

**Internship Report:**  
**Mern Stack: Chatting App in React and NOSQL**

Project report submitted in partial fulfillment of the requirement for the degree  
of Bachelor of Technology

in

**Computer Science and Engineering/Information Technology**

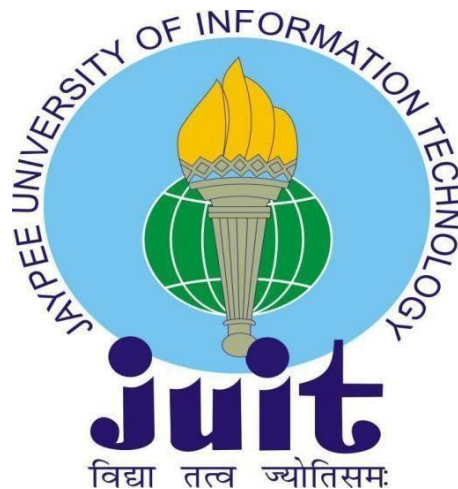
By

**Ojaswi Awasthi (181391)**

Under the supervision of

**Deepak Gupta**

To



Department of Computer Science & Engineering and Information Technology  
**Jaypee University of Information Technology Waknaghat, Solan-173234,**

**Himachal Pradesh**

---

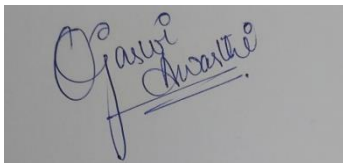
# CERTIFICATE

## Candidate's Declaration

I hereby declare that the work presented in this report entitled “**MERN Stack: Chatting Application**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from February 2022 to July 2022 under the supervision of **Dr. Deepak Gupta**

(Assistant professor Senior grade Computer Science Engineering).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

A rectangular box containing a handwritten signature in blue ink. The signature reads "Ojaswi Awasthi" with a horizontal line underneath.

Ojaswi Awasthi-181391

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Deepak Gupta

Assistant Professor (SG)

Computer Science Engineering Dated:

## **ACKNOWLEDGEMENT**

To begin with, I want to express my heartfelt gratitude to almighty God for His divine blessing, which has enabled us to successfully complete the project work.

Supervisor Dr. Deepak Gupta, Assistant Professor (SG), Department of CSE Jaypee University of Information Technology, Waknaghat, deserves my deepest gratitude. My supervisor's deep knowledge and keen interest in the field of "Data Science and Machine Learning" made it possible to complete this project, as well as his relentless patience, scholarly guidance, persistent and enthusiastic supervision, constructive criticism, insightful advice, and reading many inferior draughts and correcting them at all stages.

I'd like to express my heartfelt gratitude to Dr. Deepak Gupta of the Department of CSE for his invaluable assistance in completing my thesis.

I'd also like to express my gratitude to everyone who has assisted me in making this project a success, whether directly or indirectly. In this specific circumstance, I'd like to express my gratitude to the numerous staff members, both teaching and non-teaching, who have provided me with valuable assistance and facilitated my project.

Finally, I must express my gratitude for my parents' unwavering and unflagging love, support and utmost patience in putting up with us.

**Ojaswi Awasthi(181391)**

# TABLE OF CONTENTS

<b>Chapter Name</b>	<b>Page No.</b>
Abstract	vi
<b>Chapter1: Introduction</b>	
1.1 Introduction	1
1.2 Problem Statement	3
1.3 Objectives	4
1.4 Methodology	5
1.5 Organization	6
<b>Chapter 2: Literature Survey</b>	
2.1 Introduction to React Framework	10
2.2 Characteristics of React Javascript	11
2.3 Proposed Algorithms	12
2.4 Introduction to Chatting Platforms	12
2.5 Project Contributions	13
2.6 Overview of the Socket	16
<b>Chapter 3: System Development</b>	
3.1 Android as an Operating System	18
3.2 Javascript Framework- React Javascript	20
3.3 Basic Rendering of the Data	22
3.4 Benefit from Native Environment	22
3.5 Node Package Manager	23
3.6 Creating a React Javascript Project	24
3.7 Google Firebase	28
3.8 API Terminology	33

3.9 Core components of React Javascript	35
3.10 Context API	37
<b>Chapter 4: Performance Analysis</b>	
4.1 User's Flowchart	41
4.2 SignIn Dashboard	43
4.3 React Suite Modal for creating a new chat room	46
4.4 Room Description Modal	47
4.5 Edit Room Modal	48
4.6 Chat Types	49
4.7 React Hooks	54
<b>Chapter 5: Conclusions</b>	
5.1 Conclusions	58
5.2 Future Scope	59
5.3 Applications	59
<b>References</b>	60
<b>Appendices</b>	61

## LIST OF FIGURES

**Figure 1:** Sign In Screen

**Figure 2:** Firebase chat and profile folders

**Figure 3:** Initialization of chat-web-app

**Figure 4:** The Visual Code File Structure

**Figure 5:** Rules for Firebase Database

**Figure 6:** The billable metrics for Firebase App

**Figure 7:** The breaking up of Android features

**Figure 8:** The break-down of App JS file into header and Body

**Figure 9:** Debugger for React Javascript

**Figure 10:** Try-catch asynchronous code block

**Figure 11:** Distinction between Sass and Scss Files

**Figure 12:** Profile Context API

**Figure 13:** Figure showing User's Flowchart

**Figure 14:** React Suite Modal

**Figure 15:** Room Description Modal

**Figure 16:** Edit Room Modal

**Figure 17:** Organization of chats according to date and time

**Figure 18:** Figure depicting Firebase Success and Error

**Figure 19:** Firebase Screenshot showing users logged in the app

**Figure 20:** Github repository for the React application

## Abstract

This project is a full stack web development project that aims at building a chatting application using the Frontend Technologies React (a framework built on top of the ever popular and evolving programming language Javascript that has been ruling the IT industry over 25 years) and many of the npm dependencies such as the React Fade –In and the **rsuite** React library for the User Interface Part. The rsuite has several built-in components in React that can simply be imported in the React component file using the direct import statement after the dependency has been installed on the system. The rsuite library will build a powerful UI with several stunning eye-catching features. The React framework is a very popular framework on JS that is used by the leading companies across different niche industries such as Instagram, Spotify, Netflix , Airbnb and many such platforms.

This app is not just a chatting application where the users can chat anonymously but are only required to become the users after logging in. The users are required to log in the application using either their gmail or facebook profile username and password. This functionality is provided to the users using the popular NOSQL database platform from Google called Google Firebase. The **GoogleAuthProvider()** and **FacebookAuthProvider()** functions provide the authentication and login functionalities. This app uses other other advanced react functions such as custom hooks and Context API. The user can send upto five photos and files too. And of course, there is no limit on the number of chats sent on the platform. The chats and files are also organized by date and time like they are in professional platforms such as Whatsapp and Telegram. Also, there is the added functionality of liking messages that the user wants. The users can create chatrooms too and only the admin of the group has been provided the edit functionality to edit the name of the group or the chatroom exclusively. Also, every logged in user can view his/her profile from the dashboard section in the left drawer on the app. The dashboard shows the name of the user, their image and the edit function for their profile. The user can upload their profile image from the device that they are using the app on. Also, the user is provided the Sign Out option in the dashboard to unsubscribe from the database.

Firebase serves as the database backend in the app that stores every detail of the user including their login credentials, the chats sent and the messages liked as in a real database.

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction

The popularity of the MERN stack is ever increasing in the field of web development for frontend and backend applications. The use of React stems from the fact that it was built on Javascript and JS has been the most popular programming language amongst developers for over 25 years. React was built on top of JS by developers from Facebook and made making the UI so much simpler than the normal HTML. It made rendering the components so much easier and faster. For instance, if we had to render a component time and again, we had to hard code it as many times as we needed it. However, in React, we only have to make a component just once and then import it in whichever component we need it.

MERN stack uses the **Express JS** and **Node JS** for developing the backend. Earlier, the entire backend client and server part used to be handled by solely Node. However, sending requests and getting back responses from the server was so much more difficult, tedious, painstaking and more complicated. Then, Express was introduced as the de facto application server for building backend API. The server and client applications were made so much easier as there was no need to build the server but it was introduced by Express itself. Using Express has become as easy as importing it in a backend file and creating its instance using the `const app = require('express')` and then listening it on a given port number just like the backend. The syntax becomes much more simpler with Express than using simple Node. Also, the **react-router-dom** provides us the Router functionality that helps us define routes in the frontend application without creating a backend directory for the routing part. The router consists of the routing endpoint and the handler function for the request handling portion.

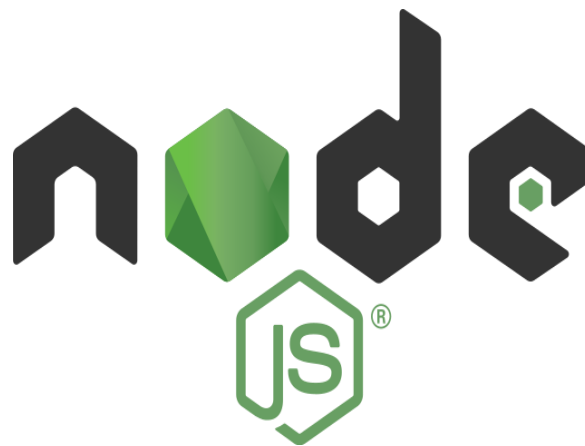
Firebase is a NOSQL backend database technology built by Google to handle the database part. NOSQL is a database technology that stores data without the storing it in the usual format of rows and columns in a table. We are exempt from the usual task of creating a table and then defining its constraints. The NOSQL will store data in the form of documents that contain clusters and collections. They contain data with or without a fixed syntax. That means, if a document contains an array field, the other might not contain it.



The fourth pillar of the MERN stack is the Node JS framework that has been discussed before with the Express framework. Node JS revolutionized the running of Javascript outside of a web browser. Earlier, it could only be run on the web browser and not in the local run-time environment. Now, it has become easier to run Javascript in the local terminal. Since, it is not a multi-threaded language, Javascript has the concept of promises and callbacks to run code asynchronously and Node Js allows us to do that. The async await syntax allows us to run code asynchronously and has been an improvement over the normal promises and callbacks. Also, async await code makes the code look as though it was synchronous.

This is because it leads to a very common problem in Javascript called the callback hell. This problem makes the asynchronous code look really complicated and for someone who has been using a synchronous multi-threaded language, it can get really difficult to decode and debug it in case of errors. Hence, we use the async-await functional syntax for fetching data from the API and handling it in React templates.

React hooks and custom hooks have been used to provide code reusability in the application over the normal usual class programming. Functional components have been built and are preferred to using class components in this application. Class components are still used but the easy syntax of functional components in React make them easier to be understood and easily applicable in the React apps.



Node JS has completely changed the way Javascript used to run in the browser. The browser was earlier used to run Javascript but with the advent of Node JS, the run-time environment has changed completely with Javascript running on the local run-time powershell or git bash terminal.

## 1.2 Problem Statement

- The internship project focuses on building a chatting application in MERN stack using the frontend Javascript technology called React and Express and Node JS for the backend for building of client and server part. Google Firebase is a free backend service that stores the login credentials of the user and the chatroom information.
- This app also gives primary focus to the admin and security permissions to the users. Only, the admin of the chatroom has the permission to edit the name of the chatroom and the chatroom description.
- The project aims to be more than just a chatting application as it allows us to send more than just chats in the server, We can upload upto 5 files in one go. The Message File has the attachment icon to attach files and the send button to finally send the files over on the server.
- The user can login the app using the authenticator provided by the Google Firebase. Also, when the user signs out from the app, the unsubscribe() function allows him to be unsubscribed from the database, The timestamp is also added which keeps a record of when the user logged in to the app.
- The user can like certain chats in the chat window and can also remove those likes. A heart emoticon will show up when the user likes the message. This is made possible through the concept of post transactions in React.
- The chats in the application are then organized by date and time when they were sent on the server. After the chats have occupied a certain space on the window screen, the **Load More** button will load the existing chats on the page that were hidden below the most recent chats in the browser.

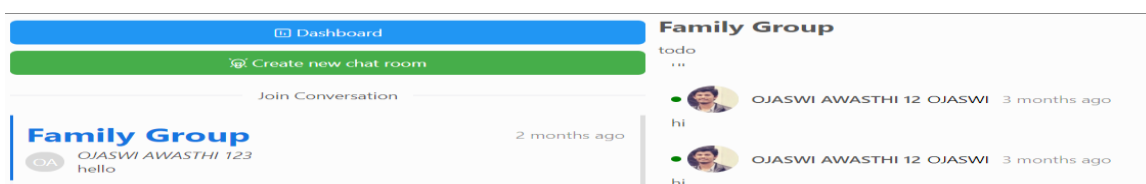


Figure showing the app with the dashboard screen after logging in of the user

### 1.3 Objectives

The most primary objective of this project is to understand the concept of building applications with the MERN stack (MongoDB, Express, React and Node). In this particular project, the focus has been built on a chatting application.

The internship project focuses on developing a talking application in the MERN stack using the frontend Javascript technology known as React and Express, as well as Node JS for the backend. Google Firebase is a free backend service that keeps the user's login credentials as well as chatroom data.

The admin and security permissions for users are also prioritized in this app. Only the chatroom's administrator has the ability to change the name and description of the chatroom.

The project aspires to be more than just a talking app by allowing us to submit more than just chats to the server; we can upload up to 5 files at once. In the chat box, the user can choose to like or unlike certain chats. When the user likes the message, a heart emoticon appears. This is feasible because to React's concept of post transactions.

The user can log in to the app using the Google Firebase authenticator. The unsubscribe() function also allows the user to be unsubscribed from the database when he logs out of the app. A timestamp is also provided to keep track of when the user registered in to the app.

The attachment icon is used to attach files to the message file, and the send button is used to transfer the files to the server.

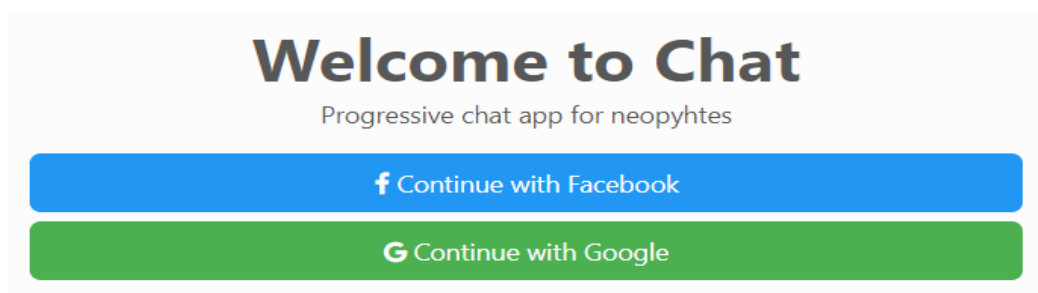


Fig 1: The app prompts the user to sign in using either the gmail or the facebook id.

## 1.4 Methodology

First and foremost, we must set up our react javascript project. Then, using some specified processes, connect our project to Google Firebase. On Firebase, enable Phone Number Authentication and Real-Time Firestore. Connect to socketio and start the server. Also, connect to the Google and Facebook servers. Create an express server and an API for real-time data transfer among users. Allow geolocation and integrate react native maps to see where your users are.

Initialise your project in the root directory with the following command after installing Node, Express, NPM (Node Package Manager):

```
npx create-react-app myproject_name
```

For styling and conditional rendering, we have used an improvement over the normal css.

The technology used for the css is **Sass** or commonly known as the **SCSS**. **Also**, the rsuite library has been used for the rendering of built-in components in React. The React-icons serve as the built-in library for the rendering of icons in the app, Also, styled components have been used in the app almost everywhere. The **react-router-dom** from React has the special Switch and Route feature to render the different routes in the app. The following can be installed by the following commands using either npm or yarn package managers.

```
npm install styled-components react-router-dom scss
```

After making the necessary folders in Firebase, it looks something like this.

