|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКИЕ РАБОТЫ**

по дисциплине «Архитектура клиент-серверных приложений»

**Студент группы** ИКБО-01-20 Стецюк О.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практических работ**  Благирев М.М.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Москва 2022

**ОГЛАВЛЕНИЕ**

[Практическая работа 1 3](#_Toc118962412)

[Практическая работа 2 6](#_Toc118962413)

[Практическая работа 1 9](#_Toc118962414)

[Практическая работа 1 12](#_Toc118962415)

[Практическая работа 1 15](#_Toc118962416)

[Практическая работа 1 18](#_Toc118962417)

[Список использованной литературы 22](#_Toc118962418)

# Практическая работа 1

**Задание**

Создать многопоточное приложение, печатающее PING PONG, поочередно использующее два потока

**Ход работы**

В качестве языка разработки в данной работе и всех последующих было решено использовать Python. В данной работе используется библиотека threading.

Переключение между потоками происходит путем получения мьютекса на выполнение одной из функций. Выполнений одной из функций предполагает разрешение выполнения другой (Рисунок 1.1)

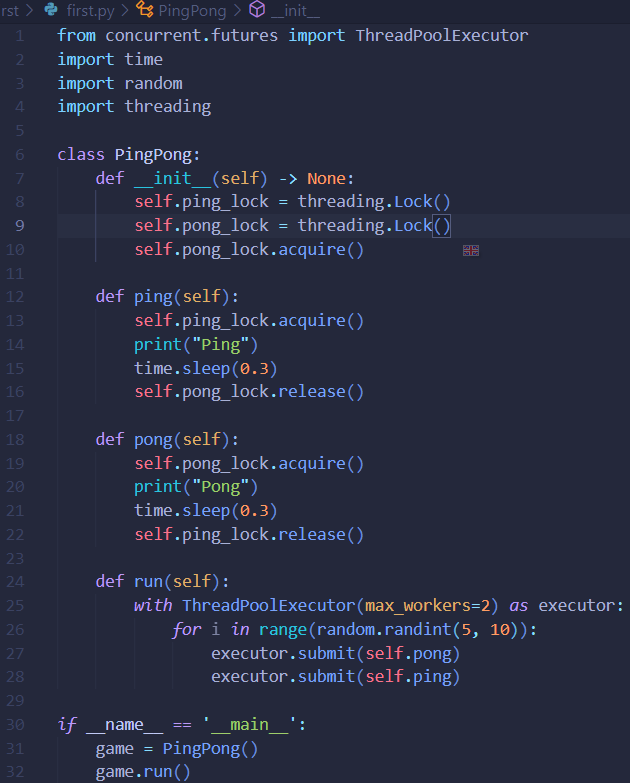


Рисунок 1.1 – Исходный код

Результат выполнения кода показан на Рисунке 1.2

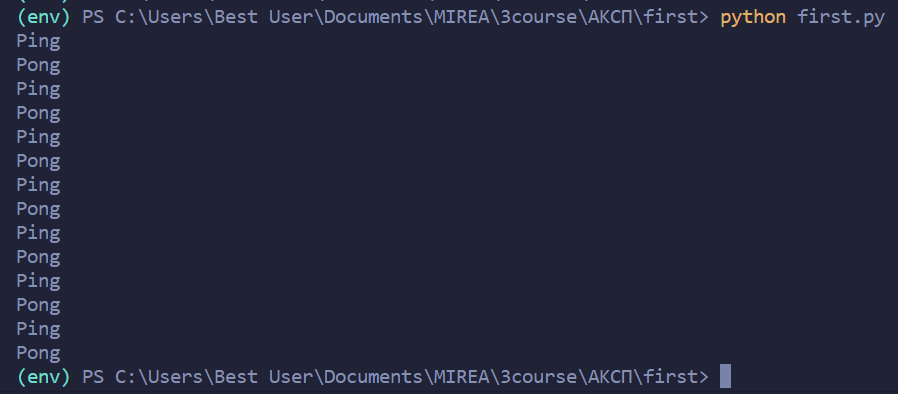


Рисунок 1.2 -- Результат

**Вывод**

В ходе практической работы были получены навыки использования нескольких потоков в разработке приложения.

# Практическая работа 2

**Задание**

Получить навыки разработки RMI (Remote Method Invocation) и реализовать программу, решающую квадратные уравнения.

**Ход работы**

В данной работе используется библиотека Pyro4 (Python Remote Objects).

