Санкт-Петербургский государственный университет

Факультет прикладной математики – процессов управления

**Отчет №3**

по дисциплине «Функциональное программирование»

**Разработка асинхронного чат-сервера**

Автор работы: Дацык Р.В.

Группа: 22.Б15-пу

Преподаватель: Киямов Ж. У.

Санкт-Петербург, 2023

**Оглавление**

[**1.** **Цель работы** 3](#_Toc152720812)

[**2.** **Задачи** 3](#_Toc152720813)

[**3.** **Теоретическая часть** 3](#_Toc152720814)

[**4.** **Алгоритм метода** 3](#_Toc152720815)

[**5.** **Описание программы** 4](#_Toc152720816)

[**6.** **Рекомендации пользователя** 7](#_Toc152720817)

[**7.** **Рекомендации программиста** 7](#_Toc152720818)

[**8.** **Контрольный пример** 7](#_Toc152720819)

[**9.** **Вывод** 9](#_Toc152720820)

[**10.** **Список использованной литературы** 9](#_Toc152720821)

# **Цель работы**

Создать асинхронный чат-сервер, который может обслуживать множество клиентов одновременно и позволяет им обмениваться сообщениями в режиме реального времени.

# **Задачи**

Для достижения цели поставлены следующие задачи:

1. **Изучить возможности асинхронных библиотек**
2. **Создание асинхронного сервера**
3. **Разработка чат-комнат**
4. **Обеспечение работы личных сообщений**
5. **Обработка ошибок и исключений**

# **Теоретическая часть**

Для создания асинхронного чат-сервера на Python с использованием asyncio и websockets, необходимо глубоко понимать асинхронное программирование. Разработка эффективного протокола обмена сообщениями и обеспечение конфиденциальности через чат-комнаты являются ключевыми аспектами. Дополнительно важна обработка ошибок в асинхронном коде для стабильной работы сервера в различных сценариях. Эти концепции обеспечат правильную реализацию взаимодействия между клиентами и сервером в режиме реального времени.

# **Алгоритм метода**

Программа представляет собой асинхронный чат-сервер, способный обслуживать несколько клиентов одновременно с использованием библиотеки asyncio и websockets. Вот общий алгоритм программы:

1. **Инициализация:**
   * Установка хоста и порта для сервера.
2. **Запуск сервера:**
   * Создание объекта сервера.
   * Запуск сервера с использованием библиотеки websockets.
3. **Ожидание подключений:**
   * Сервер ожидает входящих подключений от клиентов.
4. **Обработка клиента:**
   * При подключении нового клиента вызывается метод handle\_client.
   * Получение уникального имени и названия комнаты от клиента.
   * Добавление информации о клиенте в словарь clients.
5. **Обмен сообщениями:**
   * Асинхронный прием и отправка сообщений между сервером и клиентами.
   * Обработка команд пользователя, таких как создание/присоединение к комнатам, отправка личных сообщений и т. д.
6. **Обработка комнат:**
   * Реализация функционала создания и присоединения к комнатам.
   * Рассылка сообщений только в пределах комнаты.
7. **Обработка личных сообщений:**
   * Отправка личных сообщений между пользователями.
8. **Завершение работы клиента:**
   * Обработка закрытия соединения клиента.
   * Удаление информации о клиенте из словаря clients.
   * Рассылка уведомления о выходе клиента из комнаты.

# **Описание программы**

Программа написана на Python 3.10 с использованием библиотеки websockets для создания асинхронного чат-сервера. Основные компоненты включают в себя класс ChatServer, предоставляющий функции обработки клиентов и сообщений, а также методы для управления комнатами и обмена личными сообщениями.

Архитектура программы обеспечивает асинхронное взаимодействие между клиентами и сервером, эффективную обработку множественных подключений через многозадачность. Реализованы функции создания и присоединения к чат-комнатам, обмена текстовыми сообщениями и отправки личных сообщений через команду /whisper. Обработка ошибок и исключений в коде направлена на поддержание стабильности сервера в различных сценариях использования.

Программа предоставляет простой и гибкий механизм для множественных пользователей для общения в реальном времени в рамках чат-комнат, обеспечивая безопасность и эффективное использование ресурсов.

Таблица 5.1. Описание методов

|  |  |
| --- | --- |
| Название функции | Назначение функции |
| \_\_init\_\_(self, host, port) | Инициализирует объект ChatServer с указанным хостом и портом, создает словари для хранения информации о клиентах и комнатах. |
| handle\_client(self, websocket: websockets.WebSocketServerProtocol) | Обрабатывает подключение нового клиента, получает уникальное имя и название комнаты, добавляет информацию о клиенте и асинхронно обрабатывает сообщения от клиента. |
| create\_room(self, client, room\_name, user\_name) | Создает новую чат-комнату, если не существует, и уведомляет всех участников. |
| join\_room(self, client, room\_name, user\_name) | Присоединяет клиента к существующей или новой комнате, уведомляет участников. |
| get\_unique\_client\_name(self, websocket) | Запрашивает уникальное имя от клиента через WebSocket и проверяет его уникальность. |
| get\_room\_name(self, websocket) | Запрашивает название комнаты от клиента, создает ее, если не существует, и добавляет клиента. |
| handle\_message(self, sender, message) | Обрабатывает сообщения от клиента в комнате, поддерживает команду /whisper для личных сообщений. |
| send\_private\_message(self, sender, target\_client, message) | Отправляет личное сообщение от отправителя к получателю. |
| handle\_room\_action(self, sender, data) | Обрабатывает действия пользователя в комнате, такие как создание и присоединение. |
| handle\_whisper(self, sender, data) | Обрабатывает личные сообщения (шептания) между пользователями. |
| is\_unique\_name(self, client\_name) | Проверяет уникальность имени пользователя. |
| broadcast(self, message, room\_name, exclude=None) | Рассылает сообщение всем участникам комнаты, исключая отправителя. |
| start\_server(self) | Запускает сервер с использованием библиотеки websockets. |

