Лабораторная работа №15

Основы работы с протоколом HTTP

Автор:

Голиков Михаил Вячеславович, БВТ2402

**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 валидирироваться по схемам.

**Ход выполнения лабораторной работы**

Сперва был разработан метод обработки тела запроса через команду Post:

@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

from pydantic import BaseModel, constr

from fastapi import status

class Search\_message(BaseModel):

    message: str | None = None

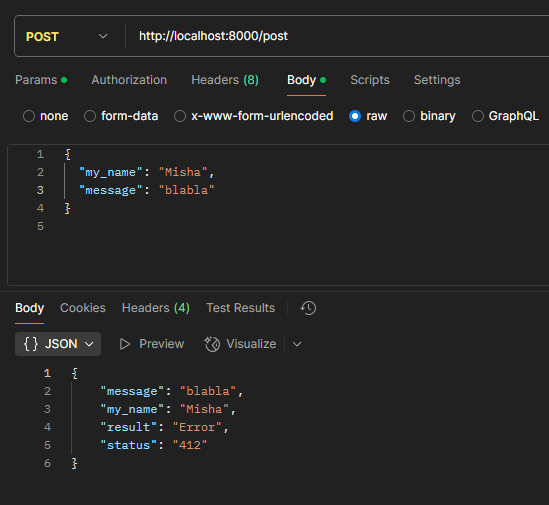
    my\_name: constr(min\_length=3, max\_length=50)

class Response\_message(Search\_message, BaseModel):

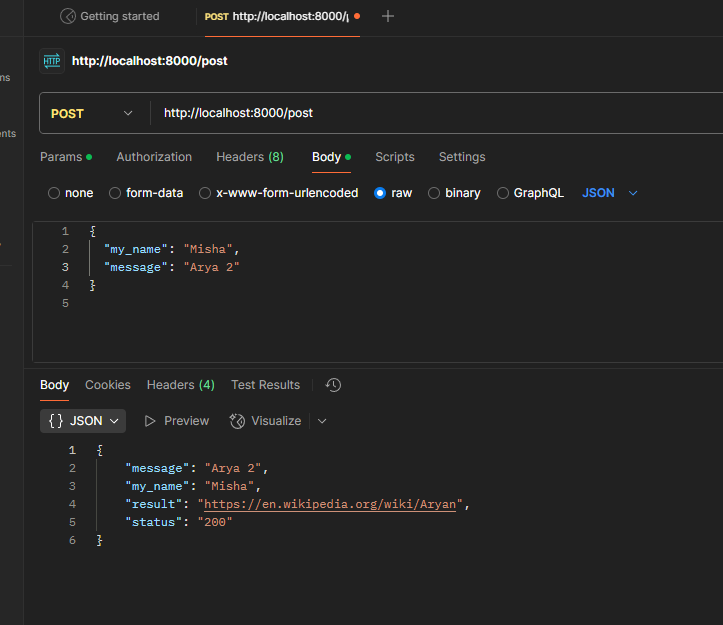
    result: str | None = None

    status: str = str(status.HTTP\_412\_PRECONDITION\_FAILED)

