

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
к курсовому проектированию  
по курсу «Программирование на языке Java»  
на тему «Разработка многомодульного приложения на языке Java»

Выполнил:  
Студент группы 23ВВП1  
Ториев Э.А.

Приняли:  
Юрова О.В.  
Карамышева Н.С.

*хорошо*  
*21.05.26*  
*о.в.*

Пенза 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"



"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

профессор М. А. Митрохин

« \_\_\_\_ » \_\_\_\_\_ 20\_\_ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование на языке Java

Студенту Морисову Игорю Викторовичу Группа 23 ВВТД1  
Тема проекта Разработка многомодульного приложения на языке Java

Исходные данные (технические требования) на проектирование

Разработать клиент-серверное приложение «Текстовый чат»

Язык программирования Java, среда разработки NetBeans.

Приложение должно обладать графическим интерфейсом и использовать следующие технологии:

1. Java Collections Framework

2. Механизм обработки исключительных ситуаций

3. Java Stream API

4. Java Multithreading

5. Сетевое взаимодействие.

## Объем работы по курсу

### 1. Расчетная часть

---

---

---

---

---

### 2. Графическая часть

Диаграммы языка UML:

диаграмма вариантов использования;

диаграмма классов;

диаграмма деятельности;

диаграмма развёртывания;

диаграмма последовательности.

---

### 3. Экспериментальная часть

Тестирование многомодульного приложения

---

---

Срок выполнения проекта по разделам

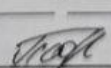
|   |  |
|---|--|
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |

Дата выдачи задания "13" февраля 2026 г.

Дата защиты проекта "19" мая 2026 г.

Руководитель Карамышева Н.С. 

Задание получил " " 2026 г.

Студент Ториев Э.А. 

|                                                                 |    |
|-----------------------------------------------------------------|----|
| ВВЕДЕНИЕ.....                                                   | 3  |
| 1. Теоретическая часть.....                                     | 4  |
| 1.1. Стек протоколов TCP/IP.....                                | 4  |
| 1.1.1. Уровни стека TCP/IP.....                                 | 5  |
| 1.1.2. Прикладной уровень.....                                  | 6  |
| 1.1.3. Транспортный уровень.....                                | 7  |
| 1.1.4. Сетевой (межсетевой) уровень.....                        | 8  |
| 1.1.5. Канальный уровень.....                                   | 8  |
| 1.2. Взаимодействие с протоколом TCP/IP на уровне программ..... | 10 |
| 1.2.1. История появления сокетов.....                           | 10 |
| 1.2.2. Понятие сокета и их классификация.....                   | 11 |
| 1.2.3. Тип данных SOCKET.....                                   | 11 |
| 2. Практическая часть.....                                      | 12 |
| 2.1. Постановка задачи.....                                     | 12 |
| 2.2. Выбор решения.....                                         | 17 |
| 2.3. Организация пользовательского интерфейса.....              | 17 |
| 2.4. Описание работы.....                                       | 18 |
| 2.4.1. Подключение и отключение.....                            | 18 |
| 2.4.2. Отправка и получение сообщений.....                      | 21 |
| 3. Отладка и тестирование приложения.....                       | 25 |
| ЗАКЛЮЧЕНИЕ.....                                                 | 30 |
| СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ.....                             | 31 |
| ПРИЛОЖЕНИЕ А.....                                               | 32 |
| А1. Файл App.java.....                                          | 33 |
| А2. Файл Server.java.....                                       | 33 |
| А3. Файл User.java.....                                         | 34 |
| ПРИЛОЖЕНИЕ Б.....                                               | 39 |
| Б1. Файл App.java.....                                          | 40 |
| Б2. Файл ActionListener.java.....                               | 41 |
| Б3. Файл DisconnectActionListener.java.....                     | 42 |
| Б4. Файл MainWindow.java.....                                   | 43 |
| Б5. Файл Receiver.java.....                                     | 45 |
| Б6. Файл SendTextActionListener.java.....                       | 46 |
| ПРИЛОЖЕНИЕ В.....                                               | 47 |
| В1. Диаграмма вариантов использования.....                      | 48 |
| В2. Диаграмма деятельности.....                                 | 49 |
| В3. Диаграмма классов.....                                      | 50 |
| В4. Диаграмма развертывания.....                                | 51 |
| В5. Диаграмма последовательности.....                           | 52 |

## ВВЕДЕНИЕ

Целью данной курсовой работы является создание многомодульной системы для обмена сообщениями в компьютерной сети с использованием стека протоколов TCP/IP. Комплекс разделен на клиентскую часть с графическим интерфейсом, вводом сообщений с клавиатуры, и серверную часть, которая обрабатывает приходящие с клиента запросы и в зависимости от них выполняет те или иные действия в сети.

Для реализации был использован следующий стек технологий:

- Стек протоколов TCP/IP как способ передачи данных в рамках сети.
- Язык программирования Java и пакет OpenJDK 26.0.1 для реализации производительной серверной части приложения.
- Библиотека JavaSwing для реализации клиентского приложения.

В ходе курсового проекта будут реализованы:

- Протокол взаимодействия клиента и сервера.
- База данных пользователей, хранящаяся напрямую в программном сервере.
- Отправка текстовых сообщений.

## **1. Теоретическая часть**

### **1.1. Стек протоколов TCP/IP**

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет. Название TCP/IP происходит из двух важнейших протоколов семейства — Transmission Control Protocol (TCP) и Internet Protocol (IP), которые были первыми разработаны и описаны в данном стандарте. Также изредка упоминается как модель DOD (Department of Defense) в связи с историческим происхождением от сети ARPANET из 1970-х годов (под управлением DARPA, Министерства обороны США).

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься. Эта функциональность организована в четыре слоя абстракции, которые классифицируют все связанные протоколы в соответствии с объёмом задействованных сетей. От самого низкого до самого высокого уровня: канальный — это уровень связи, содержащий методы связи для данных, которые остаются в пределах одного сегмента сети; сетевой уровень — обеспечивает межсетевое взаимодействие между независимыми сетями; транспортный уровень — обрабатывает связь между хостами; и прикладной уровень — обеспечивает обмен данными между процессами для приложений.

### 1.1.1. Уровни стека TCP/IP

Стек протоколов TCP/IP включает в себя четыре уровня:

- прикладной уровень (Application Layer);
- транспортный уровень (Transport Layer);
- межсетевой уровень (Сетевой уровень) (Internet Layer);
- канальный уровень (Link Layer).

Протоколы этих уровней полностью реализуют функциональные возможности модели OSI. На стеке протоколов TCP/IP построено всё взаимодействие пользователей в IP-сетях. Стек является независимым от физической среды передачи данных, благодаря чему, в частности, обеспечивается полностью прозрачное взаимодействие между проводными и беспроводными сетями.

Таблица 1 – Распределение протоколов по уровням модели TCP/IP

|                                                            |                                                                                                                                                                                                                                     |
|------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Прикладной<br>(Application Layer)                          | напр., HTTP, RTSP, FTP, DNS                                                                                                                                                                                                         |
| Транспортный<br>(Transport Layer)                          | напр., TCP, UDP, SCTP, DCCP<br>(RIP, протоколы маршрутизации, подобные OSPF, что работают поверх IP, являются частью сетевого уровня)                                                                                               |
| Сетевой<br>(Межсетевой)<br>(Internet Layer)                | Для TCP/IP это IP<br>(вспомогательные протоколы, вроде ICMP и IGMP, работают поверх IP, но тоже относятся к сетевому уровню; протокол ARP является самостоятельным вспомогательным протоколом, работающим поверх канального уровня) |
| Уровень<br>сетевого доступа<br>(Канальный)<br>(Link Layer) | Ethernet, IEEE 802.11, WLAN, SLIP, Token Ring, ATM и MPLS, физическая среда и принципы кодирования информации, T1, E1                                                                                                               |

### 1.1.2. Прикладной уровень

На прикладном уровне (Application layer) работает большинство сетевых приложений.

Эти программы имеют свои собственные протоколы обмена информацией, например, Интернет-браузер для протокола HTTP, ftp-клиент для протокола FTP (передача файлов), почтовая программа для протокола SMTP (электронная почта), SSH (безопасное соединение с удалённой машиной), DNS (преобразование символьных имён в IP-адреса) и многие другие.

В массе своей эти протоколы работают поверх TCP или UDP и привязаны к определённому порту, например:

- HTTP на TCP-порт 80 или 8080;
- FTP на TCP-порт 20 (для передачи данных) и 21 (для управляющих команд);
- SSH на TCP-порт 22;
- запросы DNS на порт UDP (реже TCP) 53;
- обновление маршрутов по протоколу RIP на UDP-порт 520.

Эти порты определены Агентством по выделению имён и уникальных параметров протоколов (IANA).

К этому уровню относятся: Echo, Finger, Gopher, HTTP, HTTPS, IMAP, IMAPS, IRC, NNTP, NTP, POP3, POPS, QOTD, RTSP, SNMP, SSH, Telnet, XDMCP.

### 1.1.3. Транспортный уровень

Протоколы транспортного уровня (Transport layer) могут решать проблему негарантированной доставки сообщений («дошло ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных. В стеке TCP/IP транспортные протоколы определяют, для какого именно приложения предназначены эти данные.

Протоколы автоматической маршрутизации, логически представленные на этом уровне (поскольку работают поверх IP), на самом деле являются частью протоколов сетевого уровня; например OSPF (IP-идентификатор 89).

TCP (IP-идентификатор 6) — «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный поток данных, дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности. В этом его главное отличие от UDP.

UDP (IP-идентификатор 17) протокол передачи датаграмм без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов. В приложениях, требующих гарантированной передачи данных, используется протокол TCP.

UDP обычно используется в таких приложениях, как потоковое видео и компьютерные игры, где допускается потеря пакетов, а повторный запрос затруднён или не оправдан, либо в приложениях вида запрос-ответ (например, запросы к DNS), где создание соединения занимает больше ресурсов, чем повторная отправка.

#### 1.1.4. Сетевой (межсетевой) уровень

Межсетевой уровень (Internet layer) изначально разработан для передачи данных из одной сети в другую. На этом уровне работают маршрутизаторы, которые перенаправляют пакеты в нужную сеть путём расчёта адреса сети по маске сети. Примерами такого протокола является X.25 и IPC в сети ARPANET.

С развитием концепции глобальной сети в уровень были внесены дополнительные возможности по передаче из любой сети в любую сеть, независимо от протоколов нижнего уровня, а также возможность запрашивать данные от удалённой стороны, например в протоколе ICMP (используется для передачи диагностической информации IP-соединения) и IGMP (используется для управления multicast-потоками).

ICMP и IGMP расположены над IP и должны попасть на следующий — транспортный — уровень, но функционально являются протоколами сетевого уровня, и поэтому их невозможно вписать в модель OSI.

