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 5 – Project Phase 3
CSCE 5430 (Spring 2024)

A. REQUIREMENTS

In Phase 1 , we started by understanding the technology stack. We faced many hurdles and were not able to complete all the requirements for this phase. Therefore many of the requirements of this phase were accommodated and allotted bandwidth in our phase 2 development. The functional components completed in phase 1 included:

- **Landing Page**
    - Registration of user
    - Login Feature
- **Dashboard for the alumni user**
- **Posting Functionality**
- **User Profile form**
- **Design Components**
    - Navigation Menu on the dashboard
    - Headers and footers of all pages
    - Carousel on the landing page

In the following phases, we changed our approach and followed a more categorised and sequential development plan which is reflected in our phases 2 and 3 which helped us in making development more efficient.

In Phase 2 we covered most of the planned features and functionalities that were discussed and reviewed by everyone after Phase 1 planning and execution. Most of these features make up the backbone of the project and were crucial to be completed before working on any newer future features. The functional components completed in phase 2 included

- **Error Handling and Database Management:**
    - Error Fixing in Database Updates: Address and rectify issues encountered while updating existing data in the database, ensuring stability and reliability during data modifications.
- **Security and User Profile Enhancements:**
    - Update Password: Implement secure methods for users to update their passwords.
    - Update Profile: Allow users to modify their personal profile details.
- **Content Management Features:**
    - Post Update/Delete: Enable users to edit or remove their posts.
    - Post Commenting: Facilitate users to comment on posts.

    - Post Report**:**
        - Every post has an options button
        - Users have option to report the post by pressing the “report” button
    -
    - Filter Posts: Introduce functionality to filter posts based on various criteria, enhancing user experience.
    - Admin User Story: Implement all functionalities required by admin users, including user management and post-moderation.
    - Post Broadcast: Develop features for broadcasting posts to the feed for everyone.
    - All Posts Delete: Allow admins to delete all posts when necessary.
- **User Interface Enhancements:**
    - UI Enhancement of All Pages: Improve the user interface across all pages for better user experience.
    - Additional UI Components: Add more UI components as required for enhanced interactivity.
    - Burger Menu on Dashboard: Implement a burger menu for better navigation and space management on the dashboard.
    - Information Tab Page for the Landing Page: Add an informational tab to the landing page to provide essential details at a glance.
    - Gallery on Landing Page: Incorporate a gallery section on the landing page to display images or media.
    - Footer on All Pages: Ensure that a footer is added to all pages for consistency and to provide useful links or information.
    - Events on Navigation Bar: Add an events section in the navigation bar to highlight upcoming events or important dates.

- **Admin User story with all functional requirements:**
    - The admin on AlmaMingle,will log in with the same credentials as other users and see a similar user interface (UI).
    - Admins act like platform moderators they have additional privileges like:
        - Post Broadcast
        - Reported Posts
        - Inquiries
        - Delete posts

Detailed descriptions of all the components developed in phase 1 and phase 2 of our development can be found in deliverable 3 and deliverable 4 respectively

- **Deliverable 3 hand in:** Final Hand in Deliverable 3
- **Deliverable 4 hand in:** **Main Deliverable 4**

We will be discussing our implementations in phase 3 in this deliverable below.

In phase 3 we consulted the teaching assistant for their suggestions on our application and implemented their suggestions, additionally we worked on UI enhancements and the deployment of our application.

- **Event Update :**
    - The user can also update his/her event on the events page
    - The edit button is present in blue color
    - When it is pressed it opens a dialog box where changes can be made to the posted event
    - Once changes are made the “save changes” button when clicked persists the new event information into the backend
- **Event Delete :**
    - The user can also Delete his/her event on the events page
    - The Delete button is present in red color that removes only the specific posts chosen by the user.

- **Update/ forgot password :**
    - The user is now able to reset his password through the Forget Password feature present in the login section.
    - The user upon clicking the reset password button will be prompted to enter his username and the answer to a security question.
    - Through validation of these entries, the user will be sent an email which he can use to enter a new password.
    - The user is also able to change his existing password after logging in through the “change password” button present above the logout button
- **UI enhancements :**
    - Several enhancements were made to the standard UI present in the previous phases. Notable ones include:
        - Restructuring the landing page and allocating new images for the slideshow
        - Cleaner UI of the navigation bar and refined layout of the pages

We have also implemented the suggestion as mentioned below -

- **Event ticket booking**
    - The user can also book tickets for events on the events page
    - The book tickets button is present in blue color
    - When it is pressed it opens an event booking dialog box where the user can select the number of tickets and enter card details to make the payment
- **Donate :**
    - Users can now donate to AlmaMingle through the donate page.

- The page asks for the donation amount which the user can enter.
- The page also asks for the card details (debit/credit), including the card number, date of expiry and CVV.

The entire code and the application’s usability was checked and the code was freezed, finally deployed on the web.

