МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Web-приложение биржа строительных работ «Халтурка»»

Выполнил студент Зыков Кирилл Андреевич

(Ф.И.О.)

Руководитель работы асс. Нистюк Ольга Александровна

(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2024

**Содержание**

[Введение 4](#_Toc160569364)

[1 Постановка задачи 5](#_Toc160569365)

[1.2 Аналитический обзор аналогов 5](#_Toc160569366)

[1.2 Разработка функциональных требование и вариантов использования 7](#_Toc160569367)

[1.3 Вывод 8](#_Toc160569368)

[2 Проектирование web-приложения 9](#_Toc160569369)

[2.1 Обобщенная структура управлением приложения 9](#_Toc160569370)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 9](#_Toc160569371)

[2.4 Вывод 10](#_Toc160569372)

[3 Разработка web-приложения 11](#_Toc160569373)

[3.1 Разработка клиентской части web-приложения 11](#_Toc160569374)

# Введение

Цель данной работы заключается в создании веб-приложения мессенджер, которое позволит пользователям обмениваться сообщениями, голосовыми сообщениями и сообщениями в виде стикеров.

Для разработки серверной части данного веб-приложения был выбран фреймворк NestJS, который позволит разработать быстрое и многофункционально веб-приложение; а для разработки клиентской части была выбрана библиотека ReactJS с написанием кода на TypeScript. Для работы с БД будет использоваться модуль Prisma, который позволит создать абстракцию над объектами базы данных PostgreSQL.

Для гарантированной безопасности пользователей приложения в курсовой работе применяется метод шифрования паролей перед их сохранением в базу данных. Также для обеспечения функциональности приложения используются мультимедийные форматы данных при сохранении аудио и картинок.

Основные требования к приложению:

* реализация ролей администратора и пользователя;
* отправка, редактирование и удаление сообщений;
* добавление в друзья;
* удаление из друзей;
* возможность подписаться на другого пользоваться;
* отписать от пользователя;

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, тестирование веб-приложения, а также инструкции по использованию приложения.

# 1 Постановка задачи

## 1.2 Аналитический обзор аналогов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

Первый аналог – WhatsApp.

WhatsApp обладает широким спектром функций, включая отправку текстовых сообщений, голосовых сообщений, видеозвонки, групповые чаты, обмен файлами и стикерами, а также возможность создания и управления каналами. Этот мессенджер также позволяет пользователям отправлять местоположение и документы.

WhatsApp имеет простой и удобный интерфейс, который легко использовать. Дизайн этого мессенджера состоит из основных функций в левой части экрана и списка чатов в правой части. WhatsApp также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид мессенджера.

Дизайн WhatsApp – довольно-таки минималистичный и простой в использовании. Цветовая схема состоит в основном из белого и зеленого цветов, что создает ощущение легкости и чистоты для пользователя.

Блок сообщения сделан закругленным. Текст для имени пользователя подсвечивается другим цветом. Текст черного цвета на белом фоне. На данное сообщение довольно-таки приятно смотреть.

Интерфейс приложения представлен на рисунке 1.1.

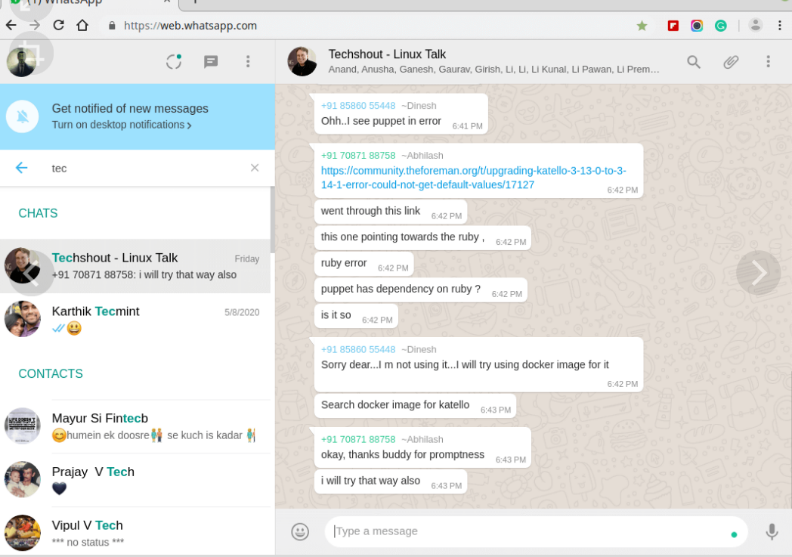


Рисунок 1.1 – Приложение «WhatsApp»

Второй аналог – это приложение Viber.