Элемент 2 ― схемы для обработки тела запроса

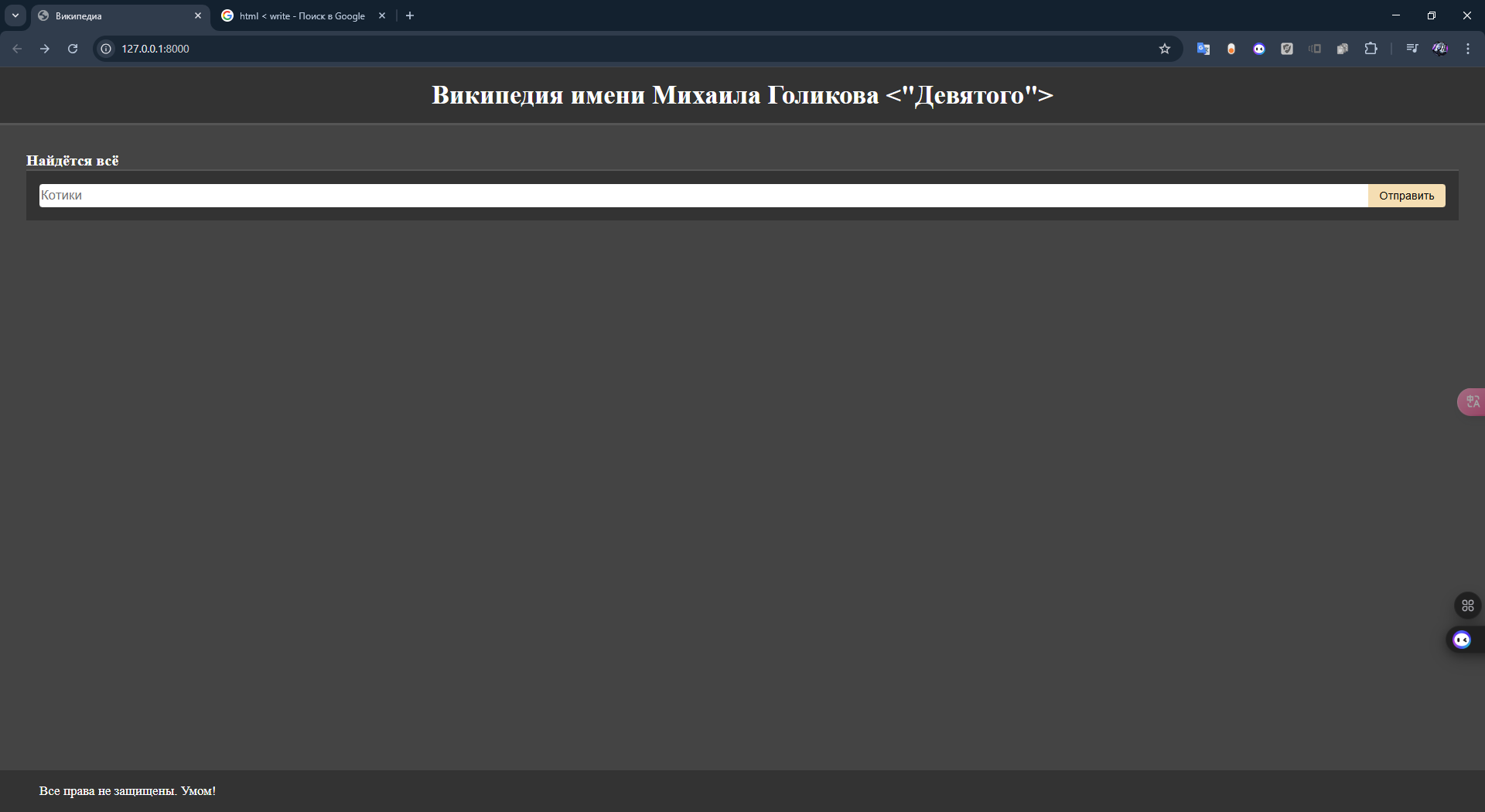


Элемент 3 ― пример неудачного запроса

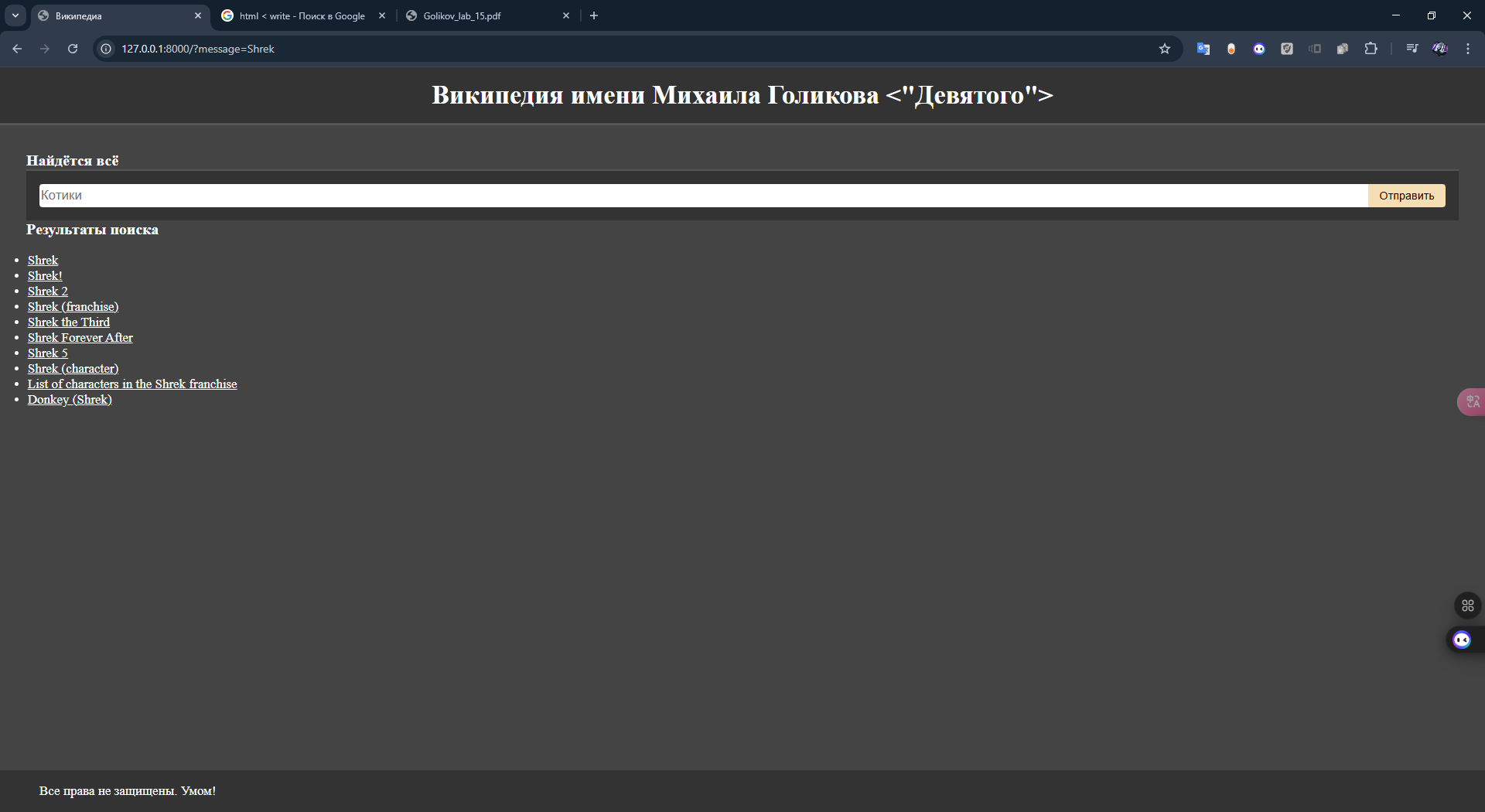


Элемент 4 ― пример удачного запроса

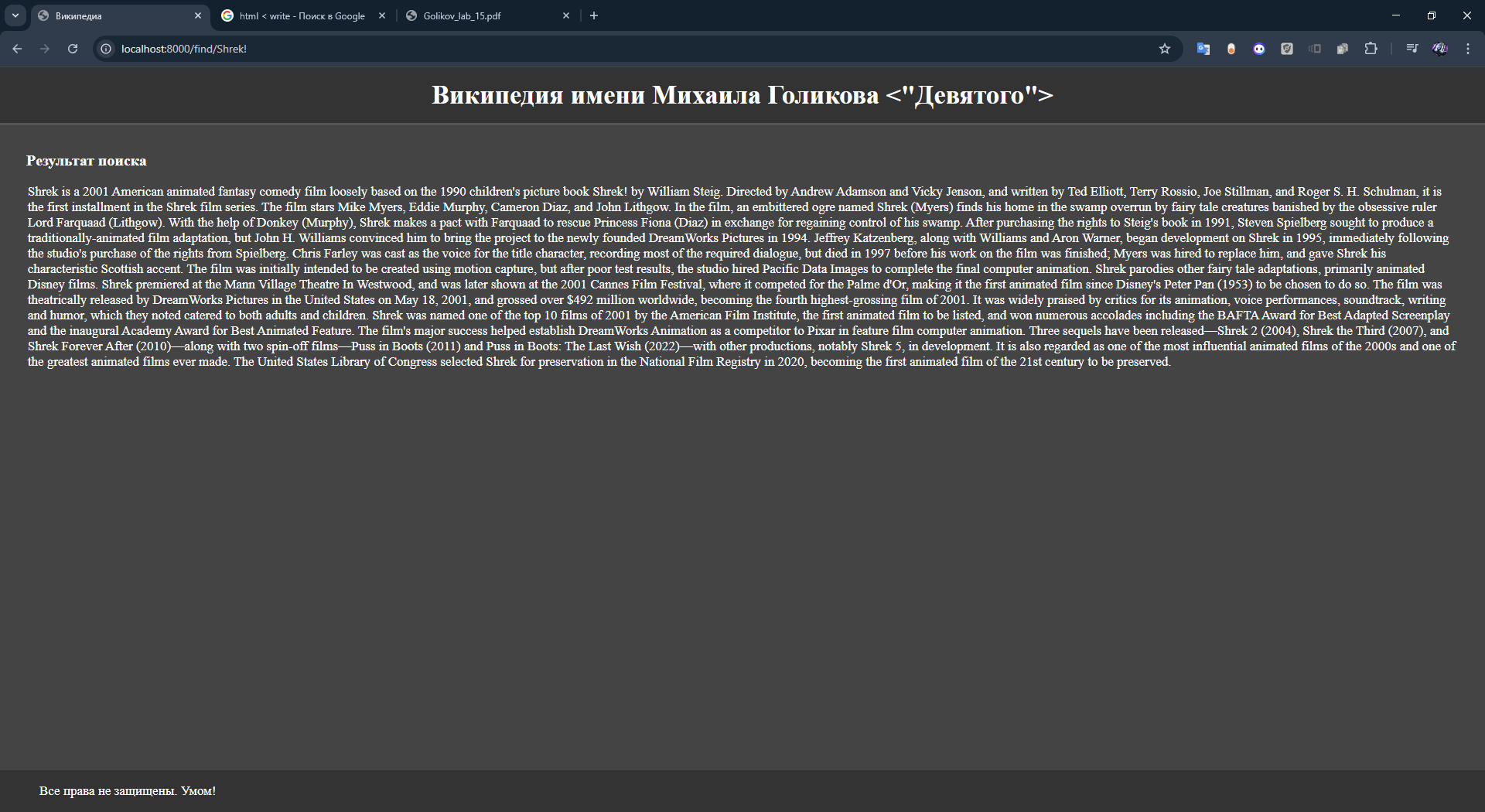
Далее был написан небольшой сайт для поиска на википедии, который необходимый запрос отправляет в качестве параметра запроса (также был введён необязательный параметр limit, ограничивающий количество совпадений). Пользователю приходят ссылки на ресурсы, по которым он может перейти через параметр пути.



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)

@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.