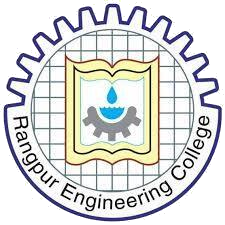
****

**SWEETCHAT:**

**A WEB BASED REAL TIME MESSAGING APPLICATION**

**MINOR PROJECT REPORT**

**Submitted in partial fulfilment of the requirements for the award of**

**the degree of**

**BACHELOR OF SCIENCE**

**In**

**COMPUTER SCIENCE & ENGINEERING**

**By**

|  |  |  |
| --- | --- | --- |
| **Md. Mahfujar Rahman**  **Roll No : 1737620108** |  | **Shitol Chandro Sarker**  **Roll No : 1737620122** |
|  |  |  |
| **Md. Nasir Bin Mahabub**  **Roll No : 1737620116** |  | **Mostasim Billah**  **Roll No : 1737620130** |

Supervised By

**Department of Computer Science and Engineering**

**Rangpur Engineering College, Rangpur**

**(Affiliated by University Of Rajshahi)**

**CANDIDATES’ DECLARATION**

It is hereby certified that the work which is being presented in the B.Sc Engg. Project Report entitled **“SWEETCHAT: A WEB BASED REAL TIME MESSAGING APPLICATION”** in partial fulfilment of the requirements for the award of the degree of **B.Sc Engineering** and submitted in the **Department of Computer Science & Engineering** of **Rangpur Engineering College, Rangpur (Affiliated by University Of Rajshahi)** is an authentic record of our own work carried out during a period from **December, 2021 to March 2022** under the guidance of **Md. Khalid Hasan, Lecturer(CSE).**

The matter presented in the B.Sc Engineering Project Report has not been submitted by us for the award of any other degree of this or any other Institute.

|  |  |  |
| --- | --- | --- |
| **Md. Mahfujar Rahman**  **Roll No : 1737620108** |  | **Shitol Chandro Sarker**  **Roll No : 1737620122** |
|  |  |  |
| **Md. Nasir Bin Mahabub**  **Roll No : 1737620116** |  | **Mostasim Billah**  **Roll No : 1737620130** |

This is to certify that the above statement made by the candidate is correct to the best of my knowledge. They are permitted to appear in the External Major Project Examination

|  |  |  |
| --- | --- | --- |
| **Md. Khalid Hasan**  **Lecturer (CSE)**  Project Supervisor |  | **(Signature of External Examiner)** |

**ACKNOWLEGEMENT**

It is a pleasure acknowledging with deep sense of gratitude the invaluable help and guidance the invaluable help and guidance rendered by Supervisor and Faculty of our Institute for the successful completion of this project report.

We express our deep gratitude to **Md. Khalid Hasan,** Lecturer, Department of Computer Science & Engineering for his valuable guidance and suggestion throughout my project work.

who never hesitated in extending their sincere cooperation from time to time to prepare an interesting and challenging project.

We would also like to thank our families and friends, who helped us in every possible way.

|  |  |  |
| --- | --- | --- |
| **Md. Mahfujar Rahman**  **Roll No : 1737620108** |  | **Shitol Chandro Sarker**  **Roll No : 1737620122** |
|  |  |  |
| **Md. Nasir Bin Mahabub**  **Roll No : 1737620116** |  | **Mostasim Billah**  **Roll No : 1737620130** |

**ABSTRACT**

Nowadays, Mobile/Desktop Messaging Applications have become a popular way to communicate with others and play an important role in our life. But web based messaging applications isn’t as much popular as mobile/desktop based messaging applications like WeChat, Messenger, WhatsApp, etc. There are some web based messaging solutions on online, but almost all are from social media websites. And also, their user interface and functionalities tends to be a bit complicated. This project is to develop a web based messaging application that enables the users to communicate with each other with very simple UI and functionalities.

**TABLE OF CONTENT**

**CANDIDATES’ DECLARATION............................................................................ ii**

**ACKNOWLEGEMENT............................................................................................ iii**

**ABSTRACT ................................................................................................................ iv**

**TABLE OF CONTENT…….................................................................................. v-vi**

**LIST OF FIGURES…….......................................................................................... vii**

**LIST OF TABLES……………………………........................................................ vii**

**CHAPTER 1: INTRODUCTION ......................................................................... 8-13**

1.0 Introduction ............................................................................................................. 8

1.1 Problem Statement .................................................................................................. 8

1.2 Innovative Ideas of Project ..................................................................................... 9

1.3 Project Objective ..................................................................................................... 9

1.4 Scope of The Project ............................................................................................... 9

1.5 What is MongoDB? ................................................................................................. 9

1.5.1 Key Components of MongoDB Architecture ....................................................... 9

1.6 What is Express.js? ................................................................................................ 10

1.6.1 Why use Express? ............................................................................................... 10

1.7 What is React? ....................................................................................................... 10

1.7.1 Why use React? .................................................................................................. 10

1.8 What is Node.js? .................................................................................................... 11

1.8.1 Features of Node.js ............................................................................................. 11

**CHAPTER 2: PRODUCT OVERVIEW ................................................................. 12**

2.0 Users and Stakeholders .......................................................................................... 12

1.1 Project Perspective: ............................................................................................... 12

1.2 Interface: ............................................................................................................... 12

1.3 Functional and Non-Functional Requirements: - ................................................... 13

1.3.1 Functional Requirements .................................................................................... 13

1.3.2 Non-Functional Requirements ........................................................................... 13

1.4 Use Case Table ...................................................................................................... 13

**CHAPTER 3: ARCHITECTURE ........................................................................... 14**

3.0 Chat Architecture: How We Approach It ............................................................... 14

3.1 Chat App or Client Side ......................................................................................... 15

3.2 Chat Server Engine ................................................................................................ 16

**CHAPTER 4: SCREENSHOTS .............................................................................. 17**

**CONCLUSION ......................................................................................................... 19**

**REFERENCES .......................................................................................................... 20**

**LIST OF FIGURES**

FIGURE 1.1 MERN STACK ........................................................................................ 8

FIGURE 2.1 GUI OF APPLICATION ........................................................................ 12

FIGURE 3.1 ARCHITECTURE ................................................................................ 14

FIGURE 3.2 CLIENT-SIDE STRUCTURE TREE ................................................... 15

FIGURE 3.3 PACKAGE.JSON FILE OF CLIENT-SIDE ........................................ 15

FIGURE 3.4 SERVER-SIDE STRUCTURE TREE .................................................. 16

FIGURE 3.5 DEPENDENCIES FOR SERVER ........................................................ 16

FIGURE 4.1 STARTING GUI AND LOGIN/SIGNUP ............................................ 17

FIGURE 4.2 ADDING USER IN GROUP CHAT/ PERSONAL CHAT ................. 17

FIGURE 4.3 CHATTING INTERFACE ................................................................... 18

**LIST OF TABLES**

TABLE 1:USERS AND STAKEHOLDER ............................................................... 12

TABLE 2: USE CASE ................................................................................................ 13

**CHAPTER 1: INTRODUCTION**

1. **Introduction**

Today Developers around the world are making efforts to enhance user experience of using application as well as to enhance the developer’s workflow of designing applications to deliver projects and rollout change requests under strict timeline. Stacks can be used to build web applications in the shortest span of time. The stacks used in web development are basically the response of software engineers to current demands. They have essentially adopted pre-existing frameworks (including JavaScript) to make their lives easier.

