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 2 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

Содержание

[Введение 5](#_Toc122399061)

[1 Постановка задачи 6](#_Toc122399062)

[1.1 Обзор аналогичных решений 6](#_Toc122399063)

[1.1.1 Веб приложение Телеграмм 6](#_Toc122399064)

[1.1.2 Веб приложение ВКонтакте 8](#_Toc122399065)

[1.2 Техническое задание 11](#_Toc122399066)

[2 Проектирование программного средства 13](#_Toc122399067)

[2.1 Выбор средства реализации 13](#_Toc122399068)

[2.2 Проектирование UML диаграммы 15](#_Toc122399069)

[2.3 Макетирование веб-страниц 16](#_Toc122399070)

[2.4 Разработка пользовательских элементов 17](#_Toc122399071)

[3 Структура приложения 18](#_Toc122399072)

[3.1 Реализация структуры клиентской части 18](#_Toc122399073)

[3.2 Реализация структуры серверной части 24](#_Toc122399074)

[4 Тестирование веб-сайта 29](#_Toc122399075)

[4.1 Адаптивный дизайн веб-сайта 29](#_Toc122399076)

[4.2 Кроссбраузерность веб-сайта 30](#_Toc122399077)

[4.3 Тестирование функциональности веб-сайта 31](#_Toc122399078)

[4.4 Ручное тестирование 32](#_Toc122399079)

[5. Руководство пользователю 34](#_Toc122399080)

[Заключение 36](#_Toc122399081)

[Приложения 38](#_Toc122399082)

[Приложение А Макет веб-сайта 38](#_Toc122399083)

[Приложение Б Структура JSX 39](#_Toc122399084)

[Приложение В Структура SASS 44](#_Toc122399085)

# 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.

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 курсовой работы:

* Проанализировать существующие языки разметки.
* Проанализировать инструменты и библиотеки для создания веб-приложения.
* Разработать макет приложения.
* Разработать структуру веб-приложения
* Разработать функциональную часть веб-приложения.
* Подключить облачное хранилище данных.
* Протестировать веб-приложение.
* Разработать руководство пользователю

# 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» [1] 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.

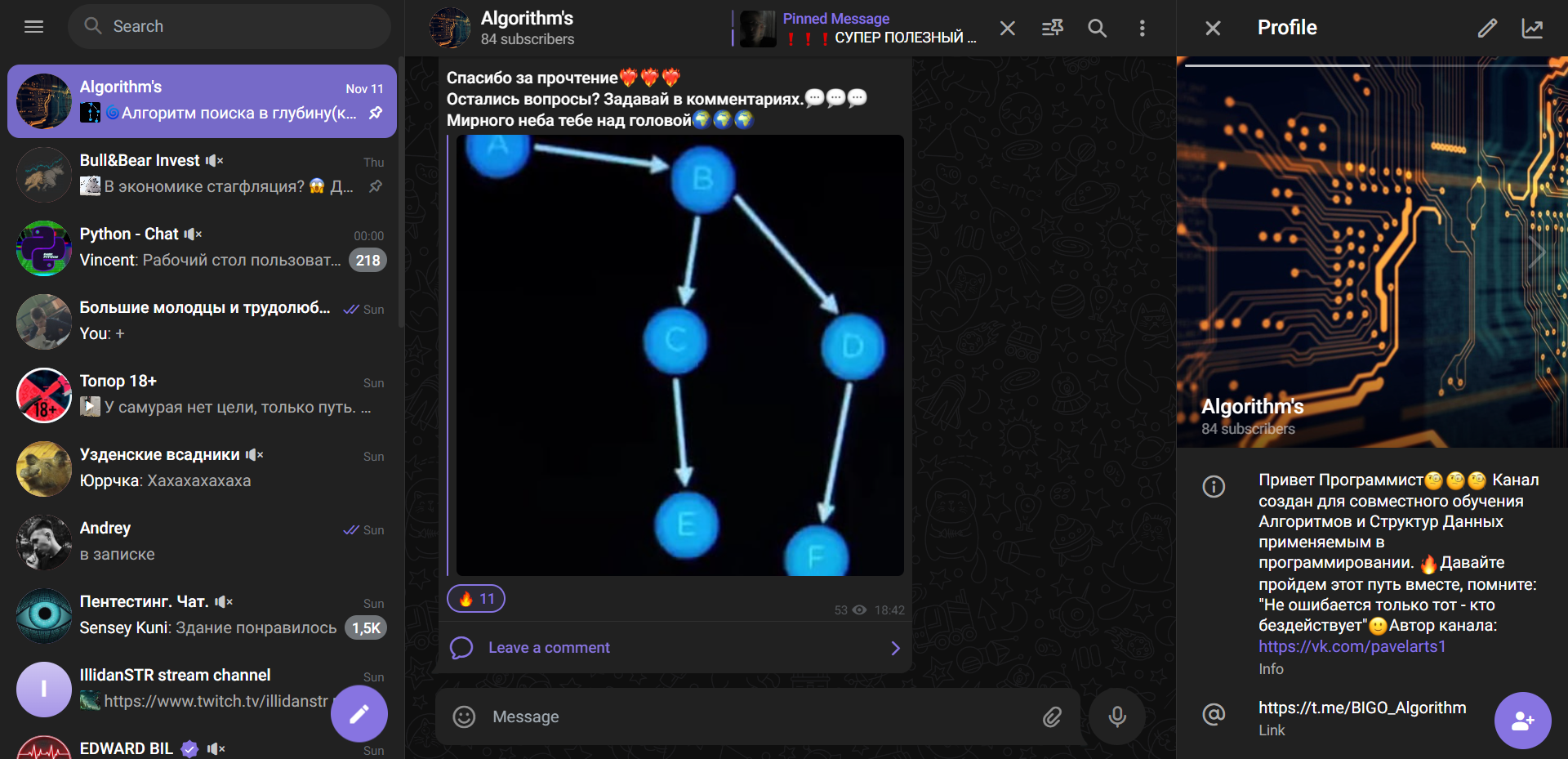
****

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.

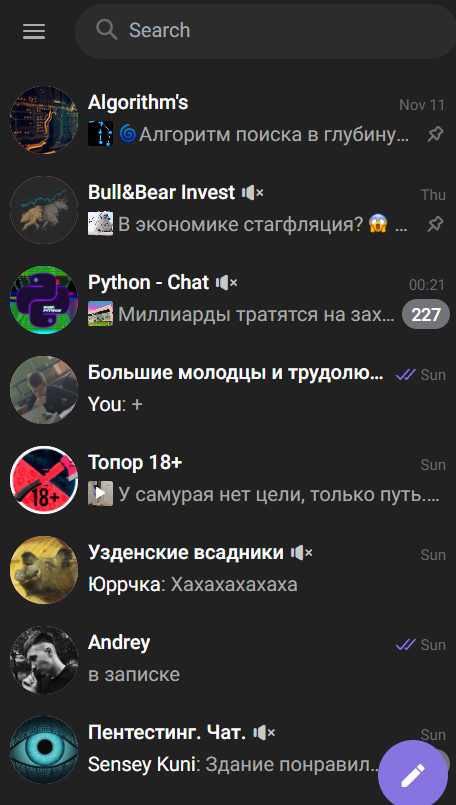


Рисунок 1.2 – Основная часть мобильной версии

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 которого можно 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 данные пользователя, переход к «слепой» версии (версия для людей с пониженным зрением), добавление контактов пользователя, а также другие вспомогательные функции.

Переключение темного мода представлено кнопкой состояния.

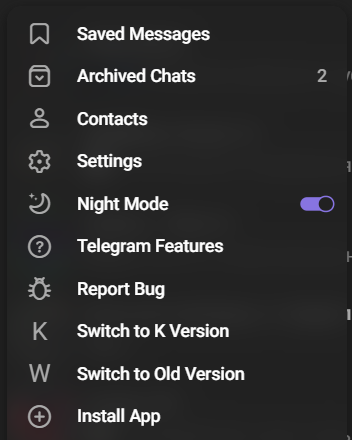


Рисунок 1.3 – Выпадающее меню мобильной версии

В пункте 1.1.1 было представлено веб-приложение «Телеграмм», которое является отличным аналогом онлайн-чата.

### Веб приложение ВКонтакте

Приложение «ВКонтакте» [2] – российская социальная сеть. 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.В данном приложении можно увидеть такие популярные функции как: Музыкальный плеер, с помощью которого пользователь может слушать интересующую его музыку, новостная лента.

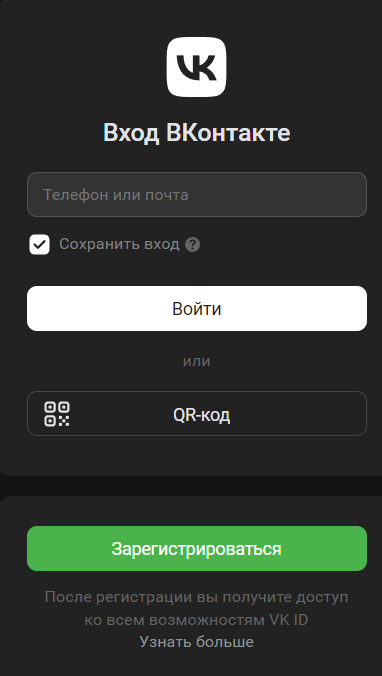


Рисунок 1.4 –Форма входа/регистрации

Также присутствует в данной форме присутсвует вход по QR-коду

На рисунках 1.5, 1.5\* представлены основная страница ВКонтакте, и вкладка мои друзья. Переходы на другие страницы реализованы с помощью «React Router».

