**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая кибернетика и информационные технологии»

**Отчет по практической работе**

по дисциплине «Основы работы с FastAPI»

Выполнила: студентка группы БПИ2401 Рябова Екатерина

Проверил: Мкртчян Грач Маратович

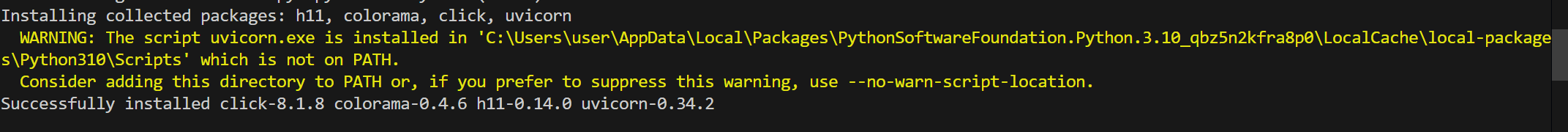
Москва

2025

1. **Цель работы:**

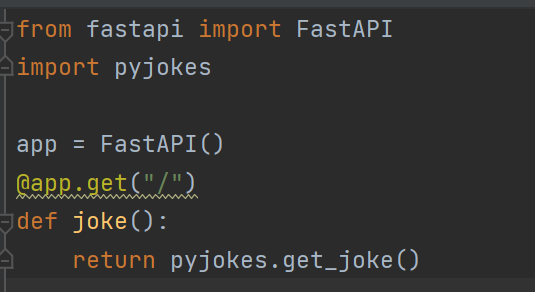
Получить практические навыки в использовании FastAPI

1. **Ход работы:**

****

Проверим что пакеты установлены и доустановим необходимые библиотеки. (Так как запуск из терминала/командной строки усложняется добавлением пути, по которому лежит uvicorn, впоследствии будем запускать сервер прям в коде)

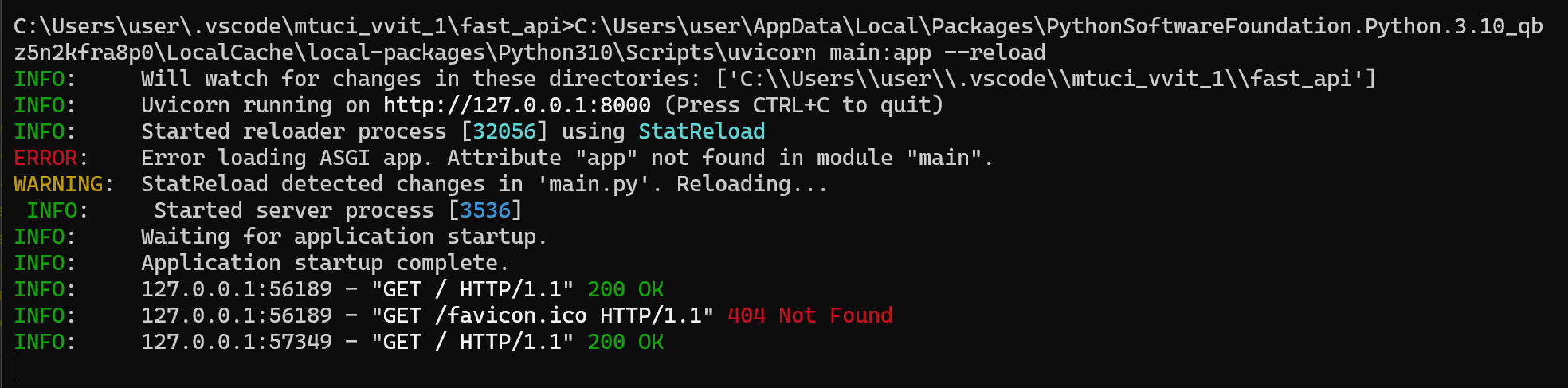
Создадим файл main, в нем - app (объект FastAPI) и напишем основной роут.



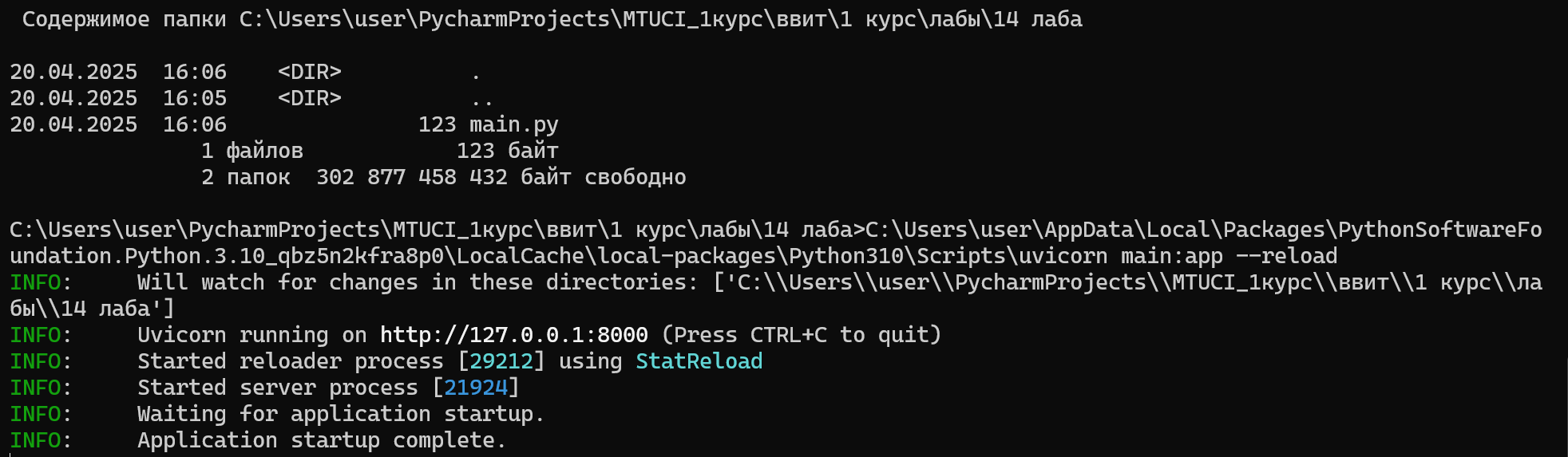
Теперь займемся запуском uvicorn – веб-сервера.

Воспользуемся командой в консоли, запущенной в месте в одной директории с main.py.

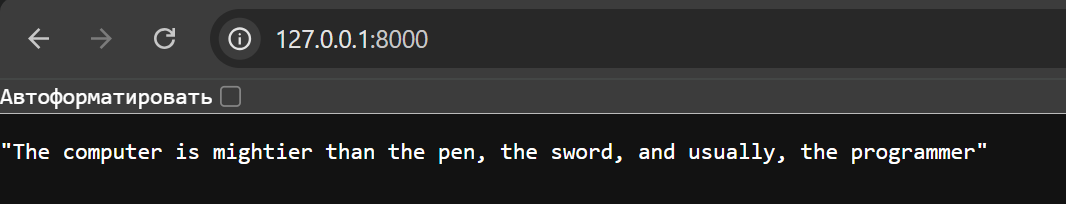
uvicorn main:app --reload

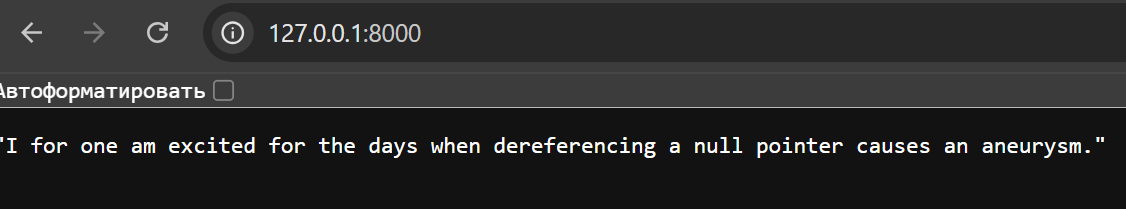


Заметим, что в vscode выдается ошибка связанная с ненахождением app. Перейдем в Pycharm.