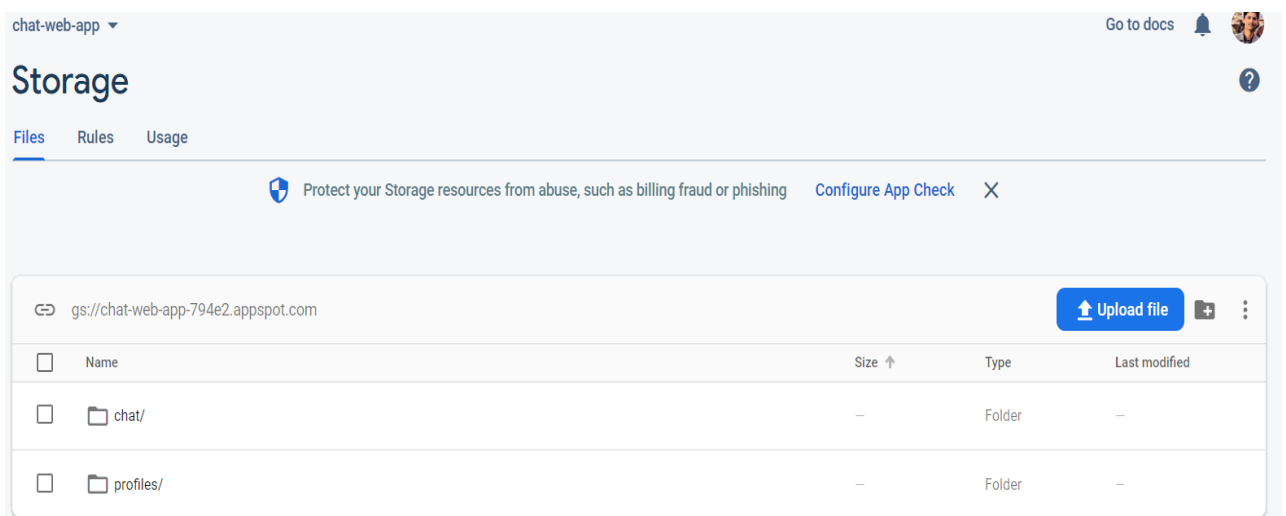


Fig 2: The figure showing the firebase chat and profile folders.

## 1.5 Organization

The organization of data and model in the algorithm is as follows:

- The dependencies installed on the system are the following: react-dom, react-router-dom, rsuite, timeago-react, node-sass and firebase. The dependencies installed for the app can be replicated by going over to the github repository and cloning it using the given link:

<https://github.com/Ojaswi2000/chat-app.git>

- Configure firebase on the system by adding a new project called chat-web-app on the system,

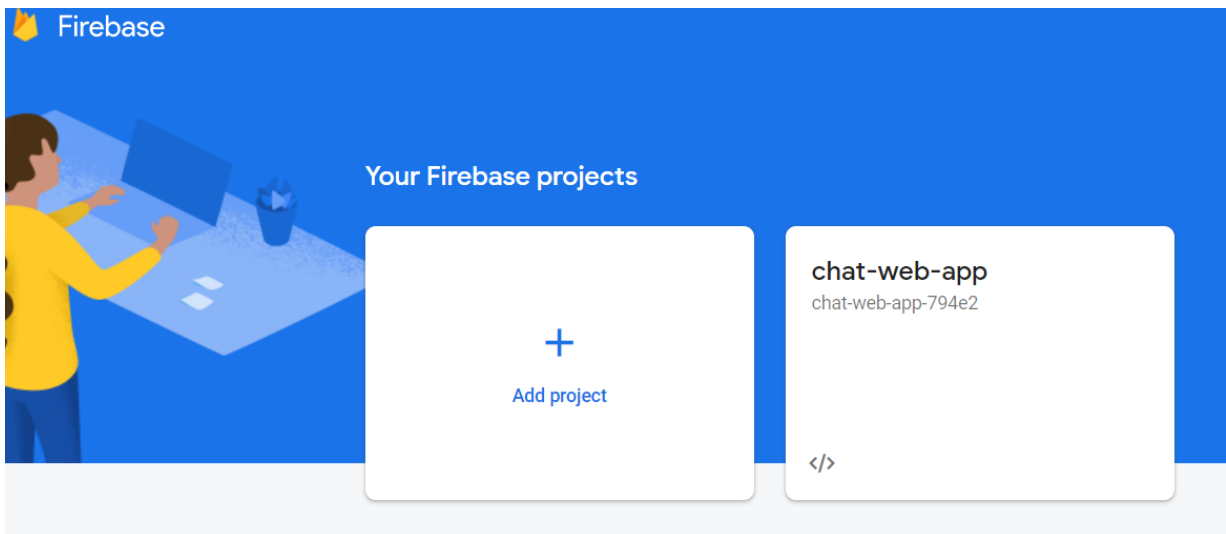


Fig 3: Figure showing initialization of chat-web-app

- **Tools and Technologies used in the project:**

i) **React:**



ii) **Express:**



iii) Node JS:



iv) Google Firebase:



v) Sass (Styled CSS):



vi) React Router Dom:



vii) Node Package Manager(NPM)



Once all the dependencies have been installed and all the project files have been initialized, the file structure looks something like this:

Also, the project needs to be run on a terminal. We can do it in the following three ways.

One of them can be adopted for doing this:

1. Open the command terminal of your system and type `npm start` in the directory of the project.
2. Run `npm start` from the VS Code terminal. It opens the project in the root directory by default,
3. We can run the project by installing and configuring git on the system.

The file structure is as follows:

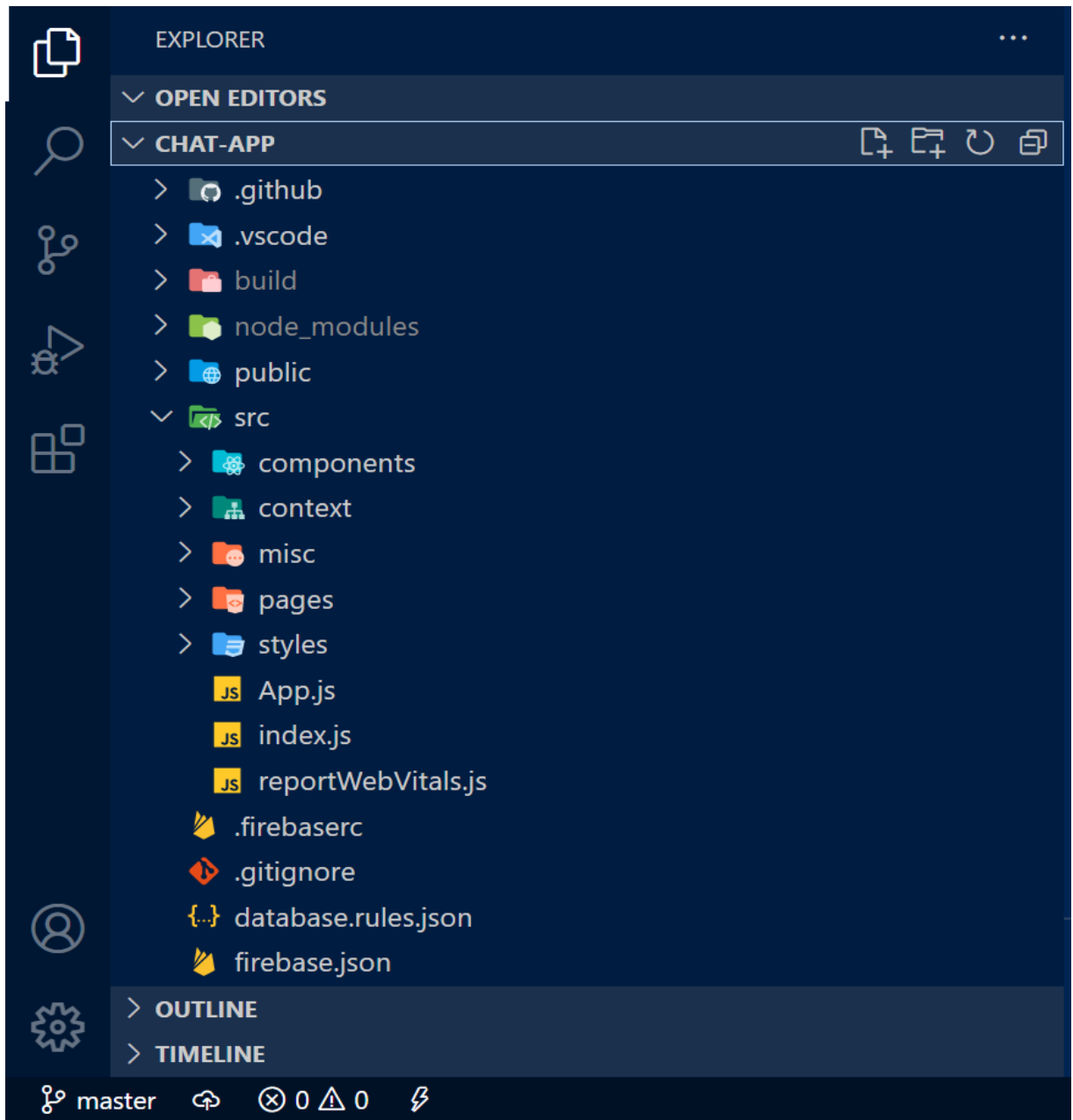


Fig 4: The file structure



## CHAPTER 2: LITERATURE SURVEY

1) Currently, millions of apps are available in different online stores to smartphone users. The most successful mobile applications have been downloaded over a billion times [1] and each day new applications are launched to the mobile market, making it extremely attractive both for companies and indie developers to invest their time and money. Such demand has often led mobile software (SW) developers to adapt established SW development methodologies or submit new proposals that fit the constraints related to mobile SW development. The mobile SW development particularities are diverse, but surely include short and frequent development cycles, frequent technological changes (platforms, operating systems, sensors, etc.), limited documentation, specific requirements and resources of the development team and the client, among others. In addition, all these possible factors are prone to constant innovation [2]. The steps being taken to develop mobile applications may differ depending on the project or established technologies. Mobile apps are the result of several activities that are carried out, such as, assigning roles to the work group, defining objectives and activities, implementing a set of good practices for teamwork and collaboration, establishing the schedule of activities, risk management, among others. In the end, guidelines must be adapted according to available resources and customer requirements [3] [4].

Furthermore, mobile application development requires continuous improvements and adaptations to meet new technological needs and changes, presenting significant challenges such as: design of user interfaces for different sizes of screens of mobile devices, user experience linked to capabilities of mobile devices, user interaction methods provided by mobile platforms, architectures, among others [5]. This need for speed of adaptability is one of the causes for the limited formal and scientific documentation in the field of mobile application development. At present, there is a lack of scientific documentation that reflects development processes focused on mobile applications and their activities [6]. SW-level mobile application development processes are not fully established by today's industry. The objective of this research is to better understand the current methodologies adopted, to identify which and how processes or methodologies relevant to the development of mobile applications are carried out in two contexts: academic and industry, counting for the latter with case studies in small, medium and large Ecuadorian and Mexican companies. This study also contributes to better understand mobile application development processes, examining the real challenges faced, the activities carried out, and considering the

Characteristics of the development team using real case scenarios. The rest of the paper is organized as follows. Section II addresses related work, while Section III sets out the applied research method. Section IV presents the results obtained based on the research sub-questions raised and, in Section V, the discussion of the results is present. Finally, Section VI presents the conclusions and future lines of research

2) Mobile technology is an exceptionally fast-growing field that is closely connected with our work and day-to-day lives. There are new developments added to its growth every day with emerging new patterns of usage, having both positive and negative implications. In the twenty-first century, higher education institutions had to be reconstructed to adapt to changes with the increasing global competition, the growing need for higher education, the changing nature of information, rapid developments in Information and Communication Technologies (ICT) and the varying expectations and demographic features of learners ([Kukulska-Hulme, 2005a](#)). The changes in the dynamics of ICT, institutions and learners influence the academics working in higher education institutions to change their teaching approaches and strategies.

However, we have not seen a noteworthy adoption of these technologies in the education sector even though they are available everywhere (ubiquitous) and have tremendous potential in addressing needs of the individual learner through their unique capabilities. Furthermore, owing to the rapid changes of mobile technologies, including devices and communication technologies have opened up new research opportunities and even change the focus of research ([Parsons, 2014](#)). [Krull and Duart \(2017\)](#) reported that in higher education, mobile learning is a growing field of research as evidenced by reviewing journal publications between 2011 and 2015. The major results of their study were that the most researched theme was on m-learning applications and systems, used both quantitative and qualitative studies and were targeted at students. As both faculty and student adoption play a crucial part in the success of mobile learning initiatives, they recommend future studies to look into the implications for both faculty and students. However, there is a scarcity of research articles related to mobile learning and methodological frameworks for designing sustainable mobile learning activities ([Nouri et al., 2016](#)). The purpose of this research study was to address this gap by applying design-based research in designing a mobile learning solution for the undergraduates of the Faculty of Health Sciences of the Open University of Sri Lanka (OUSL). It reports on the findings of the testing phase of the mobile solution by five groups of stakeholders: content experts, educational technologists, developer, novice users and researchers prior to the delivery of the first cycle. The first

section of the paper defines briefly the mobile learning, design-based research and employing design-based research in mobile applications, and stresses the importance of conducting design-based research for future technological innovations. The second section briefly describes the context. The third section examines the methodology adopted for the design-based research for new technological innovations in teaching learning using mobile applications. The fourth section is dedicated to the findings which were collected from all the stakeholders illustrating the potential for innovative teaching practices through mobile learning. The final section is a critical examination of the viewpoints expressed by all stakeholders and formulating guiding principles for both designing mobile learning solutions and on how to conduct design based research in mobile applications.

3.) Chatting applications or rather the Teleconferencing chatting applications have been a technique of getting together people across different countries and barriers. This technology has existed in the tech domain but its acceptance has been quite new and recent. This project builds a react chatting application hosted on Firebase for the database part and built on node JS and react-router-dom. To begin the chatting application, the user needs to run the npm start command which is entrusted with the responsibility of connecting to localhost:3000 on the machine. This connects the app to a server on port 3000 which is by default for React and in the range of 4000 for Angular.

4.) Since the introduction of the i-phone in 2007, app stores have become a popular mechanism for hosting and distributing applications for mobile devices, so-called apps. Apps generated \$53 billion revenue in 2012, and are predicted to generate \$68 billion revenue in 2013 [22]. App developments much more constricted in the choice of technologies as every mobile platform favors different frameworks. Besides native apps that are written in the preferred programming language of the respective platform, HTML5 technologies gain traction for the development of mobile apps [12, 4]. A recent survey among developers revealed that more than half (52%) are using HTML5 technologies for developing mobile apps [22]. HTML5 technologies enable the reuse of the presentation layer and high-level logic across multiple platforms. However, an earlier survey [21] on cross-platform developer tools revealed that access to native device APIs is the biggest shortcoming of HTML5 compared to native apps. Several different approaches exist to overcome this limitation. Besides the development of native apps for specific platforms, popular approaches include cross-platform compilation [14] and packaging common HTML5 code into native apps [23]. The use of various wireless technologies like GPS, GPRS, and Bluetooth leads us to monitor the patient remotely. The system is said to be an



intelligent system because of its diagnosis capability, timely alert for medication etc. In this paper we propose a novel approach that uses a generic device-local service, or service for short, that runs on the mobile device and that acts as a gateway, exposing native device APIs to HTML5-based web apps running inside a regular browser. We show how Web Sockets and HTTP can be used for efficient bi-directional communication between web apps and the service. The service approach provides a clear separation between a web app, a web browser, and a device local service, thereby generalizing the established packaging approach. By bundling the device-local service with the app it is also possible to mimic the packaging approach.

Specifically, this project makes the following contributions:

- A service-based approach to expose native APIs to web apps
- A reliable and efficient Web Socket-based communication protocol between the native shell and the web app
- An authentication and authorization scheme to address security and privacy concerns
- Implementations for Android and Windows Phone and the web application have been focused primarily on.

The report is structured as follows:

In Literature Survey, we present related work and provide a taxonomy by which the various approaches can be compared with each other.

System Development explains in detail the device-local service approach proposed in this paper. Section 4 discusses implementations of the device-local service for the Android and the Windows Phone platform. Finally, in the Conclusions Section we provide a conclusion and an outlook for future work.

The table below shows the extensibility, hosting, performance and dependencies

	Platform	Packaging	Compilation	Service
Extensibility	Difficult	Easy	Easy	Easy
Hosting	Web/App Store	App Store	App Store	Web/App Store
Performance	High	Low	High	High
Dependencies	None	SDK	SDK	Service

Figure showing the comparison of platform, packaging, compilation and service approaches

5.) The firebase component works on the basis of MongoDB software or rather the MongoDB Atlas which is a NO-SQL technology in the MERN stack (MongoDB, Express, React and Node JS). The mongo intends to build a mongoose schema much rather like the schema in a SQL table that comprises of rows and columns with the only difference being the fact that the mongoDB or firebase focus on building a schema that does not have a table consisting of tables, rows and columns, We can have constraints in the values held in the database. We can insert the documents in the database inside a cluster of documents which is also known as a collection of documents. The most common CRUD operations, that is, the create, read, update and delete operations have been performed in the database on the basis of the firebase or mongo schema. This schema creates a firebase.json file in the root directory of the project and we can export several functions pertaining to firebase out of this file. These exported functions can then be imported in other files wherever they are needed and be used accordingly.

