**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Функциональное программирование»**

**на тему**

**«Разработка асинхронного чат-сервера»**

**Студент гр. 23Б15-пу**

**Сурин И. С.**

**Преподаватель**

**Киямов Ж. У.**

**Санкт-Петербург**

**2024 г.**

Оглавление

1. [Цель работы 3](#_Toc185851723)
2. [Описание задачи (формализация задачи) 3](#_Toc185851724)
3. [Теоретическая часть 4](#_Toc185851725)
4. [Основные шаги программы 6](#_Toc185851726)
5. [Описание программы 7](#_Toc185851727)
6. [Рекомендации пользователя 9](#_Toc185851728)
7. [Рекомендации программиста 10](#_Toc185851729)
8. [Исходный код программы 10](#_Toc185851730)
9. [Контрольный пример 10](#_Toc185851731)
10. [Вывод 13](#_Toc185851732)
11. [Источники 13](#_Toc185851733)

# **Цель работы**

Целью данной работы является разработка чат-клиента и серверной части с использованием асинхронного ввода-вывода на основе библиотеки **asyncio** для языка **Python**. В рамках работы будет реализована возможность обмена текстовыми сообщениями между пользователями, подключёнными к серверу, а также добавлен функционал передачи файлов между клиентами. Дополнительно будет обеспечено переключение между комнатами и обработка подключения нескольких пользователей. Основное внимание уделяется созданию простого, эффективного и функционального взаимодействия между клиентом и сервером в реальном времени.

# **Описание задачи (формализация задачи)**

Задача заключается в разработке асинхронного чат-сервера и клиентского приложения, которые позволяют пользователям обмениваться сообщениями в реальном времени. Для реализации серверной части используется асинхронная библиотека **asyncio**, что обеспечивает эффективную обработку большого количества одновременных подключений без блокировки основного потока выполнения программы.

**Функциональные требования к серверу**

1. **Обработка подключений**: Сервер должен принимать подключения от клиентов и управлять ими с использованием асинхронных задач. При подключении клиент передаёт своё имя и выбирает чат-комнату для общения.
2. **Чат-комнаты**: Сервер должен поддерживать несколько комнат для общения. Клиенты могут присоединяться к выбранной комнате и обмениваться сообщениями с другими участниками этой комнаты. При отключении клиента сервер обновляет список активных пользователей и уведомляет об этом остальных участников комнаты.
3. **Рассылка сообщений**: Сообщения, отправленные одним пользователем, должны доставляться всем остальным клиентам, находящимся в той же комнате. Сервер должен корректно обрабатывать и направлять сообщения по назначению.
4. **Обновление списка активных пользователей**: При подключении или выходе клиента сервер должен обновлять список пользователей в каждой комнате и передавать актуальные данные всем участникам.

**Функциональные требования к клиенту**

1. **Подключение к серверу**: Клиент должен устанавливать соединение с сервером, вводить имя пользователя и выбирать чат-комнату для взаимодействия.
2. **Отправка сообщений**: Клиент должен иметь возможность отправлять текстовые сообщения в выбранную комнату. Каждое сообщение должно включать имя пользователя и метку времени.
3. **Получение сообщений**: Клиент должен асинхронно принимать сообщения от сервера и отображать их в пользовательском интерфейсе.
4. **Отображение списка активных пользователей**: Клиент должен отображать актуальный список пользователей, находящихся в комнате на данный момент.

# **Теоретическая часть**

**1. Асинхронное программирование**  
Асинхронное программирование – это подход, позволяющий выполнять задачи параллельно без блокировки основного потока выполнения. В языке **Python** для создания асинхронного кода используется библиотека **asyncio**, которая предоставляет инструменты для эффективного выполнения операций ввода-вывода (например, чтение данных из сети) без необходимости ожидания завершения каждой задачи. Такой подход особенно эффективен при сетевом программировании, поскольку позволяет обрабатывать множество подключений одновременно.  
Основные компоненты **asyncio** включают:

* **Корутины**: функции, определённые с использованием async def. Их выполнение можно приостанавливать с помощью await, чтобы ожидать завершения асинхронных операций.
* **Цикл событий**: ключевой механизм **asyncio**, который управляет выполнением корутин, распределяя задачи и ресурсы между ними.
* **Задачи**: корутины, запущенные с помощью asyncio.create\_task(), которые могут выполняться параллельно.