Viber позволяет отправлять текстовые сообщения, стикеры, фотографии, видео и аудиофайлы. Вы также можете создавать групповые чаты для общения с несколькими людьми одновременно. Viber позволяет совершать бесплатные голосовые звонки и видеозвонки с другими пользователями Viber по всему миру. Для этого необходимо, чтобы оба пользователя были подключены к интернету. имеет светлый интерфейс с фиолетовыми элементами дизайна.

Главный экран отображает список ваших чатов, а также кнопки для совершения голосовых и видеозвонков. В верхней части экрана расположены кнопки для доступа к контактам, магазину стикеров и настройкам. В целом, интерфейс Viber интуитивно понятен и удобен в использовании.

Дизайн Viber представляет собой современный и лаконичный интерфейс с использованием ярких цветов и плоских иконок. Он имеет минималистичный дизайн, который делает приложение легким и простым в использовании.

Страница пользователя имеет картинку человека, на которого мы зашли. Кнопки в нем подсвечены и выделяются на общем фоне. Сразу видно, куда жать.

Интерфейс приложения представлен на рисунке 1.2.

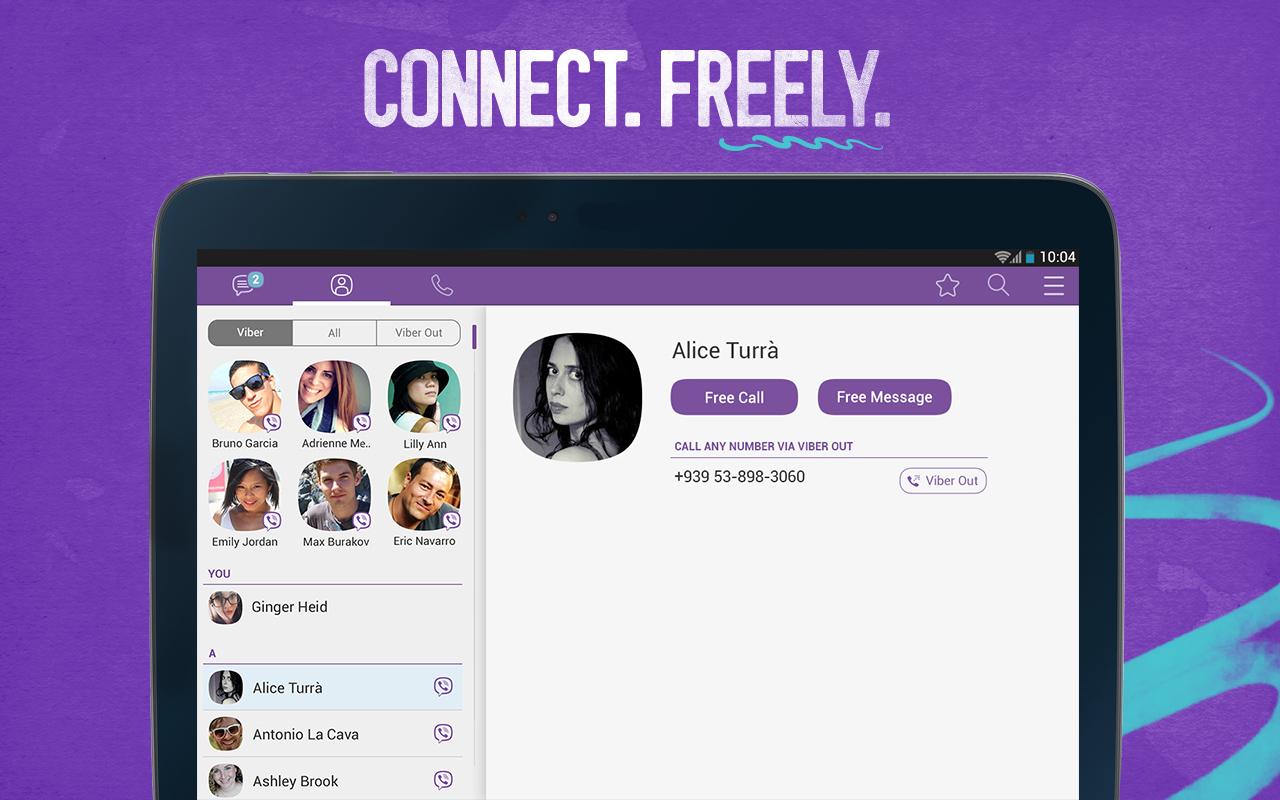


Рисунок 1.2 – Приложение «Viber»

Третий аналог – это Telegram.

Telegram обладает широким спектром функций, включая отправку текстовых сообщений, голосовых сообщений, видеозвонки, групповые чаты, демонстрация экрана, обмен файлами и стикерами, а также возможность создания каналов и ботов. Этот мессенджер также предлагает возможность создания секретных чатов с функцией автоматического удаления сообщений.

