

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Соловейчик Ю.Г.

(фамилия, имя, отчество)

(подпись)

«_____» _____ 2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

..... Коренко Юлии Олеговны

(фамилия, имя, отчество студента – автора работы)

***..... Разработка на языке программирования Java раздела сайта компании
с социальными сервисами для сотрудников***

(тема работы)

Факультет Прикладной математики и информатики

(полное название факультета)

Направление подготовки 01.03.02. Прикладная математика и информатика

(код и наименование направления подготовки бакалавра)

**Руководитель
от НГТУ**

Вагин Д.В.

(фамилия, имя, отчество)

Д.Т.Н., доцент

(ученая степень, ученое звание)

(подпись, дата)

**Автор выпускной
квалификационной работы**

Коренко Ю.О.

(фамилия, И.О.)

ФПМИ, ПМ-15

(факультет, группа)

(подпись, дата)

Новосибирск, 2025 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой

Соловейчик Ю.Г.

(фамилия, имя, отчество)

«14» марта 2025 г.

(подпись)

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

студенту Коренко Юлии Олеговне
(фамилия, имя, отчество студента)

Направление подготовки 01.03.02. Прикладная математика и информатика

Факультет Прикладной математики и информатики

Тема Разработка на языке программирования Java раздела сайта компании
с социальными сервисами для сотрудников

Исходные данные (или цель работы):

Изучение фреймворка Spring для Java-платформы и разработка раздела сайта
компании с социальными сервисами для сотрудников

Структурные части работы:

1. Изучение модуля Spring Boot фреймворка Spring.
2. Изучение протокола связи WebSocket.
3. Изучение фреймворка Spring Security.
4. Реализация подсистемы авторизации и аутентификации.
5. Разработка механизма активации пользователя.
6. Реализация профиля сотрудника и интеграция с базой данных.
7. Реализация чата с поддержкой обмена документами.
8. Реализация списка пользователей с возможностью поиска и фильтрации.
9. Реализация административной панели управления.

10.Разработка связи отдела кадров и администратора сайта.

11.Разработка онлайн-ежедневника.

Задание согласовано и принято к исполнению.

**Руководитель
от НГТУ**

Вагин Д.В.

(фамилия, имя, отчество)

д.т.н., доцент

(ученая степень, ученое звание)

14.03.2025 г.

(подпись, дата)

Студент

Коренко Ю.О.

(фамилия, имя, отчество)

ФПМИ, ПМ-15

(факультет, группа)

14.03.2025 г.

(подпись, дата)

Тема утверждена приказом по НГТУ № 1457/2 от «14» марта 2025 г.

ВКР сдана в ГЭК №____, тема сверена с данными приказа

(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Отчет 112 с., 2 ч., 43 рис., 8 табл., 10 источников, 2 прил.

ВЕБ-ПРИЛОЖЕНИЕ, JAVA, SPRING BOOT, SPRING SECURITY, WEB-SOCKET, POSTGRESQL, JAVASCRIPT.

Объект разработки – раздел сайта компании с социальными сервисами.

Цель работы – спроектировать и реализовать сайт с возможностью обмена сообщениями между пользователями, ведения ежедневника, администрирования сайта.

В результате работы был разработан раздел сайта с возможностью обмена сообщениями между пользователями, ведения ежедневника, администрирования сайта.

СОДЕРЖАНИЕ

Введение.....	5
1. Программные средства.....	7
1.1. Spring Boot.....	7
1.2. Apache Maven.....	7
1.3. Thymeleaf.....	8
1.4. Spring Security	9
1.5. WebSocket.....	10
1.6. JavaScript.....	11
1.7. PostgreSQL.....	11
2. Реализация сайта	13
2.1. Анализ функциональности веб-приложения.....	13
2.2. Структура базы данных	14
2.3. Страница авторизации	18
2.4. Страница активации	21
2.5. Страница профиля	23
2.6. Страница пользователей.....	24
2.7. Страница чата.....	27
2.8. Страница ежедневника.....	31
2.9. Страница администратора	33
2.10. Страница HR	37
Заключение	41
Список литературы	42
Приложение 1. Серверная часть	43
Приложение 2. Клиентская часть	68

ВВЕДЕНИЕ

Современные корпоративные порталы играют ключевую роль в организации внутренних коммуникаций и повышении эффективности работы сотрудников. Одним из важнейших элементов таких порталов являются социальные сервисы, включающие чаты, ежедневники и инструменты для взаимодействия между коллегами.

Целью выпускной квалификационной работы бакалавра является проектирование и реализация раздела сайта компании с социальными сервисами для сотрудников.

План выполнения выпускной квалификационной работы:

1. Разработка прототипа сайта;
2. Выбор программных средств;
3. Проектирование базы данных;
4. Разработка серверной части;
5. Разработка клиентской части;
6. Тестирование.

Востребованность данного проекта заключается в создании специализированного раздела корпоративного сайта, направленного на улучшение взаимодействия внутри коллектива. Такой раздел не только упростит обмен информацией, но и повысит вовлеченность сотрудников, способствуя более слаженной работе команды.

Внутренний корпоративный чат — это удобный и быстрый способ обмена информацией между сотрудниками. В отличие от электронной почты или телефонных звонков, чат позволяет вести диалоги в реальном времени, оперативно решать рабочие вопросы и согласовывать действия. Это особенно важно в условиях динамичной работы, когда задержки в коммуникации могут привести к потере времени или ошибкам.

Персональный ежедневник помогает сотрудникам структурировать рабочий процесс, фиксировать важные встречи, дедлайны и задачи. В отличие от бумажных или разрозненных электронных записей, интегрированный ежедневник в

корпоративном портале обеспечивает централизованное хранение информации и напоминания о предстоящих событиях.

1. ПРОГРАММНЫЕ СРЕДСТВА

1.1. SPRING BOOT

Spring Boot – это фреймворк для быстрой разработки приложений на основе Spring. Он упрощает настройку и развертывание, предоставляя готовые конфигурации и встроенные серверы (например, используемый в проекте Tomcat).

Этот фреймворк значительно ускоряет процесс управления зависимостями посредством предоставления этих зависимостей в виде starter-пакетов.

Так же он упрощает конфигурацию приложения, например, через файл `application.properties`. В проекте этот файл содержит такие конфигурационные настройки, как:

- Настройка Thymeleaf (шаблонизатора HTML) для указания пути к шаблонам;
- Настройка сервера - указание адресов, с которых сервер может принимать запросы и указание порта, на котором запускается приложение;
- Настройка базы данных – содержит данные для подключения к PostgreSQL;
- Автоматическое создание таблиц в базе данных для хранения сессий;
- Настройка почты для отправки писем через SMTP-сервер.

1.2. APACHE MAVEN

Apache Maven представляет собой мощный инструмент для автоматизации сборки проектов и управления зависимостями в экосистеме Java. В основе его работы лежит концепция Project Object Model (POM), реализованная через XML-файл `pom.xml`, который служит центральным конфигурационным файлом проекта. Этот подход позволяет Maven автоматически разрешать, загружать и управлять всеми необходимыми библиотеками и их зависимостями из репозиторий, таких как Maven Central, избавляя разработчиков от ручного скачивания и подключения JAR-файлов.

Одной из ключевых особенностей Maven является его стандартизированная структура проекта, которая обеспечивает единообразие в организации исходного кода, тестов и ресурсов. Например, исходные файлы Java размещаются в `src/main/java`, тесты — в `src/test/java`, а ресурсы — в `src/main/resources`. Такая структура упрощает навигацию по проекту и делает его более понятным для других разработчиков.

Maven также предоставляет предопределённые фазы жизненного цикла сборки, такие как `clean`, `compile`, `test`, `package`, `install` и `deploy`. Эти фазы позволяют автоматизировать ключевые этапы разработки, включая компиляцию кода, запуск тестов, упаковку в архивные форматы (JAR, WAR) и развёртывание. Для расширения функциональности Maven поддерживает множество плагинов, которые можно настроить для выполнения специфических задач, таких как генерация документации, интеграция с Docker или создание исполняемых файлов.

1.3. THYMELEAF

Thymeleaf – это современный серверный шаблонизатор для Java-приложений, разработанный специально для веб-разработки. Он интегрируется с фреймворками Spring и Spring Boot, позволяя создавать динамические HTML-страницы, которые обрабатываются на стороне сервера перед отправкой клиенту. Thymeleaf сочетает в себе простоту HTML-шаблонов с мощными возможностями для работы с данными, что делает его популярным выбором для разработки веб-интерфейсов.

Ключевые особенности Thymeleaf:

- Thymeleaf использует стандартные HTML-атрибуты (например, `th:text`, `th:each`), что позволяет шаблонам оставаться валидными и корректно отображаться даже без обработки сервером. Это особенно удобно для дизайнеров и фронтенд-разработчиков, которые могут работать с шаблонами напрямую;
- Thymeleaf поддерживает передачу данных из Spring MVC-модели напрямую в HTML;

- Thymeleaf поддерживает модульный подход к разработке интерфейсов. Общие элементы, такие как шапка, подвал или навигационное меню, можно вынести в отдельные файлы и подключать их на нужных страницах с помощью атрибутов `th:replace` или `th:insert`;
- Thymeleaf предоставляет мощные инструменты для управления логикой отображения данных, включая условные операторы (`th:if`, `th:unless`) и циклы (`th:each`);

1.4. SPRING SECURITY

Spring Security – это фреймворк, предоставляющий инструменты для организации процессов аутентификации и авторизации, а также включает дополнительные функции по защите программного обеспечения.

В проекте используются такие его возможности, как:

- Регулировка доступа. Класс `Security Config` позволяет настроить доступ к страницам с учетом того, авторизован ли пользователь или имеет достаточные права для посещения той или иной страницы. С помощью данной регулировки в проекте не допускается посещение никаких страниц кроме авторизации, если пользователь еще не вошел в приложение (исключением является страница HR, к которой доступ совершается по IP). Так же доступ будет закрыт ко всем страницам, если пользователь еще не прошел активацию и к странице администратора имеет доступ только пользователь с соответствующими правами.
- Хэширование паролей. Использование `BCryptPasswordEncoder` позволяет хранить пароли в базе данных в зашифрованном виде. При регистрации нового пользователя пароль сохраняется в зашифрованном виде и, в дальнейшем, при авторизации, проект будет получать пароль от пользователя и сравнивать его хэш с тем, что находится в базе данных.

1.5. WEBSOCKET

WebSocket является протоколом связи, обеспечивающим полнодуплексное взаимодействие клиента и сервера через одно постоянное TCP-соединение.

На рисунке 1 наглядно отображен принцип работы WebSocket соединения. Как можно увидеть, единожды установив соединение с сервером, клиент остается прослушивать открытый канал до получения ответа от сервера. Например, рассмотрим взаимодействие клиента 1, клиента 2 и сервера. Оба клиента сначала совершают “рукопожатие” и переходят в режим ожидания. Затем клиент 2 решает отправить сообщение клиенту 1, и для этого он отправляет запрос, на что получит ответ от сервера, а клиент 1 – полезные данные (в данном случае это будет текст сообщения и другие параметры, такие как время, отправитель и прочее).

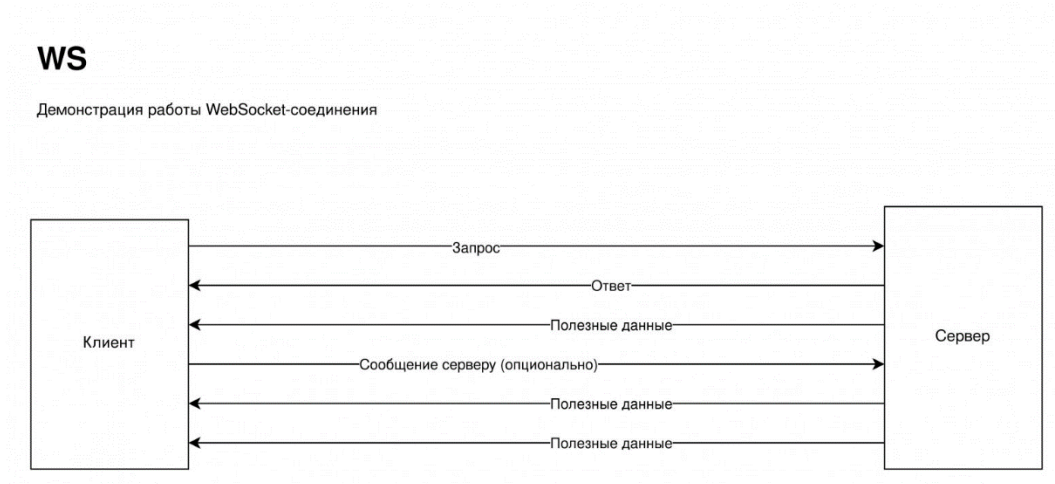


Рисунок 1 – Принцип работы WebSocket-соединения.

В проекте WebSocket был использован при реализации One-To-One чата (мгновенный обмен сообщениями, загрузка истории сообщений) и для отслеживания онлайн-статуса. Для расширения возможностей так же был использован STOMP - простой текстовый протокол обмена сообщениями, который работает поверх WebSocket и других транспортных протоколов.

1.6. JAVASCRIPT

JavaScript представляет собой высокоуровневый, интерпретируемый язык программирования, который играет фундаментальную роль в современной веб-разработке. Наряду с HTML (структура) и CSS (оформление), JavaScript образует третью ключевую технологию веб-платформы, отвечая за интерактивность и динамическое поведение страниц. В отличие от статических технологий разметки, JavaScript позволяет создавать сложные клиентские приложения, реагирующие на действия пользователя в реальном времени без необходимости перезагрузки страницы.

В контексте данного проекта JavaScript выполняет критически важную функцию, обеспечивая двустороннюю интерактивность между пользователем и системой. На клиентской стороне язык реализует сложную логику управления интерфейсом: плавное открытие и закрытие диалоговых окон чата, интеллектуальный поиск по списку пользователей с мгновенной фильтрацией результатов, динамическое обновление ленты сообщений без перезагрузки страницы. Особое внимание уделено механизму сохранения состояния интерфейса - JavaScript отслеживает текущий активный чат, запоминая выбор пользователя между сеансами работы через локальное хранилище браузера.

Серверная интеграция реализована через технологию WebSocket, где JavaScript выступает в роли посредника - собирает пользовательский ввод, форматирует данные в соответствии с API сервера, отправляет запросы и обрабатывает ответы. При получении новых сообщений или изменении статусов пользователей JavaScript динамически обновляет DOM-дерево, обеспечивая плавные переходы и визуальную обратную связь.

1.7. POSTGRESQL

PostgreSQL представляет собой мощную, свободно распространяемую объектно-реляционную систему управления базами данных (СУБД) с открытым исходным кодом, которая сочетает в себе надежность реляционной модели с гибкостью работы со сложными типами данных. Эта СУБД отличается строгим

соблюдением стандартов SQL, что обеспечивает высокую степень совместимости с другими системами, поддерживающими SQL, и позволяет легко переносить существующие приложения между различными платформами.

Одной из ключевых особенностей PostgreSQL является расширенная поддержка сложных и специализированных типов данных, включая JSON, XML, геопространственные данные, массивы и составные типы. Такая функциональность делает PostgreSQL особенно полезной для современных приложений, где требуется хранить и обрабатывать полуструктурированные или иерархические данные без потери преимуществ реляционной модели. Система предлагает полноценные механизмы работы с JSON, включая индексацию, сложные запросы и возможность преобразования между реляционными и документоориентированными представлениями данных.

PostgreSQL обеспечивает высокий уровень надежности и целостности данных благодаря строгой ACID-совместимости (Atomicity, Consistency, Isolation, Durability), сложным механизмам блокировок и многопоточности. Система поддерживает расширенные функции безопасности, включая строгий контроль доступа, шифрование данных и интеграцию с внешними системами аутентификации.

Производительность PostgreSQL оптимизирована для работы с большими объемами данных благодаря сложному планировщику запросов, поддержке параллельного выполнения операций и разнообразным методам индексирования (B-деревья, GiST, SP-GiST, GIN, BRIN). Система предлагает встроенную репликацию, точки восстановления и другие механизмы обеспечения отказоустойчивости, что делает ее подходящей для критически важных приложений.

Будучи проектом с открытым исходным кодом, PostgreSQL обладает активным сообществом разработчиков и постоянно совершенствуется, добавляя поддержку современных технологий и стандартов. Эта СУБД успешно применяется в самых разных сферах - от веб-приложений и геоинформационных систем до сложных аналитических платформ и систем принятия решений, демонстрируя исключительную гибкость, надежность и производительность в различных рабочих нагрузках.

2. РЕАЛИЗАЦИЯ САЙТА

В демонстрационных целях в интерфейсе программы используются вымышленные профили пользователей с искусственно сгенерированными именами и данными. Любое сходство с реальными лицами случайно.

2.1. АНАЛИЗ ФУНКЦИОНАЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЯ

При разработке сайта большое внимание стоит уделить ключевому моменту – чату. В первую очередь он должен реализовывать отправку и принятие сообщений в режиме реального времени. Так же, для удобства, пользователь должен видеть, было ли прочитано его сообщение другим пользователем или появились ли новые.

Стоит учесть допустимый размер сообщений для хранения в базе данных и возможность обмена не только текстовыми сообщениями, но и файлами (изображениями, текстовыми документами, архивами и т.д.). Каждое отправленное сообщение может быть отредактировано или удалено.

Для создания диалога с новым пользователем можно реализовать страницу, где будут выведены все сотрудники компании, среди которых нужного можно будет найти по имени, по статусу (в сети пользователь или нет) и некоторым данным (например, должность, пол и т.д.). На странице так же должна быть возможность просмотра профиля каждого сотрудника из списка и ссылка, открывающая диалог.

У каждого сотрудника будет существовать свой ежедневник, но количество одновременных записей должно быть ограничено.

Для контроля сайта следует реализовать административную панель управления, в которой пользователь с соответствующими правами будет иметь возможность добавлять и удалять пользователей как самостоятельно, так и по запросу отдела кадров.

Стоит учесть вероятность ситуации, когда сотрудник забудет данные для входа, в таком случае требуется разработать логику напоминания или обновления пароля.

2.2. СТРУКТУРА БАЗЫ ДАННЫХ

База данных проекта состоит из 8 таблиц. Схема азы данных приведена на рисунке 2.

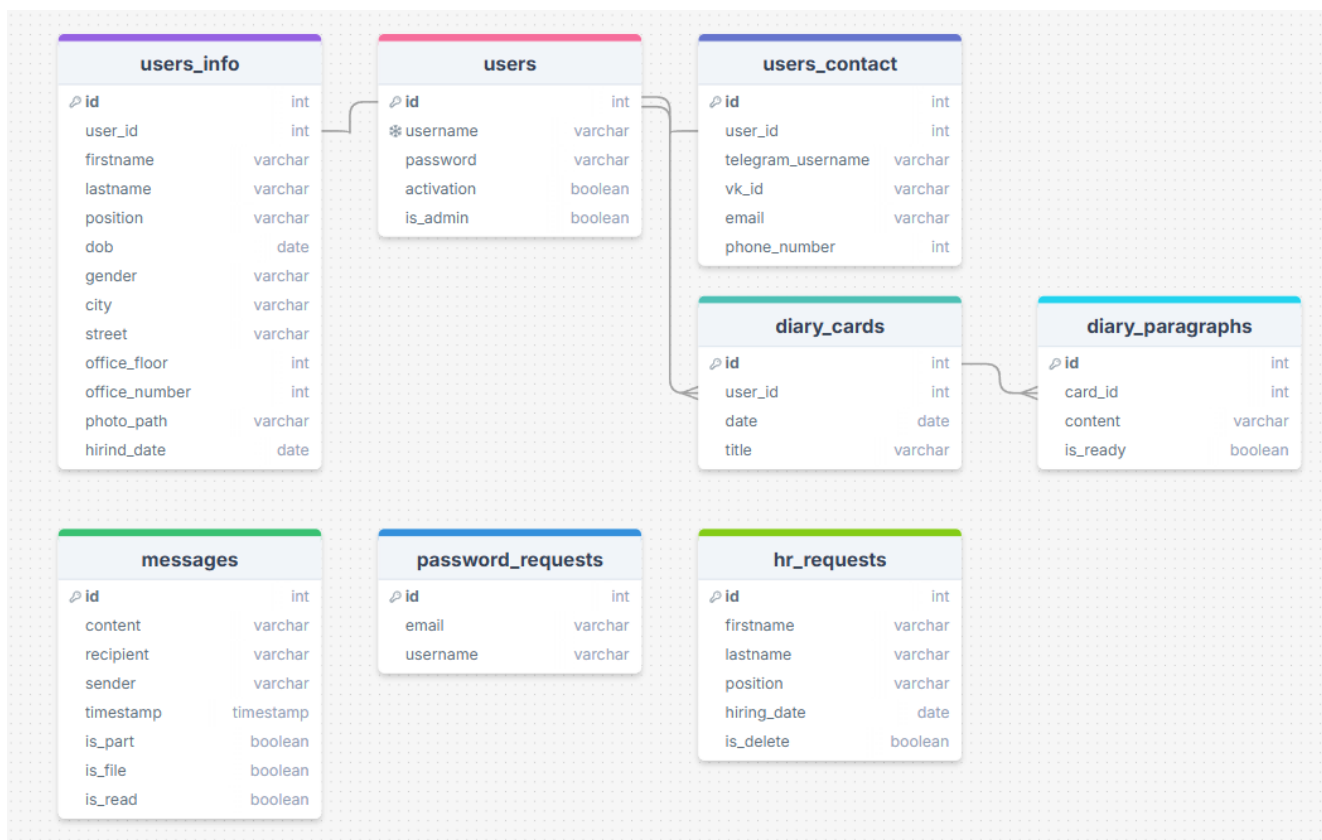


Рисунок 2 – ER-диаграмма.

Ключевой таблицей проекта является таблица users (см. таблицу 1), изменение, добавление и удаление строк в которой производится только администратором сайта. Имеет вид связи один к одному с таблицами 2 и 3, и один к многим с таблицей 7.

Таблица 1 – таблица users.

№	Столбец	Назначение
1	2	3
1	username	Юзернейм пользователя, является уникальным для каждого пользователя, не может быть null.

Продолжение таблицы 1.

1	2	3
2	password	Пароль пользователя, хранящийся в зашифрованном виде, не может быть null.
3	activation	Boolean значение несущее информацию об активации пользователя (false – пользователь еще не прошел активацию, true – прошел). По умолчанию – false.
4	is_admin	Boolean значение информирующее о наличии прав администратора для последующего доступа к административной панели. По умолчанию – false.

Таблицы 2 и 3 содержат дополнительные пользовательские данные, обязательные столбцы заполняются администратором в процессе регистрации пользователя (или могут быть частично заполнены отделом кадров, при отправке запроса на создание нового пользователя). Необязательные столбцы заполняются по желанию самого пользователя в процессе активации.

Таблица 2 – таблица users_info.

№	Столбец	Назначение
1	2	3
1	user_id	Реализует связь с таблицей users.
2	firstname	Имя сотрудника, не может быть null.
3	lastname	Фамилия сотрудника, не может быть null.
4	position	Должность сотрудника, не может быть null.
5	dob	Дата рождения сотрудника.
6	gender	Пол сотрудника.

Продолжение таблицы 2.

1	2	3
7	city	Точный адрес офиса/предприятия (город, улица, этаж, кабинет). Столбец city не может быть null.
8	street	
9	office_floor	
10	office_number	
11	photo_path	Ссылка на фотографию пользователя на сервере.
12	hiring_date	Дата найма.

Таблица 3 – таблица users_contact.

№	Столбец	Назначение
1	2	3
1	user_id	Реализует связь с таблицей users.
2	telegram_username	Данные для ссылок на соцсети (Telegram и VK).
3	vk_id	
4	email	Корпоративная рабочая почта, не может быть null.
5	phone_number	Рабочий номер телефона.

Таблица 4 хранит сообщения в чате.

Таблица 4 – таблица messages.

№	Столбец	Назначение
1	2	3
1	content	Содержание сообщения (текст или ссылка на файл).
2	recipient	Имя получателя.
3	sender	Имя отправителя.
4	timestamp	Время отправки сообщения.

Продолжение таблицы 4.

№	Столбец	Назначение
1	2	3
5	is_part	Boolean поле, по умолчанию имеет значение false, что говорит о том, что сообщение не превышает ограничение столбца content (255 символов) и не разделено на части.
6	is_file	Boolean поле, по умолчанию имеет значение false, говорит о том, что сообщение не является текстовым, а передает ссылку на файл.
7	is_read	Boolean поле, по умолчанию имеет значение false, сообщает о том, было ли прочитано сообщение получателем или нет.

Таблицы 5 и 6 содержат запросы для администратора сайта. Таблица 5 – запросы от отдела кадров на создание или удаление сотрудника, а таблица 6 – запросы на напоминание пароля от пользователей сайта.

Таблица 5 – таблица hr_requests.

№	Столбец	Назначение
1	firstname	Имя сотрудника.
2	lastname	Фамилия сотрудника.
3	position	Должность сотрудника.
4	hiring_date	Дата найма.
5	is_delete	Boolean поле, сообщающее о том, требуется добавить нового сотрудника или удалить.

Таблица 6 – таблица password_requests.

№	Столбец	Назначение
1	Email	Почта сотрудника.
2	Username	Юзернейм сотрудника.

Таблица 7 хранит все созданные пользователями записи в ежедневнике и имеет связь с таблицей 8 (хранящей пункты этих записей) один ко многим.

Таблица 7 – diary_cards.

№	Столбец	Назначение
1	user_id	Реализует связь с таблицей users.
2	date	Дата создания записи в ежедневнике.
3	title	Заголовок записи.

Таблица 8 – diary_paragraphs.

№	Столбец	Назначение
1	card_id	Реализует связь с таблицей diary_cards.
2	content	Текст пункта записи в ежедневнике.
3	is_ready	Boolean поле, сообщающее о том выполнена или не выполнена задача пункта.

2.3. СТРАНИЦА АВТОРИЗАЦИИ

На рисунке 3 можно увидеть вид страницы авторизации. На нем есть динамическая надпись, изменяющаяся в зависимости от времени суток (“добрый день” – днем, “добрый вечер” – вечером и т.д.).

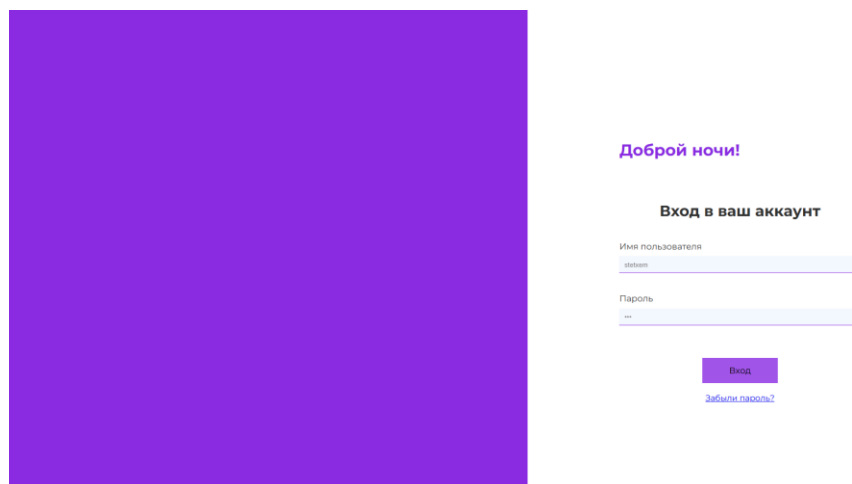


Рисунок 3 – Вид страницы авторизации.

