|  |  |
| --- | --- |
|  | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение высшего профессионального образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова» (БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-02 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Факультет |  | О |  | | Естественнонаучный |
|  |  | шифр |  | | наименование |
| Кафедра |  | О7 |  | | Информационные системы и программная инженерия |
|  |  | шифр |  | | наименование |
| Дисциплина |  | | | Системное программное обеспечение | |

КУРСОВАЯ РАБОТА

на тему

|  |
| --- |
| Разработка клиент-серверного приложения на языке программирования Python |
| с использованием библиотек asyncio, threading, socket, CustomTkinter и реализацией |
| функции чата с графическим интерфейсом и дополнительными функциями |

|  |  |
| --- | --- |
| Выполнил студент группы | О718Б |
| Папанов К.К. |  |
| Фамилия И.О. |  |
| **РУКОВОДИТЕЛЬ** | |
| Устиновский Г.С. | |
| Фамилия И.О. Подпись | |
| Оценка | |
| « » | 20 г. |

СОДЕРЖАНИЕ

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 3](#_Toc136999020)

[ВВЕДЕНИЕ 4](#_Toc136999021)

[1 Описание разработанной программы 5](#_Toc136999022)

[1.1 Формулировка задания 5](#_Toc136999023)

[1.2 Структура программы 6](#_Toc136999024)

[2 Описание классов сервера 7](#_Toc136999025)

[2.1 Класс «Async\_server» 7](#_Toc136999026)

[3 Описание классов клиента 8](#_Toc136999027)

[3.1 Класс «Window\_chat» 8](#_Toc136999028)

[3.2 Класс «Chat\_frame» 9](#_Toc136999029)

[3.3 Класс «Client» 10](#_Toc136999030)

[4 Описание классов библиотеки 12](#_Toc136999031)

[5 Схема работы 13](#_Toc136999032)

[6 Демонстрация работы 14](#_Toc136999033)

[**ЗАКЛЮЧЕНИЕ** 20](#_Toc136999034)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc136999035)

[ПРИЛОЖЕНИЕ А 22](#_Toc136999036)

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие сокращения и обозначения:

CustomTkinter – библиотека написанная на языке python, которая используется для создания графического интерфейса. Наработка над базовым модулем python - Tkinter.

Asyncio – python-модуль, который используется для асинхронного программирования c применением конкурентного выполнения кода. Взаимодействие с сервером здесь реализовано наработкой над другим модулем - socket.

Threading - python-модуль, который используется для создания многопоточности и более удобным управлением потоками.

Atom — это текстовый редактор с открытым исходным кодом, который используется в качестве интегрированной среды разработки (IDE) для самых разных языков программирования. В данном проекте, он используется, как IDE для написания кода на языке python

Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.

Dict – объект python, реализующий словарь.

ВВЕДЕНИЕ

Клиент - серверное приложение является одной из самых распространенных архитектурных моделей для создания программного обеспечения. Эта модель представляет собой совокупность двух компонентов: клиентской и серверной частей, которые взаимодействуют между собой через сеть. Клиент, это программа, которая запускается на компьютере пользователя и обеспечивает доступ к различным ресурсам, которые находятся на сервере. Сервер, это программа, которая работает на каком-либо устройстве и обеспечивает доступ к ресурсам, которые находятся на нем.

Клиент и сервер обмениваются данными по определенному протоколу, который определяет формат передачи данных и правила взаимодействия между компонентами. Клиент может отправлять запросы на сервер, а сервер отвечать на них, обрабатывая запросы и отдавая необходимые данные.

Такая модель позволяет достичь высокой масштабируемости и удобства в разработке приложений, так как клиенты и серверы могут работать на различных платформах и операционных системах.

Однако, клиент - серверная архитектура может иметь некоторые недостатки, такие как узкое место в виде сервера, который может стать единой точкой отказа или, если соединение между клиентом и сервером прерывается, то клиент не сможет получить доступ к ресурсам сервера.

Несмотря на это, клиент - серверная модель по-прежнему является одной из самых популярных архитектурных моделей для создания программного обеспечения, так как она обеспечивает многие преимущества, такие как масштабируемость, удобство в разработке, и универсальность.

Целью данной работы является создание клиент-серверной связи, реализованной в качестве приложения-чата, с дополнительными функциями. Для написания работы был выбран язык программирования Python, для реализации графического интерфейса была использована библиотека CustomTkinter.

# 1 Описание разработанной программы

## 1.1 Формулировка задания

Целью курсовой работы является разработка чата с подключенным пользовательским графическим интерфейсом, а также с дополнительными опциями для пользователей чата.

На первом этапе необходимо ознакомиться с поставленной задачей и обговорить с преподавателем особенности реализации.

Задача включает в себя следующее:

* для разработки использовать языка программирования python;
* использование библиотек asyncio, socket и threading для реализации клиента и сервера.

