Министерство науки высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовой работе

по курсу «Программирование на языке Java»

на тему «Разработка многомодульного приложения на языке Java»

Выполнил студент группы 21ВВП1:

Кабачек Н.А.

Приняла:

Юрова О.В.

Пенза 2024

# Содержание

[Содержание 2](#_Toc167368202)

[Введение 3](#_Toc167368203)

[1. Постановка задачи 4](#_Toc167368204)

[2. Выбор решения 5](#_Toc167368205)

[3. Описание протокола взаимодействия. 6](#_Toc167368206)

[4. Архитектура сервера 8](#_Toc167368207)

[5. Архитектура клиента 9](#_Toc167368208)

[6. Описание работы программы 10](#_Toc167368209)

[Заключение 13](#_Toc167368210)

[Литература 14](#_Toc167368211)

[Приложение А. Листинг программы 15](#_Toc167368212)

[Приложение А.1. Файл «Server.java». 15](#_Toc167368213)

[Приложение А.2. Файл «ServerObject.java». 15](#_Toc167368214)

[Приложение А.3. Файл «ClientObject.java». 19](#_Toc167368215)

[Приложение А.4. Файл «Client.java». 20](#_Toc167368216)

[Приложение А.5. Файл «ClientObject.java». 21](#_Toc167368217)

[Приложение А.6. Файл «LoginFormController.java». 24](#_Toc167368218)

[Приложение А.7. Файл «MessengerFormController.java». 26](#_Toc167368219)

[Приложение Б. UML-диаграммы 30](#_Toc167368220)

[Приложение Б.1. UML – диаграмма вариантов использования. 30](#_Toc167368221)

[Приложение Б.2. UML – диаграмма классов. 30](#_Toc167368222)

[Приложение Б.3. UML – диаграмма деятельности. 32](#_Toc167368223)

[Приложение Б.4. UML – диаграмма развертывания. 33](#_Toc167368224)

[Приложение Б.5. UML – диаграмма последовательности. 33](#_Toc167368225)

# Введение

«Клиент - сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

# 1. Постановка задачи

Необходимо разработать систему программ клиент-серверной архитектуры, позволяющую пользователю общаться с другими пользователями при помощи текста.

Функции клиента:

● Обеспечение интерфейса с пользователем

● Взаимодействие с сервером

● Регистрация на сервере

Функции сервера:

● Управление трафиком данных

● Взаимодействие с программами-клиентами

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.

ОС – Windows. Язык программирования: Java. Среда разработки NetBeans.

# 2. Выбор решения

Важной частью курсовой работы является передача сообщений от клиента к серверу. Для реализации этого используются TCP-сокеты. TCP гарантирует доставку пакетов, их очередность, автоматически разбивает данные на пакеты и контролирует их передачу, в отличии от UDP. ПротоколTCP был выбран так как нам необходимо гарантировать доставку данных, UDP.

Для взаимодействия между компьютерами используются адреса и порты. Адрес представляет собой 32-битную структуру. Номер порта – целое число от 0 до 65535.33

# 3. Описание протокола взаимодействия.

Запускаем сервер и создаем сокет. Устанавливаем локальную точку для прослушивания подключений. Запускаем прослушивание входящих подключений и принимаем новых клиентов и запросы от них в бесконечном цикле. После того, как клиент или запрос был принят, другой запрос отправляется обратно клиенту. На этом установление соединения между клиентом и сервером заканчивается. После сервер в бесконечном цикле ожидает других команд от клиента.

Диаграмма последовательности находиться в [Приложение Б.5](#_Приложение_Б.5._UML).

Всего будет реализовано два типа сообщений протокола обмена клиента с сервером. Первый состоит только из команды:

“/close” – клиент может отправить такую команду серверу, когда требуется закрыть соединение.

“/online” – клиент запрашивает список пользователей, находящихся в сети.

“#leave” – сервер получает сообщение о том, что клиент вернулся на страницу авторизации.

Второй тип протокола обмена состоит из команды и сообщения, разделенные специальным символом ‘|’:

“/online|name,name…:id,id…” – после авторизации клиент получает список пользователей, находящихся онлайн.

“#new|Name:Id” – клиент получает уведомление о том, что появился новый пользователь в сети.

“#reg|name login password” – клиент отправляет информацию серверу с данными, указанными в окне регистрации.

“#log|login password” – клиент отправляет информацию серверу с данными, указанными в окне авторизации.

“#reg|accept” и “#reg|reject” – сервер отправляет флаг об успешной/неудачной регистрации.

“#log|online” – сервер отправляет клиенту флаг о том, что пользователь с указанными данными уже находится в сети.

“#log|Name” и “#log|reject” – сервер отправляет имя клиенту в случае успешной авторизации, иначе - “reject”.

“#leave|Id” – клиент получает уведомление о том, что пользователь вышел из сети.

“%Message” – клиент получил сообщение от сервера.

“&Id|Message” – клиент получил сообщение от пользователя.

“@TargetId:SenderId|Message” – клиент получил сообщение от пользователя в общем чате.

“@Id|message” – отправка конкретному пользователю по Id. Такое сообщение клиент отправляет серверу.

# 4. Архитектура сервера

В курсовой работе для организации взаимодействия сервера с клиентами используется следующий подход: при запуске сервера происходит подключение к базе данных, затем сервер начинает ждать ответ от каждого клиента.

При поступлении запросов от клиентов, он ищет совпадение с базой данных и отвечает на основе существующей у него информации.

Диаграмма классов сервера находиться в [Приложение Б.2](#_Приложение_Б.2._UML).

# 5. Архитектура клиента

После запуска клиента, он сразу же пытается установить соединение с сервером. После того, как соединение было установлено, пользователь может войти в свой аккаунт или создать новый.

После входа в аккаунт пользователь может отправить сообщение в общий чат или выбрать собеседника из списка и отправить ему личное сообщение. Так же клиент может вернуться на страницу авторизации, выйдя из сети из текущего профиля.

Диаграмма классов клиента находиться в [Приложение Б.2](#_Приложение_Б.2._UML).

# 6. Описание работы программы

Ниже представлена работа клиент-серверного приложения.

Для начала необходимо запустить сервер:

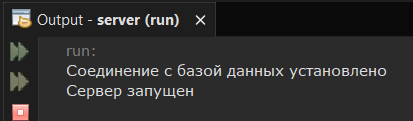


Рисунок 1 - Сервер запущен

Запускаем клиента(-ов). Открывается окно авторизации:

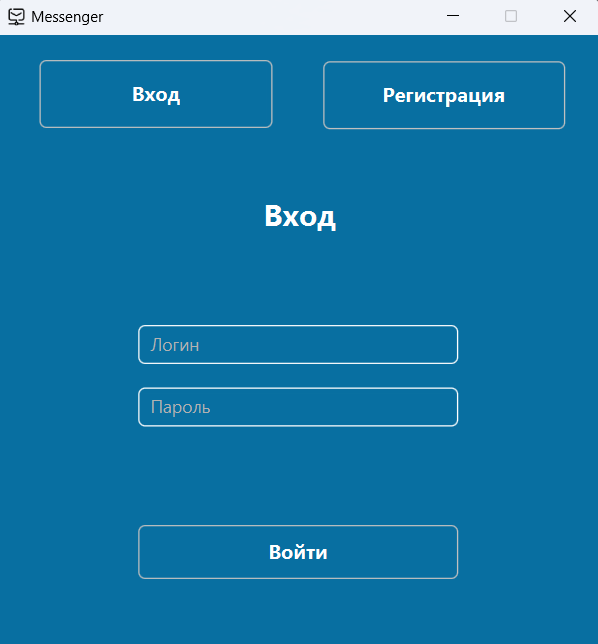
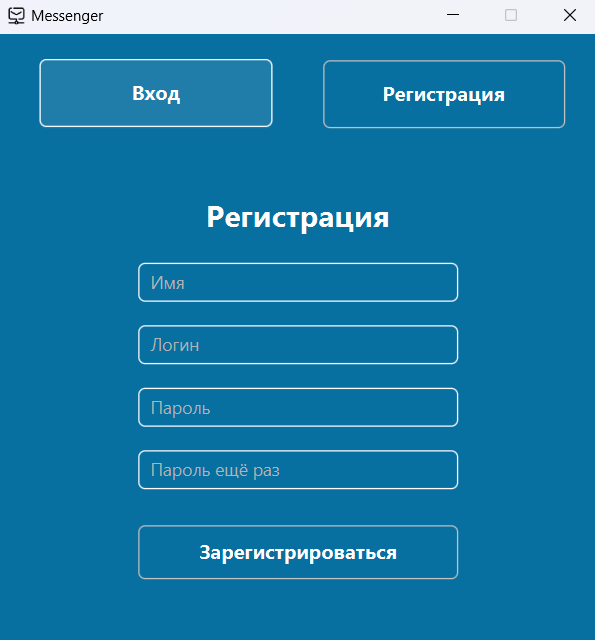
 

