МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Скриптовые языки программирования»

Тема «Веб-приложение Онлайн чат»

**Исполнитель**

студент 1 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.А. Арцыхович

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.С. Кантарович

должность, учен. степень, ученое звание подпись, дата

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.С. Кантарович

подпись дата инициалы и фамилия

# **Введение**

Наличие Интернета в нашей жизни, во многом, облегчает нам жизнь. В той или этой сфере мы часто обращаемся к этому феномену, чтобы получить достоверные знания о том, что нам интересно. В XXI веке овладеть любой информацией не составляет никакого труда. Получать информацию можно, как читая какие-либо форумы, смотреть различные обучающие видеоролики, и наиболее эффективный метод, получая информацию из прямого общения с человеком. К примеру - онлайн чат.

По статистике, примерно 79% потребителей выбирают для коммуникации онлайн-чат, поскольку он предоставляет практически мгновенные ответы.

Онлайн-чат — это инструмент коммуникации, который позволяет потенциальным пользователям взаимодействовать между собой. В наше время популярность онлайн чатов приобретается в геометрической прогрессии. Это связано с тем, что на любом форуме, интернет-магазине и т.д. часто требуется помощь связана с конкретным вопросом, живое общение – идеально подходит для решения данной проблемы.   
 В данном проекте будет реализован чат, предназначен преимущественно для общения. Будет предоставлен функционал выборки фотографии вашего профиля. Также будет реализована функция добавления файлов. Что, кстати, очень полезная функция при обучении. Но, в любом случае, функции данного чата будут направлены, прежде всего, именно на обычное общение, но, не исключено, что данный чат может быть встроены как в сайты, по получению какой-либо полезной информации, в интернет-магазины, для удобства клиента. И, наиболее эффективно, было бы встроить данный чат именно в Мессенджер, как отдельная секция для повседневного общения пользователей данного Мессенджера.

Преимущества данного проекта будет «Realtime» соединение, которое позволит пользователям мгновенно получать и отправлять информацию в чате. Данная функция наиболее характерно определяет сложность и производительность проекта.

Цель курсовой работы: разработать веб-приложение Онлайн чата, которое работало бы в реальном времени, с использованием REACT, HTML5, Sass , REDUX , а также с применением Облачного хранилища FIREBASE (Облачное хранилище данных о пользователях).

Задачи:

1. Проанализировать существующие языки разметки, инструменты и библиотеки для создания веб-приложения.
2. Разработать макет приложения.
3. Разработать структуру веб-приложения.
4. Разработать функциональную часть веб-приложения.
5. Подключить облачное хранилище данных.
6. Протестировать веб-приложение.
7. Разработать руководство пользователя.

**1. Постановка задачи**

**1.1 Обзор аналогичных решений**

Курсовой проект представляет из себя веб-приложение онлайн чата. В данной сфере огромное количество аналоговых продуктов, это связано с популярностью данной темы. В этом разделе будут приведены веб-сайты существующих онлайн-чатов.

**1.1.1 Веб приложение Телеграмм**

Приложение «Телеграмм» известен своей скоростью, плавностью и кросс-платформенным дизайном. Сайт имеет приятную для глаз цветовую палитру. Сайт динамический, интерактивный. Дизайн представлен на рисунках ниже. Данный сайт написан на таких языках как JS, PHP, а также подключение React JS. Сразу бросается в глаза минимализм данного сайта, который отражается в большом объеме области под сообщения, либо же новостную ленту определенного канала.

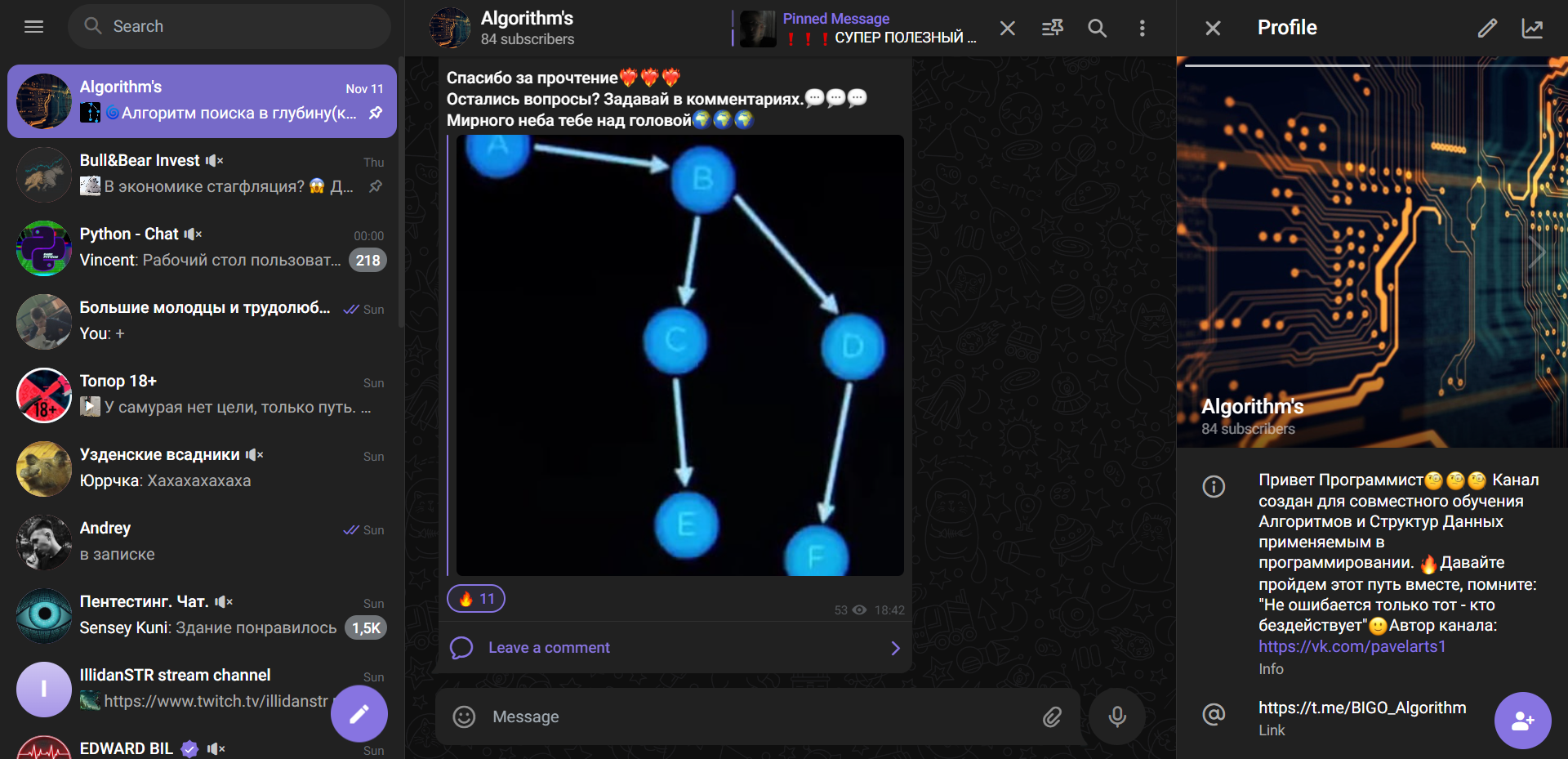
****

Рисунок 1.1 — «Главная страница веб приложения Телеграмм»

На рисунке 1.1 представлена главная Телеграмма. На ней мы можем увидеть так называемые «Три составляющие хороших чатов» Часть с поиском: Пользователей, групп, каналов. Основная часть, где, собственно, и располагается основная информация. Заключительно составляющая, которая отвечает за то, чтобы отвечать за информацию о пользователе, группе, канале.

