

Министерство образования Новосибирской области
Государственное бюджетное профессиональное образовательное учреждение
Новосибирской области
«Новосибирский колледж электроники и вычислительной техники»
(ГБПОУ НСО «Новосибирский колледж электроники и вычислительной техники»)

УТВЕРЖДАЮ
Зам. директора по УР
ГБПОУ НСО «Новосибирский колледж
электроники и вычислительной техники»
_____ Е. А. Борисова
« ____ » _____ 2023 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту на тему:

Разработка корпоративного мессенджера

Автор дипломного проекта (работы)	_____	Костиков П.А.
	подпись	
Обозначение дипломного проекта	<u>ДП НКЭиВТ - 09.02.07-9-23</u>	Группа <u>9ис-430</u>
Специальность	<u>09.02.07 «Информационные системы и программирование»</u>	
	номер, наименование	
Руководитель проекта	_____	Легостаева А.Б.
	подпись, дата	
Рецензент	_____	Щерба Е.А.
	подпись, дата	
Консультант по разделам:		
Консультант по экономической части	_____	Горбаченко Ю.В.
	подпись, дата	
Нормоконтролер	_____	Щерба Е.А.
	подпись, дата	

Новосибирск 2023

Министерство образования Новосибирской области
Государственное бюджетное профессиональное образовательное учреждение
Новосибирской области
«Новосибирский колледж электроники и вычислительной техники»
(ГБПОУ НСО «Новосибирский колледж электроники и вычислительной техники»)

УТВЕРЖДАЮ
Зам. директора по УР
ГБПОУ НСО «Новосибирский колледж
электроники и вычислительной техники»
_____ Е. А. Борисова
«_____» _____ 2023г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

По специальности 09.02.07 «Информационные системы и программирование»
Студенту Костикову П.А. Группы 9ис-430
Тема работы: Разработка корпоративного мессенджера

Утверждена приказом по ГБПОУ НСО № 123 от 26.01.2023
«Новосибирский колледж электроники и
вычислительной техники»

Сроки представления проекта к защите: 13.06.2023
Исходные данные для проектирования: Техническое задание по разработке
корпоративного мессенджера

Содержание пояснительной записки:

1. Общая часть (теоретическая)
2. Основная часть
 - 2.1. Взаимодействие с СУБД
 - 2.2. Spring Boot приложение
 - 2.3. Клиентское приложение
3. Охрана труда
4. Организационно-экономический раздел
- Заключение
- Список используемых источников

Студент _____	Костиков П.А.
Руководитель проекта _____	Легостаева А.Б.
Консультант по разделам:	
Основной раздел _____	Легостаева А.Б.
Охрана труда _____	Легостаева А.Б.
Организационно-экономический _____	Горбаченко Ю.В.

Заключение руководителя проекта.

Дипломный проект закончен. Считаю возможным допустить студента

Костикова Петра Александровича

К защите дипломного проекта

Рецензентом назначить Щерба Е.А.

Председатель ПЦК _____ Десюк А.М.

Зав. отделением _____ Ломова А.И.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1. ОБЩАЯ ЧАСТЬ (ТЕОРЕТИЧЕСКАЯ).....	7
1.1. Цель разработки и анализ её использования	7
1.2. Анализ технологий и возможных средств решения проблемы	8
1.3. Выбор средств и технологий.....	9
2. ОСНОВНАЯ ЧАСТЬ.....	16
2.1. Взаимодействие с СУБД.....	16
2.1.1. Развертывание MySQL для разработки	16
2.1.2. Организация структуры базы данных	18
2.2. Spring Boot приложение.....	21
2.2.1. Инициализация и настройка зависимостей проекта.....	21
2.2.2. Подключение приложения к базе данных	23
2.2.3. Сущности и репозитории.....	24
2.2.4. Сервисы и их имплементации.....	27
2.2.5. Обработка HTTP запросов и их ошибок.....	30
2.2.6. WebSocket соединение.....	33
2.2.7. Итоговая структура системы, сторона сервера	36
2.3. Клиентское приложение	36
2.3.1. Организация сетевых соединений с сервером	36
2.3.2. Обработка данных полученных с сервера	39
2.3.3. Интерфейс приложения	40
3. ОХРАНА ТРУДА	43
3.1. Общие требования охраны труда	43
3.2. Требования охраны труда перед началом работы	45

3.3. Требования охраны труда во время работы	46
3.4. Требования охраны труда по окончании работы	47
4. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКИЙ РАЗДЕЛ	48
4.1. Организация рабочего места	48
4.2 Расчет затрат на разработку корпоративного мессенджера	49
4.2.1. Расчет затрат на материалы.....	49
4.2.2 Расчет основной заработной платы разработчика	50
4.2.3 Дополнительная заработная плата	50
4.2.4 Отчисления на единый социальный налог	51
4.2.5 Расчет накладных расходов.....	51
4.2.6. Расчет стоимости работы оборудования.....	52
4.2.7. Расчет цены программного продукта.....	54
ЗАКЛЮЧЕНИЕ	55
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	56
ПРИЛОЖЕНИЕ	58

ВВЕДЕНИЕ

В настоящее время организовать рабочий процесс взаимодействия людей в компании без различных средств связи невероятно сложно, если вообще возможно. Для этого могут использоваться разные средства: Miro, Jira, Trello, телефонная связь, корпоративная или обычная почта и так далее. Но самым удобным и быстрым способом связаться с другим человеком является мессенджер. Мессенджер – программа для мгновенного обмена текстовыми сообщениями, аудиозаписями, фотографиями и другими мультимедиа файлами. Существующие приложения могут не соответствовать требованиям компании, а также не могут обеспечить должный уровень безопасности: данные хранятся не на носителях компании, а на носителях создателя приложения. Используя сторонний софт, мы не всегда можем его модернизировать под наши цели и задачи. В таком случае, проще будет разработать свой собственный.

Цели:

- разработать функциональный и эффективный мессенджер;
- создать удобный и интуитивно понятный пользовательский интерфейс;
- обеспечить безопасность и конфиденциальность данных, хранящихся в системе, чтобы предотвратить несанкционированный доступ и утечку информации.

Задачи:

- изучение существующих приложений;
- определение требований и потребностей предприятия;
- проектирование архитектуры и структуры системы, включая определение базы данных, интерфейсов пользователя и функциональных модулей;
- создание пользовательского интерфейса;
- разработка механизмов безопасности и конфиденциальности данных, включая аутентификацию пользователей и контроль доступа;
- проведение тестирования системы, включая проверку ее функциональности, надежности и соответствия требованиям предприятия.

1. ОБЩАЯ ЧАСТЬ (ТЕОРЕТИЧЕСКАЯ)

1.1. Цель разработки и анализ её использования

Целью разработки является создание цельной системы для обмена сообщениями внутри компании, то есть приложение, которым должны пользоваться сотрудники для передачи данных, и сервер, который обрабатывает и хранит сообщения.

Можно выделить основные идеи, которые не будут изменяться и останутся идентичными для вне зависимости от требования компании:

- должны существовать каналы, которые собой представляют набор пользователей, в которые можно добавить или же удалить участника, и список сообщений, которые в него отправили;
- каждое сообщение должно иметь канал и владельца, который может взаимодействовать со своими, уже отправленными, сообщениями, то есть удалить или же изменить;
- после того как пользователь отправил сообщение, оно должно прийти всем пользователям, у которого есть доступ в канал;
- человек не может читать, отправлять и как-либо взаимодействовать с сообщениями и каналами, до входа в систему.

С помощью результата разработки можно организовать контакт внутри и между командами, отделами, помимо этого в дальнейшем развивать систему, например, можно добавить календарь для групп, где будут указаны дни, в которые будут проводится какие-либо мероприятия, или же добавить возможность отправлять голосовые сообщения, если это требуется компании.

1.2. Анализ технологий и возможных средств решения проблемы

Изучив популярные магазины приложений для разных устройств, такие как: Play Market, AppStore, Microsoft Store, можно увидеть большое количество мессенджеров (см. рис. 1), которые можно использовать в компании: Telegram, WhatsApp, Skype, Discord, но как было указано выше, в данном подходе имеются недостатки, которые могут повлиять на всю работу компании.

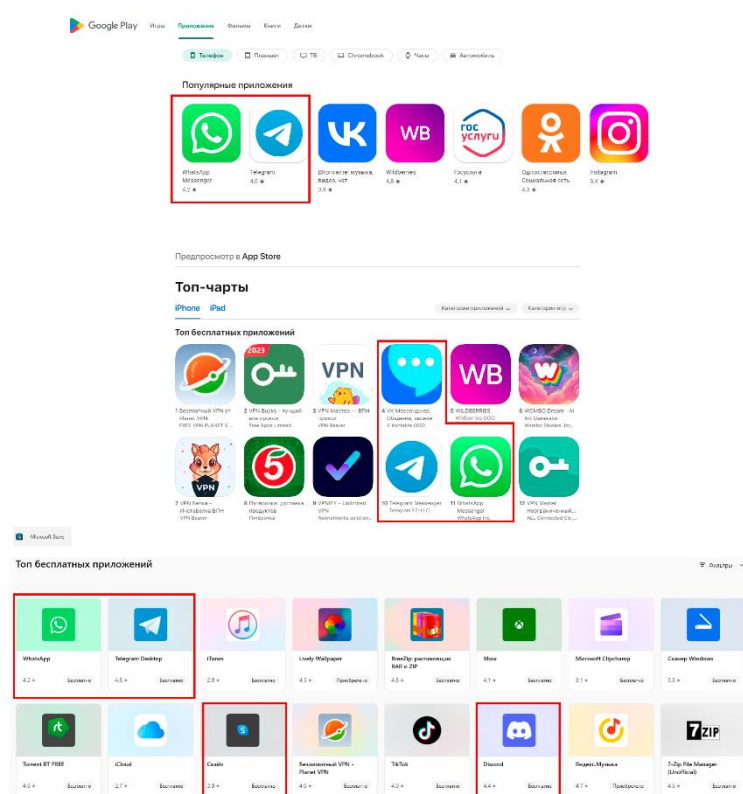


Рис. 1. Мессенджеры, имеют красную обводку, в списках самых скачиваемых приложений разных магазинов

Поскольку в настоящее время существует огромное количество мессенджеров конкуренция в данной сфере невероятна велика. Стоит также учитывать, что на разработку, тестирование и введение в эксплуатацию уйдет время. Но из-за того, что приложение будет узконаправлено на конкретную компанию, её цели и задачи, то создание системы резонно.

1.3. Выбор средств и технологий

Одной из главных проблем является передача данных другим членам группы, как только один из пользователей отправил сообщение. Вариантом решения данной проблемы является открытие соединения, которое будет уведомлять пользователя о действиях, которые произошли в доступных ему каналах. Под данный вариант подходит протокол WebSocket («веб-сокет»), описанный в спецификации RFC 6455, обеспечивающий возможность обмена данными между клиентом и сервером через постоянное соединение. Данные передаются по нему в обоих направлениях в виде «пакетов», без разрыва соединения и дополнительных HTTP-запросов.

Но помимо обмена сообщениями пользователю требуется пройти аутентификацию или же регистрацию, узнать доступные ему чаты, получить последние сообщения в определенном канале, загрузить более старые сообщения. Данные действия происходят единственный раз, либо достаточно редко, для этого можно использовать HTTP-запросы к серверу.

HTTP (от англ. HyperText Transfer Protocol) — протокол передачи гипертекста. Это набор правил, по которым данные в интернете передаются между разными источниками, обычно между компьютерами и серверами. Интернет-протокол HTTP — это шаблон, по которому формируется запрос на передачу данных, после чего происходит ответ и передаются интернет-страницы, видео, аудио и текст.

Протокол HTTP нужен для стандартизации. Благодаря ему все компьютеры в интернете могут расшифровать присланные данные и отправлять их в виде, понятном другим компьютерам.

Структура HTTP-запроса (см. рис. 2) всегда одинакова:

1. стартовая строка, в которой определяется адрес, по которому отправляется запрос, и тип сообщения. Указывается метод, который определяет действия при получении этого сообщения. Это может быть чтение данных, их отправка, изменение или удаление;

2. заголовки (Headers), в которых прописаны определённые параметры сообщения. Например, может быть напрямую задан язык;

3. тело запроса (Request Body), текст сообщения — данные, которые передаются. Например, файлы, отправляемые на сервер.

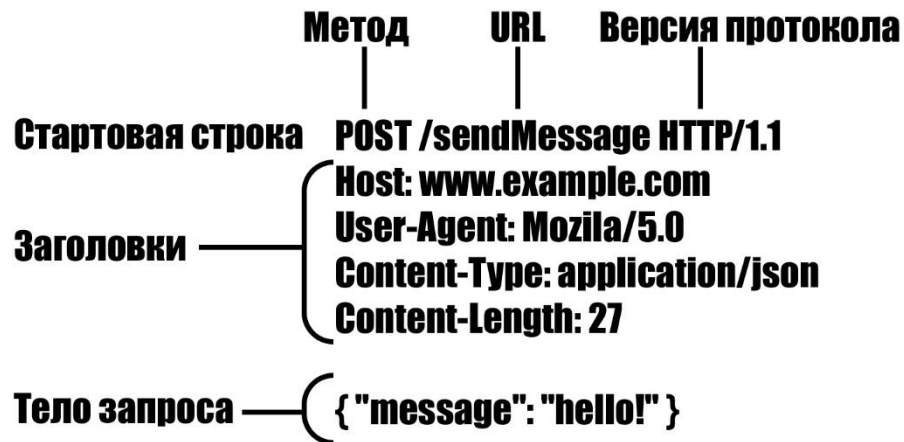


Рис. 2. Структура HTTP запроса

Как указано ранее, после запроса должен быть ответ со стороны сервера, разница его структуры только в том, что вместо стартовой строки строка статуса (см. рис. 3), в которой указана версия протокола, код состояния (см. табл. 1), показывающий был ли успешен запрос, и пояснение, текстовое описание кода состояния, поясняющее пользователю код.



Рис. 3. Структура строки статуса HTTP ответа

Классификация кодов состояния HTTP ответов

Область группы	Наименование	Описание
100 - 199	Информационные	В этот класс выделены коды, информирующие о процессе передачи. При работе через протокол версии 1.0 сообщения с такими кодами должны игнорироваться. В версии 1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но серверу отправлять что-либо не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. Прокси-серверы подобные сообщения должны отправлять дальше от сервера к клиенту.
200 – 299	Успешные	Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.
300 – 399	Перенаправление	<p>Коды этого класса сообщают клиенту, что для успешного выполнения операции необходимо сделать другой запрос, как правило, по-другому URI. Из данного класса пять кодов 301, 302, 303, 305 и 307 относятся непосредственно к перенаправлениям. Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке Location. При этом допускается использование фрагментов в целевом URI.</p> <p>По последним стандартам клиент может производить перенаправление без запроса пользователя только если второй ресурс будет запрашиваться методом GET или HEAD. В предыдущих спецификациях говорилось, что для избежания круговых переходов пользователя следует спрашивать после 5-го подряд перенаправления. При всех перенаправлениях, если метод запроса был не HEAD, то в тело ответа следует включить короткое гипертекстовое сообщение с целевым адресом, чтобы в случае ошибки пользователь смог сам произвести переход.</p>
400 – 499	Ошибки клиента	Класс кодов 4xx предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя
500 - 599	Серверные ошибки	Коды 5xx выделены под случаи необработанных исключений при выполнении операций на стороне сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.

Помимо технологий передачи информации в проекте требуются технология для создания приложения. Большая часть персональных компьютеров работает на операционной системе Windows (см. рис. 4), пользователи хорошо знакомы с этой операционной системой.

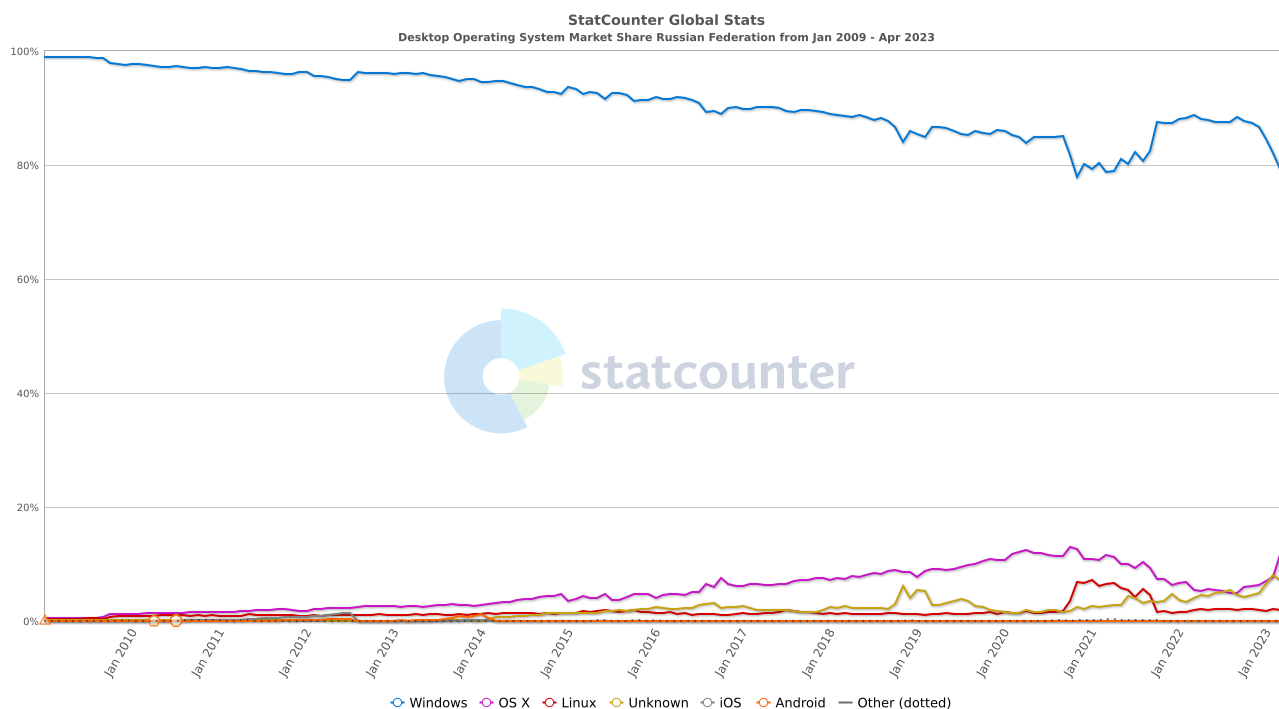


Рис. 4. Процентное соотношение использования семейств операционных систем на персональных компьютерах в РФ, за период с января 2009 по апрель 2023, по версии StatCounter

Операционная система, сокр. ОС (англ. operating system, OS) — комплекс управляющих и обрабатывающих программ, которые, с одной стороны, выступают как интерфейс между устройствами вычислительной системы и прикладными программами, а с другой стороны — предназначены для управления устройствами, управления вычислительными процессами, эффективного распределения вычислительных ресурсов между вычислительными процессами и организации надёжных вычислений. Это определение применимо к большинству современных операционных систем общего назначения.