Серверная часть приложения состоит из класса для решения уравнения, который мы в качестве интерфейса предоставляем в RMI. Также в файле server.py запускается демон RMI, в который и добавляется интерфейс (Рисунок 2.1)

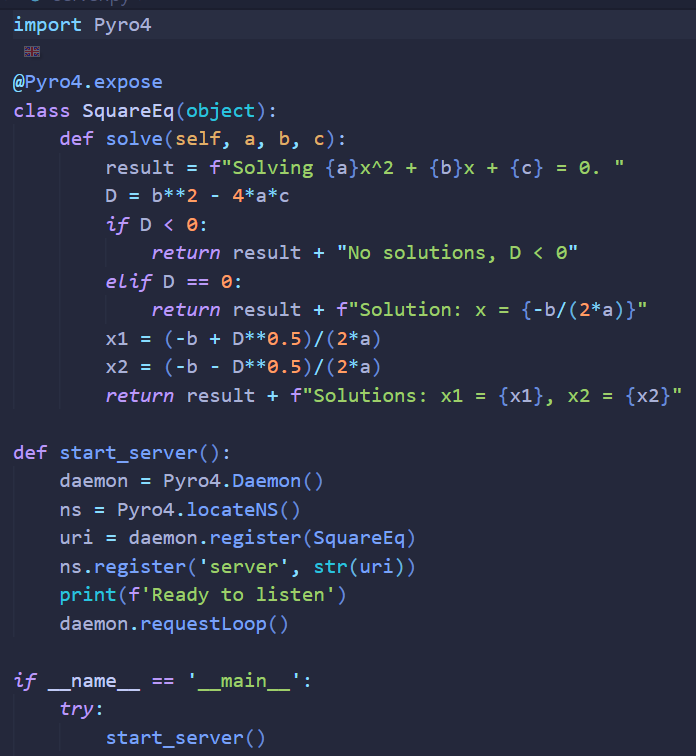


Рисунок 2.1 – Серверная часть

В клиентской части можно получить доступ к интерфейсу, поэтому в ней просто задаются случайные входные параметры для теста программы (Рисунок 2.2)

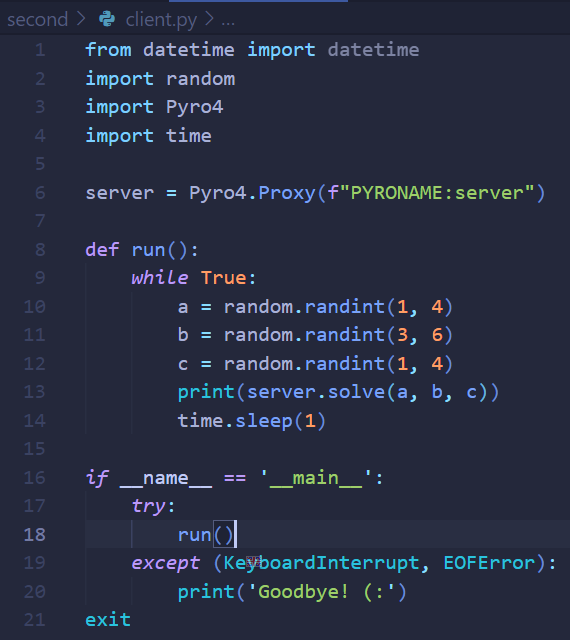


Рисунок 2.2 – Клиентская часть

Далее необходимо запустить сервер сопоставления наименований объектов и их URI (Рисунок 2.3)

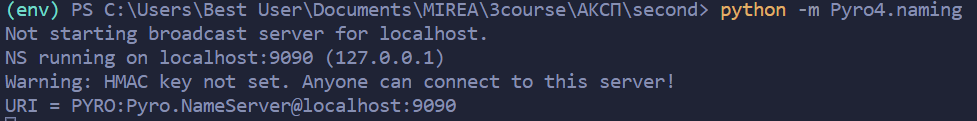


Рисунок 2.3 – Запуск naming-сервера

Запустим основной сервер (Рисунок 2.4)

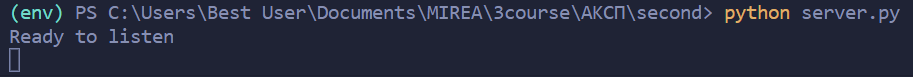


Рисунок 2.4 – Запуск сервера

Запустим клиентскую часть (Рисунок 2.5)

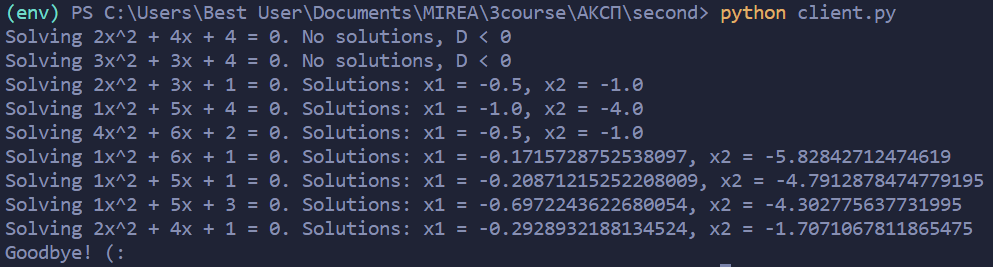


Рисунок 2.5 – Запуск клиентской части

**Вывод**

В ходе практической работы были получены навыки использования RMI на языке Python

# Практическая работа 3

**Задание**

Ознакомиться с сокетами и написать приложение, накапливающее сообщения в течение 5 секунд и устраивающее их рассылку всем клиентам

**Ход работы**

В данной работе было принято решение разделить программу на три сущности: клиент-отправитель, клиент и сервер.