Мобильная версия сайта также проработано с высоким качеством. На данном сайте хорошо проработана адаптивность. На первый взгляд бросается в глаза тёмный фон цветов. На рисунке 1.2 сразу можно заметить наиболее полезную информацию, например: Список доступных переписок и каналов. Возможность создания собственного канала или беседы, а также выпадающее меню с возможность просмотра информации о пользователе. Настроек своего профиля, а также просмотр конфиденциальной информации.

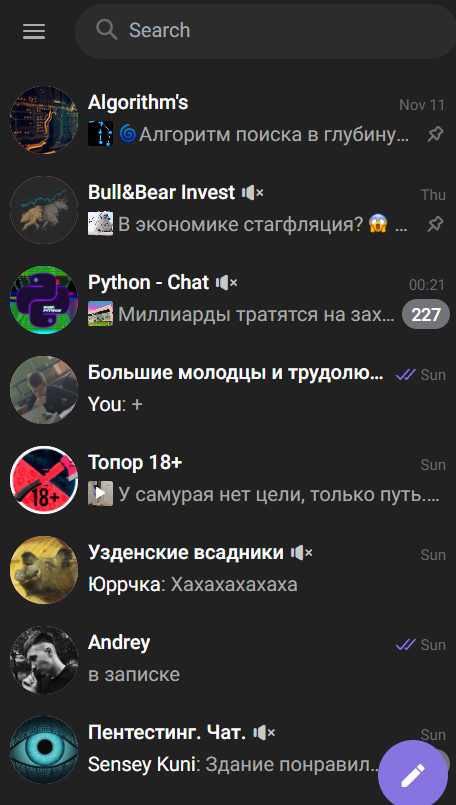


Рисунок 1.2 «Основная часть мобильной версии»

Стоит отметить, что в мобильной версии данного сайта будет доступно уже две основные функции в рамках области видимости. Очень важно построить структуру мобильной версии именно так, чтобы приложение умело выполнять все функции основные функции из сайта, который написан под более широкие устройства, а также уместить все эти функции на видном месте, что, обычно, довольно трудная задача, с которым прекрасно справились разработчики Телеграмма, в данном продукте все основные функции располагаются на самых видных местах, что, при данных условиях не нарушает структуры и выглядит очень эффектно. На рисунке 1.3 представлена то самое выдвигающее меню, с помощью его мы можем просмотреть основную информацию, к примеру: Сохраненные сообщения, чаты находящиеся в архиве, контакты, которые будут состоять из ваших контактов из телефона, а также друзей в самом приложении, тем самым будет присутствовать связь между данными из самого телефона, а также данные находящеюся в рамках самого приложения, настройки, где можно просмотреть ваши данные, темный мод, а также другие вспомогательные функции.

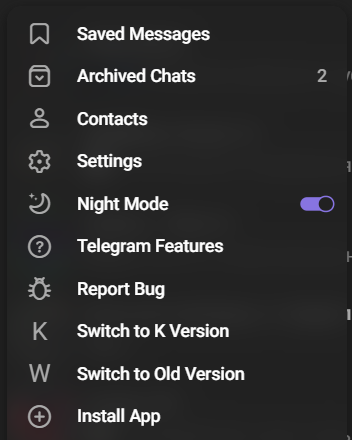


Рисунок 1.3 «Выпадающее меню мобильной версии»

* + 1. **Веб приложение ВКонтакте**

Приложение «ВКонтакте» — российская социальная сеть со штаб-квартирой в Санкт-Петербурге. Сайт доступен на 86 языках; особенно популярен среди русскоязычных пользователей. Приложение прославилось благодаря своей скорости, а также очень быстро воспользовалось моделью «Single Page Application». Данный сайт написан на таких языках как JS, PHP, библиотек React JS, Django. Дизайн данного сайта представлен в светло-синих тонах. В данном приложении можно увидеть такие популярные функции как: Музыкальный плеер, с помощью которого пользователь может слушать интересующую его музыку, новостная лента. Рассмотрим функционал сайта.

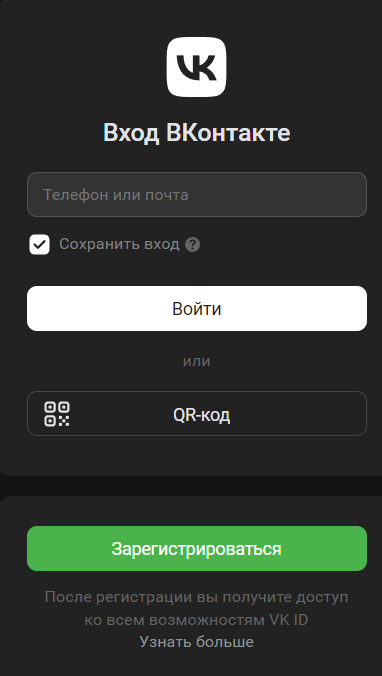


Рисунок 1.4 «Форма входа/регистрации»

На рисунке 1.4 показана форма входа, внизу видна кнопка регистрации. Преимущество данной форме в том, что вход можно реализовать с помощью QR-кода.