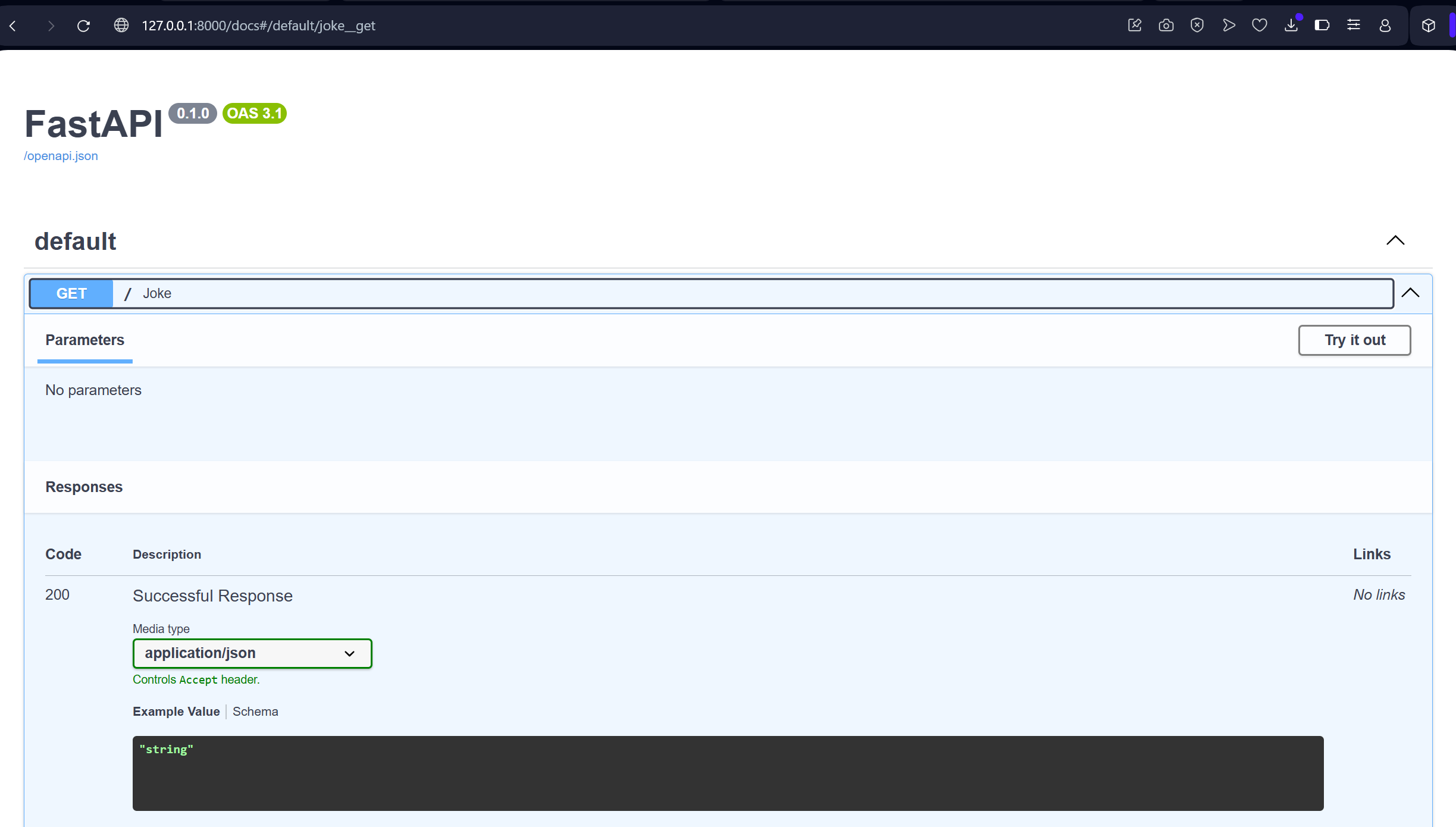


Здесь сервер великолепно запускается и шутит.

