САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Web-разработка

Отчет

Лабораторная работа №1

Выполнил:

Егоров Мичил

Группа

К33401

Проверил:
Говоров А. И.

Санкт-Петербург

2021 г.

Задача

Оладеть практическими навыками и умениями реализации web-серверов и использования сокетов.

Ход работы

Определим базовый класс для сервера (файл base.py), чтобы не писать каждый раз базовый функционал:

- Базовая настройка сервера: адрес, порт, закрепление.
- Распараллеливание потоков для запросов пользователей
- Отправка строчного сообщения пользователю
- Запуск сервера

Также определим методы:

- _handle – обработка запроса пользователя
- _recv_data – получение и подготовка запроса

1. Реализовать клиентскую и серверную часть приложения. Клиент отсылает серверу сообщение «Hello, server». Сообщение должно отразиться на стороне сервера. Сервер в ответ отсылает клиенту сообщение «Hello, client». Сообщение должно отобразиться у клиента.

```python
import socket
import logging
from base import BaseServer


logger = logging.getLogger(__name__)


class HelloWorldServer(BaseServer):

    def _handle(self, client_socket: socket.socket, address, data: str):
        print("Data: ", data)
        logger.info("data: %s", data)
        client_socket.send(b"Hello, Client")


if __name__ == "__main__":
    server = HelloWorldServer(address="127.0.0.1", port=8000)
    server.loop()
```

Рисунок 1. Сервер "Hello world"

```python
import socket


if __name__ == "__main__":
    conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    conn.connect(("127.0.0.1", 8000))
    conn.send(b"Hello, Server")
    answer = conn.recv(16340)
    print(answer)
    conn.close()
```

Рисунок 2. Клиент "Hello world"

2. Реализовать клиентскую и серверную часть приложения. Клиент запрашивает у сервера выполнение математической операции, параметры, которые вводятся с клавиатуры. Сервер обрабатывает полученные данные и возвращает результат клиенту.

Зададим интерфейс пользователя: нужно будет отправлять строки вида «op_name op_arguments», где op_name – название задачи, которую нужно решить (solve_pythagorean, solve_quadratic_equation, solve_trapezoid_area, solve_parallelogram_area) и нужные аргументы (Рис. 3)

```python
import socket

TASKS = [
    'pythagorean 3 4',
    'quadratic_equation 1 2 1',
    'trapezoid_area 4 5 3',
    'parallelogram_area 4 2',
    '-1 2 4 asdasd',
    'pythagorean ',
    'quadratic_equation 0 2 1',
    'trapezoid_area -4 5 0',
    'trapezoid_area 4 asdg',
    'trapezoid_area 4',
    'parallelogram_area 4 0',
]

if __name__ == "__main__":
    for task in TASKS:
        conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        conn.connect(("127.0.0.1", 8000))
        conn.send(task.encode('utf-8'))
        server_answer = conn.recv(16348).decode('utf-8')
        print(task, ':', server_answer)
        conn.close()
```

Рисунок 3. Отправка задач на сервер.

Нужно будет переопределить метод _handle и в зависимости от запроса пользователя, например, решение задачи квадратичного уравнения (Рис. 4), вызывать нужный метод класса и отправить результат пользователю (Рис. 5)

```python
def solve_quadratic_equation(self, a: float, b: float, c: float) -> str:
    discriminant = b ** 2 - 4 * a * c

    if a == 0:
        raise ValueError("Уравнение не является квадратичным. ")

    x1 = (b - discriminant ** 0.5) / (2 * a)
    x2 = (b + discriminant ** 0.5) / (2 * a)

    return "Решения квадратного уравнения {}*x^2+({})*x+({}): {} и {}".format(a, b, c, x1, x2)
```

Рисунок 4. Решение квадратичного уравнения.

```python
def _handle(self, client_socket: socket.socket, address: Tuple[str, int], data: str) -> None:
    op_name, *args = data.split()
    method = 'solve_' + op_name

    if not hasattr(self, method):
        raise ValueError(f'Метода {method} не существует')

    handler = getattr(self, method)

    try:
        args = list(map(float, args))
        answer = handler(*args)
    except (TypeError, ValueError):
        answer = 'Неверный формат входных данных'
    except Exception as e:
        answer = repr(e)

    self._send_message(client_socket, answer)
```

Рисунок 5. Обработка запроса пользователя.

3. Необходимо написать простой web-сервер для обработки GET и POST http запросов средствами Python и библиотеки socket.