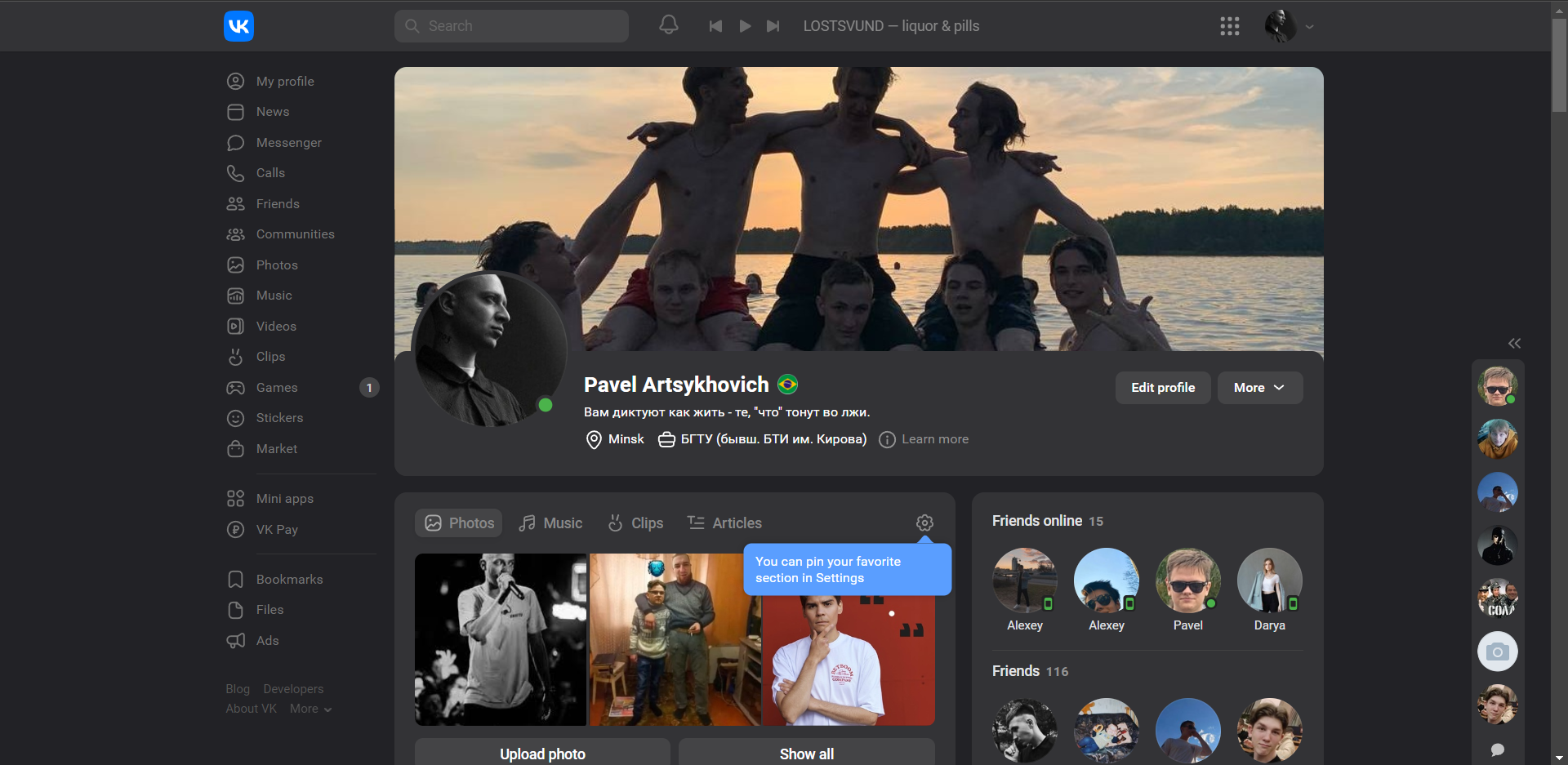


Рисунок 1.5 – Основная страница «ВКонтакте»

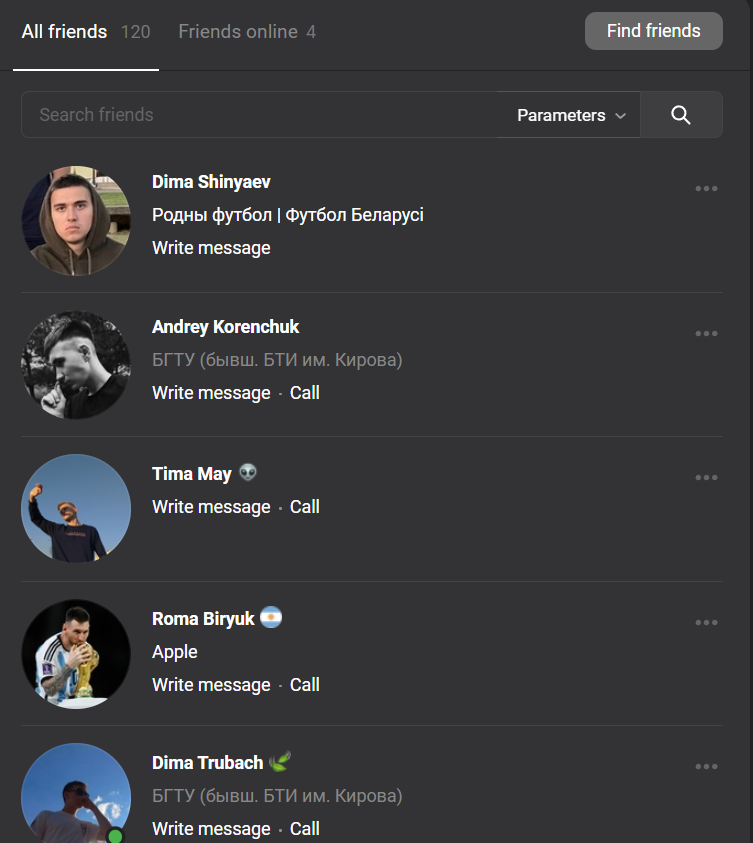


Рисунок 1.5\* – Вкладка «Мои Друзья»

На рисунке 1.5\* представлен список добавленных пользователей.

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

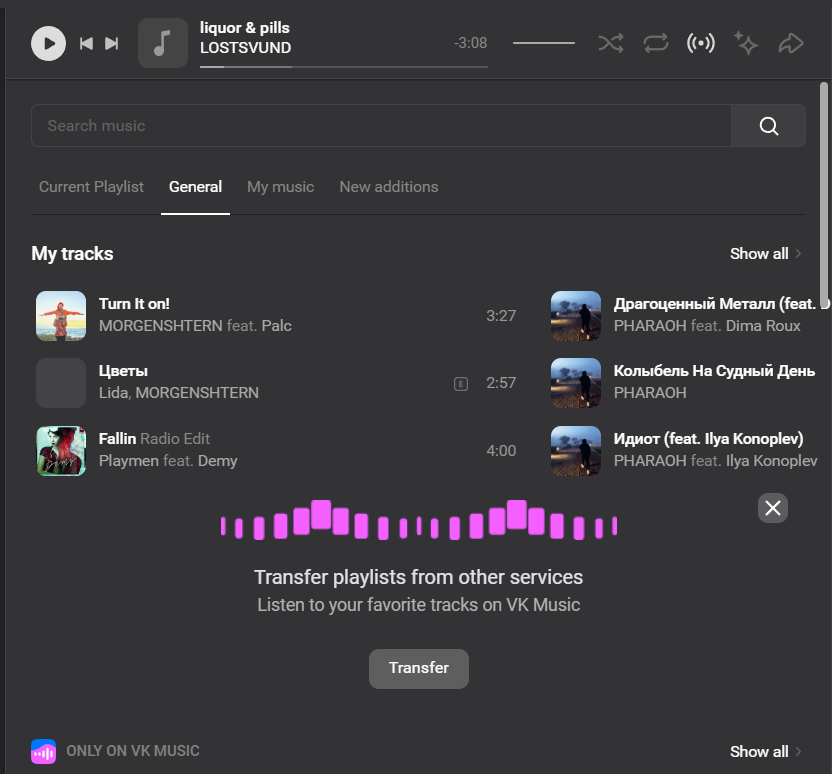


Рисунок 1.6 – Музыкальный плеер ВКонтакте

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.

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

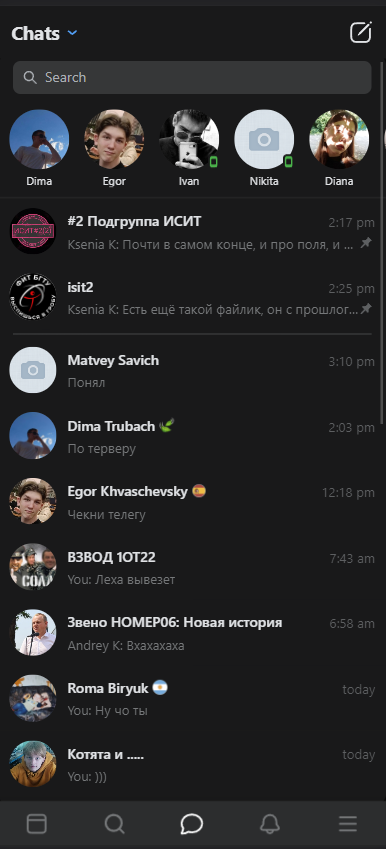


Рисунок 1.7 – Мобильная версия чата ВКонтакте

На рисунке 1.8 представлен вид выдвигающегося меню, а также возможность перехода на группы и музыку в приложении «ВКонтакте». Также на рисунке 1.7 представлен поиск, и возможность создания бесед (чатов с возможностью переписываться множества пользователей)

Также доступна вкладка перехода на профиль пользователя, а также вкладка настроек.