**2. Протокол TCP и сокеты**  
Для передачи данных между клиентом и сервером используется протокол **TCP** (Transmission Control Protocol). TCP гарантирует надёжную и упорядоченную доставку данных, а также контролирует их целостность. В данном проекте реализована следующая схема:

* **Сервер** принимает входящие подключения от клиентов и использует асинхронные сокеты для обработки данных. Асинхронные сокеты позволяют серверу избегать блокировки при ожидании данных от каждого клиента.
* **Клиенты** подключаются к серверу по IP-адресу и порту, устанавливая соединение для обмена сообщениями. Через это соединение клиенты могут отправлять и получать данные, поддерживая двустороннюю связь.

**3. Многозадачность с использованием asyncio**  
Для обеспечения взаимодействия множества пользователей в реальном времени сервер должен обрабатывать несколько задач одновременно. В **asyncio** многозадачность реализуется следующим образом:

* Сервер одновременно выполняет несколько задач, таких как приём сообщений, их пересылка другим пользователям и обновление статуса участников чата.
* На стороне клиента **asyncio** позволяет асинхронно обрабатывать отправку и получение данных, что обеспечивает бесперебойную работу приложения. Пользователь может отправлять сообщения и получать их в реальном времени без задержек и блокировок.

**4. Работа с интерфейсом на основе Tkinter**  
Библиотека **Tkinter** предоставляет простые инструменты для создания графического интерфейса (GUI) в **Python**, однако изначально она не поддерживает асинхронные операции. Для интеграции **asyncio** с **Tkinter** используются следующие подходы:

* **Запуск цикла событий**: цикл событий **asyncio** запускается в отдельном потоке, что позволяет интерфейсу оставаться отзывчивым и не блокироваться.
* **Асинхронные функции**: операции отправки и получения данных выполняются в асинхронных корутинах, что позволяет работать с сетью без блокировки интерфейса.
* **Обновление интерфейса**: данные, полученные от сервера, отображаются в окне чата и обновляют список активных пользователей, что даёт возможность пользователям видеть актуальные сообщения и статус участников комнаты.

Таким образом, использование **asyncio** в сочетании с **Tkinter** позволяет создать асинхронный чат с поддержкой множества пользователей. Приложение обеспечивает минимальные задержки при передаче сообщений, удобный графический интерфейс и стабильную работу в реальном времени.

# **Основные шаги программы**

**1. Запуск серверной части**

* Сервер запускается и начинает прослушивать входящие подключения на заданном IP-адресе и порту (например, 127.0.0.2 и порт 8888).
* При подключении нового клиента сервер устанавливает соединение, регистрирует пользователя в указанной чат-комнате и добавляет его в список активных участников этой комнаты.
* Всем участникам комнаты отправляется уведомление о подключении нового пользователя, а также обновлённый список активных участников.

**2. Запуск клиентской части**

* Клиентская программа запускается и отображает графический интерфейс, созданный на основе **Tkinter**, где пользователю предлагается ввести имя и название комнаты для подключения.
* После ввода данных клиентская программа пытается подключиться к серверу.
* В случае успешного соединения пользователь получает доступ к чату, а сервер регистрирует его в выбранной комнате и уведомляет других участников.

**3. Обмен сообщениями**

* После подключения клиент запускает асинхронную задачу для получения сообщений от сервера. Все входящие сообщения от других пользователей и обновлённый список участников отображаются в окне чата.
* Пользователь может ввести сообщение в текстовое поле интерфейса. При нажатии кнопки **"Отправить"** сообщение отправляется на сервер.
* Сервер принимает сообщение, рассылает его всем пользователям в комнате, и они видят это сообщение в своих интерфейсах.

**4. Отключение клиента**

* Если пользователь решает покинуть чат (например, нажимает кнопку **"Выйти"**), клиентская программа завершает соединение с сервером.
* Сервер удаляет пользователя из списка активных участников комнаты, обновляет его для всех оставшихся пользователей и отправляет уведомление о выходе участника.

**5. Закрытие сервера**

* Сервер продолжает ожидать подключения новых клиентов и обработки сообщений, пока его работа не будет остановлена вручную (например, через прерывание в консоли).

# **Описание программы**

Программная реализация разработана на языке Python **3.12.7** с использованием библиотеки **asyncio** для построения асинхронного взаимодействия между клиентом и сервером, а также **tkinter** для создания графического интерфейса клиента. Серверная часть представляет собой асинхронный **TCP-сервер**, который обрабатывает подключения клиентов, управляет чат-комнатами, пересылает сообщения между пользователями и поддерживает актуальный список активных участников каждой комнаты. Клиентская часть реализует соединение с сервером и предоставляет удобный интерфейс для отправки и получения сообщений в реальном времени.