Рисунок 2-3 - Авторизация

Информация о пользователях хранится в базе данных PostgreSQL:

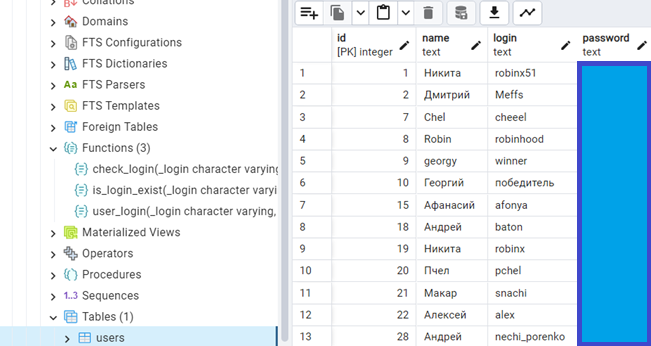


Рисунок 4 – База данных

Регистрируем нового пользователя или заходим в существующий аккаунт. Открывается окно мессенджера:

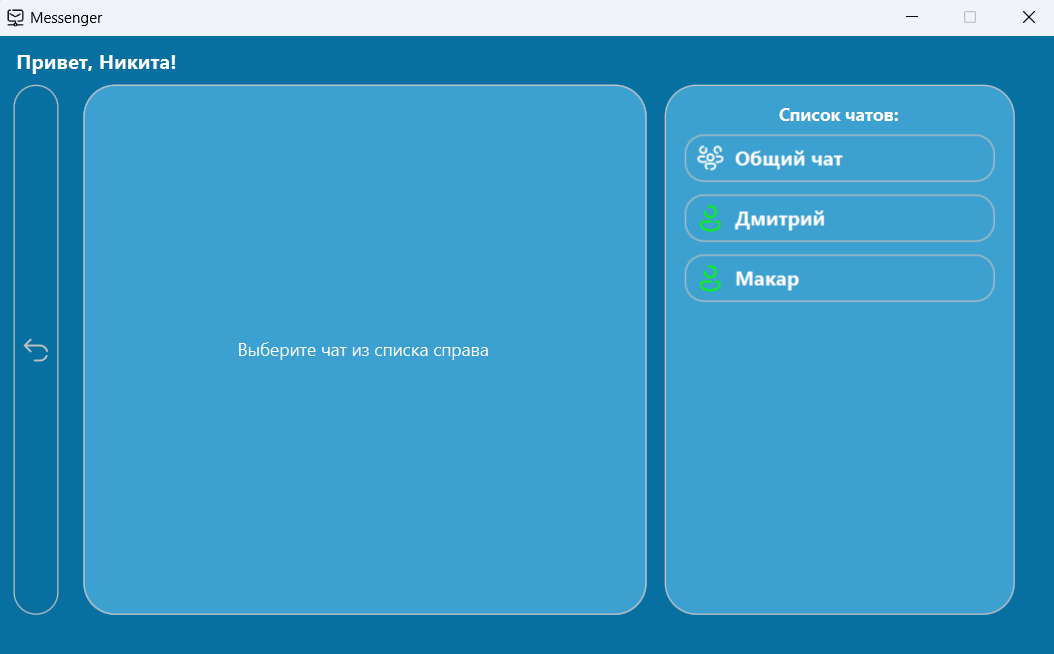


Рисунок 5 – Окно мессенджера

Выбираем собеседника и отправляем сообщение:

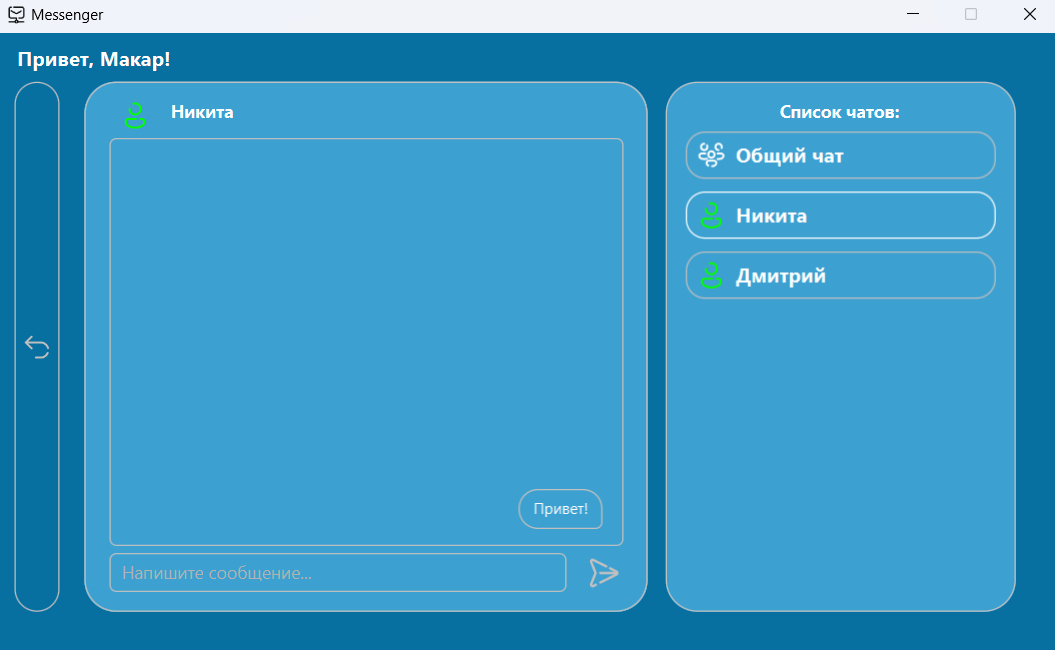


Рисунок 6 – Отправка сообщения

Тем временем пользователь «Никита» получает уведомление о новом сообщении:

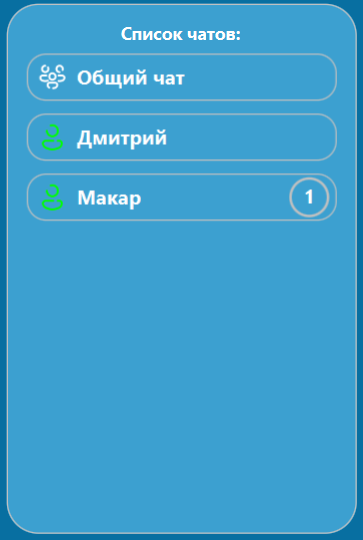
 

Рисунок 7-8 – Получение сообщения

Клиенты могут общаться через общий чат:

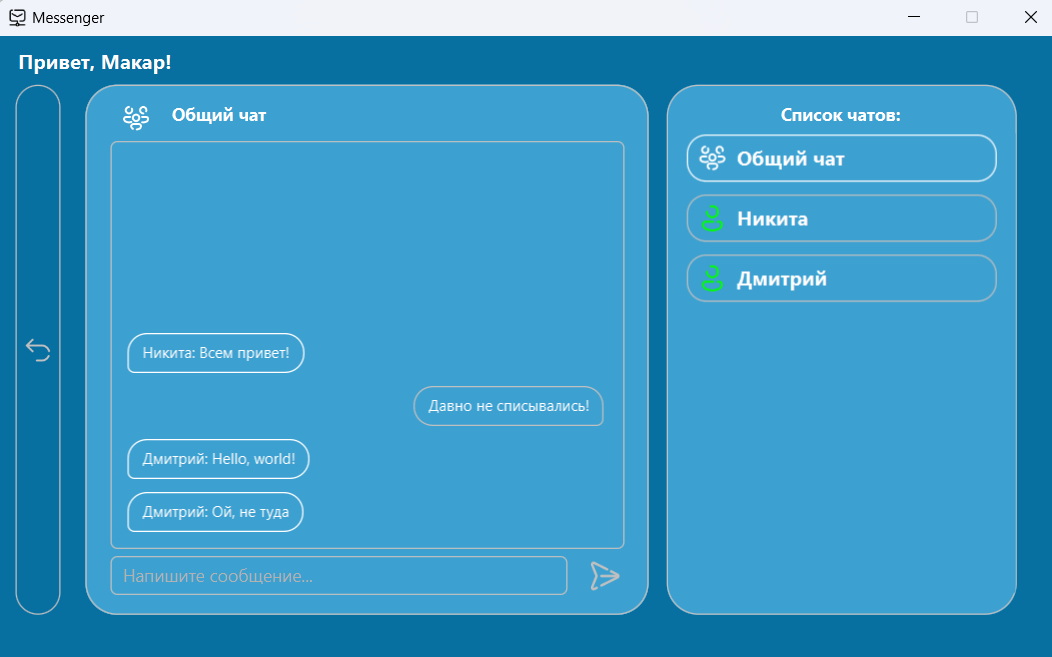


Рисунок 9 – Общий диалог

При выходе из сети, остальные клиенты получают уведомление о смене статуса собеседника:

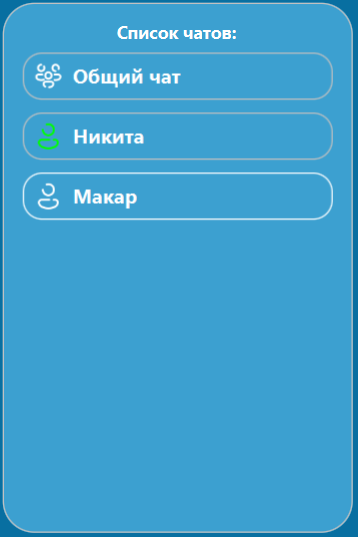


Рисунок 10 – Смена статуса

# Заключение

В ходе выполнения курсового проектирования были изучены принципы работы протоколов TCP/IP. Были получены навыки разработки приложения с пользовательским интерфейсом.

В итоге было разработано клиент-серверное приложение для передачи текстовых сообщений.

# Литература

1. "Head First Java, Изучаем Java", Кэти Сьерра, Берт Бэйтс
2. "Java. Руководство для начинающих", Герберт Шилдт
3. "Java для чайников", Барри Бёрд
4. "Java. Полное руководство", Герберт Шилдт

# Приложение А. Листинг программы

## Приложение А.1. Файл «Server.java».

package server;

import java.util.Scanner;

public class Server {

private static final ServerObject server = new ServerObject();

private static void StartServer()

{

if (server.isAlive() )

System.out.println("Сервер уже запущен");

else server.start();

}

public static void main(String[] args) {

StartServer();

Scanner console = new Scanner(System.in);

OUTER:

while (!server.serverSocket.isClosed()) {

String str = console.nextLine();

switch (str) {

case "stop", "стоп" -> {

server.StopServer();

break OUTER;

}

case "start", "старт" -> StartServer();

default -> server.HandleMessage(str);

}

}

}

}

## Приложение А.2. Файл «ServerObject.java».

package server;

import java.io.\*;

import java.net.\*;

import java.sql.\*;

import java.util.\*;

public class ServerObject extends Thread {

private final SortedMap<Integer, ClientObject> onlineList;

private final SortedMap<Integer, ClientObject> offlineList;

public ServerSocket serverSocket;

private Connection ConDB;

public ServerObject() {

onlineList = new TreeMap<>();

offlineList = new TreeMap<>();

ConnectToDB();

}

public void HandleMessage(String message) {

try {

if (message.contains("|")) {

String options, text;

{

String[] strings = message.split("\\|");

options = strings[0];

text = strings[1];

}

if (options.startsWith("@")) {

options = options.substring(1);

int TargetId = Integer.parseInt(options);

switch (TargetId) {

case 0 -> {

text = "&0|" + 0 + ':' + text;

BroadcastMessage(0, text);

} default -> {

text = "%" + text;

SendMessage(TargetId, text);

}

}

} else if ("/close".equals(options) && text.matches("[0-9]+")) {

int TargetId = Integer.parseInt(text);

CloseClient(TargetId);

}

} else

System.out.println("Необработанный запрос: " + message);

} catch (NumberFormatException e) {

System.err.println(e.getMessage());

}

}

public String GetOnlineClients(int SenderId){

if(onlineList.size() < 2)

return("/online|null");

List<String> ids = new ArrayList<>();

List<String> names = new ArrayList<>();

for (Map.Entry<Integer, ClientObject> client : onlineList.entrySet()) {

int id = client.getKey();

if (id != SenderId && id > 0) {

String name = client.getValue().GetName();

ids.add("" + id);

names.add(name);

}

}

String message = "/online|" + String.join(",", names) + ":" + String.join(",", ids);

return message;

}

private void ConnectToDB() {

try

{

Class.forName("org.postgresql.Driver");

String url = "jdbc:postgresql://localhost:5432/messenger";

Properties authorization = new Properties();

authorization.put("user", "postgres");

authorization.put("password", "123");

ConDB = DriverManager.getConnection(url, authorization);

System.out.println("Соединение с базой данных установлено");

} catch (ClassNotFoundException | SQLException e) {

System.err.println("Error accessing database!");

}

}

private Boolean IsLoginExist(String login) {

String query = "select is\_login\_exist(?)";

try{

PreparedStatement statement = ConDB.prepareStatement(query);

statement.setString(1, login);

ResultSet resultSet = statement.executeQuery();

if (resultSet.next()) {

return resultSet.getInt(1) == 1;

}

} catch (SQLException e)

{ System.err.println("Error accessing database!"); }

return false;

}

public Boolean RegClient(String message) {

String[] data = message.split(" "); // "name login password"

if (IsLoginExist(data[1]))

return false;

String query = "insert into users(name, login, password) values(?, ?, ?)";

try{

PreparedStatement statement = ConDB.prepareStatement(query);

statement.setString(1, data[0]);

statement.setString(2, data[1]);

statement.setString(3, data[2]);

statement.executeUpdate();

System.out.println("Пользователь " + data[0] + " успешно зарегистрирован.");

return true;

} catch (SQLException e)

{ System.err.println("Error accessing database!"); }

return false;

}

public Boolean LogClient(String message, ClientObject client) {

String[] data = message.split(" "); // "login password"

String query = "select \* from check\_login(?, ?)";

int id = 0;

try {

PreparedStatement statement = ConDB.prepareStatement(query);

statement.setString(1, data[0]);

statement.setString(2, data[1]);

ResultSet resultSet = statement.executeQuery();

if (resultSet.next()) {

id = resultSet.getInt(1); // Чтение значения из первого столбца результата

String name = resultSet.getString(2);

if (onlineList.containsKey(id)) {

SendMessage(client.GetId(), "#log|online");

return false;