После введения данных для входа они будут отправлены на сервер, для проверки на существование юзернейма и сверки хранящегося хэша пароля в базе данных и введенного. При неудаче будет выведено соответствующее сообщение (см. рисунок 4).

Неверный логин или пароль!

Доброй ночи!

Вход в ваш аккаунт

Имя пользователя

stetxem

Пароль

Вход

[Забыли пароль?](#)

Рисунок 4 – Неуспешный вход.

В случае если пользователь забыл пароль он может отправить запрос администратору на его восстановление. Восстановить пароль можно по юзернейму (см. рисунок 5) или почте (см. рисунок 6). Если пользователь с такими данными существует – запрос будет успешно создан.

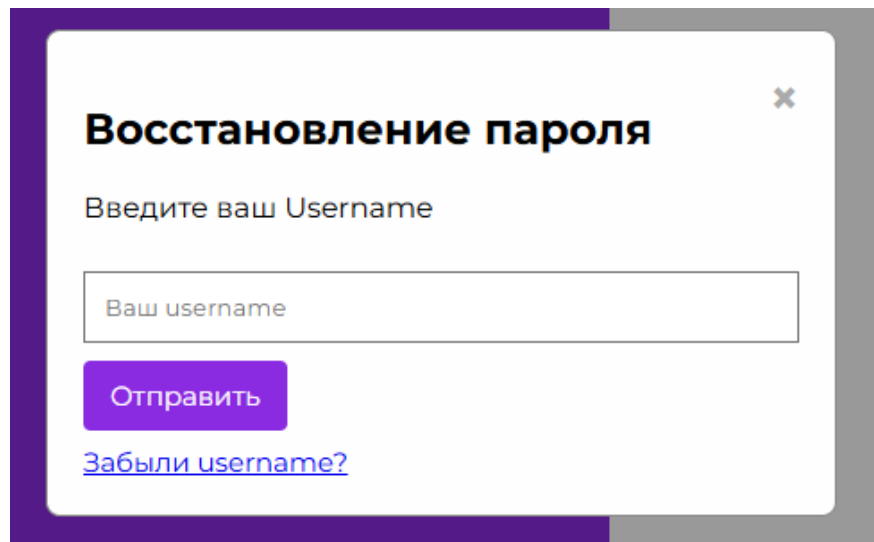


Рисунок 5 – Восстановление пароля по юзернейму.

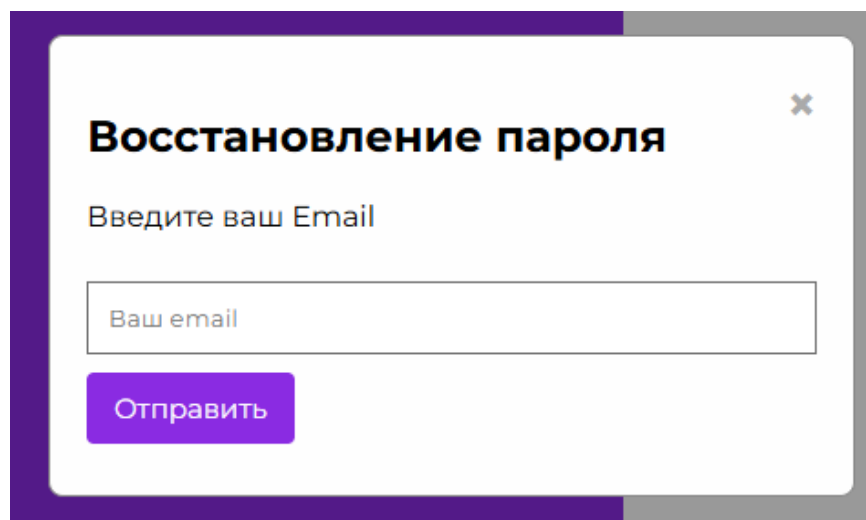


Рисунок 6 – Восстановление пароля по почте.

Пользователь так же будет осведомлен о результате запроса на восстановление пароля (см. рисунки 7 и 8).

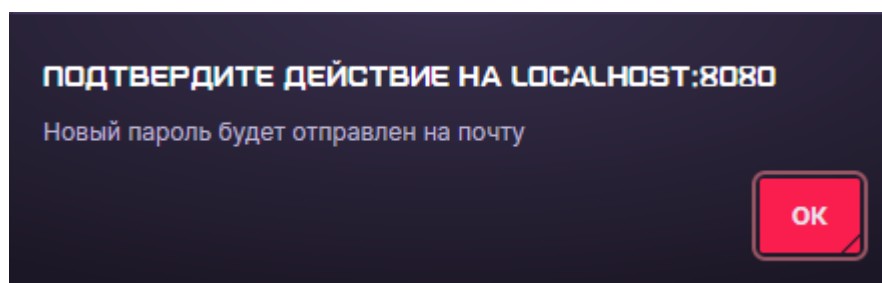


Рисунок 7 – Успешный запрос.

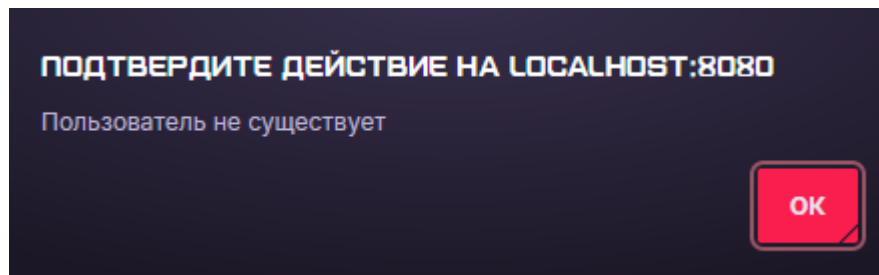


Рисунок 8 – Неуспешный запрос.

При попытке ввести адрес другой страницы, неавторизованному пользователю будет отказано в доступе и откроется страница авторизации.

2.4. СТРАНИЦА АКТИВАЦИИ

При первом входе пользователь будет переведен на страницу активации, где он может заполнить дополнительную информацию о себе и загрузить фото профиля. При указании данных соцсетей можно ввести как и сразу ссылку, так и просто указать id имя пользователя, ссылка сгенерируется автоматически на серверной стороне.

Основная информация

Имя*

Фамилия*

Должность*

Год рождения

Пол

Информация о рабочем месте

Город*

Улица

Этаж

Кабинет

Контактные данные

Номер телефона

Почта*

ID ВК

Имя пользователя Телеграм




Рисунок 9 – Вид страницы активации.

Все обязательные поля, кроме города, уже будут заполнены администратором сайта или предоставлены отделом кадров. При попытке отправки формы с незаполненными обязательными полями будет выведено предупреждение (см. рисунок 10).

Город*

Улица

Этаж

 Заполните это поле.

Рисунок 10 – Предупреждение при незаполненном обязательном поле.

Пользователь, не прошедший активацию, не имеет доступа к другим страницам.

2.5. СТРАНИЦА ПРОФИЛЯ

На данную страницу пользователь попадает после успешной авторизации (если статус активации true) или активации профиля.

Если выводится профиль текущего пользователя – ему будет доступна функция редактирования личных данных (см. рисунок 11). При редактировании он будет переведен на страницу активации. В случае если просматривается профиль другого сотрудника редактирование будет недоступно (см. рисунок 12).

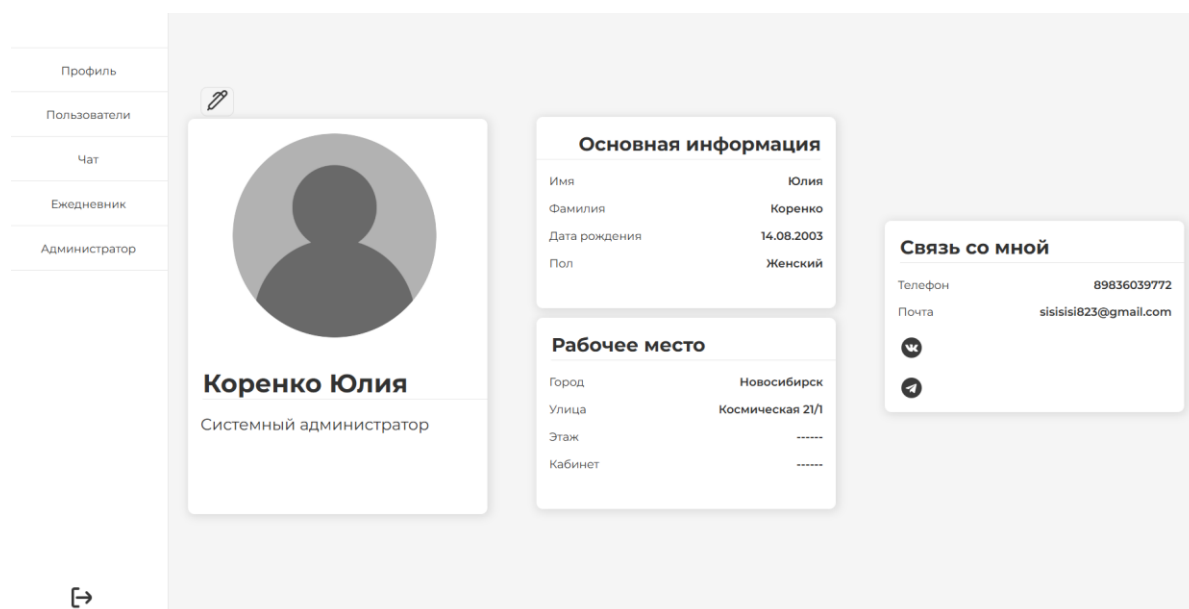


Рисунок 11 – Вид страницы профиля.

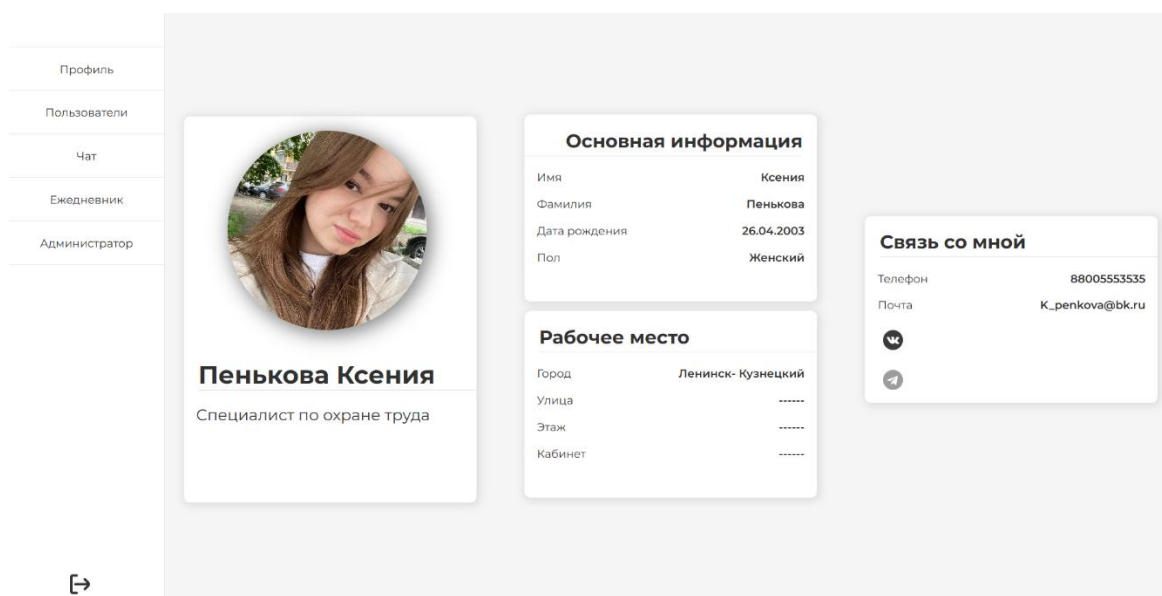


Рисунок 12 – Вид страницы профиля другого пользователя.

Для отсутствующих данных будет выведено “-----”, в случае ссылок – они будут неактивны.

2.6. СТРАНИЦА ПОЛЬЗОВАТЕЛЕЙ

На данной странице выводятся все пользователи, зарегистрированные в системе и имеющие активированный аккаунт (см. рисунок 13). Для нахождения нужного сотрудника можно воспользоваться или простым поиском по имени и фамилии (см. рисунок 14) или детальным поиском с фильтрами (см. рисунок 15).

Доступные города и должности для поиска берутся из существующих в базе данных, поле “В сети” отфильтрует только тех пользователей, что на данный момент находятся в сети.

Для каждого пользователя так же выводятся ссылки на их профиль и кнопка, открывающая диалог с ними.

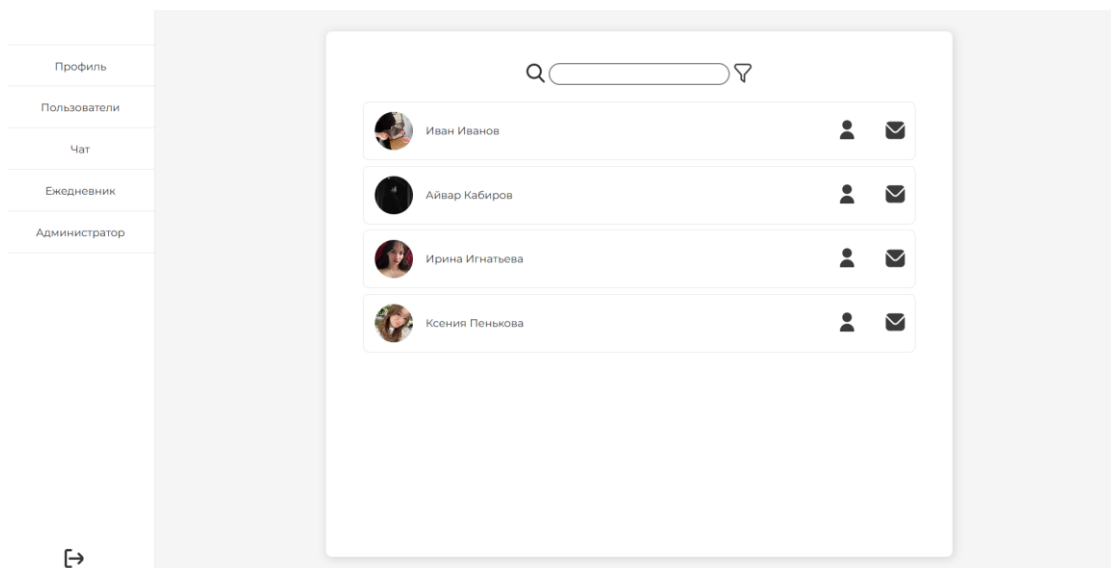


Рисунок 13 – Вид страницы пользователей.

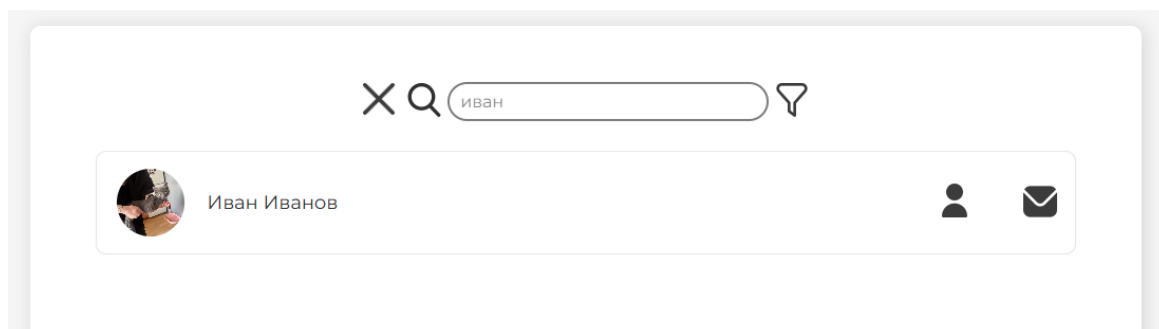


Рисунок 14 – Пример успешного поиска.

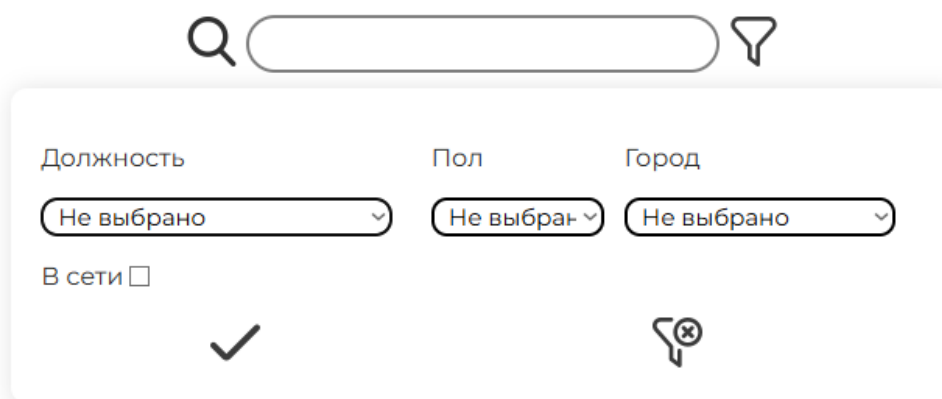


Рисунок 15 – Окно поиска с фильтрами.

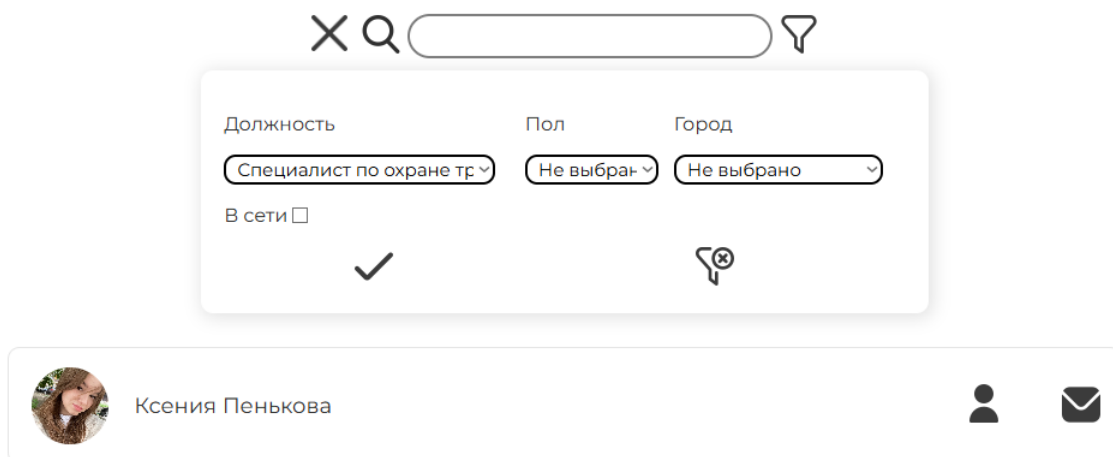


Рисунок 16 – Пример успешного поиска с фильтрами.

При неуспешном поиске будет выведено соответствующее сообщение. При нажатии на сброс поиска или при нажатии клавиши “Enter” с пустым полем поиска список будет приведен в исходное состояние.

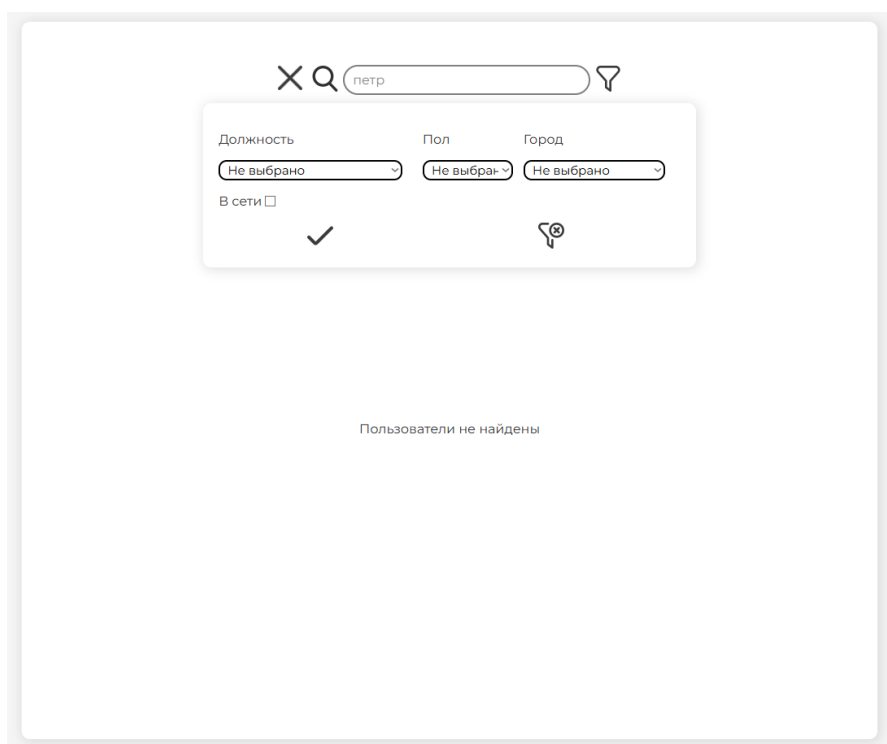


Рисунок 17 – Пример неуспешного поиска.

2.7. СТРАНИЦА ЧАТА

На страницу чата можно попасть как и через меню, находящееся слева, так и через страницу пользователей. Окно чата разделено на две основные области – список пользователей и сам чат. В списке пользователей выводятся только те, с кем существует переписка, их последние сообщения, отправитель этого сообщения и его статус. По списку можно произвести простой поиск по имени.

На рисунке 18 можно увидеть, что диалог можно закрыть. Вид окна чата при закрытых диалогах отображен на рисунке 19.

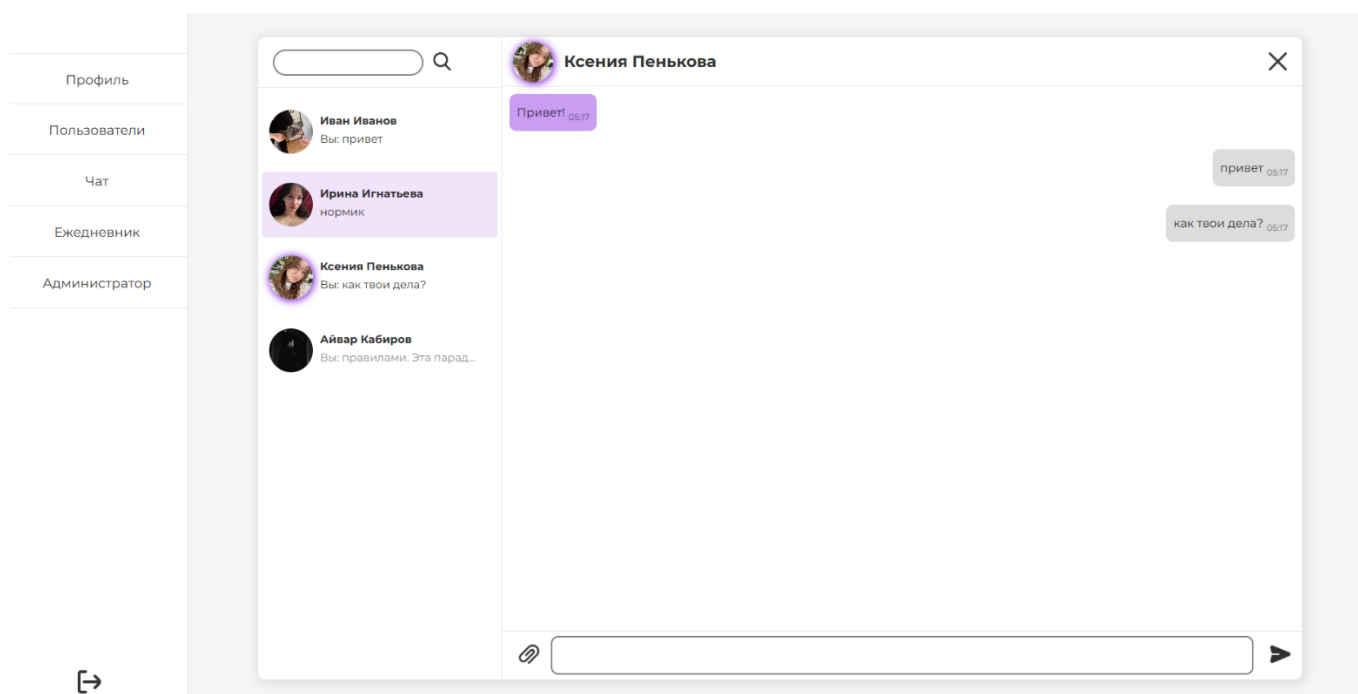


Рисунок 18 – Вид страницы чата.

Стоит обратить внимание на последнее сообщение в диалоге. По его виду можно определить отправителя – если сообщение отправлено текущим пользователем, оно будет начинаться с “Вы:...”.

Так же виден статус этого сообщения. Если отправителем является другой пользователь и текущий его еще не прочитал – сообщение будет выделено жирным шрифтом, в обратном случае (отправитель – текущий пользователь, а получатель не прочитал сообщение) сообщение будет серым. В диалоге вид непрочитанного

сообщения можно увидеть на рисунке 20. При открытии диалога получателем все сообщения помечаются как прочитанные, если при получении сообщения пользователь уже находится в диалоге – на сервер тут же отправляется информация, что сообщение является прочитанным.