К разрабатываемой программе предъявлены следующие требования:

* наличие пользовательского интерфейса;
* возможность обработки сложных запросов (не только как чат);
* отображение всех взаимодействий клиента с сервером в пользовательском интерфейсе (отображение истории обращений к серверу);
* выводить дату и время последнего сообщения.

На втором этапе необходимо ознакомиться с инструментами разработки.

Asyncio была выбрана для создания серверной части т.к. нам надо одновременно обрабатывать запросы нескольких клиентов. Конкурентная реализация сервера в этом модуле позволяют делать это с минимальными затратами ресурсов. Для клиента используется модуль socket, потому что он имеет всего одну связь только с сервером. Также в клиенте используется модуль threading для обновления графического интерфейса одновременно с прослушиванием сервера.

Для реализации графического интерфейса была выбрана библиотека CustomTkinter из-за простоты её использования и удобного функционала. Также она более гибкая нежели python-модуль Tkinter.

На третьем этапе необходимо приступить к описанию логики клиент-серверного взаимодействия. Взаимодействие будет двухсторонним и будет происходить многократно. На стороне сервера будет регистрация по имени клиента.

На четвертом этапе необходимо добавить пользовательский интерфейс, которые нужен для более удобного взаимодействия пользователя с приложением.

## 1.2 Структура программы

Программа написана на языке Python с использованием CustomTkinter, разработана в среде программирования Atom.

Для реализации сервера были создан класс - Async\_server.

Для реализации клиента были созданы следующие классы:

* Window\_chat;
* Chat\_frame;
* Client.

Также для реализации пользовательского интерфейса использовались

классы, взятые из библиотеки CustomTkinter, являющие собой объекты интерфейса:

* CTk;
* CTkLabel;
* CTkEntry;
* CTkButton;
* CTkScrollableFrame.

# 2 Описание классов сервера

## 2.1 Класс «Async\_server»

Класс отвечает за создание самого асинхронного сервера на базе модуля asyncio. Сервер является консольным приложением и сообщает туда все события, чтобы разработчик при написании кода мог отследить, что на нем происходит.

Поля класса «Async\_server»:

* self.host – хранит информацию о хосте сервера;
* self.port – хранит информацию о порте подключения сервера;
* self.users – объект dict, который хранит данные о клиентах в формате “имя пользователя : объект writer”. Writer – объект модуля asyncio для передачи данных клиенту;
* self.status – объект dict, который хранит данные о статусах клиентов в формате “имя пользователя : статус”.

Методы класса «Async\_server»:

* \_\_init\_\_(self, host = 'localhost', port = 9090) – конструктор класса, принимающий аргументы host и port, и передающий их в поля self.host и self.port соответственно;
* \_\_del\_\_(self) – деструктор класс, который отвечает за корректный конец работы сервера;
* all\_send(self, data) – асинхронный метод, который отправляет всем клиентам из словаря self.users сообщение data;
* message\_handler(self, \_message, writer) – асинхронный метод, который, обрабатывает сообщение \_message от определенного клиента, а также получает объект writer этого клиента для корректной обработки и отправки данных клиентам. Эта функция, обрабатывает нет только простые сообщения, но и сложные запросы: вывод случайной масти карт или добавление пользователю статуса перед его именем. Возвращает переменную типа boolean, которая позволяет понять взаимодействует ли клиент с сервером или нет;
* speak\_with\_client(self, reader, writer) – асинхронный метод – прослушка, который ждет сообщения от клиента, пока тот подключен к серверу и дальше отправляет это сообщение на обработку;
* main(self) – асинхронный метод, создающий асинхронный и задающий его характеристики;
* run\_server(self) – метод асинхронного запуска сервера.

# 3 Описание классов клиента

## 3.1 Класс «Window\_chat»

Класс отвечает за создание самого окна с графическим интерфейсом для приложения клиента. Включает в себя чат, кнопки, поля ввода и т.д. Наследуется от класса CTk библиотеки CustomTkinter и по сути является «обёрткой» главного окна.

Поля класса «Window\_chat»:

* self.client – объект класса Client, который описывает серверную часть клиента;
* self.chat\_frame – объект класса Chat\_frame, который описывает чат;
* self.info – объект класса CTkLabel, который описывает поле с текстом информации о пользовании клиентским приложением;
* self.label – объект класса CTkLabel, который описывает поле с текущими состояниями приложения;
* self.entry – объект класса CTkEntry, который описывает поле для ввода текста ;
* self.button – объект класса CTkButton, который описывает кнопку для отправки сообщений, а также в случае сбоя эта кнопка становится кнопкой выхода.

