FastAPI

Знакомство с FastAPI

ITcoder

FastAPI

Оглавление:

Введение

FastAPI - это современный **API-**фреймворк**,** который может похвастаться исключительно высокой производительностью**.** На момент написания статьи проекту менее двух лет**,** но он завоевал огромную популярность за относительно короткое время**.**

Как следует из названия**, FastAPI** отлично подходит для создания веб-интерфейсов **API,** которые обычно возвращают **JSON** для обмена данными между приложениями**.** Однако он также хорошо подходит для веб-приложений**,** которые генерируют **HTML** для браузеров и взаимодействия с пользователем**.**

Поскольку **FastAPI** является современным**,** он поставляется с современными функциями по умолчанию**,** такими как **"async"/"await"** и **ASGI-**серверы для создания более масштабируемых приложений**. FastAPI -** это захватывающий проект**,** набирающий популярность**.** Давайте вкратце расскажем о том**,** как начать работу с **FastAPI.**

Было бы невежливо нарушить традицию и не привести пример **Hello World.** Реальные проекты будут логически организованы в отдельные части**,** о чем будет рассказано позже**.**

Прежде чем приступить к этому**,** я хотел бы поблагодарить Майкла Кеннеди**,** ведущего потрясающего подкаста **Talk Python To Me** и основного автора книги **Talk Python Training.** Я благодарен Майклу**,** которым я очень восхищаюсь**,** и его материалы обеспечили большую часть моего понимания **FastAPI.**

Окружение

Я использую **Red Hat Enterprise Linux 8** в качестве повседневного драйвера для стабильности**,** вы можете включить и использовать более актуальную версию **Python 3,** включив в нее поток модулей**:**

$ sudo dnf module enable python3.8

Создайте новый рабочий каталог проекта**:**

```
$ mkdir exampleforyou && cd exampleforyou
```

Создайте новую виртуальную среду**,** явно ссылаясь**,** в данном случае**,** на **Python 3.8:**

```
$ python3.8 -m venv venv
```

Или вы можете использовать ту виртуальную среду которую предпочитаете**,**самый популярный способ это **:**

```
$ virtualenv venv
```

Активируйте виртуальную среду и установите **fastapi** и **unicorn. Uvicorn -** это молниеносная реализация **ASGI-**сервера**,** используемая для запуска приложения**:**

```
$ source venv/bin/activate
$ pip install --upgrade pip
$ pip install fastapi uvicorn
```

Hello FastAPI

Создайте файл **main.py,** это самый простой пример**:**

```
$ vi main.py
```

```
import fastapi
import uvicorn

motd = fastapi.FastAPI()

@motd.get('/')
def message():
    return {
        'message': "Hello FastAPI!"
    }
if __name__ == '__main__':
    uvicorn.run(motd, host='127.0.0.1', port=8000)
```

Пример включает два импорта для **FastAPI** и **Unicorn,** сервера для запуска приложения**.** Затем он создает экземпляр объекта **FastAPI** под названием **hello** и определяет функцию**,** украшенную **HTTP-**методом **GET.** Наконец**,** он запускается**,** используя объект **FastAPI** и**,** в качестве опции**,** определение хоста и порта**.**

Приложение может быть запущено либо с помощью **uvicorn:**

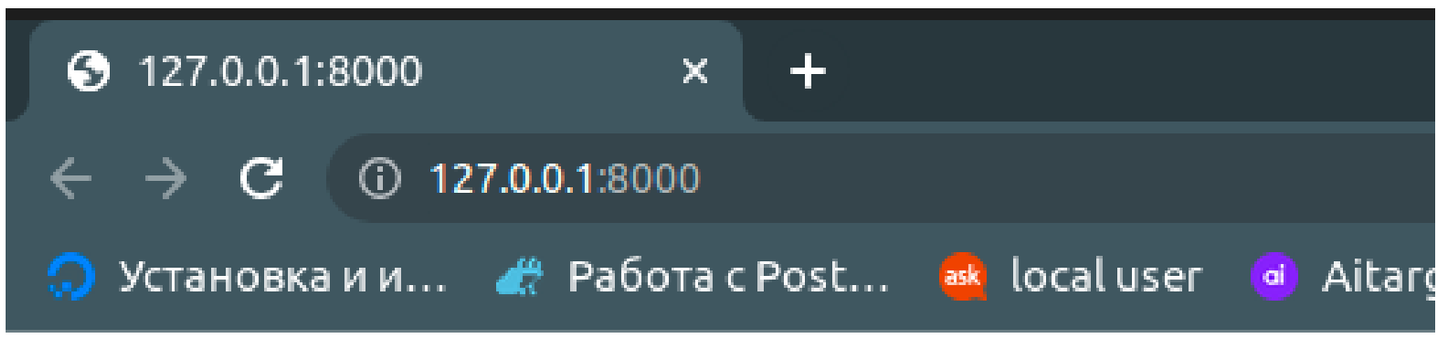
```
$ uvicorn main:motd --reload --port 8000
```