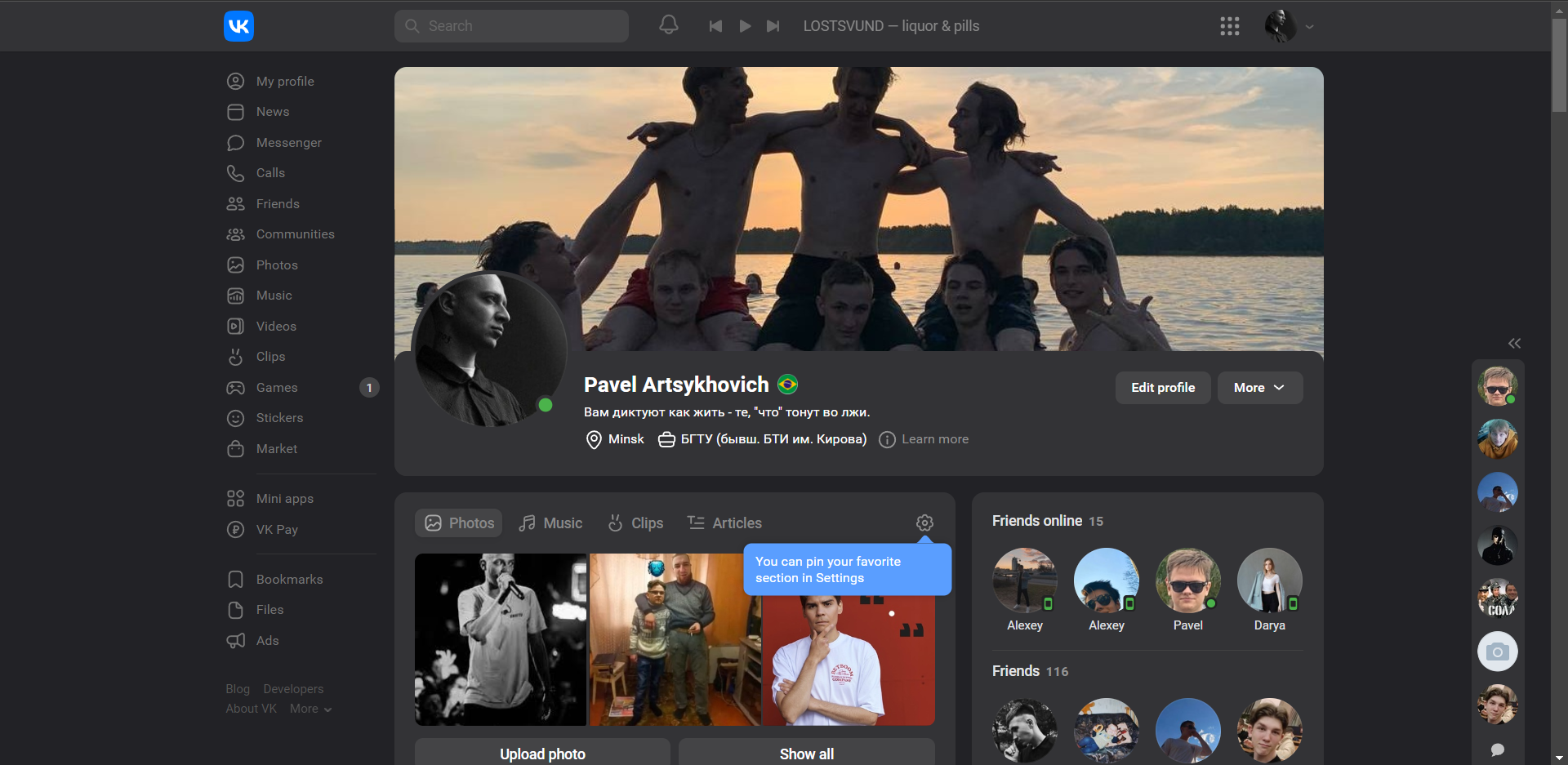


Рисунок 1.5 «Основная страница ВКонтакте»

На рисунке 1.5 представлена основная страница ВКонтакте. Переходы на другие страницы реализованы с помощью «React Router».

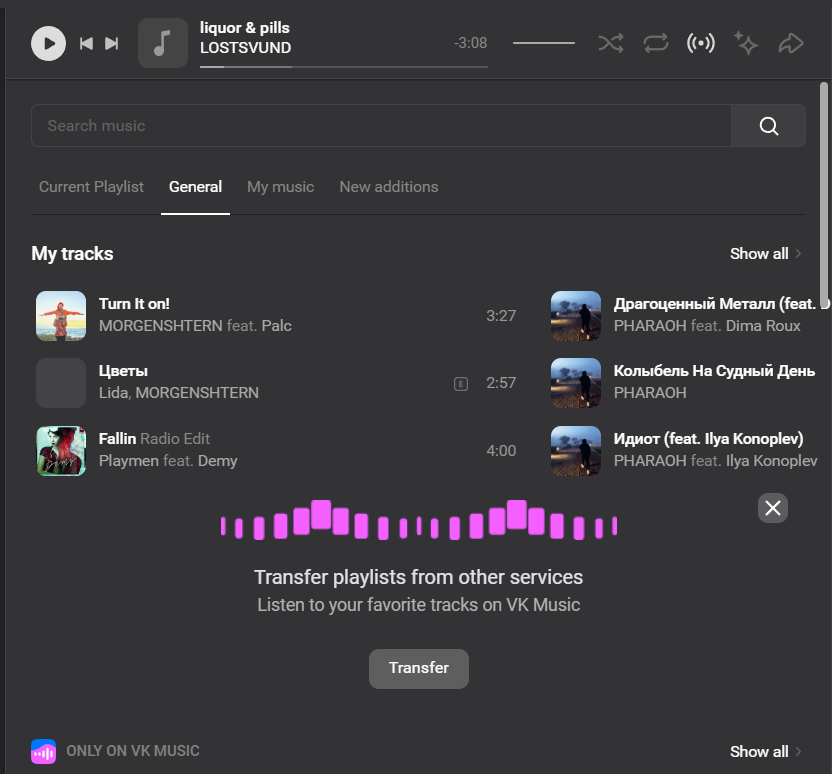


Рисунок 1.6 «Музыкальный плеер ВКонтакте»

На рисунке 1.6 представлен музыкальный плеер «ВКонтакте», в котором можно добавлять и слушать музыку.

Мобильная версия сайта не менее красочно выглядит, адаптивность проработана с высоким качеством. Наиболее структурированная информация, одно из главных качество сайта. Данный сайт явное представления структурированной информации. Пользователю с первого взгляда должно стать понятно о чем сайт. Обычно для заинтересованности пользователей, вводятся именно такие тона, которые большинству пользователей с первой секунды настраивают на приобретенные положительные эмоции во время серфинга данного сайта.

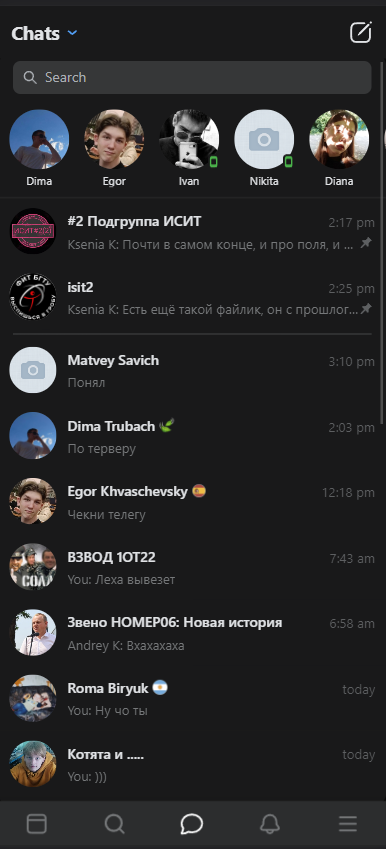


Рисунок «1.7 Мобильная версия чата ВКонтакте»

На рисунке 1.7 представлен чат в мобильной версии «ВКонтакте, справа внизу можно заметить выдвигающееся меню, с выбором остального функционала мобильной версии.

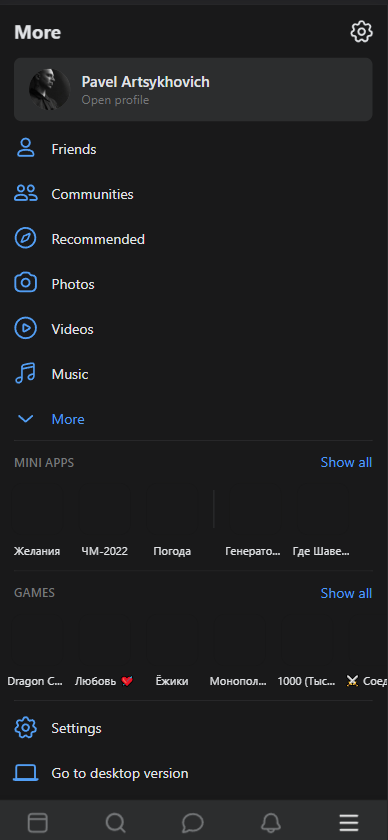


Рисунок 1.8 «Выдвигающееся меню ВКонтакте»

На рисунке 1.8 представлен вид выдвигающегося меню, а также возможность перехода на группы и музыку в приложении «ВКонтакте».

Таким образом мобильная версия отражает практически все функции полноценной версии данного приложения, это показатель того, что мобильная версия проработано очень качественно.

