Интеллектуальные информационные системы

Практическое задание 3

Кафедра информатики
Институт кибербезопасности и цифровых технологий
РТУ МИРЭА

Разработка первого API сервиса с использованием фреймворка FastApi

- **Что такое FastAPI?**
- **FastAPI** — это фреймворк для создания веб-приложений и API на Python. API (Application Programming Interface) — это интерфейс, с помощью которого программы могут общаться друг с другом. FastAPI позволяет легко создавать API, через которые можно обмениваться данными с другими приложениями.

Почему FastAPI? Он:

- Быстрый: как для разработки, так и для работы.
- Лёгкий в использовании: позволяет начать работу всего с нескольких строк кода.
- Предоставляет встроенную документацию для всех функций, которые вы создаёте.

Установка FastAPI и запуск первого приложения

Чтобы начать, нам понадобятся две библиотеки: **FastAPI** и **Uvicorn** (для запуска сервера).

1. Как в прошлых практической создайте папку, откройте терминал и зайдите в эту папку. Создайте виртуальное окружение Python.

2. Установите FastAPI и Uvicorn:

```
pip install fastapi
pip install uvicorn
```

3. Откроете VSCode и создайте файл, например, *main.py*

Разработка первого API сервиса с использованием фреймворка FastApi

- **Первое приложение на FastAPI**

В нашем первом приложении мы создадим простой веб-ресурс, который при обращении к нему скажет «Привет, мир!»: для этого созданном файле main.py наберите следующий код

```python
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Привет, мир!"}
```

Здесь мы:
1.Импортируем *FastAPI* и создаём экземпляр приложения *app*.
2.Используем декоратор ***@app.get("/")***, чтобы указать, что функция ***read_root*** будет запускаться, когда кто-то обращается к корневому пути (/). Внутри этой функции мы возвращаем словарь *{"message": "Привет, мир!"}*.

Разработка первого API сервиса с использованием фреймворка FastApi

Запуск FastAPI приложения

Чтобы запустить приложение, выполните следующую команду в терминале:

```
uvicorn main:app --reload
```

- ***main:app*** означает, что мы запускаем *app* из файла *main.py*.
- Параметр *--reload* активирует режим автообновления, что удобно при разработке.

После запуска приложения оно будет доступно по адресу http://127.0.0.1:8000.

Проверка работы приложения

Откройте браузер и перейдите по адресу *http://127.0.0.1:8000*. Вы увидите:

```
{
    "message": "Привет, мир!"
}
```

Это значит, что приложение работает, и наш API отвечает.

Разработка первого API сервиса с использованием фреймворка FastApi

Преимущество: Автоматическая документация

FastAPI автоматически создаёт документацию для всех маршрутов, которые мы добавляем. Попробуйте перейти по адресу http://127.0.0.1:8000/docs, и вы увидите визуальный интерфейс (Swagger), который позволяет вам тестировать ваш API без написания дополнительного кода.

Добавление новых маршрутов

Теперь добавьте ещё один маршрут, который будет принимать параметр и возвращать сообщение с этим параметром.

```python
# main.py
from fastapi import FastAPI

app = FastAPI()

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

@app.get("/hello/{name}")
def read_name(name: str):
    return {"message": f"Привет, {name}!"}
```

Разработка первого API сервиса с использованием фреймворка FastApi

- Теперь, если мы обратимся к адресу *http://127.0.0.1:8000/hello/Иван*, то получим ответ:

```
{
    "message": "Привет, Иван!"
}
```

Этот маршрут принимает параметр ***name*** и добавляет его в ответ. FastAPI автоматически обрабатывает ввод данных и проверяет типы, поэтому если передать что-то другое, кроме строки, будет ошибка.

Разработка первого API сервиса с использованием фреймворка FastApi

Отправка данных на Сервер

Давайте добавим возможность отправлять данные на сервер с использованием метода **POST** и проверять входные данные.

Предположим, мы хотим создать маршрут, через который пользователи могут отправлять данные о книге, а наш сервер будет проверять их перед сохранением.

