**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования «Московский технический университет связи и информатики».**

**Предмет: ВвИТ**

Лабораторная работа №14

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("/")

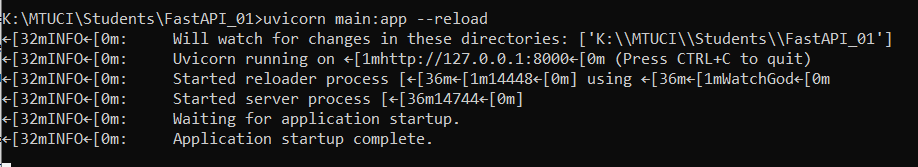
def joke():

    return pyjokes.get\_joke()

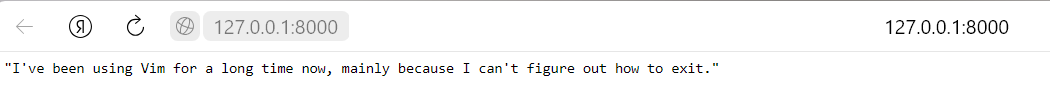
Для начала работы необходимо запустить uvicorn – наш веб-сервер. Воспользуемся командой в консоли, запущенной в месте в одной директории с main.py.

uvicorn main:app --reload

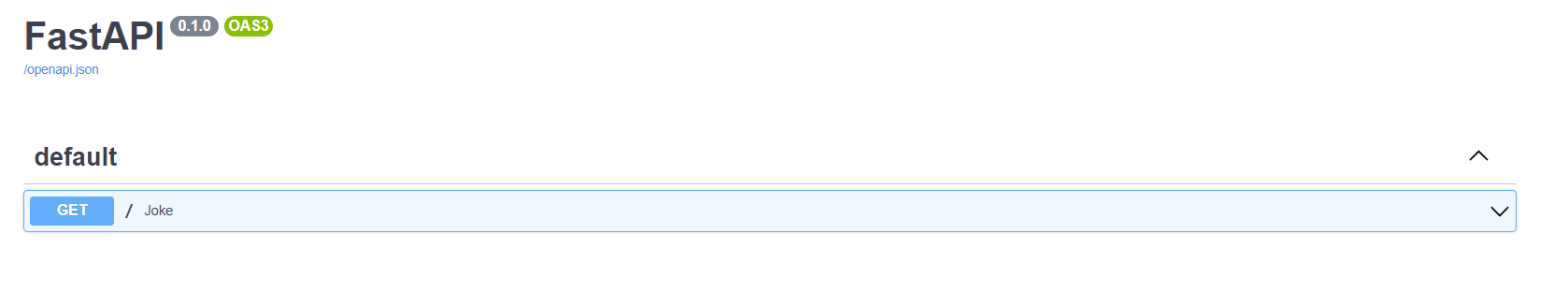
Результат запуска uvicorn.