Исходя из вышесказанного, для создания пользовательского приложения необходим стек технологий, позволяющий создавать программы под Windows. Существует два варианта: WPF и Windows Forms. Последний является устаревшим,

обладает меньшим функционалом на фоне первого, не позволяет отделить интерфейс от логики, по этой причине стоит использовать WPF.

Windows Presentation Foundation (WPF) — аналог WinForms, система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования.

XAML — это декларативный язык разметки. Как и в случае с моделью программирования .NET, XAML упрощает создание пользовательского интерфейса для приложения .NET. Можно создать видимые элементы пользовательского интерфейса в декларативной XAML-разметке, а затем отделить определение пользовательского интерфейса от логики времени выполнения, используя файлы кода программной части, присоединенные к разметке с помощью определений разделяемых классов. Язык XAML напрямую представляет создание экземпляров объектов в конкретном наборе резервных типов, определенных в сборках. В этом заключается его отличие от большинства других языков разметки, которые, как правило, представляют собой интерпретируемые языки без прямой связи с системой резервных типов. Язык XAML обеспечивает рабочий процесс, позволяющий нескольким участникам разрабатывать пользовательский интерфейс и логику приложения, используя потенциально различные средства.

При представлении в виде текста файлы XAML являются XML-файлами, которые обычно имеют расширение .xaml. Файлы можно сохранять в любой кодировке, поддерживаемой XML, но обычно используется кодировка UTF-8.

Для удобства разработки, сборки, тестирования используется IDE Visual Studio 2022, в которой имеется функционал дающий возможность простым способом создать проект WPF, подключить сторонние библиотеки, управлять структурой проекта.

Серверная часть системы помимо технологий передачи данных должна уметь взаимодействовать с базами данных, в которых хранится вся информация. С этой целью можно использовать систему управления базами данных, в настоящий момент существует большое количество разнообразных вариантов, одними из наиболее используемыми в данный момент являются: PostgreSQL, Microsoft SQL Server, MySQL, SQLite, MongoDB, Redis и Oracle.

Системы управления базами данных предназначены, являются прикладным программным обеспечением общего назначения, предназначены для работы с большими массивами информации, которые обычно представлены в виде табличных структур. С помощью СУБД автоматизируют технологические процессы по созданию, хранению и анализу электронных данных. Такие системы служат основой для разработки современных информационно-справочных программных комплексов.

БД классифицируются по модели данных:

- сетевые;
- иерархические;
- реляционные;
- объектно-ориентированные;
- XML-ориентированные и другие.

По причине того, что сервера, на которых планируется запустить часть системы, которая обрабатывает данные отосланные пользователями, и база данных, могут использовать разные ОС, требуется использовать максимально кроссплатформенные средства для создания серверной части.

Для решения данной проблемы можно воспользоваться фреймворком Spring Boot, который базируется на Java. Spring Boot - это популярный фреймворк для создания веб-приложений с использованием Java. Это часть фреймворка Spring, которая представляет собой набор инструментов и библиотек для создания приложений корпоративного уровня. Spring Boot упрощает создание автономных приложений производственного уровня, которые можно легко развернуть и запустить с минимальной конфигурацией.

Благодаря тому, что приложения Java транслируются в специальный байт-код, который исполняется на виртуальной Java-машине (JVM), данное приложение можно запустить на любом устройстве, независимо от архитектуры, при условии, что для данного устройства существует JVM. Существует два вида Java: Java Runtime Environment и Java Development Kit. Первая является средой выполнения, наиболее важным компонентом является java, он запускает JVM, который в свою очередь запускает скомпилированное приложение. В свою очередь JDK полноценный набор инструментов для разработки программного обеспечения, который так же включает в себя все компоненты JRE (см. рис. 5), компилятор языка, разнообразные утилиты и инструменты, например, отладчик jdb.

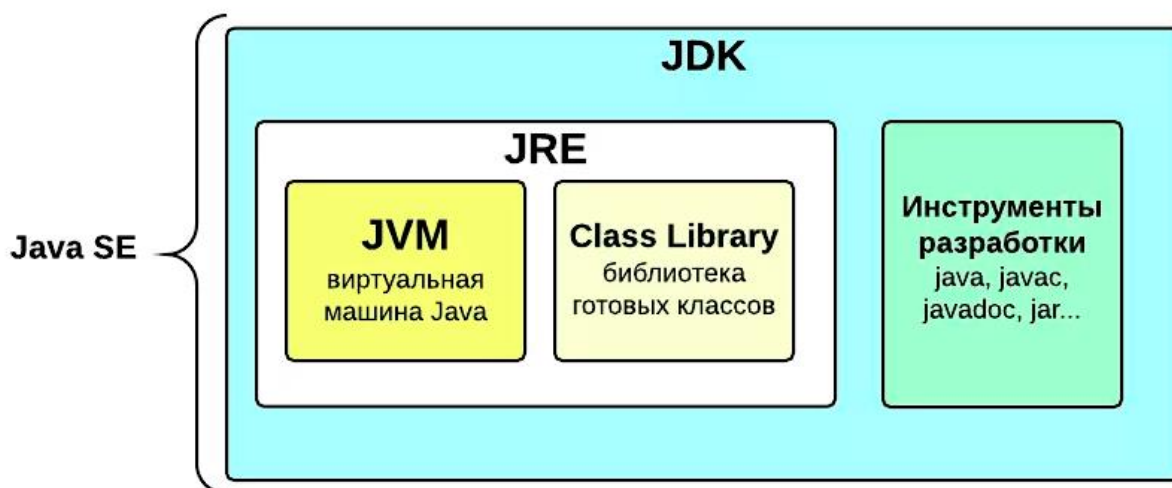


Рис. 5. Составные части JDK и JRE

Как и для клиентского приложения, может потребоваться удобная среда разработки, которая облегчит создания серверной части системы. На данный момент самыми используемыми являются: NetBeans, IntelliJ IDEA, Eclipse, Android Studio.

2. ОСНОВНАЯ ЧАСТЬ

2.1. Взаимодействие с СУБД

В качестве СУБД для данной системы была выбрана MySQL, поскольку часто можно увидеть её использование в тандеме с Spring Boot приложением.

2.1.1. Развертывание MySQL для разработки

Но из-за того, что разработка происходит на персональном компьютере, который использует ОС Windows 10, возникают проблемы с установкой и запуском таковой. Для удобства и упрощения запуска был использован программная платформа для быстрой разработки, тестирования и развертывания приложений Docker. С её помощью можно быстро, а что самое главное просто запустить таковую. В дальнейшем, при запуске системы на сервере он может не понадобится.

Сам по себе Докер, это виртуальная машина. И нужно рассматривать его именно как полноценную виртуальную машину, в которой есть свой жёсткий диск, оперативная память и сетевая карта. Если вы работали с виртуальными машинами, например, VirtualBox, то многие вещи будут уже знакомы, правда, с той разницей, что докер не имеет красивую GUI-оболочку, а управляется в основном из консоли.

Основные компоненты докера:

- образ (image) - образ диска, который будет запускаться в контейнере;
- контейнер (container) - виртуальная машина, в которой запущен образ;
- репозиторий образов (image repository) - удалённое или локальное хранилище для образов.

Нужно помнить, что в контейнере всегда запускается образ. При этом, если докер не нашёл образ в локальном хранилище, то он будет искать его в удалённом репозитории, как правило, на Dockerhub.

Для установки докера на Windows требуется установить подсистему Windows для Linux, в дальнейшем WSL, позволяющую разработчикам устанавливать

дистрибутив Linux, например, Ubuntu, OpenSUSE, Kali, Debian, Arch Linux, и использовать приложения Linux, служебные программы и программы командной строки Bash непосредственно в Windows без изменений, без дополнительных затрат на традиционную виртуальную машину или двойную установку. Что бы это сделать нужно выполнить данную команду (см. рис. 6) в командной строке, э та команда

```
PS C:\Users\User> wsl --install
```

включит функции, необходимые для запуска WSL и установки дистрибутива Ubuntu для Linux.

Рис. 6. Команда для установки WSL

После установки WSL нужно скачать сам установщик Докера с сайта Docker Docs во вкладке Manuals, переходя в навигации по пути Docker Desktop → Install Docker Desktop → Install on Windows (см. рис. 7).



Рис. 7. Путь в навигации сайта для скачивания Докера для Windows

После установки нужно в папке проекта создать файл docker-compose.yml, добавив в данный файл скрипт, взятый с Docker Hub из описания официального образа MySQL, при этом нужно его настроить под свои нужды, указать пароль и логин для главного пользователя (см. рис. 8). Так же в примере скрипта с сайта имеется веб-приложение, написанное на PHP для администрирования баз данных, благодаря этой части можно выполнить большое количество действий, например, добавить пользователя, создать таблицу, добавить запись, удалить, обновить.

```
# Use root/example as user/password credentials
version: '3.1'

services:
  db:
    image: mysql
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: password12345
    ports:
      - 3306:3306

  adminer:
    image: adminer
    restart: always
    ports:
      - 8081:8080
```

Рис. 8. Пример скрипта в docker-compose.yml

Важным моментом является поле портов. По умолчанию контейнер закрыт от любых контактов извне. Вы не можете ни скопировать в него файл, ни подключиться к сокету. Он закрыт. Но при запуске контейнера можно пробросить порт. Первый порт относится к локальной машине, второй порт контейнера, получается, что докер связывает локальный и виртуальные порты.

Для установки образов и запуска требуется написать в консоль команду docker-compose up.

2.1.2. Организация структуры базы данных

Основываясь на требованиях и идеях, описанных ранее, нужно составить структуру базы данных, которая планируется использоваться для хранения в ней данных системы. При это для оптимизации хранения, стоит придерживаться нормальных форм с самого начала, для минимизации затрат на дальнейшую работу. 3 форма является наиболее используемой, по данной причине выбор пал на неё.

В первую очередь нужно создать таблицы, которые не имеют зависимость от других. Таковыми таблицами являются:

- аккаунты пользователей, с полями: идентификационный уникальный ключ, логин для входа в аккаунт, пароль и дату создания;

- каналов или же чатов, в которой требуется id, название и описание;
- логирование запросов пришедших на сервер.

Далее создать таблицы, которые зависят от выше стоящих:

- наличия каналов у пользователя, которая является прослойкой для связи многих ко многим, как ясно по названию, между таблицей каналов и аккаунтов, опирается на id обеих;
- сообщений, которая является чуть ли не самой главной таблицей, с которой будет производиться больше всего операций, в ней должны быть ссылки на пользователя и канал, которому принадлежит, время отправки и изменения, сам текст сообщения.

Так как планируется дать возможность пользователям изменять свои сообщения, нужна таблица, которая хранит предыдущие состояния сообщений. Отличным решением будет добавить триггер после обновления записи сообщения, чтобы сохранять старое состояние используя только СУБД, что уменьшит количество работы приложения.

Важным моментом будет создание пользователя СУБД, который имеет все права, но только для данной базы данных, он будет использоваться в серверном приложении для взаимодействия.

После создания структуры базы данных (см. рис. 9), её требуется представить в языке SQL (приложение 1), чтобы выполнить через веб-приложение администрирования данный код, для переноса её в СУБД.

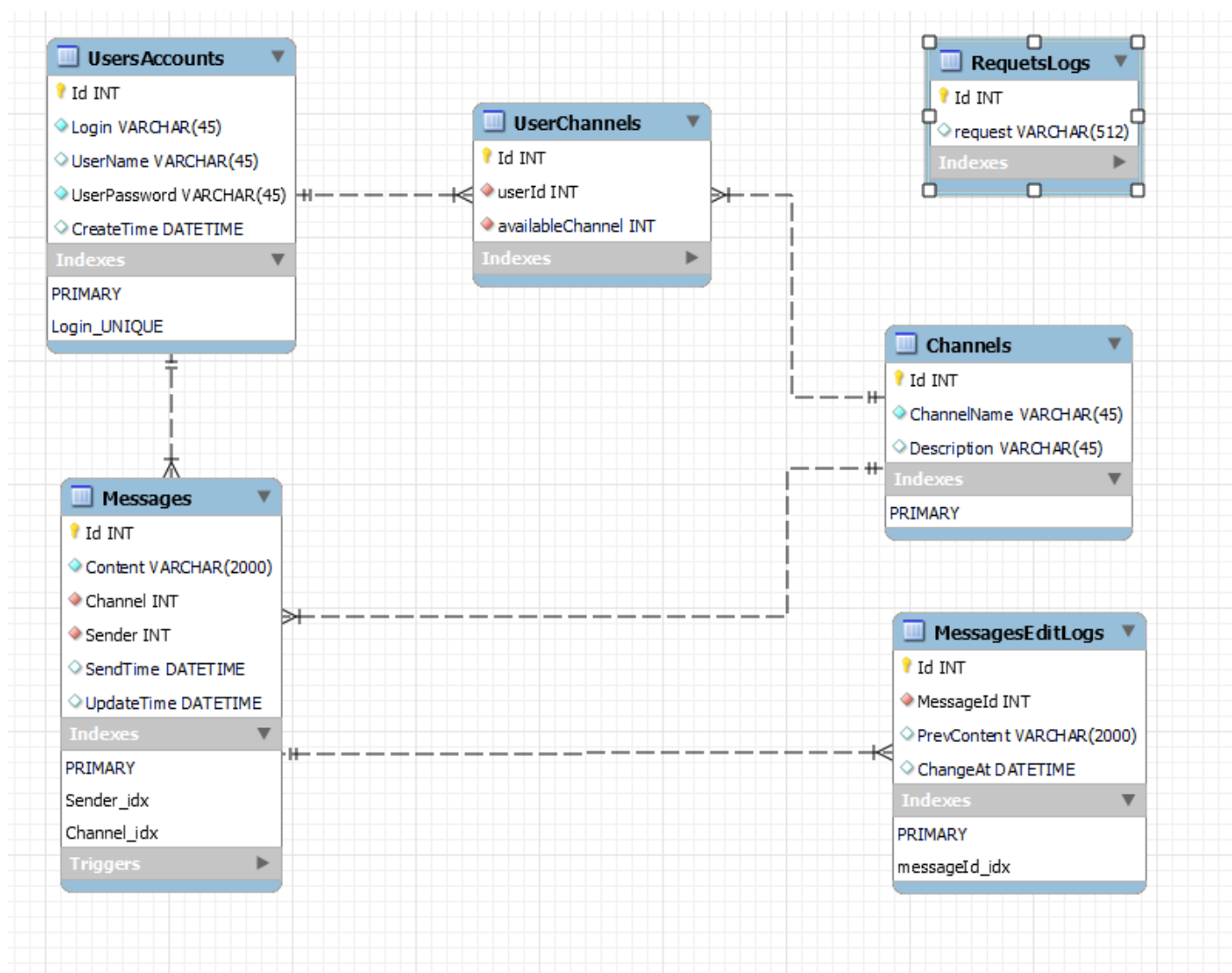


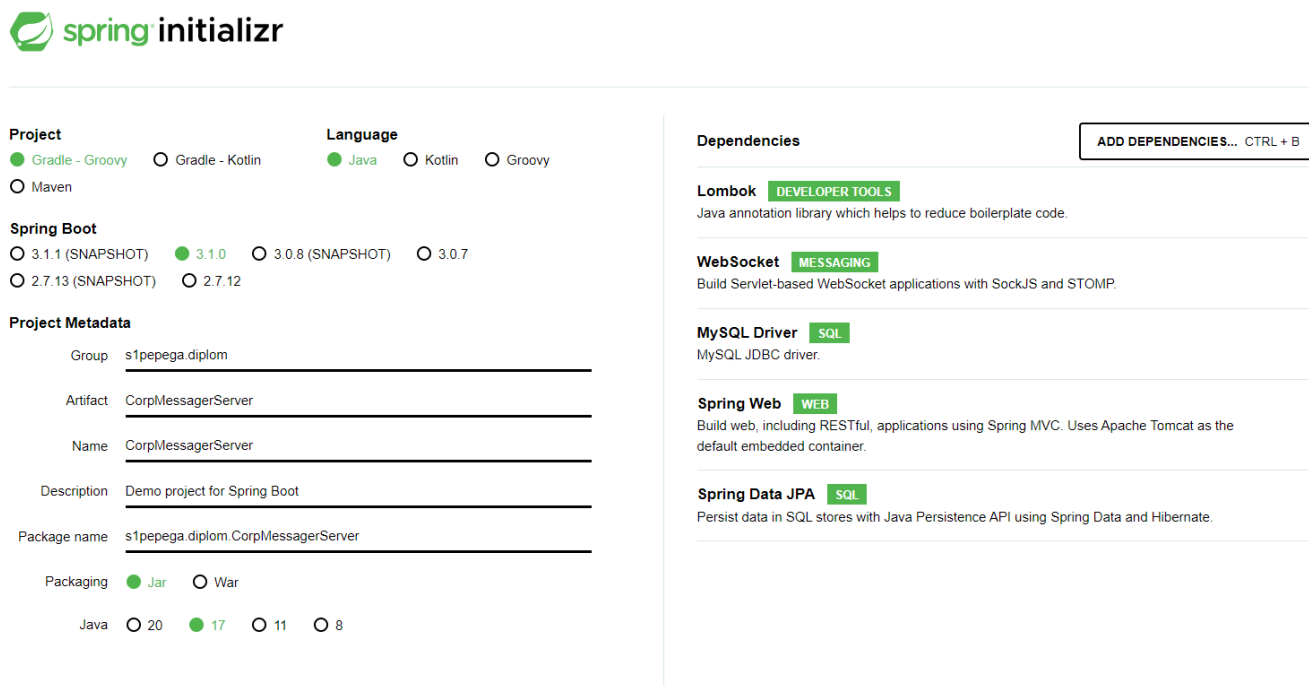
Рис. 9. EER диаграмма базы данных

Результатом будет полностью рабочая БД, с которой уже можно взаимодействовать с помощью панели администрирования, половина серверной стороны системы готова. В дальнейшем данную базу можно будет модифицировать, в качестве примера можно взять, ранее указанную идею, систему событий, указанных в календаре.

