

Software Engineering Project

Reminder Web Application

Salmanuddin Rehan Mohammed

CS 630 Software Engineering

May 30, 2025

Contents

Problem Statement	3
Solution Overview	3
Key User Stories	3
Functional Requirements	3
Non-Functional Requirements	4
Conceptual Wireframes	4
High Level Architecture Diagram	5
Technology Stack Justification	5
Initial Working MVP	6
Automated Tests	6
Test Case 1: Schedule Reminder Successfully	6
Test Case 2: Input Validation (check empty fields).....	7
Test Case 3: Fetch and Display Reminders	7
Developer Log	8
Final MVP.....	9
Post-Mortem Analysis	17
Things that went well:.....	17
Things that were challenging and how I adapted:	17
Things that I learned Technically and Process-wise:.....	18
Final Result vs Initial Plan:.....	18

Software Engineering Project

Problem Statement

Users often forget important tasks, events, and meetings. Missed appointments, forgotten and delayed tasks can lead to loss in productivity, reduced efficiency, and poor time management. A simple reminder app can help the user to schedule reminders at specific times. While calendar apps can do this task, they lack timely notification that can prompt the user to take timely action.

Solution Overview

This lightweight web application aims to allow users to schedule reminders sent via SMS. It helps in reducing missed tasks. In addition, it will help in improving time management and increase efficiency. This tool can be beneficial to busy individuals and remote teams.

Key User Stories

1. As a user, I want to schedule a reminder message with a specific time and recipient so that it is sent automatically.
2. As a user, I want to receive confirmation that my reminder was sent successfully.
3. As a user, I want to view all the reminders so that I can keep a track of all my upcoming tasks and deadlines.

Functional Requirements

1. The user can enter a recipient, message, and select reminder type.
2. The system shall validate all the required fields before saving the reminder.
3. The system shall allow users to view a list of all their scheduled and sent reminders.
4. The system shall provide a dashboard to view all the upcoming reminders.

Non-Functional Requirements

1. Must respond to reminder requests within 10 seconds.
2. The system shall support 50 users concurrently.
3. The user interface should be accessible in web and mobile devices.

Conceptual Wireframes

A conceptual wireframe for a 'Schedule Reminder' form. The form is titled 'Schedule Reminder' and contains three input fields: 'Recipient', 'Message', and a date field with a placeholder 'dd-mm-yyyy --:--'. Below the date field are two radio buttons labeled 'Email' (selected) and 'SMS'. At the bottom is a blue button labeled 'Set Reminder'.

Fig 1: Design for Scheduling Reminder

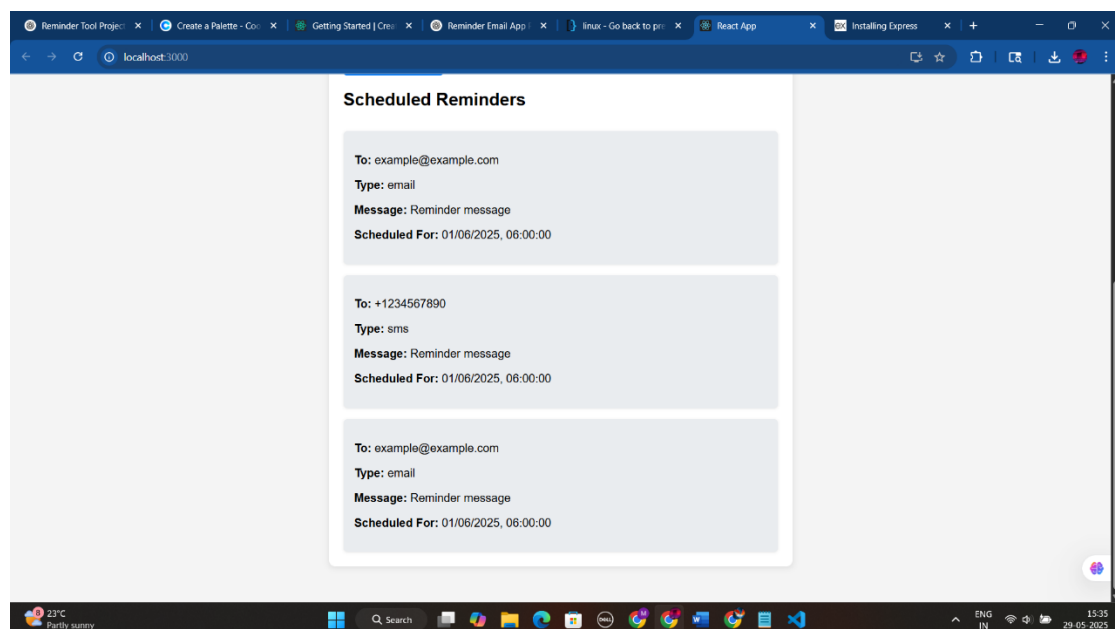


Fig 2: List of Scheduled Reminders

High Level Architecture Diagram

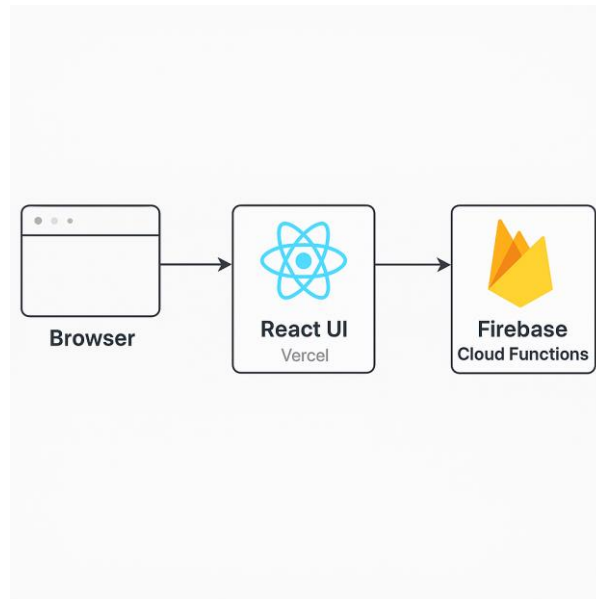


Fig 3: High Level Architecture Diagram

Technology Stack Justification

1. I am using React for front end development due to its simplicity and ease of use.
2. Firestore database: for storing the reminder list and displaying it in the dashboard.
3. I am using Firebase scheduled cloud functions to send emails at the desired date and time.
4. Firebase: It allows for automatic deployments using GitHub.
5. Jest: Testing framework as it has a good community support for any queries and questions that I may run into.