B. UML DIAGRAMS

I) CLASS DIAGRAM

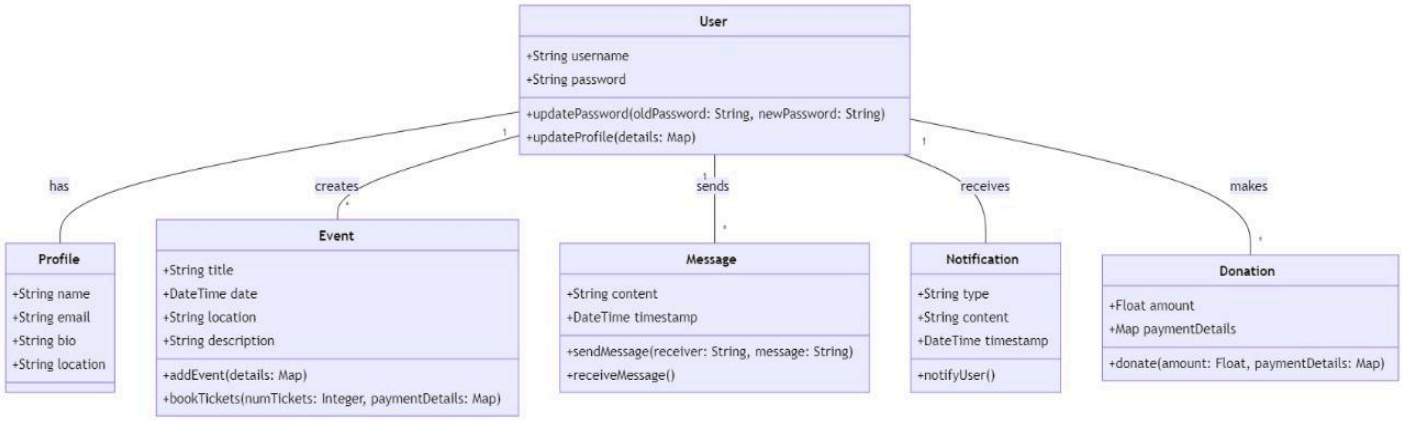


Figure B.1

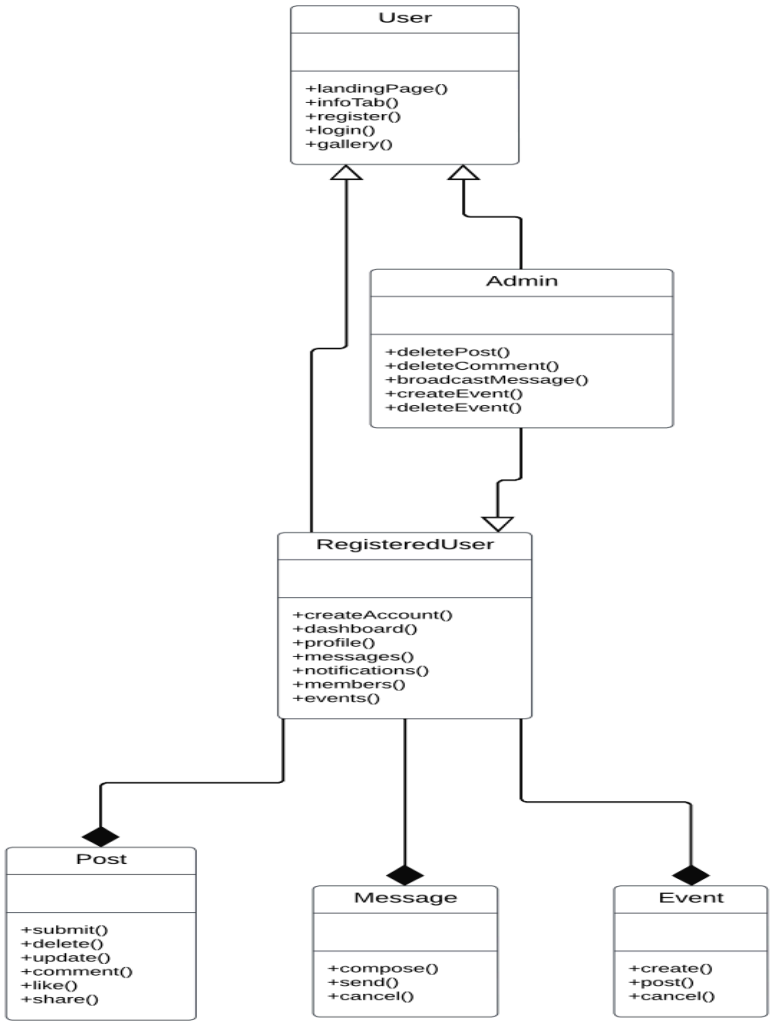


Figure B.2 Requirements Phase Class Diagram

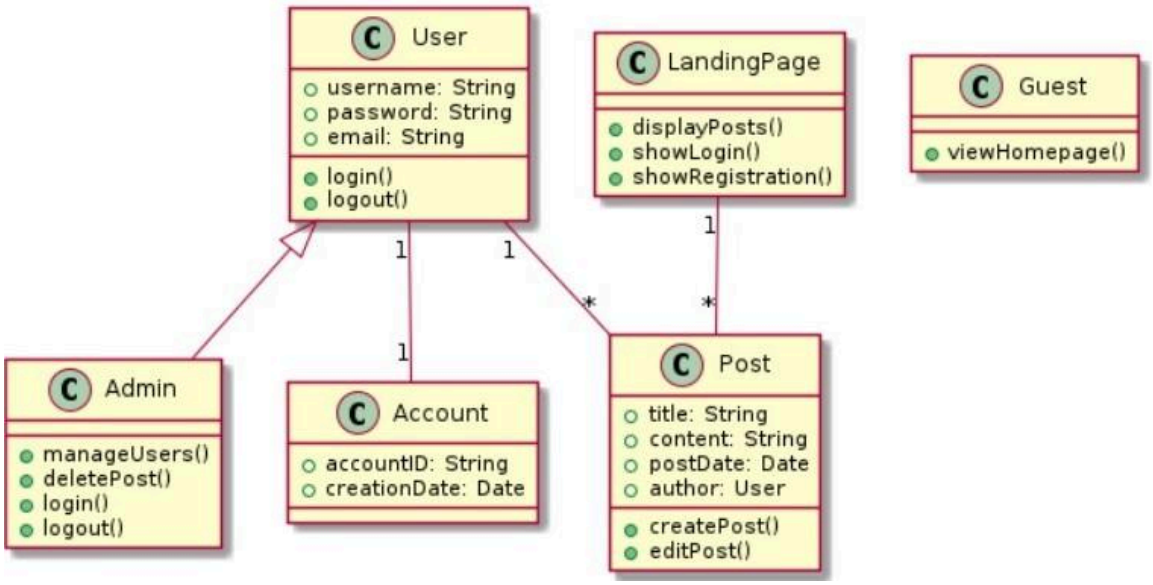


Fig B.3 Phase 1 Class Diagram

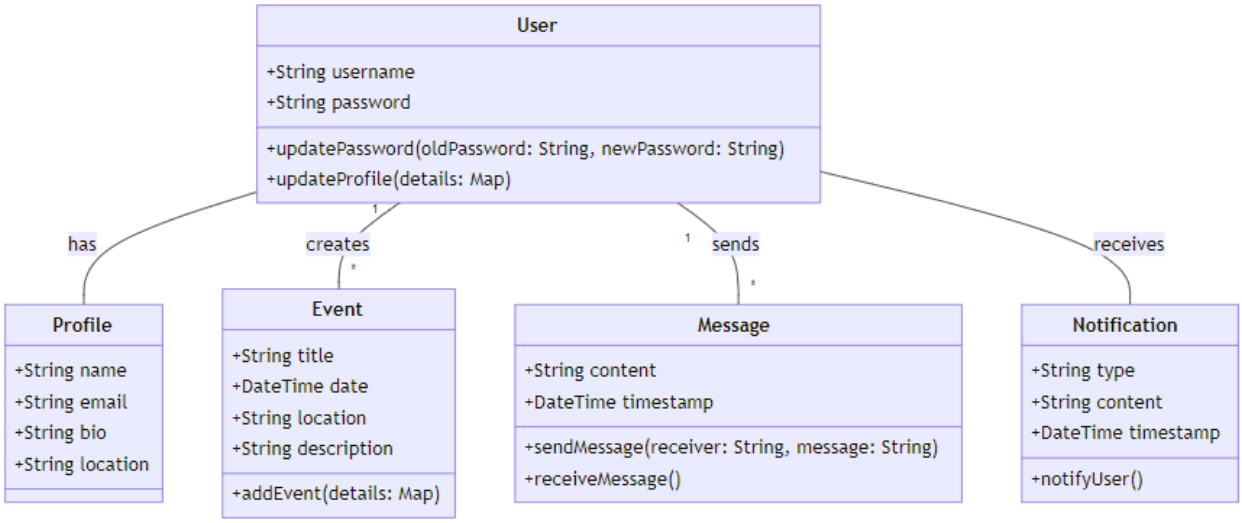


Fig B.4 Phase 2 Class Diagram

II)SEQUENCE DIAGRAM

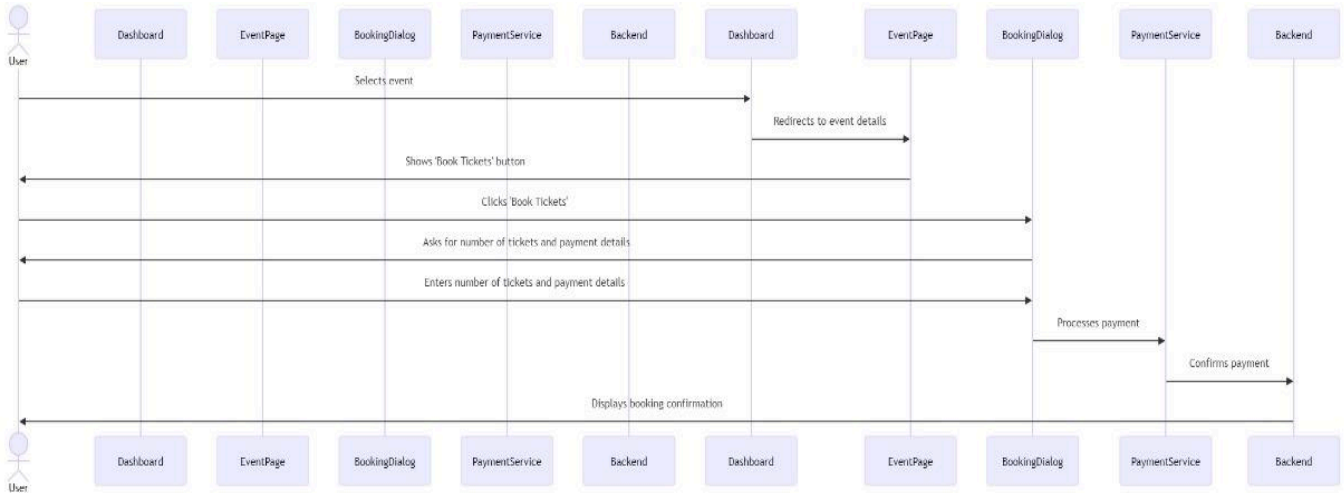


Figure B.5 Booking Tickets Sequence

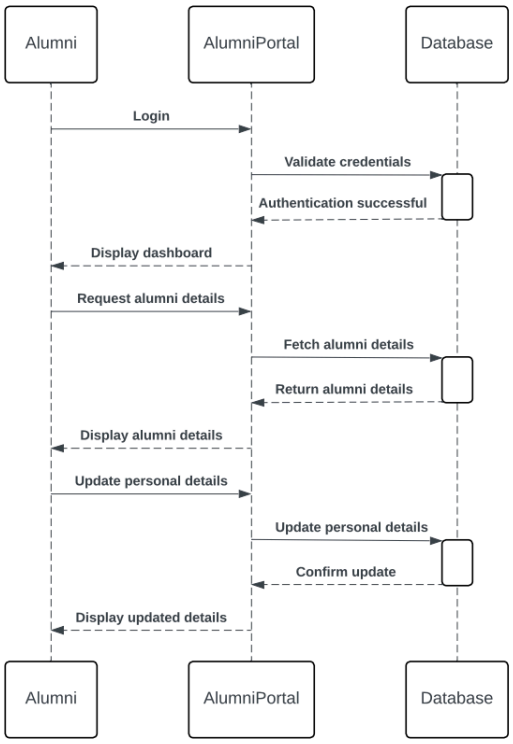


Fig B.6: Sequence diagram explaining few operations for AlmaMingle

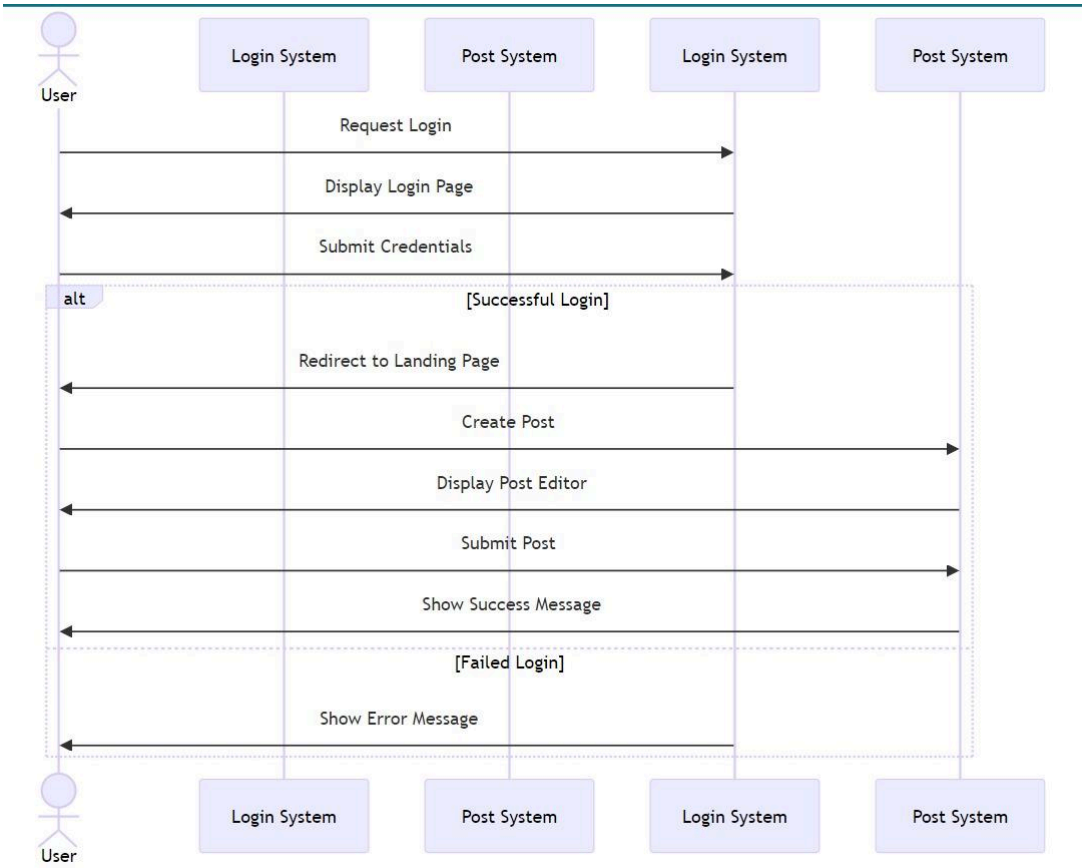


Fig B.7 Post Sequence

Update Password

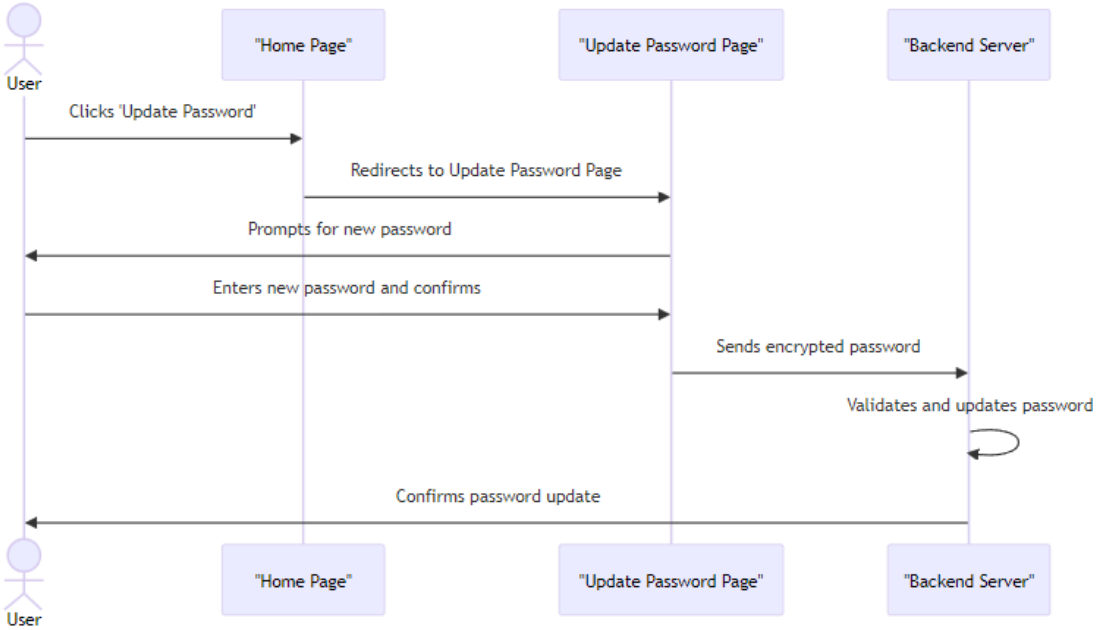


Fig B.8

Update Profile

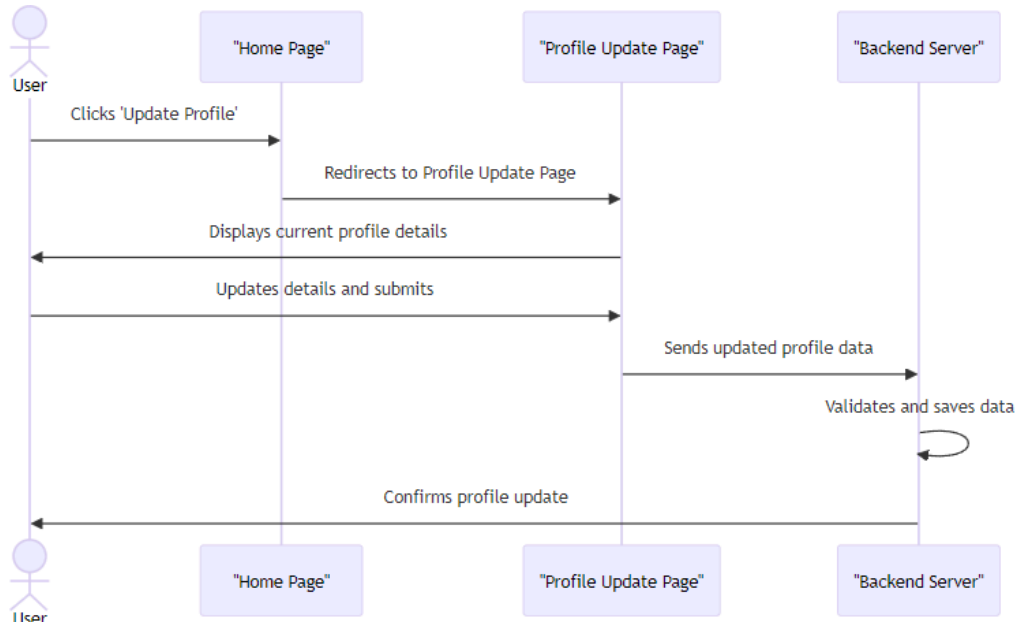


Fig B.9

Create Event

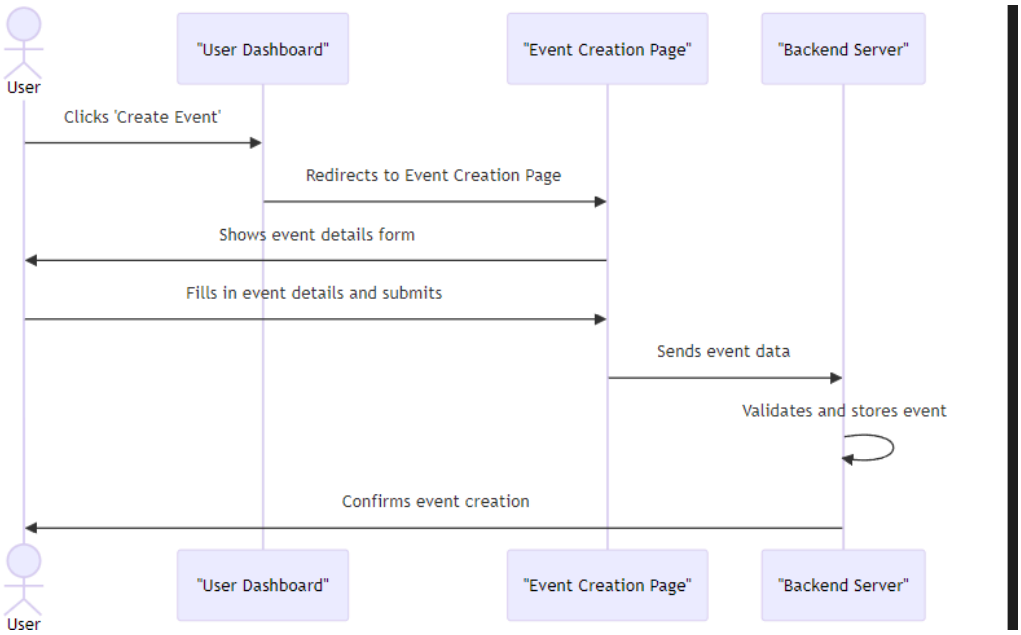


Fig B.10

Messages

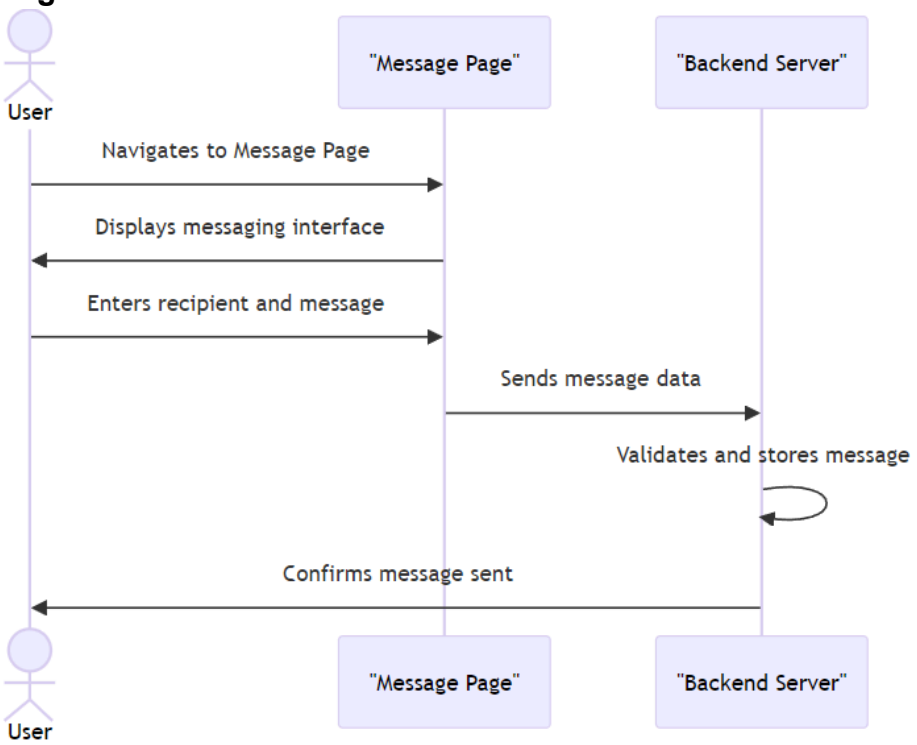


Fig B.11

Notifications

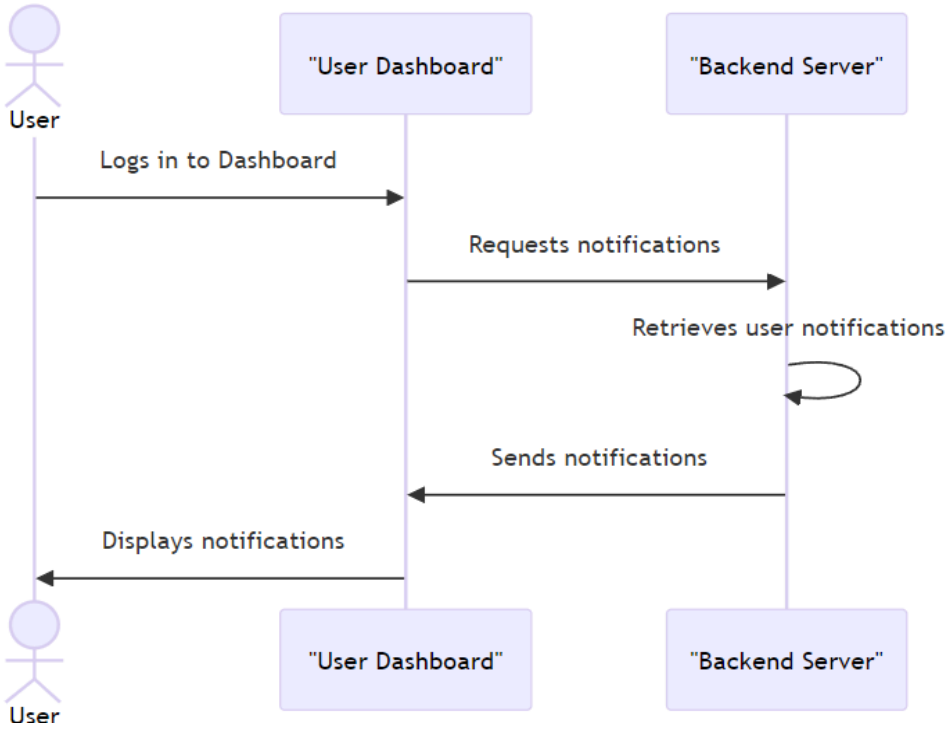


Fig B.12

III) USE CASE DIAGRAMS WITH SUCCESS AND FAILURE CASES

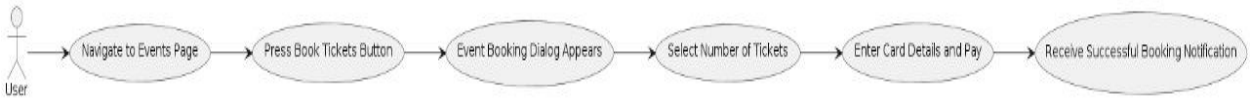


Figure B.13

Successful Ticket Booking

Figure B.14: Failure Ticket Booking

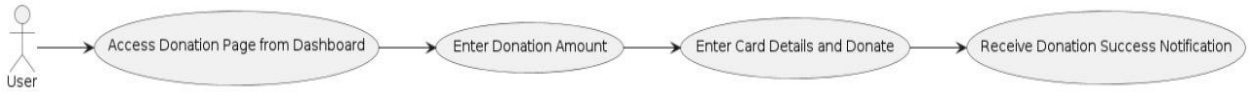


Figure B.15