2.2. Spring Boot приложение

2.2.1. Инициализация и настройка зависимостей проекта

Для создания приложения требуется перейти на сайт инициализатор проекта (см. рис. 10), который можно найти в документации Spring Boot.



The screenshot shows the Spring Initializr web interface. It is divided into several sections for configuring a new Spring Boot project:

- Project:** Includes radio buttons for **Gradle - Groovy** (selected), **Gradle - Kotlin**, and **Maven**.
- Language:** Includes radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for versions: **3.1.1 (SNAPSHOT)**, **3.1.0** (selected), **3.0.8 (SNAPSHOT)**, **3.0.7**, **2.7.13 (SNAPSHOT)**, and **2.7.12**.
- Project Metadata:** A form with fields for:
 - Group:** s1pepega.diplom
 - Artifact:** CorpMessengerServer
 - Name:** CorpMessengerServer
 - Description:** Demo project for Spring Boot
 - Package name:** s1pepega.diplom.CorpMessengerServer
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Java:** Includes radio buttons for versions: **20**, **17** (selected), **11**, and **8**.
- Dependencies:** A section on the right with a button **ADD DEPENDENCIES... CTRL + B**. It lists several dependencies with their categories in green boxes:
 - Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
 - WebSocket** (MESSAGING): Build Servlet-based WebSocket applications with SockJS and STOMP.
 - MySQL Driver** (SQL): MySQL JDBC driver.
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Рис. 10. Конфигурация проекта для инициализации проекта

Стоит немного рассказать о системе сборки приложения, в данном случае используется Gradle.

Gradle - система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но предоставляющая DSL на языках Groovy и Kotlin вместо традиционной XML-образной формы представления конфигурации проекта. В отличие от Apache Maven, основанного на концепции жизненного цикла проекта, и Apache Ant, в котором порядок выполнения задач (targets) определяется отношениями зависимости (depends-on), Gradle использует направленный ациклический граф для определения порядка выполнения задач. Gradle был разработан для расширяемых много проектных сборок, и поддерживает каскадную (waterfall) модель разработки,

определяя, какие компоненты дерева сборки не изменились и какие задачи, зависящие от этих частей, не требуют перезапуска.

После генерации разархивируем полученный архив и открываем проект с помощью среды разработки, в текущем случае это IntelliJ IDEA, в которой будет удобно ориентироваться в проекте. Так же требуется добавить ещё несколько зависимостей в build.gradle сборки (см. рис. 11) для данного приложения, в дальнейшем они сильно упростят разработку.

```
1 plugins {
2     id 'java'
3     id 'org.springframework.boot' version '3.0.4'
4     id 'io.spring.dependency-management' version '1.1.0'
5 }
6
7 group = 'sipepega.diplom'
8 version = '0.0.1-SNAPSHOT'
9 sourceCompatibility = '17'
10
11 repositories {
12     mavenCentral()
13 }
14
15 dependencies {
16     implementation 'org.springframework.boot:spring-boot-starter'
17     implementation "org.springframework.boot:spring-boot-starter-web"
18     implementation 'org.springframework.boot:spring-boot-starter-websocket'
19     implementation "org.springframework.boot:spring-boot-starter-data-jpa"
20
21     implementation "org.apache.commons:commons-lang3"
22
23     compileOnly "org.projectlombok:lombok"
24     annotationProcessor "org.projectlombok:lombok"
25
26     //sql
27     runtimeOnly 'com.mysql:mysql-connector-j'
28
29     //websocket
30     implementation 'org.webjars:webjars-locator-core'
31     implementation 'org.webjars:stomp-websocket:2.3.3'
32
33     //test
34     testImplementation 'org.springframework.boot:spring-boot-starter-test'
35 }
36
37 tasks.named('test') {
38     useJUnitPlatform()
39 }
```

Рис. 11. Содержание файла build.gradle, точками помечены добавленные зависимости

2.2.2. Подключение приложения к базе данных

Для подключения к базе данных требуется создать файл `application.property` (см. рис. 12), в котором будут указаны URL адрес по которому будет происходить подключение к базе данных, логин и пароль пользователя СУБД, от лица которого будет происходить взаимодействие с таковой, и драйвер класса, информацию о том какой драйвер использовать с какой базой можно узнать в документации Spring.

```
spring.datasource.url=jdbc:mysql://localhost:3306/mymessenger
spring.datasource.username=serverUser
spring.datasource.password=s1pepega
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Рис. 12. Конфигурация `application.property`

В случае с MySQL это `com.mysql.cj.jdbc.Driver` из библиотеки `com.mysql:mysql-connector-j`. Важно отметить, данные из `application.property` можно будет в дальнейшем изменить при запуске приложения, это можно сделать с помощью переменной среды `SPRING_APPLICATION_JSON`, либо аргумента `spring.application.json`, в обоих случаях требуется подать json с соответствующей конфигурацией. Допустим перед запуском с помощью команды, без изменения `env`, нужно поставить значению параметру `app.exception.message` для этого нужен JSON {“`app.exception.message`”:”value”} (см. рис. 13).

```
java -jar .\path\SpringApp.jar --spring.application.json='{\"app.exception.message\": \"value\"}'
```

Рис. 13. Пример команды для запуска

2.2.3. Сущности и репозитории

Сущность — это представление записи в виде объекта класса.

Для каждой таблицы требуется создать сущности, кроме логов сообщений. Из-за того, что сущности не имеют огромной разницы в структуре, то разобрав один тип, понимание оставшихся придет, само собой. В качестве примера можно взять сущность сообщений (см. рис. 14).

```
@Getter
@Setter
@Accessors(chain = true)
@Entity
@Table(name = "messages")
@JsonIgnoreProperties({"hibernateLazyInitializer"})
public class Message {
    4 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    4 usages
    @Column(name = "content", nullable = false)
    private String content;
    4 usages
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "channel", nullable = false, foreignKey = @ForeignKey(name = "fk_message_channel_id"))
    private Channel channel;
    4 usages
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name="sender", nullable = false, foreignKey = @ForeignKey(name = "fk_message_sender_id"))
    private User sender;
    4 usages
    @JsonFormat(shape = JsonFormat.Shape.STRING)
    @Column(name = "sendtime")
    private Date sendTime;
    4 usages
    @JsonFormat(shape = JsonFormat.Shape.STRING)
    @Column(name = "updatetime")
    private Date updateTime;
```

Рис. 14. Класс сущности сообщения

Данный класс имеет большое количество аннотаций на себе и своих полях, самыми главными являются Table и Entity, первая в передаваемом параметре имеет наименование таблицы, это используется для нахождения таковой в подключённой СУБД и связывает это с сущность, вторая в свою очередь указывает что данный класс является сущностью, так же её параметр имени, по стандарту совпадающим с наименованием класса, используется в запросах, которыми оперируют репозитории.

Так же важными аннотациями являются Getter и Setter, принадлежащие lombok, они добавляют соответствующие методы для полей класса. Так же к lombok

относиться Accessors, параметр chain превращает указывает на то нужно ли сеттеру вернуть объект, над которым производились действия, или нет.

У полей так же большое количество аннотаций, самой главной является Column, она указывает на столбец, его имя, может ли не быть значения, может быть уникальное ли поле и иные характеристики, в котором находится данная информация. Так же есть аннотация Id, которая относит переменную к Id столбцу в таблице, если обратить внимание, то у поля id имеется и другая аннотация GeneratedValue, она требуется если при запуске приложения не существует таблицы, указывает параметры генерации для аннотации Id.

В свою очередь JoinColumn выполняет роль аннотации Column и получает данные из другой таблице с помощью вторичного ключа. В свою очередьManyToOne показывает, как связана данный тип сущностей и поле, так же в нем указывается стратегия загрузки, ленивая или же жажущий. Вторым вариантом подгружает данные сразу же как загрузилась сущность в репозиторий, первый наоборот.

Так же стоит обратить внимание на аннотацию JsonFormat, с её помощью можно указать в каком формате будут представлены данные, когда сущность будет конвертирована в JSON для отправки или каким образом требуется читать и конвертировать при принятии запроса или же сообщения. Так же в сущности пользователя у поля пароля имеется аннотация @JsonProperty(access = JsonProperty.Access.WRITE_ONLY), которая не дает отправить пароль в ответе, что защищает аккаунт.

Репозиторий – это несколько интерфейсов которые используют JPA Entity для взаимодействия с ней. Так например интерфейс public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> обеспечивает основные операции по поиску, сохранения, удалению данных (CRUD операции).

Для создания репозитория требуется создать интерфейс и наследовать его от вышеуказанного интерфейса и добавить аннотацию Repository. Но иногда могут потребоваться специфические запросы в базу. Для этого требуется добавить интерфейс метод с аннотацией Query, в которую уже можно написать запрос, который будет производиться для выборки данных. Данные запросы могут иметь параметры, в методе требуется указать аргумент и аннотацию Param для него, при этом указав строку, под которой подразумевается данный параметр в запросе, в свою очередь для того что бы запрос знал этот параметр в нем так же требуется указать тоже название, при этом поставив перед ним двоеточие. В качестве примера репозитория и методов со специфическими запросами можно привести репозиторий сообщений (см. рис. 15).

```
@Repository
public interface MessageRepository extends JpaRepository<Message, Integer> {

    1 usage  ↗ S1dechko
    @Query(value = "SELECT * FROM messages m WHERE m.channel = :channelId LIMIT :skip, :count", nativeQuery = true)
    List<Message> getMessageInChannelWithStartAndCount(
        @Param("channelId")Integer channelId,
        @Param("skip")Integer skip,
        @Param("count")Integer count);

    1 usage  ↗ S1dechko
    @Query(value = "SELECT * FROM messages m WHERE m.channel = :channelId", nativeQuery = true)
    List<Message> getMessageAllInChannel(
        @Param("channelId")Integer channelId);
}
```

Рис. 15. Репозиторий сообщений

2.2.4. Сервисы и их имплементации

После создания всех связей с базой данных, требуется реализовать обработчики, сервисы, которые будут являться прослойкой между действиями пользователя и данными на сервере. Всего таких сервисов должно быть 5:

- сервис, обрабатывающий действия с сообщениями;
- сервис, обрабатывающий действия с каналами;
- сервис, обрабатывающий действия с пользователем;
- сервис, связывающий каналы и пользователей;
- сервис сессий.

Стоит выделить общие положения сервисов, практически у всех сервисов есть методы удаления, обновления, создания, получения по id и получения всех записей. Они мало отличаются друг от друга, иногда название, зачастую меняется только репозиторий к которому происходит запрос и небольшая часть логики.

Так же стоит ответить на вопрос, почему у сервисов есть интерфейсы и реализации. Интерфейсы нужны для возможности замены реализации, не затрагивая основной код. При использовании интерфейсов основной код ничего не знает об деталях реализации (слабая связанность). Соответственно, реализацию можно вынести в отдельный модуль (изолировать сложность). Если основной код ссылается непосредственно на класс, содержащий реализацию (сильная связанность), то сложность программы возрастает и сопровождение программы усложняется. Поэтому интерфейсы особенно нужны на границах подсистем.

Начать стоит с сервиса, обрабатывающего пользователей (см. рис 16), ведь он используется в первую очередь при взаимодействии с системой.

```

@Service("UserServiceImpl")
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

    8 usages
    @Autowired
    private UserRepository userRepository;

    S1dechko
    @Override
    @Transactional(readOnly = true)
    public List<User> findAll() { return new ArrayList<>(userRepository.findAll()); }

    2 usages S1dechko
    @Override
    @Transactional(readOnly = true)
    public User findById(Integer id) {
        return userRepository.findById(id)
            .orElseThrow(()->new EntityNotFoundException("User with id "+id+" not found"));
    }

    1 usage S1dechko
    @Override
    @Transactional(readOnly = true)
    public User findByName(String name) {
        return userRepository.findByName(name)
            .orElseThrow(()->new EntityNotFoundException("User with login "+name+" not found"));
    }

    1 usage S1dechko
    @Override
    @Transactional(readOnly = true)
    public User login(User userRequest) {
        User user = userRepository.findByName(userRequest.getLogin())
            .orElseThrow(()->new EntityNotFoundException("User with login "+userRequest.getLogin()+" not found"));
        if(!user.getPassword().equals(userRequest.getPassword()))
            throw new EntityNotFoundException("Illegal password. Try again");
        return user;
    }
}

```

Рис. 16. Часть сервиса, обрабатывающего пользователей

На примере данной имплементации можно увидеть основные черты всех сервисов, кроме сервиса сессий, его требуется рассмотреть отдельно.

Каждая реализация сервиса помечается аннотацией `Service` с указанием названия, так же сервис наследует интерфейс данного типа сервисов, причина сего решения была указана выше. Так же обратив внимание на единственное поле можно увидеть репозиторий пользователей с аннотацией `Autowired`, которая указывает фреймворку установить на данное место некий бин, по причине того, что нет других бинов такого типа, установится единственно верный. Конечно же подобное поле может быть не одиноким в классе, такое появляется и в других сервисах. По аннотации `Override`

можно увидеть, что все методы прописаны в интерфейсе, и реализуются в данном классе.

В свою очередь сервис сессий является особенным (см. рис 17), он имеет в себе статичную `HashMap<Integer, Integer>` которая хранит как пару ключ-значение номер сессии и `id` пользователя соответственно. Это в дальнейшем понадобится для проверки, может ли пользователь данной сессии выполнить действие, которое он пытается сделать.

```
@Service("SessionServiceImpl")
public class SessionServiceImpl implements SessionService {

    7 usages
    private static final HashMap<Integer, Integer> sessions = new HashMap<>();

    1 S1dechko
    @Override
    public Integer getUserIdAsSession(Integer sessionId) {
        if(!sessions.containsKey(sessionId))
            throw new RuntimeException("unknown session, user not found");
        return sessions.get(sessionId);
    }

    2 usages 1 S1dechko
    @Override
    public void appendNewSession(Integer sessionId, Integer userId) { sessions.putIfAbsent(sessionId,userId); }

    1 usage 1 S1dechko
    @Override
    public Boolean canSendRequest(Integer sessionId, Integer userId) {
        if(userId == null)
            throw new RuntimeException("userId not present");
        if(!sessions.containsKey(sessionId))
            throw new RuntimeException("unknown session, send request canceled");
        return sessions.get(sessionId).equals(userId);
    }

    1 S1dechko
    @Override
    public void closeSession(Integer sessionId) { sessions.remove(sessionId); }

    2 usages 1 S1dechko
    @Override
    public Integer getNewSessionId() { return (int)System.currentTimeMillis(); }
```

Рис. 17. Сервис сессий

Результатом текущей работы будет следующая схема, приложение подключается к СУБД, взаимодействует с таблицами с помощью репозиторийев, в свою очередь, сервисы берут взаимодействуют с сущностями в репозиториях, если в этот момент возникает какая-либо ошибка, метод, обрабатывающий действие, создается экшпешен, которая должна будет обработаться далее.

2.2.5. Обработка HTTP запросов и их ошибок

Началом взаимодействия с серверной частью системы является вход в систему, также нужно узнать какие каналы доступны пользователю, все эти действия носят разовый характер. Для данных действий не требуется открывать WebSocket соединения, хватит обычных HTTP запросов. Это является экономией ресурсов как пользователя, так и сервера. Таких образом можно разделить на два контроллера UserControllerREST и ChannelControllerREST.

По причине того, что оба контроллера имеют одинаковую структуру, рассмотреть можно один, пускай это будет UserControllerREST (см. рис. 18).