Методы класса «Window\_chat»:

* \_\_init\_\_(self) – конструктор класса, который задает характеристики окна и запускает само окно с графическим интерфейсом;
* on\_closing(self) – функция, которая отвечает за корректный выход из приложения с отправкой серверу специально сообщения-маркера об этом событии;
* screen\_of\_input\_name(self) – функция, которая отвечает за первоначальное меню с полями ввода имени, хоста и порта сервера. В случае успешной ввода запускает чат, иначе сообщает об ошибке и не дает запустить чат;
* run\_speak\_with\_server(self, name, host, port) – основная функция, которая отвечает за создание окна с чатом, проверки корректности введенного хоста и порта и имени. В случае некорректного ввода снова возвращает в функцию screen\_of\_input\_name.

## 3.2Класс «Chat\_frame»

Класс отвечает за создание чата и взаимодействие его с модулем threading, для отображения сообщений от сервера в реальном времени. Наследуется от класса CTkScrollableFrame библиотеки CustomTkinter.

Поля класса «Chat\_frame»:

* self.client – объект класса Client, который описывает серверную часть клиента;
* self.label – объект класса CTkLabel, который описывает поле с текущими состояниями приложения;
* self.button – объект класса CTkButton, который описывает кнопку для отправки сообщений, а также в случае сбоя эта кнопка становится кнопкой выхода.

Методы класса «Chat\_frame»:

* \_\_init\_\_(self, master, client, label, button, \*\*kwargs) – конструктор класса, принимающий аргументы client, label и button из основного окна, с которыми можно будет взаимодействовать теперь и из этого класса. В конструкторе запускается новый поток при помощи модуля threading, который запускает функцию print\_message;
* print\_message(self) – функция, которая запускается в отдельном потоке, чтобы в режиме реального времени параллельно основному потоку слушать сообщения с сервера и выводить их в чат.

## 3.3Класс «Client»

Класс «Client» обеспечивает полную работу клиента и отвечает за все внутренние процессы взаимодействия с сервером, происходящие во время его работы. Именно в нём происходит подключение к серверу, отключение от него, обработка сообщений, которые приходят от других пользователей и отправка тех сообщений, что пишет текущий пользователь клиента.

Поля:

* self.name – переменная, хранящая имя пользователя;
* self.server – кортеж, хранящий данные хоста и порта сервера;
* self.client\_sock – объект класса socket модуля socket, который

отвечает за взаимодействие с сервером.

Методы:

* \_\_init\_\_(self, name = 'user', host = socket.gethostbyname(socket.gethostname()), port = 9090) – конструктор класса, принимающий аргументы имени, хоста и порта, которые заносятся в self.name и self.server соответственно;
* connect(self) – функция, которая создает такой объект, как socket и подключается к серверу, при успешном подключении возвращает True, иначе возвращает False;
* disconnect(self) – функция, которая отвечает за отключение от сервера, при этом отправляя на него сообщение с соответствующим событием;
* \_\_del\_\_(self) – деструктор класса, отвечающий за корректный конец работы класса ;
* listen(self) – функция-прослушка, которая ждет сообщение от сервера и возвращает False и само сообщение при успешном получении или True и текст ошибки при получении ошибки от сервера;
* new\_message(self, state, text = '') – функция, которая отправляет новое сообщение серверу по шаблону в соответствии с состоянием, которое указывается при её вызове. (Список состояний: [CON] – сообщение серверу о состоянии подключения; [DIS] – сообщение серверу о состоянии отключения; [MES] – сообщению серверу о состоянии отправки сообщения; [COM] – сообщение серверу о состоянии отправки команды.).

# 4 Описание классов библиотеки

* CTk – класс, описывающий окно приложения;
* CTkLabel – класс, описывающий поле для текста;
* CTkEntry – класс, описывающий поле ввода текста;
* CTkButton – класс, описывающий поле кнопку;
* CTkScrollableFrame – класс, описывающий поле со специальным элементом для перемещения и позволяющий вместить в себя элементы.