Или запустите файл **main.py** с помощью команды**:**

```
$ python main.py
```

FastAPI

И перейдите по адресу **http://127.0.0.1:8000/** и увидите вот такой **JSON**




```
{"message":"Hello FastAPI!"}
```

Добавление маршрутизации

Хорошей идеей является структурирование проекта с самого начала**. API** могут находиться в каталоге и конфигурироваться с помощью маршрутизатора**.**

Создайте директорию в проекте**:**

```
$ mkdir api
```

Переместите и измените декоратор**,** в данном случае также добавив новый контекстный путь**:**

```
$ vi api/motd.py
```

```
import fastapi

router = fastapi.APIRouter()

@router.get('/api/motd')
def message():
      return {
            'message': "Hello FastAPI!"
     }
```

И обновите **main.py,** для наглядности вызывая объект **FastAPI main_app** и немного изменив структуру с помощью функции **configure()** для включения **API:**

```
$ vi main.py
```

FastAPI

```
import fastapi
import uvicorn

from api import motd

main_app = fastapi.FastAPI()

def configure():
    configure_routing()

def configure_routing():
    main_app.include_router(motd.router)

if __name__ == '__main__':
    configure()
    uvicorn.run(main_app, host='127.0.0.1', port=8000)
else:
    configure()
```

Эта структура закладывает основу для развития и развития проекта**.**

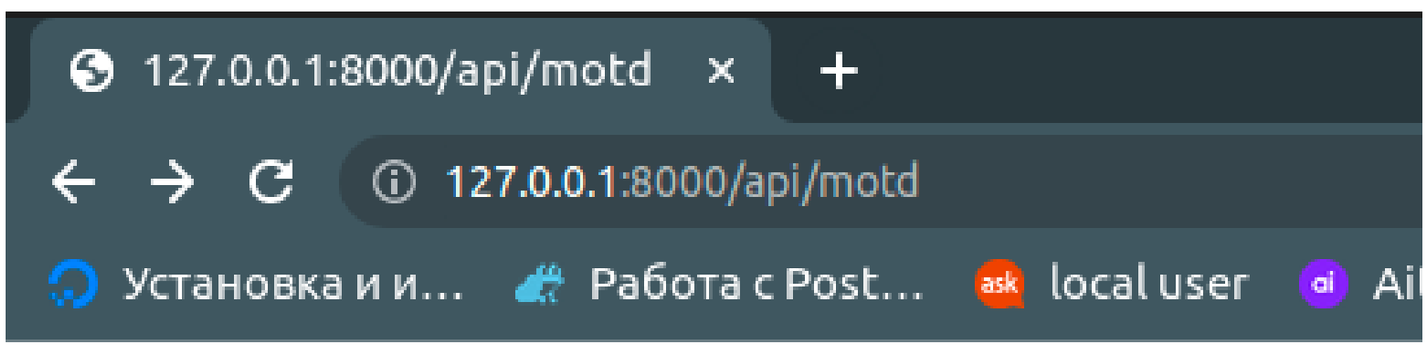
Приложение может быть запущено либо с помощью **uvicorn:**

```
$ uvicorn main:main_app --reload --port 8000
```

Или выполнение файла **main.py:**

```
$ python main.py
```

На главной странице теперь будет отображаться **"Not Found",** но **API** теперь доступен по адресу **http://127.0.0.1:8000/api/motd.**



{"message":"Hello FastAPI!"}

Добавление домашней страницы

Поддержка **FastAPI** Шаблоны **Jinja2** для рендеринга **HTML**

Чтобы использовать шаблоны **Jinja2,** установите пакет**:**

```
$ pip install jinja2
```

Создайте в проекте директорию **views** и **templates:**

```
$ mkdir views templates
```

Использование **Jinja2** означает**,** что вы можете разбить **HTML** на многократно используемые фрагменты**,** это должно показаться знакомым шаблоном из других веб**-**фреймворков**.** Добавьте базовый **HTML-**шаблон для **base** и **home:**

