# UNIVERSITY OF NORTH TEXAS
# DENTON, TEXAS


## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

# ALMA MINGLE

**Submitted By:**

**Andre Sharp - 11374558**
**Madison Mccauley - 11509676**
**Ponnuru Raja Lakshmi Kamalini - 11659081**
**Shashank Verma - 11703352**
**Suhaibuddin Ahmed -11699042**
**Usama Bin Faheem - 11698740**
**Varun Mahankali - 11708358**
**Rahman Mehmood - 11707971**

# INDEX

# Team Project
# Deliverable 4 – Project Phase 2
# CSCE 5430 (Spring 2024)

## REQUIREMENTS

In phase 2, the team worked in greater sync and the development process was more efficient and less time consuming.

**Post Update:**

- The users can now update the content as well as the title of their posts.
- Special section with the edit feature is present that allows the user to individually edit their posts.
- This increases flexibility and makes the user experience more forgiving to user error.

**Post Delete:**

- The user can also Delete his/her posts in a different section that displays only their posts with the option to delete them.
- The user can pick which specific post needs to be removed now.
- The Delete button is present that removes only the specific posts chosen by the user.

**Post Comment:**

- Each post has a comments section very similar to other social media websites
- Multiple posts can host a variety of different comments by different people and act as a method for users to interact with different posts.
- Each post comes with a comment button that when clicked - allows the user to type out a comment and submit it for that post.

**Filter Posts:**

- Posts can now be filtered through different categories allowing ease of use and improved the user accessibility.
- The posts can now be filtered based on category, visibility, title, etc.

**Update Password:**

- The user can use this option to update Almamingle account's password
- The button is available on the signup page and the users' home page
- When pressed it takes the user to page where he enters his password and confirms it

- This persists the data in the backend where the password is encrypted for the safety and privacy of the user.

**Forgot password on home page: Missing**

- This functionality was missed in error , since we already have the update password we will have to tweak it a little and update the home page
- This will be completed in phase 3.

**Update Profile:**

- The user can use this option to update Almamingle account's Profile details
- The button is available on the users' home page
- When pressed it takes the user to page where he can edit the details he entered or add new details
- When saved the it  persists the data in the backend

**Events:**

- This allows the user to add a new event
- When create event is pressed on the dashboard  the user is taken to the event page where the event details are entered
- After details are entered the data is persisted in the backend

**UI Enhancement of all pages:**

Significant changes to the overall user interface is carried out on the majority of the pages; including home page, user dashboard pages, contact us, messages, posts etc. These changes were made with these factors in mind:

1. *Clear navigation:* To ensure easy navigation by prominently displaying buttons leading to other sections of the website  and using intuitive icons or labels for navigation elements to guide users effectively.
2. *Content Organization:*
    - To present content in a structured manner, emphasizing key features and services offered.
    - Implementing clear headings, sections, and concise descriptions to convey information efficiently.
3. Utilizing high-quality images or graphics that resonate with the users and reflect the purpose of the platform.

**Additional UI Components:**

1. Burger menu in the dashboard has been implemented that shows the hidden menu options on click (similar to expanding a hidden menu)
2. Information tab page on the landing page has been implemented to provide information of the website to the users
3. Gallery on the landing page has been implemented - displaying a range of pictures for user appeal and bringing dynamicity to the page.
4. Footer of the AlmaMingle website has been developed that is present on all the website's pages.

**User Authentication:**

- User login has been revamped to involve logging in and signup through the username and password while having functionality such as password reset through "forget password".
- Accounts can now be only made through the official UNT email ids (the emails with @my.unt.edu).
- Blockers have been made to prevent the user from moving ahead if he has not provided a unt email id or if he has missed out on entering any mandatory details.

**Messages:**

- This will have three sections:
    - a text box for the message,
    - a send message button at the button,
    - area where the user can input the username of the person to whom a message is to be sent.
- This user can use this module to view a list of all the messages that have been received
- Two sections with messages from users and messages from admin
- The background messages from users is white
- The background messages from admin is red
- Messages when viewed the background turns white marked

**Notifications:**

- There is a notification component on the dashboard of all users
- If there is any activity that involves the user , the notification component displays this activity to the users
    - Likes
    - Comments
    - Messages from admin and users

    - If post deleted by admin
    - New post created by the user

**Post Report:**

- Every post has an options button
- Users have option to report the post by pressing the “report” button

**Contact Us:**

- It is a special page that allows the website to obtain feedback from the user
- The users can access this through the Navigation Bar present on the landing page
- The users have to fill in the mandatory details like : Name, name, email, subject, message to submit the feedback.

**Likes Button:**

- A special heart shaped likes button is present that users can click on to “like” a post
- On click, the heart turns pink
- And also shows the number of likes that the post has received

**Comments:**

- This feature allows the user to express their interest in a post.
- Users can click on the comments button to open an accordion under the posts displaying all the different comments from different people.
- Within this accordion there will be a text box that the user can use to enter his/her

Each comment will be shown with the username of the user and the timestamp at which the comment is made.

**Admin User story with all functional requirements:**

Username : admin
Password : admin

- The admin on  AlmaMingle,will log in with the same credentials as other users and see a similar user interface (UI), but they will have additional privileges .
- This user is added by the University in the database
- The number of admins can grow as needed.
- On the login page the admin radio button allows the user to login as an admin
- Once logged in the admin will be able to see all the additional features that the application provides to him

**Post Broadcast:**

- In the messages component the admin have additional option

- In the “To” section the admin can select the option
- This allows him to send a message to  all the users of Almamingle

**Deleting all posts:**

- On the posts page the admin can see an option to delete any post
- When the delete button is pressed the data is deleted from the backend

**Reported Posts:**

- The tab “reported posts” shows all the posts that have been reported by the users
- This page shows the posts that have been reported with the report counts
- The button on top left allows the admin to sort the posts with the ascending or descending order of report count

**Inquiries:**

- The tab “ Inquiries” shows all the inquiries sent from the contact us from from the home page
- The page shows cards that displays the details and the messages sent from the user

**Post Visibility:** This functionality is removed from the scope of our project.
This functionality was optional in our requirement , presently the team believes that this function is not required as Almamingle does not have a large categorized group of users  which would make this function more useful for a large number of  users from different groups. Therefore we have removed this from the scope of our project.

**Missing Functionalities:**

**Event update and Event Delete**
**Explanation:** The team worked on implementing this but due to the code redundancy that is using the similar code for all the posting functionalities in our application there was an error which deleted only the event id whereas the other details remained. We have planned to work on this as we draft this report , if completed we will update this in this report or document it in the next phase report.

**Member management:**
**Explanation:** The team members responsible for the member management task are very new to development in react and have encountered minor bugs and issues that arose while final testing which had ultimately delayed the implementation of this feature into the final product. The team has decided to involve more members to ensure the implementation of this task while covering all the bugs that have come up.

Overall, we have achieved most of the functional components that we had decided to complete in this phase. Furthermore, we have added additional optional functionalities which we had worked on such as the “post comments”  and “post like”.

For phase 3 , we will continue to adopt the approach we followed in phase 2 such as a sequential categorisation of tasks , regular team meetings for updates on tasks allotted and reviewing all the work every week. A few pending enhancements from phase 2 will be accorded in phase 3.

| | | | |
|---|---|---|---|
| Phase 3 | Phase 2 pending functional requirements implementation<br><br>&<br><br>Finalize the code and Website Launch | In Phase 3,<br><br>1. Make sure the issues that crept up in phase 1 and phase 2 are thoroughly avoided.<br>2. Complete the pending requirements from phase 1 and phase 2 this includes<br>a. Forgot password on home page<br>b. Event Delete<br>c. Event update<br>d. UI Enhancements<br>3. Review and make final adjustments to the website code before freezing it.<br>4. Initiate User Acceptance Testing (UAT) and address any issues identified during this phase.<br>5. Proceed with the deployment process, addressing various deployment-related tasks.<br>6. Once all tasks are completed, the website will go live and be ready for use.<br><br>Criticality: Handling different kind Deployment issues and responsiveness of the website. | April 28 2024<br>Deliverable 5 |

