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»  
(МТУСИ)

Лабораторная работа №2  
по дисциплине  
«DevOps»

Выполнили:  
студенты группы БВТ2305  
Жирнова П. В.

Проверил:  
Преподаватель DevOps  
Тимчук А.В.

Москва,

2025

## Задание

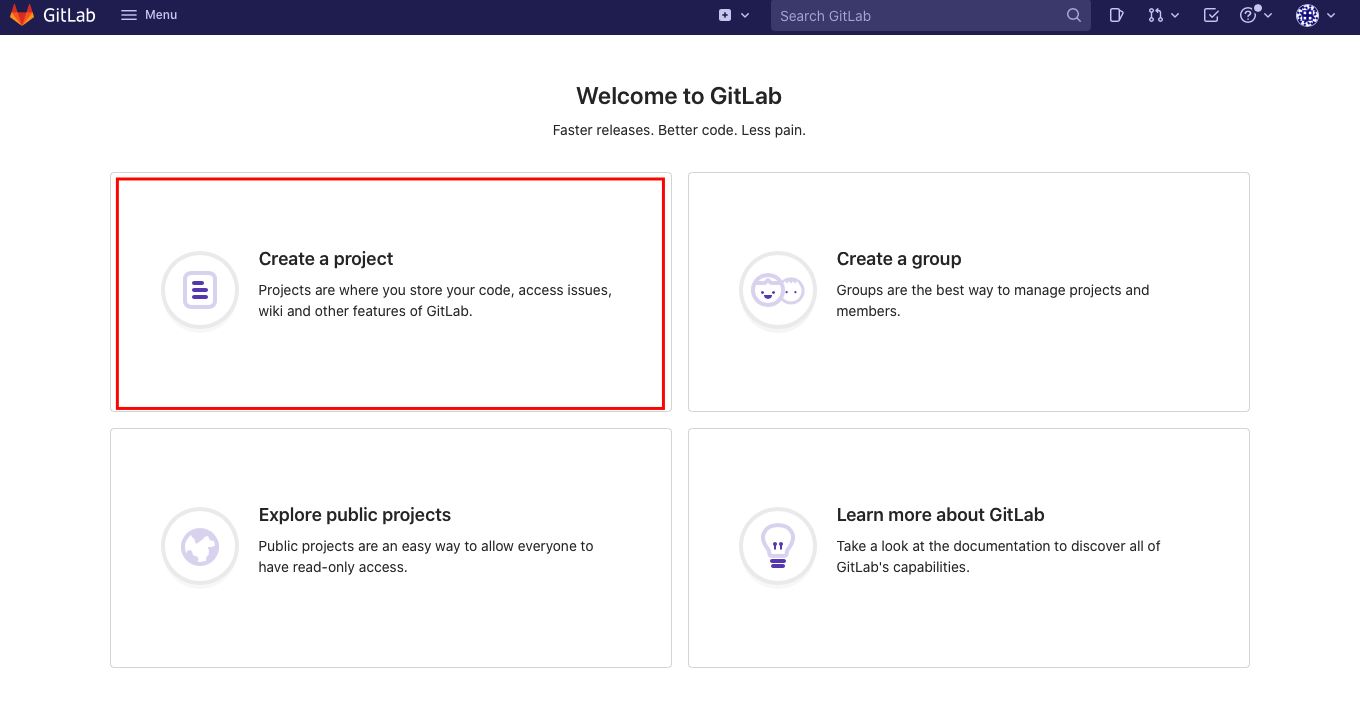
Необходимо будет создать простой REST сервис (без GUI) и написать к нему тесты (желательно воспользоваться встроенным в фреймворк инструментами, либо pytest + requests, что не очень правильно).

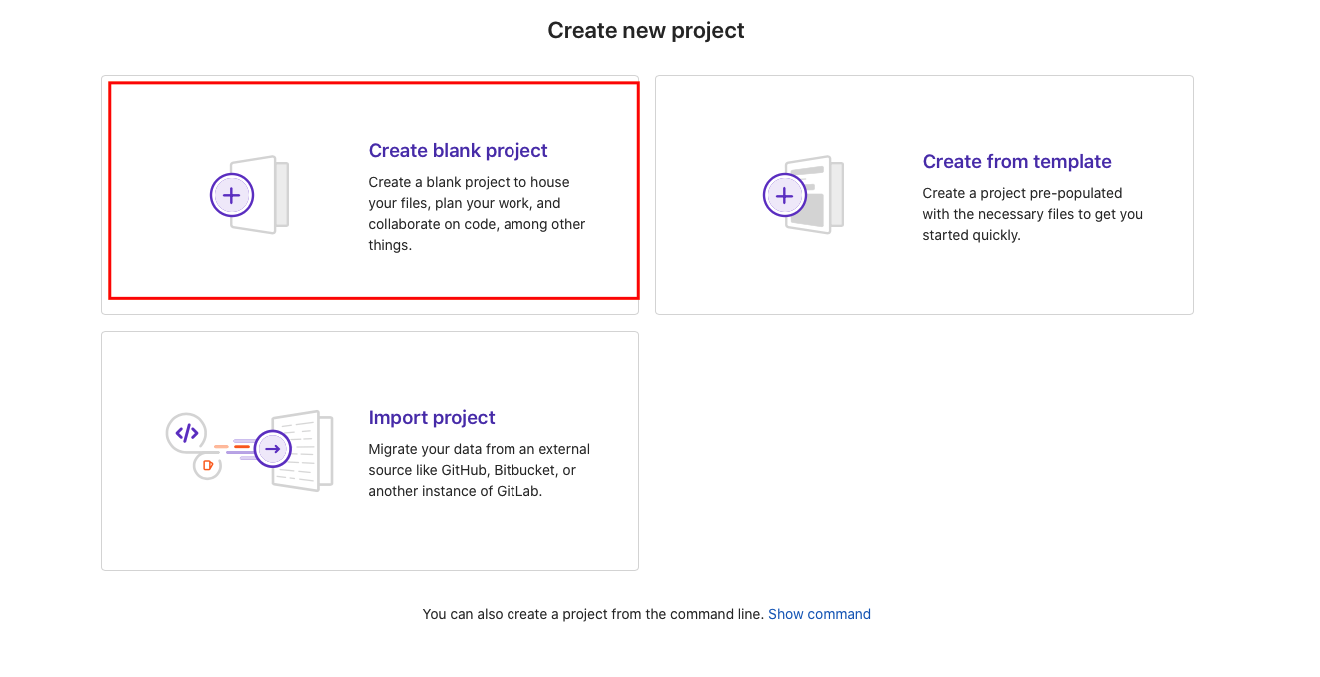
Выбор серверной библиотеки и задания осуществляется по номеру в журнале в следующей таблице (10 в списке снова берет 1 и т.д.):