Клиент отправитель в блокирующем режиме принимает ввод с клавиатуры и отправляет на сервер (Рисунок 3.1)



Рисунок 3.1 – Клиент-отправитель

Простой клиент во многом похож на отправитель, только он слушает сообщения от сервера вместо ввода с клавиатуры (Рисунок 3.2)

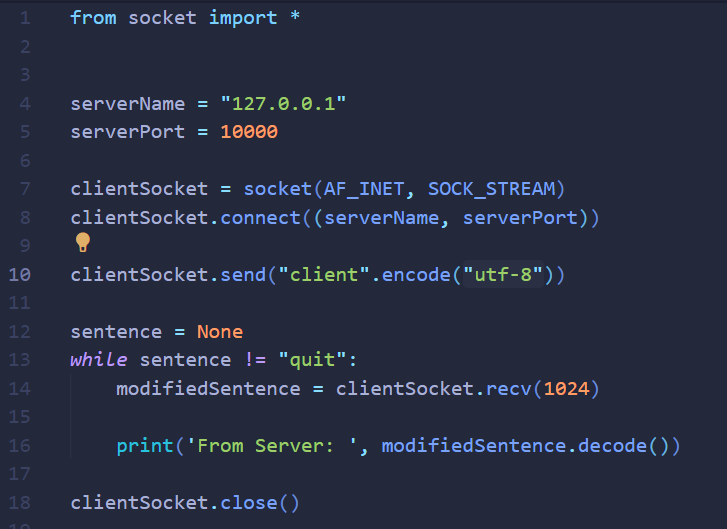


Рисунок 3.2 – Простой клиент

Серверная часть реализует класс Server. В нем прописан метод обработки клиента, по которому сервер определяет, слушать ли с него ввод или отправлять ему сообщения (Рисунок 3.3)

Рисунок 3.3 – Серверная часть

Также в этом классе есть метод массовой рассылки (Рисунок 3.4)



Рисунок 3.4 – Метод рассылки

Запустим сервер (Рисунок 3.5)

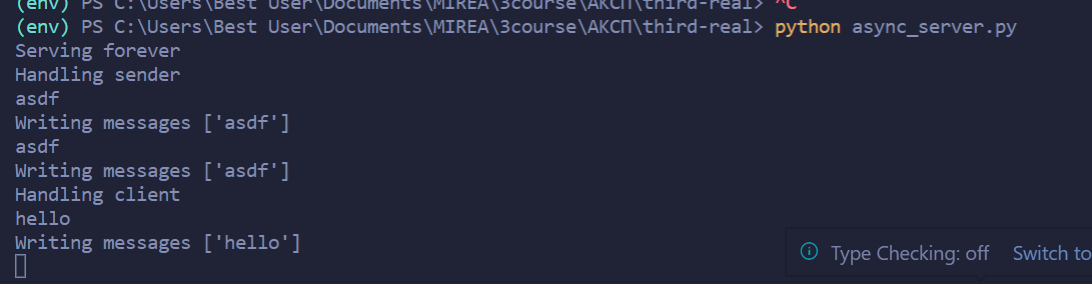


Рисунок 3.5 – Работа сервера

Запустим клиент-отправитель (Рисунок 3.6)

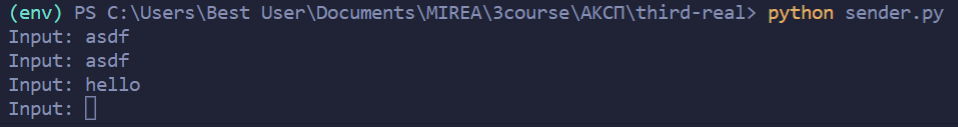


Рисунок 3.6 – Клиент-отправитель

Запустим клиент (Рисунок 3.7)

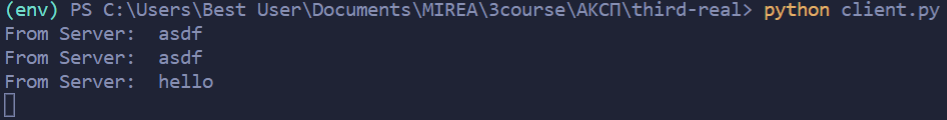


Рисунок 3.7 -- Клиент

**Вывод**

В ходе практической работы были получены навыки работы с сокетами

# Практическая работа 4

**Задание**

Получить навыки работы с вебсокетами и реализовать приложение

**Ход работы**

В данной работе используется фреймворк FastAPI. Было реализовано приложение, аналогичное приложению в предыдущей работе.