# 5 Схема работы

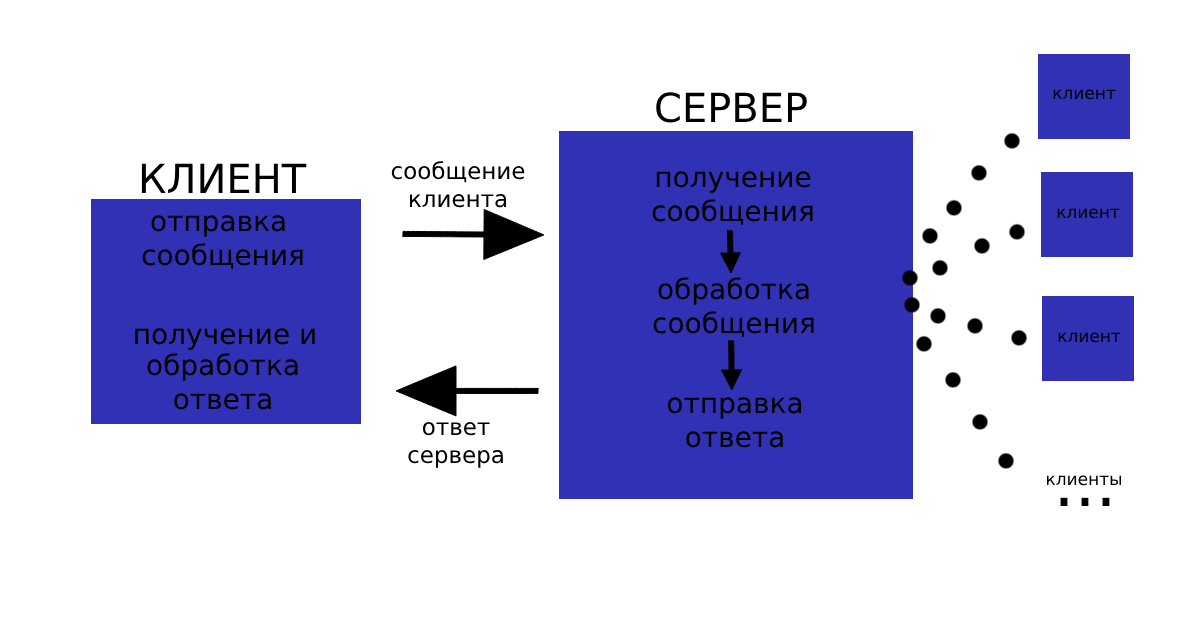


Рисунок 5.1 – Схема работы клиент-серверного приложения

Все сообщения между клиентом и сервером в проекте создаются по специальным шаблонам для удобства разработки и корректной обработки сообщений.

У клиента есть 4 состояния, которые регистрируются сервером: [CON] – сообщение серверу о состоянии подключения; [DIS] – сообщение серверу о состоянии отключения; [MES] – сообщению серверу о состоянии отправки сообщения; [COM] – сообщение серверу о состоянии отправки команды.



Рисунок 5.2 – Шаблон для сообщений клиента

У сервера в свою очередь, есть только 2 состояния, которые регистрируются клиентом: [ANS] – сообщению клиенту о состоянии отправки сообщения; [ERR] – сообщение клиенту об возникшей на сервере ошибке, тогда вместо текста сообщения будет текст ошибки для корректной обработки.