Пакеты сетевого протокола IP могут содержать код, указывающий, какой именно протокол следующего уровня нужно использовать, чтобы извлечь данные из пакета. Это число — уникальный IP-номер протокола. ICMP и IGMP имеют номера, соответственно, 1 и 2. К этому уровню относятся: DVMRP, ICMP, IGMP, MARS, PIM, RIP, RIP2, RSVP.

#### 1.1.5. Канальный уровень

Канальный уровень (англ. Link layer) описывает способ кодирования данных для передачи пакета данных на физическом уровне (то есть специальные последовательности бит, определяющих начало и конец пакета данных, а также обеспечивающие помехоустойчивость). Ethernet, например, в полях заголовка пакета содержит указание того, какой машине или машинам в сети предназначен этот пакет.

Примеры протоколов канального уровня — Ethernet, IEEE 802.11 (WLAN), SLIP, Token Ring, ATM и MPLS.

PPP не совсем вписывается в такое определение, поэтому обычно описывается в виде пары протоколов HDLC/SDLC.

MPLS занимает промежуточное положение между канальным и сетевым уровнем и, строго говоря, его нельзя отнести ни к одному из них.

Канальный уровень иногда разделяют на 2 подуровня — LLC и MAC.

Кроме того, канальный уровень описывает среду передачи данных (будь то коаксиальный кабель, витая пара, оптическое волокно или радиоканал), физические характеристики такой среды и принцип передачи данных (разделение каналов, модуляцию, амплитуду сигналов, частоту сигналов, способ синхронизации передачи, время ожидания ответа и максимальное расстояние).

При проектировании стека протоколов на канальном уровне рассматривают помехоустойчивое кодирование — позволяющие обнаруживать и исправлять ошибки в данных вследствие воздействия шумов и помех на канал связи.

## **1.2. Взаимодействие с протоколом TCP/IP на уровне программ**

### **1.2.1. История появления сокетов**

Спецификация сокетов Windows определяет интерфейс сетевого программирования, совместимый с двоичными файлами для Microsoft Windows. Сокеты Windows основаны на реализации сокетов UNIX в дистрибутиве программного обеспечения Berkeley (BSD, выпуск 4.3) из Калифорнийского университета в Беркли. Спецификация включает как подпрограммы сокетов в стиле BSD, так и расширения, относящиеся к Windows. Использование сокетов Windows позволяет приложению взаимодействовать в любой сети, соответствующей API сокетов Windows. На платформе Win32 сокет Windows обеспечивает потокобезопасность.

Многие поставщики сетевого программного обеспечения поддерживают сокет Windows среди сетевых протоколов, включая протокол управления передачей/интернет-протокол (TCP/IP), сетевую систему Xerox (XNS), протокол DECNet корпорации Digital Equipment Corporation, протоколы Интернет-пакетной/последовательной пакетной передачи корпорации Novell (IPX/SPX) и другие. Хотя текущая спецификация сокетов Windows определяет абстракции сокетов для TCP/IP, любой сетевой протокол может соответствовать сокетам Windows, предоставив собственную версию библиотеки динамического канала (DLL), которая реализует сокет Windows. Примеры коммерческих приложений, написанных с помощью сокетов Windows, включают X серверы Windows, эмуляторы терминалов и электронные почтовые системы.

### 1.2.2. Понятие сокета и их классификация

Сокет — это конечная точка связи — объект, через который приложение сокетов Windows отправляет или получает пакеты данных в сети. Сокет имеет тип и связан с запущенным процессом, и он может иметь имя. В настоящее время сокеты обычно обмениваются данными только с другими сокетами в том же «домене связи», который использует Набор протоколов Интернета.

Оба типа сокетов являются двунаправленными; они представляют собой потоки данных, с которыми можно обмениваться данными в обоих направлениях одновременно (полный дуплекс).

Доступны два типа сокета:

- Потоковые сокеты. Сокеты потоков предоставляют поток данных без границ записи: поток байтов. Потоки гарантированно доставляются и должны быть правильно расположены по порядку и не дублируются.

- Сокеты датаграмм. Сокеты датаграмм поддерживают поток данных с ориентированием на записи, который не гарантируется к доставке и может не следовать в той же последовательности, как отправлено, или может содержать дубликаты.

«Последовательность» означает, что пакеты доставляются в порядке отправки. «Без дублирования» означает, что вы получаете определенный пакет только один раз.

### 1.2.3. Тип данных SOCKET

Каждый объект сокета MFC инкапсулирует дескриптор объекта Windows Sockets. Тип данных этого дескриптора — **SOCKET**. Дескриптор **SOCKET** аналогичен **HWND** для окна. Классы сокета MFC предоставляют операции с инкапсулированным дескриптором.

Тип данных **SOCKET** подробно описан в пакете SDK для Windows. См. раздел "Тип данных сокета и значения ошибок" в сокетах Windows.

## 2. Практическая часть

### 2.1. Постановка задачи

Для начала необходимо определиться с функционалом клиентской и серверной части проекта. Клиент должен иметь возможность подключаться к серверу с фиксированным IP-адресом и отключаться от него, отправлять и получать сообщения.

Сервер должен иметь возможность обрабатывать входящие запросы, видеть всех подключенных клиентов, а также пересылать поступающие сообщения другим подключенным клиентам, или же всем подключенным клиентам, также должна быть предусмотрена возможность логирования всех событий, происходящих в сети.

Все эти возможности можно проиллюстрировать на диаграмме вариантов использования (Рисунок 1):



Рисунок 1 – Диаграмма вариантов использования

Обработка поступающих на сервер запросов должна быть поэтапной. Первой частью запроса должен быть его тип. Всего будет 4 типа запросов:

- Подключение пользователя;
- Отключение пользователя;
- Отправка личного сообщения;
- Отправка широковещательного сообщения;

Обработку запроса можно проиллюстрировать на диаграмме деятельности (Рисунок 2):

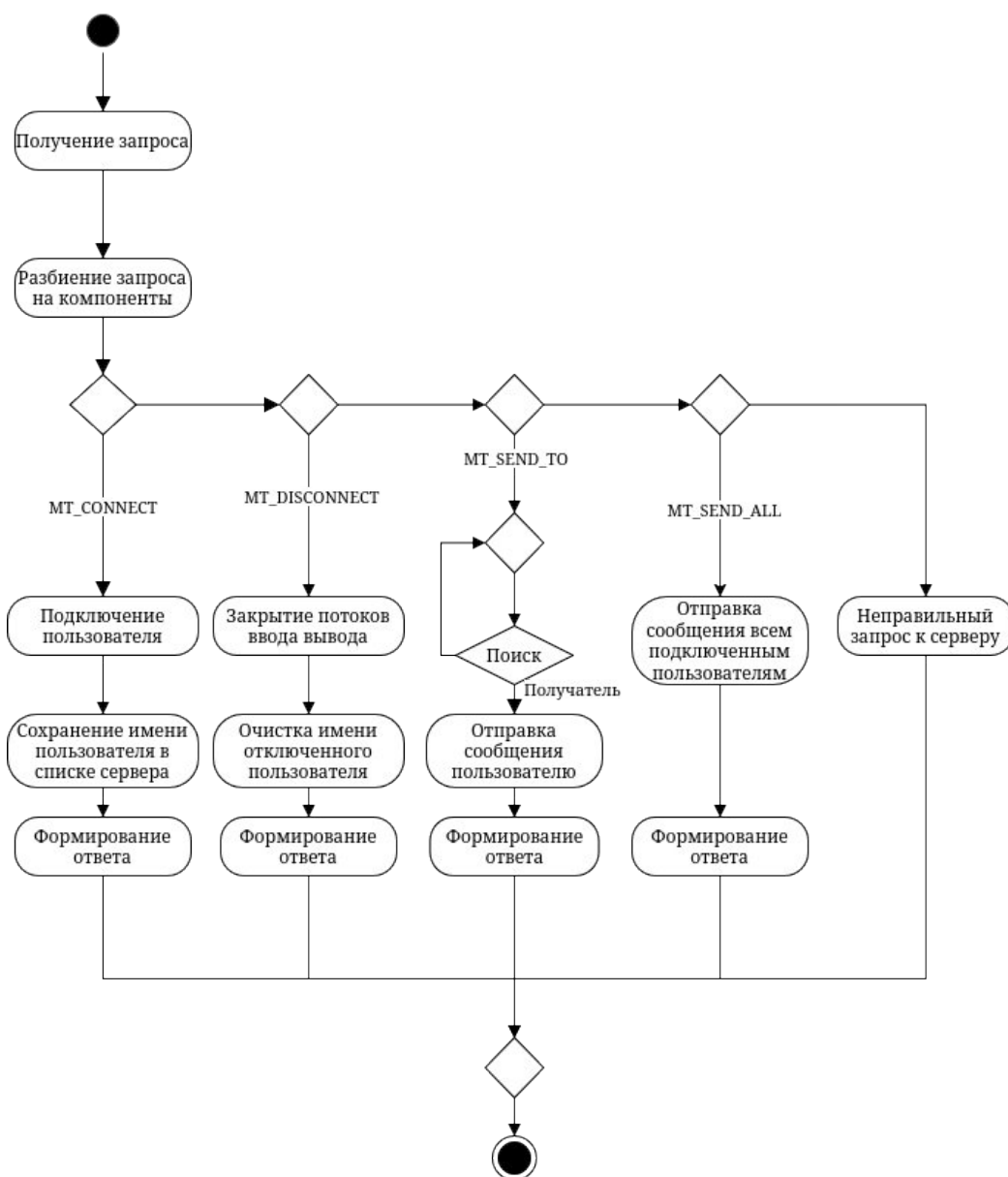


Рисунок 2 – Диаграмма деятельности (обработка запроса на сервере)

Исходя из задания, в качестве сетевой архитектуры напрашивается топология «Звезда» (Рисунок 3):

