Урок 15. Выбор темы, структура. Разработка API с помощью FastAPI. Часть 2.

Теория:

Работа с FastAPI и ее преимущества

Каждый из нас сталкивается с API регулярно, пользуясь Google или Яндекс. Вы отправляете запрос в поисковую строку, взаимодействуя с веб-страницей, а та в свою очередь взаимодействует с системой и возвращает вам результаты – это и есть работа API. Оно позволяет различным программам взаимодействовать друг с другом.

Асинхронность

FastAPI поддерживает асинхронное программирование "по умолчанию". Это означает, что ваш код может выполнять несколько задач одновременно, не блокируя выполнение других операций. Это особенно полезно для веб-приложений, где часто приходится обрабатывать множество запросов одновременно.

Автоматическая документация

Одна из самых удобных особенностей FastAPI – это автоматическая генерация документации. Используя аннотации типов Python, FastAPI создает интерактивную документацию для вашего API, которую можно посмотреть прямо в браузере.

Работа с HTTP-запросами

FastAPI отлично работает с различными типами HTTP-запросов: GET, POST, PUT, DELETE и другими.

Принципы разработки

FastAPI – это фреймворк для создания back-end приложений. Он помогает вам создать сервер, который будет обрабатывать запросы от front-end части вашего приложения и возвращать соответствующие ответы. Ваша front-end часть может быть написана на любом языке и с использованием любого фреймворка, и она будет взаимодействовать с back-end, созданным на FastAPI, через API.

Разница между front-end и back-end

Front-end – это та часть веб-приложения, с которой взаимодействует пользователь. Это всё, что вы видите и с чем взаимодействуете на

веб-странице: кнопки, текстовые поля, изображения, навигация и т.д. Front-end разрабатывается с использованием таких технологий, как HTML, CSS и JavaScript. Он отвечает за отображение данных и обработку пользовательского ввода.

Back-end – это "невидимая" часть веб-приложения, которая работает на сервере. Он отвечает за обработку данных, взаимодействие с базой данных, аутентификацию пользователей и другие задачи, которые происходят "за кулисами". Back-end разрабатывается с использованием серверных языков программирования, таких как Python, Java, PHP и других.

Пример и этапы разработки веб-приложения. Подготовка.

Ваша задача на сегодня: разбиться на команды для совместной разработки и поддержания вашего программного продукта и определится с темой. Пример, приведенный ниже можно взять за основу, но функционал вашего веб-приложения все же должен отличаться.

В связи с тем, что работать с базами данных мы еще не умеем, все данные в текущем уроке будут хранится в JSON формате.

Сегодня мы начнем писать API некоего учебного заведения (школы). Как и в любой школе есть, как ученики, так и учителя. Кроме того, имеется расписания, классы, предметы и прочее.

Представим что у нас есть только ученики и учителя, а данные по ним у нас хранятся в JSON-файлах.

Пример:

```
{
    "student_id": 1,
    "first_name": "Иван",
    "last_name": "Иванов",
    "date_of_birth": "2017-05-15",
    "email": "ivan.ivanov@example.com",
    "phone_number": "+7 (123) 456-7890",
    "address": "г. Москва, ул. Пушкина, д. 10, кв. 5",
    "enrollment_year": 2017,
    "grade": 3,
    "special_notes": null
  }
```

Создание проекта (не забываем отправлять на GIT)

Создайте JSON с именем students.json и поместите в него 10 выдуманных учеников.

Структура проекта:

```
myfirstapi/
  .venv/                  # виртуальное окружение
  src/

    myproject/
      __init__.py
      app.py
      utils.py

  tests/
    test_utils.py

  data/
    students.json

  .vscode/
    settings.json
    launch.json

  requirements.txt
  .gitignore
  README.md
```

Добавляем зависимости requirements.txt**:** fastapi[all], uvicorn[standard] и requests
Активируем зависимости: pip install -r requirements.txt

Заполняем файл utils.py:

```
import json

# Функции, для преобразования данных из JSON в dict и обратно

def dict_list_to_json(dict_list, filename):
    -code-
    # дописываем код самостоятельно

def json_to_dict_list(filename):
    -code-
```

Начинаем заполнять app.py:

```
from fastapi import FastAPI, HTTPException
from myproject.utils import json_to_dict_list
import os
from pathlib import Path
```

Описание импортов:

- json_to_dict_list: наша функция, которая будет возвращать всех студентов
- os: модуль, который поможет нам настроить относительные пути к JSON
- pathlib: модуль в Python, который предоставляет классы для работы с путями к файлам и директориям.

Прописываем путь к JSON-файлу:

```
DATA = Path(__file__).resolve().parents[1] / "data" / "students.json"
```

Создаем наш сервер:

Первая функция сработает при первом обращении к корню нашего сервера, вторая функция выдаст список всех учеников.

```
app = FastAPI()

@app.get("/")
def home_page():
    return {"message": "Привет, Мир!"}

@app.get("/students")
def get_all_students():
    try:
        return json_to_dict_list(DATA)
    except FileNotFoundError:
        raise HTTPException(500, "students.json not found")
```

Декоратор @app.get("/students"):

- Этот декоратор используется в FastAPI для обозначения того, что данная функция будет обрабатывать HTTP GET запросы, направленные на маршрут /students.
- Когда клиент (например, веб-браузер или другой сервис) делает GET запрос по этому маршруту, FastAPI вызывает функцию get_all_students.

Теперь запустим приложение.

- Запуск необходимо выполнять именно с корневой директории FastApi проекта, а не из папки app!
- Для запуска воспользуемся командой: uvicorn myproject.app:app --reload --app-dir src или

  cd src
  uvicorn myproject.app:app --reload

Для того чтоб остановить приложение необходимо использовать комбинацию CTRL+C.

Для проверки переходим в корень: http://127.0.0.1:8000 или http://127.0.0.1:8000/students

Проверяем, что запросы обрабатываются.

Видим все данные, а это значит, что наш сервер корректно отработал и вернул те данные, что мы запросили. В боевом проекте переходом по ссылке http://127.0.0.1:8000/students мы бы выполнили запрос к базе данных и вернули бы информацию на страницу. Пока, как вы уже знаете, в качестве хранилища информации мы используем JSON.

Добавление параметра в запрос:

```
@app.get("/students/{grade}")
def get_all_students_grade(grade: int):
    students = json_to_dict_list(DATA)
    return [s for s in students if s.get("grade") == grade]
```

Проверяем: http://127.0.0.1:8000/students/3

Обратите внимание на синтаксис. Путь мы указываем в фигурных скобках, давая имя переменной. Далее необходимо в саму функцию передать эту переменную.

После будет происходить следующее: функция будет принимать путь, тем самым генерируя ссылку по типу /students/3. Далее параметр пути будет автоматически передан в функцию и нам останется только перехватить его и обработать.

Поэкспериментируйте с учениками разных классов.

P.S. Порт, на котором вы запускаете сервер может быть занят, проверьте, что порт свободен:

```
import socket
with socket.socket() as s:
    try:
        s.bind(("127.0.0.1", 8000))
        print("Свободен")
    except OSError:
        print("Занят")
```

И если занят, перейдите на другой порт:

```
uvicorn myproject.app:app --reload --port 8001
```