Сириус
IT-Колледж

Продвинутые функции FastAPI

WSGI-приложения. Разделение маршрутов. Тестирование. Форматирование и линтеры.

Сириус IT-Колледж

WSGI-приложения

FastAPI — веб-фреймворк, использующий клиент-серверный протокол **ASGI (Asynchronous Server Gateway Interface)**, который дает доступ к функциям параллельного выполнения кода. Но, с помощью связки совместимых по рабочим классам HTTP-сервера Gunicorn и ASGI-сервера Uvicorn, FastAPI также поддерживает протокол **WSGI (Web Server Gateway Interface)** с последовательной обработкой запросов.

FastAPI позволяет монтировать любые WSGI-приложения, например, на фреймворках Dash, Flask, Django или CherryPy, внутри приложения FastAPI. На корневом уровне у вас может быть основное приложение FastAPI, дополненное WSGI-приложениями (например, на Flask), которые будут обрабатывать все запросы для этого конкретного пути.

> **Dash** — популярный веб-фреймворк для визуализации данных на базе графиков plotly, включающий набор компонентов для работы с HTML и Bootstrap.

Сириус IT-Колледж

WSGI-приложения

Файл `dashboard.py` с кодом сервера Plotly dash:

```
from dash import html
from dash import dcc
import dash_bootstrap_components as dbc
import dash


dash_app = dash.Dash(__name__, requests_pathname_prefix='/dashboard/', title='Demo')

header = dbc.Row(dbc.Col([
            html.Div(style={"height": 30}),
            html.H1("Demo", className="text-center"),
        ]), className="mb-4",
)

dash_app.layout = dbc.Container(
    [header,],
    fluid=True,
    className="bg-light",
)
```

Сириус IT-Колледж

WSGI-приложения

Файл `main.py` с кодом приложения FastAPI:

```python
from fastapi import FastAPI
from fastapi.middleware.wsgi import WSGIMiddleware
from demo_dash import dash_app
import uvicorn

app = FastAPI()
app.mount("/dashboard", WSGIMiddleware(dash_app.server))

@app.get('/')
def index():
    return "Hello"

if __name__ == "__main__":
    uvicorn.run("main:app", host='127.0.0.1', port=8000, reload=True)
```

Сириус
IT-Колледж

Разделение маршрутов

Правила монтирования приложений WSGI позволяют монтировать внутри одного приложения FastAPI несколько других самостоятельных приложений. Каждое подобное субприложение FastAPI будет иметь свою документацию, работать независимо от других приложений и будет обрабатывать свои запросы, зависящие от пути.

Чтобы реализовать подобную возможность, просто создайте файл первого субприложения (`apiv1.py`), как показано ниже:

```python
from fastapi import FastAPI


apiv1 = FastAPI()

@apiv1.get('/returnName')
def index():
    return {"Name": "demo"}
```

Сириус IT-Колледж

Разделение маршрутов

Теперь создайте файл второго субприложения (`apiv2.py`):

```python
from fastapi import FastAPI


apiv2 = FastAPI()

@apiv2.get('/returnLocation')
def index():
    return {"Location": "Sirius"}
```

Сириус IT-Колледж

Разделение маршрутов

Теперь мы можем смонтировать оба этих субприложения в главном приложении (`master.py`) с помощью импорта объектов:

```python
from fastapi import FastAPI
from apiv1 import apiv1
from apiv2 import apiv2
import uvicorn

app = FastAPI()
app.mount("/api/v1", apiv1)
app.mount("/api/v2", apiv2)


@app.get('/')
def index():
    return "Hello"

if __name__ == "__main__":
    uvicorn.run("master:app", host='127.0.0.1', port=8000, reload=True)
```

Сириус
IT-Колледж

Разделение маршрутов

Если вы делаете запросы к эндпоинтам соответствующих путей, запрос будет обрабатываться этими субприложениями.

Удобство метода заключается в том, что он не требует промежуточного программного обеспечения.

Сириус IT-Колледж

Разделение маршрутов

FastAPI имеет собственную систему API-маршрутизации. **API-роуты (APIRouters)** можно рассматривать как мини-приложения FastAPI, которые являются частью более крупного приложения. Это позволяет разбить большие маршруты приложений на небольшие блоки API-роутов и смонтировать их в основном приложении.