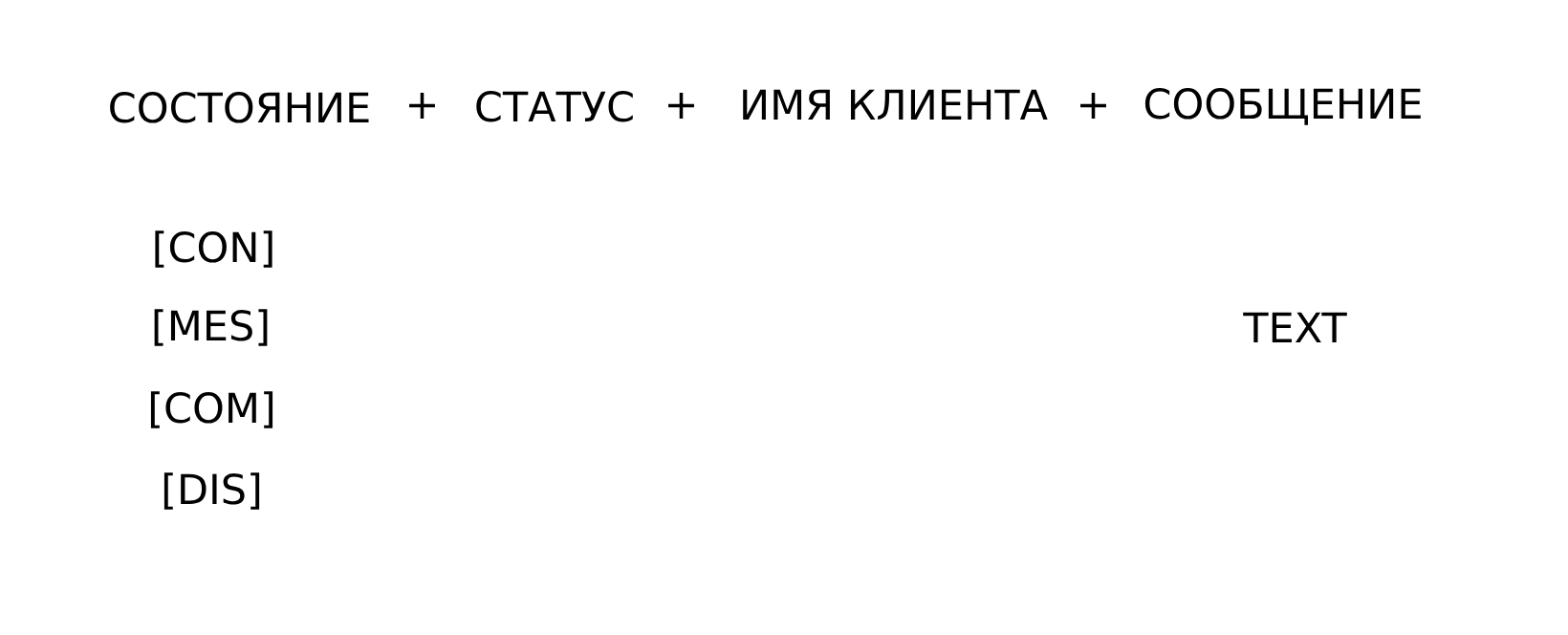


Рисунок 5.3 – Шаблон для ответа сервера

# 6 Демонстрация работы

Так как у нас чат, то подразумевается первоначальный запуск сервера. Это консольное приложение, которое выводит информацию о событиях на сервере. Интерфейс серверу не нужен т.к. пользователи с ним никак не взаимодействуют.

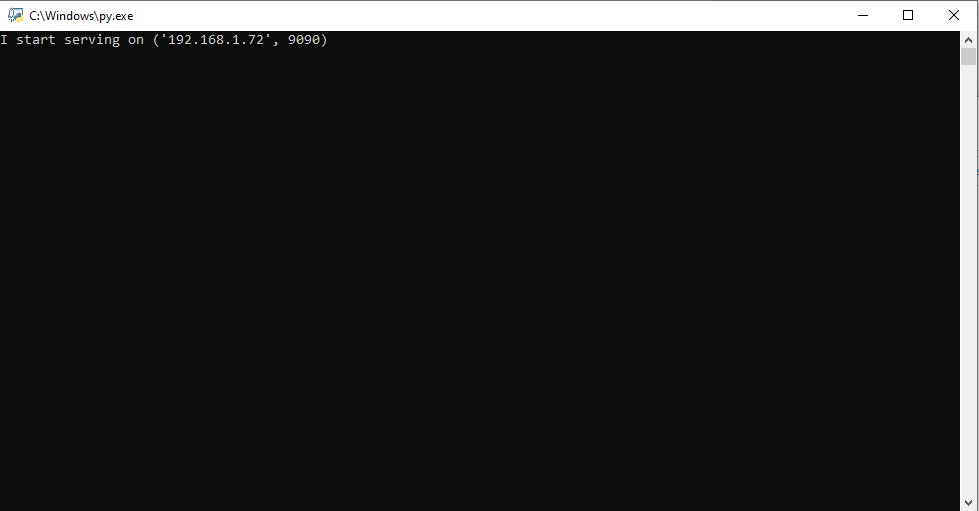


Рисунок 6.1 – Окно запуска сервера

Сервер фиксирует и выводит информацию о подключившемся клиенте, о полученных сообщениях и отключениях клиентов от сервера, как показано на рисунке 5.2.

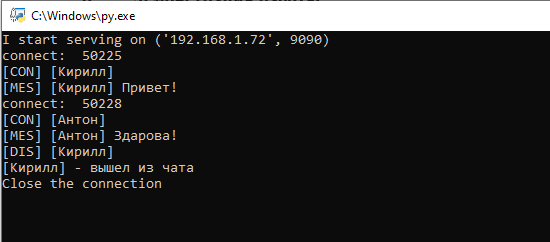


Рисунок 6.2 – Сервер включен

В клиентской части создан интерфейс т.к. именно с ней взаимодействует пользователь. При запуске приложения сначала будет окно подключения, где сначала надо заполнить ваше имя, хост и порт сервера, чтобы подключится. Если не указывать хост и порт, то будет попытка подключения к localhost с портом 9090. Окно подключения показано на рисунке 5.3.