Successful Donation

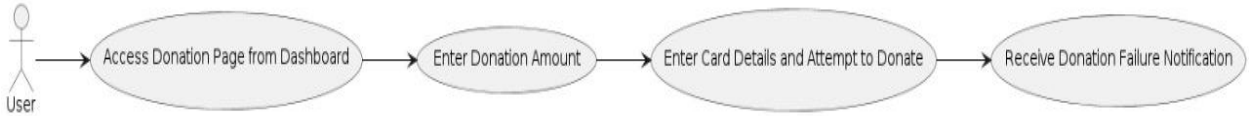


Figure B.16

Failed Donation

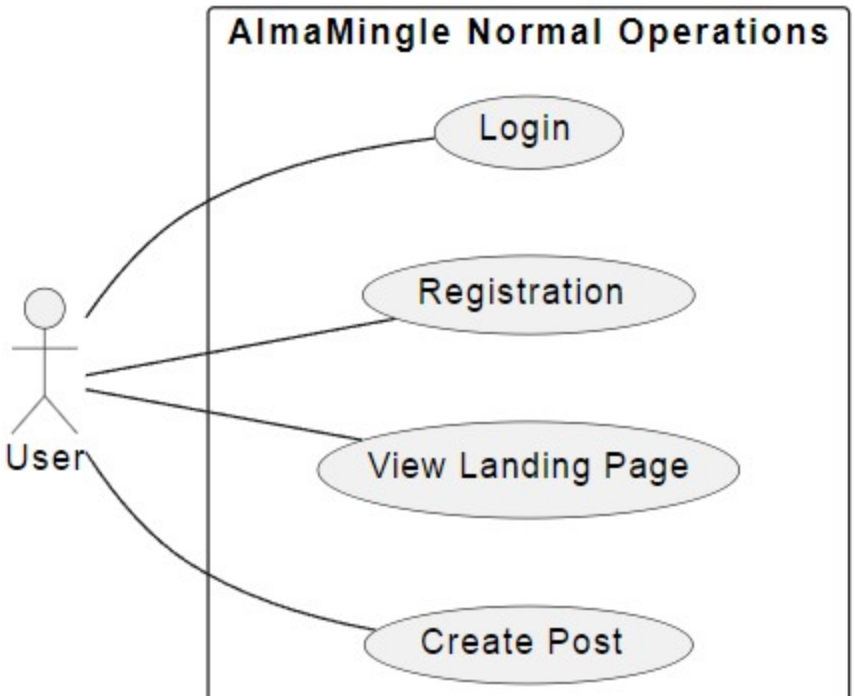


Fig B.17 Login Success Scenario

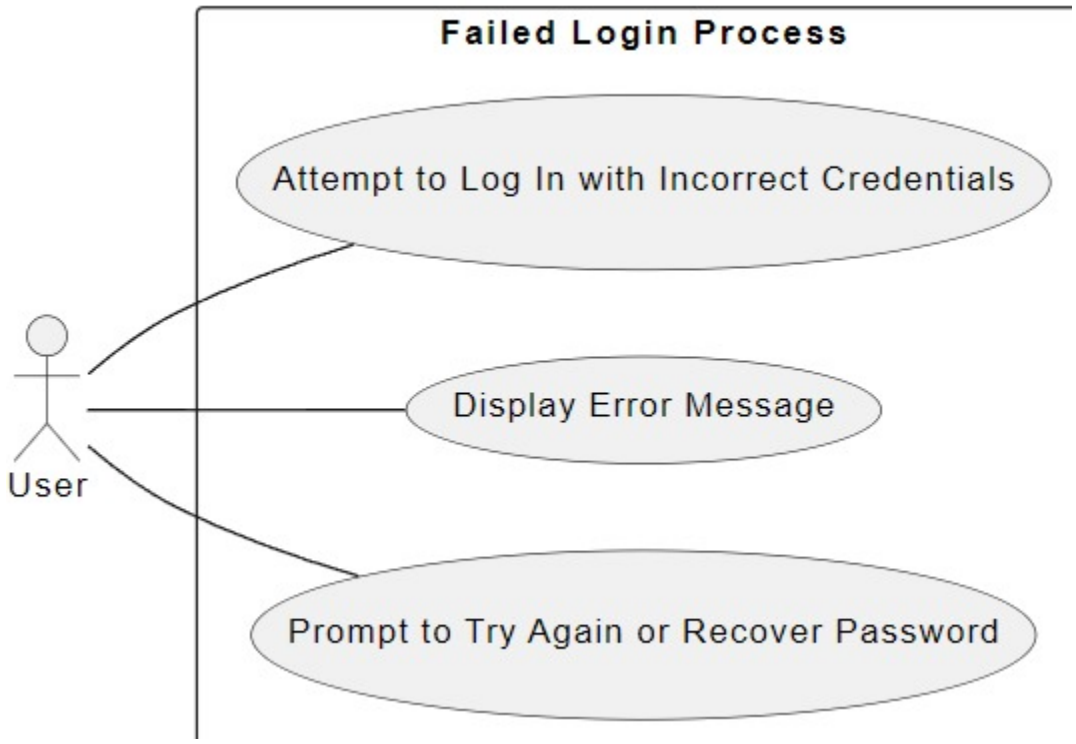


Fig B.18 Login Failure Scenario

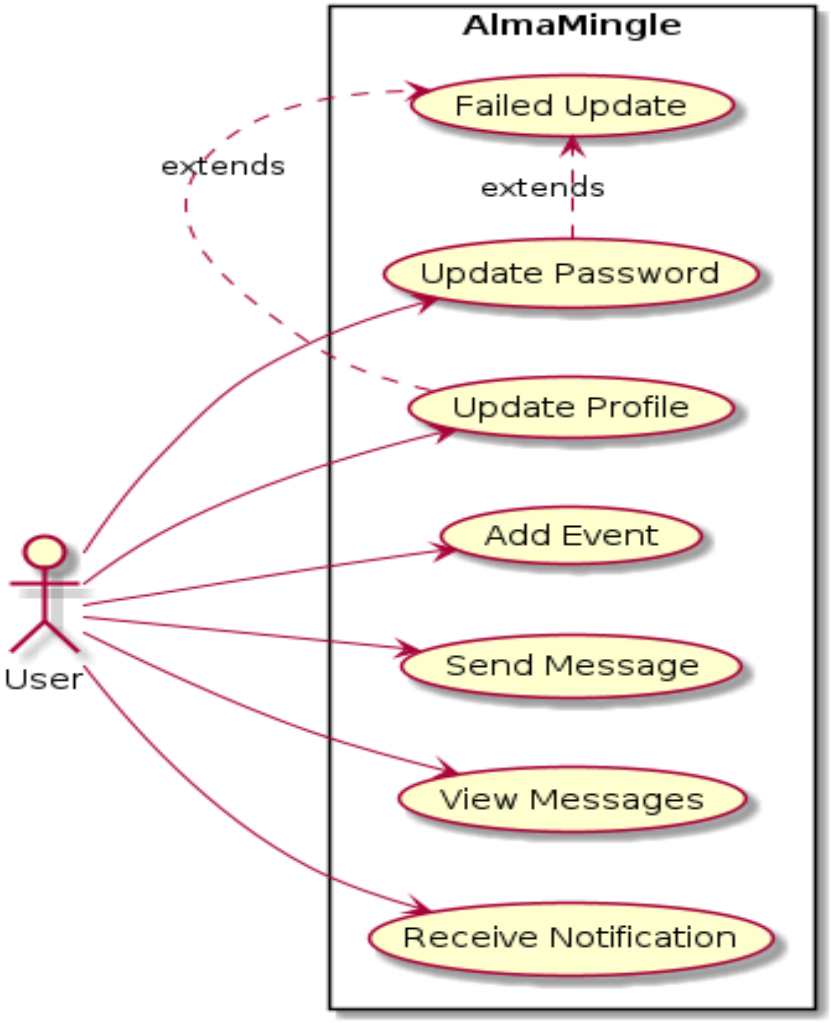


Fig B.19 Update Password Success

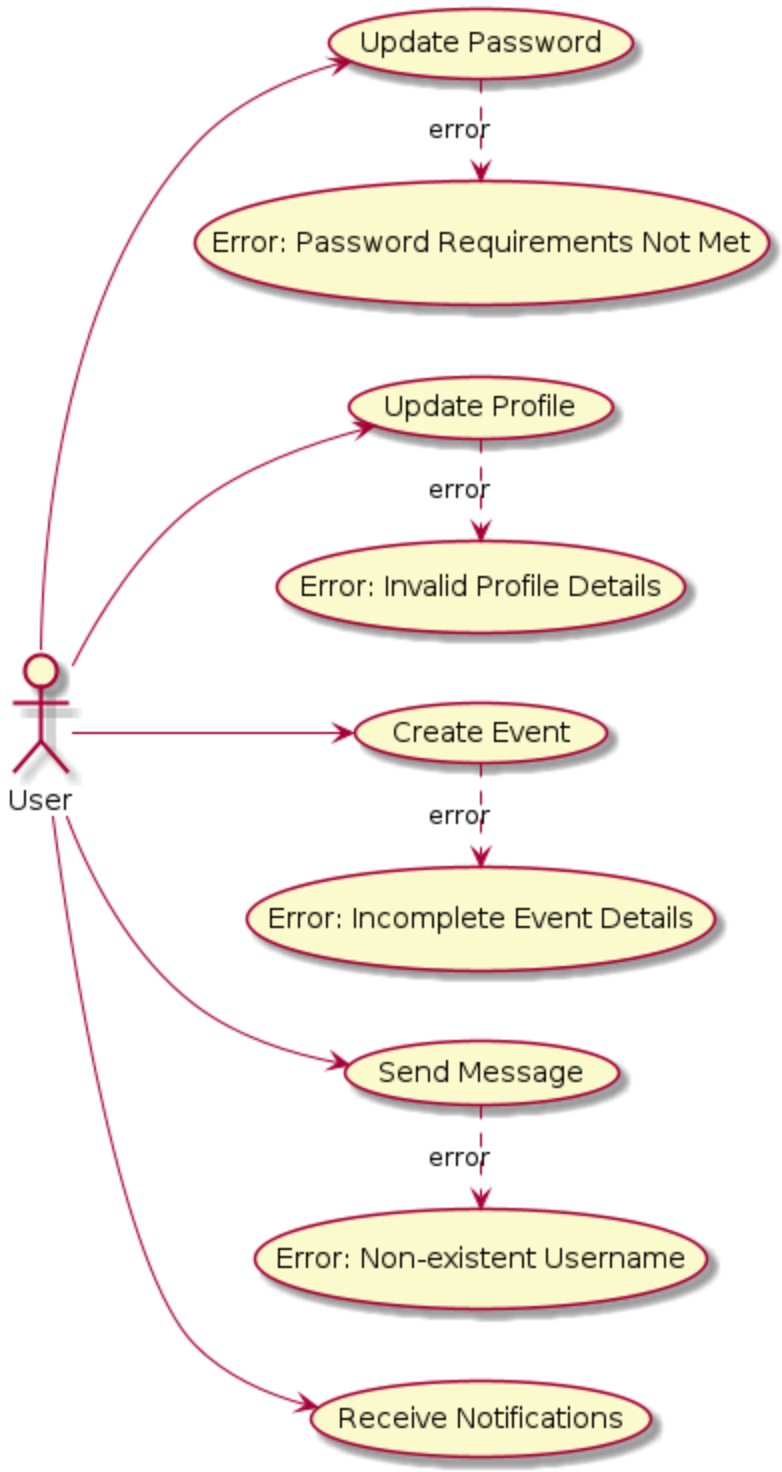


Fig B.20 Update Password Failure

C. TEST CASES

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

We've created various test case scenarios corresponding to different development stages, and we've executed both unit and functional testing. In summary, our testing encompasses all phases of development.

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.

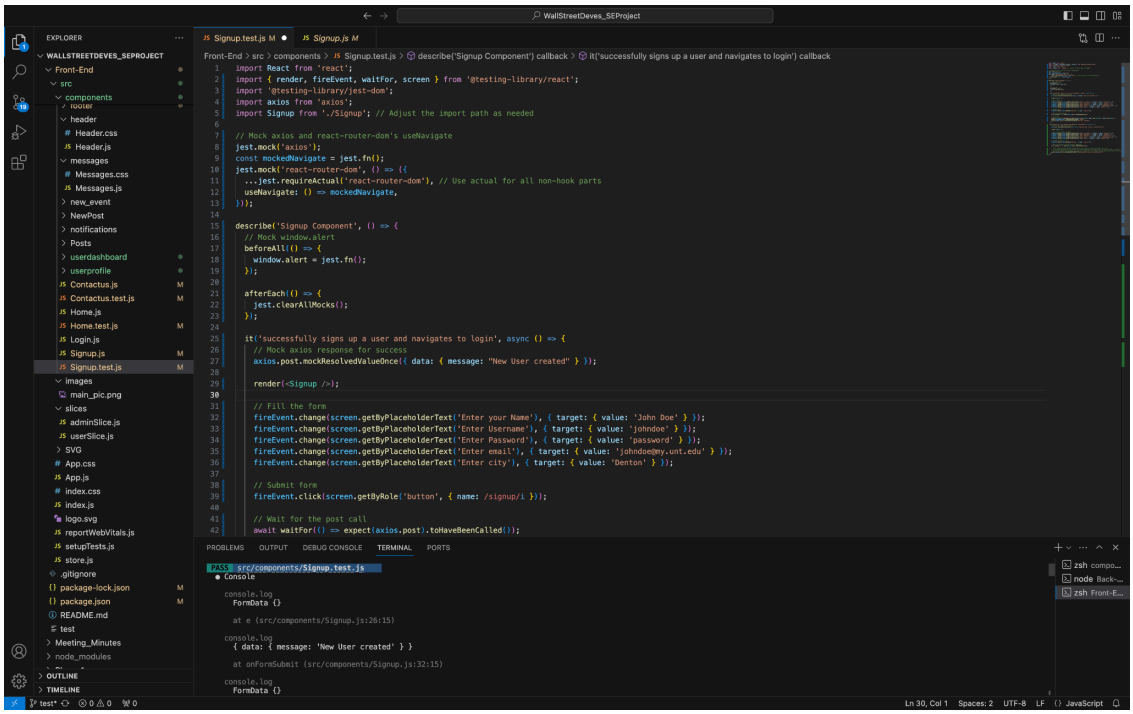


Fig:C.1

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.

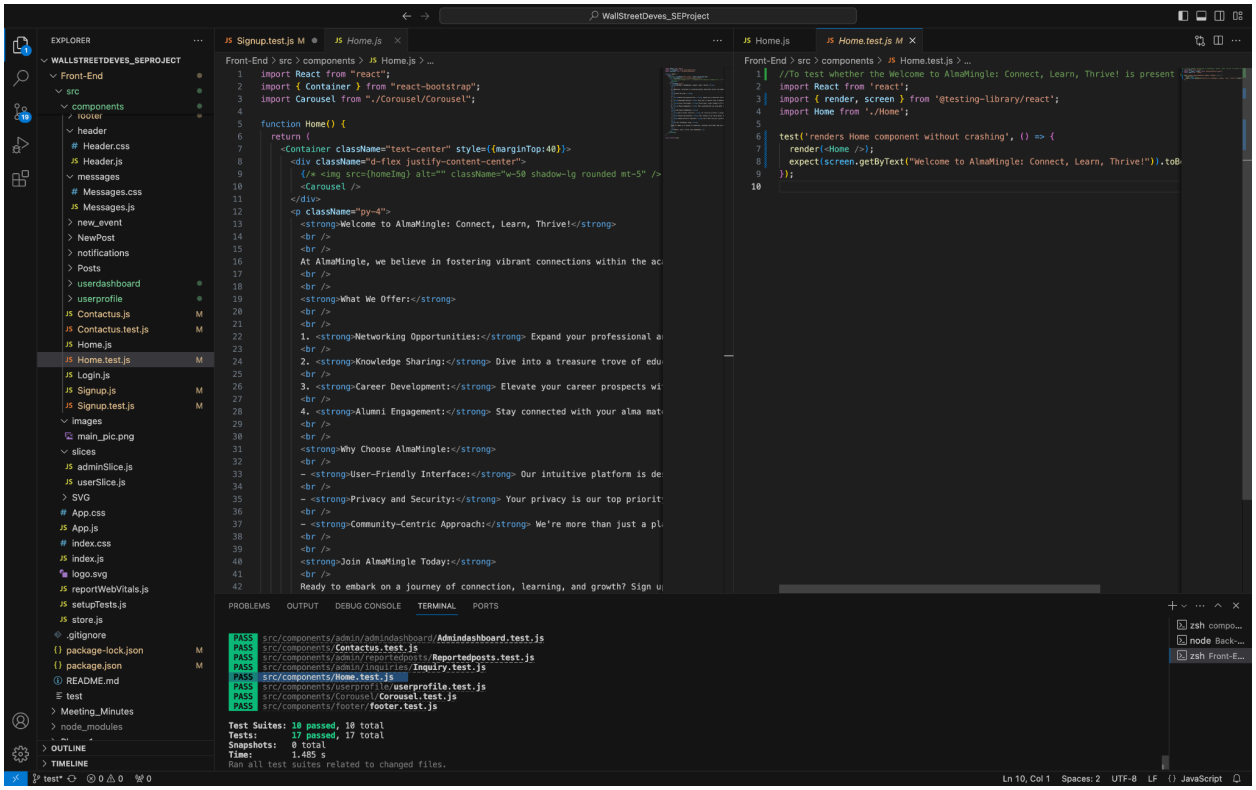


Fig:C.2

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.

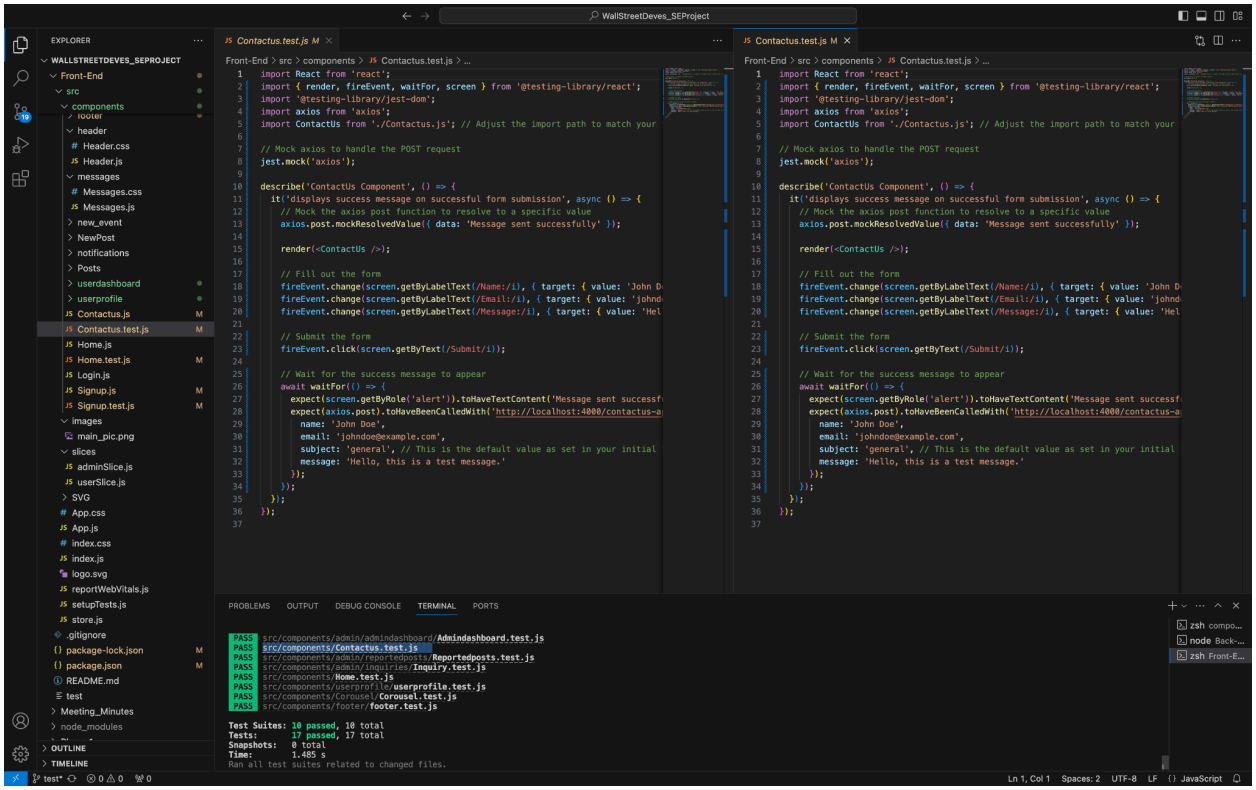


Fig:C.3

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.

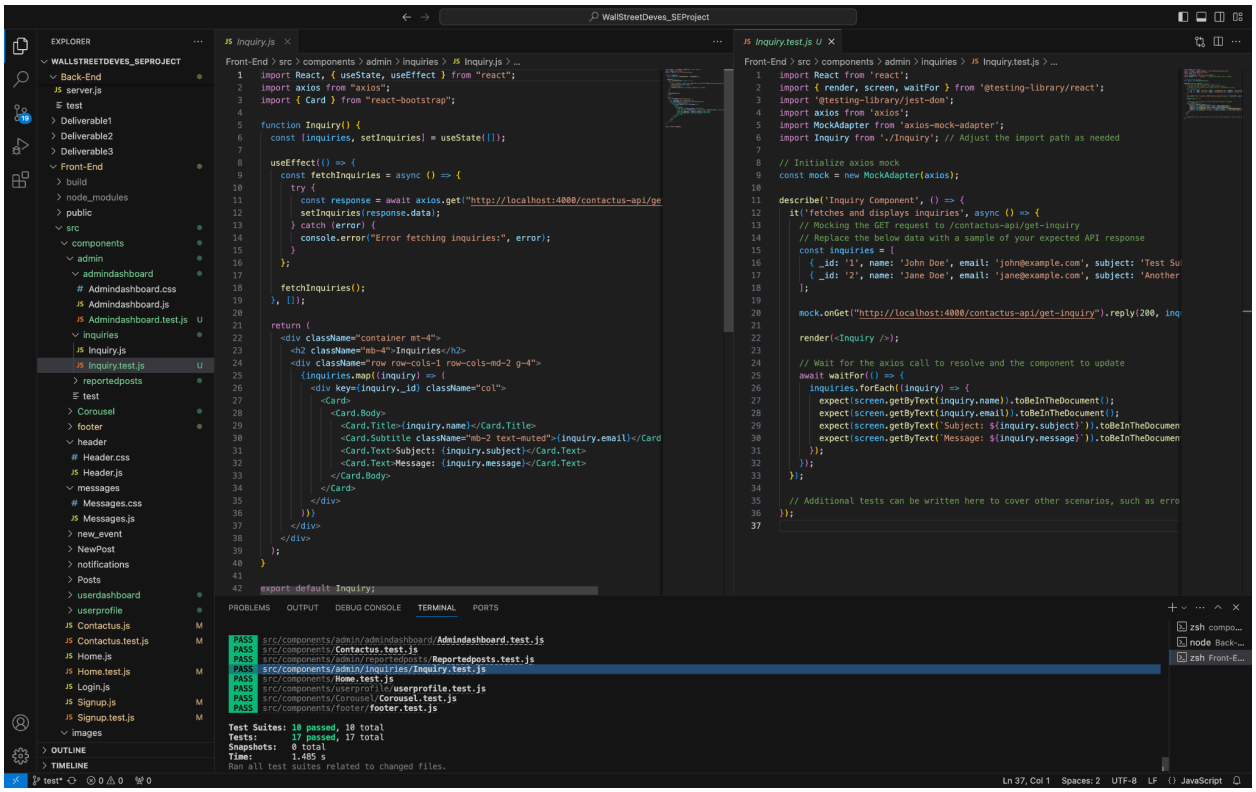


Fig:C.4

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.

