**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»

Лабораторная работа №15

Работа с FastAPI

Выполнила: Студентка группы

БВТ2402

Шилинцева Татьяна

Москва

2025

**Цель работы:** 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.

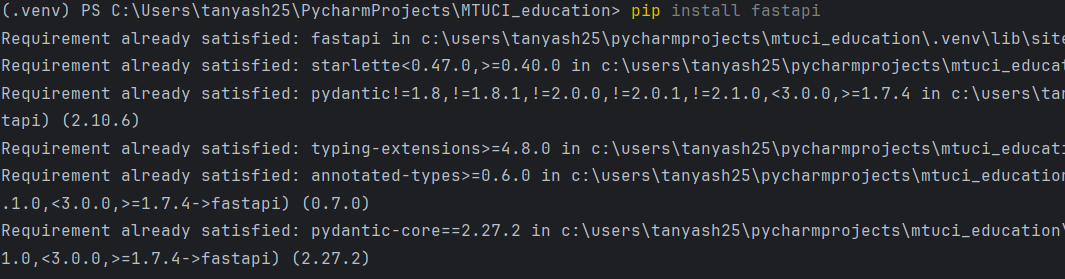
***Задание 1 Установка и подготовка***

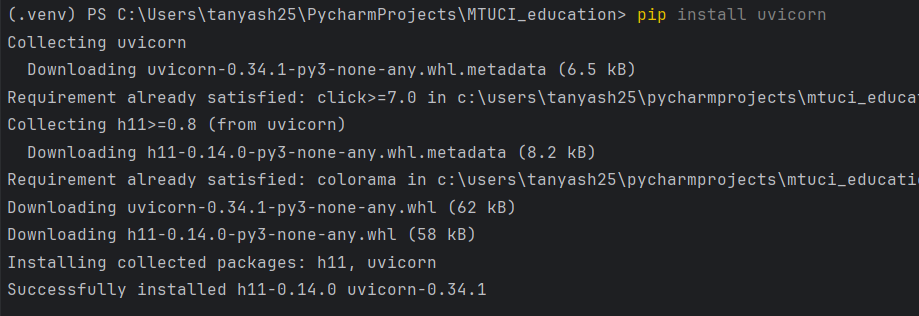
Инструкция:

* Ввести в консоль: pip install fastapi и pip install uvicorn.
* Создать файл main.py. Импортировать fastapi и pyjokes в main.py.
* Создать объект fastapi, куда далее будут подключаться роуты. Далее будет называться приложением fastapi.

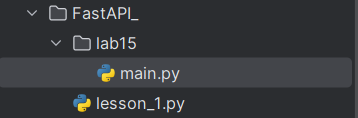
Выполнение:

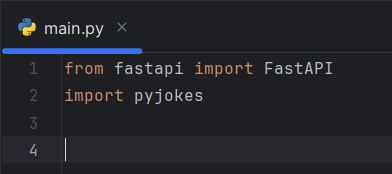
Шаг 1:



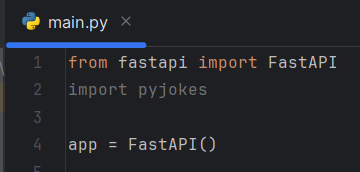


Шаг 2:





Шаг 3:



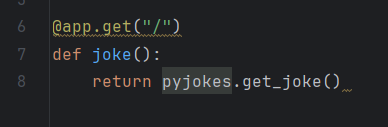
***Задание 2 Создание первого роута***

Инструкция:

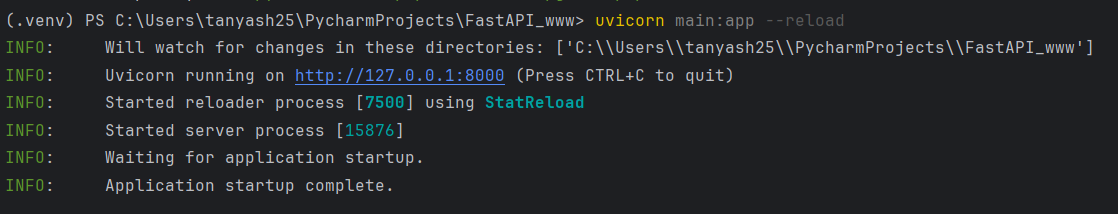
* Написать простую функцию, обернуть её специальным декоратором. Декоратор использует приложение созданное ранее, http метод и путь по которому будет работать данный роут.
* Запустить uvicorn – веб-сервер. Для этого нужно воспользоваться командой в консоли, запущенной в месте в одной директории с main.py. Правильный ввод: **uvicorn main:app --reload**. Зафиксировать результат запуска uvicorn.
* Перейти по базовому адресу, который указывается при запуске uvicorn - <http://127.0.0.1:8000>. Ознакомиться с результатом.
* Для удобной работы с нашим приложением следует использовать swagger, который открывается по ссылке <http://127.0.0.1:8000/docs>. На странице swagger отображаются все добавленные роуты.
* Развернуть роут и попробовать выполнить его. Для этого сначала нажать на кнопку “Try it out” и далее execute.

Выполнение:

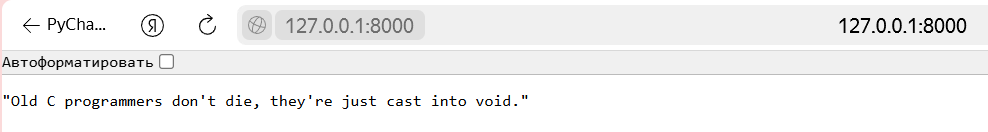
Шаг 1:



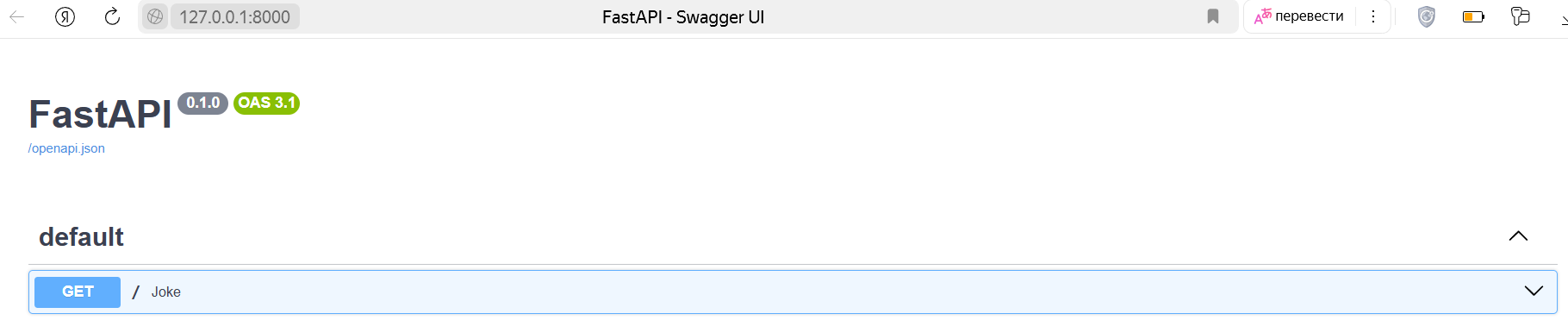
Шаг 2:



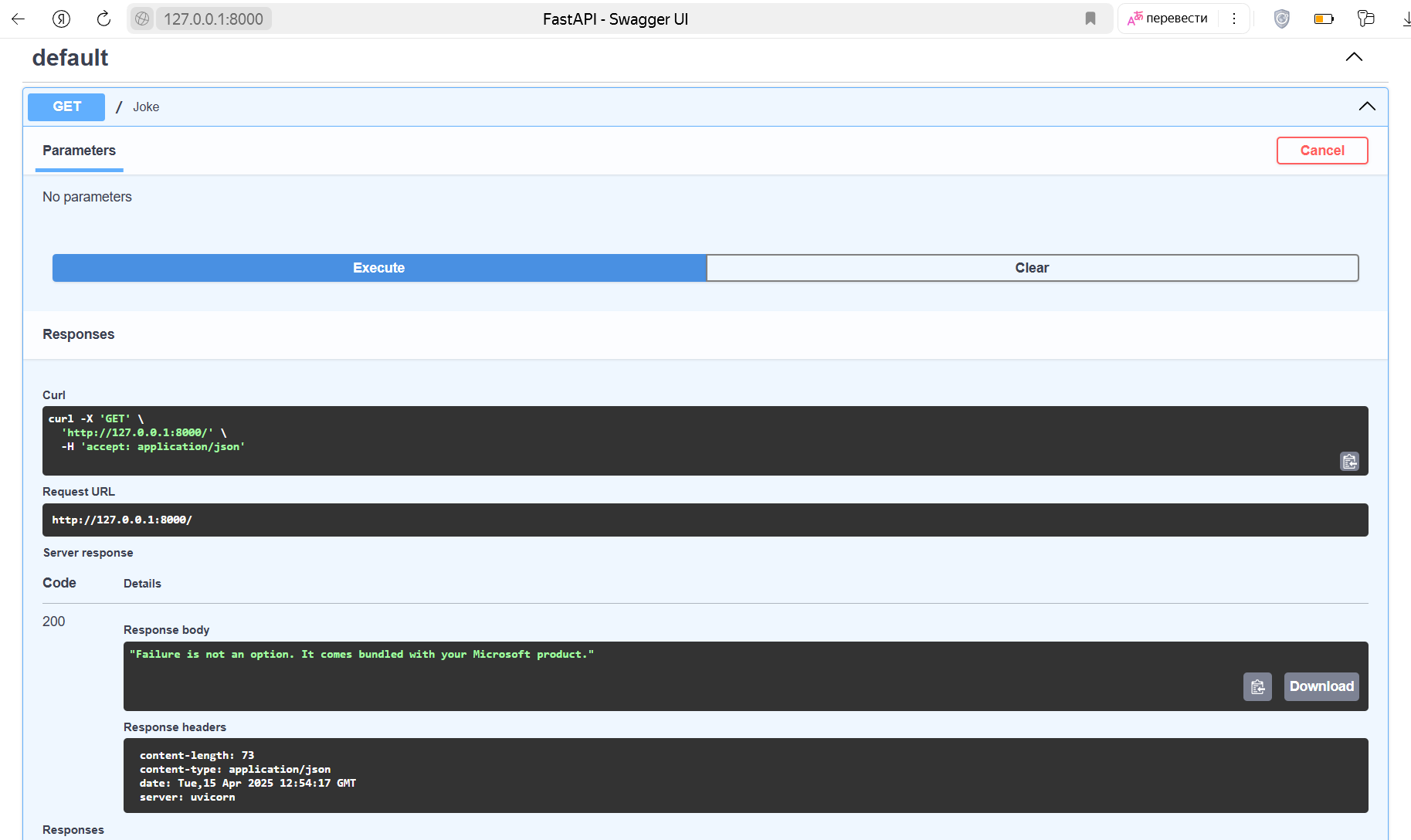
Шаг 3:

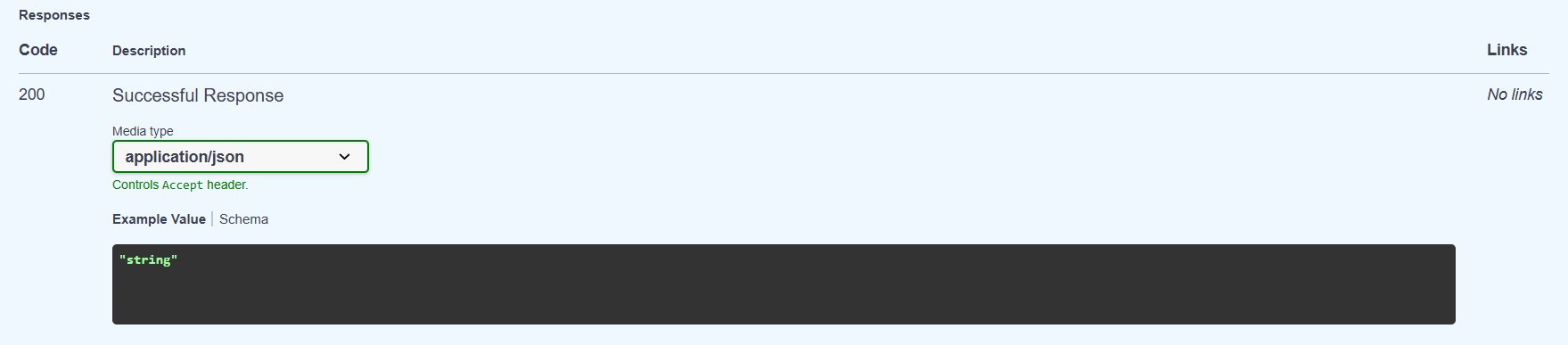


Шаг 4:



Шаг 5:





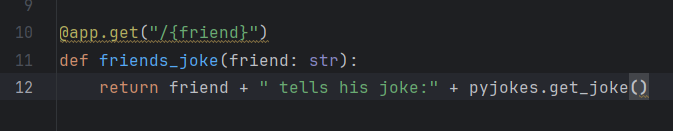
***Задание 3 Создание нового роута***

Инструкция:

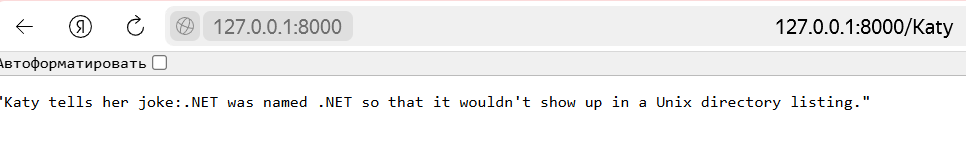
* Добавить ещё один роут. В этот раз в параметре декоратора указать параметр friend в фигурных скобках после слеша, чтобы можно было представить шутку от какого-то конкретного человека. Его же нужно добавить в параметры функции.
* Добавить к базовому пути [http://127.0.0.1:8000/](http://127.0.0.1:8000/John) через слеш желаемое значение параметра name. Зафиксировать результат. В данном случае параметр name = Katy.
* Посмотреть, как выглядит новый роут в swagger. Сделать скрин.
* Провести запуск в swagger нового роута с введёнными параметрами.