Telegram имеет простой и удобный интерфейс, который легко использовать. Дизайн этого мессенджера состоит из основных функций в нижней части экрана и списка чатов в верхней части. Telegram также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид мессенджера.

Дизайн Telegram отличается от других мессенджеров своей минималистичностью и простотой в использовании. Он имеет темно-синий цветовой фон, который выделяет контент на экране. В верхней части экрана расположено главное меню, которое позволяет быстро переключаться между разделами мессенджера. В центре экрана находится список чатов и диалогов. Каждый чат представлен в виде миниатюрного значка с изображением фото профиля пользователя или группы, а также отображается последнее сообщение. При нажатии на любой из чатов, открывается окно переписки, которое также имеет темно-синий фон и белый шрифт. Дизайн приятный и легко пользоваться им.

Интерфейс приложения представлен на рисунке 1.3.

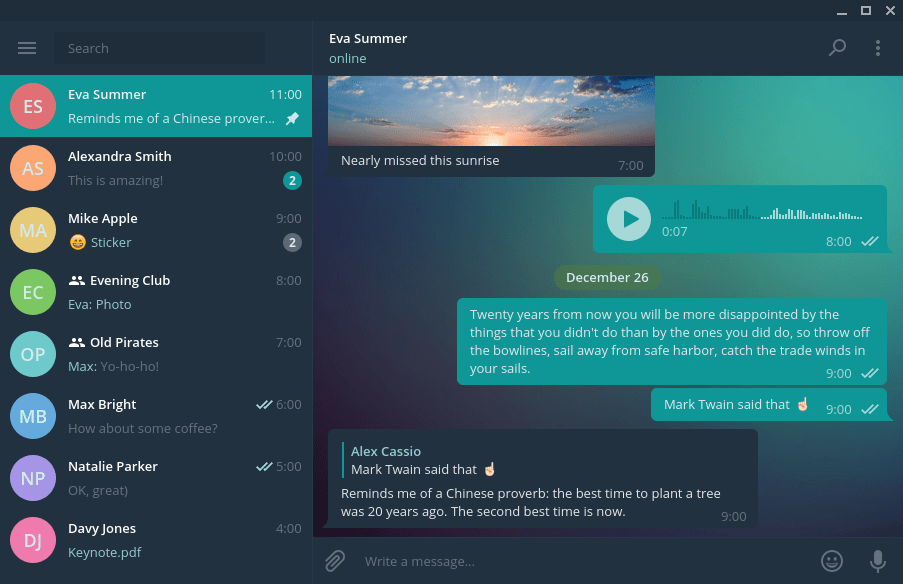


Рисунок 1.3 – Приложение «Telegram»

Анализируя аналоги WhatsApp, Viber и Telegram, можно сделать вывод, что все три приложения предлагают широкий спектр функций, включая обмен сообщениями, голосовые и видеозвонки, групповые чаты и возможность отправки файлов. Каждое из приложений имеет свою уникальную цветовую схему и дизайн интерфейса, но общий тренд – простота и удобство использования. Все они предлагают настраиваемый интерфейс с возможностью выбора тем оформления.