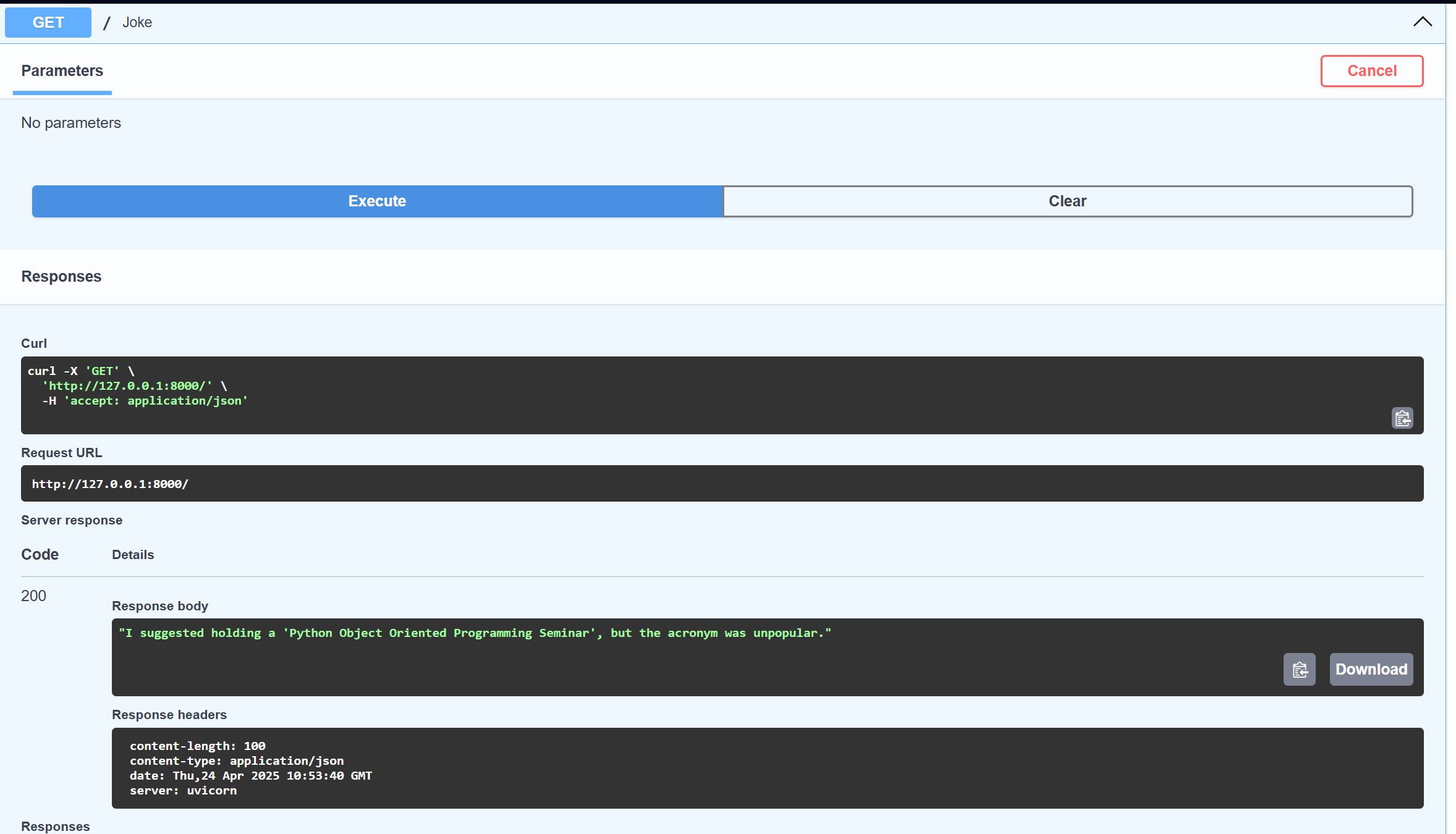




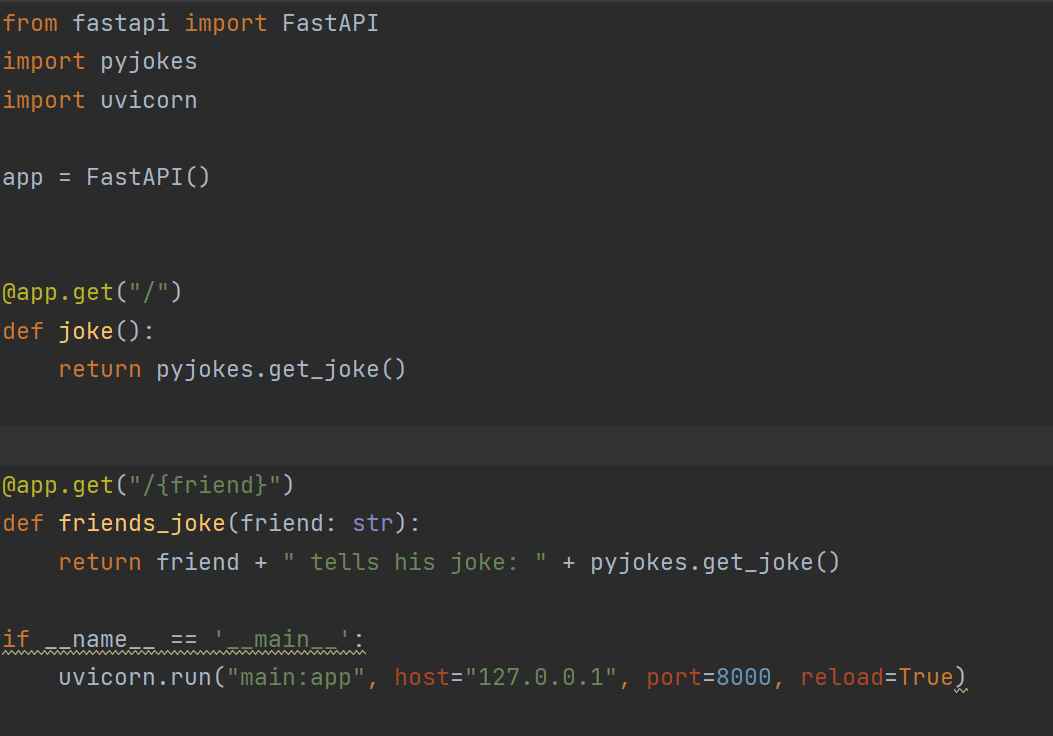
Откроем Swagger (/docs)



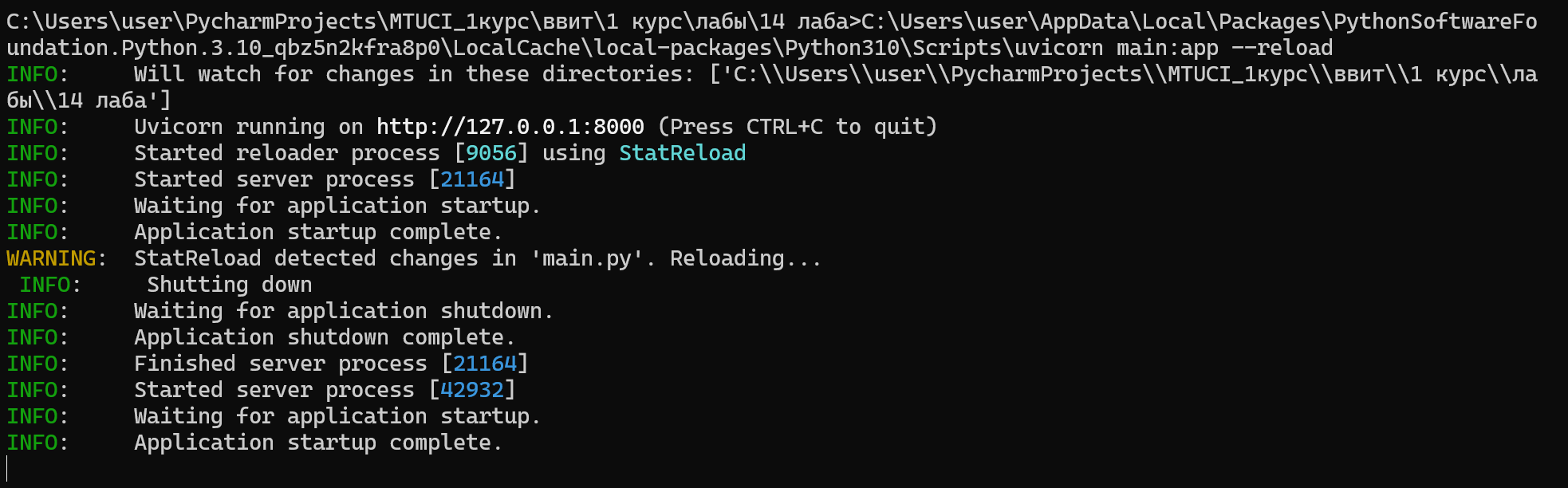
Отправим запрос серверу.



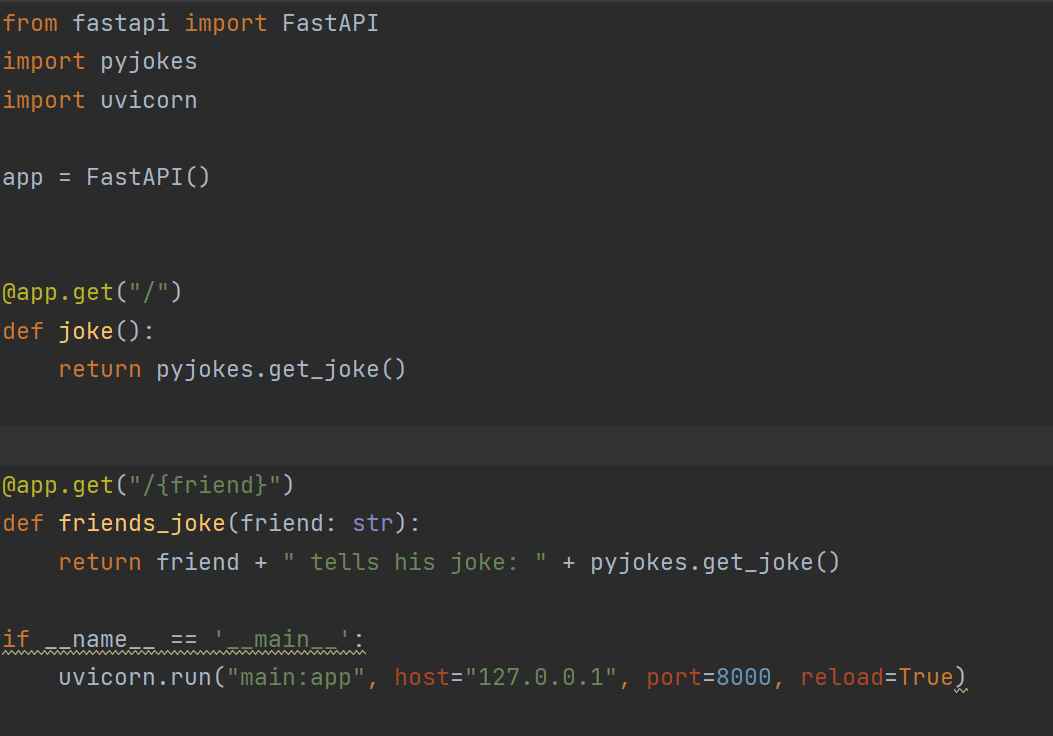
Добавим еще один роут, где будет параметр в пути, чтобы мы могли представить шутку от какого-то конкретного человека. Для этого в фигурных скобках добавим название желаемого параметра и добавим его же в параметрах функции.

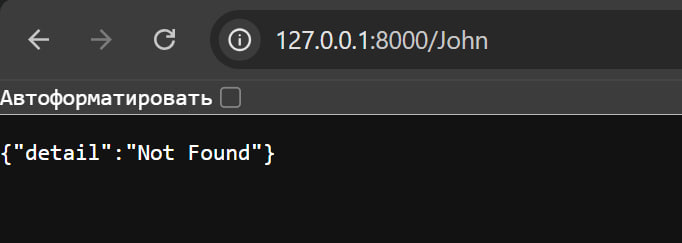


Заметим, что сервер обновляется при каждом изменении кода.