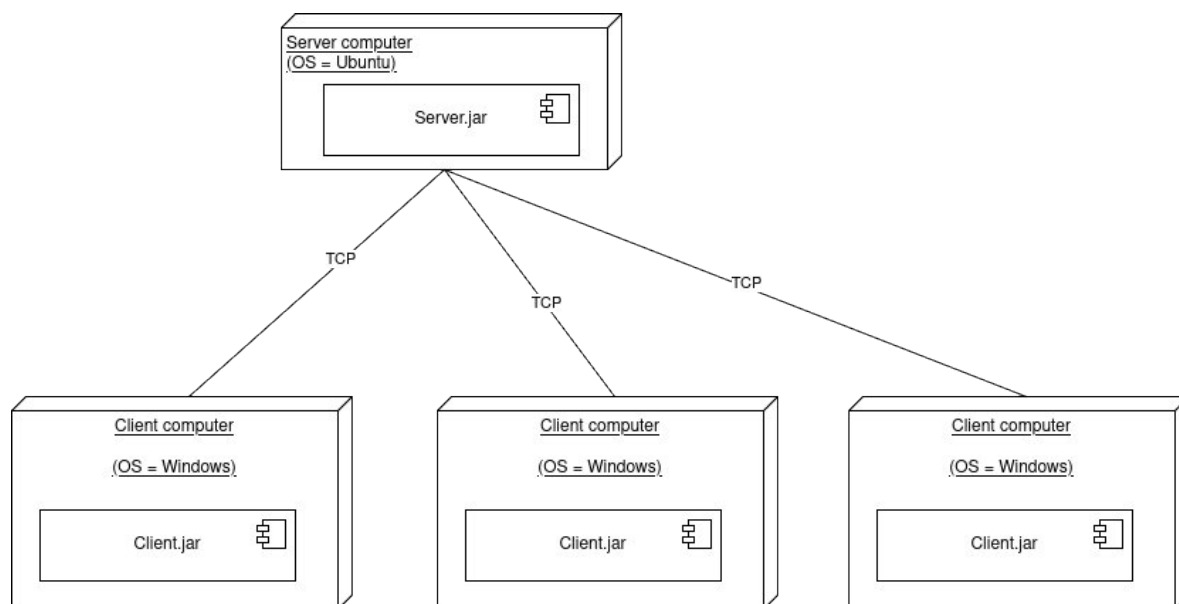


Рисунок 3 – Диаграмма развертывания сетевой архитектуры

Далее приведена диаграмма классов и диаграмма последовательности проекта, в ней показаны сущности, задействованные как на клиенте, так и на сервере.

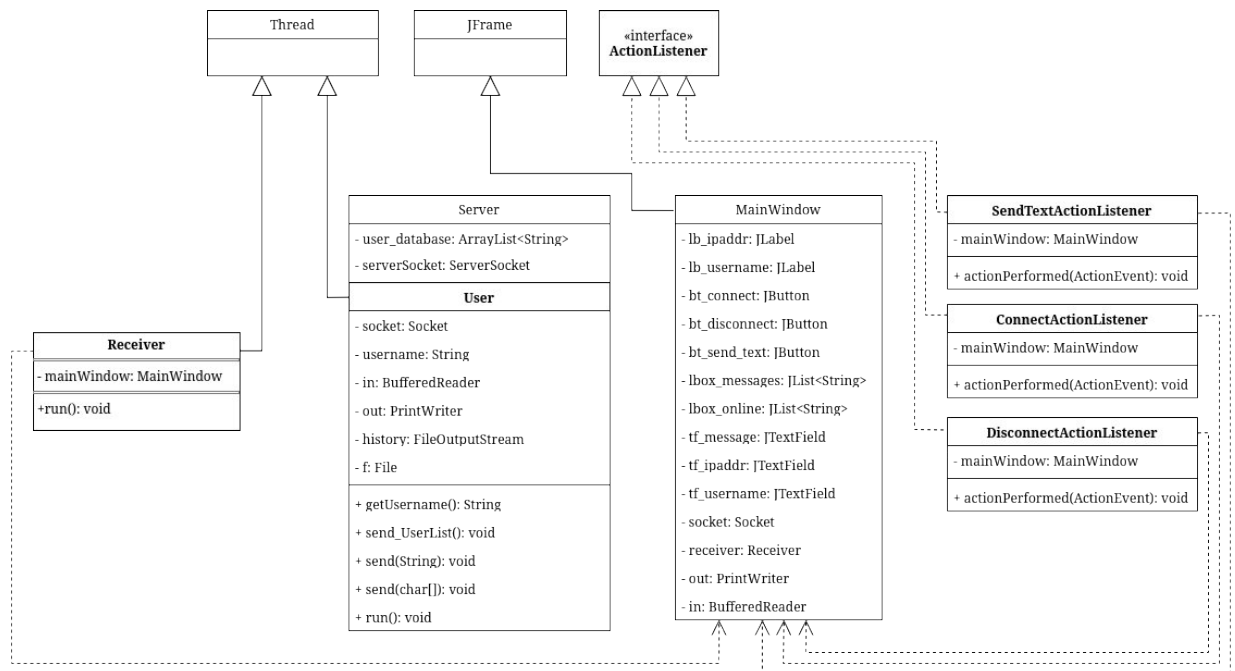


Рисунок 4 – Диаграмма классов проекта

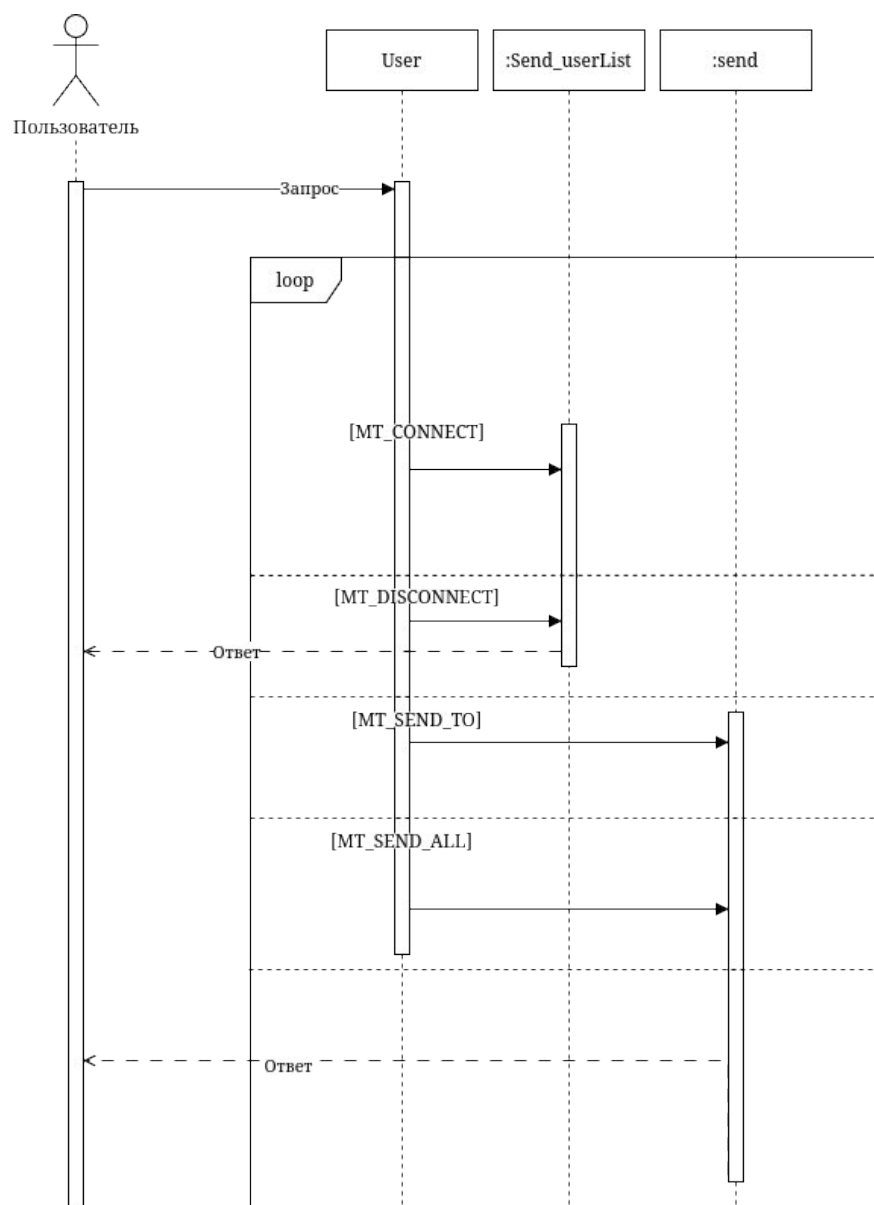


Рисунок 5 – Диаграмма последовательности (Отправка запроса и получение ответа)

## 2.2. Выбор решения

Для реализации поставленной задачи будет использоваться следующий стек технологий:

- Библиотека JavaSwing для реализации графического интерфейса;
- Библиотека Java net для доступа сокетам;

## 2.3. Организация пользовательского интерфейса

На следующем рисунке показано окно клиентского приложения.

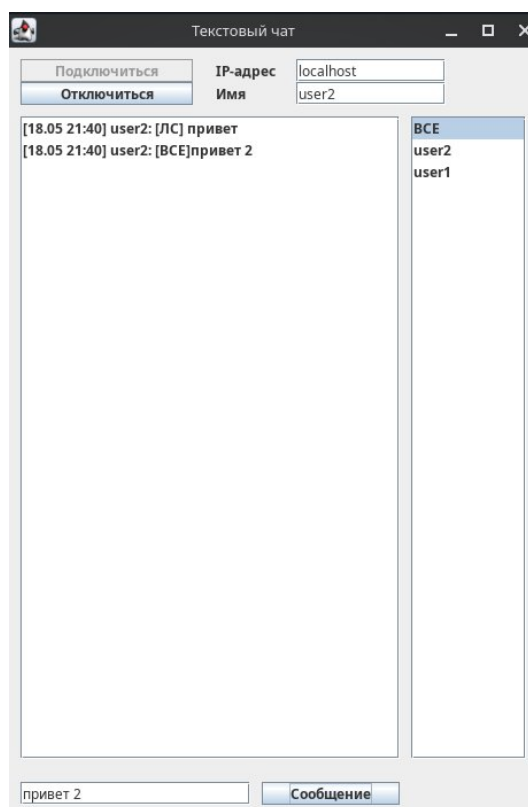


Рисунок 6 – Окно приложения

В левом столбце отображаются подключенные пользователи, в самом крупном окне показаны сообщения с датой и временем отправки. . В поле IP-адрес можно вписать статический IP-адрес компьютера, на котором запущена программа сервер, также опытным путем было выявлено, что поле поддерживает доменные имена.

## 2.4. Описание работы

Вся логика работы сервера описана в классе User. Класс User содержит в себе ссылку на серверный сокет, пишущий клиентский сокет, потоки ввода и вывода, статическое поле файла для логирования событий. Этот класс наследуется от класса Thread и в нем переопределен метод run(). В методе run реализована обработка запроса по принципу схемы на рисунке 2.

