Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

«Чат»

Студент гр. 753501
В. Н. Яньчук
Руководитель
И. А. Удовин

Минск 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В XXI веке сложно представить сферу жизни, в которой мы бы не использовали интернет. Он позволяет нам за несколько секунд найти любую интересующую нас информацию, узнать все самые свежие новости, пока едешь домой. Это далеко не все возможности интернета, он также является незаменимым инструментом для коммуникации в наши дни. Тема данной курсовой работы – мессенджеры. Их популярность растёт из года в год. Они удобны и не требуют особых навыков при использовании, поэтому даже тот человек, который далёк от современных технический изобретений, легко разберётся с применением. Если раньше мессенджеры были популярны, в основном, при неформальном общении с друзьями и родственниками, то в последние годы они стали неотъемлемой частью рабочего процесса. Это касается не только работы в сфере программирования — коммуникация в чатах осуществляется почти повсеместно, начиная со школьных групп, где в условиях пандемии учитель может сообщить детям домашнее задание, заканчивая обсуждением подробностей заключения контрактов в крупных фирмах.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Краткие теоретические сведения

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных [1].

Система управления базами данных, сокр. СУБД (англ. Database Management System, сокр. DBMS) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных. По модели данных выделяют следующие типы СУБД: иерархическую, объектную и объектно-ориентированную, объектно-реляционную, реляционную, сетевую, функциональную [2].

Реляционная база данных — это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных. В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке – значение атрибута. Каждая стока таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей. К этим данным можно получить доступ многими способами, и при этом реорганизовывать таблицы БД не требуется [3].

Нереляционная база данных — это база данных, в которой, в отличие от большинства традиционных систем баз данных, не используется табличная схема строк и столбцов. В этих базах данных применяется модель хранения, оптимизированная под конкретные требования типа хранимых данных. Например, данные могут храниться как простые пары "ключ — значение", документы JSON или граф, состоящий из ребер и вершин. Термин NoSQL применяется к хранилищам данных, которые не используют язык запросов SQL, а запрашивают данные с помощью других языков и конструкций [4].

1.2. Технологии и средства разработки

Visual Studio Code — редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией. Visual Studio Code основан на Electron и реализуется через веб-редактор Monaco, разработанный для Visual Studio Online (см. рис. 1.1) [5].

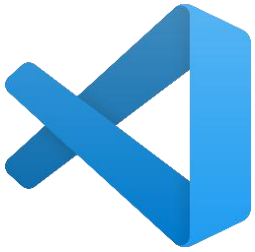

Рисунок 1.1. - Иконка VS Code.

HTML5 (HyperText Markup Language) — язык для структурирования и представления содержимого всемирной паутины. Хотя стандарт был завершён (рекомендованная версия к использованию) только в 2014 году (предыдущая, четвёртая, версия опубликована в 1999 году), уже с 2013 года браузерами оперативно осуществлялась поддержка, а разработчиками — использование рабочего стандарта (англ. HTML Living Standard). Цель разработки HTML5 — улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров (см. рис. 1.2) [6].

Рисунок 1.2. - Иконка HTML5.

CSS3 (Cascading Style Sheets) — формальный язык описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки (чаще всего HTML или XHTML). Также может применяться к любым XML-документам, например, к SVG или XUL. CSS используется создателями веб-страниц для задания цветов, шрифтов, стилей, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось отделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS) (см. рис. 1.3) [7].

Рисунок 1.3. - Иконка CSS3.

JavaScript — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262). JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам. Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса. (см. рис. 1.4) [8].

Рисунок 1.4. - Иконка JavaScript.

Firebase — американская компания, поставщик облачных услуг, основанная в 2011 году Эндрю Ли и Джеймсом Тэмплином, и поглощённая в 2014 году корпорацией Google. Основной сервис — облачная СУБД класса NoSQL , позволяющая разработчикам приложений хранить и синхронизировать данные между несколькими клиентами. Поддержаны особенности интеграции с приложениями под операционные системы Android и iOS, реализовано API для приложений на JavaScript, Java, Objective-C и Node.js, также возможно работать напрямую с базой данных в стиле REST из ряда JavaScript-фреймворков, включая AngularJS, React, Vue.js, Ember.js и Backbone.js. Предусмотрено API для шифрования данных.

Среди других услуг, предоставлявшихся компанией — запущенный 13 мая 2014 года хостинг для хранения статических файлов (таких как CSS, HTML, JavaScript), обеспечивающий доставку через CDN и сервис аутентификации клиента с использованием кода только на стороне клиента с поддержкой входа через Facebook, GitHub, Twitter и Google (Firebase Simple Login) (см. рис. 1.5) [9].

Рисунок 1.5. - Иконка Firebase.

Firebase Realtime Database — это облачная база данных NoSQL. Данные хранятся в формате JSON и синхронизируются в реальном времени с каждым подключённым клиентом.

1.3. Модули программ и их описание

В рамках исследования эффективности различных подходов к созданию двух баз данных в данном курсовом проекте реализовано несколько модулей, каждый из которых выполняет определенную задачу:

- **Клиент**, который представлен файлами HTML, CSS, JS и необходим для выполнения задач отрисовки элементов в браузере пользователя. Также выполняет функцию взаимодействия с пользователем, включая запись данных в Firebase Realtime Database и Firebase Cloud Storage.
- **Firebase Server** обрабатывает всю серверную логику: аутентификация, база данных реального времени, хостинг, облачное хранилище.

Кроме того, в состав проекта вошло множество файлов, которые отвечают за различные операции в программе (см. рис. 1.6):

- **public/index.html** — основной исходный файл, в котором происходит подключение всех библиотек и модулей. В нем также содержится код подключения к удалённому серверу и хранилищу. Индексный файл ищется по умолчанию веб-сервером, если в URL указан не файл, а каталог.
- **public/app.js** — файл, который содержит логику системы маршрутизации, рендеринга уникального контента, ссылки на все страницы, имеющиеся в проекте.
- **public/services/** — вспомогательные js-файлы для аутентификации и взаимодействия с базой данных.
- **public/views/pages/** — файлы макетов страниц. Так как приложение использует модель SPA (Single Page Application), рендеринг реализован непосредственно в js-коде.
- **public/views/components/** — файлы отдельных компонентов страниц, содержат код для отображения в браузере и описание взаимодействия с пользователем.
- **public/styles/** — css-файлы, описывающие внешний вид страниц приложения.
- **public/images/** — файлы изображений.