Серверная часть использует менеджер соединений, содержащий списки сообщений и соединений, и обрабатывающий их (Рисунок 4.1)

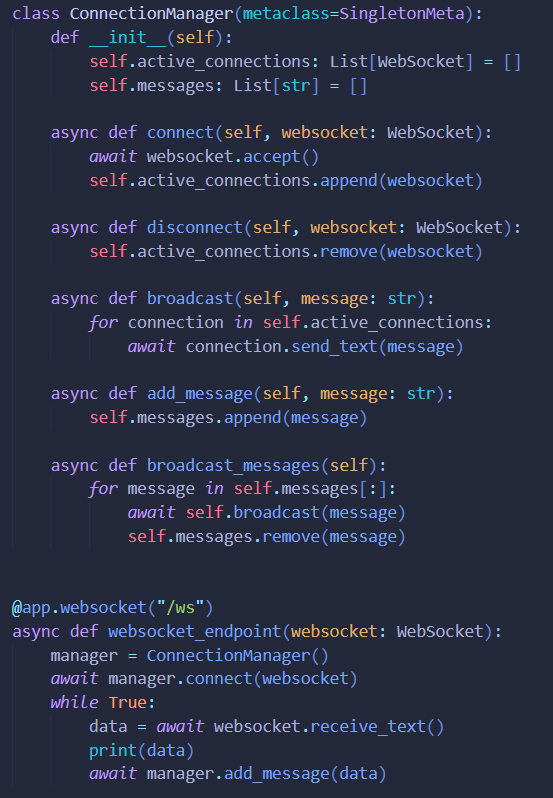


Рисунок 4.1 – Серверная часть

Клиент содержит форму отправки данных на вебсокет. Получение данных показано на Рисунках 4.2 - 4.4