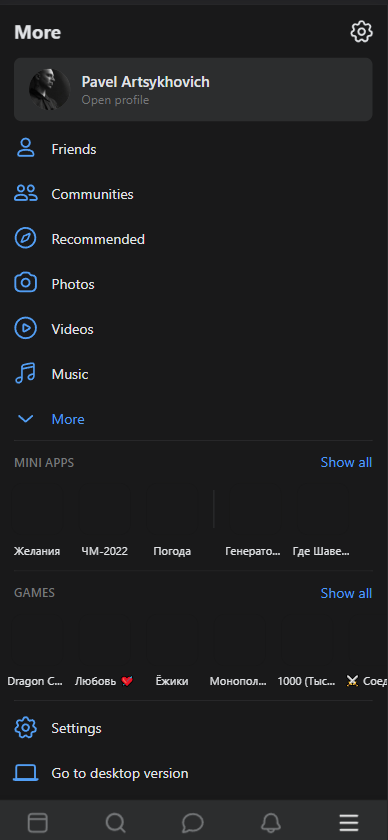


Рисунок 1.8 «Выдвигающееся меню ВКонтакте»

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». 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 входа. 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.

Таким образом можно сделать вывод: 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.

# 2 Проектирование программного средства

## 2.1 **Выбор средства реализации**

В основу основных алгоритмов данного продукта лягут алгоритмы с наибольшей эффективностью. 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. 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.

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

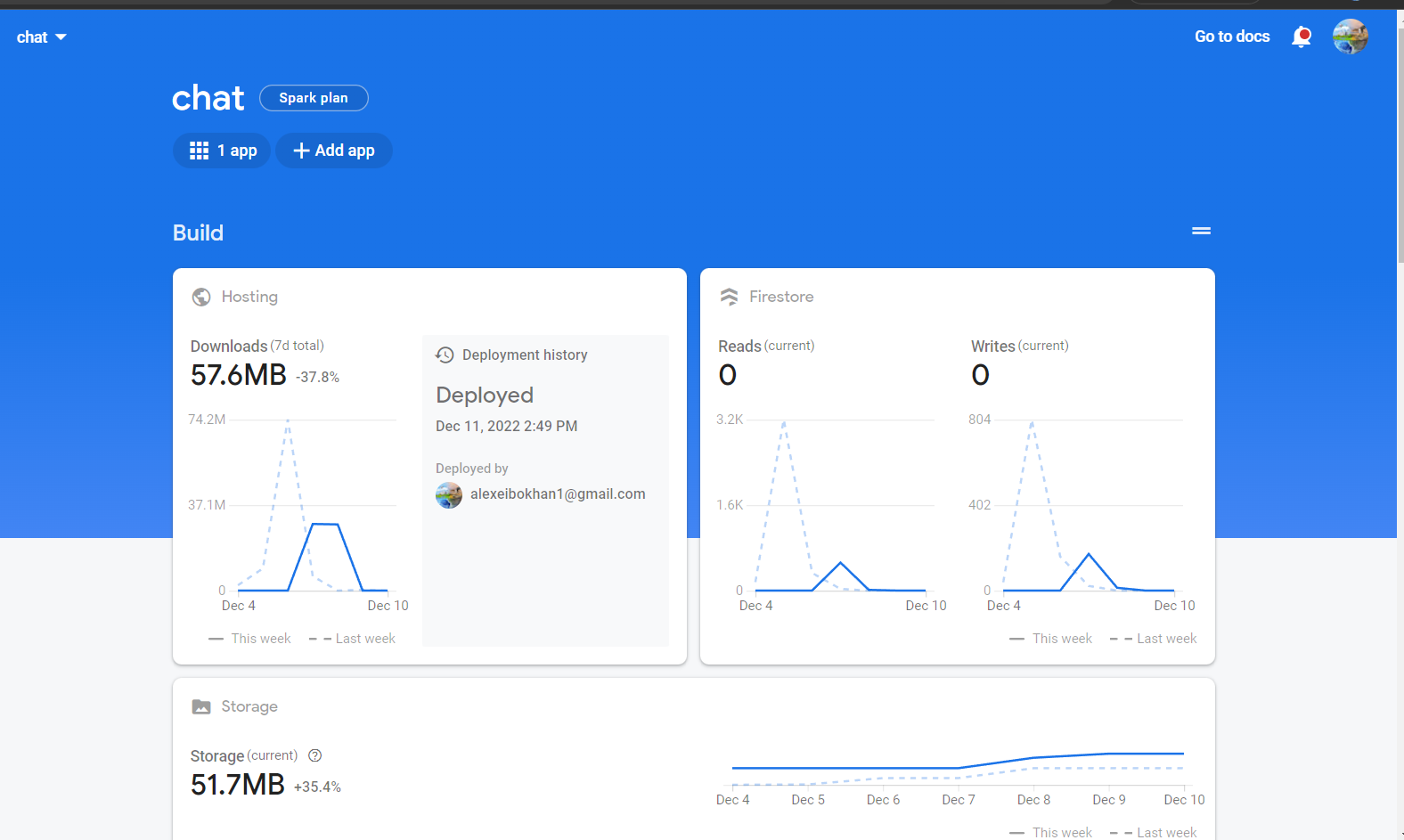


Рисунок 2.1 «Главная страница хранилища»

На рисунке 2.3 представлена таблица с пользователями.

Данные пользователей хранятся в зашифрованном виде на сервере «Firebase».

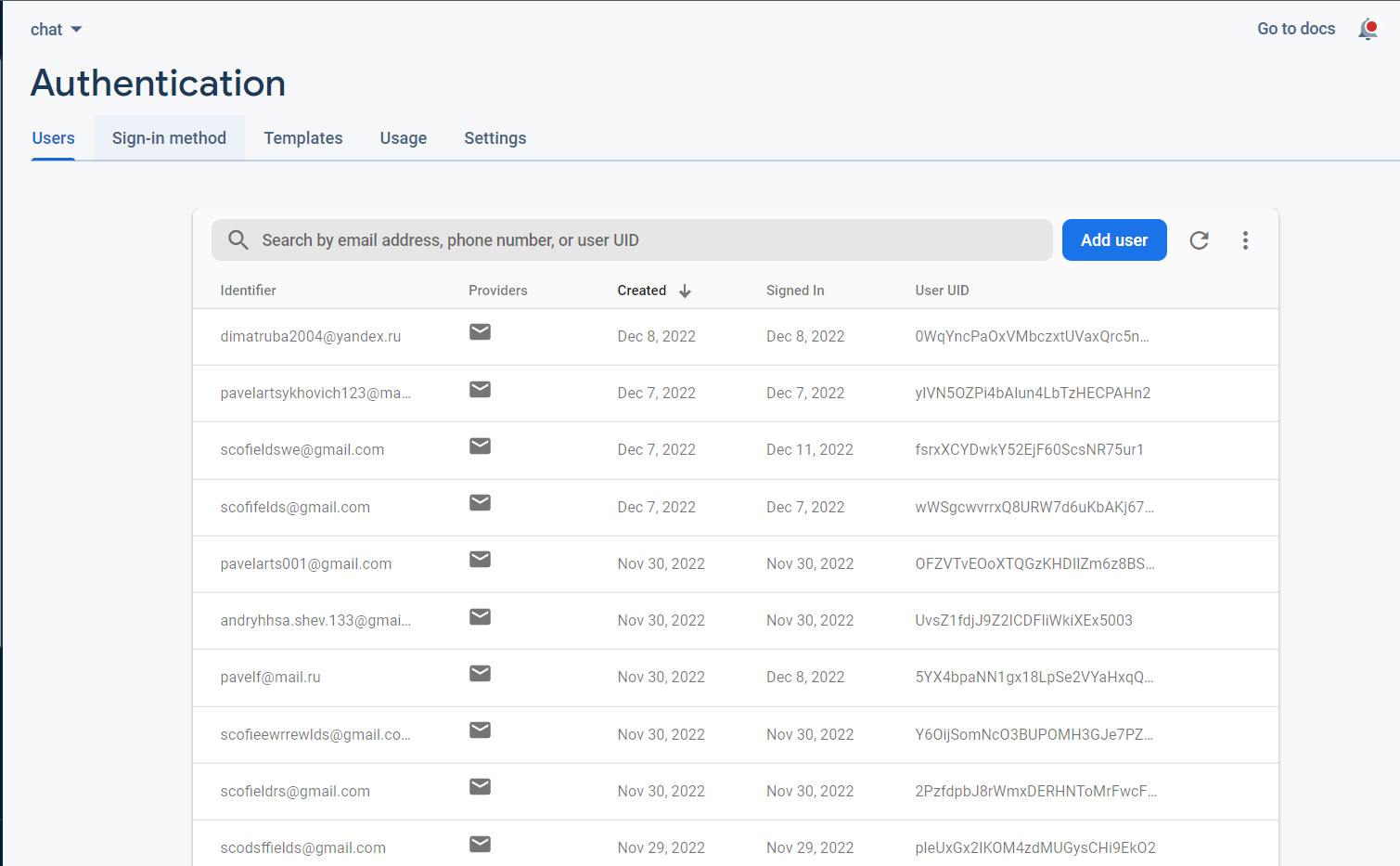


Рисунок 2.3 «Таблица Пользователей»

|  |
| --- |
| 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»