# **1.2** **Техническое задание**

В данном проекте задача создать приложение «Онлайн чат». В приложении будет выполняться «Realtime model» передачи сообщений, это значит то, что сообщения будут доставляться в реальном времени, что позволит принимать и отправлять сообщения без какой-либо задержки. Проект подразумевает собой синхронизацию с облачным хранилищем данных, с помощью которого будет выложен на хостинг с наименьшими потерями пакетов. Также в проекте будет реализован аудиоплеер и другие функции.

Главной задачей данного приложения является предоставление удобного, конфиденциального чата. В перспективе сделать данный проект наиболее динамически быстрым, для того, чтобы пользователь имел возможность комфортно вести диалоги с другими пользователями, а также пользоваться другими благами приложения.

Основные требования к исполнителю: создание дизайна, проведение анализа существующих решений, создание прототипа, адаптивность и кроссбраузерность сайта (корректное отображение в браузерах, таких как Google Chrome, Mozilla Firefox, Microsoft Edge, Opera.). В проекте используется гипертекстовый язык разметки HTML, CSS/Sass, библиотека ReactJS, Redux Toolkit. Графические элементы в формате SVG; Аудио элементы в формате mp3. Данные будут храниться в обачном хранилище firebase.

Visual Studio Code станет отличной IDE для комфортного и ускоренного написания кода для веб-страницы. А именно выбор необходимых плагинов и цветовой темы.

В качестве архитектуры будет выполнена стандартная архитектура в виде: Frontend, Backend, Database.

В начале работы с сайтом пользователь является гостем (Guest). Ему будет доступна только страница регистрации и вхожа. После регистрации пользователь становится пользователем (User).

Пользователь (User) может просматривать основную часть сайта, где и будет располагаться сам чат, прослушивать музыку, пользоваться основным функционалом приложение.

В роли администратора пользователь может изменять данные пользователей, блокировать их, удалять аккаунт.

Также он может просматривать историю сообщений, информацию о них. Но администратор будет управлять всеми данными только через базу данных.

Для реализации стилевого оформления сайта было выбрано минималистическое направление с элементами классического веб-дизайна. Такой стиль позволяет привлечь внимание новых пользователей. Сайт не загромождается лишними деталями (боковыми панелями и т.п.), большим количеством спецэффектов. Акцент на сайте ориентируется на основной контент.

Сайт будет состоять из 2-х и основных цветов – серого и сине-зеленого градиента. Также на сайте будет реализована анимация падающего снега. Эти цвета были подобраны так, потому что они будут приятны для глаз и это цвета, на которое пользователь сразу обратит внимания на сайт.

В данном проекте будут использован один основной шрифт: Monserrat. Шрифт подключался с помощью библиотеки Google Fonts.

Шрифт Monserrat является одним из самых популярных шрифтов в 2022 году. Он был разработан в 2013 году, но в последние годы его популярность растет с каждым днем. Этот шрифт отличается своей читабельностью и простотой. Он подходит для любых типов дизайна, в том числе и для веб-дизайна.

**2. Проектирование**

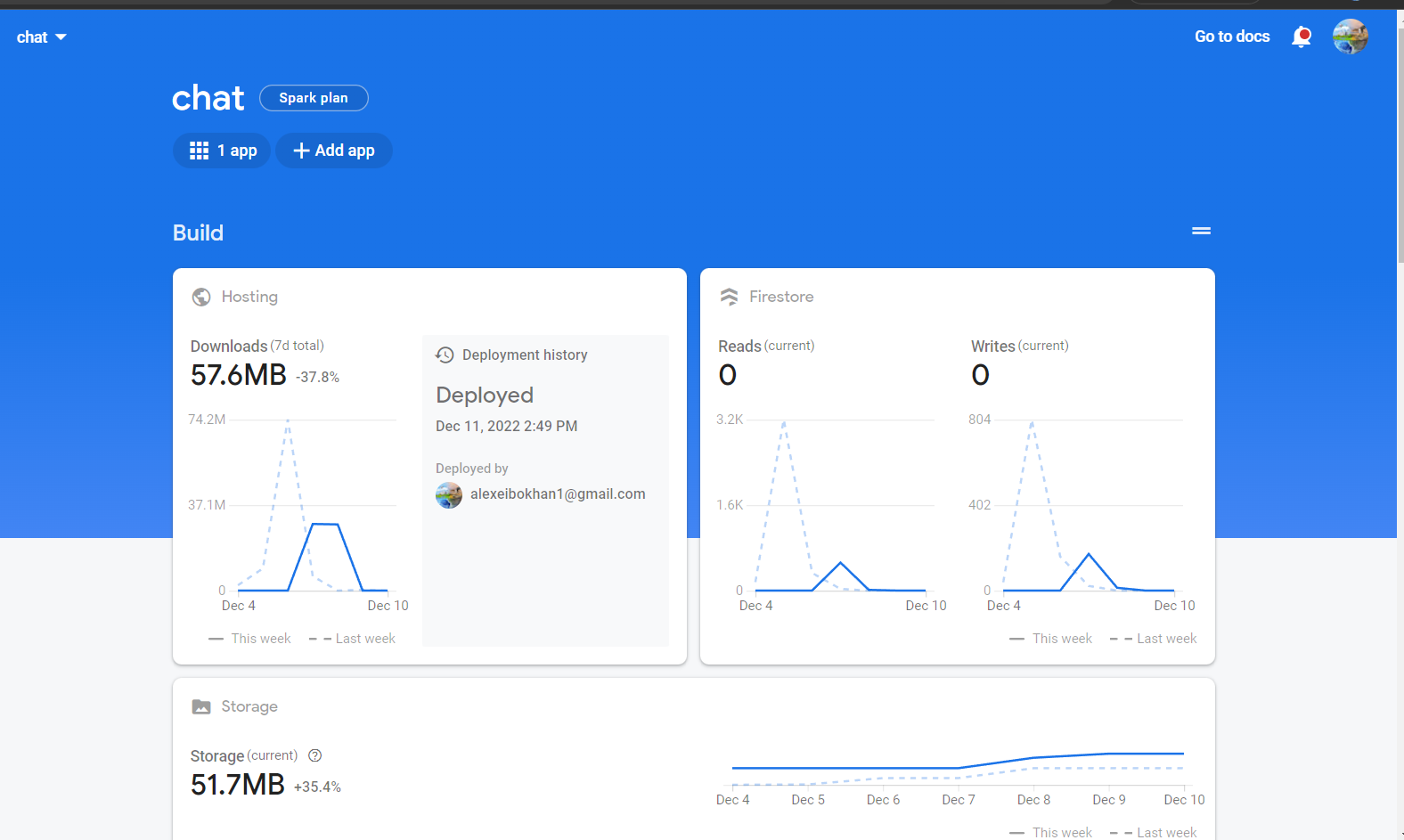
В основу основных алгоритмов данного продукта лягут алгоритмы с наибольшей эффективностью. К примеру алгоритм поиска пользователей реализуется с помощью поисков: бинарного поиска, и поиска алгоритмом поиска сортировкой. Данные алгоритмы были выбраны для получения наивысшей скоростью, а если учесть, что пользователи будут найдены в облачном хранилище, то стоит учесть, что скорость алгоритма очень важна, так как запросы к самому хранилищу тоже требуют времени. Бинарный поиск — тип поискового алгоритма, который последовательно делит пополам заранее отсортированный массив данных, чтобы обнаружить нужный элемент. Этот алгоритм состоит из трёх шагов. Сначала из массива нужно выбрать один элемент — его обычно называют опорным. Затем другие элементы в массиве перераспределяют так, чтобы элементы меньше опорного оказались до него, а большие или равные — после. Данные в облачном хранилище «Firebase» будут располагаться в виде таблиц в облаке, к которым пользователь будет писать запрос. Преимущества данного хранилища в том, что пользователь один раз подключившись к нему может на сайте корректировать данные, в удобном приложении разработанным Google. Также, приложение предоставляет возможность занятие бесплатного хостинга, на который курсовой проект и будет загружен. 