```
@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserControllerREST {

    8 usages:
    @Autowired
    private UserService userService;
    1 usage:
    @Autowired
    private UserChannelService ucService;
    5 usages:
    @Autowired
    private SessionService sessionService;

    1 Sidekick
    @GetMapping(produces = APPLICATION_JSON_VALUE)
    public List<User> findAll() {return userService.findAll();}

    1 Sidekick
    @GetMapping(value = "/{userId}", produces = APPLICATION_JSON_VALUE)
    public User findById(@PathVariable Integer userId, @RequestHeader("sessionId") Integer sessionId) {
        return userService.findById(userId);
    }

    1 Sidekick
    @GetMapping(value = "/{userName}", produces = APPLICATION_JSON_VALUE)
    public User findByName(@PathVariable String userName, @RequestHeader("sessionId") Integer sessionId) {
        return userService.findByName(userName);
    }

    1 Sidekick
    @GetMapping(value = "/{userId}/getChannels", produces = APPLICATION_JSON_VALUE)
    public List<Channel> getChannels(@PathVariable Integer userId, @RequestHeader("sessionId") Integer sessionId) {
        // checkSession(userId, sessionId);
        return ucService.getUserChannels(userId);
    }

    1 Sidekick
    @PostMapping(value = "/create", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public LoginResponse create(@RequestBody User request) {
        User newUser = userService.createUser(request);
        Integer sessionId = sessionService.getNewSessionId();
        sessionService.appendNewSession(sessionId, newUser.getId());
        return new LoginResponse().setUser(newUser).setSessionId(sessionId);
    }

    1 Sidekick
    @PostMapping(value = "/login", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public LoginResponse login(@RequestBody User request) {
        User user = userService.login(request);
        Integer sessionId = sessionService.getNewSessionId();
        sessionService.appendNewSession(sessionId, user.getId());
        return new LoginResponse().setUser(user).setSessionId(sessionId);
    }

    1 Sidekick
    @PatchMapping(value = "/update", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public User update(@RequestBody User request, @RequestHeader("sessionId") Integer sessionId) {
        checkSession(request.getId(), sessionId);
        return userService.update(request);
    }

    1 Sidekick
    @DeleteMapping(value = "/delete/{id}", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public User delete(@PathVariable Integer id, @RequestHeader("sessionId") Integer sessionId) {
        checkSession(id, sessionId);
        User user = userService.findById(id);
        userService.delete(id);
        return user;
    }

    2 usages: 1 Sidekick
    private void checkSession(Integer userId, Integer sessionId){
        if(!sessionService.canSendRequest(sessionId, userId))
            throw new IllegalSessionIdException("access denied");
    }
}
```

Рис. 18. Класс UserControllerREST

Как и у ранее продемонстрированных классов данный пример не лишен аннотаций, указывающих на его принадлежность к определенным механизмам фреймворка. В данном случае у нас имеются аннотации RestController и RequestMapping("/api/users"), второй указывает на то с какого URL начинаются все эндпоинты, то есть все запросы будут выглядеть http://IP/api/users/*, под звездочкой подразумевается дальнейший путь к запросам.

Первая в свою очередь удобная аннотация, которая сама по себе аннотируется с помощью Controller и ResponseBody. Типы, которые содержат эту аннотацию, рассматриваются как контроллеры, где методы RequestMapping по умолчанию принимают семантику ResponseBody.

Эндпоинты описываются методами данного класса. Например, методы login и update (см. рис. 19) имеющие аннотацию маппинга, первый для POST запроса, второй для PATCH, value в котором означает продолжение ссылки, consumes указывает какого формата должно быть тело запроса, produces тело ответа.

```

@S1dechko
@PostMapping(value = "/login", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
public LoginResponse login(@RequestBody User request) {
    User user = userService.login(request);
    Integer sessionId = sessionService.getNewSessionId();
    sessionService.appendNewSession(sessionId, user.getId());
    return new LoginResponse().setUser(user).setSessionId(sessionId);
}

@S1dechko
@PatchMapping(value = "/update", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
public User update(@RequestBody User request, @RequestHeader("sessionId") Integer sessionId) {
    checkSession(request.getId(), sessionId);
    return userService.update(request);
}
```

Рис. 19. Методы-эндпоинты в REST контроллере

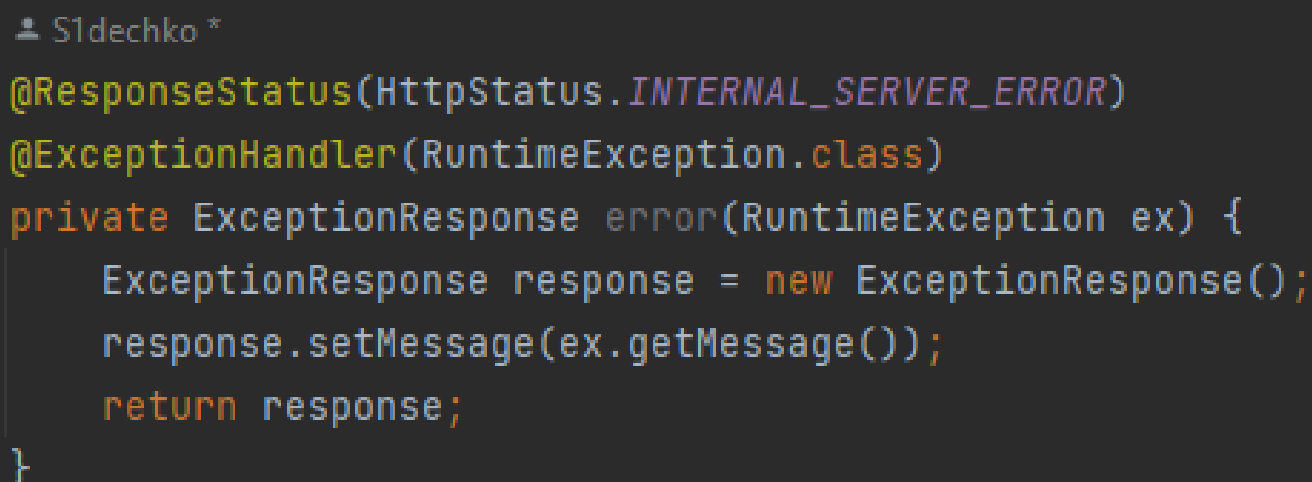
Во время выполнения запроса может произойти сбой, возможно неверно указанная ссылка или тело запроса не верно или же не указан id сессии. Требуется обрабатывать такие запросы и возвращать ошибку в виде понятного сообщения, которое может прочитать человек и понять, что произошло.

Для данной цели требуется создать новый обработчик с аннотацией RestControllerAdvice, предназначенный для написания общего кода обработки исключений, то есть, не требуется в каждый контроллер добавлять методы обработки.

В нем требуется создать несколько обработчиков для исключений:

- `EntityNotFoundException`, с кодом 404;
- `WIPException`, с кодом 501;
- `IllegalSessionIdException`, с кодом 401;
- `RuntimeException`, с кодом 500.

Последний требуется для непреднамеренных ошибок, которые могут возникнуть в любой момент. Разница обработчиков, только в том какой они код выдают и на какую ошибку вызываться, даже структура тела ответа будет идентичная (см. рис. 20). В аннотации `ResponseStatus` указывается статус HTTP, который вернется с ответом, в свою очередь `ExceptionHandler` класс ошибки, которая будет обработана.



```

S1dechko *
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
@ExceptionHandler(RuntimeException.class)
private ExceptionResponse error(RuntimeException ex) {
    ExceptionResponse response = new ExceptionResponse();
    response.setMessage(ex.getMessage());
    return response;
}

```

Рис. 20. Пример обработчика ошибок HTTP запросов

2.2.6. WebSocket соединение

Как и предыдущим типом соединения Spring позволяет организовать WS соединение достаточно простым образом, для этого потребуется только 2 класса, конфигурации, в котором будут настроена, конвертация сообщений и пути URL откуда и куда будут отсылаться сообщения, и регистрация эндпоинта (см. рис. 21).

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    1 usage  S1dechko
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker( ...destinationPrefixes: "/state");
        config.setApplicationDestinationPrefixes("/app");
    }

    1 usage  S1dechko
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint( ...paths: "/ws").withSockJS();
    }

    S1dechko
    @Override
    public boolean configureMessageConverters(List<MessageConverter> messageConverters) {
        DefaultContentTypeResolver resolver = new DefaultContentTypeResolver();
        resolver.setDefaultMimeType(MimeTypeUtils.APPLICATION_JSON);
        MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
        converter.setObjectMapper(new ObjectMapper());
        converter.setContentTypeResolver(resolver);
        messageConverters.add(converter);
        return false;
    }
}
```

Рис. 21. Класс контроллера

Сообщения, передаваемые данным способом, работают через STOMP протокол. Это надстройка над вебсокетом, который нормирует передачу данных, ведь WS не обязывает представлять данные в типе. Это упрощит задачу, к тому же фреймворк, обязывает использовать данный протокол, при данном методе организации.

Клиент и сервер будут взаимодействовать, используя кадры STOMP, отправляемые по потоку. Структура фрейма сторон одинакова (см. рис. 22), так же

важным аспектом этих кадров, является команда, указываемая в самой первой части фрейма, некоторые требуют обязательны заголовок, например, путь (см. рис. 22).

```
COMMAND
header1:value1
header2:value2

Body^@
```

Рис. 22. Структура сообщения STOMP

Для создания и отправки сообщений, требуется контроллер, по виду он мало отличим от HTTP контроллера, но по логике аннотаций значительно. Это и есть второй класс (см. рис. 23).

```
@Controller
public class EverythingWebSocketController {
    4 usages
    @Autowired
    private UserChannelService ucService;
    1 usage
    @Autowired
    private ChannelService channelService;
    3 usages
    @Autowired
    private MessageService messageService;
    2 usages
    @Autowired
    private SessionService sessionService;|
    1 usage
    @Autowired
    private SimpMessagingTemplate simpMessagingTemplate;

    //USER
    ▲ SIdchko
    @MessageMapping("/add_user_to_channel/{userId}")
    @SendTo("/state/user/{userId}/append_to_channel")
    public UserChannelLink addUserToChannel(
        @DestinationVariable Integer userId,
        @Payload UserChannelLink link,
        @Header("sessionId") Integer sessionId
    ){
        return ucService.create(link);
    }
}
```

Рис. 23. Начало класса контроллера и пример обработчика.

Метод обрабатывающий сообщения имеет две аннотации, в первой, `MessageMapping`, указывается с какого url обрабатывать сообщения данным образом, вторая, `SendTo`, куда отправлять результат ответ. Так же у аргументов тоже указываются аннотации, которые используются для модификации пути отправки, `DestinationVariable`, в качестве тела, `Payload`, который обрабатывается ранее указанным конвертором в конфигурации и дополнительные заголовки, которые будут считываться.

Так же некоторые сообщения могут вызывать преднамеренное или случайное исключение, которое требуется обработать. Для этого требуется в контроллер добавить метод (см. рис. 24), который будет выполнять данную работу. Так же придётся использовать `SimpMessagingTemplate`, чтобы иметь возможность отправить конкретному человеку сообщение.

```

@ExceptionHandler
public void handleException(Throwable exception){
    if(exception instanceof IllegalSessionIdException){
        try{
            Integer userId = sessionService.getUserIdAsSession(((IllegalSessionIdException) exception).sessionId);
            simpMessagingTemplate.convertAndSend("destination: "/state/user/"+userId+"/exception", exception.getMessage());
        }catch (Exception ignored){}
    }
}

```

Рис. 24. Обработчик исключений контроллера, имеющий обратную связь

Для проверки данного функционала требуется воспользоваться либо своей разработкой, которая будет взаимодействовать с соединением, либо postman, относительно недавно у него появилась возможность открыть соединение такого типа.

2.2.7. Итоговая структура системы, сторона сервера

Результатом разработки серверного приложения является большая часть системы (см. рис. 25), ведь именно в ней находится вся бизнес логика, благодаря таковой и будет происходить все волшебство мгновенной передачи сообщений.

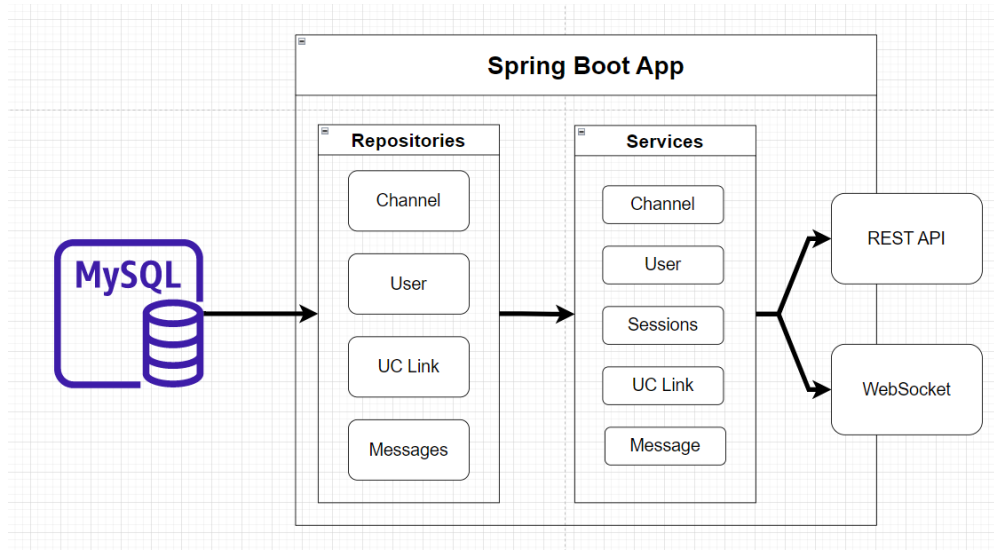


Рис. 25. Схема серверной стороны системы

2.3. Клиентское приложение

2.3.1. Организация сетевых соединений с сервером

Для одной из частей приложения был использована библиотека System.Net.Http за авторством Microsoft, предоставляющей компоненты, которые позволят использовать веб-службы через HTTP. Также важным для создания HttpClient, предоставляющий класс для отправки HTTP-запросов и получения HTTP-ответов от ресурса, определяемого URI, информацию из которых в дальнейшем приложение будет обрабатывать. Для более удобной работы в дальнейшем для данного типа взаимодействия стоит создать отдельный класс, в котором будут URI, номер сессии и сам клиент, а также методы, для формирования запроса и записи требуемых заголовков, для отправки и считывания сообщения, после того как сервер вернул ответ (см. рис. 26).

```

public class RESTClient
{
    private string? url;

    private HttpClient? Client;

    private int? sessionId;

    СОМАРК 1
    public RESTClient Initialize()
    {
        if (url is null)
            throw new MessengerHardClientException("url is null.");
        if (Client is not null)
            Client.Dispose();

        Client = new HttpClient();

        return this;
    }

    СОМАРК 1
    public RESTClient SetURL(string url)
    {
        this.url = url;
        return this;
    }

    СОМАРК 2
    public RESTClient SetSessionId(int sessionId)
    {
        this.sessionId = sessionId;
        return this;
    }

    СОМАРК 3
    private HttpRequestMessage buildMessage(HttpMethod method, string subUrl, string? content)
    {
        HttpRequestMessage message = new HttpRequestMessage()
        {
            Method = method,
            RequestUri = new Uri(this.url + subUrl)
        };

        if (this.sessionId is not null)
            message.Headers.Add("sessionId", this.sessionId.ToString());

        if (content is not null)
            message.Content = new StringContent(content, Encoding.UTF8, "application/json");

        return message;
    }

    СОМАРК 5
    public string SendAndResiveRequest(HttpMethod method, string subUrl, string? content = null)
    {
        if (Client is null)
            throw new MessengerHardClientException("HTTP client initialized.");
        Exception? ex = null;
        var response = Client.SendAsync(buildMessage(method, subUrl, content)).ContinueWith((responseTask) =>
        {
            string result = "";
            try
            {
                result = responseTask.Result.Content.ReadAsStringAsync().Result;
            }
            catch (Exception _ex) { ex = _ex; }
            return result;
        }).Result;
        if (ex is not null)
            throw ex;
        return response;
    }
}

```

Рис. 26. Класс и его методы

В свою очередь для WS клиент требует затрату больших сил на свое создание, для решение данной проблемы можно воспользоваться готовым решением. В данном случае используется библиотека websocketsharp.core. Помимо самого клиента, принимающего и отправляющего запросы, требуется ещё десериализатор сообщений, который будет преобразовывать STOMP сообщения в объекты, для удобства взаимодействия в других частях приложения. Итогом будут классы, один непосредственно сериализатор (см. рис. 27), другой надстройка над клиентом, где хранятся номер сессии, URI, сам клиент и обработчик принятых сообщений, а также удобные методы, для работы, подписки, отписки, отправки сообщения (см. рис. 28).

```

Source 2
public class StompMessageSerializer
{
    Source 4
    public string Serialize(StompMessage message)
    {
        var buffer = new StringBuilder();
        buffer.Append(message.Command + "\n");
        if (message.Headers != null)
        {
            foreach (var header in message.Headers)
            {
                buffer.Append(header.Key + ":" + header.Value + "\n");
            }
        }
        buffer.Append("\n");
        buffer.Append(message.Body);
        buffer.Append("\0");
        return buffer.ToString();
    }

    Source 1
    public StompMessage Deserialize(string message)
    {
        var reader = new StringReader(message);
        var command = reader.ReadLine();
        var headers = new Dictionary<string, string>();
        var header = reader.ReadLine();
        while (!string.IsNullOrEmpty(header))
        {
            var split = header.Split(':');
            if (split.Length == 2) headers[split[0].Trim()] = split[1].Trim();
            header = reader.ReadLine() ?? string.Empty;
        }
        var body = reader.ReadToEnd() ?? string.Empty;
        body = body.TrimEnd('\r', '\n', '\0');
        return new StompMessage(command, body, headers);
    }
}

Source 14
public class StompMessage
{
    private readonly Dictionary<string, string> _headers = new Dictionary<string, string>();

    Source 3
    public StompMessage(string command)
    {
        : this(command, string.Empty)
    }

    Source 2
    public StompMessage(string command, string body)
    {
        : this(command, body, new Dictionary<string, string>())
    }

    Source 2
    internal StompMessage(string command, string body, Dictionary<string, string> headers)
    {
        Command = command;
        Body = body;
        _headers = headers;
        this["content-length"] = body.Length.ToString();
    }

    Source 2
    public Dictionary<string, string> Headers
    {
        get { return _headers; }
    }

    Source 3
    public string Body { get; private set; }

    Source 3
    public string Command { get; private set; }

    Source 11
    public string this[string header]
    {
        get { return _headers.ContainsKey(header) ? _headers[header] : string.Empty; }
        set { _headers[header] = value; }
    }
}

```

Рис. 27. Сериализатор и объект STOMP сообщения

```

Source 6
public class WSClient
{
    private WebSocket? Client;
    private int? sessionId;
    private string? url;
    private static readonly StompMessageSerializer serializer = new();
    private IMessageHandler? messageHandler;
    public bool ConnectedToServer = false;

    Source 1
    public WSClient Initialize()
    {
        if (url is null)
            throw new MessengerHardClientException("url is null");
        Client?.Close();
        Client = new WebSocket(url);
        return this;
    }

    Source 1
    public void Connect()
    {
        if (Client is null)
            throw new MessengerHardClientException("WS client not initialized");
        if (sessionId is null)
            throw new MessengerHardClientException("sessionId is null");
        if (messageHandler is null)
            throw new MessengerHardClientException("Message handler is null");
        Client.OnOpen += (s, e) =>
        {
            var connect = new StompMessage("CONNECT");
            connect["accept-version"] = "1.1";
            connect["heart-beat"] = "10000,10000";
            Client.Send(serializer.Serialize(connect));
        };
        Client.OnError += (s, e) => { Console.WriteLine("WS ERROR: " + e.Message); };
        Client.OnMessage += (s, e) => messageHandler.OnMessageResive(serializer.Deserialize(e.Data));
        Client.Connect();
    }

    Source 1
    public WSClient SetURL(string url)
    {
        this.url = url;
        return this;
    }

    Source 2
    public WSClient SetSessionId(int sessionId)
    {
        this.sessionId = sessionId;
        return this;
    }

    Source 0
    public int GetSessionId()
    {
        return this.sessionId.HasValue ? this.sessionId.Value : -1;
    }

    Source 2
    public WSClient SetMessageHandler(IMessageHandler handler)
    {
        this.messageHandler = handler;
        return this;
    }

    Source 11
    public void SubscribeTo(string destination)
    {
        if (Client is null)
            throw new MessengerHardClientException("Client is null. Sub canceled");
        var subscribeMessage = new StompMessage(StompFrame.SUBSCRIBE);
        subscribeMessage["id"] = "sub-" + sessionId;
        subscribeMessage["destination"] = "/" + state + destination;
        Client.Send(serializer.Serialize(subscribeMessage));
    }

    Source 4
    public void UnsubscribeFrom(string destination)
    {
        if (Client is null)
            throw new MessengerHardClientException("Client is null. Unsub canceled");
        var unsubscribeMessage = new StompMessage(StompFrame.UNSUBSCRIBE);
        unsubscribeMessage["id"] = "sub-" + sessionId;
        unsubscribeMessage["destination"] = "/" + state + destination;
        Client.Send(serializer.Serialize(unsubscribeMessage));
    }

    Source 3
    public void Send<T>(string destination, T body)
    {
        if (Client is null)
            throw new MessengerHardClientException("Client is null. Send canceled");
        var message = new StompMessage(StompFrame.SEND, JsonSerializer.Serialize(body));
#pragma warning disable CS8601
        message["sessionid"] = sessionId.ToString();
#pragma warning restore CS8601
        message["id"] = "sub-" + sessionId;
        message["destination"] = "/" + app + destination;
        Client.Send(serializer.Serialize(message));
    }

    Source 1
    public void Disconnect()
    {
        Client?.Close();
    }
}

```

Рис. 28. Класс клиента.