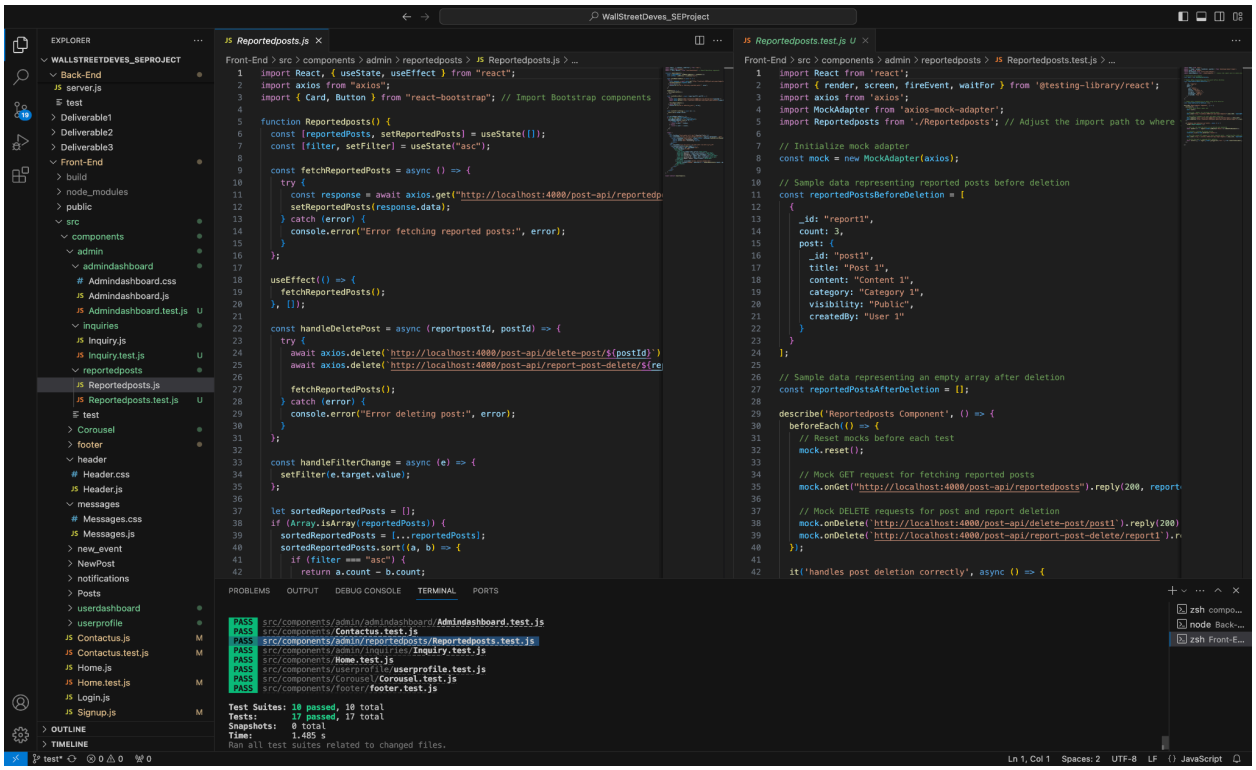


Fig: C.5

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.

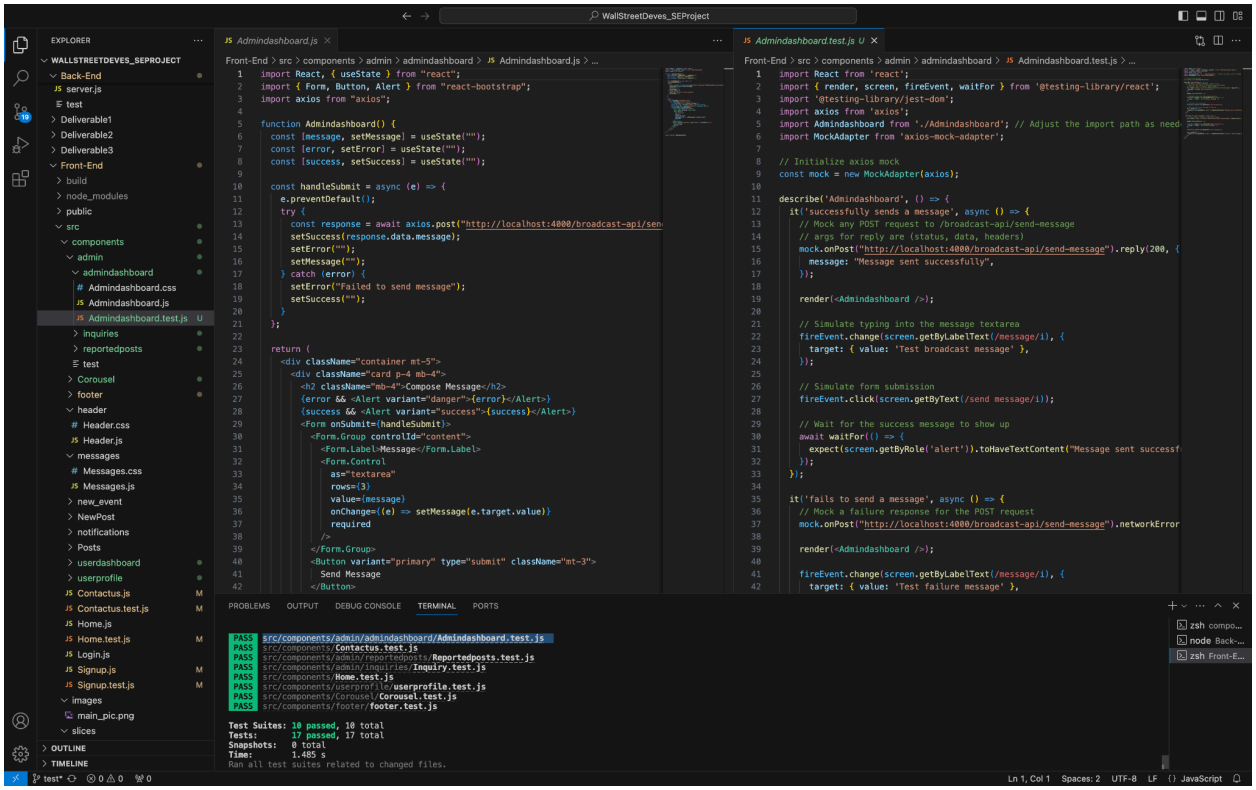


Fig: C.6

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.

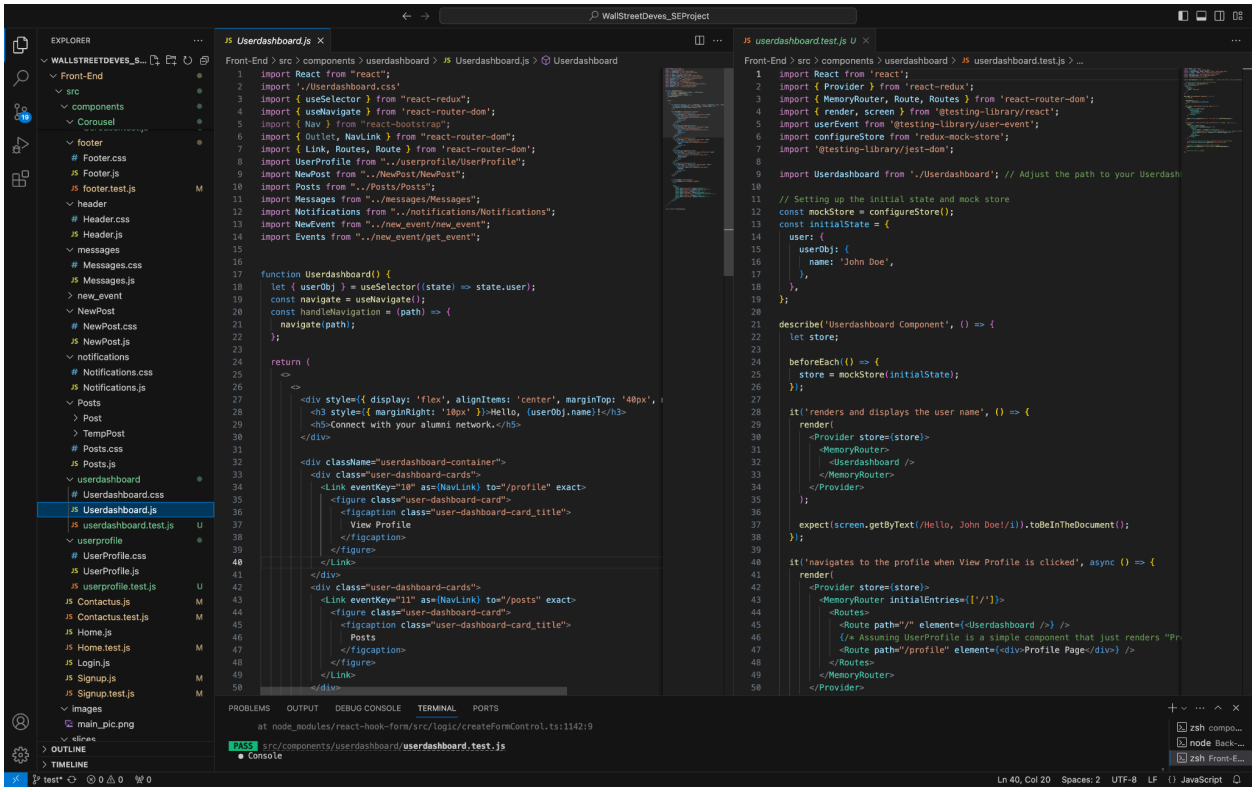


Fig: C.7

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.

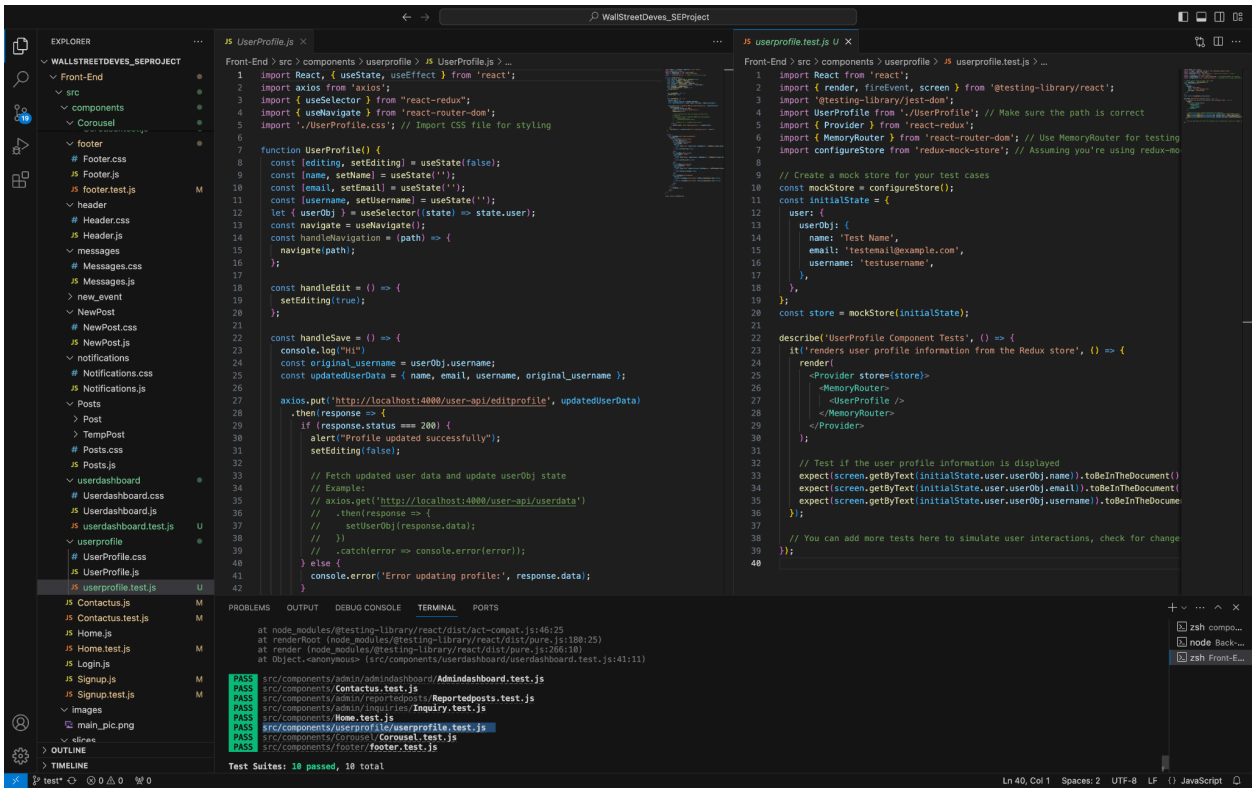


Fig: C.8

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.

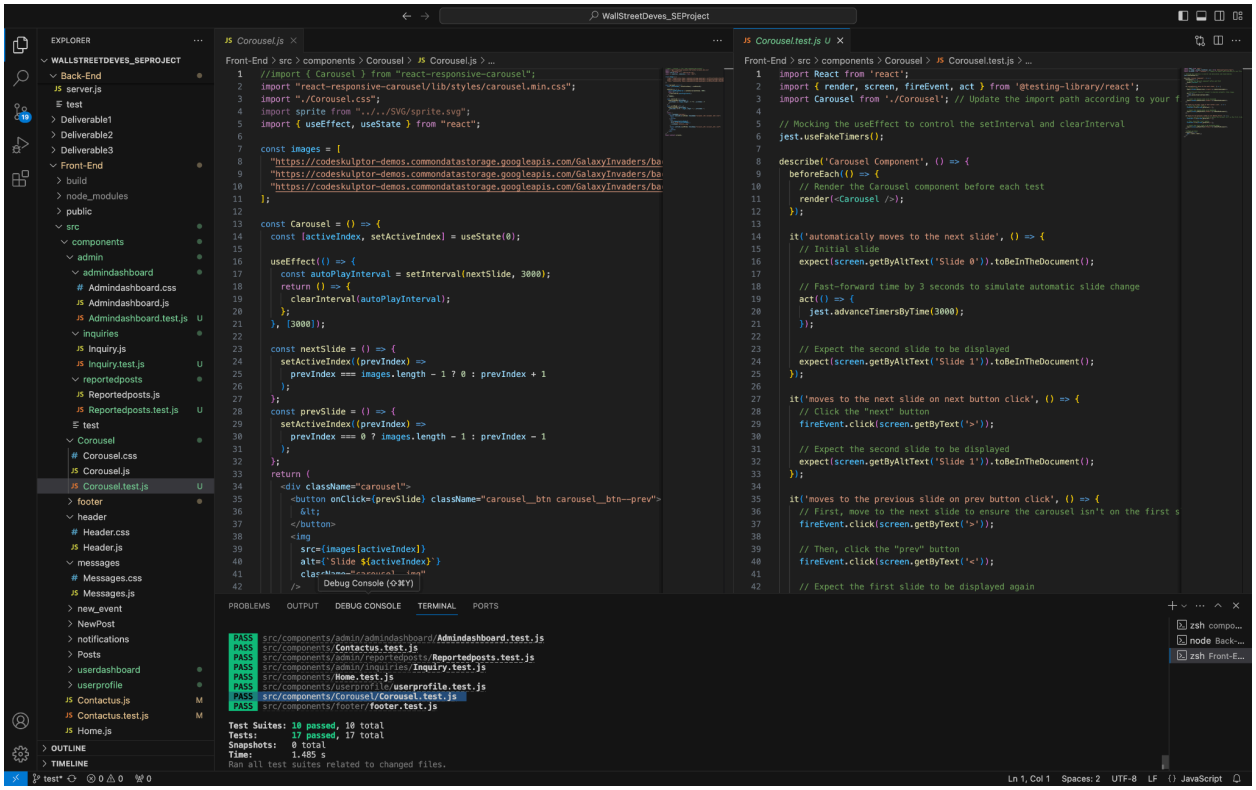


Fig:C.9

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.