**1.1 Problem Statement**

* This project is to create a chat application to enable the users to chat with each other’s.
* To develop an instant messaging solution to enable users to seamlessly communicate with each other.
* The project should be very easy to use enabling even a novice person to use it.
* This project can play an important role in social field can connect through internet.
* The main purpose of this project is to provide easy way of chatting functionality through network.

**1.2 Innovative Ideas of Project**

* **GUI:** Easy to use GUI (Graphical User Interface), hence any user with minimal knowledge of operating a system can use the software.
* **Platform independence:** The messenger operates on any system irrelevant of the underlying operating system.
* **One Click Authentication:** A easy login system called one click authentication through Gmail.
* **Unlimited clients:** “N” number of users can be connected without any performance degradation of the server.

**1.3 Project Objective**

* Communication: To develop an instant messaging solution to enable users to seamlessly communicate with each other.
* User friendliness: The project should be very easy to use enabling even a novice person to use it.

**1.4 Scope of The Project**

* Broadcasting Chat Server Application is going to be a text communication software, it will be able to communicate between two computers using point to point communication.
* The limitation of Live Chat is it does not support audio conversations. To overcome this limitation, we are concurrently working on developing better technologies.
* Companies would like to have a communication software wherein they can communicate instantly within their organization.
* The fact that the software uses an internal network setup within the organization makes it very secure from outside attacks.

**1.5 What is Firebase[1]?**

* Firebase is a cross-platform document-oriented NoSQL database used for high volume data storage that provides high performance, high availability and easy scalability.
* Firebase stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in the application code, making data easy to work with.
* The data model available within Firebase allows users to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
* Firebase works on concept of collections and documents. Each database contains collections which in turn contains documents. Each document can have varying number of fields. The size and content of each document can also be different from each other.

**1.5.1 Key Components of Firebase Architecture**

**1. Authentication**– Firebase supports multiple type of authentication system. Such as Providers(Google, Facebook, Twitter, etc), Email & Password, Anonymous, Phone Number Authentication, etc.

**2. id** – This is a 20-digit unique identifier field required in every Firebase document in the collection. The \_id field is like the document's primary key. If the user creates a new document without an \_id field, Firebase will automatically create the field.

**3. Collection** - Collection is a group of Firebase documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Typically, all documents in a collection are of similar or related purpose.

**4. Storage** – Cloud storage for media files and other, it stores the media files and provide a link of the files. Using Type fetch, the file can be access through URL of the media.

**5. Firestore Database** - Database is a physical container for collections. Each database gets its own set of files on the file system. A single Firebase server typically has multiple databases.

**6. Field** - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

**1.6 What is Next.js[2]?**

* Next is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is an open source framework developed and maintained by the Node.js foundation.
* Next provides us the tools that are required to build our app, be it single-page, multi-page or hybrid web applications. It is flexible as there are numerous modules available on npm(Node Package Manager), which can be directly plugged into Express.
* Unlike its competitors like Rails and Django, which have an opinionated way of building applications, Next has no "best way" to do something. It is very flexible and pluggable.
* Pug (earlier known as Jade) is a terse language for writing HTML templates. It produces HTML, supports dynamic code and code reusability (DRY). It is one of the most popular template languages used with Next.
* Next can be thought of as a layer built on the top of the Node.js that helps manage a server and routes. It allows users to setup middleware to respond to HTTP Requests and defines a routing table which is used to perform different actions based on HTTP method and URL.
* Next allows to dynamically render HTML Pages based on passing arguments to templates.
* Next is asynchronous and single threaded and performs I/O operations quickly.

**1.6.1 Why use Next?**

* Ultra-fast I/O.
* Asynchronous and single threaded.
* MVC like structure.
* Robust API makes routing easy.

**1.7 What is React[3]?**

* ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front-end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.
* A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.
* Instead of using regular JavaScript, React codes are written in something called JSX (JavaScript Syntax Extension). JSX is basically a syntax extension of regular JavaScript and is used to create React elements. These elements are then rendered to the React DOM. JSX is faster than normal JavaScript as it performs optimizations while translating to regular JavaScript.

**1.7.1 Why use React[4]?**

* Uses virtual DOM which is a JavaScript object. This will improve apps performance, since JavaScript virtual DOM is faster than the regular DOM.
* Can be used on client and server side as well as with other frameworks.
* Component and data patterns improve readability, which helps to maintain larger apps.

**1.8 What is Node.js[5]?**

* Node.js is a very powerful JavaScript-based platform built on Google Chrome's JavaScript V8 Engine. It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications. Node.js is open source, completely free, and used by thousands of developers around the world.
* Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009.
* Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.
* Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

**1.8.1 Features of Node.js**

**1. Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.

**2. I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So, a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.

**3. Single threaded:** Node.js follows a single threaded model with event looping.

**4. Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.

**5. No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.

**6. Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js application.

**CHAPTER 2: EXISTING SYSTEMS AND LITERATURE REVIEW**

1. **Introduction**

A messaging system is responsible for transferring data from one application to another so the applications can focus on data without getting bogged down on data transmission and sharing. Distributed messaging is based on the concept of reliable message queuing.

* 1. **Real Time Messaging**

Real time Messaging refers to the distribution and delivery of messages that are designed to be consumed or otherwise used in real time (i.e. as events occur and no later).

Real time messaging on the web typically involves a system in which [data is streamed or pushed to users](https://www.pubnub.com/learn/glossary/what-is-real-time-data/), applications, or devices with an appropriate real time delivery mechanism such as [Publish/Subscribe](https://www.pubnub.com/learn/glossary/what-is-publish-subscribe/). The concept can also be compared with something as simple as a [webhook](https://www.pubnub.com/learn/glossary/what-is-a-webhook/" \t "_blank), which provides the ability to push updates out from an app to external URLs as they occur.