После реализации вышеуказанного, можно провести тестирование работы соединения и правильности данных. По результатам данного тестирования можно

приступить к созданию механизма обработки этих данных, представляющий их в объекты.

2.3.2. Обработка данных полученных с сервера

Поскольку все данные передаваемые в телах являются JSON объектами, то требуется представить их в виде объектов языка. В С# существует инструментарий работы с JSON, что невероятно облегчит работу. Так же требуется создать модель для каждой сущности и объекта, который может быть отправлен, например, ошибки или ответ на вход в систему, в соответствие с имеющимися на сервере (см. рис. 29).

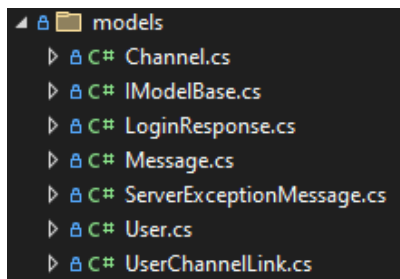


Рис. 29. Модели на стороне клиента

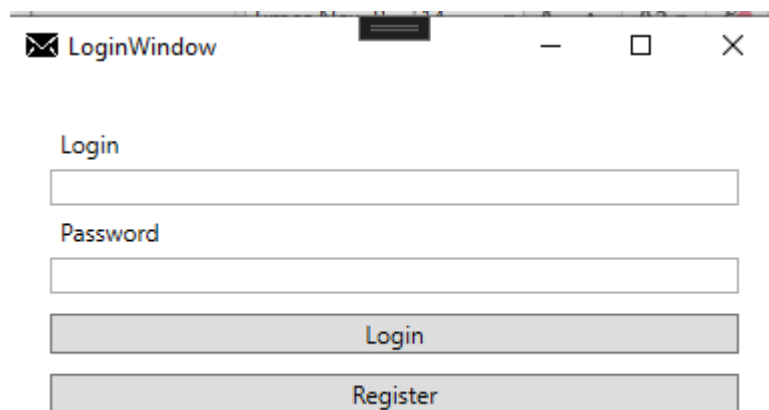
Помимо моделей понадобится класс, который хранит в себе методы для преобразования JSON, который будет при некоторых условиях прокидывать ошибки, которые будут обработаны в других местах приложения. Он нужен только для удобства работы в дальнейшем.

Центром всего приложения будет являться класс, который обрабатывает все полученные данные, сохраняет их в неких пулах и позволяет с ними взаимодействовать. Главными задачами такового будет обеспечивать обновление состояния сообщений и каналов при получении или изменении, должны существовать события, на которые будут подписаны окна приложения, а также он должен иметь интерфейс, который позволит получить эти данные из других мест, такие как вход или регистрация нового аккаунта, создание канала и пользователей в него.

Так же при запуске требуется считывать URL с файла настройки.

2.3.3. Интерфейс приложения

При запуске приложения пользователь в первую очередь видит окно авторизации (см. рис. 30), которое позволяет как совершить вход в приложение, так и зарегистрироваться в нём.



The screenshot shows a window titled "LoginWindow" with a standard Windows title bar. Inside the window, there are two text input fields. The first field is labeled "Login" and the second is labeled "Password". Below these fields are two buttons: "Login" and "Register". The "Login" button is positioned above the "Register" button. Both buttons have a light gray background and black text.

Рис. 30. Экран логина

На данном экране имеется две кнопки, разница которых только в запросе, отправляемом к серверу. По итогу активации кнопок, при правильно указанных данных, должны всплыть информационные сообщения: первое сообщает о том, что данные были введены верно, а второе о том, что соединение с сервером установлено по вебсокету, только после этого открывается главное окно.

На главном окне располагается список доступных чатов, список сообщений, поле для ввода и кнопка для отправки. (см. рис. 31)

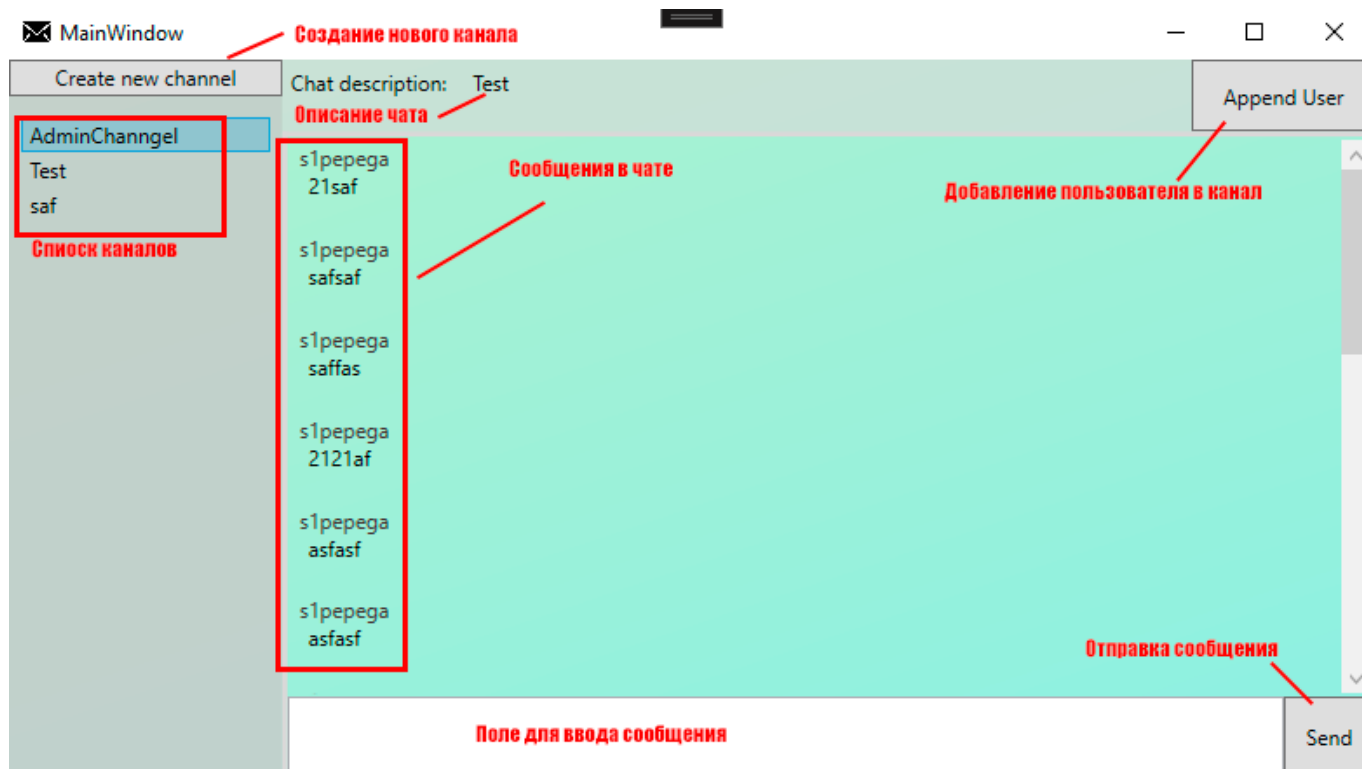


Рис. 31. Главное окно приложения

Данное окно является основным средством коммуникации, в нем отображаются все сообщения, отправленные пользователями. Так же на нём имеются кнопки, которые позволяют создать канал (см. рис. 32) или же добавить пользователя (см. рис. 33) в существующий. После ввода данных и подтверждения диалоговые окна будут закрыты автоматически.

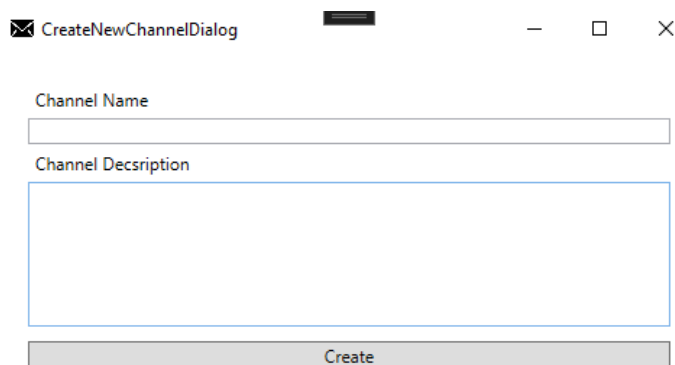


Рис. 32. Диалоговое окно для создания канала

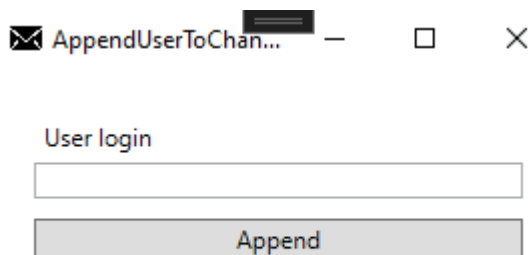


Рис. 33. Диалоговое окно для добавления пользователя в канал

Помимо видимого функционала, представленного на рисунке 31, у главного окна есть ещё несколько возможностей: для этого требуется нажать правой кнопкой мыши на любую свободную область, откроется контекстное меню, с единственным элементом, кнопкой перезагрузки. При нажатии на неё полностью перезагрузится главное окно и все соединения. Это может помочь при возникновении неожиданных обстоятельств, например, ошибок, не отображение актуального состояния чата и так далее. Кнопка удобна как для пользователей приложения, так и для тестировщиков и разработчиков.

Контекстное меню также существует и для сообщений: оно открывается при клике на правую кнопку мыши (см. рис. 34) и имеет две кнопки: изменить и удалить. С помощью этих кнопок пользователь имеет возможность взаимодействовать с ранее отправленными сообщениями. При этом ограничение на изменение сообщений существует с обеих сторон системы, человек может взаимодействовать только со своими сообщениями, к тому же они не будут удалены до конца, их предыдущее состояние будет сохранено в базе данных.

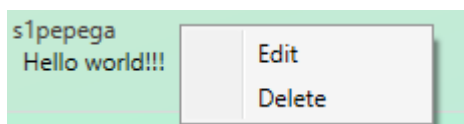


Рис. 34. Контекстное меню сообщений

3. ОХРАНА ТРУДА

Охрана труда является важным аспектом работы, без знания которого высока вероятность травмироваться, дестабилизировать здоровье и иные пагубные последствия.

3.1. Общие требования охраны труда

К работе допускается лицо, в возрасте старше 18 лет (мужчина или женщина), прошедшее вводный инструктаж по охране труда, пожарной безопасности и электробезопасности и по оказанию первой доврачебной помощи пострадавшему при несчастном случае, допущенное к работе по медицинскому заключению.

При выполнении программистом обязанностей необходимо соблюдать нормы трудового законодательства РФ, правила внутреннего трудового распорядка, а также режим труда и отдыха.

Осуществление работы, а также взаимодействие со всеми заинтересованными лицами по вопросам, связанных с деятельностью, возможно только при наличии безопасного состояния рабочих мест, отвечающих требованиям охраны труда и производственной санитарии.

На программиста процессе работы могут оказывать действие следующие опасные и вредные производственные факторы:

1. физические:
 - 1.1. повышенные уровни электромагнитного излучения;
 - 1.2. повышенные уровни рентгеновского излучения;
 - 1.3. повышенные уровни ультрафиолетового излучения;
 - 1.4. повышенный уровень инфракрасного излучения;
 - 1.5. повышенный уровень статического электричества;
 - 1.6. повышенные уровни запыленности воздуха рабочей зоны;
 - 1.7. повышенное содержание положительных аэроионов в воздухе рабочей зоны;
 - 1.8. пониженное содержание отрицательных аэроионов в воздухе рабочей

зоны;

- 1.9. пониженная или повышенная влажность воздуха рабочей зоны;
- 1.10. пониженная или повышенная подвижность воздуха рабочей зоны;
- 1.11. повышенный уровень шума;
- 1.12. повышенный или пониженный уровень освещенности;
- 1.13. повышенный уровень прямой блёскости;
- 1.14. повышенный уровень отраженной блёскости;
- 1.15. повышенный уровень ослепленности;
- 1.16. неравномерность распределения яркости в поле зрения;
- 1.17. повышенная яркость светового изображения;
- 1.18. повышенный уровень пульсации светового потока;
- 1.19. повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека;
2. химические, повышенное содержание в воздухе рабочей зоны двуокиси углерода, озона, аммиака, фенола, формальдегида и полихлорированных бифенилов;
3. психофизиологические:
 - 3.1. напряжение зрения;
 - 3.2. напряжение внимания;
 - 3.3. интеллектуальные нагрузки;
 - 3.4. эмоциональные нагрузки;
 - 3.5. длительные статические нагрузки;
 - 3.6. монотонность труда;
 - 3.7. большой объем информации, обрабатываемой в единицу времени;
 - 3.8. нерациональная организация рабочего места;
4. биологические, повышенное содержание в воздухе рабочей зоны микроорганизмов.

Программист должен соблюдать правила пожарной безопасности, знать места расположения первичных средств пожаротушения и при необходимости уметь ими пользоваться.

Программисту необходимо немедленно извещать начальника отдела о любой

ситуации, угрожающей жизни и здоровью работников, о каждом несчастном случае, происшедшем на производстве, или об ухудшении состояния своего здоровья.

Программисту в случаях травмирования работников необходимо прекратить работу и обратиться в медицинское учреждение. Программист должен уметь оказать первую помощь пострадавшим.

Программист должен проходить обучение безопасным методам и приемам выполнения работ, и оказанию первой помощи пострадавшим на производстве, инструктаж по охране труда, проверку знаний требований охраны труда, а также готовить предложения по улучшению условий труда для включения в комплексный план.

Программист обязан своевременно предоставлять в отдел охраны труда запрашиваемую информацию.

Программист обязан участвовать в проведении специальной оценки условий труда работников по условиям труда.

Программист обязан принимать предписания отдела охраны труда под подпись и исполнять их в указанные сроки.

Программист должен быть обеспечен СИЗ в соответствии с Межотраслевыми правилами обеспечения работников специальной одеждой, специальной обувью и другими средствами индивидуальной защиты, утвержденными Приказом Минздравсоцразвития России от 01.06.2009 N 290н; выдаваемые средства индивидуальной защиты должны соответствовать характеру и условиям работы и обеспечивать безопасность труда. Не допускаются приобретение и выдача работникам средств индивидуальной защиты без сертификата соответствия. Характеристика выданных СИЗ (номенклатура, срок выдачи и нормы соответствия) устанавливается из личных карточек работников, занятых на определенном рабочем месте.

3.2. Требования охраны труда перед началом работы

Программист перед началом работы обязан:

Осмотреть и привести в порядок рабочее место, убрать посторонние предметы,

которые могут отвлекать внимание и затруднять работу.

Проверить визуальным осмотром достаточность освещенности рабочей поверхности, отсутствие свисающих и оголенных концов электрической проводки, состояние полов, окон.

Проверить исправность оборудования и правильность его подключения к сети, проверить устойчивость производственных столов, стеллажей, прочность крепления оборудования.

Включить питание ПК, соблюдая последовательность: сетевой фильтр, монитор, периферийные устройства, процессор.

Программисту не разрешается приступать к работе в случае обнаружения неисправности оборудования.

Программисту обязан сообщить об обнаруженной неисправности оборудования административно - хозяйственной службе и приступить к работе только после устранения нарушений в работе или неисправностей оборудования.

3.3. Требования охраны труда во время работы

Во время работы программист обязан: следить за соблюдением требований охраны труда работниками; не загромождать рабочее место и проходы к нему; соблюдать правила использования оборудования.

При работе с ПК программист обязан: соблюдать установленные режимы рабочего времени: регламентированные перерывы в работе и выполнять в физкультурных паузах рекомендованные упражнения для глаз, шеи, рук, туловища, ног; соблюдать расстояние глаз до экрана в пределах 60-70 см., но не ближе 50 см с учетом размеров алфавитно-цифровых знаков и символов.