Выполнение:

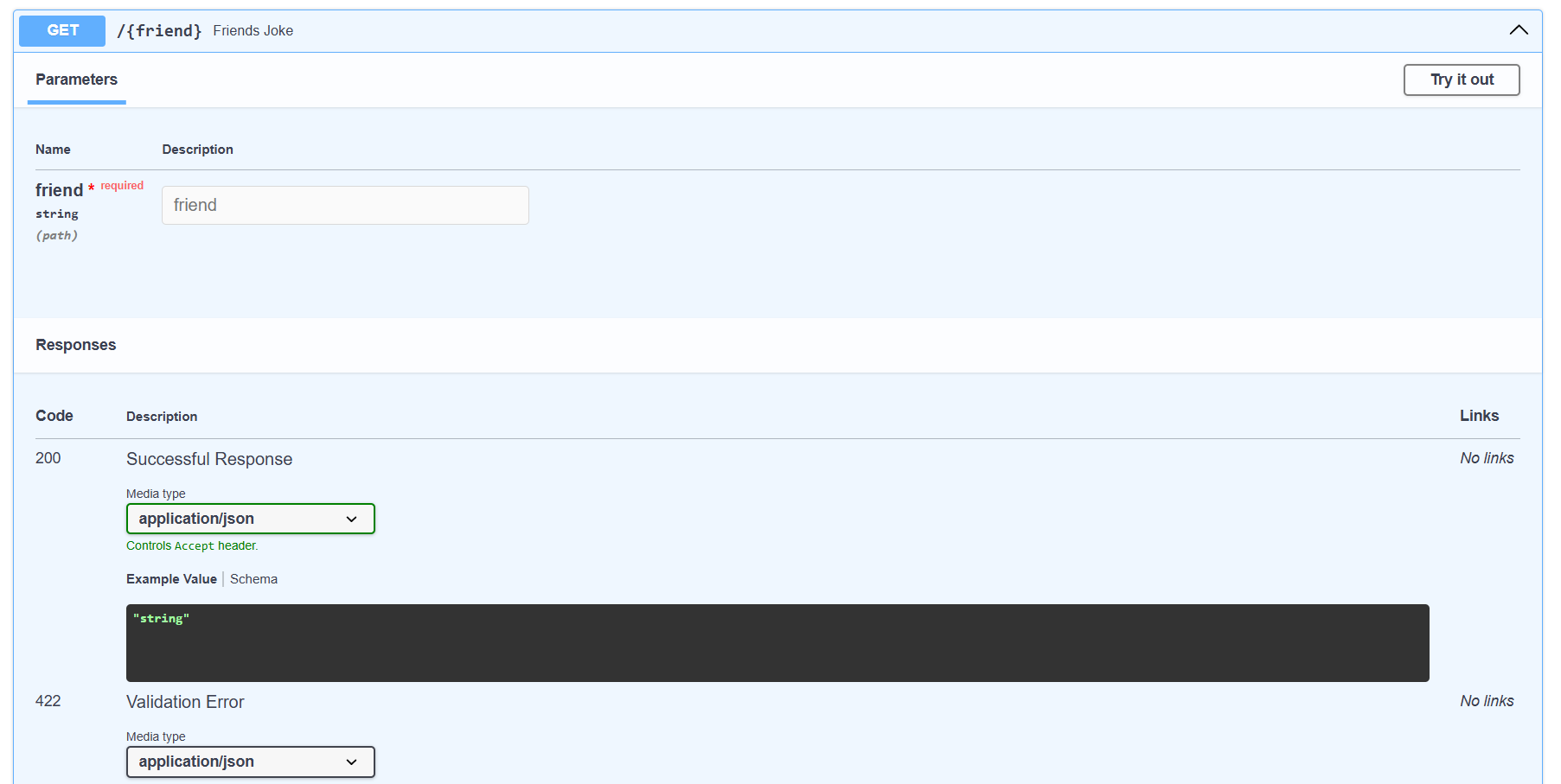
Шаг 1:



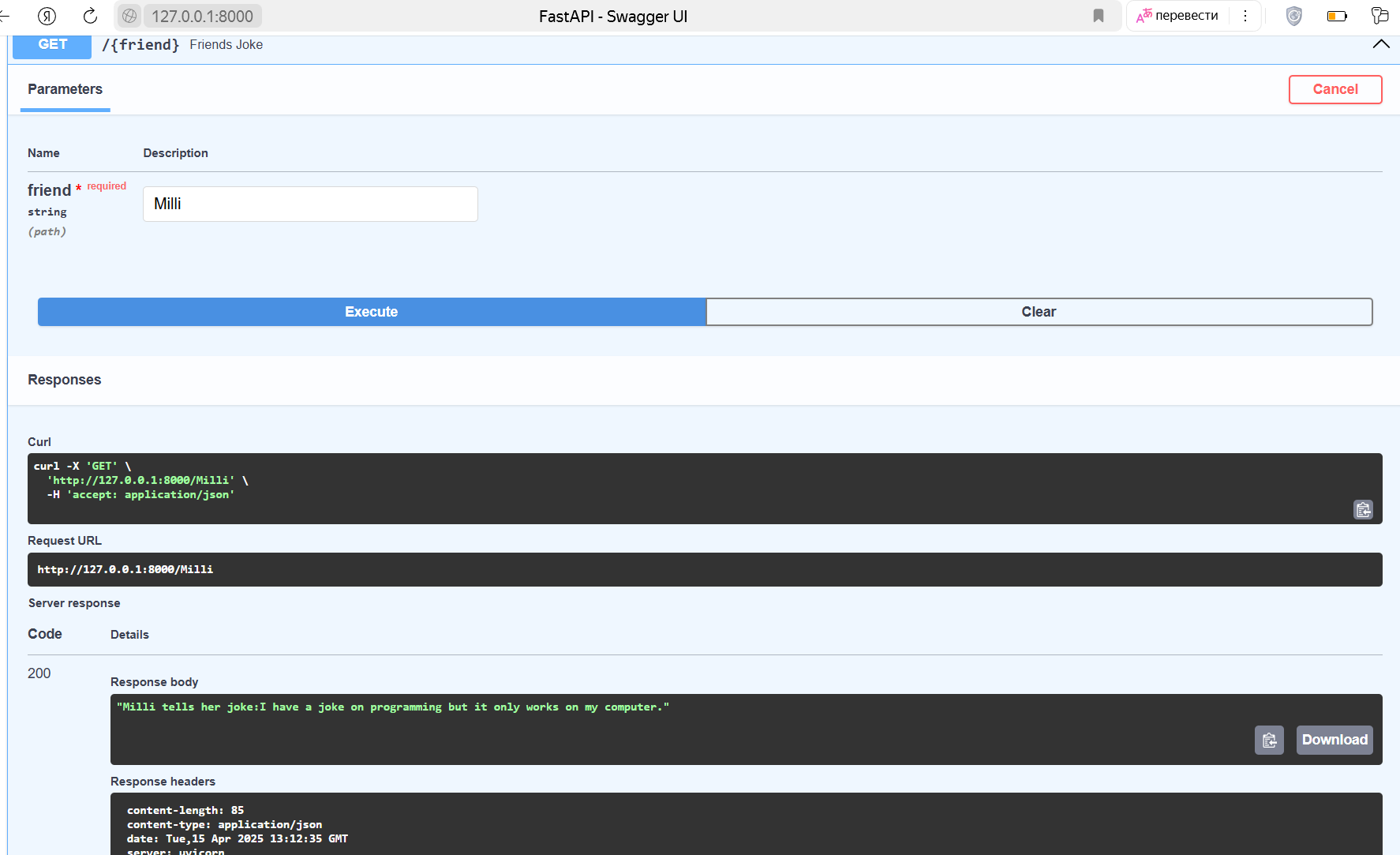
Шаг 2:



Шаг 3:



Шаг 4:



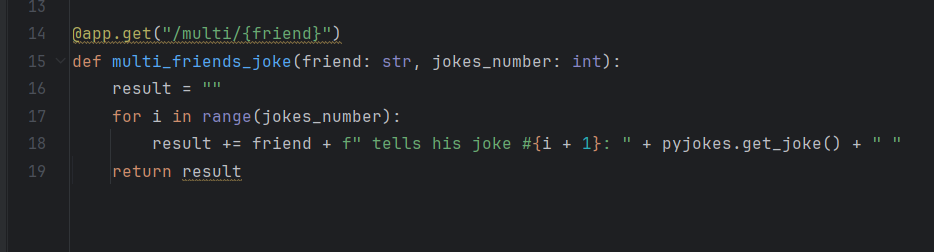
***Задание 4 Создание роута с возможностью выбора количества шуток***

Инструкция:

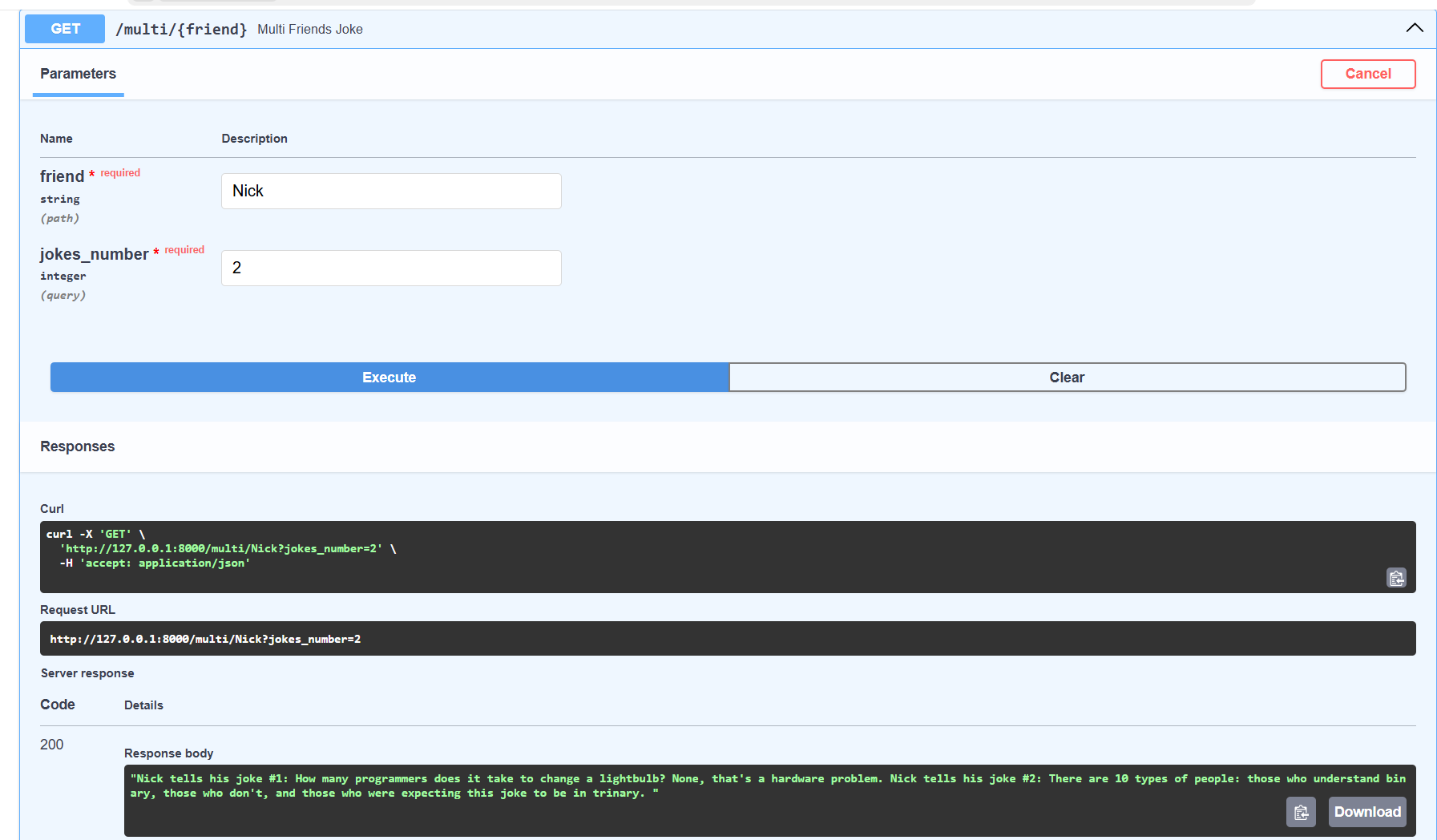
* Создать новый роут и добавить в него новый параметр jokes\_number, который не будет указан в пути. Добавить в параметр декоратора /multi перед существующим слешем.
* Запустить новый роут в swagger.