# UML DIAGRAMS

## i) Class Diagram

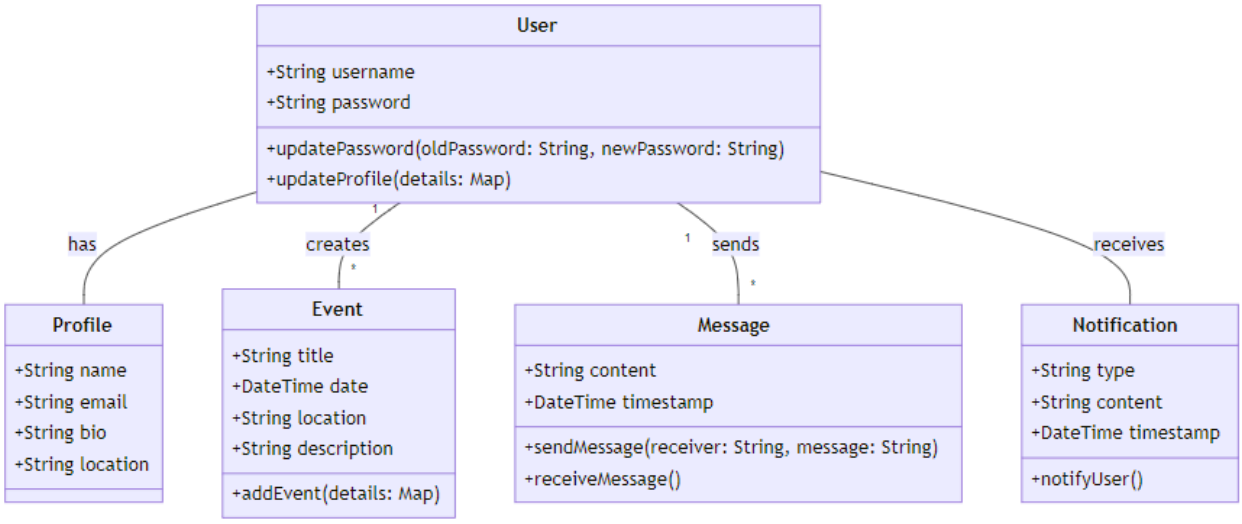


Fig 2.B.1

## ii) Sequence Diagrams

### a) Update Password

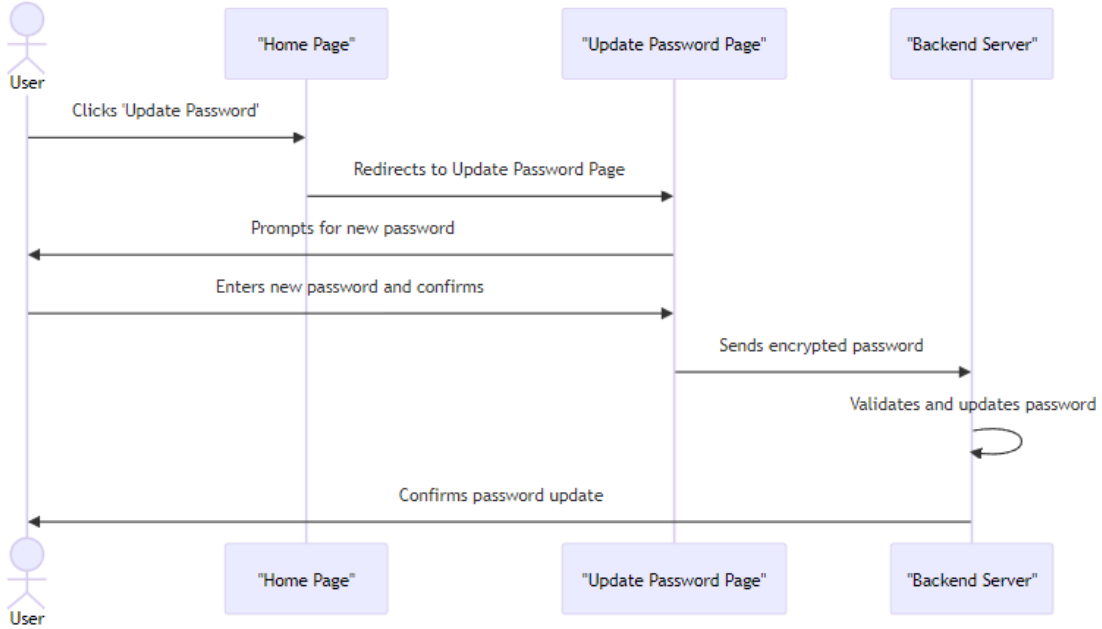


Fig 2.B.2

## b)Update Profile

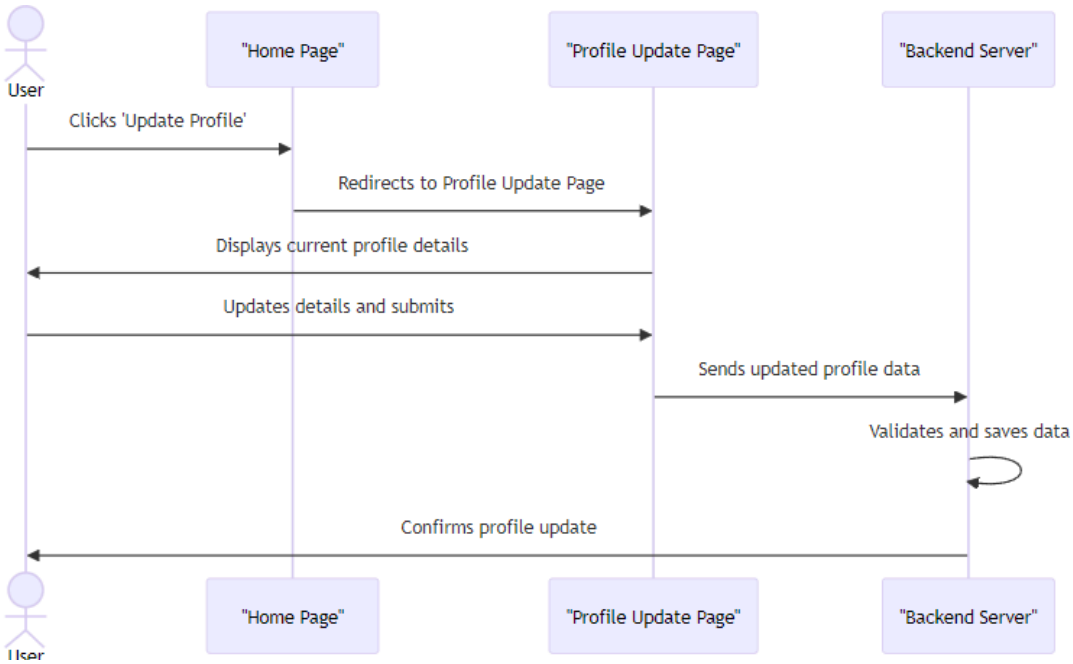


Fig 2.B.3

## c)Create Event

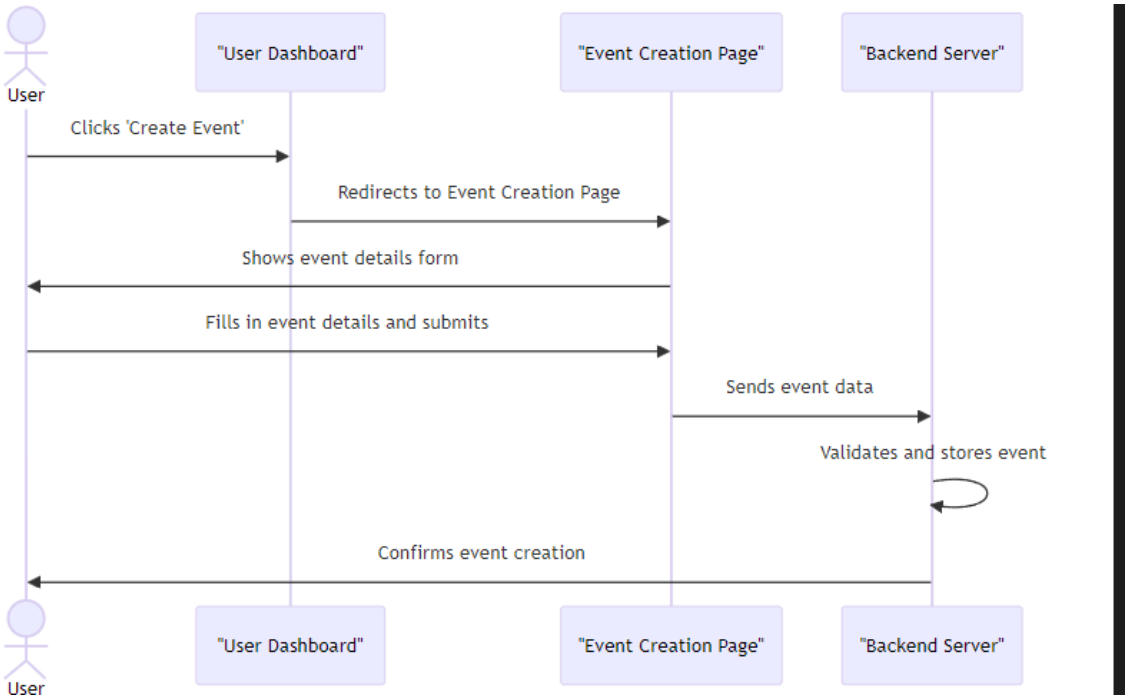


Fig 2.B.4

**d)Messages**

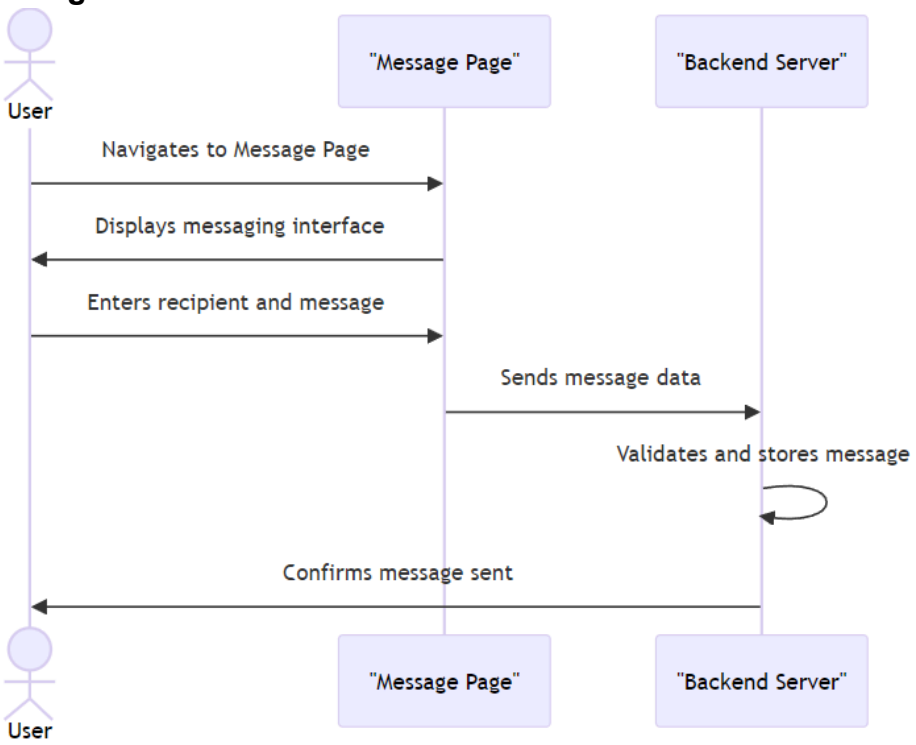


Fig 2.B.5

**e)Notifications**

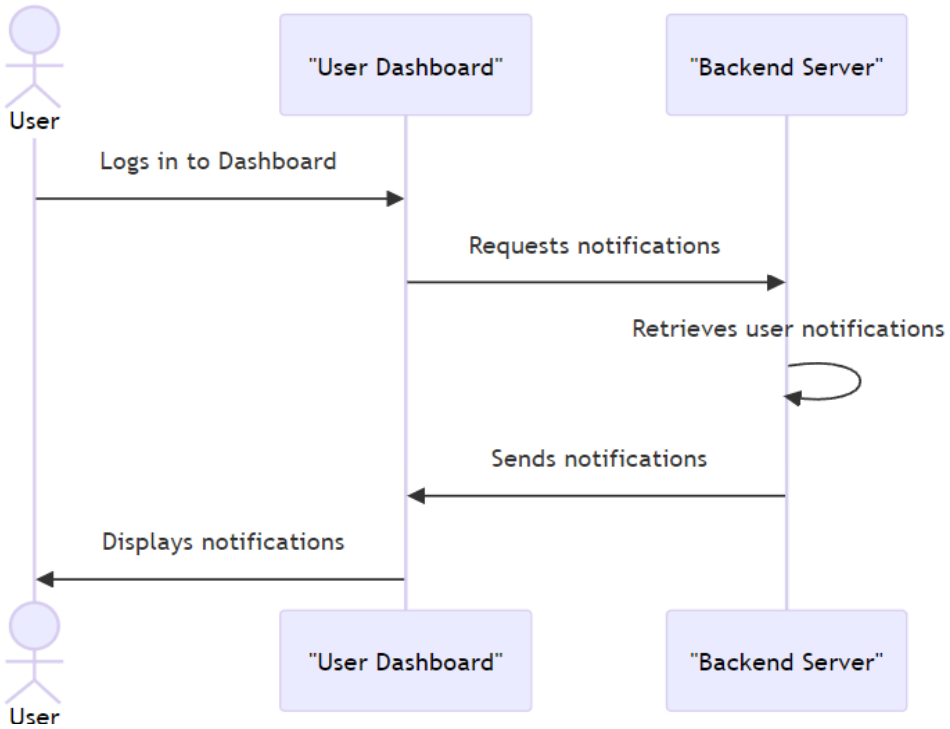


Fig 2.B.6

### iii) Use Case Diagrams

#### a) Normal Scenario

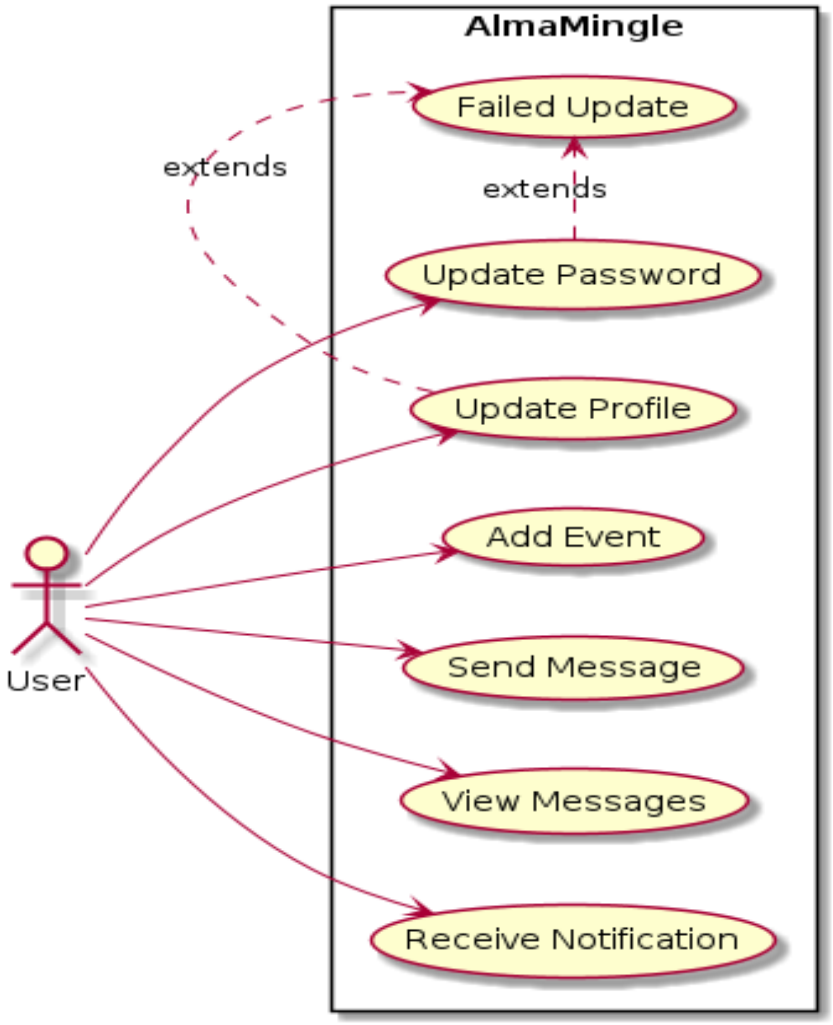


Fig 2.B.7

**b) Failure Scenario**

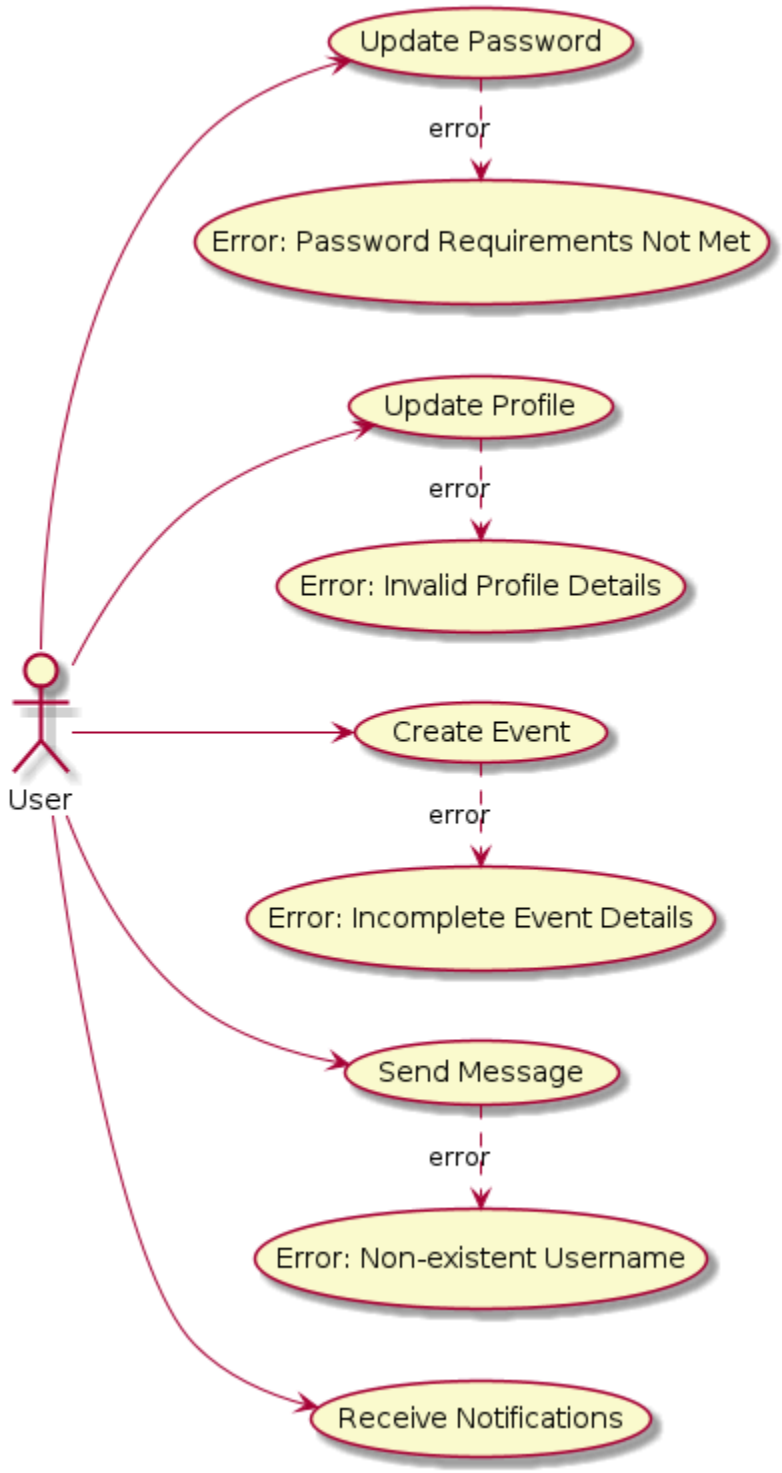


Fig 2.B.8

# UNIT TEST CASES

For our web platform, to ensure the smooth process, we have tested our code for different scenarios using jest framework.

## C.1 Signup.test.js

These test cases cover both successful and unsuccessful scenarios of user signup and ensure proper functioning of the component under different conditions.

**Successful Signup:**
**Description:** Tests whether the component successfully signs up a user when valid input is provided and navigates to the login page upon successful signup.
**Input:** Fill all input fields with valid data.
**Output:** Axios post request is made with the correct data, and upon successful signup, the user is navigated to the login page with an alert message indicating successful signup.

**Unsuccessful Signup:**
**Description:** Tests whether the component displays an error message when sign up fails due to an error response from the server.
**Input:** Fill all input fields with valid data, but mock the axios request to return an error response.
**Output:** Axios post request is made with the correct data, and an error alert is displayed indicating a failure in signup.

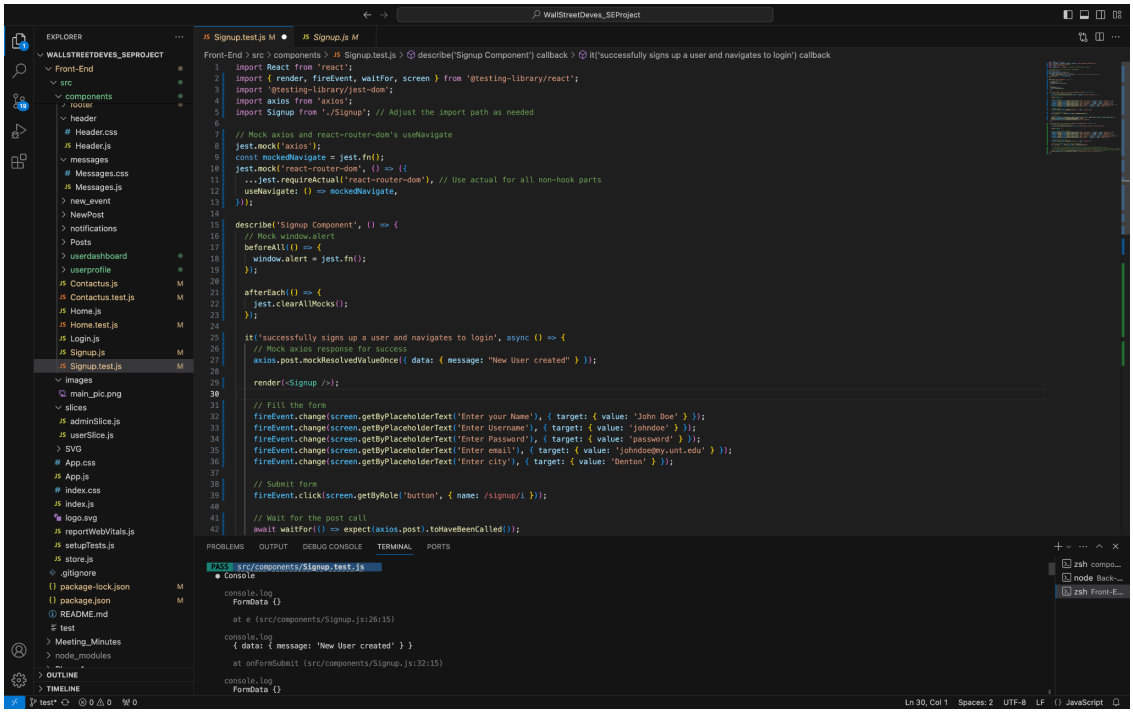

**Fig:C.1**

Additional test cases are going to be implemented to test for edge cases involving user input at signup. These test cases are testing to make sure valid information is being entered and that entered data is being handled correctly so that future use of the info can be used in function calls in the backend. These are all sample test cases that test for characters, numbers and special characters and ensure that information is entered and that they get stored as strings for future use.

```
import React from 'react';
import { render } from '@testing-library/react';
import Signup from './Signup';

describe('Signup', () => {
    test('renders Signup', () => {
      // Render Signup
      const { Text, oText } = render(<Signup />);

      const logindetail = Text('');
      expect(logindetail).toBeDefined();
      expect(logindetail).typeof().toBe("string");

      // testing for login data is handled correctly
      // letters only
      const info = oText(/abc/i);
      expect(info).toBeInTheDocument();
      expect(info).toBeDefined();
      expect(info).typeof().toBe("string");

      // numbers only
      const info2 = oText(/123/i);
      expect(info2).toBeInTheDocument();
      expect(info2).toBeDefined();
      expect(info2).typeof().toBe("string");

      // special characters only
      const info3 = oText(/!@#/i);
      expect(info3).toBeInTheDocument();
      expect(info3).toBeDefined();
      expect(info3).typeof().toBe("string");
```

```
      // letters and numbers
      const info4 = oText(/abc123/i);
      expect(info4).toBeInTheDocument();
      expect(info4).toBeDefined();
      expect(info4).typeof().toBe("string");

      // letters and special characters
      const info5 = oText(/abc!@#/i);
      expect(info5).toBeInTheDocument();
      expect(info5).toBeDefined();
      expect(info5).typeof().toBe("string");

      // letters special characters and numbers
      const info6 = oText(/abc123!@#/i);
      expect(info6).toBeInTheDocument();
      expect(info6).toBeDefined();
      expect(info6).typeof().toBe("string");

    });
});
```

**Fig C.1.2 and C.1.3**

## C.2 Home.test.js

This test ensures that the `Home` component renders successfully and contains the expected welcome message "Welcome to AlmaMingle: Connect, Learn, Thrive!".

The purpose of this test is to verify that the `Home` component displays the correct introductory message, providing users with a welcoming experience when they land on the homepage of the AlmaMingle application. By checking if the specified text is present in the rendered component, we confirm that the component renders as expected and contains the essential content.

This test is crucial for maintaining consistency in the user experience and ensuring that the introductory message is prominently displayed to users visiting the application's home page. It helps catch any unexpected changes or errors in the rendering of the `Home` component that could affect user engagement or comprehension of the application's purpose.

**Input:** `Home` component is rendered.

**Output:** Presence of the text "Welcome to AlmaMingle: Connect, Learn, Thrive!" within the rendered `Home` component is verified.

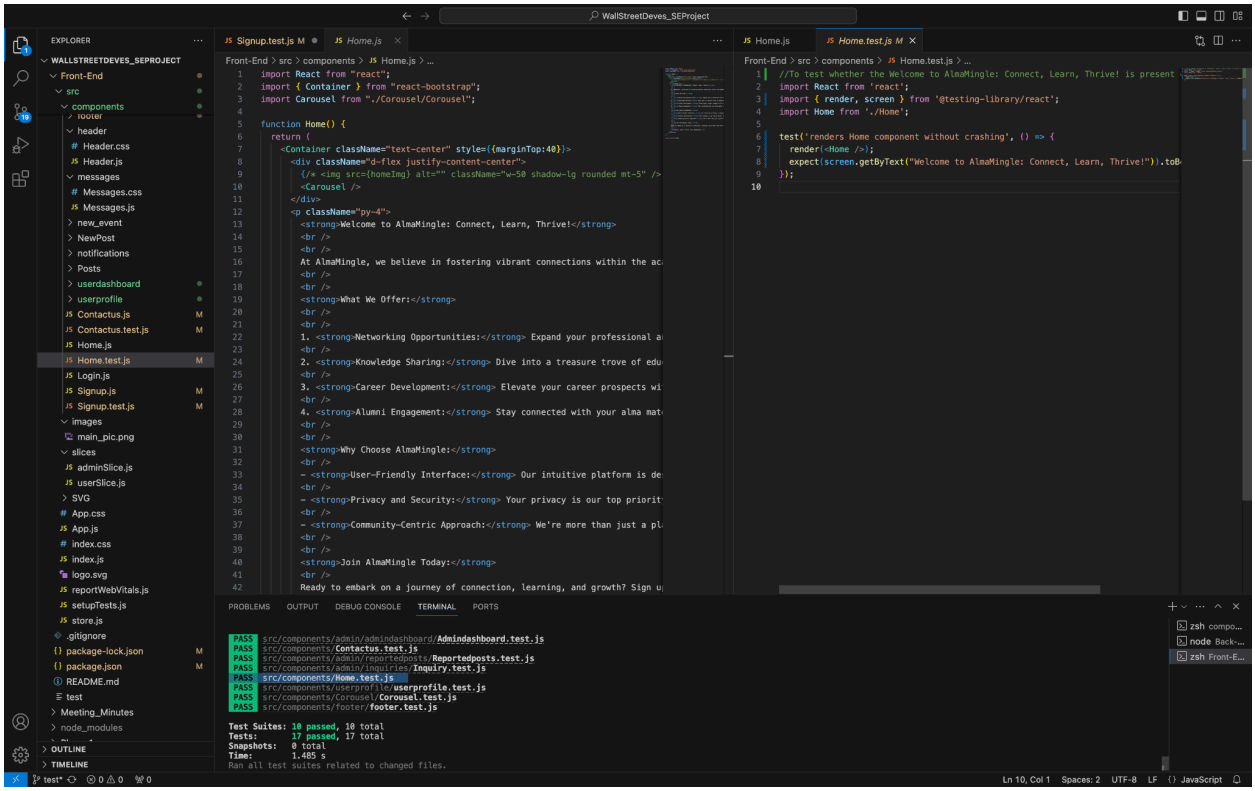

**Fig:C.2**

# C.3 Contactus.test.js

This test case verifies the behavior of the `ContactUs` component by simulating a form submission. It ensures that when the form is submitted with valid input, a success message is displayed indicating that the message has been sent successfully.

**Input:**
- Fill out the form with valid data:
  - Name: “John Doe”
  - Email: “johndoe@example.com”
  - Message: “Hello, this is a test message.”

**Output:**
- The component successfully submits the form.
- The axios post request is made with the correct data.

- After form submission, the component displays a success message "Message sent successfully".
- The test verifies that the success message appears in the DOM and that the axios post function is called with the expected data.

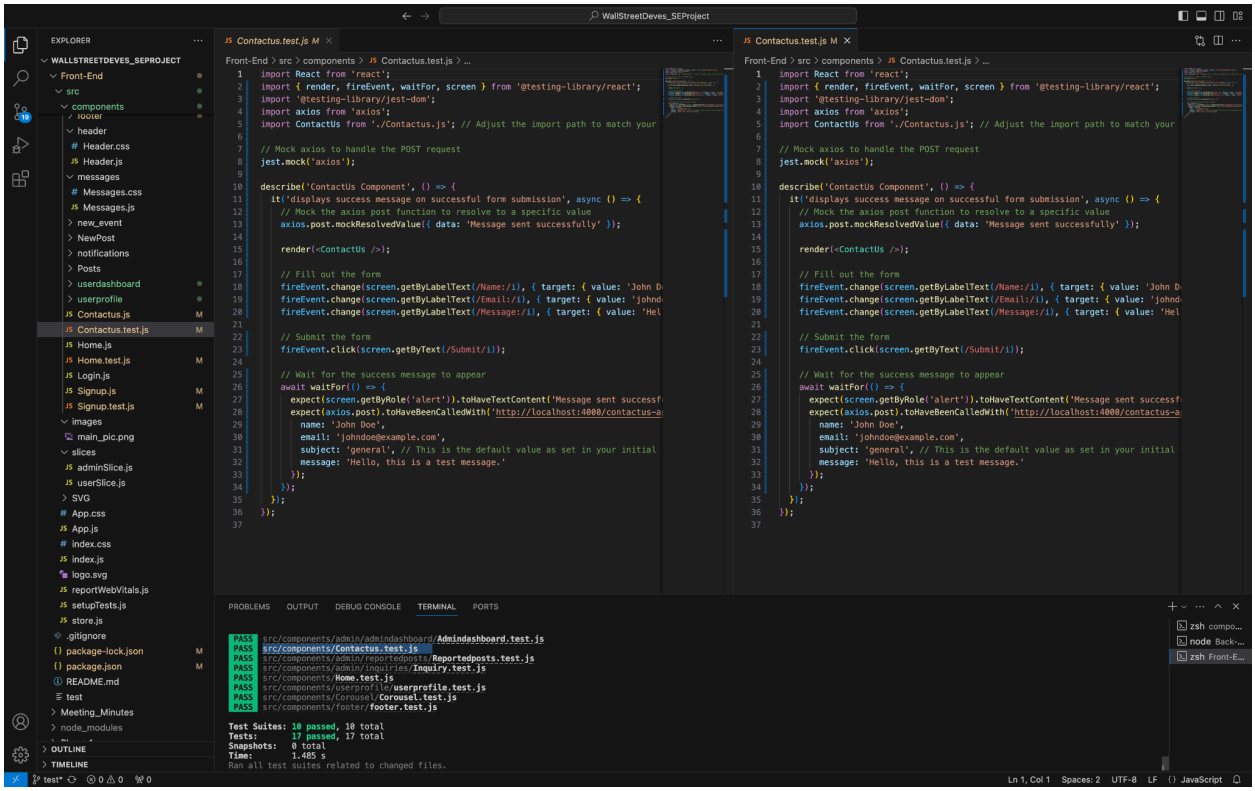


**Fig:C.3**

# C.4 Inquiry.test.js

This test case validates the behavior of the `Inquiry` component by ensuring that it correctly fetches and displays inquiries from an API endpoint. It simulates a GET request to the specified endpoint (`http://localhost:4000/contactus-api/get-inquiry` in this case) and expects the component to render the fetched inquiries appropriately.

**Input:**

- Mocked GET request to the API endpoint `/contactus-api/get-inquiry`.
- Sample data representing inquiries fetched from the API:

*javascript*

```
const inquiries = [
  { _id: '1', name: 'John Doe', email: 'john@example.com', subject: 'Test Subject', message: 'Test Message' },
  { _id: '2', name: 'Jane Doe', email: 'jane@example.com', subject: 'Another Test Subject', message: 'Another Test Message' }, ];
```

**Output:**

- The `Inquiry` component successfully fetches inquiries from the API endpoint.
- For each inquiry fetched:
  - The component displays the name of the sender (`inquiry.name`).

- The component displays the email of the sender (`inquiry.email`).
- The component displays the subject of the inquiry (`inquiry.subject`).
- The component displays the message of the inquiry (`inquiry.message`).

- The test verifies that each inquiry's details are rendered correctly within the component.