Рисунок 2.1 «Главная страница хранилища»

Как видно из рисунка 2.1 представлен график использования памяти, на котором расположена зависимость кол-во памяти затраченной в промежуток времени.

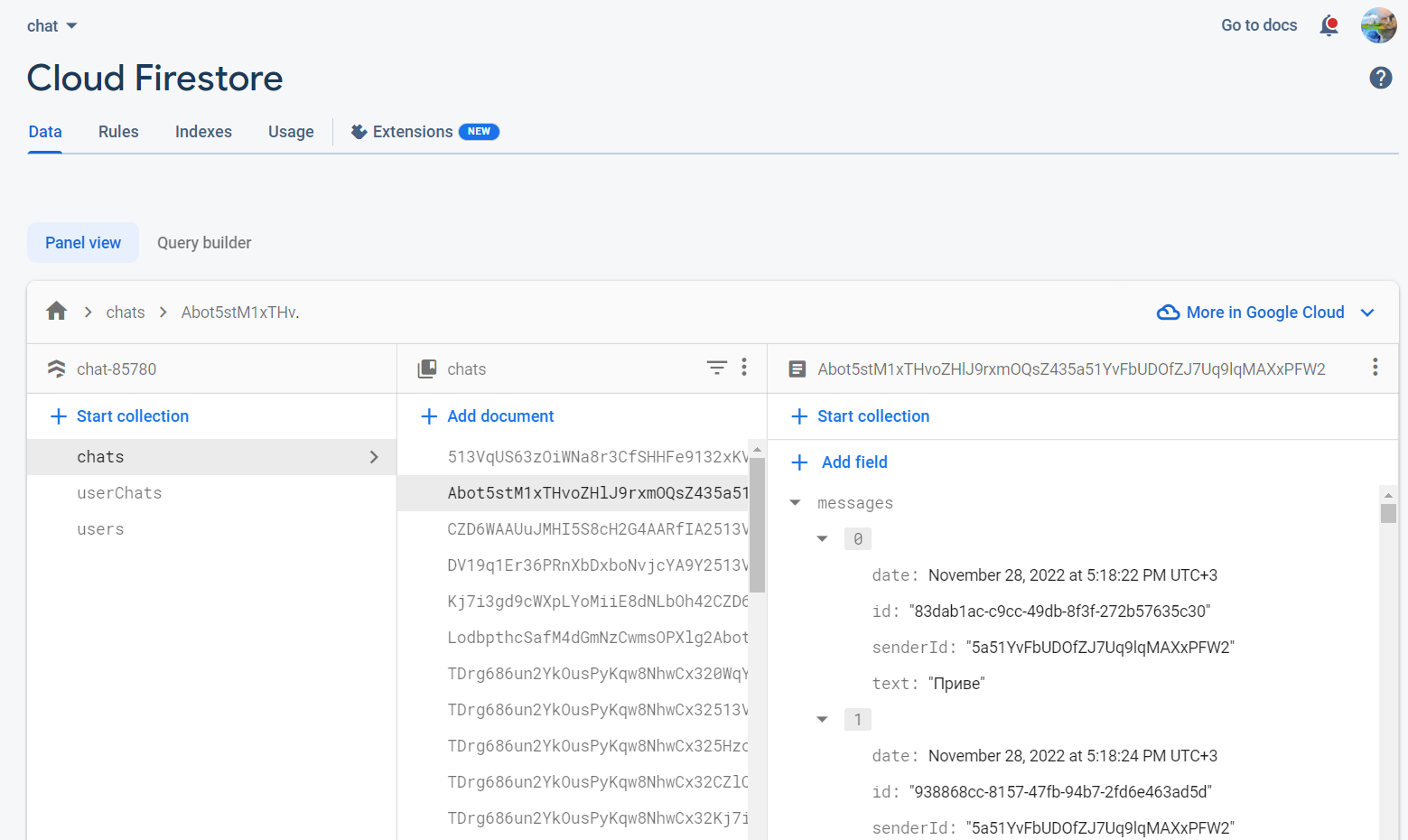


Рисунок 2.2 «Таблицы и их состовляющие»

На рисунке 2.2 нам представлена структура хранения таблиц в коллекциях на сервисе «Firebase google». Документы(столбцы) заполняющие эти таблицы показывают все, нужные, данные о пользователях.

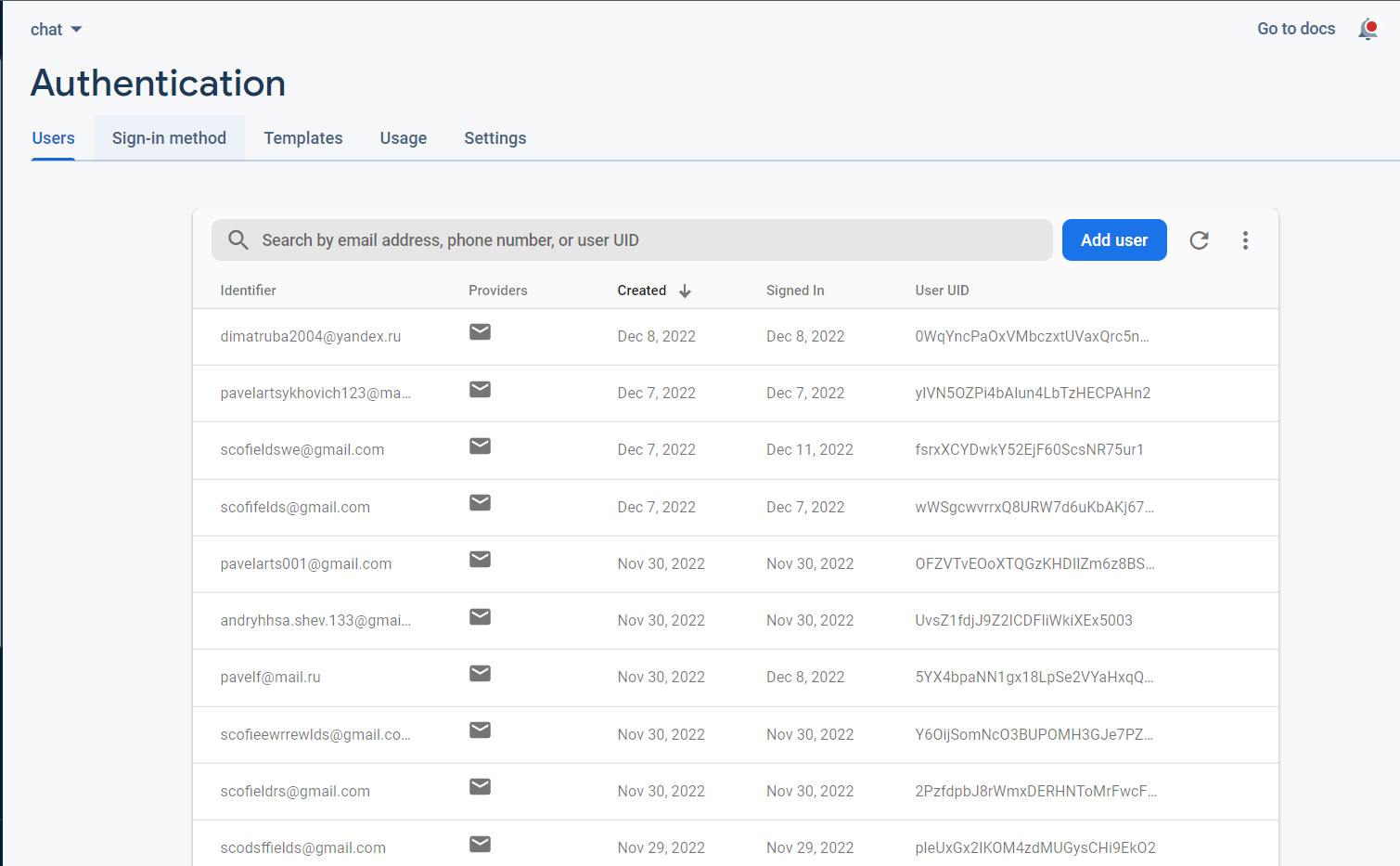


Рисунок 2.3 «Таблица Пользователей»

