МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Факультет «Информационные технологии»

Кафедра «Искусственный интеллект и машинное обучение»

Лабораторная работа №15
Основы работы с протоколом HTTP

Автор:

Голиков Михаил Вячеславович, БВТ2402

Москва, 2025

Цель лабораторной 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 валидироваться по схемам.

Ход выполнения лабораторной работы

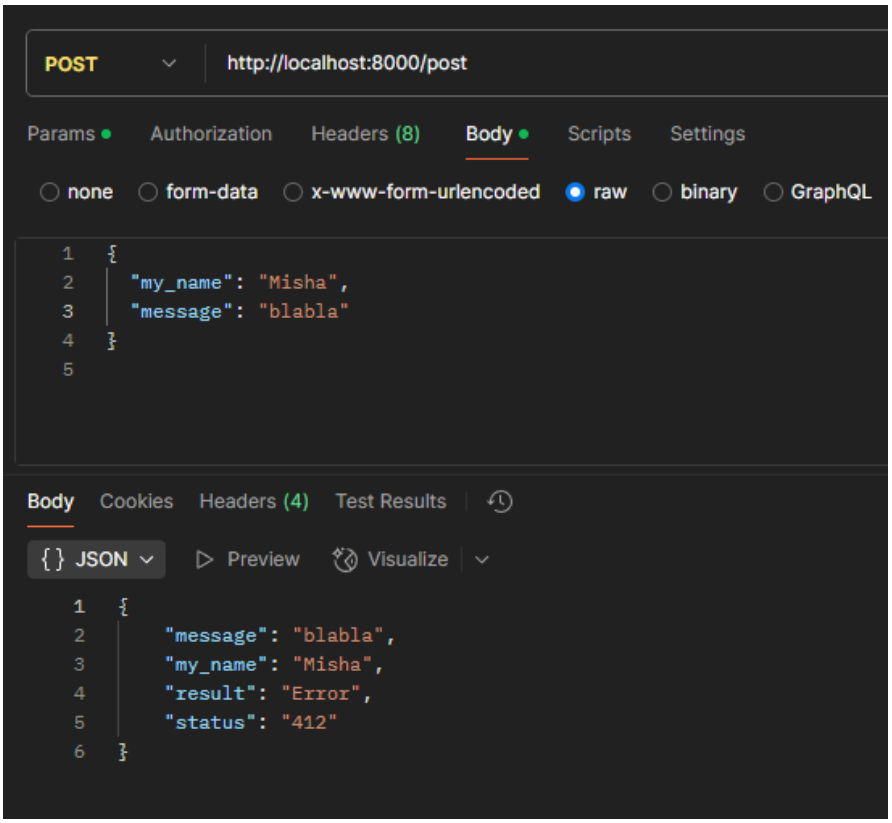
Сперва был разработан метод обработки тела запроса через команду Post:

```python
@app.post("/post", response_model=Response_message)
async def post_port(content: Search_message):
    try:
        url = wikipedia.page(content.message).url
        status_code = str(status.HTTP_200_OK)
    except:
        result = "Error"
        status_code = str(status.HTTP_412_PRECONDITION_FAILED)

    return Response_message(
        message=content.message,
        my_name=content.my_name,
        result=url,
        status=status_code
    )
```