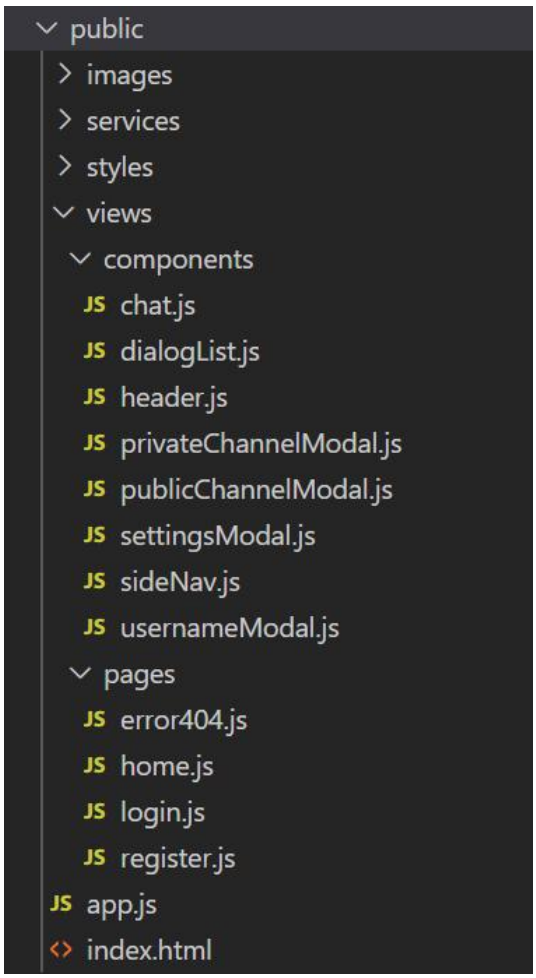

Рисунок 1.6. - Структура проекта.

2. РЕАЛИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ БАЗЫ ДАННЫХ В ПРИЛОЖЕНИИ

2.1. Firebase Realtime Database

База данных Firebase Realtime — это база данных, размещённая в облаке. Данные хранятся в формате JSON и синхронизируются в реальном времени с каждым подключённым клиентом.

При использовании хостинга Firebase можно настроить приложение для динамической загрузки библиотек Firebase JavaScript SDK с зарезервированных URL-адресов. Подключение всех необходимых сервисов, в том числе Realtime Database, и инициализация Firebase происходит в файле «index.html» (см. рис. 2.1):

```
<!-- update the version number as needed -->
<script defer src="/__/firebase/7.20.0/firebase-app.js"></script>
<!-- include only the Firebase features as you need -->
<script defer src="/__/firebase/7.20.0/firebase-auth.js"></script>
<script defer src="/__/firebase/7.20.0/firebase-database.js"></script>
<script defer src="/__/firebase/7.20.0/firebase-messaging.js"></script>
<script defer src="/__/firebase/7.20.0/firebase-storage.js"></script>
<!-- initialize the SDK after all desired features are loaded -->
<script defer src="/__/firebase/init.js"></script>

<script type="module" src="app.js"></script>
```

Рисунок 2.1. - Подключение Realtime Database и инициализация Firebase.

2.2. Авторизация/Регистрация

Сперва рассмотрим систему авторизации и регистрации. Код находится в файлах auth.js. Страница авторизации выглядит следующим образом (см. рис. 2.2):

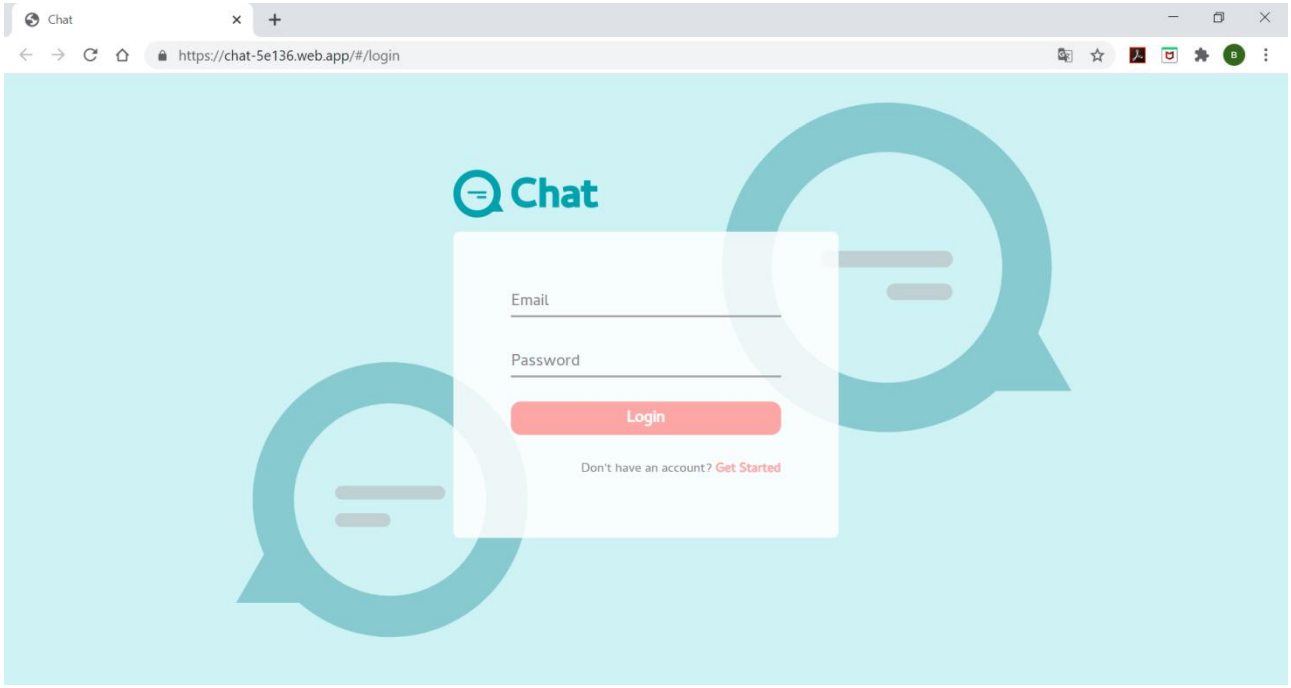

Рисунок 2.2. - Страница авторизации.

При нажатии на кнопку «Login» срабатывает обработчик события формы, который проверяет данные на корректность и выполняет авторизацию пользователя, если поля прошли валидацию (см. рис. 2.3):

```
afterRender : async () => {
    const loginForm = document.forms.login;
    const error = document.querySelector(".error");

    loginForm.addEventListener("submit", (e) => {
        e.preventDefault();

        const email = loginForm.email.value;
        const password = loginForm.password.value;

        Auth.login(email, password).catch(err => {
            console.error(err.code);
            console.error(err.message);

            if (err.code === "auth/wrong-password" ||
                err.code === "auth/user-not-found") {
                error.innerHTML = "Wrong email or password";
                error.style.display = "block";
            } else {
                alert("An error occured");
            }
        });
    });
}
```

Рисунок 2.3. - Код обработчика события отправки формы авторизации.

За добавление нового пользователя в базу данных отвечает метод signInWithEmailAndPassword объекта auth. В качестве параметров он принимает два значения: логин пользователя и его пароль. Метод автоматически сверяет введённые значения с имеющимися в базе данных — эта особенность в какой-то степени есть удобство встроенного сервиса Firebase Authentication, хотя имеется и обратная сторона монеты: метод уже прописан и мы не знаем, что конкретно творится за кулисами, поэтому могут возникнуть некоторые сложности при попытке внести какие-либо изменения в его код.