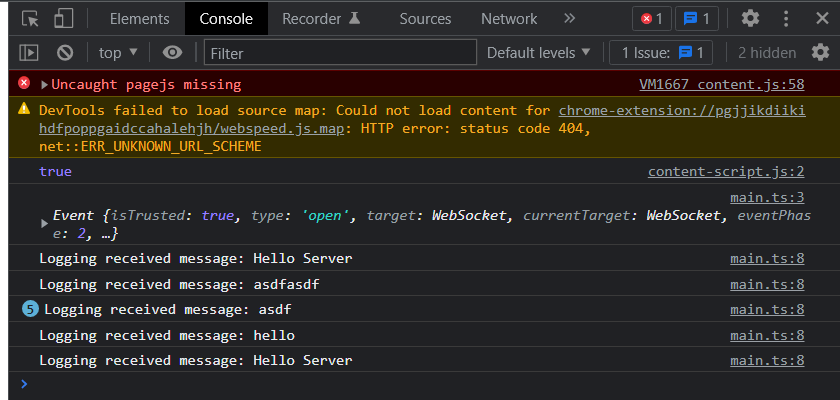


Рисунок 4.2 – Работа клиента

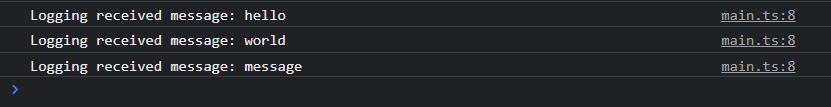


Рисунок 4.3 – Работа клиента

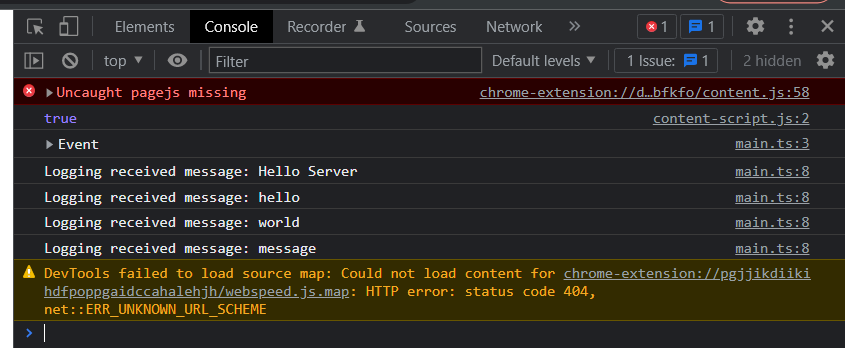


Рисунок 4.4 – Работа клиента

Работа сервера показана на Рисунках 4.5 – 4.6

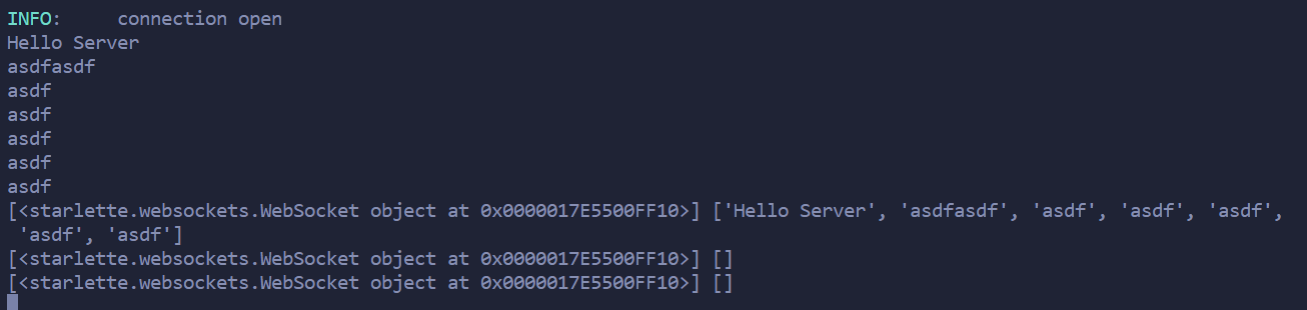


Рисунок 4.5 – Работа сервера

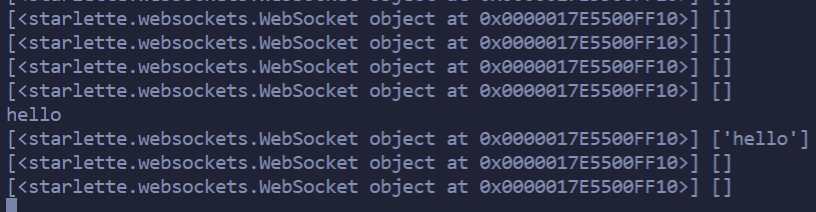


Рисунок 4.6 – Работа сервера

**Вывод**

В ходе практической работы были получены навыки использования вебсокетов в приложениях

# Практическая работа 5

**Задание**

Реализовать асинхронное взаимодействие через вебсокеты.

**Ход работы**

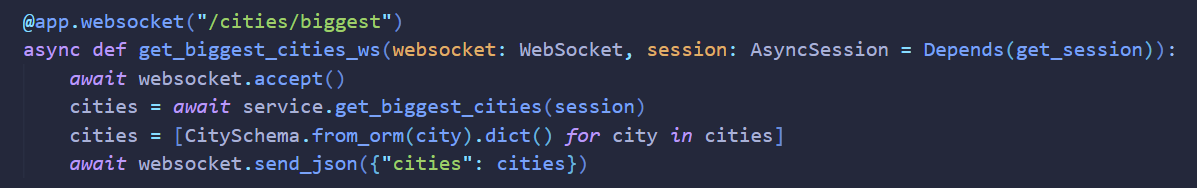
Для работы приложения был реализован вебсокет-эндпоинт, который вызывает сервис работы с базой данных (Рисунок 5.1)

Рисунок 5.1 – Вебсокет-эндпоинт

Клиентская часть приложения делает запрос на этот эндпоинт (Рисунок 5.2)

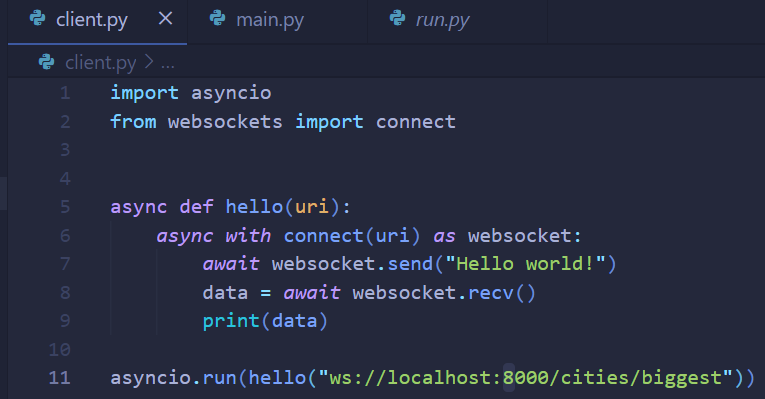


Рисунок 5.2 – Клиент

Работа клиента показана на Рисунке 5.3

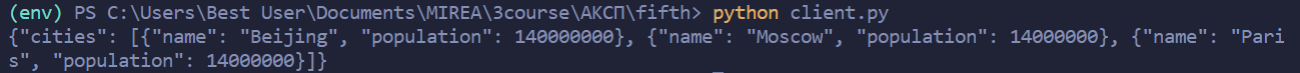


Рисунок 5.3 – Работа клиента

Также HTTP запрос на аналогичный (не вебсокет) эндпоинт возвращает те же данные (Рисунок 5.4)

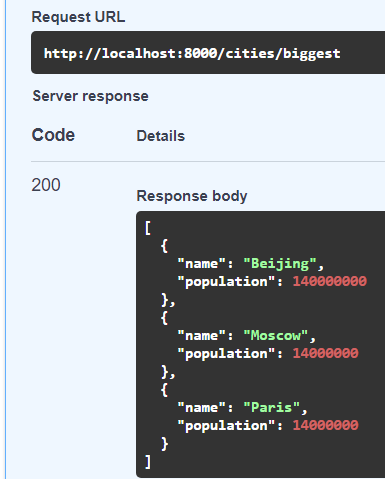


Рисунок 5.4 – Вебсокет-эндпоинт

**Вывод**

В ходе практической работы были получены навыки разработки асинхронных приложений с вебсокетами.