## 1.2 Разработка функциональных требование и вариантов использования

Приложение мессенджер должно обеспечивать функционал для регистрации авторизации, который позволит идентифицировать пользователей и для каждого пользователя предлагать чаты, которые относятся к нему. Также мессенджер должен обеспечивать функционал отправки различных типов сообщений: текстовые, голосовые и стикеры. Для работы с сообщениями должны быть созданы функции удаления и редактирования сообщений. Удаление должно быть сделано полным, чтобы сообщение полностью удалялось из базы данных. Также должен быть реализован механизм друзей, чтобы пользователь мог быстро искать человека для отправки ему сообщения. Также должен быть создан схожий с друзьями механизм – подписчики. Это создано для того, чтобы у пользователя был выбор, добавлять ли определенного пользователя к себе в друзья. Диаграмма вариантов использования представлена на рисунке 1.4.

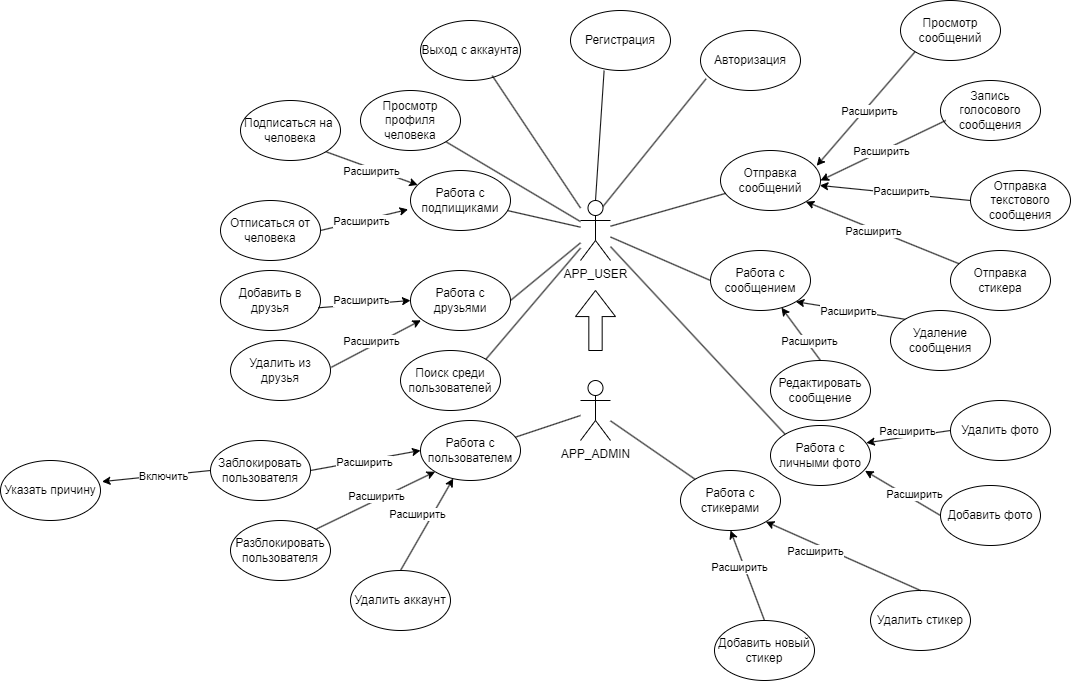


Рисунок 1.4 – Диаграмма вариантов использования

Диаграмма UML вариантов использования позволяет понять, что доступно каждой роли, доступной в данном веб-приложении.