На рисунке 2.3 представлена таблица с пользователями, присутствующими в приложении чата.

|  |
| --- |
| import { initializeApp } from "firebase/app";  import { getAuth } from "firebase/auth";  import { getFirestore } from "firebase/firestore";  import { getStorage } from "firebase/storage";  const firebaseConfig = {    apiKey: "AIzaSyCsatflhlSuFcmPRfc9keb9q6EMHjUIHlE",    authDomain: "chat-85780.firebaseapp.com",    projectId: "chat-85780",    storageBucket: "chat-85780.appspot.com",    messagingSenderId: "159067188821",    appId: "1:159067188821:web:002488538825ea34c72edc"  };  // Initialize Firebase  export const app = initializeApp(firebaseConfig);  export const auth = getAuth();  export const storage = getStorage();  export const db = getFirestore() |

Листинг 2.1 «Подключение к хранилищу»

**3. Структура приложения**

**3.1 Реализация структуры клиентской части**

Структура данного проекта будет состоять из компонентного подхода. Компонентный подход в программировании — это относительно недавний подход, если сравнивать его с другими. Пришел он с массовым появлением библиотек. Суть его сводится к тому, что разработка одной большой программы производится с применением «заготовок» кода. Заготовками будем называть отельные JSX файлы, которые возвращают что-либо своё. Компонент — это функция, которая возращает JSX фор-мат. JavaScript XML (JSX) — это расширение синтаксиса JavaScript, которое позволяет использовать HTML-подобный синтаксис для описания структуры интерфейсов. Как правило, компоненты написаны с использованием JSX, но также есть возможность использования обычного JavaScript.

|  |
| --- |
| const Chat = ({GetFind,states}) => {  const [isOpened, setIsOpened] = useState(false);    const { data } = useContext(ChatContext);  const[state,setState] = useState(false);  const Find = ()=>{    setIsOpened(true)    GetFind(true)  }  useEffect(()=>{    setIsOpened(states)  },[states])    return (      <div className="chat">        <div className="chatInfo">        <button className="find" onClick={Find}>          Find        </button>          <span className="UserName">{data.user?.displayName}</span>          {state && <Game\_1/>}          <div className="chatIcons">         <img src={Cam} alt=""         onClick={()=>setState(!state)} />            {/\* <a href="https://www.vk.com">            <img src={Add} alt="" /></a>            <a href="https://www.facebook.com"><img src={More} alt="" /></a>             \*/}          </div>        </div>        <Messages />        <Input/>          </div>    );  };  export default Chat; |

Листинг 1.1 – Структура JSX-формата

В листинге 1.1 представлен JSX формат, который отвечает за отображение сообщений в чате. Также на данной странице будет приниматься «Avatar» пользователя, которую участник приложения сможет благополучно загрузить со своего устройства. После заполнение формы, участник получит контрольное сообщение, подтверждающие или отклоняющее его вхождение в чат. Сразу после заполнение данных и входа пользователя на главную страницы чата данные отобразяться в базе данных на сервере.

|  |
| --- |
| import React, { useState } from "react";  import { useNavigate, Link } from "react-router-dom";  import { signInWithEmailAndPassword } from "firebase/auth";  import { auth } from "../firebase";  const Login = () => {    const [err, setErr] = useState(false);    const navigate = useNavigate();    const handleSubmit = async (e) => {      e.preventDefault();      const email = e.target[0].value;      const password = e.target[1].value;      try {        await signInWithEmailAndPassword(auth, email, password);        navigate("/")      } catch (err) {        setErr(true);      }    };    return (      <div className="formContainer">        <div className="formWrapper">          <span className="logo">ART\_CHAT</span>          <span className="title">Login</span>          <form onSubmit={handleSubmit}>            <input type="email" placeholder="Email" />            <input type="password" placeholder="Password" />            <button>Sign in</button>            {err && <span>Something went wrong</span>}          </form>          <p>Do u don't have account? <Link to="/Register">Register</Link></p>        </div>      </div>    );  };  export default Login; |

Листинг 1.2 – Структура «Login page»

В листинге 1.2 представлена страница входа, на которой отображена форма входа. Пользователю предложено будет ввести свой зарегистрированный адрес электронный почты, а также пароль, который пользователь указывал при регистрации, если данные верны из режима Гость, участник будет переведён в роль User. Данные, опять же, будут сверяться с данными из облачной базы данных.

|  |
| --- |
| const Chat = ({GetFind,states}) => {    const [isOpened, setIsOpened] = useState(false);    const { data } = useContext(ChatContext);  const[state,setState] = useState(false);  const Find = ()=>{    setIsOpened(true)    GetFind(true)  }  useEffect(()=>{    setIsOpened(states)  },[states])    return (      <div className="chat">        <div className="chatInfo">        <button className="find" onClick={Find}>          Find        </button>          <span className="UserName">{data.user?.displayName}</span>          {state && <Game\_1/>}          <div className="chatIcons">           <img src={Cam} alt=""         onClick={()=>setState(!state)} />            {/\* <a href="https://www.vk.com">            <img src={Add} alt="" /></a>            <a href="https://www.facebook.com"><img src={More} alt="" /></a>             \*/}          </div>        </div>        <Messages />        <Input/>            </div>    );  };  export default Chat; |

Листинг 1.3 – Структура отображения отдельного чата

В листинге 1.3 отображена структурного отдельного одного чата, в котором будет отображаться отдельный user который будет вести диалог, с тем, кто зарегистрирован на странице. Также в данном компоненте будут использоваться и компоненты, применяемые непосредственно о отправке самого сообщения и его ввода.

|  |
| --- |
| const Message = ({ message }) => {    const dispatch = useDispatch();  const messaget=useSelector((state)=>state.message.value);    const { currentUser } = useContext(AuthContext);    const { data } = useContext(ChatContext);    const ref = useRef();    useEffect(() => {      ref.current?.scrollIntoView({ behavior: "smooth" });      console.log(message.date)    }, [message]);    return (      <div        ref={ref}        className={`message ${message.senderId === currentUser.uid && "owner"}`}      >        <div className="messageInfo">          <img            src={              message.senderId === currentUser.uid                ? currentUser.photoURL                : data.user.photoURL            }            alt=""          />    <span>    {message.date.seconds}   </span>        </div>        <div className="messageContent">          <p>{message.text}</p>          {message.img && <img src={message.img} alt="" />}        </div>      </div>    );  };  export default Message; |

Листинг 1.4 – Структура «Message»