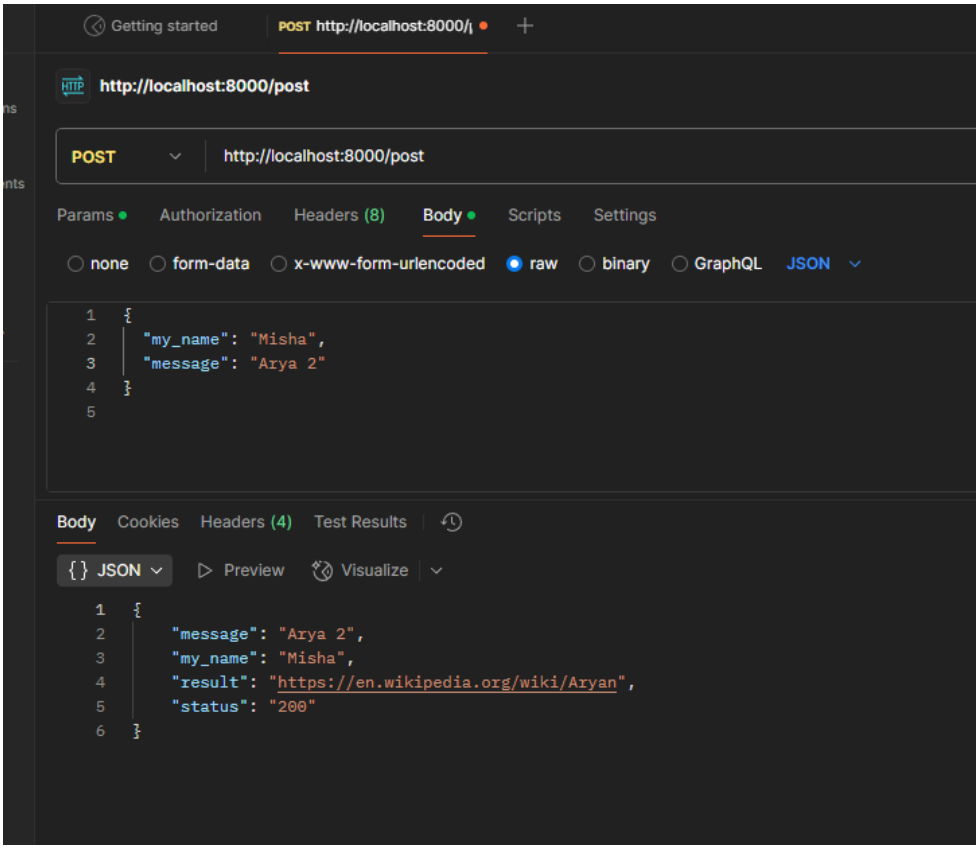
Элемент 1 — обработка тела запроса

```python
from pydantic import BaseModel, constr
from fastapi import status

class Search_message(BaseModel):
    message: str | None = None
    my_name: constr(min_length=3, max_length=50)

class Response_message(Search_message, BaseModel):
    result: str | None = None
    status: str = str(status.HTTP_412_PRECONDITION_FAILED)
```