# Практическая работа 7

**Цель работы**

Целью данной практической работы является знакомство обучающихся с набирающим популярность современным подходом к проектированию и реализации API на основе графовых моделей и с реализующей данный подход технологией на основе спецификации GraphQL.

**Выполнение практической работы**

Задание:

Используя теоретические сведения из данной практической работы, открытые интернет-источники, официальную документацию по GraphQL необходимо, с использованием SDL создать схему, реализовать сервер и клиента GraphQL для следующих бизнес-задач (по выбору)

Создание приложения для хранения списка книг в библиотеке. Схема должна определять требуемые типы данных с полями, возвращающими определённые данные. Должны быть определены поля: name (название книги), genre(жанр книги), id (уникальный идентификатор книги), name (ФИО автора книги). Дополнительные поля и соответствующие типы, если они будут нужны для решения данной задачи, определяются самостоятельно

## Решение:

Для выполнения данной работы была выбрана библиотека “graphql-request” и React, с помощью которых был создан следующий интерфейс (Рисунок 7.1)

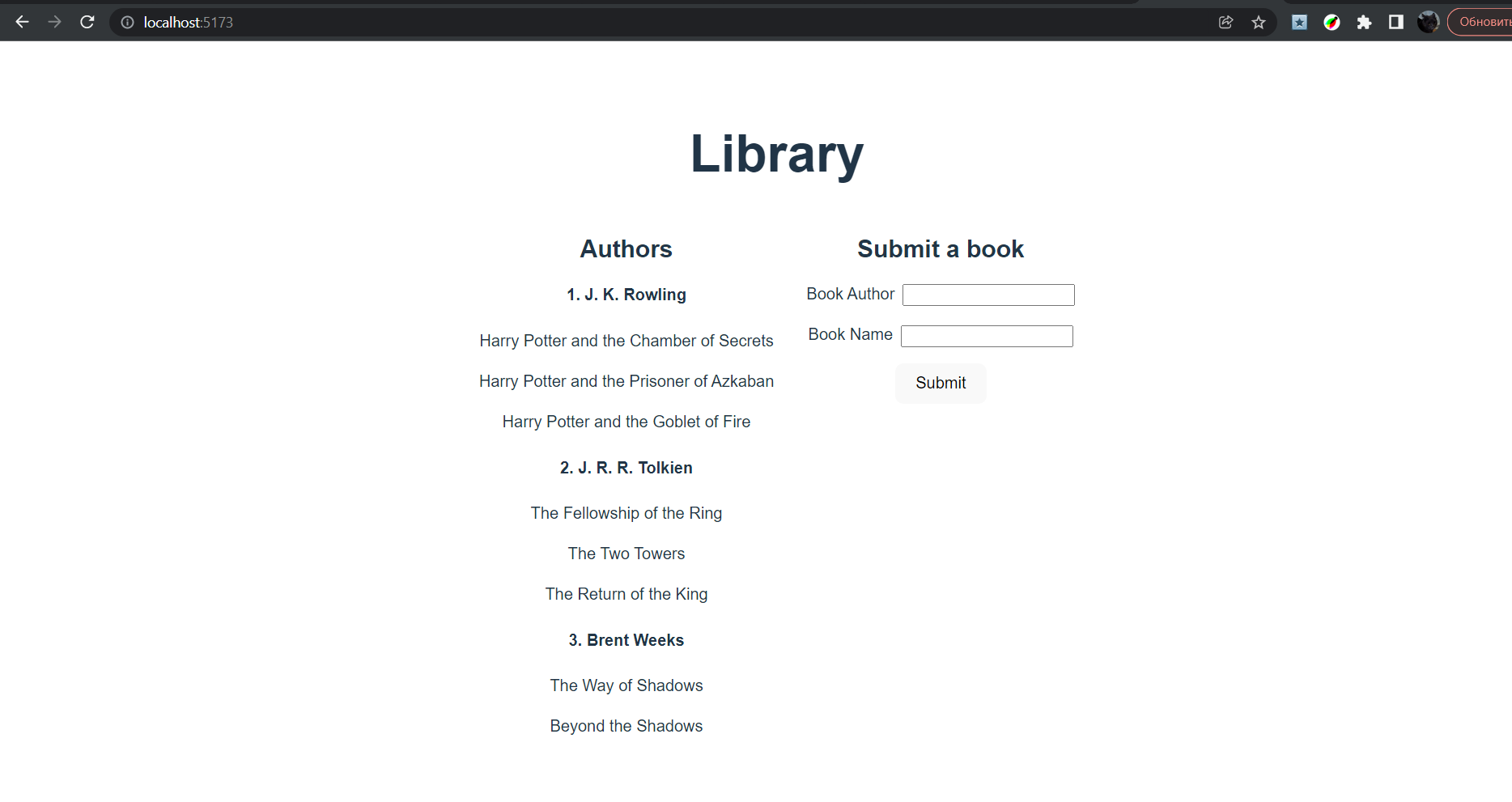


Рисунок 7.1 – UI решения.