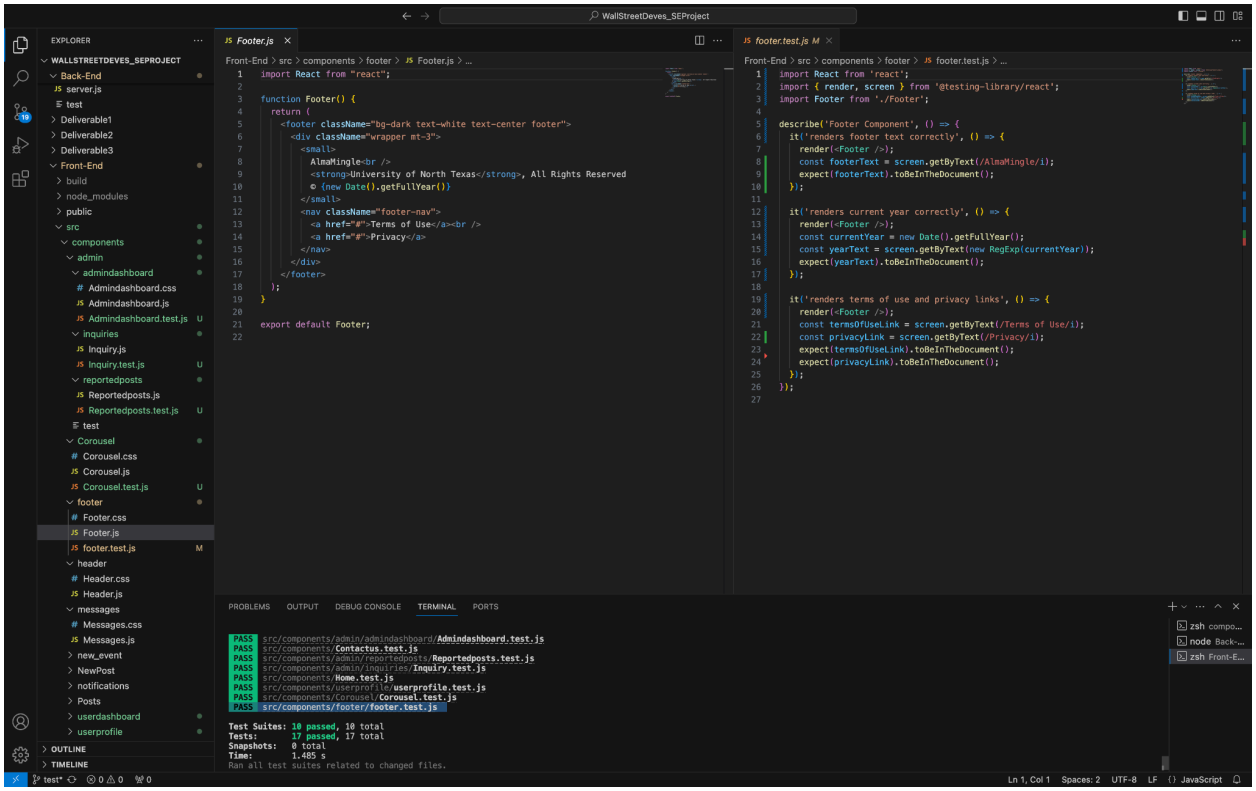


Fig: C.1

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.

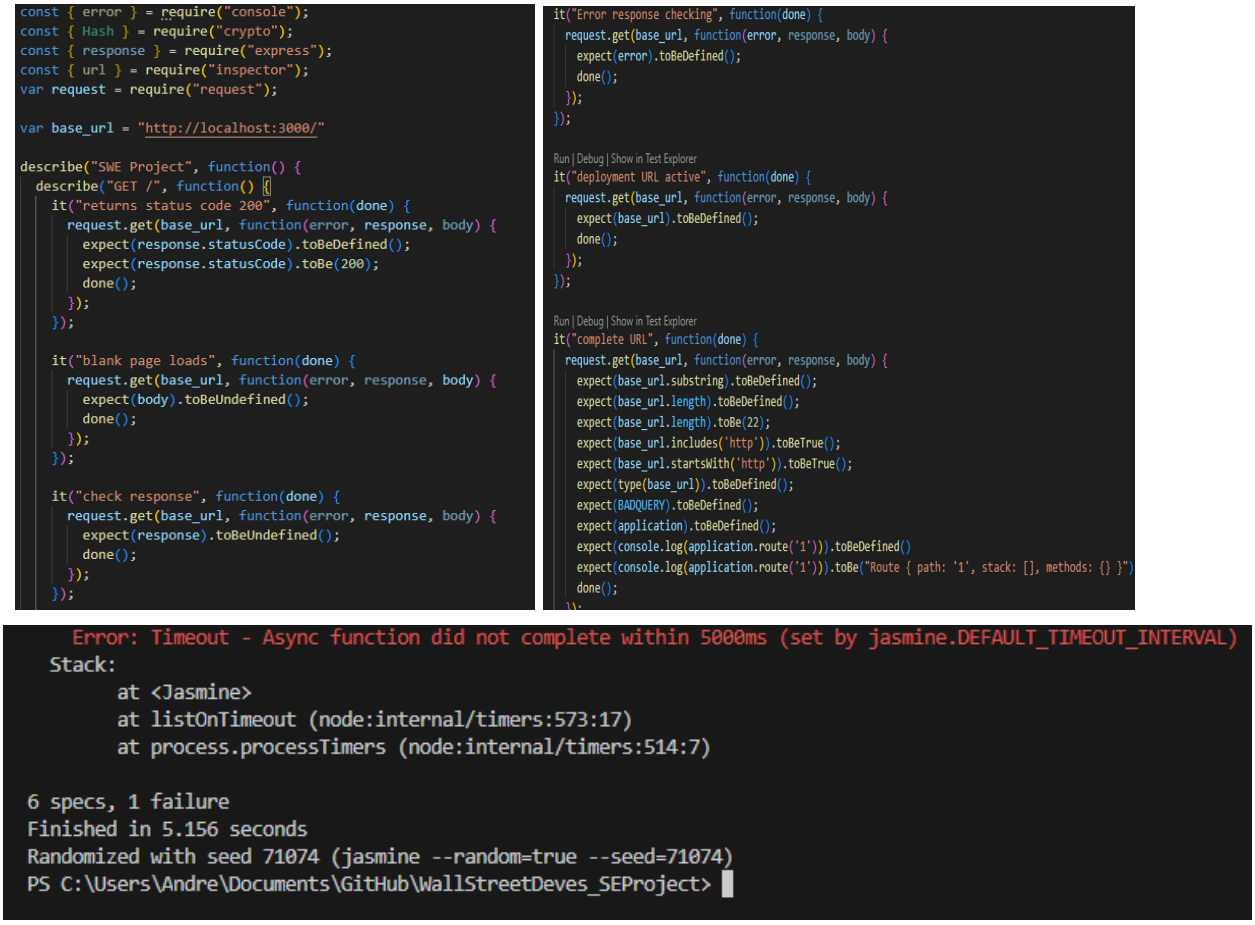


Fig: C.11, C.12 and C.13

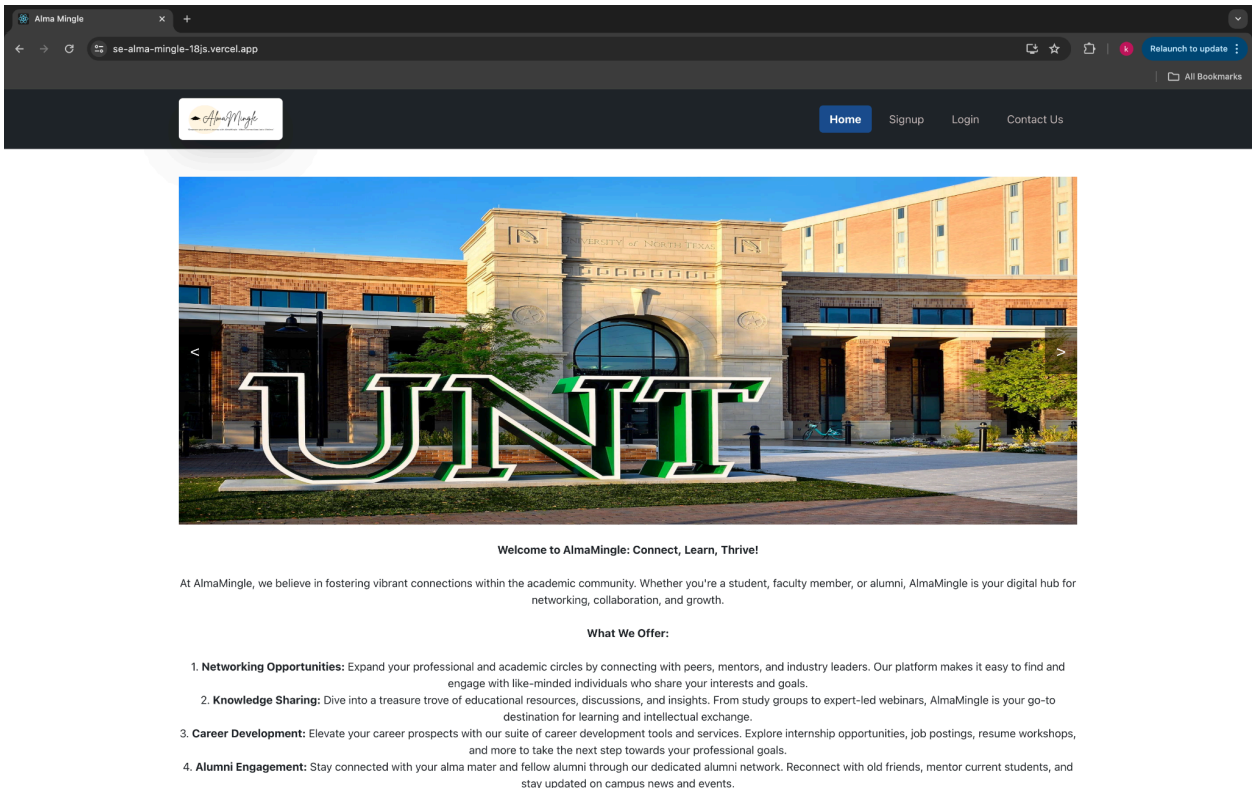
*note the single failure is to be expected because the website is not active and is tested in isolation. It will not return a status code hence the time out error.

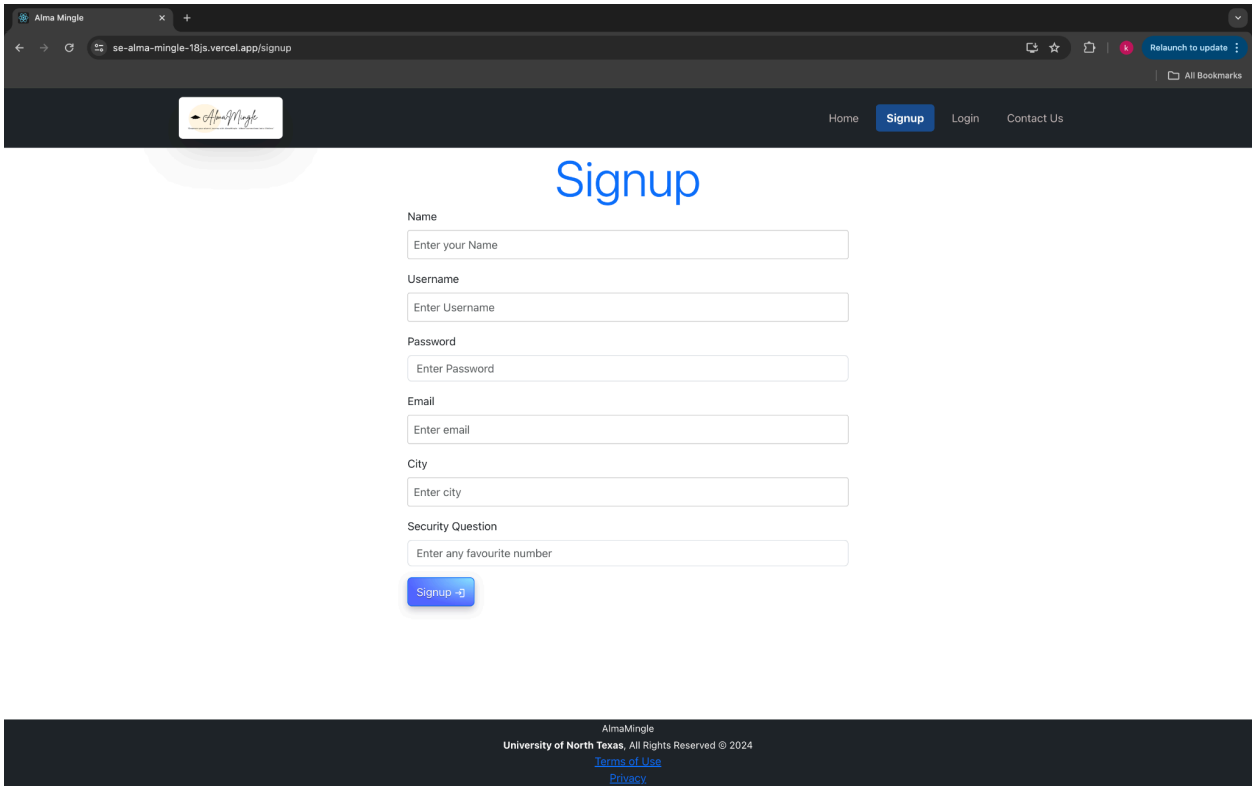
C.12 Production Testing - User Acceptance Testing

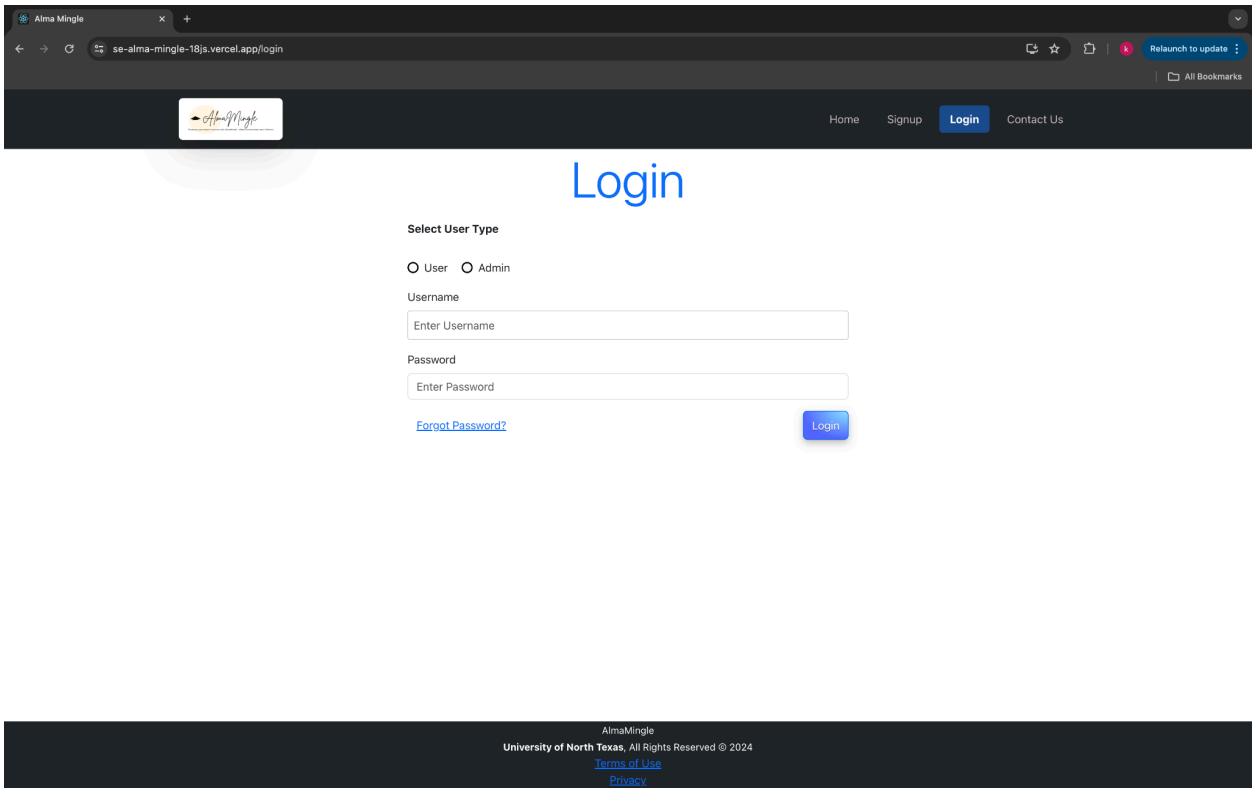
Testing performed after the site or application has been deployed and is accessible to end-users is typically referred to as "post-deployment testing" or "production testing." This type of testing ensures that the deployed system functions correctly in its live environment and that any issues arising from deployment are identified and addressed promptly. It may include various types of testing such as smoke testing, sanity testing, and regression testing to validate the system's stability and performance in the production environment.
In this scenario, we performed sanity testing, where we manually logged in to test different scenarios and functionalities of our project. We did it from three different perspectives.