Initial Working MVP

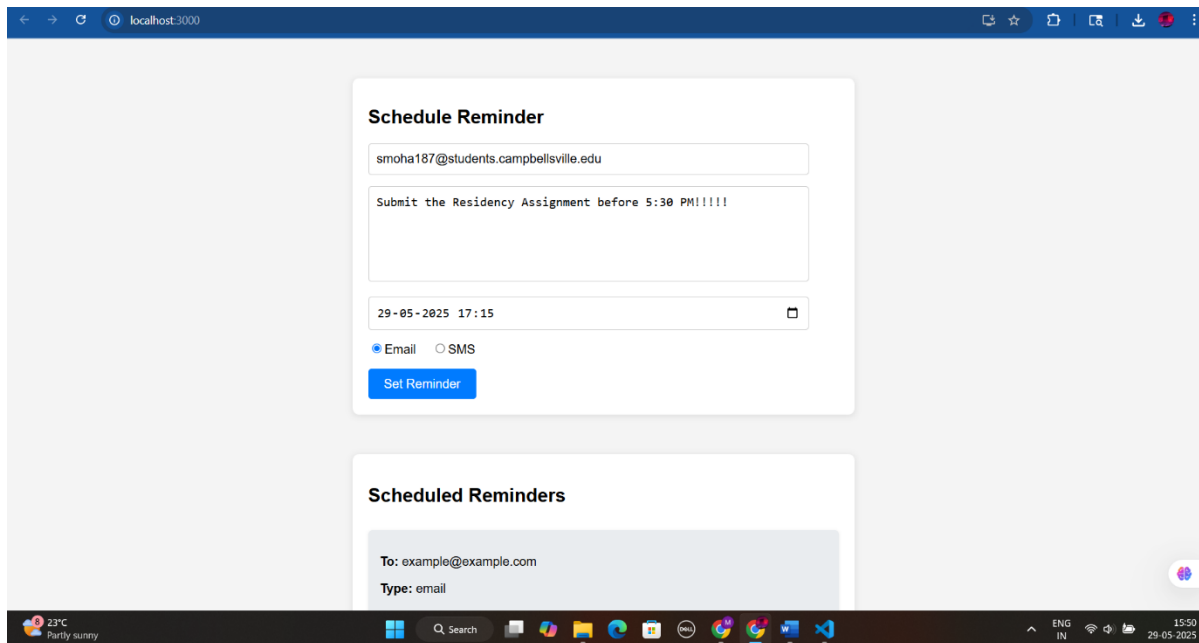


Fig 3: Actual Working application with scheduling a reminder with a specific date and time

Automated Tests

Test Case 1: Schedule Reminder Successfully

Objective: To ensure that the valid reminder is successfully stored in the Firestore database.

Input:

- To: test@example.com
- Message: This is a test reminder
- Datetime: 2025-29-05

Steps:

1. User fills in all the fields.

2. User selects email.
3. Clicks “set reminder”.

Expected Result:

- Reminder appears in the dashboard.
- Firestore reminders collection contains the new document.

Test Case 2: Input Validation (check empty fields)

Objective: To ensure that the valid reminder is successfully stored in the Firestore database.

Input:

- To: “ ” (empty)
- Message: “This is a test reminder”
- Datetime: “2025-06-01T14:00”

Steps:

- Leave recipient field blank.
- Fill out other fields.
- Clicks “set reminder”.

Expected Result:

- Form should not submit.
- Error message appears: “please fill in all fields”.

Test Case 3: Fetch and Display Reminders

Objective: Ensure the dashboard correctly fetches and displays reminders.

Precondition:

- At least one reminder exists in the firestore database to display.

Steps:

- Open the application.
- Wait for useEffect to run fetchReminders.

Expected Result:

- Reminders are fetched from Firestore.
- They appear under the “Scheduled Reminders section” section with correct data.

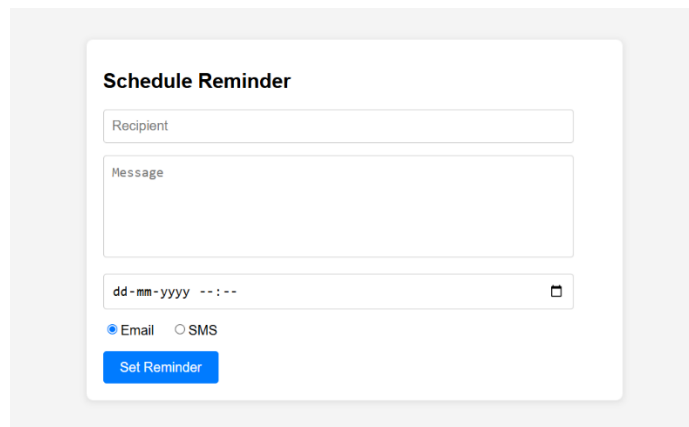
Developer Log

- Today I focused on finalizing the project for this residency assignment.
- I focused on finalizing the technology stack and developing the initial working prototype of the project.
- I switched back and forth between choosing a full stack project versus building a project using the firebase cloud services.
- I finalized on using firebase due to limited time as choosing a full stack MERN technology would have created complexities in building the front end as well as the back end.
- Key challenges included integrating the front end and the back end.
- I decided to use Firebase deployment for easy deployment using GitHub.

