD для Genre
def create_genre(db: Session, genre_data: dict):
    db_genre = Genre(**genre_data)
    db.add(db_genre)
    db.commit()
    db.refresh(db_genre)
    return db_genre

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

def get_genre(db: Session, genre_id: int):
    return db.query(Genre).filter(Genre.id == genre_id).first()

def update_genre(db: Session, genre_id: int, genre_data: dict):
    db_genre = db.query(Genre).filter(Genre.id == genre_id).first()
    if db_genre:
        for key, value in genre_data.items():
            setattr(db_genre, key, value)
        db.commit()
        db.refresh(db_genre)
    return db_genre

def delete_genre(db: Session, genre_id: int):
    db_genre = db.query(Genre).filter(Genre.id == genre_id).first()
    if db_genre:
        db.delete(db_genre)
        db.commit()
    return db_genre

# CRUD для Author
def create_author(db: Session, author_data: dict):
    db_author = Author(**author_data)
    db.add(db_author)
    db.commit()
    db.refresh(db_author)
    return db_author

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

def get_author(db: Session, author_id: int):
    return db.query(Author).filter(Author.id == author_id).first()

def update_author(db: Session, author_id: int, author_data: dict):
    db_author = db.query(Author).filter(Author.id == author_id).first()
    if db_author:
        for key, value in author_data.items():
            setattr(db_author, key, value)
        db.commit()
        db.refresh(db_author)
    return db_author

def delete_author(db: Session, author_id: int):
    db_author = db.query(Author).filter(Author.id == author_id).first()
    if db_author:
        db.delete(db_author)
        db.commit()
    return db_author

# CRUD для Collection
def create_collection(db: Session, collection_data: dict):
    db_collection = Collection(**collection_data)
    db.add(db_collection)
    db.commit()
    db.refresh(db_collection)
    return db_collection

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

def get_collection(db: Session, collection_id: int):
    return db.query(Collection).filter(Collection.id == collection_id).first()

def update_collection(db: Session, collection_id: int, collection_data: dict):
    db_collection = db.query(Collection).filter(Collection.id == collection_id).first()
    if db_collection:
        for key, value in collection_data.items():
            setattr(db_collection, key, value)
        db.commit()
        db.refresh(db_collection)
    return db_collection

def delete_collection(db: Session, collection_id: int):
    db_collection = db.query(Collection).filter(Collection.id == collection_id).first()
    if db_collection:
        db.delete(db_collection)
        db.commit()
    return db_collection

# Добавление и удаление книги в подборку
def add_book_to_collection(db: Session, collection_id: int, book_id: int):
    collection = db.query(Collection).filter(Collection.id == collection_id).first()
    book = db.query(Book).filter(Book.id == book_id).first()
    if collection and book:
        collection.books.append(book)
        db.commit()
        db.refresh(collection)
    return collection

def remove_book_from_collection(db: Session, collection_id: int, book_id: int):
    collection = db.query(Collection).filter(Collection.id == collection_id).first()
    book = db.query(Book).filter(Book.id == book_id).first()
    if collection and book:
        collection.books.remove(book)
        db.commit()
        db.refresh(collection)
    return collection
```

const.py:

```
from dataclasses import dataclass

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

database.py:

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

# Создаем движок для работы с базой данных
engine = create_engine(Const.DATABASE_URL, connect_args={"check_same_thread": False})

# Создаем локальную сессию для работы с БД
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Базовый класс для моделей
Base = declarative_base()