Страница регистрации выглядит следующим образом (см. рис. 2.4):

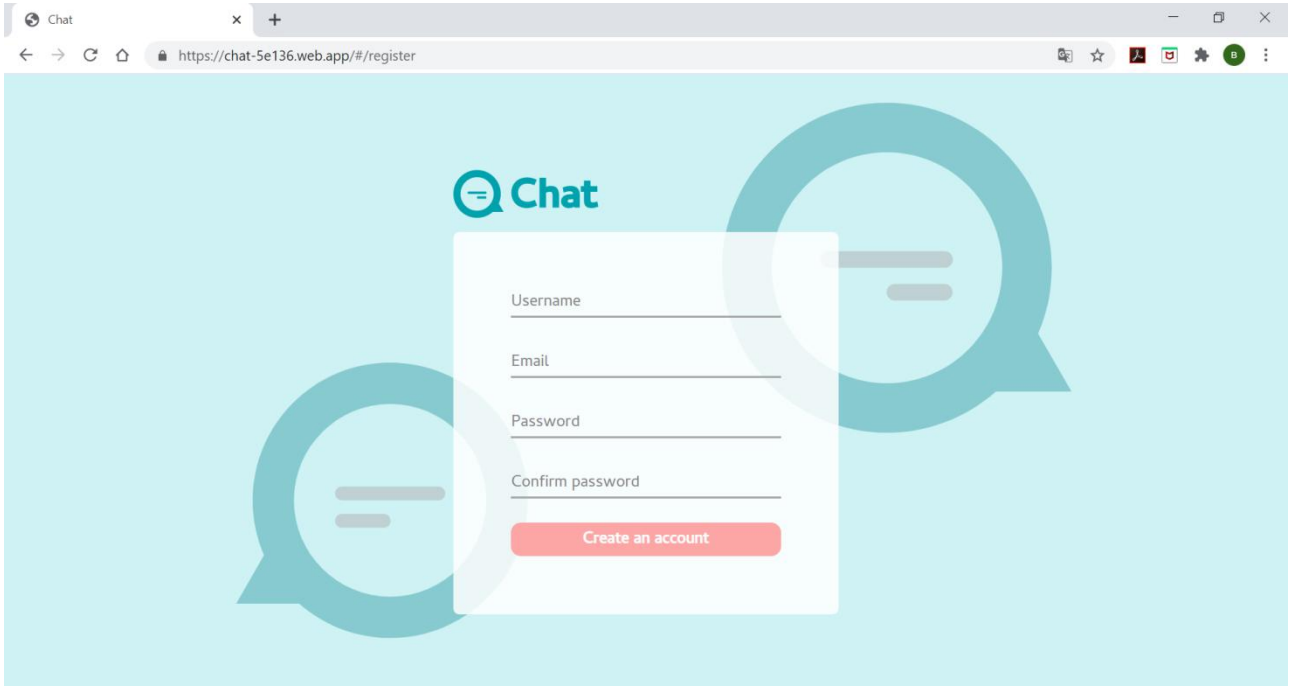

Рисунок 2.4. - Страница регистрации.

При нажатии на кнопку «Create an account» срабатывает обработчик события формы, который проверяет данные на корректность и добавляет нового пользователя в базу данных, если поля прошли валидацию (см. рис. 2.5):

```
afterRender : async () => {
    const registrationForm = document.forms.register;

    registrationForm.addEventListener("submit", (e) => {
        e.preventDefault();

        const username = registrationForm.username.value;
        const email = registrationForm.email.value;
        const password = registrationForm.password.value;
        const passwordConfirm = registrationForm.password_confirm.value;

        const error = document.querySelector(".error");

        if (password === passwordConfirm) {
            Auth.register(username, email, password).catch(err => {
                if (err.code === "auth/email-already-in-use") {
                    error.innerHTML = "Email already in use";
                    error.style.display = "block";
                } else {
                    alert("An error occured");
                }
            });
        } else {
            error.innerHTML = "Passwords do not match";
            error.style.display = "block";
        }
    });
}
```

Рисунок 2.5. - Код обработчика события отправки формы регистрации.

Важным же отличием от предыдущего метода является наличие вспомогательного метода addUser() (см. рис. 2.6). Появление этого кода связано с наличием второй проблемы сервиса Firebase Authentication: отсутствие возможности «кастомизировать» поля базы данных пользователей. Есть несколько основных полей, таких как логин (в моем случае это email), пароль, id, дата создания и дата последнего входа. Также в качестве необязательных параметров может служить номер телефона, страна, почта и другое.

```
async addUser(uid, username, email) {
    await firebase.database().ref(`users/${uid}`).set({
        displayName: username,
        photoURL: "../images/Dark_Blue_Moon_Emoji_grande.png",
        email: email
    }, function (err) {
        if (err) {
            console.error(err.message);
        }
    }
)},
```

Рисунок 2.6. - Метод addUser().

Собственно, теперь мы и подошли к необходимости метода addUser: так как мы не можем добавить новые поля в Authentication, мы должны добавить их в нашу личную базу данных, именно поэтому при помощи объекта db я создаю новое поле в таблице пользователей (ref('users/') – метод получения таблицы по названию и новому полю) и с помощью метода update добавляю нового пользователя в базу данных для дальнейшей работы с ним. Значения таблицы пользователей выглядит следующим образом (см. рис. 2.7):

Рисунок 2.7. - Сущность пользователя в базе данных.

Вот так выглядит сущность пользователя, который впервые зашёл на сайт и зарегистрировался (см. рис. 2.8):

Рисунок 2.8. - Сущность пользователя, который только что зарегистрировался

В Firebase Console также есть возможность просмотреть таблицу имеющихся пользователей (см. рис. 2.9):

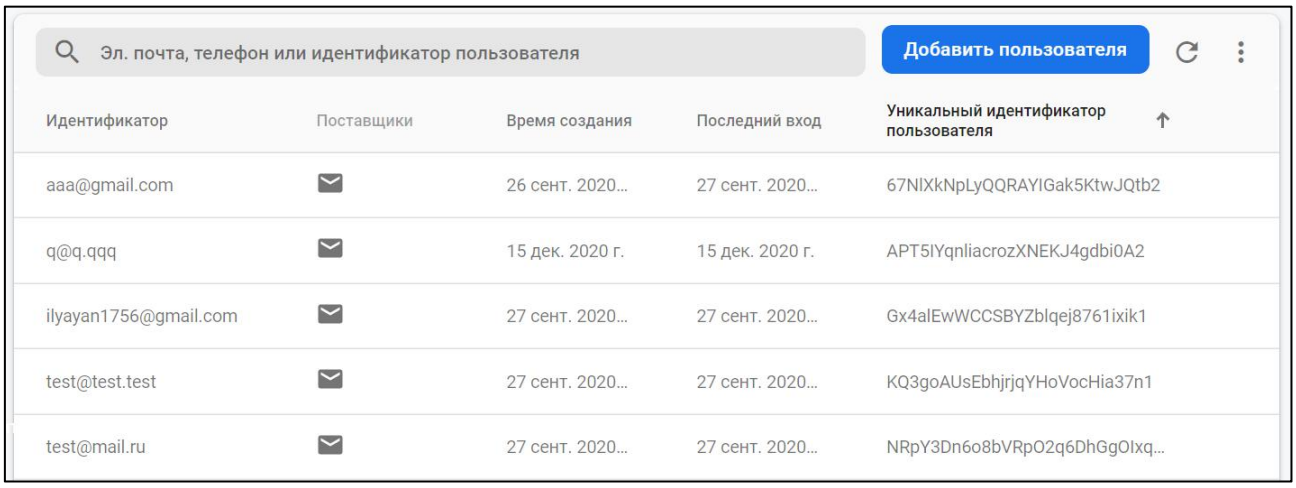

Рисунок 2.9. - Таблица пользователей в Firebase Authentication в Firebase Console.