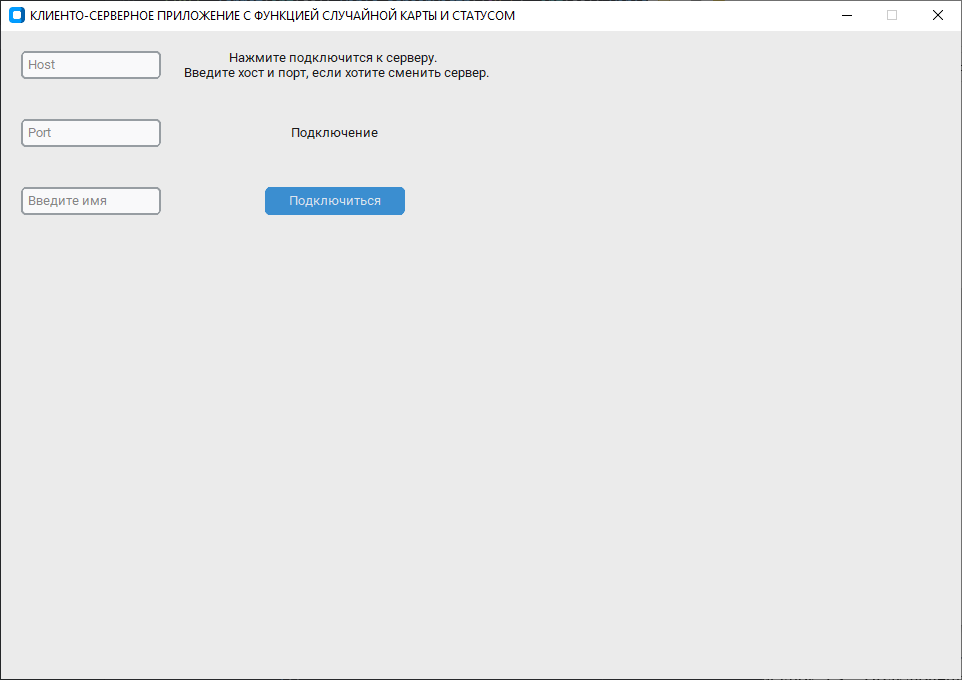


Рисунок 6.3 – Интерфейс окна подключения

Дальше на рисунках показаны ограничения окна подключения. На рисунке 6.4 неправильный ввод имени. Стоит ограничение на слитные и не пустые имена без пробелов. На рисунке 6.5 клиент не может подключится к серверу из-за некорректных данных. На рисунке 6.6 рассмотрена ситуация, если на сервере уже есть клиент с таким именем.

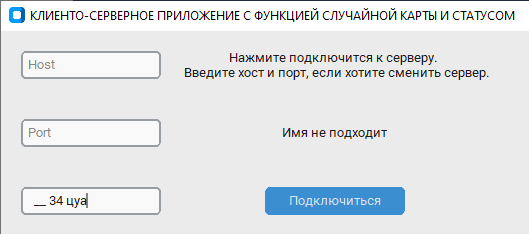


Рисунок 6.4 – Неправильное имя пользователя

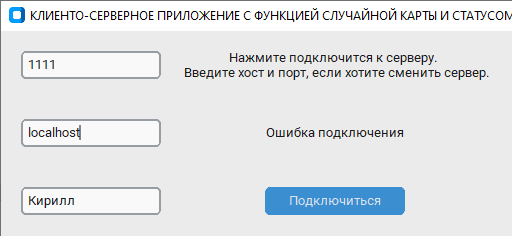


Рисунок 6.5 – Неправильные данные сервера

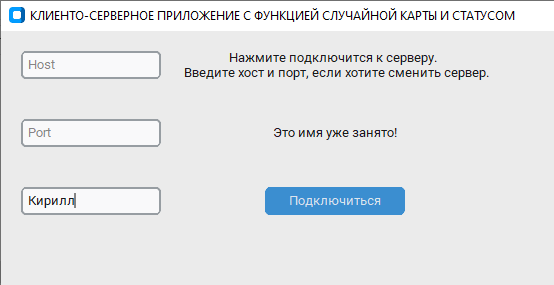


Рисунок 6.6 – Одинаковые имена пользователей на сервере

В случае удачного подключения у вас откроется окно чата, где будет написана справка и будет расположен сам чат, что показано на рисунке 6.7.

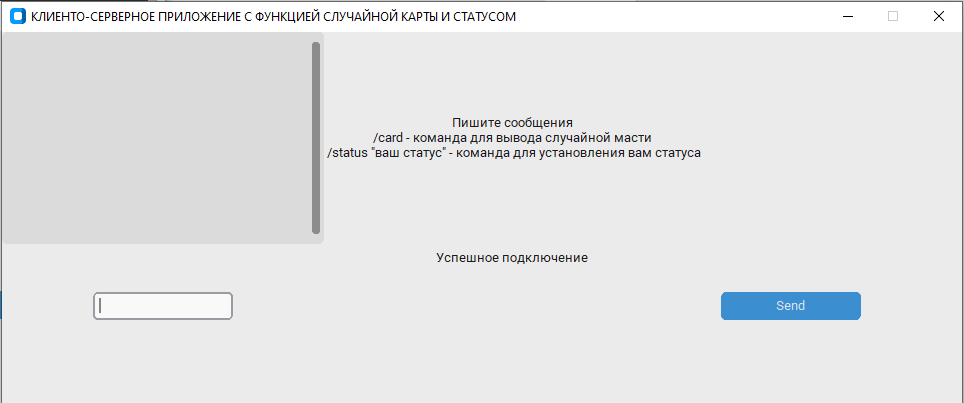


Рисунок 6.7 – Удачное подключение к серверу

Если какой-то пользователь зайдет или выйдет из приложения, то это все отобразится в чате клиента вместе со временем события, что показано на рисунке 6.8.