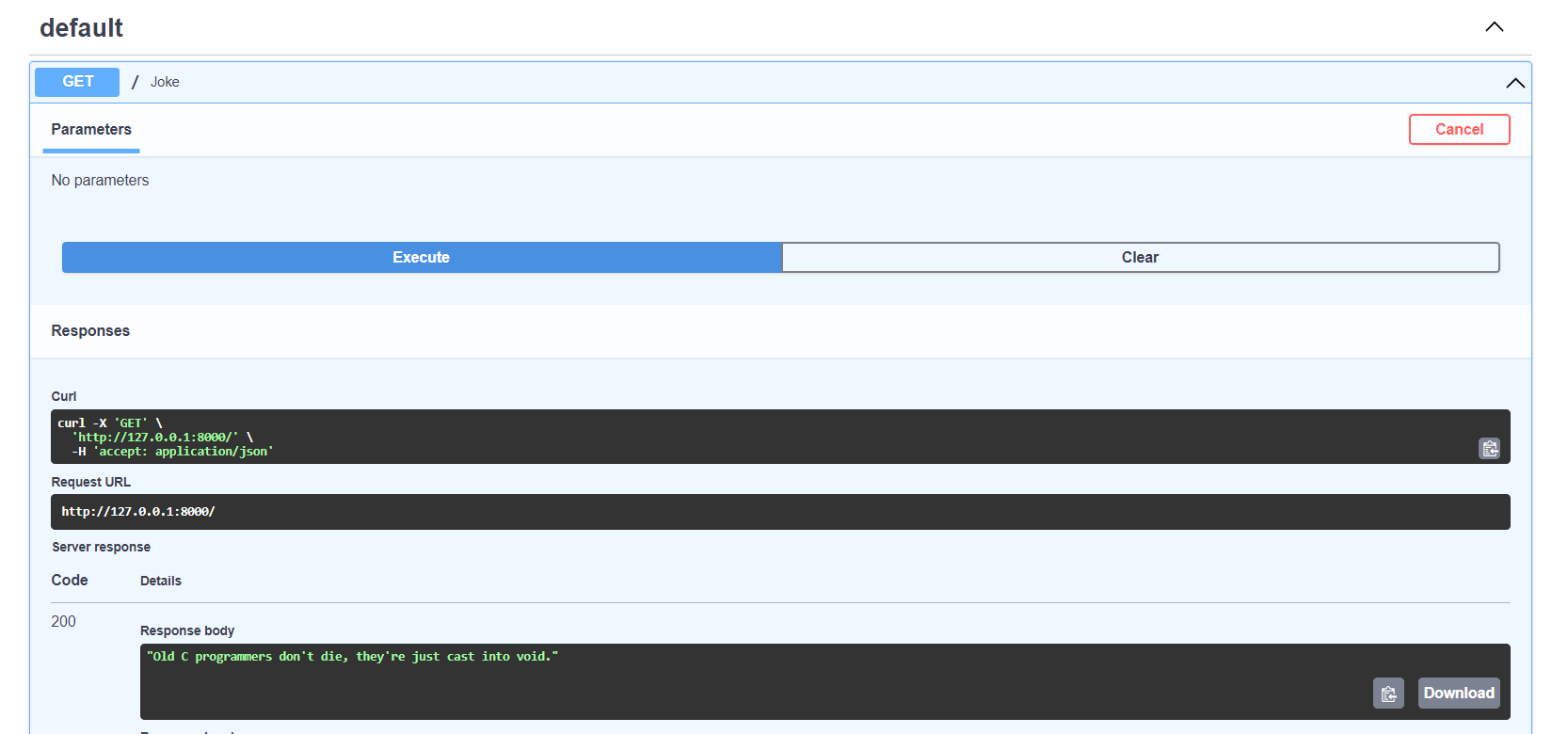
Перейдем по базовому адресу, который указывается при запуске uvicorn - <http://127.0.0.1:8000>. Перейдя по ссылке, увидим результат, как на рисунке ниже.



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.



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.



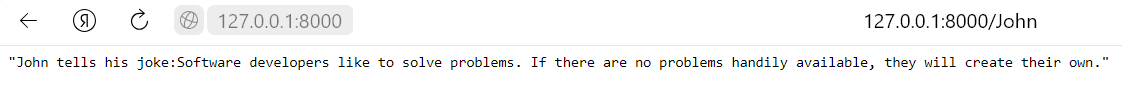
Добавим еще один роут, где будет параметр в пути, чтобы мы могли представить шутку от какого-то конкретного человека. Для этого в фигурных скобках добавим название желаемого параметра и добавим его же в параметрах функции. Итоговый вид у роута будет, как на рисунке ниже.