2.3. Поиск пользователей и каналов

Все методы взаимодействия с базой данных (добавление чатов, создание каналов, отправка сообщений, получение данных для отображения на странице) находятся в объекте db. Рассмотрим некоторые из них.

У новых пользователей список диалогов пуст, для того, чтобы начать общение, необходимо найти нужного пользователя или канал в меню поиска (см. рис. 2.10):

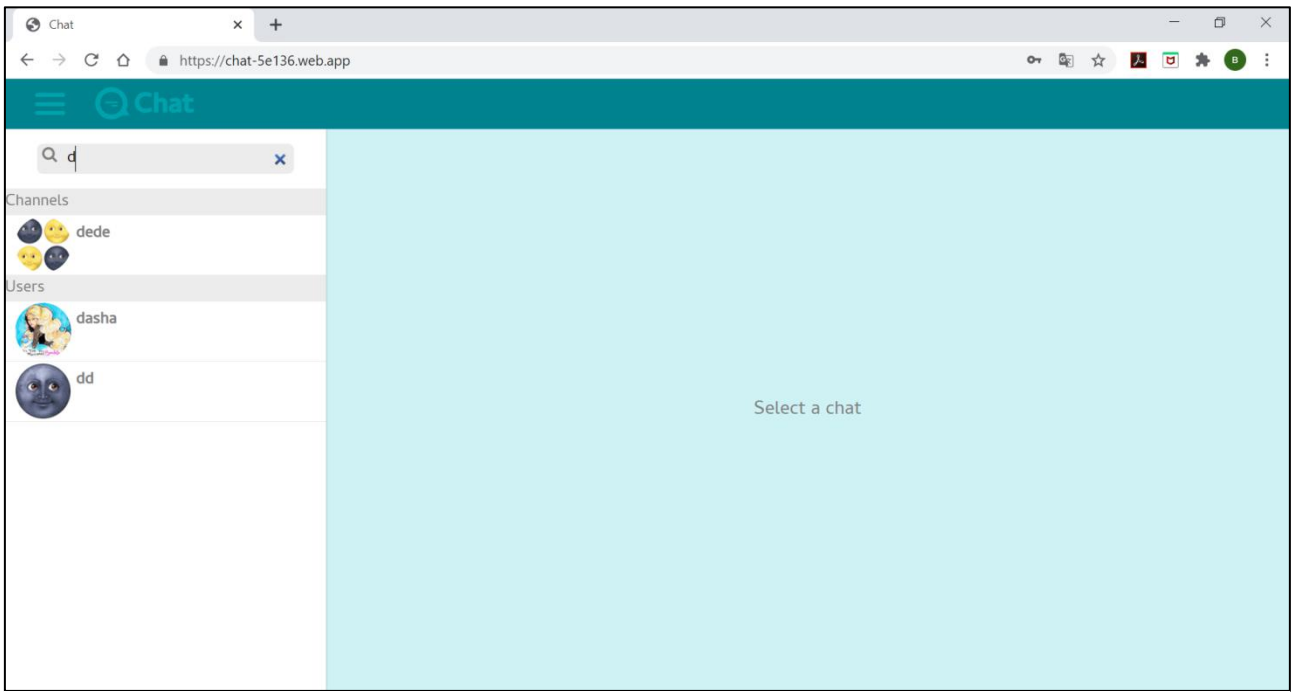

Рисунок 2.10. - Поиск пользователей и каналов.

Метод объекта db, осуществляющий поиск пользователей (аналогичный метод используется для поиска каналов) (см. рис. 2.11):

```
async getUsersByQuery(query) {
    let users = [];

    await firebase.database().ref("users").orderByChild("displayName")
    .once("value").then(async (snapshot) => {
        const snap = snapshot.val();

        for (let uid in snap) {
            if (snap[uid].displayName.startsWith(query) &&
            !(await this.isUserInCurrentUserChats(uid)) &&
            uid !== Auth.currentUserID()) {

                const user = {
                    uid: uid,
                    displayName: snap[uid].displayName,
                    photoURL: snap[uid].photoURL
                };

                users.push(user);
            }
        }
    });

    return users;
},
```

Рисунок 2.11. - Метод поиска пользователей.

2.4. Отправка и чтение сообщений

При отправке первого сообщения пользователю (см. рис. 2.12) в базу данных добавляется запись о создании чата между двумя пользователями (см. рис. 2.13). Там содержатся поля «members», содержащие идентификаторы пользователей-участников чата и идентификаторы последнего прочитанного сообщения для каждого пользователя, и «timestamp» - временная метка последнего сообщения в чате. Также в сущность каждого пользователя добавляется запись с идентификатором чата (см. рис. 2.14).

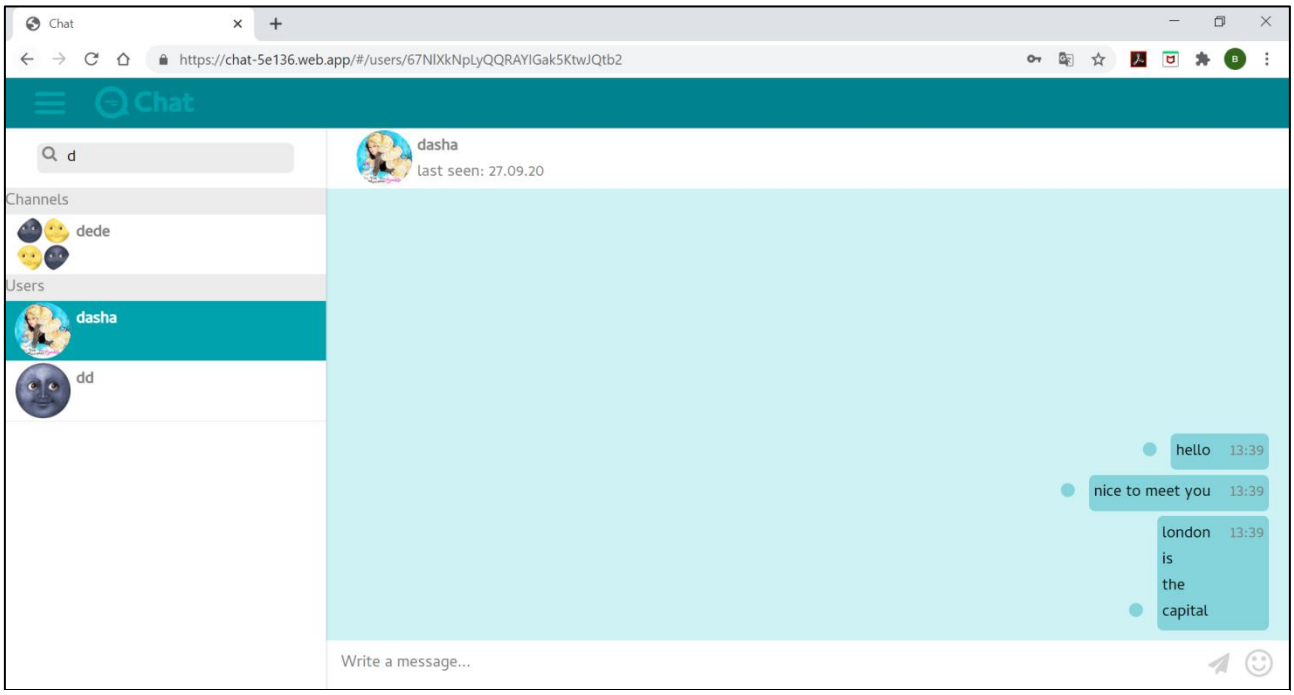

Рисунок 2.12. - Написание сообщений.

