# Интеллектуальные информационные системы

Практическое задание 5

**Кафедра информатики**
**Институт кибербезопасности и цифровых технологий**
**РТУ МИРЭА**

# Разработка проекта для реализации  API сервиса «Библиотека»

В продолжении предыдущий практики мы создадим API сервис который реализует базовую логику из предметной области **"Библиотека"**, которая включает в себя сущности:

1. Книга
2. Читатель
3. Выдача

Со следующей логикой:
Каждый **Читатель** может получить несколько **Книг**, каждая книга может быть выдана только одному читателю.
Запись выданных книг читателю ведется через сущность **Выдача**.

У нас получится следующая структура проекта:

```
├── app/
│   ├── __init__.py
│   ├── main.py              # Точка входа в приложение
│   ├── api/                 # Маршруты API
│   │   ├── __init__.py
│   │   ├── books.py         # Маршруты для сущности "Книга"
│   │   ├── readers.py       # Маршруты для сущности "Читатель"
│   │   └── issues.py        # Маршруты для сущности "Выдача"
│   ├── core/                # Основная конфигурация
│   │   ├── __init__.py
│   │   └── config.py        # Настройки приложения
│   ├── db/                  # Работа с базой данных
│   │   ├── __init__.py
│   │   ├── base.py          # Базовые модели и инициализация БД
│   │   ├── session.py       # Сессия SQLAlchemy
│   │   └── models.py        # ORM-модели (Книга, Читатель, Выдача)
│   ├── models/              # Pydantic-схемы
│   │   ├── __init__.py
│   │   ├── book.py          # Схемы для "Книга"
│   │   ├── reader.py        # Схемы для "Читатель"
│   │   └── issue.py         # Схемы для "Выдача"
│   ├── services/            # Бизнес-логика
│   │   ├── __init__.py
│   │   ├── books_service.py   # Логика для книг
│   │   ├── readers_service.py # Логика для читателей
│   │   └── issues_service.py  # Логика для выдачи
│   └── tests/               # Тесты
│       ├── __init__.py
│       ├── test_books.py    # Тесты для книг
│       ├── test_readers.py  # Тесты для читателей
│       └── test_issues.py   # Тесты для выдачи
├── requirements.txt         # Зависимости проекта
```

# Разработка проекта для реализации  API сервиса «Библиотека»

- Для этого:
- 1. Создайте новую папку
  2. Внутри нее создайте виртуальное окружение Python также в прошлых практических работах
  3. Откройте папку в Visual Studio code
  4. Создайте структуру папок такую как показано на рисунке выше.
  5. В папке **api** создайте файлы:

**__init__.py**

**books.py**

```python
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from app.services.books_service import BookService
from app.db.session import get_db
from app.models.book import BookCreate, BookResponse

router = APIRouter()

@router.post("/", response_model=BookResponse)
def create_book(book: BookCreate, db: Session = Depends(get_db)):
    """Создать книгу"""
    return BookService.create_book(db, book)

@router.get("/{id}", response_model=BookResponse)
def get_book(id: int, db: Session = Depends(get_db)):
    """Получить информацию о книге"""
    return BookService.get_book(db, id)
```

issues.py

```python
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from app.services.issues_service import IssueService
from app.db.session import get_db
from app.models.issue import IssueCreate, IssueResponse

router = APIRouter()

@router.post("/", response_model=IssueResponse)
def create_issue(issue: IssueCreate, db: Session = Depends(get_db)):
    """Создать выдачу книги читателю"""
    try:
        return IssueService.create_issue(db, issue)
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))

@router.get("/{id}", response_model=IssueResponse)
def get_issue(id: int, db: Session = Depends(get_db)):
    """Получить информацию о выдаче"""
    try:
        return IssueService.get_issue(db, id)
    except ValueError as e:
        raise HTTPException(status_code=404, detail=str(e))

@router.delete("/{id}")
def close_issue(id: int, db: Session = Depends(get_db)):
    """Закрыть выдачу (вернуть книгу)"""
    try:
        return IssueService.close_issue(db, id)
    except ValueError as e:
        raise HTTPException(status_code=404, detail=str(e))
```