@app.get("/{friend}")

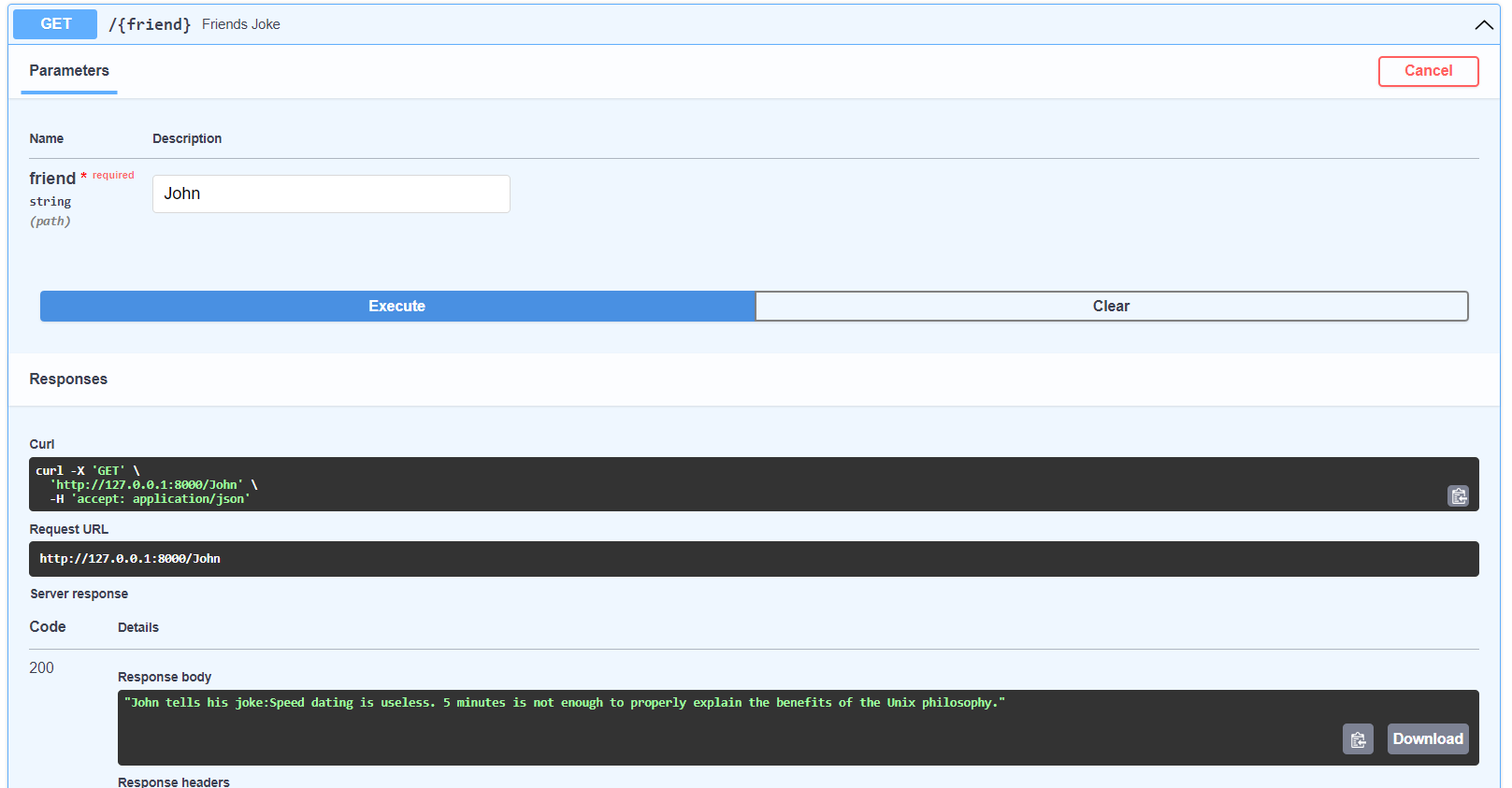
def friends\_joke(friend: str):

    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.