Рисунок 2.13. - Сущность «чат» в базе данных.

```
async addChat(memberA, memberB) {
    const chatID = Utils.getTwoUsersChatID(memberA, memberB);
    let updates = {};

    updates[`chats/${chatID}/members/${memberA}`] = "";
    updates[`chats/${chatID}/members/${memberB}`] = "";
    updates[`users/${memberA}/chats/${chatID}`] = true
    updates[`users/${memberB}/chats/${chatID}`] = true

    await firebase.database().ref().update(updates);
},
```

Рисунок 2.14. - Метод для добавления диалога между пользователями в базу данных.

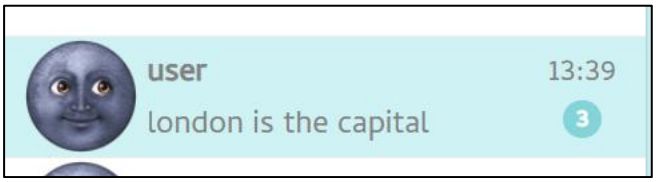

Рисунок 2.15. - Отображение непрочитанных сообщений в списке диалогов пользователя.

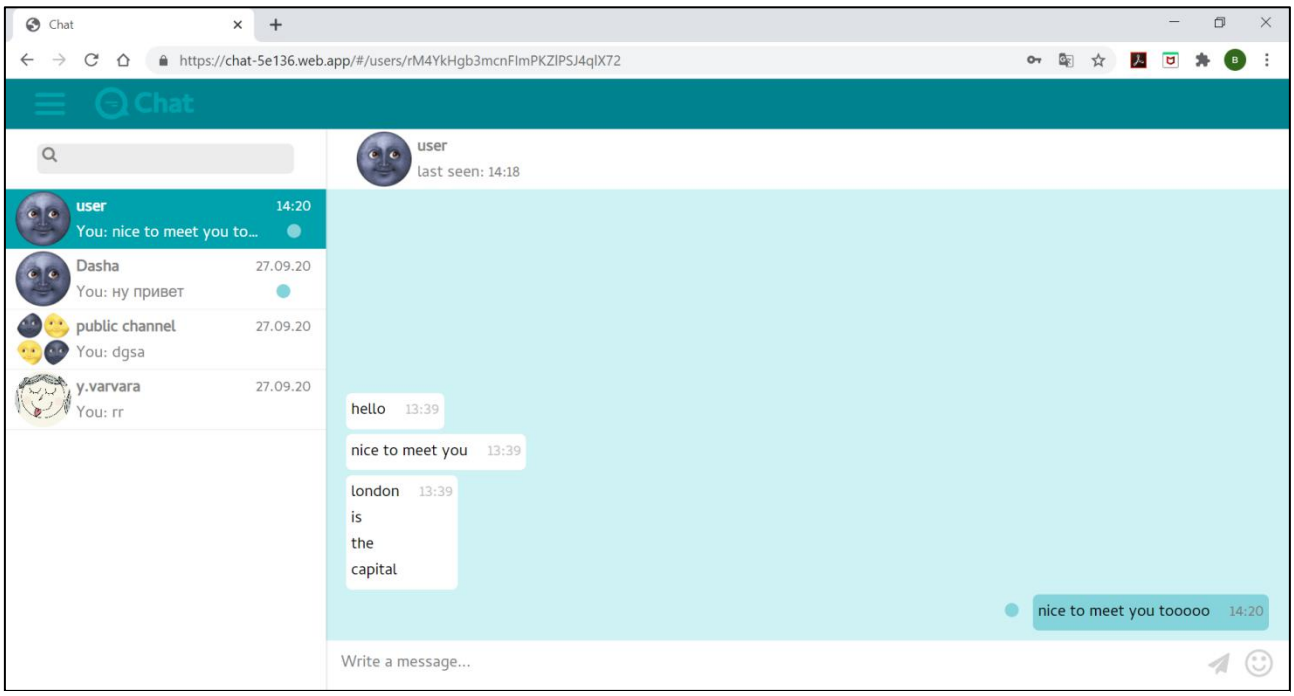

Рисунок 2.16. - Чтение сообщений.

Для чтения данных по определённому пути и прослушивания изменений используется метод on() firebase.database.

```
async addChatMessagesChildAddedListener(chatID, callback) {
    firebase.database().ref(`messages/${chatID}`).orderByChild("timestamp").
    startAt(Date.now()).on("child_added", callback);
},
```

Рисунок 2.17. - Прослушивание изменений в сущности «сообщения».

2.5. Создание каналов

В приложении есть возможность создавать канаты для общения многих пользователей. Канал может быть публичным, то есть стать его участником сможет любой пользователь, либо приватным, в таком случае для просмотра и отправки сообщений необходимо будет ввести пароль (заданный пользователем, создавшим канал).

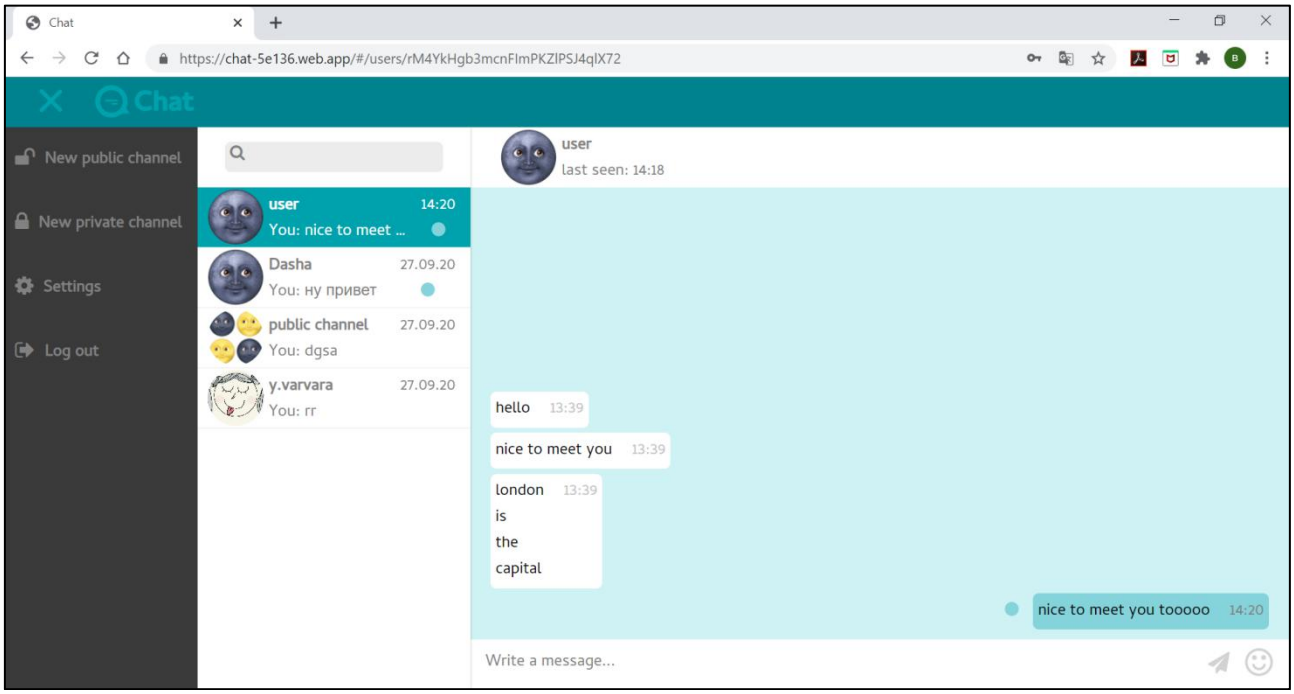

Рисунок 2.17. - Боковое меню с кнопками для добавления каналов.

