## 

## СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#_Toc530175086)

[ВВЕДЕНИЕ 2](#_Toc530175087)

[1.1. Система мгновенного обмена сообщениями 4](#_Toc530175088)

[1.2. Анализ существующих аналогов 5](#_Toc530175089)

[1.3. Постановка задачи 7](#_Toc530175090)

[2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 8](#_Toc530175091)

[2.1. Обмен данными по протоколу связи WebSocket 8](#_Toc530175092)

[2.2. Описание функциональности ПС 10](#_Toc530175093)

[2.3. Спецификация функциональных требований 11](#_Toc530175094)

[3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 12](#_Toc530175095)

[3.1. Проектирование структур данных 12](#_Toc530175096)

[3.2. Структура пересылаемого сообщения 13](#_Toc530175097)

[3.3 . Разработка алгоритма обработки регистрационного сообщения 14](#_Toc530175098)

[3.3. Разработка алгоритма обработки информационного сообщения 15](#_Toc530175099)

[3.4. Разработка алгоритма выхода пользователей из комнаты 16](#_Toc530175100)

[4. СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА 17](#_Toc530175101)

[4.1. Разработка интерфейса программы 17](#_Toc530175102)

[4.1.1 Разработка интерфейса настольного приложения 17](#_Toc530175103)

[4.1.2Разработка интерфейса настольного приложения 19](#_Toc530175104)

[4.1.2. Разработка интерфейса веб-приложения 19](#_Toc530175105)

[5.ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 21](#_Toc530175106)

[5.1. Тестирование регистрации и подключения пользователя 21](#_Toc530175107)

[5.4. Вывод из прохождения тестирования 22](#_Toc530175108)

[СПИСОК ЛИТЕРАТУРЫ 23](#_Toc530175109)

# ВВЕДЕНИЕ

Обмен мгновенными сообщениями был развивающимся направлением в течение многих лет, так что довольно трудно точно определить момент появления первых мессенджеров. Но зато достаточно легко отследить, когда сервисы обмена мгновенными сообщениями сделали качественный скачок и стали действительно популярными. Первым мессенджером, из известных на сегодняшний день, стал ICQ (название происходит от английских слов I SEEK YOU – я ищу тебя), появившийся в 1996 году. Все началось с того, что четверо школьников из Израиля создали компанию Mirabilis и начали работать над программой для общения в Интернете и локальных сетях. Создав программу, они разослали ее бесплатно друзьям и знакомым. Те, в свою очередь, приводили в «аську» своих друзей и знакомых. Количество пользователей росло в геометрической прогрессии. А через некоторое время талантливая четверка выпустила корпоративную версию ICQ. «Аська» стала пионером рынка мессенджеров

С тех пор чаты стали неотъемлемой частью человеческой жизни. Данная курсовая работа посвящена разработке программного средства для общения в сети на примере Бизнес-мессенджер для общения с клиентами

В ходе выполнения данной работы будут рассмотрены и применены следующие подходы:

1. Работа с Web-socket
2. Разработка web-версии приложения
3. Работа с Java GlassFish сервером
4. Использование Maven для сборки проектов

В пояснительной записке отображены этапы написания курсовой работы

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. **АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

## Система мгновенного обмена сообщениями

Система мгновенного обмена сообщениями, Система обмена мгновенными сообщениями— службы мгновенных сообщений программы онлайн-консультанты и программы-клиенты для обмена сообщениями в реальном времени через Интернет. Могут передаваться текстовые сообщения, звуковые сигналы, изображения, видео, а также производиться такие действия, как совместное рисование или игры. Многие из таких программ-клиентов могут применяться для организации групповых текстовых чатов или видеоконференций.

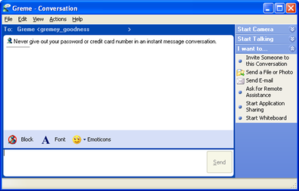


Рисунок 1.1 - Разговор в Windows Messenger 4.7.2009 с включенной боковой панелью и панелью инструментов

Для подобного рода коммуникации необходима клиентская программа, так называемый мессенджер. Отличие от электронной почты здесь в том, что обмен сообщениями идёт в реальном времени. Большинство IM-клиентов позволяет видеть, подключены ли в данный момент абоненты, занесённые в список контактов. В ранних версиях программ всё, что печатал пользователь, тут же передавалось. Если он делал ошибку и исправлял её, это тоже было видно. В таком режиме общение напоминало телефонный разговор. В современных программах сообщения появляются на мониторе собеседника уже после окончания редактирования и отправки сообщения.

Как правило, мессенджеры не работают самостоятельно, а подключаются к центральному компьютеру сети обмена сообщениями, называемому сервером. Поэтому мессенджеры и называют клиентами (клиентскими программами). Термин является понятием из клиент-серверных технологий.

## Анализ существующих аналогов

На данный момент существует большое количество систем мгновенного обмена сообщениями. Многим мессенджерам присуща “региональность” использования. Например, в Беларуси популярен “Viber” и “Telegram”, в Западной Европе “WhatsApp” и “Facebool Messenger” в Китае “WeChat” и “Mobile QQ”. Рассмотрим некоторые из них