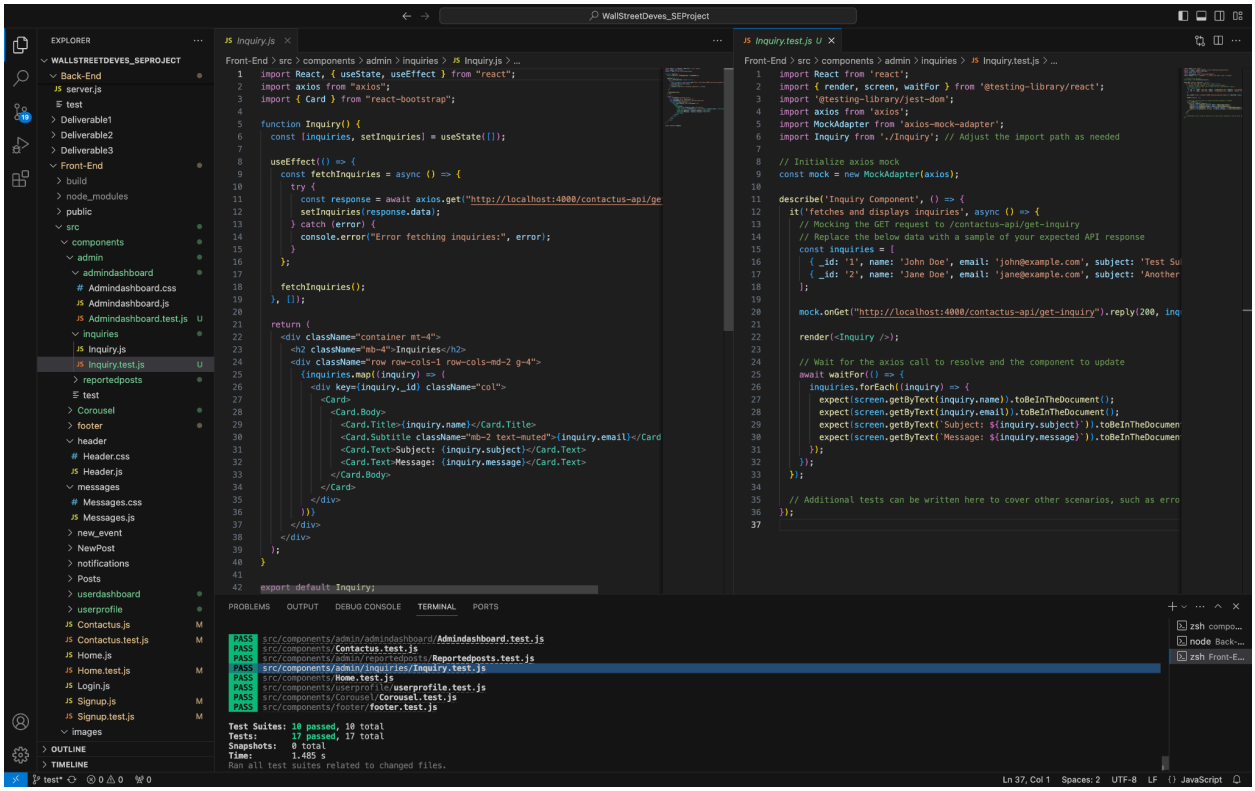

**Fig:C.4**

# C.5 ReportedPosts.test.js

This test case validates the behavior of the `Reportedposts` component by testing its ability to handle post deletion. It simulates the process of fetching reported posts, displaying them, and then deleting a post. It verifies that the post is removed from the UI after deletion.

**Input:**

- Mocked GET request to fetch reported posts from the API endpoint `http://localhost:4000/post-api/reportedposts`.
- Mocked DELETE requests for deleting a post and its associated report from the API endpoint `http://localhost:4000/post-api/delete-post/post1` and `http://localhost:4000/post-api/report-post-delete/report1` respectively.
- Sample data representing reported posts before and after deletion:

*javascript*

```
const reportedPostsBeforeDeletion = [
  {
    _id: "report1",
    count: 3,
    post: {
```

```
      _id: "post1",
      title: "Post 1",
      content: "Content 1",
      category: "Category 1",
      visibility: "Public",
      createdBy: "User 1"
    }
  }
];

const reportedPostsAfterDeletion = [];
```

**Output:**

- The `Reportedposts` component successfully fetches reported posts from the API endpoint.
- The fetched reported posts are displayed correctly in the UI.
- Upon clicking the delete button for a reported post, the component correctly sends a DELETE request to the API to delete the post and its associated report.
- After successful deletion, the deleted post is no longer displayed in the UI.
- The test verifies that the post is removed from the UI after deletion, ensuring proper handling of post deletion by the component.

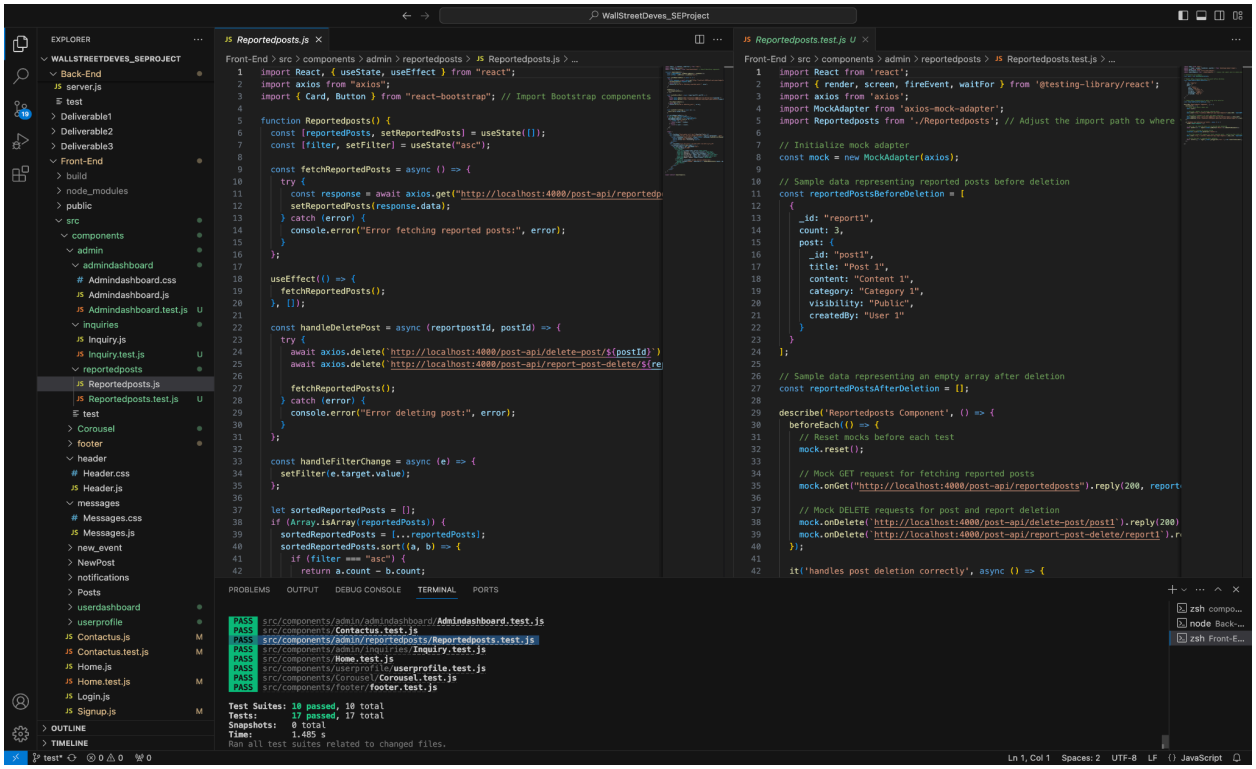

**Fig: C.5**

## C.6 AdminDashboard.test.js

These test cases validate the functionality of the `Admindashboard` component, specifically its ability to send messages. Two scenarios are tested: successful message sending and failed message sending.

**Input:**

**1. Successful Message Sending:**
- Render the `Admindashboard` component.
- Mock a successful POST request to the `/broadcast-api/send-message` endpoint.
- Simulate typing into the message textarea.
- Simulate form submission by clicking the "Send Message" button.

**2. Failed Message Sending:**
- Render the `Admindashboard` component.
- Mock a failure response for the POST request to the `/broadcast-api/send-message` endpoint.
- Simulate typing into the message textarea.
- Simulate form submission by clicking the "Send Message" button.

**Output:**

**1. Successful Message Sending:**
- The component successfully sends the message.
- The success message "Message sent successfully" is displayed in an alert.
- The test verifies that the success message appears in the DOM.

**2. Failed Message Sending:**
- The component fails to send the message due to a network error.
- The failure message "Failed to send message" is displayed in an alert.
- The test verifies that the failure message appears in the DOM.

These tests ensure that the `Admindashboard` component behaves correctly under different conditions when sending messages, handling both successful and failed scenarios appropriately.

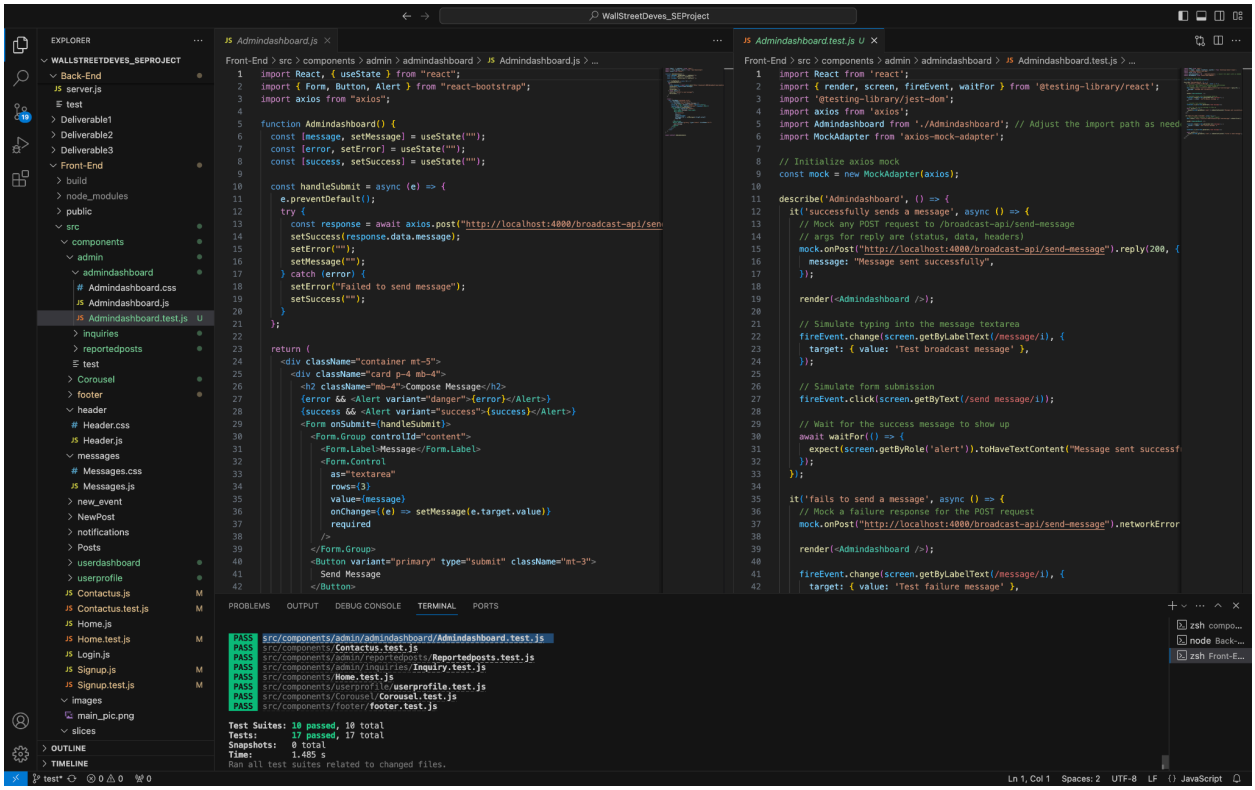

**Fig: C.6**

# C.7 Userdashboard.test.js

These test cases validate the functionality of the `Userdashboard` component. Two scenarios are tested: rendering and displaying the user's name, and navigation to the profile page when the "View Profile" link is clicked.

**Input:**

**1. Rendering and Displaying User Name:**
- Render the `Userdashboard` component wrapped in a Redux `<Provider>` with a mock store.
- Set up the initial state with a user object containing the name "John Doe".
- Check if the user's name is displayed in the rendered component.

**2. Navigation to Profile Page:**
- Render the `Userdashboard` component wrapped in a Redux `<Provider>` with a mock store.
- Use `MemoryRouter` and `Routes` from `react-router-dom` to simulate routing.
- Set up routes with the `Userdashboard` component at the root path and a mock profile page component at the "/profile" path.
- Simulate clicking the "View Profile" link.
- Check if the navigation to the profile page occurs by verifying the presence of the "Profile Page" text.

**Output:**

**1. Rendering and Displaying User Name:**
- The component successfully renders.
- The user's name "John Doe" is displayed in the component.
- The test verifies that the user's name appears in the DOM.

**2. Navigation to Profile Page:**
- Clicking the "View Profile" link navigates to the profile page.
- The profile page component content "Profile Page" is rendered.
- The test verifies that navigation to the profile page occurs and the content of the profile page is rendered.

These tests ensure that the `Userdashboard` component renders correctly and behaves as expected, displaying user information and facilitating navigation to other pages within the application.

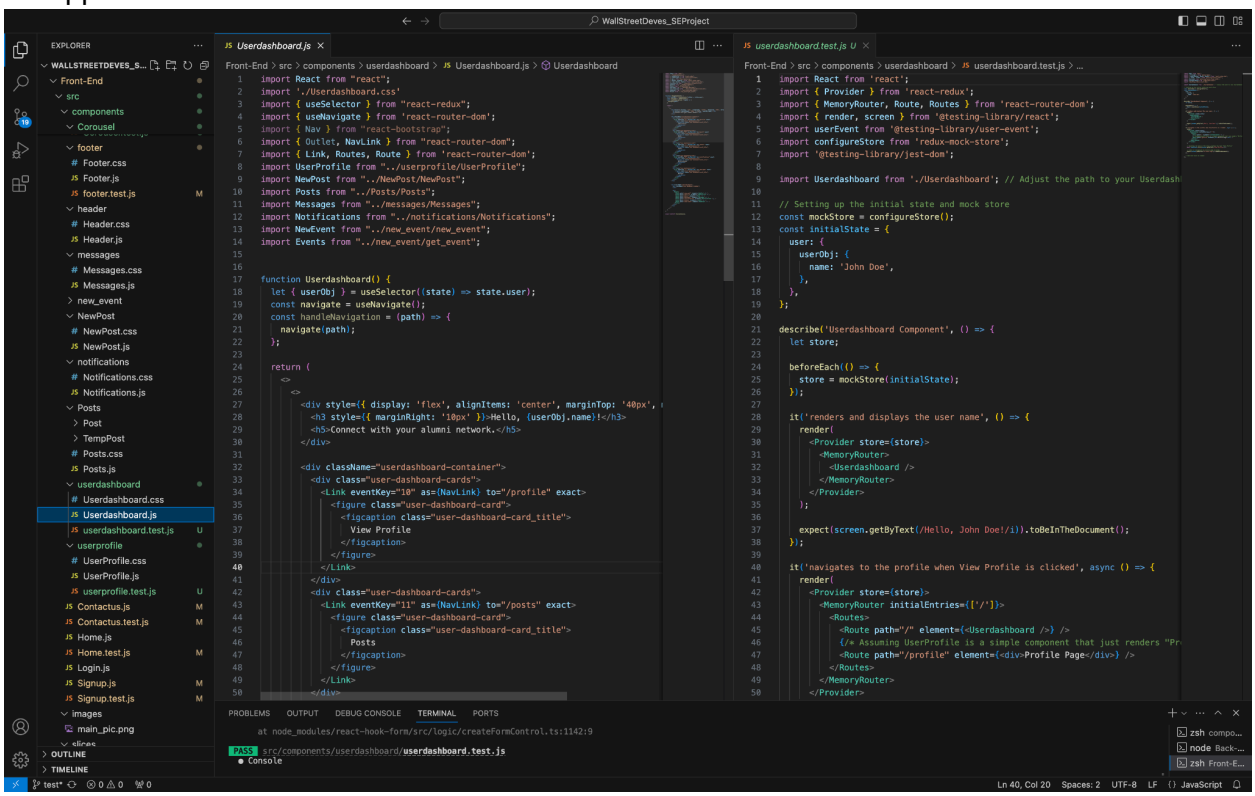

**Fig: C.7**

# C.8 Userprofile.test.js

These test cases verify the behavior of the `UserProfile` component, specifically testing its ability to render user profile information from the Redux store.

**Input: Rendering User Profile Information:**
- Render the `UserProfile` component wrapped in a Redux `<Provider>` with a mock store.

- Set up the initial state of the mock store with user profile information, including name, email, and username.
- Use `MemoryRouter` from `react-router-dom` to simulate routing.
- Check if the user profile information is correctly displayed in the rendered component.

**Output: Rendering User Profile Information:**

- The component successfully renders.
- The user's name, email, and username from the Redux store are displayed in the component.
- The test verifies that the user profile information appears in the DOM.

These tests ensure that the `UserProfile` component correctly renders user profile information from the Redux store, maintaining consistency with the application state. Additional tests can be added to simulate user interactions or to check for changes in the component's behavior.

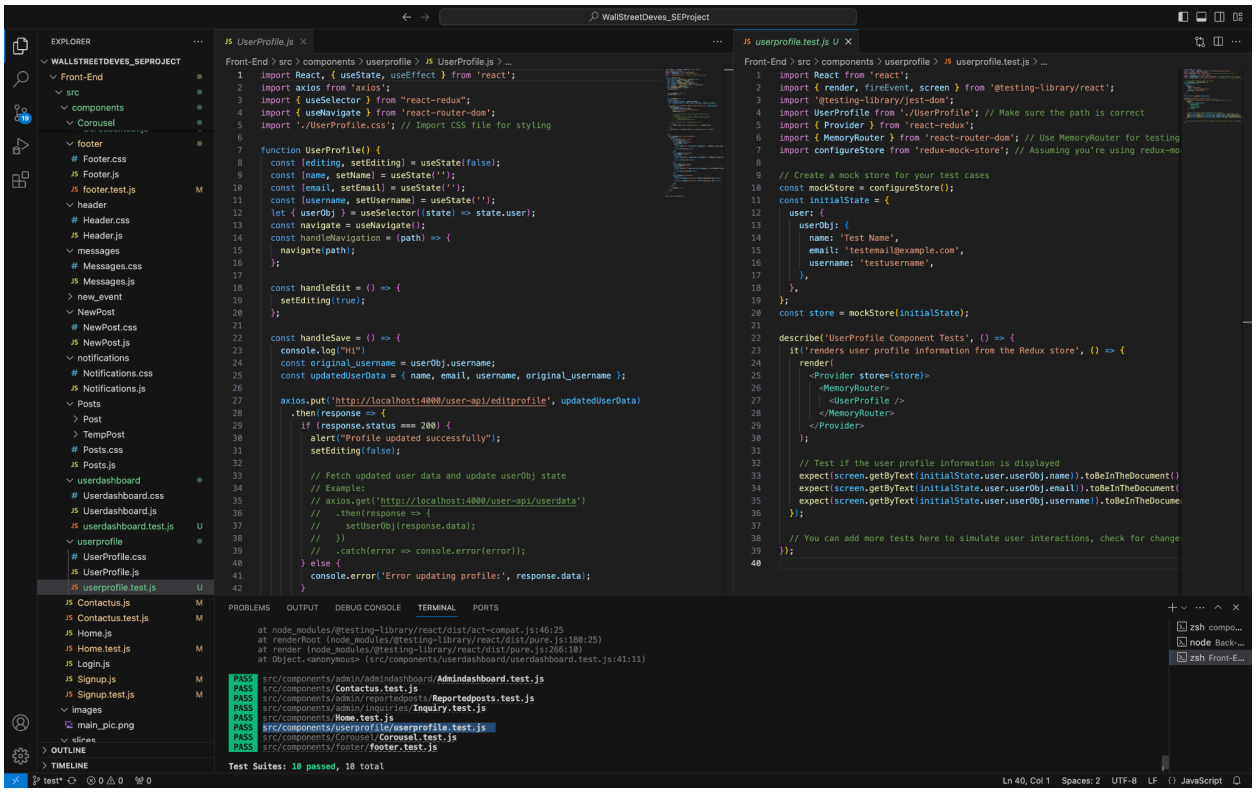

**Fig: C.8**

# C.9 Corousel.test.js

These test cases validate the functionality of the `Carousel` component, ensuring that it properly transitions between slides and responds correctly to user interaction with the next and previous buttons.

**Input:**

1. Automatically Move to Next Slide:
   - Render the `Carousel` component.
   - Check if the initial slide is displayed.
   - Fast-forward time by 3 seconds to simulate automatic slide change.
   - Check if the next slide is displayed after the specified time interval.

2. Move to Next Slide on Next Button Click:
   - Render the `Carousel` component.
   - Click the "next" button.
   - Check if the next slide is displayed.

3. Move to Previous Slide on Previous Button Click:
   - Render the `Carousel` component.
   - Move to the next slide first to ensure the carousel isn't on the first slide.
   - Click the "prev" button.
   - Check if the previous slide is displayed again.

**Output:**

1. Automatically Move to Next Slide:
   - The component automatically transitions to the next slide after the specified time interval.

2. Move to Next Slide on Next Button Click:
   - Clicking the "next" button successfully moves to the next slide.

3. Move to Previous Slide on Previous Button Click:
   - Clicking the "prev" button successfully moves to the previous slide.

These tests ensure that the `Carousel` component functions correctly, providing a smooth slideshow experience and responding appropriately to user interaction with the navigation buttons. Additionally, timers are cleaned up after each test to avoid interference with subsequent tests.