# Функция для инициализации базы данных
def init_db():
    Base.metadata.create_all(bind=engine)
```

models.py:

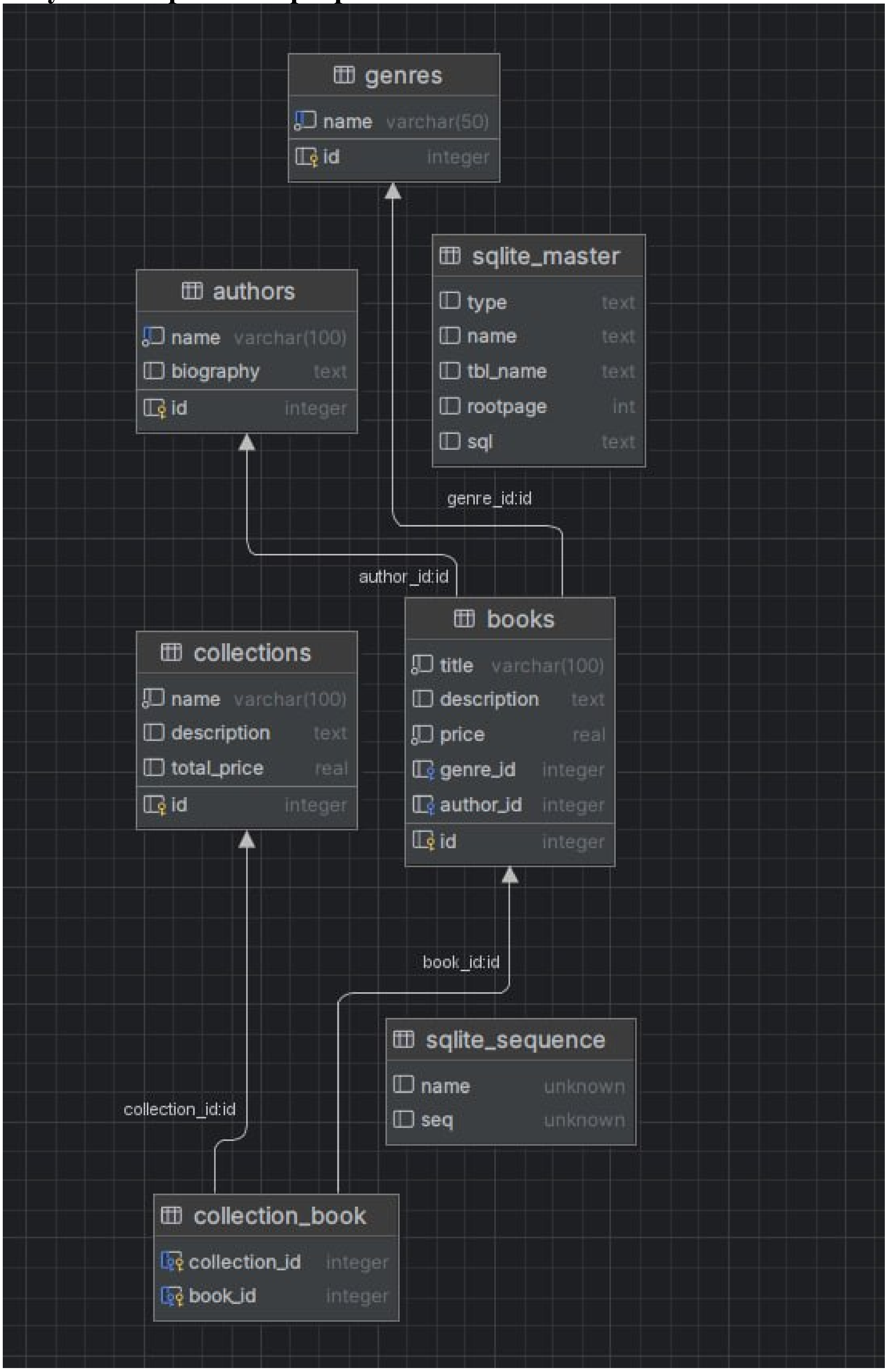
```
from sqlalchemy import Column, Integer, String, Float, ForeignKey, Table
from sqlalchemy.orm import relationship
from database import Base

# Связь многие-ко-многим между Collection и Book
collection_book = Table(
    'collection_book', Base.metadata,
    Column('collection_id', Integer, ForeignKey('collections.id')),
    Column('book_id', Integer, ForeignKey('books.id'))
)

class Genre(Base):
    __tablename__ = "genres"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(50), nullable=False, unique=True)
    books = relationship("Book", back_populates="genre")

class Author(Base):
    __tablename__ = "authors"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False, unique=True)
    biography = Column(String(500))
    books = relationship("Book", back_populates="author")

class Book(Base):
    __tablename__ = "books"
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String(100), nullable=False)
    description = Column(String(250))
    price = Column(Float, nullable=False)
    genre_id = Column(Integer, ForeignKey("genres.id"))
    author_id = Column(Integer, ForeignKey("authors.id"))
    genre = relationship("Genre", back_populates="books")
    author = relationship("Author", back_populates="books")
    collections = relationship("Collection", secondary=collection_book, back_populates="books")