Importantly, it's not only humans that benefit from real time messaging (e.g. social media), but it is also essential for autonomous, distributed, or "smart" devices that must work in tandem to achieve a common goal. Environmental control, industrial process monitoring, smart infrastructure and many other technologies make use of real time messaging to:

* Drive processes that depend on intrinsically unpredictable events.
* Ensure logical / situational awareness of the present state of a system.
* Provide a synchronous overview of different parts of a larger environment.
  1. **Existing Systems**

There are some applications on internet almost like our proposed project. Such as:

* **Messenger:** Facebook Messenger is owned by “Meta”. It’s connected to social media website “Facebook”. Initially released in 2008 as “Facebook Chat” & revamped its messaging service in 2010.
* **WhatsApp:** WhatsApp is also owned by “Meta”. It was initially released by WhatsApp Inc. in 2009. It allows users to send text messages, voice messages, share files and VoIP calling.
* **WeChat:** The messaging app “WeChat” was “Weixin” and developed by Tencent. It was initially released in 2011. Weixin was re-branded as “WeChat” in 2012 after reaching 100 million users.
  1. **Drawback Of Existing Work**

It’s known that all of those are full functional and highly developed apps. But to

us, those has some drawbacks in some features.

* **Messenger:** First of all, it’s from social media website “Facebook”. If we want to send instant message to someone through Messenger, we need to create account with Facebook. Which is actually fairly time consuming and a bit of a hassle. To create account, we need to go to social media “Facebook”, then need to sign up with phone number or email, and also verify phone number or email address. After that we can login in Messenger to message someone.

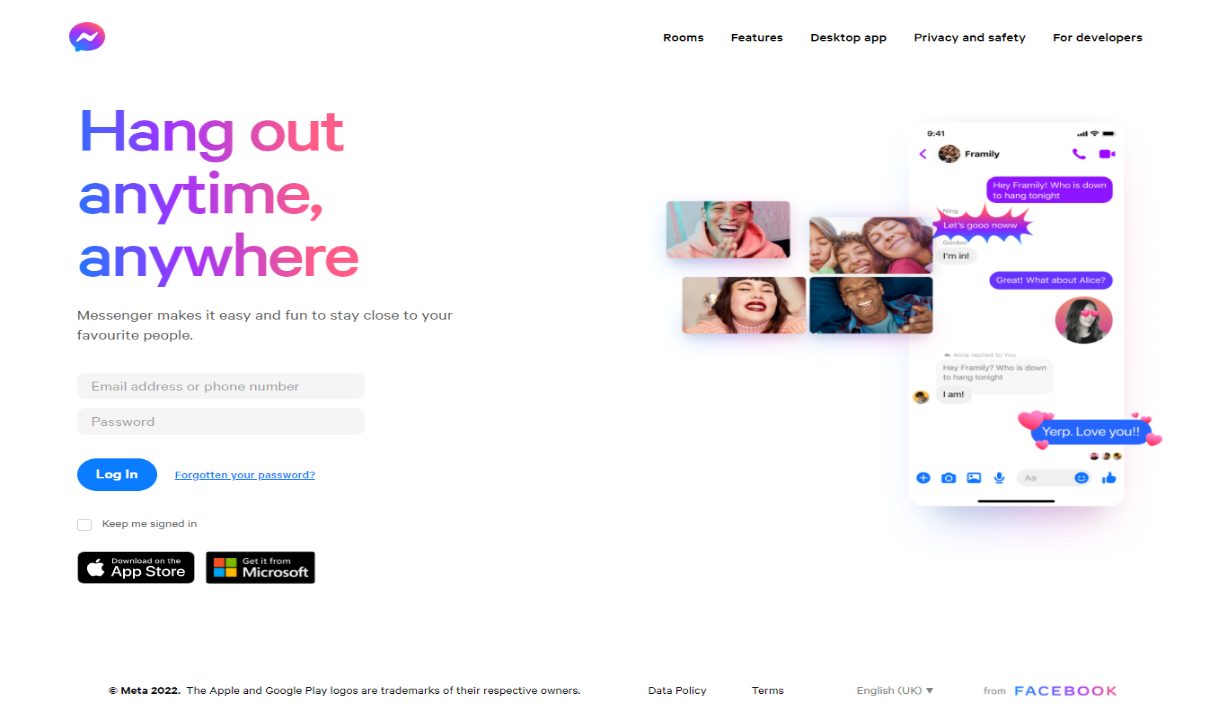


Figure 2. 1 Messenger Web

* **WhatsApp:** It’s basically a mobile based messaging application, not web based. Though it has a web based version, but we can’t use “WhatsApp Web” without WhatsApp mobile application. To use WhatsApp Web, we need WhatsApp Mobile application, Login to Mobile Application, and then scan QR code. Which is very time consuming and a hassle. Furthermore, you can’t be able to create WhatsApp account in Web, you must need WhatsApp mobile application and a phone number.