Определим базу данных, где будем хранить предметы (Рис. 6)

```python
from typing import Iterable


class DataBase:
    storage = {}

    def create(self, name: str, value: dict) -> None:
        self.storage[name] = value

    def get(self, name: str) -> dict:
        if name in self.storage:
            return self.storage[name]

        raise ValueError("Ключа %s нет базе данных")

    def all(self) -> Iterable[dict]:
        for key, value in self.storage.items():
            yield value

    def filter(self, **search_parameters) -> dict:
        for key, data in self.storage.items():
            for parameter, value in search_parameters.items():
                if parameter in self.storage and self.storage[parameter] == value:
                    yield data
```

Рисунок 6. Класс базы данных.

При вызове GET запроса отберем все имеющиеся предметы и вернем пользователю, а при POST запросе сохраним в базу, если необходимо (Рис. 7)

```
def _GET_method(self, data: dict) -> None:
    subjects = list(db.all())
    return HTMLResponse('subjects.html', {'subjects': subjects_to_html(subjects)})

def _POST_method(self, data: dict) -> None:
    error_message = ''
    grade = data['grade']

    if not grade.isdigit():
        error_message = "Неверный формат оценки"
    elif 0 < int(data['grade']) <= 5:
        db.create(data['subject_name'], {'name': data['subject_name'], 'grade': data['grade']})
    else:
        error_message = "Оценка должна быть в интервале (0, 5]"

    subjects = list(db.all())

    return HTMLResponse('subjects.html', {
        'subjects': subjects_to_html(subjects),
        'error_message': error_message
    })
```

Рисунок 7. Методы GET и POST.

При вызове _handle будем смотреть на метод запроса и вызывать советующий функционал, аналогично логике в рисунке 5. Переопределим метод _send_message, чтобы сначала привести строковое сообщение в нужный вид: определить заголовок и тело запроса HTTP (Рис. 8).

```
def _send_message(self, client_socket: socket.socket, message: Union[str, Response]) -> None:
    if isinstance(message, str):
        return self._send_message(
            client_socket,
            HTMLResponse('bad.html', {'message': message})
        )

    message = str(message)
    super(TasksServer, self)._send_message(client_socket, message)
```

Рисунок 8. Переопределение метода отправки сообщения для пользователя.

Вспомогательные классы и функции лежат в файле 2_html/server.py

4. Реализовать двухпользовательский или многопользовательский чат. Реализация многопользовательского часа позволяет получить максимальное количество

Определим интерфейс пользователя: будем отправлять сообщения вида room_id=1&message=hello, world! I'm user 3!, т.е. говорим в какую комнату, какое сообщения отправить (Рис. 9).

```python
import socket

if __name__ == "__main__":
    sockets = [socket.socket(socket.AF_INET, socket.SOCK_STREAM) for _ in range(3)]
    [sock.connect(("127.0.0.1", 8000)) for sock in sockets]

    user1, user2, user3 = sockets

    user1.send(b"room_id=1&message=hello, world! I'm user 1!")
    print(user1.recv(1024))
    user2.send(b"room_id=1&message=hello, world! I'm user 2!")
    print(user1.recv(1024))
    user3.send(b"room_id=1&message=hello, world! I'm user 3!")
    print(user1.recv(1024))
    print(user2.recv(1024))

    user1.close()
    user2.close()
    user3.close()
```

Рисунок 9. Интерфейс многопользовательского чата.

При получении запроса сервер будет распределять пользовательские сокеты по нужным комнатам и при отправке сообщения в эту комнату будет им всем отсылать запрос (Рис. 10).

```python
def _change_client_room(self, address: Tuple[str, int], room_id: str) -> None:
    if address in self.clients_rooms and self.clients_rooms[address] == room_id:
        return

    if self.clients_rooms[address] in self.rooms:
        self.rooms[room_id].discard(address)

    self.clients_rooms[address] = room_id
    self.rooms[room_id].add(address)

def _close_client(self, client_socket: socket.socket, address: Tuple[str, int]) -> None:
    if address not in self.clients_rooms:
        return

    client_room_id = self.clients_rooms[address]
    self.rooms[client_room_id].discard(address)
    self.clients_rooms.pop(address)
    self.clients_sockets.pop(address)

    super()._close_client(client_socket, address)

def _handle(self, client_socket: socket.SocketIO, address: Tuple[str, int], data: str) -> None:
    body = self._prepare_data(data)
    self.clients_sockets[address] = client_socket
    self._change_client_room(address, body['room_id'])
    for client_address in self.rooms[body['room_id']]:
        try:
            sock = self.clients_sockets[client_address]
            self._send_message(sock, body['message'])
        except Exception as e:
            logger.warning("Сообщение не отправилось: %s %s", client_address, repr(e))
```

Рисунок 10. Сервер многопользовательского чата.

Вывод:

Научились работать с встроенной функцией socket на языке программирования Python. Изучили структуру и синтаксис HTTP, написали свой собственный многопользовательский чат и сервис для хранения оценок по предметам.