Элемент 2 — схемы для обработки тела запроса

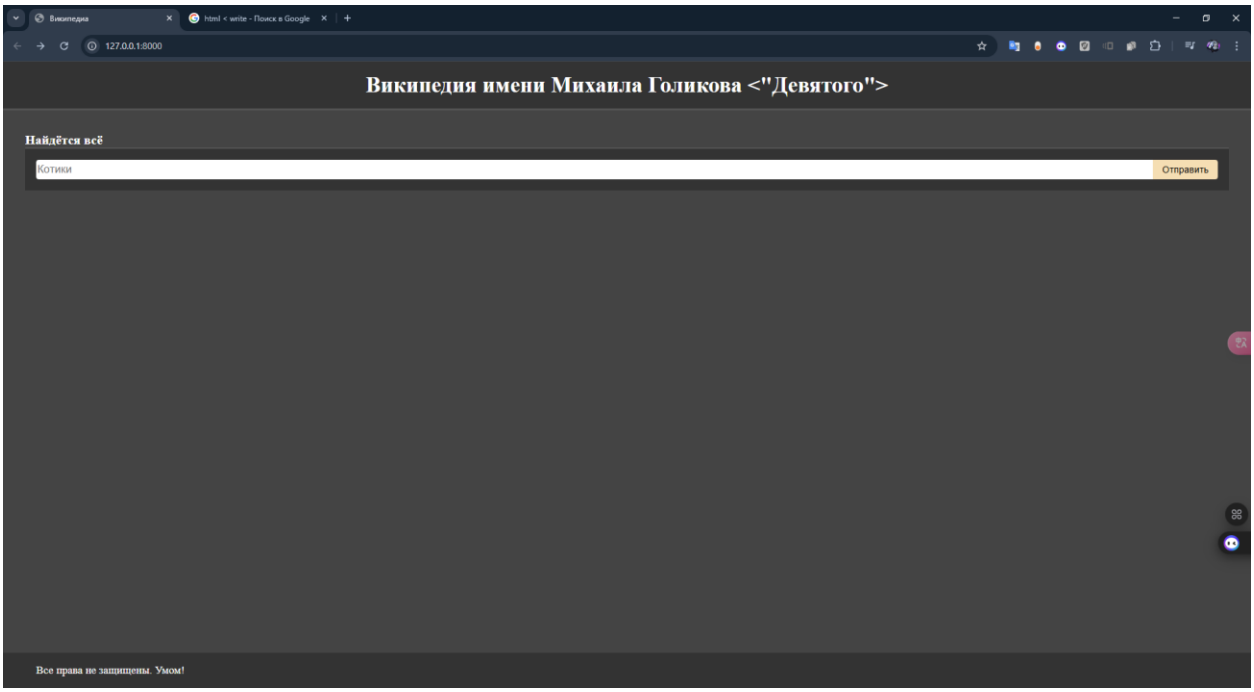


Элемент 3 — пример неудачного запроса



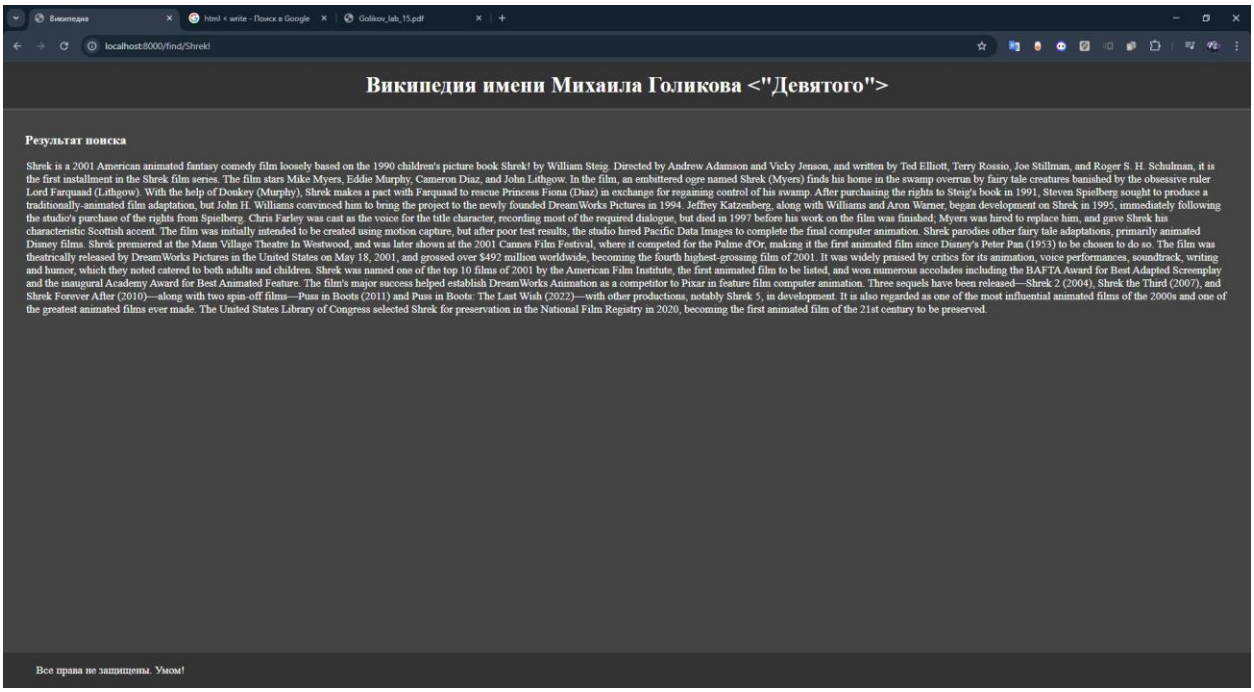
Элемент 4 — пример удачного запроса

Далее был написан небольшой сайт для поиска на википедии, который необходимый запрос отправляет в качестве параметра запроса (также был введён необязательный параметр limit, ограничивающий количество совпадений). Пользователю приходят ссылки на ресурсы, по которым он может перейти через параметр пути.



Элемент 5 — пример базовой страницы поиска

Элемент 6 — пример страницы поиска при запросе



Элемент 7 — пример перехода по ссылке

```
@app.get("/", response_class=HTMLResponse)
async def main(request: Request, message: str | None = None, limit: int = 10):
    if message is None:
        return FileResponse("index_main.html")
    else:
        decoded_message = parse.unquote(message)
        list_research = wikipedia.search(decoded_message, results=limit)
        return templates.TemplateResponse("index_search_reasults.html", {"request":
request, "content": list_research})
```

Элемент 8 — код для работы с параметрами запросами

```
@app.get("/find/{content}", response_class=HTMLResponse)
async def find_page(request: Request, content: str):
    decoded_content = parse.unquote(content)
    try:
        page_summary = wikipedia.summary(decoded_content)
        return templates.TemplateResponse("result_summary.html", {"request": request,
"content": page_summary})
    except:
        return RedirectResponse(url="/")
```

Элемент 9 — код для работы с параметрами пути

Также был написан код для обработки ошибок путей сервера (на данный момент при любой ошибке происходит переадресация на начальную страницу)

```python
@app.exception_handler(HTTPException)
async def custom_404_handler(exc: HTTPException):
    if exc.status_code == 404:
        return RedirectResponse(url="/", status_code=status.HTTP_302_FOUND)
    return RedirectResponse(url="/", status_code=status.HTTP_302_FOUND)
```

Элемент 10 — код для работы с исключениями

Заключение

Был успешно изучен процесс разработки Backend части сайтов. Все задачи выполнены, результаты соответствуют ожидаемым.