} else if (id != 0) {

offlineList.remove(client.GetId());

ChangeIdClient(client, id);

client.SetName(name);

onlineList.put(id, client);

SendMessage(id, "#log|" + name);

if (onlineList.size() > 1)

BroadcastMessage(id, "#new|" + name + ":" + id);

} else {

SendMessage(client.GetId(), "#log|reject");

id = 0;

}

}

} catch (SQLException e)

{ System.err.println("Error accessing database!"); }

return id != 0;}

public void StopServer() {

try {

for (Map.Entry<Integer, ClientObject> client : onlineList.entrySet()) {

ClientObject clientObj = client.getValue();

clientObj.socket.close();

} for (Map.Entry<Integer, ClientObject> client : offlineList.entrySet()) {

ClientObject clientObj = client.getValue();

clientObj.socket.close();

}

if (serverSocket != null && !serverSocket.isClosed())

serverSocket.close();

this.interrupt();

} catch(IOException ex) {

System.err.println("Ошибка при закрытии сокета сервера");

}}

public void CloseClient(int id) {

ClientObject client;

if (onlineList.containsKey(id)) {

client = onlineList.get(id);

onlineList.remove(id);

} else {

client = offlineList.get(id);

offlineList.remove(id);

}

try {

if (!client.socket.isClosed()) {

client.socket.close();

BroadcastMessage(id, "#leave|" + id);

}

} catch (IOException e) {

System.err.println(e.getMessage());

}

}

public void LeaveClient(int id) {

BroadcastMessage(id, "#leave|" + id);

ClientObject client = onlineList.get(id);

onlineList.remove(id);

int port = client.socket.getPort();

ChangeIdClient(client, port);

offlineList.put(port, client);

}

public void ChangeIdClient (ClientObject client, int newId) {

int oldId = client.GetId();

client.SetId(newId);

System.out.println("Смена id клиента " + oldId + " -> " + newId);

}

public void SendMessage(int id, String message) {

try {

if (onlineList.containsKey(id)) onlineList.get(id).out.println(message);

else offlineList.get(id).out.println(message);

System.out.println("@" + id + ": " + message);

} catch (Exception e) {

System.err.println(e.getMessage());

}

}

public void BroadcastMessage(int SenderId, String message) {

try {

for (Map.Entry<Integer, ClientObject> client : onlineList.entrySet()) {

int id = client.getKey();

if (id != SenderId && id > 0) {

ClientObject clientObj = client.getValue();

clientObj.out.println(message);

System.out.println("@" + id + ": " + message);

}

}

} catch (Exception e) {

System.err.println(e.getMessage());

}

}

@Override

public void run()

{

serverSocket = null;

try {

serverSocket = new ServerSocket(8080);

System.out.println("Сервер запущен");

while (!this.isInterrupted()) {

Socket socket = serverSocket.accept();

try {

int port = socket.getPort();

ClientObject client = new ClientObject(socket, port, this);

client.start();

System.out.println("Новый клиент, порт: " + port );

offlineList.put(port, client);

}

catch (IOException e) {

System.out.println("Ошибка создания ClientObject внутри цикла в ServerObject");

socket.close();

}

}

} catch (SocketException e) {

System.out.println("Сокет сервера был закрыт");

} catch (IOException e) {

System.err.println(e.getMessage());

}

finally {

StopServer();

System.out.println("Сервер был остановлен");

}}}

## Приложение А.3. Файл «ClientObject.java».

package server;

import java.io.\*;

import java.net.\*;

public class ClientObject extends Thread {

private final ServerObject server;

public final Socket socket;

private final BufferedReader in;

public final PrintWriter out;

private int ClientId;

private String ClientName;

public ClientObject(Socket socket, int port, ServerObject server) throws IOException {

this.server = server;

this.socket = socket;

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())), true);

ClientId = port;

ClientName = "$NONAME$";

}

public void SetId(int id) { ClientId = id; }

public void SetName(String name){ ClientName = name; }

public int GetId() { return ClientId; }

public String GetName() { return ClientName; }

public void HandleMessage(String message) {

if (message.contains("|")) {

String options, text;

{

String[] strings = message.split("\\|");

options = strings[0];

text = strings[1];

}

switch (options.substring(0, 1)) {

case "@" -> {

options = options.substring(1);

//if (options.contains(",")) {} // Нереализованные беседы

int TargetId = Integer.parseInt(options);

switch (TargetId) {

case 0 -> {

text = "@0:" + ClientId + "|" + text;

server.BroadcastMessage(ClientId, text);

} default -> {

text = "&" + ClientId + "|" + text;

server.SendMessage(TargetId, text);

}

}

}

case "#" -> {

options = options.substring(1);

switch (options) {

case "log" -> server.LogClient(text, this);

case "reg" -> {

if(server.RegClient(text))

server.SendMessage(ClientId, "#reg|accept");

else server.SendMessage(ClientId, "#reg|reject");

} default -> System.err.println("Unprocessed request: " + text);

}

}

default -> System.err.println("Unprocessed request: " + message);

}

} else if ("/online".equals(message)) {

server.SendMessage(ClientId, server.GetOnlineClients(ClientId));

} else {

System.err.println("Unprocessed request: " + message);

}

}

@Override

public void run() {

try {

OUTER:

while (!socket.isClosed() && !server.serverSocket.isClosed()) {

String message = in.readLine();

System.out.println("Клиент " + ClientId + ": " + message);

if (message == null) {

break;

} else {

switch (message) {

case "/online" -> {

server.SendMessage(ClientId, server.GetOnlineClients(ClientId) );

} case "/close" -> {

break OUTER;

} case "#leave" -> {

server.LeaveClient(ClientId);

} default -> HandleMessage(message);

}

}

}

}

catch (IOException e) {

System.err.println("IOException from client " + ClientId);

}

finally {

System.out.println("Закрытие клиента " + ClientId + "...");

server.CloseClient(ClientId);

}

}

}

## Приложение А.4. Файл «Client.java».

package clientFX;

import clientFX.LoginForm.LoginFormController;

import clientFX.MessengerForm.MessengerFormController;

import java.io.IOException;

import javafx.application.\*;

import javafx.fxml.FXMLLoader;

import javafx.scene.\*;

import javafx.scene.image.Image;

import javafx.scene.layout.\*;

import javafx.stage.Stage;

import javafx.stage.WindowEvent;

public class Client extends Application {

private LoginFormController LoginForm;

private MessengerFormController MessengerForm;

private ClientObject client;

private Stage authStage;

private Stage MessengerStage;

public void ConnectToServer() {

client = new ClientObject(LoginForm, this);

client.ConnectToServer();

}

public void OpenMessenger(String name) {

Platform.runLater(() -> {

MessengerStage = new Stage();

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("MessengerForm/MessengerForm.fxml"));

AnchorPane root = loader.load();

MessengerForm = loader.getController();

client.SetForm(MessengerForm);

MessengerForm.SetClient(client, name);

MessengerStage.setTitle("Messenger");

String icon = this.getClass().getResource("icons/messenger.png").toExternalForm();

MessengerStage.getIcons().add(new Image(icon));

MessengerStage.setResizable(false);

MessengerStage.setScene(new Scene(root));

MessengerStage.setOnCloseRequest((WindowEvent event) -> {

System.out.println("Closing app...");

client.CloseClient();

});

authStage.hide();

client.SetAuth(true);

client.SendMessage("/online");

MessengerStage.show();

LoginForm.ClearLoginPanel();

} catch (IOException e) {

System.err.println(e.getMessage());

}

});

}

public void OpenAuth() {

Platform.runLater(() -> {

MessengerStage.close();

client.SendMessage("#leave");

client.SetAuth(false);

authStage.show();

});