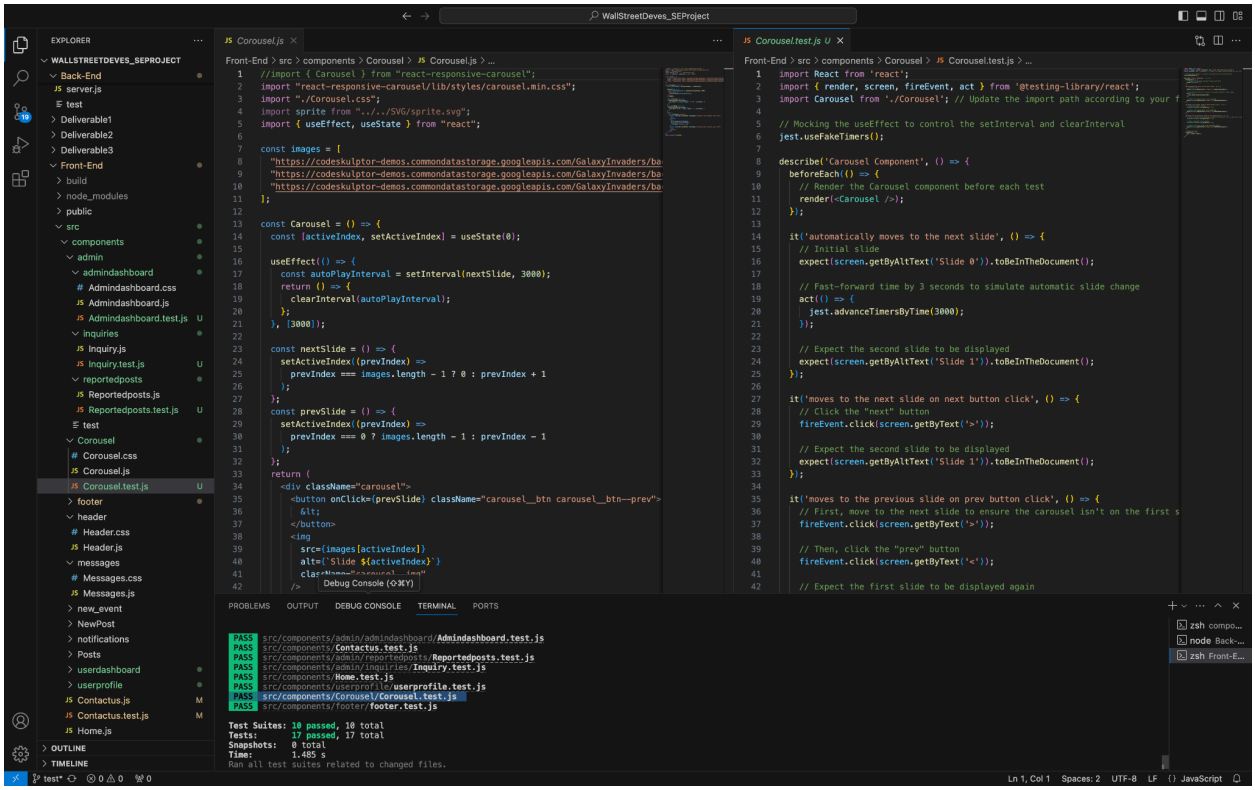

**Fig:C.9**

# C.10 Footer.test.js

These test cases verify the correctness of the `Footer` component by checking if it renders the footer text, the current year, and links to the terms of use and privacy policy correctly.

**Input:**

**1. Rendering Footer Text:**
- Render the `Footer` component.
- Check if the footer text containing "AlmaMingle" is rendered correctly.

**2. Rendering Current Year:**
- Render the `Footer` component.
- Get the current year.
- Check if the footer contains the current year.

**3. Rendering Terms of Use and Privacy Links:**
- Render the `Footer` component.
- Check if the terms of use and privacy links are rendered correctly.

**Output:**

**1. Rendering Footer Text:** The footer text containing "AlmaMingle" is rendered correctly.

**2. Rendering Current Year:** The current year is rendered correctly in the footer.
**3. Rendering Terms of Use and Privacy Links:** The terms of use and privacy links are rendered correctly in the footer.

These tests ensure that the `Footer` component displays the necessary information and links accurately, providing users with essential information and maintaining consistency in the application's user interface.

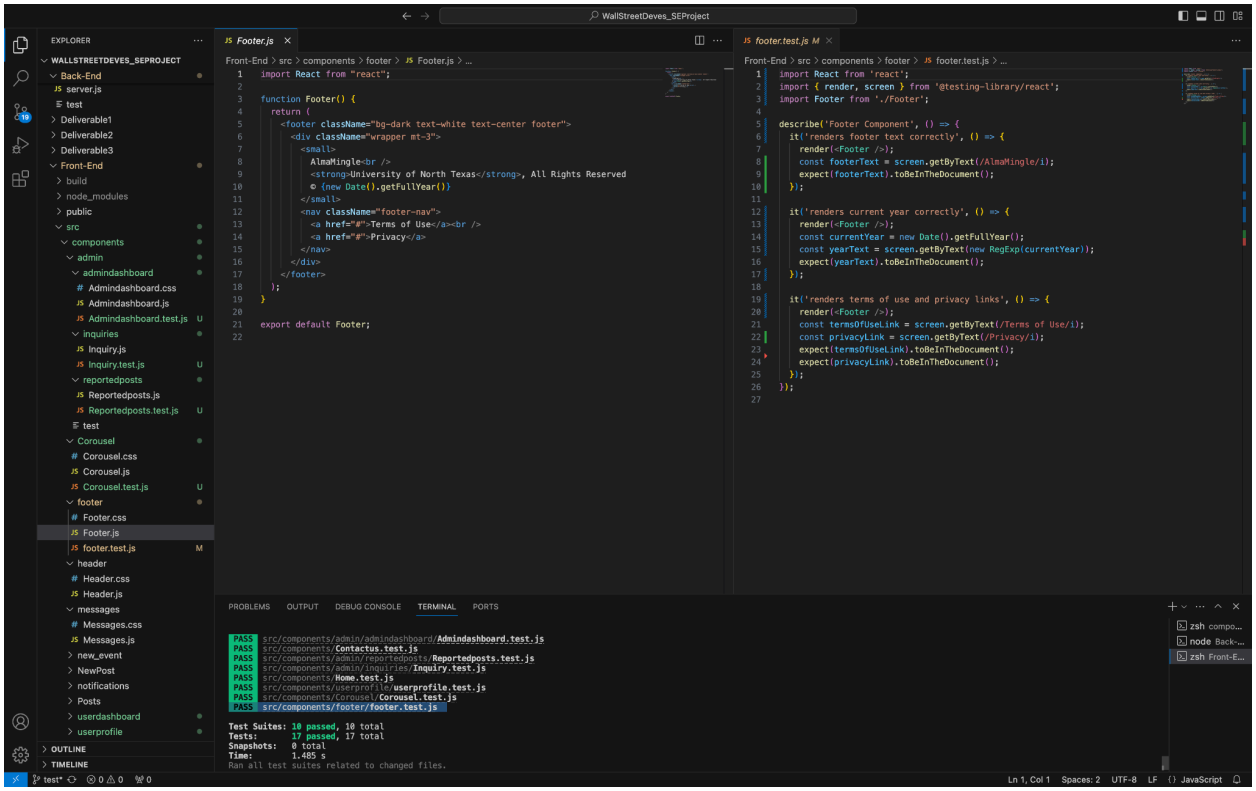


**Fig: C.10**

# C.11 testspec.js

Additional utility tests around deployment and security are conducted to ensure the product as a whole functions as expected and issues can be quickly diagnosed as the project moves into the final aspects of the software lifecycle. Checking for a proper response to the host, page loading and URL is properly working. Additional tests can be implemented around checking the protocol and network information can be implemented to better test for security.

```
const { error } = require("console");
const { Hash } = require("crypto");
const { response } = require("express");
const { url } = require("inspector");
var request = require("request");

var base_url = "http://localhost:3000/"

describe("SWE Project", function() {
  describe("GET /", function() {
    it("returns status code 200", function(done) {
      request.get(base_url, function(error, response, body) {
        expect(response.statusCode).toBeDefined();
        expect(response.statusCode).toBe(200);
        done();
      });
    });

    it("blank page loads", function(done) {
      request.get(base_url, function(error, response, body) {
        expect(body).toBeUndefined();
        done();
      });
    });

    it("check response", function(done) {
      request.get(base_url, function(error, response, body) {
        expect(response).toBeUndefined();
        done();
      });
    });
```

```
    it("Error response checking", function(done) {
      request.get(base_url, function(error, response, body) {
        expect(error).toBeDefined();
        done();
      });
    });

    it("deployment URL active", function(done) {
      request.get(base_url, function(error, response, body) {
        expect(base_url).toBeDefined();
        done();
      });
    });

    it("complete URL", function(done) {
      request.get(base_url, function(error, response, body) {
        expect(base_url.substring).toBeDefined();
        expect(base_url.length).toBeDefined();
        expect(base_url.length).toBe(22);
        done();
      });
    });


  });
});
```

```
    Error: Timeout - Async function did not complete within 5000ms (set by jasmine.DEFAULT_TIMEOUT_INTERVAL)
  Stack:
        at <Jasmine>
        at listOnTimeout (node:internal/timers:573:17)
        at process.processTimers (node:internal/timers:514:7)

6 specs, 1 failure
Finished in 5.156 seconds
Randomized with seed 71074 (jasmine --random=true --seed=71074)
PS C:\Users\Andre\Documents\GitHub\WallStreetDeves_SEProject>
```

**Fig: C.11, C.12 and C.13**

*note the single failure is to be expected because the website has not been deployed and will not return a status code hence the time out error.

# USER MANUAL

## USER SIGNUP, LOGIN and DASHBOARD SCREENS

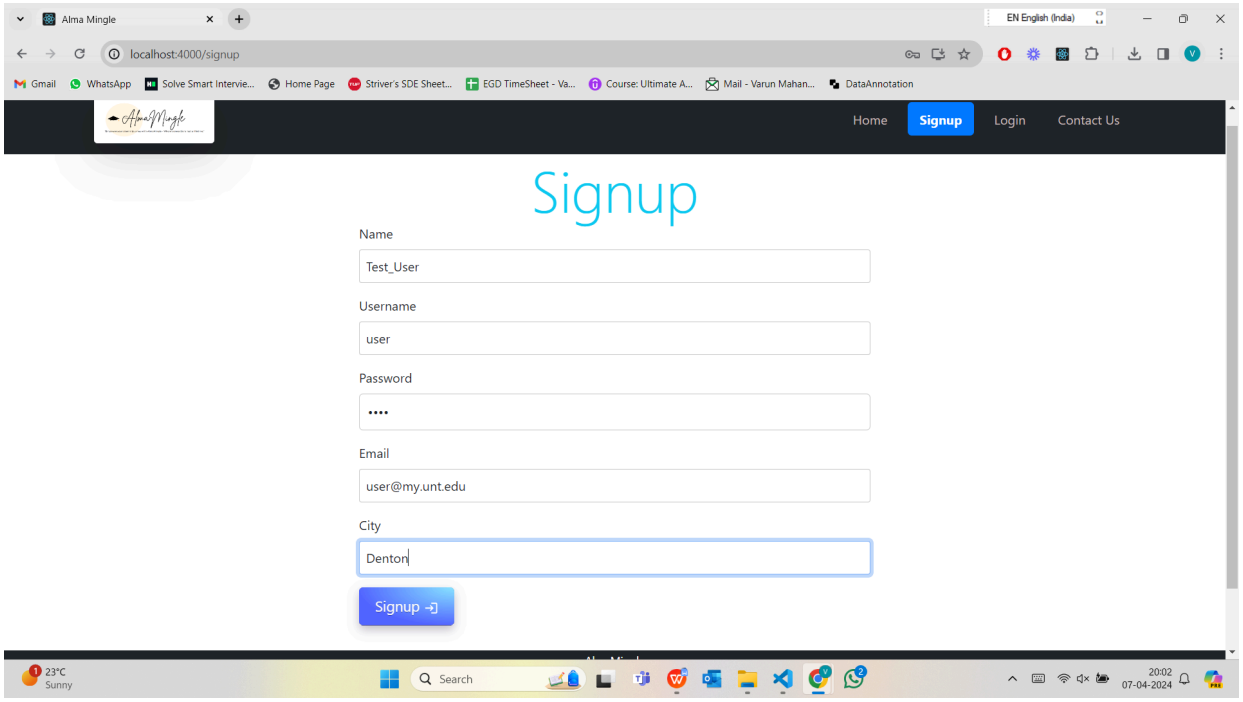


Fig D.1

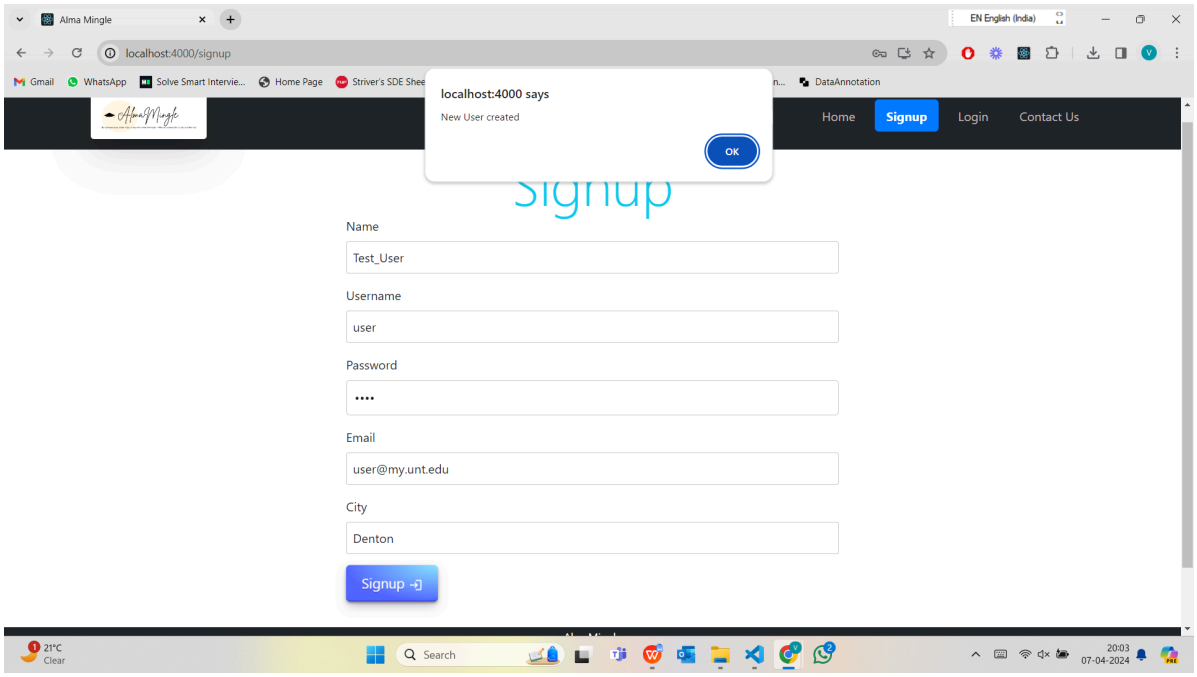


Fig D.2

Fig D.1 is the signup page for our application. This page is used to create a

profile for the UNT student. The user needs to enter Name, Username, Password, Email, and City for signing up. The email address should contain @my.unt.edu domain. After filling this information user clicks on signup and if all the information is correct there will be a popup saying New-user created as shown in Fig D.2.

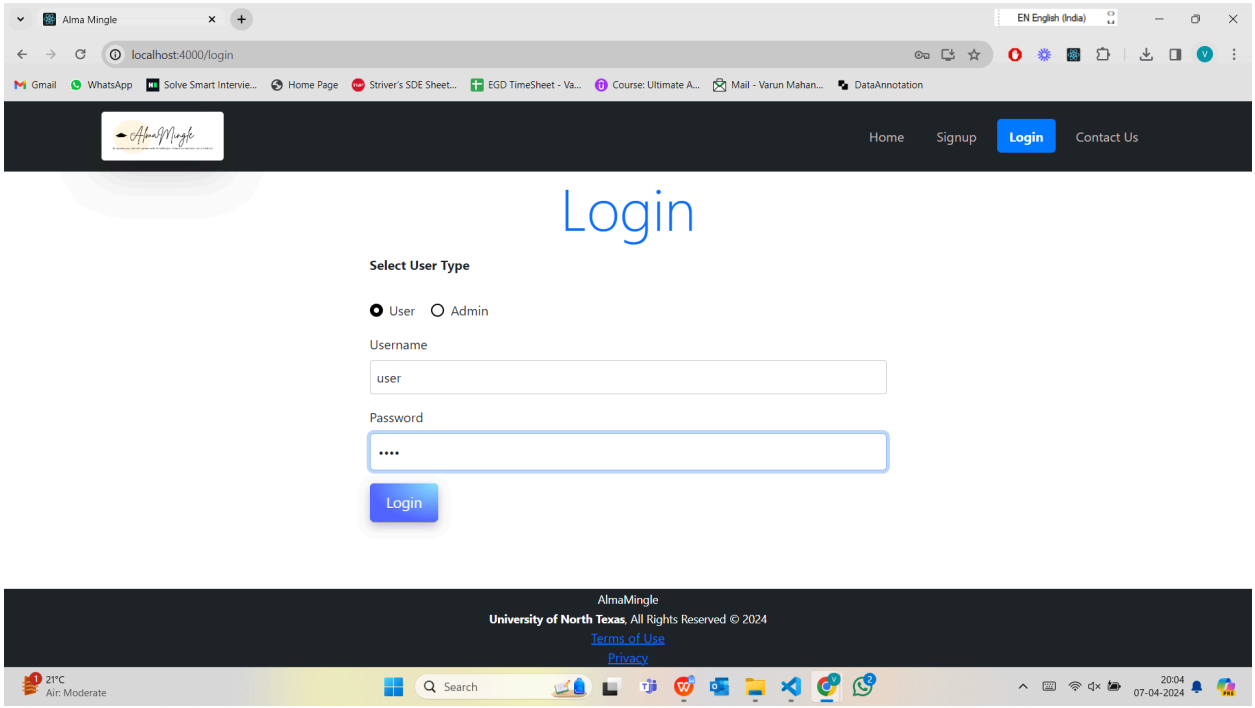


Fig D.3

Fig D.3 shows the login screen which the user or an admin can use to enter their credentials(i.e. Username and password). The User Type needs to be selected first before entering the login credentials. When the Login button is clicked, the screen will navigate to the User Dashboard screen.

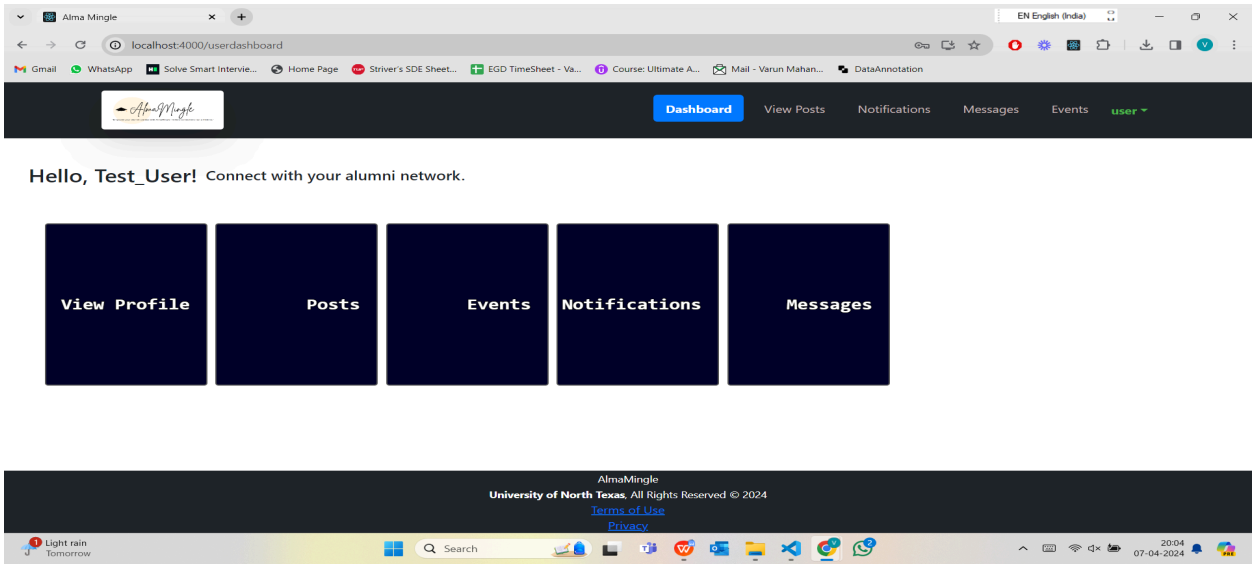


Fig D.4

Fig D.4 shows the User Dashboard screen. In the navbar, the user can see View Posts, Notifications, Messages, Events, and the user dropdown option. In the dropdown option user can see their profile or logout.

On the screen the user can see the welcome message to the user and then 5 cards below of the message. View Profile, Posts, Events, Notification and Messages.

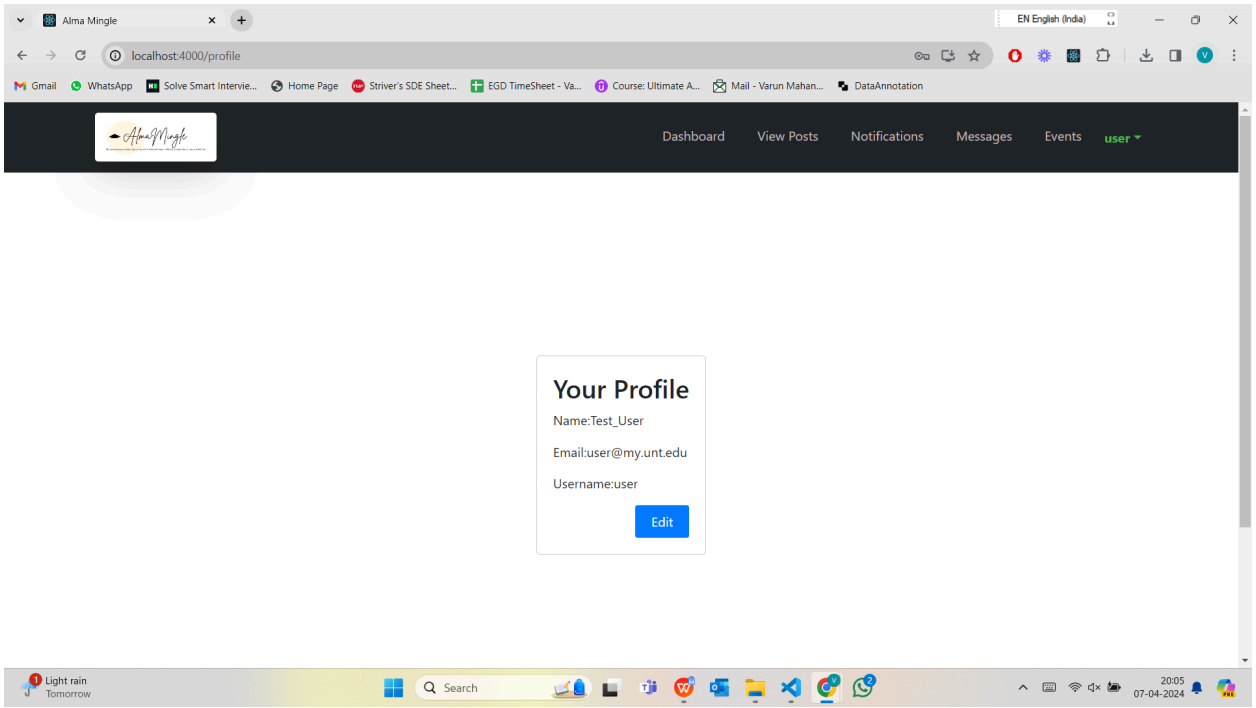


Fig D.5

Fig D.5 is the screen visible to users if the user clicks on the view profile. In this screen, the details of the user are visible such as Name, Email, and Username. If the user wants to edit the details the user can click on the edit button and the user will be able to edit its own information.