Final MVP

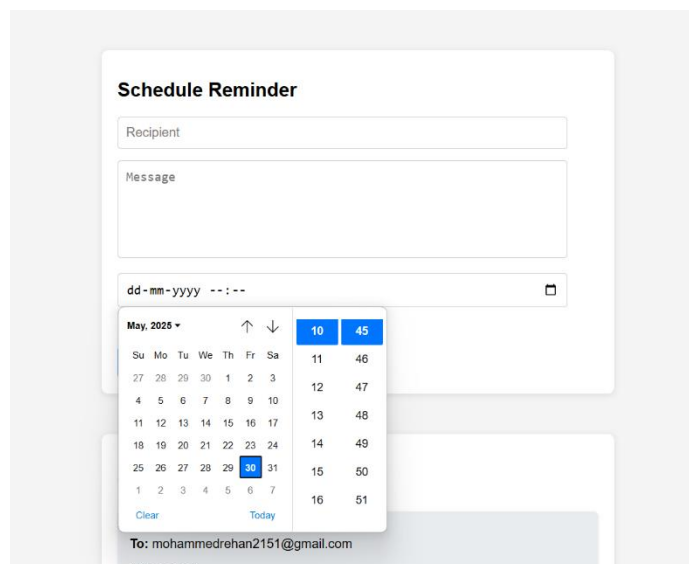
I have completed developing the email reminder web application with all its core functionality. Below are the screenshots of the working application with each screenshot displaying one feature or functionality.

1. User Design: I have the designed the web application simple keeping in mind of the limited availability of time.



The screenshot shows a web form titled "Schedule Reminder". It contains three input fields: "Recipient", "Message", and a date/time picker. The date/time picker is set to "dd-mm-yyyy --:--". Below the date/time picker, there are two radio buttons: "Email" (selected) and "SMS". At the bottom of the form is a blue button labeled "Set Reminder".

Fig 4: User input form for entering Recipient, message, and time.



The screenshot shows the same "Schedule Reminder" form as in Fig 4, but with a calendar overlay for date and time selection. The calendar is for May 2025. The date "29" is selected. The time "10:45" is selected. The "Email" radio button is selected. At the bottom of the form, the "To" field is populated with "mohammedrehan2151@gmail.com".

Fig 5: User input form with calendar for date and time.

- Testing if the user has filled all the fields and details, if all the fields are not filled, the application will throw a message displaying “Please fill in all the fields”.

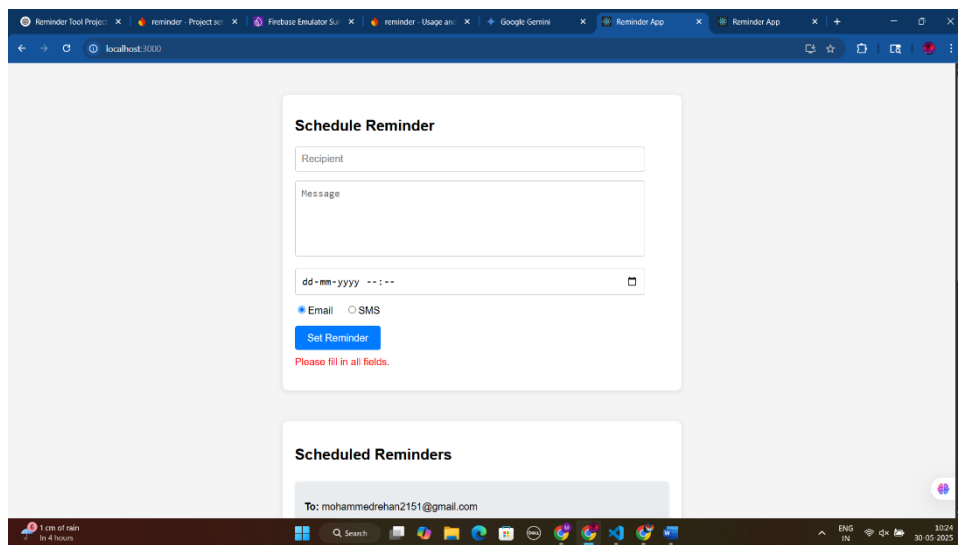


Fig 6: Testing Form data and displaying error.

- Displaying successful message if the reminder was successfully added to the database.