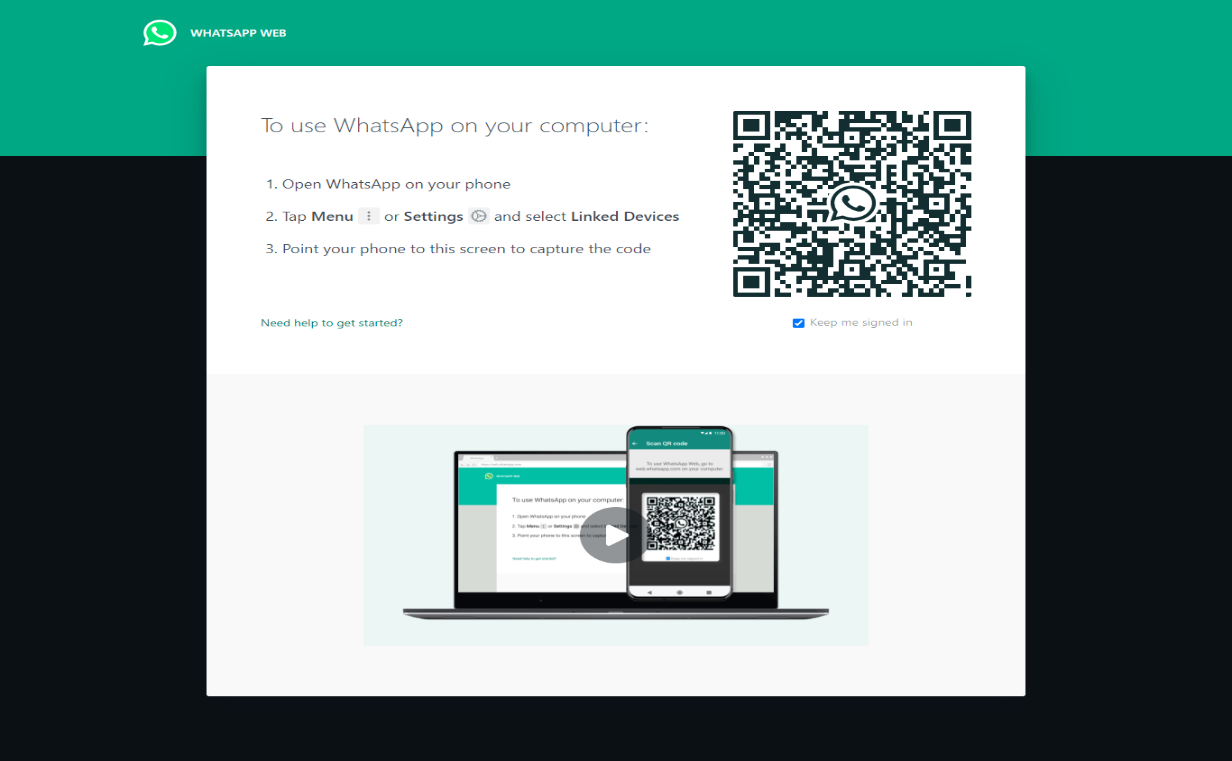


Figure 2. 2 WhatsApp Web

* **WeChat:** It’s also similar to WhatsApp. It’s also mobile based, we need WeChat pre-installed mobile and pre login to WeChat mobile application to use Web Version.



Figure 2. 3 WeChat Web

**CHAPTER 3: REQUIREMENT SPECIFICATION**

1. **Business Process Modeling**

Business process modeling (BPM) is a modern process and methodology. Which the represented the activity of an enterprise of a system engineering to improve or analysis the current process. In this process one can easily represent their workflow of a system. The main characteristic of the methodology is based on diagram as ‘Flow Diagram’. Here we are trying to describe our project’s business model using data flow diagram. Data flow diagram describes how data is processed through a system or project [8]. Data flow diagram is one of the most useable Diagram to show the work flow of a system. It’s easy and understand to any workflow. We used level 1data flow diagram for our work. Figure 3.1 shows the data flow diagram of the propose system.

Already Logged In?

(Check Session)

Yes

No

One Click Login/Signup

Is Successful?

No

User Account

Private Chat

Add Contacts

Log Out

*Figure 3.1: Data Flow Diagram of the Propose System*

* 1. **Use Case Modeling and Description**

A use case is a list of actions or event steps typically defining the interactions between a role and a system to achieve a goal. Figure 3.2 shows the use case modeling of SWEETCHAT application.

Figure 3.2 Use Case Modeling of SWEETCHAT



User

* 1. **Use Case Description**

Table 3.1: Use case description of Create Account

|  |  |
| --- | --- |
| Use Case | Create Account |
| Primary Actor | User |
| Secondary Actor | Null |
| Pre-Condition | Has Gmail Account |
| Scenario | Enter Email address |
| Post-Condition | Create account successfully or not |

Table 3. 1 Use case description of create account

Table 3.1: Use case description of Registration/Login

|  |  |
| --- | --- |
| Use Case | Registration/Login |
| Primary Actor | User |
| Secondary Actor | Null |
| Pre-Condition | Has Gmail Account |
| Scenario | Authenticate gmail account |
| Post-Condition | Registration successfully or failed |

Table 3. 1 Use case description of Registration/Login

Table 3.1: Use case description of Create New Chat

|  |  |
| --- | --- |
| Use Case | Create New Chat |
| Primary Actor | User |
| Secondary Actor | Receiver |
| Pre-Condition | Already logged in |
| Scenario | Create New Chat  Add Gmail |
| Post-Condition | Send Message |

Table 3. 1 Use case description of Create New Message

Table 3.1: Use case description of Send Message

|  |  |
| --- | --- |
| Use Case | Send Message |
| Primary Actor | User |
| Secondary Actor | Receiver |
| Pre-Condition | Already logged in |
| Scenario | Select Friend  Type Message  Send Message |
| Post-Condition | Show Message successfully send or not View Message |

Table 3. 1 Use case description of Send Message

Table 3.1: Use case description of Receive Message

|  |  |  |
| --- | --- | --- |
| Use Case |  | Receive Message |
| Primary Actor |  | User |
| Secondary Actor |  | Sender |
| Pre-Condition |  | Already logged in |
| Scenario |  | Select Friend  Receive Messaage  View Messgae |
| Post-Condition |  | View Message |

Table 3. 1 Use case description of Receive Message

Table 3.1: Use case description of LogOut

|  |  |
| --- | --- |
| Use Case | LogOut |
| Primary Actor | User |
| Secondary Actor | Null |
| Pre-Condition | Already logged in |
| Scenario | LogOut |
| Post-Condition | Logout successfully or failed |

Table 3. 1 Use case description of Logout

* 1. **Design Requirements**
     1. **Functional Requirements**

1. **User Registration:** User must be able to register for the application through an Email, Username and Password. On Opening the application, user must be able to register themselves or they can directly login if there have an account already. If user skips this step, user should able to chat. The user’s email will be the unique identifier of his/her account on Chat Application.
2. **Adding New Contacts:** The application should detect all contacts from the server database. If any of the contacts have user entered with Chat Application, those contacts must automatically be added to the users contact list on Chat Application.
3. **Send Message:** User should be able to send instant message to any contact on his/her Chat Application contact list. User should be notified when message is successfully delivered to the recipient by coloring message.
4. **Broadcast Message:** User should be able to send message to contacts. User should be able to broadcast messages.
   * 1. **Non-Functional Requirements**
5. **Privacy:** Messages shared between users should be encrypted to maintain privacy.
6. **Robustness:** In case user’s system crashes, a backup of their chat history must be stored on remote database servers to enable recoverability.
7. **Performance:** Application must be lightweight and must send messages instantly.

**CHAPTER 4: METHODOLOGY**

The functionality of the chat application is to give the ability to chat with whoever is online on the application. The users and stakeholders will be a small group for now, the use cases will be what is available to the user, and the functional/nonfunctional requirements will be covered, as well as the milestones of the chat application.

1. **Project Perspective**

* The system to be developed here is a Chat facility. It is a centralized system. It is Client-Server system with centralized database server. All local clients are connected to the centralized server via internet.
* There is a two-way communication between different clients and server. This chat application can be used for group discussion. It allows users to find other logged in users.
  1. **System Design**

This section will describe the details of the proposed web based applications. In chatting application, all data are stored in digital database (Firebase). Data must be transferable in real time, there will be a peer to peer connection within the users.

* 1. **Conceptual Framework**

Our proposed WebApp named “SweetChat” enables users to perform basic messaging features. Firstly, we need to develop a system and basic web pages for messaging. Following figure shows a complete overview of our proposed web application.