}

public void ClosedServer() {

Platform.runLater(() -> {

MessengerForm.MessageBox("Потеряно соединение с сервером", "Пожалуйста, подключитесь заново и войдите в профиль", "error");

MessengerStage.close();

client.SetAuth(false);

authStage.show();

});

}

@Override

public void start(Stage stage) throws Exception {

FXMLLoader loader = new FXMLLoader(getClass().getResource("LoginForm/LoginForm.fxml"));

Parent root = loader.load();

authStage = stage;

LoginForm = loader.getController();

ConnectToServer();

LoginForm.SetClient(client, this);

stage.setTitle("Messenger");

String icon = this.getClass().getResource("icons/messenger.png").toExternalForm();

stage.getIcons().add(new Image(icon));

stage.setResizable(false);

stage.setScene(new Scene(root));

stage.setOnCloseRequest((WindowEvent event) -> {

System.out.println("Closing app...");

client.CloseClient();

});

stage.show();

}

public static void main(String[] args) {

launch(args);

}

}

## Приложение А.5. Файл «ClientObject.java».

package clientFX;

import clientFX.LoginForm.LoginFormController;

import clientFX.MessengerForm.MessengerFormController;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.io.PrintWriter;

import java.net.InetAddress;

import java.net.Socket;

import java.net.UnknownHostException;

import javafx.application.Platform;

public class ClientObject extends Thread {

private static Socket socket;

private static BufferedReader in;

private static PrintWriter out;

private final LoginFormController LoginForm;

private MessengerFormController MessengerForm;

public final Client mainThread;

public ClientObject(LoginFormController fxml, Client client) {

this.LoginForm = fxml;

this.mainThread = client;

}

public void SetForm(MessengerFormController form) {

MessengerForm = form;

}

public void SetAuth(boolean IsLogined) {

if (!IsLogined) MessengerForm = null;

}

public void SendMessage(String message) {

Platform.runLater(() -> {

if (!socket.isClosed())

out.println(message);

});

}

public void CloseClient() {

try{

if (socket != null && !socket.isClosed()) {

out.println("/close");

socket.close();

MessengerForm = null;

this.interrupt();

}

} catch (IOException ex) {

System.out.println(ex.getMessage());

}

}

public void ConnectToServer() {

try{

// Передаем null в getByName()

InetAddress addr = InetAddress.getByName("localhost");

socket = new Socket(addr, 8080);

System.out.println("socket = " + socket);

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

// Вывод автоматически Output выталкивается PrintWriter'ом.

out = new PrintWriter(new BufferedWriter(

new OutputStreamWriter(socket.getOutputStream())), true);

LoginForm.SetConn(true);

LoginForm.SetClient(this);

start();

} catch (UnknownHostException e){

System.out.println(e.getMessage());

} catch (IOException e){

System.out.println(e.getMessage());

}

}

private void HandleMessage(String message) {

try{

if (message.contains("|")) {

String options, text;

{

String[] strings = message.split("\\|");

options = strings[0];

text = strings[1];

}

switch (options.substring(0, 1)) {

case "&" -> {

options = options.substring(1);

int TargetId = Integer.parseInt(options);

MessengerForm.NewMessage(text, TargetId, true);

}

case "@" -> { // P: "@to:from|message"

options = options.substring(1);

int TargetId, FromId;

{

String[] data = options.split(":");

TargetId = Integer.parseInt(data[0]);

FromId = Integer.parseInt(data[1]);

}

text = MessengerForm.GetName(FromId) + ": " + text;

MessengerForm.NewMessage(text, TargetId, true);

}

case "#" -> {

options = options.substring(1);

if (null != options) switch (options) {

case "log" -> {

switch (text) {

case "online" -> LoginForm.MessageBox("Получен ответ от сервера", "Пользователь с таким логином уже в сети", "warning");

case "reject" -> LoginForm.MessageBox("Получен ответ от сервера", "Ошибка входа", "error");

default -> mainThread.OpenMessenger(text);

}

}

case "reg" -> {

if ("accept".equals(text)) {

LoginForm.MessageBox("Получен ответ от сервера", "Успешная регистрация!", "confirm");

LoginForm.SuccessfulReg();

}

else if ("reject".equals(text))

LoginForm.MessageBox("Получен ответ от сервера", "Ошибка регистрации", "error");

}

case "new" -> {

String[] data = text.split(":");

MessengerForm.NewUser(data[0], Integer.parseInt(data[1]), true);

}

case "leave" -> MessengerForm.LeaveUser(Integer.parseInt(text));

default -> {

}

}

}

case "/" -> {

options = options.substring(1);

if ("online".equals(options)) {

if (!"null".equals(text)) {

String[] data = text.split(":");

String[] names = data[0].split(",");

String[] ids = data[1].split(",");

for (int i = 0; i < names.length; i++) {

MessengerForm.NewUser(names[i],Integer.parseInt(ids[i]),true);

}

}

}

}

}

} else if (message.startsWith("%")) {

LoginForm.MessageBox("Сообщение от сервера", message.substring(1), "info");

} else

System.err.println("Unprocessed request: " + message);

} catch (Exception e) {

System.err.println(e.getMessage());

}

}

@Override

public void run()

{

try {

OUTER:

while (!socket.isClosed()) {

String msg = in.readLine();

if (null == msg) {

break;

} else {

switch (msg) {

case "/close" -> {

break OUTER;

}

default -> HandleMessage(msg);

}

}

System.out.println("Server: " + msg);

}

} catch(IOException ex) {

System.out.println(ex.getMessage());

} finally {

if (LoginForm.statusApp) LoginForm.SetConn(false);

if (MessengerForm != null ) mainThread.ClosedServer();

try{

if (!socket.isClosed())

socket.close();

} catch (IOException ex) {

System.out.println(ex.getMessage());

}

}

}

}

## Приложение А.6. Файл «LoginFormController.java».

package clientFX.LoginForm;

import clientFX.Client;

import clientFX.ClientObject;

import java.net.URL;

import java.util.ResourceBundle;

import javafx.application.Platform;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.input.KeyCode;

import javafx.scene.input.KeyEvent;

import javafx.scene.layout.AnchorPane;

public class LoginFormController implements Initializable {

public Boolean statusApp;

private ClientObject client;

private Client mainThread;

@FXML

private Button LoginButton; @FXML

private TextField LoginFieldLog; @FXML

private TextField PasswordFieldLog; @FXML

private Button LoginPanelButton; @FXML

private Button RegPanelButton; @FXML

private TextField NameFieldReg; @FXML

private TextField LoginFieldReg; @FXML

private PasswordField PasswordFieldReg; @FXML

private PasswordField ConfirmPasswordFieldReg; @FXML

private Button RegButton; @FXML

private AnchorPane LoginPanel; @FXML

private AnchorPane RegPanel; @FXML

private Label ConnStatusLabel; @FXML

private Button ReconnButton;

public void SetClient(ClientObject client, Client mainTh) {

this.client = client;

mainThread = mainTh;

}

public void SetClient(ClientObject client) {

this.client = client;

}

public void SetConn(Boolean status) {

Platform.runLater(() -> {

if (status) {

statusApp = true;

ConnStatusLabel.setVisible(false);

ReconnButton.setVisible(false);

} else {

statusApp = false;

ConnStatusLabel.setVisible(true);

ReconnButton.setVisible(true);

}

});

}

public void ClearLoginPanel() {

LoginFieldLog.setText("");

PasswordFieldLog.setText("");

}

public void SuccessfulReg() {

Platform.runLater(() -> {

LoginFieldLog.setText(LoginFieldReg.getText());

LoginPanelButton.fire();

});

}

public void MessageBox(String title, String message, String type) {

Platform.runLater(() -> {

Alert alert = new Alert(AlertType.INFORMATION);

switch (type){

case "info" -> alert.setAlertType(AlertType.INFORMATION);

case "confirm" -> alert.setAlertType(AlertType.CONFIRMATION);

case "warning" -> alert.setAlertType(AlertType.WARNING);

case "error" -> alert.setAlertType(AlertType.ERROR);

case "none" -> alert.setAlertType(AlertType.NONE);