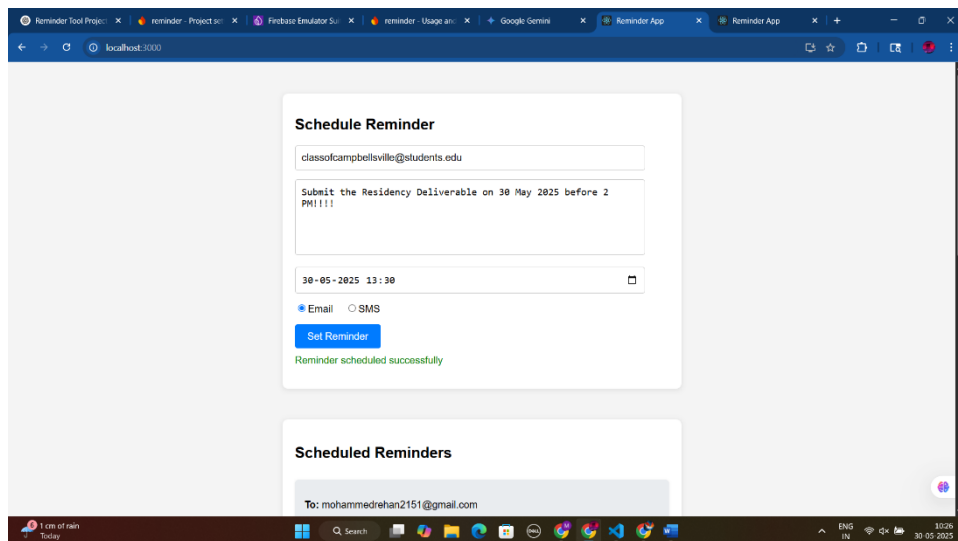


Fig 7: Success message

4. Throwing an error message if there is an issue in adding the reminder to the database.

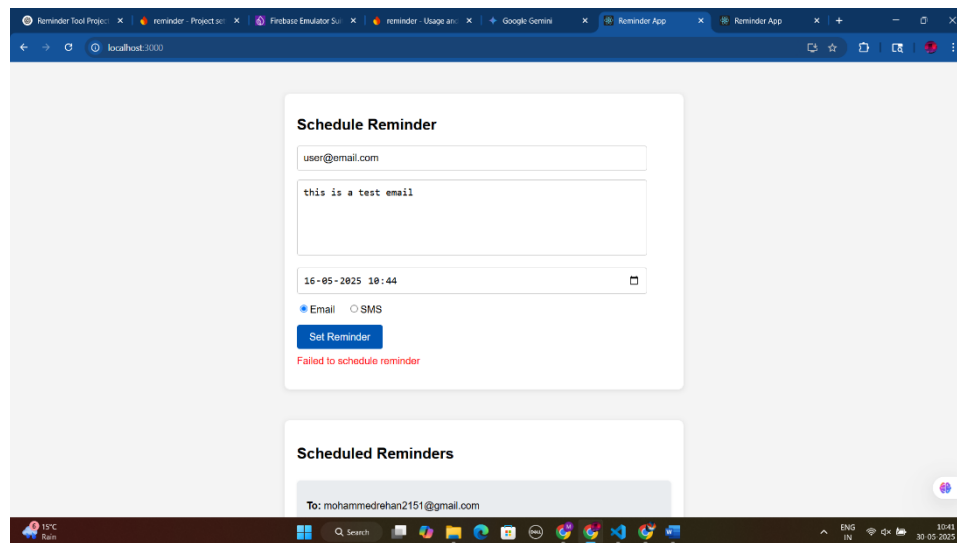


Fig 8: Error Message

5. Displaying all the scheduled reminders from the database. The application will fetch all the reminders from the Firestore database once the application is loaded initially. The database will be retrieved again to display updated reminder list after successful addition of a new reminder in the database.

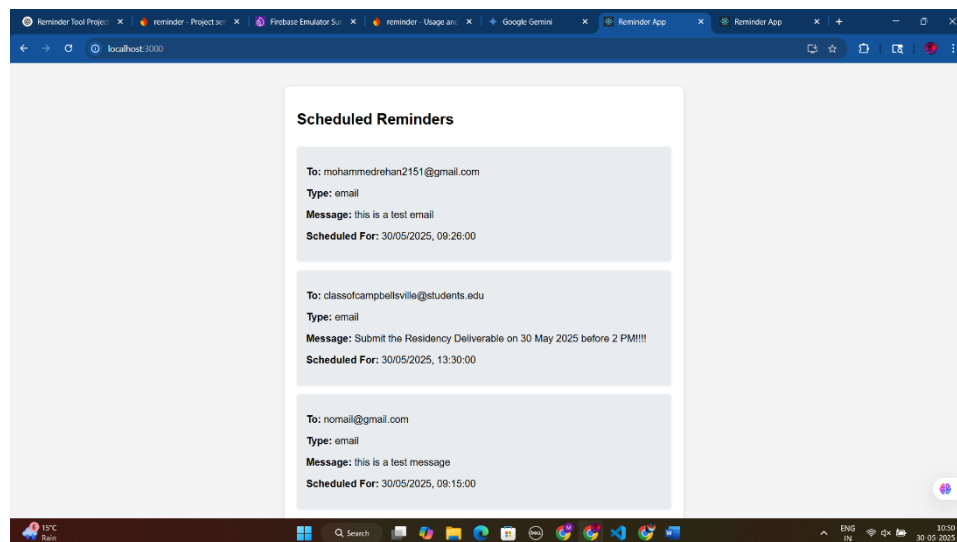


Fig 9: Displaying all the reminder list in the application.

6. Database structure and data stored in the database.
 - a. The database is a NoSQL database.
 - b. The main collection is named as 'reminders'.
 - c. Each reminder has a unique ID associated with it.

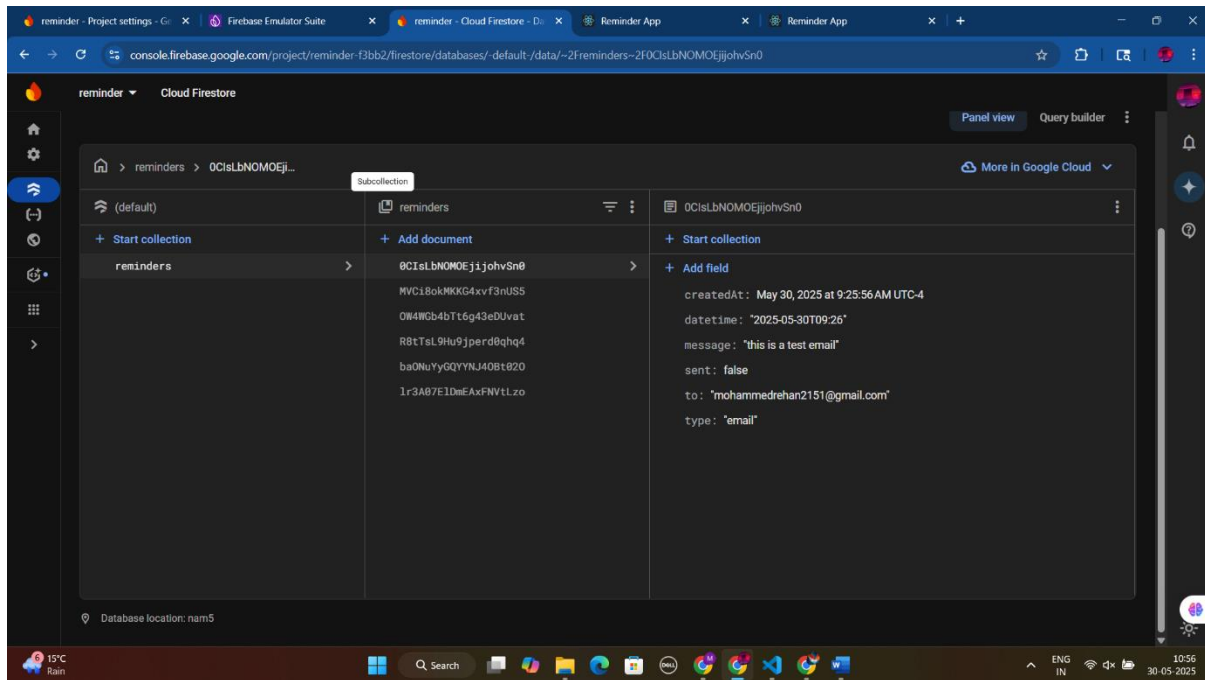


Fig 10: Firestore Database Structure

- d. Then each reminder is a collection of objects that has the fields –
 - i. createdAt: Holds the timestamp of the reminder when it was created.
 - ii. Datetime: Contains the date and time of when the email has to be sent.
 - iii. Message: Contains the message of the email.
 - iv. Sent: Boolean value specifying if the email was sent or it is to be done in future.
 - v. Type: email or phone, the functionality is now limited to emails only. But I have added the phone field for future upgrades.