readers.py

```python
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from app.services.readers_service import ReaderService
from app.db.session import get_db
from app.models.reader import ReaderCreate, ReaderResponse

router = APIRouter()

@router.post("/", response_model=ReaderResponse)
def create_reader(reader: ReaderCreate, db: Session = Depends(get_db)):
    """Создать читателя"""
    try:
        return ReaderService.create_reader(db, reader)
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))

@router.get("/{id}", response_model=ReaderResponse)
def get_reader(id: int, db: Session = Depends(get_db)):
    """Получить информацию о читателе"""
    try:
        return ReaderService.get_reader(db, id)
    except ValueError as e:
        raise HTTPException(status_code=404, detail=str(e))

@router.delete("/{id}")
def delete_reader(id: int, db: Session = Depends(get_db)):
    """Удалить читателя"""
    try:
        return ReaderService.delete_reader(db, id)
    except ValueError as e:
        raise HTTPException(status_code=404, detail=str(e))
```

# Разработка проекта для реализации  API сервиса «Библиотека»

6. В папке **core** создайте файлы:
**__init__.py**

**config.py**

app > core > config.py > ...

```
1   DATABASE_URL = "sqlite:///./library.db"
2
```

7. В папке **db** создайте файлы:
**__init__.py**

**base.py**

app > db > base.py > ...

```
1   from app.db.models import Base
2   from app.db.session import engine
3
4   def init_db():
5       Base.metadata.create_all(bind=engine)
6
```

# Разработка проекта для реализации  API сервиса «Библиотека»

- **models.py**

```python
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Book(Base):
    __tablename__ = "books"
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, nullable=False)
    author = Column(String, nullable=False)

class Reader(Base):
    __tablename__ = "readers"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False)

class Issue(Base):
    __tablename__ = "issues"
    id = Column(Integer, primary_key=True, index=True)
    book_id = Column(Integer, ForeignKey("books.id"), nullable=False)
    reader_id = Column(Integer, ForeignKey("readers.id"), nullable=False)
    book = relationship("Book")
    reader = relationship("Reader")
```

# Разработка проекта для реализации  API сервиса «Библиотека»

- **sessions.py**

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.core.config import DATABASE_URL

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

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

# Разработка проекта для реализации  API сервиса «Библиотека»

- 8. В папке **models** создайте файлы:
- **__init__.py**

**config.py**

```python
from pydantic import BaseModel

class BookBase(BaseModel):
    title: str
    author: str

class BookCreate(BookBase):
    pass

class BookResponse(BookBase):
    id: int

    class Config:
        orm_mode = True
```

**issue.py**

```python
from pydantic import BaseModel

class IssueBase(BaseModel):
    book_id: int
    reader_id: int

class IssueCreate(IssueBase):
    pass

class IssueResponse(IssueBase):
    id: int

    class Config:
        orm_mode = True
```

**reader.py**

```python
from pydantic import BaseModel

class ReaderBase(BaseModel):
    name: str

class ReaderCreate(ReaderBase):
    pass

class ReaderResponse(ReaderBase):
    id: int

    class Config:
        orm_mode = True

```

# Разработка проекта для реализации  API сервиса «Библиотека»

- 9. В папке **services** создайте файлы:
- **__init__.py**

**books_service.py**

```python
from sqlalchemy.orm import Session
from app.db.models import Book
from app.models.book import BookCreate

class BookService:
    @staticmethod
    def create_book(db: Session, book_data: BookCreate):
        book = Book(**book_data.model_dump())
        db.add(book)
        db.commit()
        db.refresh(book)
        return book

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

readers_service.py

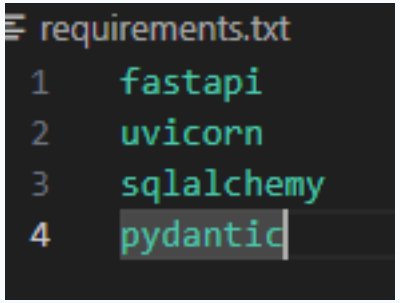
```python
from sqlalchemy.orm import Session
from app.db.models import Reader
from app.models.reader import ReaderCreate