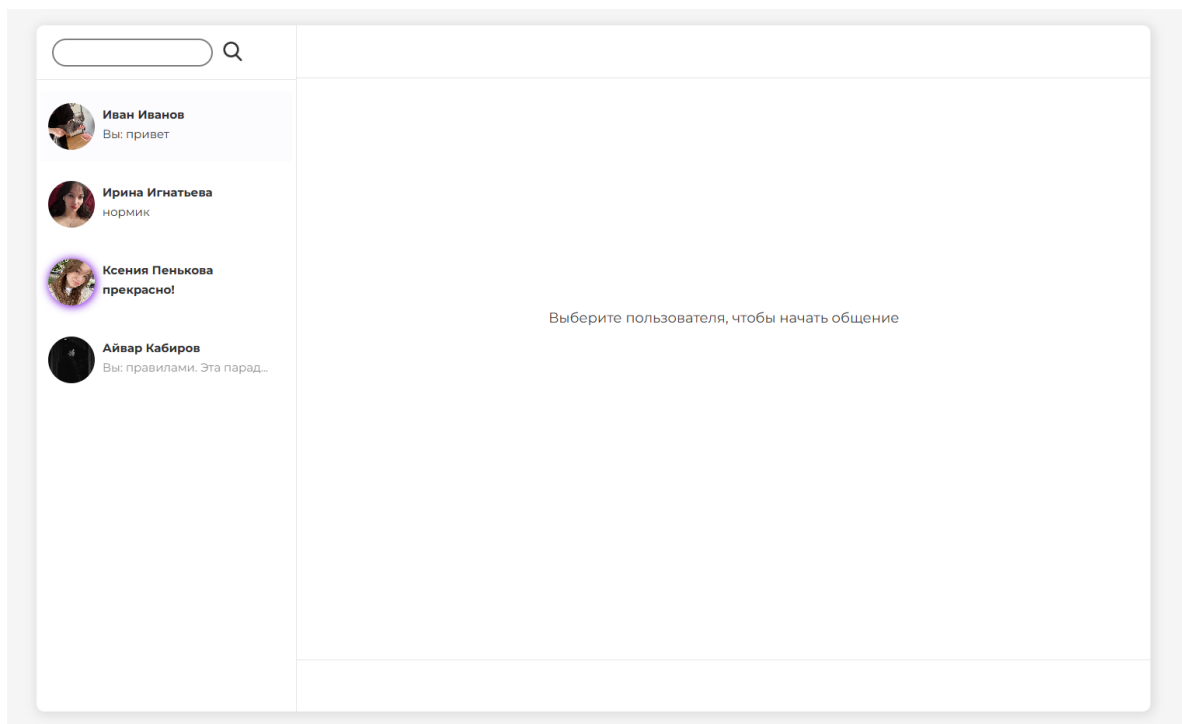


Рисунок 19 – Окно чата с закрытым диалогом.



Рисунок 20 – Вид непрочитанного сообщения в диалоге.

Так как размер хранимого сообщения имеет ограничение, то текстовые сообщения, превышающие размер в 255 символов будут разделены на несколько частей на сервере. Пример такого сообщения можно увидеть на рисунке 21.

dolor sit amet, consectetur adipiscing elit. Aenean
commodo ligula eget dolor. Aenean massa. Cum sociis
natoque penatibus et magnis dis parturient montes,
nascetur ridiculus mus. Donec quam felis, ultricies nec,
pellentesque eu, pretium quis, sem. Nulla

05:34

consequat massa quis enim. Donec pede justo, fringilla
vel, aliquet nec, vulputate eget, arcu. In enim justo,
rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam
dictum felis eu pede mollis pretium. Integer tincidunt.
Cras dapibus. Vivamus elementum

05:34

semper nisi. Aenean vulputate eleifend tellus. Aenean
leo ligula, porttitor eu, consequat vitae, eleifend ac,
enim. Aliquam lorem ante, dapibus in, viverra quis,
feugiat a, tellus. Phasellus viverra nulla ut metus varius
laoreet. Quisque rutrum. Aenean

05:34

imperdiet. Etiam ultricies nisi vel augue. Curabitur
ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus.
Maecenas tempus, tellus eget condimentum rhoncus,
sem quam semper libero, sit amet adipiscing sem
neque sed ipsum. Nam quam nunc, blandit vel,

05:34

Рисунок 21 – Пример разделенного сообщения.

При отправке файла он будет сохранен на сервере, а в диалоге появится сообщение с ссылкой для скачивания этого файла (см. рисунок 22).

Файл: [отчет.pdf](#) 05:37

Рисунок 22 – Пример сообщения с файлом.

Пользователь так же имеет возможность изменять и удалять свои сообщения. При наведении на нужное сообщение всплывет соответствующее меню (см. рисунок 23). Новое сообщение в диалоге и все действия с ним (редактирование и удаление) тут же отображаются в чате без перезагрузки страницы.

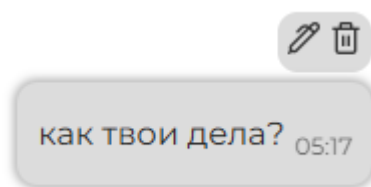


Рисунок 23 – Меню сообщения.

При наведении курсора на сообщение видна и другая полезная информация – дата отправления сообщения (см. рисунок 24).

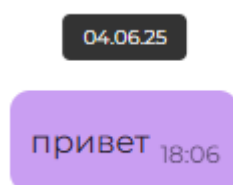


Рисунок 24 –Дата отправки сообщения.

В процессе общения можно увидеть, набирается ли сообщение другим пользователем (см. рисунок 25).

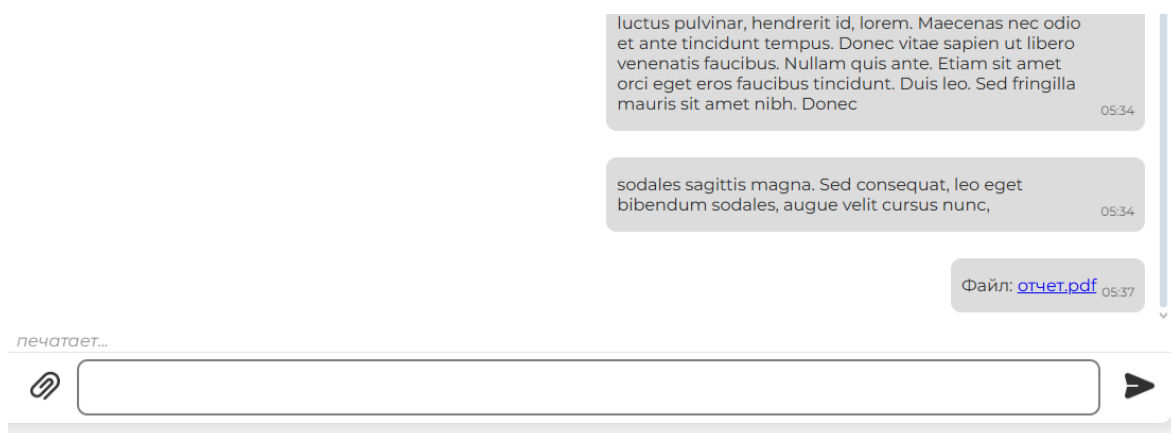


Рисунок 25 – Оповещение о том, что другой пользователь набирает сообщение.

Стоит отметить и то, что страница сохраняет свое состояние. То есть, при переходе на другие страницы и последующее возвращение на текущую или при обновлении будет выведен тот диалог, который был открыт на момент закрытия страницы или перезагрузки (если диалог не был открыт состояние так же сохранится).

2.8. СТРАНИЦА ЕЖЕДНЕВНИКА

Ежедневник реализован в виде слайдера, управляемого или кнопками в интерфейсе или клавишами (см. рисунок 26).

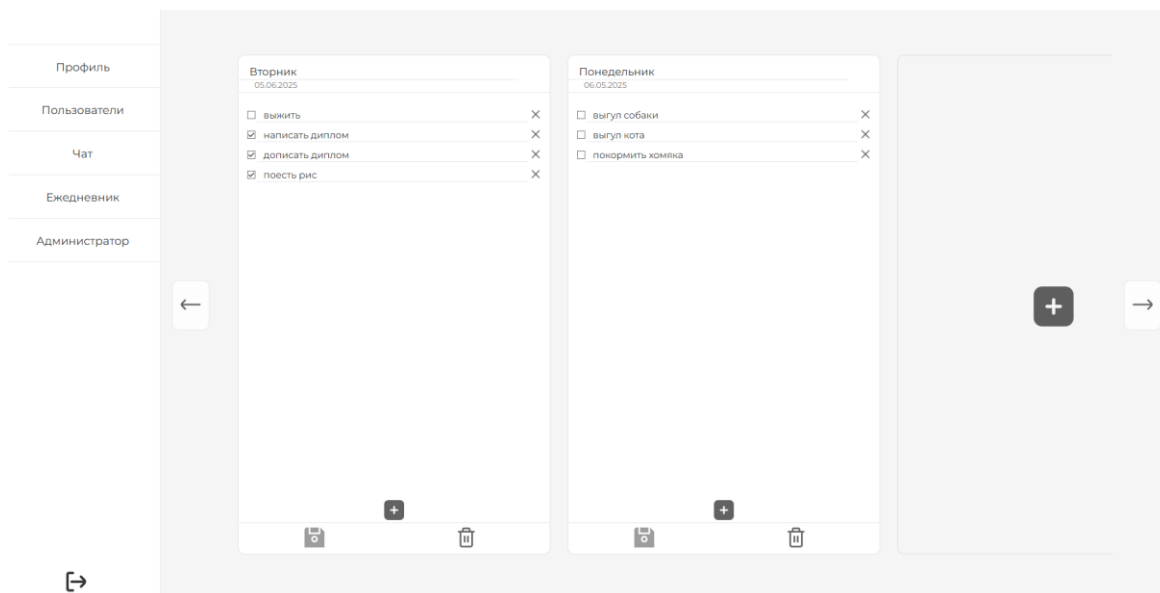


Рисунок 26 – Вид страницы слайдера.

При создании новой записи будет автоматически вставлена дата и кнопка сохранения сразу отобразится активной. В существующих записях кнопка неактивна до тех пор, пока не будут внесены изменения в запись, такие как изменения заголовка, изменение пунктов или их статуса, удаление пункта.

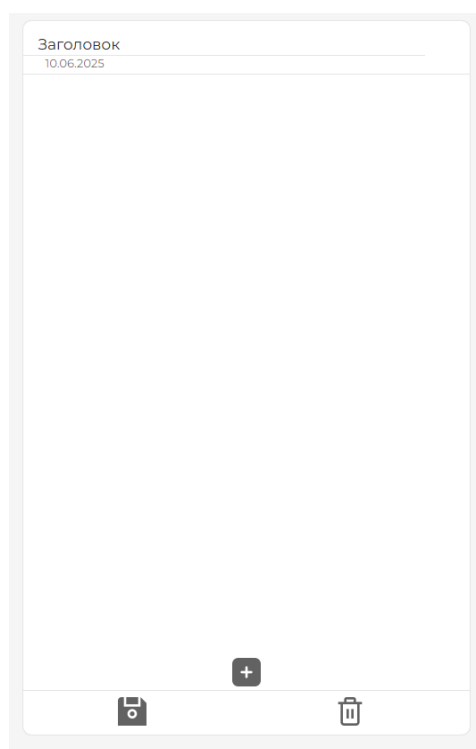


Рисунок 27 – Новая запись в ежедневнике.

Новые пункты добавляются по соответствующей кнопке в конце карточки и имеют ограничение в 16 пунктов на запись, количество карточек так же ограничено – 10 одновременно существующих карточек. При попытке создать новые пункты или записи по достижению лимита на странице появятся сообщения, приведенные на рисунках 28 и 29.

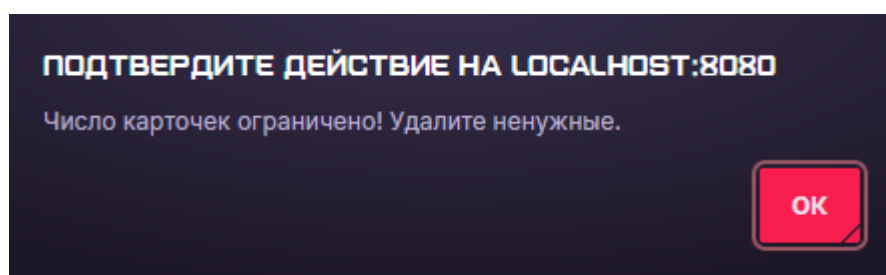


Рисунок 28 – Предупреждение о достижении лимита записей.

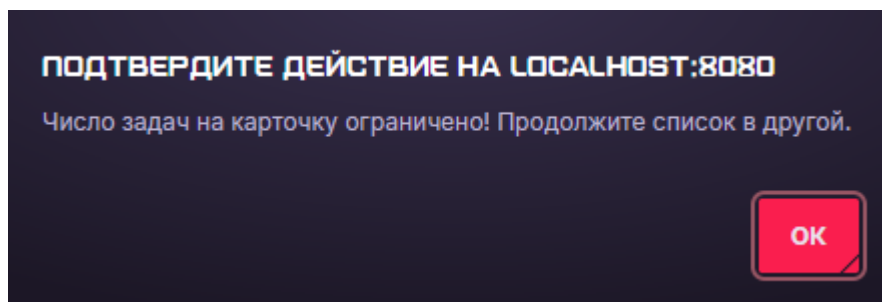


Рисунок 29 – Предупреждение о достижении лимита пунктов.

Удаление записи требует подтверждения (см. рисунок 30).

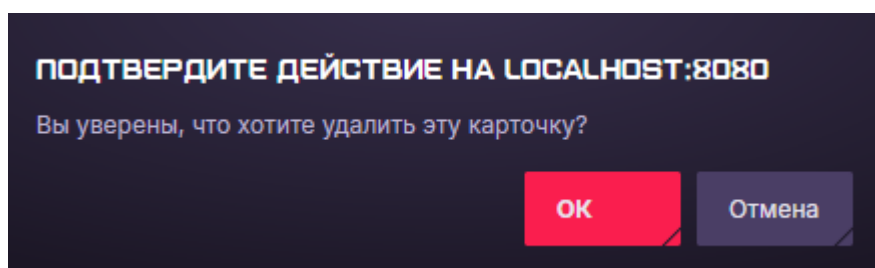


Рисунок 30 – Подтверждение удаления записи в ежедневнике.

При сохранении карточки с пустыми пунктами сохранится только заголовок (если он есть) и дата создания записи.

2.9. СТРАНИЦА АДМИНИСТРАТОРА

К странице администратора имеют доступ только пользователи с соответствующими правами. Страница имеет три раздела: пользователи (см. рисунок 31), запросы от отдела кадров (см. рисунок 32), запросы на смену пароля (см. рисунок 33).

В разделе пользователей выведены все существующие пользователи. Для каждого выводятся id, юзернейм, имя, должность, статус активации и наличие прав администратора (такие пользователи будут помечены жирным шрифтом). В этом разделе администратор совершает управление пользователями сайта – он может просмотреть профили (если аккаунт активирован), удалить пользователя или создать нового. Так же он может найти нужного пользователя по имени, юзернейму или статусу активации.

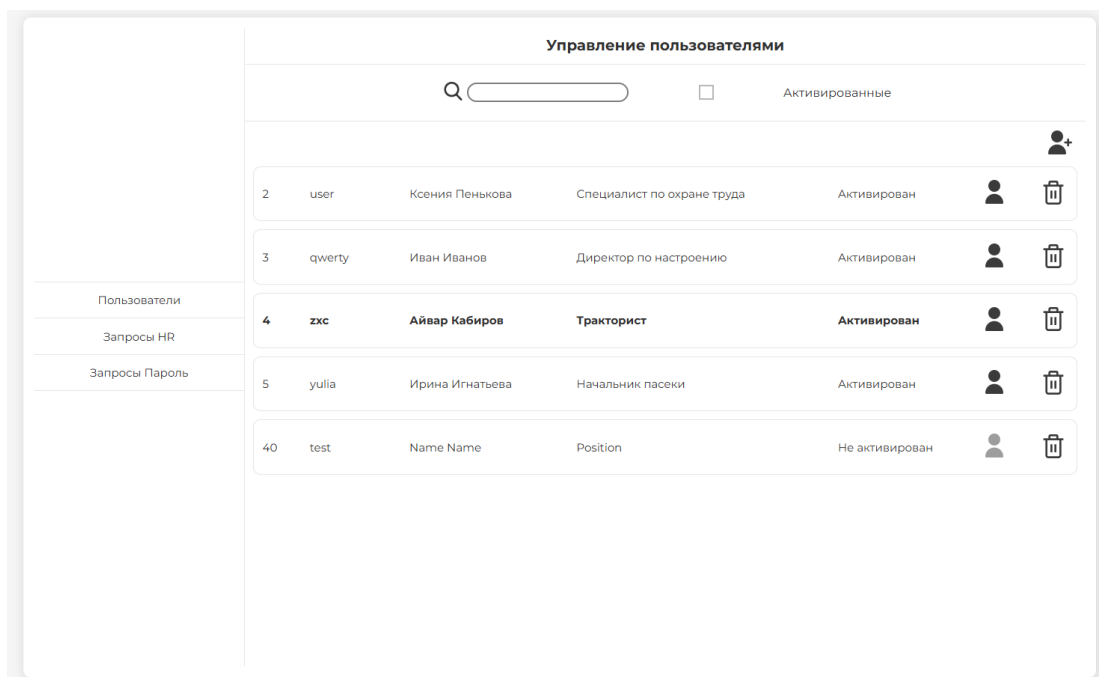


Рисунок 31 – Вид административной панели, раздел пользователей.

Во втором разделе все запросы от отдела кадров разделены на две категории – создание или удаление. Администратор имеет право как подтвердить запрос, так и отклонить его. В случае удаления пользователь будет тут же удален, в случае создания – откроется окно создания пользователя.

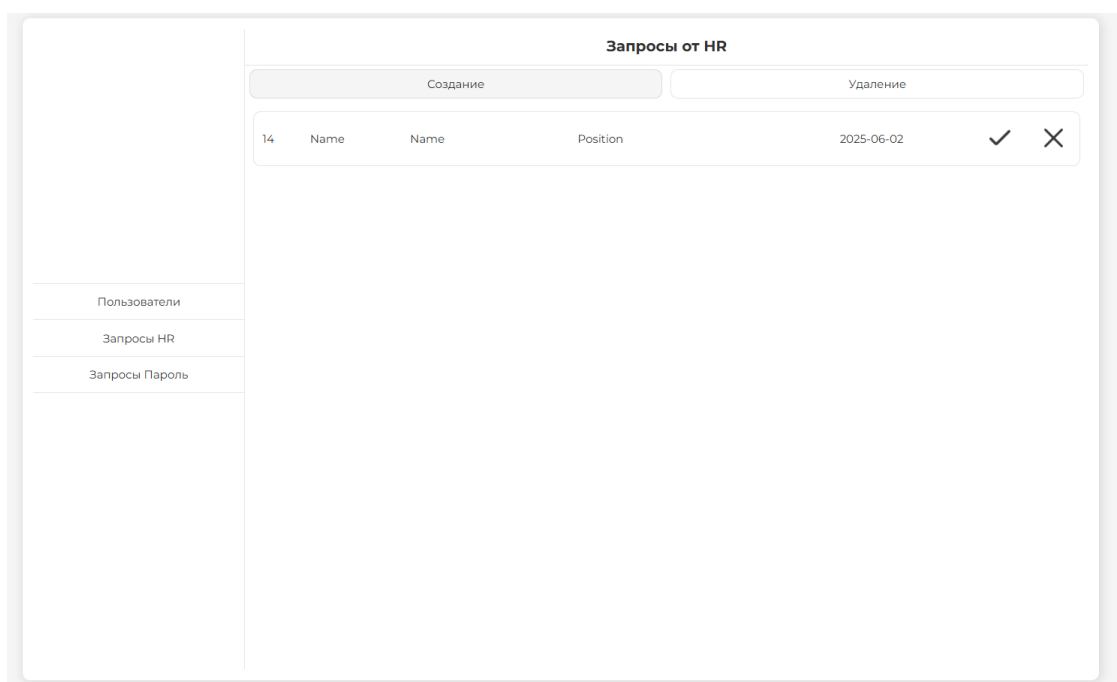


Рисунок 32 – Вид административной панели, раздел запросов от отдела кадров.

В разделе запросов на смену пароля выводятся непосредственно эти запросы. Как и в случае с запросами, присылаемыми отделом кадров, администратор имеет право подтверждать и отклонять. И при подтверждении, и при отклонении на почту пользователя будет отправлено письмо (см. рисунки 34 и 35).

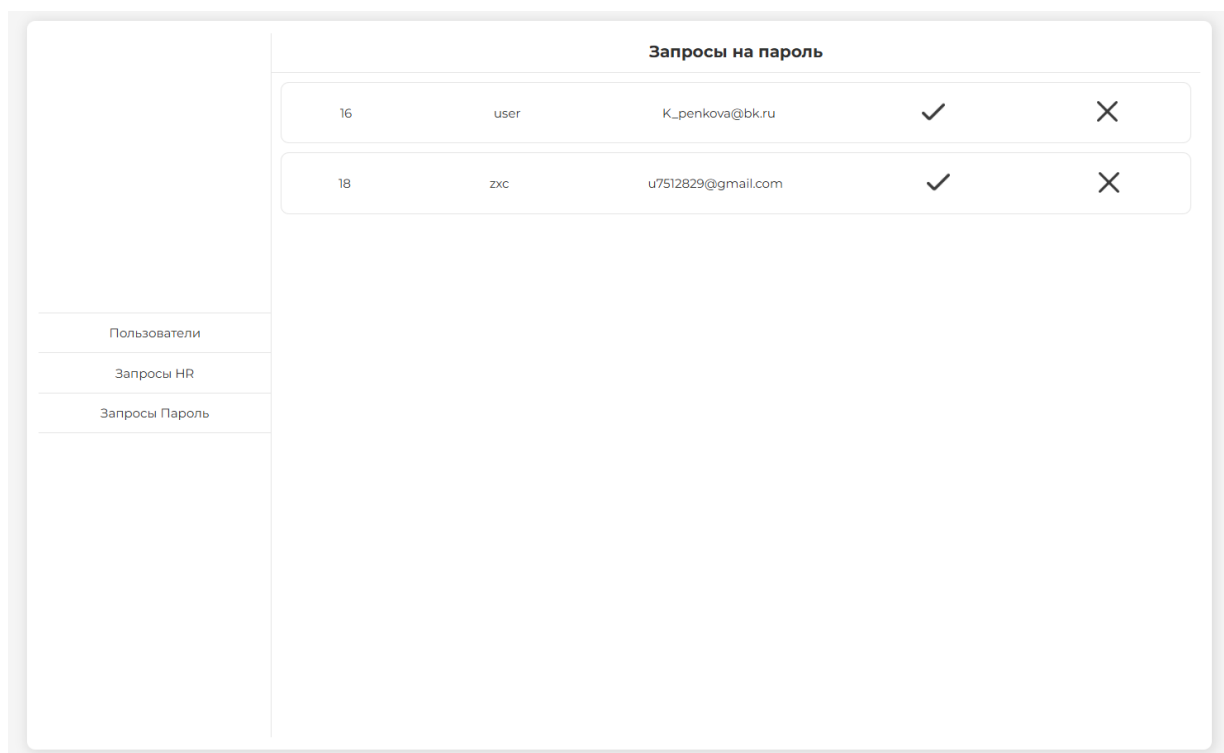


Рисунок 33 – Вид административной панели, раздел запросов на смену пароля.

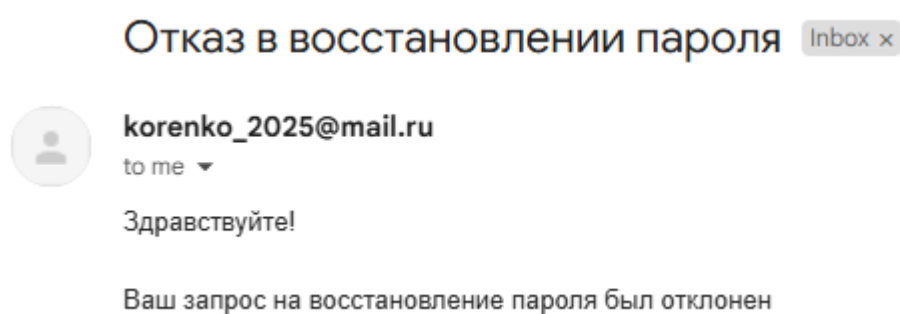


Рисунок 34 – Письмо при отклонении запроса на восстановление пароля.

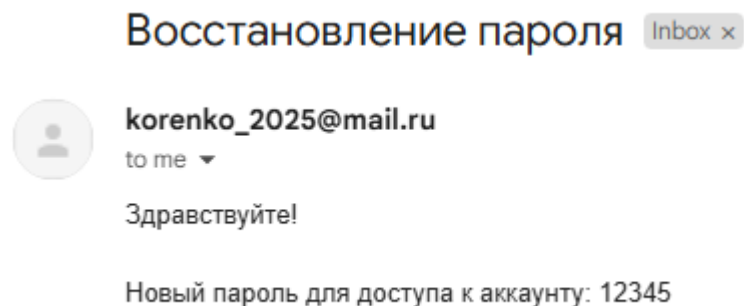


Рисунок 35 – Письмо при подтверждении запроса на восстановление пароля.

Так же, при подтверждении запроса администратору требуется ввести новый пароль (см. рисунок 36).

The image shows a form for password recovery. It is enclosed in a light grey rounded rectangle. Inside, the word 'Пароль' (Password) is followed by a text input field. Below the input field is a button with a checkmark icon, indicating a confirmation or 'next' step.

Рисунок 36 – Форма для ввода нового пароля.

При создании нового пользователя форма будет несколько отличаться в зависимости от того, создается ли он по запросу отдела кадров или же самостоятельно. В первом случае некоторые поля уже будут заполнены (см. рисунок 37). Перед записью в базу данных отправляемая форма так же будет проверена на то, существует ли уже такой пользователь или нет.

Создание пользователя


Юзернейм	<input type="text"/>
Пароль	<input type="password"/>
Почта	<input type="text"/>
Имя	<input type="text" value="Name"/>
Фамилия	<input type="text" value="Name"/>
Должность	<input type="text" value="Position"/>
Дата найма	<input type="text" value="02.06.2025"/> 
Права администратора	<input type="checkbox"/>
<input type="button" value="✓"/>	

Рисунок 37 – Создание нового пользователя по запросу.

2.10. СТРАНИЦА HR

Доступ к странице отдела кадров осуществляется по IP. На нее допускаются только компьютеры, принадлежащие к конкретной подсети, авторизация не требуется. На рисунке 38 видно, что отдел кадров может послать запрос на добавление нового сотрудника или удаление существующего. Сами формы имеют идентичный вид (см. рисунки 39 и 40).