1. Basic point of view
2. Admin point of view
3. User point of view

1. **Basic point of view**
   We can access our website through this active link.
   https://se-alma-mingle-18js.vercel.app/




Alma Mingle
se-alma-mingle-18js.vercel.app/signup
Relaunch to update
All Bookmarks
Home
Signup
Login
Contact Us
Signup
Name
Enter your Name
Username
Enter Username
Password
Enter Password
Email
Enter email
City
Enter city
Security Question
Enter any favourite number
Signup
AlmaMingle
University of North Texas, All Rights Reserved © 2024
Terms of Use
Privacy


Alma Mingle
se-alma-mingle-18js.vercel.app/login
Relaunch to update
All Bookmarks
Home
Signup
Login
Contact Us
Login
Select User Type
User
Admin
Username
Enter Username
Password
Enter Password
Forgot Password?
Login
AlmaMingle
University of North Texas, All Rights Reserved © 2024
Terms of Use
Privacy

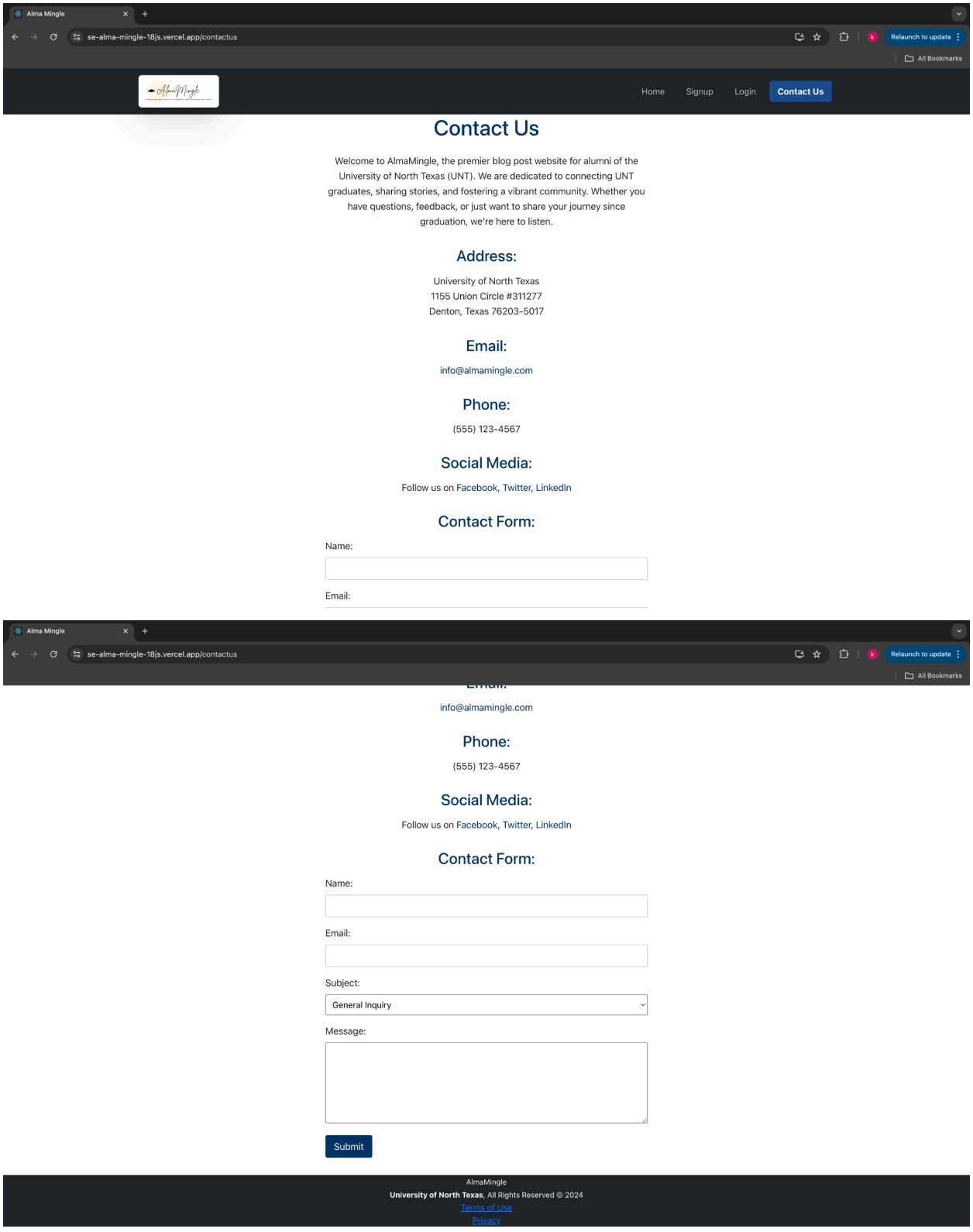


Fig: C.12.1 Basic View

This is the basic homepage, Login, Signup and contact page of our website. And, they are the expected and actual output.

2. Admin point of view

After receiving the administrative credentials from the Database Administrator (DBA), we utilized them to log in to the website's admin interface via the login page. These credentials are pivotal in ensuring the security of the system and granting authorized access to administrative functionalities. We also have forgot password functionality here. This can be used when the admin forgets their password, admin needs to provide basic details such as username and security question.

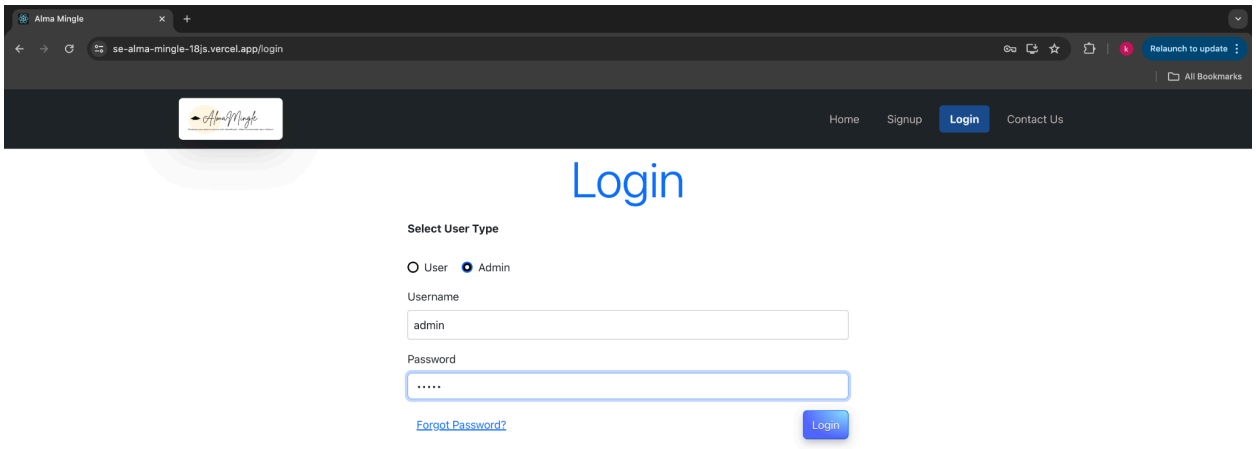


Fig: C.12.2 Admin Login View

The wait time may depend upon the bandwidth of the network.

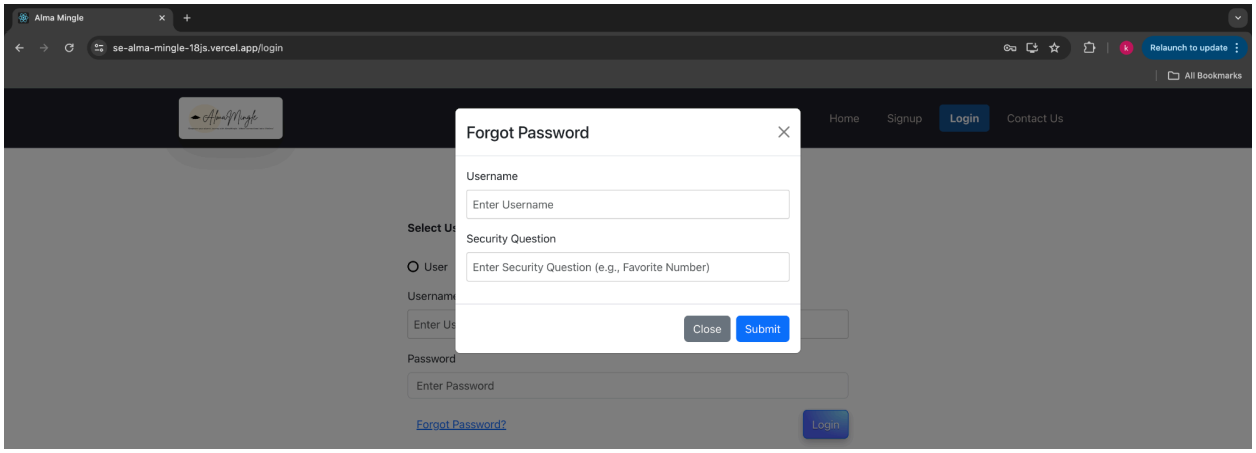


Fig: C.12.3 Admin View - Forgot password feature

Once the admin gets logged in, it will show the admin dashboard. Here, admin posted a message to other users. This message will be received by other users and this can be seen in

their notifications as well. Also, one they will receive an acknowledgement saying the message “sent successfully”.

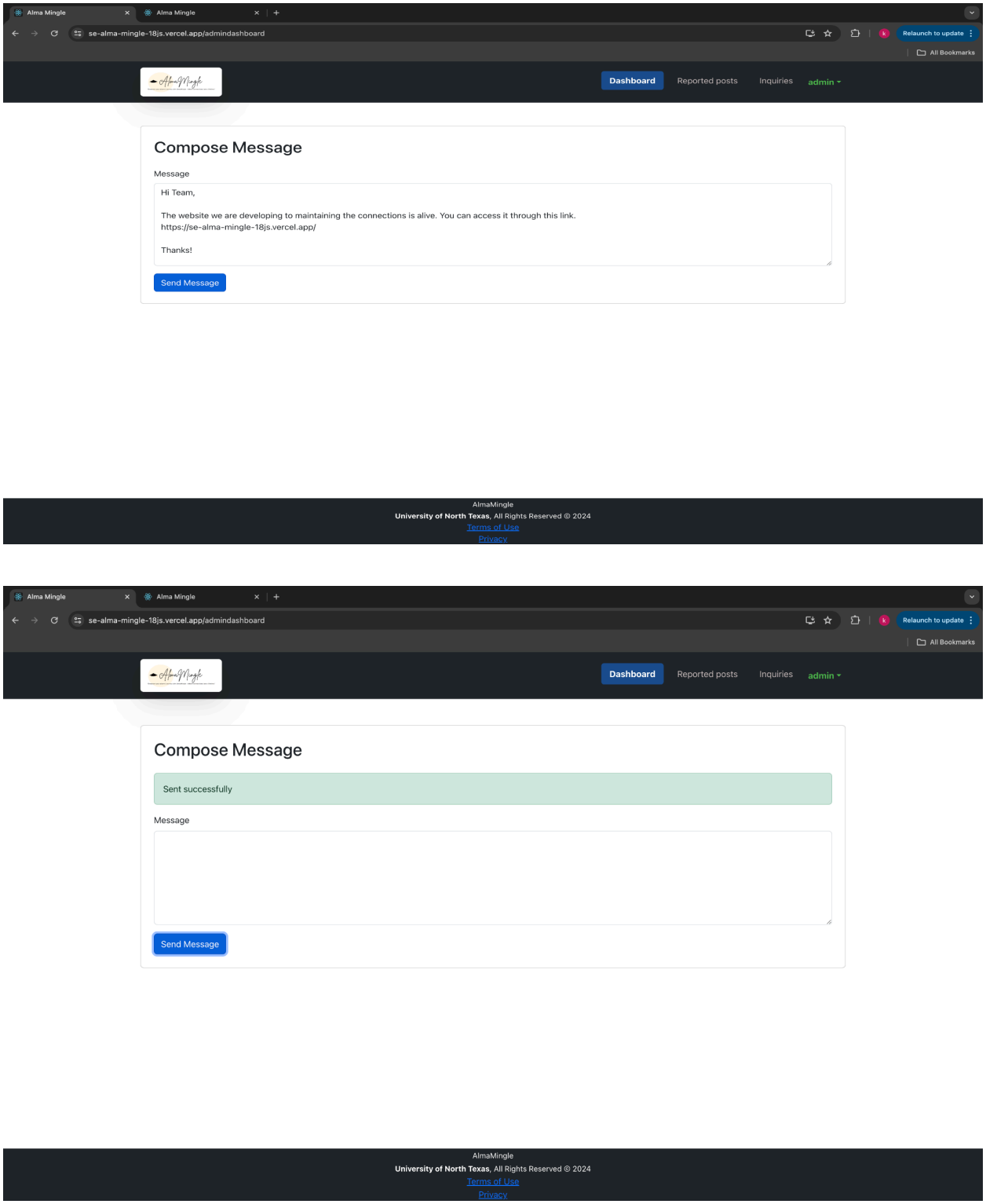


Fig: C.12.4 Admin dashboard

Inquiries, if the user has any query and wants to check with the admin then those can be visible to the admin in this section. The admin can get back to the user by using their mail Ids or the username. It is similar to the ticket concept.

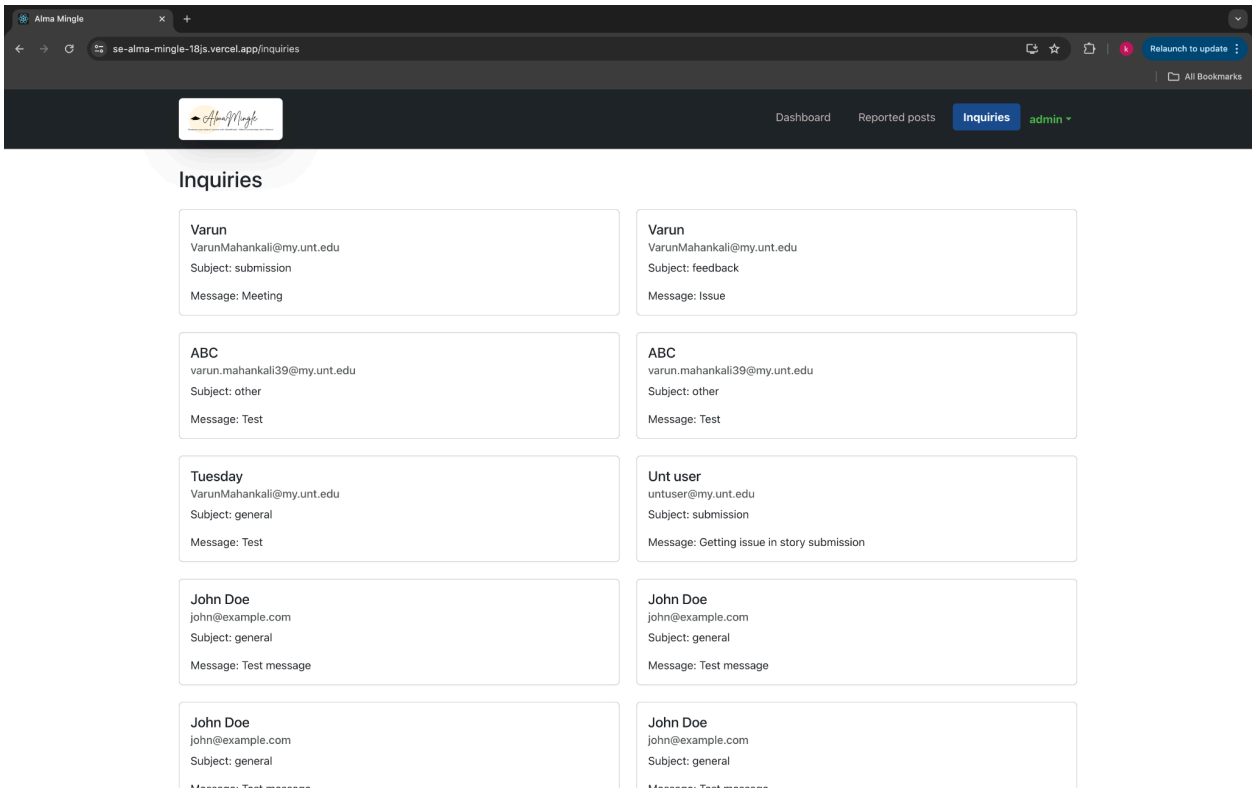


Fig: C.12.5 Inquires view

And, if a user posts something that is inappropriate, the admin has the right to delete the user posts. Then can be handled in the reported posts section.

When the admin deletes the reported post, it will get removed from the reported posts and the users window as well.
Once everything is done, if the user wants to logout or end the session, they can select the logout option for the same. After sometime if there is any inactivity it will get logged out automatically. It will redirect to the login page.

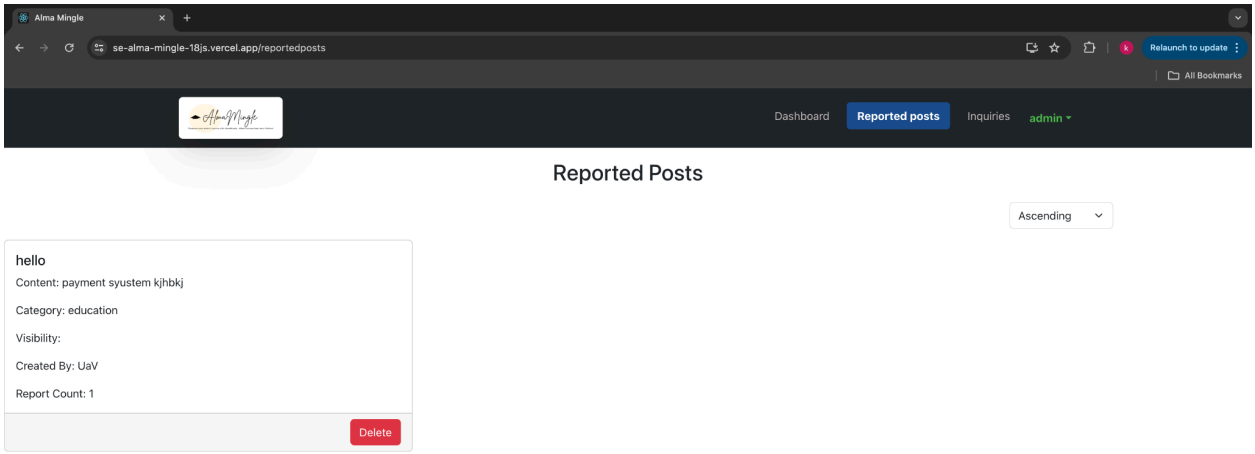
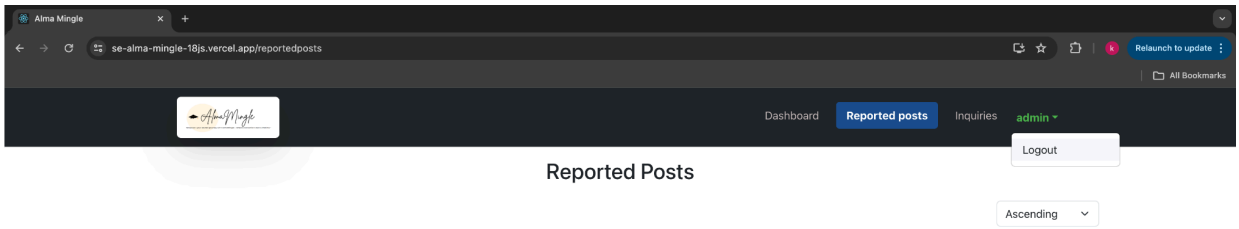


Fig: C.12.6 Reported Posts view



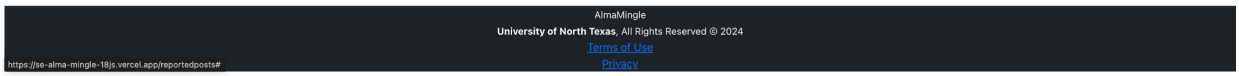


Fig: C.12.7 Logout

3. User point of view

To evaluate the user experience, we'll initiate the creation of a new account via the signup page. Ensuring adherence to standard criteria, we'll assign a distinct username and password. Given that our platform caters to alumni of UNT, it's imperative that the signup email addresses end with "@my.unt.edu" to proceed successfully. Any deviation from these requirements will prompt

error notifications highlighting the issue. Recognizing the likelihood of users forgetting their passwords, a security question feature has been incorporated to facilitate password recovery in such instances. After the successful signup the person is provided with the pop-up message from the website saying “New User Created”. When the person clicks “OK” it will redirect to the login page. This implies the successful authentication and authorization has been implemented while maintaining the security standards.