Выполнение:

Шаг 1:



Шаг 2:



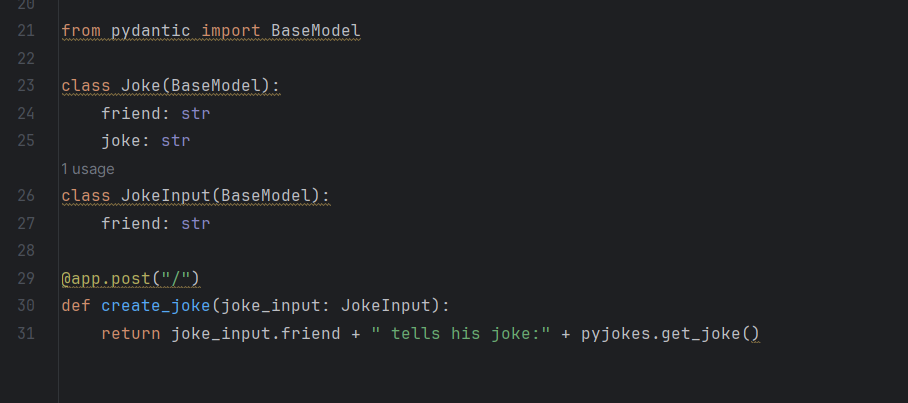
***Задание 5 Другой способ передачи информации в роут***

Инструкция:

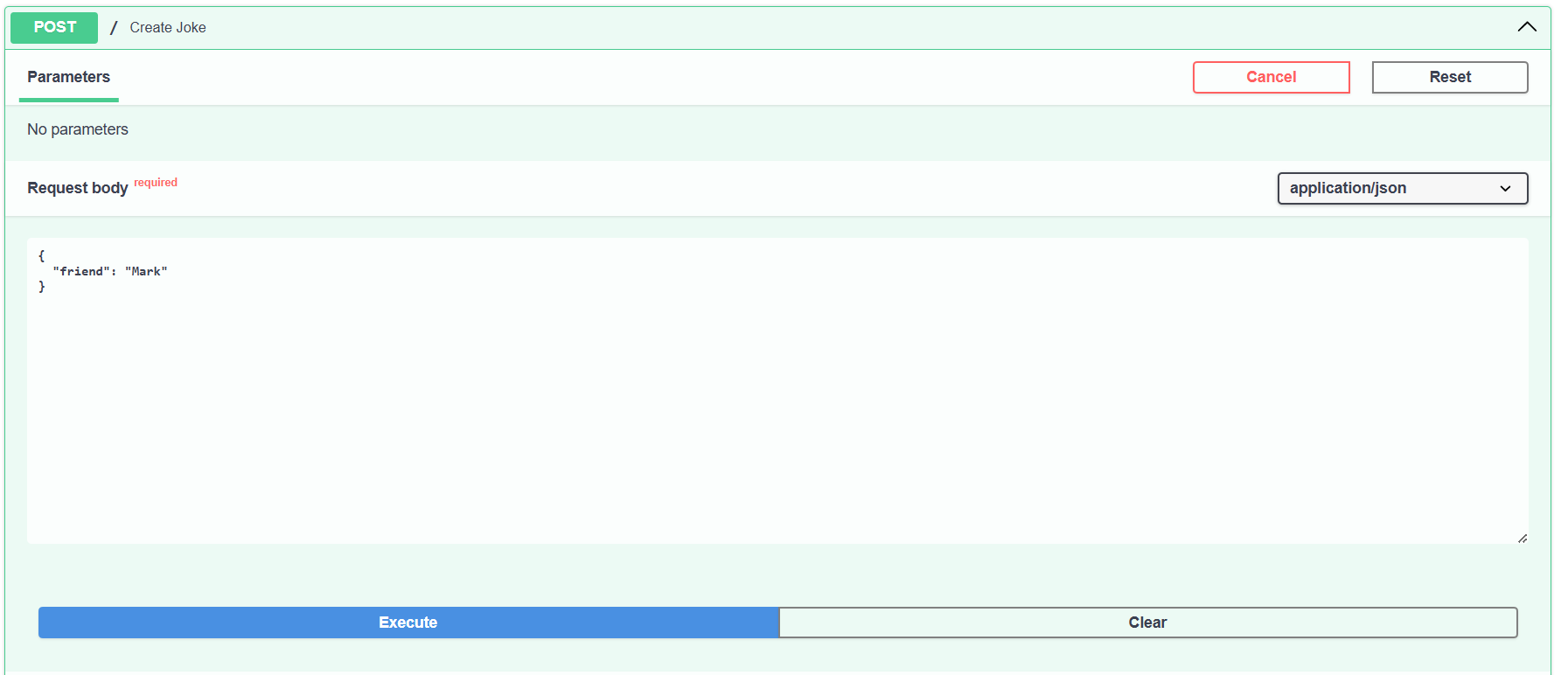
* Импортировать из библиотеки pydantic класс BaseModel. Создать новый роут, используя метод POST, и схему тела запроса, которую будет принимать роут для корректной работы. На основе BaseModel составить схему получения шутки, которая будет передана функции на вход.
* Открыть swagger и протестировать новый метод.
* Обновить ранее созданный POST метод, чтобы ответ выдавался в формате ранее описанным в схеме Joke.
* Посмотреть на изменения в swagger и протестировать обновленный роут.
* Добавить комментарий с описанием роута и поставить в параметрах response\_model, чтобы добавить валидацию ответа функции. Обновленный эндпоинт зафиксировать.

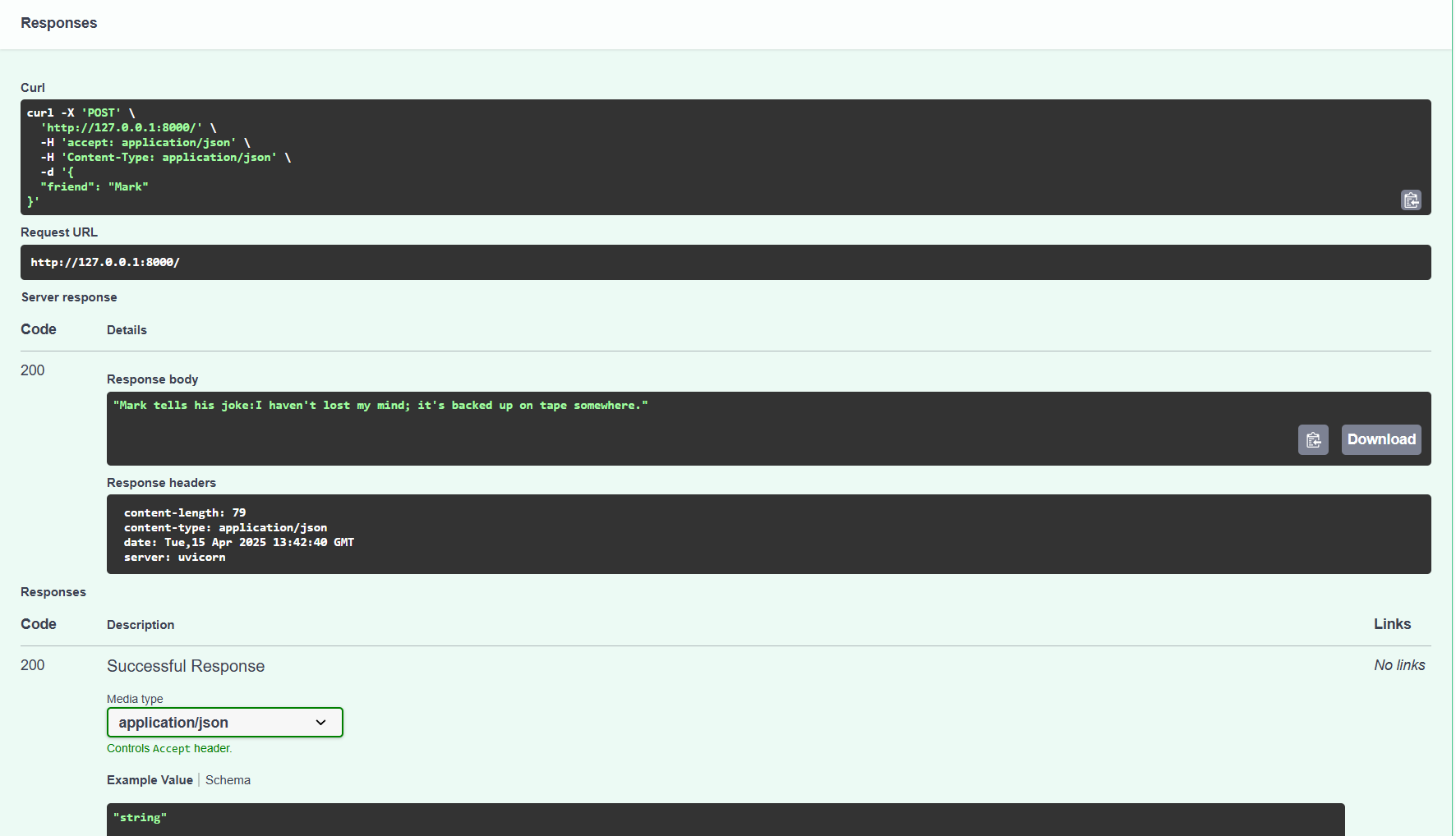
Выполнение:

Шаг 1:

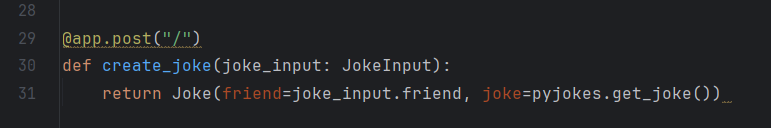


Шаг 2:

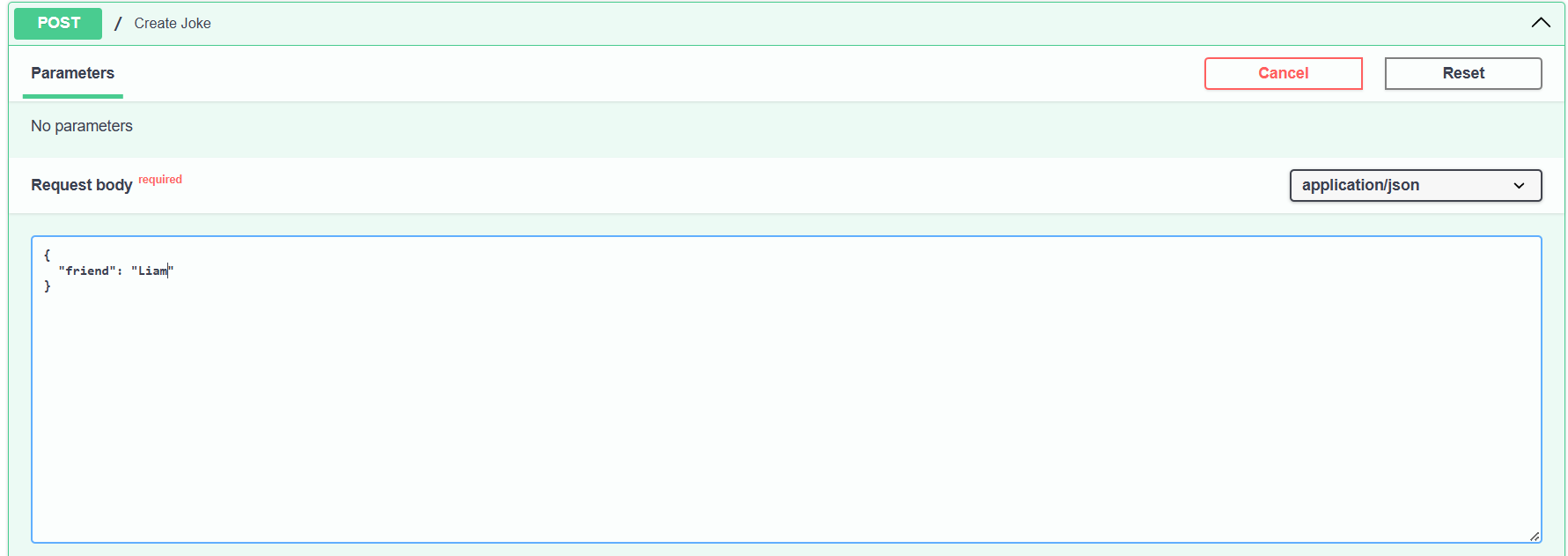




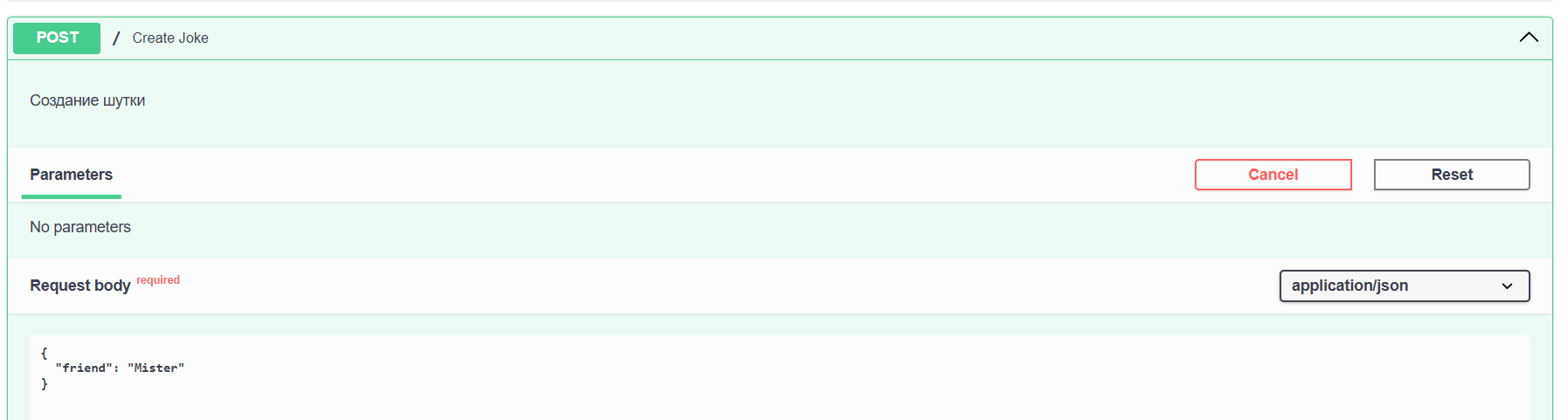
Шаг 3:



Шаг 4:



Шаг 5:





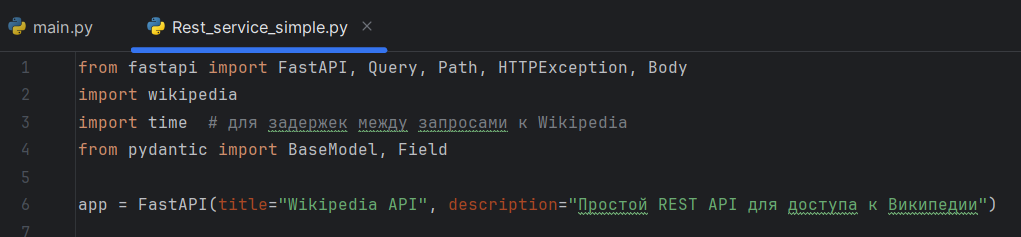
***Задание 6 Создать простой REST сервис с использованием библиотеки wikipedia.***

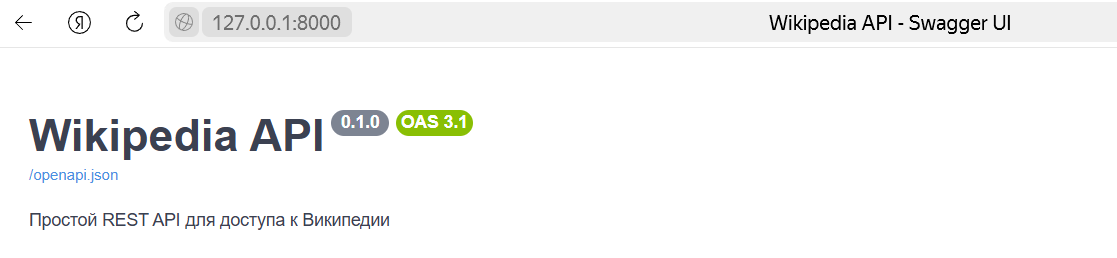
Инструкция:

* Импортировать нужные библиотеки. Задать настройки приложения FastAPI
* Создать 1 роут с параметром path.
* Создать 1 роут с параметром query.
* Создать 1 роут с передачей параметров в теле запроса.
* Провести тестирование.

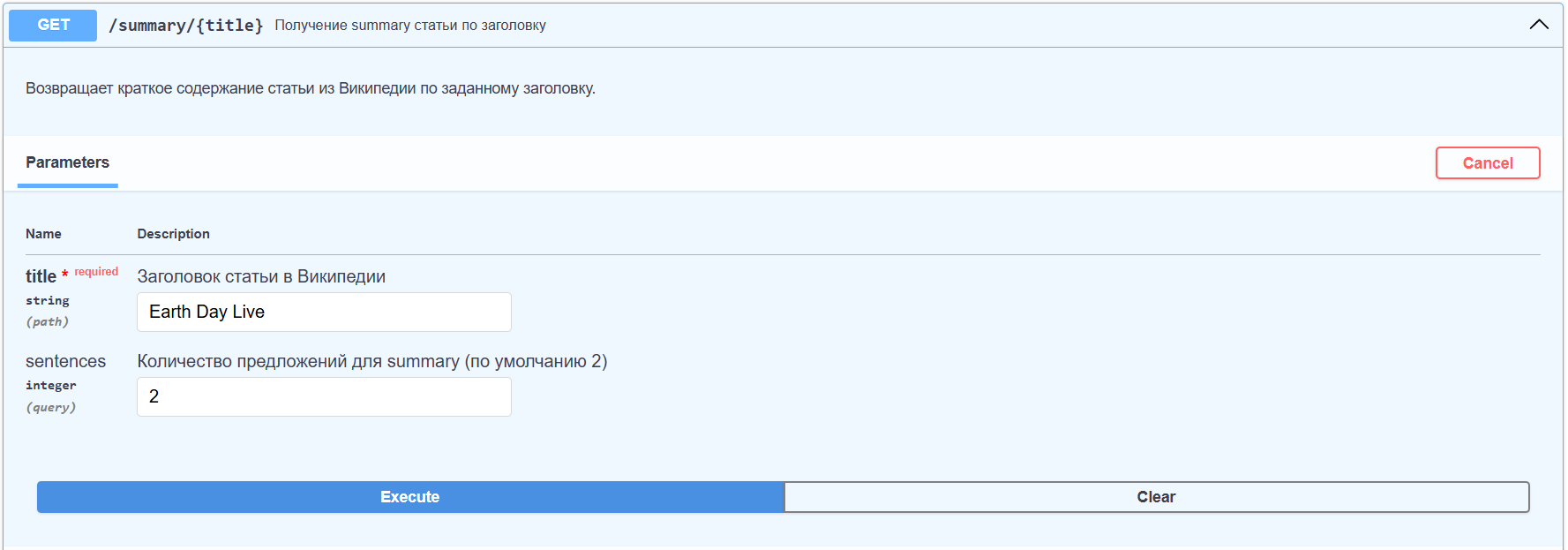
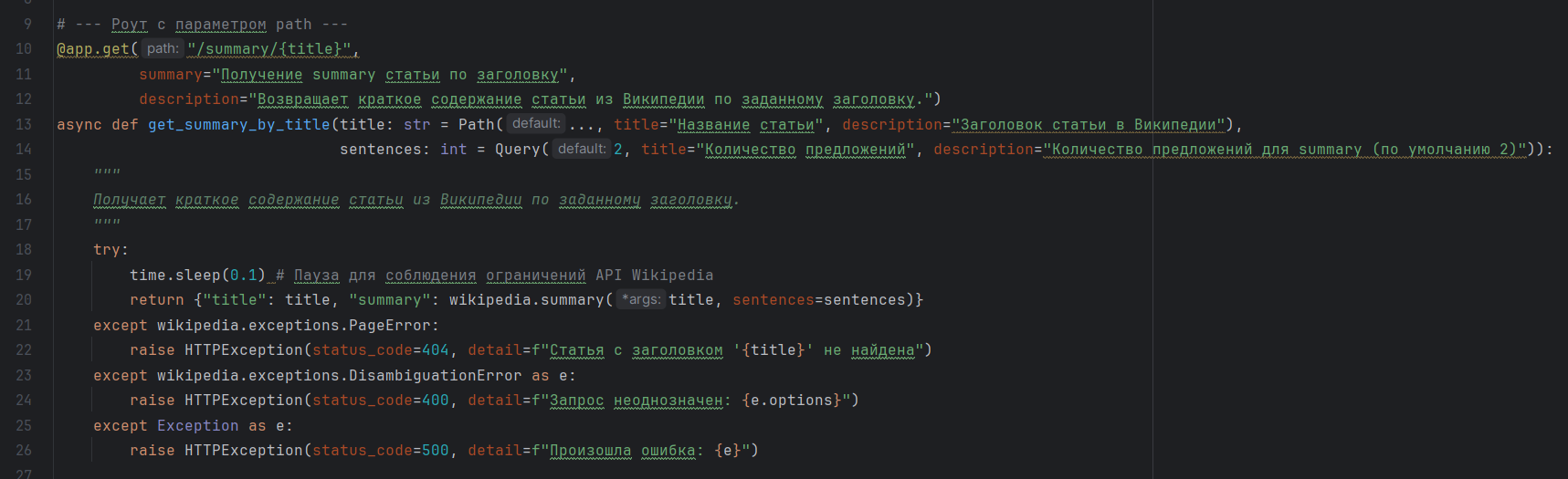
Выполнение:

Шаг 1:

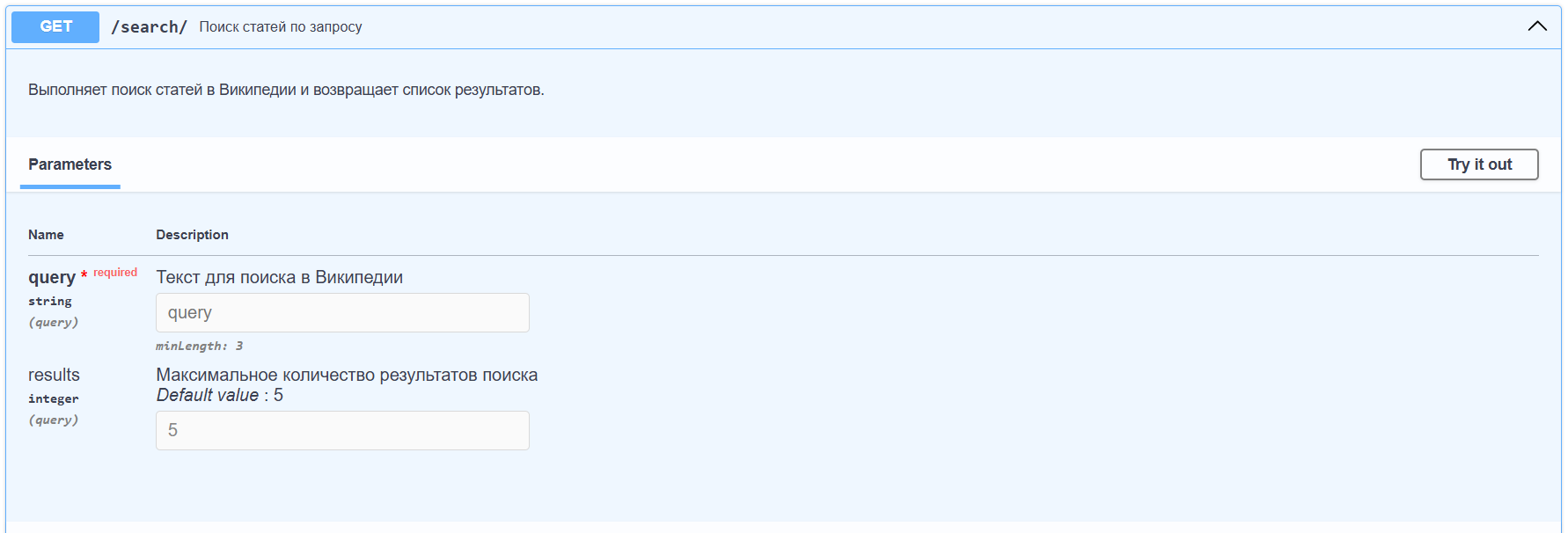
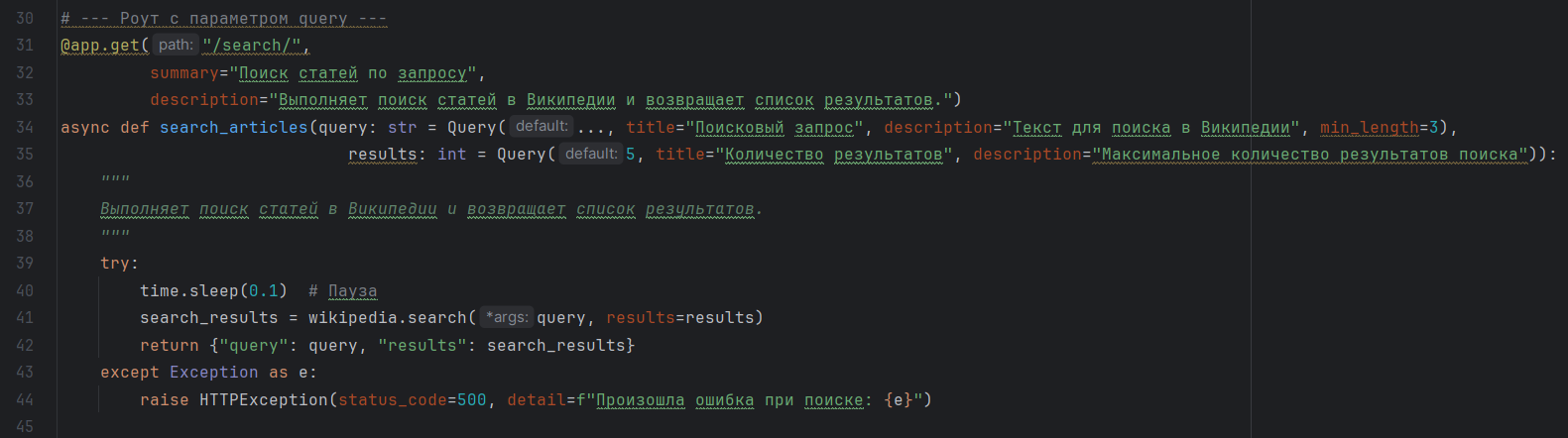




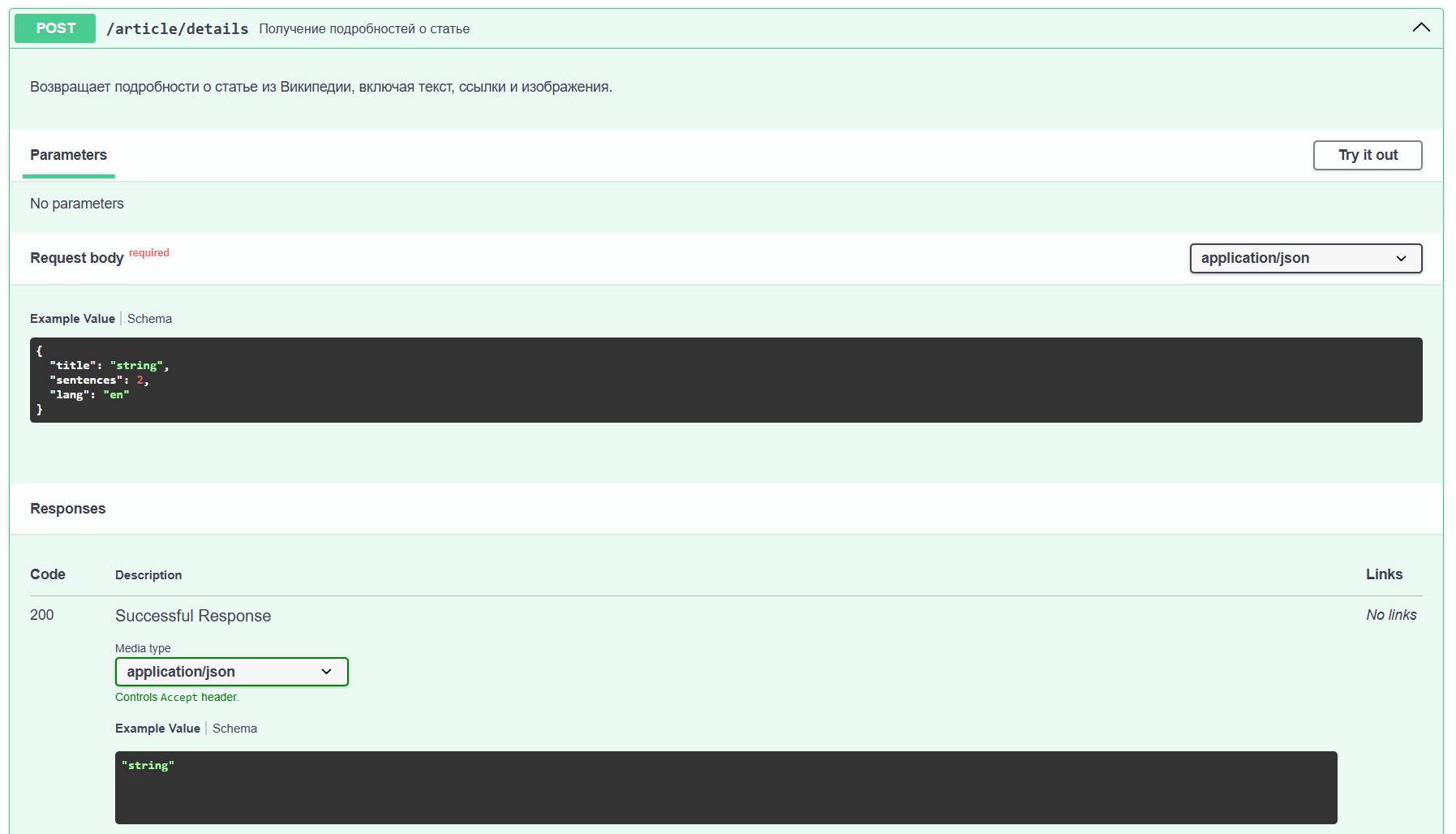
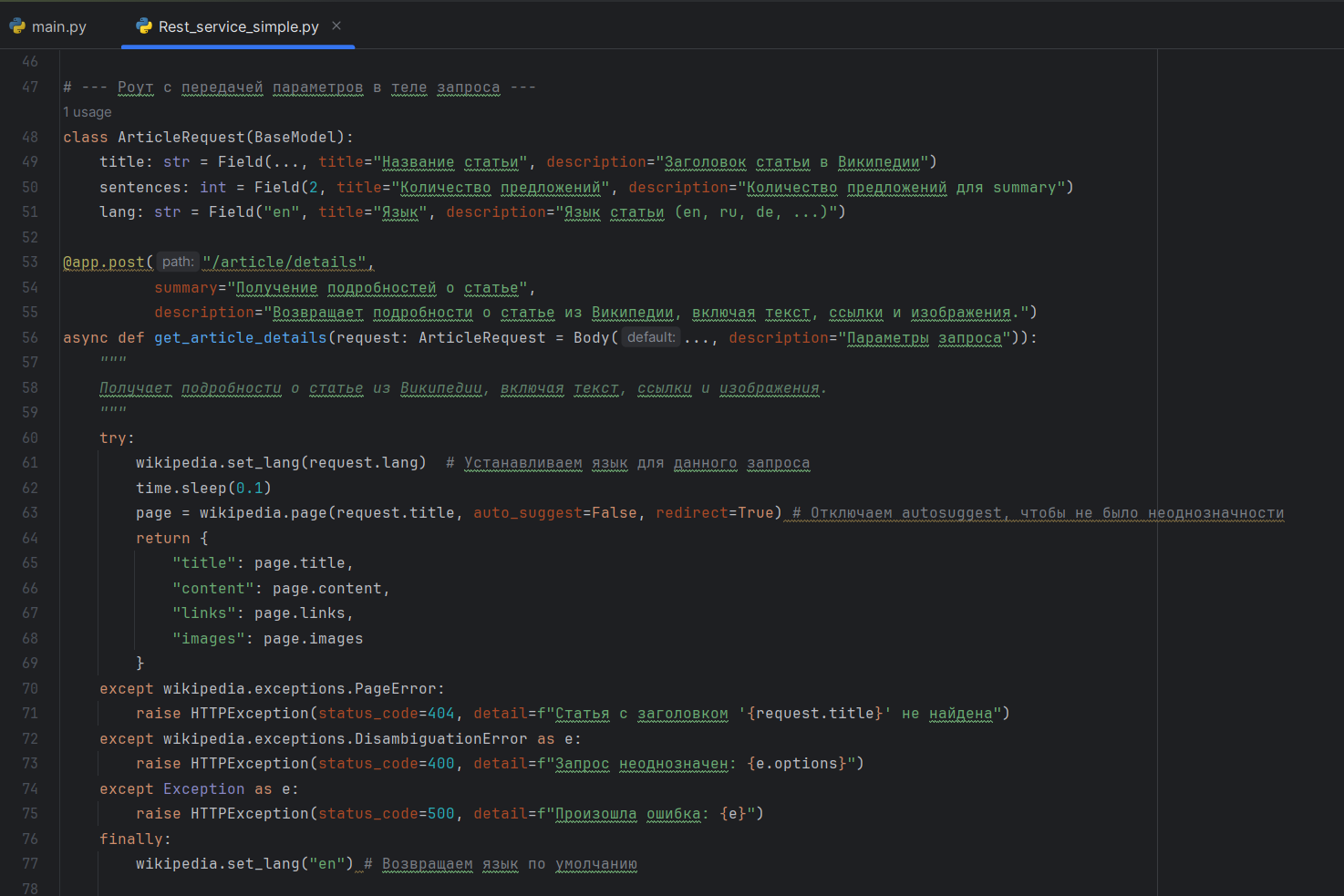
Шаг 2:



Шаг 3:

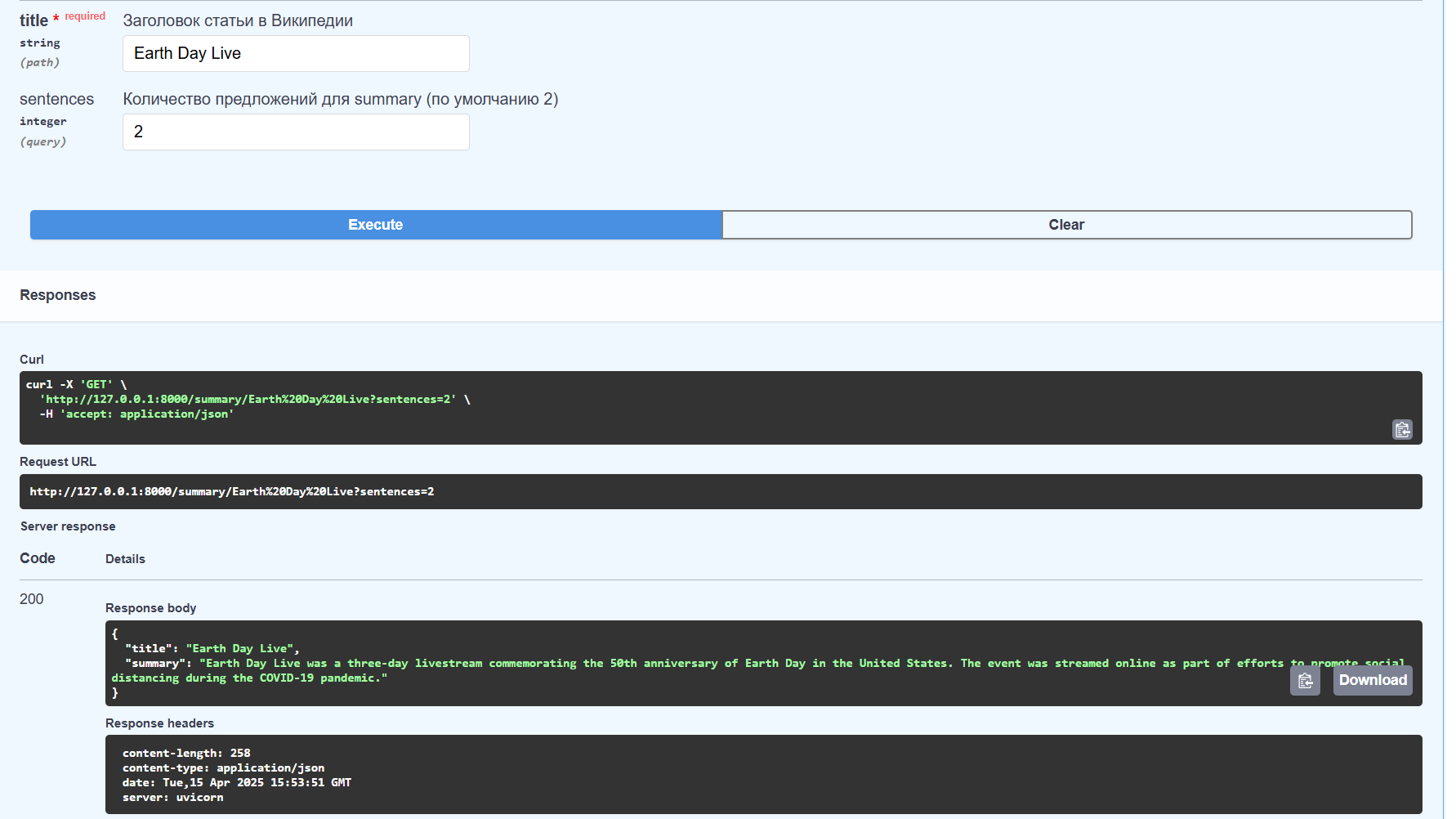


Шаг 4:



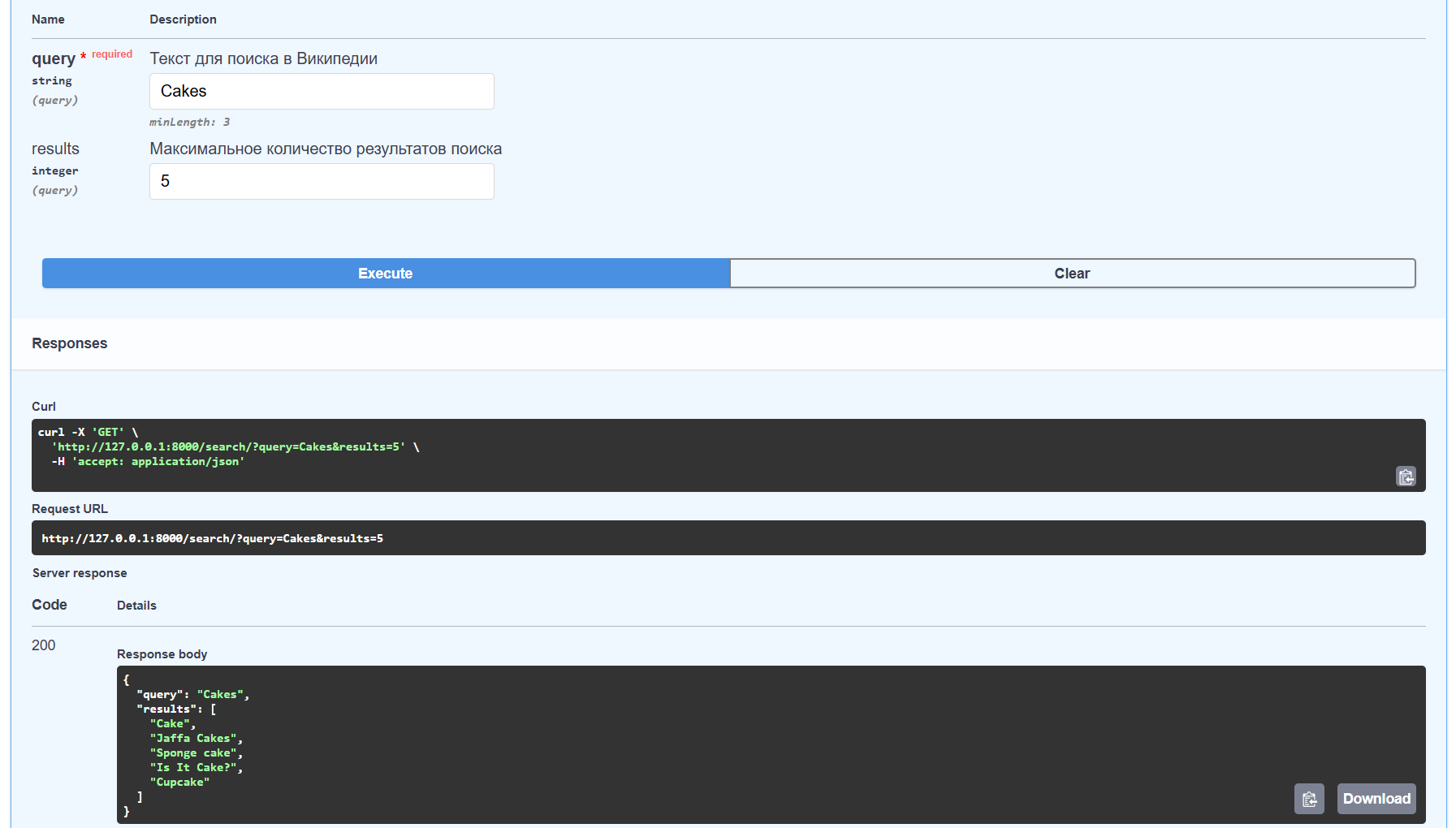
Шаг 5:

* Тестирование роута path.

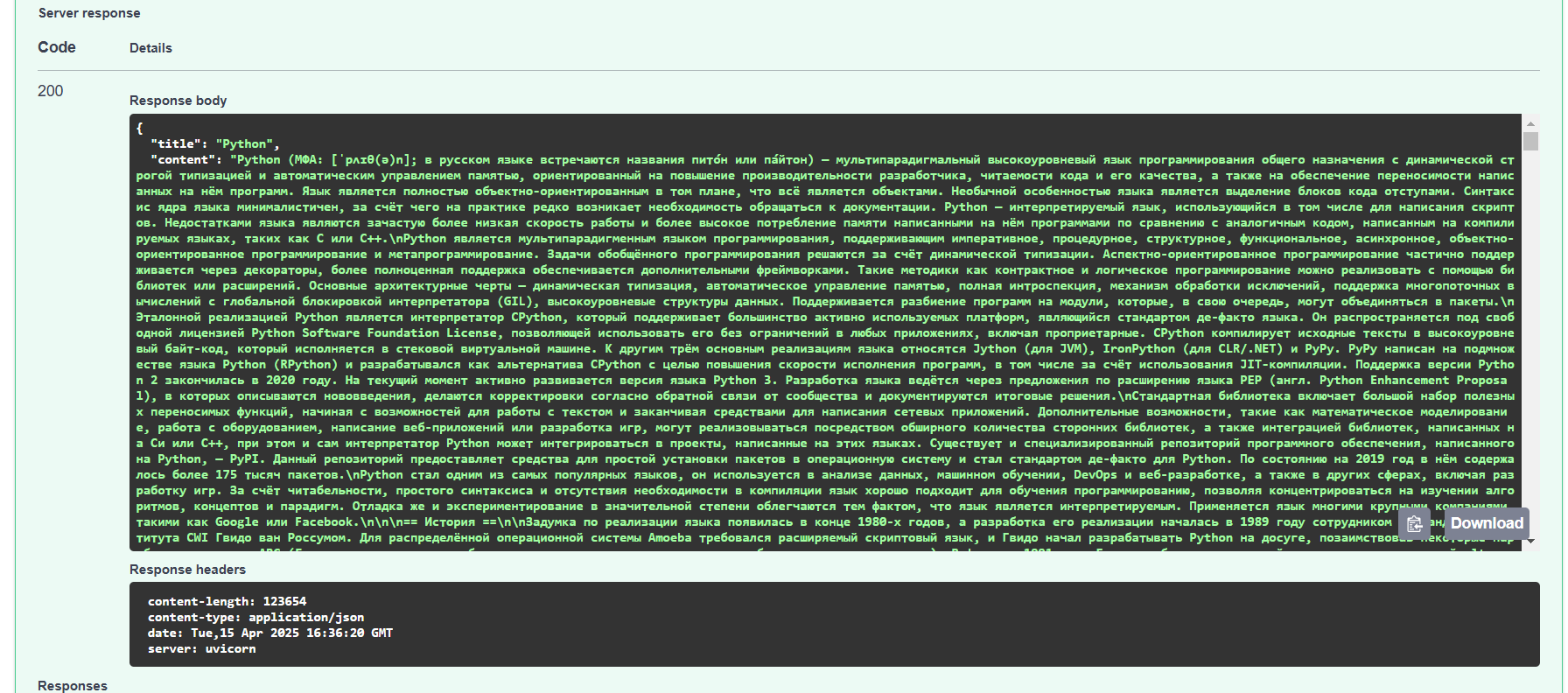
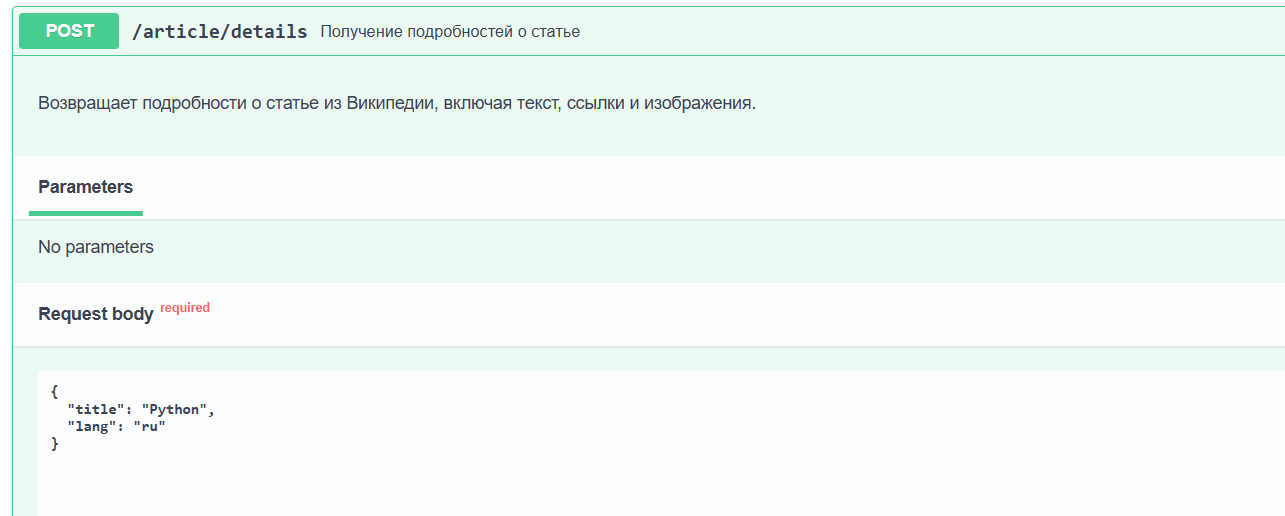


* Тестирование роута query.

Результат пользовательского вопроса:



* Тестирование роута с передачей параметров в теле запроса.



**Вывод**:

В ходе данной работы мы научились работать с технологией FastAPI и другими библиотеками для создания собственного сервера, написали свои роуты (правила для взаимодействия с сервером) и создали своё собственное клиент-серверное приложение для работы с сайтом Wikipedia по правилам архитектуры REST.

## Ответы на вопросы:

1. Что такое REST сервис?

**REST (Representational State Transfer) – это архитектурный стиль, который определяет набор ограничений, используемых для создания веб-сервисов.** Сервис, соответствующий этим ограничениям, называется RESTful. REST – это не протокол, а скорее архитектурный подход. Он не накладывает жестких правил о реализации, а предоставляет рекомендации и принципы.

