МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное учреждение высшего образования

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

Лабораторная работа №15

**Реализация REST API на FastAPI**

**по дисциплине**

**«Введение в информационные технологии»**

Выполнил: студент гр. БВТ2403

Лашков Ю.Е.

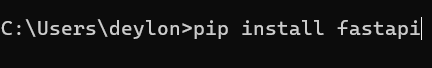
Проверил: Павликов А.Е.

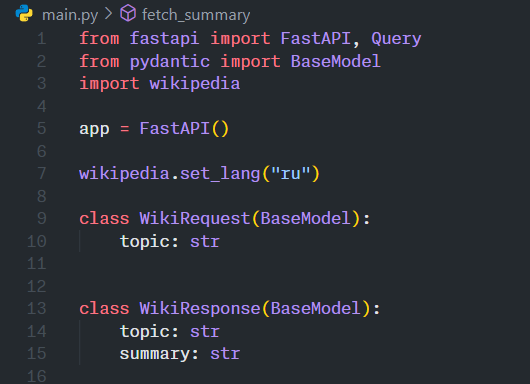
Москва, 2025 г.

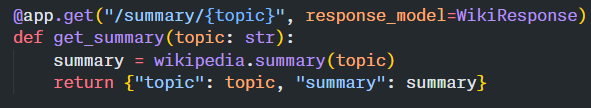
**Цель работы:**  
Создание простого REST API с использованием FastAPI для демонстрации основных концепций, таких как маршруты, параметры пути, параметры запроса и передача данных в теле запроса  
  
**Оборудование:**  
1. Установленное программное обеспечение: Python, FastAPI, Uvicorn, pyjokes, pydantic.  
  
**Введение:**  
FastAPI — это современный Python веб-фреймворк для быстрого создания API с автоматической генерацией документации, высокой производительностью и встроенной валидацией данных. FastAPI был выбран для этой лабораторной работы из-за его скорости, простоты и надежности. Он позволяет быстро разрабатывать и тестировать API, а также обеспечивает валидацию данных и автоматическую генерацию документации, что важно для поддержания высокого качества кода. Фреймворк предоставляет возможности для определения маршрутов, валидации входных и выходных данных с использованием Pydantic, а также автоматического создания документации API с помощью Swagger UI

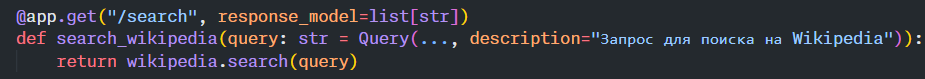
.

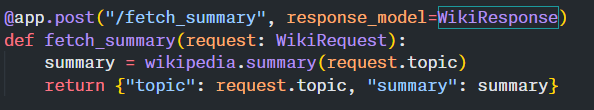
**ПРАКТИЧЕСКАЯ РАБОТА**

  
  
  
  
  
Для начала работы с FastAPI и Uvicorn, убедился в их установке, выполнив команды pip install fastapi и pip install uvicorn. После установки проверил наличие установленных пакетов с помощью команды pip list. Убедился, что в списке присутствуют fastapi версии 0.115.12 и uvicorn версии 0.34.1