class ReaderService:
    @staticmethod
    def create_reader(db: Session, reader_data: ReaderCreate):
        """
        Создает нового читателя и сохраняет его в базе данных.
        """
        reader = Reader(**reader_data.model_dump())
        db.add(reader)
        db.commit()
        db.refresh(reader)
        return reader

    @staticmethod
    def get_reader(db: Session, reader_id: int):
        """
        Получает информацию о читателе по его ID.
        """
        reader = db.query(Reader).filter(Reader.id == reader_id).first()
        if not reader:
            raise ValueError(f"Читатель с ID {reader_id} не найден")
        return reader

    @staticmethod
    def delete_reader(db: Session, reader_id: int):
        """
        Удаляем читателя
        """
        reader = db.query(Reader).filter(Reader.id == reader_id).first()
        if not reader:
            raise ValueError(f"Читатель с ID {reader_id} не найдена")

        db.delete(reader)
        db.commit()
        return {"message": f"Читатель с ID {reader_id} закрыта"}
```

## issues_service.py

```python
from sqlalchemy.orm import Session
from app.db.models import Issue, Book, Reader
from app.models.issue import IssueCreate

class IssueService:
    @staticmethod
    def create_issue(db: Session, issue_data: IssueCreate):
        """
        Создает новую выдачу книги читателю.
        """
        # Проверка, что книга существует и не выдана
        book = db.query(Book).filter(Book.id == issue_data.book_id).first()
        if not book:
            raise ValueError(f"Книга с ID {issue_data.book_id} не найдена")

        existing_issue = db.query(Issue).filter(Issue.book_id == issue_data.book_id).first()
        if existing_issue:
            raise ValueError(f"Книга с ID {issue_data.book_id} уже выдана")

        # Проверка, что читатель существует
        reader = db.query(Reader).filter(Reader.id == issue_data.reader_id).first()
        if not reader:
            raise ValueError(f"Читатель с ID {issue_data.reader_id} не найден")

        # Создание новой выдачи
        issue = Issue(**issue_data.model_dump())
        db.add(issue)
        db.commit()
        db.refresh(issue)
        return issue

    @staticmethod
    def get_issue(db: Session, issue_id: int):
        """
        Получает информацию о выдаче по ее ID.
        """
        issue = db.query(Issue).filter(Issue.id == issue_id).first()
        if not issue:
            raise ValueError(f"Выдача с ID {issue_id} не найдена")
        return issue

    @staticmethod
    def get_all_issues(db: Session):
        """
        Возвращает список всех выдач.
        """
        return db.query(Issue).all()

    @staticmethod
    def close_issue(db: Session, issue_id: int):
        """
        Закрывает выдачу, удаляя запись о ней (возврат книги).
        """
        issue = db.query(Issue).filter(Issue.id == issue_id).first()
        if not issue:
            raise ValueError(f"Выдача с ID {issue_id} не найдена")