Шаг 1: Создание модели для данных с помощью Pydantic

Pydantic - это библиотека для Python, которая обеспечивает удобную и быструю проверку и сериализацию данных с помощью аннотаций типов. В основе Pydantic лежит идея создания моделей данных (или схем), определяющих, какие типы данных ожидаются, и выполняющих автоматическую проверку значений. Это упрощает обработку данных и повышает их надёжность, так как позволяет легко проверять входные данные и предотвращать ошибки, связанные с неверными типами или значениями. FastAPI использует **Pydantic** для проверки данных.

.

Разработка первого API сервиса с использованием фреймворка FastApi

Создайте схему (или модель) для данных о книге (**Class Book**): имя автора, название книги и год

```python
# main.py
from fastapi import FastAPI
from pydantic import BaseModel, Field

app = FastAPI()

# Модель данных для книги
class Book(BaseModel):
    title: str = Field(..., title="Название книги", max_length=100)
    author: str = Field(..., title="Имя автора", max_length=50)
    year: int = Field(..., title="Год издания", ge=1450, le=2024)
```

Здесь:
Book — это Pydantic-модель, которая описывает, как должны выглядеть данные о книге.
Поля *title, author* и *year* будут проверяться на наличие, тип и длину, а также на диапазон значений (например, *year* должен быть между 1450 и 2024).

Разработка первого API сервиса с использованием фреймворка FastApi

Шаг 2: Добавление POST-маршрута для создания книги

Теперь добавьте POST-маршрут */books/*, который будет принимать данные о книге и возвращать сообщение с подтверждением, если данные прошли проверку.

```python
@app.post("/books/")
def create_book(book: Book):
    # Здесь могли бы быть операции сохранения данных
    return {"message": f"Книга '{book.title}' добавлена!", "data": book}
```

Как работает этот маршрут:
Маршрут */books/* принимает объект *book* типа *Book*.
FastAPI автоматически проверит данные, отправленные в запросе, чтобы они соответствовали требованиям модели *Book*.
Если данные корректны, API вернёт сообщение о добавлении книги вместе с данными.

Разработка первого API сервиса с использованием фреймворка FastApi

Шаг 3: Тестирование POST-запроса

После запуска приложения (командой *uvicorn main:app --reload*) перейдите в браузере на http://127.0.0.1:8000/docs и найдите метод **POST /books/**.

Попробуйте отправить данные (используя в качестве клиентя для API модуль **Request** из прошлой лабораторной работы, для этого создайте еще один файл client.py и напишите нужный код вызова API используя следующие параметры:

```
{
    "title": "Война и мир",
    "author": "Лев Толстой",
    "year": 1869
}
```

Помните что для вызова Post метода в API при работе с Request надо использовать соответствующий метод request.post). После чего запустите скрипт в файле client.py. Вы должны получить следующий ответ:

```
{
    "message": "Книга 'Война и мир' добавлена!",
    "data": {
        "title": "Война и мир",
        "author": "Лев Толстой",
        "year": 1869
    }
}
```

Разработка первого API сервиса с использованием фреймворка FastApi

- **Обработка ошибок**

Если данные не соответствуют требованиям модели, например, если ***year*** выходит за пределы диапазона (значение больше 2024 года), FastAPI автоматически вернёт ошибку с сообщением. Для этого измените параметры в файле client.py указав year 2025 и еще раз запустите client.py, То вы должны получить примерно следующую ошибку:

```
{
    "detail": [
        {
            "loc": ["body", "year"],
            "msg": "ensure this value is less than or equal to 2024",
            "type": "value_error.number.not_le",
            "ctx": {"limit_value": 2024}
        }
    ]
}
```

Теперь вы можете:

- Использовать методы **GET** и **POST** для получения и отправки данных на сервер.
- Создавать модели данных с помощью **Pydantic** для проверки и валидации.
- Автоматически обрабатывать ошибки, если входные данные не соответствуют требованиям.