One Click Login

Select Existing Contact

Add New Contact

Start Messaging

Figure 4.1 Conceptual Framework

* + 1. **Architecture: How We Approach It**

**Chat Server Engine**

Pool of external servers responsible for the chat operation

**Chat App**

Web based application

**Chat Client Engine**

Communicate with the chat server engine via its components

**Chat UI**

Display data to the user via its widgets

**Chat Dialog UI**

**Chat Contact list UI**

**Chat REST API Client Library**

**Chat REST API**

**Chat WebSocket Server**

**Chat WebSocket Client Library**

*Figure 4.2 Chat Architecture*

* + 1. **UI/UX Design**

Simply put, User Interface Design is important because it can make or break your customer base. It creates fewer problems, increases user involvement, perfects functionality and creates a strong link between your customers and your website.

It shows us the basic structures/demo and interactions of the conceptual product.

Furthermore it play an important role in development, makes the implementation easy.

* + - 1. **Conceptual Design**

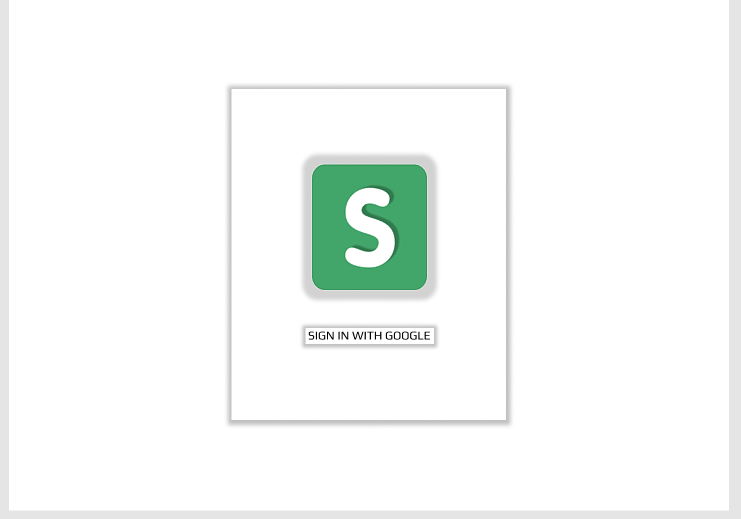
****

Figure 4. 3 Conceptual Login Page

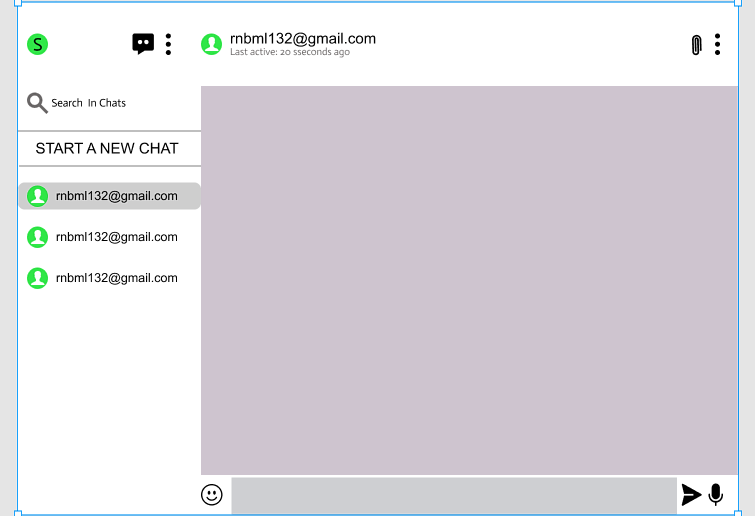


Figure 4. 4 Conceptual Main Page

* 1. **System Implementation**

This chapter details the most relevant parts of the application development, decisions taken and algorithms. We have divided this chapter into three sections: databases (design), features (the most important ones) and a brief overview on how we tested our features.

* + 1. **Database and Models**

A key defining aspect of any database-dependent application is its database structure.

The database design can vary depending on many different factors, such as the number of reads over writes or the values that the user is likely to request the most. That is because as full stack developers we want the database to have the best performance, which can often be achieved by focusing the optimizations on the most common actions.

We concentrated on the Firestore database, which is the most complex data storage and the one which stores the most data.

* + - 1. **Users**

To start, we needed somewhere to store our users. Since we were expecting a significant number of entries, an individual collection for the users’ themselves was the most appropriate. What we mean by that is that it was best for the users’ collection to solely store the information that made reference to their authentication and personal data. Their chats, and messages should be stored somewhere else. Given that we were expecting a lot of rooms, chats, and messages per user, we refrained from even making references to them in this collection. We are querying these other collections directly.

**Schema fields:**

* + id: identifier.
  + email: email address.
  + lastSeen: timestamp.
  + photoURL: google photo URL.

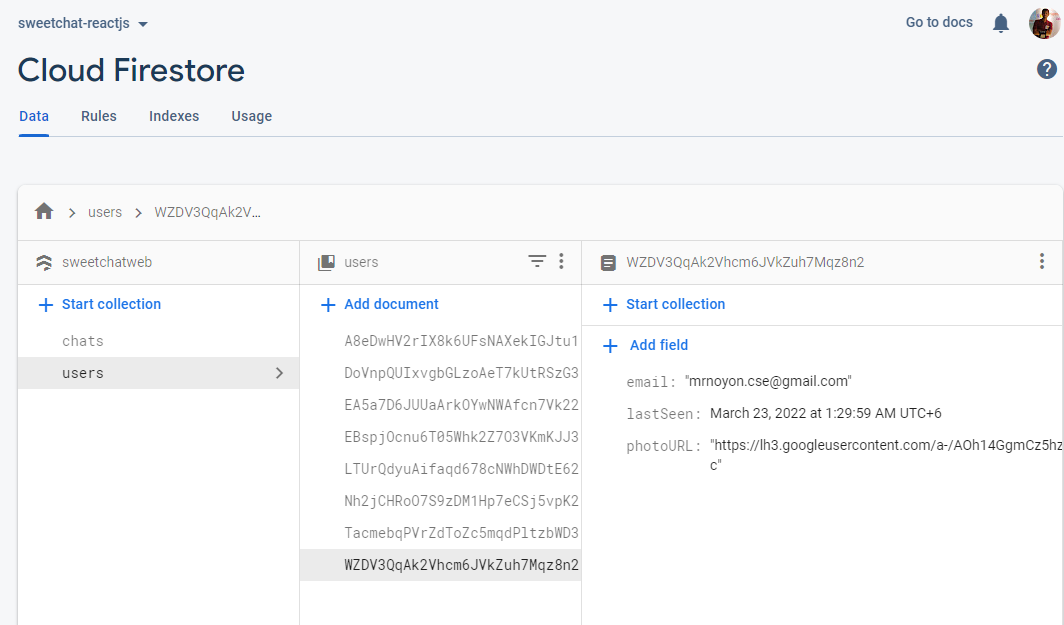


Figure 4. 3 User Data Model

* + - 1. **Chats**

As we stated earlier, our chats were going to be in individual collections. There might be rooms in which their members have few chats, but others might have hundreds (even if that leads to having a few inactive ones). Once again, we had to think whether it was worth embedding or referring messages inside the Chats collection or keeping them isolated in another one. In this case, it was evident.

