

Установка и подготовка

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

Создаем файл main.py. Импортируем fastapi и pyjokes в main.py

```
from fastapi import FastAPI
import pyjokes
```

Создадим объект фастапи, куда далее будут подключаться роуты. Будем называть его далее приложение фастапи.

```
app = FastAPI()
```

Создадим простой роут. Для этого напишем простую функцию, которую обернем декоратором. Декоратор использует приложение созданное ранее, http метод и путь по которому будет работать данный роут.

```
@app.get("/")
def joke():
    return pyjokes.get__joke()
```

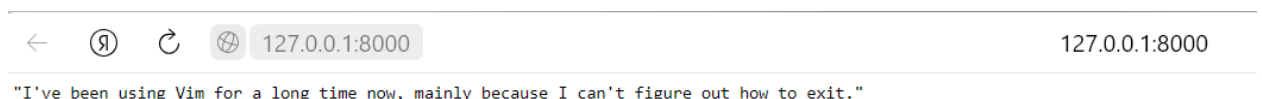
Для начала работы необходимо запустить uvicorn – наш веб-сервер. Воспользуемся командой в консоли, запущенной в месте в одной директории с main.py.

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

Результат запуска uvicorn.

```
K:\MTUCI\Students\FastAPI_01>uvicorn main:app --reload
[32mINFO[0m: Will watch for changes in these directories: ['K:\\MTUCI\\Students\\FastAPI_01']
[32mINFO[0m: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
[32mINFO[0m: Started reloader process [14448] using Watchdog
[32mINFO[0m: Started server process [14744]
[32mINFO[0m: Waiting for application startup.
[32mINFO[0m: Application startup complete.
```

Перейдем по базовому адресу, который указывается при запуске uvicorn - <http://127.0.0.1:8000>. Перейдя по ссылке, увидим результат, как на рисунке ниже.

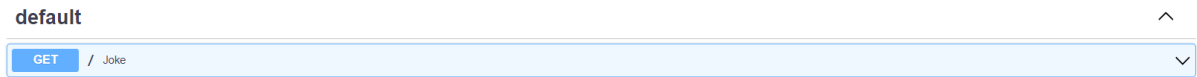


← ⓘ ↻ 🌐 127.0.0.1:8000 127.0.0.1:8000

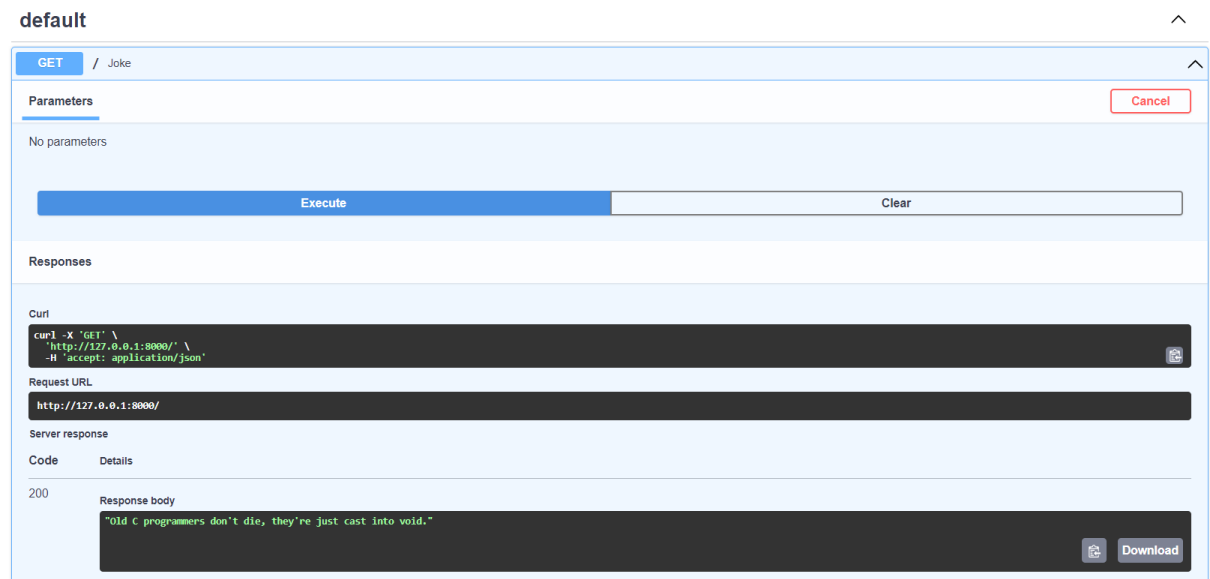
"I've been using Vim for a long time now, mainly because I can't figure out how to exit."

Для удобной работы с нашим приложением будем использовать swagger. Он открывается по следующей ссылке <http://127.0.0.1:8000/docs>. По данной ссылке мы должны увидеть изображение ниже.