Later, the user needs to provide the credentials in the login section using the login page. This will redirect the user dashboard. The wait period depends on the bandwidth of the website.

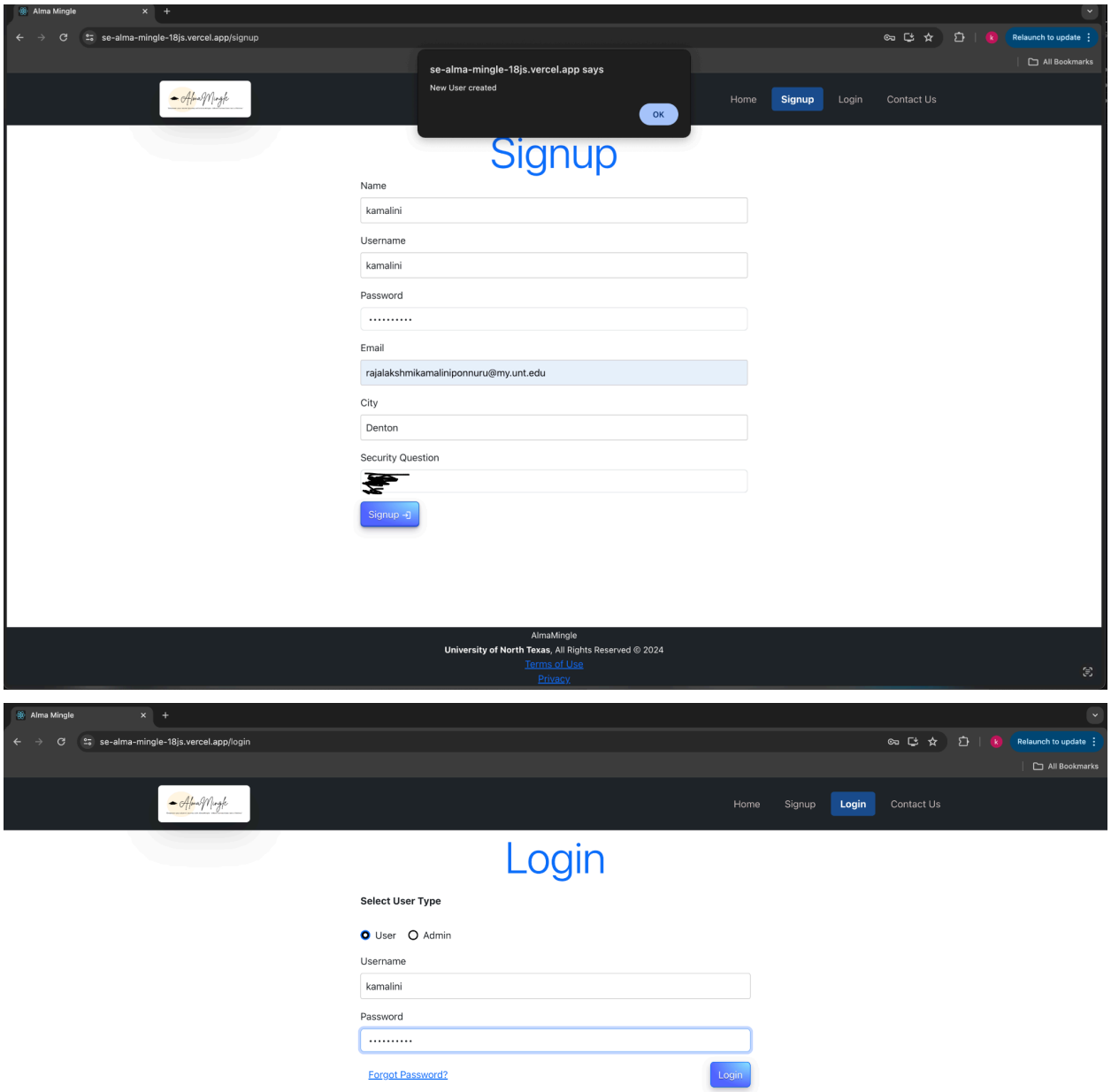


Fig: C.12.8 User login

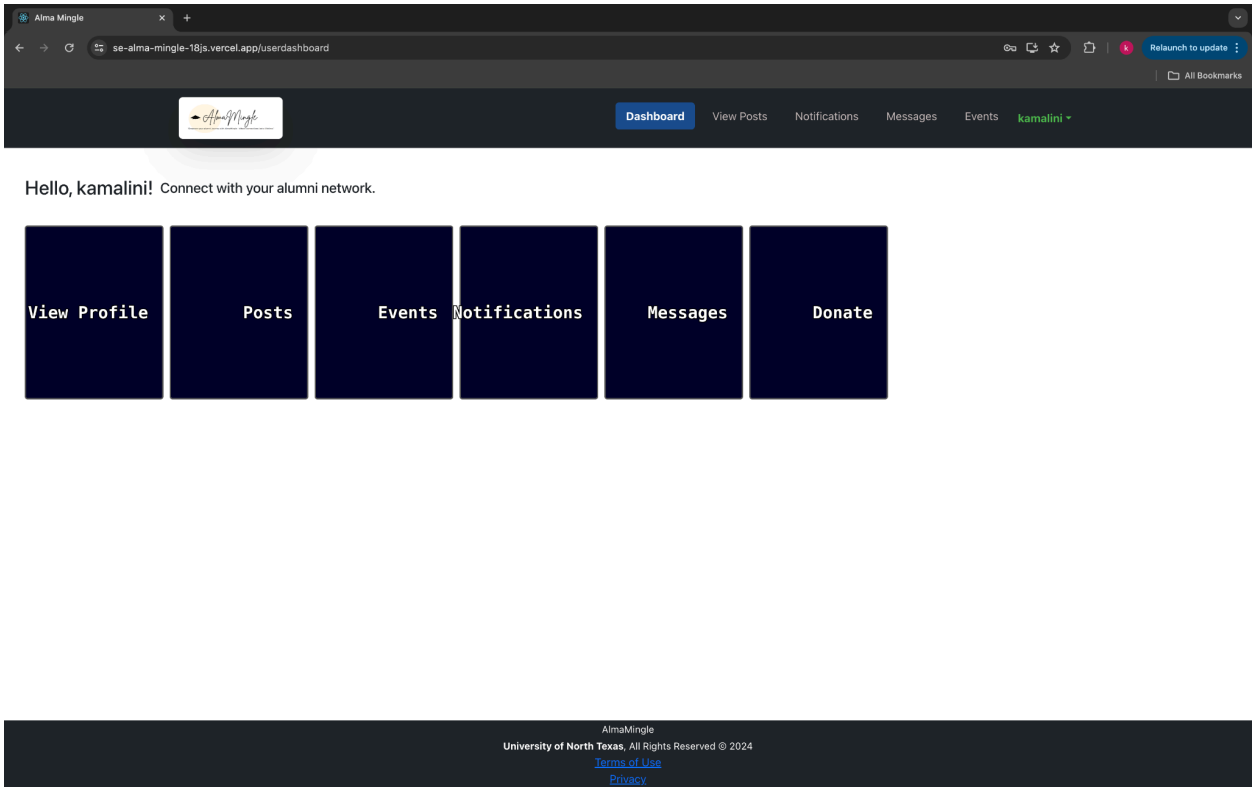
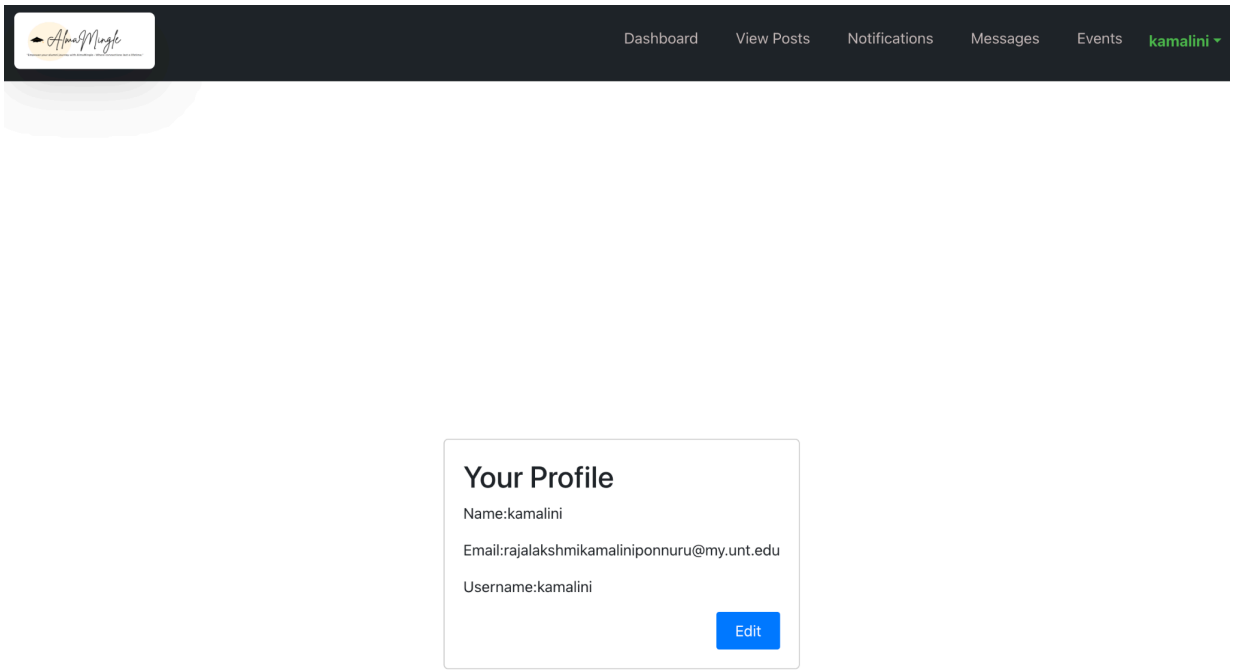


Fig: C.12.9 User Dashboard

When we click on the View profile section, we can edit the user name, by providing the requested details.



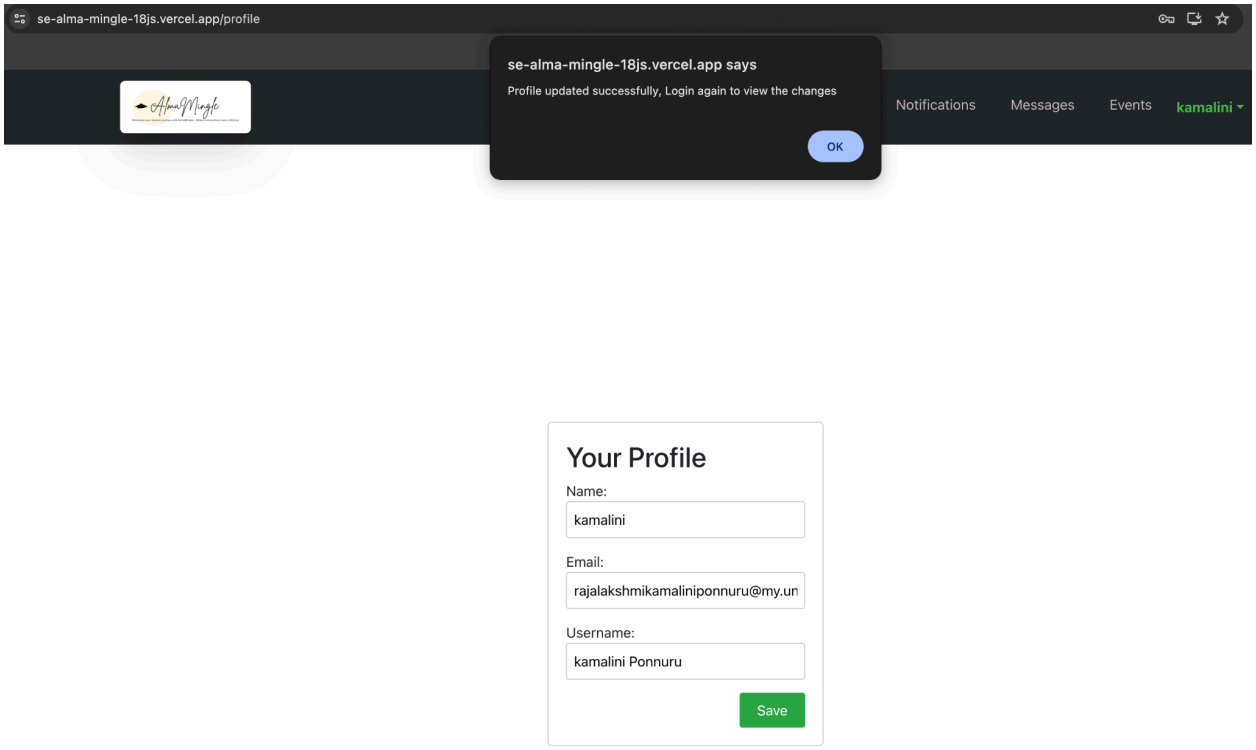
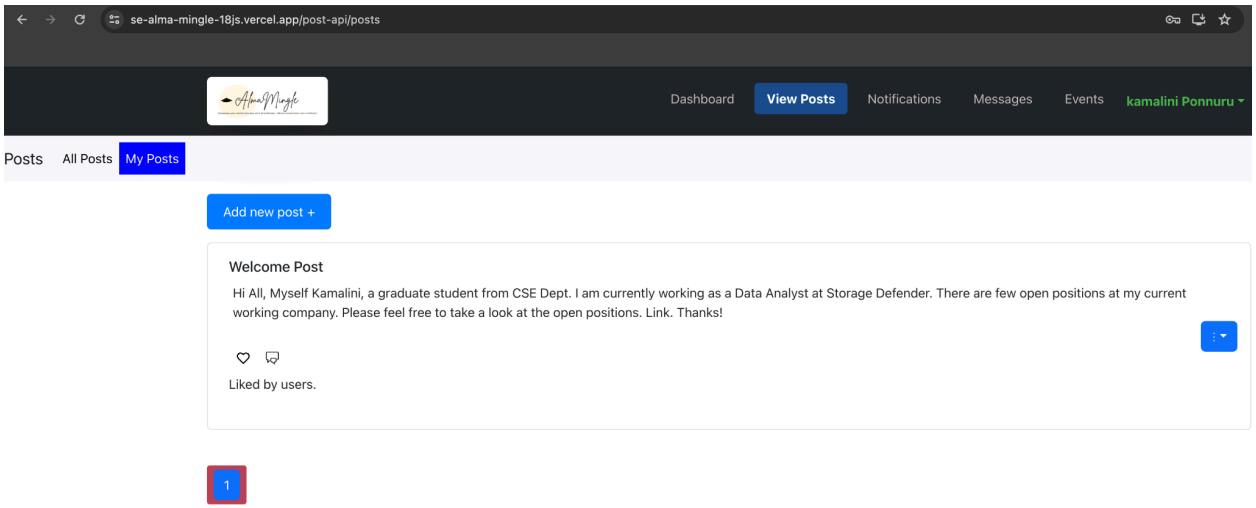


Fig: C.12.10 Profile view

Once the edit gets done it will give up a pop-up box saying “Profile updated successfully, Login again to view the changes”.

Moving to **posts section**, we can create a post similar to post functionality, in famous platforms like linkedIn by providing the required details. After clicking on create a pop-up box will be created saying “New Post Created”. The created post will be visible in the “ My posts ” section. In the “ All posts ” section, users can see all other posts posted by other users. The users can react to the posts and add their comments if they wish to. Also, we can see the user name being updated from Kamalini to Kamalini Ponnuru.



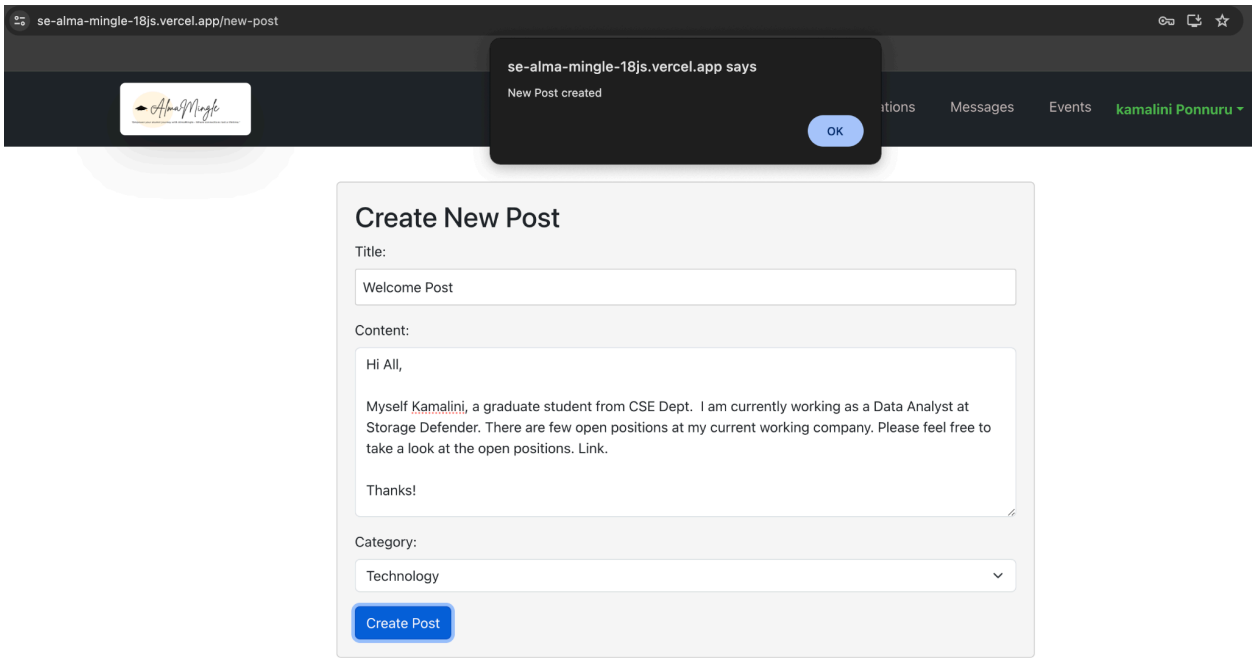


Fig: C.12.11 Post implementation

Events Section

Figure C.12.12 shows the pop up message confirming the user after creation of new event by entering the details.