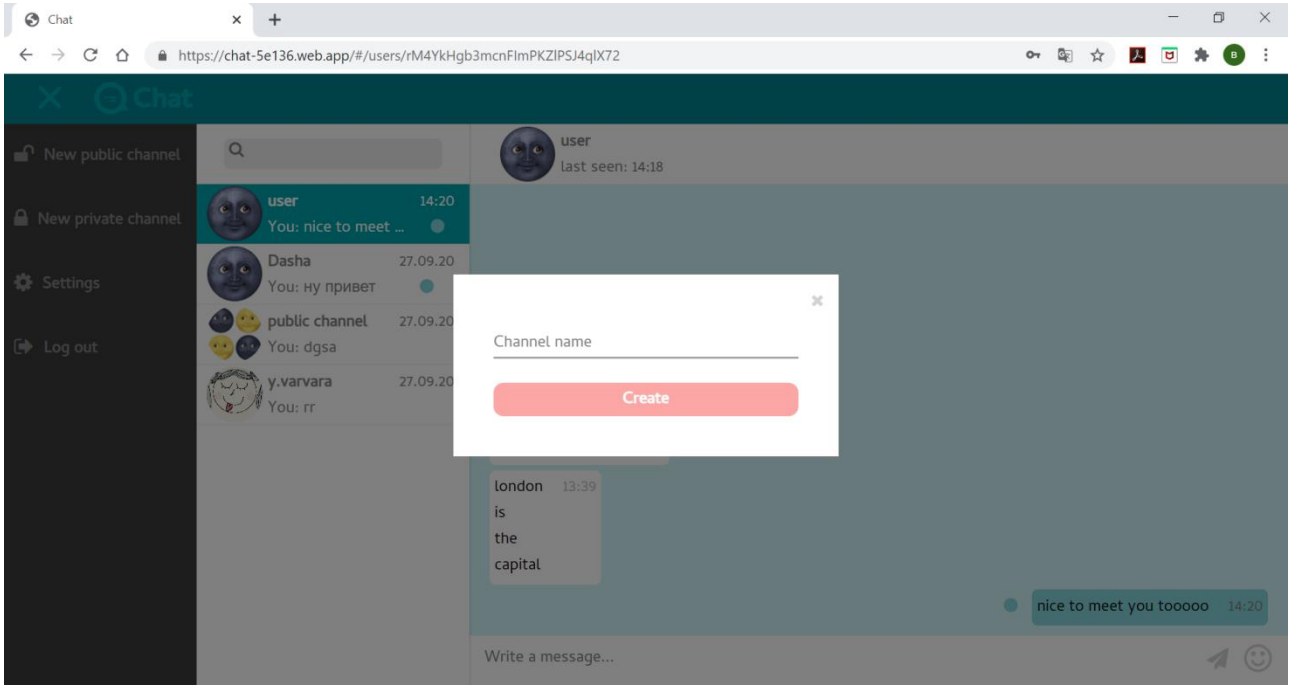

Рисунок 2.18. - Форма для создания публичного канала.

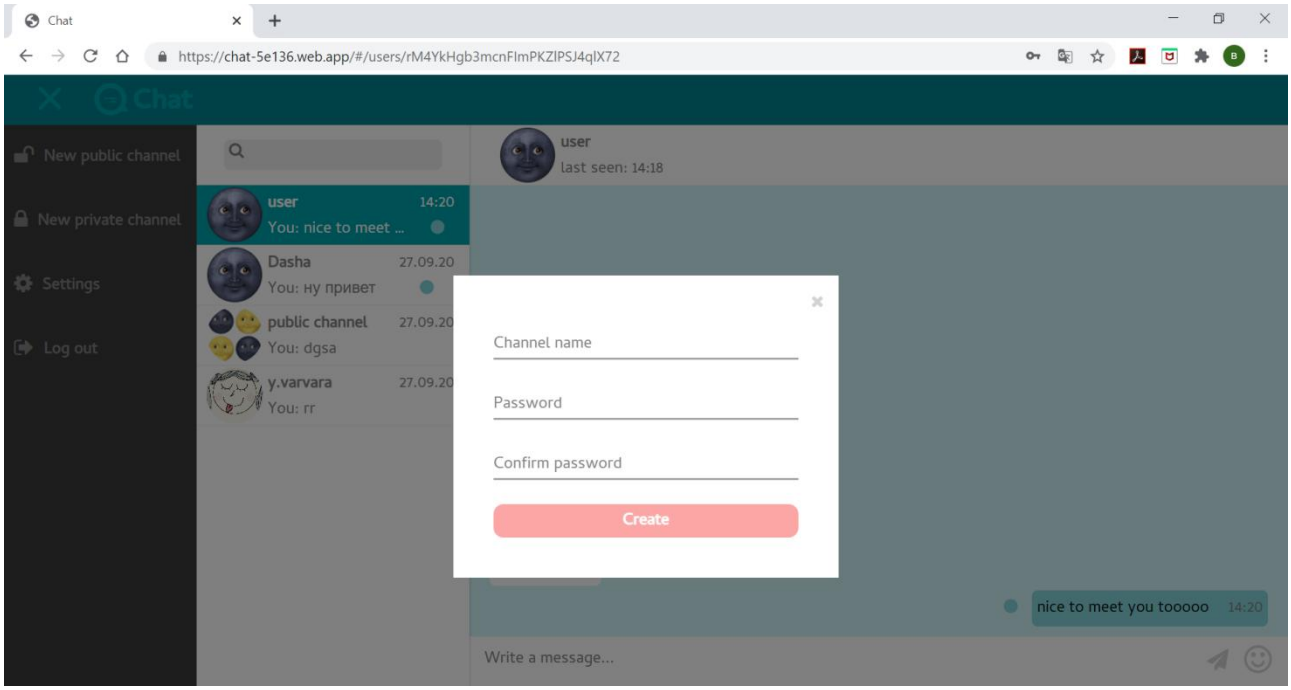

Рисунок 2.19. - Форма для создания приватного канала.