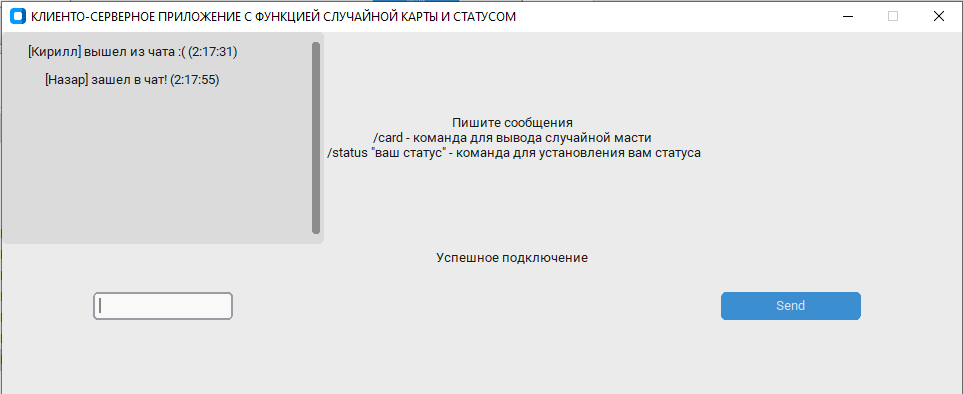


Рисунок 6.8 – Отображения событий в чате

Любые сообщения пользователей также отображаются в чате вместе с именем пользователя, отправившего сообщения, и временем отправки этого сообщения, что показано на рисунке 6.9.

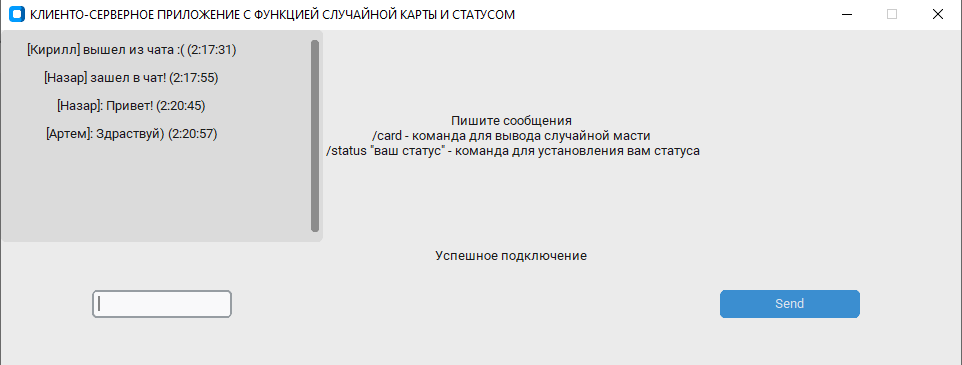


Рисунок 6.9 – Отображение сообщений

Первая дополнительная функция сервера – добавление статуса к пользователю. Если написать серверу команду /status, а дальше его указать, то сервер присвоит этот статус этому пользователю и будет указывать при каждом его сообщении, как показано на рисунке 6.10.

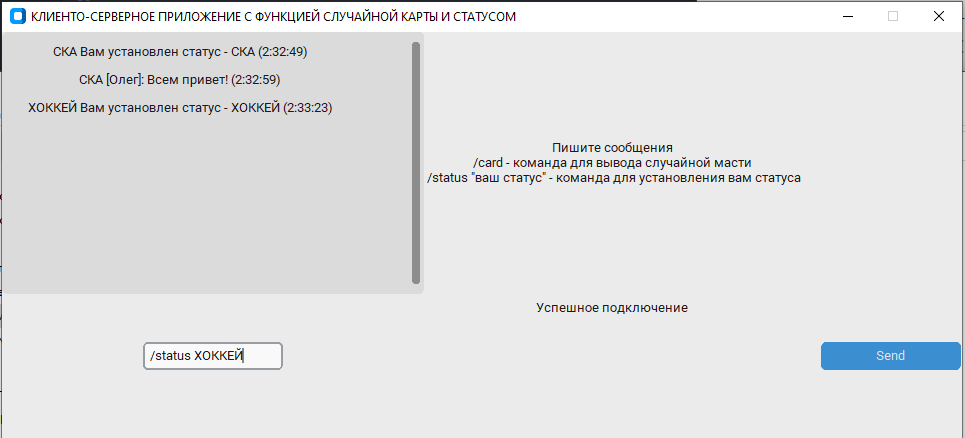


Рисунок 6.10 – Добавление статуса пользователю

Вторая дополнительная функция сервера – это получение случайной масти карт при помощи команды /card, что показано на рисунке 6.11.