The Real-time Firebase database corresponding to the project called “Chat-web-app” has a fixed set of rules that can be found at the backend on the rules page.

```
1 ▾ {
2 ▾   "rules": {
3 ▾     "profiles":{
4 ▾       "$user_id":{
5 ▾         ".read":"$user_id === auth.uid",
6 ▾         ".write":"$user_id === auth.uid"
7 ▾       }
8 ▾     },
9 ▾     "rooms":{
10 ▾       ".read":"auth !== null",
11 ▾       "$room_id":{
12 ▾         ".read":"auth !== null",
13 ▾         ".write":"!data.exists() || data.child('admins').child(auth.uid).val()=== true",
14 ▾         "lastMessage":{
15 ▾           ".write":"auth !== null"
16 ▾         }
17 ▾       }
18 ▾     },
```

Fig 5: The figure shows the rules for the profiles and chatrooms files.

```

19 ▾      "messages":{
20         ".read":"auth !== null",
21         ".write":"auth !== null",
22         ".indexOn": "roomId",
23 ▾      "$message_id":{
24         ".read":"auth !== null",
25         ".write":"auth !== null"
26     }
27     },
28 ▾      "status":{
29 ▾      "$user_id":{
30         ".read":"auth !== null",
31         ".write":"$user_id === auth.uid"
32     }
33     },
34     ".read": false,
35     ".write": false
36 }
37 }

```

The figure shows the rules defined for the messages section and the status for user id

The connections, storage and downloads and other billable metrics are also shown on Firebase in the Monitor Rules section of the database. The subscriptions are shown everyday on the same Usage Page. This page is present in the Usage Section of **Realtime Firebase**. The figure shown below shows the new number of users that logged in to the app.

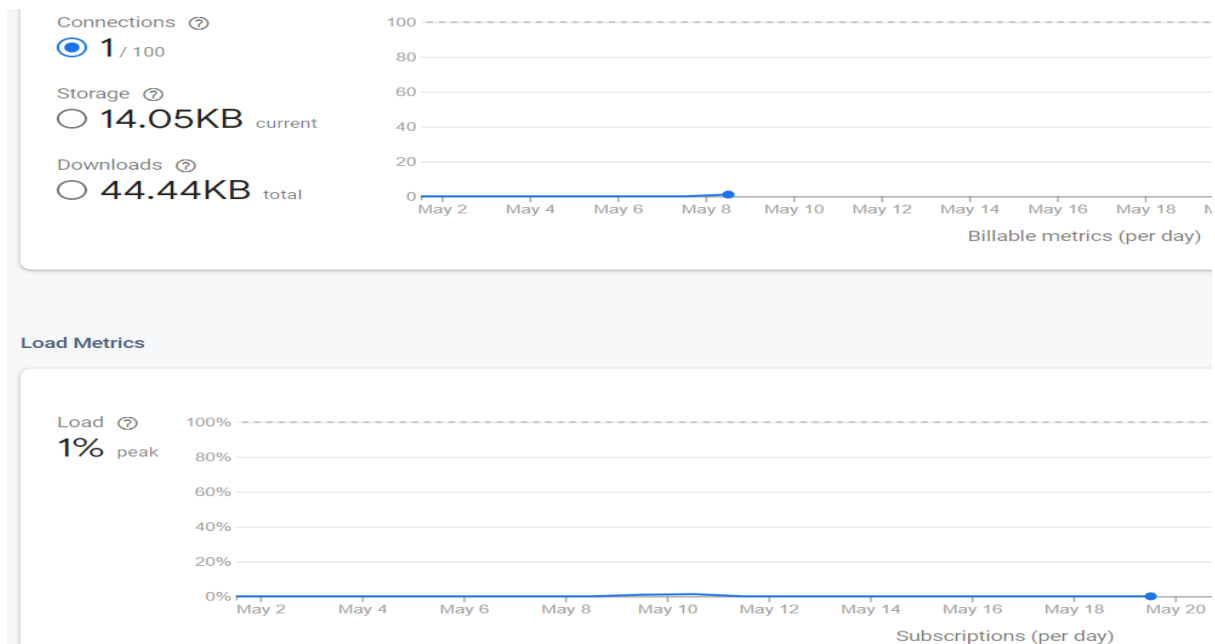


Fig 6:Figure showing the billable metrics of the project

## 6.) Overview of the Socket:

A socket is an object that represents a low-level IP stack access point. This socket can be opened, closed, or in one of a few different states. Down disconnection, a socket can send and receive data. For efficiency, data is typically transferred in blocks of a few kilobytes at a time; each of these blocks is referred to as a packet.

The Internet Protocol must be used by all packets travelling across the internet. This means that the packet must include the source IP address and destination address. A port number is also included in most packets. A port is a number between 1 and 65,535 that is used to distinguish between higher protocols. When it comes to designing your own network apps, ports are crucial since no two applications can use the same port.

UDP and TCP/IP are the two types of packets that contain port numbers. UDP is faster than TCP/IP, especially during startup. UDP can be easier to use than TCP when data integrity isn't as crucial, but it should never be used when data integrity is more important than performance; nonetheless, data received via UDP can sometimes arrive in the wrong sequence, rendering it unusable to the receiver. TCP/IP is more complicated than UDP and has greater latency in general, but it ensures that data is not damaged when travelling across the internet. TCP is good for file transfers, because a defective file is more unacceptable than a slow download;

nevertheless, it is unsuitable for internet radio, where an occasional out-of-place sound is preferable than long periods of quiet.

The User Datagram Protocol (UDP) is a connectionless protocol that employs an IP address to identify the destination host and a port number to identify the destination application. Any physical port on a computer, such as a COM port or a I/O port address, is not the same as the UDP port number. The UDP port is a 16-bit address that exists solely to send specific sorts of datagram information to the correct point above the protocol stack's transport layer. A UDP datagram header consists of four (4) fields of two bytes each:

1. source port number
2. destination port number
3. datagram size
4. checksum



## CHAPTER 3: SYSTEM DEVELOPMENT

### 3.1 Android as an Operating System

Android is a Linux-based open source operating system for mobile devices such as smartphones and tablets. The Open Handset Alliance, lead by Google, and other firms collaborated to create Android.

Android takes a unified approach to mobile application development, which means that developers simply have to code for Android, and their apps should run on a variety of Android-powered devices.

Google published the first beta version of the Android Software Development Kit (SDK) in 2007, followed by the first commercial version, Android 1.0, in September 2008. Google revealed the next Android version, 4.1 Jelly Bean, during the Google I/O conference on June 27, 2012. Jelly Bean is a step-by-step upgrade aimed at improving the user interface, both in terms of functionality and efficiency.

Android's source code is distributed under free and open source software licences. The Apache License version 2.0 covers the majority of the code, whereas the GNU General Public License version covers the Linux kernel changes.

#### 3.1.1 Software development kit

SDK stands for software development kit, sometimes known as dev-kit. It's a collection of software tools and programmes that developers utilise to create apps for various platforms.

SDK tools will feature a variety of items that developers may utilise and incorporate into their own projects, such as libraries, documentation, code examples, workflows, and instructions. SDKs are software development kits that are tailored to specific platforms or programming languages.

To create an Android app, you'd need an Android SDK toolkit, an iOS SDK toolkit, a VMware SDK for connecting with the VMware platform, or a Nordic SDK for creating Bluetooth or wireless products, etc.

#### 3.1.2 Dalvik Virtual Machine Architecture

Modified version of JAVA language is used for application development with the help of Dalvik VM which is used to run the mobile app on Android devices. This Dalvik VM can be viewed as the modified version of JVM which constrained in term of storage and processing speed and convert the java bytecode in form of JVM compatible .class files to be compatible by Dalvik VM and is converted in. dex which is executable before installation.



Fig 7: Figure showing a break-up of Android features

### 3.2 JavaScript framework –React JS

React Native is a JavaScript framework for creating real-time, natively rendered web apps that can also generate a mobile view for iOS and Android. It's built on React, Facebook's JavaScript toolkit for creating user interfaces, but it's designed for local environments and mobile platforms rather than the browser. In other words, web developers can now create mobile applications that look and feel fully "native," all while using the familiar JavaScript library. Furthermore, because most of the code you create can be shared across platforms, React Native makes it simple to develop for both Android and iOS at the same time. React and Native applications, like React Native on the Web, are created with JSX, a combination of JavaScript and XML syntax. The React Native "bridge" then calls the native rendering APIs in Objective-C (for iOS) or Java (for Android) (for Android). As a result, your app will appear and feel like any other mobile app, as it will be rendered using genuine mobile UI components rather than web-views. React Native also offers JavaScript interfaces for platform APIs, allowing your React Native apps to exploit features like the phone camera or the user's location. React Native is now available for iOS and Android, with the possibility to expand to other platforms in the future. Both iOS and Android will be covered in this book. We'll be writing cross-platform code for the most part. And, yes, React Native can be used to create production-ready mobile apps. Facebook, Netflix, AirBnb, Instagram, and TaskRabbit are among the apps that use it in production for user-facing applications.

#### 3.2.1 JavaScript Runtime

You'll be running your JavaScript code in two settings while utilising React Javascript:

- 1) JavaScriptCore, the JavaScript engine that underpins Safari, will be used in most circumstances by React or React Native. Due to the lack of writable executable memory in iOS apps, JavaScriptCore does not employ JIT on iOS.
- 2) When debugging with Chrome, all JavaScript code runs inside Chrome, connecting with native code via WebSockets. The JavaScript engine in Chrome is V8. While both surroundings are relatively similar, there may be minor discrepancies. We'll probably experiment with various JavaScript engines in the future, so don't get too hung up on runtime specifics.

### 3.2.2 JavaScript Syntax Transformers

Syntax transformers make creating code more fun by allowing you to use new JavaScript syntax without waiting for all interpreters to implement it.

The Babel JavaScript compiler is included with React Native. More information about Babel's supported transformations may be found in its documentation. The metro-react-native-babel-preset contains a complete list of React Native's supported transformations.

#### ES5

- Reserved Words: `promise.catch(function() { });`

#### ES6

- Arrow functions: `this.setState({pressed: true})` />
- Block scoping: `let greeting = 'hi';`
- Call spread: `Math.max(...array);`
- Classes: `class C extends React.Component { render() { return ; } }`
- Constants: `const answer = 42;`
- Destructuring: `var {isActive, style} = this.props;`
- for...of: `for (var num of [1, 2, 3]) {};`
- Modules: `import React, { Component } from 'react';`
- Computed Properties: `var key = 'abc'; var obj = {[key]: 10};`
- Object Concise Method: `var obj = { method() { return 10; } };`
- Object Short Notation: `var name = 'vjeux'; var obj = { name };`
- Rest Params: `function(type, ...args) {};`
- Template Literals: `var who = 'world'; var str = `Hello ${who}`;`

ES8

- Function Trailing Comma: `function f(a, b, c,) {};`
- Async Functions: `async function doStuffAsync() { const foo = await doOtherStuffAsync(); };` Stage 3
- Object Spread: `var extended = { ...obj, a: 10 }`

### 3.3 React rendering of the data

React is a popular Javascript framework that allows you to create user interfaces as a DOM tree of discrete code pieces called React components. In React, a component is a mix of HTML and JSX that contains all of the code needed to render a tiny piece of a larger User Interface. All of these React components are stacked one on top of the other to create increasingly complex elements of an app. The details are the rest.

React primitives render to native platform UI, which means the app uses the same APIs as other apps on the platform.

Many platforms, one React. Create platform-specific components to share a single codebase across many platforms. React Native allows a single team to manage two platforms while sharing a common technology—React.

### 3.4 Everyone Can Benefit From Native Development

With React Javascript and React Native, you can build truly native apps without sacrificing the user experience. It provides a platform-independent base of native components including View, Text, and Image. directly to the platform's native UI building components.

#### Cross-Platform Integrity

React components wrap existing native code and interface with native APIs using React's declarative UI. The paradigm and JavaScript This enables for the creation of entirely new teams of native app developers.

This could speed up the work of present local teams.

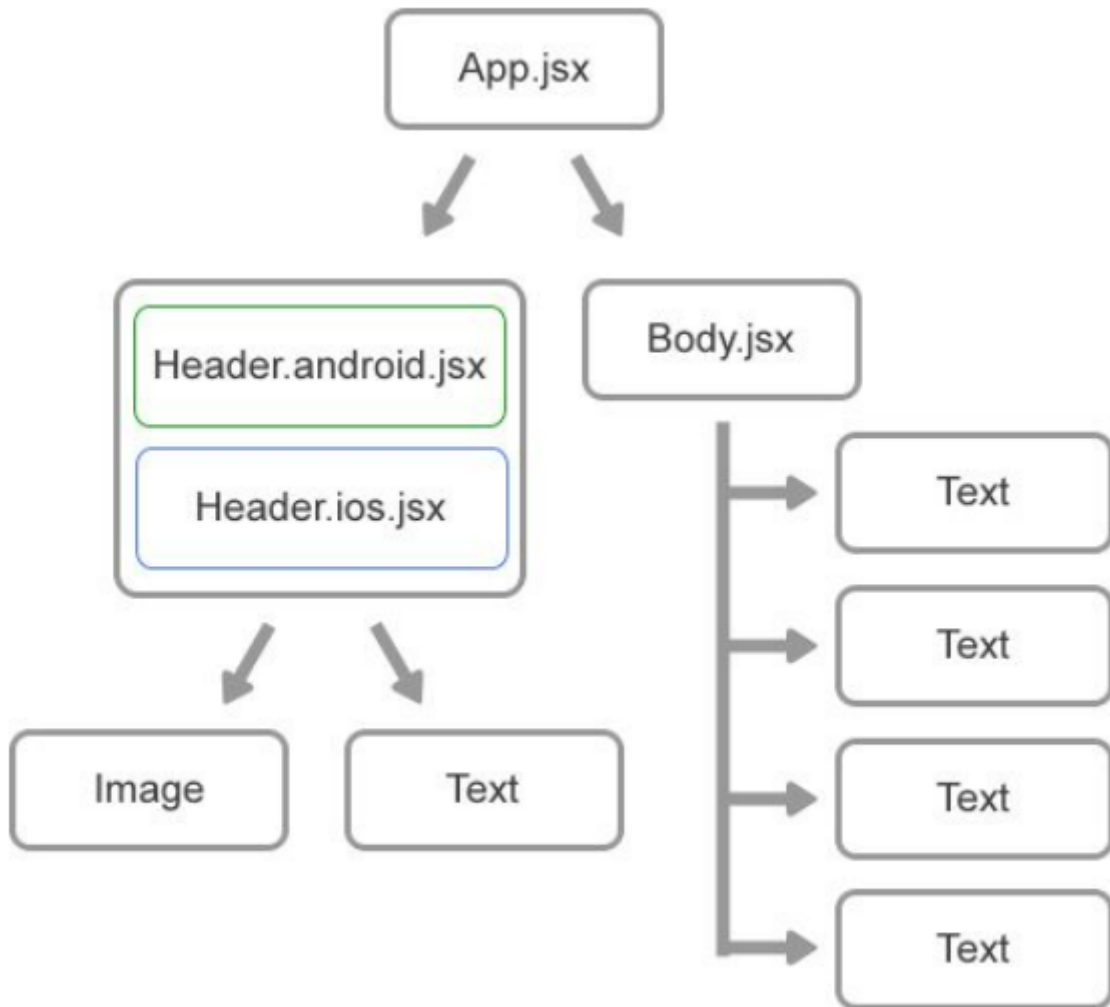


Fig 8: Figure showing the break-up of App javascript file into header and body

### 3.5 Node Package Manager

npm, or Node Package Manager, is a command-line programme for interfacing with an online repository for publishing open-source Node.js projects. On npm, there are thousands of Node.js libraries and applications, and more are added every day. On <http://npmjs.org/>, you can look for these programmes. Once you have a package that you wish to install, you can do it with only one command.

Let's pretend you're working on the Next Great Application one day. You run into a difficulty and decide it's time to use that amazing library you've heard so much about - let's use as an example, consider Caolan McMahon's `async`.

Fortunately, `npm` is really easy to use: simply execute `npm install async`, and the desired module will be installed in the current directory under `./node_modules/`. You can use `require()` on them once they've been installed in your node modules folder, just like built-ins.

Consider the following example of a global install: `coffee-script`. The `npm` command is straightforward:

```
coffee-script -g npm install
```

This will usually install the software and create a symlink in `/usr/local/bin/` for it. This will allow you to launch the programme like any other CLI tool from the console. Running `coffee` in this situation will allow you to use the `coffee-script` REPL.

Another important use for `npm` is dependency management. When you have a node project with a `package.json` file, you can run `npm install` from the project root and `npm` will install all the dependencies listed in the `package.json`.