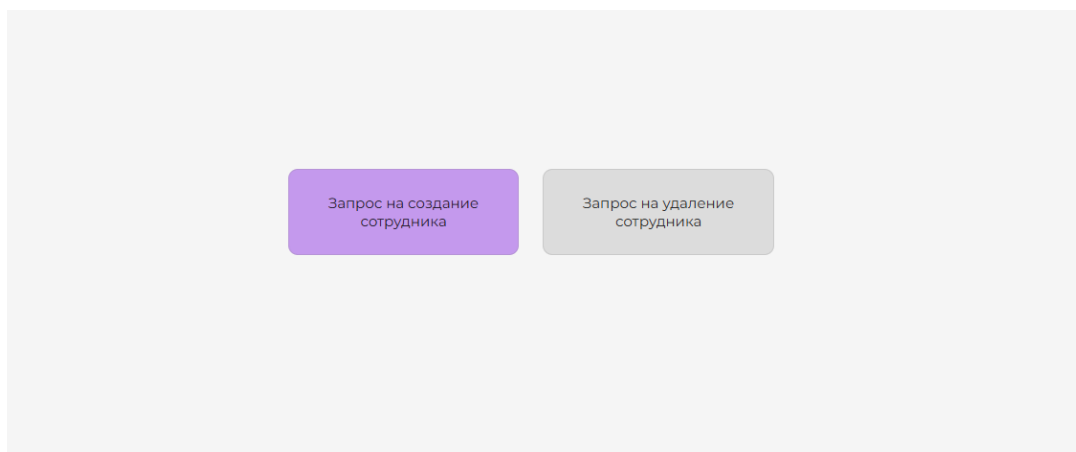
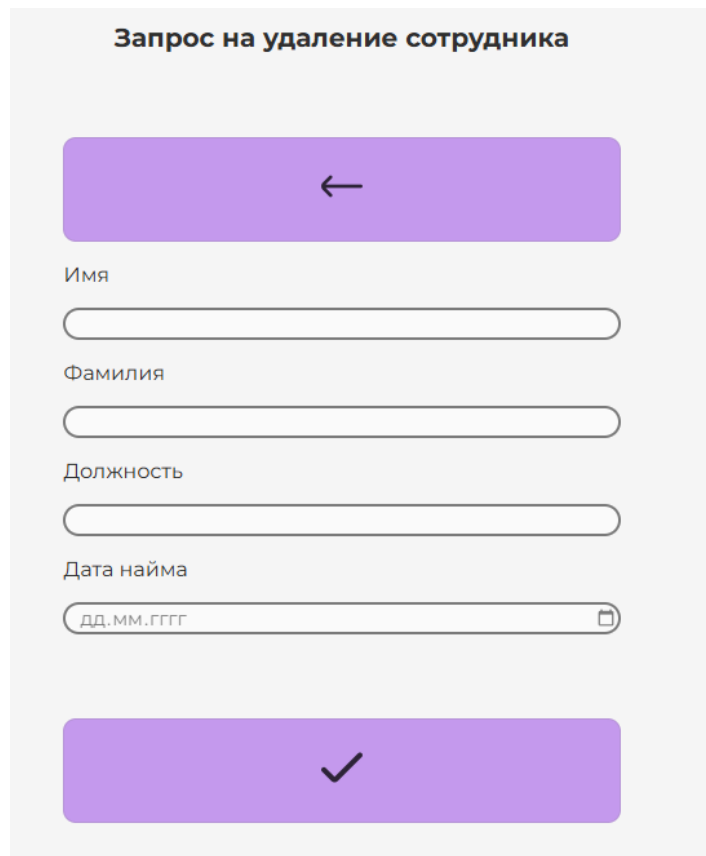


Рисунок 38 – Вид страницы для отдела кадров.

The image shows a form titled 'Запрос на создание сотрудника' (Request for employee creation) in bold black text. Below the title is a purple button with a white left-pointing arrow. Underneath are four input fields: 'Имя' (Name), 'Фамилия' (Surname), 'Должность' (Position), and 'Дата найма' (Hire date). The 'Дата найма' field has a date format placeholder 'дд.мм.гггг' and a calendar icon. At the bottom is a purple button with a white checkmark.

Рисунок 39 – Форма для запроса на создание пользователя.



The form is titled "Запрос на удаление сотрудника" (Request for employee deletion). It features a purple header bar with a left-pointing arrow icon. Below the header are four input fields: "Имя" (Name), "Фамилия" (Surname), "Должность" (Position), and "Дата найма" (Hire date). The "Дата найма" field includes a date picker icon. At the bottom is a purple bar with a checkmark icon.

Рисунок 40 – Форма для запроса на удаление пользователя.

В каждом разделе перед отправкой запроса проводится проверка на существование пользователя. Если в ходе создания запроса на нового сотрудника сервер вернет положительный ответ на существование, то запрос не будет создан (см. рисунок 41). В случае удаления сотрудника ситуация противоположная (см. рисунок 42), т.к. удалить возможно только существующего пользователя. Успешное создание запроса так же подтверждается (см. рисунок 43).

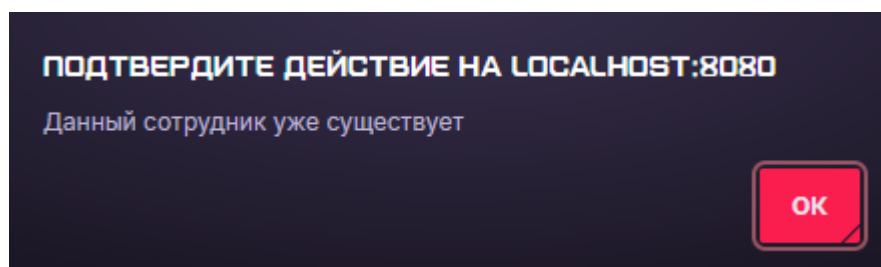


Рисунок 41 – Предупреждение в случае, если создаваемый пользователь уже существует.

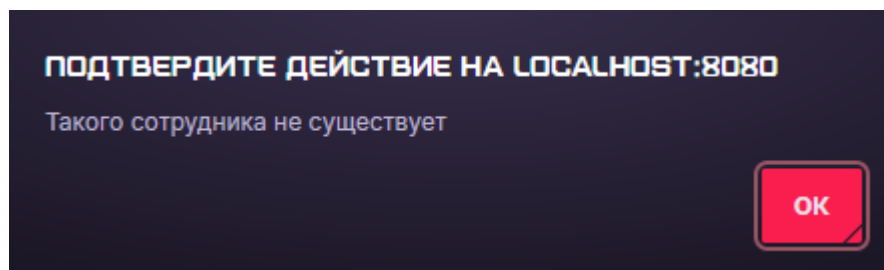


Рисунок 42 – Предупреждение в случае, если удаляемый пользователь не существует.

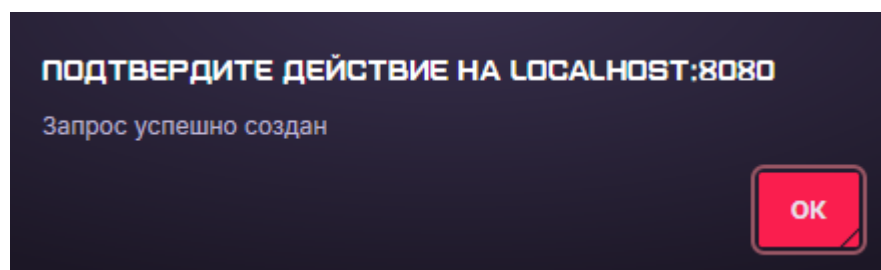


Рисунок 43 – Оповещение об успешно созданном запросе.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было разработано веб-приложение с социальными сервисами для сотрудников на языке Java с использованием фреймворка Spring.

Разработанный проект представляет собой готовое решение, которое может быть внедрено в корпоративную среду для повышения эффективности взаимодействия между сотрудниками.

В перспективе может быть расширен путем добавление таких функций как групповые чаты, видеозвонки и таких разделов, как новостной внутрикорпоративный блог, многопользовательские игры.

СПИСОК ЛИТЕРАТУРЫ

1. Официальная документация Spring Framework [Электронный ресурс]. - Режим доступа: <https://spring.io/projects/spring-framework> (дата 07.04.2025).
2. Официальная документация Spring Boot [Электронный ресурс]. – Режим доступа: <https://spring.io/projects/spring-boot> (дата обращения: 07.04.2025).
3. Руководство по Spring Security [Электронный ресурс]. - Режим доступа: <https://docs.spring.io/spring-security/reference/> (дата 07.04.2025).
4. Официальная документация Thymeleaf [Электронный ресурс]. - Режим доступа: <https://www.thymeleaf.org/documentation.html> (дата 07.04.2025).
5. Руководство по PostgreSQL [Электронный ресурс]. – Режим доступа: <https://postgrespro.ru/docs> (дата обращения: 15.04.2025).
6. Spring WebSocket: документация [Электронный ресурс]. – Режим доступа: <https://docs.spring.io/spring-framework/reference/web/websocket.html> (дата обращения: 22.04.2025).
7. WebSocket API: спецификация MDN [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/API/WebSockets_API (дата обращения: 24.04.2025).
8. Документация по JavaScript (MDN Web Docs) [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата обращения: 24.04.2025).
9. Флэнаган, Д. JavaScript: Полное руководство / Д. Флэнаган. – 7-е изд. – СПб.: Символ-Плюс, 2022. – 1088 с.
10. Вагин, Д. В., Петров, Р. В. Современные технологии разработки веб-приложений / Вагин, Д. В., Петров, Р. В. – Новосибирск: Изд-во НГТУ, 2019. – 52 с.

ПРИЛОЖЕНИЕ 1. СЕРВЕРНАЯ ЧАСТЬ

Файл «ChatController»

@Controller

```
public class ChatController {
```

```
    @Autowired
```

```
    private MessageService messageService;
```

```
    @Autowired
```

```
    private SimpMessagingTemplate messagingTemplate;
```

```
    @Autowired
```

```
    MyUserService userService;
```

```
    @PostMapping("/handleMessage")
```

```
    public void handleMessage(@Payload MessageDTO message, Principal principal) {
```

```
        String sender = principal.getName();
```

```
        String recipientUS = message.getRecipient();
```

```
        List<String> participantUsernames =
messageService.findChatParticipants(recipientUS);
        boolean existChat = participantUsernames.contains(sender);
        System.out.println(existChat);
        if (!existChat) {
            System.out.println("Create new dialog");
            UsersDto profile = userService.getUserProfile(sender);
            messagingTemplate.convertAndSendToUser(
                recipientUS,
                "/queue/newDialog",
                profile
            );
        }

        if (message.getContent().length() > 255) {
            List<MessageEntity> messageParts =
messageService.saveLongMessageToDatabase(
                sender,
                recipientUS,
                message.getContent()
            );

            for (MessageEntity part : messageParts) {
                messagingTemplate.convertAndSendToUser(
                    recipientUS,
                    "/queue/newMessages",
                    part
                );
                messagingTemplate.convertAndSendToUser(
                    sender,
                    "/queue/newMessages",
                    part
                );
            }
        } else {

            MessageEntity savedMessage = messageService.saveToDatabase(
                sender,
                recipientUS,
                message.getContent(),
```

```

        false
    );
    messagingTemplate.convertAndSendToUser(
        sender,
        "/queue/newMessages",
        savedMessage
    );

    messagingTemplate.convertAndSendToUser(
        recipientUS,
        "/queue/newMessages",
        savedMessage
    );
}

}

@GetMapping("/requestMessages")
public void getMessages(@Payload MessageDTO request, Principal principal) {
    String sender = principal.getName();
    String recipient = request.getOtherUser();

    messagingTemplate.convertAndSendToUser(
        sender,
        "/queue/messages",
        messageService.getMessagesBetweenUsers(sender, recipient)
    );
}

@PostMapping("/upload")
public ResponseEntity<?> handleFileUpload(@RequestParam("file") MultipartFile
file,
                                           @RequestParam("recipient") String
recipient,
                                           Principal principal) {
    try {
        String sender = principal.getName();
        String uploadDir = "uploads/";
        File uploadPath = new File(uploadDir);
        if (!uploadPath.exists()) {
            uploadPath.mkdirs();
        }

        String fileName = UUID.randomUUID().toString() + "_" +
file.getOriginalFilename();
        Path filePath = Paths.get(uploadDir + fileName);
        Files.copy(file.getInputStream(), filePath,
StandardCopyOption.REPLACE_EXISTING);

        String fileLink = "/download/" + fileName;
        String messageContent = "Файл: <a href=\"" + fileLink + "\" download>"
+ file.getOriginalFilename() + "</a>";

        MessageEntity savedMessage = messageService.saveToDatabase(
            sender,
            recipient,
            messageContent,
            true
        );

        messagingTemplate.convertAndSendToUser(
            recipient,
            "/queue/newMessages",
            savedMessage
        );
    }
}

```

```

        messagingTemplate.convertAndSendToUser(
            sender,
            "/queue/newMessages",
            savedMessage
        );

        return ResponseEntity.ok(Map.of(
            "success", true,
            "filePath", fileLink,
            "message", savedMessage
        ));
    } catch (Exception e) {
        return ResponseEntity.status(500).body(Map.of("success", false));
    }
}

@GetMapping("/download/{filename:.+}")
public ResponseEntity<Resource> downloadFile(@PathVariable String filename) {
    try {
        Path filePath = Paths.get("uploads/" + filename);
        Resource resource = new UrlResource(filePath.toUri());

        return ResponseEntity.ok()
            .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + resource.getFilename() + "\"")
            .body(resource);
    } catch (Exception e) {
        return ResponseEntity.notFound().build();
    }
}

@MessageMapping("/deleteMessage")
public void deleteMessage(@Payload Integer id) {
    MessageEntity message = messageService.getMessageById(id);
    messageService.deleteMessage(id);

    messagingTemplate.convertAndSendToUser(
        message.getRecipient(),
        "/queue/messages",
        messageService.getMessagesBetweenUsers(message.getSender(),
message.getRecipient())
    );
}

@MessageMapping("/updateMessage")
public void updateMessage (@Payload Map<String, Object> message){
    Integer id = (Integer) message.get("id");
    String content = (String) message.get("content");
    messageService.updateMessage(id, content);
    MessageEntity mess = messageService.getMessageById(id);

    messagingTemplate.convertAndSendToUser(
        mess.getRecipient(),
        "/queue/messages",
        messageService.getMessagesBetweenUsers(mess.getSender(),
mess.getRecipient())
    );
}

@MessageMapping("/requestAllLastMessages")
@SendToUser("/queue/allLastMessages")
public Map<String, LastMessageDTO> getAllLastMessages(@Payload String
currentUser) {

```

```

        return messageService.getLastMessagesForUser(currentUser);
    }

    @PostMapping("/msgStatus")
    public void msgStatus(@Payload Map<String, Object> message) {
        String recipient = (String) message.get("recipient");
        Integer id = (Integer) message.get("msgId");
        Boolean isRead = (Boolean) message.get("msgStatus");

        messageService.messageIsRead(id, isRead);
        messagingTemplate.convertAndSendToUser(
            recipient,
            "/queue/messageIsRead",
            id
        );
    }

    @PostMapping("/allMessagesRead")
    public void allMessagesRead(@Payload Map<String, Object> message) {
        String sender = (String) message.get("sender");
        String opponent = (String) message.get("opponent");
        List<Integer> unreadIds =
messageService.markMessagesAsReadBetweenUsers(sender, opponent);

        messagingTemplate.convertAndSendToUser(
            opponent,
            "/queue/allMessagesIsRead",
            unreadIds
        );
    }
}

```

Файл «DiaryController»

```

@Controller
public class DiaryController {
    @Autowired
    private DiaryService diaryService;

    @PostMapping("/saveCard")
    public void saveCard(DiaryCardDTO cardDTO) {
        diaryService.saveCard(cardDTO);
    }

    @PostMapping("/deleteCard")
    public void deleteCard(DiaryCardDTO cardDTO) {
        diaryService.deleteCard(cardDTO.getId());
    }
}

```

Файл «MyController»

```

@Controller
public class MyController {

    @Autowired
    private MessageService messageService;

    @Autowired
    private PasswordRequestsRepo passwordRequestsRepo;

    @Autowired
    private HRRequestsRepo hrRequestsRepo;

    @GetMapping("/login")
    public String auth(Model model) {

```

```

String helloString = "Здравствуйте!";

LocalTime now = LocalTime.now();
LocalTime morning = LocalTime.of(5, 0);
LocalTime day = LocalTime.of(12, 0);
LocalTime evening = LocalTime.of(18, 0);
LocalTime night = LocalTime.of(0, 0);

if (now.isAfter(morning) && now.isBefore(day)) {
    helloString = "Доброго утра!";
} else if (now.isAfter(day) && now.isBefore(evening)) {
    helloString = "Доброго дня!";
} else if (now.isAfter(evening) && now.isBefore(night)) {
    helloString = "Доброго вечера!";
} else if (now.isAfter(night) && now.isBefore(morning)) {
    helloString = "Доброй ночи!";
}

model.addAttribute("today", helloString);
return "login";
}

@Autowired
private MyUserDetailService userDetailsService;

@GetMapping("/profile")
public String profile(Model model, Principal principal) {
    String username =
SecurityContextHolder.getContext().getAuthentication().getName();
    Users user = userDetailsService.findByUsername(username);
    model.addAttribute("currentUser", user);

    UsersDto profile = userDetailsService.getUserProfile(principal.getName());
    model.addAttribute("profile", profile);
    return "profile";
}

@Autowired
private ActivationService activationService;

@GetMapping("/activation")
public String activationPage(Model model, Authentication authentication) {
    UserPrincipal userPrincipal = (UserPrincipal)
authentication.getPrincipal();
    String username = userPrincipal.getUsername();
    Users user = userDetailsService.findByUsername(username);
    UsersInfo usersInfo = user.getUsersInfo() != null ? user.getUsersInfo() :
new UsersInfo();
    UsersContact usersContact = user.getUsersContact() != null ?
user.getUsersContact() : new UsersContact();

    model.addAttribute("userId", user.getId());
    model.addAttribute("usersInfo", usersInfo);
    model.addAttribute("usersContact", usersContact);
    return "activation";
}

@PostMapping("/activation")
public String processActivation(
    @RequestParam Integer userId,
    @ModelAttribute UsersInfo usersInfo,
    @ModelAttribute UsersContact usersContact,
    @RequestParam(value = "photo", required = false) MultipartFile
photoFile) {

```



```

        if (userInfo.getGender() != null) {
            switch (userInfo.getGender()) {
                case "male":
                    userInfo.setGender("Мужской");
                    break;
                case "female":
                    userInfo.setGender("Женский");
                    break;
                case "none":
                    userInfo.setGender(null);
                    break;
            }
        }
    }

    String VK = usersContact.getVKid();
    String telegram = usersContact.getTelegramUsername();

    if (!VK.isEmpty()) {
        if (!VK.contains("https://vk.com/")) {
            usersContact.setVKid("https://vk.com/" + VK);
        } else {
            usersContact.setVKid(VK);
        }
    }

    if (!telegram.isEmpty()) {
        if (!telegram.contains("https://t.me/")) {
            usersContact.setTelegramUsername("https://t.me/" + telegram);
        } else {
            usersContact.setTelegramUsername(telegram);
        }
    }

    activationService.activateUser(userId, userInfo, usersContact,
photoFile);
    return "redirect:/profile";
}

@GetMapping("/chat")
public String chat(@CookieValue(name = "currentChatUser", required = false)
String selectedUsername, Model model, Principal principal) {
    Users currentUser = userDetailsService.findByUsername(principal.getName());
    model.addAttribute("currentUser", currentUser);

    List<String> participantUsernames =
messageService.findChatParticipants(currentUser.getUsername());

    List<UsersDto> chatParticipants = participantUsernames.stream()
        .map(username -> userDetailsService.findByUsername(username))
        .filter(Objects::nonNull)
        .map(user -> new UsersDto(user))
        .collect(Collectors.toList());

    model.addAttribute("usersAll", chatParticipants);

    if (selectedUsername == null || selectedUsername.isEmpty()) {
        UsersDto profile =
userDetailsService.getUserProfile(currentUser.getUsername());
        model.addAttribute("selectedUser", profile);
    } else {
        UsersDto profile = userDetailsService.getUserProfile(selectedUsername);
        model.addAttribute("selectedUser", profile);
    }
}

```

```

        return "chat";
    }

    @GetMapping("/allUsers")
    public String allUsers(Model model, Principal principal) {
        List<Users> usersAll = userDetailsService.findByActivationTrue();
        List<UsersDto> usersNamesAll = usersAll.stream()
            .map(user -> new UsersDto(user))
            .collect(Collectors.toList());
        model.addAttribute("usersAll", usersNamesAll);

        List<String> allPositions = userDetailsService.findAllPositions();
        List<String> allCities = userDetailsService.findAllCities();
        model.addAttribute("positions", allPositions);
        model.addAttribute("cities", allCities);

        Users user = userDetailsService.findByUsername(principal.getName());
        model.addAttribute("currentUser", user);
        return "allUsers";
    }

    @GetMapping("/profile/{username}")
    public String viewUserProfile(@PathVariable String username, Model model,
        Principal principal) {
        Users currentUser = userDetailsService.findByUsername(principal.getName());
        model.addAttribute("currentUser", currentUser);

        UsersDto profile = userDetailsService.getUserProfile(username);
        model.addAttribute("profile", profile);

        return "profile";
    }

    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/admin")
    public String admin(Model model, Principal principal) {
        Users currentUser = userDetailsService.findByUsername(principal.getName());
        model.addAttribute("currentUser", currentUser);

        List<UsersInfo> usersAll = userDetailsService.findAllUsersInfo();

        List<UsersDto> users = usersAll.stream()
            .map(user -> new UsersDto(user))
            .collect(Collectors.toList());

        List<UsersDto> sortedUsers = users.stream()
            .sorted(Comparator.comparing(UsersDto::getId))
            .collect(Collectors.toList());
        model.addAttribute("usersAll", sortedUsers);

        List<PasswordRequests> passwordRequests = passwordRequestsRepo.findAll();
        model.addAttribute("passwordRequests", passwordRequests);

        List<HRRequests> hrRequests = hrRequestsRepo.findAll();
        model.addAttribute("hrRequests", hrRequests);

        return "admin";
    }

    @GetMapping("/diary")
    public String diary(Model model, Principal principal) {
        Users currentUser = userDetailsService.findByUsername(principal.getName());
        model.addAttribute("currentUser", currentUser);
    }

```

```

        List<DiaryCards> cards = currentUser.getDiaryCards();
        List<DiaryParagraphs> paragraphs = cards.stream()
            .flatMap(card -> card.getDiaryParagraphs().stream())
            .toList();

        model.addAttribute("cards", cards);
        model.addAttribute("paragraphs", paragraphs);

        return "diary";
    }

    @GetMapping("/HR")
    public String HR() {
        return "HR";
    }
}

Файл «UserController»
@Controller
public class UserController {

    @Lazy
    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @Autowired
    MyUserService userService;

    @Autowired
    MessageService messageService;

    @Autowired
    PasswordRequestsRepo passwordRequestsRepo;

    @Autowired
    private UserRepo userRepo;

    @Autowired
    private UserInfoRepo userInfoRepo;

    @Autowired
    EmailService emailService;

    @Autowired
    HRRequestsRepo hrRequestsRepo;

    @PostMapping("/addNewUser")
    public void addNewUser(@Payload Map<String, Object> userInfo) {
        String username = (String) userInfo.get("username");
        String password = (String) userInfo.get("password");
        String email = (String) userInfo.get("email");
        String firstname = (String) userInfo.get("firstname");
        String lastname = (String) userInfo.get("lastname");
        String position = (String) userInfo.get("position");
        String hiringDateString = (String) userInfo.get("hiringDate");
        Boolean admin = (Boolean) userInfo.get("admin");

        LocalDate hiringDate = LocalDate.parse(hiringDateString,
            DateTimeFormatter.ISO_LOCAL_DATE);

        userService.saveUserWithInfo(username, password, email, firstname,
            lastname, position, hiringDate, admin);
    }
}

```

```

    @PostMapping("/deleteUser")
    public void deleteUser(@Payload String username) {
        userDetailsService.deleteUserById(username);
        messageService.deleteAllMessagesByUser(username);
    }

    @PostMapping("/requestUserExist")
    @SendToUser("/queue/userExist")
    public Boolean userExist(@Payload String username) {
        Users user = userDetailsService.findByUsername(username);
        if (user != null) {
            UsersContact usersContact = user.getUsersContact();
            PasswordRequests passwordRequests = new PasswordRequests();
            passwordRequests.setUsername(username);
            passwordRequests.setEmail(usersContact.getEmail());
            passwordRequestsRepo.save(passwordRequests);
            return true;
        } else {
            return false;
        }
    }

    @PostMapping("/requestEmailExist")
    @SendToUser("/queue/emailExist")
    public Boolean emailExist(@Payload String email) {
        UsersContact user = userDetailsService.findByEmail(email);
        if (user != null) {
            Users users = user.getUser();
            PasswordRequests passwordRequests = new PasswordRequests();
            passwordRequests.setUsername(users.getUsername());
            passwordRequests.setEmail(email);
            passwordRequestsRepo.save(passwordRequests);
            return true;
        } else {
            return false;
        }
    }

    @PostMapping("/changePassword")
    public void changePassword(@Payload Map<String, Object> userInfo) {
        Integer id = (Integer) userInfo.get("id");
        String password = (String) userInfo.get("password");
        PasswordRequests passwordRequests =
passwordRequestsRepo.findById(id).orElse(new PasswordRequests());

        Users user = userRepo.findByUsername(passwordRequests.getUsername());
        user.setPassword(passwordEncoder.encode(password));
        userRepo.save(user);
        passwordRequestsRepo.deleteById(id);

        emailService.sendPasswordRecoveryEmail(passwordRequests.getEmail(),
password);
    }

    @PostMapping("/cancelRequest")
    public void cancelRequest(@Payload Integer id) {
        PasswordRequests passwordRequests =
passwordRequestsRepo.findById(id).orElse(new PasswordRequests());
        emailService.sendCancelRequest(passwordRequests.getEmail());
        passwordRequestsRepo.deleteById(id);
    }

    @PostMapping("/createNewUserRequest")

```

```

@SendToUser("/queue/actionRequest")
public String createNewUserRequest(@Payload Map<String, Object> userInfo) {
    String firstname = (String) userInfo.get("firstName");
    String lastname = (String) userInfo.get("lastName");
    String position = (String) userInfo.get("position");
    String hiringDateString = (String) userInfo.get("hiringDate");
    LocalDate hiringDate = LocalDate.parse(hiringDateString,
DateTimeFormatter.ISO_LOCAL_DATE);

    Boolean isExist = userDetailsService.existsByFourParameters(firstname,
lastname, position, hiringDate);
    if (isExist) {
        return "Данный сотрудник уже существует";
    } else {
        HRRequests request = new HRRequests();
        request.setFirstname(firstname);
        request.setLastname(lastname);
        request.setPosition(position);
        request.setHiringDate(hiringDate);
        request.setIsDelete(false);
        hrRequestsRepo.save(request);
        return "Запрос успешно создан";
    }
}

@MessageMapping("/deleteUserRequest")
@SendToUser("/queue/actionRequest")
public String createDeleteUserRequest(@Payload Map<String, Object> userInfo) {
    String firstname = (String) userInfo.get("firstName");
    String lastname = (String) userInfo.get("lastName");
    String position = (String) userInfo.get("position");
    String hiringDateString = (String) userInfo.get("hiringDate");
    LocalDate hiringDate = LocalDate.parse(hiringDateString,
DateTimeFormatter.ISO_LOCAL_DATE);

    Boolean isExist = userDetailsService.existsByFourParameters(firstname,
lastname, position, hiringDate);
    if (isExist) {
        HRRequests request = new HRRequests();
        request.setFirstname(firstname);
        request.setLastname(lastname);
        request.setPosition(position);
        request.setHiringDate(hiringDate);
        request.setIsDelete(true);
        hrRequestsRepo.save(request);
        return "Запрос успешно создан";
    } else {
        return "Такого сотрудника не существует";
    }
}

@MessageMapping("/cancelHRRequest")
public void cancelHRRequest(@Payload Integer id) {
    hrRequestsRepo.deleteById(id);
}

@MessageMapping("/acceptDeleteRequest")
public void deleteHRRequest(@Payload Map<String, Object> userInfo) {
    String firstname = (String) userInfo.get("firstName");
    String lastname = (String) userInfo.get("lastName");
    String position = (String) userInfo.get("position");
    String hiringDateString = (String) userInfo.get("hiringDate");
    LocalDate hiringDate = LocalDate.parse(hiringDateString,
DateTimeFormatter.ISO_LOCAL_DATE);