### 2.4.1. Подключение и отключение

Для корректной обработки подключений, отключений и отправки сообщений сервер использует клиентские потоки. В каждом потоке подключен пишущий сокет клиента, поток ввода и вывода. Все потоки создаются и запускаются в конструкторе класса Server.

```
public class Server {
    static ServerSocket serverSocket;
    ArrayList<User> user_database;

    Server(int capacity) {
        this.user_database = new ArrayList<>(capacity);
        try {
            serverSocket = new ServerSocket(5000);
            for (int i = 0; i < capacity; i++) {
                user_database.add(new User(this));
                user_database.getLast().start();
            }
        } catch (
            Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

Кнопка «Подключиться» посылает на сервер запрос MT\_CONNECT вместе с именем пользователя. На сервере происходит установка соединения и запись имени пользователя в поле username, а затем отправка списка пользователей всем подключенным клиентам с помощью метода send\_UserList().

#### Клиентский код:

```
public void actionPerformed(ActionEvent e) {
    try {
        this.mainWindow.socket = new Socket(this.mainWindow.tf_ipaddr.getText(), 5000);
        this.mainWindow.in = new BufferedReader(new
InputStreamReader(this.mainWindow.socket.getInputStream()));
        this.mainWindow.out = new PrintWriter(this.mainWindow.socket.getOutputStream(),
true);

        this.mainWindow.receiver = new Receiver(this.mainWindow);
        String request = "MT_CONNECT|" + this.mainWindow.tf_username.getText();
        this.mainWindow.out.println(request);

        this.mainWindow.bt_connect.setEnabled(false);
        this.mainWindow.bt_disconnect.setEnabled(true);
        this.mainWindow.bt_send_file.setEnabled(true);
        this.mainWindow.bt_send_text.setEnabled(true);
        this.mainWindow.receiver.start();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this.mainWindow.getRootPane(), ex.getMessage());
    }
}
```

#### Серверный код:

```
case "MT_CONNECT":
    this.username = answer[1];
    System.out.printf("[%s] %s: \t %s подключен.\n",
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),
        answer[0],
        username);
    history.write(String.format("[%s] %s: \t %s подключен.\n",
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),
        answer[0],
        username).getBytes());

    for (var user : this.server.user_database) {
        user.send_UserList();
    }
    break;

public void send_UserList() {
    if (this.out == null) {
        return;
    }
    String user_list = "";
    for (var user : this.server.user_database) {
        if (user.getUsername() == "")
            continue;
        user_list += user.getUsername() + "|";
    }
    this.out.println("MT_CONNECT|" + user_list);
}
```

Кнопка «Отключиться» отправляет на сервер специальный запрос MT\_DISCONNECT, после чего на сервере клиентский поток закрывает потоки ввода-вывода и перезапускается путем вызова метода run().

## Клиентский код:

```
public void actionPerformed(ActionEvent e) {
    try {
        this.mainWindow.out.println("MT_DISCONNECT");
        this.mainWindow.in.close();
        this.mainWindow.out.close();
        this.mainWindow.socket.close();
        this.mainWindow.bt_connect.setEnabled(true);
        this.mainWindow.bt_disconnect.setEnabled(false);
        this.mainWindow.default_lm_messages.removeAllElements();
        this.mainWindow.bt_send_file.setEnabled(false);
        this.mainWindow.bt_send_text.setEnabled(false);

    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this.mainWindow.getRootPane(), ex.getMessage());
    }
}
```

## Серверный код:

```
case "MT_DISCONNECT":
    /*
     * Закрытие потоков ввода вывода
     */
    this.in.close();
    this.out.close();
    System.out.printf("[%s] %s: \t %s отключен.\n",
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
        answer[0],
        username);
    history.write(String.format("[%s] %s: \t %s отключен.\n",
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
        answer[0],
        username).getBytes());
    this.username = null;
    this.out.println("MT_DISCONNECT");
    for (var user : this.server.user_database) {
        user.send_UserList();
    }
    run();
    break;
```

Если же закрыть окно приложения через крестик, то метод потока `in` на сервере выбросит `IOException`, что позволяет в блоке `catch` освободить ресурсы и также заново запустить клиентский поток.

```
catch (
    Exception ex) {
    /* При отключении перезапуск клиентского потока */
    try {
        System.out.printf("[%s]%s:\t%s отключен.\n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")), "MT_DISCONNECT",
        username);
        history.write(String.format("[%s]%s:\t%s отключен.\n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")), "MT_DISCONNECT",
        username).getBytes());
        this.in.close();
        this.out.close();
        this.socket.close();
        this.username = null;
        this.run();
    } catch (Exception e) {
    }
}
```

## 2.4.2. Отправка и получение сообщений

Отправка личных и публичных текстовых сообщений происходит при помощи запросов `MT_SEND_TO` и `MT_SEND_ALL` соответственно. При отправке личного сообщения происходит линейный поиск имени получателя в базе данных, а затем отправка сообщения на пишущий сокет найденного пользователя. Для отправки сообщения на сокет другого потока был реализован метод `send()`.

Клиентский код:

```
public void actionPerformed(ActionEvent e) {
    String send_user = this.mainWindow.lbox_online.getSelectedValue();
    if (send_user == null || send_user == "")
        this.mainWindow.out.println(String.format("%s|%s|%s",
            "MT_SEND_ALL",
            "MSG_TEXT",
            this.mainWindow.tf_message.getText()));
    else
        this.mainWindow.out.println(String.format("%s|%s|%s|%s",
            "MT_SEND_TO",
            "MSG_TEXT",
            send_user,
            this.mainWindow.tf_message.getText()));
}
```

Получение сообщений происходит в классе Receiver в отдельном потоке.

```
class Receiver extends Thread {
    /**
     *
     */
    private final MainWindow mainWindow;

    Receiver(MainWindow mainWindow) {
        super();
        this.mainWindow = mainWindow;
    };

    public void run() {
        String sender, message;
        while (true) {
            try {
                String[] answer = this.mainWindow.in.readLine().split("\\s*\\\\\\\\s*");
                switch (answer[0]) {
                    case "MT_CONNECT":
                        this.mainWindow.default_lm_online.removeAllElements();
                        for (int i = 1; i < answer.length; i++) {
                            this.mainWindow.default_lm_online.addElement(answer[i]);
                        }
                        break;
                    case "MT_DISCONNECT":
                        return;
                    case "MSG_TEXT":
                        sender = answer[1];
                        message = answer[2];
                        this.mainWindow.default_lm_messages
                            .addElement(String.format("[%s] %s: %s",
                                LocalDateTime.now()
                                    .format(DateTimeFormatter.ofPattern("dd.MM
HH:mm"))),
                                sender, message));
                        break;
                    case "MSG_FILE":
                        break;
                    default:
                        break;
                }
            } catch (Exception e1) {
                try {
                    this.mainWindow.default_lm_online.removeAllElements();
                } catch (Exception e2) {
                }
                break;
            }
        }
    }
}
```

## Серверный код:

```
/* Отправка текстового сообщения */
public void send(String message) {
    if (this.out == null)
        return;
    this.out.println(message);
}

case "MT_SEND_TO":
    send_user = answer[2];
    message = answer[3];
    System.out.printf("[%s] %s: \t %s -> %s: %s\n",
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss"))),
        answer[0],
        username,
        send_user, message);
    history.write(String.format("[%s] %s: \t %s -> %s: %s\n",
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss"))),
        answer[0],
        username,
        send_user, message).getBytes());
    switch (answer[1]) {
        case "MSG_TEXT":
            for (var user : this.server.user_database) {
                if (user.getUsername().equals(send_user)) {
                    user.send(
                        String.format("%s| %s| %s", answer[1], this.username,
                            message));
                    break;
                }
            }
            break;
        case "MSG_FILE":
            /* Отправка байтов чанками */
            break;
    }
    break;
```

Отличие широковещательной отправки отличается лишь в том, что сообщение отправляется всем подключенным пользователям при помощи метода `send()`.

```
case "MT_SEND_ALL":
    switch (answer[1]) {
        case "MSG_TEXT":
            message = answer[2];
            System.out.printf("[%s] %s:\n",
                LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
                answer[0]);
            history.write(String.format("[%s] %s:\n",
                LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
                answer[0]).getBytes());
            for (var user : this.server.user_database) {
                System.out.printf("\t%s -> %s: %s\n",
                    username,
                    user.getUsername(),
                    message);
                history.write(String.format("\t%s -> %s: %s\n",
                    username,
                    user.getUsername(),
                    message).getBytes());
                user.send(String.format("%s| %s| %s", answer[1], this.username,
                    "[BCE]" + message));
            }
            break;
        case "MSG_FILE":
            /* Отправка байтов чанками */
            break;
    }
    break;
```

### 3. Отладка и тестирование приложения

Тестирование приложения будет проводиться по следующим сценариям:

1. Попытка подключиться к серверу, если он не запущен.
2. Отключение от сервера через кнопку «Отключиться»
3. Отключение от сервера через закрытие приложения.
4. Отправка личного сообщения
5. Отправка широковещательного сообщения.

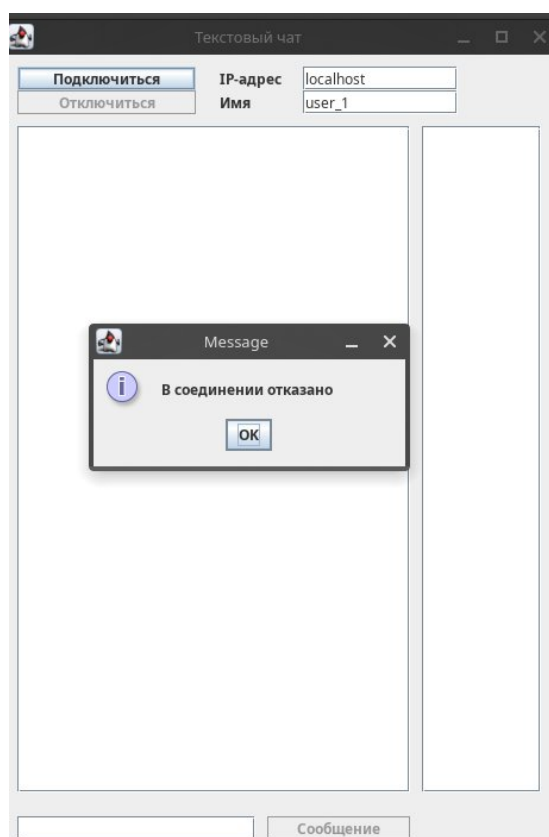


Рисунок 7 – Попытка подключиться к не запущенному серверу

Так как сервер не запущен, конструктор клиентского сокета выбрасывает исключение «В соединении отказано».