}

alert.setTitle(title);

alert.setHeaderText(message);

alert.showAndWait();

});

}

@Override

public void initialize(URL url, ResourceBundle rb) {

//Tooltip tooltip = new Tooltip("Это кнопка для выполнения какого-то действия");

}

@FXML

private void HandleTextBoxLog(KeyEvent event) {

if (event.getCode() == KeyCode.ENTER)

LoginButton.fire();

}

@FXML

private void HandleLoginButton(ActionEvent event) {

String login = LoginFieldLog.getText();

String pass = PasswordFieldLog.getText();

if (login.length() > 2 && pass.length() > 2) {

String message = "#log|" + login + ' ' + pass;

client.SendMessage(message);

} else {

MessageBox("Ошибка ввода","Минимальная длина каждого поля - 3 символа","warning");

}

}

@FXML

private void HandleTextBoxReg(KeyEvent event) {

if (event.getCode() == KeyCode.ENTER)

RegButton.fire();

}

@FXML

private void HandleRegButton(ActionEvent event) {

String login = LoginFieldReg.getText();

String name = NameFieldReg.getText();

String pass = PasswordFieldReg.getText();

String confPass = ConfirmPasswordFieldReg.getText();

if (login.length() > 2 && name.length() >= 2 && pass.length() > 2 && confPass.length() > 2 ) {

if (!login.matches(".\*\\W+.\*") && name.matches("[a-zA-Zа-яА-Я]+")) {

if (pass.equals(confPass)) {

String message = "#reg|" + name + ' ' + login + ' ' + pass;

client.SendMessage(message);

} else {

MessageBox("Ошибка ввода","Пароль не совпадает","warning");

}

} else {

MessageBox("Ошибка ввода","Имя и Логин должны содержать только буквы и цифры","warning");

}

} else {

MessageBox("Ошибка ввода","Минимальная длина каждого поля - 3 символа","warning");

}

}

@FXML

private void HandleLogPanelButton(ActionEvent event) {

LoginPanel.toFront();

NameFieldReg.setText("");

LoginFieldReg.setText("");

PasswordFieldReg.setText("");

ConfirmPasswordFieldReg.setText("");

}

@FXML

private void HandleRegPanelButton(ActionEvent event) {

RegPanel.toFront();

ClearLoginPanel();

}

@FXML

private void HandleReconnButton(ActionEvent event) {

mainThread.ConnectToServer();

}

@FXML

private void HandleFastLogin(ActionEvent event) {

String message = "#log|robinx51 123";

client.SendMessage(message);

} @FXML

private void HandleMeffsLogin(ActionEvent event) {

String message = "#log|Meffs 123";

client.SendMessage(message);

}

}

## Приложение А.7. Файл «MessengerFormController.java».

package clientFX.MessengerForm;

import clientFX.ClientObject;

import java.net.URL;

import java.util.\*;

import javafx.application.Platform;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.geometry.Pos;

import javafx.scene.Node;

import javafx.scene.control.\*;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.input.\*;

import javafx.scene.layout.\*;

public class MessengerFormController implements Initializable {

public Boolean statusApp;

private ClientObject client;

private int activeFriendId = -1;

private final SortedMap<Integer, UserProfile> FriendMap = new TreeMap<>();

@FXML

private Button BackButton; @FXML

private Label HelloLabel;

@FXML

private AnchorPane noChatPanel; @FXML

private AnchorPane ChatPanel; @FXML

private Label NameFriend; @FXML

private Label FriendImg; @FXML

private ScrollPane MessagesScrollPane; @FXML

private VBox MessagesList; @FXML

private TextField MessageField; @FXML

private Button SendButton;

@FXML

private VBox FriendList;

private final class UserProfile extends Pane {

private final String name;

private final int id;

private final boolean isOnline;

private int newMessagesCount;

private ObservableList<Node> messages;

private Button button;

private Label status;

private final Label nameLabel;

private Label newMessagesLabel;

private final class Message extends Pane {

private final Label messageLabel;

public Message(String message, boolean isDerived) {

this.setId("MessagePane");

HBox msgBox = new HBox();

messageLabel = new Label(message);

messageLabel.setWrapText(true);

msgBox.getChildren().add(messageLabel);

if (isDerived) {

messageLabel.setId("DerivedMessage");

} else {

messageLabel.setId("SendedMessage");

msgBox.setAlignment(Pos.CENTER\_RIGHT);

msgBox.setPrefWidth(381);

}

getChildren().add(msgBox);

}

}

public UserProfile(String name, int id, boolean isOnline) {

this.name = name;

this.id = id;

this.isOnline = isOnline;

newMessagesCount = 0;

messages = FXCollections.observableArrayList();

button = new Button("");

status = new Label("");

nameLabel = new Label(name);

newMessagesLabel = new Label("" + newMessagesCount);

this.setId("UserProfile");

button.setId("UserProfileButton");

if (id == 0) status.setId("UserProfileGroup");

else SetStatus(this.isOnline);

nameLabel.setId("UserProfileName");

newMessagesLabel.setId("UserProfileCount");

newMessagesLabel.setVisible(false);

newMessagesLabel.setLayoutX(210);

newMessagesLabel.setLayoutY(3);

status.setLayoutY(3);

getChildren().addAll(status, nameLabel, newMessagesLabel, button);

button.setOnAction((ActionEvent event) -> {

SetActive(name, id, status, messages);

ChatPanel.toFront();

newMessagesLabel.setVisible(false);

newMessagesCount = 0;

button.setId("UserProfileActiveButton");

//button.setDisable(true);

});

}

public void SetInactive() {

button.setId("UserProfileButton");

button.setDisable(false);

}

public void SetStatus(boolean IsOnline) {

if (IsOnline) status.setId("UserProfileOnline");

else status.setId("UserProfileOffline");

FriendImg.setId(status.getId());

}

public boolean GetStatus() { return isOnline; }

public int GetId() { return id; }

public String GetName() { return name; }

public Node NewMessage(String strMessage, boolean isDerived) {

Message msg = new Message(strMessage, isDerived);

if (isDerived && activeFriendId != id) {

newMessagesLabel.setText("" + ++newMessagesCount);

newMessagesLabel.setVisible(true);

}

messages.add(msg);

return msg;

}

}

public String GetName(int id) {

return FriendMap.get(id).GetName();

}

public void NewUser(String name, int id, boolean IsOnline) {

Platform.runLater(() -> {

if (!FriendMap.containsKey(id)) {

UserProfile Alex = new UserProfile(name, id, IsOnline);

FriendMap.put(id, Alex);

FriendList.getChildren().add(Alex);

} else {

FriendMap.get(id).SetStatus(true);

}

});

}

public void LeaveUser(int id) {

Platform.runLater(() -> { FriendMap.get(id).SetStatus(false); });

}

public void NewMessage(String message, int id, boolean isDerived) {

Platform.runLater(() -> {

Node Message = FriendMap.get(id).NewMessage(message, isDerived);

if (activeFriendId == id) MessagesList.getChildren().add(Message);

});

}

private void SetActive(String name, int id, Label Status, ObservableList<Node> messages) {

if (activeFriendId != -1) SetInactive(activeFriendId);

activeFriendId = id;

NameFriend.setText(name);

FriendImg.setId(Status.getId());

MessagesList.getChildren().setAll(messages);

}

private void SetInactive(int id) {

FriendMap.get(id).SetInactive();

}

public void SetClient(ClientObject client, String name) {

this.client = client;

HelloLabel.setText("Привет, " + name + '!');