## 1.3 Вывод

В данном разделе мы рассмотрели аналогичные веб-приложения. Исходя из этих аналогов можно взять пару идей для своего веб-приложения. Например, данный мессенджер должен обеспечивать удобные интерфейс, как и Viber, данный мессенджер должен предоставлять безопасность при отправке сообщений, как Telegram.

Также в данном разделе рассмотрели весь основной функционал, который должен присутствовать в конечном приложении. Данный функционал рассчитан на то, что он позволит довольно-таки просто начать пользоваться этим веб-приложением, а также что этот функционал позволит общаться разным людям при помощи разных способах. Была приведена UML диаграмма, на которой все и приведено.

# 2 Проектирование web-приложения

## 2.1 Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения для общения должны входить функции для удобного поиска людей в приложении, функции добавления в друзья, различные виды сообщение и другие подобные функции.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.1.

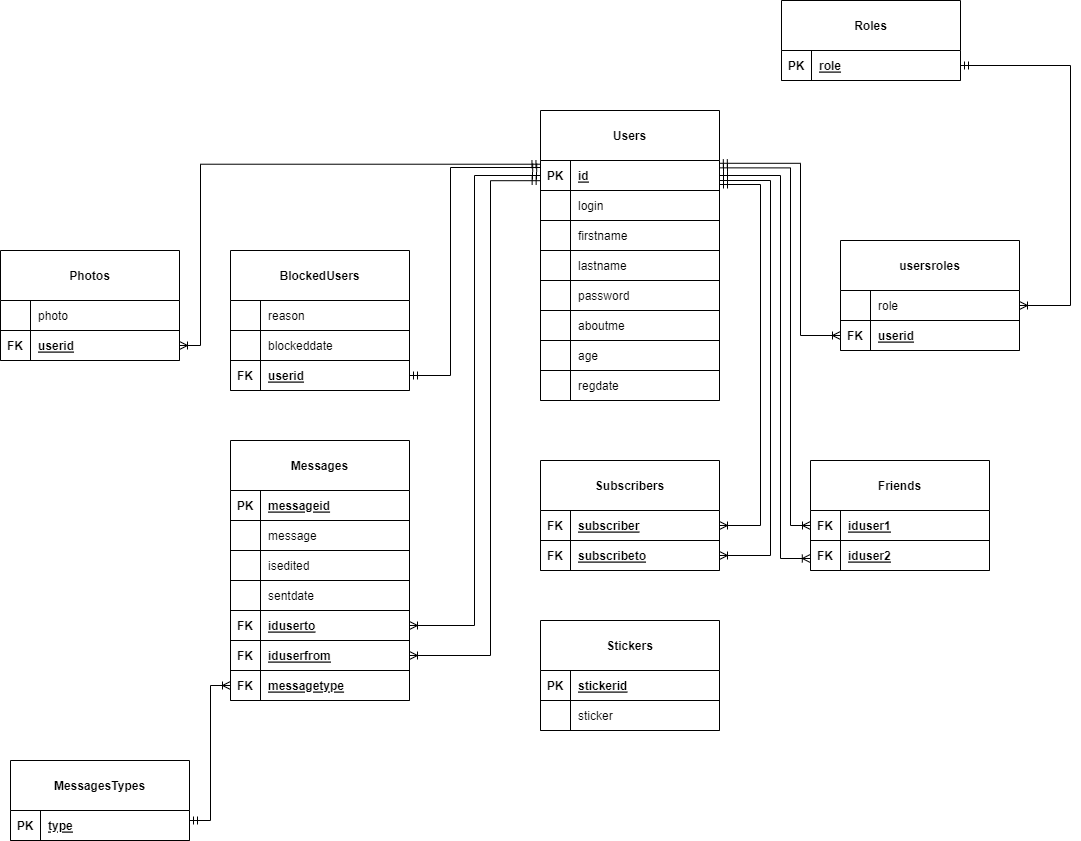


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-одному", "один-ко-многим", "многие-ко-многим". Например, таблица Users связана с таблицами Photos, BlockedUsers, UsersRoles, Messages, Subsribers, Friends через внешние ключи.