## 2.2 **Проектирование UML диаграммы**

Использование ролей прописано в Техническом задании

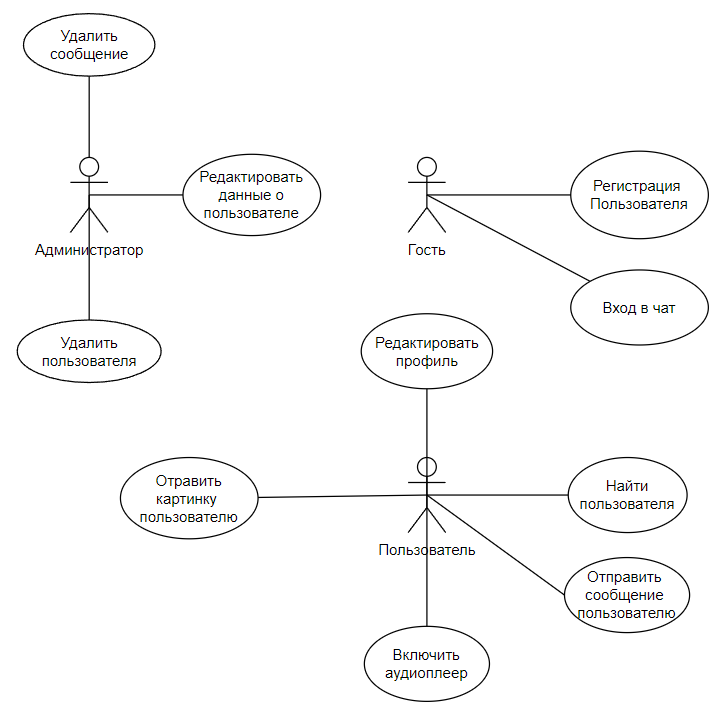


Рисунок 2.4 – Диаграмма вариантов использования

|  |  |
| --- | --- |
| Роль | Прецедент (пояснение) |
| Гость | Регистрация Пользователя |
| Гость | Вход в чат |
| Редактировать профиль | Редактирование профиля пользователя |
| Найти пользователя | Поиск пользователя. |
| Отправить сообщение пользователю | Отправка сообщения. |
| Включить аудиоплеер | Прослушивание музыки. |
| Отправить картинку пользователю | Отправка сообщения в виде картинки. |

Таблица 1 – Таблица прецедентов

## 2.3 **Макетирование веб-страниц**

На рисунке 2.5 представлен макет страницы регистрации сайта

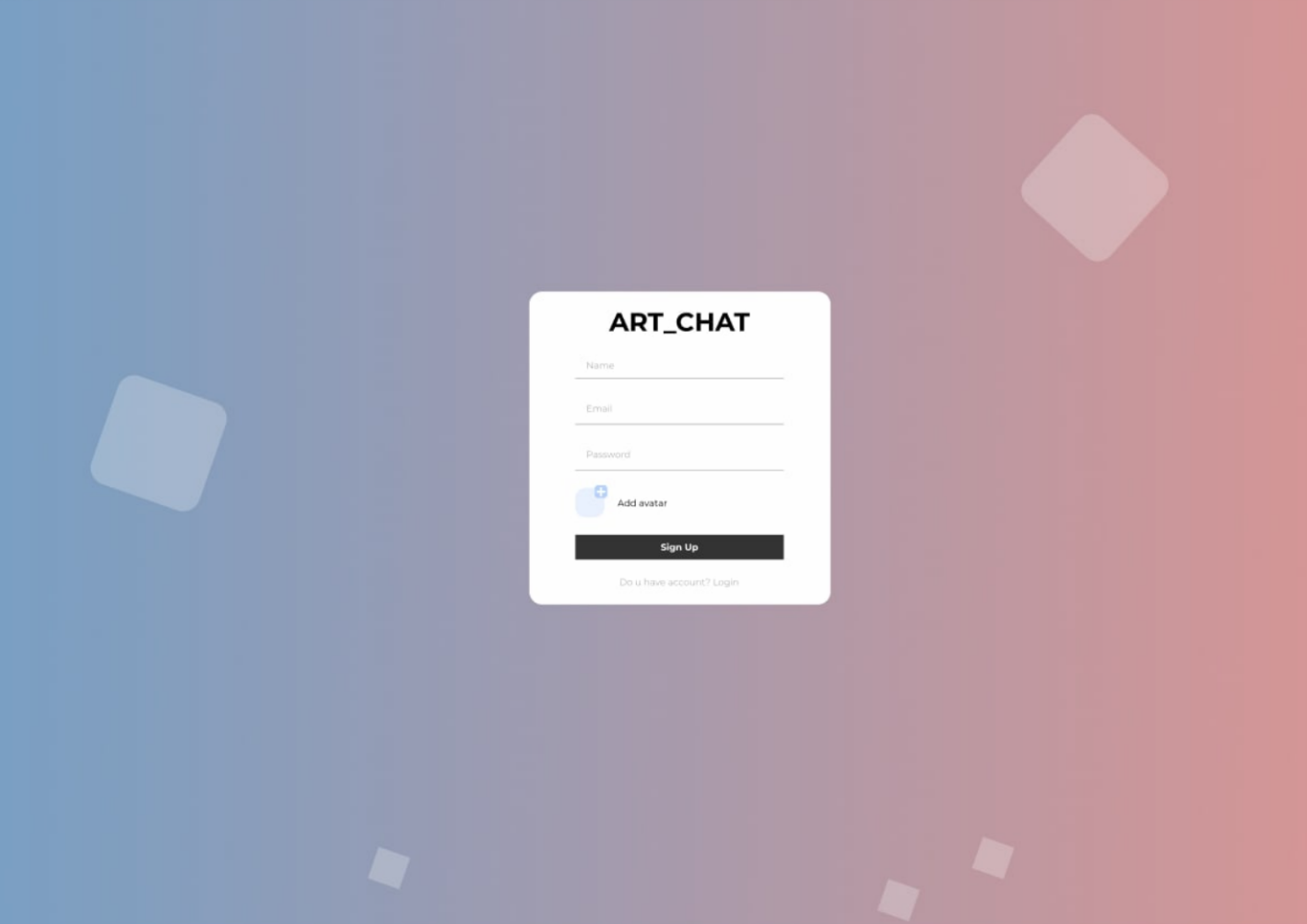
****

Рисунок 2.5 – Макет Регистрации

На рисунке 2.6 представлен макет основной части приложения.

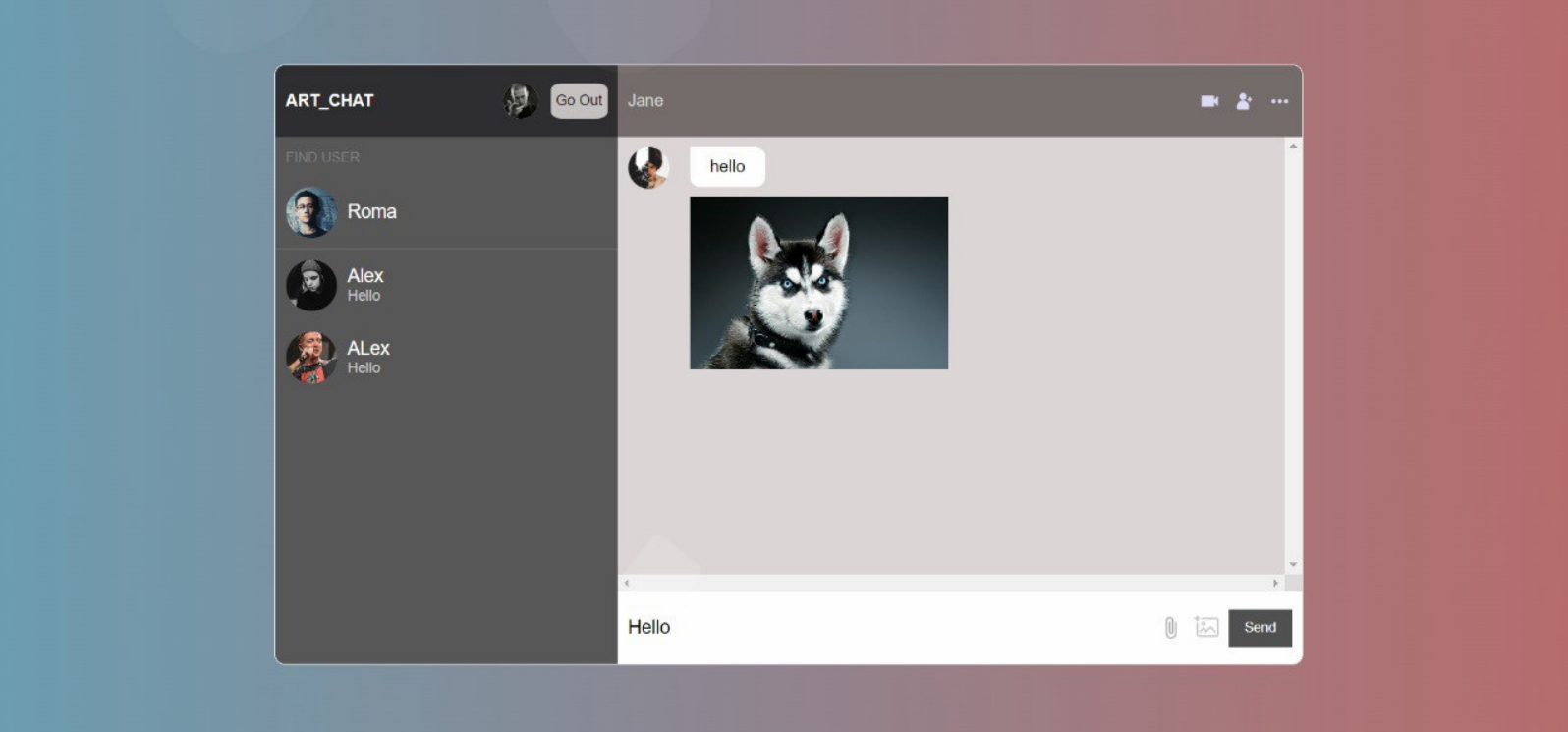


Рисунок 2.6 – Основная страница

В ходе разработки элементы макета могут изменяться

## 2.4 Разработка пользовательских элементов

В проекте будут реализованы такие элементы пользовательского интерфейса, как иконка отправки сообщения, а также стрелочки переключения музыки в аудиоплеере, представлены на рисунках 2.7, 2.8.