Программисту при работе на ПК не должен: касаться одновременно экрана монитора и клавиатуры; допускать попадания влаги на поверхность системного блока, монитора, рабочую поверхность клавиатуры, принтера и других устройств; производить самостоятельное вскрытие и ремонт оборудования.

Программист обязан отключить ПК из электросети: при обнаружении неисправности; при внезапном снятии напряжения электросети; во время чистки и

уборки оборудования.

Продолжительность непрерывной работы с ПК не должна превышать 2 часов.

При работе на копировально-множительном оборудовании программист не должен: освобождать заевшую бумагу при включенном питании; выключать оборудования, не дожидаясь его автоматического отключения; производить самостоятельное вскрытие и ремонт копировально-множительного устройства; класть и ставить на копировально-множительный аппарат посторонние предметы, подвергать его механическим воздействиям.

При передвижении пешком программист обязан выполнять правила дорожного движения для пешеходов: выбрать маршрут передвижения с соблюдением мер личной безопасности. Если на маршруте движения есть (или появились) опасные участки, то выбрать обходной путь; двигаться по тротуарам или пешеходным дорожкам, а при их отсутствии - по обочинам; пересекать проезжую часть по пешеходным переходам, в том числе по подземным и надземным; в местах, где движение регулируется, программист должен руководствоваться сигналами регулировщика или пешеходного светофора, а при его отсутствии - транспортного светофора. Не успев закончить переход, программист должен остановиться на линии, разделяющей транспортные потоки противоположных направлений. Продолжать переход можно, лишь убедившись в безопасности дальнейшего движения и с учетом сигнала светофора (регулировщика).

3.4. Требования охраны труда по окончании работы

Выключить электрические приборы, отключить от электрической сети средства оргтехники и другое оборудование.

4. ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКИЙ РАЗДЕЛ

4.1. Организация рабочего места

Рабочее место программиста должно занимать площадь не менее 6 м, высота помещения должна быть не менее 4 м, а объем — не менее 20 м³ на одного человека. Высота над уровнем пола рабочей поверхности, за которой работает оператор, должна составлять 720 мм. Желательно, чтобы рабочий стол оператора при необходимости можно было регулировать по высоте в пределах 680 – 780 мм. Оптимальные размеры поверхности стола 1600 х 1000 кв. мм. Под столом должно иметься пространство для ног с размерами по глубине 650 мм. Рабочий стол оператора должен также иметь подставку для ног, расположенную под углом 15° к поверхности стола. Длина подставки 400 мм, ширина — 350 мм. Удаленность клавиатуры от края стола должна быть не более 300 мм, что обеспечит оператору удобную опору для предплечий. Расстояние между глазами оператора и экраном видеодисплея должно составлять 40 – 80 см. Рабочий стул программиста должен быть снабжен подъемно-поворотным механизмом. Высота сиденья должна регулироваться в пределах 400 – 500 мм. Глубина сиденья должна составлять не менее 380 мм, а ширина — не менее 400 мм. Высота опорной поверхности спинки не менее 300 мм, ширина — не менее 380 мм. Угол наклона спинки стула к плоскости сиденья должен изменяться в пределах 90 – 110° (см. рис. 35).

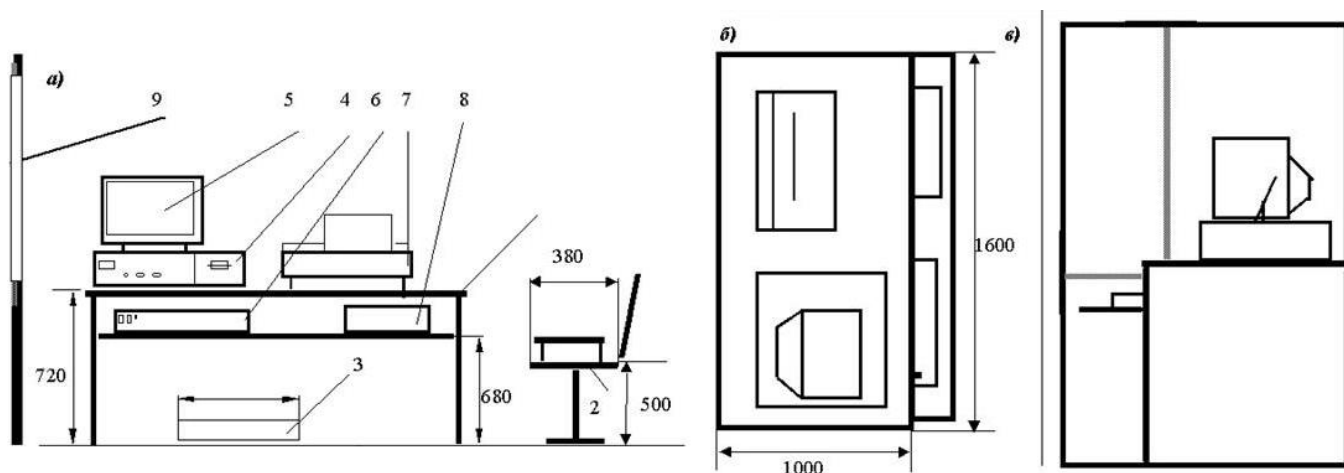


Рис. 35. Схема рабочего места программиста: вид спереди; вид с верху; вид с боку

4.2 Расчет затрат на разработку корпоративного мессенджера

Целью расчетного раздела является расчет затрат на разработку корпоративного мессенджера.

Затраты на разработку состоят из:

1. Затраты на материалы.
2. Основной заработной платы разработчика.
3. Дополнительной заработной платы.
4. Отчисления на единый социальный налог.
5. Накладных расходов.
6. Расчет затрат на эксплуатацию оборудования.

В плановую себестоимость включены все затраты независимо от источника финансирования, которые рассматриваются постатейно.

4.2.1. Расчет затрат на материалы

В затраты на материалы входят:

Затраты на основные и вспомогательные материалы, при этом они определяются по рыночным ценам на момент проведения разработки.

Расчет затрат на материалы производится по формуле:

$$З_m = Ц * К \quad (1)$$

Где:

З_м – затраты на материалы;

Ц - цена материала (руб);

К – количество материала (ед);

Затраты на материалы представлены в таблице. (см. табл. 2).

Таблица 2

Затраты на материалы

№	Наименование материала	Цена (руб)	Количество	Сумма (руб)
1.	Бумага, лист	1,1	500	550
2.	Картридж принтера, шт.	1 150	1	1150
3.	Флеш накопитель 16ГБ, шт.	299	2	598
Итого				2298

Затраты на вспомогательные материалы рассчитываются в процентном соотношении к основным материалам и составляют 20%, или 460 руб.

Суммарные затраты на материалы составляют: $2298 + 460 = 2758$ руб.

4.2.2 Расчет основной заработной платы разработчика

Заработная плата учитывает выполнение основных видов работ по данной теме и складывается из затраченного времени на разработку и оплаты труда разработчика.

Расчет основной заработной платы.

Исходя из того, что тарифная часовая ставка разработчика составляет 130 руб./час, время, требуемое на данную разработку – 100 часов. Основная заработная плата составит 13000 руб.

$$З_{Посн} = 130 * 100 = 13000 \text{ рублей}$$

4.2.3 Дополнительная заработная плата

Дополнительная заработная плата составляет 10% от основной.

Формула для расчета дополнительной заработной платы:

$$З_{Пдоп} = З_{Посн} * 10\% / 100\% \quad (2)$$

$$З_{Пдоп} = 13000 * 10\% / 100\%$$

$$З_{Пдоп} = 1300 \text{ рублей.}$$

4.2.4 Отчисления на единый социальный налог

Включают в себя обязательные платежи во внебюджетные фонды с учетом основной и дополнительной заработной платы и составляют: 30%

Из которых:

- пенсионный фонд – 22%;
- медицинское страхование – 5,1%
- фонд социального страхования - 2,9%;

Отчисления определяются по формуле:

$$\text{Отч.} = (\text{з/Посн} + \text{з/Пдоп}) * 0,30 \text{ руб.} \quad (3)$$

$$\text{Отч.} = (13000 + 1300) * 0,30 = 4290 \text{ руб.}$$

Не зависимо от вида заработной платы, на неё должен быть начислен поясной коэффициент, который в Новосибирской области составляет -25%.

Отчисления с районным коэффициентом определяются по формуле:

$$\text{Отч. с р. к-том} = (\text{з/Посн.} + \text{з/Пдоп.} + \text{отч.}) * 0,25 \text{ руб.} \quad (4)$$

$$\text{Отч. с р. к-том} = (13000 + 1300 + 4290) * 0,25 = 4647 \text{ руб.}$$

4.2.5 Расчет накладных расходов

В эту статью включаются расходы на управление и хозяйственное обслуживание, на содержание и текущий ремонт оборудования, амортизационные отчисления и капитальный ремонт.

Накладные расходы – это те расходы, которые не имеют непосредственного отношения к созданию продукции (аренда помещений, коммуникационные расходы и т.д.)

Накладные расходы рассчитываются в процентном отношении к основной заработной плате и составляют 25%, в данном случае - 3250 рублей.

4.2.6. Расчет стоимости работы оборудования

Расчет включает в себя:

- Определение амортизации от стоимости оборудования;
- Затраты на электроэнергию, потребляемую оборудованием.

Для определения этих показателей необходимо найти эффективный фонд времени работы оборудования, который определяется по формуле:

$$T_{\text{эф. об}} = ((D_r * T_{\text{см.}} * h - P_{\text{ч}}) * (100 - P_r)) / 100 \text{ (час.)} \quad (5)$$

Где:

D_r – количество дней, требуемых на разработку – 20 дней;

$T_{\text{см.}}$ – продолжительность смены – 5 часов;

h – количество смен – 1;

$P_{\text{ч}}$ – количество предпраздничных часов;

P_r – процент простоя оборудования согласно графику планового предупредительного ремонта (15%);

$$T_{\text{эф. об.}} = ((20 * 5 * 1) * (100 - 15)) / 100 = 85 \text{ ч.}$$

Амортизационные отчисления от стоимости оборудования, приходящиеся на 1 машино / час, рассчитываются по формуле:

$$A_{\text{отч. обр.}} = (P * A_n) / T_{\text{эф. об.}} \text{ (руб.)} \quad (6)$$

Где:

P – первоначальная стоимость оборудования;

A_n – норма амортизации;

Норма амортизации устанавливается государством и составляет 15% в год.

$$A_{\text{отч. обр.}} = (25000 * 0,15) / 85 = 44 \text{ (руб.)}$$

Расчет затрат на электроэнергию определяется по формуле:

$$Z_{\text{эл. эн}} = D * C_{\text{эл.}} \text{ (руб.)} \quad (7)$$

Где D – потребляемая мощность оборудования – 0,4 кВт/ч;

$C_{\text{эл.}}$ – цена 1 кВт/ч – 3,65 руб.

$$Z_{\text{эл. эн}} = 0,4 * 3,65 = 1,46 \text{ руб.}$$

Суммарные затраты на 1 машино/час определяются по формуле:

$$З_{м/ч} = А_{отч.обр.} + З_{эл.эн.} \text{ (руб.)} \quad (8)$$

$$З_{м/ч} = 44 + 1,46 = 45,46 \text{ руб.}$$

Суммарные затраты на эксплуатацию оборудования определяются по формуле:

$$З_{экс.ком} = З_{м/ч} * Т_{раб.об.} \text{ (руб.)} \quad (9)$$

$$З_{экс.ком} = 45,46 * 85 = 3864 \text{ руб}$$

Затраты на разработку корпоративного мессенджера сведем в итоговую таблицу (см. табл. 3).

Таблица 3

Сумма затрат на разработку

№	Наименование статьи	Сумма (руб.)	Процентное соотношение
1	Затраты на материалы	2758	8%
2	Основная заработная плата	13000	39%
3	Дополнительная заработная плата	1300	4%
4	Отчисления на единый социальный налог	4290	13%
5	Начисление поясного коэффициента	4647	14%
6	Накладные расходы	3250	10%
7	Затраты на эксплуатацию оборудования	3864	12%
8	Итог:	33109	100%

В результате проведенных расчетов, затраты на разработку интернет-магазина составляют 33109 руб. Анализируя итоговую таблицу, можно сделать следующий вывод, что большая часть издержек приходится на основную заработную плату и составляет 13000 руб. или 39%. Снизить расходы по данной статье можно путем увеличения производительности труда и сокращения времени на разработку.

4.2.7. Расчет цены программного продукта

Цена складывается из нескольких компонентов:

$$Ц = С + П + НДС \quad (10)$$

Где:

С - себестоимость программного продукта;

П – прибыль (планируемая), которую берем в размере 40%, от себестоимости;

НДС - налог на добавленную стоимость 20%, от суммы себестоимости и прибыли.

$$П = 33109 * 40\% / 100\% = 13243 \text{ руб.}$$

$$НДС = (33109 + 13243) * 20\% / 100\% = 9270 \text{ руб.}$$

Подставляя значения в формулу №10, определяем цену программного продукта:

$$Ц = 33109 + 13243 + 9270 = 55622 \text{ руб.}$$

Результаты расчетов сведем в итоговую таблицу (см. табл. 4).

Таблица 4

Сводная таблица показателей

Наименование показателя	Сумма (руб).
Себестоимость программного продукта	33109
Прибыль	13243
НДС	9270
Цена программного продукта	55622

ЗАКЛЮЧЕНИЕ

В ходе разработки корпоративного мессенджера были выполнены следующие цели и задачи:

- изучены существующие приложения;
- определены требования и потребности пользователей;
- спроектирована архитектура системы;
- разработана серверная часть системы;
- разработано функциональное приложение;
- в клиентском приложении разработан интерфейс;
- выполнено тестирование и отладка готовой системы.

Обеспечение безопасности было реализовано частично: функционирует идентификация сессий и проверка прав пользователя. К сожалению, так и не было реализовано шифрование передаваемой информации, а также информации о самих пользователях. Тем не менее, взаимодействие с паролем происходит только на серверной стороне. На данном этапе шифрование не является приоритетной задачей. Его можно будет реализовать по требованию заказчика.

За время работы над проектом я познакомился с фреймворком SpringBoot и его модулями, а также со сборщиком Gradle, научился пользоваться Докером, повысил уровень знания git, освежил знания по C# и WPF.

Считаю, что работа прошла довольно успешно, я смог реализовать большинство целей и задач, которые передо мной стояли. В дальнейшем я планирую развивать систему и улучшать созданное приложение. В дальнейшем можно будет добавить шифрование, создать дополнительные модули, которые были описаны во введении.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Перлова О. Н. , Ляпина О. П. , Гусева А. В. Проектирование и разработка информационных систем [Текст] / Перлова О. Н. , Ляпина О. П. , Гусева А. В. — 2-е изд. стер.. — : , 2018 — 256 с.
2. Сильвия Ботрос, Джереми Тинли MySQL по максимуму [Текст] / Сильвия Ботрос, Джереми Тинли — Санкт-Петербург: Издательский дом «Питер», 2023 — 433 с.
3. Эйдан Хобсон Сейерс, Иан Милл Docker на практике [Текст] / Эйдан Хобсон Сейерс, Иан Милл — Москва: ДМК Пресс, 2020 — 516 с.
4. Спилкэ Л. Spring быстро [Текст] / Спилкэ Л.— 4-е изд.. — Санкт-Петербург: Издательский дом «Питер», 2023 — 448 с.
5. Стандарт RFC 6455 — The WebSocket protocol / [Электронный ресурс] // IETF Datatracker : [сайт]. — URL: <https://datatracker.ietf.org/doc/html/rfc6455> (дата обращения: 18.02.2023).
6. Стандарт RFC 2616 — HTTP/1.1 / [Электронный ресурс] // IETF Datatracker : [сайт]. — URL: <https://datatracker.ietf.org/doc/html/rfc2616> (дата обращения: 20.02.2023).
7. Стандарт RFC 1945 — HTTP/1.0 / [Электронный ресурс] // IETF Datatracker : [сайт]. — URL: <https://datatracker.ietf.org/doc/html/rfc1945z> (дата обращения: 20.02.2023).
8. Описание протокола STOMP [Электронный ресурс] // The Simple Text Oriented Messaging Protocol : [сайт]. — URL: <https://stomp.github.io/> (дата обращения: 21.02.2023).
9. Обзор XAML (WPF .NET) [Электронный ресурс] // Техническая документация Microsoft : [сайт]. — URL: <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0&redirectedfrom=MSDN> (дата обращения: 22.02.2023).

10. Документация по Windows Presentation Foundation [Электронный ресурс] // Техническая документация Microsoft : [сайт]. — <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/?view=netdesktop-7.0> (дата обращения: 22.02.2023).
11. Windows Forms [Электронный ресурс] // Техническая документация Microsoft : [сайт]. — URL: <https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/?view=netdesktop-7.0&preserve-view=true> (дата обращения: 22.02.2023).
12. Полное руководство по языку программирования C# 11 и платформе .NET 7 [Электронный ресурс] // METANIT.COM Сайт о программировании : [сайт]. — URL: <https://metanit.com/sharp/tutorial/> (дата обращения: 26.02.2023).
13. Документация по Visual Studio [Электронный ресурс] // Техническая документация Microsoft : [сайт]. — URL: <https://learn.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2019> (дата обращения: 1.03.2023).
14. Документация по фреймворку Spring Boot [Электронный ресурс] // Spring Boot Reference Documentation : [сайт]. — URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> (дата обращения: 10.03.2023).
15. Документация Docker [Электронный ресурс] // Docker Docs: [сайт]. — URL: <https://docs.docker.com/> (дата обращения: 1.04.2023).