Для того понимания структуры данного приложения и как его разворачивать нам может помочь диаграмма развертывания. Данная диаграмма показана на рисунке 2.2.

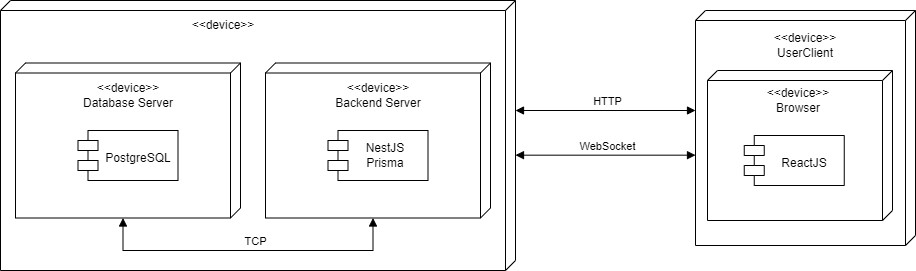


Рисунок 2.2 – Диаграмма развертывания

По данной диаграмме развертывания видно, что наше веб-приложение состоит из трех основных частей. Первая часть – это клиентская часть нашего приложения. Эта часть реализована при помощи одноименной библиотеки для языка программирования JavaScript ReactJS. Данная библиотека поможет создать динамически SPA приложение, которое будет делать запросы по протоколу HTTP к серверной части приложения. Также оно будет отправлять и получать сообщения по протоколу WebSocket. Вторая часть данного веб-приложения – это серверная часть, разработанная на платформе NodeJS при помощи фреймворка NestJS, который позволяет построить структуру из всех основных модулей нашего приложения. Данная часть приложения будет обращаться к базе данных по протоколу TCP. Обращение к базе данных будет упрощено за счет использования ORM модуля – Prisma, который позволит нам создать абстракцию над объектами базы данных. Третья часть нашего приложения – это сам сервер базы данных PostgreSQL, который будет отвечать за хранение всех данных, способствующих работе нашего приложения.

## 2.4 Вывод

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой.

В данном разделе рассмотрели структуру базы данных PostgreSQL, ее основные сущности и как они связаны. Также рассмотрели, какие есть основные части нашего веб-приложения, точнее, во что оно развертывается.

# 3 Разработка web-приложения

## 3.1 Разработка клиентской части web-приложения

Клиентская часть – главная часть приложения. Она должна быть интуитивно понятна и интерактивна. Клиентская часть будет состоять из трех основных частей: библиотека React, библиотека RTK Query и WebSocket клиент.

Часть React состоит из четырех страниц. Когда пользователь заходит первый раз в веб-приложение ему доступна лишь одна страница – страница авторизации. В библиотеки React для маршрутизации используется компонент Router. Пример использования данной компоненты показан в листинге 3.1.

<div className="min-vh-100 max-height-100vh w-100 bg-color height-100vh">  
 {  
 !isLoading

&&  
 <Routes>  
 {  
 currentUser ?  
 <>  
 <Route index path="/\*" element={<MainPage/>}/>  
 </>  
 :  
 <>  
 <Route index path="/auth/\*" element={<AuthPage/>}/>  
 <Route index path="/\*" element={<AuthPage/>}/>  
 </>  
 }  
 </Routes>  
 }  
</div>

Листинг 3.1 – Изначальные настройки маршрутизации

По данному коду видно, что если пользователь авторизован, то его будет пересылать на MainPage, где будет происходить другая маршрутизация, о которой будет сказано чутка позже. Если же пользователь не авторизован, то ему будет доступна лишь страница AuthPage, в которой также есть своя маршрутизация.