Рисунок 2.7 – Кнопка отправки сообщения



Рисунок 2.8 – Иконки в аудиоплеере

На рисунках 2.9, 2.10 представлены авторские кнопки перехода на домашнюю страницу, а также добавления фотографии профиля соответственно.



Рисунок 2.9 – Кнопка перехода на домашнюю страницу



Рисунок 2.10 – Кнопка добавления фотографии профиля

Таким образом можно сделать вывод: во втором разделе были затронуты темы проектирования программного средства, в котором были затронуты все основные моменты связанные с внешним видом сайта, а также разобрана часть, отвечающая за роли пользователей, которые использует приложение и часть, отвечающая за хранение данных на сервере.

# 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 3.1 – Структура JSX-формата

В листинге 3.1 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 3.2 – Структура «Login page»

В листинге 3.2 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 3.3 – Структура отображения отдельного чата

В листинге 3.3 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 3.4 – 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 3.5 – Структура основной страницы

В листинге 3.5 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 3.5 – Структура SASS

В листинге 3.5 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 3.6 – Подключение к хранилищу

В листинге 3.6 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 3.7 – Маршрутизация

В листинге 3.7 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 3.8 – Поиск Пользователей

В листинге 3.8 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 3.9 – Подключение «Redux-toolkit»

В листинге 3.9 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,    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 3.10 – Представление контекстной части

В листинге 3.10 описывается реализация контекстной части. С помощью её будет обернут файл index.js.

Таким образом можно сделать вывод: в третьем разделе была разработана структура приложения, клиентская и серверная часть. В клиентской части показана разработка визуальной части и взаимодействия между пользователями. В пользовательской части показана разработка обработки сообщений пользователями, через соединение с сервером.

# 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.

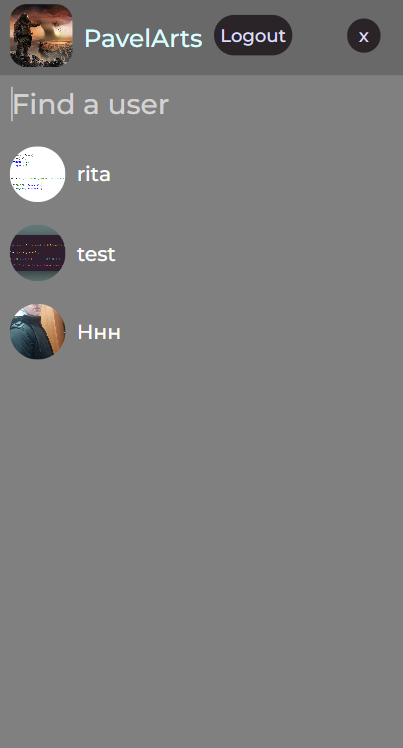


Рисунок 4.1– Основная часть мобильной версии приложения

Как видно из рисунка 4.1, чтобы контактировать с определенным пользователем нужно сначала найти его, как только пользователь пишет имя другого пользователя, браузер делает запрос к облаку, и при нахождении показывает его.

## 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. Анимация снега показывала себя не с самой хорошей стороны.

Firefox также показал хороший результат, текст не съехал, кнопки работали отлично. Единственным отличием была анимация выпадающего снега, иногда он был выходил за рамки размеров самого браузера, это связано с тем, что размеры задающие анимацию изменяются динамически, для того чтобы снег мог лететь в различных состояний размера. Но преимуществом работы приложения в этом браузере является то, что данный браузер показывает наиболее благоприятную скорость ответа сервера на запросы, отображая их на самой странице. В связи с этим Firefox отличный браузер для работы с данным приложением.

Internet Explorer показал неудовлетворительный результат. Ни отображая практически никакие элементы на странице.

На рисунке 4.2 представлено отображения самого приложения на странице Google Chrome. Демонстрация отличного сочетания размеров браузера с приложением.

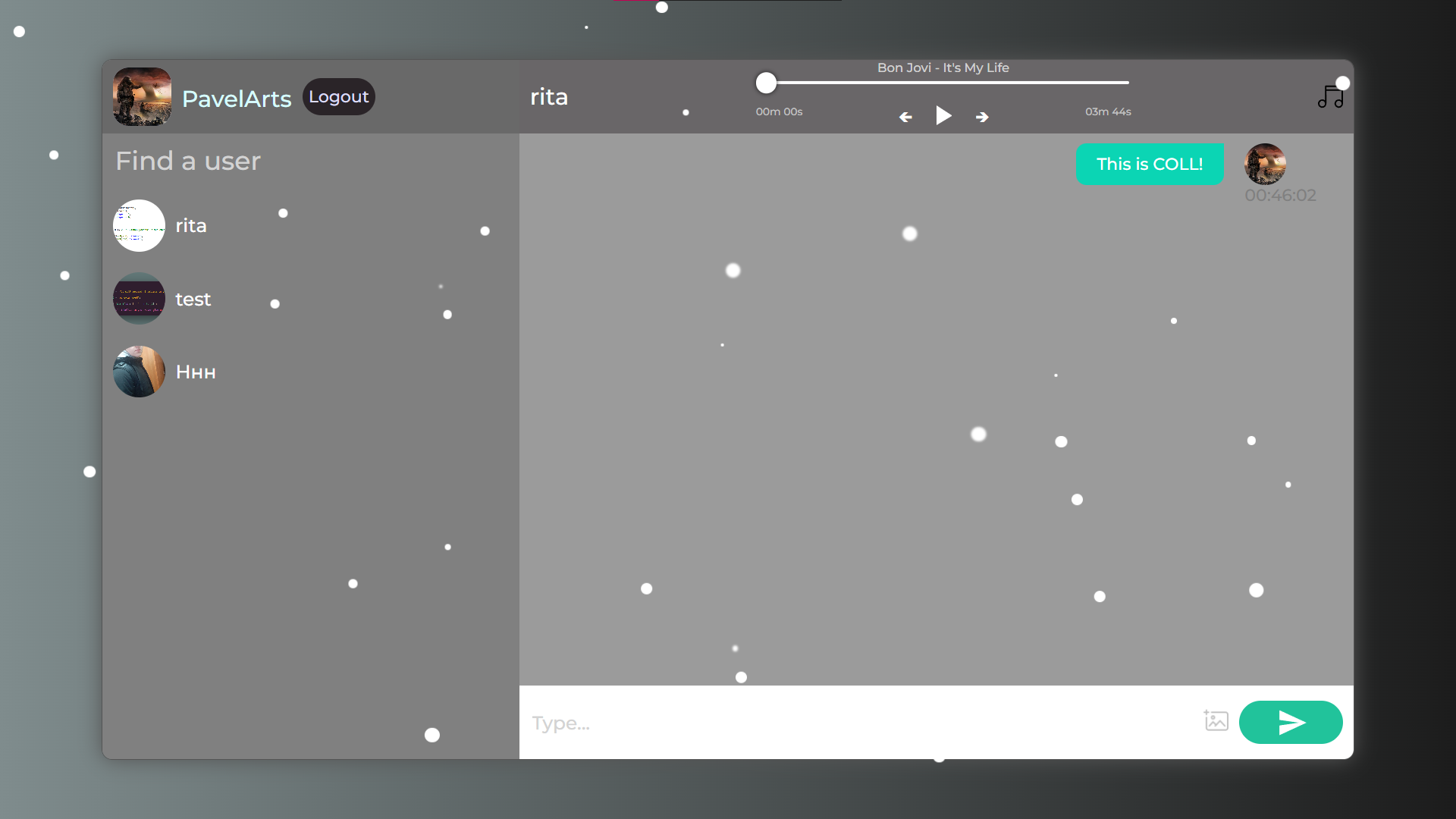


Рисунок 4.2– Отображение приложения в Google Chrome

## 4.3 Тестирование функциональности веб-сайта

Тест на функциональность веб-сайта – это проверка всех функций сайта, которые должны работать корректно, а также проверка всех элементов, которые должны быть доступны для пользователя. Основными функциями приложения являются отправка сообщения, поиск пользователей, воспроизведение аудиоплеера, регистрация и аутентификация пользователей, а также просмотр и редактирование личных данных. Тестирование функционала сайта было выполнено вручную с использованием следующие браузеров Google Chrome , Opera и Firefox.

Google Chrome корректно отобразил все функции веб сайта, отлично проработаны все разделы приложения. А также отлично показана анимация падения снега.

Opera показала, также как и Google Chrome отличные результаты в выполнении всех функций приложений. Но возникли проблемы с шириной анимации снега. Иногда она превышала размеры Opera, в следствии чего появляется горизонтальный scrool.

Firefox показал хороший результат в выполнении большинства функций, но возникли проблемы с отображением анимации снега, а именно выход за пределы высоты и ширины экрана браузера. В следствии чего появление горизонтального и вертикального scrool.

Динамика сайта на разных устройствах – наиболее важная часть в написании сайта, поэтому, на данном этапе было проанализировано поведение веб-сайта при использовании с разных устройств, а также веб-ресурс был протестирован в большом количестве браузеров. По итогу выполнения данного этапа для веб-сайта был разработан адаптивный дизайн и кроссбраузерность. Работа, проделанная с сайтом, позволяет пользователям ориентироваться по сайту. Лишних элементов нет, что упрощает пользование сайтом. При пользовании сайтом при помощи мобильного устройства некоторые элементы меняют свое положение, для более удобного пользования.

## 4.4 Ручное тестирование

Тестирование frontend части включает в себе проверку правильности осуществления перехода между страницами, которые совершают пользователи системы, также проверку правильности отправления запросов на сервер, их обработки frontend сервером.