Таблица 1. server.py

|  |  |  |  |
| --- | --- | --- | --- |
| Функция | Описание | Параметры функции | Описание параметров |
| handle\_client | Обрабатывает подключение клиента, регистрирует его в комнате, пересылает сообщения другим участникам комнаты и обрабатывает выход клиента. | reader — объект для чтения данных от клиента.  writer — объект для записи данных клиенту. | reader используется для чтения данных клиента, а writer — для отправки сообщений обратно. |
| send\_message\_to\_room | Отправляет переданное сообщение всем пользователям в указанной комнате. | room — имя комнаты, для которой отправляется информация о пользователях. | Передаётся строка с именем комнаты. |
| |  | | --- | | send\_active\_users\_to\_room |  |  | | --- | |  | | Отправляет список активных пользователей всем участникам указанной комнаты. | room — имя комнаты, в которую отправляется сообщение.  message — сообщение, которое нужно отправить. | Передается название комнаты, в которую отправляется список активных пользователей. |
| main | Запускает сервер и начинает прослушивание подключений на заданном IP и порту | Нет параметров, функция сама инициирует запуск сервера. | Настроен сервер на прослушивание порта и IP-адреса. |

Таблица 2. client.py

|  |  |  |  |
| --- | --- | --- | --- |
| Функция | Описание | Параметры функции | Описание параметров |
| register\_client | Устанавливает соединение с сервером, отправляет данные о пользователе (имя и комнату) и подключает клиента к чату. | ip - IP-адрес сервера.  username - имя пользователя.  room - номер комнаты. | ip — адрес сервера, к которому подключается клиент. username и room — данные пользователя. |
| get\_messages | Получает сообщения от сервера и отображает их в текстовом поле чата, а также обновляет список активных пользователей | reader — объект для чтения данных. text\_widget — текстовый виджет. active\_users\_widget — виджет для списка пользователей. | reader читает данные с сервера, text\_widget выводит сообщения, а active\_users\_widget обновляет список участников. |
| send\_message | Формирует сообщение пользователя с временной меткой и отправляет его на сервер. | writer — объект для записи данных. message — текст сообщения. | writer отправляет сообщение на сервер, а message содержит текст отправляемого сообщения. |
| disconnect\_client | |  | | --- | |  |  |  | | --- | | Закрывает соединение клиента с сервером и завершает программу. | | Нет параметров | Подключение к серверу (объект writer) |
| start\_client | |  | | --- | |  |  |  | | --- | | Запускает основной цикл событий **asyncio** для асинхронного взаимодействия клиента с сервером. | | Нет параметров | Запускает цикл событий **asyncio** для обработки асинхронных задач |
| prompt\_user\_info | Открывает окно для ввода данных пользователя: IP-адреса сервера, имени пользователя и номера комнаты. | Нет параметров | Возвращает введённые данные пользователя (IP-адрес, имя и комнату). |

# **Рекомендации пользователя**

Сервер: cохраните серверный код в файл, например server.py и запустите: python server.py. Сервер начнет работу на 0.0.0.0:5002.

Особенности: Сообщения и файлы обрабатываются внутри комнат. Клиенты подключаются командой /join <имя\_комнаты>. Файлы сохраняются в папке uploads и передаются всем участникам комнаты.

Клиент: Сохраните код клиента в файл, например client.py. Убедитесь, что установлен PyQt5. запустите: python client.py. Сервер должен быть запущен до подключения клиента. Если связь потеряна, клиент автоматически завершит работу.

В окне авторизации введите имя и название комнаты.

Отправка сообщений: Введите текст и нажмите Enter или Send.

Отправка файлов: Нажмите Send File, выберите файл. У получателя появится окно сохранения.

Смена комнаты: Нажмите Change Room и введите новое имя комнаты.

Выход: Нажмите Exit для завершения.