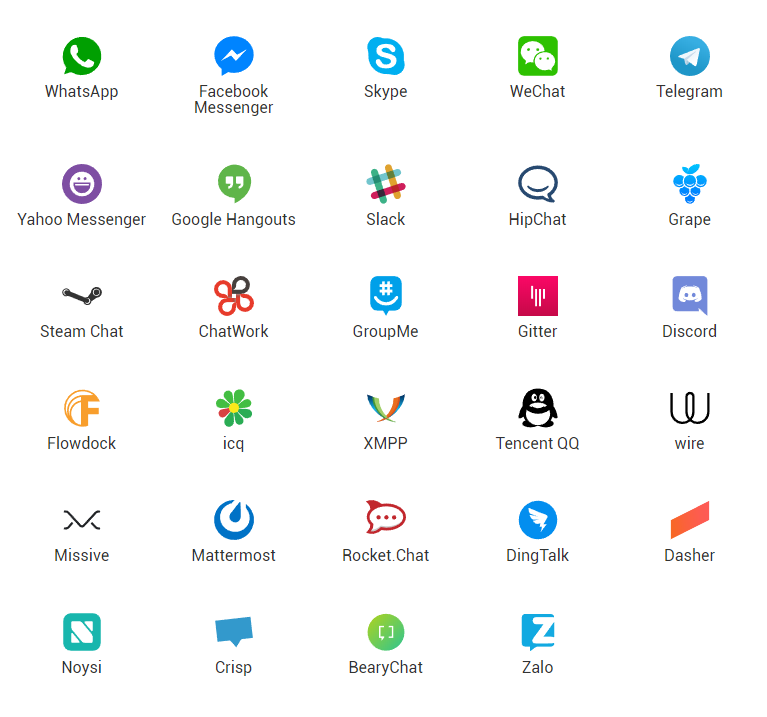


Рисунок 1.2 – Наиболее популярные мессенджеры

**Viber**

Viber существует для всех платформ. Только веб-версии нет. Все интерфейсы русифицированы. Приложения максимально простые и понятные, при этом обладают одинаковой функциональностью как мобильное, так и “десктопное”, вплоть до совершения звонков. Контакты подтягиваются автоматически, друзьям, уже использущим Viber, приходит сообщение, что вы установили приложение. Сообщения на серверах не хранятся.

**Telegram**

Telegram – самый молодой мессенджер среди представленных, быстро набирает популярность. Установить можно практически всюду, кроме BlackBerry, но благодаря популярности, сервис обрастает десятками неофициальных приложений. Всюду имеет схожий интерфейс, очень похож на WhatsApp. Контакты подтягивает из телефонной книги, отправляет сообщение пользователям о “прибавлении в семействе”. Все сообщения отображаются абсолютно на всех устройствах, ибо хранятся на серверах.

Рисунок 1.3 – Рейтинг мессенджеров на основании оценок экспертов

Так же не маловажную роль играет конфиденциальность информации. Потому хранятся ли сообщения на устройстве, или на сервере мессенджера, доступность личных данных пользователей спецслужбам стран порой имеют определяющую роль при выборе коммуникационных сервисов.

В моём случае речь идёт о разработке чата для решения такой бизнес-задачи как общение клиента и агента, к примеру, на сайте интернет-магазина. В данном направлении конкурентность несколько ниже, и назвать каких-либо именитых издателей такого ПО сходу сложно. Поэтому было принято решение о разработке своего собственного коммуникационного сервиса по моментальному обмену сообщениями.

## Постановка задачи

Так как программное обеспечение является встраиваемым, необходимо создать легко расширяемую и адаптируемую базу. Программное средство должно обладать средствами логгирования. Для демонстрации работы сервиса необходимо создать десктопную версию клиента для приложения агента и клиента, а так же, при помощи веб языков программирования разработать браузерные версии.

Требования к серверу:

* Автоматизировать подключение пользователей к агенту/клиенту
* Обеспечить поддержку многопользовательского подключения.
* Использовать выдачу идентификаторов пользователям, для избежания конфликта имён.
* Предусмотреть возможность подключения агентского приложения к множественным клиентским приложениям (возможность общаться сразу с несколькими клиентами)
* Предусмотреть получение доступа к данным с использованием архитектурого стиля RESTful API.

Требования к приложениям:

* Реализовать возможность регистрации, отправки сообщения, выхода из диалога и выхода из клиента приложения.
* Осуществлять общение с сервером по протоколу связи WebSocket

# МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## Обмен данными по протоколу связи WebSocket

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

В настоящее время в W3C осуществляется стандартизация API Web Sockets. Черновой вариант стандарта этого протокола утверждён IETF.

WebSocket разработан для воплощения в web-браузерах и web-серверах, но он может быть использован для любого клиентского или серверного приложения. Протокол WebSocket — это независимый протокол, основанный на протоколе TCP. Он делает возможным более тесное взаимодействие между браузером и web-сайтом, способствуя распространению интерактивного содержимого и созданию приложений реального времени.[1]

Протокол 76

2 июня 2010 года в протокол WebSocket были внесены поправки, изменившие процедуру установления соединения WebSocket без сохранения обратной совместимости. В 76-й редакции черновика протокола WebSocket добавлена защита от поддельных запросов. Клиент, поддерживающий новую схему, присылает следующий запрос:



Рисунок 2.1 – запрос, отправляемый клиентом для установления соединения по технологии WebSocket

В запрос добавлены новые заголовки «Sec-WebSocket-Key1» и «Sec-WebSocket-Key2» и 8-байтовое тело запроса. Все они генерируются клиентом случайным образом.

Ответ сервера, подтверждающий переход на WebSocket:

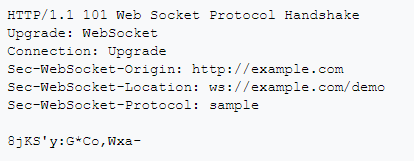


Рисунок 2.2 – ответ от Сервера при удачном подключении клиента.