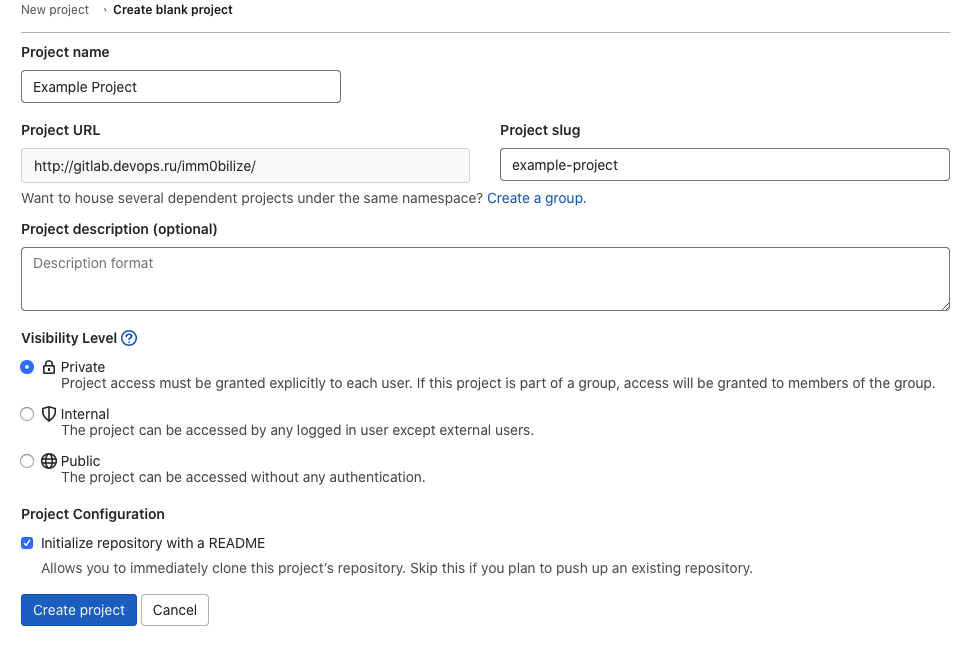
|  |  |  |
| --- | --- | --- |
| № | Фреймворк | Задание |
| 1 | FastAPI | Калькулятор |
| 2 | FastAPI | Форматирование текста |
| 3 | FastAPI | N-ое число ряда простых чисел или ряда Фибоначчи |
| 4 | Flask | Калькулятор |
| 5 | Flask | Форматирование текста |
| 6 | Flask | N-ое число ряда простых чисел или ряда Фибоначчи |
| 7 | Tornado | Калькулятор |
| 8 | Tornado | Форматирование текста |
| 9 | Tornado | N-ое число ряда простых чисел или ряда Фибоначчи |

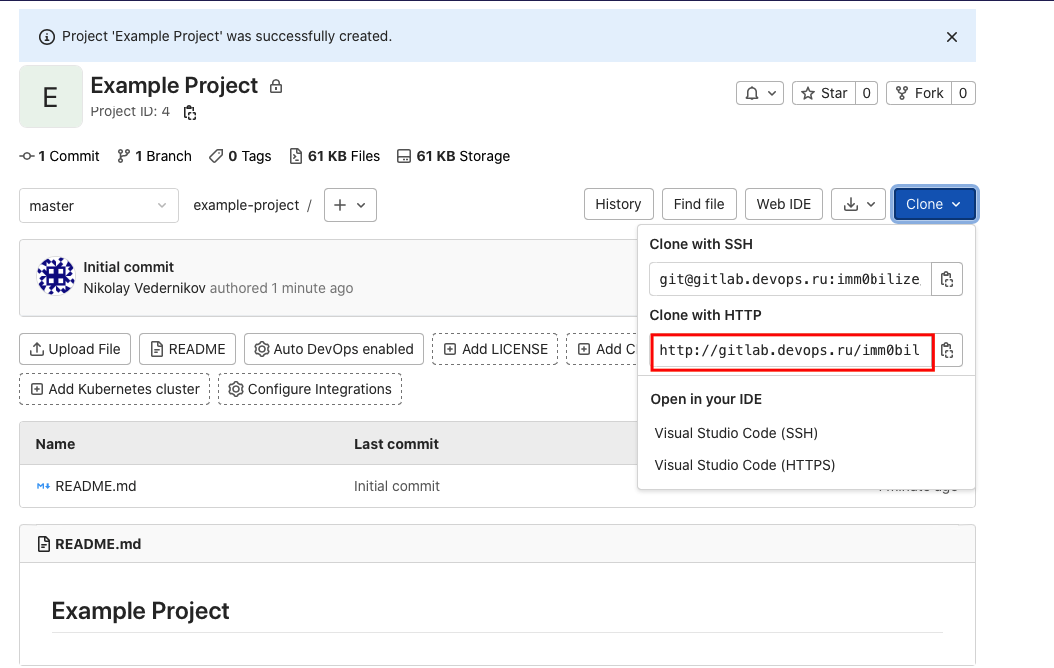
Далее необходимо будет сделать запушить полученный код в репозиторий в гитлаб.