### 3.6 Creating react Javascript project

Install React JS globally: `npm install -g react-native-cli`

- Create a new React Bootstrapped project (Note: this step may take a while):

```
npx create-react-app chat-app
```

- CD into your new project

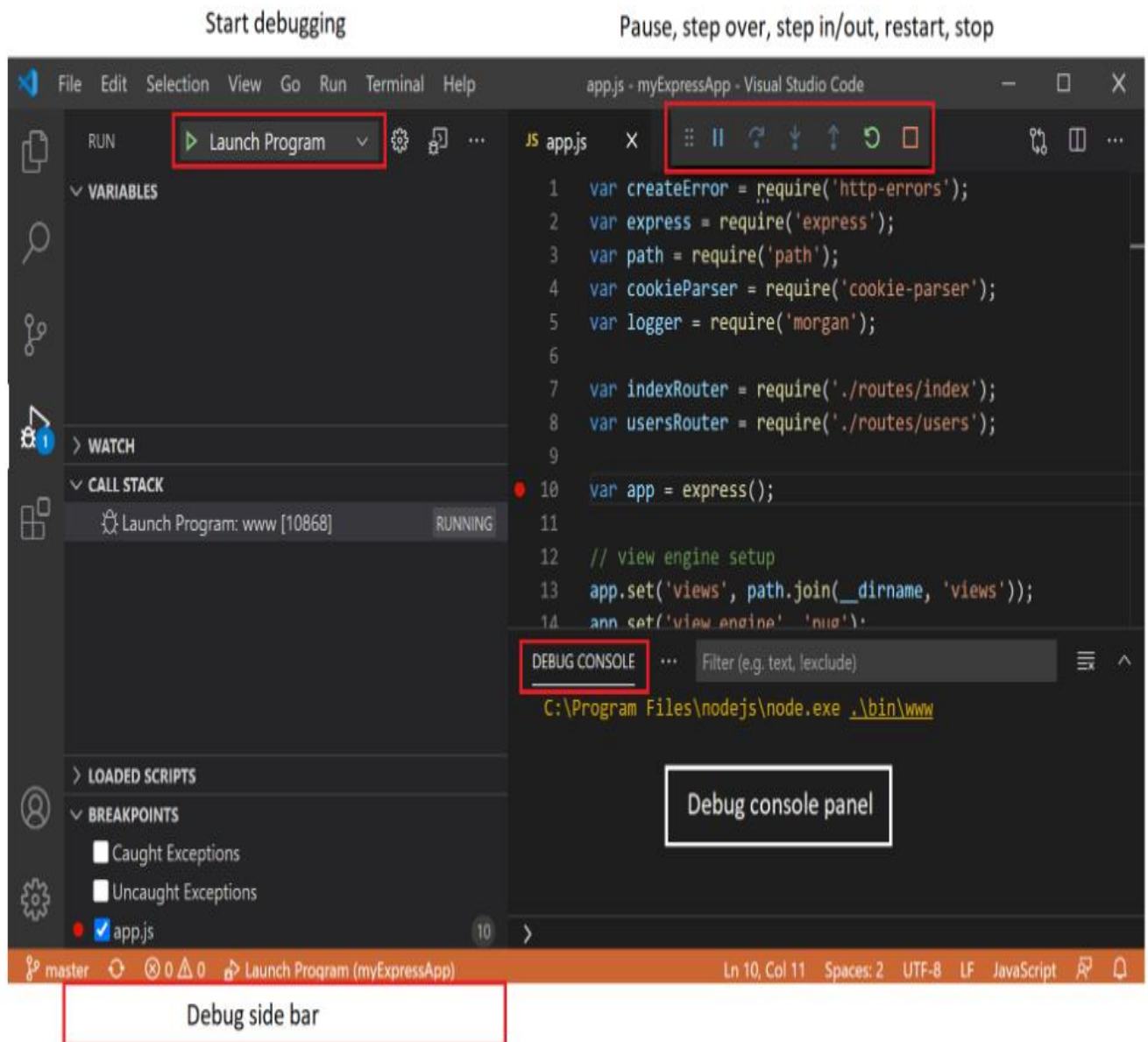
```
cd documents/chat-app (assuming the project is in the documents folder)
```

- Optional: Configure Gradle Daemon for faster compiling

### 3.6 Setting up the Debugger in launch.json file in VSCode

One of the most primary features of VS Code is its very popular feature of debugging the code. Visual Studio Code's built-in debugging tool helps us to accelerate and accentuate our edit, compile, and debug loop. The very traditional way of debugging the code and finding the errors has been commenting the coded part which contains the error portion, running the app again

and then looking for errors in the browser and then rectifying them from there. Testing the code becomes redundant hence the React devTools is not preferred on the browser but is instead the option of VSCode Debugger is rather preferred.



The following steps have to be followed for debugging the code:

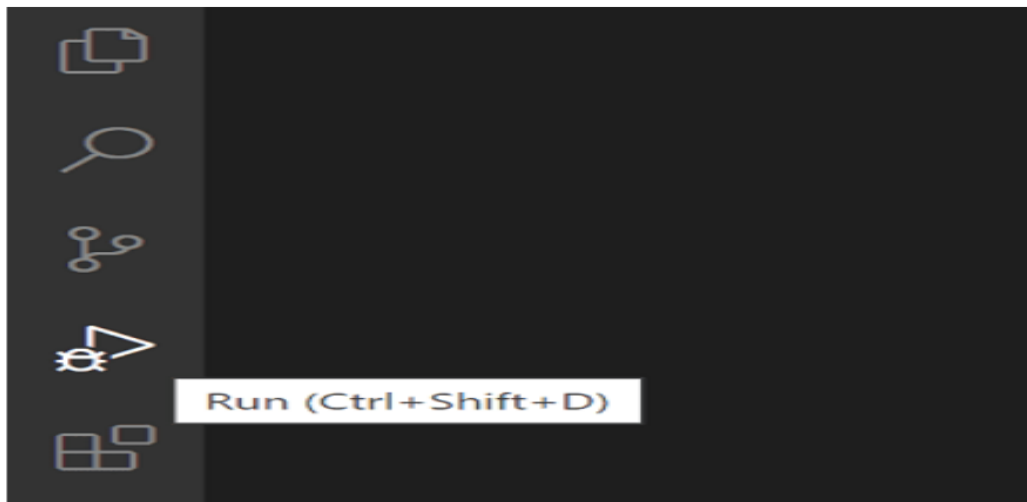
1. Click on the red gutter icon before the line number that you want to check for errors. This will create a red bright spot called the break point.
2. Then click on The **Run and Debug** in the icon list.



3. Launch the run and debug by clicking on the green Play icon.
4. This creates a slider on the screen with many different options.
5. The variables, call stack and loaded scripts will appear in a Sidebar to the left to the IDE Screen.
6. The Debug Console will display all the errors in the console inside VSCode.

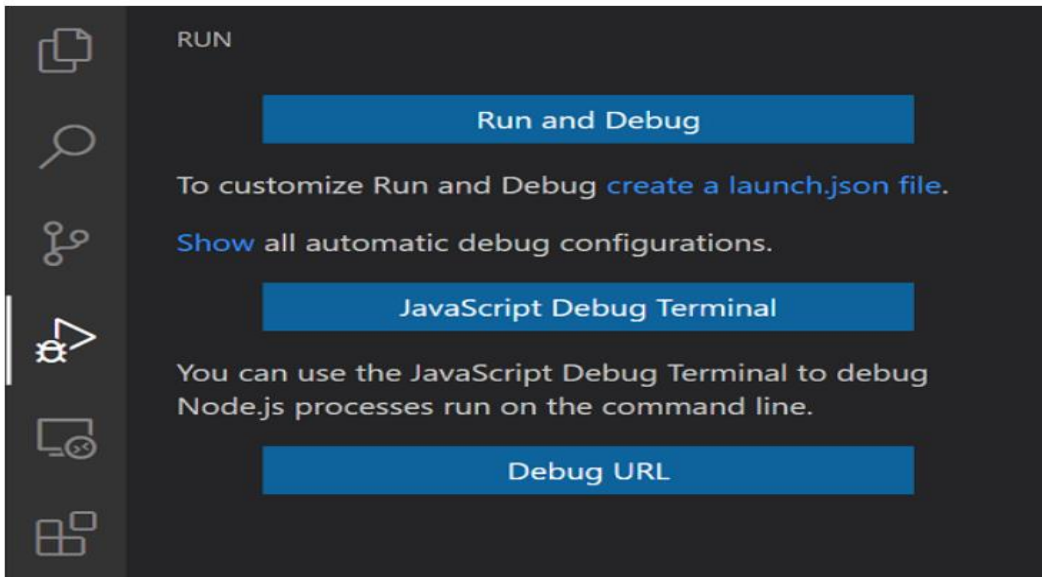
### Run View:

In order for the user to bring up the Run View, use the short cut for it : **Ctrl+Shift+D** or select the Run Icon in the Activity Bar on the top of the VSCode.



The Run view includes a top bar containing debugging commands and configuration settings, as well as any information related to running and debugging.

VS Code displays the Run start view if running and debugging have not yet been configured (no launch.json has been produced).



Click on the launch.json file. This adds a launch.json file and a configuration by default too. We can also add another configuration also in the same file. The structure of the launch file has been shown below:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "skipFiles": ["<node_internals>/**"],
      "program": "${workspaceFolder}\\app.js"
    }
  ]
}
```

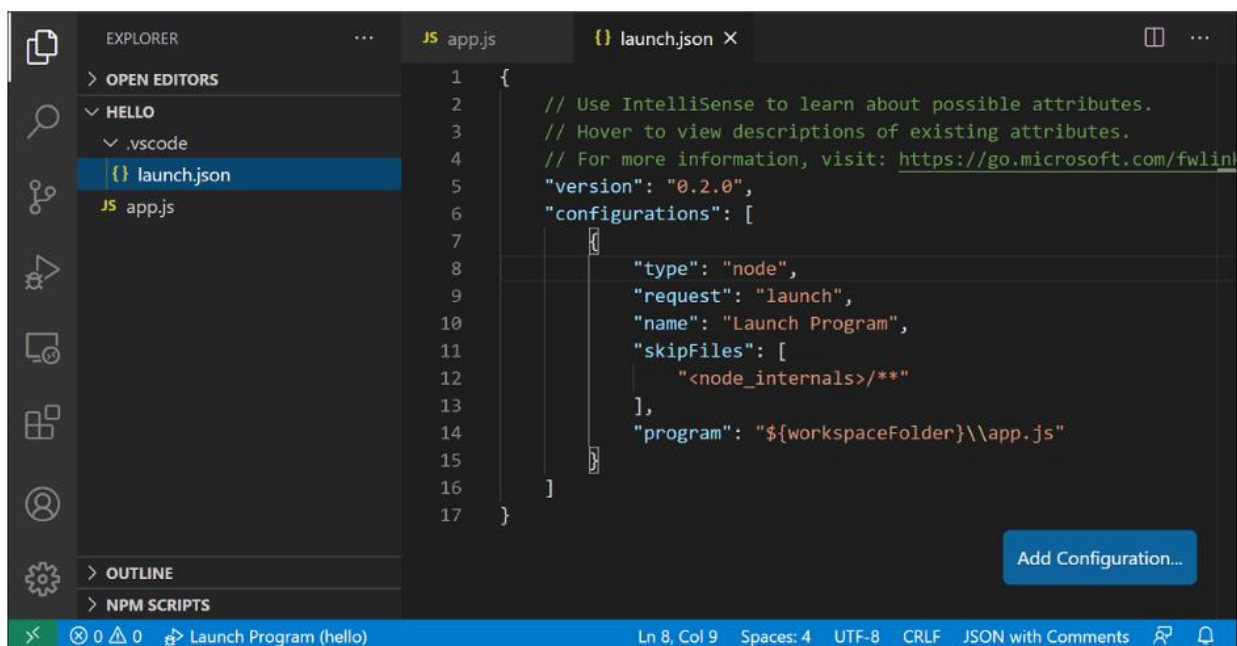


Fig 9: The figure shows the launch.json file properly configured in the system

With the advent of the React technology, The Switch has been replaced by the Routers component that can be imported as the BrowserRouter component. Each App file which is the start of the process of execution will contain one Router (earlier Switch) and then a number of different routes in the app will be present inside the Router tag.

### 3.7 Google FireBase

Google Firebase is a Google-backed app development platform that allows developers to create apps for iOS, Android, and the web. Firebase delivers analytics reporting, tracking and app issue fixes, as well as marketing and product experimentation capabilities.

**Some of the features that it provides are as follows:**

- Google Analytics for Firebase provides free, limitless reporting on up to 500 different events. Analytics gives data about user activity in iOS and Android apps, allowing for better performance and app marketing decisions.
- Authentication — Firebase Authentication makes it simple for developers to create secure authentication systems while also improving user sign-in and onboarding. This feature provides

a comprehensive identification solution, including email and password accounts, phone authentication, and Google, Facebook, GitHub, Twitter, and other social media logins.

- Firebase Cloud Messaging (FCM) is a free cross-platform messaging application that allows businesses to reliably receive and send messages on iOS, Android, and the web.
- The Firebase Realtime Database is a cloud-hosted NoSQL database that allows data to be stored and synced in real time across users. The data is synced in real time across all clients and is remains accessible when an app goes offline.
- Firebase – Crashlytics Crashlytics is a real-time crash reporter that aids developers in tracking, prioritising, and resolving stability issues that degrade app quality. Developers may spend less time organising and investigating crashes and more time implementing features for their apps with crashlytics.
- Firebase Performance Monitoring gives developers visibility into the performance characteristics of their iOS and Android apps, allowing them to understand where and when their apps' performance may be enhanced.
- Firebase Test Lab is a cloud-based infrastructure for app testing. Developers may test their iOS or Android apps on a multitude of devices and configurations with just one action. They may view the results in the Firebase console, which includes videos, images, and logs.

### **3.7.1 Integrating firebase with react**

To run the command on the terminal, here gitbash has been used. After merging the modules you use with a package bundler, the Firebase JavaScript npm package contains code that can be launched in the browser (e.g., Browserify, Webpack).

Install the npm module for Firebase:

```
$ npm init
```

```
$ npm install --save firebase
```

Access Firebase using:

```
var firebase = require('firebase');
```

```
var app = firebase.initializeApp({ ... })
```

The following screenshot shows an attempt using try-catch block to make a connection to Firebase in the Sign In page.

```
try {
  const {additionalUserInfo,user}= await auth.signInWithPopup(provider);

  if(additionalUserInfo.isNewUser){
    await database.ref(`/profiles/${user.uid}`).set({
      name: user.displayName,
      createdAt: firebase.database.ServerValue.TIMESTAMP
    })
  }

  Alert.success("Signed In", 4000);
} catch (err) {
  Alert.error(err.message, 4000);
}
```

Fig 10: The figure shows an asynchronous function using a try-catch block to make a connection to the firebase server using the user's gmail id

### 3.7.2 Including features needed from firebase

The full Firebase JavaScript client includes support for Firebase Authentication, the Firebase Realtime Database, Firebase Storage, and Firebase Cloud Messaging. Including code via the above snippets will pull in all of these features.

You can reduce the amount of code your app uses by just including the features you need. The individually installable services are:

- firebase-app - The core firebase client (required).
- firebase-auth - Firebase Authentication (optional).
- firebase-database - The Firebase Realtime Database (optional).

- firebase-firestore - Cloud Firestore (optional).
- firebase-storage - Firebase Storage (optional).
- firebase-messaging - Firebase Cloud Messaging (optional).
- firebase-functions - Firebase Cloud Functions (optional)

Identifier	Providers	Created ↓	Signed In	User UID
snh901002@gmail.com		Oct 27, 2021	Oct 27, 2021	xax8PLAdsNX3BaCIXiaaRTrqPrI2
181391@juitsolan.in		Apr 8, 2021	Jun 2, 2021	jqUKNiVUgVXi4wXI4vSrP9YpL4z2
ojaswiawasthi2025@gmail...		Mar 30, 2021	Aug 11, 2021	Po5YQ3lxA9Vjm54QQQ9uFmOY1...

Rows per page: 50    1 - 3 of 3

The figure shows the number and details of users that are currently signed in the App using their Google IDs. The users can also use their Facebook IDs to login the app. The user can remain connected to the app using both Google and Facebook IDs as well

```
https://chat-web-app-794e2-default-rtdb.firebaseio.com/
└─ messages
  └─ -MYBaXcUQdoxkXyH8ntc
    └─ author
      ├── avatar: "https://firebasestorage.googleapis.com/v0/b/chat-web-app-794e2.appspot.com/o/profiles%2FPo5YQ3lxA9Vjm54QQ9uFm0Y1Hu2%2Favate"
      ├── createdAt: 1617046911994
      ├── name: "OJASWI AWASTHi 34"
      └── uid: "Po5YQ3lxA9Vjm54QQ9uFm0Y1Hu2"
    ├── createdAt: 1618340359216
    ├── likeCount: 1
    └─ likes
```

The structure of nested messages is shown above in the following diagram where every message contains the name of the author who wrote the message, his avatar profile image, the timestamp of when the message was sent on the server and the user-id of the author which is unique for every user. The number of likes I represented using the likeCount symbol that toggles between 1 and 0 for liked and un-liked messages.