7. Using firebase functions that run every minute to check if there's any reminder that must be sent. For the sake of this project, I am running the firestore functions locally on the emulator. As deploying them on the production environment would require setting up payment on the account.

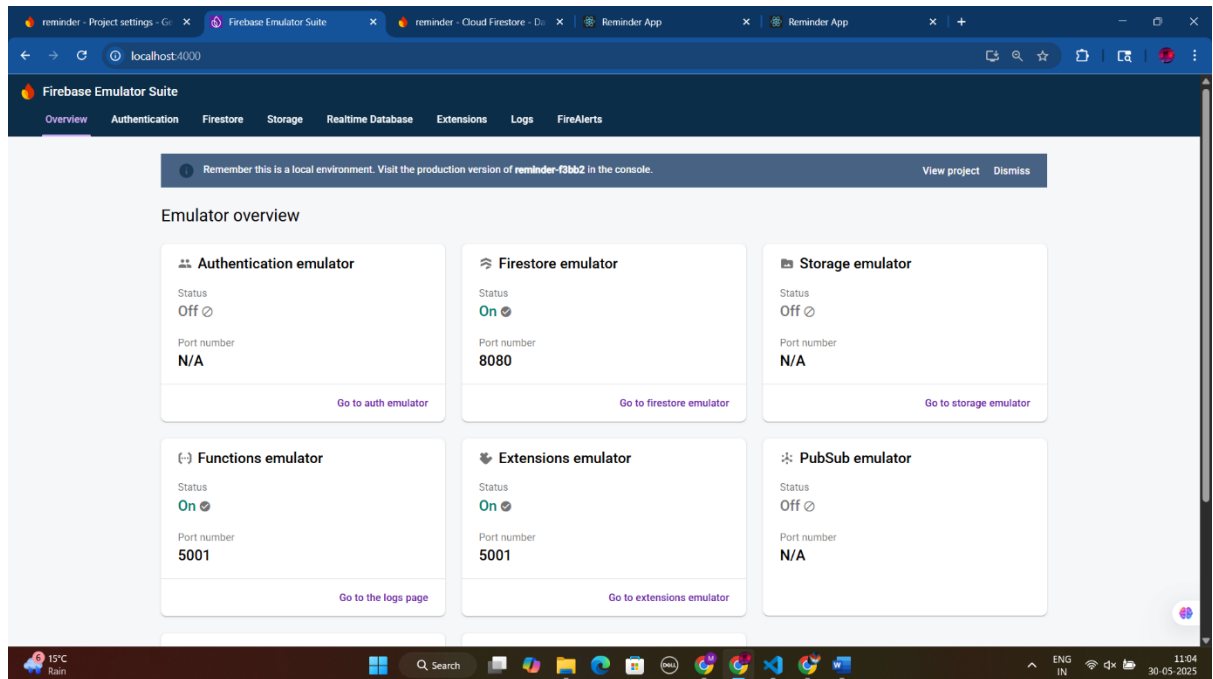


Fig 11: Running emulators locally to emulate the functionality.

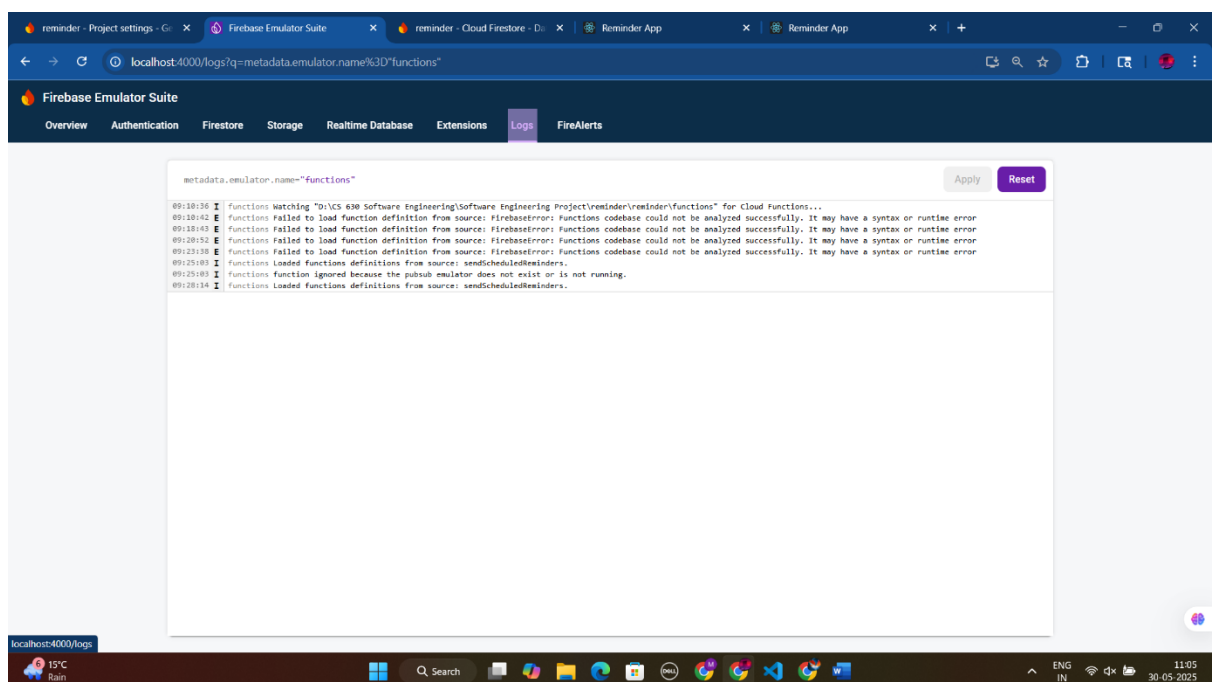


Fig 12: Running Google Cloud Functions on the emulator to schedule the functions.