Пример:

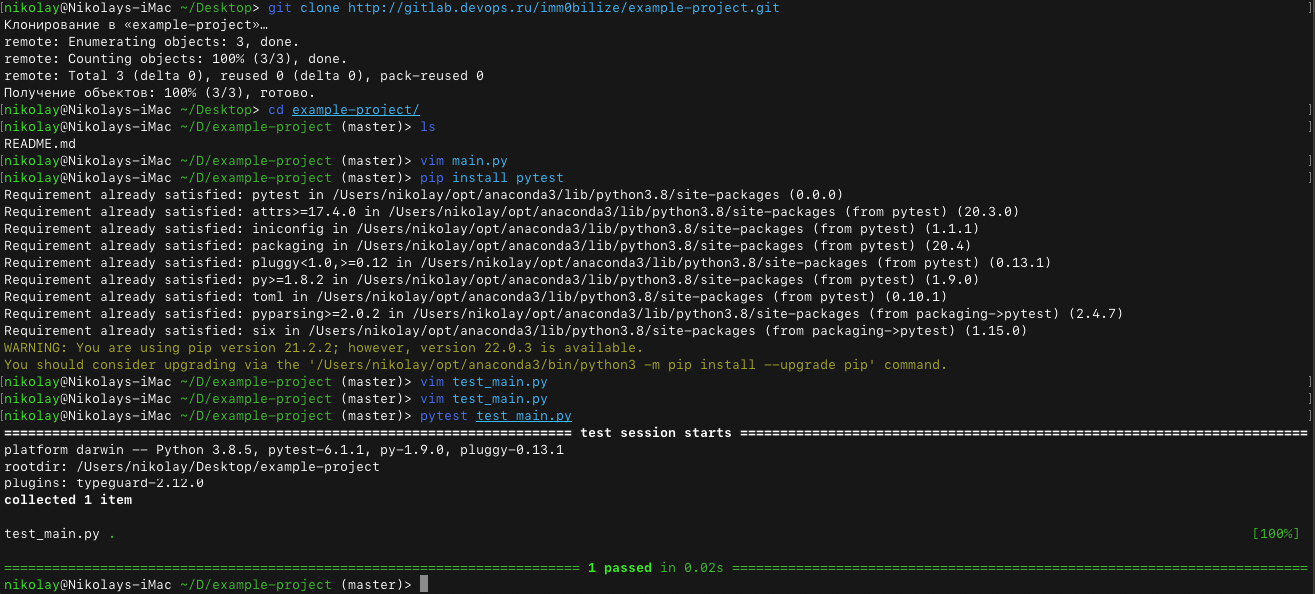








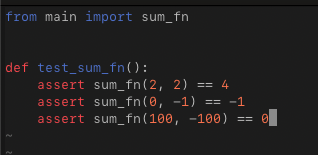
Пример подготовки кода. Обратите внимание, в файлах main.py и test\_main.py представлен код не rest сервиса, а демонстрирующий принцип разделения кода на рабочий и тестирующий. Для понимания, как тестировать rest сервис, обратитесь к документации на фреймворк



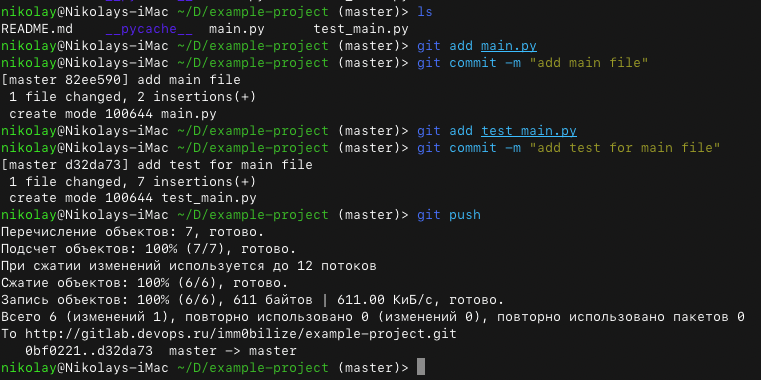
Файл main.py:



Файл test\_main.py:

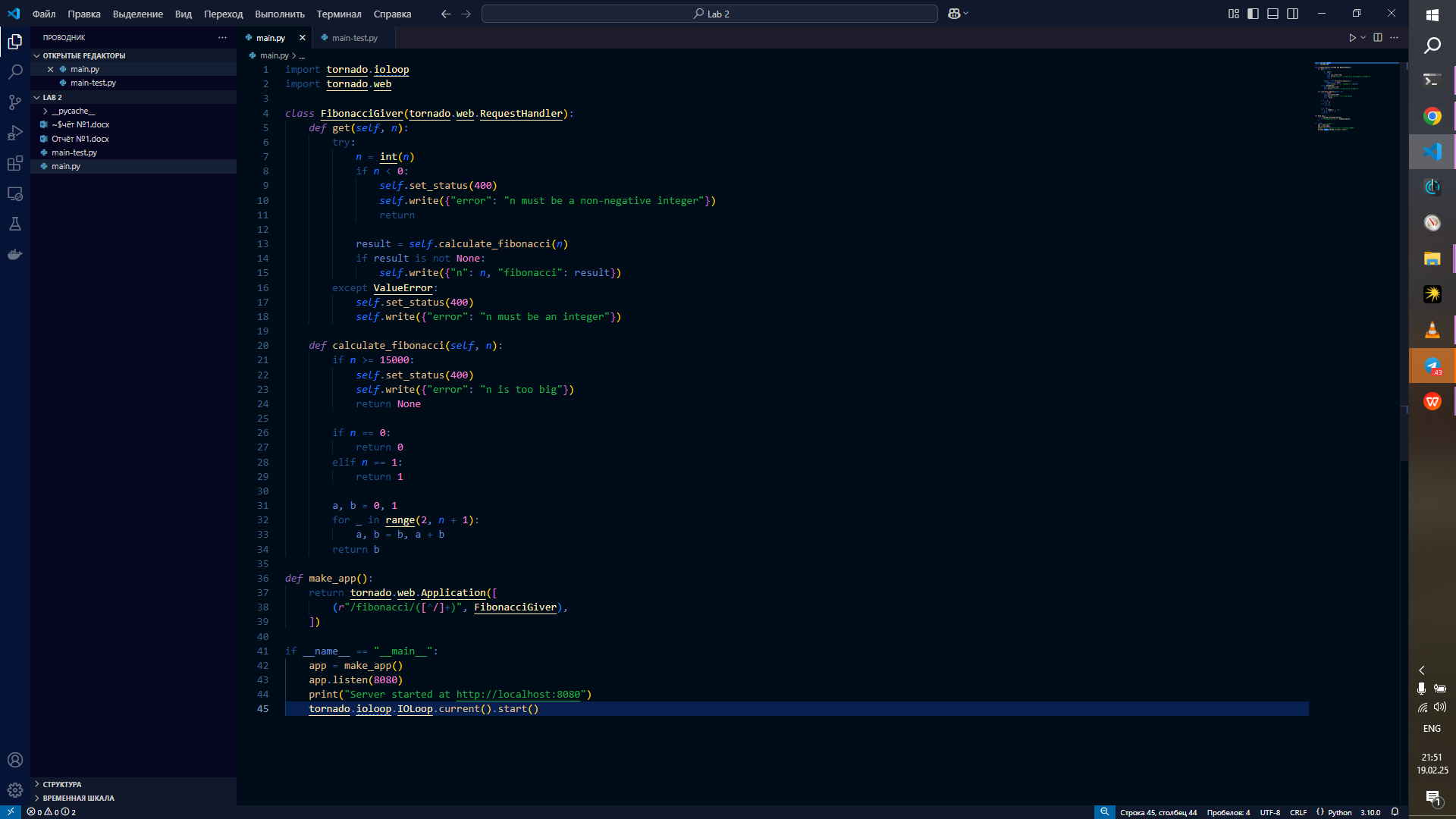


Пушим получившийся результат в созданный репозиторий.