```

```

        Optional<UserInfo> user =
userInfoRepo.findUserByFourParameters(firstname, lastname, position, hiringDate);

        if (user.isPresent()) {
            Users deletedUser = user.get().getUser();
            userDetailsService.deleteUserById(deletedUser.getUsername());
        }
    }
}

```

Файл «WebSocketController»

@Controller

public class WebSocketController {

```

    private final Set<String> onlineUsers = ConcurrentHashMap.newKeySet();

```

```

    @Autowired

```

```

    private SimpMessagingTemplate messagingTemplate;

```

```

    @Autowired

```

```

    private MyUserDetailsService userDetailsService;

```

```

    public WebSocketController(SimpMessagingTemplate messagingTemplate) {

```

```

        this.messagingTemplate = messagingTemplate;

```

```

    }

```

```

    @MessageMapping("/register")

```

```

    public void registerUser(@Payload String username) {

```

```

        onlineUsers.add(username);

```

```

        notifyStatusChange(username, true);

```

```

    }

```

```

    @MessageMapping("/unregister")

```

```

    public void unregisterUser(@Payload String username) {

```

```

        onlineUsers.remove(username);

```

```

        notifyStatusChange(username, false);

```

```

    }

```

```

    @MessageMapping("/requestStatuses")

```

```

    public void sendStatuses(@Payload String requestingUsername) {

```

```

        List<String> allUsers = userDetailsService.findAllActiveUsernames();

```

```

        Map<String, Boolean> userStatuses = new HashMap<>();

```

```

        for (String user : allUsers) {

```

```

            userStatuses.put(user, onlineUsers.contains(user));

```

```

        }

```

```

        messagingTemplate.convertAndSendToUser(

```

```

            requestingUsername,

```

```

            "/queue/statuses",

```

```

            userStatuses

```

```

        );

```

```

    }

```

```

    @MessageMapping("/requestStatus")

```

```

    public void sendStatus(@Payload Map<String, Object> message) {

```

```

        String requestingUser = (String) message.get("requestingUser");

```

```

        String requestedUser = (String) message.get("requestedUser");

```

```

        Boolean response = onlineUsers.contains(requestedUser);

```

```

        messagingTemplate.convertAndSendToUser(

```

```

            requestingUser,

```

```

            "/queue/status",

```

```

            response

```

```

    );
}

private void notifyStatusChange(String username, boolean isOnline) {
    messagingTemplate.convertAndSend(
        "/topic/onlineStatus",
        Map.of("username", username, "isOnline", isOnline)
    );
}

@MessageMapping("/typing")
public void handleTyping(@Payload Map<String, Object> payload, Principal
principal) {
    boolean typing = Boolean.parseBoolean(payload.get("isTyping").toString());
    String recipient = payload.get("recipient").toString();

    Map<String, Object> response = new HashMap<>();
    response.put("sender", principal.getName());
    response.put("recipient", recipient);
    response.put("isTyping", typing);

    messagingTemplate.convertAndSendToUser(
        recipient,
        "/queue/typing",
        response
    );
}
}

```

Файл «DiaryCardDTO»

```

@Getter
@Setter
public class DiaryCardDTO {
    private Integer id;
    private String title;
    private LocalDate date;
    private List<DiaryParagraphDTO> diaryParagraphs;
    private Integer userId;
}

```

Файл «DiaryParagraphDTO»

```

@Getter
@Setter
public class DiaryParagraphDTO {
    private Integer id;
    private String content;
    private Boolean isReady;
    private Integer cardId;
}

```

Файл «LastMessageDTO»

```

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
public class LastMessageDTO {
    private Integer id;
    private String sender;
    private String content;
    private LocalDateTime timestamp;
    private Boolean isRead;
}

```

Файл «MessageDTO»

```

@Setter
@Getter
public class MessageDTO {
    private String content;
    private String recipient;

    private String otherUser;

    public MessageDTO(String content, String recipient) {
        this.content = content;
        this.recipient = recipient;
    }
}

Файл «UsersDTO»
@Setter
@Getter
public class UsersDto {

    private Integer id;
    private String username;
    private String firstName;
    private String lastName;
    private String position;
    private LocalDate dob;
    private String gender;
    private String city;
    private String street;
    private String officeFloor;
    private String officeNumber;
    private String photoPath;
    private Boolean activation;
    private Boolean admin;
    private String phoneNumber;
    private String email;
    private String VKid;
    private String TelegramUsername;

    public UsersDto(Users user) {
        this.username = user.getUsername();

        UsersInfo userInfo = user.getUsersInfo();
        UsersContact usersContact = user.getUsersContact();
        this.firstName = userInfo.getFirstname();
        this.lastName = userInfo.getLastname();
        this.position = userInfo.getPosition();
        this.dob = userInfo.getDob();
        this.gender = userInfo.getGender();
        this.city = userInfo.getCity();
        this.street = userInfo.getStreet();
        this.officeFloor = userInfo.getOfficeFloor();
        this.officeNumber = userInfo.getOfficeNumber();
        this.photoPath = userInfo.getPhotoPath();
        this.activation = user.getActivation();
        this.admin = user.getIsAdmin();
        this.phoneNumber = usersContact.getPhoneNumber();
        this.email = usersContact.getEmail();
        this.VKid = usersContact.getVKid();
        this.TelegramUsername = usersContact.getTelegramUsername();
    }

    public UsersDto(UsersInfo user) {
        this.id = user.getUser().getId();
        this.username = user.getUser().getUsername();
    }
}

```



```

        this.firstName = user.getFirstname();
        this.lastName = user.getLastname();
        this.position = user.getPosition();
        this.activation = user.getUser().getActivation();
        this.admin = user.getUser().getIsAdmin();
    }
}

Файл «DiaryCards»
@Entity
@Table(name="diary_cards")
@Setter
@Getter
public class DiaryCards implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private Users user;

    private String title;
    private LocalDate date;

    @OneToMany(mappedBy = "card", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<DiaryParagraphs> diaryParagraphs = new ArrayList<>();
}

Файл «DiaryParagraphs»
@Entity
@Table(name="diary_paragraphs")
@Setter
@Getter
public class DiaryParagraphs implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @ManyToOne
    @JoinColumn(name = "card_id")
    private DiaryCards card;

    private String content;
    private Boolean isReady;
}

Файл «HRRequests»
@Entity
@Table(name="hr_requests")
@Setter
@Getter
public class HRRequests {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String firstname;
    private String lastname;
    private String position;
    private LocalDate hiringDate;
    private Boolean isDelete = false;
}

```

Файл «MessageEntity»

```
@Entity
@Table(name = "messages")
@Setter
@Getter
public class MessageEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String sender;
    private String recipient;
    private String content;
    private LocalDateTime timestamp;
    @Column(nullable = false)
    private Boolean isPart = false;
    @Column(nullable = false)
    private Boolean isLastPart = false;
    private Integer partNumber;
    @Column(nullable = false)
    private Boolean isFile;
    @Column(nullable = false)
    private Boolean isRead = false;
}
```

Файл «PasswordRequests»

```
@Entity
@Table(name="password_requests")
@Setter
@Getter
public class PasswordRequests {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String username;
    private String email;
}
```

Файл «Users»

```
@Entity
@Table(name="users")
@Setter
@Getter
public class Users implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String username;
    private String password;
    @Column(nullable = false)
    private Boolean activation = false;
    @Column(nullable = false)
    private Boolean isAdmin;

    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private UsersInfo usersInfo;

    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private UsersContact usersContact;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<DiaryCards> diaryCards = new ArrayList<>();
}
```

```

@Override
public String toString() {
    return "Users{" +
        "id=" + id +
        ", username='" + username + '\'' +
        ", password='" + password + '\'' +
        ", activation=" + activation +
        '\'';
}
}

```

Файл «UsersContact»

```

@Entity
@Table(name="users_contact")
@Setter
@Getter

```

```

public class UsersContact implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String phoneNumber;
    private String email;
    private String VKid;
    private String TelegramUsername;

    @OneToOne
    @JoinColumn(name = "user_id")
    private Users user;
}

```

Файл «UsersInfo»

```

@Entity
@Table(name="users_info")
@Setter
@Getter

```

```

public class UsersInfo implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String firstname;
    private String lastname;
    private String position;
    private LocalDate dob;
    private String gender;
    private String city;
    private String street;
    private String officeFloor;
    private String officeNumber;
    private String photoPath;
    private LocalDate hiringDate;

    @OneToOne
    @JoinColumn(name = "user_id")
    private Users user;
}

```

Файл «DiaryParagraphsRepo»

```

public interface DiaryParagraphsRepo extends JpaRepository<DiaryParagraphs,
Integer> {
    @Modifying
    @Query("DELETE FROM DiaryParagraphs p WHERE p.card.id = :cardId AND p.id NOT
IN :ids")

```

```

        void deleteByCardIdAndIdNotIn(@Param("cardId") Integer cardId, @Param("ids")
List<Integer> ids);

        Optional<DiaryParagraphs> findByIdAndCardId(Integer id, Integer cardId);
    }
}

Файл «MessageRepo»
public interface MessageRepository extends JpaRepository<MessageEntity, Integer> {
    List<MessageEntity> findBySenderOrderByTimestampAsc(String sender);

    @Query("SELECT m FROM MessageEntity m WHERE " +
        "(m.sender = :user1 AND m.recipient = :user2) OR " +
        "(m.sender = :user2 AND m.recipient = :user1) " +
        "ORDER BY m.timestamp ASC")
    List<MessageEntity> findMessagesBetweenUsers(@Param("user1") String user1,
        @Param("user2") String user2);

    void deleteById(Integer id);

    MessageEntity getMessageEntityById(Integer id);

    @Modifying
    @Query("UPDATE MessageEntity m SET m.content = :content WHERE m.id = :id")
    int updateMessageContentById(@Param("id") int id, @Param("content") String
content);

    @Query("SELECT DISTINCT CASE WHEN m.sender = :username THEN m.recipient ELSE
m.sender END " +
        "FROM MessageEntity m WHERE m.sender = :username OR m.recipient =
:username")
    List<String> findChatParticipants(@Param("username") String username);

    @Modifying
    @Query("UPDATE MessageEntity m SET " +
        "m.sender = CASE WHEN m.sender = :username THEN 'DELETED' ELSE
m.sender END, " +
        "m.recipient = CASE WHEN m.recipient = :username THEN 'DELETED' ELSE
m.recipient END " +
        "WHERE m.sender = :username OR m.recipient = :username")
    void markDeletedUser(@Param("username") String username);

    @Modifying
    @Query("DELETE FROM MessageEntity m WHERE m.sender = :username OR m.recipient
= :username")
    void deleteAllMessagesByUser(@Param("username") String username);

    @Query("SELECT DISTINCT CASE " +
        "WHEN m.sender = :currentUser THEN m.recipient " +
        "ELSE m.sender END " +
        "FROM MessageEntity m " +
        "WHERE m.sender = :currentUser OR m.recipient = :currentUser")
    List<String> findOpponents(@Param("currentUser") String currentUser);

    @Query("SELECT m FROM MessageEntity m " +
        "WHERE (m.sender = :user1 AND m.recipient = :user2) " +
        "OR (m.sender = :user2 AND m.recipient = :user1) " +
        "ORDER BY m.timestamp DESC LIMIT 1")
    MessageEntity findLastMessageBetweenUsers(
        @Param("user1") String user1,
        @Param("user2") String user2);

    @Modifying
    @Query("UPDATE MessageEntity m SET m.isRead = :isRead WHERE m.id = :id")

```

```

    int updateMessageStatusById(@Param("id") int id, @Param("isRead") Boolean
isRead);

    @Query("SELECT m.id FROM MessageEntity m WHERE " +
        "(m.sender = :user2 AND m.recipient = :user1) " +
        "AND m.isRead = false")
    List<Integer> findUnreadMessagesIdsBetweenUsers(@Param("user1") String user1,
        @Param("user2") String user2);

    @Modifying
    @Query("UPDATE MessageEntity m SET m.isRead = true WHERE m.id IN :ids")
    void markMessagesAsRead(@Param("ids") List<Integer> ids);
}

```

Файл «UserInfoRepo»

@Repository

public interface UserInfoRepo extends JpaRepository<UsersInfo, Integer> {

```

    @Query("SELECT DISTINCT ui.position FROM UsersInfo ui WHERE ui.position IS NOT
NULL")
    List<String> findAllPositions();

```

```

    @Query("SELECT DISTINCT ui.city FROM UsersInfo ui WHERE ui.city IS NOT NULL")
    List<String> findAllCities();

```

```

    Optional<UsersInfo> findById(Integer userId);

```

```

    @Query("SELECT CASE WHEN COUNT(u) > 0 THEN true ELSE false END " +
        "FROM UsersInfo u WHERE " +
        "u.firstname = :firstname AND " +
        "u.lastname = :lastname AND " +
        "u.position = :position AND " +
        "u.hiringDate = :hiringDate")

```

```

    boolean existsByFourParameters(
        @Param("firstname") String firstname,
        @Param("lastname") String lastname,
        @Param("position") String position,
        @Param("hiringDate") LocalDate hiringDate);

```

```

    @Query("SELECT u FROM UsersInfo u WHERE " +
        "u.firstname = :firstname AND " +
        "u.lastname = :lastname AND " +
        "u.position = :position AND " +
        "u.hiringDate = :hiringDate")

```

```

    Optional<UsersInfo> findUserByFourParameters(
        @Param("firstname") String firstname,
        @Param("lastname") String lastname,
        @Param("position") String position,
        @Param("hiringDate") LocalDate hiringDate);
}

```

Файл «UserRepo»

@Repository

public interface UserRepo extends JpaRepository<Users, Integer> {

```

    Users findByUsername(String username);

```

```

    List<Users> findByActivationTrue();

```

```

    @Query("SELECT u.username FROM Users u WHERE u.activation = true")
    List<String> findAllActiveUsernames();

```

```

    @Query("SELECT u.id FROM Users u WHERE u.username = :username")
    Integer getIdByUsername(@Param("username") String username);
}

```

```

}

Файл «ActivationService»
@Service
@Transactional
public class ActivationService {

    @Autowired
    private UserRepo userRepo;

    @Autowired
    private UserInfoRepo userInfoRepo;

    @Autowired
    private UserContactRepo userContactRepo;

    @Autowired
    private FileStorageService fileStorageService;

    public void activateUser(Integer userId, UsersInfo usersInfo, UsersContact
usersContact, MultipartFile photoFile) {
        Users user = userRepo.findById(userId)
            .orElseThrow(() -> new RuntimeException("User not found"));
        user.setActivation(true);

        UsersInfo existingInfo = userInfoRepo.findById(userId)
            .orElse(new UsersInfo());

        existingInfo.setFirstname(usersInfo.getFirstname());
        existingInfo.setLastname(usersInfo.getLastname());
        existingInfo.setPosition(usersInfo.getPosition());
        existingInfo.setDob(usersInfo.getDob());
        existingInfo.setGender(usersInfo.getGender());
        existingInfo.setCity(usersInfo.getCity());
        existingInfo.setStreet(usersInfo.getStreet());
        existingInfo.setOfficeFloor(usersInfo.getOfficeFloor());
        existingInfo.setOfficeNumber(usersInfo.getOfficeNumber());

        if (photoFile != null && !photoFile.isEmpty()) {
            String photoPath = fileStorageService.storeFile(photoFile, userId);
            existingInfo.setPhotoPath(photoPath);
        }

        existingInfo.setUser(user);
        userInfoRepo.save(existingInfo);

        UsersContact newUsersContact = userContactRepo.findById(userId)
            .orElse(new UsersContact());

        newUsersContact.setPhoneNumber(usersContact.getPhoneNumber());
        newUsersContact.setEmail(usersContact.getEmail());
        newUsersContact.setVKid(usersContact.getVKid());
        newUsersContact.setTelegramUsername(usersContact.getTelegramUsername());
        newUsersContact.setUser(user);
        userContactRepo.save(newUsersContact);
    }
}

Файл «DiaryService»
@Service
@Transactional
public class DiaryService {

    @Autowired

```

```

private DiaryCardsRepo diaryCardsRepo;

@Autowired
private DiaryParagraphsRepo diaryParagraphsRepo;

@Autowired
private UserRepo userRepo;

public DiaryCards saveCard(DiaryCardDTO cardDTO) {
    DiaryCards card;
    if (cardDTO.getId() != null) {
        card = diaryCardsRepo.findById(cardDTO.getId())
            .orElseThrow(() -> new RuntimeException("Карточка не
найдена"));
    } else {
        card = new DiaryCards();
        card.setDate(LocalDate.now());
    }

    Users user = userRepo.findById(cardDTO.getUserId())
        .orElseThrow(() -> new RuntimeException("Пользователь не
найден"));

    card.setTitle(cardDTO.getTitle());
    card.setUser(user);

    DiaryCards savedCard = diaryCardsRepo.save(card);

    if (cardDTO.getDiaryParagraphs() != null &&
!cardDTO.getDiaryParagraphs().isEmpty()) {

        if (savedCard.getId() != null) {
            List<Integer> existingParagraphIds =
cardDTO.getDiaryParagraphs().stream()
                .map(DiaryParagraphDTO::getId)
                .filter(Objects::nonNull)
                .collect(Collectors.toList());

            if (!existingParagraphIds.isEmpty()) {
diaryParagraphsRepo.deleteByCardIdAndIdNotIn(savedCard.getId(),
existingParagraphIds);
            }
        }

        for (DiaryParagraphDTO paragraphDTO : cardDTO.getDiaryParagraphs()) {
            DiaryParagraphs paragraph = new DiaryParagraphs();

            if (paragraphDTO.getId() != null && savedCard.getId() != null) {
                paragraph =
diaryParagraphsRepo.findByIdAndCardId(paragraphDTO.getId(), savedCard.getId())
                    .orElse(new DiaryParagraphs());
            }

            paragraph.setContent(paragraphDTO.getContent());
            paragraph.setIsReady(paragraphDTO.getIsReady() != null ?
paragraphDTO.getIsReady() : false);
            paragraph.setCard(savedCard);

            diaryParagraphsRepo.save(paragraph);
        }
    }

    return savedCard;
}

```

```

    }

    public void deleteCard(Integer cardId) {
        DiaryCards card = diaryCardsRepo.findById(cardId)
            .orElseThrow(() -> new EntityNotFoundException("Карточка не
найдена"));

        diaryCardsRepo.delete(card);
    }
}

Файл «EmailService»
@Service
public class EmailService {

    @Autowired
    private JavaMailSender mailSender;

    public void sendPasswordRecoveryEmail(String toEmail, String password) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("korenko_2025@mail.ru");
        message.setTo(toEmail);
        message.setSubject("Восстановление пароля");
        message.setText("Здравствуйте!\n\n"
            + "Новый пароль для доступа к аккаунту: "
            + password);

        mailSender.send(message);
    }

    public void sendCancelRequest(String toEmail) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("korenko_2025@mail.ru");
        message.setTo(toEmail);
        message.setSubject("Отказ в восстановлении пароля");
        message.setText("Здравствуйте!\n\n"
            + "Ваш запрос на восстановление пароля был отклонен");

        mailSender.send(message);
    }
}

Файл «FileStorageService»
@Service
public class FileStorageService {
    private final Path fileStorageLocation;

    @Autowired
    public FileStorageService(FileStorageConfig config) {
        this.fileStorageLocation = config.fileStorageLocation();
    }

    public String storeFile(MultipartFile file, Integer id) {
        String filename = StringUtils.cleanPath(file.getOriginalFilename());
        String extension = filename.substring(filename.lastIndexOf("."));
        String newFilename = "avatar_" + id + extension;

        try {
            Path targetLocation = this.fileStorageLocation.resolve(newFilename);
            Files.copy(file.getInputStream(), targetLocation,
StandardCopyOption.REPLACE_EXISTING);
            return newFilename;
        } catch (IOException ex) {

```



```

        throw new RuntimeException("Не удалось сохранить файл " + filename,
ex);
    }
}

public Resource loadFile(String filename) {
    try {
        Path filePath =
this.fileStorageLocation.resolve(filename).normalize();
        Resource resource = new UrlResource(filePath.toUri());
        if (resource.exists()) {
            return resource;
        } else {
            throw new RuntimeException("Файл не найден " + filename);
        }
    } catch (MalformedURLException ex) {
        throw new RuntimeException("Файл не найден " + filename, ex);
    }
}
}

```

Файл «IpAddressMatcher»

```

public class IpAddressMatcher implements RequestMatcher {
    private final Set<String> allowedSubnets;

    public IpAddressMatcher(Set<String> allowedSubnets) {
        this.allowedSubnets = allowedSubnets;
    }

    @Override
    public boolean matches(HttpServletRequest request) {
        String clientIp = request.getRemoteAddr();

        if ("0:0:0:0:0:0:0:1".equals(clientIp) || "::1".equals(clientIp)) {
            clientIp = "127.0.0.1";
        }

        for (String subnet : allowedSubnets) {
            boolean inSubnet = isInSubnet(clientIp, subnet);
            if (inSubnet) {
                return true;
            }
        }
        return false;
    }

    private boolean isInSubnet(String ip, String subnet) {
        SubnetUtils subnetUtils = new SubnetUtils(subnet);
        return subnetUtils.getInfo().isInRange(ip);
    }
}

```

Файл «MessageService»

```

@Service
@Transactional
public class MessageService {

    @Autowired
    private MessageRepository messageRepository;

    private static final int MAX_MESSAGE_LENGTH = 255;

    public List<MessageEntity> saveLongMessageToDatabase(String sender, String
recipient, String text) {

```

```

List<String> messagePart = splitMessage(text);
List<MessageEntity> savedMessage = new ArrayList<>();

for (String part : messagePart) {
    MessageEntity messageEntity = new MessageEntity();
    messageEntity.setSender(sender);
    messageEntity.setRecipient(recipient);
    messageEntity.setContent(part);
    messageEntity.setTimestamp(LocalDateTime.now());
    messageEntity.setIsPart(true);
    messageEntity.setPartNumber(savedMessage.size() + 1);

    savedMessage.add(messageRepository.save(messageEntity));
}

if (!savedMessage.isEmpty()) {
    MessageEntity lastMessage = savedMessage.get(savedMessage.size() - 1);
    lastMessage.setIsLastPart(true);
    messageRepository.save(lastMessage);
}

return savedMessage;
}

private List<String> splitMessage(String text) {
    List<String> parts = new ArrayList<>();
    if (text.length() <= MAX_MESSAGE_LENGTH) {
        parts.add(text);
        return parts;
    }

    int start = 0;
    while (start < text.length()) {
        int end = Math.min(start + MAX_MESSAGE_LENGTH, text.length());

        if (end < text.length()) {
            int lastSpace = text.lastIndexOf(' ', end);
            if (lastSpace > start) {
                end = lastSpace;
            }
        }
        parts.add(text.substring(start, end).trim());
        start = end;
    }
    return parts;
}

public MessageEntity saveToDatabase(String sender, String recipient, String
text, Boolean isFile) {

    MessageEntity messageEntity = new MessageEntity();

    messageEntity.setSender(sender);
    messageEntity.setRecipient(recipient);
    messageEntity.setContent(text);
    messageEntity.setTimestamp(LocalDateTime.now());
    messageEntity.setIsFile(isFile);

    messageRepository.save(messageEntity);
    return messageEntity;
}

public void deleteAllMessagesBetweenUsers(String user1, String user2) {

```

```

        List<MessageEntity> allMessages =
messageRepository.findMessagesBetweenUsers(user1, user2);
        allMessages.forEach(msg -> {
            messageRepository.deleteById(msg.getId());
        });
    }

    public Map<String, LastMessageDTO> getLastMessagesForUser(String currentUser)
    {
        List<String> opponents = messageRepository.findOpponents(currentUser);
        Map<String, LastMessageDTO> lastMessages = new HashMap<>();

        opponents.forEach(opponent -> {
            MessageEntity lastMsg =
messageRepository.findLastMessageBetweenUsers(currentUser, opponent);
            if (lastMsg != null) {
                lastMessages.put(opponent,
                    new LastMessageDTO(
                        lastMsg.getId(),
                        lastMsg.getSender(),
                        lastMsg.getContent(),
                        lastMsg.getTimestamp(),
                        lastMsg.getIsRead()
                    ));
            }
        });
        return lastMessages;
    }

    public List<Integer> markMessagesAsReadBetweenUsers(String user1, String
user2) {
        List<Integer> unreadMessages =
messageRepository.findUnreadMessagesIdsBetweenUsers(user1, user2);
        messageRepository.markMessagesAsRead(unreadMessages);
        return unreadMessages;
    }
}

```

Файл «MyUserDetailService»

```

@Service
@Transactional
public class MyUserDetailService implements UserDetailsService {

    @Autowired
    private UserRepo userRepo;

    @Autowired
    private UserInfoRepo userInfoRepo;

    @Lazy
    @Autowired
    private BCryptPasswordEncoder passwordEncoder;
    @Autowired
    private UserContactRepo userContactRepo;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Users user = userRepo.findByUsername(username);
        return new UserPrincipal(user);
    }

    public void saveUserWithInfo(String username, String password,
                                String email,

```

```

        String firstname, String lastname,
        String position, LocalDate hiringDate, Boolean
admin) {

    Users user = new Users();
    user.setUsername(username);
    user.setPassword(passwordEncoder.encode(password));
    user.setIsAdmin(admin);

    UsersInfo userInfo = new UsersInfo();
    userInfo.setFirstname(firstname);
    userInfo.setLastname(lastname);
    userInfo.setPosition(position);
    userInfo.setHiringDate(hiringDate);

    UsersContact userContact = new UsersContact();
    userContact.setEmail(email);

    user.setUsersInfo(userInfo);
    userInfo.setUser(user);

    user.setUsersContact(userContact);
    userContact.setUser(user);

    userRepo.save(user);
}

public void deleteUserById(String username) {
    Users user = userRepo.findByUsername(username);
    userRepo.delete(user);
}

public Boolean existsByFourParameters(String firstname, String lastname,
String position, LocalDate hiringDate) {
    return userInfoRepo.existsByFourParameters(firstname, lastname, position,
hiringDate);
}
}