# **Рекомендации программиста**

* **Установка необходимых библиотек**
* Программа использует стандартные библиотеки Python **asyncio** и **tkinter**, которые устанавливаются автоматически с Python и не требуют дополнительной установки.
* Если вы работаете в окружении, где эти библиотеки отсутствуют (например, минимальная установка Python), убедитесь, что tkinter доступен.
* **Проверка версии Python**
* Код протестирован и совместим с Python 3.7 и выше, но рекомендуется использовать Python 3.10 или новее для наилучшей поддержки **asyncio**.
* Убедитесь, что версия Python установлена корректно, запустив:



* **Проверка сети и порта**
* Убедитесь, что IP 127.0.0.1 и порт 8888 не заблокированы брандмауэром и доступны для локальных подключений, так как сервер и клиент взаимодействуют через локальный IP.
* Для работы на нескольких устройствах убедитесь, что в **server.py** указан IP-адрес сервера, доступный в сети.

# **Исходный код программы**

<https://github.com/Ignatio27/Func_prog_corse_2_spbu>

# **Контрольный пример**

1. Запуск сервера: для запуска сервера используйте файл **server.py.** Программа должна запустить сервер и вывести в терминал соответствующую надпись. (Рис. 1)

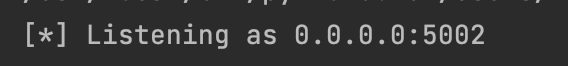


Рис. 1 Запуск сервера

2. Запуск клиента: при запуске клиента откроется окно регистрации в котором необходимо указать некоторые данные (имя, номер чат-комнаты). (Рис.2)

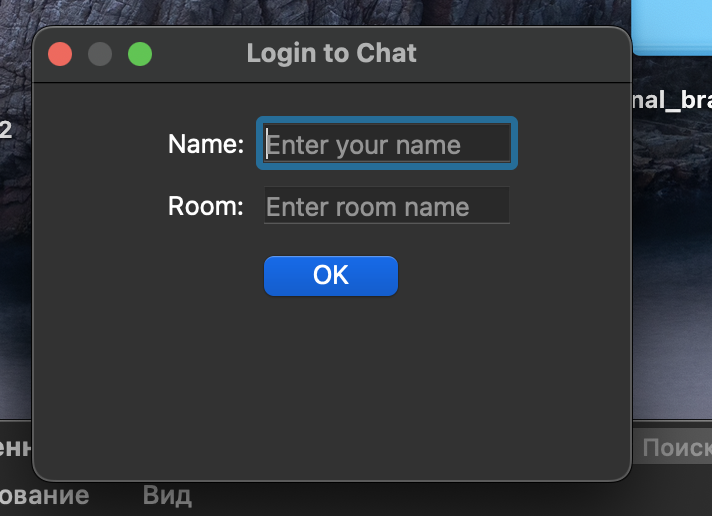


Рис.2 Окно регистрации

3. Вход в чат-комнату: после нажатия: “Enter” откроется выбранная чат-комната, обновится окно активных пользователей, и всех уведомят о том, что вы подключились. (Рис. 3)