```
→ Java_Kurs git:(master) cd /home/cent2/Projects/Java_Kurs ; /usr/bin/env /usr/lib/jvm/java-26-openjdk/bin/jav
a -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:37769 -XX:+ShowCodeDetailsInExceptionM
essages -cp /home/cent2/.config/Code\ -\ OSS\User/workspaceStorage/47fa694ecdc181d7e689a3451d0feb63/redhat.java/
jdt_ws/Java_Kurs_2a1569f9/bin Server.App
41: ожидание подключения...
42: ожидание подключения...
38: ожидание подключения...
39: ожидание подключения...
40: ожидание подключения...
█
```

Рисунок 8 – Запуск сервера

The screenshot shows an IDE with a Java class named `User` that extends `Thread`. The `run()` method implements a loop to handle incoming connections. It uses `ServerSocket` to accept connections on port 329. For each connection, it creates a `BufferedReader` and a `PrintWriter`. It then enters a `while (true)` loop to read messages. A `switch` statement handles the `MT_CONNECT` message by setting the username to `user_1` and logging the connection. The terminal output shows the server running and waiting for connections, with a log entry: `[17.05.2026 15:38:38]MT_CONNECT: user_1 подключен.`

Рисунок 9 – Обработка запроса на подключение MT\_CONNECT

При подключении нового пользователя происходит отправка всех подключенных пользователей на клиентские компьютеры.

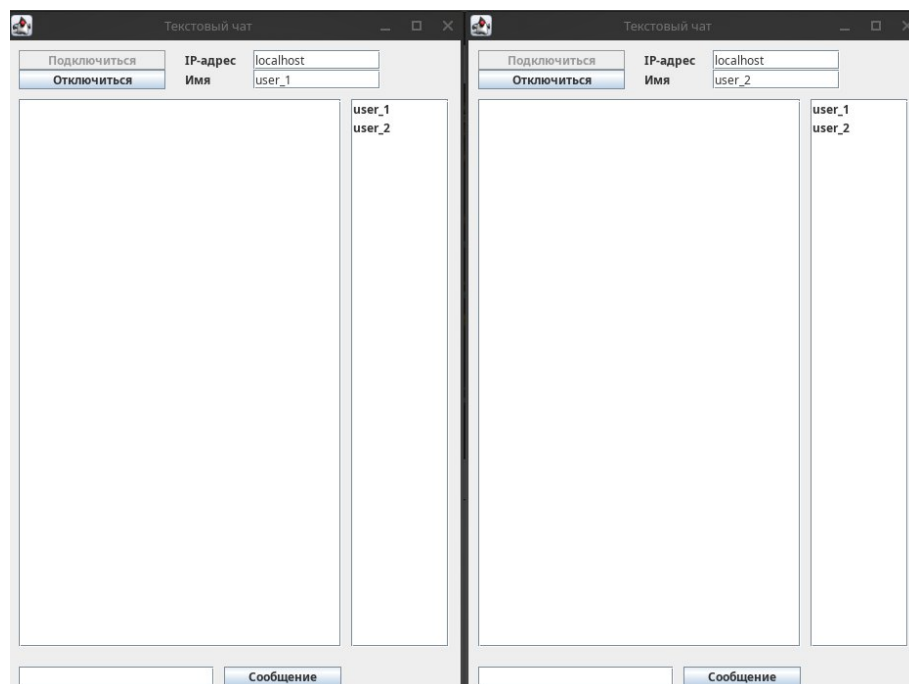


Рисунок 10 – Демонстрация списка подключенных пользователей

```
101 for (var user : this.server.user_database) { server = Server@83
102 user.send_UserList();
103 }
104 break;
105 case "MT_DISCONNECT":
106     /*
107     * Закрытие потоков ввода вывода, возможность повторного подключения ???
108     * Сохранение учетной записи пользователя в базе ???
109     */
110     this.in.close(); in = BufferedReader@228
111     this.out.close(); out = PrintWriter@218
112
113     System.out.printf("[%s] %s: \t %s отключен.\n",
114         LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
115         answer[0], answer = String[1]@227
116         username); username = null
117     history.write(String.format("[%s] %s: \t %s отключен.\n", history = FileOutputStream@62
118         LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
119         answer[0], answer = String[1]@227
120         username).getBytes()); username = null
121     this.username = null; username = null
122     this.out.println("MT_DISCONNECT"); out = PrintWriter@218
123     for (var user : this.server.user_database) { server = Server@83
124         user.send_UserList();
125     }
126
127 run();
128 break;
129 case "MT_SEND_TO":
130     send_user = answer[2];
131     message = answer[3];
132     System.out.printf("[%s] %s: \t %s -> %s: %s\n",
133         LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
134         ...
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
➔ Java_Kurs git:(master) cd /home/cent2/Projects/Java_Kurs ; /usr/bin/env /usr/lib/jvm/j
a -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:37769 -XX:+ShowC
[17.05.2026 15:50:57]MT_DISCONNECT: user_1 отключен.
41: ожидание подключения...
[17.05.2026 15:51:19]MT_CONNECT: user_1 подключен.
[17.05.2026 15:51:23]MT_DISCONNECT: user_1 отключен.
```

Рисунок 11 – Обработка отключения пользователя MT\_DISCONNECT

```

181         }
182     }
183     break;
184     default:
185         System.out.println("Неправильный запрос к серверу:");
186         for (var i : answer) {
187             System.out.println(i);
188         }
189     }
190 }
191 } catch (
192
193 Exception ex) { ex = NullPointerException@389 "java.lang.NullPointerException: Cannot invoke "String.spl
194
195 /* При отключении перезапуск клиентского потока */
196 try {
197     System.out.printf("[%s] %s: \t %s отключен.\n",
198         LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")), "MT_DISC
199         username); username = null
200     history.write(String.format("[%s] %s: \t %s отключен.\n", history = FileOutputStream@62
201         LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")), "MT_DISC
202         username).getBytes()); username = null
203     this.in.close(); in = BufferedReader@296
204     this.out.close(); out = PrintWriter@294
205     this.socket.close(); socket = Socket@295 "Socket[addr=/127.0.0.1,port=39566,localport=5000]"
206     this.username = null; username = null
207     this.run();
208 } catch (Exception e) {
209 }
210 }
211 }
212 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

→ Java_Kurs git:(master) cd /home/cent2/Projects/Java_Kurs ; /usr/bin/env /usr/lib/jvm/j
a -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:37769 -XX:+ShowC
38: ожидание подключения...
[17.05.2026 15:58:14]MT_CONNECT: user_1 подключен.
[17.05.2026 15:58:18]MT_CONNECT: user_2 подключен.
[17.05.2026 15:58:53]MT_DISCONNECT: user_1 отключен.

```

Рисунок 12 – Обработка отключения пользователя через блок catch

```

case "MT_SEND_TO":
    send_user = answer[2]; send_user = "user_2", answer = String[4]@86
    message = answer[3]; message = "12345", answer = String[4]@86
    System.out.printf("[%s] %s: \t %s -> %s: %s\n",
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
        answer[0], answer = String[4]@86
        username, username = "user_1"
        send_user, message); send_user = "user_2", message = "12345"
    history.write(String.format("[%s] %s: \t %s -> %s: %s\n", history = FileOutputStream@78
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
        answer[0], answer = String[4]@86
        username, username = "user_1"
        send_user, message).getBytes()); send_user = "user_2", message = "12345"
    switch (answer[1]) { answer = String[4]@86
        case "MSG_TEXT":
            for (var user : this.server.user_database) { user = User@17 "Thread[#38,Thread-0,5,main]", server = Server@76
                if (user.getUsername().equals(send_user)) { user = User@17 "Thread[#38,Thread-0,5,main]", send_user = "user_2"
                    user.send(
                        String.format("%s %s %s", answer[1], this.username, "[ЛС] " + message));
                    break;
                }
            }
        }
    }
    break;

```

Рисунок 13 – Обработка отправки личного сообщения

```

73     public void run() {
74         break;
75         case "MSG_FILE":
76             /* Отправка байтов чанками */
77             break;
78     }
79     break;
80     case "MT_SEND_ALL":
81         switch (answer[1]) { answer = String[3]@155
82             case "MSG_TEXT":
83                 message = answer[2]; message = "привет", answer = String[3]@155
84                 System.out.printf("[%s] %s: \n",
85                     LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
86                     answer[0]); answer = String[3]@155
87                 history.write(String.format("[%s] %s: \n", history = FileOutputStream@78
88                     LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss")),
89                     answer[0]).getBytes()); answer = String[3]@155
90                 for (var user : this.server.user_database) {
91                     System.out.printf("\t %s -> %s: %s\n",
92                         username,
93                         user.getUsername(),
94                         message);
95                     history.write(String.format("\t %s -> %s: %s\n",
96                         username,
97                         user.getUsername(),
98                         message).getBytes());
99                     user.send(String.format("%s %s %s", answer[1], this.username, "[BCE]" + message));
100                 }
101             }
102         }
103     }
104     break;

```

VARIABLES

Local

```

> answer = String[3]@155
> message = "привет"
> this = User@17 "Thread[#38,Thread-0,5,main]"

```

WATCH

```

+ - [x] [x] [x]
> answer = String[3]@155
> 0 = "MT_SEND_ALL"
> 1 = "MSG_TEXT"
> 2 = "привет"

```

Рисунок 14 – Обработка отправки широковещательного сообщения