ПРИЛОЖЕНИЕ

Приложение 1

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,
NO_ENGINE_SUBSTITUTION';
```

```
CREATE SCHEMA IF NOT EXISTS `mymessenger` DEFAULT CHARACTER SET utf8 ;
USE `mymessenger` ;
```

```
-----
-- Table `mymessenger`.`usersaccounts`
-----
```

```
CREATE TABLE IF NOT EXISTS `mymessenger`.`usersaccounts` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `login` VARCHAR(45) NOT NULL,
  `username` VARCHAR(45) NULL DEFAULT 'login',
  `userpassword` VARCHAR(45) NOT NULL,
  `createtime` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `login_UNIQUE` (`login` ASC) INVISIBLE)
ENGINE = InnoDB;
```

```
-----
-- Table `mymessenger`.`channels`
-----
```

```
CREATE TABLE IF NOT EXISTS `mymessenger`.`channels` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `channelname` VARCHAR(45) NOT NULL,
  `Description` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

```
-----
-- Table `mymessenger`.`messages`
-----
```

```
CREATE TABLE IF NOT EXISTS `mymessenger`.`messages` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `content` VARCHAR(2000) NOT NULL,
  `channel` INT NOT NULL,
  `sender` INT NOT NULL,
  `sendtime` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `updatetime` DATETIME NULL,
  PRIMARY KEY (`id`),
  INDEX `sender_idx` (`sender` ASC) VISIBLE,
  INDEX `channel_idx` (`channel` ASC) VISIBLE,
  CONSTRAINT `fk_message_sender_id`
    FOREIGN KEY (`sender`)
      REFERENCES `mymessenger`.`usersaccounts` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_message_channel_id`
    FOREIGN KEY (`channel`)
      REFERENCES `mymessenger`.`channels` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-----
-- Table `mymessenger`.`messageseditlogs`
-----
```

```
CREATE TABLE IF NOT EXISTS `mymessenger`.`messageseditlogs` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Messageid` INT NOT NULL,
  `Prevcontent` VARCHAR(2000) NULL,
  `ChangeAt` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  INDEX `messageid_idx` (`Messageid` ASC) INVISIBLE,
  CONSTRAINT `fk_edited_message_id`
    FOREIGN KEY (`Messageid`)
      REFERENCES `mymessenger`.`messages` (`id`)
    ON DELETE NO ACTION
```

```

    ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```

-----
-- Table `mymessenger`.`userchannels`
-----

```

```

CREATE TABLE IF NOT EXISTS `mymessenger`.`userchannels` (
  `id` INT NOT NULL,
  `userid` INT NOT NULL,
  `availablechannel` INT NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `iduserchannels_UNIQUE` (`id` ASC) VISIBLE,
  INDEX `fk_userid_idx` (`userid` ASC) VISIBLE,
  INDEX `fk_availablechannel_channelid_idx` (`availablechannel` ASC) VISIBLE,
  CONSTRAINT `fk_userid_accountid`
    FOREIGN KEY (`userid`)
      REFERENCES `mymessenger`.`usersaccounts` (`id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_availablechannel_channelid`
    FOREIGN KEY (`availablechannel`)
      REFERENCES `mymessenger`.`channels` (`id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

```

-----
-- Table `mymessenger`.`requestslogs`
-----

```

```

CREATE TABLE IF NOT EXISTS `mymessenger`.`requestslogs` (
  `id` INT NOT NULL,
  `request` VARCHAR(512) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

```

```

USE `mymessenger`;

```

```

DELIMITER $$

```

```

USE `mymessenger`$$

```

```

CREATE DEFINER = CURRENT_USER TRIGGER `mymessenger`.`messages_AFTER_UPDATE` AFTER UPDATE ON `messages` FOR EACH ROW
BEGIN
  INSERT INTO messageseditlogs(Messageid, Prevcontent, ChangeAt) VALUE (NEW.id, OLD.content, current_time());
END$$

```

```

DELIMITER ;

```

```

CREATE USER 'serverUser' IDENTIFIED BY 'slpepega';

```

```

GRANT ALL ON `mymessenger`.* TO 'serverUser';

```

```

GRANT SELECT, INSERT, TRIGGER, UPDATE, DELETE ON TABLE `mymessenger`.* TO 'serverUser';

```

```

GRANT SELECT ON TABLE `mymessenger`.* TO 'serverUser';

```

```

GRANT SELECT, INSERT, TRIGGER ON TABLE `mymessenger`.* TO 'serverUser';

```

```

SET SQL_MODE=@OLD_SQL_MODE;

```

```

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

```

```

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/configs/WebSocketConfig.java

```
package s1pepega.diplom.CorpMessengerServer.configs;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.converter.DefaultContentTypeResolver;
import org.springframework.messaging.converter.MappingJackson2MessageConverter;
import org.springframework.messaging.converter.MessageConverter;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.util.MimeTypeUtils;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

import java.util.List;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker( "/state");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws").withSockJS();
    }

    @Override
    public boolean configureMessageConverters(List<MessageConverter> messageConverters) {
        DefaultContentTypeResolver resolver = new DefaultContentTypeResolver();
        resolver.setDefaultMimeType(MimeTypeUtils.APPLICATION_JSON);
        MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
        converter.setObjectMapper(new ObjectMapper());
        converter.setContentTypeResolver(resolver);
        messageConverters.add(converter);
        return false;
    }
}
```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/controllers/ExceptionHandler.java

```
package s1pepega.diplom.CorpMessengerServer.controllers;

import jakarta.persistence.EntityNotFoundException;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import s1pepega.diplom.CorpMessengerServer.exceptions.IllegalSessionIdException;
import s1pepega.diplom.CorpMessengerServer.exceptions.WIPException;
import s1pepega.diplom.CorpMessengerServer.models.ExceptionResponse;
```

```

@RestControllerAdvice
public class ExceptionController {

    @ResponseStatus(HttpStatus.NOT_FOUND)
    @ExceptionHandler(EntityNotFoundException.class)
    private ExceptionResponse notFound(EntityNotFoundException ex) {
        ExceptionResponse response = new ExceptionResponse();
        response.setMessage(ex.getMessage());
        return response;
    }

    @ResponseStatus(HttpStatus.NOT_IMPLEMENTED)
    @ExceptionHandler(WIPEXception.class)
    private ExceptionResponse notIMPL(WIPEXception ex){
        ExceptionResponse response = new ExceptionResponse();
        response.setMessage(ex.getMessage());
        return response;
    }

    @ResponseStatus(HttpStatus.UNAUTHORIZED)
    @ExceptionHandler(IllegalSessionIdException.class)
    private ExceptionResponse secure(IllegalSessionIdException ex){
        ExceptionResponse response = new ExceptionResponse();
        response.setMessage(ex.getMessage());
        return response;
    }

    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ExceptionHandler(RuntimeException.class)
    private ExceptionResponse error(RuntimeException ex) {
        ExceptionResponse response = new ExceptionResponse();
        response.setMessage(ex.getMessage());
        return response;
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/controllers/rest/ChannelControllerREST.java

```

package s1pepega.diplom.CorpMessengerServer.controllers.rest;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.handler.annotation.Header;
import org.springframework.web.bind.annotation.*;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.Message;
import s1pepega.diplom.CorpMessengerServer.entities.User;
import s1pepega.diplom.CorpMessengerServer.exceptions.WIPEXception;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.ChannelService;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.MessageService;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.UserChannelService;

import java.util.List;

import static org.springframework.util.MimeTypeUtils.APPLICATION_JSON_VALUE;

@RestController
@RequestMapping("/api/channels")
@RequiredArgsConstructor

```

```

public class ChannelControllerREST {
    @Autowired
    private ChannelService channelService;
    @Autowired
    private UserChannelService ucService;
    @Autowired
    private MessageService messageService;

    @GetMapping(produces = APPLICATION_JSON_VALUE)
    public List<Channel> findAll() {
        return channelService.findAll();
    }

    @GetMapping(value = "/{channelId}", produces = APPLICATION_JSON_VALUE)
    public Channel findById(@PathVariable Integer channelId) {
        return channelService.findById(channelId);
    }

    @GetMapping(value = "/{channelId}/getUsers", produces = APPLICATION_JSON_VALUE)
    public List<User> getUsers(@PathVariable Integer channelId) {
        return ucService.getChannelUsers(channelId);
    }

    @GetMapping(value = "/{channelId}/getMessagesAll", produces = APPLICATION_JSON_VALUE)
    public List<Message> getMessages(@PathVariable Integer channelId) {
        Channel channel = channelService.findById(channelId);
        return messageService.getMessagesAllInChannel(channel);
    }

    @GetMapping(value = "/{channelId}/getMessages/from{skip}/to{count}", produces = APPLICATION_JSON_VALUE)
    public List<Message> getMessagesFromTo(
        @PathVariable Integer channelId,
        @PathVariable Integer skip,
        @PathVariable Integer count) {
        Channel channel = channelService.findById(channelId);
        return messageService.getMessagesInChannel(channel, skip, count);
    }

    @PostMapping(value = "/create", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public Channel create(@RequestBody Channel request, @RequestHeader("sessionId") Integer sessionId) {
        return channelService.create(request);
    }

    @PatchMapping(value = "/update", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public Channel update(@RequestBody Channel request, @RequestHeader("sessionId") Integer sessionId) {
        return channelService.update(request);
    }

    @DeleteMapping(value = "/delete/{id}", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public Channel delete(@PathVariable Integer id) {
        throw new WIPException();
    }
    // Channel channel = channelService.findById(id);
    // channelService.delete(id);
    // return channel;
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/controllers/rest/UserControllerREST.java

```

package s1pepega.diplom.CorpMessengerServer.controllers.rest;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.*;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.User;
import s1pepega.diplom.CorpMessengerServer.exceptions.IllegalSessionIdException;
import s1pepega.diplom.CorpMessengerServer.models.LoginResponse;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.SessionService;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.UserChannelService;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.UserService;

import java.util.HashMap;
import java.util.List;

import static org.springframework.util.MimeTypeUtils.APPLICATION_JSON_VALUE;

@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserControllerREST {

    @Autowired
    private UserService userService;
    @Autowired
    private UserChannelService ucService;
    @Autowired
    private SessionService sessionService;

    @GetMapping(produces = APPLICATION_JSON_VALUE)
    public List<User> findAll() {
        return userService.findAll();
    }

    @GetMapping(value =("/{userId}", produces = APPLICATION_JSON_VALUE)
    public User findById(@PathVariable Integer userId, @RequestHeader("sessionId") Integer sessionId) {
        return userService.findById(userId);
    }

    @GetMapping(value =("/{username}", produces = APPLICATION_JSON_VALUE)
    public User findByName(@PathVariable String username, @RequestHeader("sessionId") Integer sessionId) {
        return userService.findByName(username);
    }

    @GetMapping(value =("/{userId}/getChannels", produces = APPLICATION_JSON_VALUE)
    public List<Channel> getChannels(@PathVariable Integer userId, @RequestHeader("sessionId") Integer sessionId) {
        // checkSession(userId,sessionId);
        return ucService.getUserChannels(userId);
    }

    @PostMapping(value = "/create", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
    public LoginResponse create(@RequestBody User request) {
        User newUser = userService.createUser(request);
        Integer sessionId = sessionService.getNewSessionId();
        sessionService.appendNewSession(sessionId,newUser.getId());
        return new LoginResponse().setUser(newUser).setSessionId(sessionId);
    }
}

```

```

@PostMapping(value = "/login", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
public LoginResponse login(@RequestBody User request) {
    User user = userService.login(request);
    Integer sessionId = sessionService.getNewSessionId();
    sessionService.appendNewSession(sessionId,user.getId());
    return new LoginResponse().setUser(user).setSessionId(sessionId);
}

@PatchMapping(value = "/update", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
public User update(@RequestBody User request, @RequestHeader("sessionId") Integer sessionId) {
    checkSession(request.getId(), sessionId);
    return userService.update(request);
}

@DeleteMapping(value = "/delete/{id}", consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
public User delete(@PathVariable Integer id, @RequestHeader("sessionId") Integer sessionId) {
    checkSession(id,sessionId);
    User user = userService.findById(id);
    userService.delete(id);
    return user;
}

private void checkSession(Integer userId, Integer sessionId){
    if(!sessionService.canSendRequest(sessionId, userId))
        throw new IllegalSessionIdException("access denied");
}
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/controllers/ws/EverythingWebSocketController.java

```

package s1pepega.diplom.CorpMessengerServer.controllers.ws;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.handler.annotation.*;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.stereotype.Controller;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.Message;
import s1pepega.diplom.CorpMessengerServer.entities.UserChannelLink;
import s1pepega.diplom.CorpMessengerServer.exceptions.IllegalSessionIdException;
import s1pepega.diplom.CorpMessengerServer.models.ExceptionResponse;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.ChannelService;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.MessageService;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.SessionService;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.UserChannelService;

import java.util.Objects;

@Controller
public class EverythingWebSocketController {
    @Autowired
    private UserChannelService ucService;
    @Autowired
    private ChannelService channelService;
    @Autowired
    private MessageService messageService;
    @Autowired
    private SessionService sessionService;
}

```



```

@Autowired
private SimpMessagingTemplate simpMessagingTemplate;

//USER
@MessageMapping("/add_user_to_channel/{userId}")
@SendTo("/state/user/{userId}/append_to_channel")
public UserChannelLink addUserToChannel(
    @DestinationVariable Integer userId,
    @Payload UserChannelLink link,
    @Header("sessionId") Integer sessionId
){
    return ucService.create(link);
}

@MessageMapping("/delete_user_from_channel/{userId}")
@SendTo("/state/user/{userId}/remove_from_channel")
public void deleteUserFromChannel(
    @DestinationVariable Integer userId,
    @Payload UserChannelLink link,
    @Header("sessionId") Integer sessionId
) throws Exception {
    ucService.delete(link);
}

//CHANNEL
@MessageMapping("/delete_channel/{id}")
@SendTo("/state/channel/{id}/delete")
public Channel deleteChannel(
    @DestinationVariable Integer id,
    @Payload Channel channel,
    @Header("sessionId") Integer sessionId
) throws Exception {
    channelService.delete(channel.getId());
    ucService.getChannelUsersLink(channel.getId()).forEach(ucService::delete);
    return channel;
}

//MESSAGE
@MessageMapping("/channel/{channelId}/send_message")
@SendTo("/state/channel/{channelId}/msg/send")
public Message sendMessage(
    @DestinationVariable Integer channelId,
    @Payload Message message,
    @Header("sessionId") Integer sessionId
) throws Exception {
    checkMsgPermission(message,sessionId);
    return messageService.sendMessage(message);
}

@MessageMapping("/channel/{channelId}/change_message")
@SendTo("/state/channel/{channelId}/msg/change")
public Message changeMessage(
    @DestinationVariable Integer channelId,
    @Payload Message message,
    @Header("sessionId") Integer sessionId
) throws Exception {
    checkMsgPermission(message,sessionId);
    return messageService.editMessage(message);
}

```

```

@MessageMapping("/channel/{channelId}/delete_message")
@SendTo("/state/channel/{channelId}/msg/delete")
public Message deleteMessage(
    @DestinationVariable Integer channelId,
    @Payload Message message,
    @Header("sessionId") Integer sessionId
) throws Exception {
    checkMsgPermission(message, sessionId);
    messageService.delete(message.getId());
    return message;
}

private void checkMsgPermission(Message message, Integer sessionId){
    if(!Objects.equals(sessionService.getUserIdAsSession(sessionId), message.getSender().getId()))
        throw new IllegalSessionIdException("this session cannot manage this user's messages");
}

//Exception handler
@MessageMapping("/debug/excTest")
public Integer testMessage(@Payload Integer sessionId){
    throw new IllegalSessionIdException("TEST MESSAGE EXCEPTION HANDLER", sessionId);
}

@MessageExceptionHandler
public void handleException(Throwable exception){
    if(exception instanceof IllegalSessionIdException){
        try{
            Integer userId = sessionService.getUserIdAsSession(((IllegalSessionIdException) exception).sessionId);
            simpMessagingTemplate.convertAndSend("/state/user/"+userId+"/exception", exception.getMessage());
        }catch (Exception ignored){}
    }
}
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/CorpMessengerServerApplication.java

```

package s1pepega.diplom.CorpMessengerServer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CorpMessengerServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(CorpMessengerServerApplication.class, args);
    }

}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/entities/Channel.java

```

package s1pepega.diplom.CorpMessengerServer.entities;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.*;
import lombok.Getter;

```

```

import lombok.Setter;
import lombok.experimental.Accessors;

import java.util.Objects;

@Getter
@Setter
@Accessors(chain = true)
@Entity
@Table(name = "channels")
@JsonIgnoreProperties({"hibernateLazyInitializer"})
public class Channel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "channelname", nullable = false)
    private String name;
    @Column(name = "description")
    private String desc;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Channel channel = (Channel) o;
        return Objects.equals(id, channel.id) && Objects.equals(name, channel.name) && Objects.equals(desc, channel.desc);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name, desc);
    }

    @Override
    public String toString() {
        return "Channel{" +
            "Id=" + id +
            ", name=" + name + "\"" +
            ", desc=" + desc + "\"" +
            '}';
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/entities/Message.java

```

package s1pepega.diplom.CorpMessengerServer.entities;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.Accessors;

import java.util.Date;
import java.util.Objects;

@Getter
@Setter

```

```

@Accessors(chain = true)
@Entity
@Table(name = "messages")
@JsonIgnoreProperties({"hibernateLazyInitializer"})
public class Message {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "content", nullable = false)
    private String content;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "channel", nullable = false, foreignKey = @ForeignKey(name = "fk_message_channel_id"))
    private Channel channel;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "sender", nullable = false, foreignKey = @ForeignKey(name = "fk_message_sender_id"))
    private User sender;
    @JsonFormat(shape = JsonFormat.Shape.STRING)
    @Column(name = "sendtime")
    private Date sendTime;
    @JsonFormat(shape = JsonFormat.Shape.STRING)
    @Column(name = "updatetime")
    private Date updateTime;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Message message = (Message) o;
        return Objects.equals(id, message.id) && Objects.equals(content, message.content) && Objects.equals(channel, message.channel) && Objects.equals(sender, message.sender) && Objects.equals(sendTime, message.sendTime) && Objects.equals(updateTime, message.updateTime);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, content, channel, sender, sendTime, updateTime);
    }

    @Override
    public String toString() {
        return "Message{" +
            "id=" + id +
            ", content='" + content + '\'' +
            ", channel=" + channel +
            ", sender=" + sender +
            ", sendTime=" + sendTime +
            ", updateTime=" + updateTime +
            '}';
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/entities/User.java