```

ПРИЛОЖЕНИЕ 2. КЛИЕНТСКАЯ ЧАСТЬ

```
Файл «/admin/actionHandler»
window.addEventListener('beforeunload', () => {
    stompClient.send("/app/unregister", {}, getCurrentUser());
});

function getCurrentUser() {
    const element = document.getElementById('currentUser');
    return element ? element.getAttribute('data-username') : null;
}

document.addEventListener('DOMContentLoaded', function() {
    connect();

    const usersButton = document.getElementById('usersButton');
    const hrButton = document.getElementById('hrButton');
    const passwordButton = document.getElementById('passwordButton');
    const addUserButton = document.getElementById('addUserButton');

    const mainViewUsers = document.getElementById('mainViewUsers');
    const mainViewHR = document.getElementById('mainViewHR');
    const mainViewPassword = document.getElementById('mainViewPassword');
    const mainViewCreateUser = document.getElementById('mainViewCreateUser');
    const mainViewNewPassword = document.getElementById('mainViewNewPassword');

    usersButton.addEventListener('click', () => switchView(mainViewUsers));
    hrButton.addEventListener('click', () => switchView(mainViewHR));
    passwordButton.addEventListener('click', () => switchView(mainViewPassword));
    addUserButton.addEventListener('click', function () {
switchView(mainViewCreateUser);
        mainViewCreateUser.querySelector('.viewHeader').textContent = "Создание
пользователя");
    });

    const searchInput = document.getElementById('searchInput');
    const searchButton = document.getElementById('searchButton');
    const cancelSearchButton = document.getElementById('cancelSearchButton');

    searchButton.addEventListener('click', function () {
        resetUserList();
        performSearch();
    });

    searchInput.addEventListener('keydown', function(e) {
        if (e.key === 'Enter') {
            e.preventDefault();
            resetUserList();
            performSearch();
        }
    });

    document.getElementById('haveActivation').addEventListener('change',
function() {
        performSearch();
    })

    cancelSearchButton.addEventListener('click', function() {
        const searchInput = document.getElementById('searchInput');
        const haveActivationCheckbox = document.getElementById('haveActivation');

        searchInput.value = '';
    });
});
```

```

        haveActivationCheckbox.checked = false;
        this.style.display = 'none';

        const noResults = document.querySelector('.no-results');
        if (noResults) {
            noResults.remove();
        }

        resetUserList();
    });

    const submitNewUser = document.getElementById('submitNewUser');

    submitNewUser.addEventListener('click', getUserInfo);

    document.addEventListener('click', function (e) {
        if (e.target.closest('.deleteUser')) {
            const button = e.target.closest('.deleteUser');
            const username = button.dataset.username;
            deleteUser(username);
        }
    });

    document.addEventListener('click', function (e) {
        if (e.target.closest('.accept')) {
            switchView(mainViewNewPassword);
            const button = e.target.closest('.accept');
            const id = +button.getAttribute('request-id');

            submitNewPassword.addEventListener('click', function () {
                const inputPassword =
document.getElementById('inputNewPassword').value.trim();
                if (inputPassword.length > 4) {
                    stompClient.send("/app/changePassword", {}, JSON.stringify({
                        id: id,
                        password: inputPassword
                    }));
                }
            });
        }
    });

    const submitNewPassword = document.getElementById('submitNewPassword');

    document.addEventListener('click', function (e) {
        if (e.target.closest('.cancel')) {
            const button = e.target.closest('.cancel');
            const username = button.getAttribute('data-username');
            const id = button.getAttribute('request-id');
            const viewPassword = document.getElementById('viewPassword');
            const listItem = document.getElementById(`listItem-password-${username}-${id}`);
            viewPassword.removeChild(listItem);
            stompClient.send("/app/cancelRequest", {}, id);
        }
    });

    const createUserRequests = document.getElementById('createUserRequests');
    const deleteUserRequests = document.getElementById('deleteUserRequests');
    const createUserItems = document.getElementById('createUserItems');
    const deleteUserItems = document.getElementById('deleteUserItems');

    createUserRequests.addEventListener('click', function () {

```

```

        createUserRequests.className = 'checked';
        deleteUserRequests.className = 'unchecked';

        createUserItems.style.display = "block";
        deleteUserItems.style.display = "none";
    });

    deleteUserRequests.addEventListener('click', function () {
        createUserRequests.className = 'unchecked';
        deleteUserRequests.className = 'checked';

        createUserItems.style.display = "none";
        deleteUserItems.style.display = "block";
    });

    document.addEventListener('click', function (e) {
        if (e.target.closest('.cancelCreateRequest')) {
            const button = e.target.closest('.cancelCreateRequest');
            const id = button.getAttribute('request-id');
            const createUserItems = document.getElementById('createUserItems');
            const listItem = document.getElementById(`listItem-request-${id}`);
            createUserItems.removeChild(listItem);
            stompClient.send("/app/cancelHRRequest", {}, id);
        }
    });

    document.addEventListener('click', function (e) {
        if (e.target.closest('.cancelDeleteRequest')) {
            const button = e.target.closest('.cancelDeleteRequest');
            const id = button.getAttribute('request-id');
            const deleteUserItems = document.getElementById('deleteUserItems');
            const listItem = document.getElementById(`listItem-request-${id}`);
            deleteUserItems.removeChild(listItem);
            stompClient.send("/app/cancelHRRequest", {}, id);
        }
    });

    document.addEventListener('click', function (e) {
        if (e.target.closest('.acceptCreateRequest')) {
            const id = document.getElementById('requestId').getAttribute('data');
            const firstname =
document.getElementById('requestFirstname').getAttribute('data');
            const lastname =
document.getElementById('requestLastname').getAttribute('data');
            const position =
document.getElementById('requestPosition').getAttribute('data');
            const hiringDate =
document.getElementById('requestHiringDate').getAttribute('data');

            document.getElementById('inputRequestId').value = id;
            document.getElementById('inputFirstname').value = firstname;
            document.getElementById('inputLastname').value = lastname;
            document.getElementById('inputPosition').value = position;
            document.getElementById('inputHiringDate').value = hiringDate;

            switchView(mainViewCreateUser);
        }
    });

    document.addEventListener('click', function (e) {
        if (e.target.closest('.acceptDeleteRequest')) {
            const id =
document.getElementById('deleteRequestId').getAttribute('data');

```

```

        const firstname =
document.getElementById('deleteRequestFirstname').getAttribute('data');
        const lastname =
document.getElementById('deleteRequestLastname').getAttribute('data');
        const position =
document.getElementById('deleteRequestPosition').getAttribute('data');
        const hiringDate =
document.getElementById('deleteRequestHiringDate').getAttribute('data');

        const deleteUserItems = document.getElementById('deleteUserItems');
        const listItem = document.getElementById(`listItem-request-${id}`);
        deleteUserItems.removeChild(listItem);

        stompClient.send("/app/acceptDeleteRequest", {}, JSON.stringify({
            firstname: firstname,
            lastname: lastname,
            position: position,
            hiringDate: hiringDate
        }));
        stompClient.send("/app/cancelHRRequest", {}, id);
    }
    });
});

function getUserInfo() {

    const inputRequestId = document.getElementById('inputRequestId').value.trim();
    const inputUsername = document.getElementById('inputUsername').value.trim();
    const inputPassword = document.getElementById('inputPassword').value.trim();
    const inputEmail = document.getElementById('inputEmail').value.trim();
    const inputFirstname = document.getElementById('inputFirstname').value.trim();
    const inputLastname = document.getElementById('inputLastname').value.trim();
    const inputPosition = document.getElementById('inputPosition').value.trim();
    const inputHiringDate =
document.getElementById('inputHiringDate').value.trim();
    const inputAdminRole = document.getElementById('inputAdminRole').checked;

    const inputElement = document.getElementById('inputUsername');

    inputElement.setCustomValidity('');

    const allUsernames = document.querySelector(`.listItem[user-
id="${inputUsername}"]`);
    if (allUsernames) {
        console.log("Пользователь существует");
        inputElement.setCustomValidity("Пользователь уже существует!");
        inputElement.reportValidity();
        inputElement.addEventListener('input', function () {
            inputElement.setCustomValidity('');
            inputElement.reportValidity();
        });
    } else if (inputUsername && inputPassword &&
inputFirstname && inputLastname &&
inputPosition && inputHiringDate) {
        inputElement.setCustomValidity('');
        inputElement.reportValidity();
        console.log(inputFirstname);
        stompClient.send("/app/addNewUser", {},
            JSON.stringify({
                username: inputUsername,
                password: inputPassword,
                email: inputEmail,
                firstname: inputFirstname,
                lastname: inputLastname,

```



```

        position: inputPosition,
        hiringDate: inputHiringDate,
        admin: inputAdminRole
    }));
    if(inputRequestId) {
        stompClient.send("/app/cancelHRRequest", {}, inputRequestId);
        document.getElementById('inputRequestId').value = '';
        const createUserItems = document.getElementById('createUserItems');
        const listItem = document.getElementById(`listItem-request-${inputRequestId}`);
        createUserItems.removeChild(listItem);
    }
    document.getElementById('inputUsername').value = '';
    document.getElementById('inputPassword').value = '';
    document.getElementById('inputEmail').value = '';
    document.getElementById('inputFirstname').value = '';
    document.getElementById('inputLastname').value = '';
    document.getElementById('inputPosition').value = '';
    document.getElementById('inputHiringDate').value = '';
    switchView(document.getElementById('mainViewUsers'));
}

function deleteUser(username) {
    stompClient.send("/app/deleteUser", {}, username);
    const usersList = document.getElementById('usersList');
    const listItem = document.getElementById(`listItem-${username}`);

    usersList.removeChild(listItem);
}

let currenView = document.getElementById('mainViewUsers');

function switchView(view) {
    removeCurrentView();
    currenView.style.display = 'none';
    view.style.display = 'grid';
    currenView = view;
}

function removeCurrentView() {
    if (currenView === null) {
        currenView = document.getElementById('mainViewUsers');
    }
    console.log(currenView);
    currenView.style.display = 'none';
}

Файл «/admin/serachUsers»
function getSelectedFilters() {
    return {
        haveActivation: document.getElementById('haveActivation').checked
    };
}

function matchesFilters(user, filters) {
    if (filters.haveActivation) {
        return user.getAttribute('data-activation') === 'true';
    }
    return true;
}

function performSearch() {

```

```

    const searchTerm =
document.getElementById('searchInput').value.trim().toLowerCase();
    const cancelButton = document.getElementById('cancelSearchButton');
    const users = document.querySelectorAll('.listItem');
    const filters = getSelectedFilters();
    let hasResults = false;

document.querySelectorAll('.no-results').forEach(el => el.remove());

    if (!searchTerm && !filters.haveActivation) {
        resetUserList();
        cancelButton.style.display = 'none';
        return;
    }

    users.forEach(user => {
        const userName =
user.querySelector('.userName').textContent.toLowerCase();
        const fullName = user.querySelector('.name').textContent.toLowerCase();
        const searchText = user.getAttribute('data-search-text').toLowerCase();

        const matchesSearch = !searchTerm ||
            userName.includes(searchTerm) ||
            fullName.includes(searchTerm) ||
            searchText.includes(searchTerm);

        const matchesFilter = matchesFilters(user, filters);

        if (matchesSearch && matchesFilter) {
            user.style.display = 'grid';
            hasResults = true;
        } else {
            user.style.display = 'none';
        }
    });

    if (!hasResults) {
        const noResults = document.createElement('div');
        noResults.className = 'no-results';
        noResults.textContent = 'Пользователи не найдены';
        document.querySelector('.usersList').appendChild(noResults);
    }

    cancelButton.style.display = 'flex';
}

function resetUserList() {
    const users = document.querySelectorAll('.listItem');
    users.forEach(user => {
        user.style.display = 'grid';
    });
}

Файл «wsConnect»
let stompClient = null;

function connect() {
    const socket = new SockJS('/ws');
    stompClient = Stomp.over(socket);
}

Файл «/allUsers/actionHandler»
window.addEventListener('beforeunload', () => {
    stompClient.send("/app/unregister", {}, getCurrentUser());
});

```

```

});

document.addEventListener('DOMContentLoaded', function() {
    connect();
    const searchInput = document.getElementById('searchInput')
    const searchButton = document.getElementById('searchButton');
    const cancelSearchButton = document.getElementById('cancelSearchButton');
    const openFilter = document.getElementById('openFilter');
    const closeFilter = document.getElementById('closeFilter');
    const applyFilterButton = document.querySelector('.applyFilter');
    const resetFilterButton = document.querySelector('.resetFilter');

    searchButton.addEventListener('click', function () {
        resetUserList();
        performSearch();
    });

    searchInput.addEventListener('keydown', function(e) {
        if (e.key === 'Enter') {
            e.preventDefault();
            resetUserList();
            performSearch();
        }
    });

    cancelSearchButton.addEventListener('click', function() {
        const searchInput = document.getElementById('searchInput');
        searchInput.value = '';
        this.style.display = 'none';

        const noResults = document.querySelector('.no-results');
        if (noResults) {
            noResults.remove();
        }

        resetUserList();
    });

    openFilter.addEventListener('click', function () {
        const filterMenu = document.getElementById('filterMenu');
        filterMenu.style.display='grid';
        this.style.display = 'none';
        closeFilter.style.display = 'block';
    });

    closeFilter.addEventListener('click', function () {
        const filterMenu = document.getElementById('filterMenu');
        filterMenu.style.display='none';
        this.style.display = 'none';
        openFilter.style.display = 'block';
    });

    applyFilterButton.addEventListener('click', function() {
        resetUserList();
        performSearch();
        const filterMenu = document.getElementById('filterMenu');
        filterMenu.style.display = 'none';
        closeFilter.style.display = 'none';
        openFilter.style.display = 'block';
    });

    resetFilterButton.addEventListener('click', function() {
        resetFilters();
    });

```

```

        resetUserList();
    });
});

function startChat(button) {
    const username = button.getAttribute('data-username');
    localStorage.setItem('currentChatUser', username);
    document.cookie = `currentChatUser=${username}; path=/; max-age=3600`;
    window.location.href = `/chat`;
}

Файл «/allUsers/searchUsers»
function getCurrentUser() {
    const element = document.getElementById('currentUser');
    return element ? element.getAttribute('data-username') : null;
}

function getSelectedFilters() {
    return {
        position: document.querySelector('.filterPosition').value,
        gender: document.querySelector('.filterGender').value,
        city: document.querySelector('.filterCity').value,
        isOnline: document.getElementById('isOnline').checked
    };
}

function matchesFilters(user, filters) {
    if (filters.position !== 'Не выбрано' &&
        user.getAttribute('data-position') !== filters.position) {
        return false;
    }

    if (filters.gender !== 'Не выбрано' &&
        user.getAttribute('data-gender') !== filters.gender) {
        return false;
    }

    if (filters.city !== 'Не выбрано' &&
        user.getAttribute('data-city') !== filters.city) {
        return false;
    }

    if (filters.isOnline) {
        const userImage = user.querySelector('.userImage');
        if (!userImage.classList.contains('online')) {
            return false;
        }
    }

    return true;
}

function performSearch() {
    const searchTerm =
document.getElementById('searchInput').value.trim().toLowerCase();
    const cancelButton = document.getElementById('cancelSearchButton');
    const users = document.querySelectorAll('.listItem');
    const filters = getSelectedFilters();
    let hasResults = false;

    document.querySelectorAll('.no-results').forEach(el => el.remove());

    const visibleUsers = Array.from(document.querySelectorAll('.listItem'))
        .filter(user => user.style.display !== 'none');

```

```

const usersList = document.getElementById('usersList');
usersList.innerHTML = '';
visibleUsers.forEach(user => usersList.appendChild(user));

if (!searchTerm && Object.values(filters).every(val => val === false || val
=== 'Не выбрано')) {
    resetUserList();
    cancelButton.style.display = 'none';
    return;
}

users.forEach(user => {
    const userName =
user.querySelector('.userName').textContent.toLowerCase();
    const searchText = user.getAttribute('data-search-text').toLowerCase();

    const matchesSearch = !searchTerm ||
        userName.includes(searchTerm) ||
        searchText.includes(searchTerm);
    const matchesFilter = matchesFilters(user, filters);

    if (matchesSearch && matchesFilter) {
        user.style.display = 'grid';
        hasResults = true;
    } else {
        user.style.display = 'none';
    }
});

if (!hasResults) {
    const noResults = document.createElement('div');
    noResults.className = 'no-results';
    noResults.id = 'tmpText';
    noResults.textContent = 'Пользователи не найдены';
    document.getElementById('usersList').appendChild(noResults);
}

cancelButton.style.display = 'flex';
}

function resetFilters() {
    document.querySelector('.filterPosition').value = 'Не выбрано';
    document.querySelector('.filterGender').value = 'Не выбрано';
    document.querySelector('.filterCity').value = 'Не выбрано';
    document.getElementById('isOnline').checked = false;
}

function resetUserList() {
    const usersList = document.getElementById('usersList');
    const users = document.querySelectorAll('.listItem');

    usersList.innerHTML = '';
    users.forEach(user => {
        usersList.appendChild(user);
        user.style.display = 'grid';
    });
}

Файл «/allUsers/wsOperation»
let stompClient = null;
let isConnected = false;

function connect() {

```

```

const socket = new SockJS('/ws');
stompClient = Stomp.over(socket);

stompClient.connect({}, () => {
    isConnected = true;

    stompClient.subscribe('/user/queue/statuses', function(message) {
        const statuses = JSON.parse(message.body);
        Object.entries(statuses).forEach(([username, isOnline]) => {
            updateUserStatus(username, isOnline);
        });
    });

    const currentUser = getCurrentUser();
    stompClient.send("/app/register", {}, currentUser);
    stompClient.send("/app/requestStatuses", {}, currentUser);

});

function updateUserStatus(username, isOnline) {
    const userElements = document.querySelectorAll(`.listItem[data-user-id="${username}"] .userImage`);
    userElements.forEach(element => {
        element.classList.remove('online', 'offline');
        element.classList.add(isOnline ? 'online' : 'offline');
    });
}

}

Файл «/chat/actionHandler»

window.addEventListener('beforeunload', () => {
    stompClient.send("/app/unregister", {}, getCurrentUser());
});

document.addEventListener('DOMContentLoaded', function() {
    connect();

    const searchInput = document.getElementById('searchInput');
    const searchButton = document.getElementById('searchButton');
    const cancelSearchButton = document.getElementById('cancelSearchButton');
    const usersList = document.getElementById('usersList');

    usersList.addEventListener('click', function(e) {
        const listItem = e.target.closest('.listItem');
        if (listItem) {
            const userId = listItem.getAttribute('data-user-id');
            const userName = listItem.querySelector('.userName').textContent;
            const userImage = listItem.querySelector('.userImage img').src;

            stompClient.send("/app/allMessagesRead", {}, JSON.stringify({
                sender: getCurrentUser(),
                opponent: userId
            }));

            const lastMessageElement =
listItem.querySelector('.lastMessage.newMessage');
            if (lastMessageElement) {lastMessageElement.className =
'lastMessage';}

            saveChatState(userId, userName, userImage);
            openChat(userName, userImage, userId);

```

```

    }
  });

  searchButton.addEventListener('click', function () {
    resetUserList();
    performSearch();
  });

  searchInput.addEventListener('keydown', function(e) {
    if (e.key === 'Enter') {
      e.preventDefault();
      resetUserList();
      performSearch();
    }
  });

  cancelSearchButton.addEventListener('click', function() {
    const searchInput = document.getElementById('searchInput');
    searchInput.value = '';
    this.style.display = 'none';

    const noResults = document.querySelector('.no-results');
    if (noResults) {
      noResults.remove();
    }

    resetUserList();
  });

  const currentChatUser = localStorage.getItem('currentChatUser');
  if (currentChatUser !== null) {
    localStorage.removeItem('currentChatUser');
    const selectedUsername = document.getElementById('selectedUsername');
    const userId = selectedUsername.getAttribute('data-user-id');
    const userImage = '/uploads/' + selectedUsername.getAttribute('data-userImage');
    const userName = selectedUsername.getAttribute('data-username');

    saveChatState(userId, userName, userImage);
    openChat(userName, userImage, userId);
  } else {
    const savedState = loadChatState();
    if (savedState) {
      restoreChat(savedState.userName, savedState.userImage,
savedState.userId);
    }
  }

  resetUserList();
});

document.querySelectorAll('.listItem').forEach(item => {
  item.addEventListener('click', function() {
    const userId = this.getAttribute('data-user-id');
    const userName = this.querySelector('.userName').textContent;
    const userImage = this.querySelector('.userImage img').src;

    saveChatState(userId, userName, userImage);

    openChat(userName, userImage, userId);
  });
});

document.addEventListener('click', function(e) {

```

```

        if (e.target.closest('#closeChatBtn')) {
            document.getElementById('chatHeader').innerHTML = '';
            document.getElementById('chatBody').innerHTML = `<div
class="noDialogMessage">Выберите пользователя, чтобы начать общение</div>`;
            document.getElementById('chatInput').innerHTML = '';
            clearChatState();
        }
    });

Файл «/chat/chatState»
function restoreChat(userName, userImage, userId) {
    openChat(userName, userImage, userId);
}

function saveChatState(userId, userName, userImage) {
    localStorage.setItem('currentChat', JSON.stringify({
        userId: userId,
        userName: userName,
        userImage: userImage,
        timestamp: new Date().getTime()
    }));

    const lastMessages = JSON.parse(localStorage.getItem('lastMessages') || '{}');
    lastMessages[userId] = new Date().getTime();
    localStorage.setItem('lastMessages', JSON.stringify(lastMessages));
}

function loadChatState() {
    const savedChat = localStorage.getItem('currentChat');
    if (savedChat) {
        return JSON.parse(savedChat);
    }
    return null;
}

function clearChatState() {
    localStorage.removeItem('currentChat');
}

Файл «/chat/message»
let typingTimer;
const TYPING_DELAY = 1000;

function getCurrentUser() {
    const element = document.getElementById('currentUser');
    return element ? element.getAttribute('data-username') : null;
}

function openChat(userName, userImage, userId) {
    const notificationDiv = document.getElementById('notification');
    notificationDiv.innerHTML = '';

    const typingIndicator = document.createElement('div');
    typingIndicator.className = 'typing-indicator';
    typingIndicator.style.display = 'none';
    typingIndicator.textContent = 'печатает...';
    notificationDiv.appendChild(typingIndicator);

    const listItems = document.querySelectorAll('.listItem');
    const userIds = [];
    listItems.forEach(item => {
        userIds.push(item.dataset.userId);
    });
    if (userIds.includes(userId)) {

```



```

        console.log('Пользователь найден');
    } else {
        createNewDialog(userId, userName, userImage);
    }

    const userImageElement = document.querySelector(`.listItem[data-user-id=${userId}] .userImage`);
    let userStatus = "userImage";
    const classes = userImageElement.className.split(' ');
    if (classes.length > 1) {
        userStatus = "userImage " + classes[1];
    }

    const chatHeader = document.getElementById('chatHeader');
    chatHeader.innerHTML = `
        <p class="${userStatus}"></p>
        <p class="opponentName ${userId}" id="opponent">${userName}</p>
        <button type="button" id="closeChatBtn">
            
        </button>
    `;
    const chatInput = document.getElementById('chatInput');
    chatInput.innerHTML = `
        <button class="docButton" type="submit"
id="docButton"></button>
        <input type="file" id="fileInput" style="display: none;">
        <textarea class="auto-resize-textarea" id="message"
rows="1"></textarea>
        <button class="sendButton" onclick="handleMessage()"></button>
    `;

    const textarea = document.getElementById('message');
    textarea.addEventListener('keydown', function(e) {
        if (e.key === 'Enter' && !e.altKey && !e.shiftKey && !e.ctrlKey) {
            e.preventDefault();
            handleMessage();
        }
    });
    textarea.addEventListener('input', function() {
        clearTimeout(typingTimer);

        const isTyping = this.value.length >= 0;
        sendTypingNotification(true, userId);

        if (isTyping) {
            typingTimer = setTimeout(() => {
                sendTypingNotification(false, userId);
            }, TYPING_DELAY);
        }
    });
    document.getElementById('docButton').addEventListener('click', function() {
        document.getElementById('fileInput').click();
    });
    document.getElementById('fileInput').addEventListener('change', function(e) {
        const file = e.target.files[0];
        if (file) {
            uploadFile(file);
        }
    });

    loadChatMessages(userId);
}

```

```

function sendTypingNotification(isTyping, userId) {
    stompClient.send("/app/typing", {},
        JSON.stringify({
            recipient: userId,
            isTyping: isTyping
        })
    );
}

function createNewDialog(userId, userName, userImage) {
    const chatBody = document.getElementById('chatBody');
    chatBody.innerHTML = ``;

    const usersList = document.getElementById('usersList');
    const newListItem = document.createElement('div');
    newListItem.className = 'listItem';
    newListItem.id = `listItem ${userId}`;
    console.log("userImage " + userImage);
    if (userImage === "/uploads/null") {
        userImage = "/img/user.png";
        console.log("userImage " + userImage);
    }
    console.log(userImage);
    newListItem.setAttribute('data-user-id', `${userId}`);
    newListItem.setAttribute('data-user-name', `${userName}`);
    newListItem.setAttribute('data-search-text', `${userId} ${userName}`);
    newListItem.innerHTML = `
        <p class="userImage">
            <img src='${userImage}'>
        </p>
        <p class="userName">${userName}</p>
        <p class="lastMessage"></p>
    `;
    usersList.appendChild(newListItem);
}

function handleMessage() {
    const textarea = document.getElementById('message');
    const content = textarea.value.trim();

    const lastMessages = JSON.parse(localStorage.getItem('lastMessages') || '{}');
    lastMessages[getCurrentUser()] = new Date().getTime();
    localStorage.setItem('lastMessages', JSON.stringify(lastMessages));

    const savedChat = JSON.parse(localStorage.getItem('currentChat'));
    const recipientUS = savedChat.userId;

    stompClient.send("/app/handleMessage", {},
        JSON.stringify({
            content: content,
            recipient: recipientUS
        })
    );

    textarea.value = '';
}

function uploadFile(file) {
    const formData = new FormData();
    formData.append('file', file);

    const savedChat = loadChatState();
    if (!savedChat) return;
}

```

```

    formData.append('recipient', savedChat.userId);

    const csrfToken = document.cookie.replace(/(?:?:(?:^|.*;)\s*)XSRF-TOKEN\s*\s*\s*(?:[^\s]*|.*$)|^.*$/ , '$1');

    fetch('/upload', {
      method: 'POST',
      body: formData,
      headers: {
        'X-XSRF-TOKEN': csrfToken
      },
      credentials: 'include'
    })
      .then(response => {
        if (!response.ok) throw new Error('Upload failed');
        return response.json();
      })
      .then(data => {
        if (data.success) {
          // Сообщение уже сохранено и отправлено сервером
          console.log('File uploaded and message saved', data.message);
        }
      })
      .catch(error => {
        console.error('Upload error:', error);
        alert('Ошибка загрузки файла');
      });
  }

  function loadChatMessages(recipientId) {
    stompClient.send("/app/allMessagesRead", {}, JSON.stringify({
      sender: getCurrentUser(),
      opponent: recipientId
    }));
    stompClient.send("/app/requestMessages", {},
      JSON.stringify({
        otherUser: recipientId
      })
    );

    stompClient.subscribe('/topic/messages', function (message) {
      const messages = JSON.parse(message.body);
      displayMessages(messages);
    });
  }

  function displayMessages(messages) {
    const chatBody = document.getElementById('chatBody');

    if (Array.isArray(messages)) {
      chatBody.innerHTML = '';

      messages.forEach(msg => displaySingleMessage(msg));
    } else {
      displaySingleMessage(messages);
    }
  }

  function displaySingleMessage(msg) {
    const chatBody = document.getElementById('chatBody');
    const currentUser = getCurrentUser();

    const otherUser = msg.sender === currentUser ? msg.recipient : msg.sender;
  }