}

public void MessageBox(String title, String message, String type) {

Platform.runLater(() -> {

Alert alert = new Alert(AlertType.INFORMATION);

switch (type){

case "info" -> alert.setAlertType(AlertType.INFORMATION);

case "confirm" -> alert.setAlertType(AlertType.CONFIRMATION);

case "warning" -> alert.setAlertType(AlertType.WARNING);

case "error" -> alert.setAlertType(AlertType.ERROR);

case "none" -> alert.setAlertType(AlertType.NONE);

}

alert.setTitle(title);

alert.setHeaderText(message);

alert.showAndWait();

});

}

@Override

public void initialize(URL url, ResourceBundle rb) {

Tooltip tooltip = new Tooltip("Отправить сообщение");

SendButton.setTooltip(tooltip);

tooltip = new Tooltip("Вернуться на страницу авторизации");

BackButton.setTooltip(tooltip);

NewUser("Общий чат", 0, true);

MessagesList.heightProperty().addListener((observable, oldValue, newValue) -> {

MessagesScrollPane.setVvalue(1.0);

});

}

@FXML

private void HandleBackButton(ActionEvent event) {

client.mainThread.OpenAuth();

} @FXML

private void HandleReconnButton(ActionEvent event) {

client.ConnectToServer();

} @FXML

private void HandleMessageField(KeyEvent event) {

if (MessageField.getText().length() >= 1)

SendButton.setId("sendButtonActive");

else SendButton.setId("sendButton");

if (event.getCode() == KeyCode.ENTER)

SendButton.fire();

} @FXML

private void HandleMessageTypedField(KeyEvent event) {

if (MessageField.getText().length() >= 1)

SendButton.setId("sendButtonActive");

else SendButton.setId("sendButton");

} @FXML

private void HandleMessagePanel(KeyEvent event) {

if (event.getCode() == KeyCode.ESCAPE) {

noChatPanel.toFront();

MessagesList.getChildren().clear();

SetInactive(activeFriendId);

activeFriendId = -1;

}

} @FXML

private void HandleSendButton(ActionEvent event) {

if (MessageField.getText().length() >= 1) {

String message = MessageField.getText();

try {

client.SendMessage("@" + activeFriendId + "|" + message);

NewMessage(message, activeFriendId, false);

MessageField.setText("");

SendButton.setId("sendButton");

} catch (Exception e) {

System.err.println(e.getMessage());