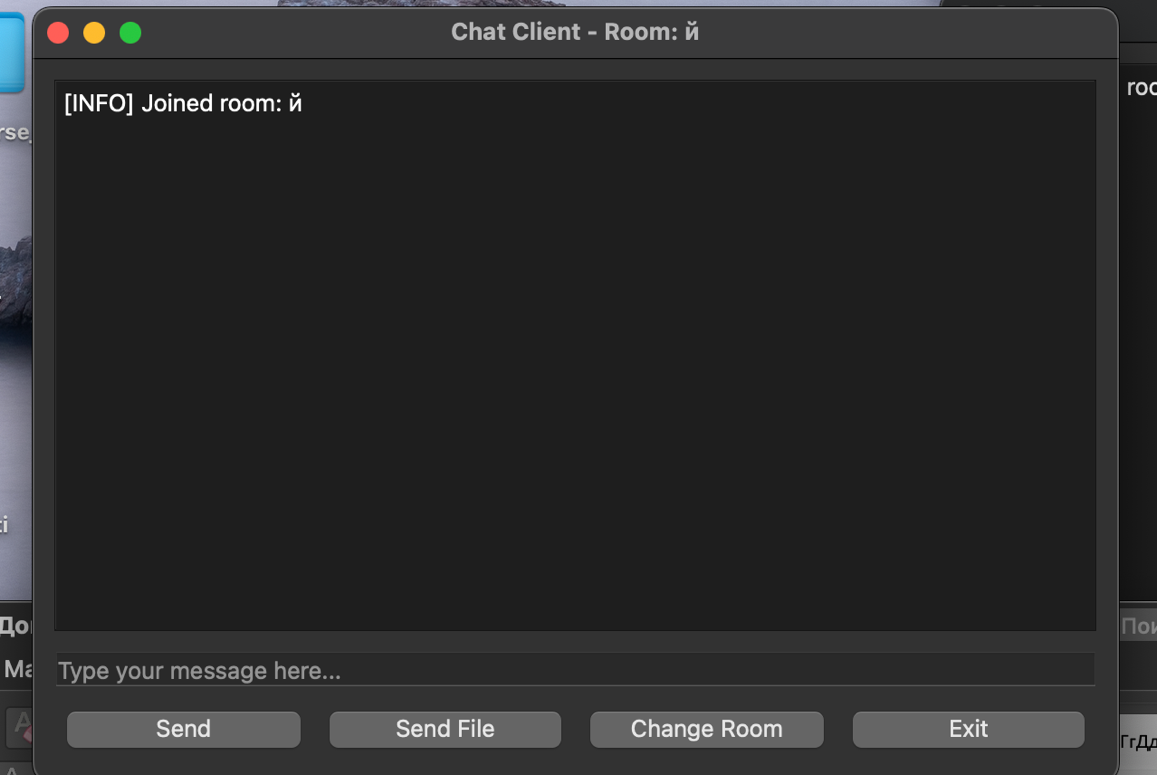


Рис. 3 Чат-комната

4. Отправка сообщения: в поле – “Enter message…” вы можете вводить что угодно и при нажатии на “Send” или Enter на клавиатуре отправлять другим пользователям в чат-комнате. (Рис. 4)

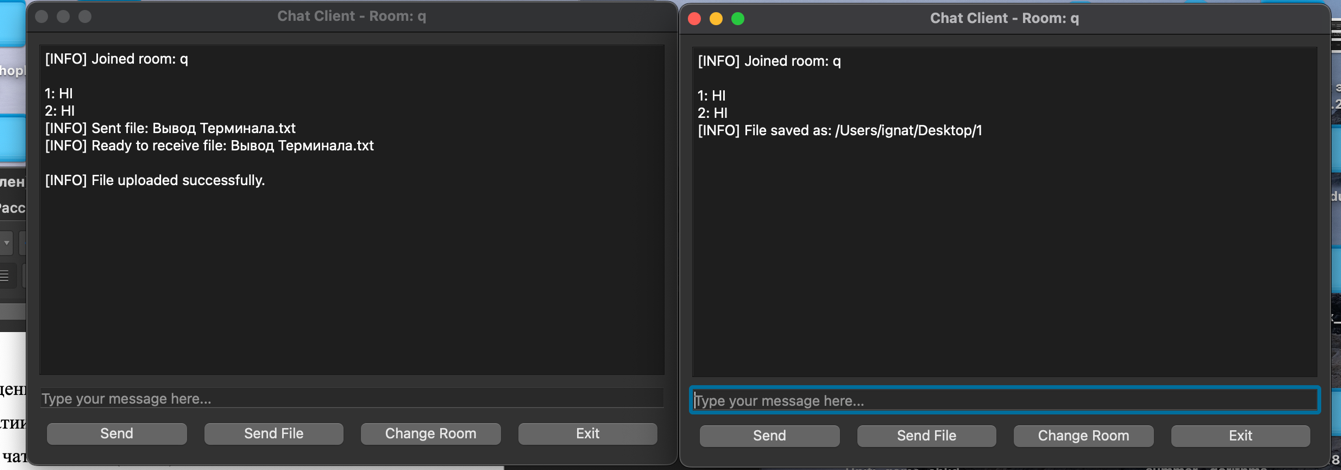
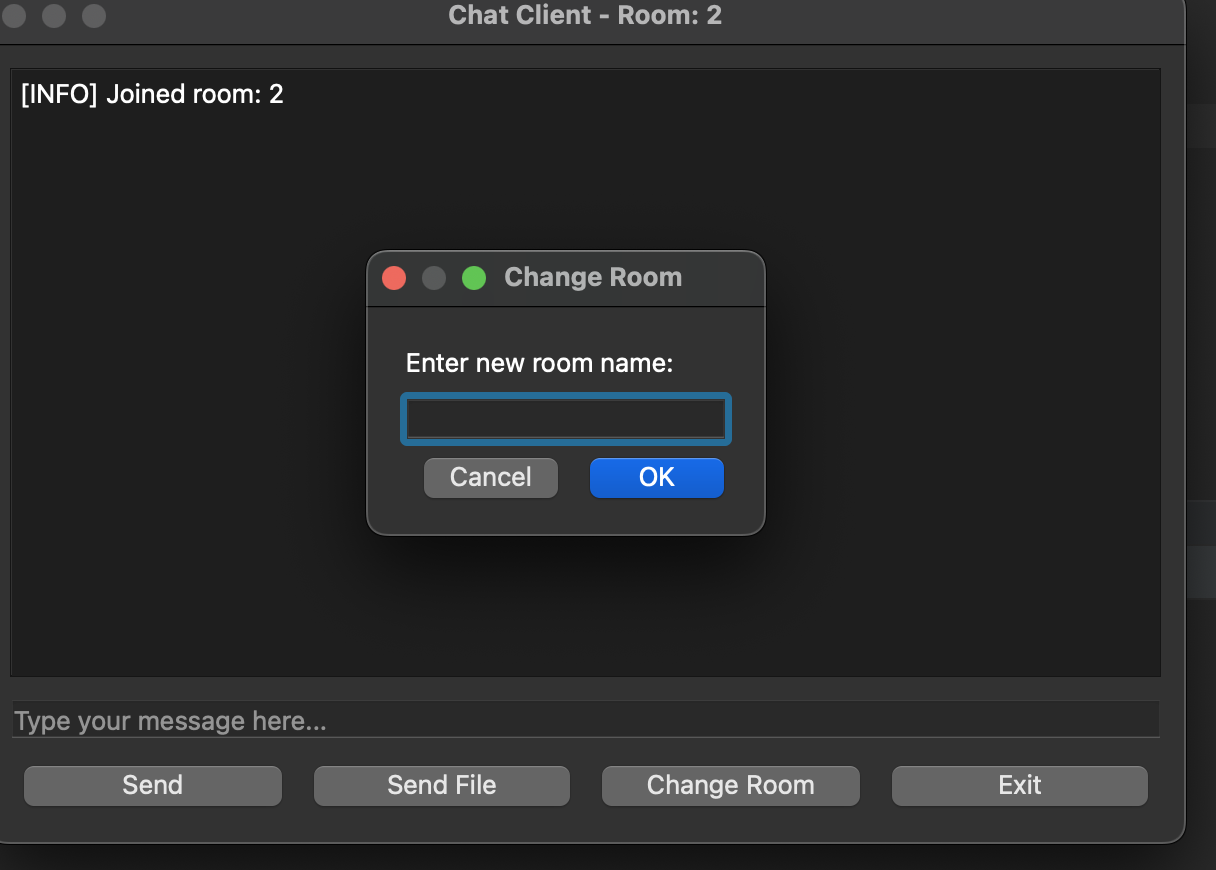