> **Обратите внимание!** Эти API-роуты не являются независимыми, как те, что мы видели в двух предыдущих разделах, а входят в основное приложение как составная часть. Поэтому все маршруты от API-роутов будут перечислены в основной документации приложения.

API-роуты могут иметь отдельные префиксы для операций пути, тегов, зависимостей и ответов. Чтобы реализовать это, нужно импортировать класс `APIRouter` из FastAPI, а затем использовать его объект для создания маршрутов, как в обычном приложении FastAPI.

Для примера предположим, что мы создаем систему управления библиотекой и хотим обрабатывать данные книг и новелл (рассказов) отдельно.

Сириус IT-Колледж

Разделение маршрутов

Реализуем это через применение APIRouter в файле `book.py` (для книг):

```
from fastapi import APIRouter


bookroute = APIRouter()

@bookroute.get('/info')
def books():
    return {"detail": "This book info is from the book APIRouter",
    "name": "Hello",
    "ISBN": "32DS3"}
```

Сириус IT-Колледж

Разделение маршрутов

А этот код — пример использования APIRouter в файле novel.py (для новелл):

```
from fastapi import APIRouter


novelroute = APIRouter()

@novelroute.get('/info')
def novels():
    return {"detail": "This novel info is from the novel APIRouter",
    "name": "I am good",
    "publication": "Somewhat"}
```

Сириус IT-Колледж

Разделение маршрутов

Пример кода для включения двух API-роутов (`book.py` и `novel.py`) в основное приложение:

```python
from fastapi import FastAPI
from book import bookroute
from novel import novelroute
import uvicorn

app = FastAPI()
app.include_router(bookroute, prefix="/book")
app.include_router(novelroute, prefix="/novel")


@app.get('/')
def index():
    return "Hello"

if __name__ == "__main__":
    uvicorn.run("demo:app", host='127.0.0.1', port=8000, reload=True)
```

Сириус IT-Колледж

Тестирование

Тестирование основано на **HTTPX**, который, в свою очередь разработан на основе **Requests**, поэтому он прост и интуитивно понятен.

В результате, можно использовать **pytest** напрямую с FastAPI.

> Для использования `TestClient` необходимо установить `httpx` и `pytest`:

```
$ pip install httpx pytest
```

Сириус IT-Колледж

Тестирование

```python
from fastapi import FastAPI
from fastapi.testclient import TestClient

app = FastAPI()


@app.get("/")
async def read_main():
    return {"msg": "Hello World"}


client = TestClient(app)


def test_read_main():
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"msg": "Hello World"}
```

Сириус
IT-Колледж

Тестирование

В реальном приложении тесты выделены в отдельный файл. Так же как и само приложение FastAPI может быть разбито на несколько файлов/модулей и тп.

Допустим у нас есть проект со следующей структурой:

```
.
├── app
│   ├── __init__.py
│   ├── main.py
│   └── test_main.py
```

Сириус IT-Колледж

Тестирование

Файл `main.py` содержит приложение FastAPI:

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def read_main():
    return {"msg": "Hello World"}
```

Сириус IT-Колледж

Тестирование

Так как наличие файла `__init__.py` задает пакет мы можем использовать относительный импорт в файле с тестами, чтобы получить приложени `app` из модуля `main`.

```
from fastapi.testclient import TestClient

from .main import app

client = TestClient(app)


def test_read_main():
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"msg": "Hello World"}
```

Сириус IT-Колледж

Тестирование

Для запуска тестов достаточно выполнить в терминале вызов `pytest`:

```
$ pytest

================ test session starts ================
platform linux -- Python 3.6.9, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /home/user/code/superawesome-cli/app
plugins: forked-1.1.3, xdist-1.31.0, cov-2.8.1
collected 6 items

████████████████████████████████████████ 100%

test_main.py ......                                [100%]