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<title>FastAPI</title>
</head>
<body>
	{% block content %}
	{% endblock %}
</body>
</html>
```

```
$ vi templates/home.html
```

```
{% extends "_base.html" %}
{% block content %}
<h1>Hello FastAPI!</h1>
<a href="/api/motd">Message Of The Day API</a>
{% endblock %}
```

Используя шаблон **TemplateResponse,** добавьте домашнее представление **home.py:**

```
$ vi views/home.py
```

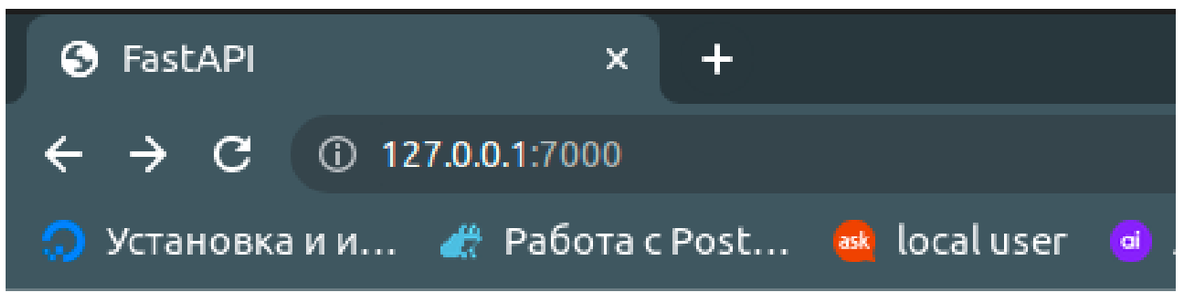
```
import fastapi
from starlette.requests import Request

from starlette.templating import Jinja2Templates

router = fastapi.APIRouter()
templates = Jinja2Templates('templates')

@router.get('/')
def home(request: Request):
	return templates.TemplateResponse('home.html', {'request': request})
```

Обновите **main.py,** чтобы импортировать представление **home** и настроить маршрутизацию**:**

```
$ vi main.py
```

```
import fastapi
import uvicorn

from api import motd
from views import home # New

main_app = fastapi.FastAPI()

def configure():
    configure_routing()

def configure_routing():
    main_app.include_router(motd.router)
    main_app.include_router(home.router) # New
if __name__ == '__main__':
    configure()
    uvicorn.run(main_app, host='127.0.0.1', port=7000)
else:
    configure()
```

Запуск сервера и посещение сайта **http://127.0.0.1:8000** теперь должно вернуть обычную **HTML-**страницу**.**



Hello FastAPI!

Message Of The Day API

Статические файлы

Чтобы включить статический директорий**,** например**,** для включения стили и изображения**, FastAPI** использует концепцию монтирования**.**

Установите зависимость для монтирования **aiofiles:**

```
$ pip install aiofiles
```

Создайте директорию для хранения статических файлов**,** в данном примере используется изображение**:**

```
$ mkdir -p static/img
```

Создайте директорию для хранения статических файлов, в данном примере используется изображение:

Я скопировал изображение под названием **fastapi_logo.png** в **static/img**

```
$ vi main.py
```

Добавьте следующий импорт:

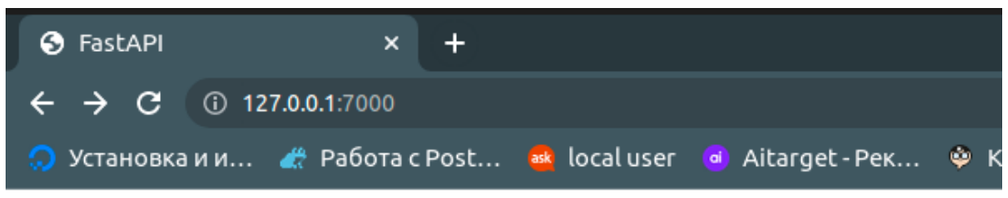
```
from starlette.staticfiles import StaticFiles
```

Добавьте следующее **/static** соединение в функцию **configure_routing():**

```
def configure_routing():
        main_app.mount('/static', StaticFiles(directory='static'), name='static') # New
        main_app.include_router(motd.router)
        main_app.include_router(home.router)
```

В **HTML-**шаблоне **Jinga2** на статические файлы можно ссылаться, например, в файле **templates/home.html** следующим образом:

```
<img src="static/img/fastapi_logo.png" alt="FastAPI">
```



Hello FastAPI!

Message Of The Day API

Переменные окружения

Существует множество подходов к управлению секретными переменными, такими как пароли и токены доступа. В долгосрочной перспективе я предпочитаю использовать переменные окружения. Такой подход позволяет избежать случайного включения таких секретных переменных в исходный код и заложить хорошую основу для использования контейнеров на более поздних этапах.