Рис. 4 Отправка сообщений и файлов

1. Смена комнаты: при нажатии кнопки “Change Room” выпадает окно, в которым вы можете поменять комнату. (Рис. 5)

 Рис 5. Смена комнаты

# **Вывод**

В ходе выполнения данного проекта была реализована клиент-серверная архитектура на Python с использованием асинхронного программирования для эффективной обработки сетевых запросов.

Графический интерфейс, разработанный с помощью tkinter, обеспечивает интуитивное взаимодействие пользователя с приложением, позволяя отправлять запросы и получать ответы в режиме реального времени.

Серверная часть постоянно ожидает подключений и обрабатывает запросы асинхронно, обеспечивая высокую производительность. Клиентская часть отвечает за отправку запросов и отображение полученных данных в понятном формате. Проект успешно продемонстрировал эффективность асинхронного подхода в Python, а также возможности интеграции сетевых функций и GUI.

Полученное приложение служит наглядным примером для изучения и дальнейшей разработки подобных асинхронных сетевых приложений с графическим интерфейсом на Python, подтверждая гибкость и функциональность Python в данной области.

Благодаря богатому набору библиотек Python показал себя как мощный и удобный инструмент для создания подобных приложений, сочетающих в себе сетевое взаимодействие и удобный пользовательский интерфейс.

# **Источники**

1. **Tkinter //** [*https://docs.python.org/3/library/tkinter.html*](https://docs.python.org/3/library/tkinter.html%20)  *(дата обращения: 22.12.2024)*
2. **asyncio //** [*https://docs.python.org/3/library/asyncio.html*](https://docs.python.org/3/library/asyncio.html) *(дата обращения:22.12.2024)*