8. Screenshots of the code.

a. Configuring Firebase with the credentials

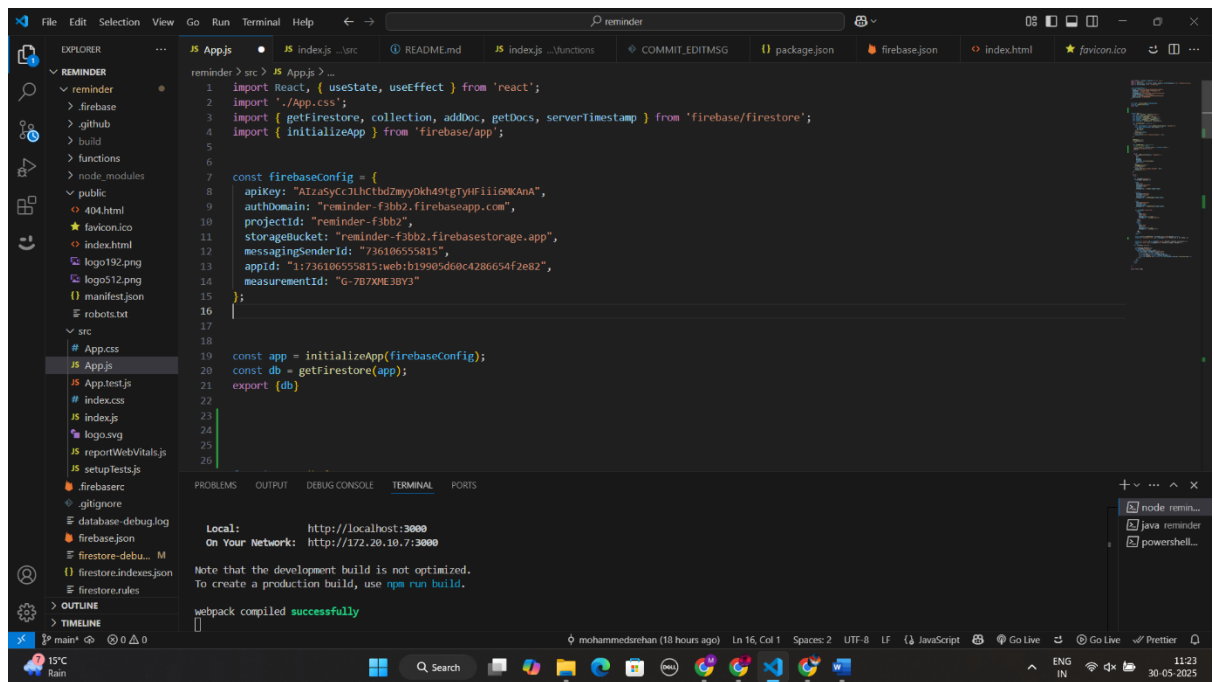


Fig 13: Firebase configuration

9. Fetching data from the database to display on the web application.

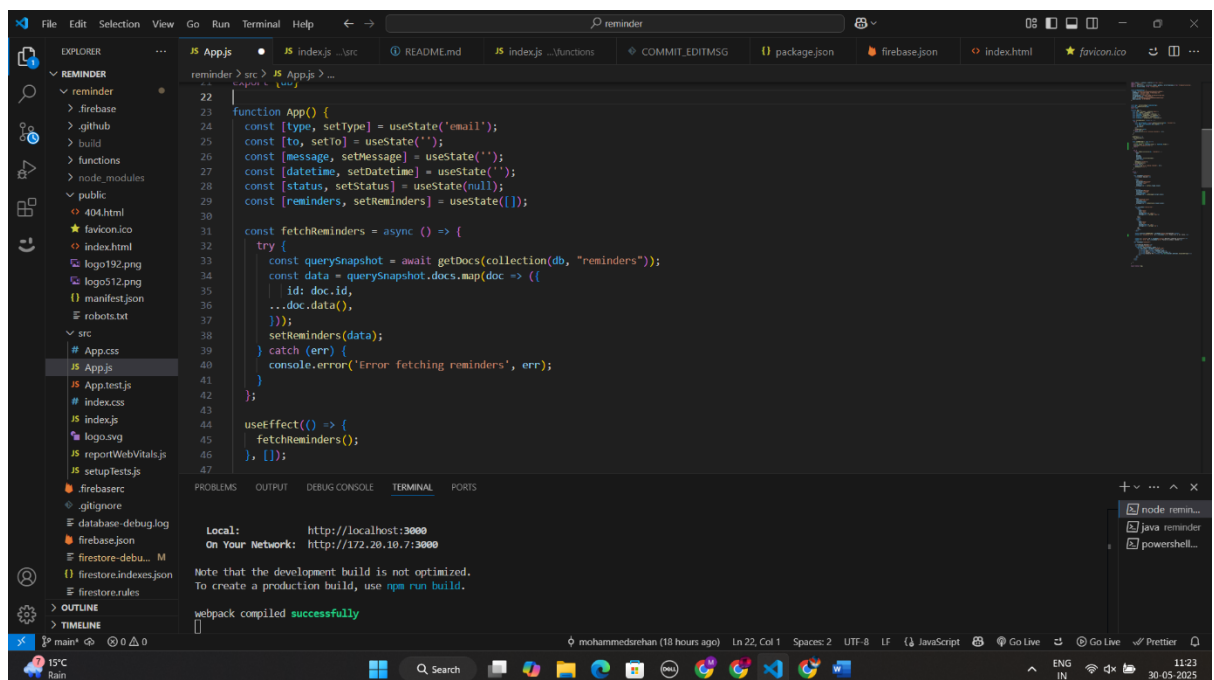


Fig 14: Fetch data from the database

10. Creating an object in the database when the user inputs the data in the fields and inserting it into the database.

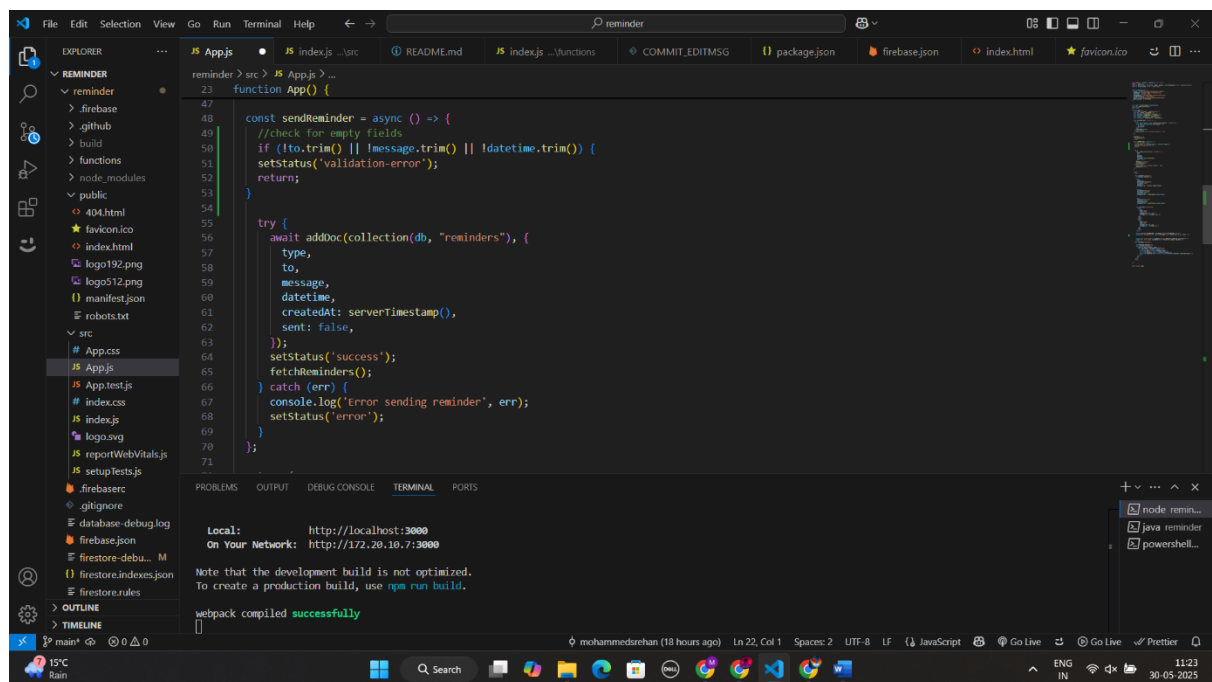


Fig 15: Creating a new reminder from user input.