**Ключевые характеристики и принципы REST:**

*Клиент-серверная архитектура:* Клиент и сервер разделены. Клиент отвечает за пользовательский интерфейс и взаимодействие с пользователем, а сервер – за хранение и обработку данных. Это позволяет независимо развивать и масштабировать клиентскую и серверную части.

*Отсутствие состояния (Stateless):* Сервер не хранит никакой информации о состоянии клиента между запросами. Каждый запрос от клиента должен содержать всю необходимую информацию для его обработки. Это упрощает масштабирование, так как любой сервер может обработать любой запрос.

*Кэширование (Cacheable):* Клиенты и промежуточные серверы могут кэшировать ответы от сервера. Ответы сервера должны указывать, можно ли их кэшировать и как долго. Это улучшает производительность и снижает нагрузку на сервер.

*Единообразный интерфейс (Uniform Interface):* Это, пожалуй, самый важный принцип REST. Он определяет общий набор операций, которые клиенты могут использовать для взаимодействия с ресурсами на сервере. Единообразный интерфейс обеспечивает независимость клиентской и серверной части. Ключевые аспекты единообразного интерфейса:

*Идентификация ресурсов (Resource Identification):*Каждый ресурс должен быть однозначно идентифицирован URI (Uniform Resource Identifier). Например, /users/123 идентифицирует пользователя с ID 123.

*Манипулирование ресурсами через представления (Resource Manipulation through Representations):*Клиенты взаимодействуют с ресурсами, обмениваясь представлениями этих ресурсов (например, в формате JSON или XML). Клиент получает представление ресурса, изменяет его и отправляет обратно на сервер.

*Самоописательные сообщения (Self-descriptive Messages):*Каждое сообщение должно содержать достаточно информации, чтобы клиент мог понять его. Это включает в себя использование правильных HTTP-методов, заголовков Content-Type и т. д.

*HATEOAS (Hypermedia as the Engine of Application State):* Клиент начинает с начальной точки (URI) и динамически обнаруживает другие доступные ресурсы и операции, используя гипермедийные ссылки, содержащиеся в ответах сервера. Это позволяет серверу развиваться независимо от клиента, так как клиент не захардкоживает URL-адреса. HATEOAS является самым сложным и редко реализуемым принципом REST.

*Многоуровневая система (Layered System):* Клиент не обязательно должен знать, подключен он напрямую к конечному серверу или через промежуточные серверы (например, прокси-серверы, балансировщики нагрузки). Это упрощает масштабирование и улучшает безопасность.

*Код по требованию (Code-On-Demand):* (Необязательный) Сервер может предоставлять клиенту исполняемый код (например, Java-апплеты или JavaScript), который клиент может выполнить. Этот принцип используется редко.

1. Что умеет библиотека wikipedia и как с ней работать?

Библиотека wikipedia предоставляет простой и удобный интерфейс для доступа к контенту Википедии, что делает ее отличным инструментом для образовательных целей, исследовательских проектов и создания приложений, связанных с Википедией.

**Основные функции:**

Поиск статей: Поиск статей по ключевым словам.

Получение информации о статье: Получение заголовка, текста, разделов, ссылок, изображений и других метаданных статьи.

Получение ссылок на статьи: Получение ссылок на статьи из статьи.

Получение ссылок на изображения: Получение ссылок на изображения из статьи.

Получение ссылок на категории: Получение списка категорий, к которым относится статья.

Получение цитат из статьи: Получение цитат из статьи (если они есть).

Получение случайной статьи: Получение случайной статьи.

Изменение языка: Работа с разными языковыми версиями Википедии.

Работа с разрешениями: Получение информации о разрешениях на использование контента Википедии.

Работа с API Википедии: Выполнение более сложных операций, используя непосредственно API Википедии.

1. Что такое роут и какие у него могут быть параметры? За что отвечают эти параметры?

**Роут (Route)** в контексте веб-разработки, особенно при создании веб-приложений и API, — это определение соответствия между URL-адресом (путем в вебе) и определенным обработчиком (функцией, методом), который должен быть выполнен при запросе этого URL. Другими словами, **это инструкция для сервера, определяющая, что делать, когда пользователь или другое приложение обращается к конкретному URL.**

**Параметры роута:**

Параметры роута позволяют делать роуты более гибкими и динамичными, позволяя передавать значения из URL в обработчик. Они могут быть двух основных типов:

**Параметры пути (Path Parameters):**

Определяются непосредственно в URL-шаблоне.

Используются для идентификации конкретных ресурсов или разделов внутри ресурса.

Обычно используются для указания ID сущности (например, пользователя, товара, записи в блоге) или для указания категории.

Примеры:

/users/{user\_id}: user\_id — параметр пути, который указывает ID конкретного пользователя. Например, /users/123 обратится к пользователю с ID 123.

/products/{category}/{product\_id}: category и product\_id — параметры пути, указывающие категорию и ID конкретного товара. Например, /products/electronics/456 обратится к товару с ID 456 в категории “electronics”.

За что отвечают эти параметры:

user\_id, product\_id: Идентифицируют конкретный ресурс (пользователя, товар).

category: Позволяет фильтровать или сегментировать ресурсы (например, отображать товары только определенной категории).

**Параметры запроса (Query Parameters):**

Передаются после вопросительного знака (?) в URL.

Состоят из пар “ключ=значение”, разделенных амперсандом (&).

Используются для фильтрации, сортировки, пагинации и других дополнительных опций запроса.

Примеры:

/users?page=2&limit=10: page и limit — параметры запроса, указывающие номер страницы и количество элементов на странице.

/products?category=electronics&sort=price&order=desc: category, sort и order — параметры запроса, указывающие категорию, поле для сортировки и порядок сортировки.

За что отвечают эти параметры:

page, limit: Управляют пагинацией (разбиением результатов на страницы).

sort, order: Управляют порядком сортировки результатов.

category: Фильтруют результаты по определенной категории.

Другие параметры: Могут использоваться для указания диапазона дат, выбора конкретных полей для возврата, выполнения поиска и т.д.

Сводная таблица:

| Параметр | Тип | Расположение в URL | Назначение | Пример |
| --- | --- | --- | --- | --- |
| user\_id | Пути | /users/{user\_id} | Идентификация конкретного пользователя. | /users/123 |
| category | Пути | /products/{category} | Идентификация категории продуктов. | /products/electronics |
| page | Запроса | /users?page=2 | Указание номера страницы для пагинации. | /users?page=2 |
| limit | Запроса | /users?limit=10 | Указание количества элементов на странице. | /users?limit=10 |
| sort | Запроса | /products?sort=price | Указание поля для сортировки продуктов. | /products?sort=price |
| order | Запроса | /products?order=desc | Указание порядка сортировки (asc — по возрастанию, desc — по убыванию). | /products?order=desc |
| search | Запроса | /products?search=keyword | Поиск продуктов, содержащих определенное ключевое слово. | /products?search=keyword |
| from\_date, to\_date | Запроса | /reports?from\_date=2023-01-01&to\_date=2023-01-31 | Указание диапазона дат для отчетов. | /reports?from\_date=2023-01-01&to\_date=2023-01-31 |

1. Что значит возвращение запросов и валидация запросов? В чём разница между этими понятиями?

**Возвращение запросов (Returning Responses):**

Возвращение запросов — это процесс, при котором веб-сервер или API после получения и обработки запроса от клиента отправляет ответ (response) обратно клиенту. Ответ содержит данные или информацию, которую клиент запрашивал, или сообщение об ошибке, если что-то пошло не так.

Что включает в себя возвращение запроса:

* HTTP-код состояния (Status Code): Это трехзначное число, указывающее на результат обработки запроса. Примеры:

200 OK: Запрос успешно обработан.

201 Created: Ресурс успешно создан (обычно в ответ на POST-запрос).

400 Bad Request: Некорректный запрос от клиента (например, неверный формат данных).

401 Unauthorized: Требуется аутентификация.

403 Forbidden: Доступ запрещен (даже после аутентификации).

404 Not Found: Ресурс не найден.

500 Internal Server Error: Ошибка на стороне сервера.

* HTTP-заголовки (Headers): Содержат дополнительную информацию о запросе и ответе, такую как:

Content-Type: Указывает тип данных, содержащихся в теле ответа (например, application/json, text/xml).

Content-Length: Указывает размер тела ответа в байтах.

Cache-Control: Управляет кэшированием ответа.

Authorization: Содержит информацию для аутентификации.

* Тело ответа (Response Body): Содержит данные, которые клиент запрашивал. Формат данных определяется заголовком Content-Type (обычно JSON или XML). В теле ответа может быть:

Представление ресурса (например, информация о пользователе в формате JSON).

Список ресурсов.

Сообщение об успехе или ошибке.

**Валидация запросов (Request Validation):**

Валидация запросов — это процесс проверки входящего запроса от клиента на соответствие ожидаемым требованиям и ограничениям. Цель валидации — убедиться, что запрос содержит корректные данные в правильном формате и что у клиента есть необходимые права для выполнения запрашиваемой операции.

Что включает в себя валидация запроса:

Проверка формата данных: Убедиться, что данные имеют правильный тип (например, число, строка, дата) и формат (например, email-адрес).

Проверка наличия обязательных полей: Убедиться, что все обязательные поля присутствуют в запросе.

Проверка диапазона значений: Убедиться, что значения полей находятся в допустимом диапазоне (например, возраст не может быть отрицательным).

Проверка соответствия схеме: Проверить, что запрос соответствует определенной схеме (например, с использованием JSON Schema).

Проверка аутентификации и авторизации: Убедиться, что клиент аутентифицирован (подтвердил свою личность) и авторизован (имеет права на выполнение данной операции).

Защита от вредоносных данных: Проверить, что данные не содержат вредоносный код (например, SQL-инъекции, XSS).

Ключевая разница:

* **Валидация запросов происходит перед обработкой запроса**, чтобы убедиться, что запрос вообще можно обработать. Если валидация не пройдена, запрос отклоняется с соответствующим кодом ошибки (например, 400 Bad Request).
* **Возвращение запросов происходит после обработки запрос**а, чтобы сообщить клиенту об успехе или неудаче и, в случае успеха, предоставить запрошенные данные.

1. Что такое FastAPI? Для чего его используют?

FastAPI — это мощный и современный фреймворк для создания API на Python, который выделяется своей производительностью, простотой использования, автоматической документацией и мощной валидацией данных. Это отличный выбор для разработки API в современных проектах.

1. Для чего была использована библиотека pyjokes? Что она из себя представляет?

Данная библиотека была использована для генерации случайных шуток программистов в качестве данных «воображаемых» пользователей.

Данная библиотека представляет из себя сборники шуток (собраны в списки строковых объектов) на разных языках. В главном файле представлен список основных функций и параметров к ним (язык, категория: нейтральная, с Чаком Норисом и объединение предыдущих двух категорий). В ней есть три основные функции: получить одну шутку, несколько шуток и запустить генератор с бесконечными шутками. Последняя функция не сработала у меня.

1. Что такое объект FastAPI?

В FastAPI, объект FastAPI — это основной класс, представляющий ваше приложение API. Это центральная точка, где вы конфигурируете и определяете все компоненты вашего API, такие как роуты, middleware, обработчики ошибок, зависимости и т.д. Объект FastAPI является экземпляром класса fastapi.FastAPI.

1. Как работает декоратор @app.get("/")?

**Декоратор @app.get("/") в FastAPI выполняет следующие действия:**

* Связывание пути и обработчика: Декоратор связывает указанный путь (в данном случае /) с функцией, которая следует за декоратором. Эта функция становится обработчиком (handler) для запросов, отправленных на этот путь. То есть, когда приходит HTTP GET запрос к корневому пути /, будет вызвана функция, которую “декорирует” @app.get("/").
* Определение HTTP-метода: @app.get указывает, что этот роут должен обрабатывать HTTP GET запросы. FastAPI поддерживает различные HTTP методы, такие как POST, PUT, DELETE, PATCH и другие, и для каждого из них есть соответствующие декораторы (@app.post, @app.put, @app.delete, @app.patch).
* Регистрация роута: Декоратор регистрирует информацию о роуте (путь, метод, обработчик) в приложении FastAPI (app). Эта информация используется FastAPI для маршрутизации входящих запросов к соответствующим обработчикам.
* Генерация документации: FastAPI использует информацию, полученную из декоратора, для автоматической генерации документации API (Swagger UI и ReDoc). Это включает в себя путь, метод, параметры и описание роута, а также структуру запроса и ответа.
* Валидация данных: FastAPI использует аннотации типов (type hints) в определении функции-обработчика для автоматической валидации данных, передаваемых в запросе.
* Преобразование данных: FastAPI автоматически преобразует данные, возвращаемые функцией-обработчиком, в JSON-формат для отправки в HTTP-ответе.