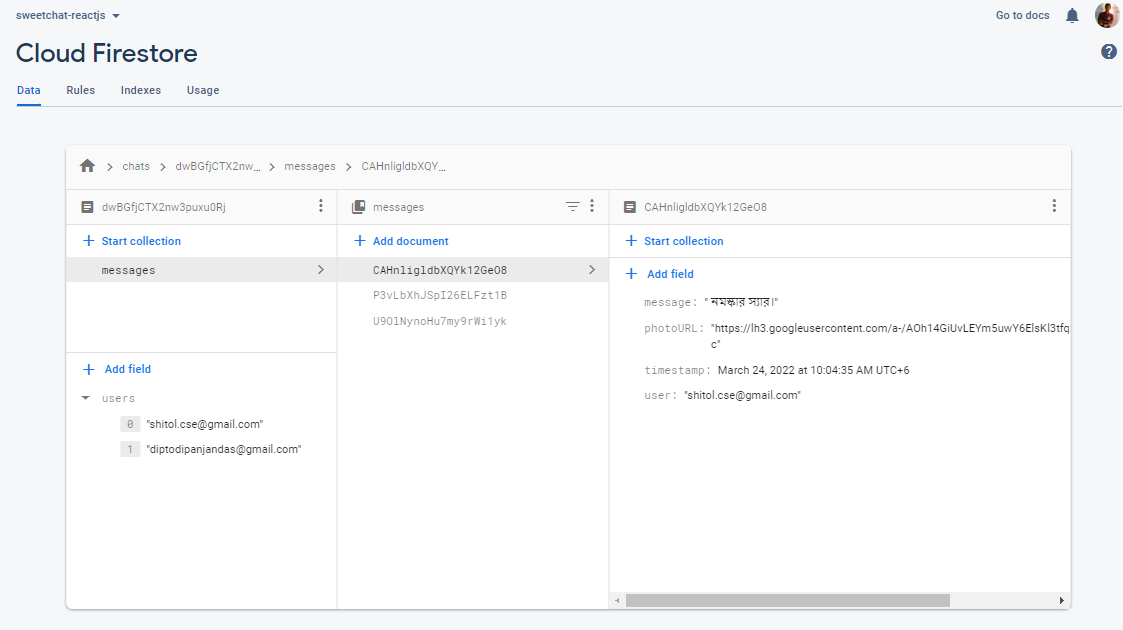


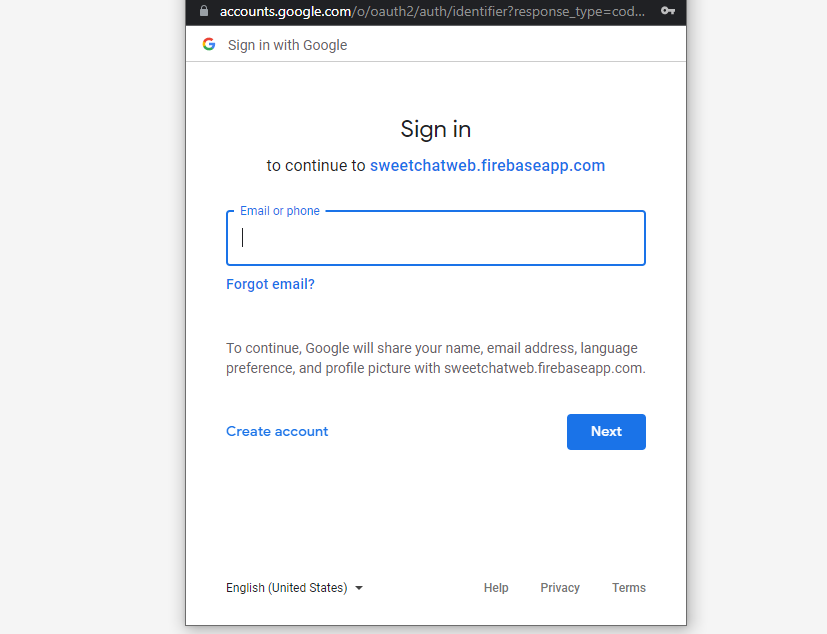
Figure 4. 4 Chat Data Model

* + 1. **UI/UX**

This chapter details the most relevant parts of the ui/ux implementation. We have divided this chapter into several sections: such as LogIn, ChatScreen, Sidebar, [id].js, and Messages etc.

* + - 1. **Log In**

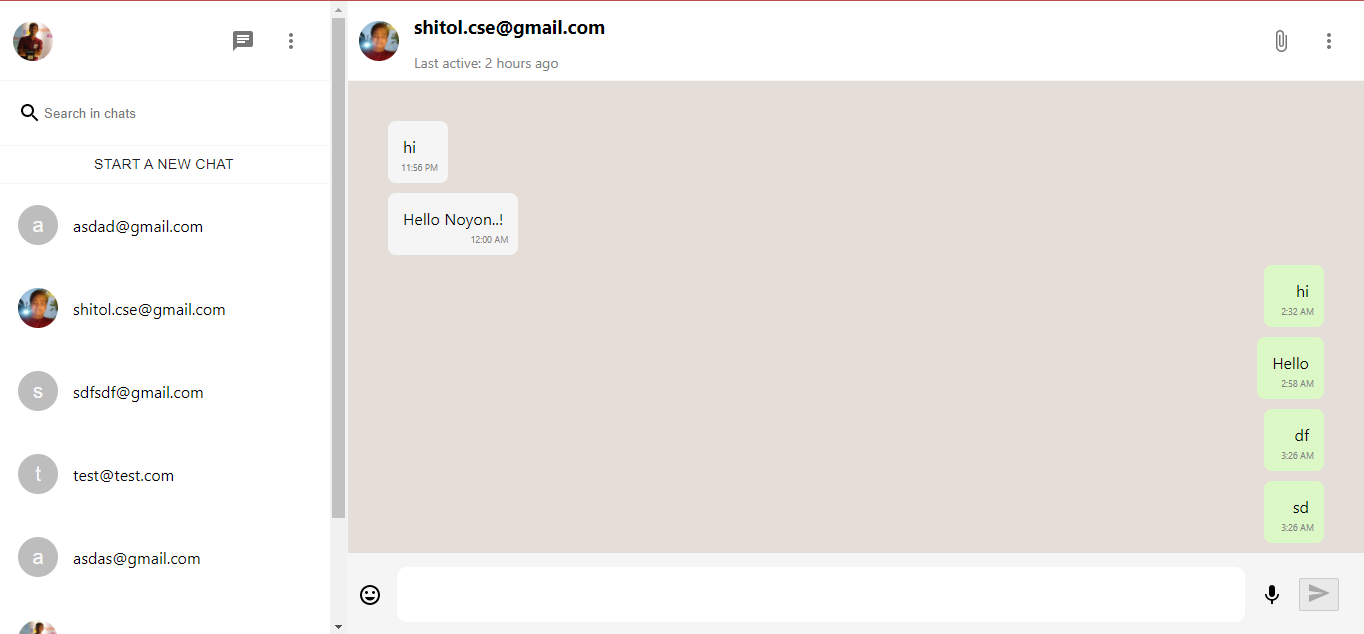
This is the page to login/signup of the application. We implement it as like simple like the design given above. Added a simple One Click authentication button, That performs google authentications by opening a popup window.