Ответ содержит новые названия заголовков («Sec-WebSocket-Origin», «Sec-WebSocket-Location», «Sec-WebSocket-Protocol» вместо «WebSocket-Origin», «WebSocket-Location», «WebSocket-Protocol») и 16-байтное тело ответа, вычисляемое следующим образом:

* из строки со значением заголовка запроса Sec-WebSocket-Key1 исключить все символы, не являющиеся цифрами (не попадающие в диапазон '0'..'9');
* полученную строку превратить в 64-битное целое число (для примера выше получим 1299853100);
* полученное число разделить целочисленным делением на количество пробелов в исходной строке со значением заголовка;
* полученное число представить в виде 4-байтового 32-битного числа в формате big endian: старший байт хранится по нулевому смещению;
* проделать то же самое с заголовком Sec-WebSocket-Key2;
* интерпретируя числа из пунктов 4) и 5) как 4-байтовые строки, конкатенировать их (сложить в одну строку) и добавить как строку тело запроса;
* вычислить от полученной 16-байтной строки значение MD5 и записать это значение в тело ответа «как есть», без преобразования в какое-либо представление;

## Описание функциональности ПС

Необходимый функционал:

* Два типа пользователей: “клиент” и “агент”, которые могут общаться друг с другом.
* Поддерживаются только личные сообщения клиента с агентом (клиент не может писать другому клиенту, а агент - другому агенту).
* Когда клиент начинает писать сообщение, он ещё не знает, с каким агентом он будет общаться.
* При получении первого сообщения клиента сервер должен выбрать свободного агента в качестве собеседника.
* Агент может вести беседу одновременно только с одним клиентом.
* Должны поддерживаться команды для регистрации в системе, завершения чата и выходы из системы.

## Спецификация функциональных требований

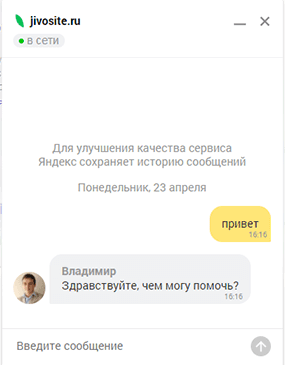
Программное обеспечение должно предоставлять возможность его пользователям осуществлять общение в сети. При этом предусмотреть возможность встраивания данного программного обеспечения в сайт, для использования в коммерческих целях

Рисунок 2.3 – Пример встроенного в сайт приложения

Так же Rest API позволит, при необходимости создать мобильное приложение для агентов или сторонних приложений.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Проектирование структур данных

В Java все объекты создаются динамически, и память под них также выделяется динамически, наличие сборщика мусора освобождает разработчика от необходимости вручную освобождать память.

Для реализации программного продукта было решено использовать потокобезопасные структуры данных, для избежания конфликта доступа к данным. Для очереди Агентов, Комнат ожидания для клиентов это ConcurrentLinkedDeque - Неограниченный параллельный deque, основанный на связанных узлах. Параллельные операции вставки, удаления и доступа выполняются безопасно для нескольких потоков. [2]

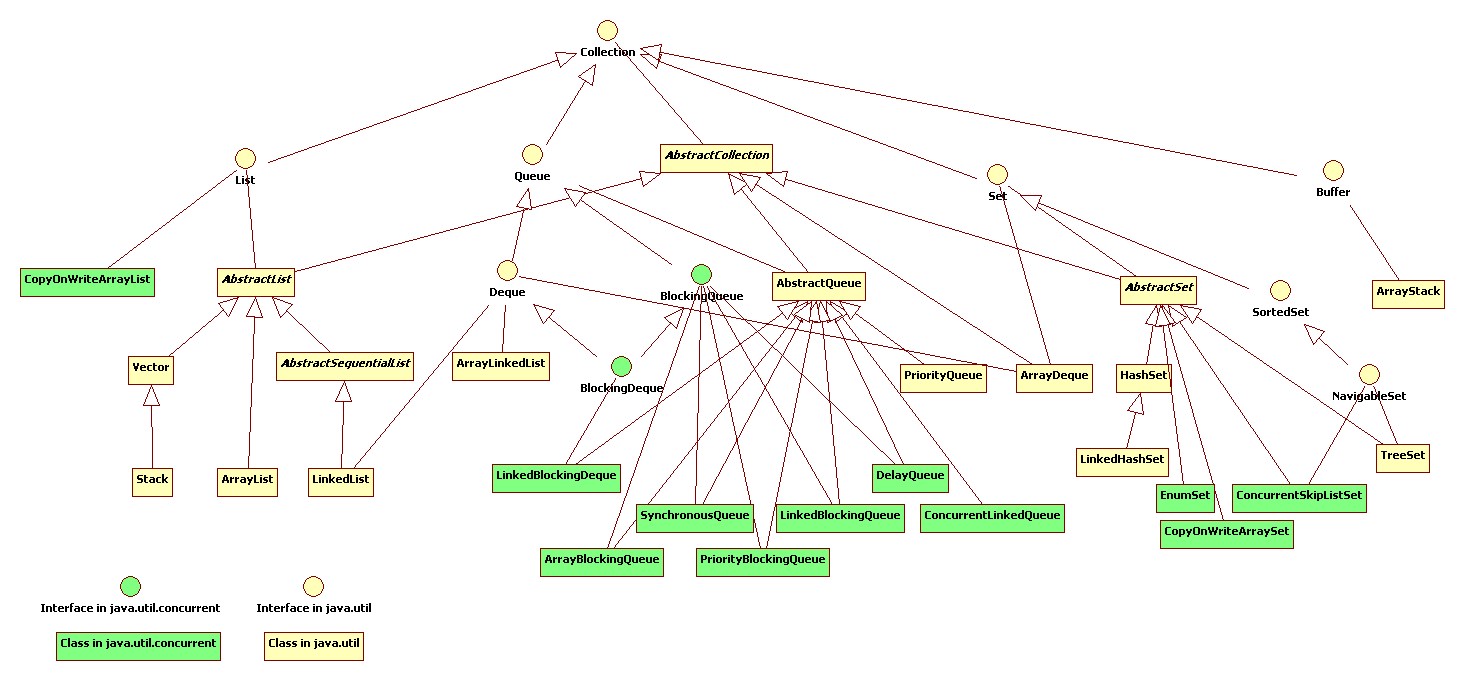


Рисунок 3.1 – иерархия Коллекций Java 8

Исходя из требований к функциональности, также необходимо было сохранять текст сообщения клиента, до того, как он подключиться к агенту. Для этих целей было решено использовать ArrayList без потокобезопасности, потому как для каждой «Комнаты ожидания» своя очередь сообщений, и необходимости в параллельном доступе нет.