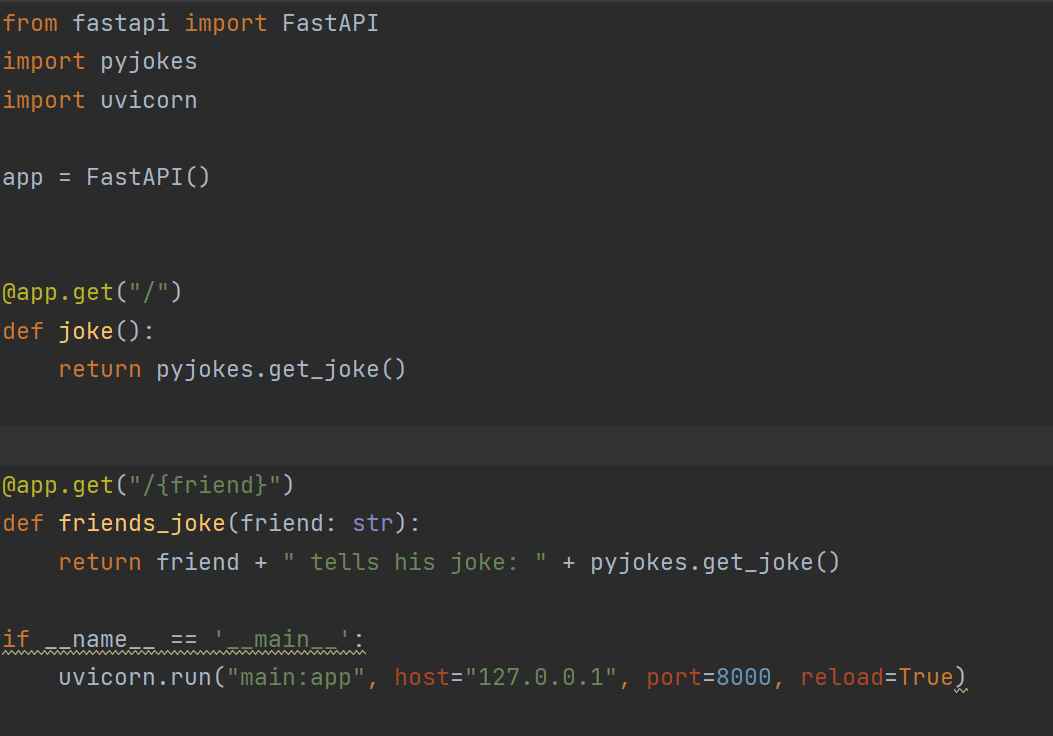


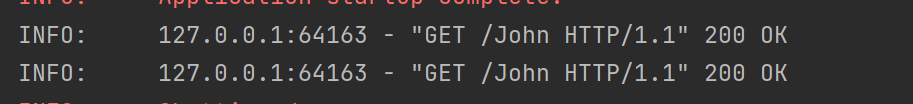
Но при этом при добавлении нового роута не замечает его и выдает ошибку.



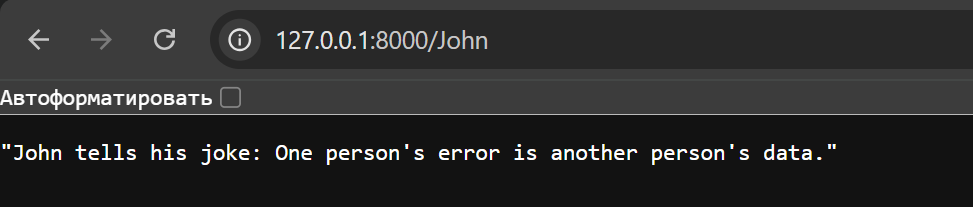


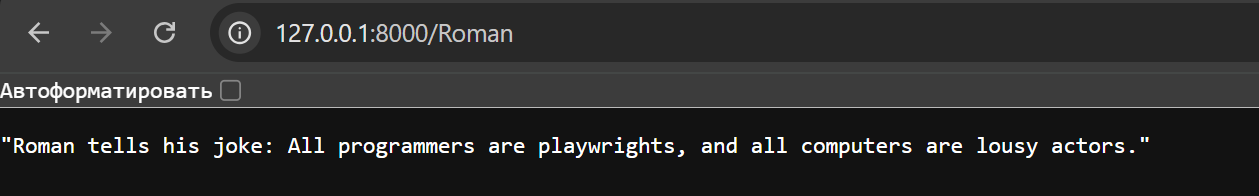
При этом, новый роут не появляется и в Swagger. Именно поэтому воспользуемся следующей конструкцией развертки сервера:

(В самом начале импортируем библиотеку uvicorn)

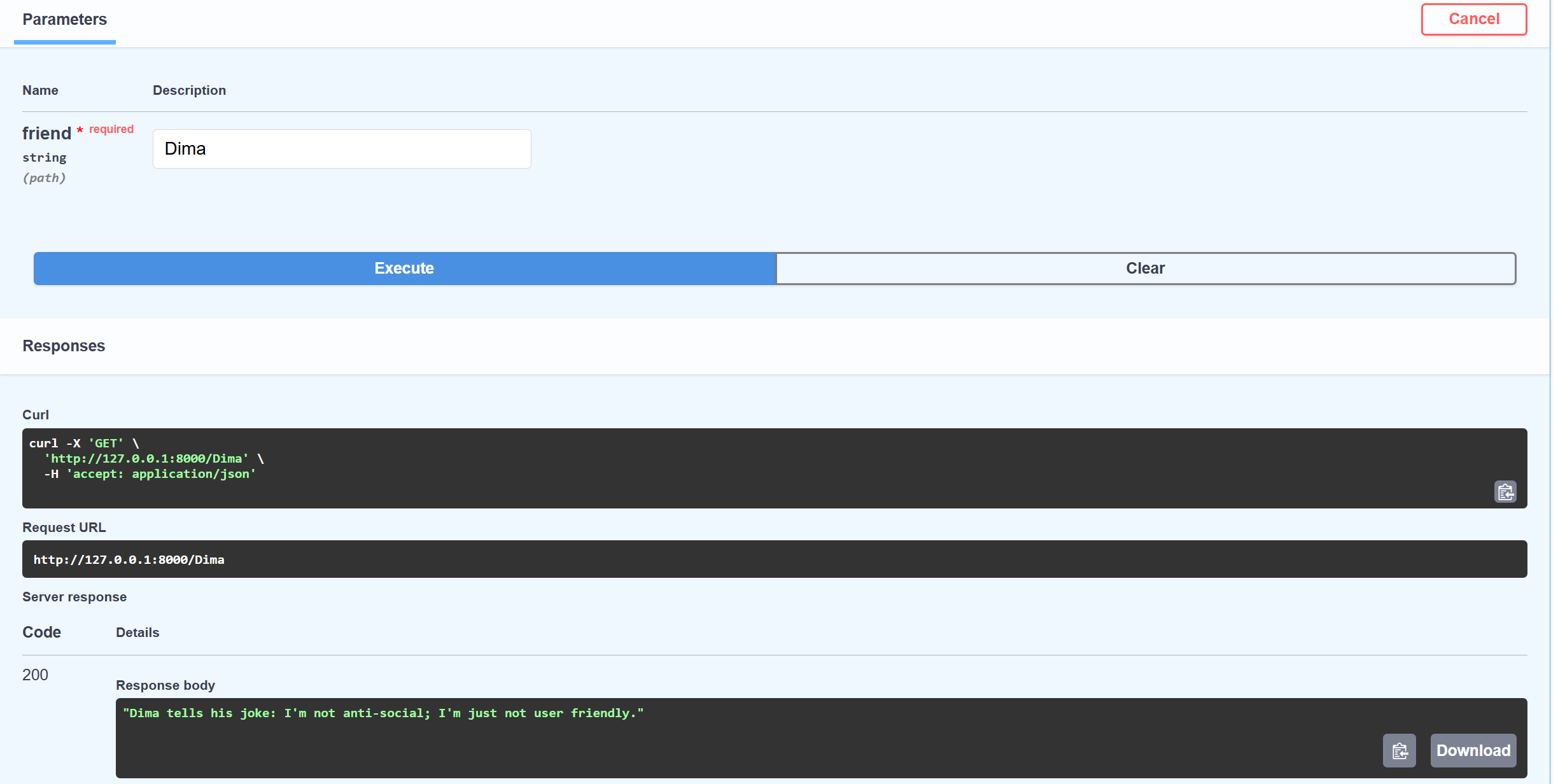


Теперь даже видны ответы сервера.





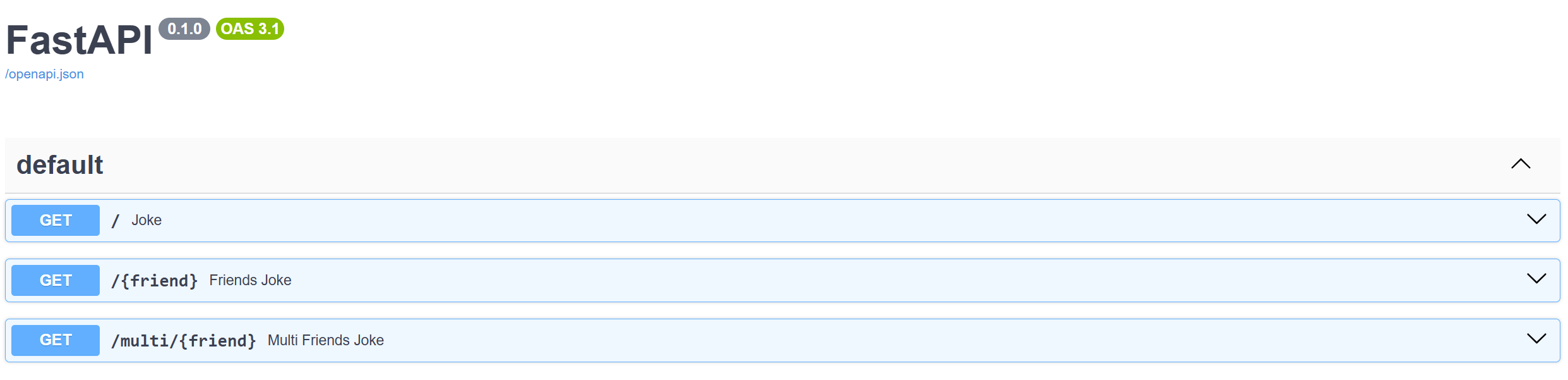
И друзья начали шутить.