*Figure 4.5 Login page and Popup window*

* + - 1. **ChatScreen**

ChatScreen holds the Sidebar, Header, Message Container and rest of the components. Sidebar holds Add New Contact Button, SearchBox, Contacts Components. Header holds the identification of chatting user and the status components.



*Figure 4.6 ChatScreen page*

The ChatScreen components shows the message calling function showMessage,

**Algorithm:** show message algorithm

**Output:** show messages

**Input:** key,user,message(json)

**if**( has messageSnapshot)

**if**(key==user)

**return** showMessages;

**else**

**return** json.parse;

**end**

**Code Sample:**

*const showMessages = () =>{*

*if (messagesSnapshot) {*

*return messagesSnapshot.docs.map((message) => (*

*<Message*

*key={message.id}*

*user={message.data().user}*

*message={{*

*...message.data(),*

*timestamp: message.data().timestamp?.toDate().getTime(),*

*}}*

*/>*

*));*

*} else JSON.parse(messages).map(message =>(*

*<Message key={message.id} user={message.user} message={message} />*

*))*

*};*

* + 1. **Version Control**

We used GitHub as the version control system of our proposed application. GitHub is a web-based version-control and collaboration platform for software developers. Git is used to store the source code for a project and track the complete history of all changes to that code. It allows developers to collaborate on a project more effectively by providing tools for managing possibly conflicting changes from multiple developers. GitHub facilitates social coding by providing a web interface to the Git code repository and management tools for collaboration.

* + - 1. **GitHub Contribution**

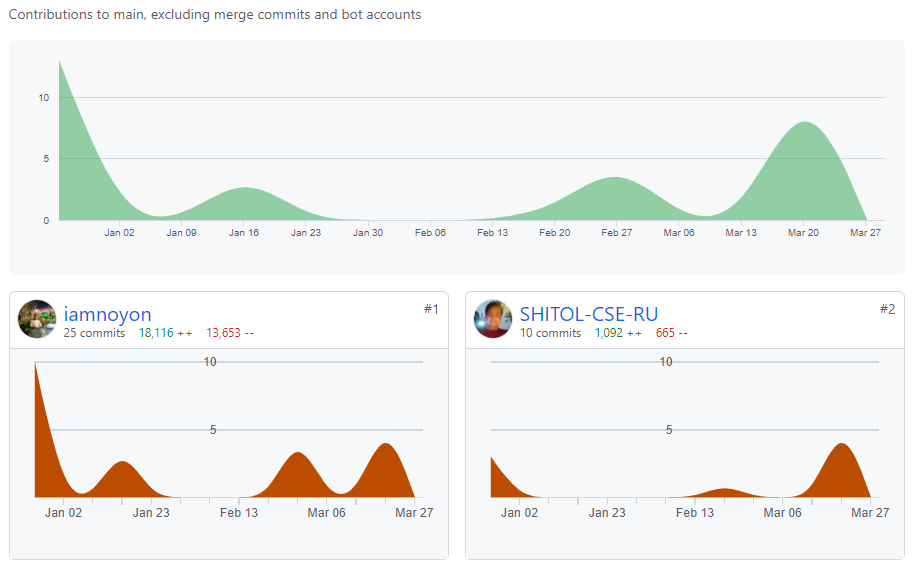
****

Figure 4. 7 GitHub Contribution

* 1. **Security**
     1. **Data Encription**

Firebase services encrypt data in transit using HTTPS and logically isolate customer data. In addition, several Firebase services also encrypt their data at rest: Cloud Firestore, Cloud Functions for Firebase, Cloud Storage for Firebase, Firebase Crashlytics, Firebase Authentication, Firebase Cloud Messaging, Firebase Realtime Database, Firebase Test Lab, Firebase App Check.

* + 1. **Firebase Service Data**

Firebase Service Data is personal information that Google collects and generates during the provision and administration of the Firebase services\*, excluding Customer Data\*\* as defined in our customer agreements covering Firebase services and Google Cloud Service Data. Examples of Firebase Service Data include information about service usage, resource identifiers like application IDs and package name/bundle IDs, technical and operational details of usage such as IP addresses, and direct communications with developers from feedback and support related conversations.

* + 1. **Server-side encryption**

Firestore automatically encrypts all data before it is written to disk. There is no setup or configuration required and no need to modify the way you access the service. The data is automatically and transparently decrypted when read by an authorized user.

With server-side encryption, Google manages the cryptographic keys on your behalf using the same hardened key management systems that we use for our own encrypted data, including strict key access controls and auditing. Each Firestore object's data and metadata is encrypted under the 256-bit Advanced Encryption Standard, and each encryption key is itself encrypted with a regularly rotated set of master keys.

Server-side encryption can be used in combination with client-side encryption. In client-side encryption, you manage your own encryption keys and encrypt data before writing it to Firestore. In this case, your data is encrypted twice, once with your keys and once with Google's keys.

**CHAPTER 5: TESTING & DEPLOYMENT**

1. **Testing**

Testing is the set of activities that can be planned in advance and conducted systematically. Numbers of testing strategies are proposed. All provide software developer with a template for testing and all have following characteristics.

Testing begins at component level & works “outward” towards the integration of the entire computer based system.