```
12 [17.05.2026 15:38:35]MT_CONNECT: user_1 подключен.
13 [17.05.2026 15:42:06]MT_CONNECT: user_2 подключен.
14 [17.05.2026 15:50:18]MT_DISCONNECT: user_2 отключен.
15 [17.05.2026 15:50:32]MT_CONNECT: user_2 подключен.
16 [17.05.2026 15:50:59]MT_DISCONNECT: user_1 отключен.
17 [17.05.2026 15:51:19]MT_CONNECT: user_1 подключен.
18 [17.05.2026 15:51:25]MT_DISCONNECT: user_1 отключен.
19 [17.05.2026 15:57:30]MT_CONNECT: user_1 подключен.
20 [17.05.2026 15:57:33]MT_DISCONNECT: user_1 отключен.
21 [17.05.2026 15:58:02]MT_DISCONNECT: user_2 отключен.
22 [17.05.2026 15:58:14]MT_CONNECT: user_1 подключен.
23 [17.05.2026 15:58:18]MT_CONNECT: user_2 подключен.
24 [17.05.2026 15:58:56]MT_DISCONNECT: user_1 отключен.
25 [17.05.2026 16:01:17]MT_CONNECT: user_1 подключен.
26 [17.05.2026 16:02:52]MT_SEND_TO: user_1 -> user_2: 12345
27 [17.05.2026 16:03:26]MT_CONNECT: user_2 подключен.
28 [17.05.2026 16:03:58]MT_DISCONNECT: user_2 отключен.
29 [17.05.2026 16:04:08]MT_CONNECT: user_1 подключен.
30 [17.05.2026 16:04:13]MT_CONNECT: user_2 подключен.
31 [17.05.2026 16:04:49]MT_SEND_TO: user_1 -> user_2: 12345
32 [17.05.2026 16:07:22]MT_SEND_ALL:
33 user_2 -> user_2: привет
34 user_2 -> : привет
35 user_2 -> user_1: привет
36 user_2 -> : привет
37 user_2 -> : привет
38
```

Рисунок 15 – Демонстрация логирования

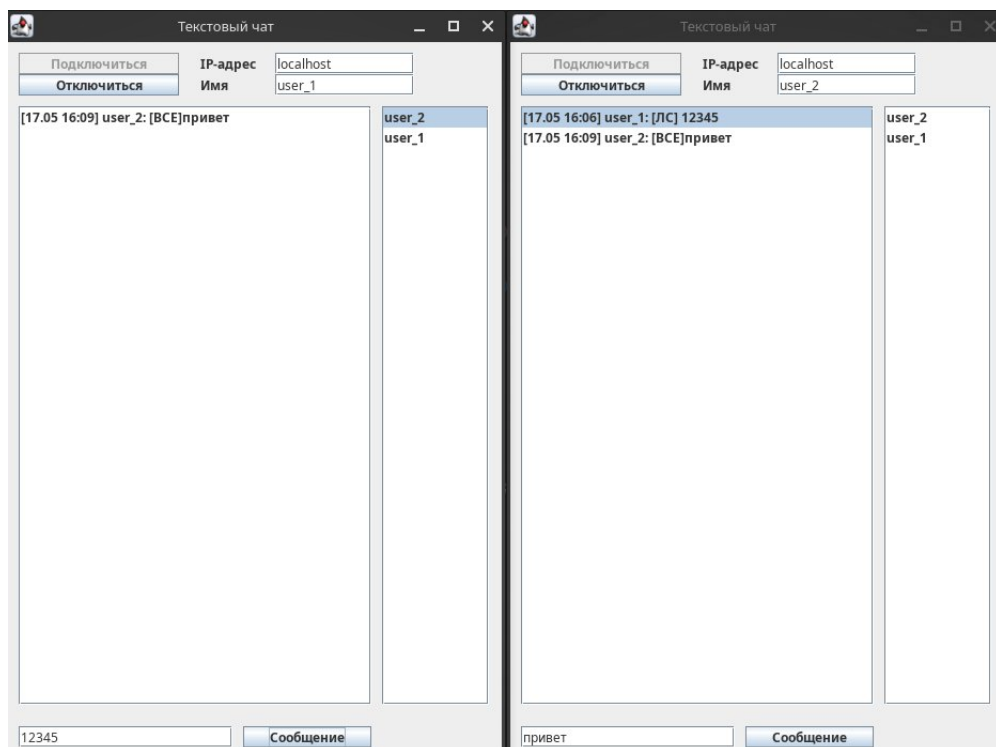


Рисунок 16 – Демонстрация получения сообщений

## ЗАКЛЮЧЕНИЕ

В ходе курсовой работы было разработано многомодульное приложение по обмену сообщениями в компьютерной сети. Для достижения цели курсовой работы был изучен теоретический материал по стеку протоколов TCP/IP, а также сокеты для взаимодействия с ним.

Для взаимодействия обеих программ был разработан протокол, содержащий 5 команд. В протоколе описаны базовые команды, которые сервер и клиент должны выполнять.

Была реализована клиентская часть с использованием библиотеки JavaSwing, которая в зависимости от действий пользователя формирует запрос для сервера.

На языке Java был реализован модуль сервера, который может принимать и обрабатывать запросы от клиента, передавать текстовые сообщения.

В дальнейшем есть перспективы расширения системы путем реализации отправки файлов через байтовые потоки, более надежного учета клиентов в базе данных SQL. Также есть возможность реализовать регистрацию нового пользователя через учетную запись администратора, реализация групп пользователей. Подводя итоги, можно сказать, что задание к курсовой работе выполнено и цель достигнута.

## СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Руководство по Java [Электронный ресурс] информационный ресурс metanit.com. Режим доступа – <https://metanit.com/java/tutorial/>. (Дата обращения 01.05.2026).
2. Swing (Java) [Электронный ресурс] информационный ресурс en.wikipedia.org. Режим доступа – [https://en.wikipedia.org/wiki/Swing\\_\(Java\)](https://en.wikipedia.org/wiki/Swing_(Java)). (Дата обращения 01.05.2026).
3. Классы Socket и ServerSocket в Java [Электронный ресурс] информационный ресурс javarush.com. Режим доступа – <https://javarush.com/groups/posts/654-klassih-socket-i-serversocket-ili-allo-server-tih-menja-slihshishjh>. (Дата обращения 01.05.2026).
4. TCP [Электронный ресурс] информационный ресурс ru.wikipedia.org. Режим доступа – <https://ru.wikipedia.org/wiki/TCP>. (Дата обращения 01.05.2026).
5. TCP/IP [Электронный ресурс] информационный ресурс ru.wikipedia.org. Режим доступа – [https://ru.wikipedia.org/wiki/TCP/IP#%D0%A3%D1%80%D0%BE%D0%B2%D0%BD%D0%B8\\_%D1%81%D1%82%D0%B5%D0%BA%D0%B0\\_TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP#%D0%A3%D1%80%D0%BE%D0%B2%D0%BD%D0%B8_%D1%81%D1%82%D0%B5%D0%BA%D0%B0_TCP/IP). (Дата обращения 01.05.2026).

**ПРИЛОЖЕНИЕ А**  
**(обязательное)**  
**Исходный код сервера**

## A1. Файл App.java

```
package Server;

public class App {
    public static void main(String[] args) throws Exception {
        new Server(Integer.parseInt(args[0]));
    }
}
```

## A2. Файл Server.java

```
package Server;

import java.net.*;
import java.util.ArrayList;

public class Server {
    static ServerSocket serverSocket;
    ArrayList<User> user_database;

    Server(int capacity) {
        this.user_database = new ArrayList<>(capacity);
        try {
            serverSocket = new ServerSocket(5000);
            for (int i = 0; i < capacity; i++) {
                user_database.add(new User(this));
                user_database.getLast().start();
            }
        } catch (
            Exception e) {
            System.err.println(e.getMessage());
        }
    }
}
```

### A3. Файл User.java

```
package Server;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/* Класс пользователя */
class User extends Thread {
    /**
     *
     */
    private final Server server;
    private Socket socket;
    private String username;
    BufferedReader in;
    PrintWriter out;
    static File f;

    static FileOutputStream history;

    User(Server server) {
        this.server = server;
        f = new File("./history.log");

        try {
            history = new FileOutputStream(f, true);
        } catch (Exception e) {
            e.getMessage();
        }
        super();
    }

    public String getUsername() {
        if (this.username == null)
            return "";
        else
            return this.username;
    }

    public void send_UserList() {
        if (this.out == null) {
            return;
        }
        String user_list = "";
        for (var user : this.server.user_database) {
            if (user.getUsername() == "")
                continue;
            user_list += user.getUsername() + "|";
        }
    }
}
```

```

        this.out.println("MT_CONNECT|" + user_list);
    }

    /* Отправка текстового сообщения */
    public void send(String message) {
        if (this.out == null)
            return;
        this.out.println(message);
    }

    /* Отправка последовательности байт. */
    public void send(char[] message) {
        if (this.out == null)
            return;
        this.out.println(message);
    }

    public void run() {
        try {
            /*
             * MT_CONNECT: Обработка подключения к серверу
             * MT_SEND_TO: Отправка сообщения пользователю
             * MT_SEND_ALL: Отправка сообщения всем пользователям в
сети
             * MSG_TEXT: Текстовое сообщение
             * MSG_FILE: Последовательность байт
             */
            System.out.println(threadId() + ": ожидание
подключения...");
            this.socket = Server.serverSocket.accept();
            in = new BufferedReader(new
InputStreamReader(this.socket.getInputStream()));
            out = new PrintWriter(this.socket.getOutputStream(),
true);

            while (true) {
                String[] answer = in.readLine().split("\\s*\\|\\s*");
                String send_user; // Получатель
                String message; // Сообщение
                switch (answer[0]) {
                    case "MT_CONNECT":
                        this.username = answer[1];
                        System.out.printf("[%s] %s: \t %s подключен.\n",
LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),
                                answer[0],
                                username);
                        history.write(String.format("[%s] %s: \t %s
подключен.\n",
LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),
                                answer[0],
                                username).getBytes());

                        for (var user : this.server.user_database) {
                            user.send_UserList();

```

```

        }
        break;
    case "MT_DISCONNECT":
        /*
         * Закрытие потоков ввода вывода, возможность
повторного подключения ???
         * Сохранение учетной записи пользователя в
базе ???
         */
        this.in.close();
        this.out.close();

        System.out.printf("[%s] %s: \t %s отключен. \n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),

            answer[0],
            username);
        history.write(String.format("[%s] %s: \t %s
отключен. \n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),

            answer[0],
            username).getBytes());
        this.username = null;
        this.out.println("MT_DISCONNECT");
        for (var user : this.server.user_database) {
            user.send_UserList();
        }
        run();
        break;
    case "MT_SEND_TO":
        send_user = answer[2];
        message = answer[3];
        System.out.printf("[%s] %s: \t %s -> %s: %s \n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),

            answer[0],
            username,
            send_user, message);
        history.write(String.format("[%s] %s: \t %s
-> %s: %s \n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),

            answer[0],
            username,
            send_user, message).getBytes());
        switch (answer[1]) {
            case "MSG_TEXT":
                for (var user :
this.server.user_database) {
                    if
(user.getUsername().equals(send_user)) {
                        user.send(

```

```

String.format("%s|%s|%s", answer[1], this.username, "[JC] " +
message));

                                break;
                                }
                                }
                                break;
                                case "MSG_FILE":
                                    /* Отправка байтов чанками */
                                    break;
                                }
                                break;
                                case "MT_SEND_ALL":
                                    switch (answer[1]) {
                                        case "MSG_TEXT":
                                            message = answer[2];
                                            System.out.printf("[%s] %s:\n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),

                                                answer[0]);

history.write(String.format("[%s] %s:\n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),

                                                answer[0]).getBytes());
                                for (var user :
this.server.user_database) {
                                    System.out.printf("\t%s
-> %s: %s\n",

                                                username,
                                                user.getUsername(),
                                                message);
                                    history.write(String.format("\t%s
-> %s: %s\n",

                                                username,
                                                user.getUsername(),
                                                message).getBytes());

user.send(String.format("%s|%s|%s", answer[1], this.username, "[BCE]"
+ message));

                                }
                                break;
                                case "MSG_FILE":
                                    /* Отправка байтов чанками */
                                    break;
                                }
                                break;
                                default:
                                    System.out.println("Неправильный запрос к
серверу:");
                                    for (var i : answer) {
                                        System.out.println(i);
                                    }
                                }
                                }
}

```