class Collection(Base):
    __tablename__ = "collections"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    description = Column(String(250))
    total_price = Column(Float)
    books = relationship("Book", secondary=collection_book, back_populates="collections")
```

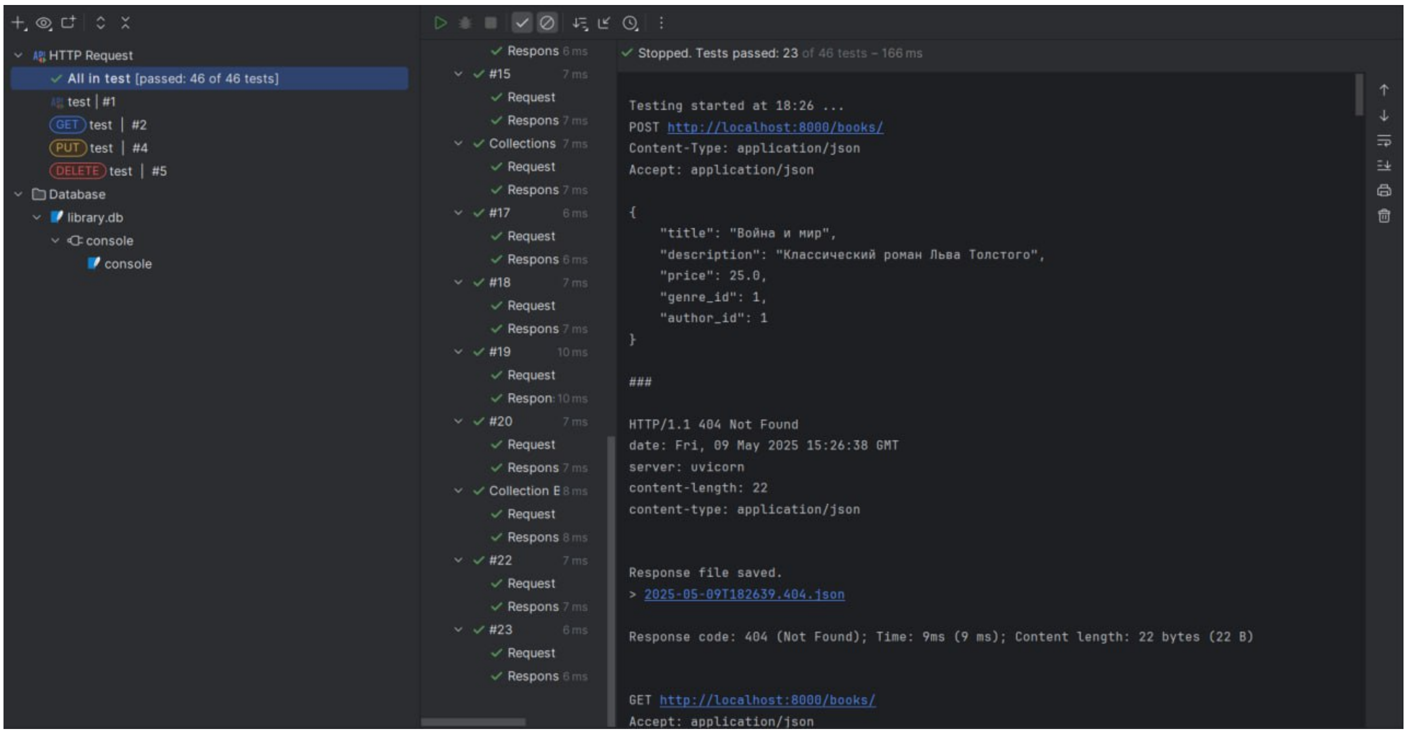
test.http:

```
### Books
# Create a new book
POST http://localhost:8000/books/
Content-Type: application/json
Accept: application/json

{
    "title": "Война и мир",
    "description": "Классический роман Льва Толстого",
    "price": 25.0,
    "genre_id": 1,
    "author_id": 1
}

###
# Get all books
GET http://localhost:8000/books/
Accept: application/json

###
# Get a specific book
GET http://localhost:8000/books/1
Accept: application/json

###
# Update a book
PUT http://localhost:8000/books/1
Content-Type: application/json
Accept: application/json

{
    "title": "Анна Каренина",
    "price": 20.0
}

###
# Delete a book
DELETE http://localhost:8000/books/1
Accept: application/json

### Genres
# Create a new genre
POST http://localhost:8000/genres/
Content-Type: application/json
Accept: application/json

{
    "name": "Роман"
}

###
# Get all genres
GET http://localhost:8000/genres/
Accept: application/json

###
# Get a specific genre
GET http://localhost:8000/genres/1
Accept: application/json

###
# Update a genre
PUT http://localhost:8000/genres/1
Content-Type: application/json
Accept: application/json

{
    "name": "Драма"
}

###
# Delete a genre
DELETE http://localhost:8000/genres/1
Accept: application/json

### Authors
# Create a new author
POST http://localhost:8000/authors/
Content-Type: application/json
Accept: application/json

{
    "name": "Лев Толстой",
    "biography": "Русский писатель, философ и мыслитель"
}

###
# Get all authors
GET http://localhost:8000/authors/
Accept: application/json

###
# Get a specific author
GET http://localhost:8000/authors/1
Accept: application/json

###
# Update an author
PUT http://localhost:8000/authors/1
Content-Type: application/json
Accept: application/json

{
    "name": "Фёдор Достоевский",
    "biography": "Русский писатель и философ"
}
```

```
###
# Delete an author
DELETE http://localhost:8000/authors/1
Accept: application/json