        db.delete(issue)
        db.commit()
        return {"message": f"Выдача с ID {issue_id} закрыта"}
```

# Разработка проекта для реализации  API сервиса «Библиотека»

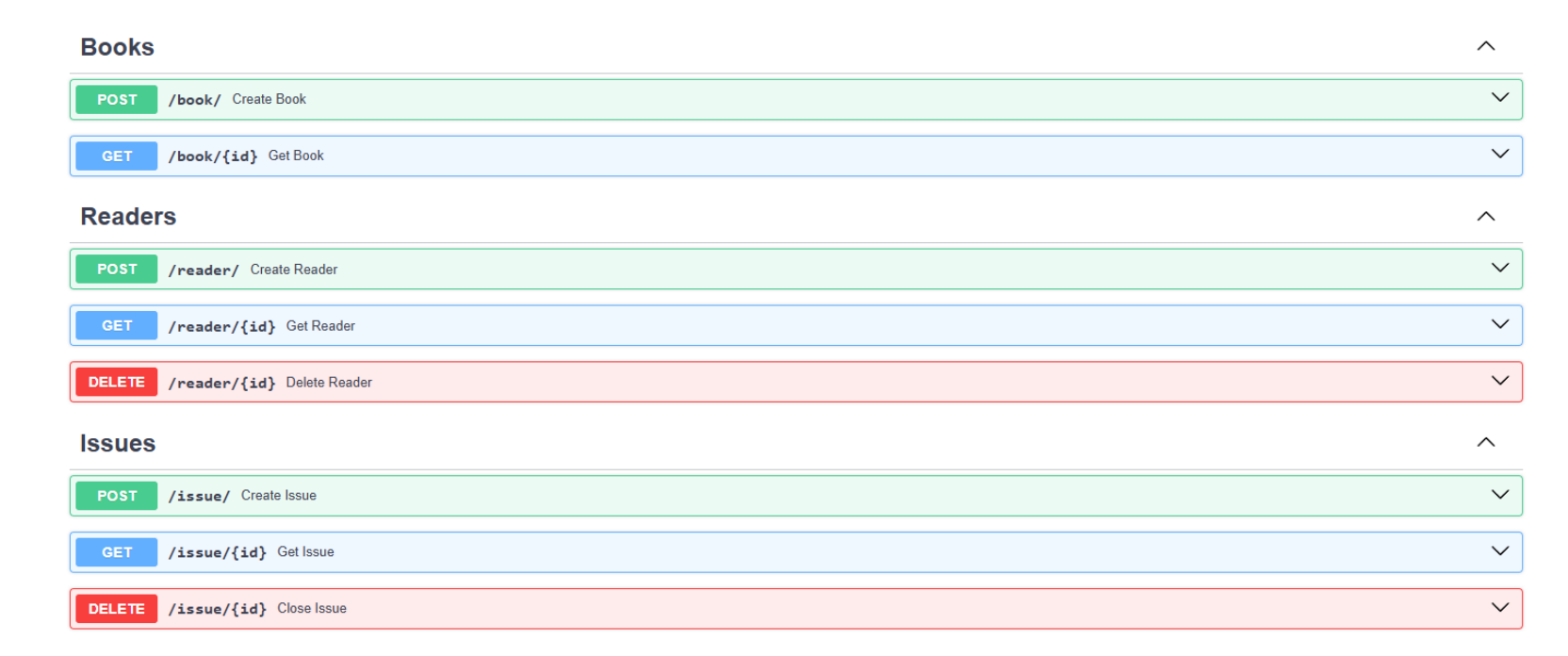
10. В папке app создайте файл **requirements.txt** с содержимым:




11. В консоли, находясь в созданной вами папки в пункте 1, выполните команду установки зависимостей проекта из файла requirements.txt:
**pip install -r requirements.txt**
12. В консоли в той же папке запустите сервис: **uvicorn app.main:app --reload**

# Разработка проекта для реализации  API сервиса «Библиотека»

13. Проверьте что по пути http://127.0.0.1:8000/docs открывается сваггер интерфейс с доступными методами (эндпоинтами):




14. Убедитесь что данные создаются и сохраняются в файле БД : library.db

# Разработка проекта для реализации  API сервиса «Библиотека»

## Доработка сервиса:

15. Доработайте файл  **api\books.py** так, чтобы он обрабатывал ошибки и возвращал http коды с пояснением к ошибке так, как это сделано в файле **api\issues.py**.
16. Создайте в каждом из файлов **api\books.py, api\issues.py, api\readers.py** недостающие маршруты: удаления и возврата всего списка соответствующей сущности. Для этого надо будет добавить код как в папке **api** так и в папке **services**.
17. Проверьте работоспособность новых маршрутов.