Компонент AuthPage содержит в себе еще три маршрутизации. Первая маршрутизация – это по URL равном «/reg». Она будет маршрутизировать на компонент RegForm, который будет содержать разметку для формы регистрации. Вторая маршрутизация – это по URL равно «/login». Она будет маршрутизировать на компонент LoginForm, который будет содержать разметку для формы авторизации. Также есть третья маршрутизация. Данная маршрутизация будет отвечать за все остальные URL, которые не были описаны ранее. Ее задача будет, если неавторизованный пользователь введет не поддерживаемый URL, возвращать компонент LoginForm, как компонент на случай ненайденной страницы. Компонента AuthPage представлена в листинге 3.2.

export const *AuthPage* = () => { return (  
 <div className="min-vh-100 w-100 d-flex justify-content-center ">  
 <Routes>  
 <Route path="/reg" element={<RegForm />}/>  
 <Route path="/login" element={<LoginForm/>}/>  
 <Route path="/\*" element={<LoginForm/>}/>  
 </Routes>  
 </div>  
 )  
}

Листинг 3.2 – Компонент AuthPage

Компонент MainPage отвечает за маршрутизацию для авторизованного пользователя. Пользователю доступно три страницы. Первая страница доступная страница – это компонент MessagePage. Данный компонент отвечает за страницу с сообщениями и чатами. Следующая страница – это компонент PeoplePage. Данная страница показывает, кто вообще пользуется приложением, своих друзей и подписчиков. Третья страница, доступная авторизованную пользователю, – это компонент BlockedUsersPage. Но данная страница доступна авторизованную пользователю, который является администратором. Так же там указана причина блокировки. Там будут показаны все пользователи, которые заблокированы в приложении. Компонент MainPage показан в листинге 3.3.

export const *MainPage* = () => {  
 const currentUser = *useCurrentStoreUser*();  
 const [\_isAdmin] = *useState*(*isAdmin*(currentUser));

useGetMessage();  
 useOnlineChecker();  
 useUserDeleteHandler();  
 useUserBlockHandler();

return (  
 <div className="main-page h-100 height-100vh max-height-100vh">  
 <Header/>  
 <Routes>  
 <Route path='/messages' element={<MessagePage />}/>  
 <Route path='/people' element={<PeoplePage />}/>  
 {  
 \_isAdmin &&  
 <Route path='/blocked-users' element={<BlockedUsersPage />}/>  
 }  
 <Route path='\*' element={<MessagePage />}/>  
 </Routes>  
 </div>  
 )  
}

Листинг 3.3 – Компонент MainPage

Также в данном компоненте видно вызов хуков useGetMessage(), useOnlineChecker(), useUserDeleteHandler(), useUserBlockHandler(). Данные хуки относятся к WebSocket-клиенту данного веб-приложения. Хук useGetMessage() отвечает за получение сообщения по WebSocket-клиенту. Хук useOnlineChecker() отвечает за отслеживанием, какие пользователи в сети. Хук useUserDeleteHandler() подписан на очень редкое событие, которое срабатывает, когда администратор удалит аккаунт пользователя. Хук useUserBlockHandler() отвечает за отслеживанием подписки на событие, которое срабатывает, когда аккаунт пользователя был заблокирован.

За хранение состояния на клиентской части отвечает библиотека Redux ToolKit Query. Она позволяет хранить состояние и делать запросы к серверу с помощью хуков в React. Хранилище Redux показано в листинге 3.4.

export const store = *configureStore*({  
 reducer: {  
 auth: authReducer,  
 users: usersReducer,  
 chats: chatsReducer,  
 messages: messageReducer,  
 ws: wsReducer,  
 friends: friendsReducer,  
 subscribers: subscribersReducer,  
 [authApi.reducerPath]: authApi.reducer,  
 [usersApi.reducerPath]: usersApi.reducer,  
 [photosApi.reducerPath]: photosApi.reducer,  
 [chatsApi.reducerPath]: chatsApi.reducer,  
 [messagesApi.reducerPath]: messagesApi.reducer,  
 [stickersApi.reducerPath]: stickersApi.reducer,  
 [audiosApi.reducerPath]: audiosApi.reducer,  
 [subscribersApi.reducerPath]: subscribersApi.reducer,  
 [friendsApi.reducerPath]: friendsApi.reducer,  
 [adminApi.reducerPath]: adminApi.reducer  
 },  
 middleware: (*getDefaultMiddleware*) => *getDefaultMiddleware*()  
 .concat(authApi.middleware, usersApi.middleware, photosApi.middleware, chatsApi.middleware, messagesApi.middleware, stickersApi.middleware, audiosApi.middleware, subscribersApi.middleware, friendsApi.middleware, adminApi.middleware)