### Collections
# Create a new collection
POST http://localhost:8000/collections/
Content-Type: application/json
Accept: application/json

{
    "name": "Классика русской литературы",
    "description": "Подборка лучших произведений русских авторов",
    "total_price": 100.0
}

###
# Get all collections
GET http://localhost:8000/collections/
Accept: application/json

###
# Get a specific collection
GET http://localhost:8000/collections/1
Accept: application/json

###
# Update a collection
PUT http://localhost:8000/collections/1
Content-Type: application/json
Accept: application/json

{
    "name": "Мировая классика",
    "total_price": 120.0
}

###
# Delete a collection
DELETE http://localhost:8000/collections/1
Accept: application/json

### Collection Books Management
# Add a book to collection
POST http://localhost:8000/collections/1/books/2
Accept: application/json

###
# Remove a book from collection
DELETE http://localhost:8000/collections/1/books/2
Accept: application/json

###
# Get collection with books
GET http://localhost:8000/collections/1
Accept: application/json
```

Результаты работы программы:

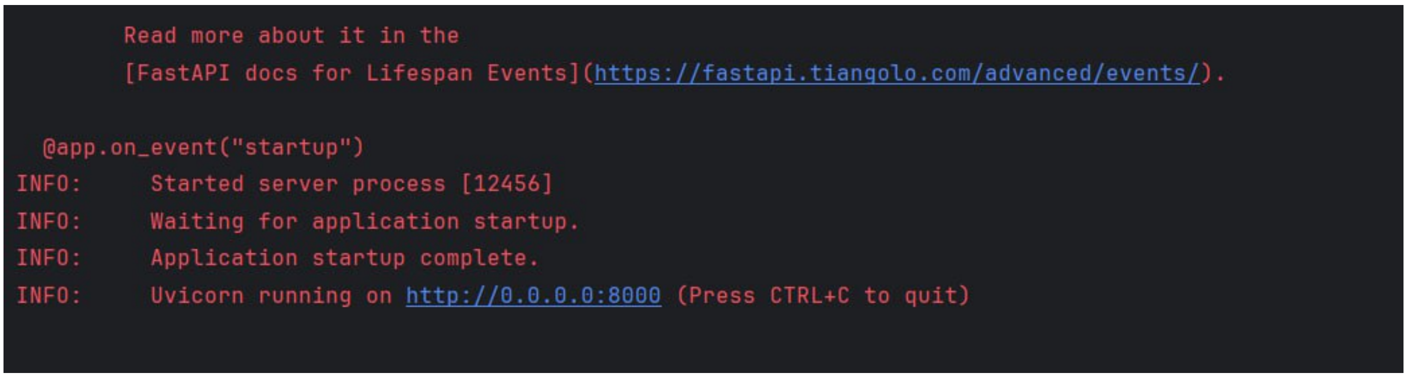




```
HTTP Request
  ✓ All in test [passed: 46 of 46 tests]
    test | #1
    GET test | #2
    PUT test | #4
    DELETE test | #5
Database
  library.db
    console
      console
```

```
  ✓ Respons 6 ms
✓ #15 7 ms
  ✓ Request
  ✓ Respons 7 ms
✓ Collections 7 ms
  ✓ Request
  ✓ Respons 7 ms
✓ #17 6 ms
  ✓ Request
  ✓ Respons 6 ms
✓ #18 7 ms
  ✓ Request
  ✓ Respons 7 ms
✓ #19 10 ms
  ✓ Request
  ✓ Respon: 10 ms
✓ #20 7 ms
  ✓ Request
  ✓ Respons 7 ms
✓ Collection E 8 ms
  ✓ Request
  ✓ Respons 8 ms
✓ #22 7 ms
  ✓ Request
  ✓ Respons 7 ms
✓ #23 6 ms
  ✓ Request
  ✓ Respons 6 ms
```

```
✓ Stopped. Tests passed: 23 of 46 tests – 166 ms

Testing started at 18:26 ...
POST http://localhost:8000/books/
Content-Type: application/json
Accept: application/json

{
    "title": "Война и мир",
    "description": "Классический роман Льва Толстого",
    "price": 25.0,
    "genre_id": 1,
    "author_id": 1
}

###

HTTP/1.1 404 Not Found
date: Fri, 09 May 2025 15:26:38 GMT
server: uvicorn
content-length: 22
content-type: application/json


Response file saved.
> 2025-05-09T182639.404.json

Response code: 404 (Not Found); Time: 9ms (9 ms); Content length: 22 bytes (22 B)


GET http://localhost:8000/books/
Accept: application/json
```



```
        Read more about it in the
        [FastAPI docs for Lifespan Events](https://fastapi.tiangolo.com/advanced/events/).

  @app.on_event("startup")
INFO:     Started server process [12456]
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.