В данном листинге показывается отправка сообщения, соединение синхронно с базой данных, использование ссылок, а также отправка особых сообщений, с картинками. Также в данном листинге реализован scroll страницы в самый конец, при входе в конкретного пользователя, для удобства пользователя.

|  |
| --- |
| const Home = () => {    const [state,setState]= useState(false);    const GetFind =(states)=>{      setState(states)      }    const CloseFind = (states)=>{      setState(states)    }    return (      <div className='home'>        <div className="container">        <Sidebar state={state} CloseFind={CloseFind}/>        <Chat GetFind={GetFind} states={state} />            </div>      </div>    )  }  export default Home |

Листинг 1.5 – Структура основной страницы

В листинге 1.5 реализован основной компонент, который собирает в себе все вспомогательные компоненты. Также здесь хранится основное состояние поиска в хранилище данных

|  |
| --- |
| .formContainer {    background: -webkit-linear-gradient(90deg, #ffffff,#5757af);    background: linear-gradient(90deg, #ffffff,#4949a6);    height: 100vh;    display: flex;    align-items: center;    justify-content: center;    .formWrapper {      background-color: white;      overflow: hidden;      margin: 10px;      padding: 20px 60px;      border-radius: 10px;      display: flex;      flex-direction: column;      gap: 10px;      align-items: center;      .logo {        color: #080473;        font-weight: bold;        font-size: 24px;      }      .title {        color: #5d5b8d;        font-size: 12px;      }        form {        display: flex;        flex-direction: column;        gap: 15px;        input {          padding: 15px;          border: none;          width: 100%;      width: 260px;          border-bottom: 1px solid #a7bcff;          &::placeholder {            color: rgb(175, 175, 175);          }        }        button {          background-color: #2bb7ba;          color: white;          padding: 10px;          font-weight: bold;          border: none;          cursor: pointer;          border-radius: 5%;        } |

Листинг 1.5 – Структура SASS

В листинге 1.5 представлена реализация основного контейнера стилей, и применения SASS.

**3.2 Реализация структуры серверной части**

В данном подразделе будут представлены листинги связанные непосредственно с серверной частью приложения, подключение к облачному хранилищу, а также обработка внутри её.

|  |
| --- |
| const firebaseConfig = {    apiKey: "AIzaSyCsatflhlSuFcmPRfc9keb9q6EMHjUIHlE",    authDomain: "chat-85780.firebaseapp.com",    projectId: "chat-85780",    storageBucket: "chat-85780.appspot.com",    messagingSenderId: "159067188821",    appId: "1:159067188821:web:002488538825ea34c72edc"  };  // Initialize Firebase  export const app = initializeApp(firebaseConfig);  export const auth = getAuth();  export const storage = getStorage();  export const db = getFirestore() |

Листинг 1.6 – Подключение к хранилищу

В листинге 1.6 представлено подключение к облачному хранилище, с помощью которого будет происходить управление проектом.

|  |  |
| --- | --- |
| function App() {      const { currentUser } = useContext(AuthContext);      const ProtectedRoute = ({ children }) => {        if (!currentUser) {          return <Navigate to="/login" />;        }        return children      };      return (            <div>    <BrowserRouter>            <Routes>              <Route path="/">                <Route                  index                  element={                    <ProtectedRoute>                      <Home/>                    </ProtectedRoute>                  }                />                <Route path="login" element={<Login />} />                <Route path="register" element={<Register />} />                <Route path ="Game\_1" element = {<Game\_1/>}/>                <Route path ="/Profile" element = {<Profile/>}/>              </Route>            </Routes>          </BrowserRouter>        );    }    export default App; |  |

Листинг 1.7 – Маршрутизация

В листинге 1.7 представлены маршруты, которые представляют собой SPA(Single Page Application) модель представления страниц на сайте. С помощью которой, мы сможем быстро не подгружая каждый html документ. А подгружать динамически один, и изменяя JavaScript подгружать данные.

|  |
| --- |
| const Search = () => {    const [username, setUsername] = useState("");    const [user, setUser] = useState(null);    const [err, setErr] = useState(false);    const { currentUser } = useContext(AuthContext);    const handleSearch = async () => {      const q = query(        collection(db, "users"),        where("displayName", "==", username)      );    const handleSelect = async () => {        const combinedId =        currentUser.uid > user.uid          ? currentUser.uid + user.uid          : user.uid + currentUser.uid;      try {        const res = await getDoc(doc(db, "chats", combinedId));        if (!res.exists()) {          //create a chat in chats collection          await setDoc(doc(db, "chats", combinedId), { messages: [] });          //create user chats          await updateDoc(doc(db, "userChats", currentUser.uid), {            [combinedId + ".userInfo"]: {              uid: user.uid,              displayName: user.displayName,              photoURL: user.photoURL,            },            [combinedId + ".date"]: serverTimestamp(),          });          await updateDoc(doc(db, "userChats", user.uid), {            [combinedId + ".userInfo"]: {              uid: currentUser.uid,              displayName: currentUser.displayName,              photoURL: currentUser.photoURL,            },            [combinedId + ".date"]: serverTimestamp(),          });        }      } catch (err) {}      setUser(null);      setUsername("")    };    return (      <div className="search">        <div className="searchForm">          <input            type="text"            placeholder="Find a user"            onKeyDown={handleKey}            onChange={(e) => setUsername(e.target.value)}            value={username}          />          </div>        )}      </div>    );  };  export default Search; |

Листинг 1.8 – Поиск Пользователей

В листинге 1.8 представлен компонент, отвечающий за поисков пользователей на сервисе, делая параллельные запросы на сервер облака.

|  |
| --- |
| import {createSlice} from '@reduxjs/toolkit';  const initialState = {      value: ""  }  export const messageSlice=createSlice ({      name: 'message',      initialState,      reducers: {          setMessage: (state, action) => {              state.value = action.payload          }      }  })  export const  {setMessage} = messageSlice.actions  export default messageSlice.reducer |

Листинг 1.9 – Подключение «Redux-toolkit»

В листинге 1.9 представлено подключение Redux, которое будет согласовываться с отправляемыми сообщениями, а также хранить состояния в сообщениях.

|  |
| --- |
| export const ChatContext = createContext();  export const ChatContextProvider = ({ children }) => {    const { currentUser } = useContext(AuthContext);    const INITIAL\_STATE = {      chatId: "null",      user: {},    };    const chatReducer = (state, action) => {      switch (action.type) {        case "CHANGE\_USER":          return {            user: action.payload,            chatId:              currentUser.uid > action.payload.uid                ? currentUser.uid + action.payload.uid                : action.payload.uid + currentUser.uid,          };        default:          return state;      }    };    const [state, dispatch] = useReducer(chatReducer, INITIAL\_STATE);    return (      <ChatContext.Provider value={{ data:state, dispatch }}>        {children}      </ChatContext.Provider>    );  };  // AuthContext  export const AuthContext = createContext();  export const AuthContextProvider = ({ children }) => {    const [currentUser, setCurrentUser] = useState({});    useEffect(() => {      const unsub = onAuthStateChanged(auth, (user) => {        setCurrentUser(user);        console.log(user);      });      return () => {        unsub();      };    }, []);    return (      <AuthContext.Provider value={{ currentUser }}>        {children}      </AuthContext.Provider>    );  }; |

Листинг 2 – Представление контекстной части

В листинге 2 описывается реализация контекстной части. С помощью её будет обернут файл index.js.

**4. Тестирование веб-сайта**

**4.1 Адаптивный дизайн веб-сайта**

На данный момент количество пользователей, которые заходят в Интернет с мобильных устройств очень велико, и с каждым днем их становится все больше. А потому адаптация веб-сайта под маленькие экраны – актуальная задача и для курсового проекта.