Выполнение:

основной сервис main.py



Листинг кода:  
  
```import tornado.ioloop

import tornado.web

*class* FibonacciGiver(tornado.web.RequestHandler):

*def* get(*self*, *n*):

        try:

*n* = int(*n*)

            if *n* < 0:

*self*.set\_status(400)

*self*.write({"error": "n must be a non-negative integer"})

                return

            result = *self*.calculate\_fibonacci(*n*)

            if result is not None:

*self*.write({"n": *n*, "fibonacci": result})

        except ValueError:

*self*.set\_status(400)

*self*.write({"error": "n must be an integer"})

*def* calculate\_fibonacci(*self*, *n*):

        if *n* >= 15000:

*self*.set\_status(400)

*self*.write({"error": "n is too big"})

            return None

        if *n* == 0:

            return 0

        elif *n* == 1:

            return 1

        a, b = 0, 1

        for \_ in range(2, *n* + 1):

            a, b = b, a + b

        return b

*def* make\_app():

    return tornado.web.Application([

        (*r*"/fibonacci/([^/]+)", FibonacciGiver),

    ])

if \_\_name\_\_ == "\_\_main\_\_":

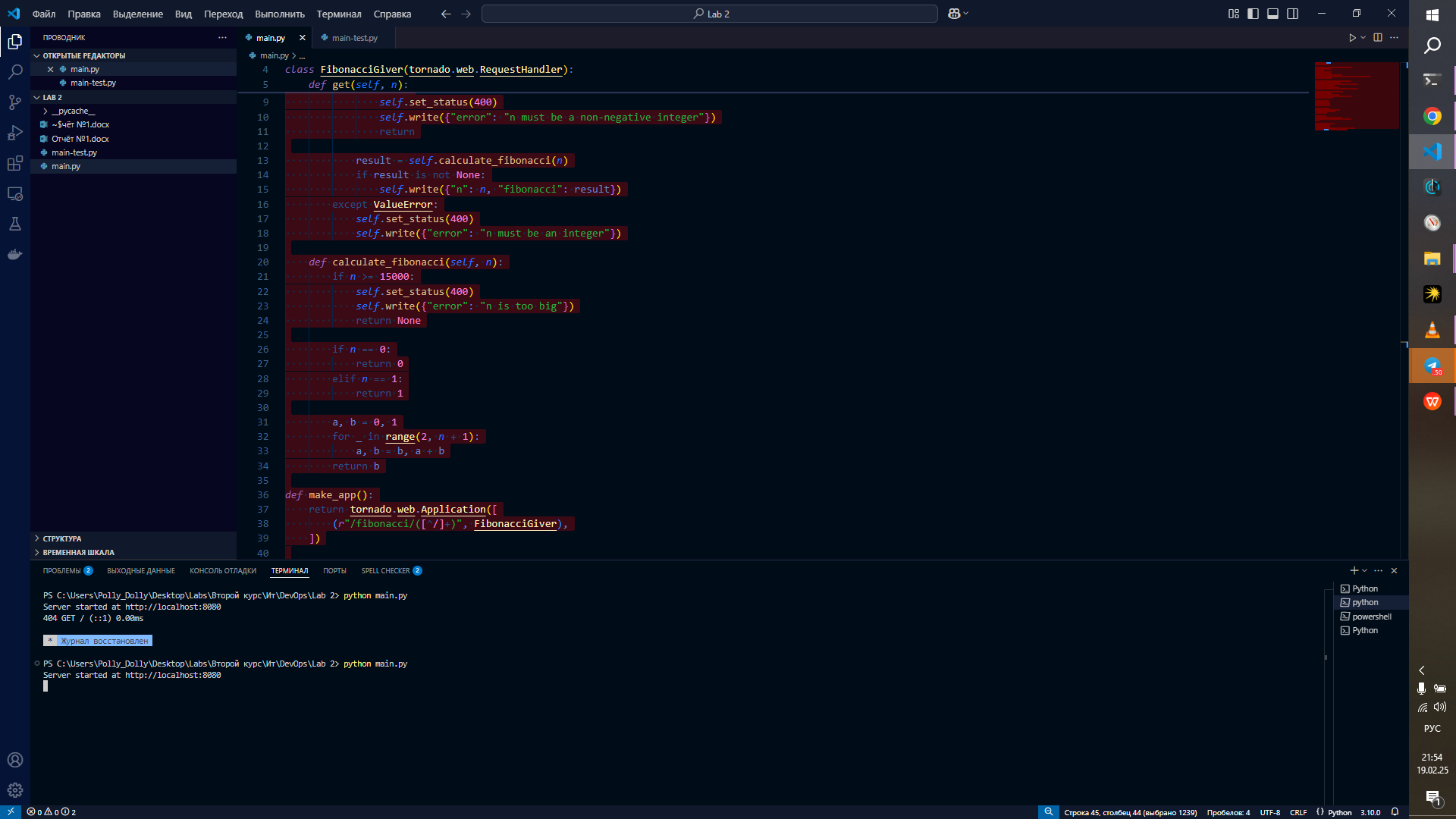
    app = make\_app()

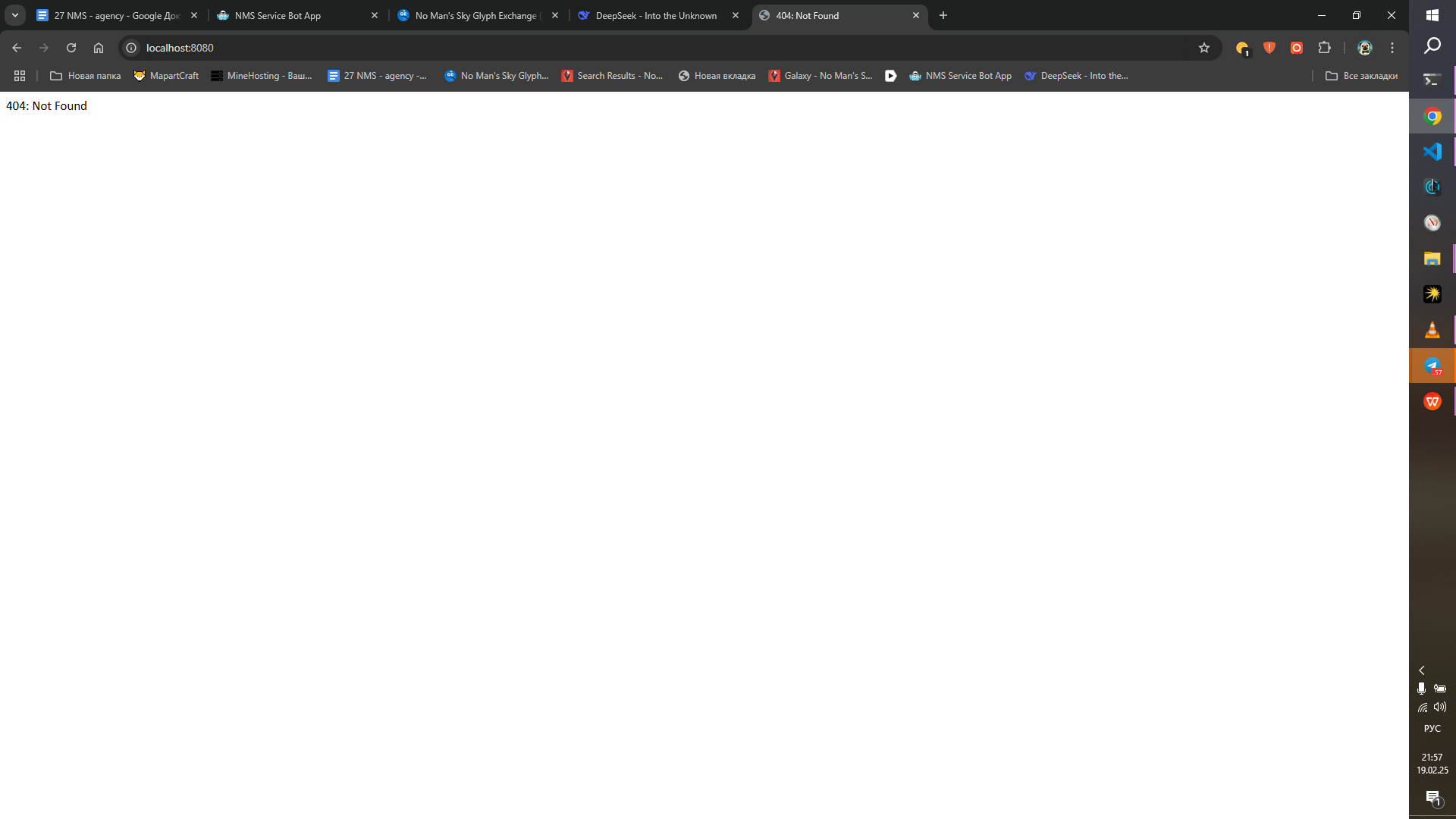
    app.listen(8080)

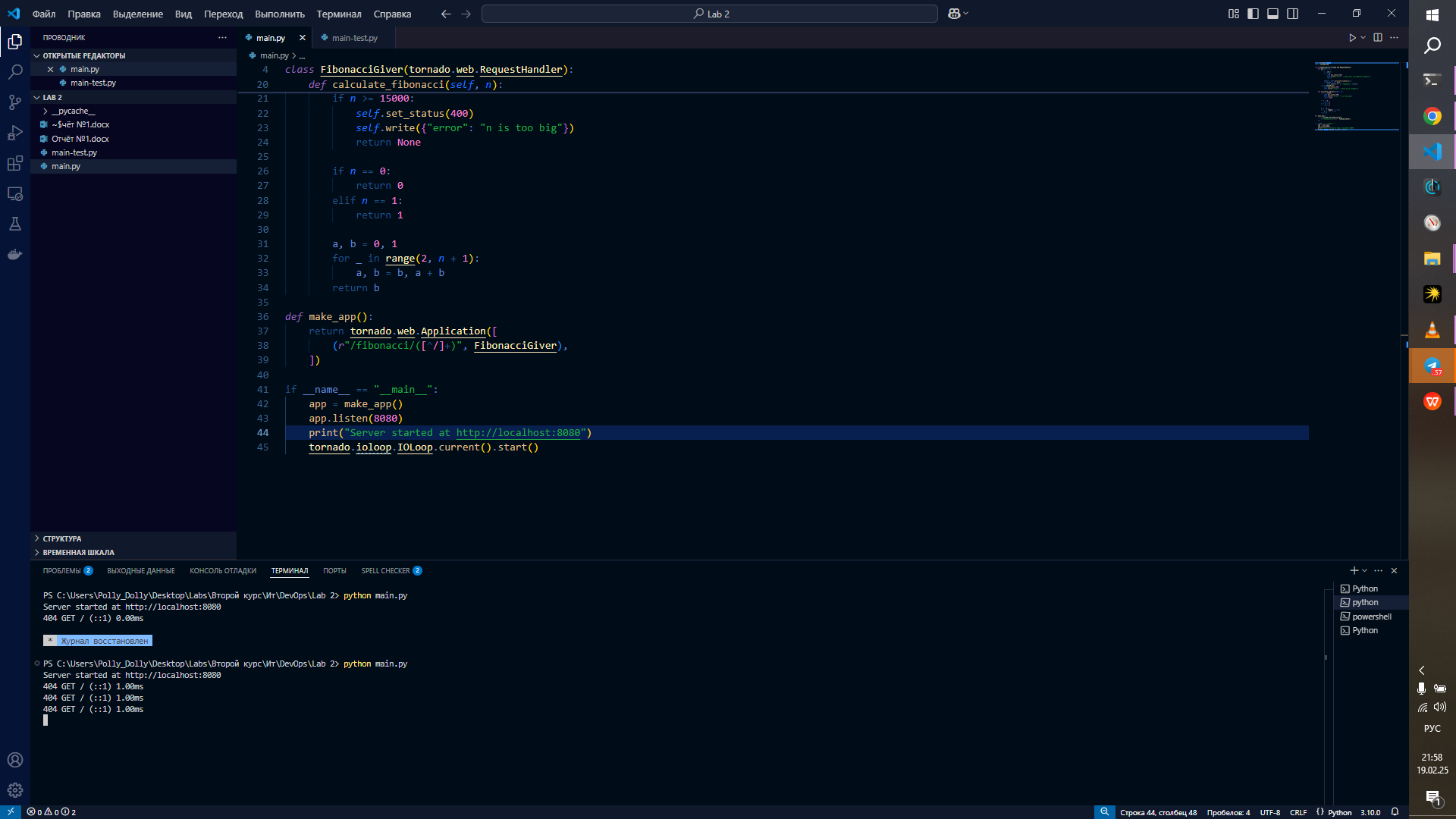
    print("Server started at http://localhost:8080")

    tornado.ioloop.IOLoop.current().start()

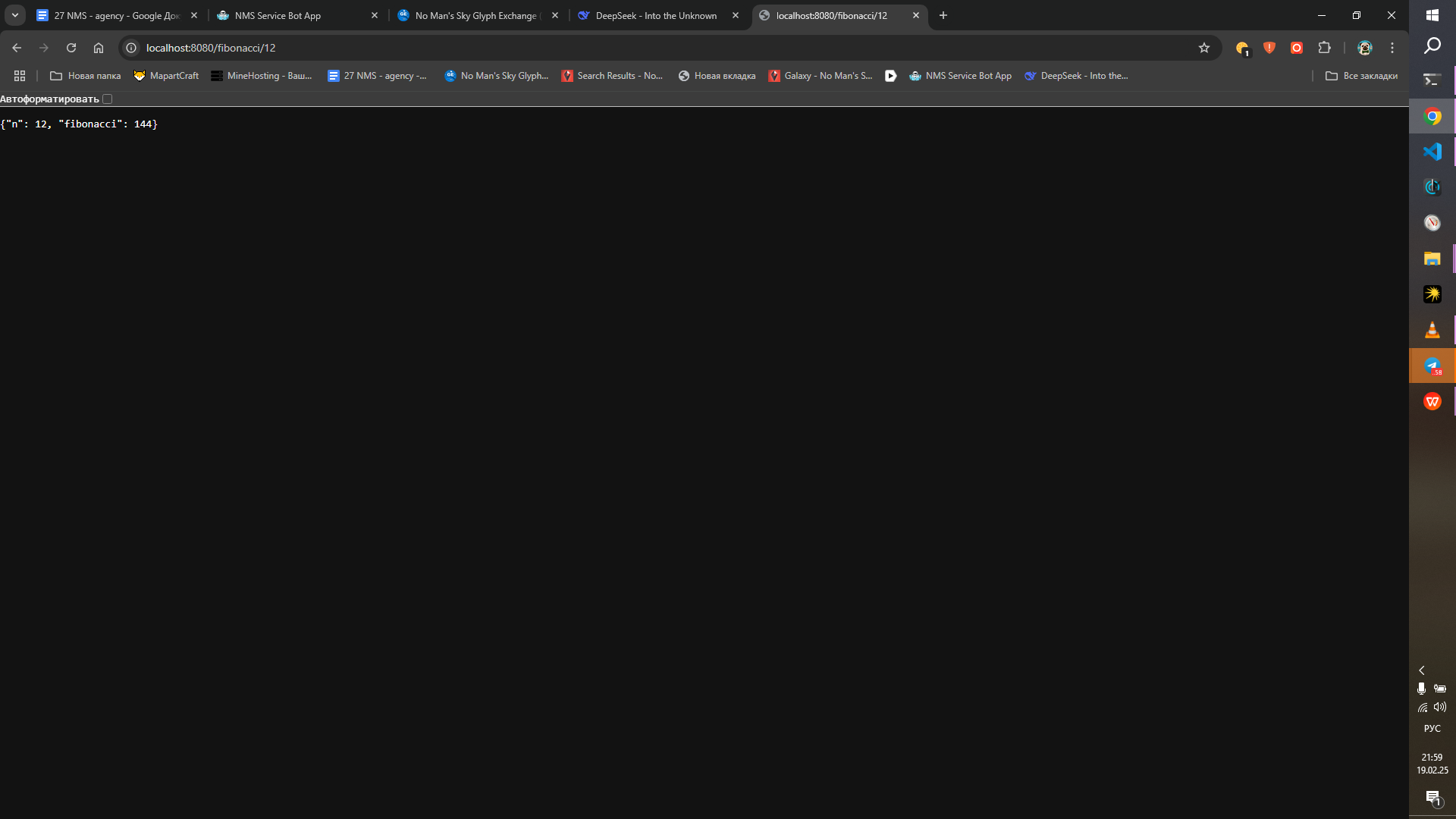
Запуск сервиса