Рассмотрим сценарий перехода к главной странице чата. Для начала необходимо зарегистрироваться. При заполнении формы неправильными данными должно быть получено оповещение об ошибки “Something went wrong». При вводе, же, верных данных пользователя перенесет на основную страницу, где из роли гостя будет произведено перенаправление на роль пользователя. Заполнение формы неправильными данными и уведомление об ошибке изображено на рисунке 4.3.

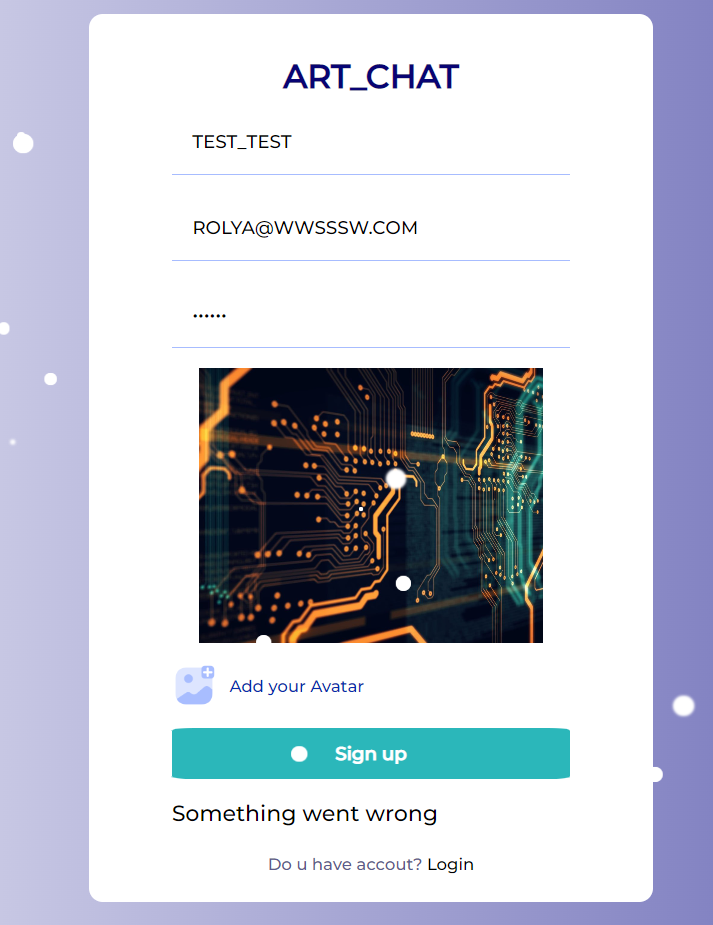


Рисунок 4.3 – Ввод неверных данных

На рисунке 4.4 представлена проверка на ввод пользователя неверных данных в форму входа. В данной форме дополнительно будет выполняться проверка на существующий аккаунт, где сравнение будет происходить по данным электронной почты. Где видно, что ввод неверного логина и пароля, введёт к ошибке входа в профиль.

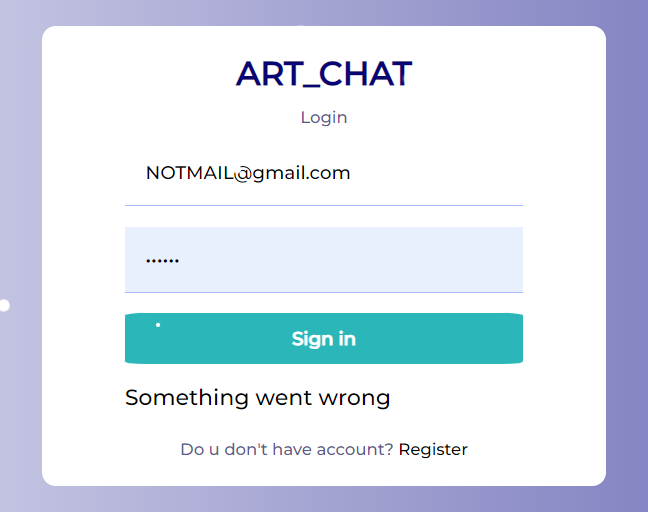


Рисунок 4.4 – Ошибка входа

Таким образом: главной целью такой тщательной 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.

# 5. Руководство пользователю

Данное приложение предоставляет возможность приятно пользоваться функциями реального чата, также прослушивать приятную музыку, также реализован приятный интерфейс, понятен даже новичку. При входе в приложение будет предложено войти, либо же зарегистрироваться новому пользователю. После входа будут доступны все функции приложения.

Для обеспечения надежности, а также удобства данного веб-сайта выполнено:

* разработан удобный динамический пользовательский интерфейс, без лишних страниц, но со всем необходимым функционалом;
* разработан удобный, быстрый аудиоплеер, в котором можно слушать музыку.

Во вкладке “Register” пользователь может зарегистрировать аккаунт на сайте, введя свои данные и фотографию.

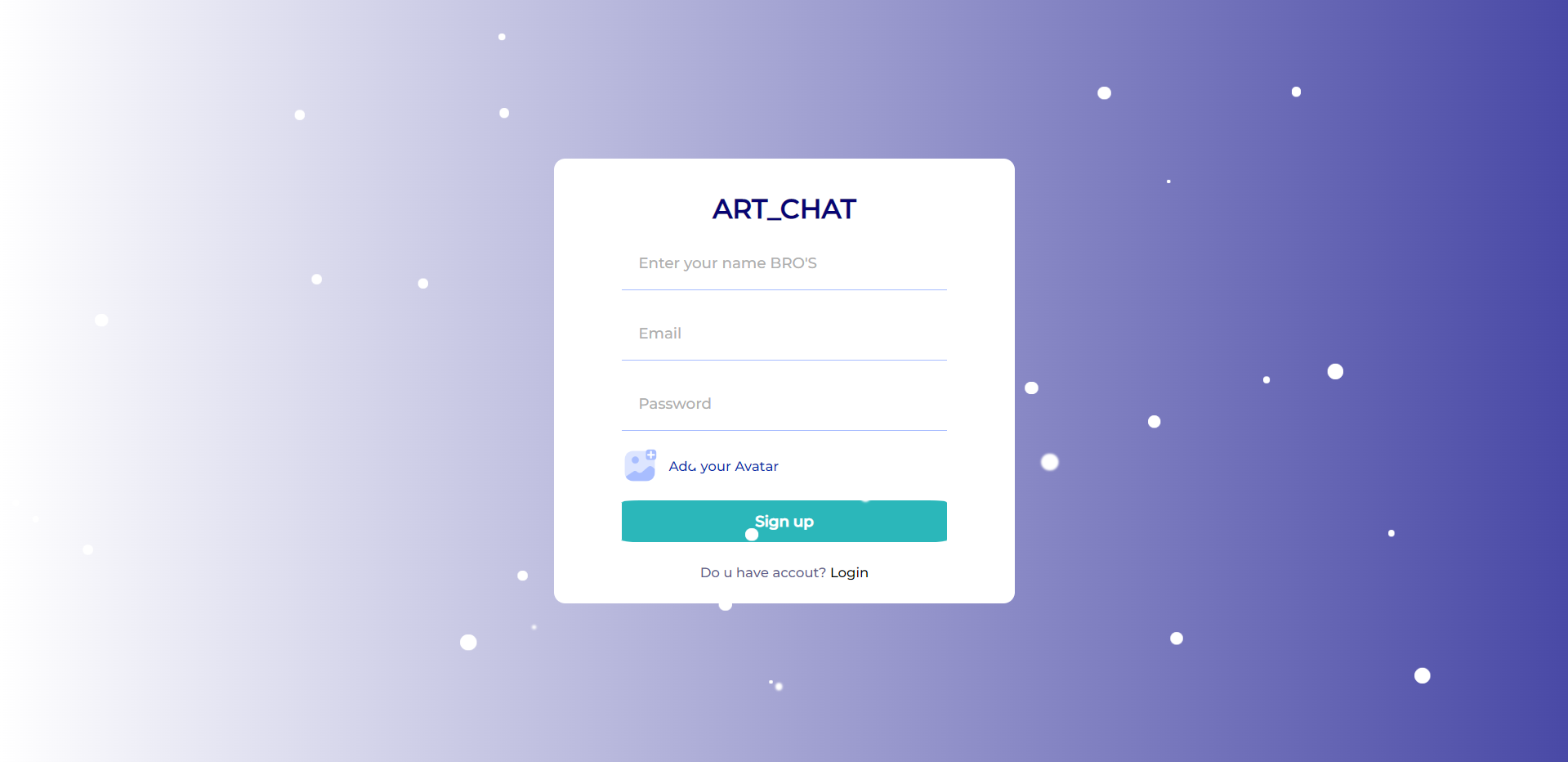


Рисунок 5.1 – Форма регистрации

Во вкладке “Login” пользователь может войти во вкладку “Home” под своими данными.

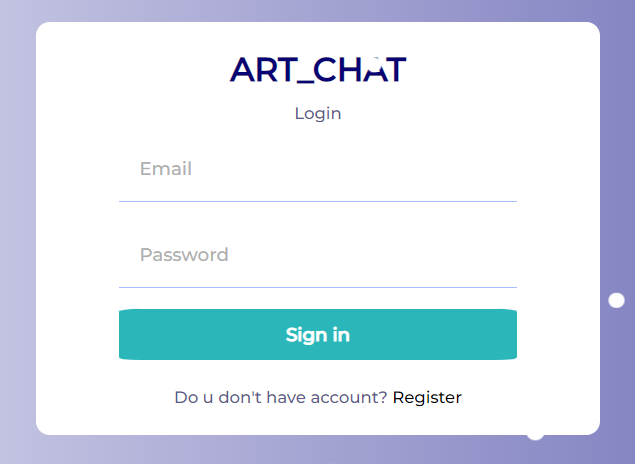


Рисунок 5.2 – Форма входа

Во вкладке “Home” показан сам чат, в котором справа можно найти нужных нам пользователей. Отображен сам аудиоплеер, поиск users, а также чат переписки.