Проверим работу сервиса добавив новую книгу. Если имя автора книги не присутствует в базе, создается новый, иначе книга добавляется к уже существующему автору по id.

Проверим добавление новой книги (Рисунок 7.1)

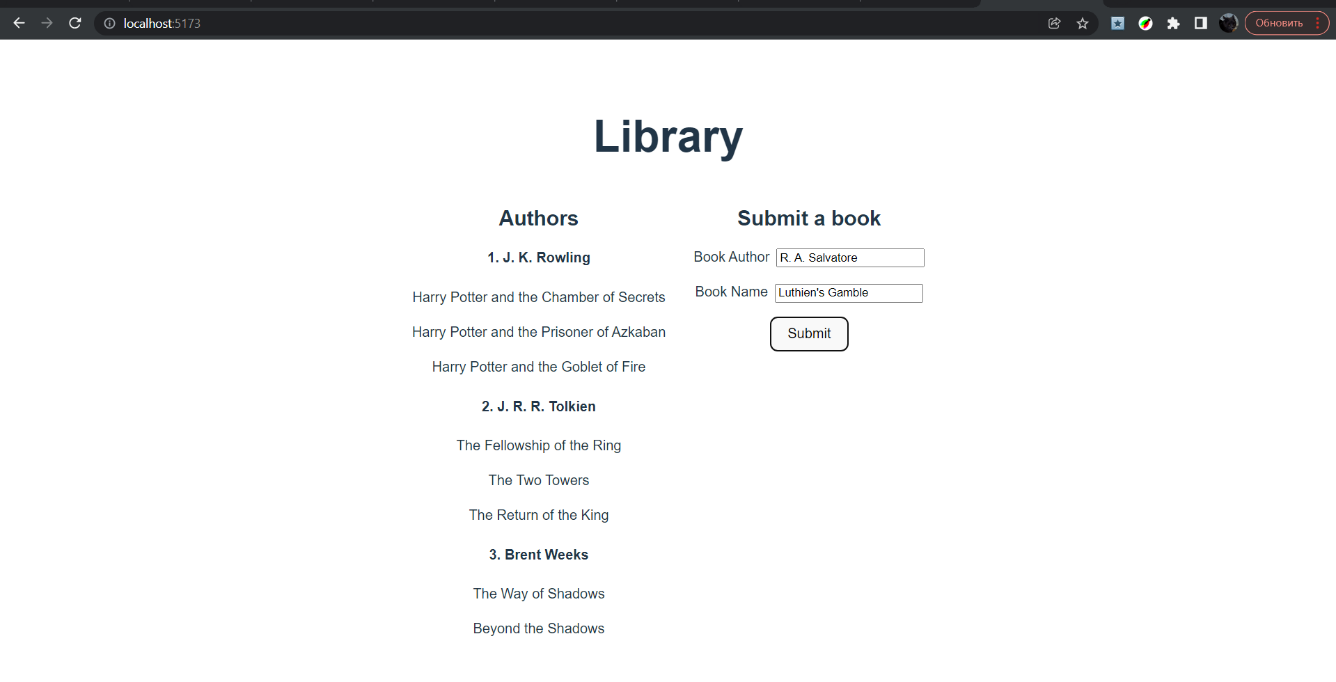


Рисунок 7.1 – Добавление новой книги

Проверим добавление новой книги к уже существующему автору (Рисунок 7.2)