При создании сайта, была поставлена задача: создать сайт, который будет сам подстраиваться под разные типы экранов. Главными условиями адаптивности является нефиксированный размер элементов страницы.

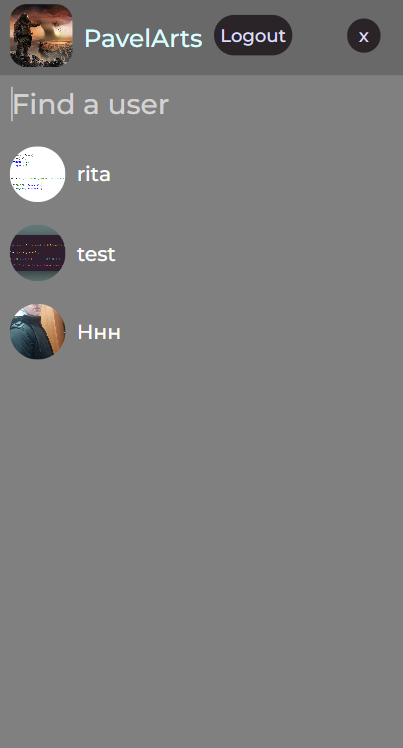


Рисунок 2.4– Основная часть мобильной версии приложения

**4.2 Кроссбраузерность веб-сайта**

Кроссбраузерность – это способность веб-ресурса отображаться одинаково и работать во всех популярных браузерах, без перебоев в функционировании и ошибок в верстке, а также с одинаково корректной читабельностью контента.

В ходе тестирования были проверены все ссылки(включая модальные окна), проверены формы регистрации пользователя. Была выполнена проверка контента. Также в ходе тестирования была проверена совместимость с браузерами.(Google Chrome, Opera, Firefox, Microsoft Edge). В браузере Internet Explorer тестирование давала некоторые сбои. А именно проблема с запуском с открытого сервера(live server), а также анимации сделанные непосредственно через CSS3 ввели себя некорректно.

У каждого браузера есть свои встроенные, экспериментальные или нестандартные свойства и для того, чтобы они корректно работали и ввелись префиксы. Они были автоматически прописаны после компиляции файла с расширением sass.

После написания основной структуры страницы на HTML и внешнего стилевого оформления появился вопрос о тестировании. Веб-сайт был открыт при помощи различных браузеров.

Также в ходе тестов были проверены конкретные браузеры, к примеру браузер Internet Explorer некорректно отображал форму регистрации, и кнопки с анимацией свечения. Анимация снега не отобразилась.

Основной целью такой доскональной проверки сайта является грамотная настройка всех показателей, одной или всех страниц сайта, оперативное обнаружение и устранение всех дефектных мест на сайте, а также его адаптация под различные устройства.

Лучше всего данное приложение демонстрирует себя в Google Chrome. Это связано с тем, что данный браузер наиболее оптимизирован под скоростные приложения, в ту же очередь, данный проект напрямую зависит от производительности браузера, так как запросы, которые пользователь будет отправлять напрямую связан с запросами исходящими внутри самого браузера, поэтому добиться максимальной скорости выполнения задач можно добиться именно с данным браузером. Также Google Chrome гарантирует удобными стилями по умолчанию.

В Opera сайт показал довольно схожий результат: никаких отклонений как для мобильной версии сайта, так и для компьютерной не было выявлено, текст не уехал.

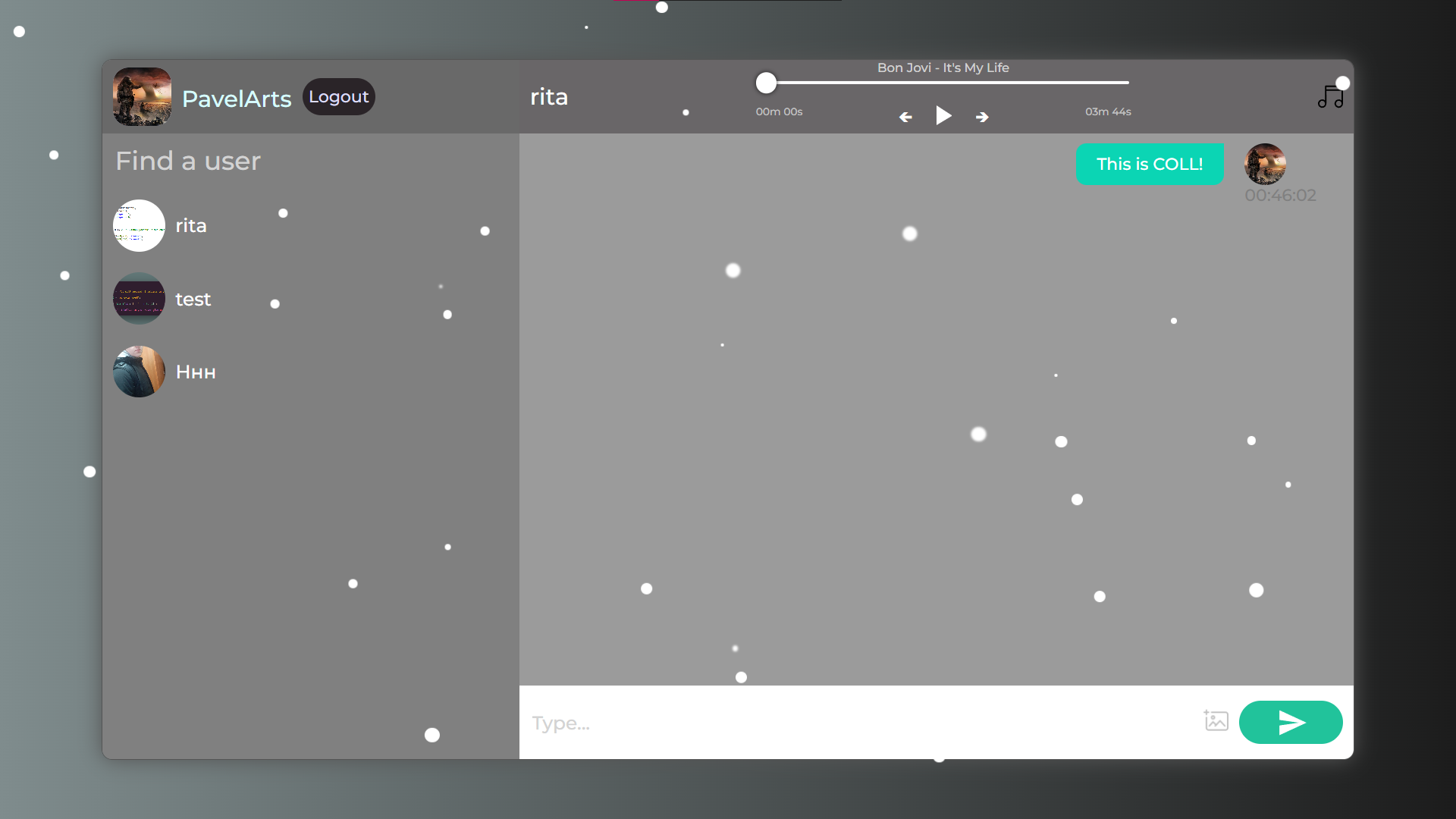


Рисунок 2.5– Отображение приложения в Google Chrome

На рисунках 2.5 представлено отображения самого приложения на странице Google Chrome. Демонстрация отличного сочетания размеров браузера с приложением.