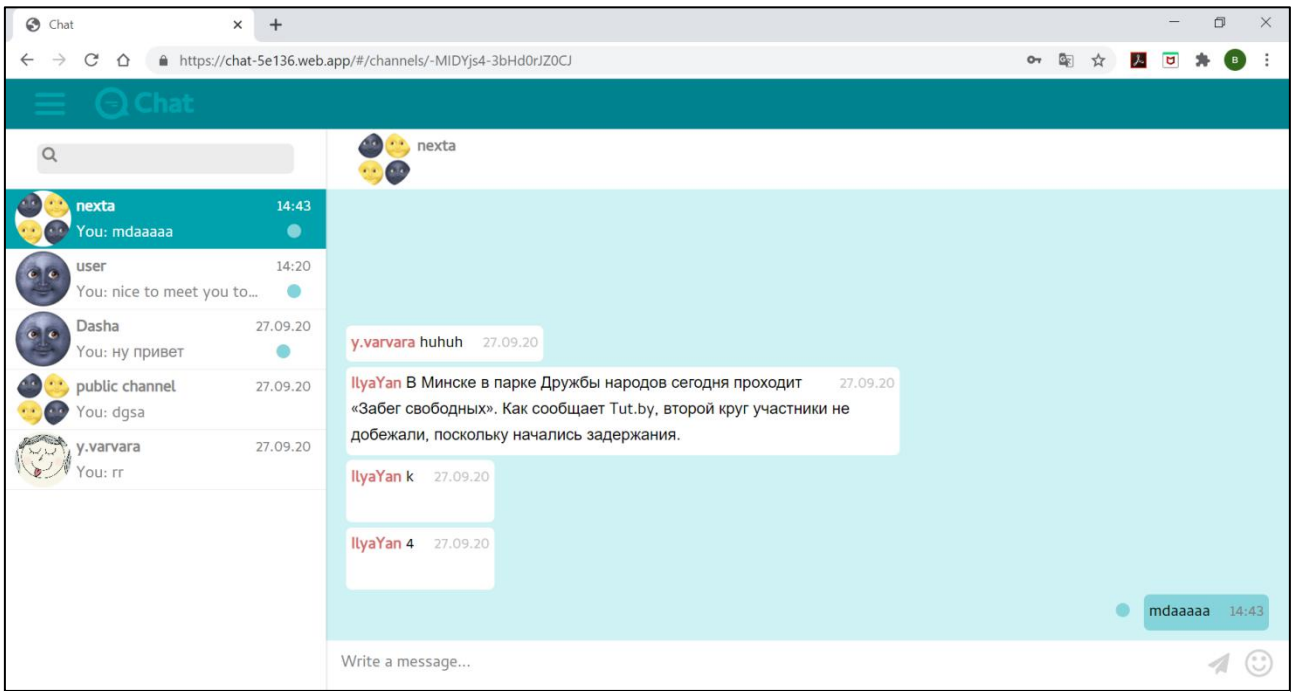

Рисунок 2.20. - Отображение сообщений в публичном канале.

```
async addChannel(name, password, members) {
    const photoURL = "../images/group_avatar.png";
    let channel = {name, password, photoURL};

    const channelID = await firebase.database().ref(`chats`).push(channel).key;

    if (members) {
        for (let member of members) {
            let updates = {};

            updates[`chats/${channelID}/members/${member}`] = "";
            updates[`users/${member}/chats/${channelID}`] = true

            await firebase.database().ref().update(updates);
        }
    }
},
```

Рисунок 2.21. - Метод добавления канала в базу данных.

2.6. Изменение данных пользователя

Пользователю доступны действия по изменению имени пользователя и аватара. Для этого необходимо открыть боковое меню и перейти на форму настроек.

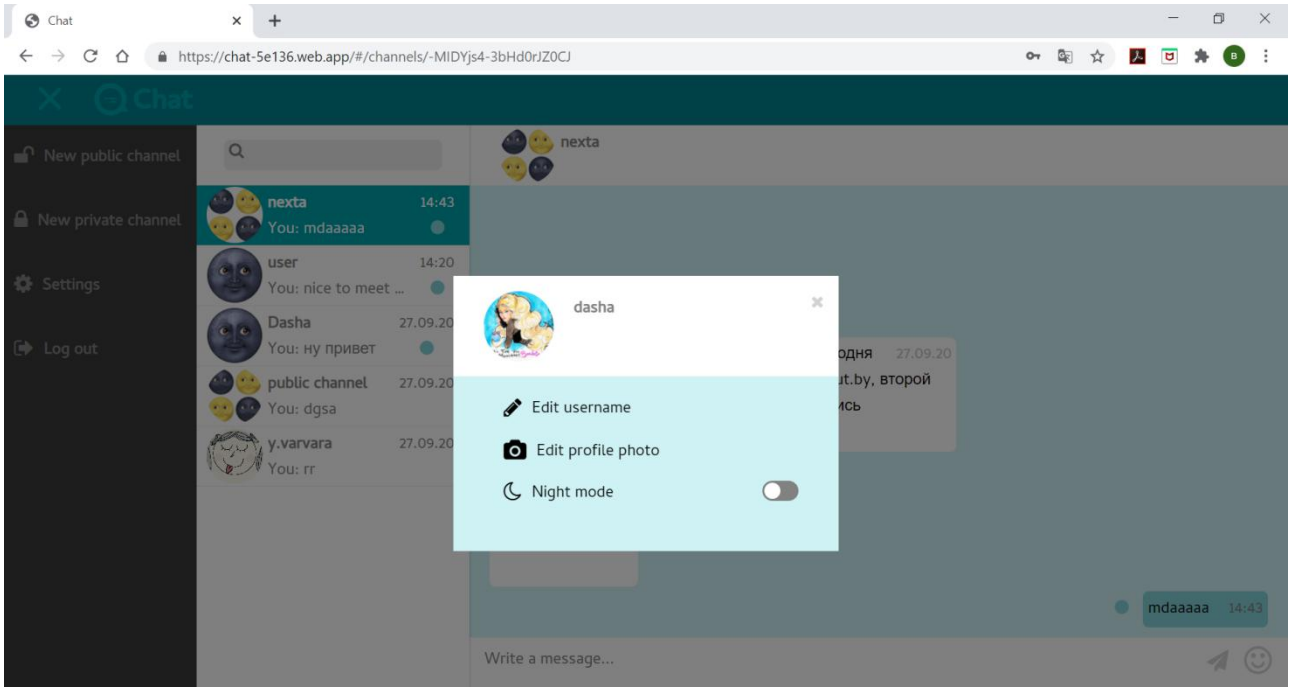

Рисунок 2.22. - Форма настроек.

Поменяем аватар пользователя. При нажатии на кнопку «Edit profile photo» открывается окно выбора файла.