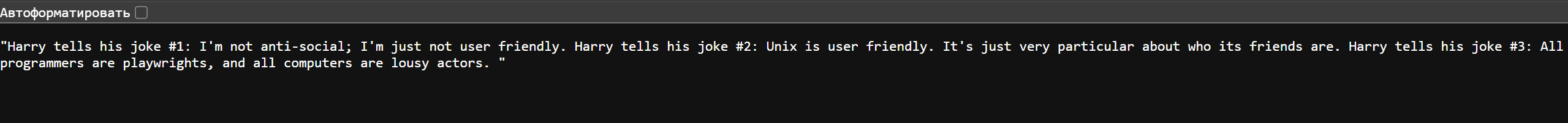
Проверим это так же через Swagger.



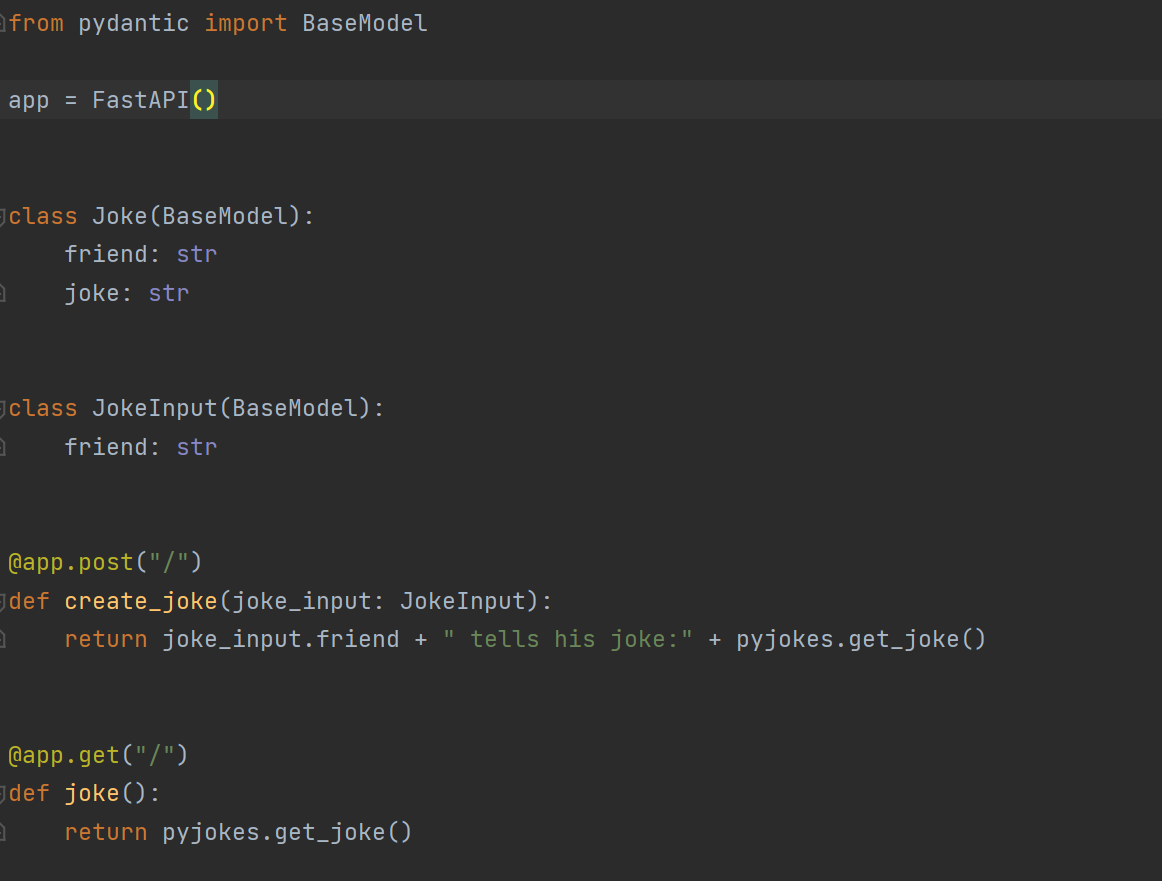
Добавим еще один роут, где будет возможность выбрать количество шуток. Для этого добавим еще один параметр, который не будем указывать в пути роута. Это будет query параметр jokes\_number. Он не будет указан в пути, но также необходим для корректной работы роута.

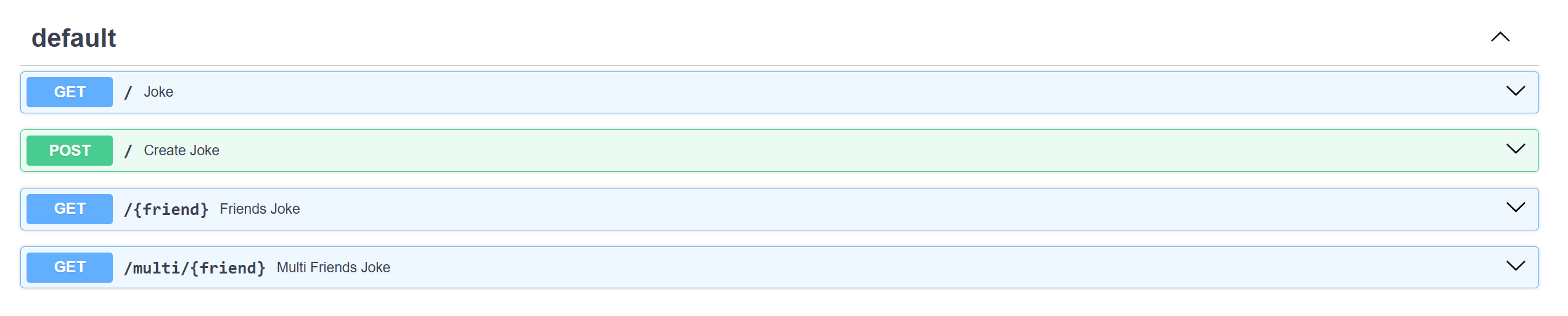






Рассмотрим и другой способ передачи информации в роут. Например, некоторые http запросы поддерживают передачу данных в теле запроса (Body). Создадим новый роут, использую метод POST, и создадим схему тела запроса, которую будет принимать роут для корректной работы. Импортируем из библиотеки pydantic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