11. Setting up cloud function that runs every minute to check if there is any reminder that is to be sent.

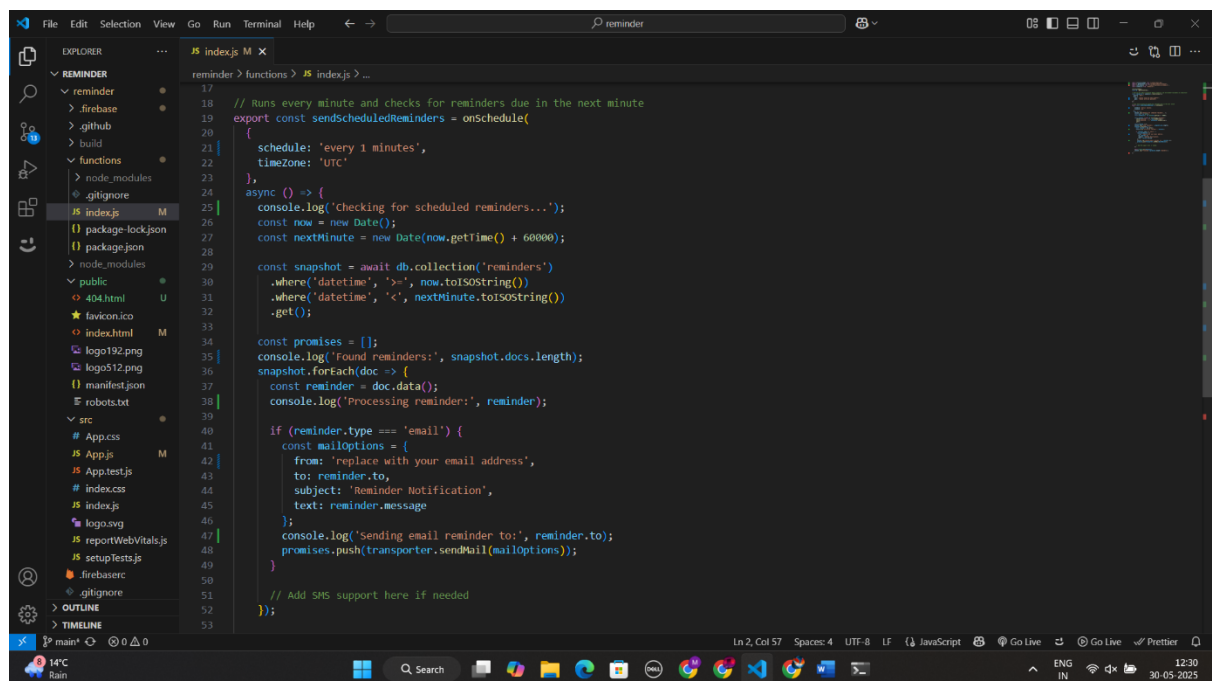


Fig 16: sendScheduledReminder() cloud function

12. Running Test using Jest and React

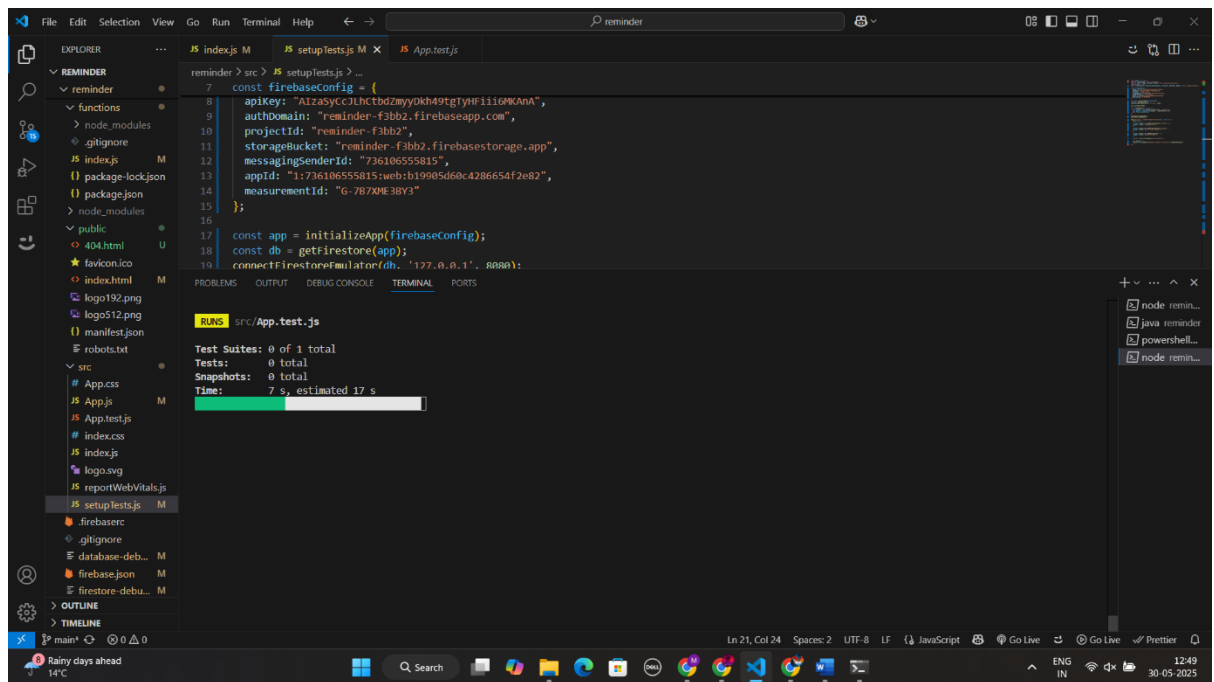


Fig 17: Running Tests

Post-Mortem Analysis

Things that went well:

- I have successfully developed the core functionality of the application.
- I was able to create the user input form with validation.
- I was able to save the input into the firestore database which I have created.
- I was then able to fetch the reminders from the database and display them in the web application.
- I have successfully setup cloud functions using the emulator which runs the function every minute to check if any reminder is to be sent.
- I was able to connect my frontend application with the cloud functions locally.

Things that were challenging and how I adapted:

- Setting up the cloud functions on the deployment server was challenging for me, since the deployment server requires me to setup the payment details for using the services in the production mode.
- So, I have adjusted the project by setting up emulators for simulating the cloud functions.
- Since, I was now using the emulator for cloud functions, I ran into a problem of using database on the actual cloud server and the functions running locally on the emulator.
- Due to this, I have to setup the database on emulator as well to not create any conflicts in the environment.
- Another challenging part for me was running the test using jest.
