# Описание

1. Удобный фреймворк для создания бэкенда С АВТОМАТИЧЕСКОЙ генерацией сваггера по умолчанию.

# Quick start

### Базовый сервис

from typing import Union

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

def read\_root():

return {"Hello": "World"}

@app.get("/items/{item\_id}")

def read\_item(item\_id: int, q: Union[str, None] = None):

return {"item\_id": item\_id, "q": q}

Запуск с помощью

fastapi dev app.py – эта команда автоматически настраивает параметры сервера

#### Но лучше

from typing import Union

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

def read\_root():

return {"Hello": "World"}

@app.get("/items/{item\_id}")

def read\_item(item\_id: int, q: Union[str, None] = None):

return {"item\_id": item\_id, "q": q}

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app, host=0.0.0.0, port=8000)

# Общий механизм работы

1. Сервер: прослушивание портов делает Uvicorn
2. Эндпоинты: ФастАПИ принимает запросы от uvicorn обрабатывает их как эндпоинты

# Сервер

Если у тебя есть структура проекта:

myapp/

├── mod/

│ ├── \_\_init\_\_.py

│ └── main.py # <- Здесь `app = FastAPI()`

И в main.py определено ASGI-приложение:

from fastapi import FastAPI

app = FastAPI() # <- Это ASGI-приложение (app)

То можно запустить Uvicorn

Из консоли:

uvicorn mod.main:app

Из файла:

if \_\_name\_\_ = “\_\_main”:

uvicorn.run(“mod.main:app”, port = …)

Если более сложная структура то как file.file.mod.main:app

# Разработка эндпоинтов FastAPI

## Общая структура

1. Типичная структура выглядит следующим образом:

from fastapi import FastAPI

app = FastAPI()

@app.get("/)

def endpoint1():

return "Hello"

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app, host=0.0.0.0, port=8000)

## Эндпоинт

1. Эндпоинт всегда состоит из

* функции, которая выполняет функционал эндпоинта
* FastAPI декоратора, который оборачивает функцию в требуемый функционал для обработки запросов.

Также декоратор содержит путь path, о чем подробнее в Части эндпоинта

Реализация эндпоинта в коде выглядит так:

@app.get("/)

def endpoint1():

return "Hello"

### Части эндпоинта

#### Декоратор эндпоинта

1. Путь так так

@app.get("/)

1. Путь может иметь параметры о чем подробнее в Обработка запросов эндпоинтом

#### Функция эндпоинта

1. Функция эндпоинта представляет собой алгоритм действий необходимых совершить при получении запроса на этот эндпоинт.

Этот алгоритм может зависить от того какой запрос придёт – то есть для разных запросов разные алгоритмы действий

def endpoint1():

return "Hello"

1. Аргументами функции эндпоинта являются – информация из http запроса (параметры и данные запроса), пришедшего к этому эндпоинту.

Об этом подробнее в Обработка запросов эндпоинтом.

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Обработка запросов эндпоинтом

1. Общая схема
2. Uvicorn – FastAPI сервер обслуживает внешние запросы\*, приходящие на него.

\* Внешними запросами в основном явлются http/https, но также поддерживается websocket

1. Uvicorn принимает запрос и передаёт его FastAPI (ASGI).
2. Проверяя path http запроса и сравнивая с декларируемыми путями эндпоинтов FastAPI находит нужный (проверка происходит сверху вниз).

Таким образом внешний запрос приходит на конкретный эндпоинт.

### Извлечение данных из запроса, приходящего на эндпоинт

1. Эндпоинт извлекает информацию из http запроса с помощью механизма dependency injection

fastapi извлекает например хэдер и передаёт в аргументы функции.

* Path параметры
* Query параметры
* Header параметры
* Body данные

# Параметры HTTP запроса

По http запросы можно передать параметры несколькими способами

1) параметры пути (path parameters)

http:/www.google/{par1}/…

Самый простой способ передать параметры

В фаст апи можно передавать их как

@app.get("/todos/{id}/", response\_model=str)

def get\_two\_params(id: str):

    return f"id: {id}"

2) query

http:/www.google/?par1=1&par2=2

В принципе тоже самое тоже передаются в самом пути, но начинаются с ? и имеют наименование

В сваггере path и query также не будут отличаться, только названием внизу соответсвенно query или path

3) header parameters

Это параметр который не виден в пути и передаётся в header-e запроса:

Запрос с хедер параметрами:

curl -X 'GET' \

'http://localhost:8000/todos/' \

-H 'accept: application/json' \

-H 'token: qwe'

При этом -H 'accept: application/json' - формат файла в котором сервер должен прислать body ответа.

А -H 'token: qwe' - header параметр нашего запроса внутри файла который нам прислал сервер.

Как обработать хедер параметры в фастапи:

@app.get("/todos/", response\_model=str)

def get\_two\_params(token: str = Header()):

        return token

Указывая token: str = Header() – мы говорим, что это из хэдера. При этом важно понимать, что хэдер параметры могут приходить ТОЛЬКО в виде string, которую потом парсят.

4) body параметры

Работает также как и с хеадер параметрами