}

}

}

}

# Приложение Б. UML-диаграммы

## Приложение Б.1. UML – диаграмма вариантов использования.

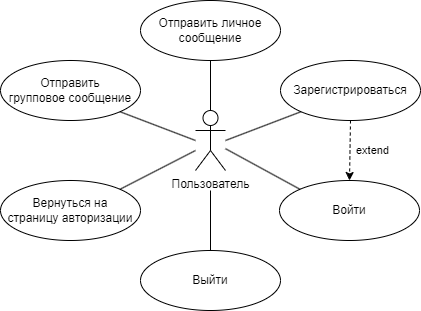


Рисунок 11 - UML- диаграмма вариантов использования

## Приложение Б.2. UML – диаграмма классов.

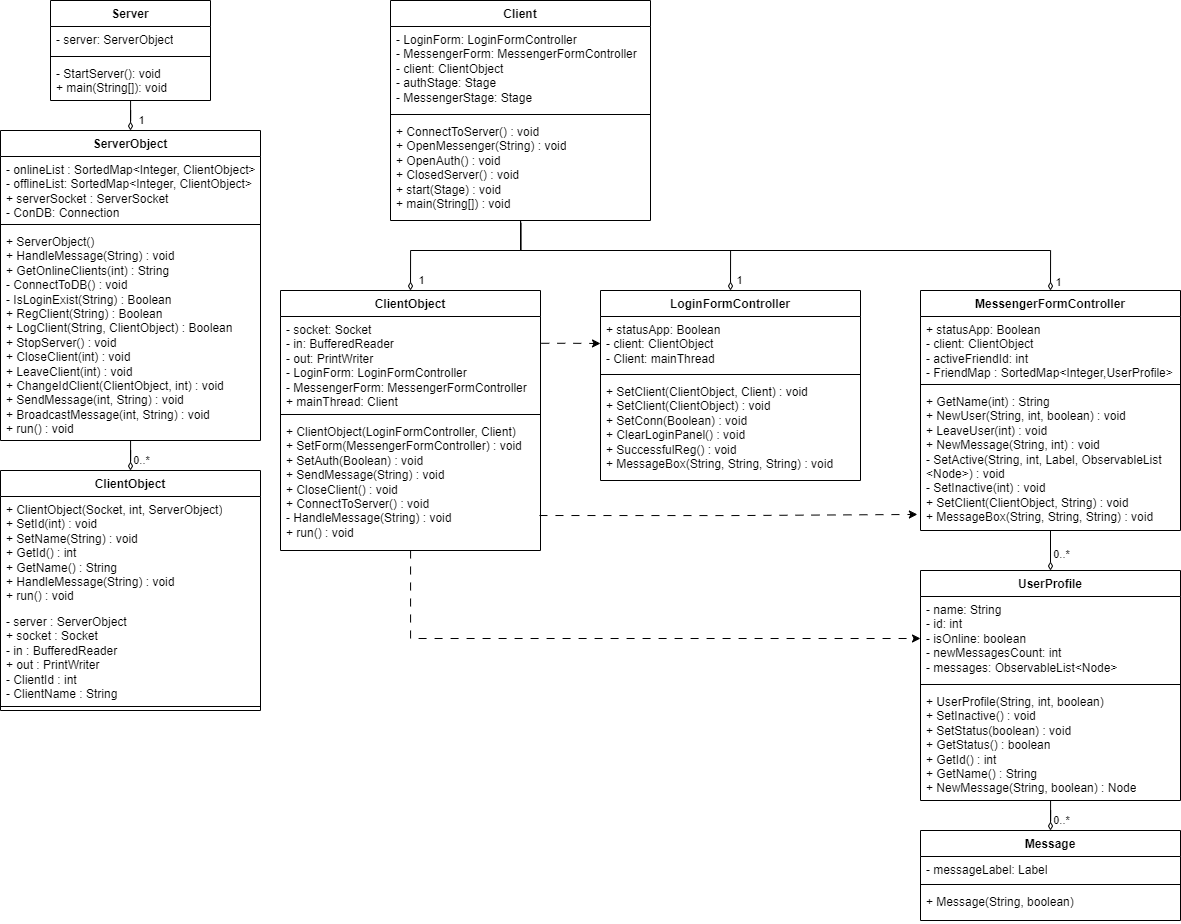


Рисунок 12 - UML- диаграмма классов сервера

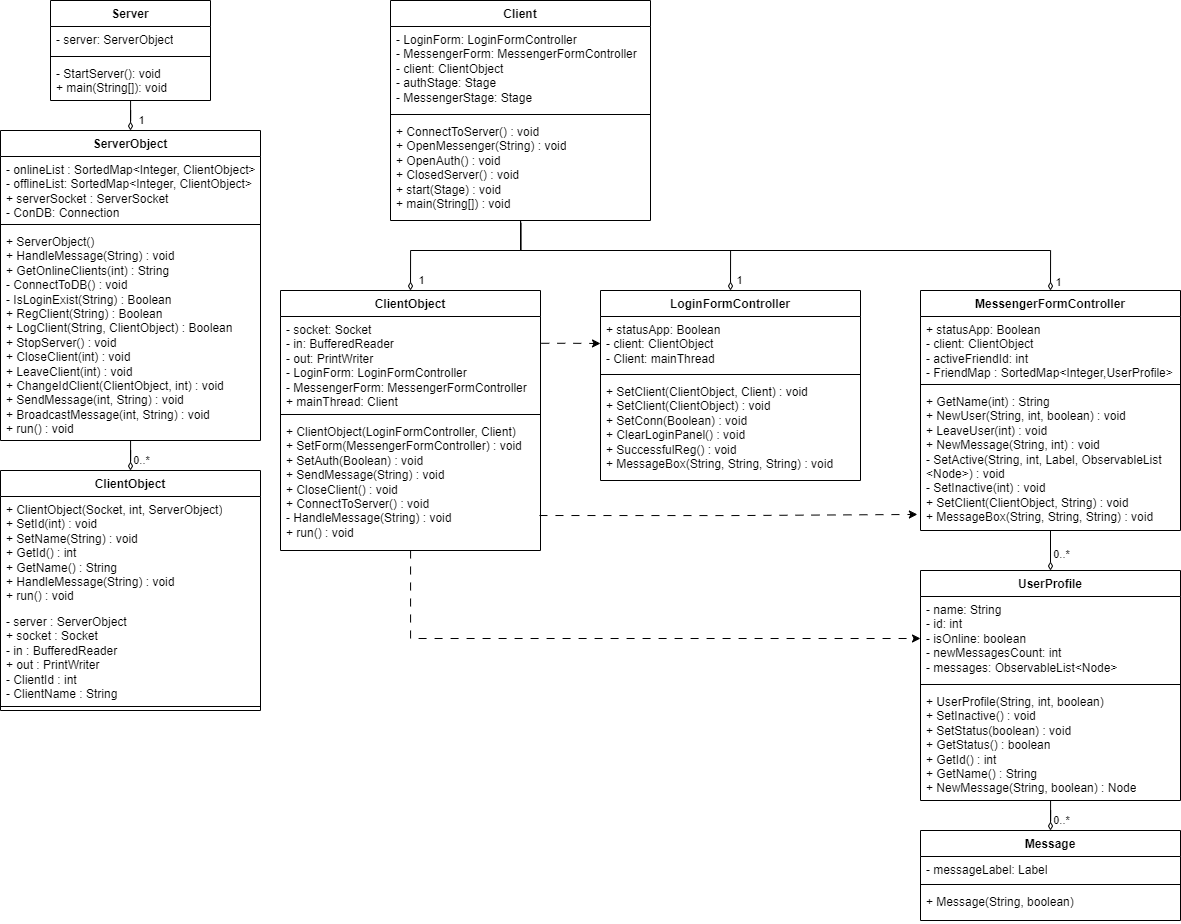


Рисунок 13 - UML- диаграмма классов клиента

## Приложение Б.3. UML – диаграмма деятельности.

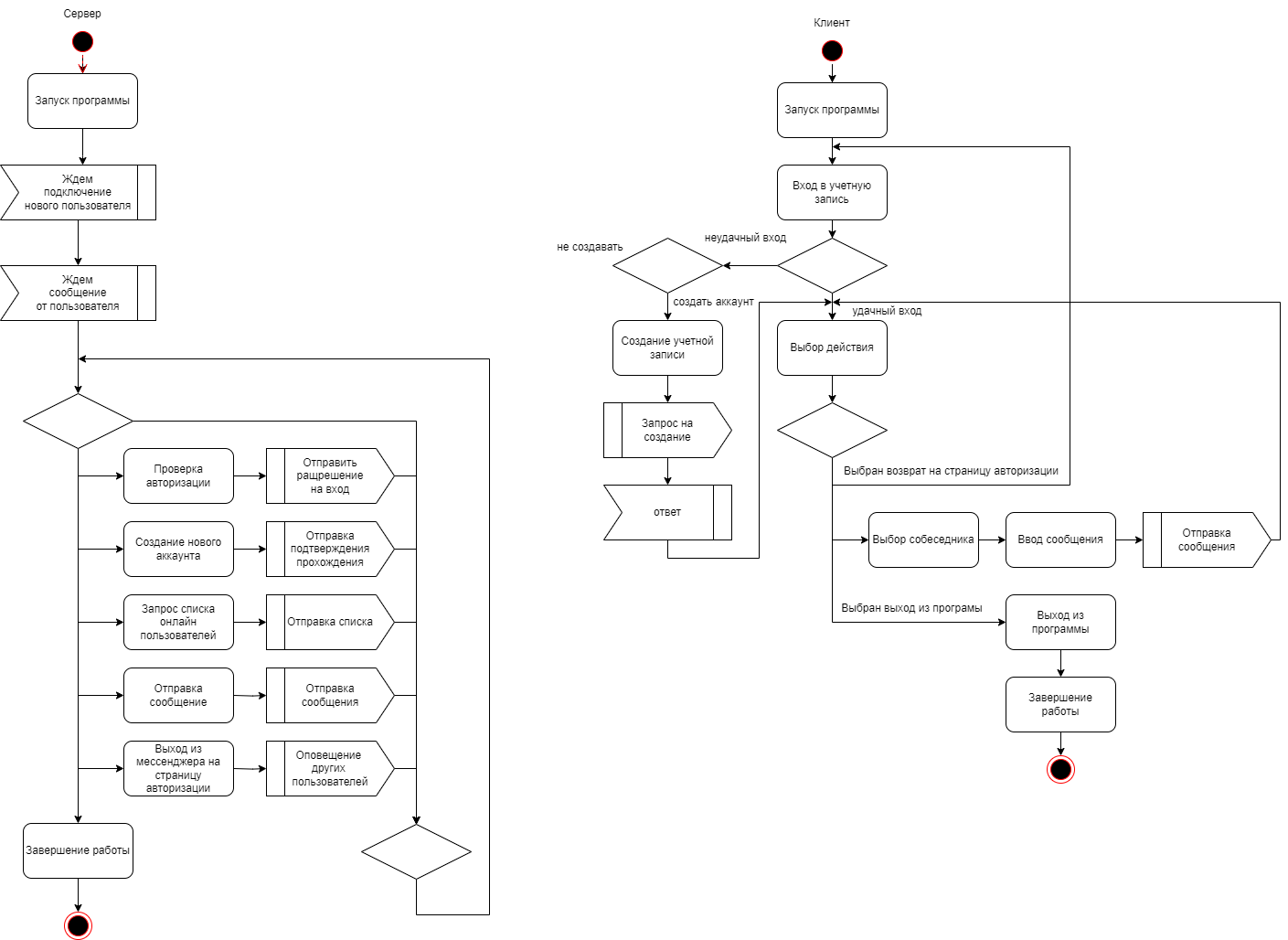


Рисунок 14 - UML- диаграмма деятельности

## Приложение Б.4. UML – диаграмма развертывания.

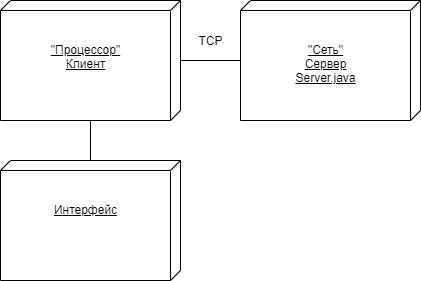


Рисунок 15 - UML- диаграмма развертывания

## Приложение Б.5. UML – диаграмма последовательности.

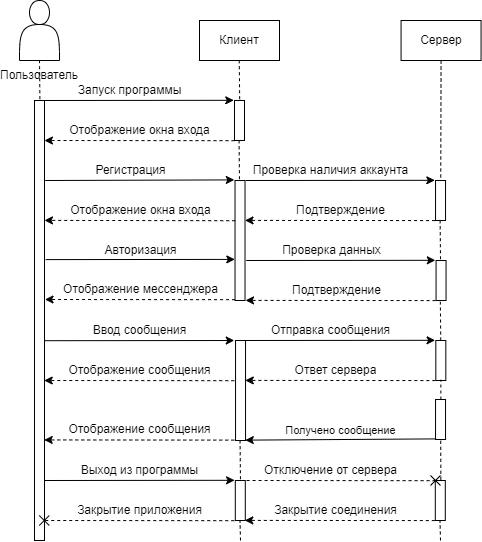


Рисунок 16 - UML- диаграмма последовательности