```
$ pip install environs
```

В этом примере установите переменную локальной среды **ENV_SECRET:**

```
$ export ENV_SECRET="MyTopSecretToken"
```

Отредактируйте файл **main.py:**

```
$ vi main.py
```

Добавьте следующий импорт**:**

```
from environs import Env
```

И следующая функция**:**

```
def configure_env_vars():
    env = Env()
    env.read_env()
    if not env("ENV_SECRET"):
        print(f"WARNING: environment variable ENV_SECRET not found")
        raise Exception("environment variable ENV_SECRET not found.")
    else:
        home.secret = env("ENV_SECRET")
```

Эта новая функция требует вызова**,** поэтому добавьте ее в функцию **configure():**

```
def configure():
    configure_routing()
    configure_env_vars()
```

Если установлена переменная окружения **ENV_SECRET,** функция устанавливает значение **home.secret,** поэтому добавьте следующее в файл **views/home.py:**

```
$ vi views/home.py
```

Следующий импорт**:**

```
from typing import Optional
```

И определите необязательный секретный параметр**:**

```
secret: Optional[str] = None
```

Чтобы проверить его работу и увидеть**,** как можно передавать значения в шаблон**,** обновите представление**,** чтобы оно возвращало значение**:**

```
@router.get('/')
def home(request: Request):
    return templates.TemplateResponse('home.html', {'request': request,
'display_secret': secret})
```

Наконец, отредактируйте шаблон домашней страницы, чтобы отобразить в нем секрет:

```
$ vi templates/home.html
```

```
<p>SECRET: {{ display_secret }}</p>
```

Запуск приложения теперь должен отобразить секрет на главной странице. Это демонстрирует, как значения могут быть получены безопасным способом на основе окружения и развязки различий между средами. При использовании **Docker**, **Podman**, **Kubernetes** или **OpenShift** этот подход принесет свои плоды.

В **Environs** также поддерживается чтение скрытого файла **.env** в корне проекта, вместо экспорта переменных на уровне пользователя их можно определить на уровне проекта. Только помните, что никогда не включайте **.env** в контроль версий!

```
$ vi .env
```

```
export ENV_SECRET="MyTopSecretToken"
```

Digital Ocean API

В этом разделе мы добавим службу, которая будет обращаться к внешнему **API**, используя **Digital Ocean** для получения списка всех доступных изображений капель. Вызов **API** к **Digital Ocean** требует аутентификации с помощью **API-**токена.

Этот подход использует другую зависимость пакета **httpx:**

```
$ pip install httpx
```

Создайте новый раздел с названием **services:**

```
$ mkdir services
```

Добавьте новый файл **Python** для службы **Digital Ocean Service:**

```
from typing import Optional
import httpx

do_api_token: Optional[str] = None
async def get_droplet_images_async():
        url = f'https://api.digitalocean.com/v2/images?type=distribution'
#Если не сработает то поднимите любой сервер например :
       # url=f'http://127.0.0.1:8000/api/v1/test.png' #здесь есть такой словарь как
{'images': "любая ссылка на фотку"}

        url_headers = {'Authorization': 'Bearer ' + do_api_token}

    async with httpx.AsyncClient() as client:
            resp = await client.get(url, headers=url_headers)
    data = resp.json()
    droplet_images = data['images']
    return droplet_images
```

Добавьте новый **API,** который потребляет сервис**:**

```
$ vi api/digital_ocean_images.py
```

```
import fastapi
from services import digital_ocean_service

router = fastapi.APIRouter()

@router.get('/api/droplet_images')
async def images():
        return await digital_ocean_service.get_droplet_images_async()
```

Обновите **main.py,** чтобы включить маршрутизатор этого нового **API,** и обновите переменную окружения**,** чтобы установить токен доступа **Digital Ocean.**

```
$ vi main.py
```

Включите импорт**:**

```
from api import digital_ocean_images
from services import digital_ocean_service
```

Включите маршрутизатор**:**