## Структура пересылаемого сообщения

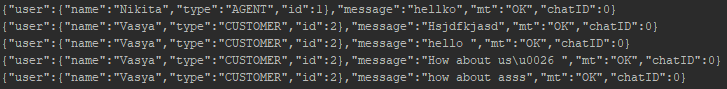
Сообщение в разрабатываемом программном обеспечении имеет не только текстовую часть, с текстом, введённым пользователем, но также информацию о самом пользователе, адресате сообщения, типе сообщения (например сообщение о регистрации нового пользователя, сообщение о выходе пользователя из сети) идентификатор комнаты, для поддержки «мультиклиентности» агента. Так как использование протокола WebSocket подразумевает передачу текстовой информации, возникла необходимость в использовании библиотеки Google Gson, предназначенной для преобразования Java-объектов в текстовый формат JSON (сериализация) и обратного преобразования (десереализация).

Рисунок 3.2 – результат преобразования Java-объектов в текстовый формат JSON

Как видно из рисунка 3.2 JSON представляет собой упорядоченный набор значений, где объект заключается в фигурные скобки, а поля представляются как пары ключ – значение. Для правильной работы обратного преобразования, принципиально важно, что б объект имел те же имена что и на отправляющей стороне.

## . Разработка алгоритма обработки регистрационного сообщения

## 

Для регистрации пользователя на сервере отправляющая сторона указывает в качестве MessageType значение “REG” После чего серверная часть приложения обрабатывает его по алгоритму представленному на рисунке 3.3

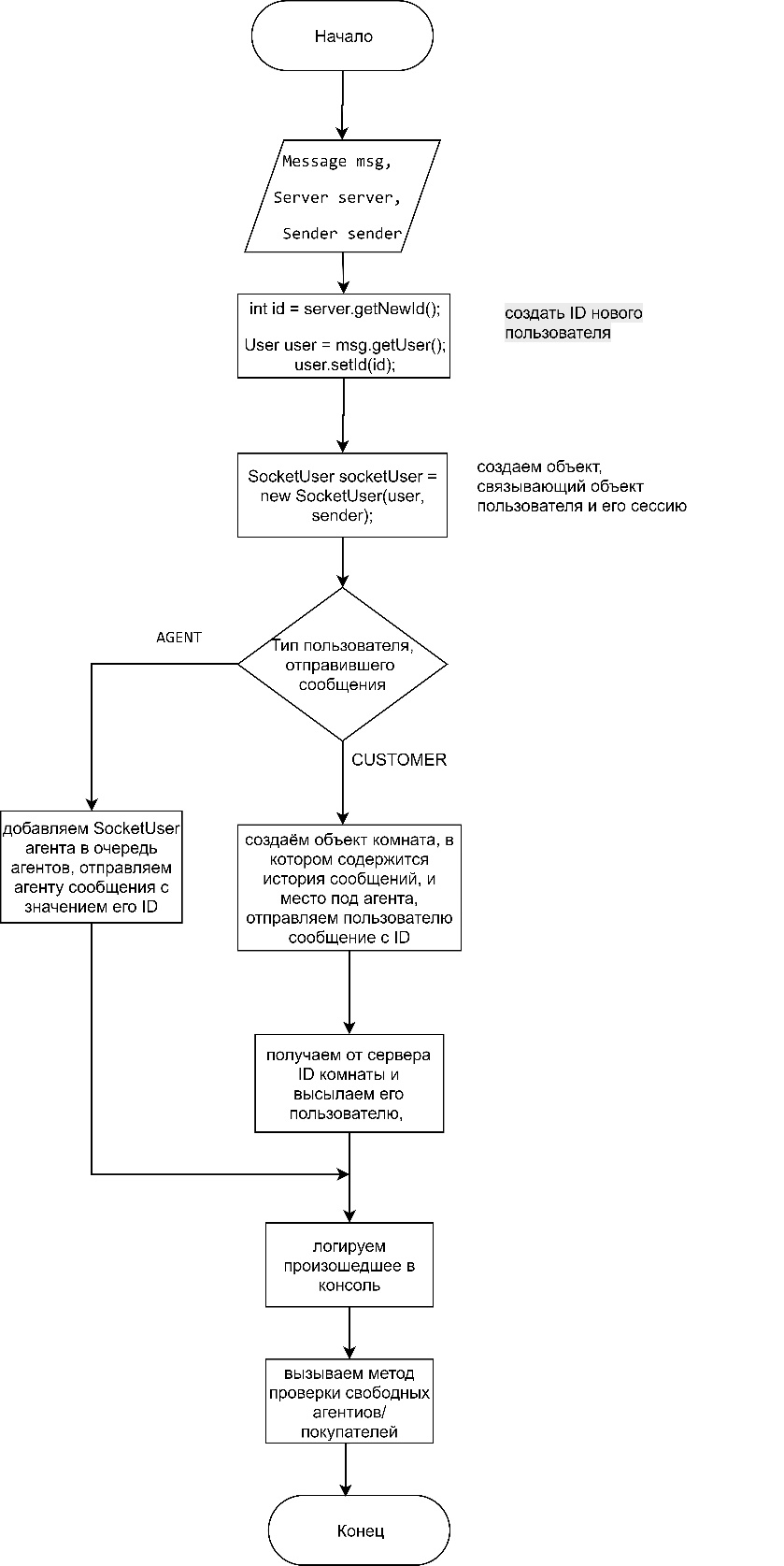


Рисунок 3.3 – Схема алгоритма обработки регистрации пользователя

## Разработка алгоритма обработки информационного сообщения

При получении сообщения с типом “OK”, сервер обрабатывает его как текстовое сообщение, которое нужно отправить адресату. Для этого проверяется, существует ли комната в которой находится отправитель, и оба ли собеседника в комнате. Если пользователь подключен, то по его идентификатору ищется комната, в которой так же записан объект адресата