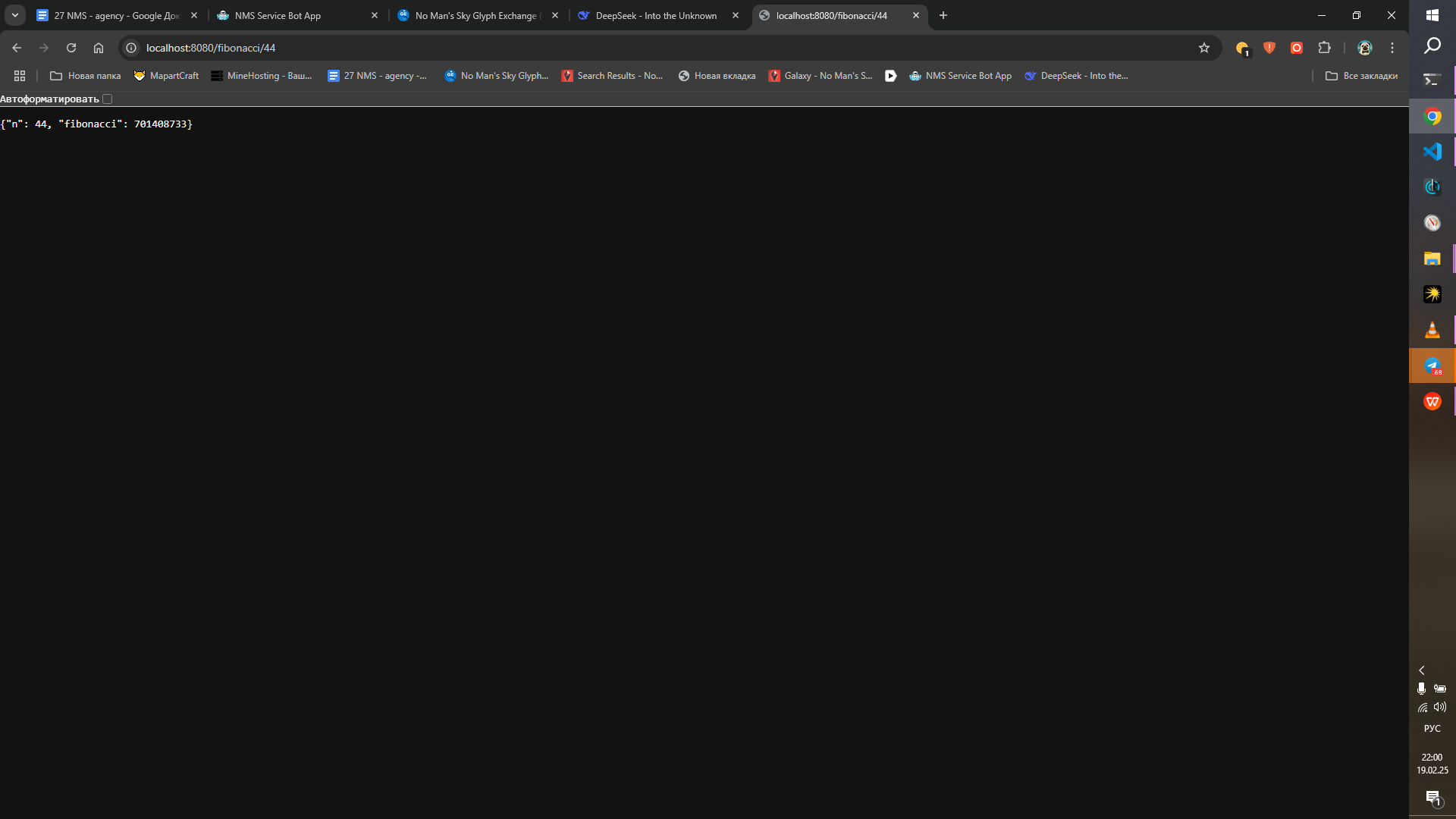
  
Проверка работоспособности



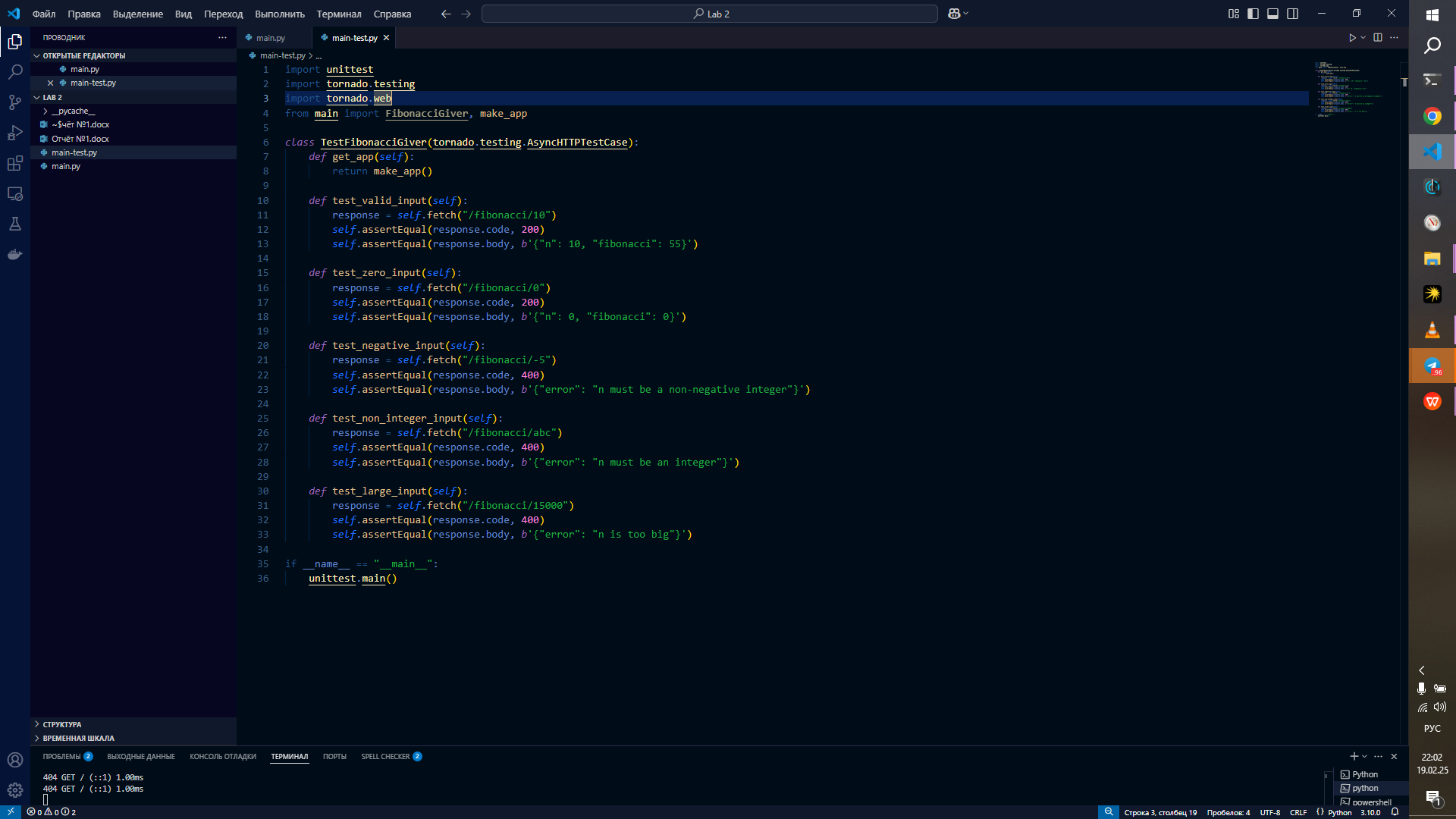


Правильный ввод





Файл с тестами



Листинг кода

import unittest

import tornado.testing

import tornado.web

from main import FibonacciGiver, make\_app

*class* TestFibonacciGiver(tornado.testing.AsyncHTTPTestCase):

*def* get\_app(*self*):

        return make\_app()

*def* test\_valid\_input(*self*):

        response = *self*.fetch("/fibonacci/10")

*self*.assertEqual(response.code, 200)

*self*.assertEqual(response.body, *b*'{"n": 10, "fibonacci": 55}')

*def* test\_zero\_input(*self*):

        response = *self*.fetch("/fibonacci/0")

*self*.assertEqual(response.code, 200)