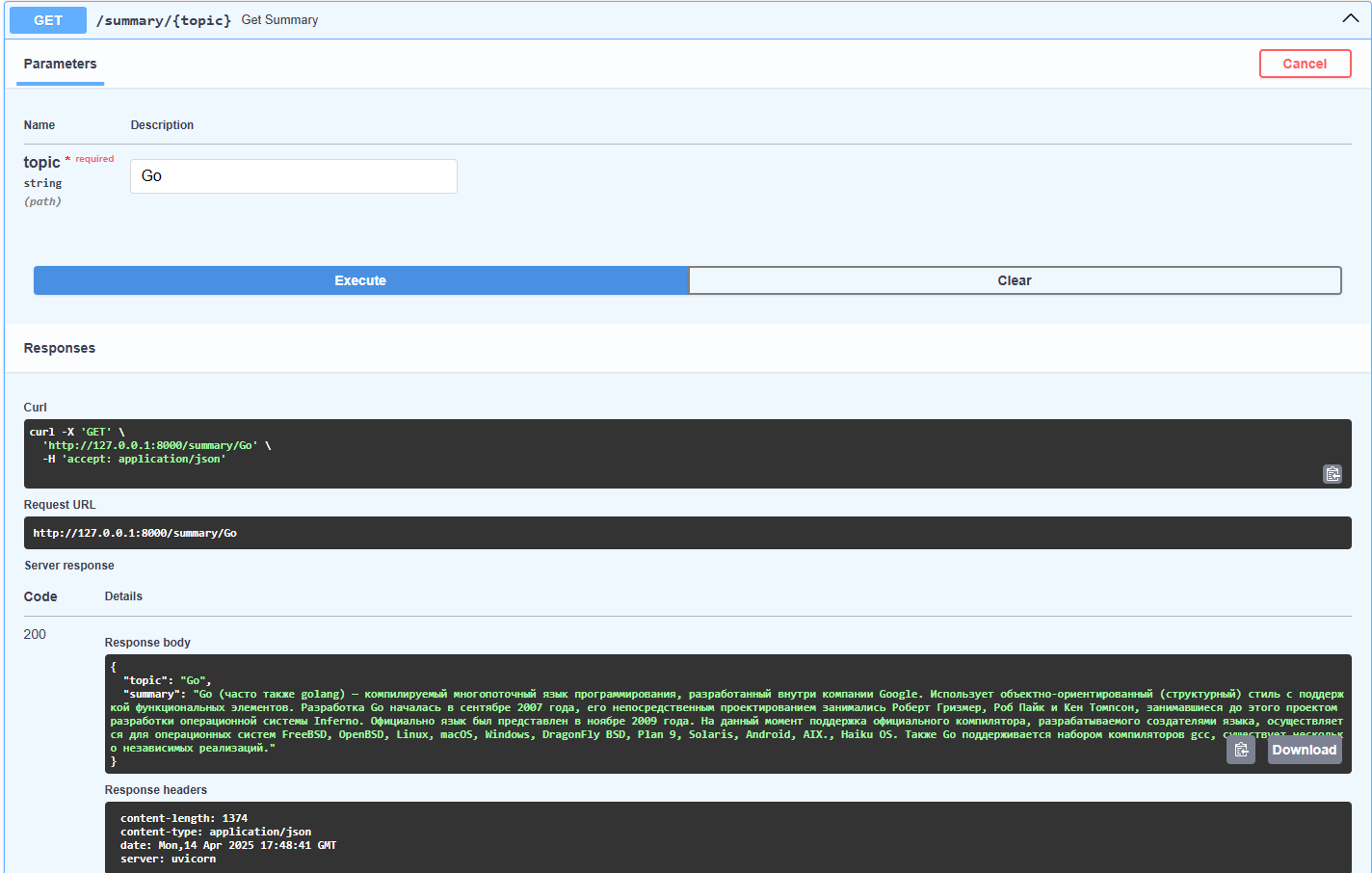
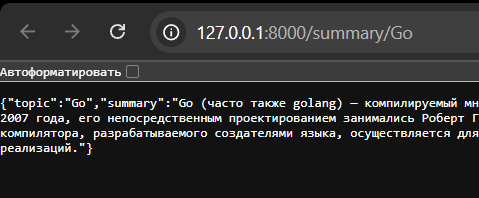
  
Здесь я импортировал необходимые библиотеки: FastAPI для создания API, Pydantic для определения моделей данных и wikipedia для работы с Википедией. Затем инициализировал приложение FastAPI, присвоив его переменной app. Также установил русский язык для использования в Википедии с помощью wikipedia.set\_lang("ru"). Определил две модели Pydantic: WikiRequest для входящего запроса, содержащую поле topic (строка) - тему для поиска, и WikiResponse для исходящего ответа, содержащую поля topic (строка) - тему поиска и summary (строка) - краткое содержание найденной статьи.

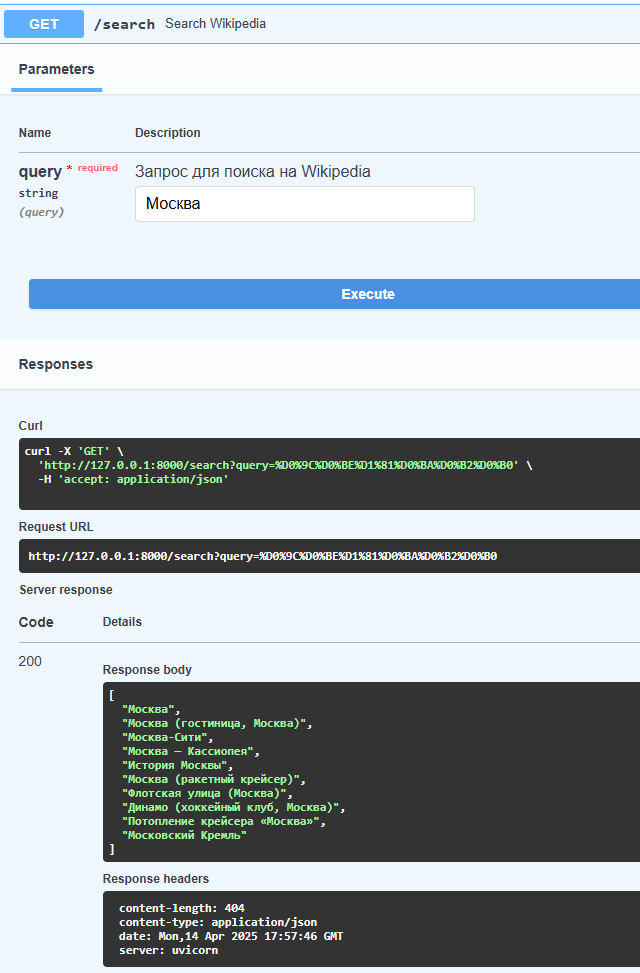
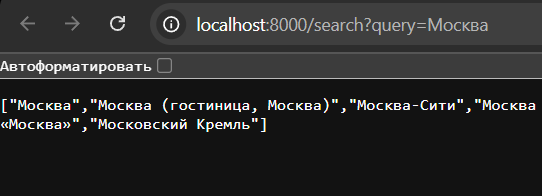
  