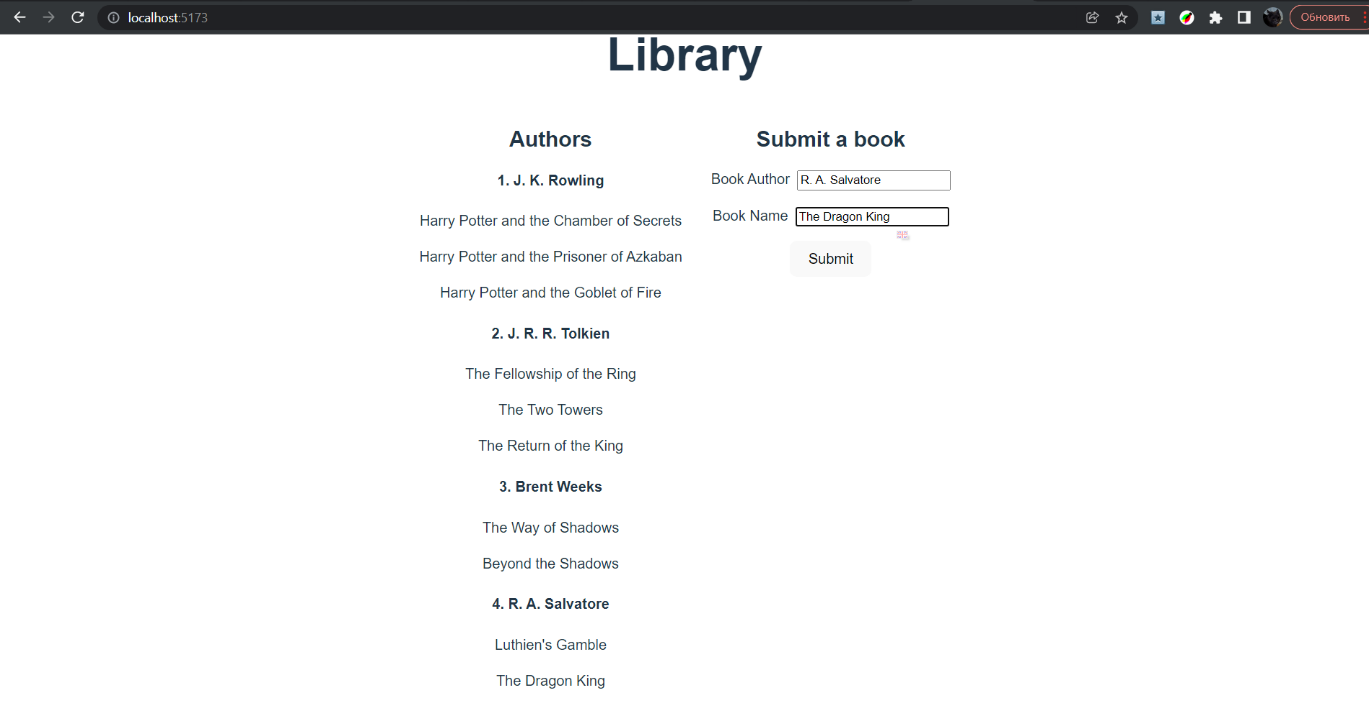


Рисунок 7.2 – Добавление новой книги к уже существующему автору

Для выполнения данных функций были созданы мутации graphql для добавления книги или автора (Рисунок 7.3)

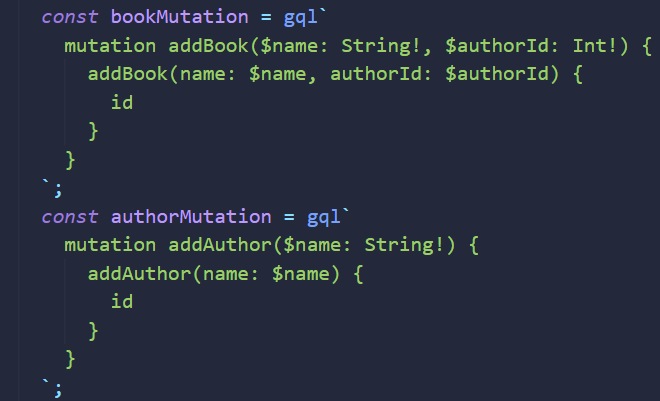


Рисунок 7.3 – Мутации graphql

Функция отправки запроса проверяет, не пустые ли поля для ввода значений, также она производит запрос на проверку существования автора в базе (Рисунок 7.4)

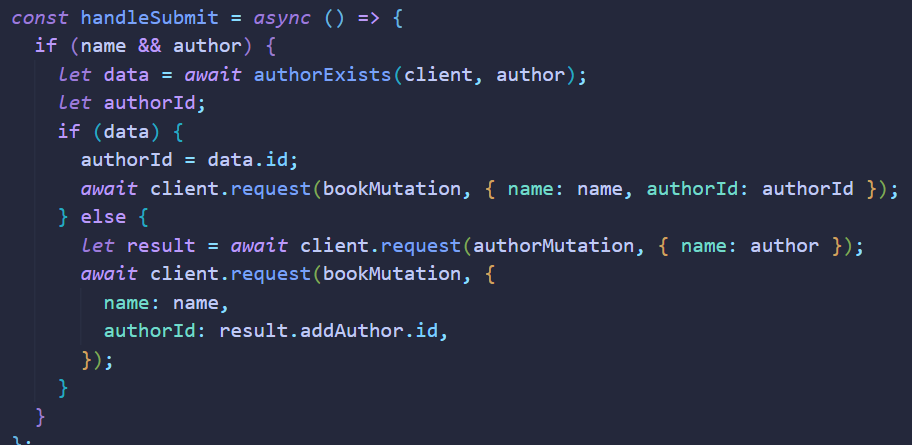


Рисунок 7.4 – Функция выполнения запроса.

**Выводы по работе**

В результате выполнения данной работы были получены навыки разработки API с использованием GraphQL.