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):

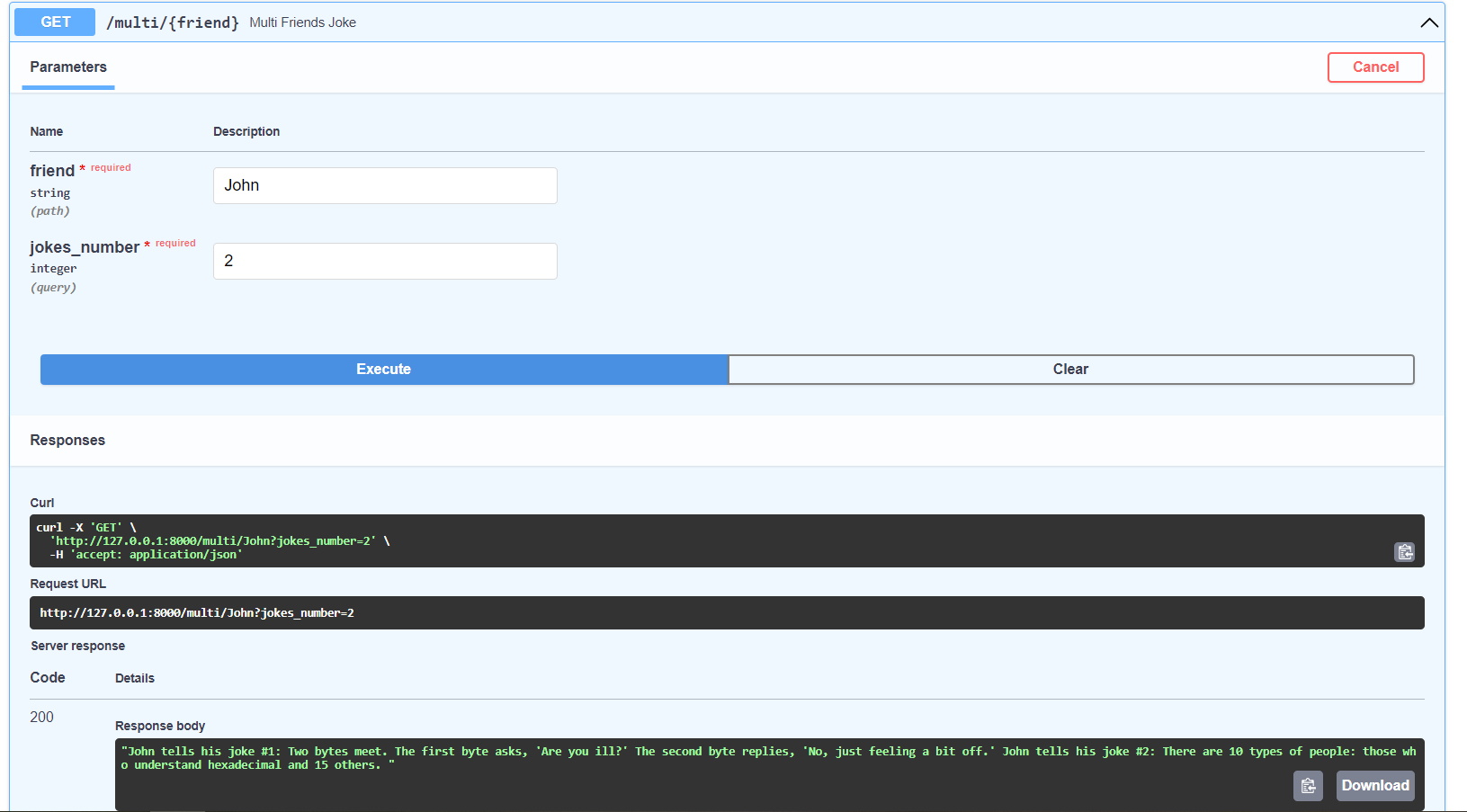
    result = ""

    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.



Рассмотрим и другой способ передачи информации в роут. Например, некоторые http запросы поддерживают передачу данных в теле запроса (Body). Создадим новый роут, использую 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):