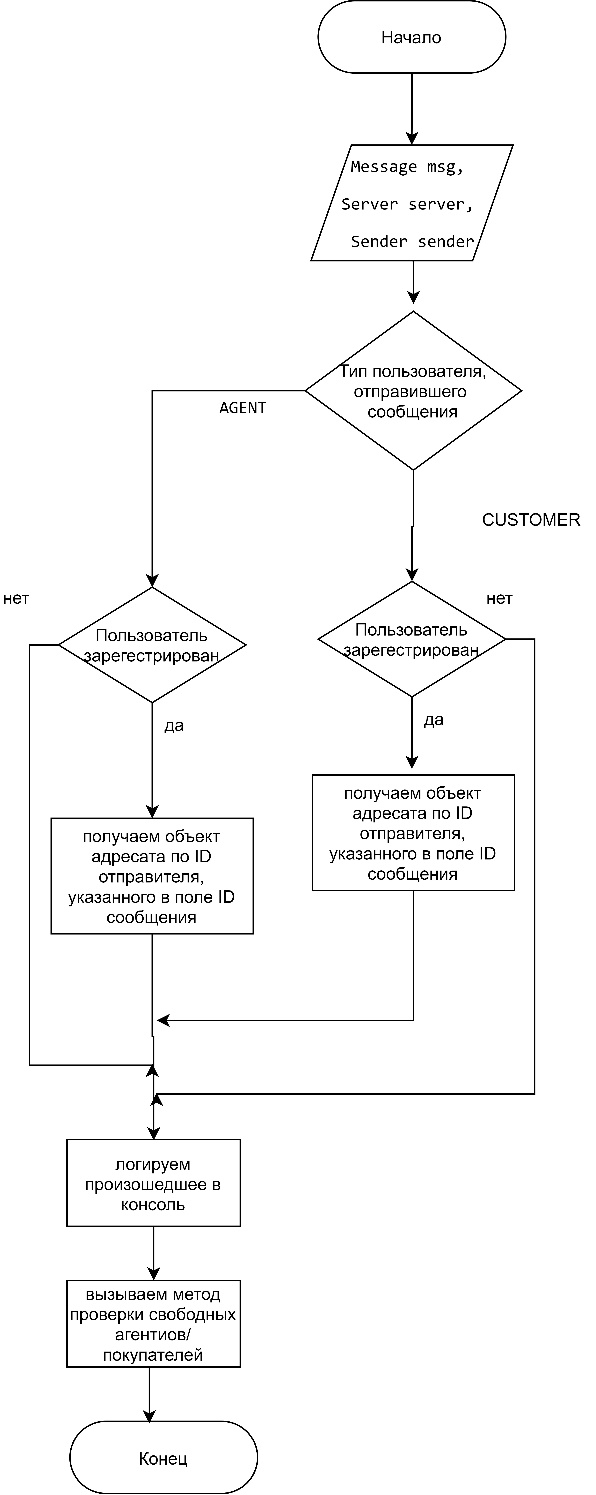
и отправляет сериализированный объект сообщения адресату.

Рисунок 3.4 – Схема алгоритма отправки сообщения

## Разработка алгоритма выхода пользователей из комнаты

Согласно требованиям к разрабатываемому программному обеспечению, необхожимо предусмотреть возможность выхода пользователя из диалога, невыходя из приложения, и, может быть, вновь в него вернуться. Схема алгоритма представлена на рисунке 3.4