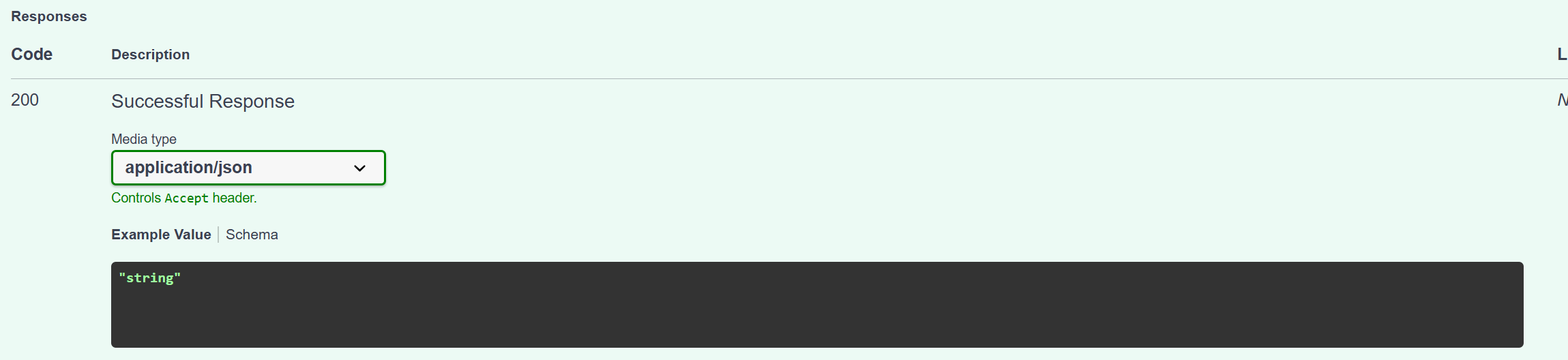


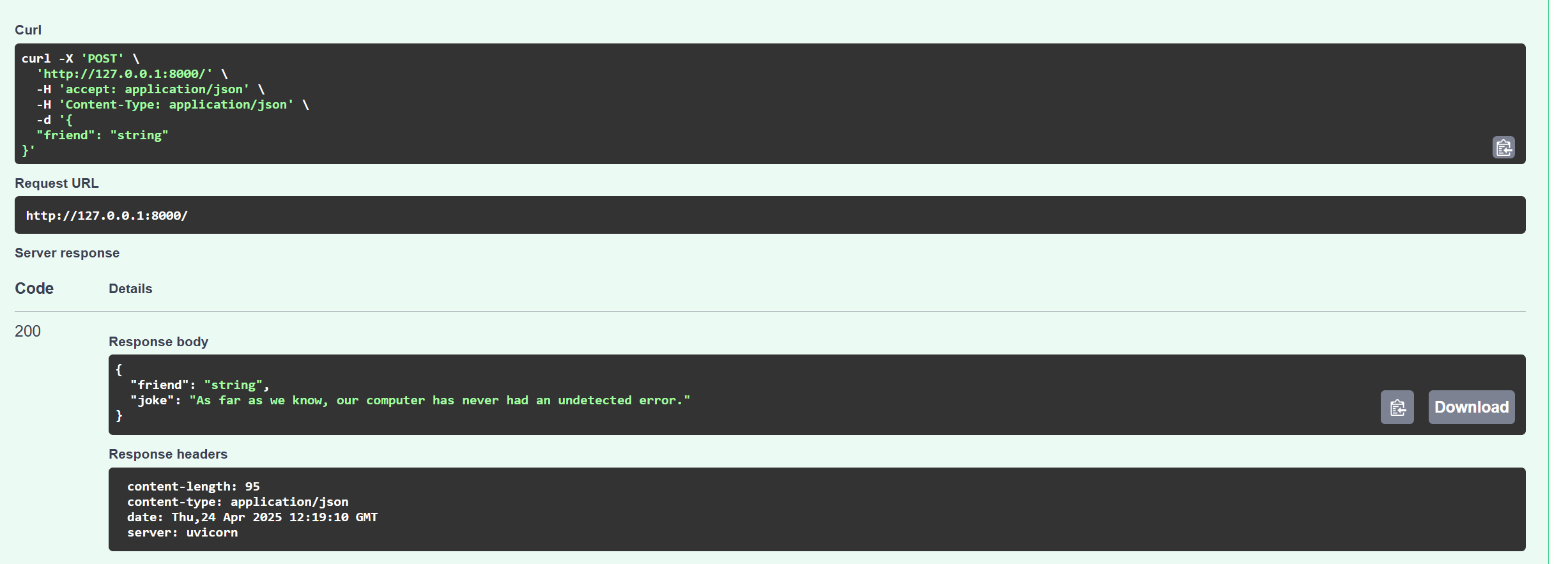




Немного модифицируем POST. Схемы позволяют и задавать образец ожидаемого ответа. Обновим ранее созданный POST метод, согласно примеру ниже, чтобы ответ выдавался в формате ранее описанным в схеме Joke.





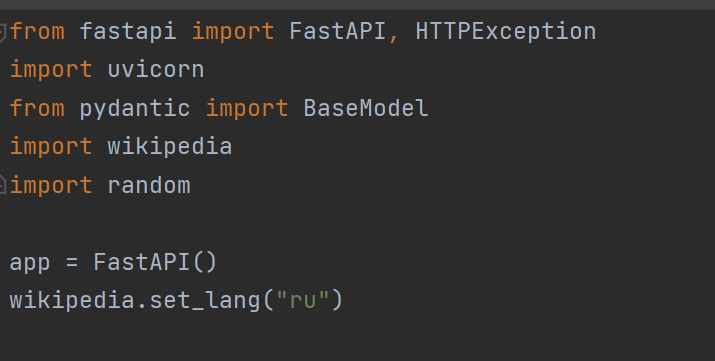




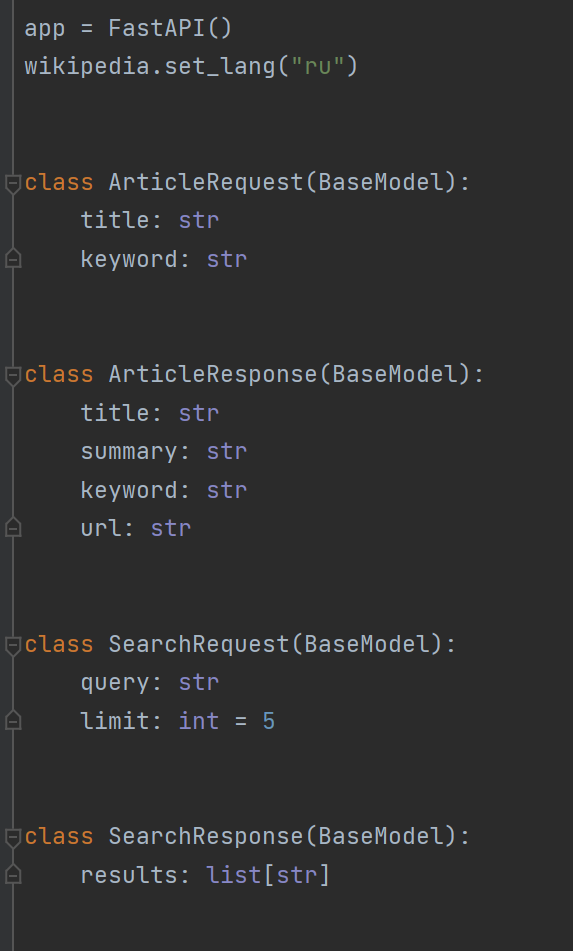
Проверяем, что все работает как надо.

Самостоятельное задание:

1. Импортируем необходимые библиотеки, в том числе uvicorn, Wikipedia и random (для запроса post). Создаем app от FastAPI и меняем язык в википедии на русский.



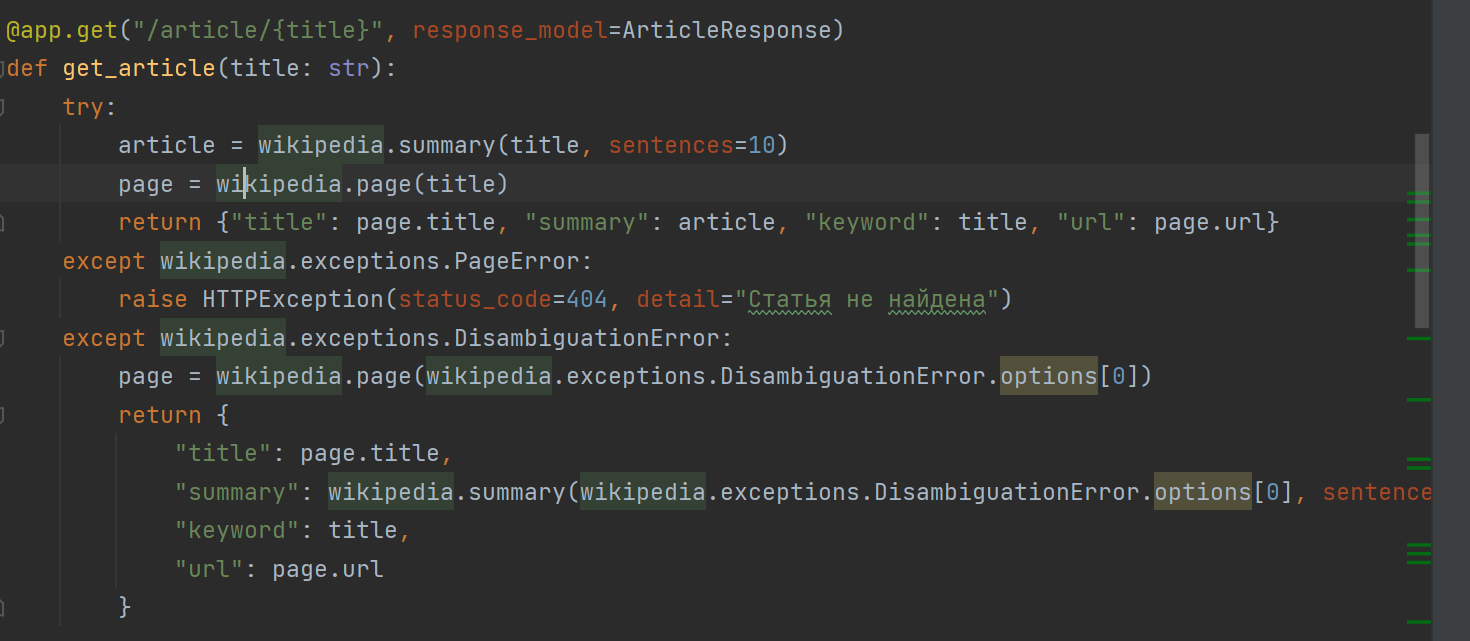
1. Создаем схемы Pydantic для валидации и форматирования (создадим схему тела запроса, которую будет принимать роут для корректной работы.)