```

```

updateLastMessageInList(otherUser, msg);

const parentDiv = document.createElement('div')
const messageDiv = document.createElement('div');
const messageMenu = document.createElement('div');

parentDiv.className = `parentMessages ${msg.id}`;
messageDiv.className = msg.sender === currentUser ? 'messages sender' :
'messages recipient';
if (currentUser === msg.sender) {
    parentDiv.id=`parentMessages${msg.id}`;
    messageDiv.id=`message${msg.id}`;
    messageDiv.className = msg.isRead ? 'messages sender' : 'messages sender
unchecked';
    messageMenu.className = 'messageMenu sender';

    messageMenu.innerHTML = `
        <div class="item change ${msg.id}" id="change"></div>
        <div class="item delete" id="delete"></div>
    `;
    parentDiv.appendChild(messageMenu);
}
messageDiv.innerHTML = `
    <p class="messagesText">${msg.content}</p>
    <p class="messagesTime">${formatTime(msg.timestamp)}</p>
    <span class="cool-tooltip-text">${formatDate(msg.timestamp)}</span>
`;

parentDiv.appendChild(messageDiv);
chatBody.appendChild(parentDiv);
chatBody.scrollTop = chatBody.scrollHeight;

parentDiv.addEventListener('click', function(e) {
    if (e.target.closest('.delete')) {
        console.log("Message id:", msg.id);
        stompClient.send("/app/deleteMessage", {}, msg.id);
        const parentDiv = document.getElementById(`parentMessages${msg.id}`);
        chatBody.removeChild(parentDiv);
    }
});

parentDiv.addEventListener('click', function(e) {
    if (e.target.closest('.change')) {
        console.log("Message id:", msg.id);

        const changeMessage = document.getElementById(`message${msg.id}`);
        const messageText =
changeMessage.querySelector('.messagesText').textContent;
        changeMessage.innerHTML = `
            <textarea class="auto-resize-textarea" id="newMessage"
rows="1">${messageText}</textarea>
        `;
        const newMessage = document.getElementById('newMessage');
        newMessage.addEventListener('keydown', function(e) {
            if (e.key === 'Enter' && !e.altKey && !e.shiftKey && !e.ctrlKey) {
                e.preventDefault();
                const content = newMessage.value.trim();
                if (content !== msg.content) {
                    stompClient.send("/app/updateMessage", {},
JSON.stringify({
                        id: msg.id,
                        content: content
                    }));
                }
            }
        });
    }
});

```

```

        }));
        changeMessage.innerHTML = `
            <p class="messagesText">${content}</p>
            <p
class="messagesTime">${formatTime(msg.timestamp)}</p>
            <span class="cool-tooltip-
text">${formatDate(msg.timestamp)}</span>
            `;
    } else {
        changeMessage.innerHTML = `
            <p class="messagesText">${msg.content}</p>
            <p class="messagesTime"
title="Подсказка">${formatTime(msg.timestamp)}</p>
            <span class="cool-tooltip-
text">${formatDate(msg.timestamp)}</span>
            `;
    }
}
});
}
});
}

function updateLastMessageInList(userId, message) {
    const userElement = document.querySelector(`.listItem[data-user-
id="${userId}"]`);
    const sender = message.sender;
    const currentUser = getCurrentUser();

    if (userElement) {
        const lastMessageElement = userElement.querySelector('.lastMessage');
        lastMessageElement.id = `lastMessage${message.id}`;
        if (lastMessageElement) {
            if (sender === currentUser) {
                lastMessageElement.textContent = message.content.length > 20 ?
'Бы: ' + message.content.substring(0, 20) + '...' : 'Бы: ' + message.content;
                lastMessageElement.className = message.isRead ? 'lastMessage' :
'lastMessage unchecked';
            } else {
                lastMessageElement.textContent = message.content.length > 20 ?
message.content.substring(0, 20) + '...' : message.content;
                lastMessageElement.className = message.isRead ? 'lastMessage' :
'lastMessage newMessage';
            }
        }
    }
}

function updateMessageStatus(id) {
    const msg = document.getElementById(`message${id}`);
    msg.className = 'messages sender';
    const lastMsg = document.getElementById(`lastMessage${id}`);
    lastMsg.className = 'lastMessage';
}

function formatTime(timestamp) {
    const date = new Date(timestamp);
    return date.toLocaleTimeString([], {hour: '2-digit', minute:'2-digit'})
}

function formatDate(timestamp) {
    const date = new Date(timestamp);
    return date.toLocaleDateString([], {year: '2-digit', month:'2-digit', day:'2-
digit'})
}

```

```

}

Файл «/chat/searchUsers»
function performSearch() {
    const searchTerm =
document.getElementById('searchInput').value.trim().toLowerCase();
    const cancelButton = document.getElementById('cancelSearchButton');
    const users = document.querySelectorAll('.listItem');
    let hasResults = false;

    document.querySelectorAll('.no-results').forEach(el => el.remove());

    const visibleUsers = Array.from(document.querySelectorAll('.listItem'))
        .filter(user => user.style.display !== 'none');

    const sortedUsers = sortUsersByLastMessage(visibleUsers);
    const userList = document.getElementById('usersList');
    userList.innerHTML = '';
    sortedUsers.forEach(user => userList.appendChild(user));

    if (!searchTerm) {
        resetUserList();
        cancelButton.style.display = 'none';
        return;
    }

    users.forEach(user => {
        const userName =
user.querySelector('.userName').textContent.toLowerCase();
        const searchText = user.getAttribute('data-search-text').toLowerCase();

        if (userName.includes(searchTerm) || searchText.includes(searchTerm)) {
            user.style.display = 'grid';
            hasResults = true;
        } else {
            user.style.display = 'none';
        }
    });

    if (!hasResults) {
        const noResults = document.createElement('div');
        noResults.className = 'no-results';
        noResults.id = 'tmpText';
        noResults.textContent = 'Пользователи не найдены';
        document.getElementById('usersList').appendChild(noResults);
    }

    cancelButton.style.display = 'flex';
}

function resetUserList() {
    const userList = document.getElementById('usersList');
    const users = document.querySelectorAll('.listItem');
    const sortedUsers = sortUsersByLastMessage(users);

    userList.innerHTML = '';
    sortedUsers.forEach(user => {
        userList.appendChild(user);
        user.style.display = 'grid';
    });
}

function sortUsersByLastMessage(users) {
    const lastMessages = JSON.parse(localStorage.getItem('lastMessages') || '{}');

```

```

    return Array.from(users).sort((a, b) => {
        const userIdA = a.getAttribute('data-user-id');
        const userIdB = b.getAttribute('data-user-id');

        const timeA = lastMessages[userIdA] || 0;
        const timeB = lastMessages[userIdB] || 0;

        return timeB - timeA;
    });
}

Файл «/chat/wsOperation»
let stompClient = null;
let isConnected = false;

function connect() {
    const socket = new SockJS('/ws');
    stompClient = Stomp.over(socket);

    stompClient.connect({}, () => {
        isConnected = true;

        stompClient.send("/app/requestAllLastMessages", {}, getCurrentUser());

        stompClient.subscribe("/user/queue/messageIsRead", function (message) {
            const msg = JSON.parse(message.body);
            updateMessageStatus(msg);
        });

        stompClient.subscribe("/user/queue/allMessagesIsRead", function (message)
        {
            const msgIds = JSON.parse(message.body);
            msgIds.forEach(id => {
                updateMessageStatus(id);
            });
        });

        stompClient.subscribe('/user/queue/allLastMessages', function(message) {
            const lastMessages = JSON.parse(message.body);

            Object.entries(lastMessages).forEach(([userId, messageData]) => {
                updateLastMessageInList(userId, messageData);

                const lastMessagesStorage =
JSON.parse(localStorage.getItem('lastMessages') || '{}');
                lastMessagesStorage[userId] = messageData.timestamp;
                localStorage.setItem('lastMessages',
JSON.stringify(lastMessagesStorage));
            });

            resetUserList();
        });

        stompClient.subscribe('/user/queue/messages', function(message) {
            const msg = JSON.parse(message.body);

            const sender = msg[0].sender;
            const recipient = msg[0].recipient;
            const opponent =
document.getElementById('opponent').classList.toString();
            const otherUser = sender === currentUser ? recipient : sender;

            console.log( recipient + ' === ' + currentUser);

```

```

        console.log( 'opponentName ' + otherUser + ' === ' + opponent);

        const lastMessages = JSON.parse(localStorage.getItem('lastMessages'))
        || '{}';
        lastMessages[msg[0].sender] = new Date().getTime();
        localStorage.setItem('lastMessages', JSON.stringify(lastMessages));

        if ('opponentName ' + otherUser === opponent) {
            displayMessages(msg);
        }
        resetUserList();
    });

    stompClient.subscribe('/user/queue/newDialog', function (message) {
        const msg = JSON.parse(message.body);
        const userName = msg.firstName + ' ' + msg.lastName;
        const userImage = '/uploads/' + msg.photoPath;
        console.log(msg);
        console.log(msg.username + ' ' + userName + ' ' + userImage)
        createNewDialog(msg.username, userName, userImage);
        stompClient.send("/app/requestStatuses", {}, getCurrentUser());
    });

    stompClient.subscribe('/user/queue/newMessages', function(message) {
        let msg = JSON.parse(message.body);
        const recipient = msg.recipient;

        const sender = msg.sender;
        const currentUser = getCurrentUser();
        const opponent =
document.getElementById('opponent').classList.toString();

        const lastMessages = JSON.parse(localStorage.getItem('lastMessages'))
        || '{}';
        lastMessages[msg.sender] = new Date().getTime();
        localStorage.setItem('lastMessages', JSON.stringify(lastMessages));

        const otherUser = sender === currentUser ? recipient : sender;
        updateLastMessageInList(otherUser, msg);

        if ((recipient === currentUser && 'opponentName ' + sender ===
opponent) || sender === currentUser) {
            if (sender !== currentUser) {
                msg.isRead = true;
                stompClient.send("/app/msgStatus", {}, JSON.stringify({
                    recipient: sender,
                    msgId: msg.id,
                    msgStatus: msg.isRead
                }));
            }
            displayMessages(msg);
        }
        resetUserList();
    });

    stompClient.subscribe('/user/queue/statuses', function(message) {
        const statuses = JSON.parse(message.body);
        Object.entries(statuses).forEach(([username, isOnline]) => {
            updateUserStatus(username, isOnline);
        });
    });

    stompClient.subscribe(`/user/queue/typing`, function(message) {
        const notification = JSON.parse(message.body);

```



```

        const currentChat = loadChatState();

        if (!currentChat || notification.sender !== currentChat.userId)
return;

        const typingIndicator = document.querySelector('#notification .typing-
indicator');
        if (typingIndicator) {
            typingIndicator.style.display = notification.isTyping ? 'block' :
'none';
        }
    });

    const savedChat = loadChatState();
    if (savedChat) {
        loadChatMessages(savedChat.userId);
    }
    const currentUser = getCurrentUser();
    stompClient.send("/app/register", {}, currentUser);
    stompClient.send("/app/requestStatuses", {}, currentUser);

});

function updateUserStatus(username, isOnline) {
    const userElements = document.querySelectorAll(`.listItem[data-user-
id="${username}"] .userImage`);
    userElements.forEach(element => {
        element.classList.remove('online', 'offline');
        element.classList.add(isOnline ? 'online' : 'offline');
    });

    const savedChat = loadChatState();
    if (savedChat && savedChat.userId === username) {
        const chatHeaderImage = document.querySelector('#chatHeader
.userImage');
        if (chatHeaderImage) {
            chatHeaderImage.classList.remove('online', 'offline');
            chatHeaderImage.classList.add(isOnline ? 'online' : 'offline');
        }
    }
}

}

Файл «/diary/actionHandler»
document.addEventListener('DOMContentLoaded', function() {
    connect();

    let cards = document.querySelectorAll('.diaryCard');
    let btnLeft = document.querySelector('.navButton.left');
    let btnRight = document.querySelector('.navButton.right');

    let currentIndex = 0;

    function updateCardsList() {
        cards = document.querySelectorAll('.diaryCard');
    }

    function goToCard(index) {
        if (cards.length === 0) return;

        if (index < 0) index = cards.length - 1;
        if (index >= cards.length) index = 0;
    }

```

```

        currentIndex = index;
        cards.forEach((card, i) => {
            card.style.transform = `translateX(-${currentIndex * 100}%)`;
        });
    }

    btnLeft.addEventListener('click', () => goToCard(currentIndex - 1));
    btnRight.addEventListener('click', () => goToCard(currentIndex + 1));

    document.addEventListener('keydown', (e) => {
        if (e.key === 'ArrowLeft') goToCard(currentIndex - 1);
        if (e.key === 'ArrowRight') goToCard(currentIndex + 1);
    });

    const createNewDiaryCard = document.getElementById('createNewDiaryCard');
    createNewDiaryCard.addEventListener('click', function () {
        const createDiaryCard = document.getElementById('createDiaryCard');
        const cardsContainer = document.getElementById(`cardsContainer`);
        const existingDivs =
            cardsContainer.querySelectorAll('.diaryCard.exist').length;
        if (existingDivs > 10) {
            createDiaryCard.style.display = "none";
            alert("Число карточек ограничено! Удалите ненужные.");
            return;
        }

        const now = formatDate();
        const cardId = 'new_' + Date.now();

        const newDiaryCard = document.createElement('div');
        newDiaryCard.className = 'diaryCard exist';
        newDiaryCard.id = cardId;
        newDiaryCard.setAttribute('data-id', cardId);
        newDiaryCard.innerHTML = `
<div class="cardTitle">
    <input class="title" type="text" value="Заголовок">
    <p class="timeCreateCard">${now}</p>
</div>

    <div class="cardBody" id="cardBody${cardId}">
        <div class="cardItemsContainer"
id="cardItemsContainer${cardId}">
        </div>
        <button class="createNewParagraph" data-id="${cardId}"></button>

        <div class="cardAction">
            <button class="saveCard" id="saveCard${cardId}"></button>

            <div class="empty"></div>
            <button class="deleteCard" id="deleteCard"></button>

        </div>
    </div>
`;
        cardsContainer.insertBefore(newDiaryCard,
            createNewDiaryCard.parentElement);
    }

```

```

        updateCardsList();
        goToCard(cards.length - 1);
    });

    document.addEventListener('click', function (e) {
        if (e.target.closest('.createNewParagraph')) {
            const button = e.target.closest('.createNewParagraph');
            const cardId = button.dataset.id;
            const paragraphId = 'new_' + Date.now();

            const cardItemsContainer =
document.getElementById(`cardItemsContainer${cardId}`);
            const existingDivs =
cardItemsContainer.querySelectorAll('.parentParagraph').length;
            if (existingDivs > 16) {
                alert("Число задач на карточку ограничено! Продолжите список в
другой.");
                return;
            }

            const parentParagraph = document.createElement('div');
            parentParagraph.className = `parentParagraph`;
            parentParagraph.id = `parentParagraph${paragraphId}`;
            parentParagraph.innerHTML = `
                <input class="taskStatus" type="checkbox">
                <input class="task" type="text">
                <button class="deleteParagraph" data-
id="${paragraphId}" data-card-id="${cardId}"></button>
            `;

            cardItemsContainer.appendChild(parentParagraph);
        }

        if (e.target.closest('.deleteParagraph')) {
            const button = e.target.closest('.deleteParagraph');
            const paragraphId = button.dataset.id;
            const cardId = button.dataset.cardId;
            document.getElementById(`saveCard${cardId}`).className = "saveCard";

            document.getElementById(`parentParagraph${paragraphId}`).remove();
        }

        if (e.target.closest('.saveCard')) {
            const button = e.target.closest('.saveCard');
            button.className = "saveCard inactive";
            const cardId = button.id.replace('saveCard', '');

            const cardData = collectCardData(cardId);
            saveCardToServer(cardData);
        }

        if (e.target.classList.contains('title')) {
            const card = e.target.closest('.diaryCard.exist');
            const cardId = card.dataset.id;
            document.getElementById(`saveCard${cardId}`).className = "saveCard";
        }

        if (e.target.classList.contains('task')) {
            const parentParagraph = e.target.closest('.parentParagraph');
            const card = e.target.closest('.diaryCard.exist');
            const cardId = card.dataset.id;
            document.getElementById(`saveCard${cardId}`).className = "saveCard";
        }
    });

```

```

    }

    if (e.target.classList.contains('taskStatus')) {
        const parentParagraph = e.target.closest('.parentParagraph');
        const card = e.target.closest('.diaryCard.exist');
        const cardId = card.dataset.id;
        document.getElementById(`saveCard${cardId}`).className = "saveCard";
    }

    if (e.target.closest('.deleteCard')) {
        const button = e.target.closest('.deleteCard');
        const card = button.closest('.diaryCard.exist');
        const cardId = card.id;

        if (confirm('Вы уверены, что хотите удалить эту карточку?')) {
            deleteCardOnServer(cardId);
            card.remove();
            updateCardsList();
        }
    }
});

});

function collectCardData(cardId) {
    const card = document.getElementById(cardId);
    if (!card) {return null;}

    const userId = document.getElementById('currentUser').dataset.id;

    const isNewCard = cardId.startsWith('new_');

    const cardData = {
        id: isNewCard ? null : cardId,
        userId: parseInt(userId),
        title: card.querySelector('.title').value,
        date: isNewCard ? new Date().toISOString().split('T')[0] : null,
        diaryParagraphs: []
    };

    const paragraphsContainer =
document.getElementById(`cardItemsContainer${cardId}`);
    if (paragraphsContainer) {
        const paragraphs =
paragraphsContainer.querySelectorAll('.parentParagraph');

        paragraphs.forEach(paragraph => {

            const taskInput = paragraph.querySelector('.task');
            const content = taskInput.value.trim();

            if (!content) return;

            const paragraphId = paragraph.id.replace('parentParagraph', '');
            const isNewParagraph = paragraphId.startsWith('new_');

            cardData.diaryParagraphs.push({
                id: isNewParagraph ? null : parseInt(paragraphId),
                content: paragraph.querySelector('.task').value,
                isReady: paragraph.querySelector('.taskStatus').checked
            });
        });
    }
});

```

```

    }

    return cardData;
}

function saveCardToServer(cardData) {
    stompClient.send(
        "/app/saveCard",
        {},
        JSON.stringify(cardData)
    );
}

function deleteCardOnServer(cardId) {
    if (cardId.startsWith('new_')) return;

    const cardData = {
        id: parseInt(cardId),
        userId: parseInt(document.getElementById('currentUser').dataset.id)
    };

    stompClient.send(
        "/app/deleteCard",
        {},
        JSON.stringify(cardData)
    );
}

function formatDate() {
    const date = new Date();
    return date.toLocaleDateString([], {year: 'numeric', month: '2-digit', day: '2-digit'});
}

Файл «/HR/actionHandler»
function getCurrentUser() {
    const element = document.getElementById('currentUser');
    return element ? element.getAttribute('data-username') : null;
}

document.addEventListener('DOMContentLoaded', function() {
    connect();

    const addNewUserRequest = document.getElementById('addNewUserRequest');
    const deleteUserRequest = document.getElementById('deleteUserRequest');

    addNewUserRequest.addEventListener('click', function () {
        document.getElementById('menu').style.display="none";
        document.getElementById('createUser').style.display = "flex";
    });

    deleteUserRequest.addEventListener('click', function () {
        document.getElementById('menu').style.display="none";
        document.getElementById('deleteUser').style.display = "flex";
    });

    const backToMenuFromCreate = document.getElementById('returnBackCreate');
    const backToMenuFromDelete = document.getElementById('returnBackDelete');

    backToMenuFromCreate.addEventListener('click', function () {
        document.getElementById('createUser').style.display = "none";
        document.getElementById('menu').style.display="flex";
    });
});

```

```

backToMenuFromDelete.addEventListener('click', function () {
    document.getElementById('deleteUser').style.display = "none";
    document.getElementById('menu').style.display="flex";
});

const submitNewUser = document.getElementById('submitNewUser');
const submitDeleteUser = document.getElementById('submitDeleteUser');

submitNewUser.addEventListener('click', function () {
    const firstName = document.getElementById('inputFirstname').value.trim();
    const lastName = document.getElementById('inputLastname').value.trim();
    const position = document.getElementById('inputPosition').value.trim();
    const hiringDate =
document.getElementById('inputHiringDate').value.trim();

    if (firstName && lastName && position && hiringDate) {
        stompClient.send("/app/createNewUserRequest", {}, JSON.stringify({
            firstName: firstName,
            lastName: lastName,
            position: position,
            hiringDate: hiringDate
        }));
    }
});

submitDeleteUser.addEventListener('click', function () {
    const firstName =
document.getElementById('inputFirstnameDelete').value.trim();
    const lastName =
document.getElementById('inputLastnameDelete').value.trim();
    const position =
document.getElementById('inputPositionDelete').value.trim();
    const hiringDate =
document.getElementById('inputHiringDateDelete').value.trim();

    if (firstName && lastName && position && hiringDate) {
        stompClient.send("/app/deleteUserRequest", {}, JSON.stringify({
            firstName: firstName,
            lastName: lastName,
            position: position,
            hiringDate: hiringDate
        }));
    }
});
});

Файл «/HR/wsOperation»
let stompClient = null;
let isConnected = false;

function connect() {
    const socket = new SockJS('/ws');
    stompClient = Stomp.over(socket);

    stompClient.connect({}, () =>{
        isConnected = true;

        stompClient.subscribe("/user/queue/actionRequest", function (message) {
            console.log(message.body);
            if (message.body === "Запрос успешно создан") {
                alert(message.body);
                document.getElementById('createUser').style.display = "none";
                document.getElementById('deleteUser').style.display = "none";
                document.getElementById('menu').style.display = "flex";
            }
        });
    });
}

```

```

        } else {
            alert(message.body);
        }
    });
});
}

Файл «activation»
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Активация</title>
    <link rel="stylesheet" th:href="@{/style/activationStyle.css}" />
    <link rel="stylesheet" th:href="@{/style/style.css}" />
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://cdnjs.cloudflare.com/ajax/libs/cropperjs/1.5.12/cropper.min.css"
rel="stylesheet">
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/cropperjs/1.5.12/cropper.min.js"></scr
ipt>
</head>
<body>
    <div class="container">
        <div th:replace="~{staticElement :: sidebar}"></div>

        <form class="mainContentContainer" th:action="@{/activation}"
method="post" enctype="multipart/form-data">
            <div class="mainInfo">
                <input type="hidden" name="userId" th:value="${userId}">
                <h3>Основная информация</h3>

                <div class="staticData">
                    <p>Имя*</p>
                    <p>Фамилия*</p>
                    <p>Должность*</p>
                    <p>Год рождения</p>
                    <p>Пол</p>
                </div>

                <div class="dynamicData">
                    <input type="text" name="firstname"
th:value="${usersInfo.firstname}" required>
                    <input type="text" name="lastname"
th:value="${usersInfo.lastname}" required>
                    <input type="text" name="position"
th:value="${usersInfo.position}" required>
                    <input type="date" name="dob" th:value="${usersInfo.dob !=
null} ? ${#temporals.format(usersInfo.dob, 'yyyy-MM-dd')} : ''">
                    <select name="gender">
                        <option value="none" th:selected="${usersInfo.gender ==
'Не указано'}">Не указано</option>
                        <option value="male" th:selected="${usersInfo.gender ==
'Мужской'}">Мужской</option>
                        <option value="female" th:selected="${usersInfo.gender ==
'Женский'}">Женский</option>
                    </select>
                </div>
            </div>

            <div class="placeInfo">
                <h3>Информация о рабочем месте</h3>

```

```

        <div class="staticData">
            <p>Город*</p>
            <p>Улица</p>
            <p>Этаж</p>
            <p>Кабинет</p>
        </div>

        <div class="dynamicData">
            <input type="text" name="city" th:value="\${usersInfo.city}"
required>
            <input type="text" name="street"
th:value="\${usersInfo.street}">
            <input type="text" name="officeFloor"
th:value="\${usersInfo.officeFloor}">
            <input type="text" name="officeNumber"
th:value="\${usersInfo.officeNumber}">
        </div>
    </div>

    <div class="contactInfo">
        <h3>Контактные данные</h3>

        <div class="staticData">
            <p>Номер телефона</p>
            <p>Почта*</p>
            <p>ID ВК</p>
            <p>Имя пользователя Телеграм</p>
        </div>

        <div class="dynamicData">
            <input type="tel" name="phoneNumber"
th:value="\${usersContact.phoneNumber}">
            <input type="text" name="email"
th:value="\${usersContact.email}" required>
            <input type="text" name="VKid"
th:value="\${usersContact.VKid}">
            <input type="text" name="TelegramUsername"
th:value="\${usersContact.telegramUsername}">
        </div>
    </div>

    <div class="avatar-upload">
        <div class="avatar-edit">
            <input type="file" id="imageUpload" name="photo"
accept="image/*" th:value="\${usersInfo.photoPath}"/>
            <label for="imageUpload"></label>
        </div>
        <div class="avatar-preview">
            <div id="imagePreview" th:style="\${usersInfo.photoPath !=
null} ?
                'background-image: url(/uploads/' + \${usersInfo.photoPath} +
                '):' :
                'background-image: url(..img/user.png)'"></div>
        </div>
    </div>
    <div class="confirm">
        <button type="submit"></button>
    </div>
</form>
</div>
<script>
    // отключение ссылок на другие страницы
    const navigation = document.querySelectorAll('#navigation .link');

```



```

        navigation.forEach(element => {
            element.style.display = 'none';
            element.addEventListener('click',function (e) {
                e.preventDefault();
            })
        });

        document.getElementById("imageUpload").addEventListener("change",
function(event) {
    const file = event.target.files[0];
    if (file) {
        const reader = new FileReader();
        reader.onload = function(e) {
            document.getElementById("imagePreview").style.backgroundImage =
"url(" + e.target.result + ")";
        };
        reader.readAsDataURL(file);
    }
});

</script>
</body>
</html>

```

Файл «admin»

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <link rel="stylesheet" th:href="@{/style/style.css}">
    <link rel="stylesheet" th:href="@{/style/adminStyle.css}">
    <meta charset="UTF-8">
    <title>Администратор</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <script src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.5.0/dist/sockjs.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
    <script src="/js/admin/actionHandler.js"></script>
    <script src="/js/admin/searchUsers.js"></script>
    <script src="/js/admin/wsConnect.js"></script>
</head>
<body>

<div class="container">
    <div th:replace=~{staticElement :: sidebar}></div>

    <div class="mainContainer">
        <div id="currentUser" th:data-username="${currentUser.username}"
style="display: none;"></div>
        <div class="adminPanel">
            <div class="menu">
                <button class="selectUsers" id="usersButton">Пользователи</button>
                <button class="selectHR" id="hrButton">Запросы HR</button>
                <button class="selectPassword" id="passwordButton">Запросы
Пароль</button>
            </div>

            <div class="mainView" id="mainViewUsers" style="display: grid;">
                <div class="viewHeader">Управление пользователями</div>

                <div class="view Users">
                    <div class="searchUsers">

```

```

        <button type="button" id="cancelSearchButton"
style="display: none;"></button>
        <button class="startSearch" id="searchButton"
type="submit"></button>
        <input class="searchUser" id="searchInput" type="text">

        <input id="haveActivation" type="checkbox">
        <p for="haveActivation">Активированные</p>
    </div>

    <div class="usersList" id="usersList">
        <button class="addUserButton" id="addUserButton"
title="Добавить пользователя">
            