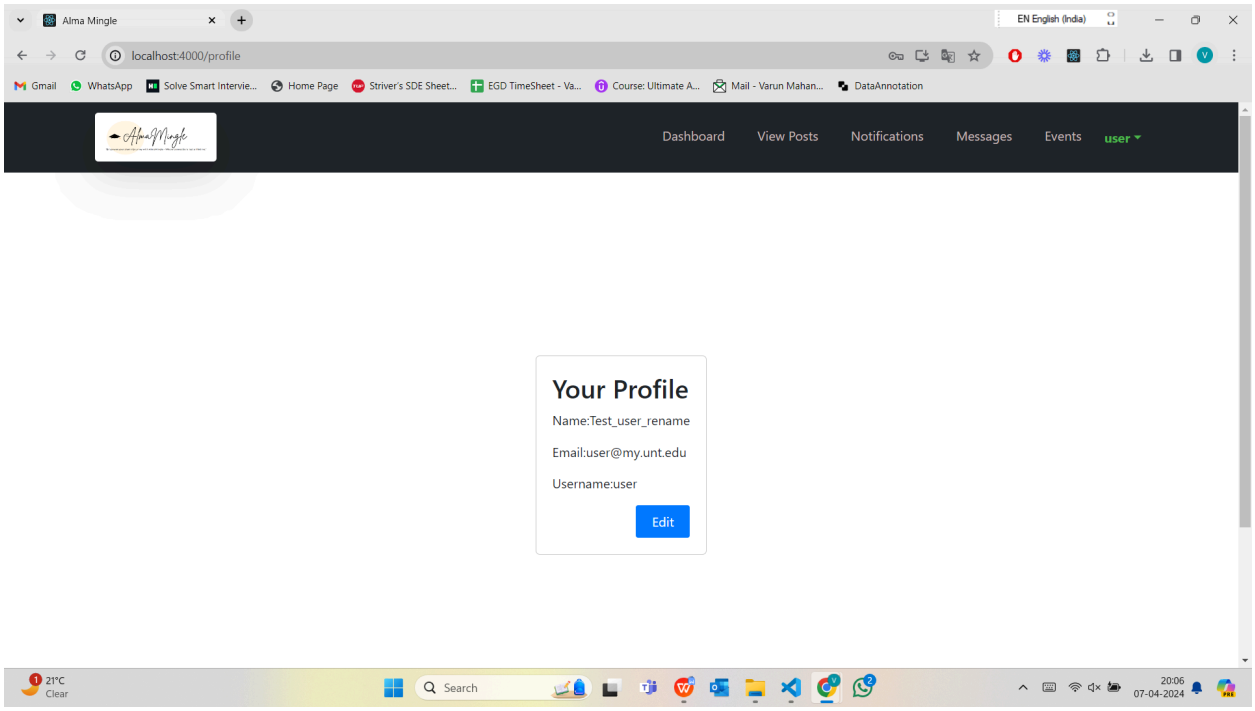


Fig D.6

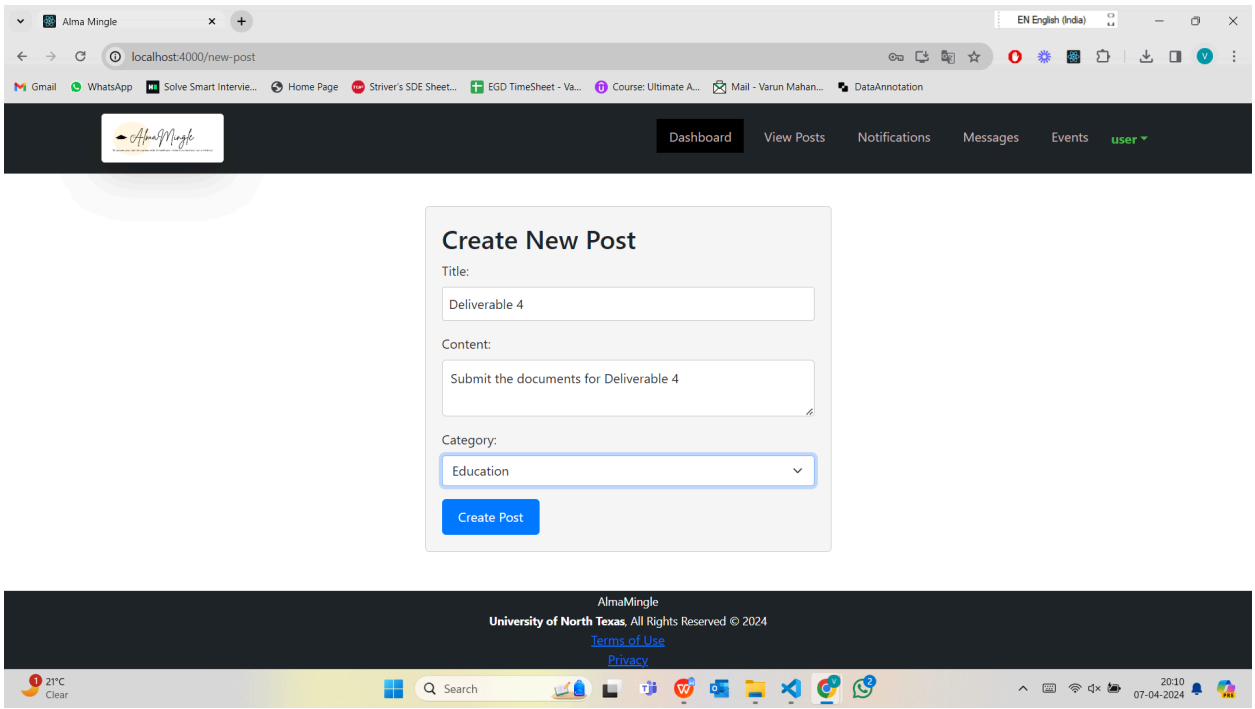


Fig D.7

When the Create New Post card is clicked, the screen in Fig 2.D.7 is shown. The user can create a post by entering the Title, Content, and Company fields. When the Create Post button is clicked, the post is created.

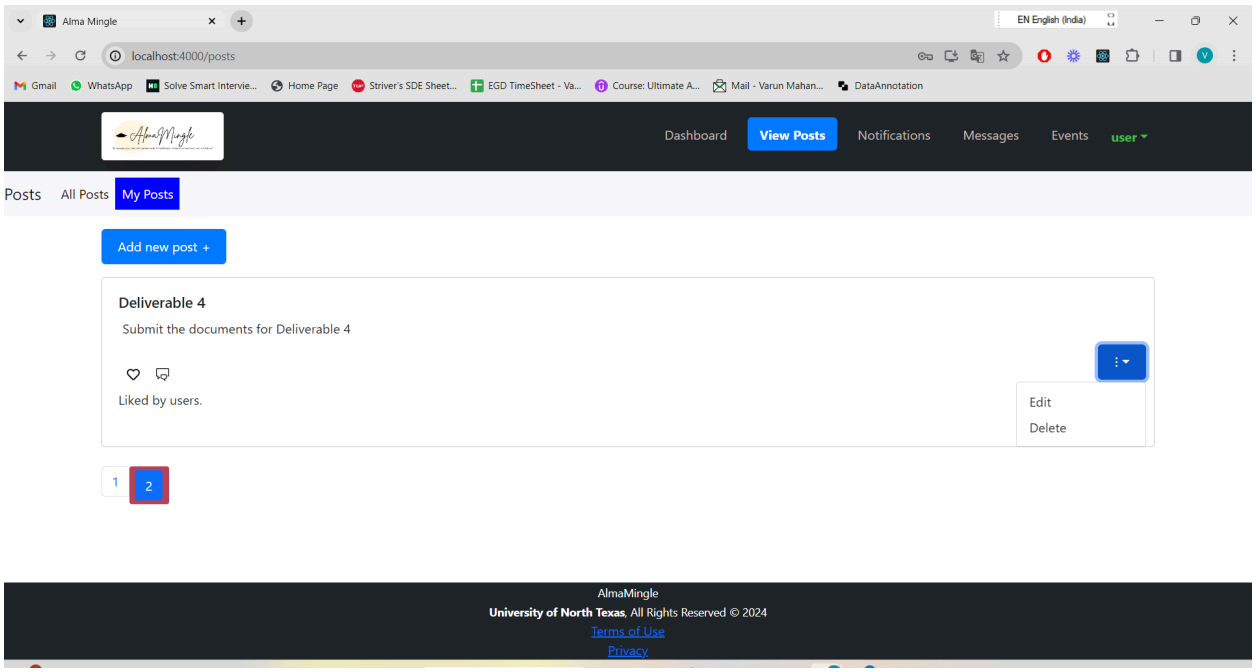


Fig D.8

In Fig D.8, the Posts Screen is shown. It shows all the posts posted by the current users. The user can edit or delete its own posts.

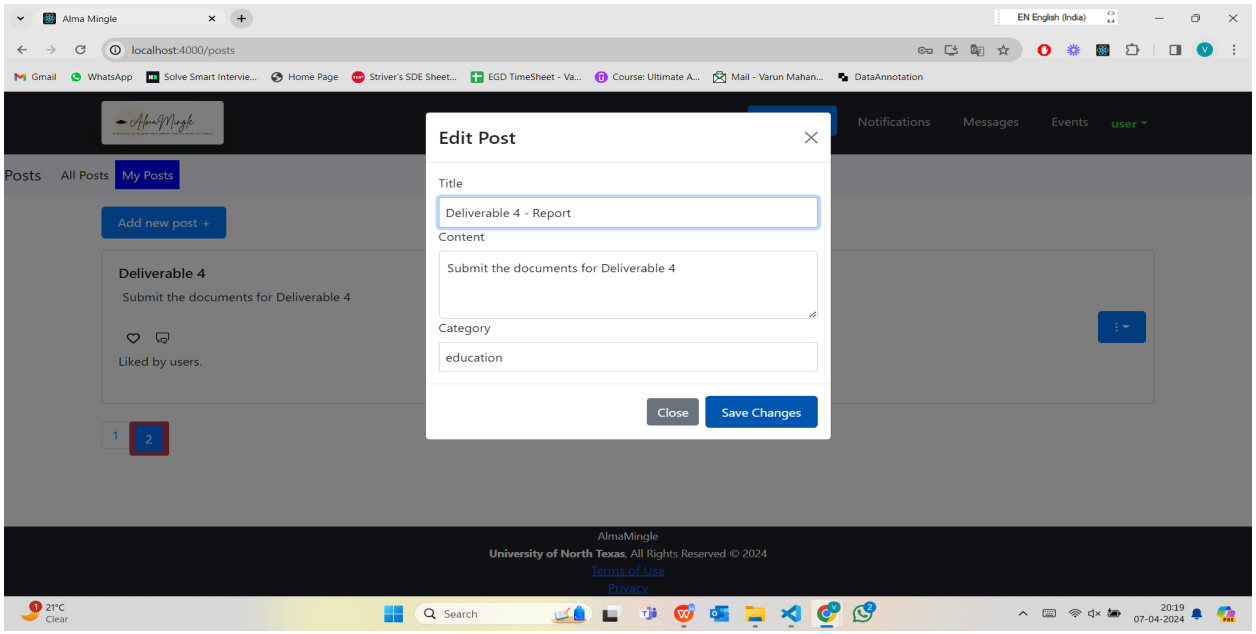


Fig D.9

If a user clicks on edit post then this screen will be shown where user can edit the details and click on save changes to update the details of its own post.

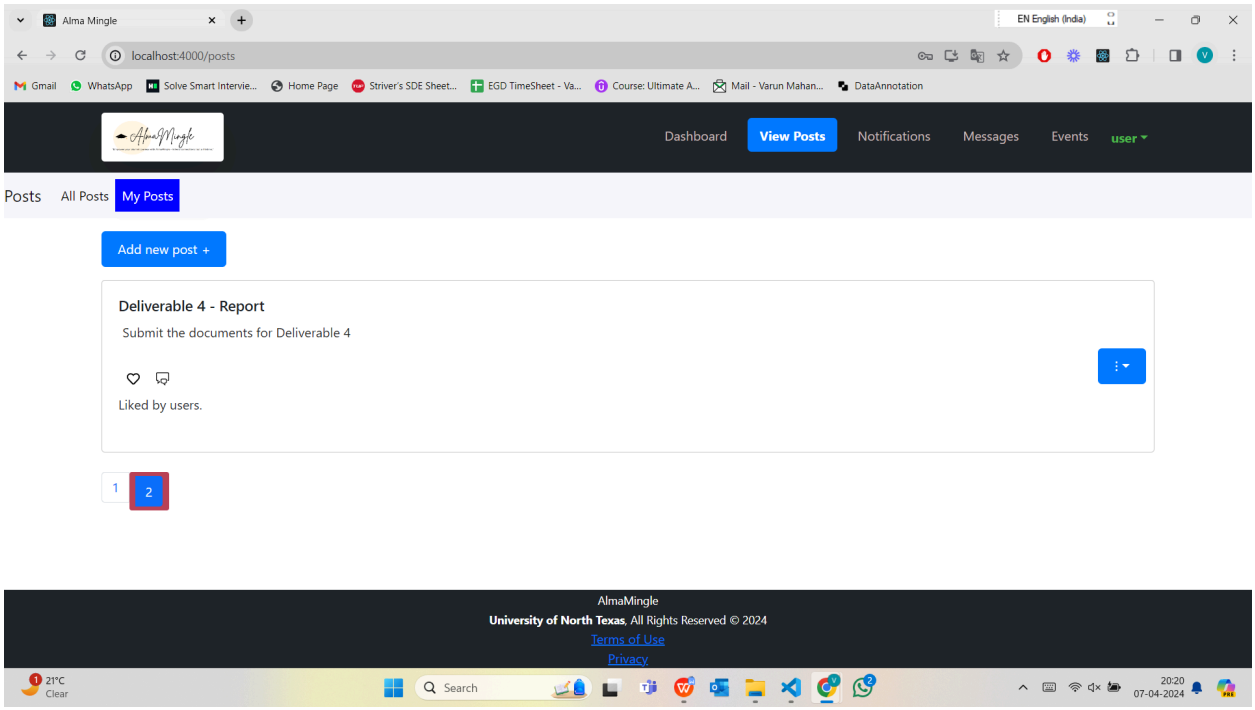


Fig D.10

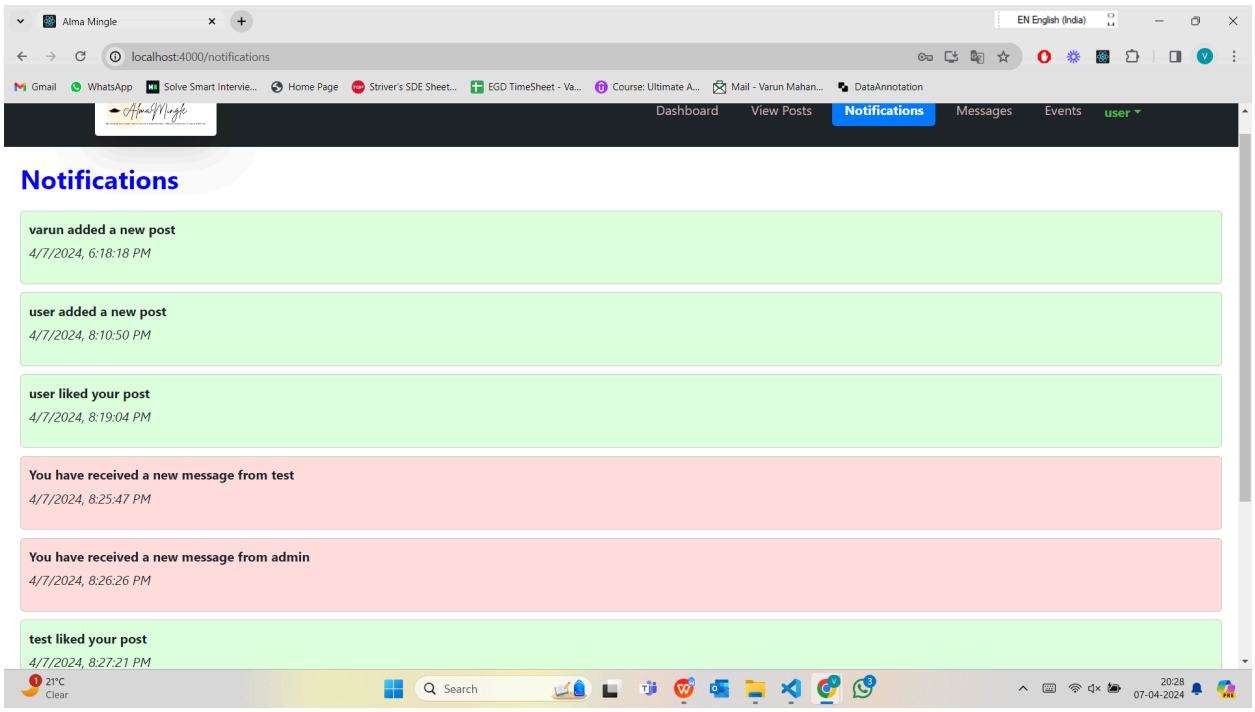


Fig D.11

Fig D.11 shows the notification screen whenever the user gets a notification it will

be shown here when the notification is unread it will be shown as read once the user clicks on that notification it will turn into red.

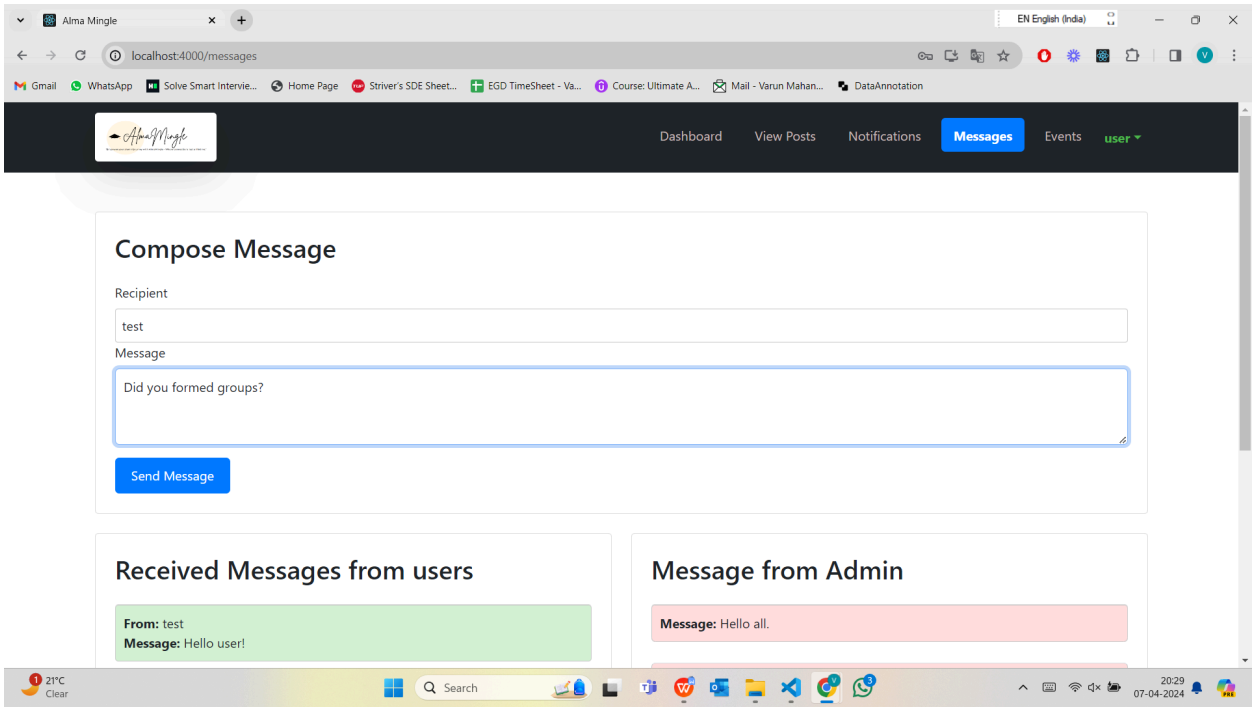


Fig D.12

Fig D.12 shows the message screen where the user can send a message to any user having a profile on the Alma Mingle portal. It also shows the received messages from other users or admins.