```
def configure_routing():
        main_app.mount('/static', StaticFiles(directory='static'), name='static')
        main_app.include_router(motd.router)
        main_app.include_router(home.router)
        main_app.include_router(digital_ocean_images.router)
```

Обновляя функцию **configure_env_vars(),** вы можете продолжать добавлять блоки **if not / else** для многочисленных переменных**,** например**:**

```
def configure_env_vars():
        env = Env()
        env.read_env()
        if not env("ENV_SECRET"):
                print(f"WARNING: environment variable ENV_SECRET not found")
                raise Exception("environment variable ENV_SECRET not found.")
        else:
                home.secret = env("ENV_SECRET")
        if not env("DO_API_ACCESS_TOKEN"):
                print(f"WARNING: environment variable DO_API_ACCESS_TOKEN not found")
                raise Exception("environment variable DO_API_ACCESS_TOKEN not found.")
        else:
                digital_ocean_service.do_api_token = env("DO_API_ACCESS_TOKEN")
```

Не забудьте экспортировать переменную окружения **token** или добавить ее**,** например**,** в **.env:**

```
export DO_API_ACCESS_TOKEN=xyzxyzxyzyz
```

Посетите сайт **http://127.0.0.1:8000/api/droplet_images,** чтобы увидеть все полученные результаты в формате **JSON.**

Отлично**,** наконец**,** обновите домашнее представление**,** чтобы отобразить результаты**,** в данном случае список **slugs** для всех доступных образов дистрибутива в **Digital Ocean.**

```
$ vi views/home.py
```

Импортируйте сервис **digital_ocean_service:**

```
from services import digital_ocean_service
```

И обновите функцию**,** обратите внимание**,** что функция преобразована с использованием **async** и **await:**

```
@router.get('/')
async def home(request: Request):
        droplet_images = await digital_ocean_service.get_droplet_images_async()
        return templates.TemplateResponse('home.html', {'request': request,
'display_secret': secret, 'droplet_images': droplet_images})
```

Наконец**,** обновите домашнюю страницу шаблона**,** чтобы отобразить **slugs:**

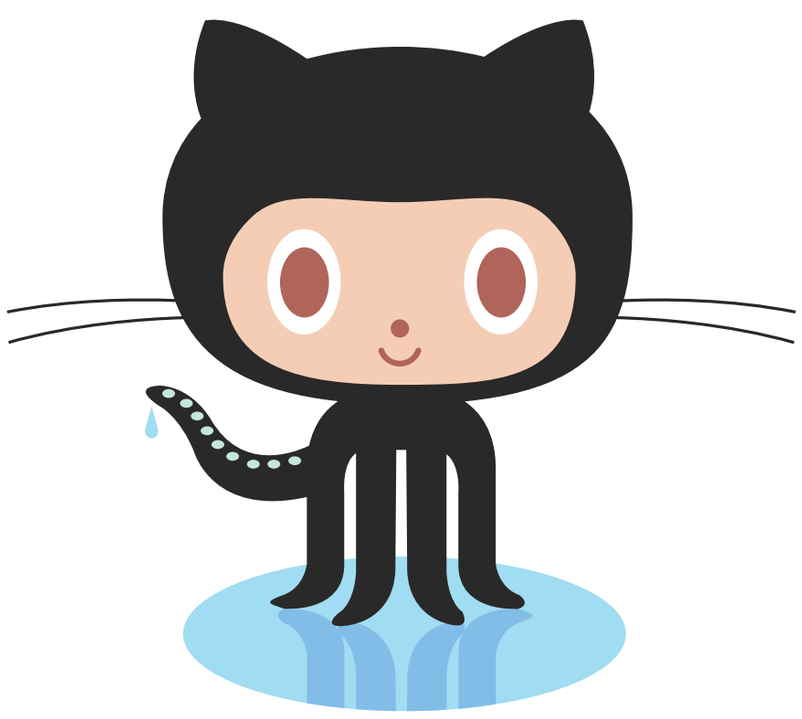
```
$ vi templates/home.html
```

```
{% for i in images %}
   <li> {{ i.slug }}</li>
{% endfor %}
```

Я думаю, что это было отличное введение в **FastAPI**, охватывающее все основы, чтобы разжечь аппетит. Документация **FastAPI** великолепна, а по мощности и, как мне кажется, скорости и простоте этот веб-фреймворк является серьезным соперником.

В следующий раз подключим **sqlalchemy +**БАЗА ДАННЫХ.
FastApi - очень интересный фреймворк , надеюсь и вам было интересно.

Исходный код можете найти тут:



https://github.com/jumabekova06/FastAPI-first-project