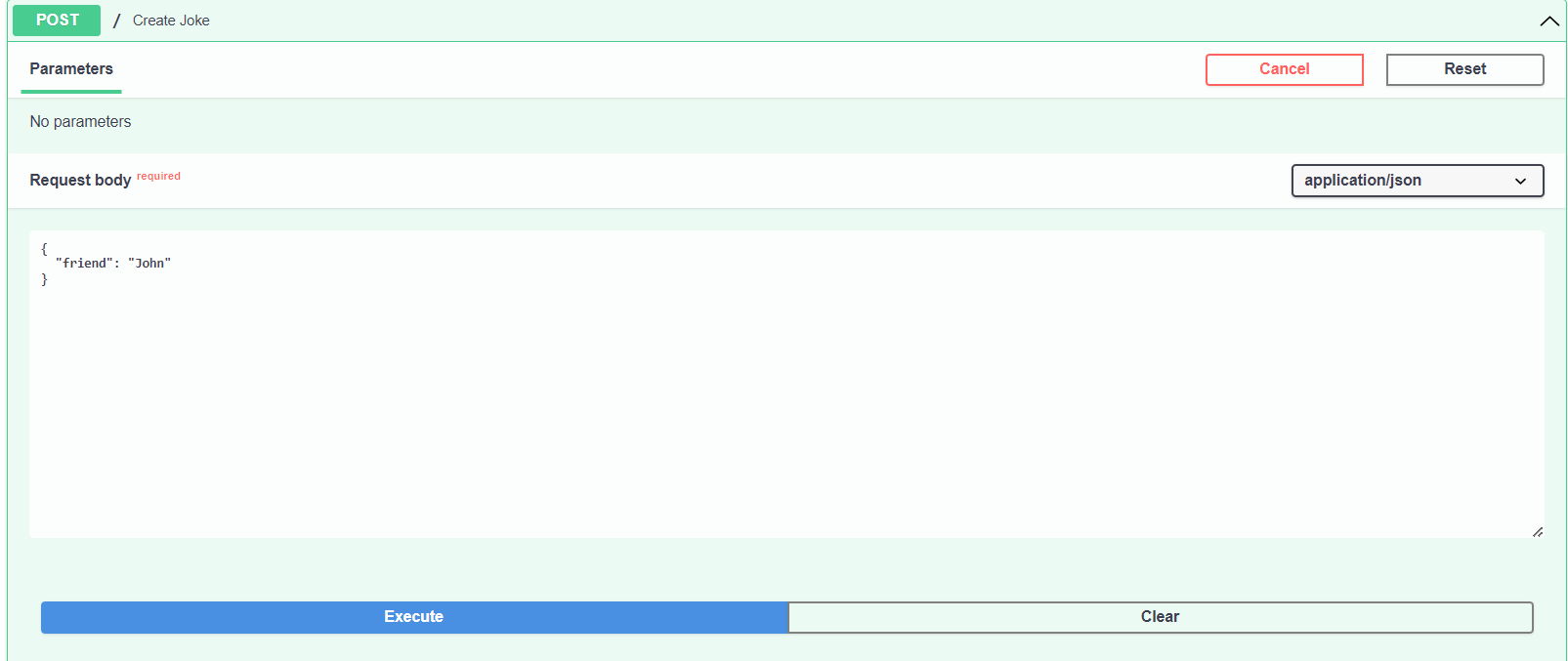
    friend: str

@app.post("/")

def create\_joke(joke\_input: JokeInput):

    return joke\_input.friend + " tells his joke:" + pyjokes.get\_joke()

Откроем сваггер и протестируем новый метод.



Результат успешного запроса показан ниже.



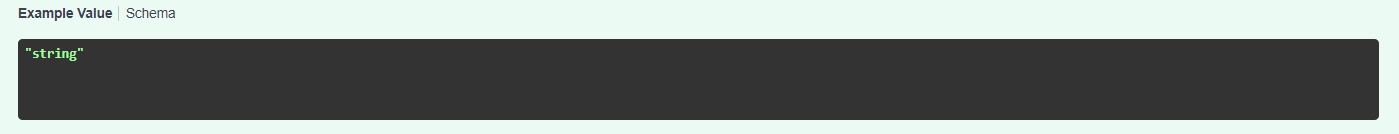
Схемы позволяют и задавать образец ожидаемого ответа. Обновим ранее созданный POST метод, согласно примеру ниже, чтобы ответ выдавался в формате ранее описанным в схеме Joke.

@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.



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.