```
└─ file
  ├── contentType: "application/json"
  ├── name: "1618405455097scaffolding.json"
  ├── url: "https://firebasestorage.googleapis.com/v0/b/chat-web-app-794e2.appspot.com/o/chat%2F-MX:"
  ├── likeCount: 0
  └── roomId: "-MX3Gmq0KXJQROElmCtQ"
- -MYFTrKuS7T7njFo_QeF
```

The file structure has been shown in the above figure from Firbase Realtime database. The content type is shown in the first row: showing the type of content that file contains which in this case is an application/json structure. The name of the file is shown in the file as the second column and the url of the file is shown in case it has been uploaded on the server. The room ID has been assigned to every chat room in the file.

## 3.8 API Terminology

When using or building APIs, you will encounter these terms frequently:

- **HTTP (Hypertext Transfer Protocol)** is the most common method of transmitting data over the internet. HTTP has a number of "methods" that specify how data should be moved and what should happen to it. The two most popular are GET and GET. POST request pushes fresh data to a server, and GET, which receives data from a server.
- **URL (Uniform Resource Locator)** - A web address for a resource, such as a website. <https://programminghistorian.org/about>. The protocol (<http://>) is the first part of a URL. Optional path (</about>) and domain ([programminghistorian.org](http://programminghistorian.org)). A URL identifies a website, a certain resource's location, such as a web page. When you're reading about APIs, the phrases URL, request, URI, and endpoint are commonly used to denote related concepts.
- **JSON (JavaScript Object Notation)** is a text-based data storage format that is intended for both humans and machines to understand. The most prevalent format for returning data through an API is JSON, with XML coming in second.
- **REST (Representational State Transfer)** is an API implementation philosophy that outlines several best practices. REST APIs are APIs that are built with some or all of these ideas in mind. While the API described in this course employs some REST principles, the name itself is fraught with controversy. As a result, I refer to the example APIs here as web or HTTP APIs rather than REST APIs.

### 3.8.1 Interacting with APIs Using React Fetch

Fetch() is a great networking API that was chosen for React Javascript, but because it is relatively and quite new, there are a few things to be aware of when using it. The React Javascript Documentation does include a single example, which is a good start, but I want to add a few more things that may not be readily obvious.



## GET requests

Sending a GET request to a JSON API is the simplest use case. Just call `fetch` and supply it with the appropriate URL. It returns a promise that can be parsed as usual:

```
fetch('/users.json')  
  
  .then(function(response)  
  
    { return response.json()  
  
    })
```

## POST requests

When submitting a POST request, supply the URL as the first argument and an object containing the request information as the second argument.

There are two things to note:

1. Make sure you send the correct headers. Otherwise, the payload won't get through.
2. Stringify the JSON payload before sending it.

As an example, we can follow the documentation on the React Javascript website:

```
fetch('https://mywebsite.com/endpoint/', {  
  
method: 'POST',  
  
headers: {  
  
  'Accept': 'application/json',  
  
  'Content-Type': 'application/json',  
  
},  
  
body: JSON.stringify({
```

```
    firstParam: 'yourValue',  
    secondParam: 'yourOtherValue',  
  })
```

### **3.9 Core components of the React Javascript:**

#### **3.9.1 Div :**

View/Div is a built-in component in React. If we are familiar with HTML, view is similar to div in that it's utilised in mobile apps. The div or rather a view from React Native is a content section where your material is shown. This allows us to organize the content effectively.

##### **Usage of view**

- When you need to wrap your content inside the container
- When you need to use different style for different element
- When you need a nested element
- It supports for Synthetic touch events, which is used for different purpose.

Figure detailing the different uses of a Div in React or a View in React Native

#### **3.9.2 State:**

A component is controlled by two forms of data. They are state and props, respectively. The state can be changed. It means that the value of a state can change at any time. Variable data is saved in a state.

Create the state in a function `Object() { [native code] }` and modify the value as needed by executing the function `'setState()'`.

#### **3.9.3 props:**

Props (abbreviated for properties) are the second data type. It is unchangeable. It can be used to transfer data between components. It establishes a connection the Container Component with the Presentation Component.

The Container Component handles all of the states and functionalities, whereas the Presentation Component is a passive region where Div is displayed.

The container component is responsible for state initialization and update. The result will be provided to the presentation component, which will use the props to display the view or the div.

### 3.9.4 Flex layout :

Flex Layout is provided to give a clean layout to the component. Children of a component layout are specified using the Flexbox. Using the flexDirection, justifyContent, alignItems properties we can archive the right layout.

#### Sass:

The styled CSS or the scss files are a part of the Sass programming language which happens to be a pre-processing scripting language that, at runtime, gets compiled and interpreted into CSS or the Cascading Style Sheets. The scripting language is called the SassScript. The original syntax is called the indented syntax. This syntax is synonymous with Haml.

# .SCSS

```
.button {
  background: cornflowerblue;
  border-radius: 5px;
  padding: 10px 20px;

  &:hover {
    cursor: pointer;
  }

  &:disabled {
    cursor: default;
    background: grey;
    pointer-events: none;
  }
}
```

# .SASS

```
.button
  background: cornflowerblue
  border-radius: 5px
  padding: 10px 20px

  &:hover
    cursor: pointer

  &:disabled
    cursor: default
    background: grey
    pointer-events: none
```

Fig 11: The figure shows both the .scss and .sass types of files and compares them syntactically

The sass and scss are often used interchangeably. Both use the styled components that can be installed from either the node package manager or the yarn package manager.

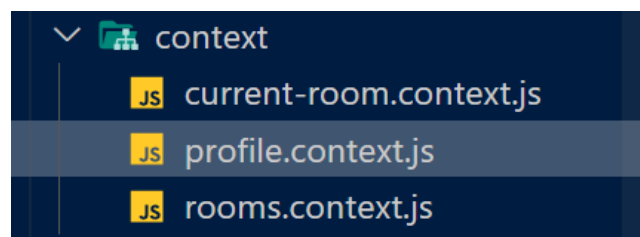
### 3.10 Context API:

This is one of the most important structures in the React JS Framework that makes us fully equipped to exchange unique and personal details. It offers a huge assistance in making the stuck developers solve the process of prop-drilling that exist across all the levels of the application,

The usage of Context API becomes more important when there happens to be some data that has to be made accessible to many different components in an application. This becomes more important when these so called components in the app happen to be at different nested states,

However, the problem that is faced by them is that they make reusing the component more tedious and we need to use it only where necessary or rather sparingly.

In our application, the context API has been used by three different files: namely the current-room, the room-context and the profile.



The context is created with the help of the createContext() function.

```

const ProfileContext= createContext();

export const ProfileProvider = ({children}) => {
  const [profile,setProfile] = useState(null);
  const [isLoading, setIsLoading] =useState(true);

  useEffect(()=>{
    let userRef;
    let userStatusRef;

    const authUnsub =auth.onAuthStateChanged(authObj => {

      if(authObj){
        userStatusRef = database.ref(`/status/${authObj.uid}`);
        userRef =database.ref(`/profiles/${authObj.uid}`);
        userRef.on('value',(snap)=>{
          const {name,createdAt,avatar}=snap.val();

```

Fig 12: Figure showing the Profile Context API

### The rules in Firebase database:

The database which is a NOSQL database needs to have certain rules, despite the fact that it need not be present in the format of a table with rows and columns. These certain rules of authentication are presented in a json file and incorporated into the app inside a file which is labeled as the database.rules.json file. It defines a certain structure for the profiles, chat-rooms, messages and status used in the app. Some of these objects can also be nested and the rules are applicable inside the nested structure too,

For instance, in our app, rooms and messages are nested while profiles and status are not. Also, the read and write permissions are also provided in the json objects.

Below is a snapshot of the rules.

```
{
  "rules": {
    "profiles": {
      "$user_id": {
        ".read": "$user_id === auth.uid",
        ".write": "$user_id === auth.uid"
      }
    },
    "rooms": {
      ".read": "auth !== null",
      "$room_id": {
        ".read": "auth !== null",
        ".write": "!data.exists() || data.child('admins').child(auth.uid).val() === true",
        "lastMessage": {
          ".write": "auth !== null"
        }
      }
    }
  },
}
```

This is the snapshot for the profiles(non-nested) and rooms(nested) objects

```
"messages":{
  ".read":"auth!==null",
  ".write":"auth!==null",
  ".indexOn": "roomId",
  "$message_id":{
    ".read":"auth!==null",
    ".write":"auth!==null"
  }
},
"status":{
  "$user_id":{
    ".read":"auth !== null",
    ".write":"$user_id === auth.uid"
  }
},
".read": false,
".write": false
}
```

This snapshot elucidates the rules for the messages and status in the app

The **@keyframes** library serves as a nice way to provide animations inside the sass files. The below example serves as a great example to show the key frames library in action.