FastAPI 0.1.0 OAS3
/openapi.json



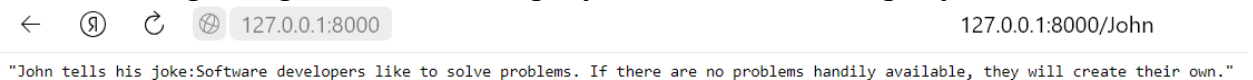
На странице сваггера отображаются все добавленные роуты. Развернем наш единственный роут и попробуем выполнить его. Для этого сначала нажмем на кнопку “Try it out” и далее execute.



Добавим еще один роут, где будет параметр в пути, чтобы мы могли представить шутку от какого-то конкретного человека. Для этого в фигурных скобках добавим название желаемого параметра и добавим его же в параметрах функции. Итоговый вид у роута будет, как на рисунке ниже.

```
@app.get("/{friend}")
def friends_joke(friend: str):
    return friend + " tells his joke:" + pyjokes.get_joke()
```

Добавим к базовому пути <http://127.0.0.1:8000/John> через слеш желаемое значение параметра name, чтобы результат был, как на рисунке ниже.



В сваггере новый роут будет выглядеть следующем образом. Попробуем снова его запустить.

GET /{friend} Friends Joke

Parameters

Name	Description
friend * required string (path)	John

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/John' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/John
```

Server response

Code	Details
200	<p>Response body</p> <pre>"John tells his joke:Speed dating is useless. 5 minutes is not enough to properly explain the benefits of the Unix philosophy."</pre> <p>Response headers</p>

Добавим еще один роут, где будет возможность выбрать количество шуток. Для этого добавим еще один параметр, который не будем указывать в пути роута. Это будет query параметр `jokes_number`. Он не будет указан в пути, но также необходим для корректной работы роута. Итоговый вид роута показан на изображении ниже.

```
@app.get("/multi/{friend}")
def multi_friends_joke(friend: str, jokes_number: int):
    result = ""
    for i in range(jokes_number):
        result += friend + f" tells his joke #{i + 1}: " + pyjokes.get_joke() + "
    "
    return result
```

Откроем новый роут в сваггере и попробуем запустить его.

GET /multi/{friend} Multi Friends Joke

Parameters

Name	Description
friend * required string (path)	John
jokes_number * required integer (query)	2

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/multi/John?jokes_number=2' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/multi/John?jokes_number=2
```

Server response

Code	Details
200	Response body "John tells his joke #1: Two bytes meet. The first byte asks, 'Are you ill?' The second byte replies, 'No, just feeling a bit off.' John tells his joke #2: There are 10 types of people: those who understand hexadecimal and 15 others."

Download

Рассмотрим и другой способ передачи информации в роут. Например, некоторые http запросы поддерживают передачу данных в теле запроса (Body). Создадим новый роут, используя метод POST, и создадим схему тела запроса, которую будет принимать роут для корректной работы. Импортируем из библиотеки pydantic класс BaseModel, как показано на рисунке ниже.

```
from pydantic import BaseModel
```

И создадим на его основе схему получения шутки, которую и передадим функции на вход, как показано ниже.

```
class Joke(BaseModel):
    friend: str
    joke: str

class JokeInput(BaseModel):
    friend: str

@app.post("/")
def create_joke(joke_input: JokeInput):
    return joke_input.friend + " tells his joke:" + pyjokes.get_joke()
```

Откроем сваггер и протестируем новый метод.

POST / Create Joke

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{  
  "friend": "John"  
}
```

Execute

Clear

Результат успешного запроса показан ниже.

Responses

Curl

```
curl -X 'POST' \  
  'http://127.0.0.1:8000/' \  
  -H 'accept: application/json' \  
  -H 'content-type: application/json' \  
  -d '{  
    "friend": "John"  
  }'
```

Request URL

```
http://127.0.0.1:8000/
```

Server response

Code	Details
200	<div>Response body<div><pre>"John tells his joke:What do you call a programmer from Finland? Nerdic."</pre></div><div><div>Download</div></div></div>

Схемы позволяют и задавать образец ожидаемого ответа. Обновим ранее созданный POST метод, согласно примеру ниже, чтобы ответ выдавался в формате ранее описанным в схеме Joke.

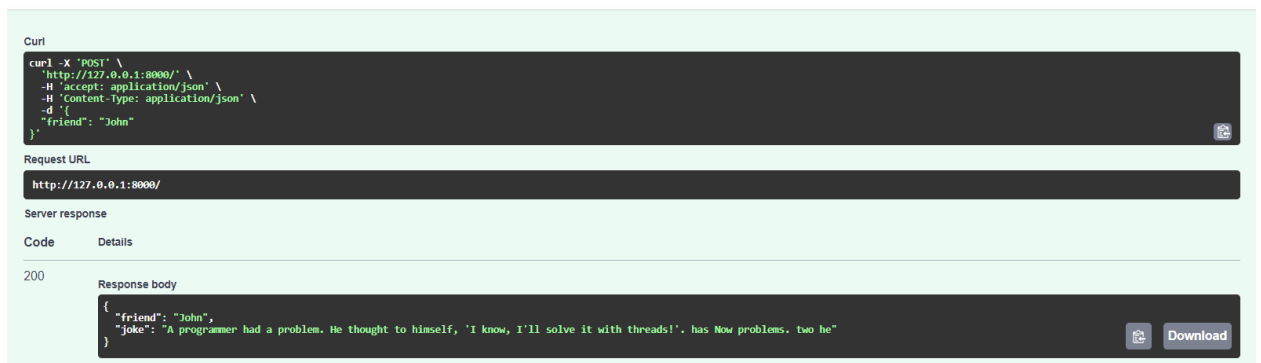
```
@app.post("/")  
def create_joke(joke_input: JokeInput):  
    return Joke(friend=joke_input.friend, joke=pyjokes.get_joke())
```

Посмотрим на изменения в сваггере и протестируем обновленный роут. Перед запуском обратим внимание, что схема не показывается в ожидаемом ответе.

Example Value | Schema