- When I was trying to run the test using the actual database, the test was timing out due to slow internet connectivity.
- I am attaching the screenshot for the failed test due to connectivity.

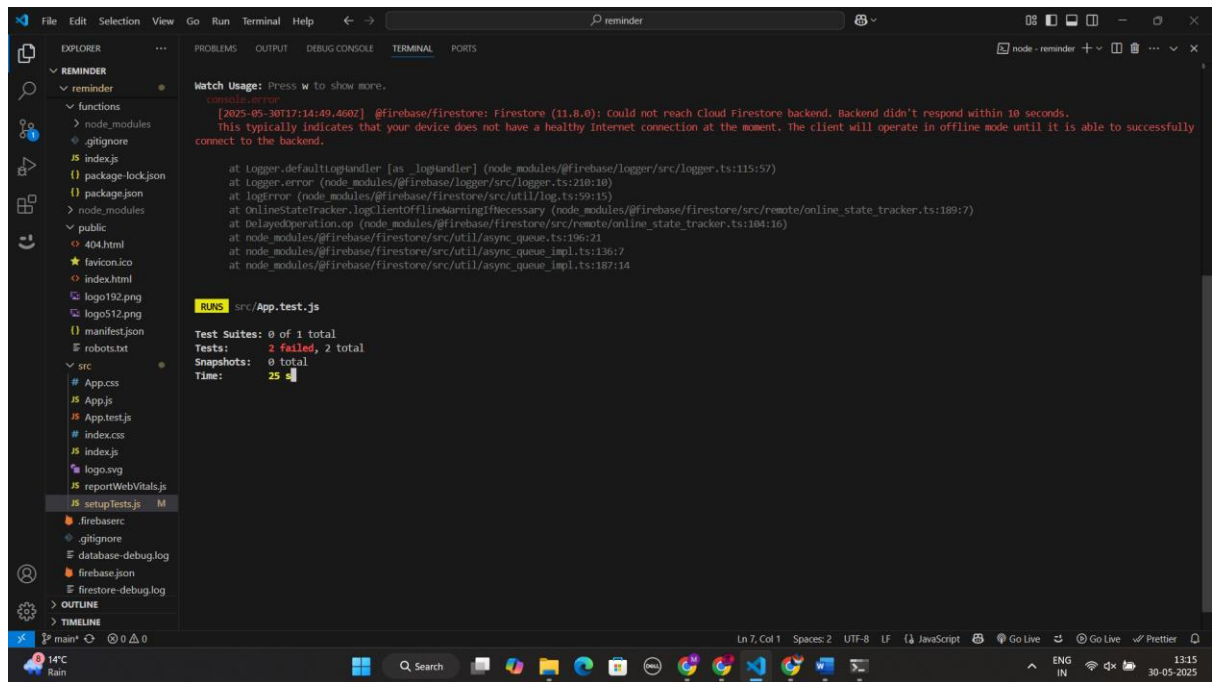


Fig 18: Failed test due to connectivity

Things that I learned Technically and Process-wise:

- I have learned to use cloud functions locally after a lot of trial and error. Before this I only used in a cloud environment.
- I have developed the ability to document the entire process with this project.
- The short duration of this project increased the difficulty and required me adjust and modify the tools.
- I have enhanced my understanding of the google cloud services and its in local environment which is preferred in development mode.

Final Result vs Initial Plan:

- My initial plan was to deploy the project to a working website, but because of using cloud functions locally on an emulator, the project could not be deployed to be an actual website online.

- I was planning to extend the functionality to sending SMS using this application. But I was unable to build the SMS functionality due to limited time.
- The final deliverable has the core functionality of adding reminder to the database and sending reminders. But it lacks features such as being able to update or delete the existing reminders.
- I was planning to use cloud functions that would only trigger when the reminder is to be sent. The final version uses a cloud function that runs every minute which increases the running cost on the deployed server.