```
.animate-blink {
  animation: blink normal 1.5s infinite ease-in-out;
  color: red !important;
  @keyframes blink {
    0% {
      opacity: 1;
    }
    50% {
      opacity: 0.3;
    }
    100% {
      opacity: 1;
    }
  }
}
```

Blink Animation in action

## CHAPTER 4: PERFORMANCE ANALYSIS

In this chapter 4, we will talk about the different screens that are built into the app and how they are related to each other. The most important screens of the Sign In Page that will appear only when the user has to log in to the app for the first time or when the user has been signed out. Once the user has been signed in, the user gets redirected to the chat screen and can either chat in the existing chat-rooms or create a chat room by himself. The flowchart of the app has been shown below:

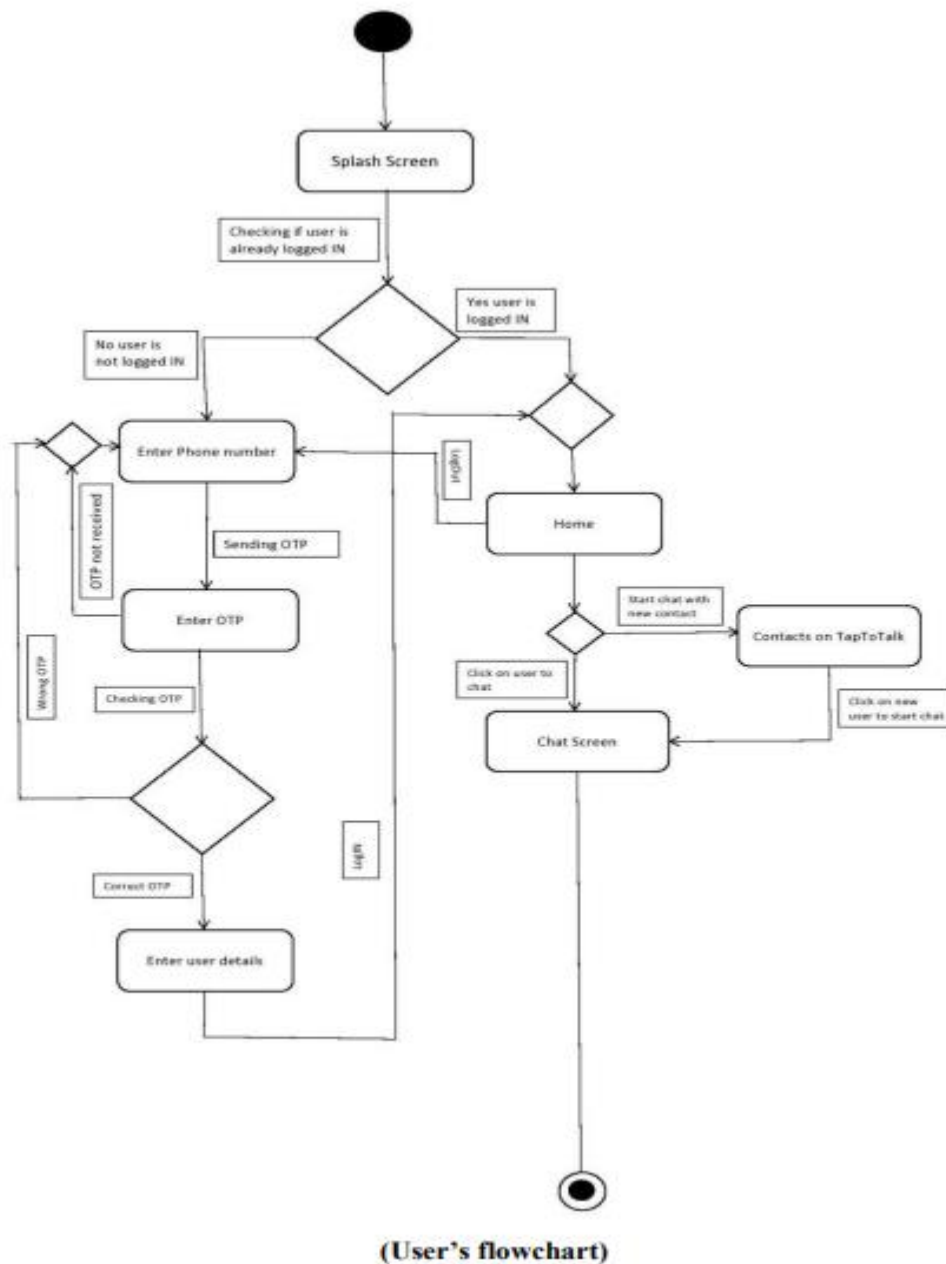


Fig 13: The flowchart of the app for the user

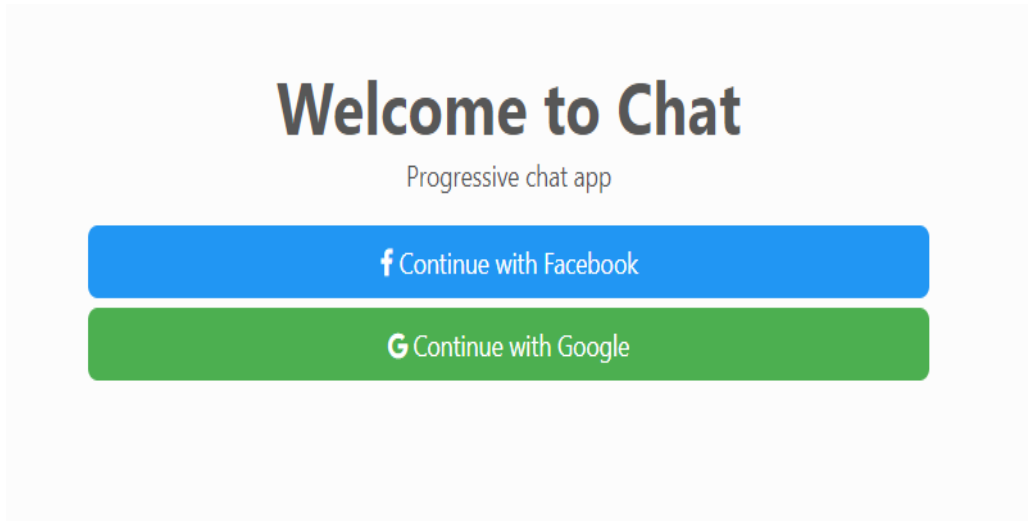


The User's flowchart has been explained in the following steps:

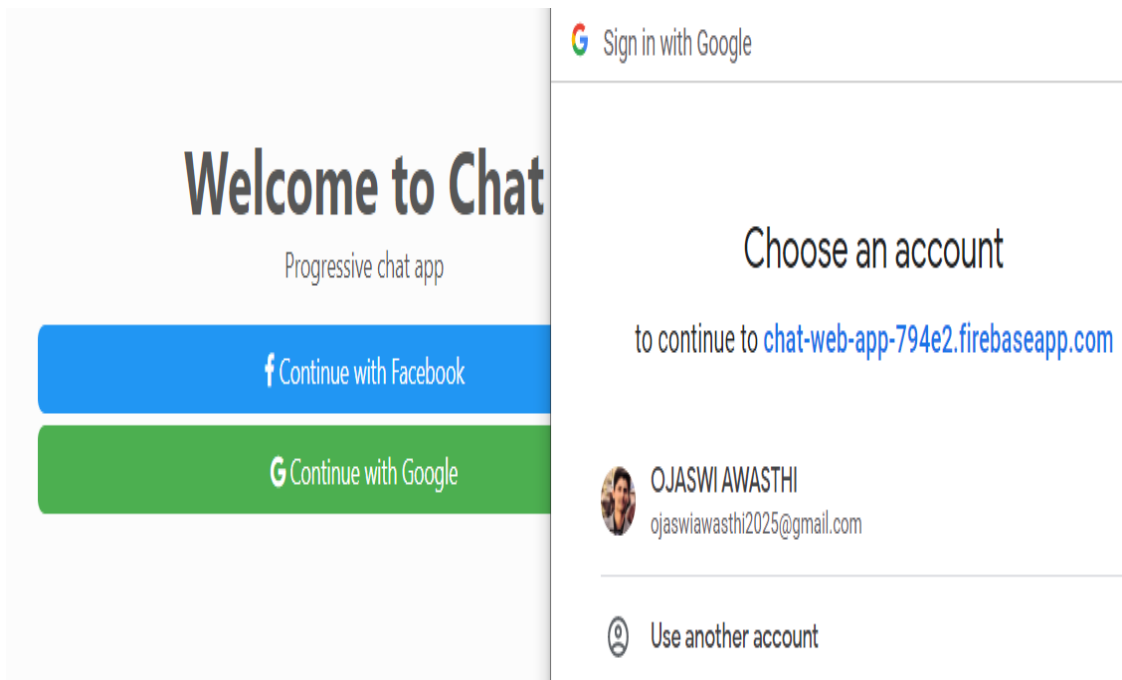
- When user first logs into the app, the Sign in screen pops up for the very first screen(or Splash Screen). In the backend, it checks if the user is already logged in.
- The user is already logged in if there is a record of the user's google or facebook mail and password in the Google Firebase database. The match between the current entered login credentials and those present in the database is made and checked for repetition.
- If the user is already logged in, the user is prompted to choose from the available gmail IDs to log in the app.
- Once the login is successful, the user is redirected to the chat screen where he comes across the dashboard with his details on it. It contains of an Editable Component so that the user can change his profile photo, his nickname or an option to sign out.
- The user can either chat in the existing chat screens or create a new chat-room. Also, only the admin of the page can edit the chat-room information when he wants to.
- The types of files that can be sent across the server include voice messages, text-messages or atmost five files from the device.
- There is a record of every message sent, profile created, chat-room created in the Firebase server. The user has the ability to like a message and his like appears next to the message in the shape of a heart and upon clicking the heart again, the heart disappears.
- The messages appear on the screen according to the date and time when they were sent. Also, the Load More Button at the top of the chat screen will load more the old messages on the screen from the firebase server.
- Rsuite library from React components has helped in building reusable components like the Modal, Drawer, Sidebar, Checkboxes, etc. and props can be easily passed into them.

- React hooks have been used to provide code reusability and make the code less complex. Also, custom hooks can be made.

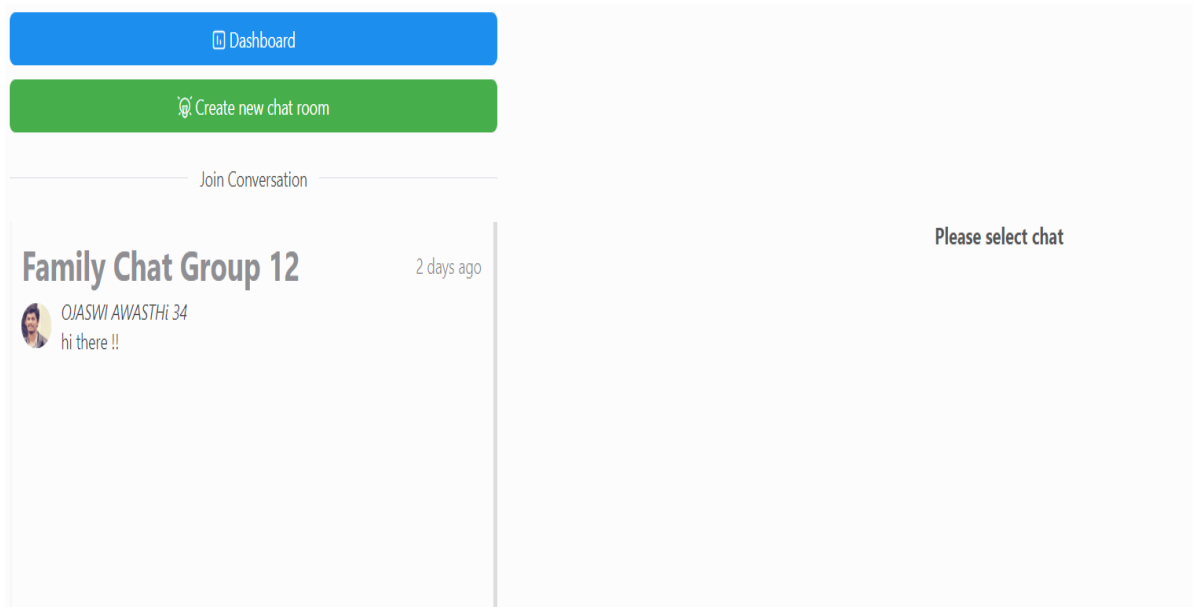
**The SignIn page-** This screen is the first screen shown on the screen before the user is logged in.



Upon clicking on Continue with Google, the sign up Popup appears.



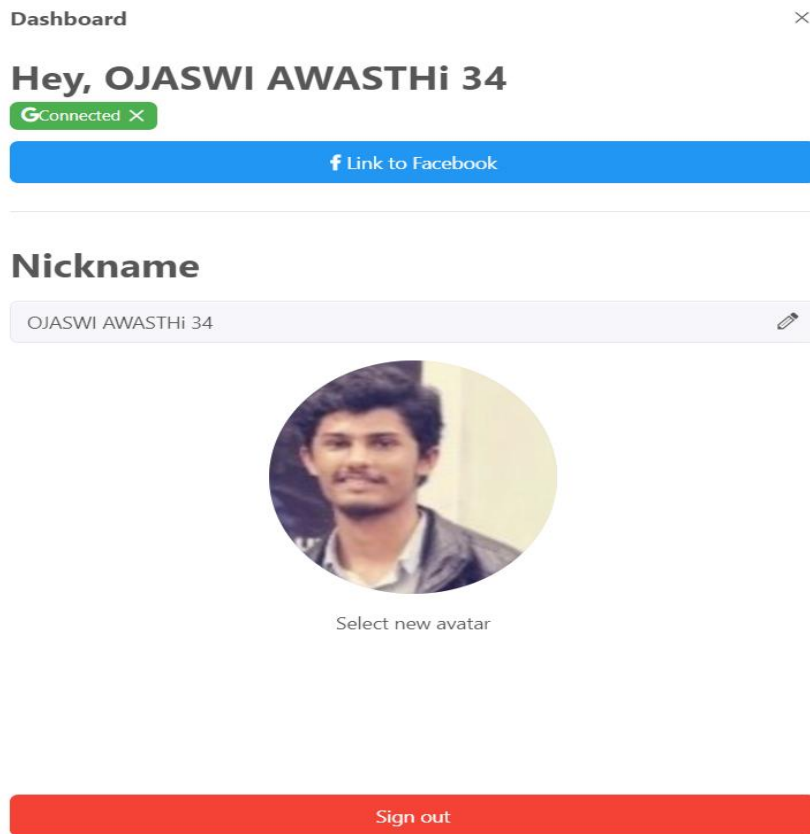
The popup shown on the screen appears on the screen when we click on **Continue With Google**. The Continue With Facebook Button will appear upon clicking on the Facebook button and the popup that appears will be from the Facebook servers instead of the Google ones.



After getting the user logged into the app, the Home Screen appears on the screen which has been divided into the Sidebar and the main Chat.

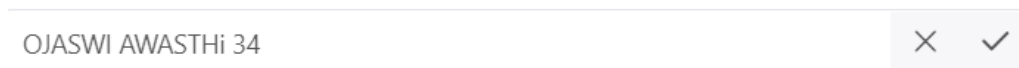
**The Dashboard:**

This is unique for every logged user and contains the user information such as the nickname of the user, the profile image of the user and the Sign Out option. These are all controlled by Context API. The Dashboard is shown below:



The Dashboard

Upon clicking on the **Pencil Icon** next to the name of the logged user, we are shown an Editable Component that replaces the normal text field and we can edit the name if we want to. Similarly, we can upload a new profile image of our liking from our device. This is the Editable Toggle Component.



Upon clicking on the Green button that says **Create a New chatroom**, we are shown a React Modal on the Screen that prompts us to enter information to create a new chat room. This is how it looks like.

**New chat room** X

Room Name

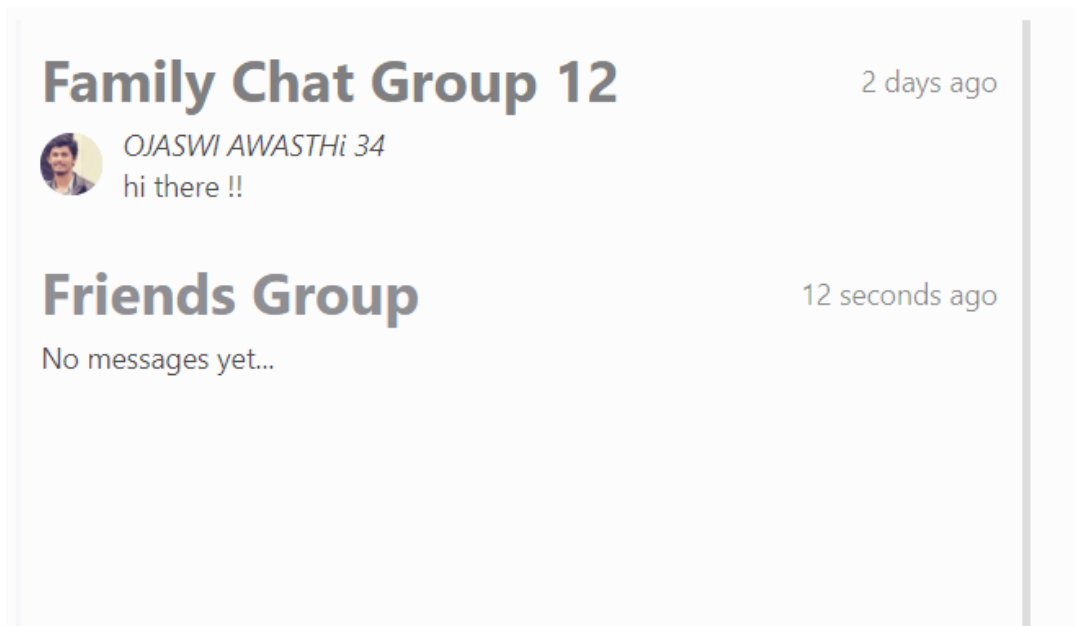
Enter chat room here...

Description

Enter room description...

Create new chat room

Fig 14: The React Suite Modal for creating a new chat room on the server.



The newly created groups appear below the groups that are already in use. In the example above, the First group called “**Family Chat Group 12**” is already in use and the newly created group called “**Friends Group**” appears below the first one. Also, the most recent message will appear in the chat-room div. The time next to the group name shows the time when the group was created.

Upon clicking any of the chats or chat-rooms, we come across the Room Information button that helps us to see the room description and room information. The Room Information for one of the chat-rooms has been described below:

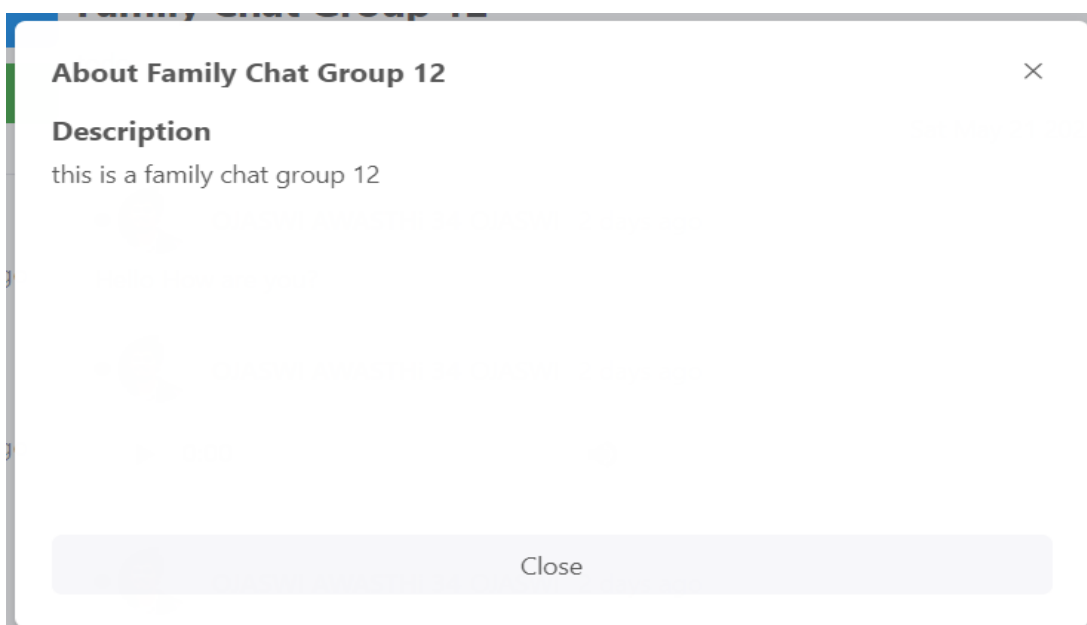
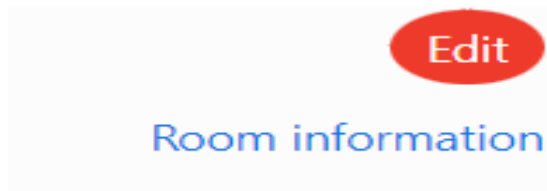


Fig 15: The Room Description Modal that appears on clicking Room Information

The Edit button at the top-right corner of the screen is shown only to the admin of the chat-room and only the admin of the group can edit the room information. The Edit button has been shown below:



The Edit Admin Button

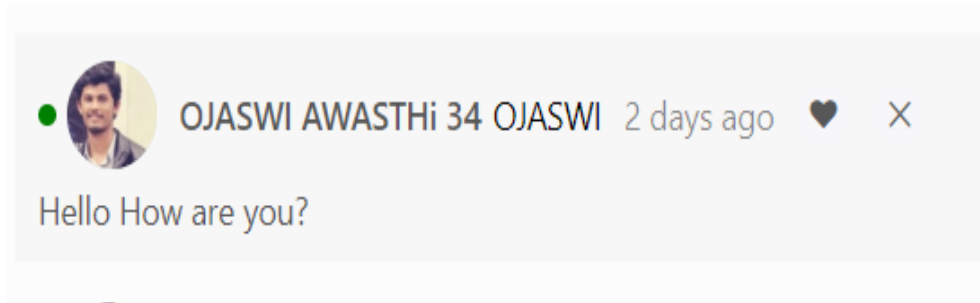
The Edit Chatroom Modal that appears on clicking the red button called Edit is shown below. Only the admin can change the information and not the other people from the chat-room,



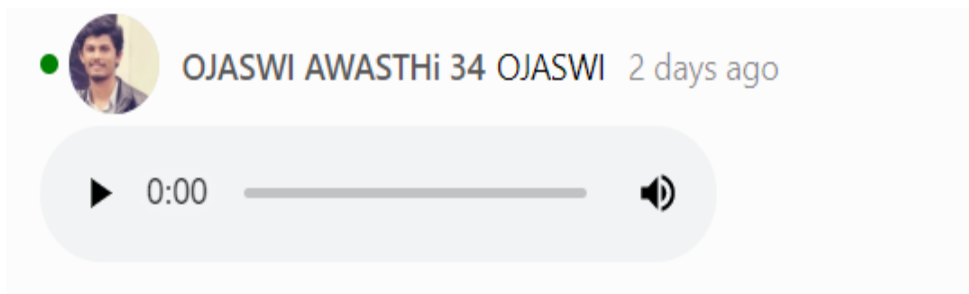
Fig 16: The Edit Room Modal

The ChatScreen has been divided into three types of chats that can be sent on the server, The chats include:

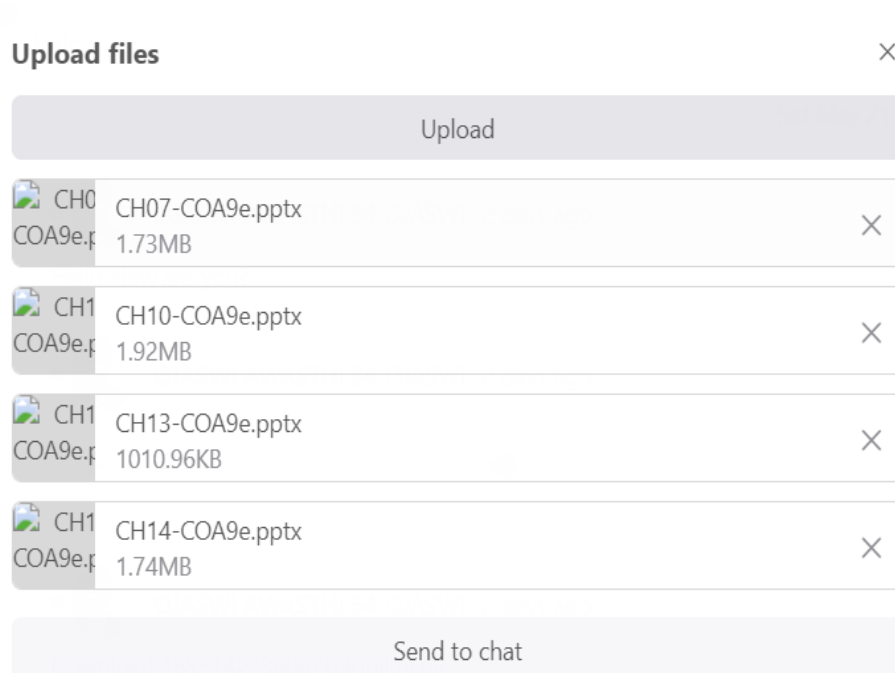
- Text Messages



- Voice Messages



- File Upload: The user can upload up to 5 files on the server from the device,



\*only files less than 5 MB are allowed



The user can only upload files whose size is less than 5 MB. This is an additional check on the number and size of files uploaded on the server.

The complete chat screen with the different voice messages, text messages and the various files uploaded on the server looks something like this when put together.

### The Chat Screen

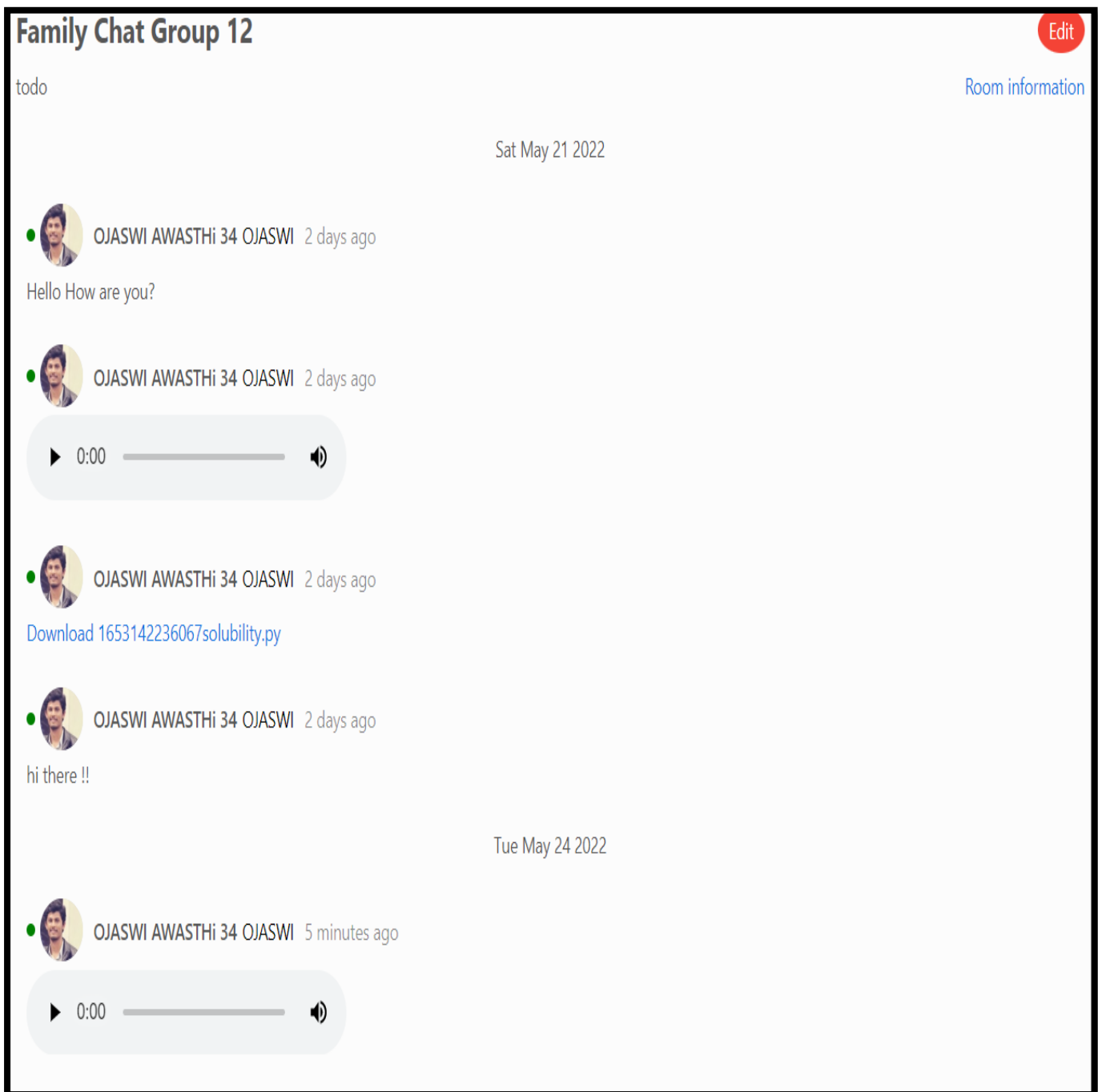
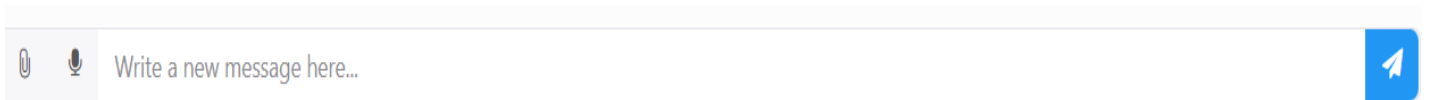


Fig 17: Figure showing the chats organized by date and time with the edit button

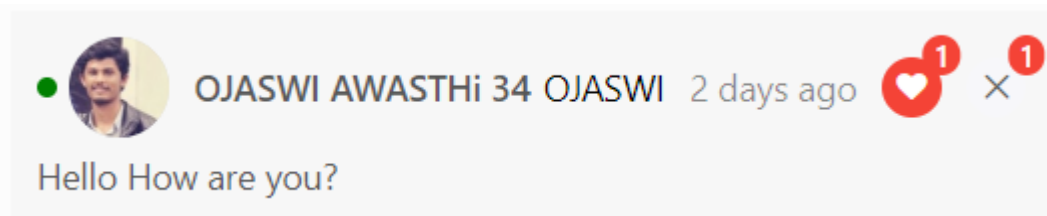
- The messages are sent on the screen with the functionality of the **Send** button.
- The react-router-mic has been used to record messages on the server and then upon toggling it again, the message stops recording and is sent on the server.
- The attachment icon helps to attach files and upload them on the server. They serve as the Chat Bottom Screen.



The chat-bottom component with the mic, attachment icon and send icon

### **The Like Message Functionality:**

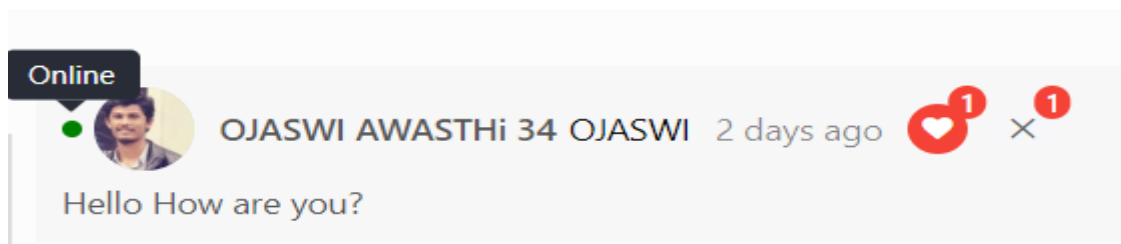
The user can like the chats that he wishes to using a toggle functionality from `React.useState()` function. Upon clicking on the liked message, the like disappears. The like will appear next to the component in the shape of a heart emoticon and upon clicking it again, it disappears.



The like heart appears when the message is liked.

### **Checking whether the user is online/ offline:**

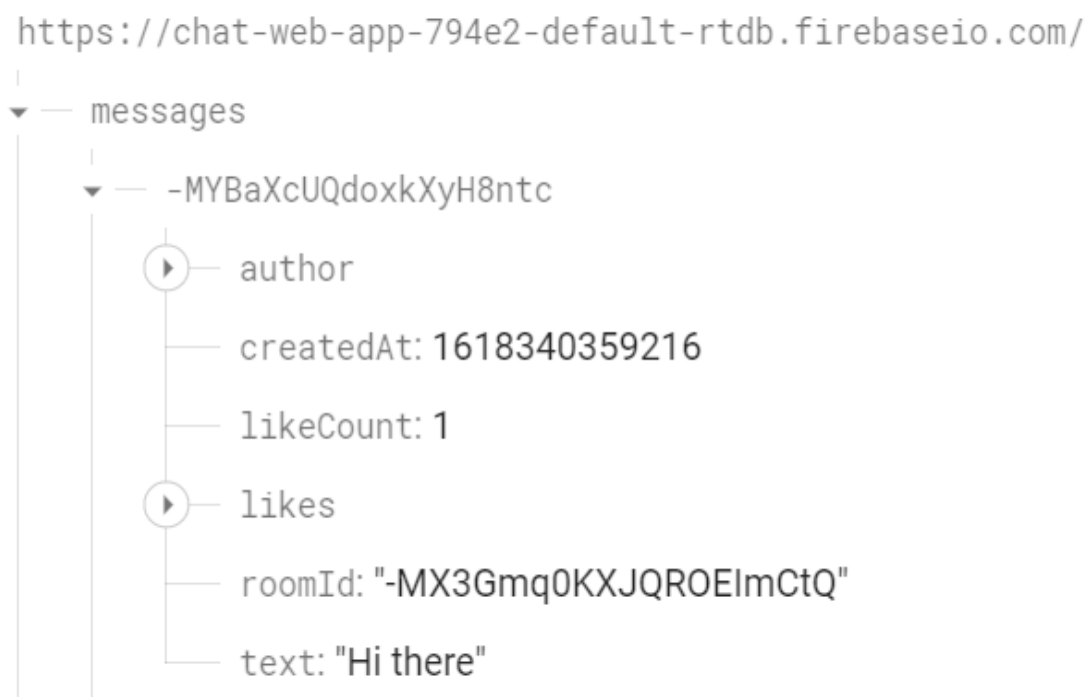
If the user is online, the PresenceDot shows a green icon and if the user is offline, the PresenceDot file shows a red dot instead of that. The Tooltip from React Suite helps us in achieving this state.



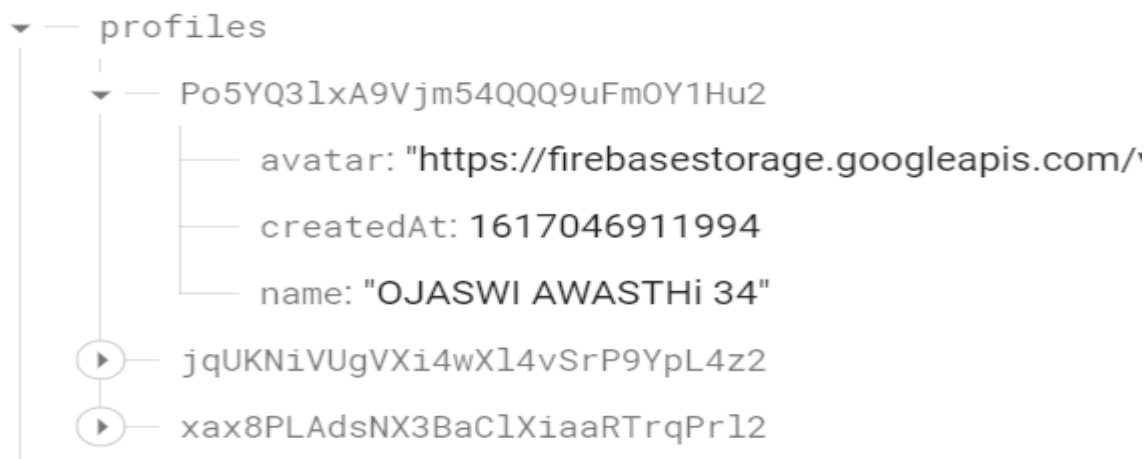
The online tooltip from React

### Firestore database used in the application:

The schema shown below is the one used for messages present in the database. Each message is stored in the database with a unique ID, the name of the author's message, timestamp when the message was created, the value of likeCount which could be 1 or 0 depending whether the message was liked or not. Also, the chat-room ID is shown in the database server inside which a particular message was sent.