# **Рекомендации пользователя**

Для запуска программы достаточно перейти по ссылке ведущей к чат-серверу. Сначала в поле для ввода вписать логин пользователя, затем номер комнаты, куда пользователь хочет присоединиться. Далее поле ввода предназначено для набора сообщений в комнате. 2 поля ниже предназначены для личных сообщений: левое – выбор пользователя, правое – сообщение.

# **Рекомендации программиста**

Для запуска программы необходима 64-битная операционная система Windows, Linux или macOS. Для работы с кодом необходима среда разработки, совместимая с python 3.10. Также требуется установить следующие библиотеки: asyncio, websockets.

Исходный код программы доступен по ссылке:

<https://github.com/CapTopGrade/Functional_Programming/blob/main/%D0%A4%D0%9F/%D0%A4%D0%9F%20%D0%B0%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9%20%D1%87%D0%B0%D1%82/ACHAT%20server.py>

# **Контрольный пример**

В данном разделе представлен контрольный пример, демонстрирующий работу программы.

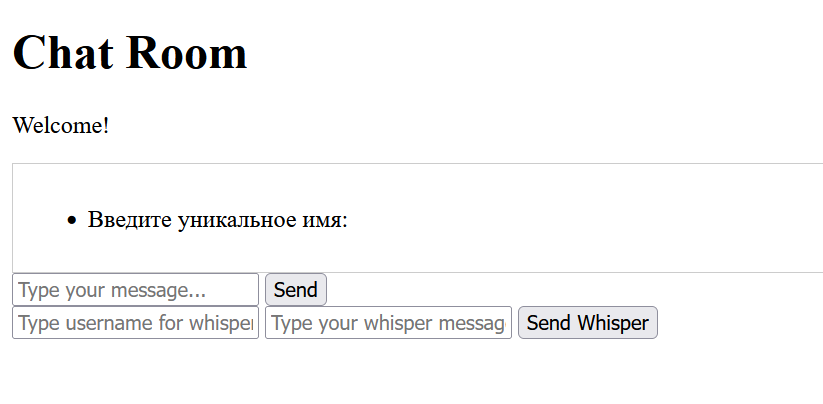


Рисунок 6.1, интерфейс программы

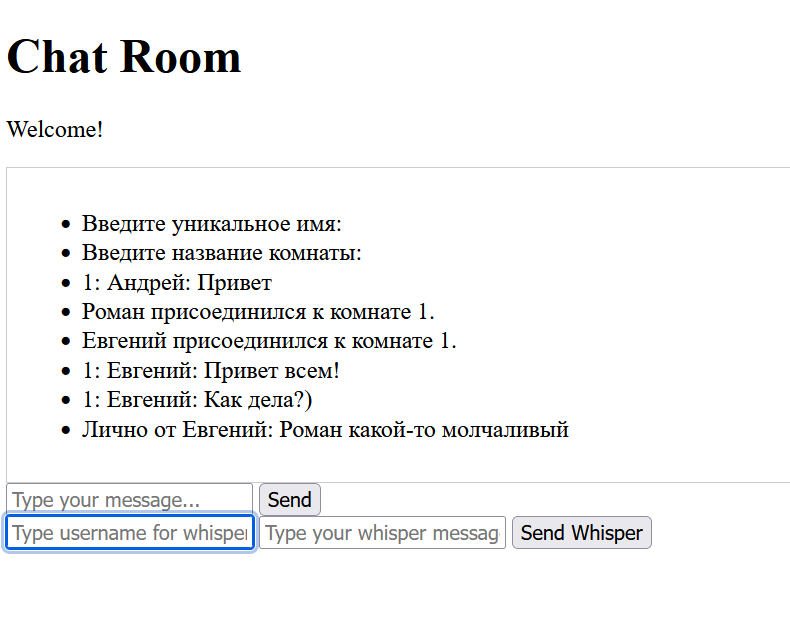


Рисунок 6.2, Пример результата работы программы

# **Вывод**

В результате лабораторной работы успешно создан асинхронный чат-сервер на Python с использованием библиотек asyncio и websockets. Сервер обеспечивает обслуживание множества клиентов, поддерживает чат-комнаты, и предоставляет дополнительные функции, такие как личные сообщения. Программа стабильна и прошла тестирование, подтверждая её эффективность и удобство использования.

# **Список использованной литературы**

[1] Документация библиотеки asyncio: <https://docs.python.org/3/library/asyncio.html>

[2] Документация библиотеки websockets: <https://websockets.readthedocs.io/en/stable/>

[3] Документация Python по обработке ошибок и исключений: <https://docs.python.org/3/tutorial/errors.html>