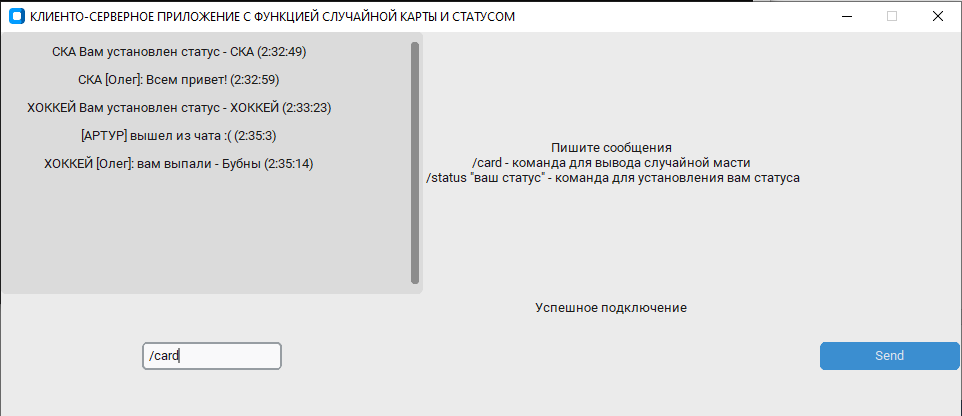


Рисунок 6.11 – Функция случайной масти

Выйти из приложения можно просто нажав на крестик. Специальная функция обработает выход и отправит серверу статус отключения, чтобы корректно завершить работу. А в случае, если сервер отключится раньше, то пользователю выведется сообщение о разорванном подключении и будет предложено выйти через кнопку, что показано на рисунке 6.12

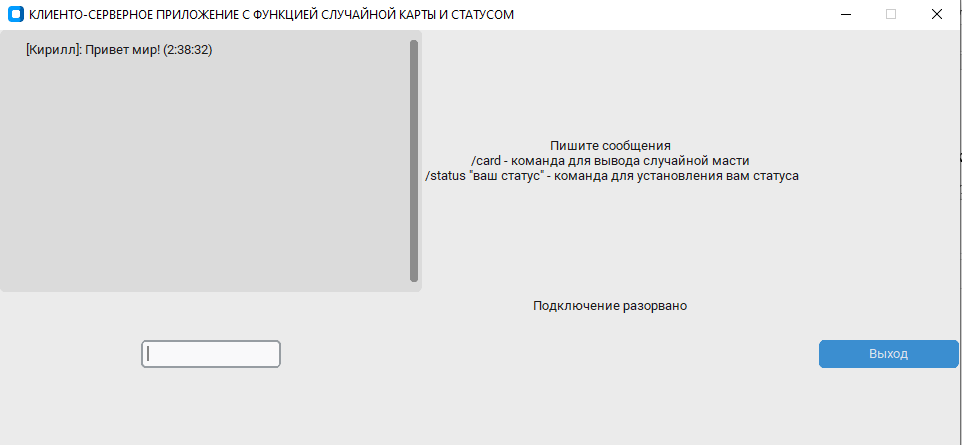


Рисунок 6.12 – Отключение сервера

**ЗАКЛЮЧЕНИЕ**

В процессе работы была реализована клиент-серверная связь в виде чата, в котором могут вести переписку пользователи, соединенные локальной сетью. Было написано две программы с реализацией серверной работы на асинхронном модуле asyncio и клиентской с помощью socket и threading. Был реализован графический интерфейс с использованием библиотеки CustoTkinter, с кнопками и текстовыми строками, упрощающими работу пользователя с программой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт с документацией по CustomTkinter. – URL: https://customtkinter.tomschimansky.com/ (дата обращения 27.03.2023).
2. Репозиторий библиотеки CustomTkinter. – URL: https://github.com/tomschimansky/customtkinter (дата обращения 27.03.2023).
3. Руководство по python-модулю asyncio, – URL: https://habr.com/ru/companies/wunderfund/articles/700474/ (дата обращения 28.03.2023).
4. Документация по встроенной библиотеке Threading. – URL: https://docs.python.org/3/library/threading.html (дата обращения 28.03.2023)
5. Учебник по созданию клиент-серверного приложения. – URL: https://metanit.com/python/network/1.3.php (дата обращения 30.03.2023)

# ПРИЛОЖЕНИЕ А

**Исходный код приложения**

Исходный текст разработанной программы находится на облачном хранилище в удаленном репозитории github по ссылке: https://github.com/papanoffk/client\_server/tree/main  
 Для запуска сервера нужно запустить файл main.py, а для запуска клиента файл Window\_chat.pyw. Перед этим загрузить зависимости из файла requirements.txt