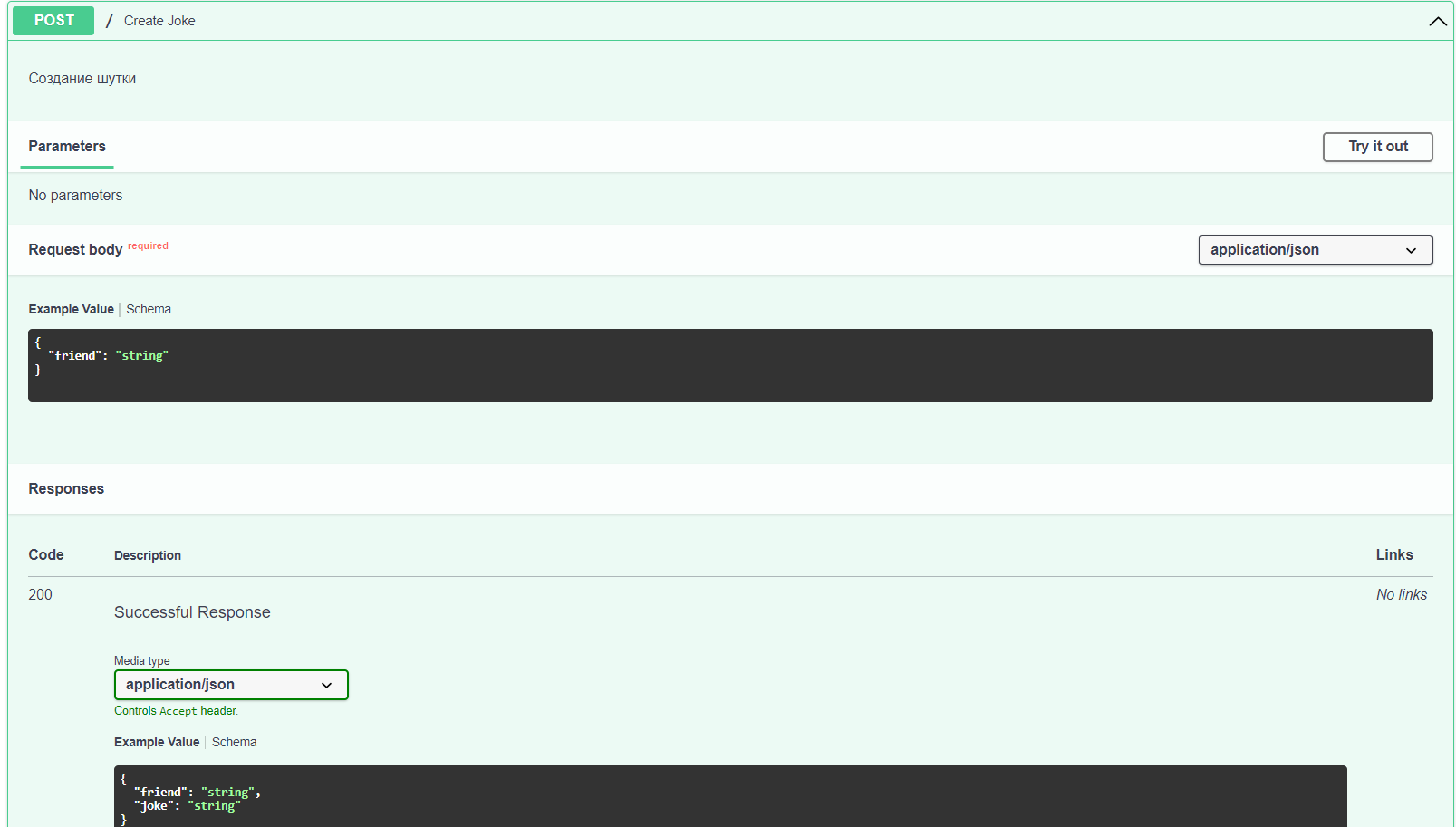
@app.post("/", response\_model=Joke)

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 валидирироваться по схемам.

Ссылка на документацию по Wikipedia в python

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