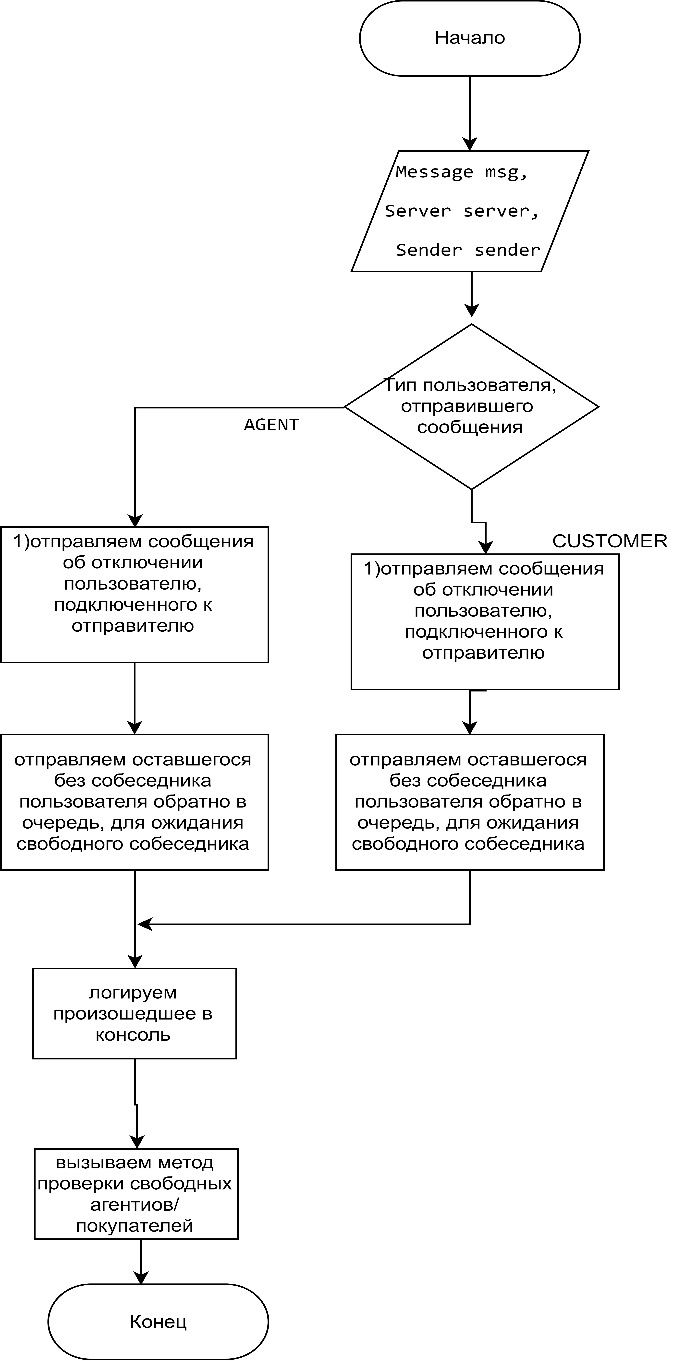


Рисунок 3.5 – Схема алгоритма выхода пользователя из системы

# СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

## Разработка интерфейса программы

В ходе постановки задачи было решено реализовать 2 интерфейса, для использования сервиса – стационарная версия для ПК и универсальная версия, для всех устройств, имеющих браузер.

## Разработка интерфейса настольного приложения

В ходе разработки активно использовался паттерн, MVС который предполагает разделение данных, их визуализации и управления. Окно приложения спроектировано при помощи инструмента SceneBuilder представленный на рисунке 4.1, который автоматически генерирует fxml файл разметки.

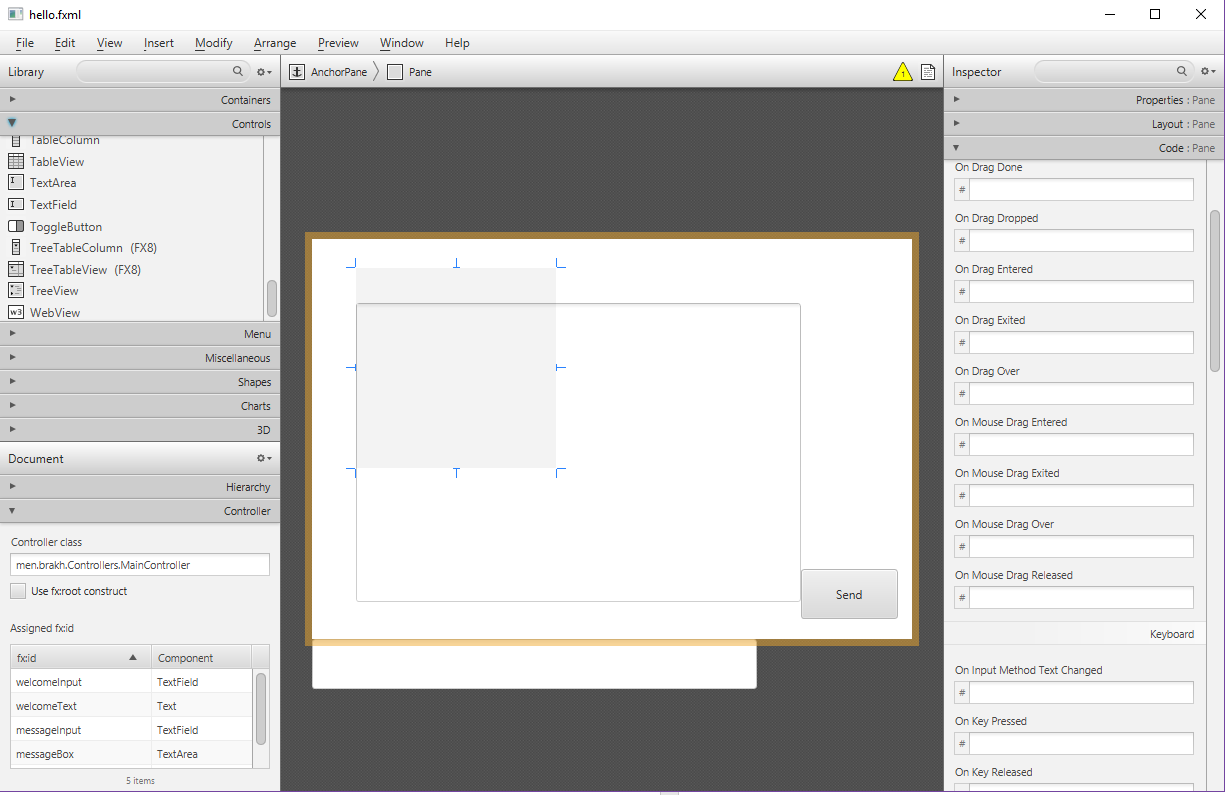


Рисунок 4.1 – Инструмент для разработки UI и генерации разметки

Для изменения вида компонентов по умолчанию используются JavaFX CSS стили, которые работают так же как обычный веб – CSS. Сразу после запуска приложения пользователю предлагают ввести своё, после нажатия на Enter приложение автоматически отправит сообщение регистрации. Процесс регистрации отображен на рисунок 4.2. При этом соединение с сервером создаётся вместе с отображением окна. При успешной регистрации появляется

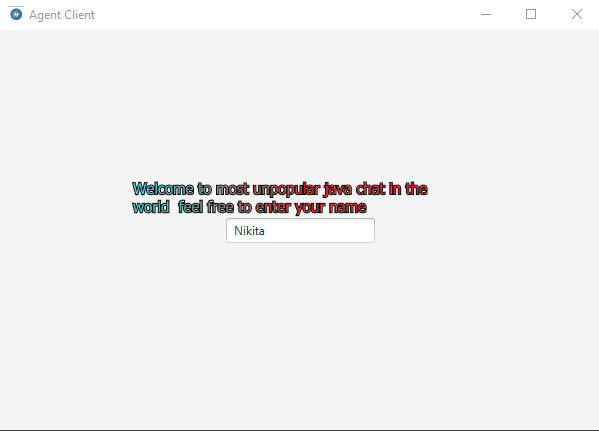


Рисунок 4.2 – Окно приветствия и регистрации

Для выхода из программы можно написать команду ''/exit'' или просто закрыть программу. Для смены собеседника для покупателей предусмотрена комманда “leave”. Интерфейс максимально интуитивно понятный и не требует никакой документации или инструкции.