* Different testing techniques are appropriate at different points in time.
* Testing is conducted by the developer of the software & independent test group.
* Testing & debugging are different activities, but debugging must be accommodated in any testing strategy.
  1. **TEST CASES FOR CLIENT**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Steps** | **Description** | **Input Data** | **Expected Result** | **Actual Result** | **Status** |
| **1** | Login info | Click on login  info tab | Login information  must be displayed | It shows the login  info. | Pass |
| **1.1** | Email insert  area | Click on submit  button with  invalid Email user | Must show  appropriate error | It shows the  appropriate error  with error cause | Pass |
| **1.2** | Submit button | Click on submit  button with  appropriate  Email | Must disable submit  button and email  field and enable all  tabs and add user to users  connected list | It disables  username field,  submit button and  enable all tab  (server status tab  if user is admin)  and adds  username to list | Pass |

Table 5.1Test cases for Login tab

* 1. **SYSTEM TESTING**
     1. **Recovery Testing**

Recovery testing is a system test that enforces the software to fail in a variety of ways and verifies that recovery is properly performed. If the recovery is automatic, re-initialization, check pointing mechanism and data recovery and restart are each evaluated for correctness. If recovery requires human intervention, the mean time to repair is evaluated to determine whether it is within acceptable limits.

* + 1. **Security Testing**

Security testing attempts to verify that protection mechanisms built into a system will infract protect it from improper penetration.

* + 1. **Stress Testing**

Stress tests are designed to handle programs with abnormal situations. Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency or volume.

* 1. **Deployment**

Since the very beginning, the chat application was meant to be a cloud service. Users would be able to access it anytime without having to install themselves any special software. Thus, we had to upload our working software on a remote server, which was accessible worldwide.

Although we marked the deployment as, we are now going to deploy our product, when we already had a decent set of utilities, and the platform was already usable through the UI. We used a **Vercell** server to conduct our deployment.

* + 1. **Vercell Deployment**

Signup for an account on Vercel. It will be great if you choose the Signup with Github option. Import the repositories from our Github account. Give Vercel the permissions to read the repositories in our GitHub account. This will list out all the repositories in our GitHub account.

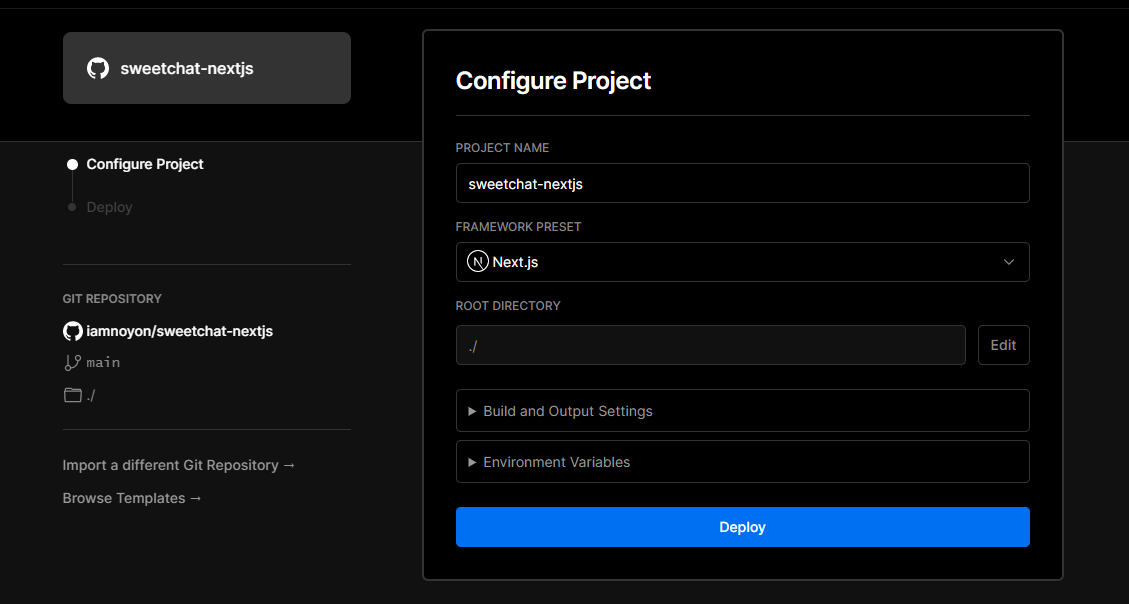
Import the repository next.js-demo-app that we created earlier. 

Figure 5. 2 Deploying to Vercell

Now deploy the project without changing anything. So our app is deployed and now can see a Congratulations window

**CHAPTER 6: FUTURE WORK & CONCLUSION**

1. **Future Work**

Although the application itself works well, much was learned during its development. For this reason, we wrote a list of possible improvements/changes, some of which are easy to execute, others might require rewriting a significant amount of the current source code. Apart from that, the Ideal application (which we described in section 2.2) was too ambitious, which resulted in many features not being able to be implemented during the course of the project.

* + 1. **Next to Hapi**

Next works well for small projects, it is easy to set up and you can have an API working within minutes. However, it is very minimalistic. As the project gets bigger, you are forced to write much middleware code yourself, which does not only take time but it can lead to security risks if not properly tested. Hapi is a more modern Node.js framework, with security in mind and designed to handle big loads. Hapi by itself can handle things such as input validation, server-side caching, cookie parsing or logging. Although moving to Hapi is not a requirement, we believe it is a wise move since it would ease a lot of future work.

* + 1. **Remaining features**

The model platform, described in the "Features" section, had plenty of features. Many of them remain undone:

• Notifications

• Status Update

• Group Rolls

• File sharing

• Emote Send

• Voice and videocalls

• Public API

• etc.

While our application already provides the basics to programmers who want to talk and share code themselves, having more of these model features done would probably attract the attention of more of them.

* 1. **Conclusion**

In this project, we try to show our concept on social messaging web application. We try to cover up some features that need to implement web messaging applications. We proposed a simple web based realtime messaging app. In the modern world the value of web apps are increasing, as it works as independent platform.

It is now a usable product to send real time messages. It is more secure, as we use firebase encryption. There are several chat apps which serve similar purpose as this project, but these apps were rather difficult to use and provide confusing interfaces. A positive first impression is essential in human relationship as well as in human computer interaction. This project hopes to develop a chat service Web app with simple modern user interface.

**REFERENCES**

[1] Howell, Bronwyn E., and Petrus H. Potgieter. "The Techno-Economic Content of Your Crypto Wallet." (2019).

[2] Bakar, Nashirah Abu, Sofian Rosbi, and KiyotakaUzaki. "E-wallet transactional framework for digital economy: a perspective from Islamic financial engineering." International Journal of Management Science and Business Administration 6.3 (2020): 50-57.