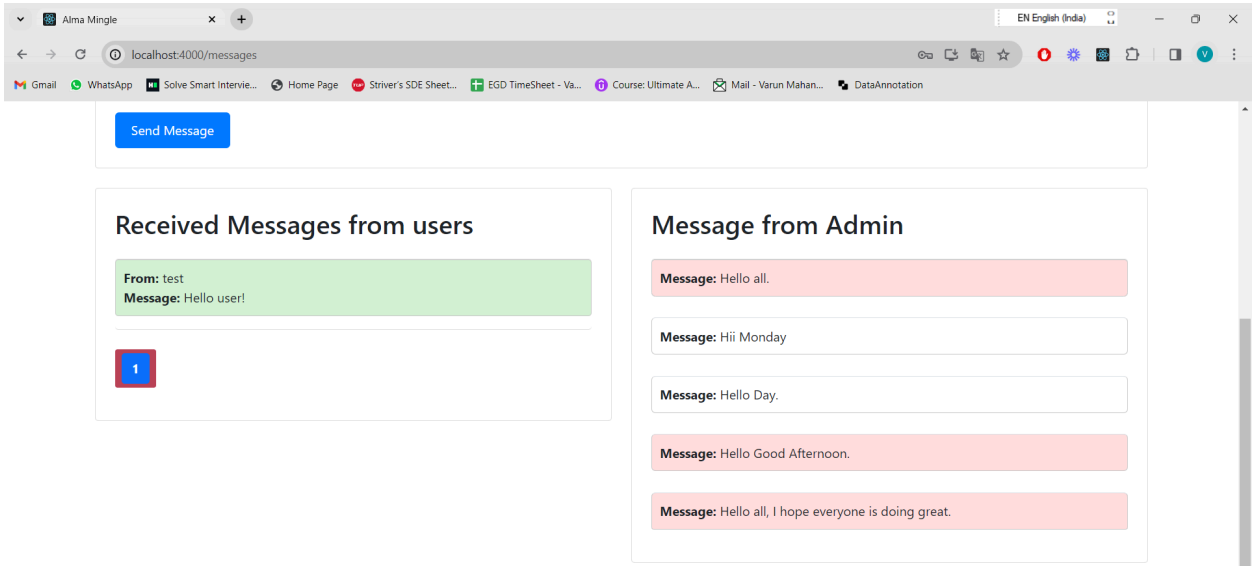


Fig D.13

If the message is unread it will be green and when it is clicked it is marked as read and it will turn into red.

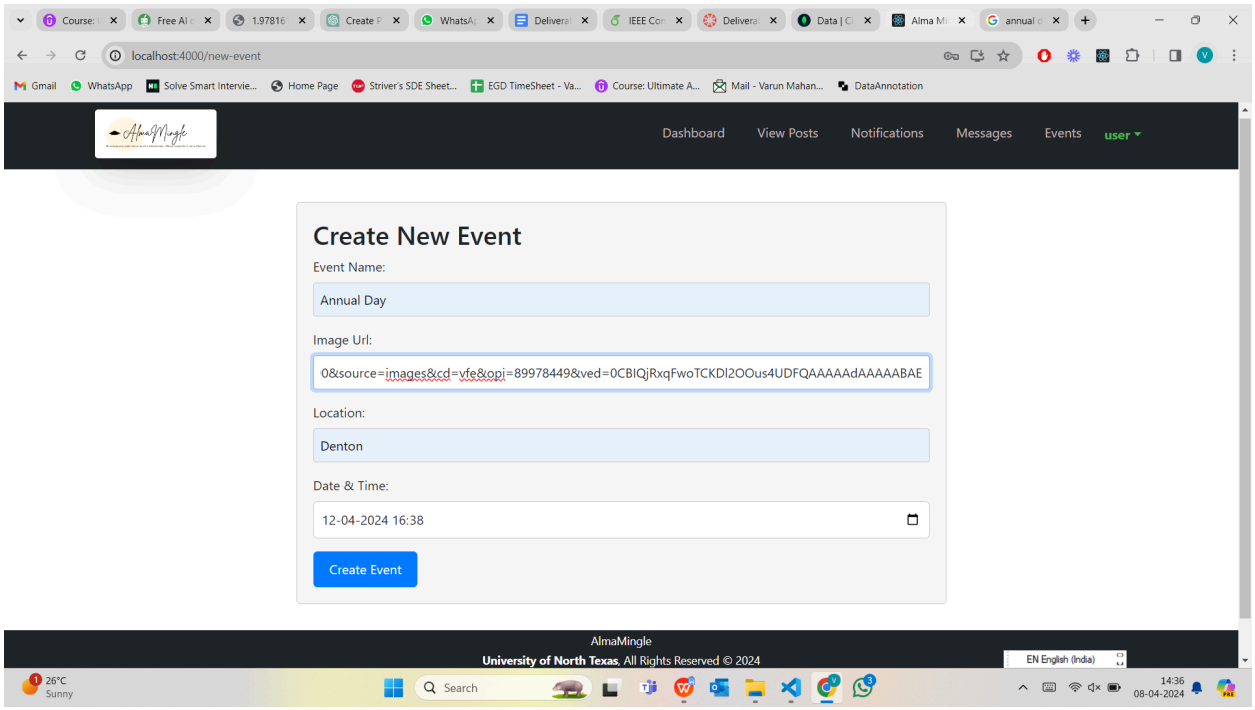


Fig D.14

Fig D.14 shows the create a new event page where the user can create a new event by giving information such as event name,image url, location and datetime and click on create event button then that event will be visible on the event page.

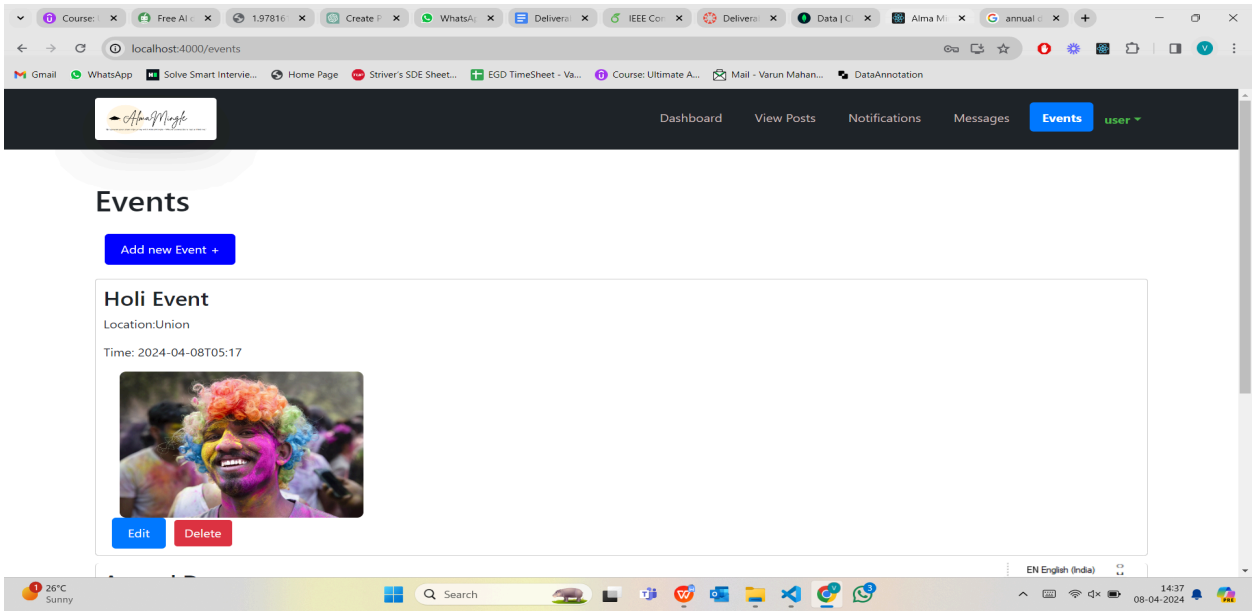


Fig D.15

This is the event page where all the events are shown which was created by users from the create new event page.

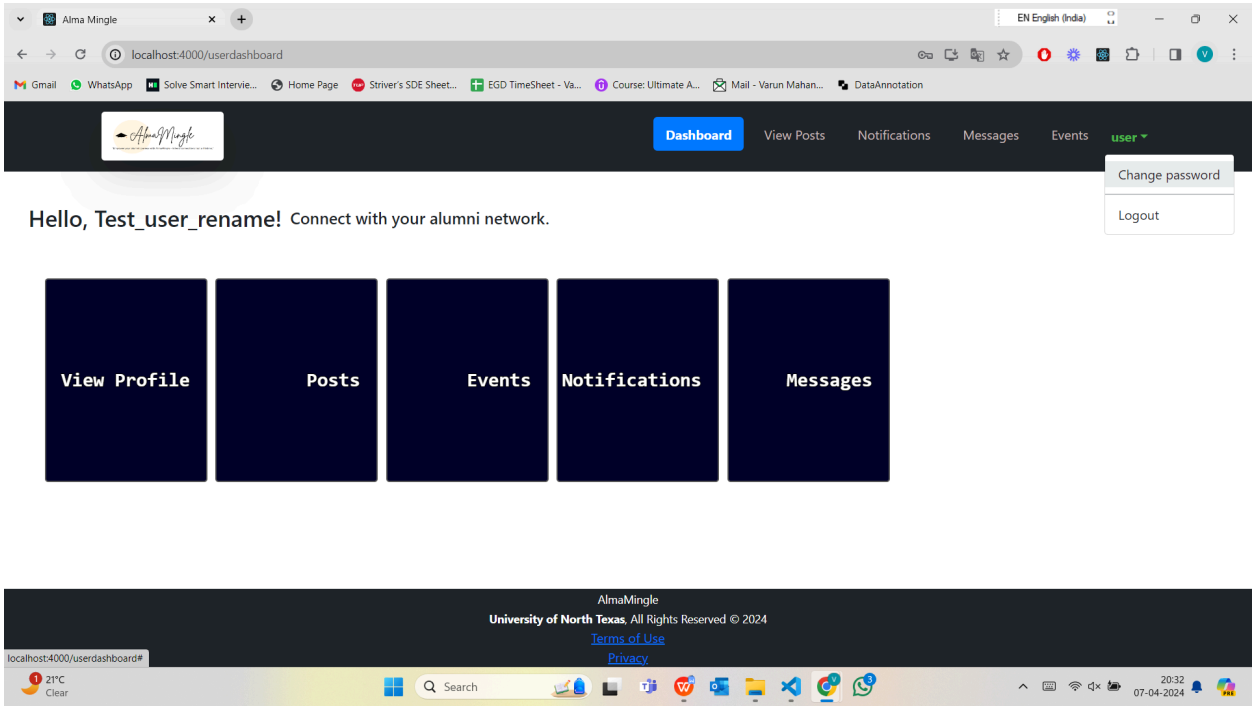


Fig D.16

If the user needs to change the password, the option of changing password is there in the dropdown menu of the user.

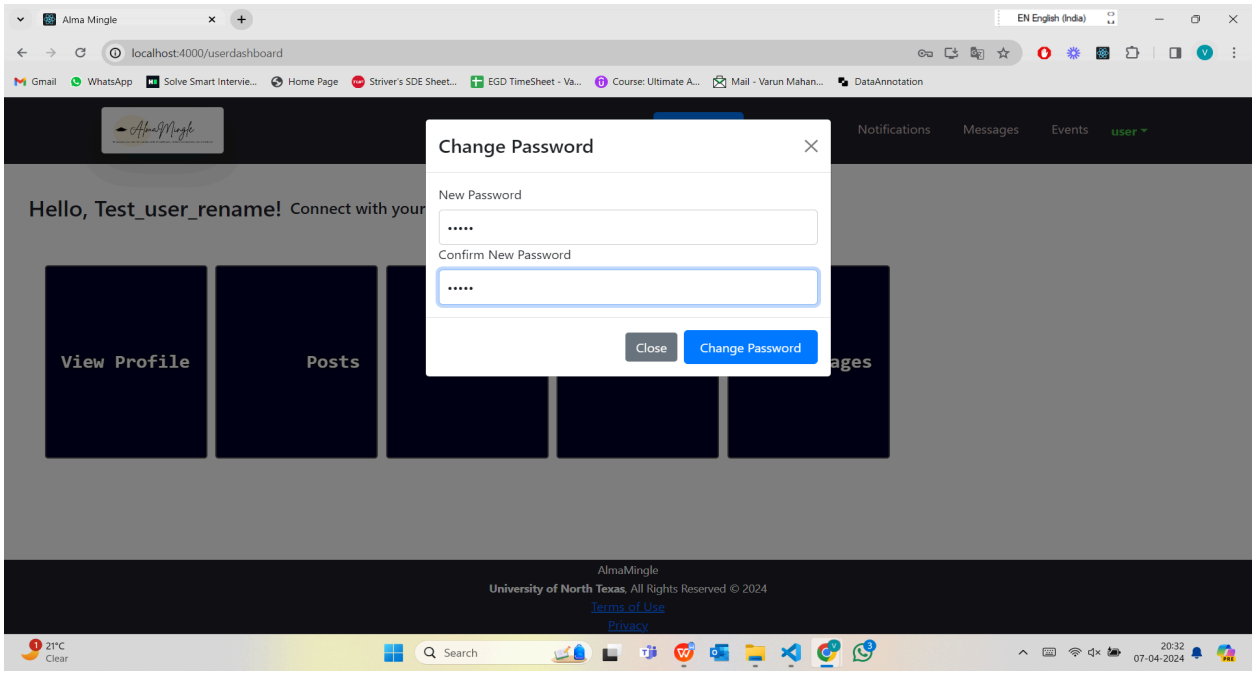


Fig D.17

The user needs to provide the new password and confirm the same on this screen and click on change password. The password will be changed after that.

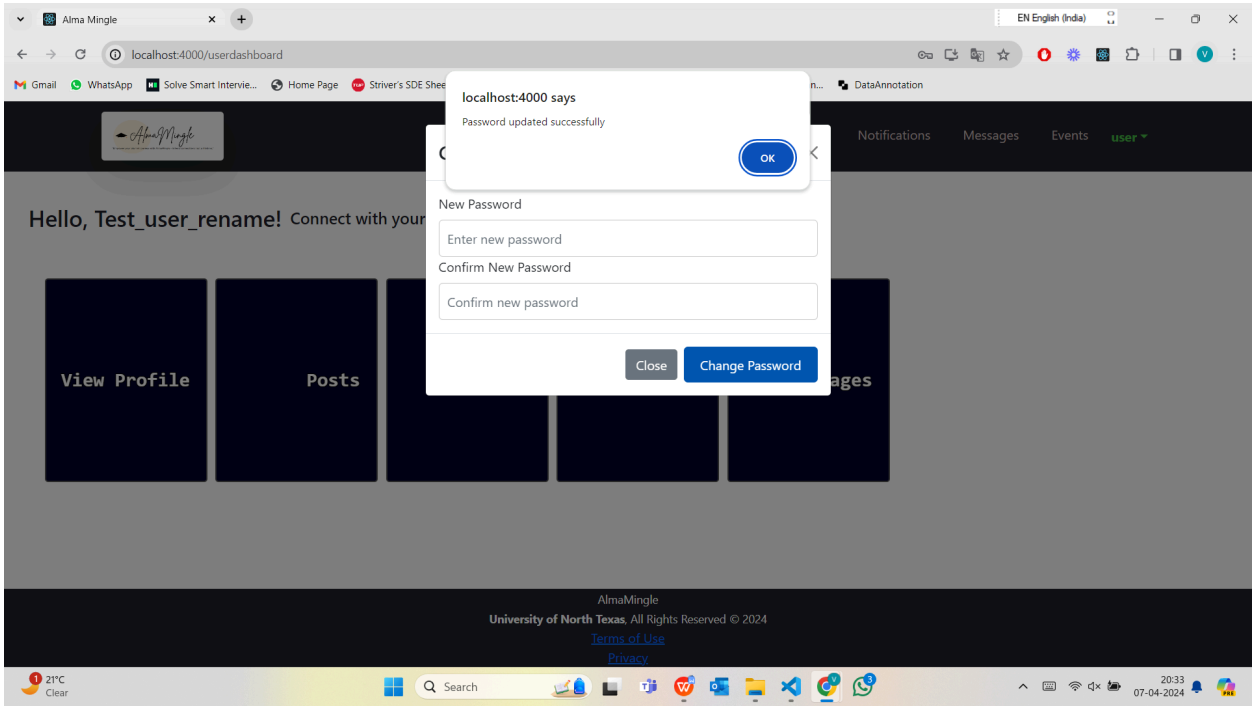


Fig D.18

The pop-up will be visible saying password changed successfully after clicking on change password.

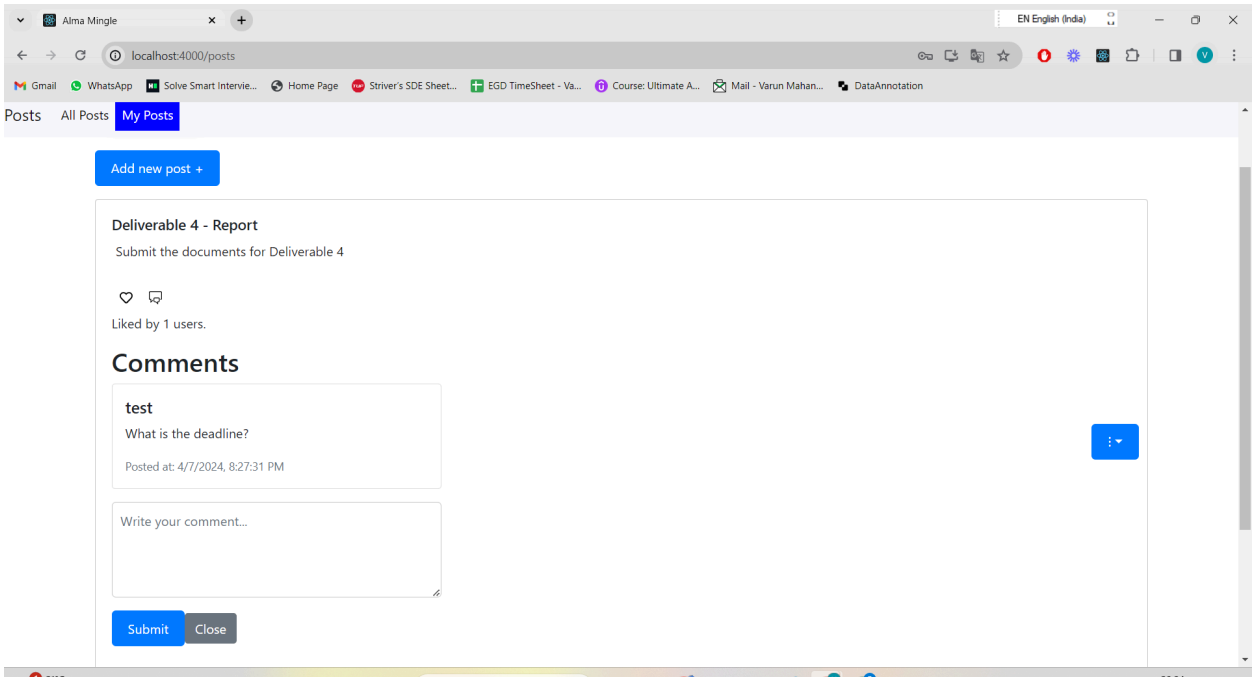


Fig D.19

This is the post section. We can like or see or give comments on that post we can like the post by clicking on the like icon and see the comments by clicking on the comments icon. All the comments can be seen there and if the user needs to comment then the new comment can be added by the text box and clicking on submit button.

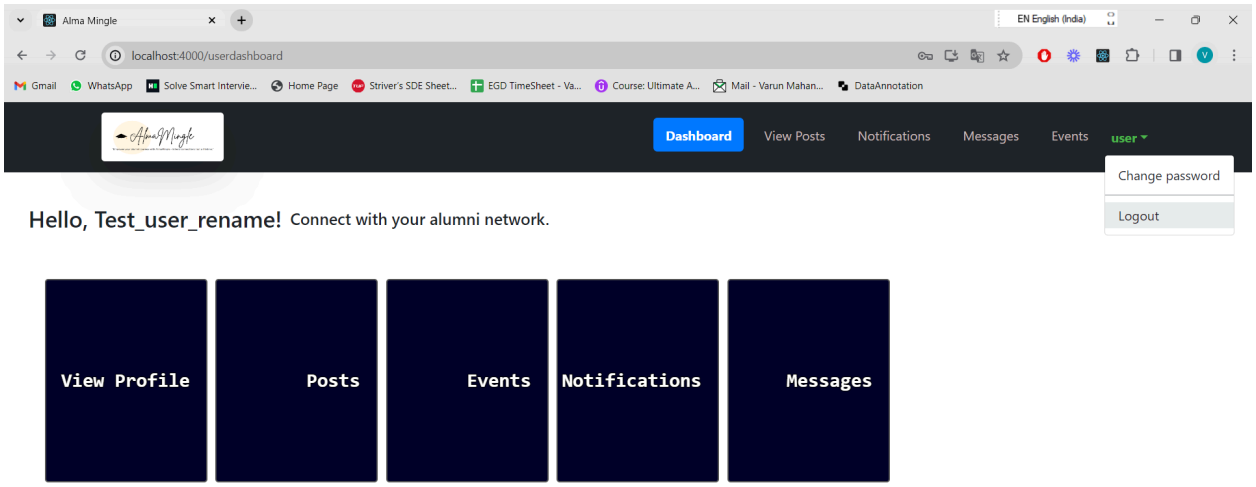


Fig D.20

If the user wants to logout. It can be done with the logout button present on the top right drop down.