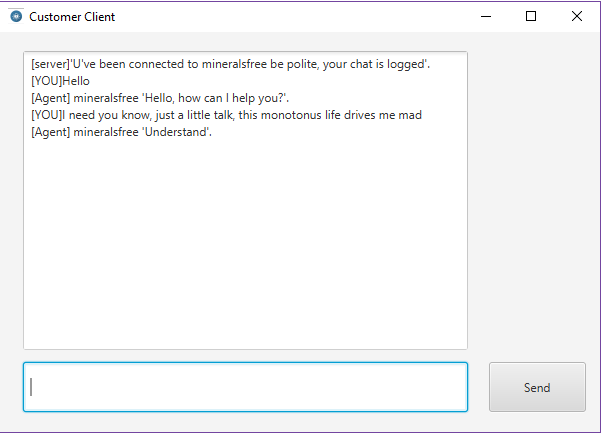


Рисунок 4.2 Интерфейс чата

## 4.1.2Разработка интерфейса настольного приложения

## Разработка интерфейса веб-приложения

Традиционно веб-приложения разрабатываются при помощи связки HTML – CSS – JavaScript и данный проект не стал исключением. В ходе из технической части были использованы новейшие нововведения JavaScript – классы, а также встроенные методы сериализации и десериализации JavaScript и реализацию WebSocket.

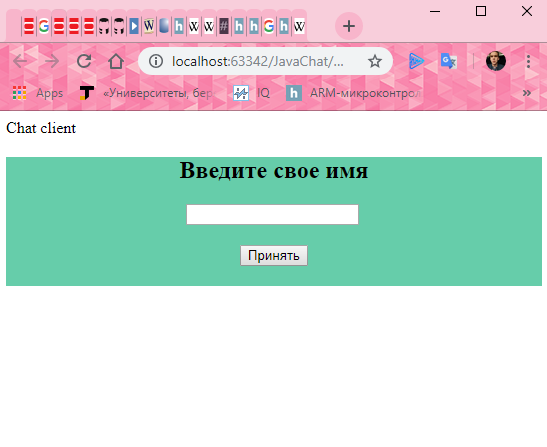
Интерфейс веб-версии приложения идентичен десктопной с небольшими отличиями. При открытии html страницы мы видим подобную страницу приветствия (рисунок 4.3).

Рисунок 4.3 – Интерфейс окна приветствия веб-версии

Однако на данном этапе разработки только веб-версия приложения имеет возможность поддерживать подключения сразу к нескольким клиентам, данный функуионал продемонстрирован на рисунке 4.4

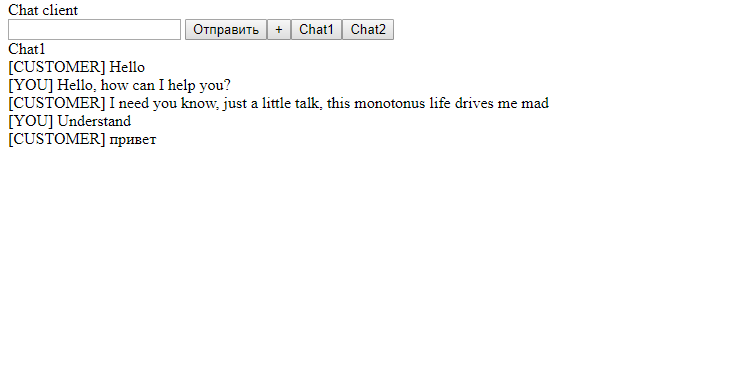


Рисунок 4.4 – Интерфейс чата веб-версии агента

# 5.ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

## 5.1. Тестирование регистрации и подключения пользователя

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Регистрация пользователя | 1. Создать объект пользователя на стороне клиента, отправить сообщение регистрации на сервер | Появление интерфейса чата | Тест пройден! |
| 1 | Подключение к Свободному агенту | Регистрируем в системе агента с помощью одного из клиента | Подключение Агента к клиенту. Сообщение от серевера о том, что произошло подключение | Тест пройден! |

Программа прошла данный этап тестирования, никаких проблем, ошибок и багов выявлено не было.

## 5.2. Тестирование отправки сообщения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Отправка сообщения | 1. Набор текста в поле, нажатие клавиши Enter | Появление сообщение в окне собеседника, появление сообщение в окне польщователя | Тест пройден! |

## 5.3. Тестирование выхода из системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Выход из системы | 1. Набор комманды /exit 2. Закрытие приложени/вкладки | Появление сообщение в окне собеседника уведомления о том что вы покинули беседу. | Тест пройден! |

## . Вывод из прохождения тестирования

Программа успешно прошла все тесты, что показывает корректность работы программы и соответствие функциональным требованиям.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта было разработано приложение, имеющий минимальный функционал требуемый для осуществления общение агента и покупателя

При тестировании и отладке были выявлены случаи некорректной работы программы, однако большинство из них были исправлены.

Простой и удобный пользовательский интерфейс позволяет пользователю легко привыкнуть к приложению.

Написанный код легко модифицируется, а изменения вида может быть модифицирован путем изменения констант. В дальнейшем возможны улучшения и доработки, вносящие новый функционал в программу.

В процессе разработки, я изучил принципы ООП, изучил коллекции в Java,узнал особенности проектировки многопоточного ПО. Научился собирать проекты при помощи maven, разрабатывать веб-приложения с применением интерпретируемого языка JavaScript.