```

package s1pepega.diplom.CorpMessengerServer.entities;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;

```

```

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.Accessors;

import java.util.Date;
import java.util.Objects;

@Getter
@Setter
@Accessors(chain = true)
@Entity
@Table(name = "usersaccounts")
@JsonIgnoreProperties({"hibernateLazyInitializer"})
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(name = "login", unique = true, nullable = false)
    private String login;

    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    @Column(name = "userpassword", nullable = false)
    private String password;
    @JsonFormat(shape = JsonFormat.Shape.STRING)
    @Column(name = "createtime")
    private Date regTime;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id) && Objects.equals(login, user.login) && Objects.equals(password, user.password) && Objects.equals(regTime, user.regTime);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, login, password, regTime);
    }

    @Override
    public String toString() {
        return "User{" +
            "id=" + id +
            ", login=" + login + "\" +
            ", password=" + password + "\" +
            ", regTime=" + regTime +
            '"';
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/entities/UserChannellink.java

```

package s1pepega.diplom.CorpMessengerServer.entities;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.*;

```

```

import lombok.Getter;
import lombok.Setter;
import lombok.experimental.Accessors;

@Getter
@Setter
@Accessors(chain = true)
@Entity
@Table(name = "userchannels")
@JsonIgnoreProperties({"hibernateLazyInitializer"})
public class UserChannelLink {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name="userid", nullable = false, foreignKey = @ForeignKey(name = "fk_userId_accountId"))
    private User user;
    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name="availablechannel", nullable = false, foreignKey = @ForeignKey(name = "fk_availableChannel_channelId"))
    private Channel channel;

    @Override
    public String toString() {
        return "UserChannelLink{" +
            "id=" + id +
            ", user=" + user +
            ", channel=" + channel +
            '}';
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/exceptions/IllegalSessionIdException.java

```

package s1pepega.diplom.CorpMessengerServer.exceptions;

public class IllegalSessionIdException extends RuntimeException{

    public Integer sessionId = -1;
    public IllegalSessionIdException(String message) {
        super(message);
    }

    public IllegalSessionIdException(String message, Integer id){
        this(message);
        this.sessionId = id;
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/exceptions/WIPEException.java

```

package s1pepega.diplom.CorpMessengerServer.exceptions;

public class WIPEException extends RuntimeException{
    public WIPEException(){super("The logic of this function is under development");}
    public WIPEException(String message){super(message);}
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/models/ExceptionResponse.java

```
package s1pepega.diplom.CorpMessengerServer.models;
```

```
import lombok.Data;
import lombok.experimental.Accessors;
```

```
@Data
@Accessors(chain = true)
public class ExceptionResponse {
    private String message;
}
```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/models/LoginResponse.java

```
package s1pepega.diplom.CorpMessengerServer.models;
```

```
import lombok.Data;
import lombok.experimental.Accessors;
import s1pepega.diplom.CorpMessengerServer.entities.User;
```

```
@Data
@Accessors(chain = true)
public class LoginResponse {
    User user;
    Integer sessionId;
}
```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/repositories/ChannelRepository.java

```
package s1pepega.diplom.CorpMessengerServer.repositories;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
```

```
@Repository
public interface ChannelRepository extends JpaRepository<Channel, Integer> {
}
```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/repositories/MessageRepository.java

```
package s1pepega.diplom.CorpMessengerServer.repositories;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import s1pepega.diplom.CorpMessengerServer.entities.Message;
```

```
import java.util.List;
```

```
@Repository
public interface MessageRepository extends JpaRepository<Message, Integer> {
```

```
    @Query(value = "SELECT * FROM messages m WHERE m.channel = :channelId LIMIT :skip, :count", nativeQuery = true)
```

```

List<Message> getMessageInChannelWithStartAndCount(
    @Param("channelId")Integer channelId,
    @Param("skip")Integer skip,
    @Param("count")Integer count);

@Query(value = "SELECT * FROM messages m WHERE m.channel = :channelId", nativeQuery = true)
List<Message> getMessageAllInChannel(
    @Param("channelId")Integer channelId);
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/repositories/UserChannelRepository.java

```

package s1pepega.diplom.CorpMessengerServer.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import s1pepega.diplom.CorpMessengerServer.entities.UserChannelLink;

import java.util.List;

public interface UserChannelRepository extends JpaRepository<UserChannelLink, Integer> {

    @Query(value = "SELECT DISTINCT * FROM userchannels uc WHERE uc.userId = :id", nativeQuery = true)
    public List<UserChannelLink> getUserChannels(@Param("id") Integer id);

    @Query(value = "SELECT DISTINCT * FROM userchannels uc WHERE uc.availableChannel = :id", nativeQuery = true)
    public List<UserChannelLink> getChannelUsers(@Param("id") Integer id);
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/repositories/UserRepository.java

```

package s1pepega.diplom.CorpMessengerServer.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.User;

import java.util.List;
import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
    @Query(value = "SELECT * FROM usersaccounts u WHERE u.login = :login", nativeQuery = true)
    Optional<User> findByName(@Param("login") String login);
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/impls/ChannelServiceImpl.java

```

package s1pepega.diplom.CorpMessengerServer.services.impls;

import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;

```



```

import org.springframework.stereotype.Service;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.repositories.ChannelRepository;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.ChannelService;

import java.util.ArrayList;
import java.util.List;

import static java.util.Optional.ofNullable;

@Service("ChannelServiceImpl")
@RequiredArgsConstructor
public class ChannelServiceImpl implements ChannelService {
    @Autowired
    private ChannelRepository channelRepository;
    @Override
    public List<Channel> findAll() {
        return new ArrayList<>(channelRepository.findAll());
    }

    @Override
    public Channel findById(Integer id) {
        return channelRepository.findById(id)
            .orElseThrow(()->new EntityNotFoundException("Channel with id "+id+" not found"));
    }

    @Override
    public Channel create(Channel channelResponse) {
        Channel channel = new Channel().setName(channelResponse.getName())
            .setDesc(channelResponse.getDesc());
        return channelRepository.save(channel);
    }

    @Override
    public Channel update(Channel channelRequest) {
        Channel channel = channelRepository.findById(channelRequest.getId())
            .orElseThrow(()->new EntityNotFoundException("Channel with id "+channelRequest.getId()+" not found"));
        ofNullable(channelRequest.getName()).map(channel::setName);
        ofNullable(channelRequest.getDesc()).map(channel::setDesc);
        return channelRepository.save(channel);
    }

    @Override
    public void delete(Integer id) {
        channelRepository.deleteById(id);
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/impls/MessageServiceImpl.java

```

package s1pepega.diplom.CorpMessengerServer.services.impls;

import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.Message;

```

```

import s1pepega.diplom.CorpMessengerServer.repositories.ChannelRepository;
import s1pepega.diplom.CorpMessengerServer.repositories.MessageRepository;
import s1pepega.diplom.CorpMessengerServer.repositories.UserRepository;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.MessageService;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import static java.util.Optional.ofNullable;

@Service("MessageServiceImpl")
@RequiredArgsConstructor
public class MessageServiceImpl implements MessageService {
    @Autowired
    private MessageRepository messageRepository;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private ChannelRepository channelRepository;

    @Override
    @Transactional(readOnly = true)
    public List<Message> findAll() {
        return new ArrayList<>(messageRepository.findAll());
    }

    @Override
    @Transactional(readOnly = true)
    public Message findById(Integer id) {
        return messageRepository.findById(id)
            .orElseThrow(() -> new EntityNotFoundException("Message with id " + id + " not found"));
    }

    @Override
    @Transactional(readOnly = true)
    public List<Message> getMessagesInChannel(Channel channel, Integer start, Integer count) {
        return new ArrayList<>(messageRepository.getMessageInChannelWithStartAndCount(
            channel.getId(),
            count,
            start
        ));
    }

    @Override
    @Transactional(readOnly = true)
    public List<Message> getMessagesAllInChannel(Channel channel) {
        return new ArrayList<>(messageRepository.getMessageAllInChannel(channel.getId()));
    }

    @Override
    @Transactional
    public Message editMessage(Message messageRequest) {
        Message message = messageRepository.findById(messageRequest.getId())
            .orElseThrow(() -> new EntityNotFoundException("Message with id " + messageRequest.getId() + " not found"));
        ofNullable(messageRequest.getContent()).map(message::setContent);
        message.setUpdateTime(new Date());
        return messageRepository.save(message);
    }
}

```

```

@Override
@Transactional
public Message sendMessage(Message sendMessageByUserRequest) {
    userRepository.findById(sendMessageByUserRequest.getSender().getId())
        .orElseThrow(() -> new EntityNotFoundException("User with id "+sendMessageByUserRequest.getSender().getId()+"
not found"));
    channelRepository.findById(sendMessageByUserRequest.getChannel().getId())
        .orElseThrow(() -> new EntityNotFoundException("Channel with id
"+sendMessageByUserRequest.getChannel().getId()+" not found"));
    sendMessageByUserRequest.setSendTime(new Date());
    return messageRepository.save(sendMessageByUserRequest);
}

@Override
@Transactional
public void delete(Integer id) {
    messageRepository.deleteById(id);
}
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/impls/SessionServiceImpl.java

```

package s1pepega.diplom.CorpMessengerServer.services.impls;

import org.springframework.stereotype.Service;
import s1pepega.diplom.CorpMessengerServer.exceptions.IllegalSessionIdException;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.SessionService;

import java.util.HashMap;

@Service("SessionServiceImpl")
public class SessionServiceImpl implements SessionService {

    private static final HashMap<Integer, Integer> sessions = new HashMap<>();

    @Override
    public Integer getUserIdAsSession(Integer sessionId) {
        if(!sessions.containsKey(sessionId))
            throw new IllegalSessionIdException("unknown session, user not found");
        return sessions.get(sessionId);
    }

    @Override
    public void appendNewSession(Integer sessionId, Integer userId) {
        sessions.putIfAbsent(sessionId,userId);
    }

    @Override
    public Boolean canSendRequest(Integer sessionId, Integer userId) {
        if(userId == null)
            throw new IllegalSessionIdException("userId not present");
        if(!sessions.containsKey(sessionId))
            throw new IllegalSessionIdException("unknown session, send request canceled");
        return sessions.get(sessionId).equals(userId);
    }

    @Override
    public void closeSession(Integer sessionId) {
        sessions.remove(sessionId);
    }
}

```

```

    }

    @Override
    public Integer getNewSessionId() {
        return (int)System.currentTimeMillis();
    }

    @Override
    public HashMap<Integer, Integer> getSessions() {
        return sessions;
    }
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/impls/UserChannelServiceImpl.java

```

package s1pepega.diplom.CorpMessengerServer.services.impls;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.User;
import s1pepega.diplom.CorpMessengerServer.entities.UserChannelLink;
import s1pepega.diplom.CorpMessengerServer.repositories.UserChannelRepository;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.UserChannelService;

import java.util.List;
import java.util.stream.Collectors;

@Service("UserChannelServiceImpl")
@RequiredArgsConstructor
public class UserChannelServiceImpl implements UserChannelService {

    @Autowired
    private UserChannelRepository ucRepository;

    @Override
    @Transactional(readOnly = true)
    public List<UserChannelLink> findAll() {
        return ucRepository.findAll();
    }

    @Override
    @Transactional(readOnly = true)
    public List<Channel> getUserChannels(Integer userId) {
        return ucRepository.getUserChannels(userId)
            .stream()
            .map(UserChannelLink::getChannel)
            .collect(Collectors.toList());
    }

    @Override
    @Transactional(readOnly = true)
    public List<UserChannelLink> getUserChannelsLink(Integer userId) {
        return ucRepository.getUserChannels(userId);
    }

    @Override

```

```

@Transactional(readOnly = true)
public List<User> getChannelUsers(Integer channelId) {
    return ucRepository.getChannelUsers(channelId)
        .stream()
        .map(UserChannelLink::getUser)
        .collect(Collectors.toList());
}

@Override
@Transactional(readOnly = true)
public List<UserChannelLink> getChannelUsersLink(Integer channelId) {
    return ucRepository.getChannelUsers(channelId);
}

@Override
@Transactional
public UserChannelLink create(UserChannelLink link) {
    return ucRepository.save(link);
}

@Override
@Transactional
public void delete(UserChannelLink link) {
    ucRepository.delete(link);
}
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/impls/UserServiceImpl.java

```

package s1pepega.diplom.CorpMessengerServer.services.impls;

import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import s1pepega.diplom.CorpMessengerServer.entities.User;
import s1pepega.diplom.CorpMessengerServer.repositories.UserRepository;
import s1pepega.diplom.CorpMessengerServer.services.interfaces.UserService;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Optional;

import static java.util.Optional.ofNullable;

@Service("UserServiceImpl")
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    @Override
    @Transactional(readOnly = true)
    public List<User> findAll() {
        return new ArrayList<>(userRepository.findAll());
    }
}

```

```

@Override
@Transactional(readOnly = true)
public User findById(Integer id) {
    return userRepository.findById(id)
        .orElseThrow(()->new EntityNotFoundException("User with id "+id+" not found"));
}

@Override
@Transactional(readOnly = true)
public User findByName(String name) {
    return userRepository.findByName(name)
        .orElseThrow(()->new EntityNotFoundException("User with login "+name+" not found"));
}

@Override
@Transactional(readOnly = true)
public User login(User userRequest) {
    User user = userRepository.findByName(userRequest.getLogin())
        .orElseThrow(()->new EntityNotFoundException("User with login "+userRequest.getLogin()+" not found"));
    if(!user.getPassword().equals(userRequest.getPassword()))
        throw new EntityNotFoundException("Illegal password. Try again");
    return user;
}

@Override
@Transactional
public User createUser(User newUser) {
    Optional<User> optionalUser = userRepository.findByName(newUser.getLogin());
    if(optionalUser.isPresent())
        throw new EntityNotFoundException("User with login "+newUser.getLogin()+" was registered.");
    newUser.setRegTime(new Date());
    return userRepository.save(newUser);
}

@Override
@Transactional
public User update(User updatedUserInfo) {
    User user = userRepository.findById(updatedUserInfo.getId())
        .orElseThrow(()->new EntityNotFoundException("User with id "+updatedUserInfo.getId()+" not found"));
    ofNullable(updatedUserInfo.getLogin()).map(user::setLogin);
    ofNullable(updatedUserInfo.getPassword()).map(user::setPassword);
    userRepository.save(user);
    return user;
}

@Override
@Transactional
public void delete(Integer id) {
    throw new EntityNotFoundException("You cannot delete message.");
}
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/interfaces/ChannelService.java

```
package s1pepega.diplom.CorpMessengerServer.services.interfaces;
```

```
import org.springframework.lang.NonNull;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
```

```

import java.util.List;

public interface ChannelService {

    @NonNull
    List<Channel> findAll();

    @NonNull
    Channel findById(@NonNull Integer id);

    @NonNull
    Channel create(@NonNull Channel createChannel);

    @NonNull
    Channel update(@NonNull Channel createChannel);

    void delete(@NonNull Integer id);
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/interfaces/MessageService.java

```

package s1pepega.diplom.CorpMessengerServer.services.interfaces;

import org.springframework.lang.NonNull;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.Message;

import java.util.List;

public interface MessageService {

    @NonNull
    List<Message> findAll();

    @NonNull
    Message findById(@NonNull Integer id);

    @NonNull
    List<Message> getMessagesInChannel(@NonNull Channel channel, Integer start, Integer count);

    @NonNull
    List<Message> getMessagesAllInChannel(@NonNull Channel channel);

    @NonNull
    Message editMessage(@NonNull Message message);

    @NonNull
    Message sendMessage(@NonNull Message message);

    void delete(@NonNull Integer id);
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/interfaces/SessionService.java

```

package s1pepega.diplom.CorpMessengerServer.services.interfaces;

import java.util.HashMap;

```

```

public interface SessionService {

    public Integer getUserIdAsSession(Integer sessionId);

    public void appendNewSession(Integer sessionId, Integer userId);

    public Boolean canSendRequest(Integer sessionId, Integer userId);

    public void closeSession(Integer sessionId);

    public Integer getNewSessionId();

    public HashMap<Integer,Integer> getSessions();
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/interfaces/UserChannelService.java

```

package s1pepega.diplom.CorpMessengerServer.services.interfaces;

import org.springframework.lang.NonNull;
import s1pepega.diplom.CorpMessengerServer.entities.Channel;
import s1pepega.diplom.CorpMessengerServer.entities.User;
import s1pepega.diplom.CorpMessengerServer.entities.UserChannelLink;

import java.util.List;

public interface UserChannelService {
    @NonNull
    List<UserChannelLink> findAll();

    @NonNull
    List<Channel> getUserChannels(@NonNull Integer userId);
    @NonNull
    List<UserChannelLink> getUserChannelsLink(@NonNull Integer userId);

    @NonNull
    List<User> getChannelUsers(@NonNull Integer channelId);
    @NonNull
    List<UserChannelLink> getChannelUsersLink(@NonNull Integer channelId);

    @NonNull
    UserChannelLink create(@NonNull UserChannelLink link);

    void delete(@NonNull UserChannelLink link);
}

```

CorpMessengerServer/srcmain/java/s1pepega/diplom/CorpMessengerServer/services/interfaces/UserService.java

```

package s1pepega.diplom.CorpMessengerServer.services.interfaces;

import org.springframework.lang.NonNull;
import s1pepega.diplom.CorpMessengerServer.entities.User;

import java.util.List;

public interface UserService {

```



```

@NonNull
List<User> findAll();

@NonNull
User findById(@NonNull Integer id);

@NonNull
User findByName(@NonNull String name);

@NonNull
User login(@NonNull User potentialUser);

@NonNull
User createUser(@NonNull User newUser);

@NonNull
User update(@NonNull User updatedUserInfo);

void delete(@NonNull Integer id);
}

```

CorpMessengerServer/srcmain/resources/application.properties

```

spring.application.name=diplom_messenger_server
# props for test
spring.datasource.url=jdbc:mysql://localhost:3306/mymessenger
spring.datasource.username=serverUser
spring.datasource.password=s1pepega
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

```