А также при наведении на картинку, анимация подсветки входа в профиль.

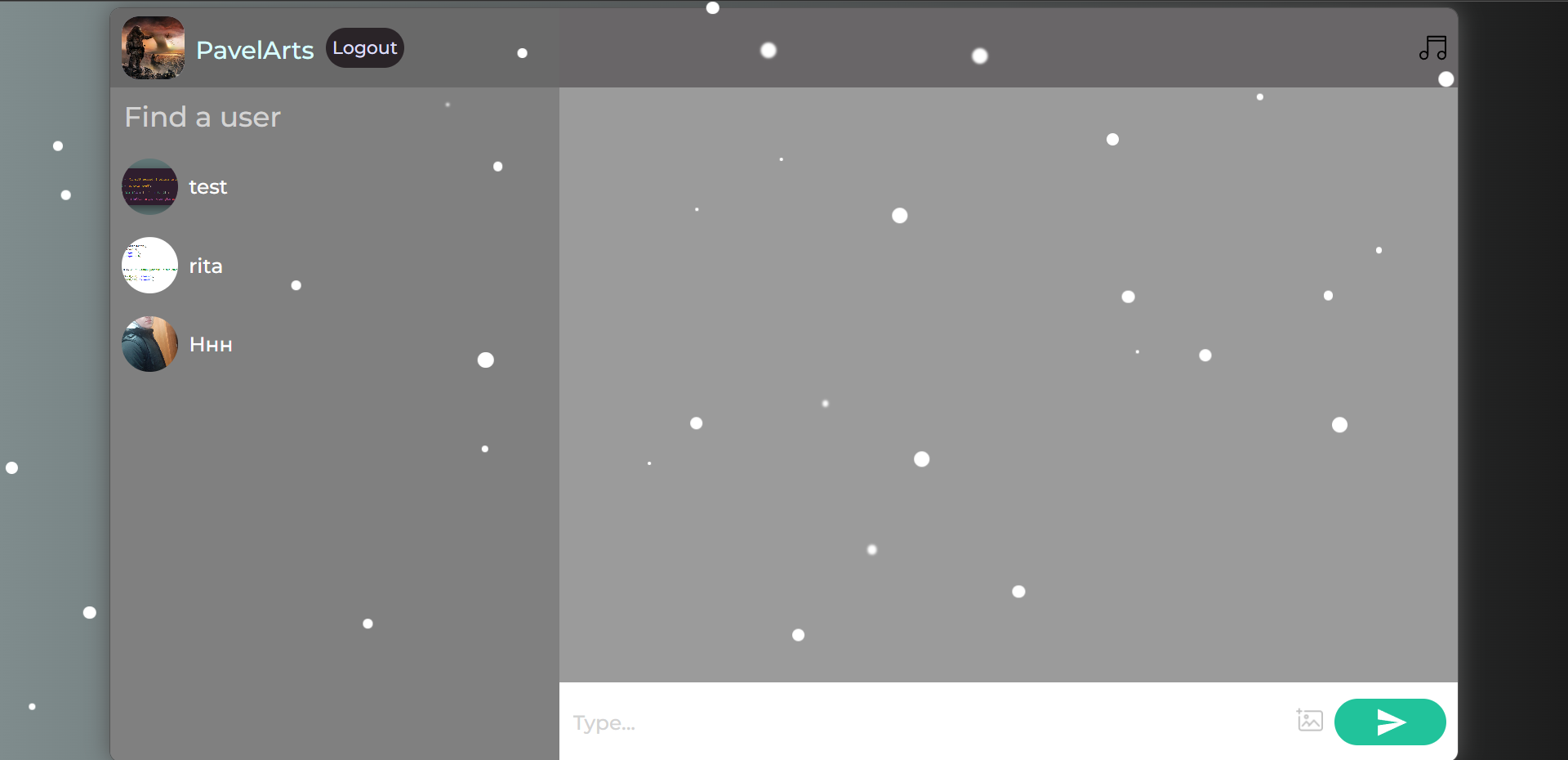


Рисунок 5.3 – Главная страница чата

Во вкладке “Profile” пользователь может изменить данные, меняя имя, фотографию и пароль.

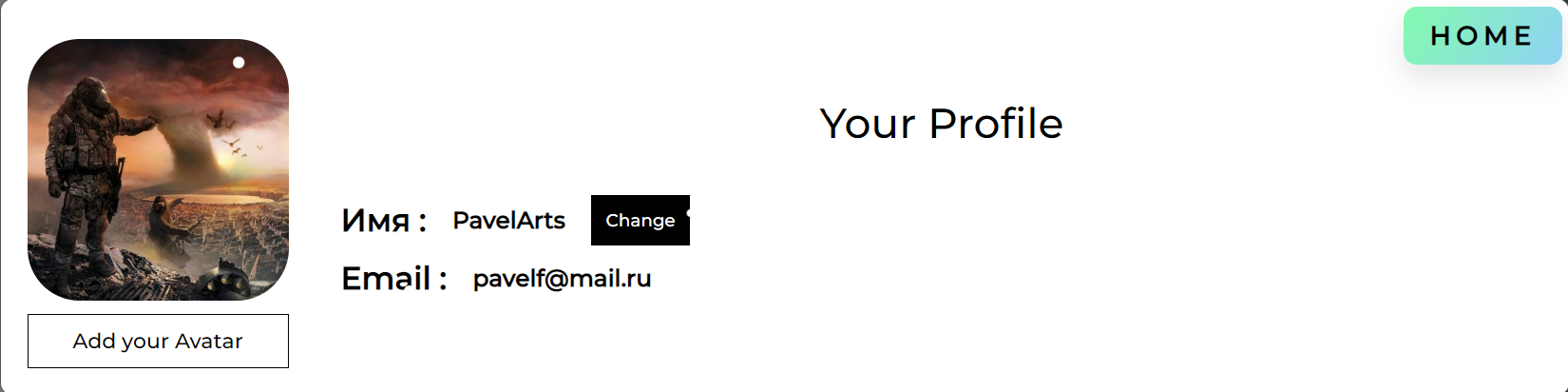
****

Рисунок 5.4 – Профиль пользователя в чате

Таким образом: в разделе Руководство пользователю представлены все основные страницы сайта, и разобран порядок начала работы с приложением, начиная от регистрации на сайте, заканчивая изменением данных о пользователе

# Заключение

Целью курсового проекта была разработка удобного приложения онлайн-чата для изучения переписки в реальном времени, с прослушиванием приятных треков.

Для реализации макета веб-страницы были исследованы сайты схожей тематикой и исследованы недостатки веб-приложения. Показана и предложено иные реализации блоков веб-страницы.

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

По итогам реализации веб-сайта, было принято решения о тестировании веб-сайта. В результате тестирования была обнаружена неспособность веб-браузера(Internet Explorer) работать с практическими всеми функциями сайта.

Реализация важных элементов страницы либо же скриптов были представлены в приложениях. Которые дают общую информацию о проекте.

Для достижения поставленной цели для курсового проекта функционально были реализованы следующие задачи:

* анализ аналогичных решений;
* выбор способа верстки;
* выбор стилевого оформления;
* реализация соединения с облачным хранилищем;
* добавление веб-сайту адаптивности;
* кроссбраузерность веб-сайта;
* создание руководства пользователя;

Результатом курсового проекта стал готовый к работе веб-сайт, выложенный на хостинг. В дальнейшем разработанный веб-сайт можно будет использовать в реальной жизни, как конкурент современным онлайн-чатам. Так же возможно расширение списка предоставляемых функций. Данный ресурс стабилен в работе и не требует высоких знаний для работы с ним.

Таким образом, из выше сказанного, можно сделать вывод, что в курсовом проекте раскрывается все теоретические и практические аспекты темы и достигнуты поставленные цели и задачи роботы.

**Список использованных литературных источников**

1. Социальная сеть «ВКонтакте» [Электронный ресурс] / Режим доступа : [https://vk.com](https://vk.com.org) – Дата доступа 20.12.2022.
2. Социальная сеть «Телеграмм» [Электронный ресурс] / Режим доступа : <https://web.telegram.org> – Дата доступа 20.12.2022.