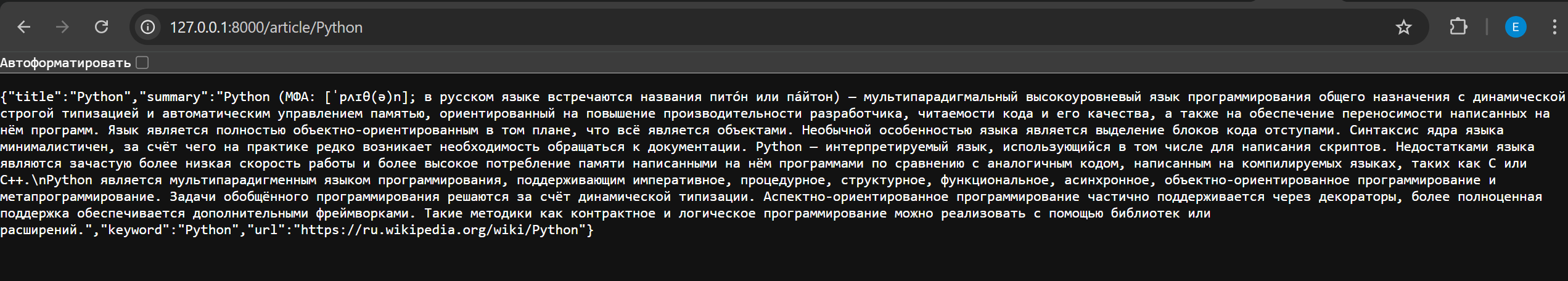
1. Первый, get-запрос с path-параметром. Пусть он ищет статью по названию. Используем метод summary, в качестве параметра укажем количество предложений – 10. (То же ограничение прописано и в шаблонах pydantic). Так же находим саму страницу с этой статьей, чтобы вернуть пользователю url. Оборачиваем это все в конструкцию try-except и проверяем на две ошибки (здесь и дальше): wikipedia.exceptions.PageError – страница не найдена, возвращаем статус-код 404 – и wikipedia.exceptions.DisambiguationError – неоднозначный ответ. В обработке второй ошибки можем тоже вернуть ошибку, а можем взять первый из всех результатов



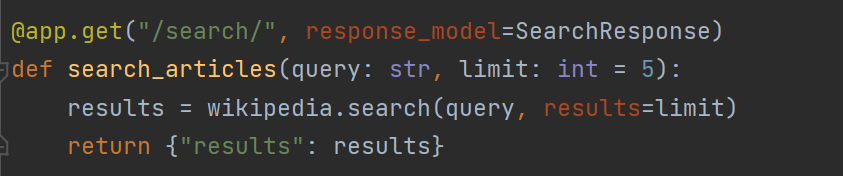
Или



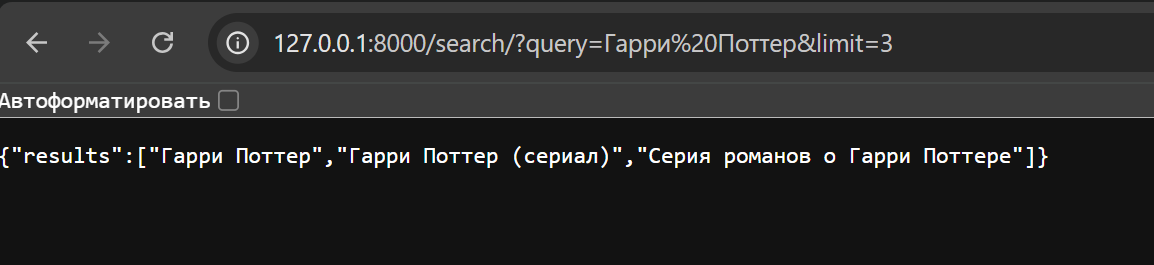
Пример запроса: <http://127.0.0.1:8000/article/Python>



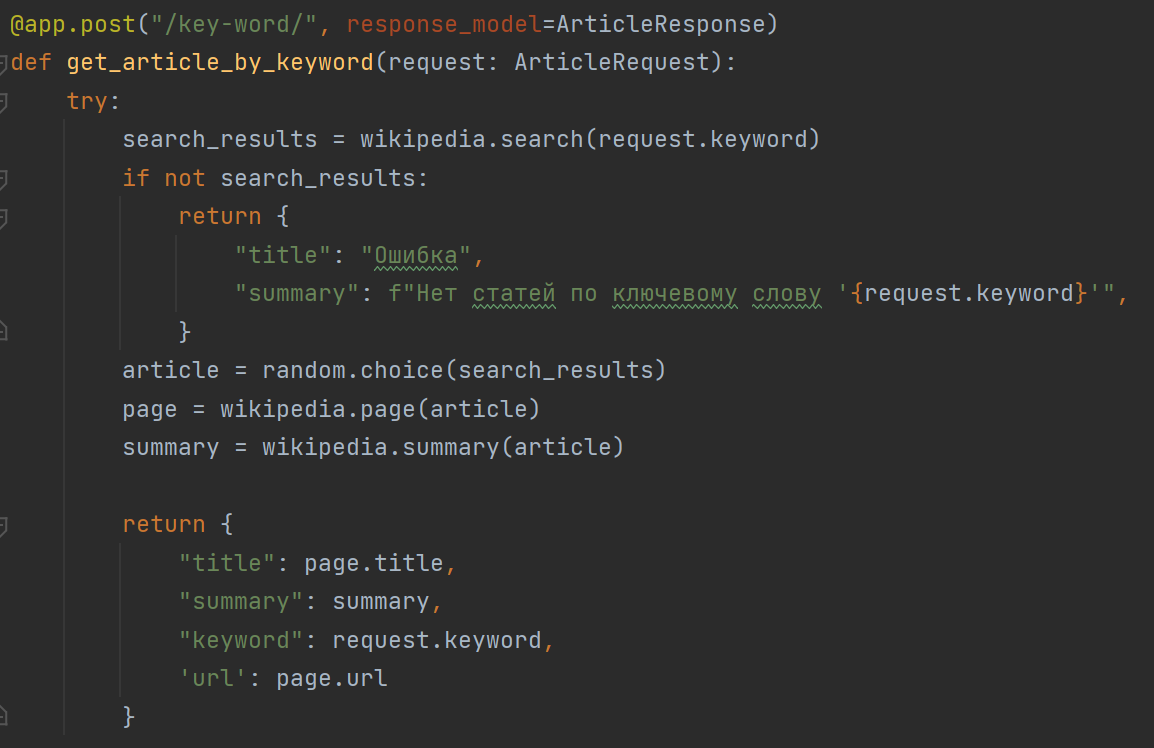
1. Напишем get-запрос с querry-параметром.