});

Листинг 3.4 – Построение хранилища Redux

Хранилище будет хранить состояния об авторизации. К этому состоянию будет относиться JWT-токен авторизации. Также хранилище будет хранить информацию о пользователях. К этой информации относится объект с пользователем, который на данный момент пользуется приложением, массив, который хранит всех пользователей, полученных из сервера, что используется в компоненте PeoplePage, массив с идентификаторами пользователей, которые находятся в сети, что используется в хуке useOnlineChecker(). Следующее состояние, хранимое в Redux, – это информация о чатах. К данной информации относятся: идентификатор текущего чата, идентификатор чата, на который надо перейти при нажатии на кнопку «Перейти в чат» на странице «Люди», массив со всеми чатами, полученных с сервера. Состояние о сообщениях – это следующее состояние, хранимое в Redux. Оно отвечает за хранение всех сообщений с определенным пользователем. Состояние о WebSocket хранит информацию о самом сокете, по которому будет идти обмен сообщениями. Также Redux хранит состояние о друзьях, где хранится массив со всеми людьми, которые являются другом текущему пользователю. Следующее состояние схоже с предыдущим – это состояние о подписчиках, которое хранит массив со всеми подписчиками текущего пользователя.

RTK Query позволяет делать запросы к серверу за счет хуков и хранить информацию о том, что вернул нам сервер. В листинге 3.5 представлен пример создания запроса на получение стикеров из сервера.

export const stickersApi = createApi({  
 reducerPath: "stickers-api",  
 baseQuery: *fetchBaseQuery*({  
 baseUrl: SERVER\_URL + "/api/stickers",  
 credentials: "include"  
 }),  
 endpoints: (*builder*) =>

({  
 getStickers: *builder*.query({  
 query: (*page*: number) => ({  
 url: "",  
 method: "GET",  
 params:

{  
 *page* }  
 }),  
 keepUnusedDataFor: 0  
 })  
 }),  
});

export const {

useGetStickersQuery,

useLazyGetStickersQuery } = stickersApi;

Листинг 3.5 – Пример постороения запросов в RTK Query

В листинге 3.5 видно, что мы формируем URL, куда будет отправлен запрос, подключаем отправку и куки, выбираем метод отправки и параметры, которые будут приложены к запросу. После построения Redux API мы экспортируем хуки, которые и будут обеспечивать получение стикеров в компонентах React.

Одна из основных частей данного веб-приложения – это WebSocket клиент. Его инициализация происходит при первом запуске данного сайта в компоненте App. За это отвечает хук useCreateSocket(). Данный хук работает так, что он отслеживает за изменением текущего пользователя, если он меняется, то сокет будет пересоздаваться. Данный хук показан в листинге 3.6.

export const *useCreateSocket* = () => {  
 const dispatch = *useAppDispatch*();  
 const user = *useCurrentStoreUser*();  
  
 *useEffect*(() => {  
 dispatch(createSocket());  
 }, [user]);  
}

Листинг 3.6 – Хук для создания WebSocket-клиента

Для того, чтобы WebSocket-сервер знал, с кем работает мы после инициализации сокета и установки соединения будем отправлять сообщение на сервер, чтобы он сохранил идентификатор пользователя и токен авторизации. Сохранением этих данных занимается хук useSetSocketName. Он представлен в листинге 3.7.

export const *useSetSocketName* = (*name*?: string): void => {  
 const socket = *useSocket*();  
 const token = *useToken*();  
  
 *useEffect*(() => {  
 if (!*name*) return;  
 socket?.emit('set-name', {*name*, token});  
 }, [*name*, socket]);  
}

Листинг 3.7 – Хук useSetSocketName

Данный хук отслеживает за изменением сокета с помощью хука useEffect. При изменении состояния клиент будет отправлять сообщение на сервер, а в параметры сообщения занесет объект с идентификатором пользователя и токеном авторизации.