```

    } catch (
        Exception ex) {

        /* При отключении перезапуск клиентского потока */
        try {
            System.out.printf("[%s] %s: \t %s отключен.\n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")), "MT_DISCONNECT",
                username);
            history.write(String.format("[%s] %s: \t %s отключен.\n",

LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")), "MT_DISCONNECT",
                username).getBytes());
            this.in.close();
            this.out.close();
            this.socket.close();
            this.username = null;
            this.run();
        } catch (Exception e) {
        }
    }
}
}

```

**ПРИЛОЖЕНИЕ Б**  
(обязательное)  
Исходный код клиентского приложения

## Б1. Файл App.java

```
package Client;
public class App {
    public static void main(String[] args) throws Exception {
        MainWindow window = new MainWindow();
        window.setVisible(true);
    }
}
```

## Б2. Файл ConnectActionListener.java

```
package Client;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

import javax.swing.JOptionPane;

public class ConnectActionListener implements ActionListener {

    /**
     *
     */
    private final MainWindow mainWindow;

    /**
     * @param mainWindow
     */
    ConnectActionListener(MainWindow mainWindow) {
        this.mainWindow = mainWindow;
    }

    public void actionPerformed(ActionEvent e) {
        try {
            this.mainWindow.socket = new
Socket(this.mainWindow.tf_ipaddr.getText(), 5000);
            this.mainWindow.in = new BufferedReader(new
InputStreamReader(this.mainWindow.socket.getInputStream()));
            this.mainWindow.out = new
PrintWriter(this.mainWindow.socket.getOutputStream(), true);
            this.mainWindow.receiver = new Receiver(this.mainWindow);
            String request = "MT_CONNECT|" +
this.mainWindow.tf_username.getText();
            this.mainWindow.out.println(request);

            this.mainWindow.bt_connect.setEnabled(false);
            this.mainWindow.bt_disconnect.setEnabled(true);
            this.mainWindow.bt_send_file.setEnabled(true);
            this.mainWindow.bt_send_text.setEnabled(true);
            this.mainWindow.receiver.start();
        } catch (Exception ex) {

JOptionPane.showMessageDialog(this.mainWindow.getRootPane(),
ex.getMessage());
        }

    }
}
```

### Б3. Файл DisconnectActionListener.java

```
package Client;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JOptionPane;

public class DisconnectActionListener implements ActionListener {

    /**
     *
     */
    private final MainWindow mainWindow;

    /**
     * @param mainWindow
     */
    DisconnectActionListener(MainWindow mainWindow) {
        this.mainWindow = mainWindow;
    }

    public void actionPerformed(ActionEvent e) {
        try {
            this.mainWindow.out.println("MT_DISCONNECT");
            this.mainWindow.in.close();
            this.mainWindow.out.close();
            this.mainWindow.socket.close();
            this.mainWindow.bt_connect.setEnabled(true);
            this.mainWindow.bt_disconnect.setEnabled(false);
            this.mainWindow.default_lm_messages.removeAllElements();
            this.mainWindow.bt_send_file.setEnabled(false);
            this.mainWindow.bt_send_text.setEnabled(false);

        } catch (Exception ex) {

            JOptionPane.showMessageDialog(this.mainWindow.getRootPane(),
                ex.getMessage());
        }
    }
}
```

## Б4. Файл MainWindow.java

```
package Client;

import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class MainWindow extends JFrame {
    /* Надписи. */
    JLabel lb_ipaddr, lb_username;
    /* Кнопки */
    JButton bt_connect, bt_disconnect, bt_send_text, bt_send_file;
    /* Текстовые поля ввода */
    JTextField tf_message, tf_ipaddr, tf_username;
    /* Окно сообщений */
    JList<String> lbox_messages, lbox_online;
    DefaultListModel<String> default_lm_messages, default_lm_online;
    /* Панель прокрутки */
    JScrollPane jsp_messages, jsp_online;

    Container container = super.getContentPane();
    Socket socket;
    Receiver receiver;

    /* Потоки ввода вывода для отправки и передачи сообщений */
    PrintWriter out;
    BufferedReader in;

    MainWindow() {
        super("Текстовый чат");
        super.setBounds(20, 20, 470, 700);
        super.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        /* Установка надписей */
        lb_ipaddr = new JLabel("IP-адрес");
        lb_username = new JLabel("Имя");
        lb_ipaddr.setBounds(180, 10, 150, 20);
        lb_username.setBounds(180, 30, 150, 20);
        /* Установка кнопок */
        bt_connect = new JButton("Подключиться");
        bt_disconnect = new JButton("Отключиться");
        bt_send_text = new JButton("Сообщение");
        bt_send_file = new JButton("Файл");
        bt_connect.setBounds(10, 10, 150, 20);
        bt_disconnect.setBounds(10, 30, 150, 20);
        bt_send_text.setBounds(220, 640, 120, 20);
        bt_send_file.setBounds(350, 640, 100, 20);

        /* Установка текстовых полей ввода */
        tf_message = new JTextField();
        tf_ipaddr = new JTextField("localhost");
        tf_username = new JTextField("user");
        tf_message.setBounds(10, 640, 200, 20);
    }
}
```

```

tf_ipaddr.setBounds(250, 10, 130, 20);
tf_username.setBounds(250, 30, 130, 20);

/* Установка границ окна сообщений и окна онлайн */
default_lm_messages = new DefaultListModel<String>();
default_lm_online = new DefaultListModel<String>();
lbox_messages = new JList<String>(default_lm_messages);
lbox_online = new JList<String>(default_lm_online);
jsp_messages = new JScrollPane(lbox_messages);
jsp_messages.setBounds(10, 60, 330, 560);

jsp_online = new JScrollPane(lbox_online);
jsp_online.setBounds(350, 60, 100, 560);
/* Добавление элементов на главное окно. */
container.setLayout(null);
container.add(lb_ipaddr);
container.add(lb_username);
container.add(bt_connect);
container.add(bt_disconnect);
container.add(bt_send_text);
// container.add(bt_send_file);
container.add(tf_message);
container.add(tf_ipaddr);
container.add(tf_username);
container.add(jsp_messages);
container.add(jsp_online);
/* Обработчики событий */
var SendText_al = new SendTextActionListener(this);
var SendFile_al = new SendFileActionListener(this);
var Connect_al = new ConnectActionListener(this);
var Disconnect_al = new DisconnectActionListener(this);

bt_connect.addActionListener(Connect_al);
bt_disconnect.addActionListener(Disconnect_al);
bt_send_text.addActionListener(SendText_al);
bt_send_file.addActionListener(SendFile_al);
bt_connect.setEnabled(true);
bt_disconnect.setEnabled(false);
bt_send_file.setEnabled(false);
bt_send_text.setEnabled(false);

}

}

```

## Б5. Файл Receiver.java

```
package Client;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

class Receiver extends Thread {
    /**
     *
     */
    private final MainWindow mainWindow;

    Receiver(MainWindow mainWindow) {
        super();
        this.mainWindow = mainWindow;
    };

    public void run() {
        String sender, message;
        while (true) {
            try {
                String[] answer =
this.mainWindow.in.readLine().split("\\s*\\|\\s*");
                switch (answer[0]) {
                    case "MT_CONNECT":

this.mainWindow.default_lm_online.removeAllElements();
this.mainWindow.default_lm_online.addElement("BCE");
                    for (int i = 1; i < answer.length; i++) {

this.mainWindow.default_lm_online.addElement(answer[i]);
                    }
                    break;
                    case "MT_DISCONNECT":
                        return;
                    case "MSG_TEXT":
                        sender = answer[1];
                        message = answer[2];
                        this.mainWindow.default_lm_messages
                            .addElement(String.format("[%s] %s: %s",
                                LocalDateTime.now()
                                    .format(DateTimeFormatter.ofP
attern("dd.MM HH:mm")),
                                sender, message));
                        break;
                    case "MSG_FILE":

                        break;
                    default:
                        break;
                }
            } catch (Exception e1) {
                try {
                    this.mainWindow.default_lm_online.removeAllElements();
                } catch (Exception e2) {

                }
                break;
            }
        }
    }
}
```

## Б6. Файл SendTextActionListener.java

```
package Client;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SendTextActionListener implements ActionListener {

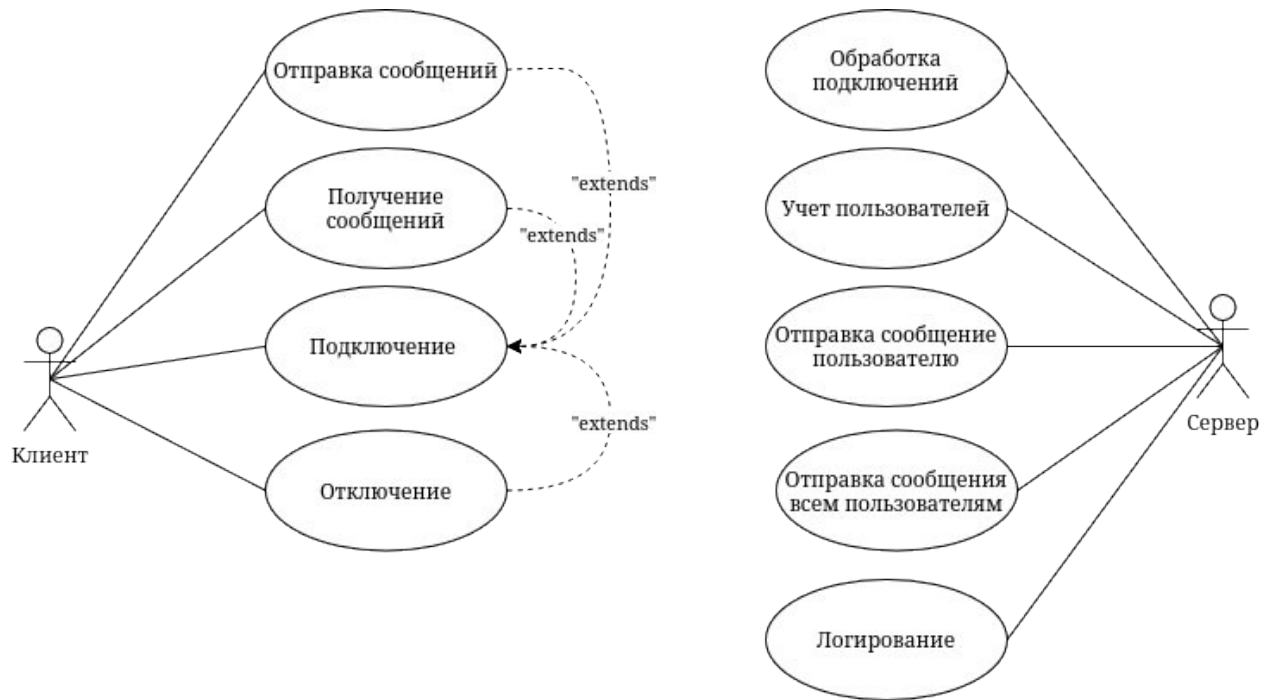
    /**
     *
     */
    private final MainWindow mainWindow;