```
"string"
```

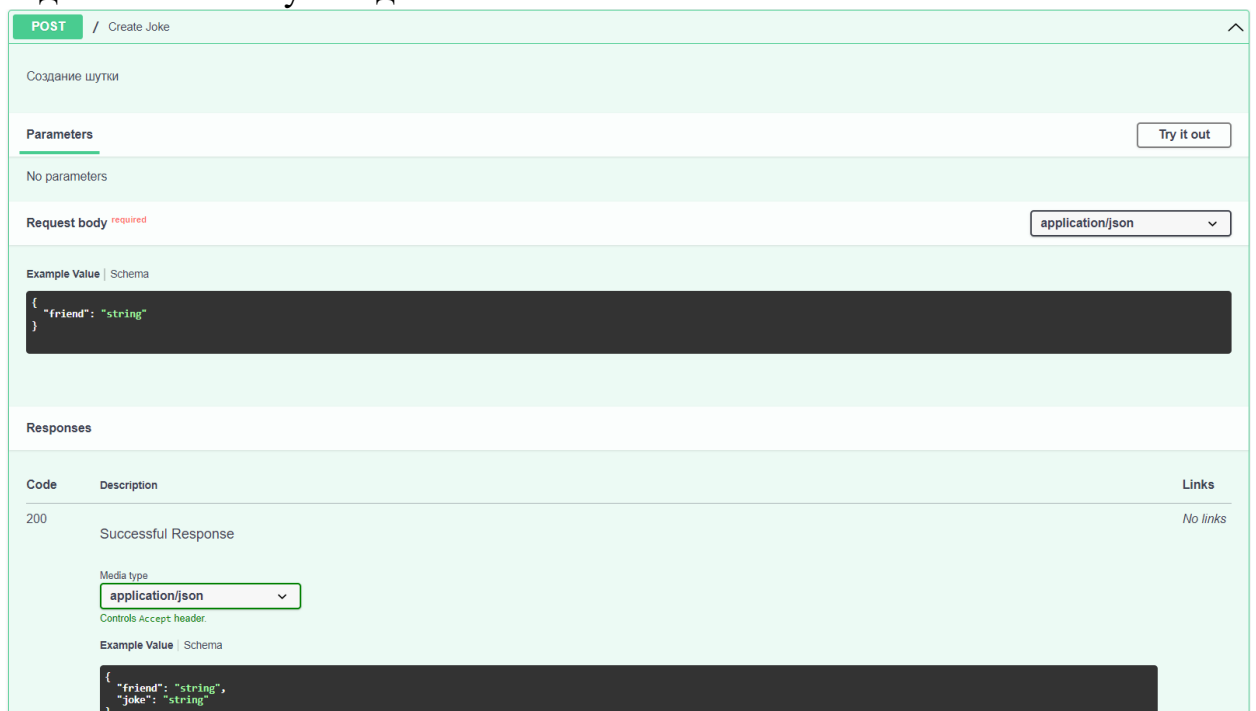
Результат успешного выполнения показан ниже.



Добавим комментарий с описанием роута и поставим в параметрах `response_model`, чтобы добавить валидацию ответа функции. Обновленный эндпоинт будет выглядеть следующим образом.

```
@app.post("/", response_model=Joke)
def create_joke(joke_input: JokeInput):
    """Создание шутки"""
    return Joke(friend=joke_input.friend, joke=pyjokes.get_joke())
```

Обновим страницу со сваггером и увидим добавленное описание эндпоинта и схему ожидаемого ответа.



Самостоятельно задание:

Создать простой REST сервис, использующий библиотеку `wikipedia`. Создайте 1 роут с параметром `path`, 1 роут с параметром `query`, 1 роут с передачей параметров в теле запроса. Все запросы должны возвращаться и валидироваться по схемам.

Ссылка на документацию по `Wikipedia` в `python`

https://rukovodstvo.net/posts/id_1061/