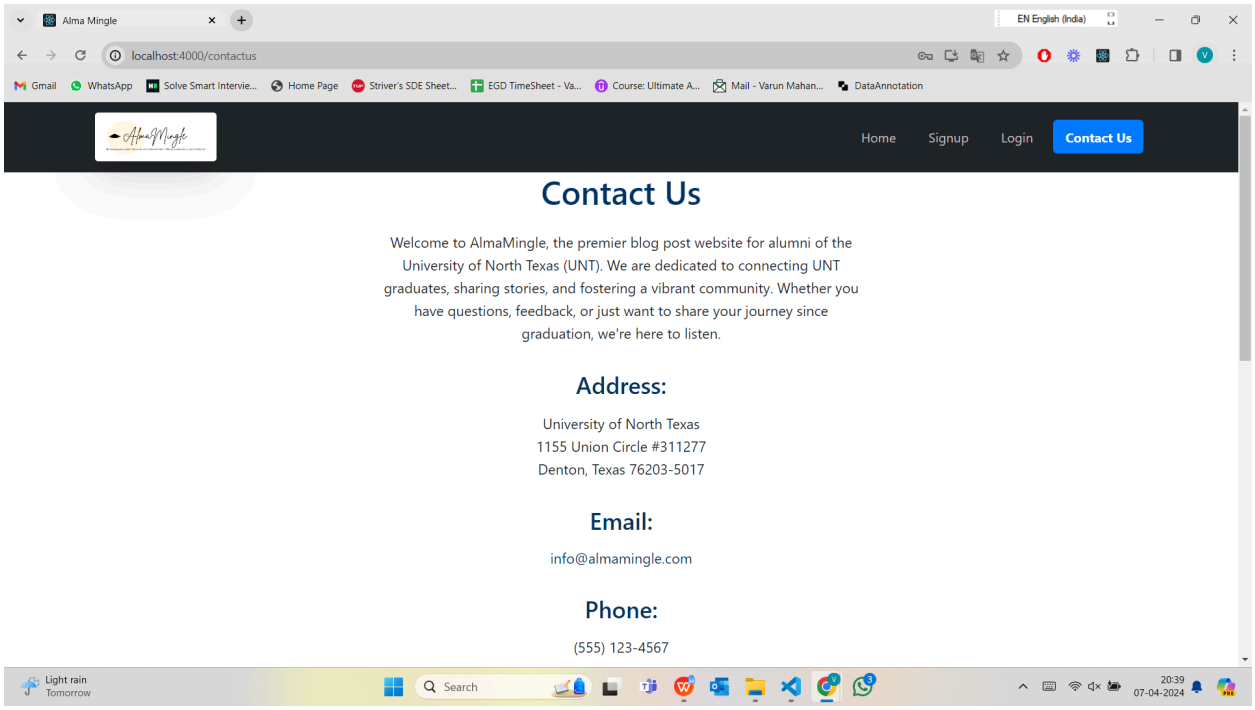


Fig D.21

Fig D.21 shows the Contact Us page, showing the details to contact the application admin. Address, Email phone.

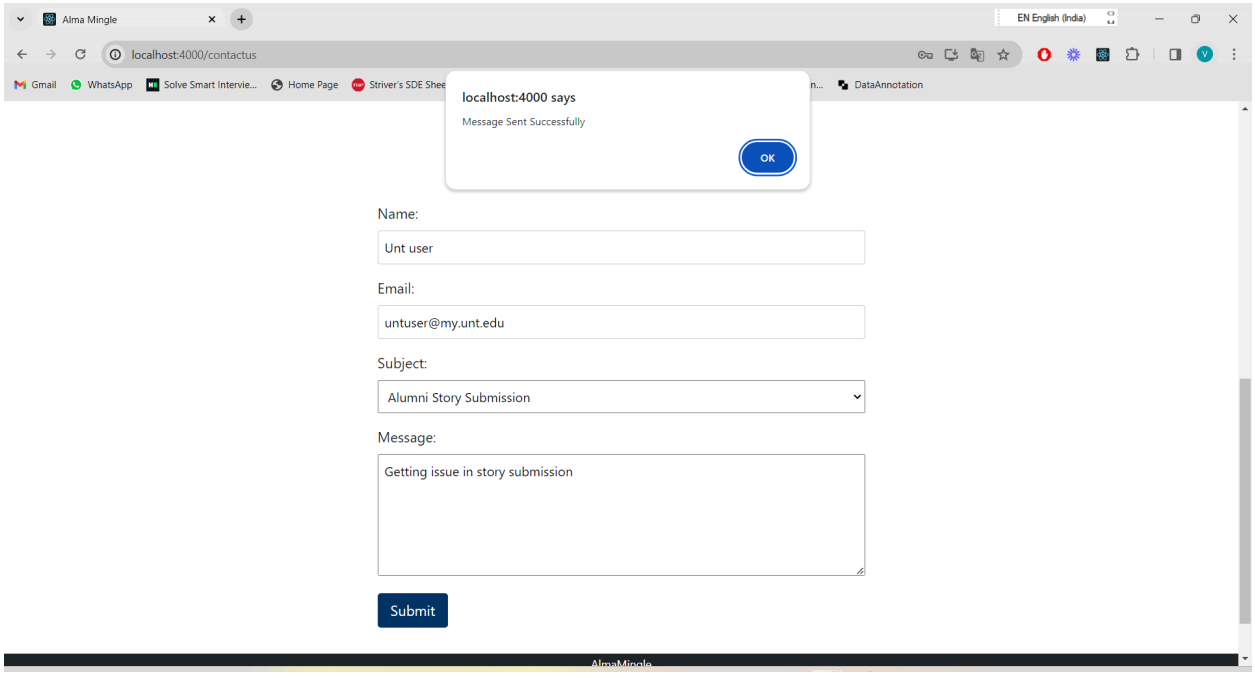


Fig D.22

If the user wants to send a message to the admin it can be done on the contact us page there is a form that will ask for name, email, subject, and message and

when the user clicks on the submit button the message will be sent to the admin.

## Admin Login and Dashboard Screens

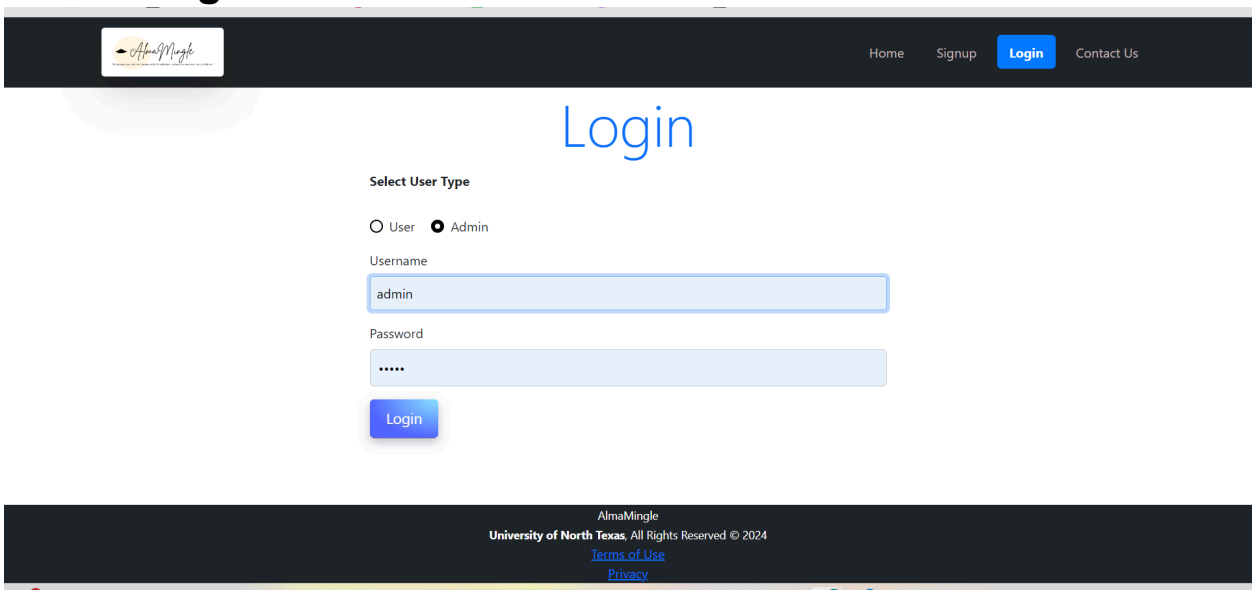


Fig D.23

If the admin wants to login into the portal then in select user types the admin option needs to be selected and enter the admin credentials.

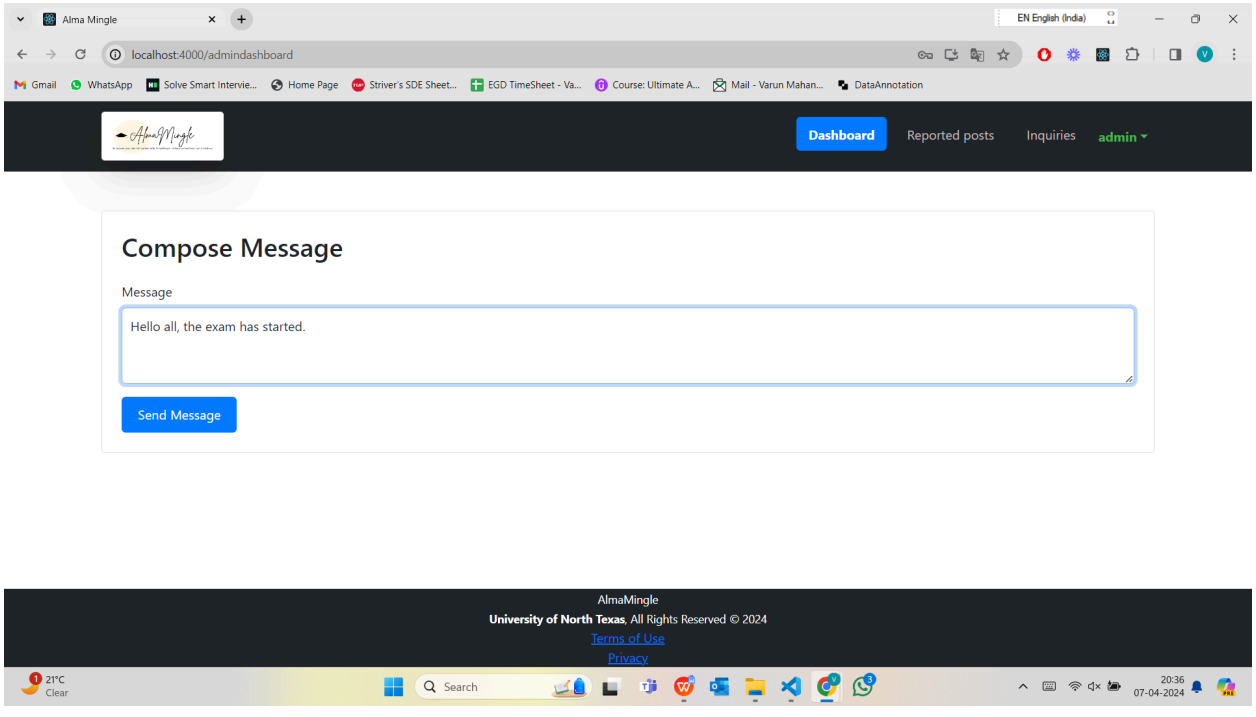


Fig D.24

In this screen if an admin needs to send a message to all users it can do it by

entering the message and clicking on send message button.

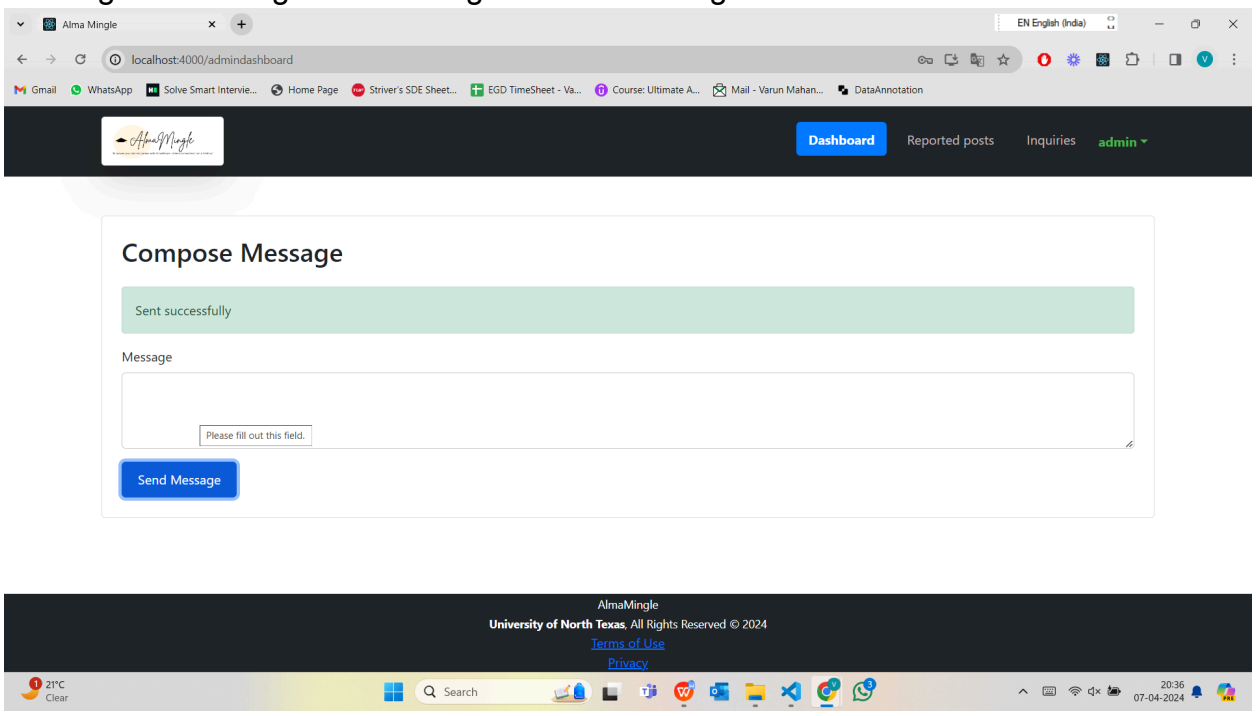


Fig D.25

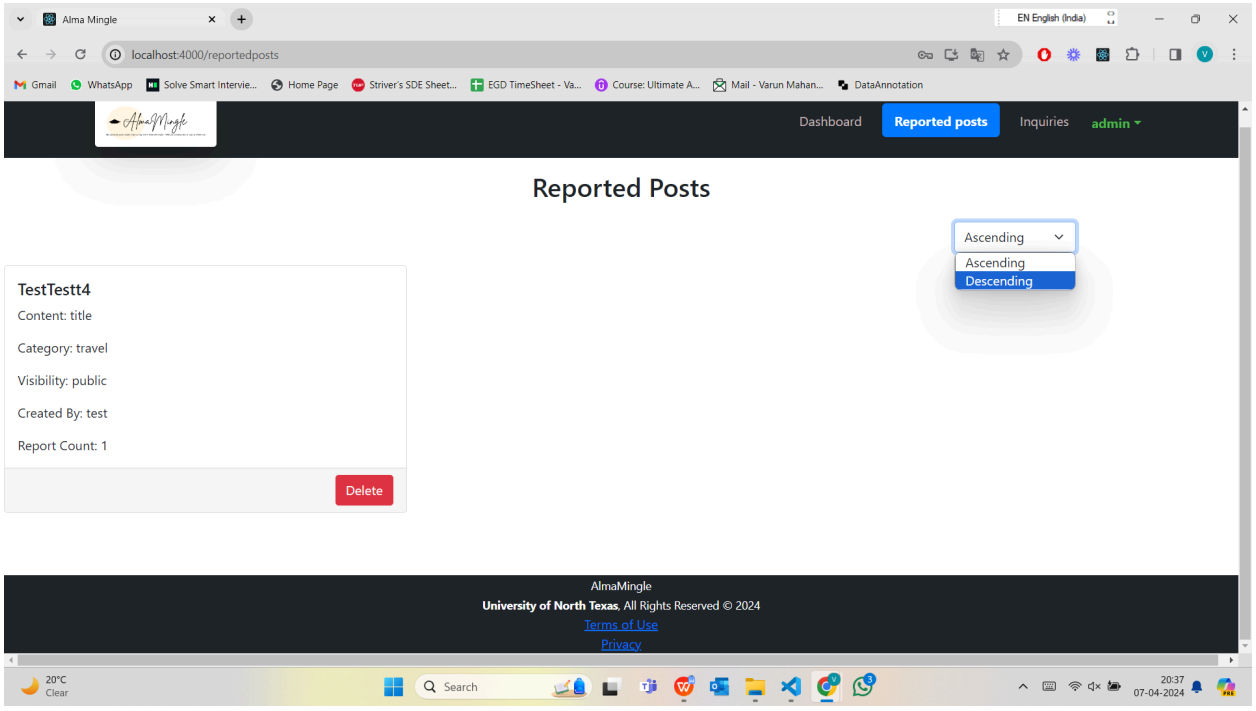


Fig D.26

Fig D.26 shows the screen where all the posts are shown which are reported by the users. The admin can sort the posts in ascending or descending order by clicking on the option provided on the top right corner.

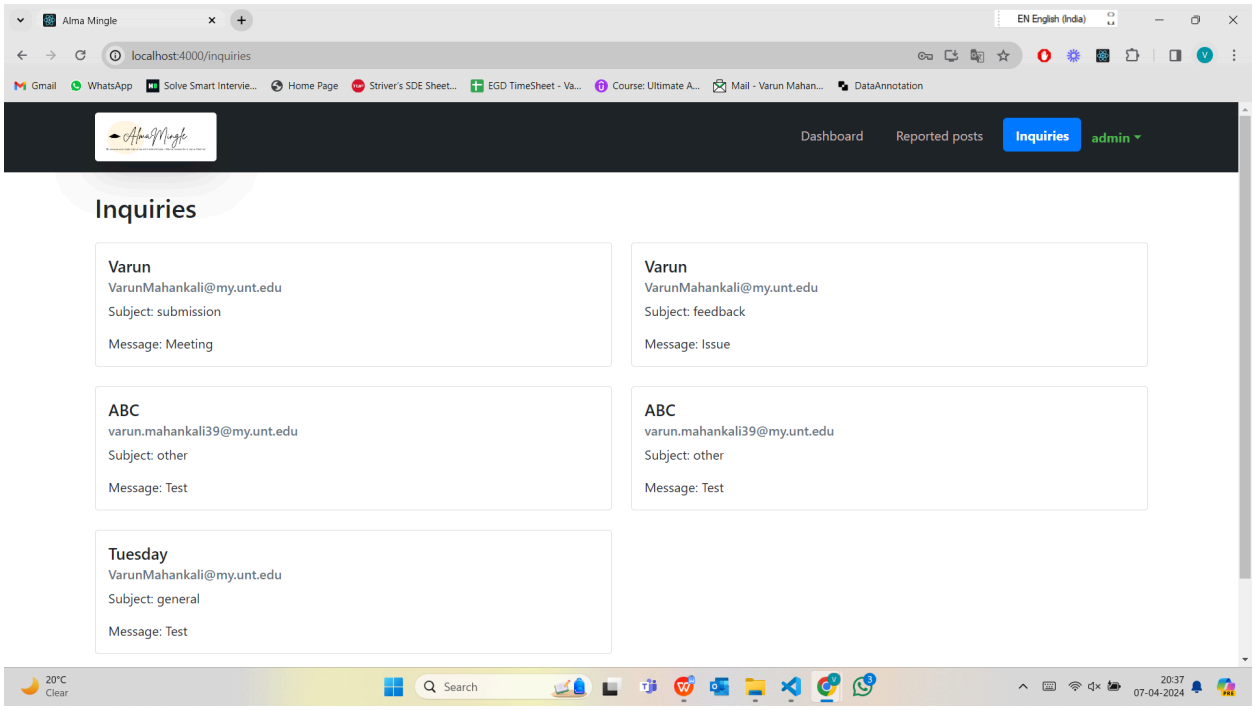


Fig D.27

Fig D.27 shows all the inquiries done by users on the contact us page.

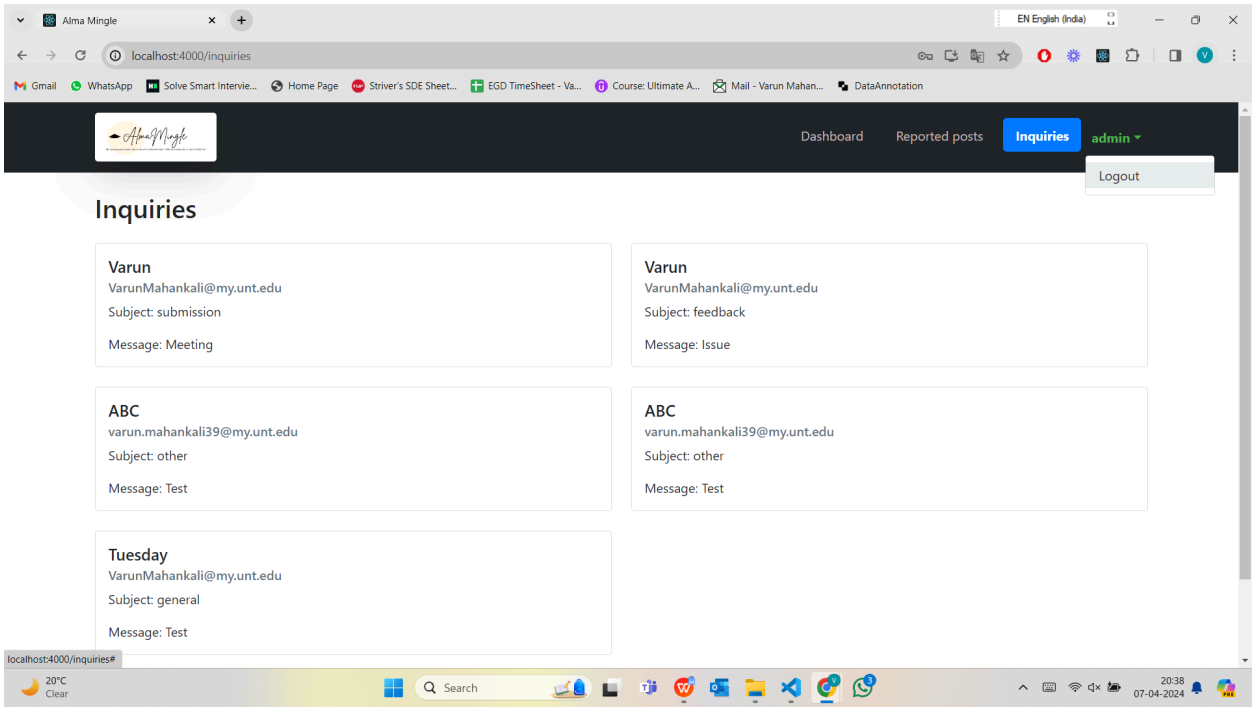


Fig D.28

If the admin wants to logout it can be done using the logout button available on the top right in the dropdown.