# Приложения

## Приложение А Макет веб-сайта

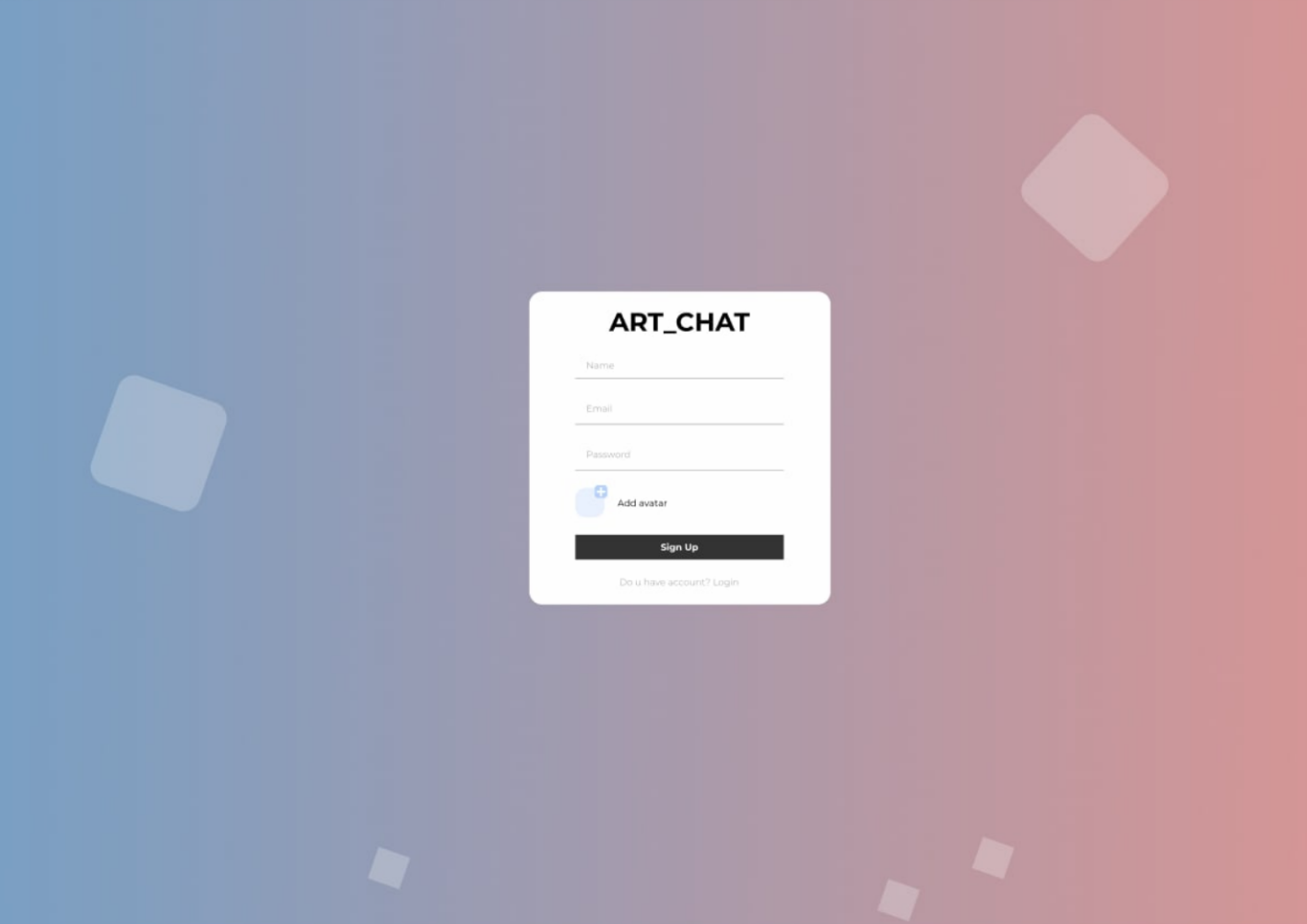
****

Рисунок 1 – Макет Регистрации

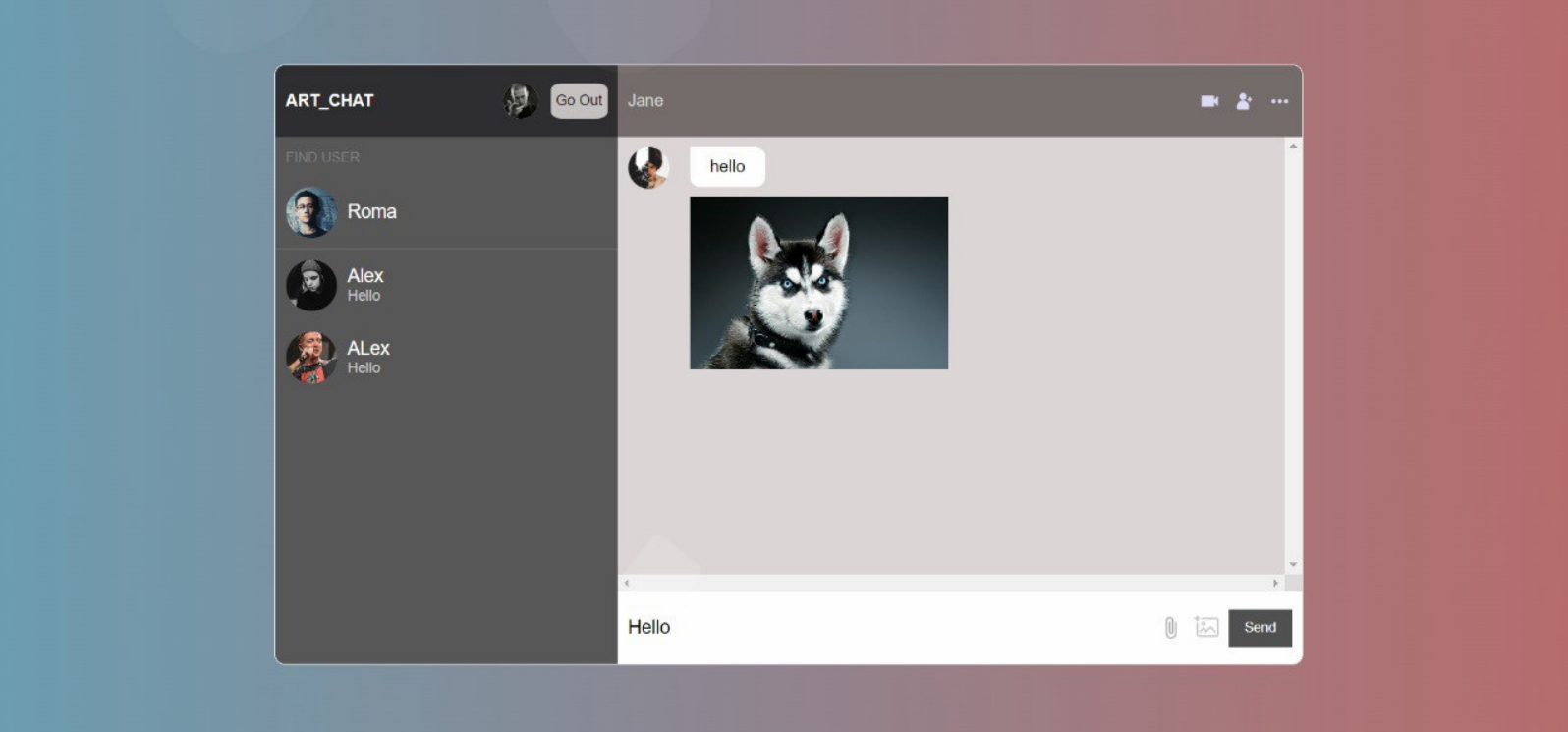


Рисунок 2 – Макет Основной страницы

## Приложение Б Структура JSX

|  |
| --- |
| 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 класса Firebase

|  |  |
| --- | --- |
| 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 класса – Routes

|  |
| --- |
| 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 класса Search.

|  |
| --- |
| 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 класса message-slice

|  |
| --- |
| 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 класса Context

## Приложение В Структура SASS

|  |
| --- |
| @mixin mobile {    @media screen and (max-width: 480px) {        @content;    }  }  @mixin tablet {    @media screen and (max-width: 768px) {      @content;    }  }  @mixin laptop {    @media screen and (max-width: 1200px) {      @content;    }  }    .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%;        }        label {          display: flex;          align-items: center;          gap: 10px;          color: #042cad;          font-size: 12px;          cursor: pointer;          img {            width: 32px;          }        }      }      p {        color: #5d5b8d;        font-size: 12px;        margin-top: 10px;      }    }  }  .home {    background: -webkit-linear-gradient(90deg, #380036,#0CBABA);    background: linear-gradient(90deg, #7F8C8D,#1c1c1c);    height: 100vh;    display: flex;    align-items: center;    justify-content: center;          \*::-webkit-scrollbar,    \*::-webkit-scrollbar-thumb {      width: 26px;      border-radius: 25px;      background-clip: padding-box;      border: 10px solid transparent;    }      \*::-webkit-scrollbar-thumb {      color: rgb(76, 75, 75);      box-shadow: inset 0 0 0 10px;    }    // Убрать пространство между горизонтальным и вертикальным скроллбарами    \*::-webkit-scrollbar-corner {      background-color: transparent;    }        .container {      margin: 10px;      border: 1px solid white;      border-radius: 10px;      max-width: 1200px;    width: 100%;      display: flex;      overflow: hidden;      border: 1px solid #555;          box-shadow:0 0 20px #555;    .sidebar {        flex: 1;        background-color: #808080;        position: relative;          height: 600px;        position: relative;        padding-top: 70px;        overflow: auto;        overflow-x: hidden;        .navbar {          display: flex;          align-items: center;          background-color: #696969;          height: 50px;          padding: 10px;          justify-content: space-between;          color: #ddddf7;          position: absolute;          top:0;          left: 0px;          width: 100%;            .logo {            font-weight: bold;            font-size: 18px;            margin-top: 10px;            color: #07021b;              @include tablet {              display: none;            }          }            .user {            display: flex;            gap: 10px;            font-size: 22px;            align-items: center;              a{              display: flex;              gap: 10px;              font-size: 22px;              align-items: center;              cursor: pointer;            }              img {              background-color: #ddddf7;              &:hover {                background-color: #2a2428;                color: #ddddf7;                box-shadow: 0 0 10px #ddddf7, 0 0 40px #ddddf7, 0 0 80px #ddddf7;                animation: neon 1.5s ease-in-out infinite alternate;              }              height: 56px;              width:  56px;              border-radius: 27%;              object-fit: cover;              }            button {                background-color: #2a2428;              color: #ddddf7;              height: 36px;              width: fit-content;                border-radius: 18px   ;              font-size: 16px;              border: none;              cursor: pointer;                        }            .close{              position: absolute;              right: 20px;              top:20px;              width: 30px;              height: 30px;              display: none;              }          }        } |

Листинг класса style