# СПИСОК ЛИТЕРАТУРЫ

[1] RFC 6455 — The WebSocket protocol [Электронный ресурс] - Режим доступа: https://tools.ietf.org/html/rfc6455

[2] Java Platform Standard Edition 8 Documentation [Электронный ресурс] Режим доступа: Дата доступа: 16.04.18

[3] Шилдт Г. – Java 8. Полное руководство. 9-е издание, 2015 – Глава 22. Работа в сети. – С.811–832.

[4] Bloch J. – Effective Java – 3rd edition, 2018 – Chapter 12. Serialization. – Pages 339–366

# ПРИЛОЖЕНИЕ 1

**Схема алгоритма работы программы**

# ПРИЛОЖЕНИЕ 2

**Исходный код программы**

Server.java

package men.brakh;

import men.brakh.Listener.WebSocketListener;

import org.springframework.context.annotation.Configuration;

import java.io.IOException;

import java.util.LinkedList;

@Configuration

public class Server {

private int id = 0;

public static final int PORT = 1488;

public AgentQueue agentQueue;

public CustomerQueue customerQueue;

public Logger logger;

public static LinkedList<ServerSomthing> serverList = new LinkedList<ServerSomthing>(); // список всех нитей - экземпляров

// сервера, слушающих каждый своего клиента

public synchronized void checkFreeAgents(){

Chat customerChat = customerQueue.GetFreeCustomers();

if (( customerChat!= null) && (!agentQueue.isQueueEmpty())) {

SocketUser agent = agentQueue.PollAgent();

Chat CustomerChat = customerQueue.GetFreeCustomers();

CustomerChat.setAgent(agent);

String chatid = String.valueOf(CustomerChat.getId());

agent.GetSender().ServerRegID(chatid);

agent.GetSender().ServerSend("U've been connected to "+CustomerChat.getCustomer().GetUser().getName()+" be polite, your chat is logged",Integer.parseInt(chatid));

CustomerChat.getCustomer().GetSender().ServerSend("U've been connected to "+CustomerChat.getAgent().GetUser().getName()+" be polite, your chat is logged",Integer.parseInt(chatid));

logger.log(String.format("Pair [Agent] %s and [Customer] %s", agent.GetUser().getName(), customerChat.getCustomer().GetUser().getName()," Created"));

}

}

public Server() throws IOException {

agentQueue = new AgentQueue();

customerQueue = new CustomerQueue();

logger = new Logger("Log.txt");

// SocketListener socketListener = new SocketListener(PORT,this);

WebSocketListener webSocketListener = new WebSocketListener(this);

// try {

// socketListener.join();

// webSocketListener.join();

// } catch (InterruptedException e) {

// e.printStackTrace();

// }

System.out.println("Server not Started");

}

public synchronized int getNewId() {

return ++id;

}

public static void main(String[] args) throws IOException {

new Server();

}

}

SpringConfig.java

package men.brakh;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.io.IOException;

@Configuration

public class SpringConfig {

private static Server server;

@Bean

public static Server server(){

try {

server = new Server();

return server;

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

public static Server getServer() {

return server;

}

}

Chat.java

package men.brakh;

import men.brakh.Sender.Sender;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

public class Chat {

private SocketUser agent;

private SocketUser customer;

private int id;

private ArrayList<Message> messageArrayList = new ArrayList<Message>();

public Chat(User user, Sender ss,int id){

this.customer = new SocketUser(user,ss);

this.agent = null;

this.id = id;

}

public Type getUserType(String name){

if (getAgent().getUser().getName().equals(name)){

return Type.AGENT;

}

if (getCustomer().getUser().getName().equals(name)){

return Type.CUSTOMER;

}

return null;

}

public Type getUserType(int id){

if (getAgent().getUser().getId()==id){

return Type.AGENT;

}

if (getCustomer().getUser().getId()==id){

return Type.CUSTOMER;

}

return null;

}

public SocketUser getAgent() {

return agent;

}

public boolean isAgentHere(){

if (agent==null) return false; else return true;

}

public SocketUser getCustomer() {

return customer;

}

public Sender getCustomerSS(){

return customer.GetSender();

}

public Sender getAgentSS(){

return agent.GetSender();

}

public void setAgent(SocketUser agent) {

this.agent = agent;

for(Message mg : messageArrayList){

this.agent.GetSender().send(mg.getString(this.agent.GetSender().toString()));

}

}

public void setCustomer(SocketUser customer) {

this.customer = customer;

}

public void addMessage(Message msg){

messageArrayList.add(msg);

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

}

WebServer.java

package men.brakh;

import men.brakh.handler.Handler;

import men.brakh.handler.MessageHandler;

import javax.websocket.\*;

import javax.websocket.server.ServerEndpoint;

import javax.websocket.CloseReason.CloseCodes;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.logging.Logger;

@ServerEndpoint(value = "/chat")

public class WebServer {

public static Server server;

public static Map<String, Integer> hashMap = new HashMap<String, Integer>();

private Logger logger = Logger.getLogger(this.getClass().getName());

@OnOpen

public void onOpen(Session session) {

logger.info("Connected ... " + session.getId());

}

@OnMessage

public String onMessage(String message, Session session) {

Message msg = Message.getMessage(message);

if (!hashMap.containsKey(session.getId())){

hashMap.put(session.getId(),msg.getUser().getId());

}

new Handler(session,msg,server);

System.out.println(Message.getMessage(message).getString("web"));

if (message.equals("quit")) {

try {

session.close(new CloseReason(CloseCodes.NORMAL\_CLOSURE, "Game ended"));

} catch (IOException e) {

throw new RuntimeException(e);

}

}

return null;

}

@OnClose

public void onClose(Session session, CloseReason closeReason) {

logger.info(String.format("Session %s closed because of %s", session.getId(), closeReason));

if (hashMap.containsKey(session.getId())){

Integer id = hashMap.get(session.getId());

Type uType = (server.customerQueue.getByID(id).getUserType(id));

Message msg = new Message(new User("any", uType,id),"",MessageType.EXIT);

new MessageHandler(msg,server,null);

}

}

}