        </button>
        <div class="listItem" th:each="user : ${usersAll}"
            th:attr="class=${user.admin} ? 'listItem admin' :
'listItem'"
                th:if="${user.username} != ${currentUser.username}"
                th:id="'listItem-' + ${user.username}"
                th:user-id="${user.username}"
                th:data-search-text="${user.firstName + ' ' +
user.lastName + ' ' + user.username+ ' ' + user.id.toString()}"
                th:data-activation="${user.activation}"
                <p class="id" th:text="${user.id.toString()}"></p>
                <p class="userName" th:text="${user.username}"></p>
                <p class="name" th:text="${user.firstName + ' ' +
user.lastName}">First Last</p>
                <p class="position"
th:text="${user.position}">Position</p>
                <p class="activation" th:if="${user.activation}"
>Активирован</p>
                <p class="activation" th:unless="${user.activation}"
>Не активирован</p>
                <a th:if = "${user.activation}"
th:href="@{/profile/{username} (username=${user.username})}" class="userAction
seeUserProfile">
                    
                </a>
                <a th:unless = "${user.activation}" class="userAction
seeUserProfile inactive">
                    
                </a>
                <button class="userAction deleteUser" th:data-
username="${user.username}">
                    
                </button>
            </div>
        </div>
    </div>

</div>

<div class="mainView" id="mainViewHR" style="display: none;">
    <div class="viewHeader">Запросы от HR</div>

    <div class="view HR" id="viewHR">
        <div class="radioButtonsContainer">
            <button class="checked"
id="createUserRequests">Создание</button>
            <button class="unchecked"
id="deleteUserRequests">Удаление</button>

```

```

</div>

<div class="itemsContainer" id="createUserItems"
style="display: block;">
    <div class="listItem HRRequest" th:each="hrRequest :
${hrRequests}"
        th:if="${!hrRequest.isDelete}"
        th:id="'listItem-request-' + ${hrRequest.id}">
            <p class="id" id="requestId" th:data="${hrRequest.id}"
th:text="${hrRequest.id}"></p>
            <p class="firstname" id="requestFirstname"
th:data="${hrRequest.firstname}" th:text="${hrRequest.firstname}"></p>
            <p class="lastname" id="requestLastname"
th:data="${hrRequest.lastname}" th:text="${hrRequest.lastname}"></p>
            <p class="position" id="requestPosition"
th:data="${hrRequest.position}" th:text="${hrRequest.position}"></p>
            <p class="hiringDate" id="requestHiringDate"
th:data="${hrRequest.hiringDate}" th:text="${hrRequest.hiringDate}"></p>

            <button class="userAction acceptCreateRequest"
id="acceptCreateButton" th:request-id="${hrRequest.id}">
                
            </button>
            <button class="userAction cancelCreateRequest"
id="cancelCreateButton" th:request-id="${hrRequest.id}">
                
            </button>
        </div>
    </div>

    <div class="itemsContainer" id="deleteUserItems"
style="display: none;">
        <div class="listItem HRRequest" th:each="hrRequest :
${hrRequests}"
            th:if="${hrRequest.isDelete}"
            th:id="'listItem-request-' + ${hrRequest.id}">
                <p class="id" id="deleteRequestId"
th:data="${hrRequest.id}" th:text="${hrRequest.id}"></p>
                <p class="firstname" id="deleteRequestFirstname"
th:data="${hrRequest.firstname}" th:text="${hrRequest.firstname}"></p>
                <p class="lastname" id="deleteRequestLastname"
th:data="${hrRequest.lastname}" th:text="${hrRequest.lastname}"></p>
                <p class="position" id="deleteRequestPosition"
th:data="${hrRequest.position}" th:text="${hrRequest.position}"></p>
                <p class="hiringDate" id="deleteRequestHiringDate"
th:data="${hrRequest.hiringDate}" th:text="${hrRequest.hiringDate}"></p>

                <button class="userAction acceptDeleteRequest"
id="acceptDeleteButton" th:request-id="${hrRequest.id}">
                    
                </button>
                <button class="userAction cancelDeleteRequest"
id="cancelDeleteButton" th:request-id="${hrRequest.id}">
                    
                </button>
            </div>
        </div>
    </div>

</div>

<div class="mainView" id="mainViewPassword" style="display: none;">

```

```

<div class="viewHeader">Запросы на пароль</div>

<div class="view Password" id="viewPassword">
  <div class="listItem passwords" th:each="passwordRequest :
${passwordRequests}"
      th:id="'listItem-password-' + ${passwordRequest.username}
+ '-' + ${passwordRequest.id.toString()}"
      th:email="${passwordRequest.email}">
    <p class="id" th:text="${passwordRequest.id}"></p>
    <p class="userName"
th:text="${passwordRequest.username}"></p>
    <p class="email" th:text="${passwordRequest.email}"></p>

    <button class="userAction accept" id="acceptButton"
th:request-id="${passwordRequest.id}">
      
    </button>
    <button class="userAction cancel" id="cancelButton"
th:data-username="${passwordRequest.username}"
      th:request-id="${passwordRequest.id}">
      
    </button>
  </div>
</div>

</div>

<div class="mainView" id="mainViewCreateUser" style="display: none">
  <div class="viewHeader">Создание пользователя</div>

  <form class="view CreateUser">
    <input class="userData Email" id="inputRequestId" type="text"
style="display: none;">

    <p class="label ForUsername">Юзернейм</p>
    <input class="userData Username" id="inputUsername"
type="text" required>

    <p class="label ForPassword">Пароль</p>
    <input class="userData Password" id="inputPassword"
type="text"
      required minlength="5"
      oninvalid="this.setCustomValidity('Заполните это поле
(минимум 5 символов) ')"
      oninput="this.setCustomValidity('')">

    <p class="label ForEmail">Почта</p>
    <input class="userData Email" id="inputEmail" type="text"
required>

    <p class="label ForFirstname">Имя</p>
    <input class="userData Firstname" id="inputFirstname"
type="text" required>

    <p class="label ForLastname">Фамилия</p>
    <input class="userData Lastname" id="inputLastname"
type="text" required>

    <p class="label ForPosition">Должность</p>
    <input class="userData Position" id="inputPosition"
type="text" required>

    <p class="label ForHiringDate">Дата найма</p>

```

```

        <input class="userData HiringDate" id="inputHiringDate"
type="date" required>

        <p class="label ForAdminRole">Права администратора</p>
        <input class="userData AdminRole" id="inputAdminRole"
type="checkbox">

        <button class="submitNewUser" id="submitNewUser"
type="submit">
            
        </button>
    </form>
</div>

<div class="mainView" id="mainViewNewPassword" style="display: none">
    <div class="viewHeader">Новый пароль</div>

    <form class="view CreateNewPassword">
        <p class="label ForPassword">Пароль</p>
        <input class="userData Password" id="inputNewPassword"
type="text"
            required minlength="5"
            oninvalid="this.setCustomValidity('Заполните это поле
(минимум 5 символов) ')"
            oninput="this.setCustomValidity('')">

        <button class="submitNewUser" id="submitNewPassword"
type="submit">
            
        </button>
    </form>
</div>
</div>
</div>
</div>
</body>
</html>

```

```

Файл «allUsers»
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Пользователи</title>
    <link rel="stylesheet" th:href="@{/style/style.css}">
    <link rel="stylesheet" th:href="@{/style/allUsersStyle.css}">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <script src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.5.0/dist/sockjs.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
    <script src="/js/allUsers/wsOperation.js"></script>
    <script src="/js/allUsers/searchUsers.js"></script>
    <script src="/js/allUsers/actionHandler.js"></script>
</head>
<body>
<div class="container">
    <div th:replace="~{staticElement :: sidebar}"></div>

    <div class="mainContainer">
        <div class="allUsers">

```

```

<div id="currentUser" th:data-username="${currentUser.username}"
style="display: none;"></div>

<div class="searchUserWithFilter">
    <button type="button" id="cancelSearchButton" style="display:
none;"></button>
    <button class="startSearch" id="searchButton" type="submit"></button>
    <input class="searchUser" id="searchInput" type="text">
    <button class="openFilter" id="openFilter"></button>
    <button class="openFilter" id="closeFilter" style="display:
none;"></button>
</div>

<div class="filterMenu" id="filterMenu" style="display: none;">

    <p class="positionLabel">Должность</p>
    <select class="filterPosition">
        <option>Не выбрано</option>
        <option th:each="position : ${positions}"
th:text="${position}"></option>
    </select>

    <p class="genderLabel">Пол</p>
    <select class="filterGender">
        <option>Не выбрано</option>
        <option>Мужской</option>
        <option>Женский</option>
    </select>

    <p class="cityLabel">Город</p>
    <select class="filterCity">
        <option>Не выбрано</option>
        <option th:each="city : ${cities}" th:text="${city}"></option>
    </select>

    <div class="isOnline">
        <p for="isOnline">В сети</p>
        <input id="isOnline" type="checkbox">
    </div>

    <button class="filterAction applyFilter" type="submit"></button>
    <button class="filterAction resetFilter" type="submit"></button>
</div>

<div class="usersList" id="usersList">
    <div class="listItem" th:each="user : ${usersAll}"
        th:if="${user.username} != ${currentUser.username}"
        th:id="'listItem-' + ${user.username}"
        th:data-user-id="${user.username}"
        th:data-user-name="${user.firstName + ' ' + user.lastName}"
        th:data-search-text="${user.firstName + ' ' + user.lastName +
' ' + user.username}"
        th:data-dob="${user.dob}"
        th:data-position="${user.position}"
        th:data-gender="${user.gender}"
        th:data-city="${user.city}">
        <p class="userImage">
            

```

```

        
    </p>
    <p class="userName" th:text="{user.firstName + ' ' +
user.lastName}"></p>
    <a th:href="@{/profile/{username} (username={user.username})}"
class="userAction seeUserProfile">
        
    </a>
    <button class="userAction sendMessage"
        th:attr="data-username={user.username}"
        onclick="startChat(this)"></button>
    </div>
</div>
</div>
</div>
</div>
</body>
</html>

```

Файл «chat»

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Чат</title>
    <link rel="stylesheet" th:href="@{/style/chatStyle.css}">
    <link rel="stylesheet" th:href="@{/style/style.css}">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <script src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.5.0/dist/sockjs.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
    <script src="/js/chat/wsOperation.js"></script>
    <script src="/js/chat/message.js"></script>
    <script src="/js/chat/chatState.js"></script>
    <script src="/js/chat/searchUsers.js"></script>
    <script src="/js/chat/actionHandler.js"></script>
</head>
<body>
<div class="container">
    <div th:replace="~{staticElement :: sidebar}"></div>

    <div class="mainContainer">
        <div id="currentUser" th:data-username="{currentUser.username}"
style="display: none;"></div>
        <div id="selectedUsername" th:data-user-id="{selectedUser.username}"
th:data-userImage="{selectedUser.photoPath}" th:data-
username="{selectedUser.firstName} + ' ' + {selectedUser.lastName}"
style="display: none;"></div>
        <div class="mainChatContainer">
            <form class="usersContainer" th:action="@{/chat}" method="get"
id="searchForm" onsubmit="return false;">
                <div class="searchUsers">
                    <button type="button" id="cancelSearchButton" style="display:
none;"></button>
                    <input id="searchInput" type="text" name="q">
                    <button type="submit" id="searchButton"></button>
                </div>

                <div class="usersList" id="usersList">

```

```

        <div th:each="user : ${usersAll}" th:if="${user.username} !=
${currentUser.username}" class="listItem" th:id="'listItem-' + ${user.username}"
        th:data-user-id="${user.username}"
        th:data-user-name="${user.firstName + ' ' +
user.lastName}"
        th:data-search-text="${user.firstName + ' ' +
user.lastName + ' ' + user.username}">
            <p class="userImage">
                
                
            </p>
            <p class="userName" th:text="${user.firstName + ' ' +
user.lastName}"></p>
            <p class="lastMessage"></p>
        </div>
    </div>
</form>

<div class="chatContainer">
    <div class="chatHeader" id="chatHeader">
    </div>

    <div class="chatBody" id="chatBody">
        <div class="noDialogMessage">Выберите пользователя, чтобы
начать общение</div>
    </div>
    <div class="notification" id="notification"></div>
    <div class="chatInput" id="chatInput">
    </div>
</div>
</div>
</div>
</body>
</html>

```

Файл «diary»

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" th:href="@{/style/style.css}">
    <link rel="stylesheet" th:href="@{/style/diaryStyle.css}">
    <title>Ежедневник</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <script src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.5.0/dist/sockjs.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
    <script src="/js/diary/actionHandler.js"></script>
    <script src="/js/diary/wsConnect.js"></script>
</head>
<body>

<div class="container">
    <div th:replace="~{staticElement :: sidebar}"></div>

    <div class="mainContainer">
        <div id="currentUser" th:data-id="${currentUser.id}" style="display:
none;"></div>
        <div class="cardsContainer" id="cardsContainer">

```



```

        <div class="diaryCard exist" th:each="card: ${cards}"
th:id="${card.id}" th:data-id="${card.id}">

            <div class="cardTitle">
                <input class="title" type="text" th:value="${card.title}">
                <p class="timeCreateCard"
th:text="${#temporals.format(card.date, 'dd.MM.yyyy')}"></p>
            </div>

            <div class="cardBody" th:id="'cardBody'+${card.id}">
                <div class="cardItemsContainer"
th:id="'cardItemsContainer' + ${card.id}">
                    <div class="parentParagraph" th:each="paragraph :
${card.diaryParagraphs}" th:id="'parentParagraph' + ${paragraph.id}">
                        <input class="taskStatus" type="checkbox"
th:checked="${paragraph.isReady}">
                        <input class="task" type="text"
th:value="${paragraph.content}">
                        <button class="deleteParagraph" th:data-
id="${paragraph.id}" th:data-card-id="${card.id}"></button>
                    </div>
                    <div>
                        <button class="createNewParagraph" id="createNewParagraph"
th:data-id="${card.id}"></button>
                        <div class="cardAction">
                            <button class="saveCard inactive" th:id="'saveCard' +
${card.id}"></button>
                            <div class="empty"></div>
                            <button class="deleteCard" id="deleteCard"></button>
                        </div>
                    </div>
                </div>

                <div class="diaryCard create" id="createDiaryCard">
                    <button class="createNewDiaryCard" id="createNewDiaryCard"></button>
                </div>
                </div>
                <button class="navButton left"></button>
                <button class="navButton right"></button>
            </div>
        </div>
    </body>
</html>

Файл «HR»
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" >
<head>

```

```

<meta charset="UTF-8">
<title>HR panel</title>
<link rel="stylesheet" th:href="@{/style/HRStyle.css}" />
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<script src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.5.0/dist/sockjs.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
<script src="/js/HR/actionHandler.js"></script>
<script src="/js/HR/wsConnect.js"></script>
</head>
<body>
<div class="firstView" id="menu" style="display: flex;">
<button class="request addNewUser" id="addNewUserRequest">Запрос на
создание сотрудника</button>
<button class="request deleteUser" id="deleteUserRequest">Запрос на
удаление сотрудника</button>
</div>

<div class="createUser" id="createUser" style="display: none;">
<div class="view createUser">
<div class="header">Запрос на создание сотрудника</div>
<button class="returnBack" id="returnBackCreate" formnovalidate>

</button>
<p class="label ForFirstname">Имя</p>
<input class="userData Firstname" id="inputFirstname" type="text"
required>

<p class="label ForLastname">Фамилия</p>
<input class="userData Lastname" id="inputLastname" type="text"
required>

<p class="label ForPosition">Должность</p>
<input class="userData Position" id="inputPosition" type="text"
required>

<p class="label ForHiringDate">Дата найма</p>
<input class="userData HiringDate" id="inputHiringDate" type="date"
required>

<button class="submitNewUser" id="submitNewUser" type="submit">

</button>
</div>
</div>

<div class="deleteUserRequest" id="deleteUser" style="display: none;">
<div class="view deleteUserRequest">
<div class="header">Запрос на удаление сотрудника</div>

<button class="returnBack" id="returnBackDelete" formnovalidate>

</button>
<p class="label ForFirstname">Имя</p>
<input class="userData Firstname" id="inputFirstnameDelete"
type="text" required>

<p class="label ForLastname">Фамилия</p>
<input class="userData Lastname" id="inputLastnameDelete" type="text"
required>

<p class="label ForPosition">Должность</p>

```

```

        <input class="userData Position" id="inputPositionDelete" type="text"
required>

        <p class="label ForHiringDate">Дата найма</p>
        <input class="userData HiringDate" id="inputHiringDateDelete"
type="date" required>

        <button class="submitNewUser" id="submitDeleteUser" type="submit">
            
        </button>
    </div>
</div>
</body>
</html>

```

Файл «login»

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8"/>
        <title>Вход</title>
        <link rel="stylesheet" th:href="@{/style/loginStyle.css}" />
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
        <script src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.5.0/dist/sockjs.min.js"></script>
        <script
src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
    </head>

    <body>
        <div class="container">

            <div class="timeDependentImage">
            </div>

            <div class="authField">

                <div th:if="${param.error}" class="alert alert-danger">
                    Неверный логин или пароль!
                </div>

                <div class="helloTimeDependentText">
                    <h1 th:text="${today}"></h1>
                </div>

                <form class="authForm" th:action="@{/login}" method="post">
                    <input type="hidden" th:name="${_csrf.parameterName}"
th:value="${_csrf.token}">

                    <h2>Вход в ваш аккаунт</h2>

                    <div class="usernameInput">
                        <label for="username">Имя пользователя</label>
                        <input type="text" id="username" name="username" required>
                    </div>

                    <div class="passwordInput">
                        <label for="password">Пароль</label>
                        <input type="password" id="password" name="password" required>
                    </div>

                    <div class="confirmButton">
                        <button class="button" type="submit"></button>

```

```

        <div><a href="#" id="forgotPasswordLink">Забыли
пароль?</a></div>
    </div>
</form>

</div>
</div>

<div id="forgotPasswordModal" class="modal">
    <div class="modal-content">
        <span class="close">&times;</span>
        <h2>Восстановление пароля</h2>
        <p>Введите ваш Username</p>
        <input id="recoveryUsername" placeholder="Ваш username" required>
        <div class="linkContainer">
            <button id="sendRecovery">Отправить</button>
            <div><a href="#" id="forgotUsernameLink">Забыли
username?</a></div>
        </div>
    </div>
</div>

<div id="forgotUsernameModal" class="modal">
    <div class="modal-content">
        <span class="close" id="closeModalEmail">&times;</span>
        <h2>Восстановление пароля</h2>
        <p>Введите ваш Email</p>
        <input id="recoveryEmail" placeholder="Ваш email" required>
        <div class="linkContainer">
            <button id="sendRecoveryEmail">Отправить</button>
        </div>
    </div>
</div>

<script>
    let stompClient = null;
    let isConnected = false;

    function connect() {
        const socket = new SockJS('/ws');
        stompClient = Stomp.over(socket);

        stompClient.connect({}, () => {
            isConnected = true;

            stompClient.subscribe("/user/queue/userExist", function (message)
{
                const msg = JSON.parse(message.body);
                if (msg) {
                    const modal =
document.getElementById("forgotPasswordModal");
                    modal.style.display = "none";
                    alert("Новый пароль будет отправлен на почту");
                } else {
                    alert("Пользователь не существует");
                }
            });

            stompClient.subscribe("/user/queue/emailExist", function (message)
{
                const msg = JSON.parse(message.body);
                if (msg) {
                    const modal =
document.getElementById("forgotUsernameModal");

```

```

        modal.style.display = "none";
        alert("Новый пароль будет отправлен на почту");
    } else {
        alert("Пользователя с указанной почтой не существует");
    }
    });
});
}

document.addEventListener('DOMContentLoaded', function() {
    connect();

    const modal = document.getElementById("forgotPasswordModal");
    const btn = document.querySelector(".confirmButton a");
    const span = document.getElementsByClassName("close")[0];
    const sendBtn = document.getElementById("sendRecovery");
    const sendEmailBtn = document.getElementById("sendRecoveryEmail");
    const forgotUsername = document.getElementById("forgotUsernameLink");
    const modalEmail = document.getElementById("forgotUsernameModal");
    const closeModalEmail = document.getElementById("closeModalEmail");

    btn.onclick = function(e) {
        e.preventDefault();
        modal.style.display = "block";
    }

    span.onclick = function() {
        modal.style.display = "none";
    }

    closeModalEmail.onclick = function () {
        modalEmail.style.display = "none";
    }

    window.onclick = function(event) {
        if (event.target == modal) {
            modal.style.display = "none";
        } else if (event.target == modalEmail) {
            modalEmail.style.display = "none";
        }
    }

    forgotUsername.addEventListener('click', function () {
        modal.style.display = "none";
        modalEmail.style.display = "block";
    });

    sendEmailBtn.onclick = function() {
        const email = document.getElementById("recoveryEmail").value;
        if (email) {
            stompClient.send("/app/requestEmailExist", {}, email);
        } else {
            alert("Пожалуйста, введите ваш email");
        }
    }

    sendBtn.onclick = function() {
        const username =
document.getElementById("recoveryUsername").value;
        if (username) {
            stompClient.send("/app/requestUserExist", {}, username);
        } else {
            alert("Пожалуйста, введите ваш username");
        }
    }

```

```

    }
  });
</script>
</body>
</html>

Файл «profile»
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8"/>
    <title>Профиль</title>
    <link rel="stylesheet" th:href="@{/style/profileStyle.css}" />
    <link rel="stylesheet" th:href="@{/style/style.css}" />
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <script src="https://cdn.jsdelivr.net/npm/sockjs-
client@1.5.0/dist/sockjs.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/stompjs@2.3.3/lib/stomp.min.js"></script>
  </head>

  <body>
    <div class="container">
      <div th:replace="~{staticElement :: sidebar}"></div>

      <div class="mainContentContainer">
        <div id="currentUser" th:data-username="${currentUser.username}"
style="display: none;"></div>
        <div id="user" th:data-username="${profile.username}"
style="display: none;"></div>

        <div class="profileCard">
          <button th:if="${profile.username ==
currentUser.username}" onclick="goToActivation()" class="editProfile">
            </button>
          <div id="profileImage" class="profileImage">
            
            
          </div>

          <div class="about">
            <h3 th:text="${profile.lastName} + ' ' +
${profile.firstName}"></h3>

            <p th:text="${profile.position}">
          </div>
        </div>

        <div class="mainInfo">
          <h3>Основная информация</h3>
          <div class="staticData">
            <p>Имя</p>
            <p>Фамилия</p>
            <p>Дата рождения</p>
            <p>Пол</p>
          </div>

          <div class="dynamicData">
            <p th:text="${profile.firstName}">
            <p th:text="${profile.lastName}">

```

```

        <p th:if="${profile.dob != null}"
            th:text="${#temporals.format(profile.dob,
'dd.MM.yyyy')} ">

        <p th:unless="${profile.dob != null}">-----</p>

        <p th:text="${profile.gender != null} ?
${profile.gender} : '-----'"></p>
    </div>
</div>

<div class="workplaceInfo">
    <h3>Рабочее место</h3>
    <div class="staticData">
        <p>Город</p>
        <p>Улица</p>
        <p>Этаж</p>
        <p>Кабинет</p>
    </div>

    <div class="dynamicData">
        <p th:text="${profile.city}"></p>

        <p th:text="${!profile.street.isEmpty()} ?
${profile.street} : '-----'"></p>

        <p th:text="${!profile.officeFloor.isEmpty()} ?
${profile.officeFloor} : '-----'"></p>

        <p th:text="${!profile.officeNumber.isEmpty()} ?
${profile.officeNumber} : '-----'"></p>
    </div>

</div>

<div class="feedback">
    <h3>Связь со мной</h3>

    <div class="staticData">
        <p>Телефон</p>
        <p>Почта</p>
    </div>

    <div class="dynamicData">
        <p th:text="${!profile.phoneNumber.isEmpty()} ?
${profile.phoneNumber} : '-----'"></p>

        <p th:text="${profile.email}"></p>
    </div>

    <a class="linkVK" th:if="${!profile.VKid.isEmpty()}"
th:href="${profile.VKid}" target="_blank"></a>

    <a class="linkTG"
th:if="${!profile.telegramUsername.isEmpty()}"
th:href="${profile.telegramUsername}" target="_blank"></a>

    <a class="linkVK inactive"
th:unless="${!profile.VKid.isEmpty()}" target="_blank"></a>

    <a class="linkTG inactive"
th:unless="${!profile.telegramUsername.isEmpty()}" target="_blank"></a>

```

```

        </div>
    </div>
</div>

<script>
    const socket = new SockJS('/ws');
    const stompClient = Stomp.over(socket);
    const element = document.getElementById('currentUser');
    const username = element.getAttribute('data-username');
    const elem = document.getElementById('user');
    const user = elem.getAttribute('data-username');
    console.log(username);
    console.log(user);

    // Подключаемся только если это профиль текущего пользователя
    if (username === user) {
        stompClient.connect({}, () => {
            stompClient.send("/app/register", {}, username);

            stompClient.subscribe('/topic/onlineStatus', (message) => {
                const data = JSON.parse(message.body);
                if (data.username === username) {
                    const element = document.getElementById('profileImage');
                    element.className = data.isOnline ? 'profileImage online'
: 'profileImage offline';
                }
            });
        });

        window.addEventListener('beforeunload', () => {
            stompClient.send("/app/unregister", {}, username);
        });
    } else {
        // Для чужого профиля просто проверяем онлайн статус
        stompClient.connect({}, () => {
            stompClient.subscribe('/user/queue/status', (message) => {
                const isOnline = JSON.parse(message.body);
                const element = document.getElementById('profileImage');
                element.className = isOnline ? 'profileImage online' :
'profileImage offline';
            });
            stompClient.send("/app/requestStatus", {},
                JSON.stringify( {
                    requestingUser: username,
                    requestedUser: user
                }));
        });
    }

    function goToActivation() {
        window.location.href = /*[[@{/activation}]]*/ '/activation';
    }
</script>
</body>
</html>

```

Файл «staticElement»

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head>
    <meta charset="UTF-8"/>
    <title>Профиль</title>
    <link rel="stylesheet" th:href="@{/style/style.css}" />

```



```

<link rel="stylesheet" th:href="@{/style/chatStyle.css}" />
<link rel="icon" type="image/png" href="/img/logoW.png">
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
</head>

<body>
  <div th:fragment="navbar" class="navbar">

  </div>

  <div th:fragment="sidebar" class="sidebar" id="navigation">
    <div class="link goToProfile">
      <a th:href="@{/profile}" class="button">
        Профиль
      </a>
    </div>

    <div class="link goToAllUsers">
      <a th:href="@{/allUsers}" class="button">
        Пользователи
      </a>
    </div>

    <div class="link goToChat">
      <a th:href="@{/chat}" class="button">
        Чат
      </a>
    </div>

    <div class="link goToDiary">
      <a th:href="@{/diary}" class="button" >
        Ежедневник
      </a>
    </div>

    <div class="link goToAdminPage" sec:authorize="hasRole('ROLE_ADMIN')">
      <a th:href="@{/admin}" class="button" >
        Администратор
      </a>
    </div>

    <div class="emptyPlace"></div>

    <form class="logout" th:action="@{/logout}" method="post">
      <button class="logout" type="submit">
        
      </button>
    </form>
  </div>
</body>
</html>

```