    /**
     * @param mainWindow
     */
    SendTextActionListener(MainWindow mainWindow) {
        this.mainWindow = mainWindow;
    }

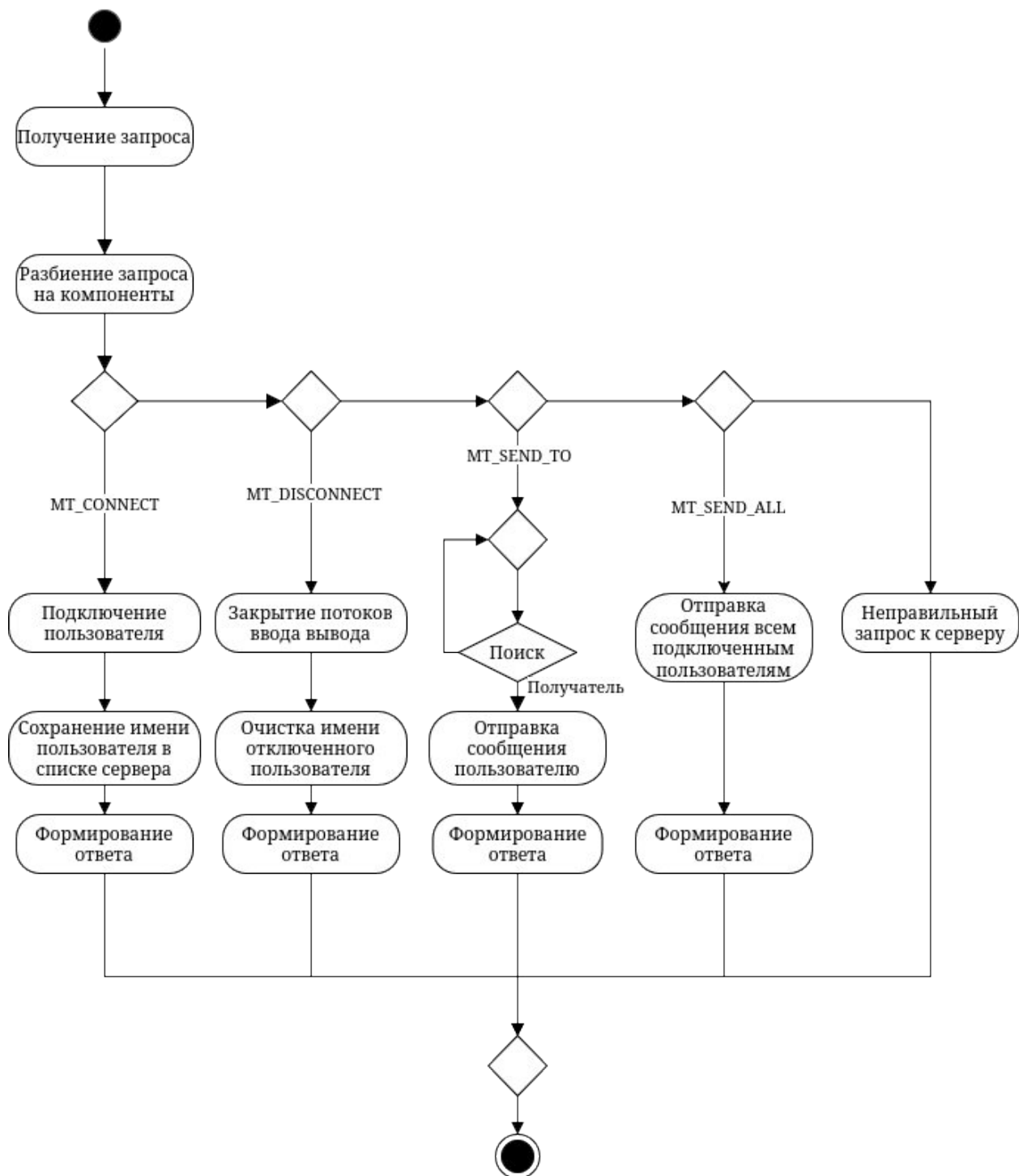
    public void actionPerformed(ActionEvent e) {
        String send_user =
this.mainWindow.lbox_online.getSelectedValue();
        if (send_user == null || send_user == "BCE")
            this.mainWindow.out.println(String.format("%s|%s|%s",
                "MT_SEND_ALL",
                "MSG_TEXT",
                this.mainWindow.tf_message.getText()));
        else
            this.mainWindow.out.println(String.format("%s|%s|%s|%s",
                "MT_SEND_TO",
                "MSG_TEXT",
                send_user,
                this.mainWindow.tf_message.getText()));
    }
}
```

**ПРИЛОЖЕНИЕ В**  
(обязательное)  
UML-диаграммы протокола

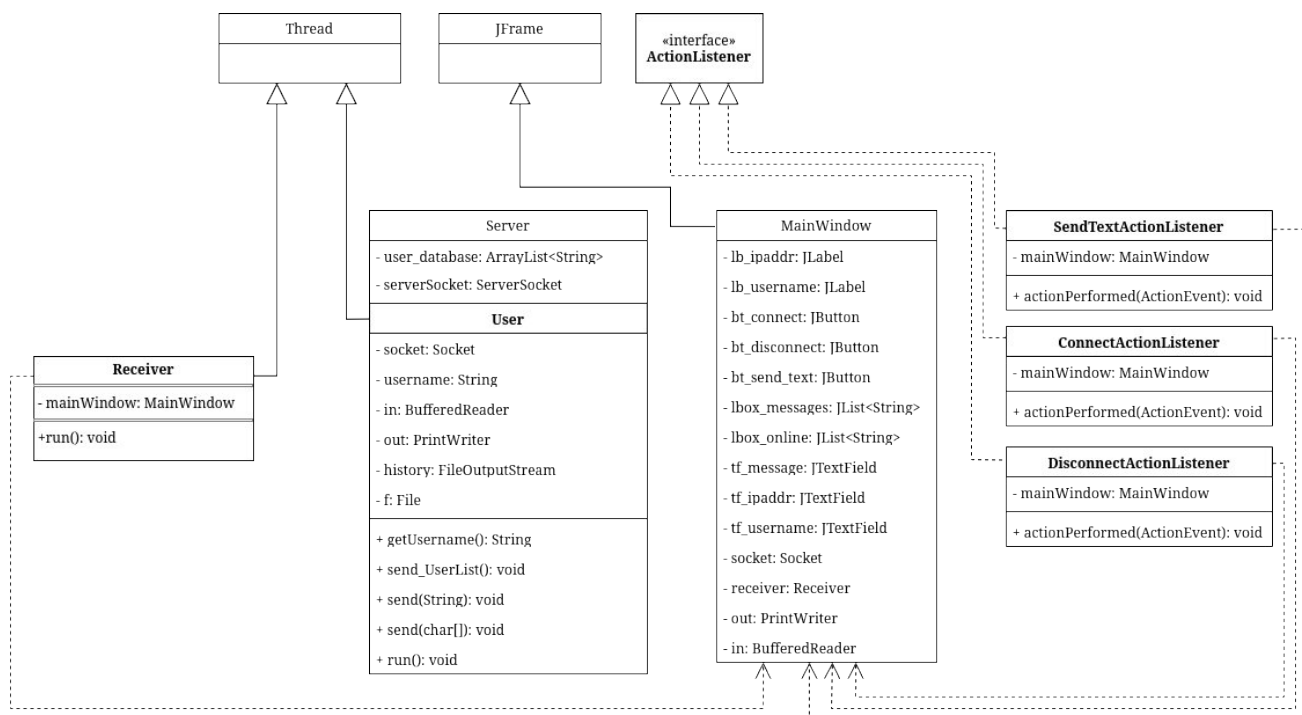
## В1. Диаграмма вариантов использования



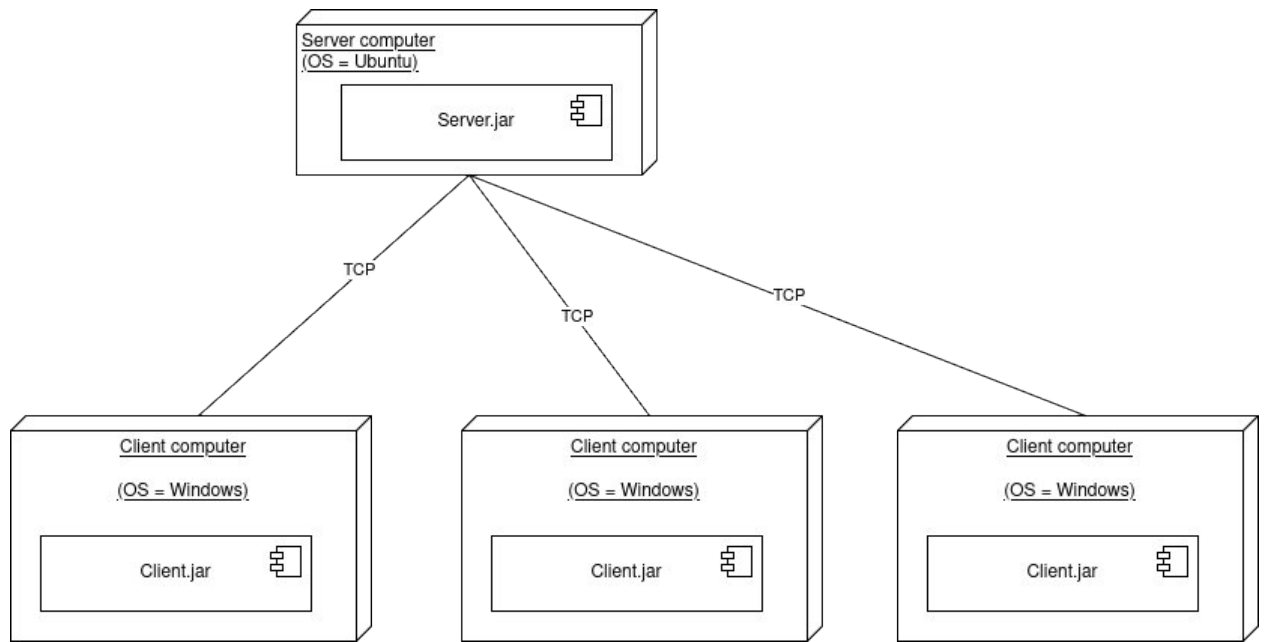
## В2. Диаграмма деятельности



### В3. Диаграмма классов



#### В4. Диаграмма развертывания



## В5. Диаграмма последовательности