================= 1 passed in 0.03s =================
```

Сириус
IT-Колледж

Форматирование и линтеры

PEP-8 - этот документ описывает соглашение о том, как писать код для языка python, включая стандартную библиотеку, входящую в состав python.

Поддержание соответствия кода стандарту обычно выполняется вручную и по собственной памяти, однако для этой цели можно использовать специальные программные средства - **форматеры**.

А для упрощения разработки проекта и экономии времени можно использовать **pre-commit hooks**, которые проверят код на соответствие стандартам, форматированию, безопасности и множеству других пунктов.

Сириус
IT-Колледж

Форматирование и линтеры

Black - форматирует код для поддержания однородного стиля. Правила форматирования основаны на множестве правил Python. Например:

```
#non black code
def myfunc(product_id, order_id):
    product = get_object(Product,   id = product_id, something = something, something_else = something_else, order_id   = order_id)
```

```
# after using black
def myfunc(product_id, order_id):
    product = get_object(
        Product,
        id=product_id,
        something=something,
        something_else=something_else,
        order_id=order_id,
    )
```

Сириус
IT-Колледж

Форматирование и линтеры

Black может быть установлен как самостоятельное приложение, но удобнее добавить его как часть pre-commit пайплайна. Для начала необходимо установить `pre-commit`:

```
$ pip install pre-commit
```

Теперь добавим файл конфигурации, в котором зададим все операции нашего пайплайна.

Сириус IT-Колледж

Форматирование и линтеры

Файл `.pre-commit-config.yaml`:

```yaml
repos:
-   repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v2.3.0
    hooks:
    -   id: check-yaml
    -   id: end-of-file-fixer
    -   id: trailing-whitespace

-   repo: https://github.com/psf/black
    rev: 22.10.0
    hooks:
    -   id: black
```

Сириус
IT-Колледж

Форматирование и линтеры

Установим пайплайн:

```
$ pre-commit install
pre-commit installed at .git\hooks\pre-commit
```

Теперь при каждом коммите Black будет выполнять форматирование кода.

Сириус
IT-Колледж

Форматирование и линтеры

Линтер — программа, которая проверяет код на соответствие стандартам в соответствии с определённым набором правил. Правила описывают отступы, названия создаваемых сущностей, скобки, математические операции, длину строк и множество других аспектов. Каждое отдельное правило кажется не очень важным, но соблюдение их всех — основа хорошего кода.

Главная задача линтера — сделать код единообразным, удобным для восприятия и самим программистом, и другими людьми, которые будут читать код. В разных командах могут использоваться разные линтеры и разные наборы правил для них, но главное — уметь работать с линтером в принципе, а привыкнуть писать по определенным правилам будет несложно.

Сириус
IT-Колледж

Форматирование и линтеры

Чтобы добавить линтер flake 8, добавьте в файл `.pre-commit-config.yaml` следующие строки:

```
repos:
-   repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v2.3.0
    ##black etc here

-   repo: https://github.com/PyCQA/flake8
    rev: 4.0.1
    hooks:
    -   id: flake8
        args: [--max-line-length=88]
        #files: ^my_appname/|^test_suite_name/
```

Сириус IT-Колледж

Форматирование и линтеры

Возможно так же изменить настройки flake8: исключить некоторые файлы, изменить максимальную длину строки или игнорировать некоторые ошибки. Для этого необходимо добавить либо файл `.flake8`, либо `setup.cfg` с требуемыми настройками. Рекомендуется использовать `setup.cfg` так как это общий файл и, при создании других хуков, можно будет добавлять туда их параметры. Пример файла `setup.cfg`:

```
[flake8]
ignore = E501, W503, E203, E402, E712
max-line-length = 88
exclude = .git, backend/alembic/versions/* , backend/db/base.py
```

Чтобы выполнить коммит без выполнения форматера и линтера используйте параметр `--no-verify`

```
$ git commit -m "my message" --no-verify
```

Сириус IT-Колледж

Форматирование и линтеры

Возможно так же подключить **import reordering** добавив следующие строки:

```
-   repo: https://github.com/asottile/reorder_python_imports
    rev: v3.10.0
    hooks:
    -   id: reorder-python-imports
```

Теперь при каждом коммите эти хуки будут "подчищать" код. Чтобы запустить хуки без коммита выполните:

```
$ pre-commit run --all-files
```

> **Важно!** Это приведет к изменениям в файлах и возможно придется исправлять некоторое количество ошибок и предупреждений от линтера, так как он не позволит выполнить коммит с нарушениями.