1. Что делает uvicorn ? Это библиотека или что?

**Что такое Uvicorn?**

Uvicorn — это ASGI (Asynchronous Server Gateway Interface) веб-сервер для Python. Его основная задача — принимать HTTP-запросы и перенаправлять их в асинхронное веб-приложение Python, а затем отправлять ответы обратно клиенту.

**Ключевые характеристики Uvicorn:**

ASGI-сервер: Uvicorn полностью реализует спецификацию ASGI. ASGI является стандартом для асинхронных веб-серверов и приложений в Python. Он позволяет серверам поддерживать асинхронные веб-приложения, использующие async и await.

Высокая производительность: Uvicorn разработан для обеспечения высокой производительности и масштабируемости. Он использует uvloop (быстрая реализация цикла событий asyncio на основе libuv) и httptools (быстрый парсер HTTP), что делает его одним из самых быстрых ASGI-серверов.

Многопроцессорность и многопоточность: Uvicorn может запускать несколько процессов и потоков для обработки запросов, что позволяет эффективно использовать ресурсы многоядерных серверов.

Поддержка HTTP/1.1 и HTTP/2: Uvicorn поддерживает как HTTP/1.1, так и HTTP/2, что позволяет использовать современные возможности веб-протоколов для повышения производительности.

Простота использования: Uvicorn легко устанавливается и настраивается. Он предоставляет простой интерфейс командной строки для запуска ASGI-приложений.

Широкая совместимость: Uvicorn совместим со многими асинхронными веб-фреймворками Python, такими как FastAPI, Starlette, Quart и Sanic.

Uvicorn — это библиотека или что?

Uvicorn — это, в первую очередь, библиотека, которую можно установить с помощью pip:

pip install uvicorn

Однако, Uvicorn — это не просто набор модулей и классов, которые вы импортируете в свой код. Он также предоставляет исполняемый файл (uvicorn), который используется для запуска вашего ASGI-приложения как веб-сервера. То есть, он выступает и как библиотека, и как исполняемый инструмент командной строки.

1. За что отвечает команда uvicorn main:app --reload? Почему её надо вводить именно через консоль?

**Эта команда используется для запуска ASGI (Asynchronous Server Gateway Interface) веб-приложения, написанного на Python с использованием фреймворка, такого как FastAPI или Starlette, в режиме разработки.**

Давайте разберем каждый компонент команды:

uvicorn: Это команда, которая вызывает исполняемый файл uvicorn, который является ASGI-сервером. Как мы уже обсуждали, Uvicorn принимает HTTP-запросы и перенаправляет их в ваше асинхронное веб-приложение Python, а затем отправляет ответы обратно клиенту. Uvicorn - это “серверная часть”, которая слушает запросы и передает их вашему приложению.

main:app: Этот аргумент указывает Uvicorn, где находится ваше ASGI-приложение.

main: Это имя файла Python, в котором находится ваше приложение (в данном случае, main.py). Uvicorn импортирует этот файл как Python-модуль.

app: Это имя переменной в файле main.py, которая содержит экземпляр вашего ASGI-приложения (например, экземпляр FastAPI). Uvicorn ищет объект app в модуле main и использует его как ASGI-приложение для обработки запросов.

--reload: Это опция командной строки, которая указывает Uvicorn автоматически перезагружать сервер при обнаружении изменений в файлах вашего проекта. Это очень удобно во время разработки, так как вам не нужно вручную перезапускать сервер каждый раз, когда вы вносите изменения в код. Uvicorn будет отслеживать изменения в файлах .py в той же директории, где запущен и автоматически перезагружать сервер.

Вместе: Команда uvicorn main:app --reload говорит: “Запусти, пожалуйста, Uvicorn-сервер, используй ASGI-приложение, определенное как переменная app в файле main.py, и перезагружай сервер автоматически, когда я вношу изменения в код.”

**Эту команду нужно вводить через консоль (терминал) по следующим причинам:**

* Запуск исполняемого файла: uvicorn — это исполняемый файл, который является частью установленной библиотеки Uvicorn. Консоль позволяет вам запускать исполняемые файлы, находящиеся в вашей системе (или добавленные в ваш PATH). Когда вы вводите uvicorn, операционная система ищет исполняемый файл с таким именем и запускает его.
* Передача аргументов командной строки: Консоль позволяет передавать аргументы командной строки (например, main:app и --reload) исполняемому файлу. Эти аргументы указывают исполняемому файлу, как он должен работать. В данном случае, мы сообщаем Uvicorn, где находится наше приложение и что нужно включить автоматическую перезагрузку.
* Запуск процесса сервера: Когда вы запускаете uvicorn из консоли, вы запускаете отдельный процесс, который будет работать в фоновом режиме и слушать входящие HTTP-запросы. Этот процесс будет работать до тех пор, пока вы не остановите его (например, нажав Ctrl+C в консоли). Консоль предоставляет вам способ контролировать этот процесс.
* Управление зависимостями и окружением: Консоль обычно связана с определенным окружением Python (например, виртуальным окружением), в котором установлены необходимые библиотеки (FastAPI, Uvicorn и т. д.). Это гарантирует, что при запуске Uvicorn будут доступны все необходимые зависимости. Запуск команды из консоли обеспечивает доступ к правильному окружению.

**Почему нельзя запустить из Python-кода?**

Хотя технически возможно запустить Uvicorn из Python-кода, это обычно не рекомендуется по следующим причинам:

* Сложность: Запуск Uvicorn из Python-кода требует более сложной настройки и управления циклом событий asyncio.
* Неудобство: Это менее удобно, чем просто запустить Uvicorn из консоли.
* Ограничения: Запуск Uvicorn из Python-кода может наложить ограничения на то, как вы можете управлять сервером (например, перезагрузка при изменении кода).
* Разделение ответственности: Запуск веб-сервера обычно считается отдельной задачей от кода вашего приложения. Разделение ответственности делает ваш код более модульным и удобным в обслуживании.

1. Что такое swagger? И почему ссылка не открывается?

**Swagger (теперь известный как OpenAPI) — это набор инструментов, спецификация и фреймворк с открытым исходным кодом для проектирования, построения, документирования и использования RESTful API**. Он помогает стандартизировать разработку API и делает их более понятными как для разработчиков, создающих API, так и для тех, кто их использует.

Ссылка не открывалась потому, что запуск приложения не был завершён до конца: были ошибки пути (или в целом программы/окружения).

1. Что нужно сделать, чтобы развернуть и выполнить свой роут?

**Общие шаги:**

* Написать код роута: Определить логику вашего роута, то есть функцию-обработчик, которая будет выполняться при запросе к этому роуту. Эта функция должна принимать входные данные, обрабатывать их и возвращать результат.
* Определить роут (определить маршрут): Использовать функциональность вашего веб-фреймворка для связать URL-путь (например, /users) и HTTP-метод (например, GET, POST) с вашей функцией-обработчиком. Это обычно делается с помощью декораторов или других механизмов маршрутизации.
* Создать веб-приложение: Инициализировать экземпляр веб-фреймворка (например, FastAPI(), Flask(\_\_name\_\_)). Ваши роуты будут зарегистрированы в этом экземпляре.
* Запустить веб-сервер: Использовать веб-сервер, чтобы слушать входящие HTTP-запросы и перенаправлять их в ваше веб-приложение. В разработке часто используются простые веб-серверы (например, встроенный сервер разработки Flask или Uvicorn для FastAPI), а в продакшене - более мощные веб-серверы (например, Gunicorn, Nginx, Apache).
* Развернуть приложение: Скопировать код вашего приложения на сервер и установить необходимые зависимости (например, с помощью pip install -r requirements.txt).
* Настроить веб-сервер: Настроить веб-сервер для перенаправления запросов к вашему веб-приложению. Это может включать в себя настройку виртуальных хостов, проксирование и другие параметры.
* Запустить веб-приложение: Запустить ваше веб-приложение с помощью веб-сервера.
* Проверить работоспособность: Отправить HTTP-запрос к вашему роуту (например, с помощью веб-браузера, curl или Postman) и убедиться, что он возвращает ожидаемый результат.

1. Что меняется, когда мы изменяем первоначальный декоратор функции на @app.get("/{friend}")?

Когда вы изменяете декоратор @app.get("/") на @app.get("/{friend}"), вы существенно меняете поведение роута и то, как он обрабатывает входящие запросы:

1. Изменение пути (URL):

@app.get("/"): Этот декоратор указывает, что функция-обработчик будет вызываться только для запросов, отправленных на корневой путь (/). Это означает, что URL должен быть точно /, без каких-либо дополнительных сегментов.

@app.get("/{friend}"): Этот декоратор определяет параметр пути ({friend}). Это означает, что функция-обработчик будет вызываться для запросов, отправленных на URL, который начинается с / и содержит один дополнительный сегмент после него (например, /john, /jane, /alice). Этот дополнительный сегмент будет передан в функцию-обработчик как значение параметра friend.

2. Изменение типа запросов, которые обрабатываются:

@app.get("/"): Обрабатывает только запросы, отправленные точно на корневой путь. Другие URL не будут соответствовать этому роуту.

@app.get("/{friend}"): Обрабатывает запросы, отправленные на корневой путь с одним дополнительным сегментом. Например, /anything будет соответствовать, и "anything" будет передано как значение параметра friend.

3. Введение параметра функции:

@app.get("/"): Функция-обработчик, связанная с этим роутом, обычно не принимает никаких параметров (если только вы не используете query-параметры).

@app.get("/{friend}"): Функция-обработчик, связанная с этим роутом, должна принимать параметр friend. FastAPI автоматически извлечет значение из URL и передаст его в функцию.

4. Изменение логики обработки запросов:

@app.get("/"): Функция-обработчик обычно выполняет действия, связанные с корневым путем (например, отображение главной страницы, возврат приветственного сообщения).

@app.get("/{friend}"): Функция-обработчик теперь должна использовать значение параметра friend для выполнения каких-то действий (например, отображение информации о друге с именем friend, возврат приветствия, персонализированного для friend).

1. Что такое query параметр?

Query параметр (Query parameter) в URL:

**Query параметр (или параметр запроса) — это часть URL-адреса**, которая используется для передачи дополнительных данных серверу. Он следует за основным путем (path) URL и начинается с вопросительного знака (?). Параметры запроса состоят из пар “ключ=значение”, разделенных амперсандом (&).

**Структура:**

URL: [схема]://[хост]/[путь]?[ключ1]=[значение1]&[ключ2]=[значение2]&...

[схема]: Протокол (например, http, https).

[хост]: Доменное имя или IP-адрес сервера (например, www.example.com).

[путь]: Основной путь к ресурсу (например, /products).

?: Разделитель между путем и параметрами запроса.

[ключ]=[значение]: Пара “ключ-значение”.

&: Разделитель между парами “ключ-значение”.

**Преимущества query параметров:**

Простота: Легко добавлять и изменять параметры запроса.

Универсальность: Поддерживаются всеми веб-серверами и браузерами.

Читаемость: URL с query параметрами обычно легко читаются.

**Недостатки query параметров:**

Ограниченная длина URL: Существует ограничение на максимальную длину URL (обычно около 2000 символов), что ограничивает количество и размер параметров.

Видимость: Параметры запроса видны в адресной строке браузера, что может быть нежелательно для конфиденциальных данных (например, паролей). Конфиденциальные данные следует передавать в теле запроса (например, с использованием HTTP POST).

Порядок параметров: В большинстве случаев порядок параметров не имеет значения, но некоторые серверы могут полагаться на определенный порядок, что может привести к проблемам.

Когда использовать query параметры:

Когда нужно передать серверу дополнительные данные, которые не являются частью идентификатора ресурса.

Для фильтрации, сортировки, пагинации и поиска данных.

Для передачи небольших объемов данных, которые не являются конфиденциальными.

**Альтернативы query параметрам:**

Параметры пути (Path parameters): Используются для идентификации конкретных ресурсов (например, /users/{user\_id}).

Тело запроса (Request body): Используется для передачи больших объемов данных, особенно при создании или обновлении ресурсов (с использованием HTTP POST, PUT, PATCH).

HTTP-заголовки (HTTP headers): Используются для передачи метаданных о запросе и ответе.

1. Что такое пагинация?

Пагинация (Pagination):

**Пагинация — это метод разбиения большого набора данных на отдельные страницы**, чтобы облегчить навигацию и представление данных пользователю. Вместо отображения всех данных сразу, пагинация показывает только определенное количество элементов на каждой странице, позволяя пользователю переходить между страницами для просмотра всех данных.