Интеллектуальные информационные системы

Практическое задание 4

Кафедра информатики
Институт кибербезопасности и цифровых технологий
РТУ МИРЭА

Разработка базового проекта для реализации API сервиса

Мы создадим API сервис по работе с сущностью Книга (Book).
API будет позволять клиентам сервиса создавать, обновлять, удалять и получать данные о книгах.

Мы будем использовать Python библиотеку **TinyDb** https://pypi.org/project/tinydb/
Это лёгкая, документно-ориентированная база данных для Python, которая хранит данные в формате JSON-файлов и не требует установки сервера. Она проста в использовании, поддерживает фильтрацию и обновление данных с помощью запросов, подобных SQL-запросам, но ограничена по масштабируемости и подходит для небольших проектов. TinyDB удобно применять в проектах, где требуется встроенное хранилище данных без сложной настройки.

У нас получится следующая структура проекта:

```
├── app/
│   ├── __init__.py
│   ├── main.py              # Основной файл приложения FastAPI
│   ├── models.py            # Описание модели книги с использованием Pydantic
│   ├── database.py          # Конфигурация и операции с базой данных (TinyDB)
│   └── routes.py            # Определение маршрутов для работы с книгами
├── requirements.txt         # Зависимости проекта
```

Разработка базового проекта для реализации API сервиса

- **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 файл **requirements.txt** и добавьте туда следующие строки:

```
fastapi
uvicorn
pydantic
tinydb
```

Файл нужен для того чтобы указывать зависимость от сторонних библиотек в одном месте с возможностью устанавливать их все сразу одной командой.

5. Здесь и далее файлы создаются в **папке app**: создайте файл **__init__.py** для более правильной работы инструкции import и чтобы Python правильно понимал структуру проекта.
6. Создайте файл **models.py** для описания Pydantic-модели книги (Book):

```python
from pydantic import BaseModel
from typing import Optional

class Book(BaseModel):
    id: Optional[int] = None  # Идентификатор книги
    title: str                # Название книги
    author: str               # Автор книги
    year: int                 # Год издания книги
```

Разработка базового проекта для реализации API сервиса

7. Создайте файл **database.py** для настройки TinyDB и основных операций с базой данных:

```python
from tinydb import TinyDB, Query
from tinydb.operations import set
import os

# Инициализация базы данных
db_path = os.path.join(os.path.dirname(__file__), "library_db.json")
db = TinyDB(db_path)
books_table = db.table("books")
BookQuery = Query()

def get_all_books():
    return books_table.all()

def get_book(book_id: int):
    return books_table.get(BookQuery.id == book_id)

def add_book(book_data: dict):
    book_id = books_table.insert(book_data)
    books_table.update({"id": book_id}, doc_ids=[book_id])  # добавляем ID в запись
    return book_id

def update_book(book_id: int, updated_data: dict):
    books_table.update(updated_data, BookQuery.id == book_id)
    return get_book(book_id)

def delete_book(book_id: int):
    books_table.remove(BookQuery.id == book_id)
```

- 8. Создайте файл **routres.py** для описания маршрутов (эндпоинтов) для работы с книгами:

```python
from fastapi import APIRouter, HTTPException
from app.models import Book
from app.database import get_all_books, get_book, add_book, update_book, delete_book

rt = APIRouter()

@rt.get("/book/all", response_model=list[Book])
async def read_books():
    """Получить список всех книг."""
    return get_all_books()

@rt.get("/book/{book_id}", response_model=Book)
async def read_book(book_id: int):
    """Получить книгу по ID."""
    book = get_book(book_id)
    if book is None:
        raise HTTPException(status_code=404, detail="Книга не найдена")
    return book

@rt.post("/book", response_model=Book)
async def create_book(book: Book):
    """Добавить новую книги."""
    book_data = book.model_dump()
    book_id = add_book(book_data)
    return get_book(book_id)

@rt.put("/book/{book_id}", response_model=Book)
async def update_book_info(book_id: int, book: Book):
    """Обновить информацию о книге."""
    existing_book = get_book(book_id)
    if existing_book is None:
        raise HTTPException(status_code=404, detail="Книга не найдена")
    updated_book = update_book(book_id, book.model_dump(exclude_unset=True))
    return updated_book

@rt.delete("/book/{book_id}")
async def delete_book_info(book_id: int):
    """Удалить книгу по ID."""
    if get_book(book_id) is None:
        raise HTTPException(status_code=404, detail="Книга не найдена")
    delete_book(book_id)
    return {"message": "Книга успешно удалена"}
```

Разработка базового проекта для реализации API сервиса

9. Создайте основной файл для запуска приложения:

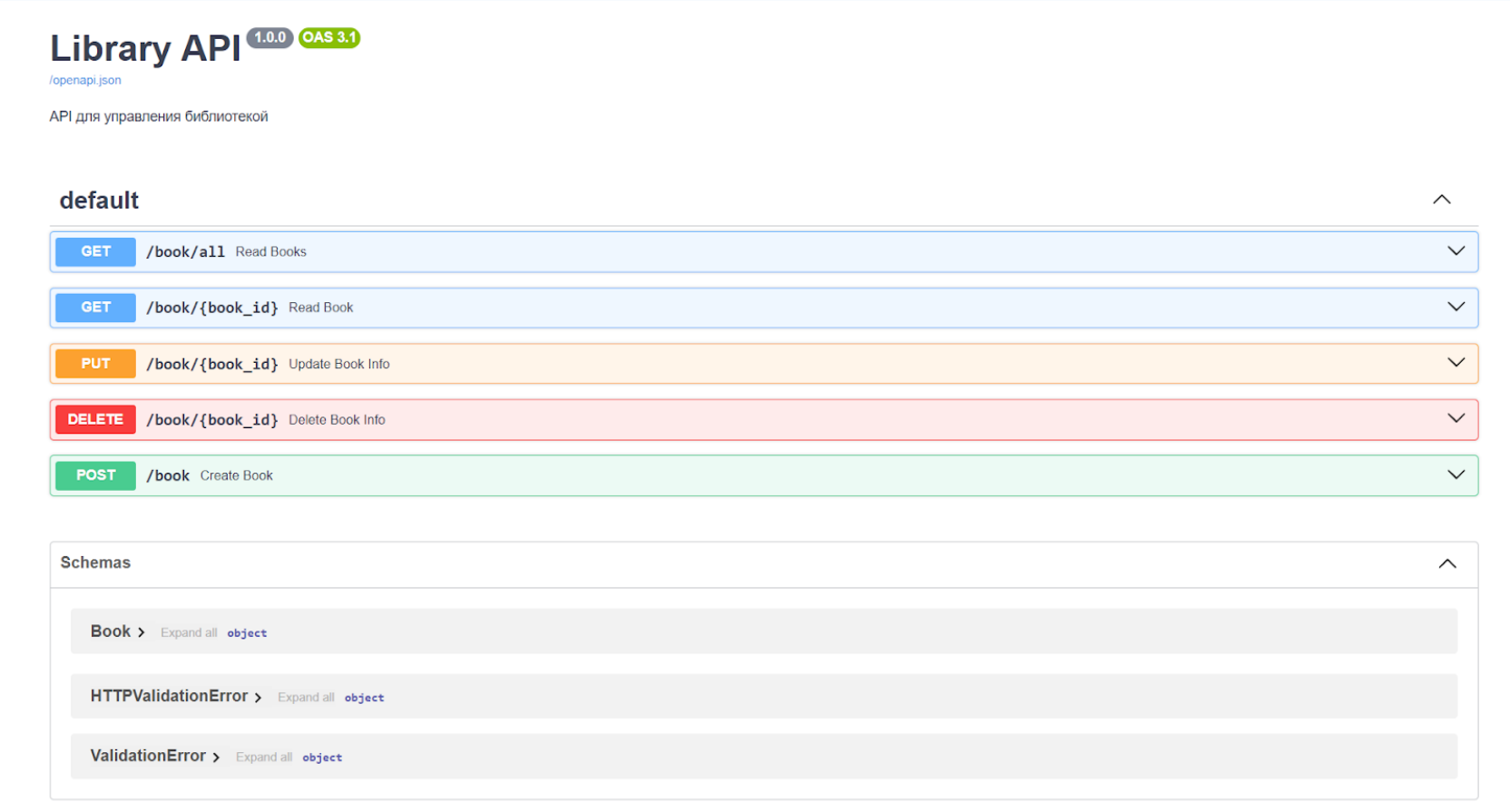
```python
from fastapi import FastAPI
from app.routes import rt

app = FastAPI(
    title="Library API",
    description="API для управления библиотекой",
    version="1.0.0"
)

# Подключение роутера
app.include_router(rt)
```

10. В консоли, находясь в созданной вами папки в пункте 1, выполните команду установки зависимостей проекта из файла requirements.txt:
pip install -r requirements.txt

11. В консоли, находясь в созданной вами папки в пункте 1, запустите приложение командой **uvicorn app.main:app --reload**

Разработка базового проекта для реализации API сервиса

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



13. Проверьте работоспособность эндпоинтов вызвав каждый из методов аналогично тому как вы делали в предыдущих практических работах: **Создайте 2 книги, получите список книг, удалите 1 книгу, измените другую книгу, получите книгу по id**