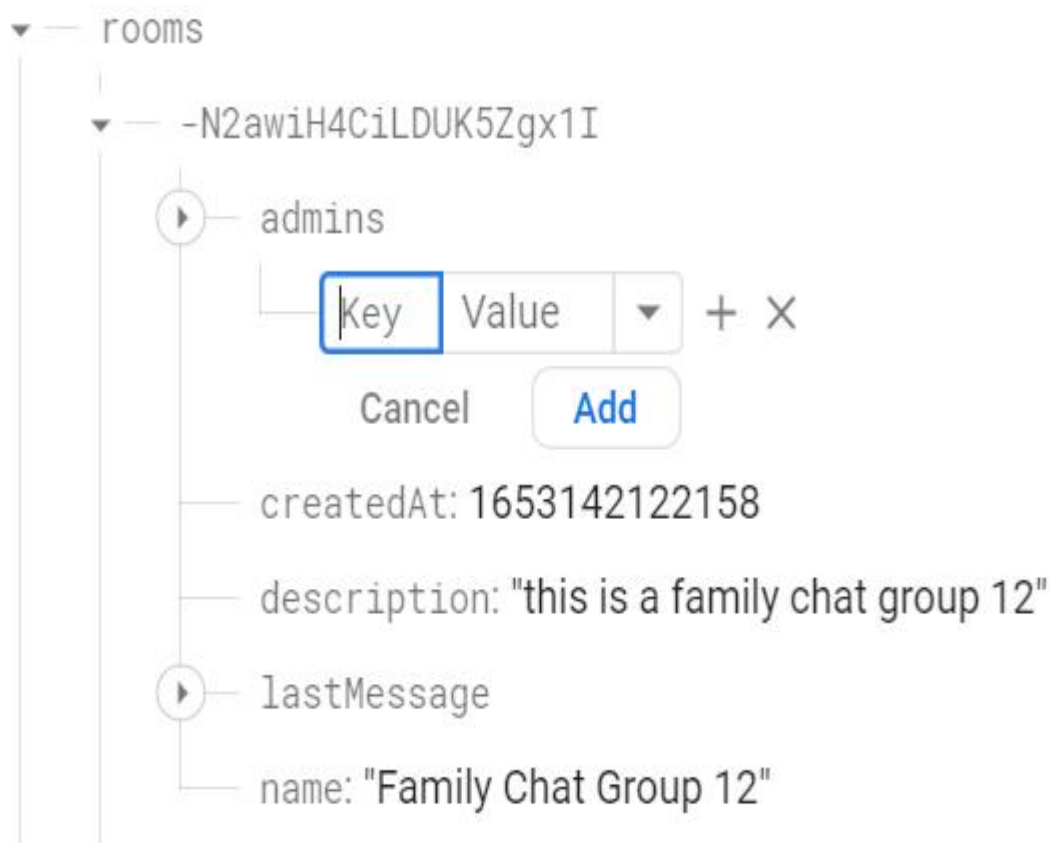


The structure for the **Profiles** Section is shown below. Each profile is divided into the profile image, the timestamp when the image was created and the name of the logged user which gets picked up from the person's Google Account.



The chat-rooms structure is shown below:

Each chat-room will contain a key-value pair for the admin of the group, the timestamp when the group was created, the chat-room description and the most recent message that was sent in the chatroom.



The fourth structure shown in the firebase server is the Status of the logged user. The last\_changed and the state is stored in the server which stores the information whether the user is online or offline.



### Hooks and Custom Hooks:

Hooks are an incredibly powerful feature in React that was introduced in React 16.8. The most important feature of hooks is to write state, other functions and `setState()` without officially declaring or writing code inside a class function.

An important example of hooks has been shown below:

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

In the example above: we declare a count variable using the **useState()** hooks in React. The `setCount()` is a function that will update the value of count variable every time the button **Click**

Me is clicked. The count is initially 0 but gets incremented when the button is clicked. The process of using hooks is much simpler than using the state in class. We do not need the **this operator** or the bind() function to bind the state to our variables while using functional components.

Uses of React Hooks have been explained below:

- It becomes incredibly complex to use stateful logical code across different components.
- Also, all complex components that use the state and setState() functions become difficult to understand for even the developer sometimes.
- Hooks can be beneficial in break a bigger component into smaller chunks of code for better understanding.

Firestore Login Success occurs when the authUser succeeds in authenticating the user's login credentials with any set of credentials present in the database, otherwise there occurs a Firestore error. The figure below shows exactly that.

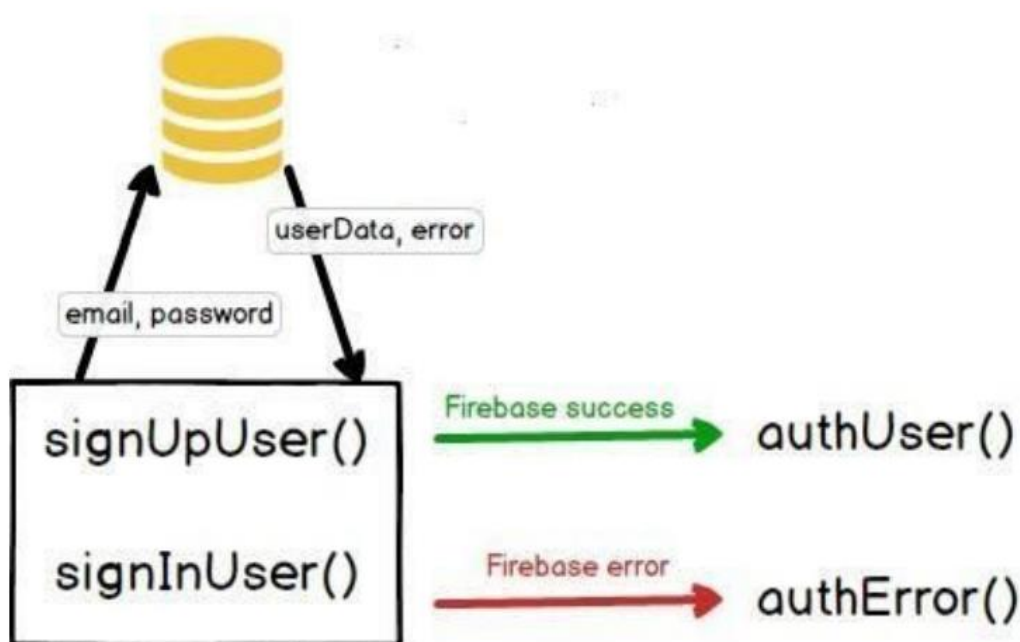


Fig 18: Firestore success and Firestore Error depicted in these figures

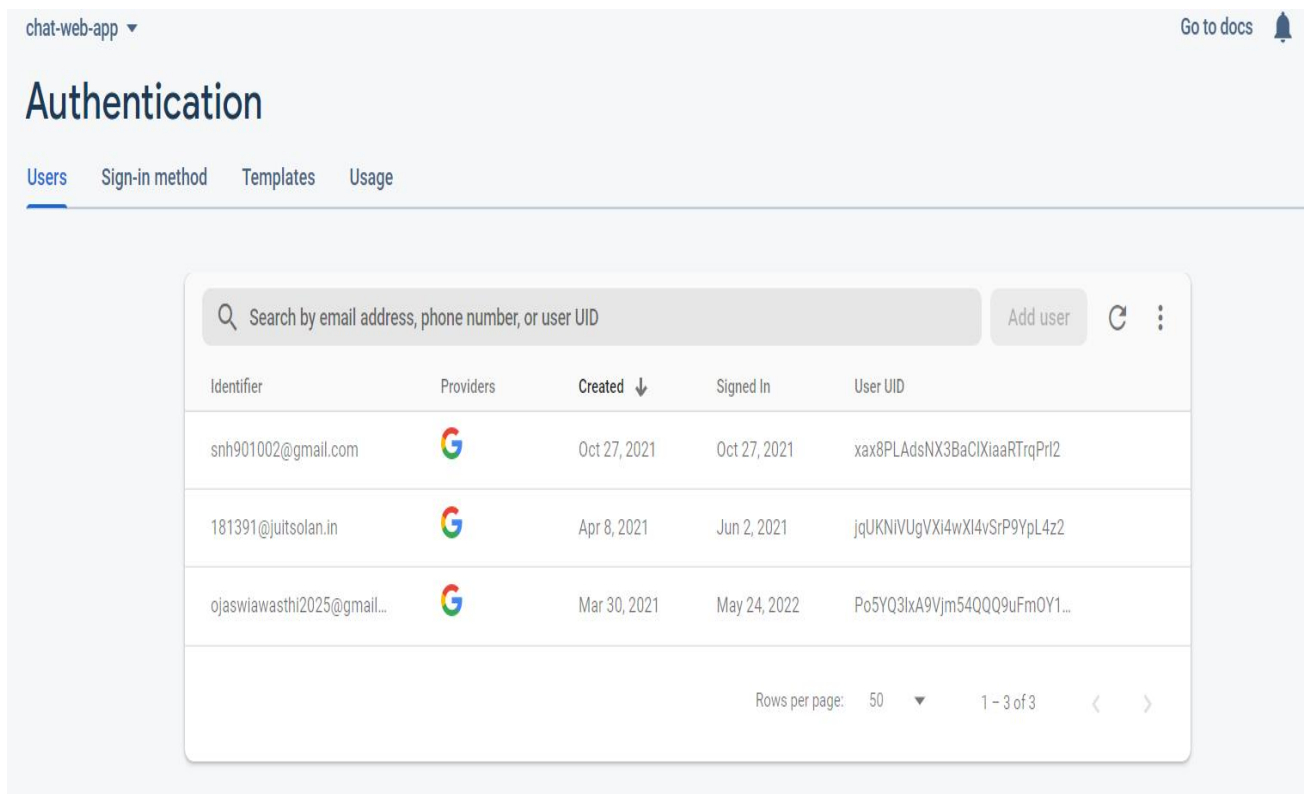


Fig 19: Figure showing the number of users logged into the app

### Hosting of the application:

The full-fledged version of the application has been made public on Github on the public repository:

<https://github.com/Ojaswi2000/chat-app>

The aforementioned repository contains all the necessary files and folders that have been used in the building the application. To run the application on your local system, we need to first clone the repository using the **git clone repository\_name** command and then to install all the used dependencies in the project , use the **npm init** command to install the node\_modules in the project. The dependencies will take some time for installation depending upon the internet speed.

The snapshot below shows all the files pushed onto github repository for the given project.

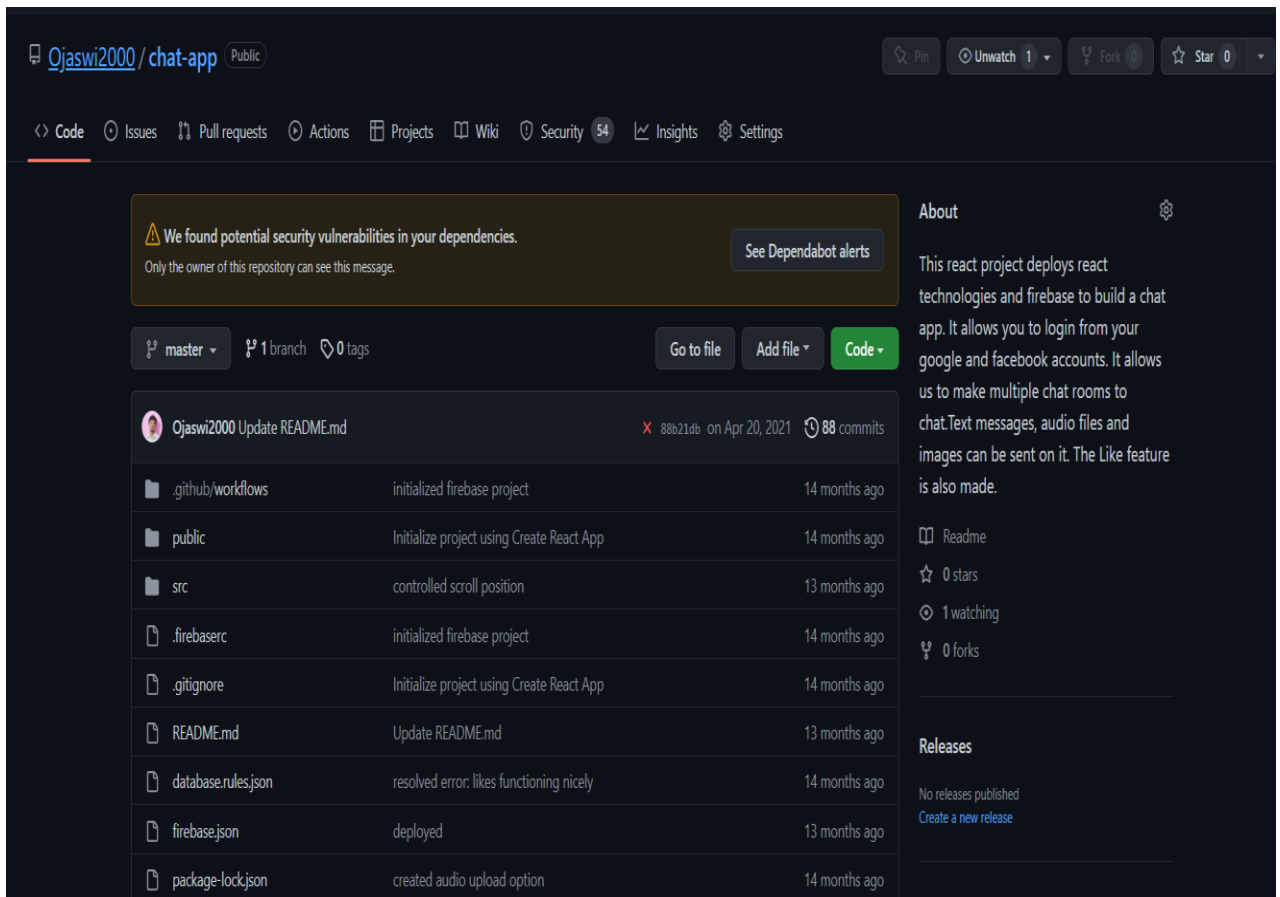


Fig 20: The github project repository for chat-web-app



## CHAPTER 5: CONCLUSIONS

**5.1** The sole aim of the project is to understand the concept and basics of MERN stack web development application by building an advanced chatting application. The way it is different from the other applications is in the following ways:

- The internship project focuses on building a chatting application in MERN stack using the frontend Javascript technology called React and Express and Node JS for the backend for building of client and server part. Google Firebase is a free backend service that stores the login credentials of the user and the chatroom information.
- This app also gives primary focus to the admin and security permissions to the users. Only, the admin of the chatroom has the permission to edit the name of the chatroom and the chatroom description.
- The project aims to be more than just a chatting application as it allows us to send more than just chats in the server. We can upload upto 5 files in one go. The Message File has the attachment icon to attach files and the send button to finally send the files over on the server.
- The user can login the app using the authenticator provided by the Google Firebase. Also, when the user signs out from the app, the unsubscribe() function allows him to be unsubscribed from the database, The timestamp is also added which keeps a record of when the user logged in to the app.
- The user can like certain chats in the chat window and can also remove those likes. A heart emoticon will show up when the user likes the message. This is made possible through the concept of post transactions in React.

There is still a lot of backend that needs to be incorporated into making the app faster and more reliable than other chat applications in the market. For example, the video calling feature needs to be incorporated for multiple people to video chat at the same time. Also, chat reactions, dark mode for the entire app are some of the features that need to be coded in the app.

## 5.2 Future Scope:

- There is still a lot of backend work to be done in order to make the app faster and more dependable than other chat apps available. For example, if numerous individuals want to video chat at the same time, the video calling feature must be included.
- Chat reactions, as well as a dark mode for the full app, are some of the features that must be coded and incorporated.
- Animation Libraries such as Framer Motion have to be used to provide UI strength and more unique design to the app,
- Alongwith Firebase database, the app needs to be tested on other NOSQL platforms such as MongoDB Atlas and NOSQL Booster to see if is compatible with other platforms,
- The app needs to load faster. Hence, useMemo and other hooks have to be used to give the app speed and reduce the DOM complexity while loading components on the screen and putting them together.

## 5.3 Applications:

- The most basic application of this app is to be used as a chatting app and then later as a video chat application.
- The Like Message Functionality enables us to like or dislike messages,
- Chats are organized according to their date and time and prevent the user from getting all confused regarding when the chat was actually sent.
- The chat application can be hosted on Google Playstore from where users can download it in Native environment and use it accordingly.

## REFERENCES

1. Buddhini Gayathri Jayatilleke, Gaya R. Ranawaka and Chamali Wijesekera ,Malinda C.B. Kumarasinha, “*Development of mobile application through design-based research*”, Asian Association of Open Universities Journal Vol. 13 No. 2, 2018
2. H. K. Flora, X. Wang, and S. V.Chande, “*An Investigation into Mobile Application Development Processes: Challenges and Best Practices*”, Int. J. Mod. Educ. Comput. Sci., vol. 6, no. 6, pp. 1–9, 2014
3. R. Colomo-Palacios, J. Calvo-Manzano, A. De Amescua, and T. San Feliu, “*Agile Estimation Techniques and Innovative Approaches to SW Process Improvement*”. 2014.
4. H. K. Flora, X. Wang, and S. V.Chande, “*An Investigation into Mobile Application Development Processes: Challenges and Best Practices*,” Int. J. Mod. Educ. Comput. Sci., vol. 6, no. 6, pp. 1–9, 2014
5. R. Alkadhi, T. Lata, E. Guzman, and B. Bruegge. Rationale in Development Chat Messages: An Exploratory Study. In *Proc. of the 14th Work. Conf. on Min. Softw. Repos.*, MSR'17, pages 436-446, 2017.
6. V. Rahimian and R. Ramsin, “*Designing an Agile Methodology for Mobile SW Development : A Hybrid Method Engineering Approach*,” pp. 351– 356, 2007.
7. M. Stoica, M. Mircea, and B. Ghilic-Micu, “*Software development: Agile vs. traditional*,” Inform. Econ., vol. 17, no. 4, pp. 64–76, 2013.
8. W. McIver, “*Software Engineering Processes for Mobile Applications Development*,” NSERC Mob. First, Frederict., vol. 1, no. 506, pp. 1–74, 2015.

## APPENDICES

Code in the application:

Returning the **SignIn** component