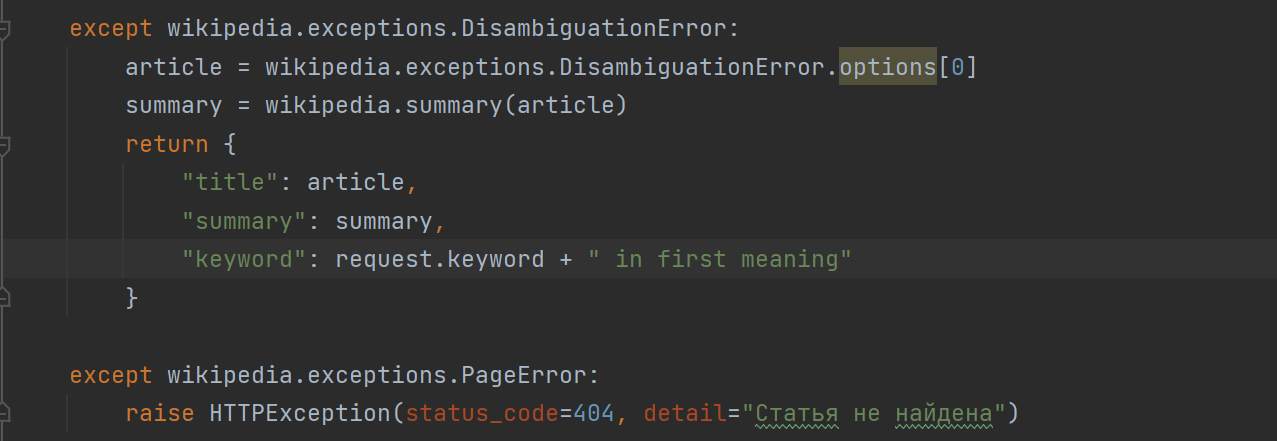


Пример запроса: http://127.0.0.1:8000/search/?query=Гарри Поттер&limit=3

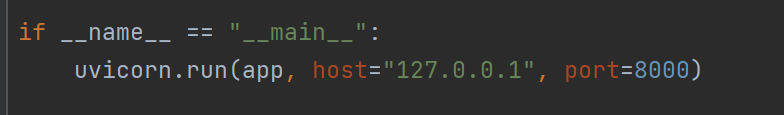


1. Напишем Post-запрос. Пусть он выдает рандомную статью по ключевому слову. Также обернем это в конструкцию try-except и так же как в первом get обработаем ошибки. Но при этом добавим в try еще одну обработку – когда у нас пустой результат не будем поднимать ошибку, а просто вернем сообщение, что нет статей. Article выберем с помощью random

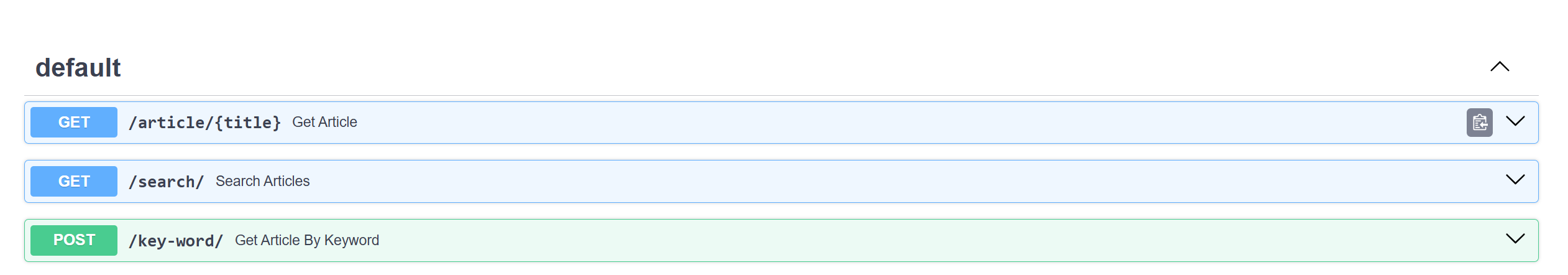


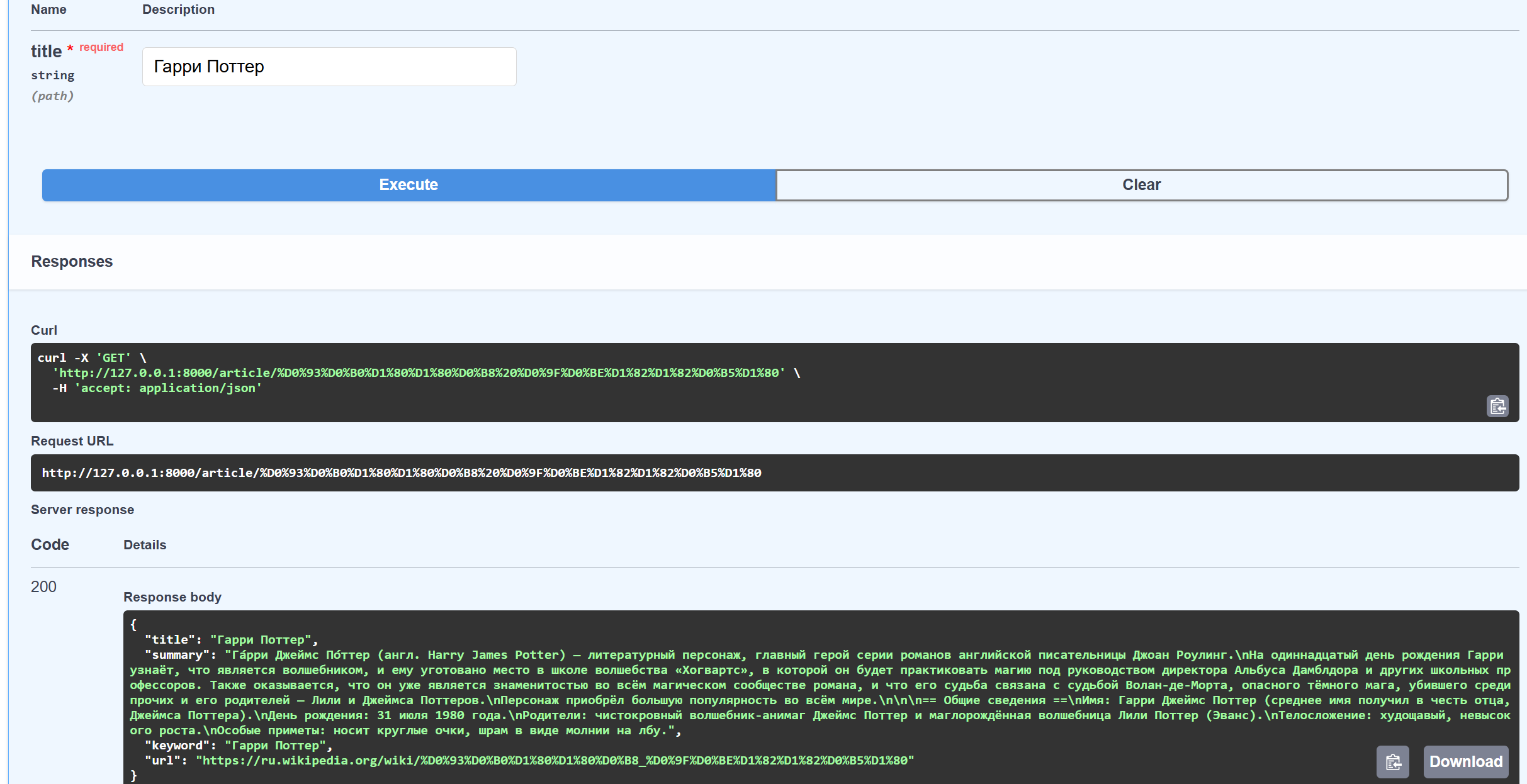


1. И напишем такую конструкцию, чтобы наш сервер запускался без помощи командной строки.

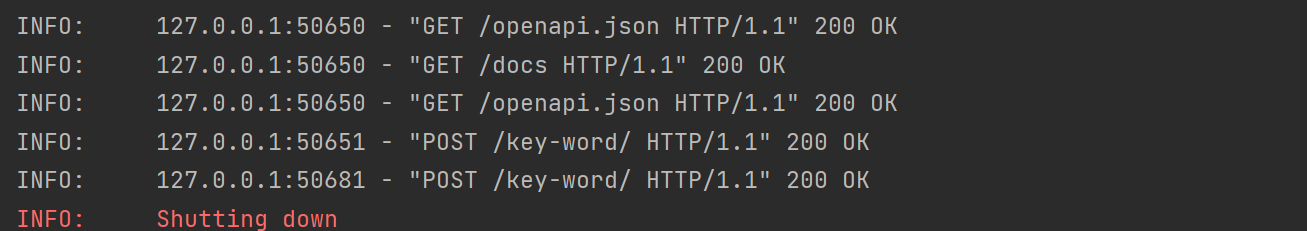


1. Запустим сервер, и откроем документацию (Swagger) и протестируем Get и Post запросы









Ответы сервера можно увидеть и в пайчарме.

1. **Вывод:** все поставленные цели были выполнены.