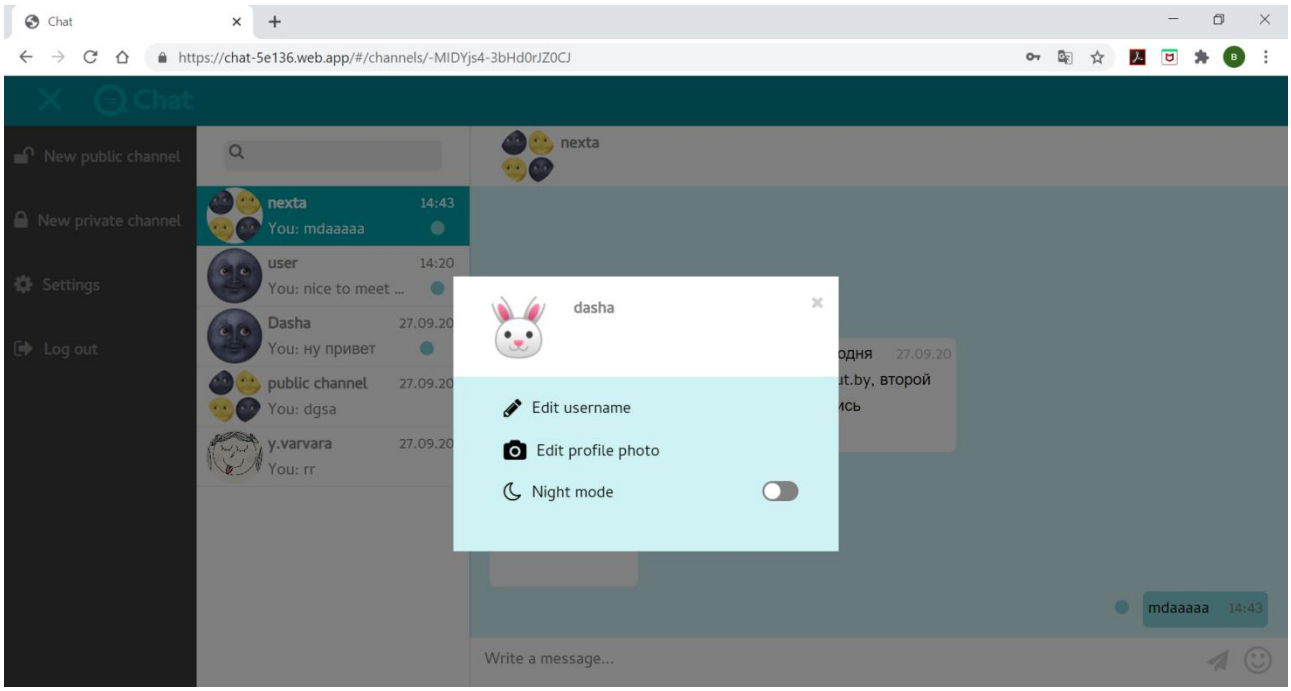

Рисунок 2.23. - Изменение аватара пользователя.

Теперь при рендеринге страницы обновится аватар пользователя. А в хранилище Firebase появится новый файл с именем, равным идентификатору пользователя:

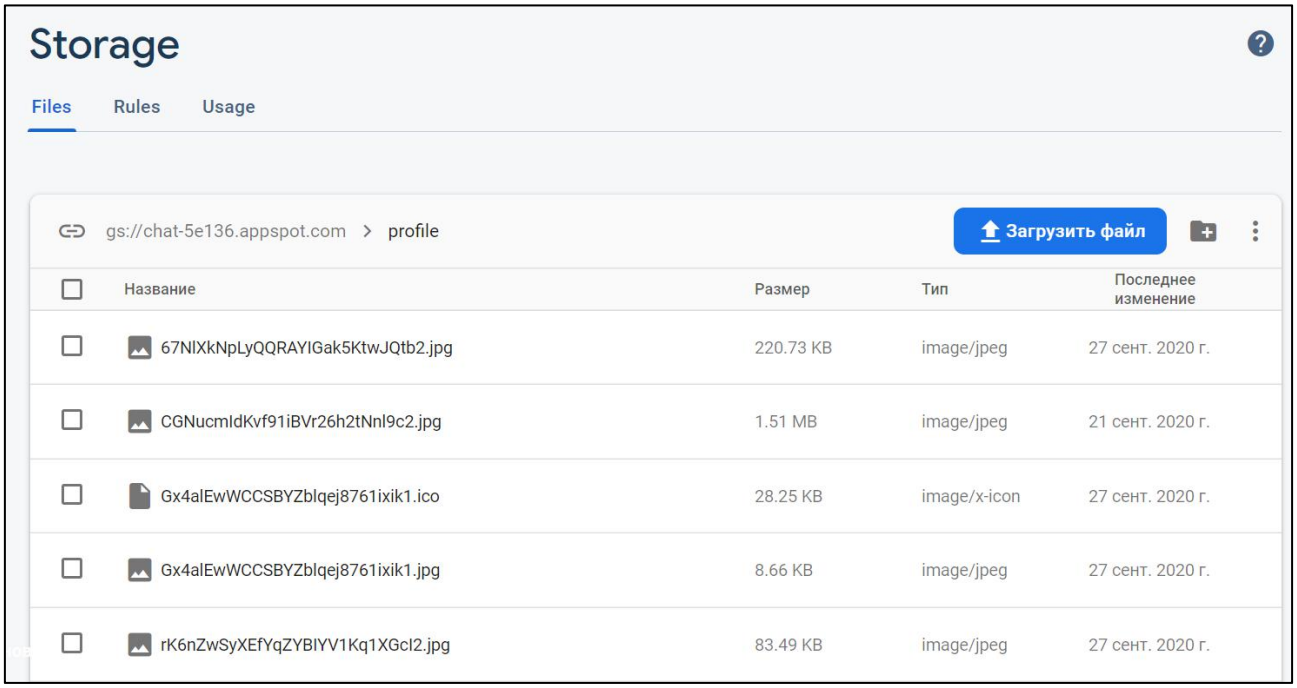

Рисунок 2.24. - Хранилище загруженных картинок.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была продемонстрирована работа базы данных NoSQL на примере создания одностраничного веб-приложения «Чат». Данное приложение представляет собой площадку, на которой пользователи могут обмениваться сообщениями. В ходе работы с базой данных NoSQL были выявлены её следующие преимущества:

- Возможность хранения больших объёмов неструктурированной информации. В NoSQL нет ограничений на типы хранимых данных, а при необходимости можно добавлять новые типы данных.
- Быстрая разработка. NoSQL базы данных не требуют большого объёма подготовительных действий, который нужен для реляционных баз.

Также присутствуют и слабые стороны:

- NoSQL решения не требуют определять схему базы данных перед началом работы, поэтому в процессе разработки можно наткнуться на непредвиденные трудности, которые могут привести к отказу от данного NoSQL решения.
- Невозможность выполнения сложных запросов.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Что такое база данных [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://www.oracle.com/ru/database/what-is-database/);
2. Система управления базами данных[Электронный ресурс]. - Электронные данные. - Режим доступа: (https://ru.wikipedia.org/wiki/ Система_управления_базами_данных);
3. What is a Relational Database? [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://aws.amazon.com/relational-database/);
4. Non-relation data and NoSQL [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://docs.microsoft.com/azure/architecture/data-guide/big-data/non-relational-data);
5. Visual Studio Code [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://ru.wikipedia.org/wiki/Visual_Studio_Code);
6. HTML5 [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://ru.wikipedia.org/wiki/HTML5);
7. CSS [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://ru.wikipedia.org/wiki/CSS);
8. JavaScript [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://ru.wikipedia.org/wiki/JavaScript);
9. Firebase [Электронный ресурс]. - Электронные данные. - Режим доступа: (https://ru.wikipedia.org/wiki/Firebase);

Приложение 1. Исходные файлы

1. Ссылка на исходный код программы:

   https://github.com/yvarvara/chat-application