```
<Container>
  <Grid className="mt-page">
    <Row>
      <Col xs={24} md={12} mdOffset={6}>
        <Panel>
          <div className="text-center">
            <h2>Welcome to Chat</h2>
            <p>Progressive chat app</p>
          </div>

          <div className="mt-3">
            <Button block color="blue" onClick={onFaceBookSignIn} >
              <Icon icon="facebook" /> Continue with Facebook
            </Button>

            <Button block color="green" onClick={onGoogleSignIn}>
              <Icon icon="google" /> Continue with Google
            </Button>
          </div>
        </Panel>
      </Col>
    </Row>
  </Grid>
</Container>
```

Returning the **Context API** for Chat-Room

```
const RoomsContext = createContext();

export const RoomsRovider = ({children}) => {

  const [rooms,setRooms] = useState(null);
  useEffect(() => {

    const roomsListRef = database.ref('rooms');

    roomsListRef.on('value', (snap)=>{
      const data= tranformToArrayWithId(snap.val());
      console.log('data',data);
      setRooms(data);
    })

    return ()=>{
      roomsListRef.off();
    }
  })
}
```

Status to show whether the user is online or offline

```
const getText= (presence) =>{
  if(!presence){
    return "Unknown state";
  }
  return presence.state === 'online'?
  "Online" :
  `Last online ${new Date(presence.last_changed).toLocaleDateString()}`;
}

const PresenceDot = ({uid}) => {
  const presence = usePresence(uid);

  return (
    <Whisper placement="top" trigger="hover" speaker={
      <Tooltip>
        {getText(presence)}
      </Tooltip>
    }>
    <Badge className="cursor-pointer" style={{backgroundColor:getColor(presence)}} />
  </Whisper>
  )
}
```

The Sidebar Component which gets shown across all the components

```
const topSidebarRef = useRef();
const [height, setHeight]= useState(0);

useEffect(() => {
  if(topSidebarRef.current){
    setHeight(topSidebarRef.current.scrollHeight);
  }
}, [topSidebarRef]);

return (
  <div className="h-100 pt-2">
    <div ref={topSidebarRef}>
      <DashboardToggle />
      <CreateRoomBtnModal />
      <Divider>Join Conversation</Divider>
    </div>

    <ChatRoomList aboveElHeight={height} />
  </div>
)
```