*self*.assertEqual(response.body, *b*'{"n": 0, "fibonacci": 0}')

*def* test\_negative\_input(*self*):

        response = *self*.fetch("/fibonacci/-5")

*self*.assertEqual(response.code, 400)

*self*.assertEqual(response.body, *b*'{"error": "n must be a non-negative integer"}')

*def* test\_non\_integer\_input(*self*):

        response = *self*.fetch("/fibonacci/abc")

*self*.assertEqual(response.code, 400)

*self*.assertEqual(response.body, *b*'{"error": "n must be an integer"}')

*def* test\_large\_input(*self*):

        response = *self*.fetch("/fibonacci/15000")

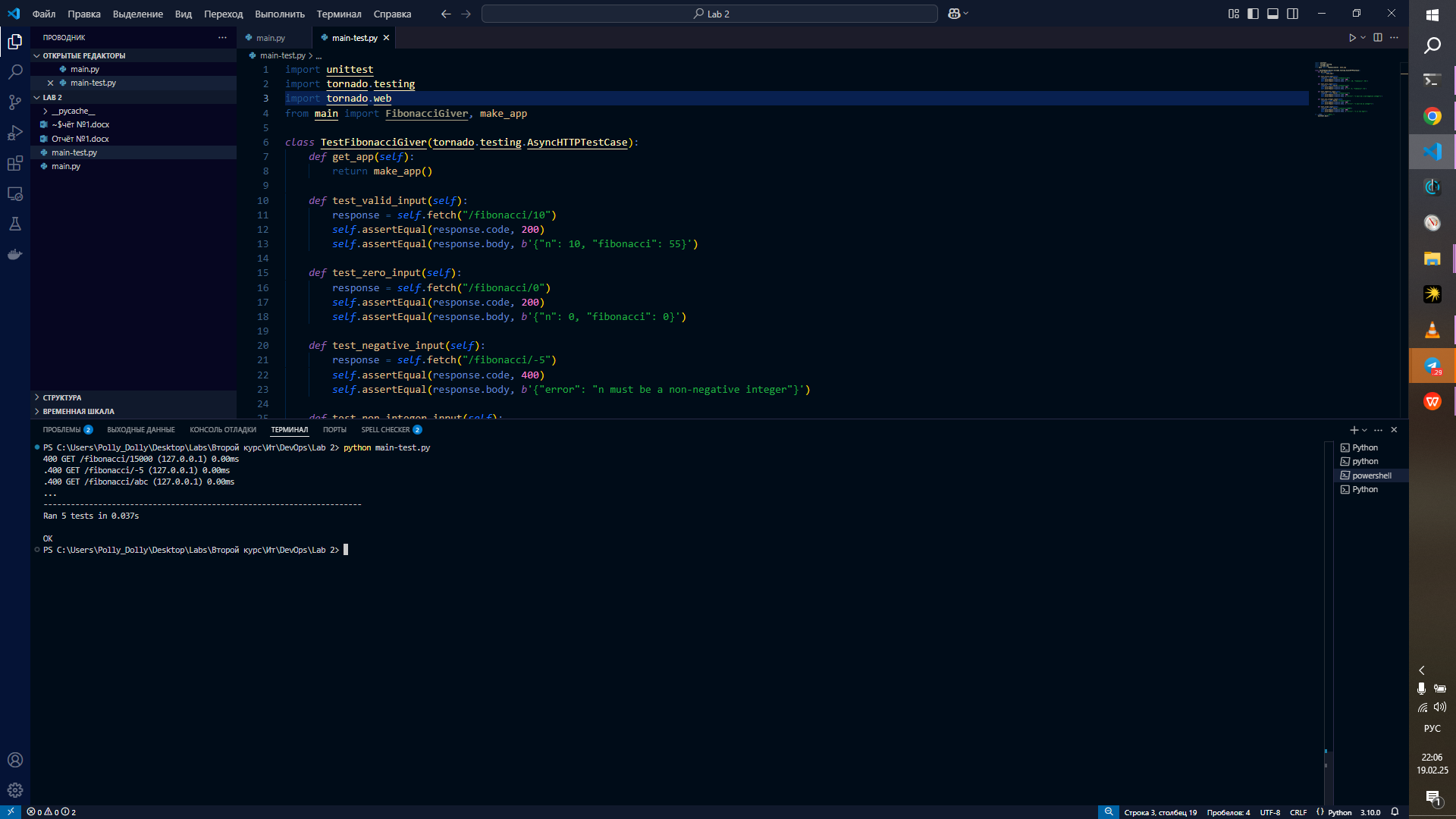
*self*.assertEqual(response.code, 400)

*self*.assertEqual(response.body, *b*'{"error": "n is too big"}')

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

Результаты тестирования:



## Ссылки:

# [What is a REST API?](https://www.redhat.com/en/topics/api/what-is-a-rest-api" \l ":~:text=A%20REST%20API%20(also%20known,by%20computer%20scientist%20Roy%20Fielding.)

# [Введение в REST API — RESTful веб-сервисы](https://habr.com/ru/post/483202/)

# [Общение микросервисов: REST, JSON, GraphQL или gRPC (Их история появления и зачем они нужны) видео](https://youtu.be/o7ogFTMJW1A)

1. [Документация по FastAPI](https://fastapi.tiangolo.com/tutorial/first-steps/)
2. [Документация по Flask](https://flask.palletsprojects.com/en/2.0.x/tutorial/layout/)
3. [Документация по Tornado](https://www.tornadoweb.org/en/stable/)