# INSTRUCTIONS TO COMPILE/ RUN THE PROGRAM and TEST CASES

## *Program:*

Initially after cloning the repository, the main code lies in the Back-End folder as it takes the build folder from the Front-End side. The build folder consists of all the files that are required to run or present in the Front-End folder. In order to run the code, follow the below steps.

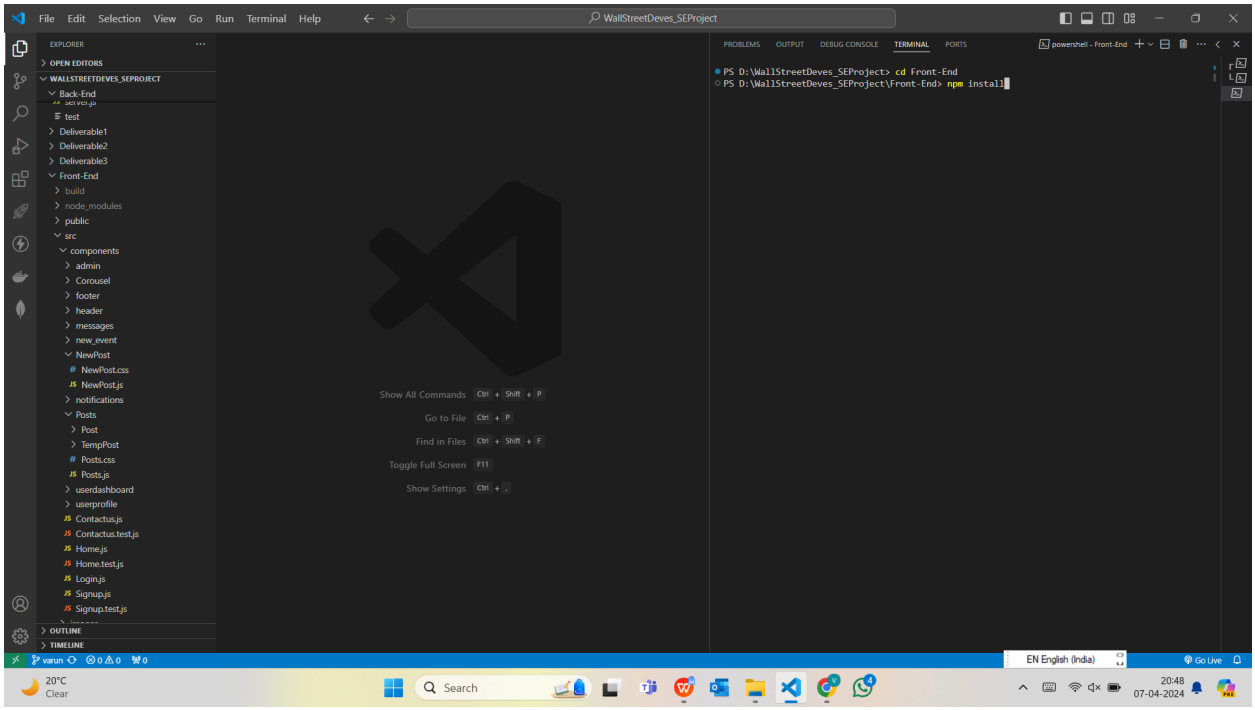


Fig E.1

In above Figure E.1, navigate to the Front-End folder and run the command “npm install” as shown above. This will install all the required packages

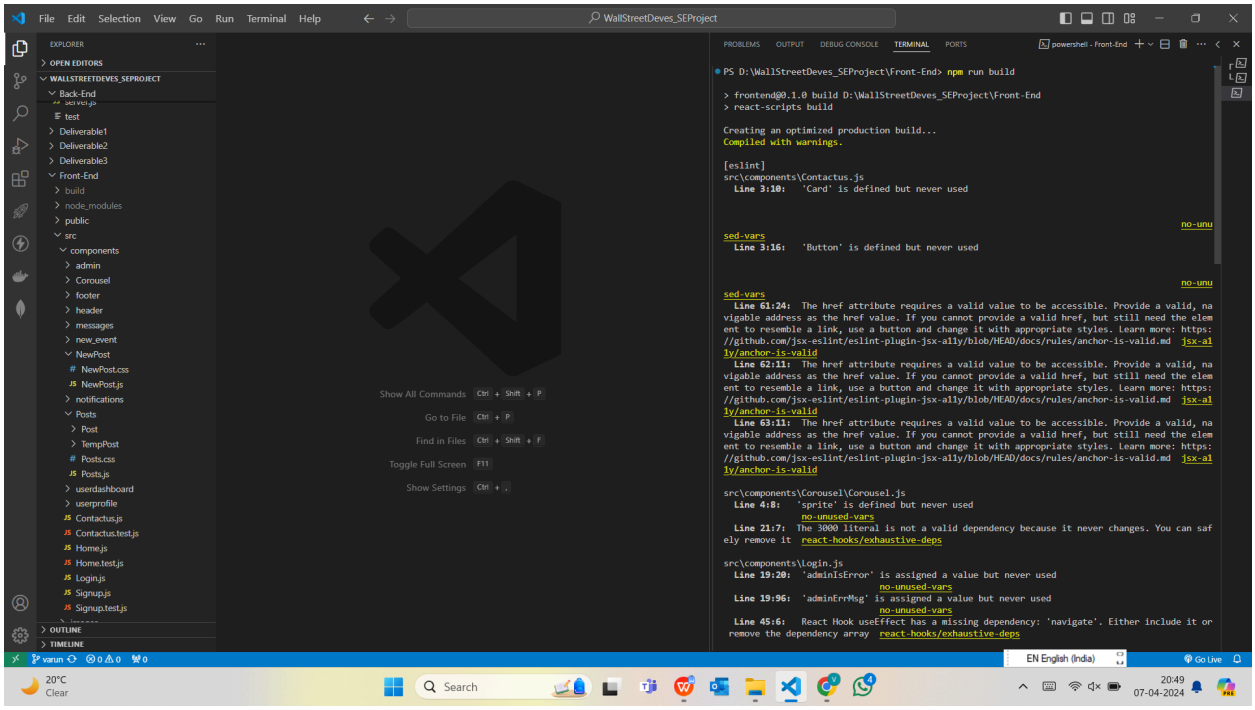


Fig E.2

Now as shown in Figure E.2, run the command “npm run build” to build all the files.

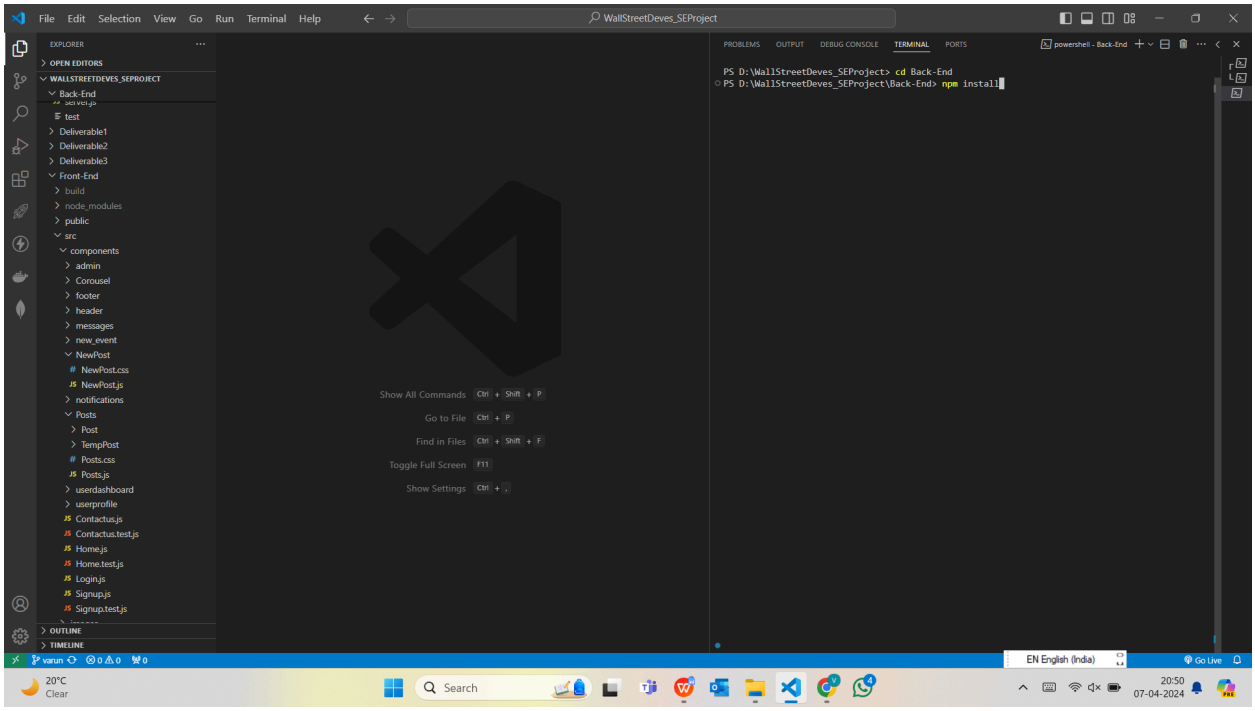


Fig E.3

Now Navigate to the Back-End folder and run the command “npm install” to run the packages in the backend side as shown in the above figure E.3.

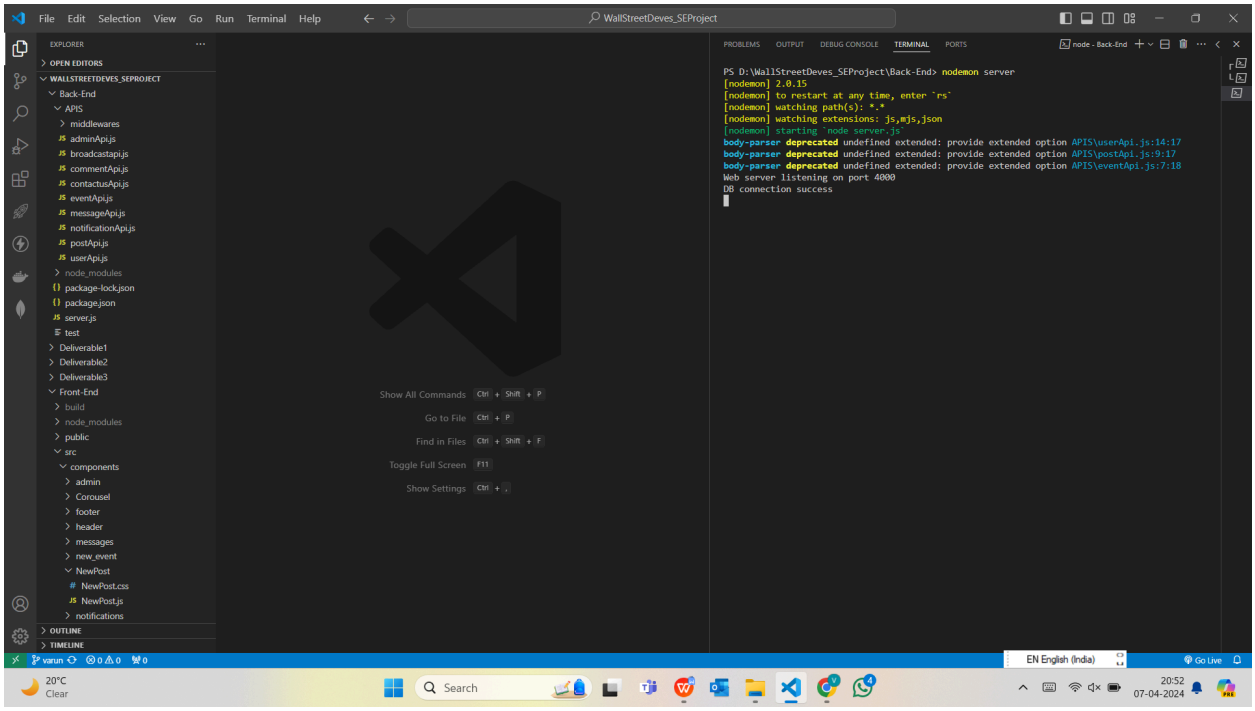


Fig E.4

Now as shown in Figure E.4 run the command “nodemon server” and the connection success message will be shown. Now the user can open the link “http://localhost:4000/” in the browser and can start creating the account and login and adding the posts the user requires.

## *Test Cases*

After connecting to another server and running the program, We can open a new terminal for running our test cases. To run them, we need to have the initial set up done which includes giving the following commands for the required installation

$ npm install
$ npm install msw --save-dev
$ npm install --save-dev redux-mock-store
$ npm install --save-dev @babel/plugin-proposal-private-property-in-object
$ npm install --save-dev @testing-library/react @testing-library/user-event
$ npm install react react-dom axios react-bootstrap @testing-library/react
$ npm install --save-dev @testing-library/react @testing-library/jest-dom
$ npm i --save-dev @testing-library/react react-test-renderer

$ npm install --save-dev @testing-library/react @testing-library/jest-dom @testing-library/user-event jest axios-mock-adapter

$ npm start

At last, to run the test cases we need to give the following command.

$ npm test -- --detectOpenHandles

It will automatically run all the test cases that are present in our project folder. We just need to make sure that for all the names of testing related files should contain “*.js” as their extension.

**Output:**

After executing all the commands that are mentioned above, it will generate this kind of output.

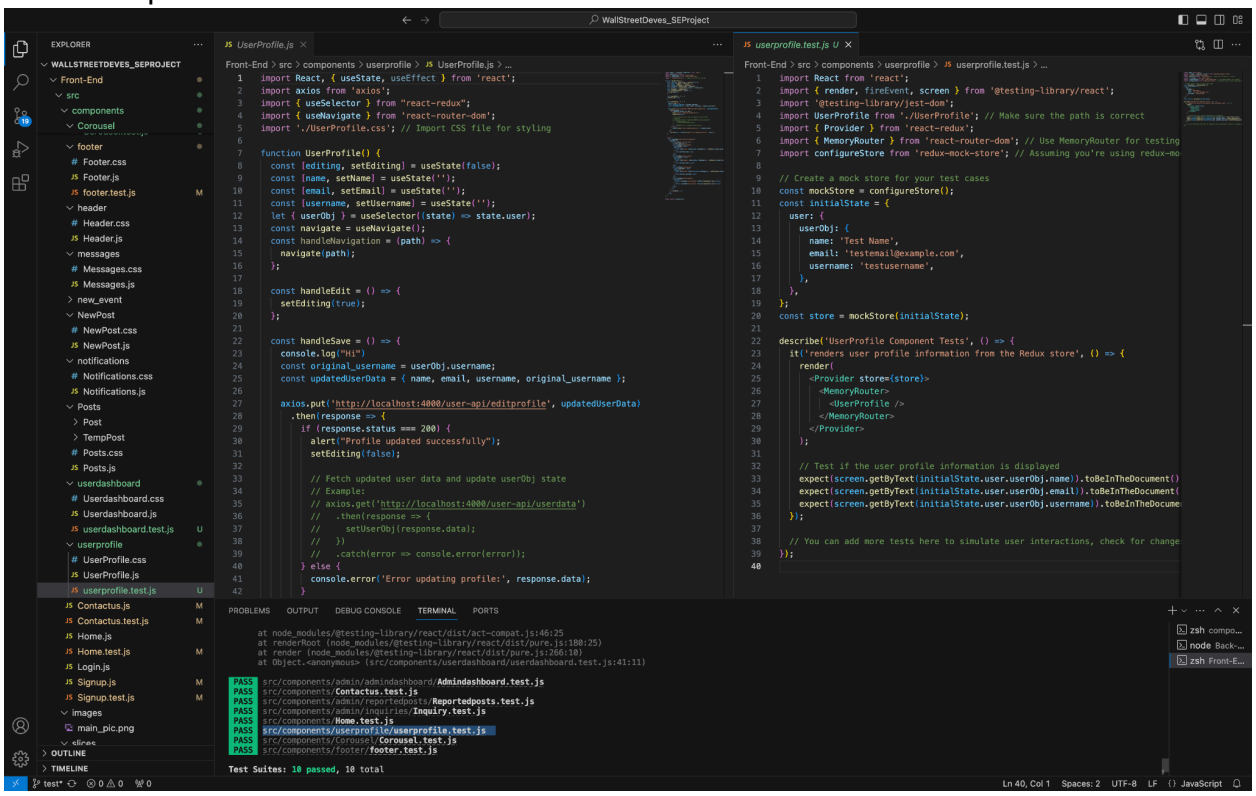


Fig E.5

## CODE INSPECTION FEEDBACK

During a recent code inspection session, we delved into discussions aimed at enhancing our testing framework. The feedback encouraged us to extend our testing scope to include backend components, leading to the integration of Jest, a popular testing framework, for rigorous assessment. This expansion aims to ensure robustness and stability across all layers of our application, instilling confidence in its performance. Additionally, feedback from "Tiny coders" suggested evaluating website accessibility through the public network space, though we encountered a challenge with connecting to MongoDB, which we're currently addressing. Ensuring data integrity and security is paramount, so we're prioritizing the implementation of rigorous data checks on the signup page, including password length and email validation.

Our discussions have been centered around implementing messaging and notification features to enrich the user experience, facilitating seamless communication and engagement within the platform. As we navigate the intricacies of implementing these functionalities, usability remains a top priority, ensuring clarity in messages and timeliness in notifications. Through iterative development and user feedback, we're dedicated to refining these features to meet evolving user needs and expectations, ultimately contributing to a more dynamic platform. Reflecting on the feedback received, we're confident in our ability to effectively enhance the project, grateful for the constructive criticism that has shaped our trajectory. Moving forward, we're committed to implementing necessary changes and enhancements to ensure the application's success. Overall, the code inspection session provided a valuable learning experience, guiding us towards continued improvement and development.

## REFLECTION

Reflecting on our progress, we've accomplished most of the functional components outlined for Phase 2, which is a significant achievement. Additionally, we successfully integrated optional functionalities like "post comments" and "post like", enhancing the project's overall value. Moving into Phase 3, we plan to maintain the approach that served us well in Phase 2: sequential categorization of tasks, regular team meetings for updates, and weekly work reviews. We acknowledge there are some pending enhancements from Phase 2 that need to be addressed in Phase 3. These include implementing forgotten password functionality on the home page, event deletion, event updating, UI enhancements, and finalizing the website code before freezing it.

To improve in phase 3, we aim to initiate User Acceptance Testing (UAT) in Phase 3 to identify and address any issues promptly. Deployment-related tasks

will be a focus, ensuring a smooth transition to the live website. Critical to our success will be handling deployment issues effectively and ensuring the responsiveness of the website. We believe that all of these functionalities we have met for the current phase and will improve going into the next phase will create a more robust and user-friendly platform, fulfilling the evolving needs and expectations of our users.

## MEMBER CONTRIBUTION TABLE

| S.No | Member Name | Contribution | Overall Contribution (%) |
|---|---|---|---|
| 1 | Kamalini Ponnuru | I was responsible for coordinating the team's efforts and assigning tasks accordingly. During our meetings, we discussed the project requirements and divided responsibilities among team members. My focus was on developing test cases for the front-end functionalities, ensuring comprehensive coverage of all features. Additionally, I dedicated time to crafting documentation for the test cases, outlining their purpose, input, output, and expected outcomes. Finally, I compiled all the contributions into the final document deliverable, presenting a cohesive overview of our testing approaches and outcomes. | 12.5% |
| 2 | Andre Sharp | More unit tests for edge cases within the login page. Additional utility test cases around deployment and security were developed for quick diagnosis of errors. | 12.5% |
| 3 | Madi McCauley | Helped coordinate meetings and kept track of participants and our contributions, peer review, accomplishments and improvements needed according to this development phase, helped with uml and ui coordination and coding, report formating | 12.5% |
| 4 | Shashank Verma | Worked on Frontend implementation of carousel on the home page, fixed navigation bar issues and implemented likes and comment frontend implementation of the posts. Worked on fixing | 12.5% |

| | | | |
|---|---|---|---|
| | | issues for all posts and my posts frontend. Fixed minor UI issues. | |
| 5 | Usama Bin Faheem | Worked on development of Event functionality , updated the phase plan 3 , Reviewed the requirements and worked with the team to narrow down essential components of the application . | 12.5% |
| 6 | Suhaibuddin Ahmed | Worked to add updates to the phase plan 3 and added descriptions to the updates, collaborated in the UI development of Events page. Tested Login and registration. Worked towards implementation. | 12.5% |
| 7 | Varun Mahankali | Divided the development work among the team members, coordinated with team members and completed all the functionalities. Implemented the Admin screens, added the likes and comment functionality to all the posts, implemented the Notifications functionality. Managed the database, helped teammates in resolving the errors, Implemented the Messages functionality and managed all the Api's in the backend and completed the D and E parts of the report. | 12.5% |
| 8 | Rahman Mehmood | Created UML Diagrams for Deliverable 4.<br>Helped in Delegating Responsibilities for the Deliverable 4.<br>Assisted in the UI of Events page. | 12.5% |