Я определил GET-запрос с помощью @app.get("/summary/{topic}", response\_model=WikiResponse). Он связывает URL /summary/{topic} с функцией get\_summary(topic), принимающей тему. response\_model=WikiResponse гарантирует, что ответ будет соответствовать заданной структуре. Функция get\_summary получает краткое содержание из Википедии с помощью wikipedia.summary(topic) и возвращает его вместе с темой в формате JSON.

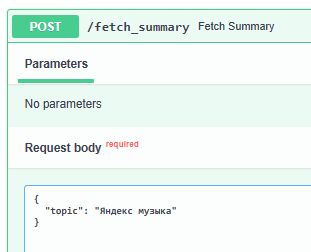
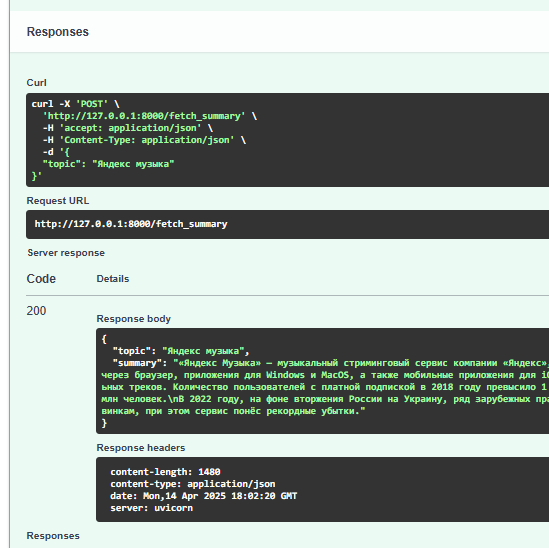
  
Здесь я определил GET-запрос к URL /search, используя декоратор @app.get("/search", response\_model=list[str]). Этот запрос принимает параметр query через Query-параметр, который описывает строку для поиска в Wikipedia. response\_model=list[str] указывает, что API возвращает список строк с результатами поиска.  
Функция search\_wikipedia(query = Query(..., description="Запрос для поиска на Wikipedia")) использует функцию wikipedia.search(query) для выполнения поиска по Википедии. description="Запрос для поиска на Wikipedia" добавляет описание для параметра query, которое будет отображаться в документации API. Функция возвращает список найденных соответствий поисковому запросу.

  
Здесь я определил POST-запрос к URL /fetch\_summary с помощью @app.post("/fetch\_summary", response\_model=WikiResponse). Он принимает данные в формате WikiRequest (тема для поиска) в теле запроса. response\_model=WikiResponse указывает на формат ответа.  
Функция fetch\_summary(request: WikiRequest) получает тему из запроса (request.topic) и извлекает краткое содержание из Википедии с помощью wikipedia.summary(request.topic). Возвращает topic и summary в формате JSON.

  
Полученный результат.

  
Изучил функциональность GET-запросов, отправив запрос на URL http://127.0.0.1:8000/summary/Go с параметром topic равным Go. В результате был получен ответ с кодом состояния 200 OK, что свидетельствует об успешном выполнении запроса. Тело ответа содержит JSON-объект с информацией о языке программирования Go, включая краткое описание.  
  
Для дополнительной проверки корректности работы API, после выполнения запроса через Swagger UI, открыл URL `http://127.0.0.1:8000/summary/Go` непосредственно в браузере. Убедился, что браузер отображает JSON-ответ, идентичный тому, что был получен в Swagger UI.

  
Для тестирования функциональности поиска, отправил GET-запрос к URL http://127.0.0.1:8000/search с параметром query, имеющим значение "Москва". Параметр query был передан в строке запроса (query string). В Swagger UI в поле для параметра query было введено значение "Москва" и выполнен запрос.  
Сервер вернул ответ с кодом состояния 200 OK. Тело ответа содержит JSON-массив со списком найденных статей, соответствующих поисковому запросу.  
  
Аналогично сделал дополнительную проверку, убедился в корректности работы API.

  
  
Для тестирования POST-запроса отправил запрос на URL http://127.0.0.1:8000/fetch\_summary с телом запроса в формате JSON, содержащим параметр topic со значением "Яндекс музыка". После выполнения запроса, был получен ответ с кодом состояния 200 OK. Тело ответа содержало JSON-объект с информацией о "Яндекс музыке".

**Заключение:**В ходе лабораторной работы было успешно разработано REST API на FastAPI для взаимодействия с Wikipedia. Реализованы основные функции: поиск статей и получение их описаний через GET и POST-запросы