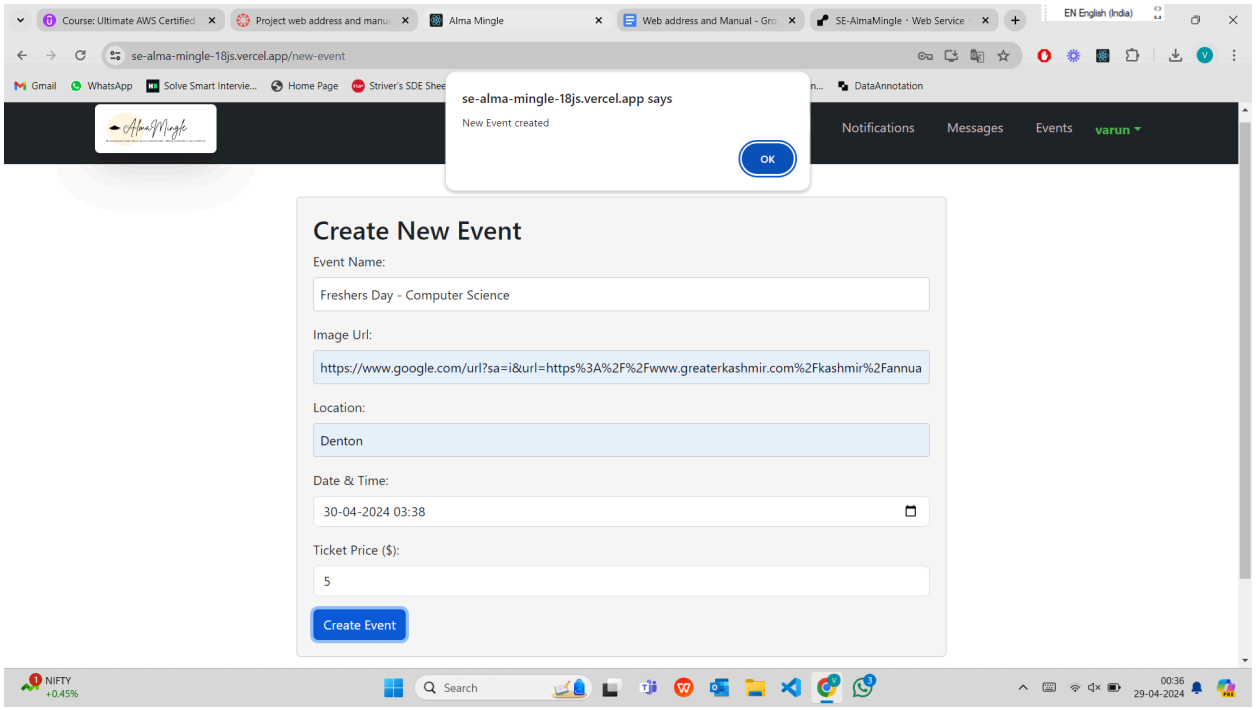


Fig: C.12.12 Events section

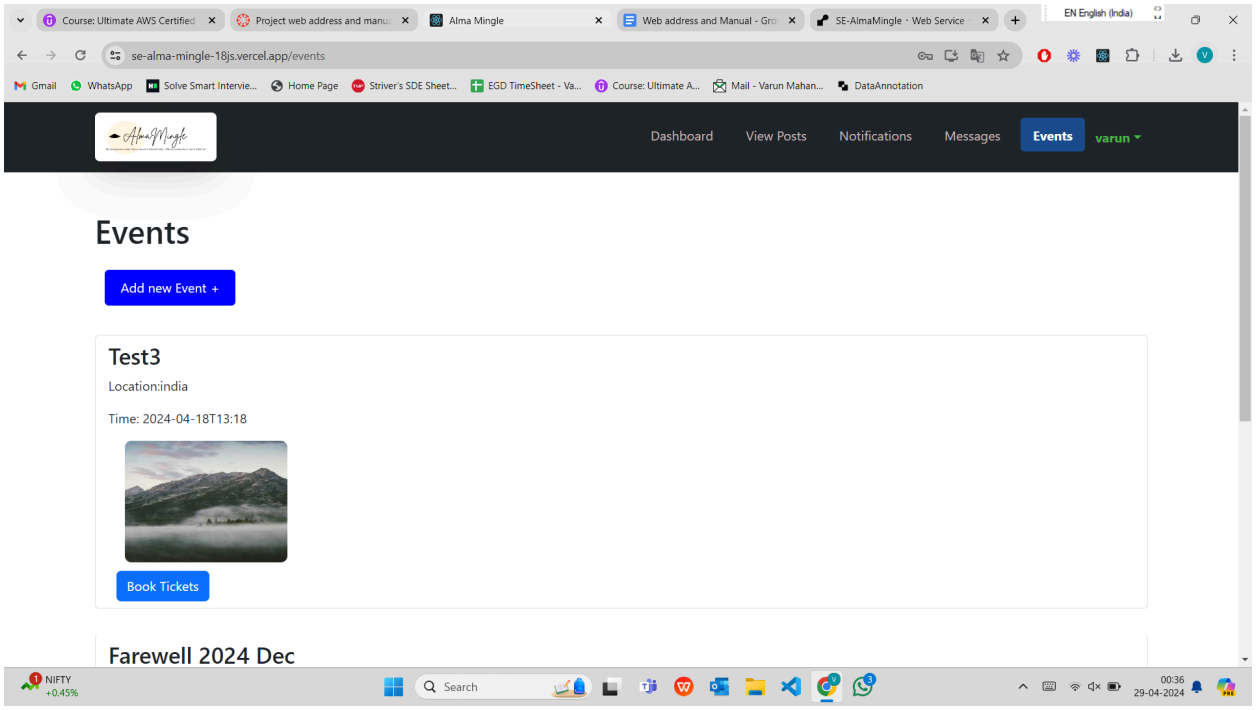


Fig: C.12.13 Events section

Figure C.12.13 shows all the events created by other users. Other users can register to the event by booking tickets.

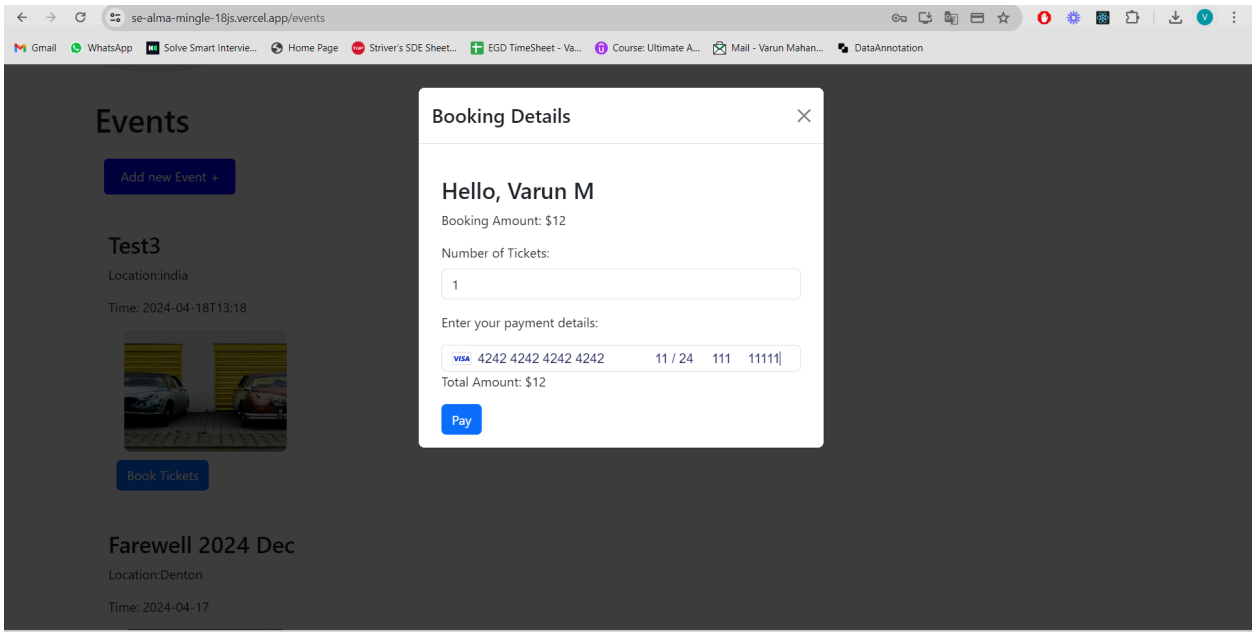


Fig: C.12.14 Events Booking

Figure C.12.14 shows the modal when the “Book Tickets” button is clicked. The user can enter details of the card and book the number of tickets they want.

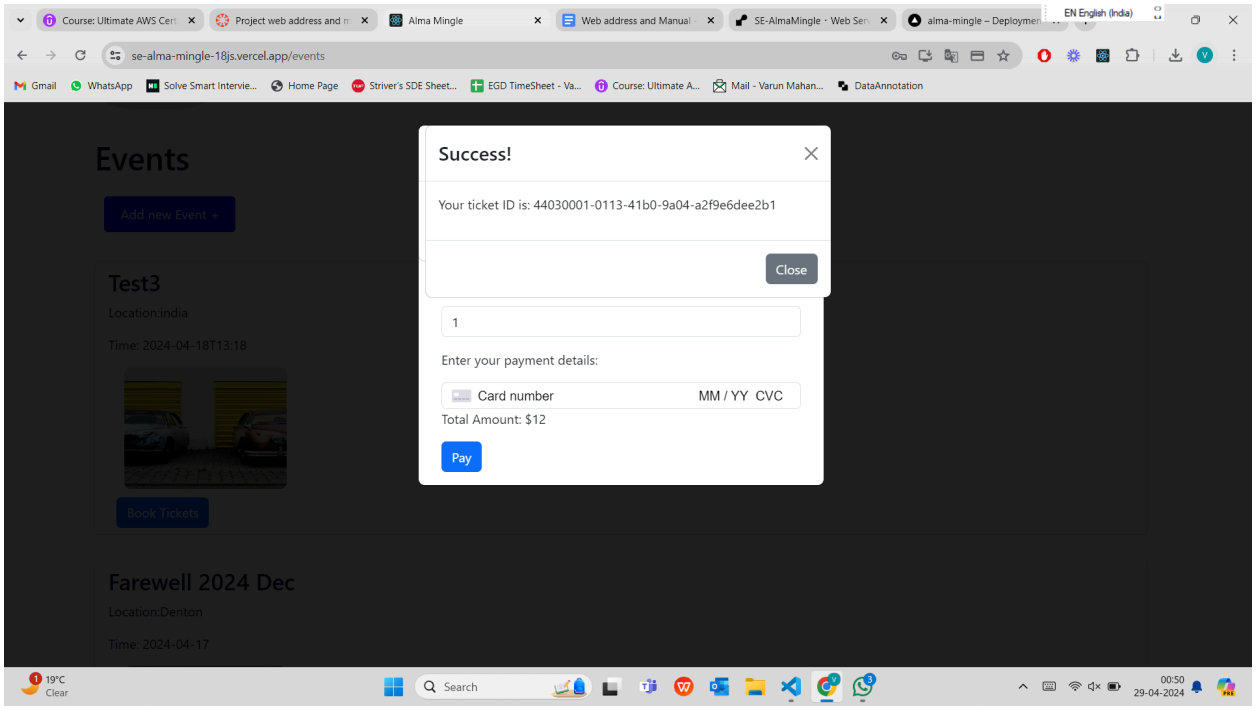


Fig: C.12.15 Booking success confirmation

Figure C.12.15 shows the Ticket ID sent to user after the payment is success.

Donate Section

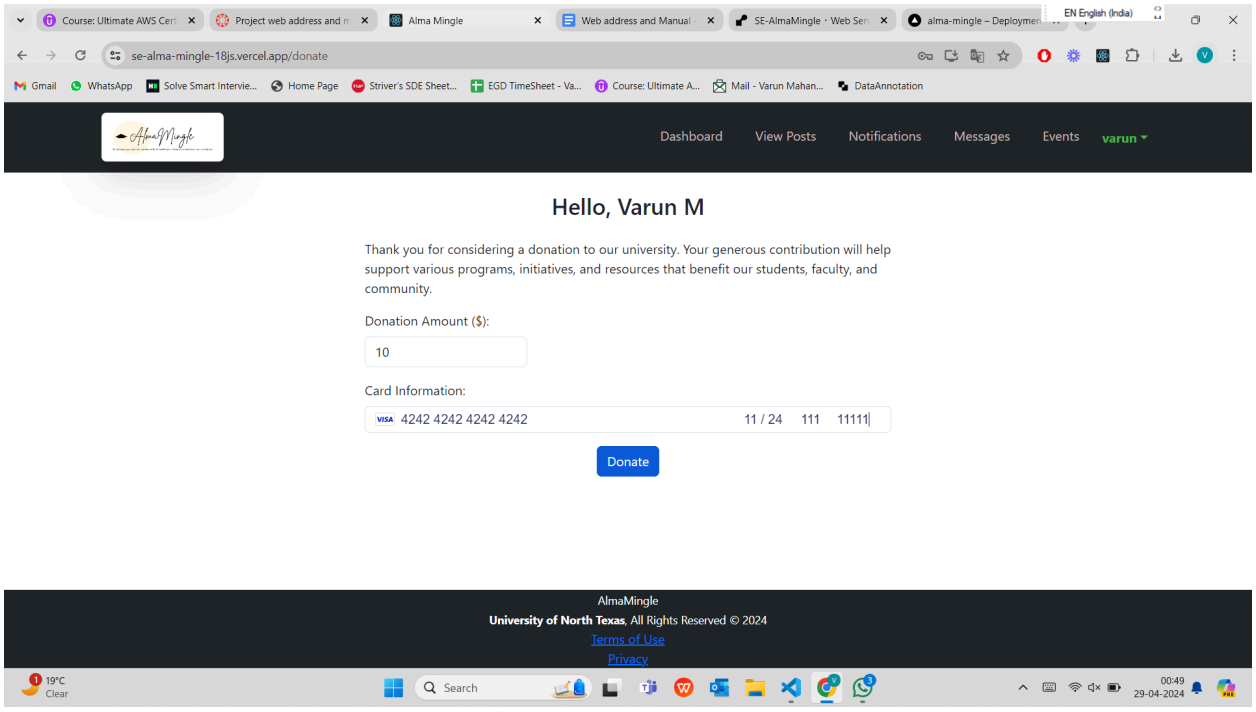


Fig: C.12.16 Donate Section

Figure C.12.16 is the Donation page which is available on the userdashboard. The user can enter the amount they want to donate and enter the card details and click donate.

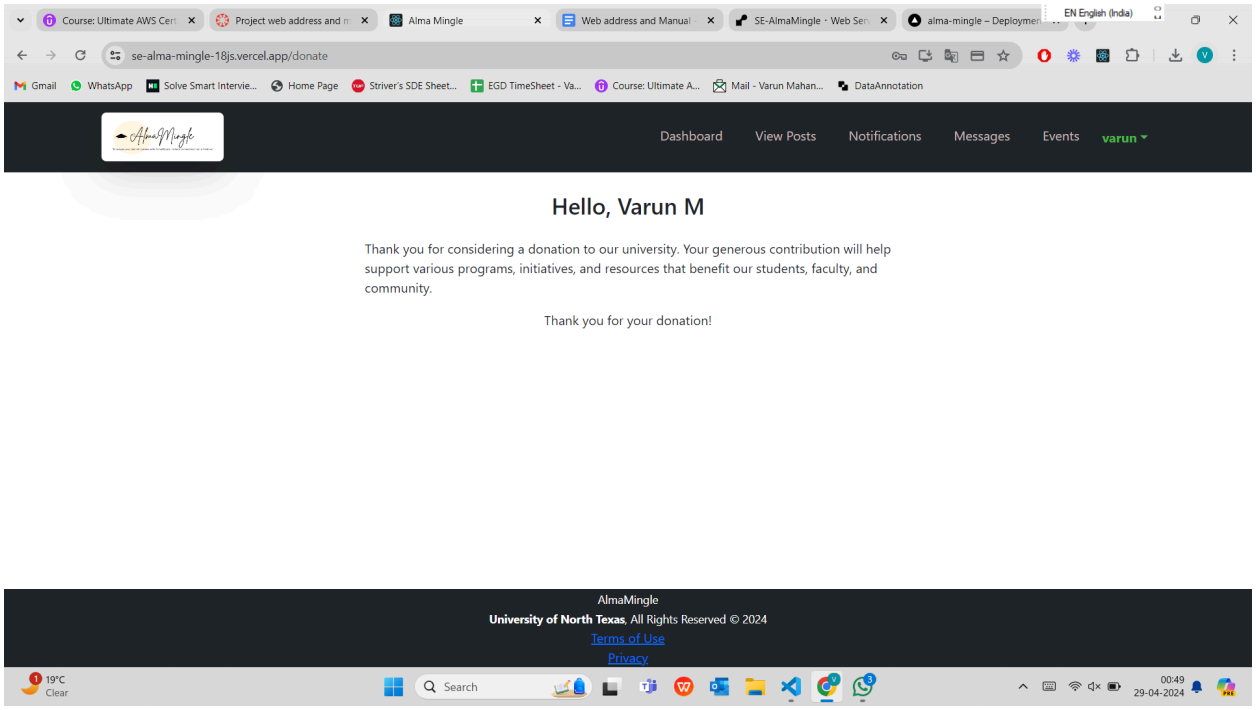


Fig: C.12.17 Confirmation after Donation

Figure **C.12.17** shows the Confirming page to the user after the donation payment is success full.

Notifications Section

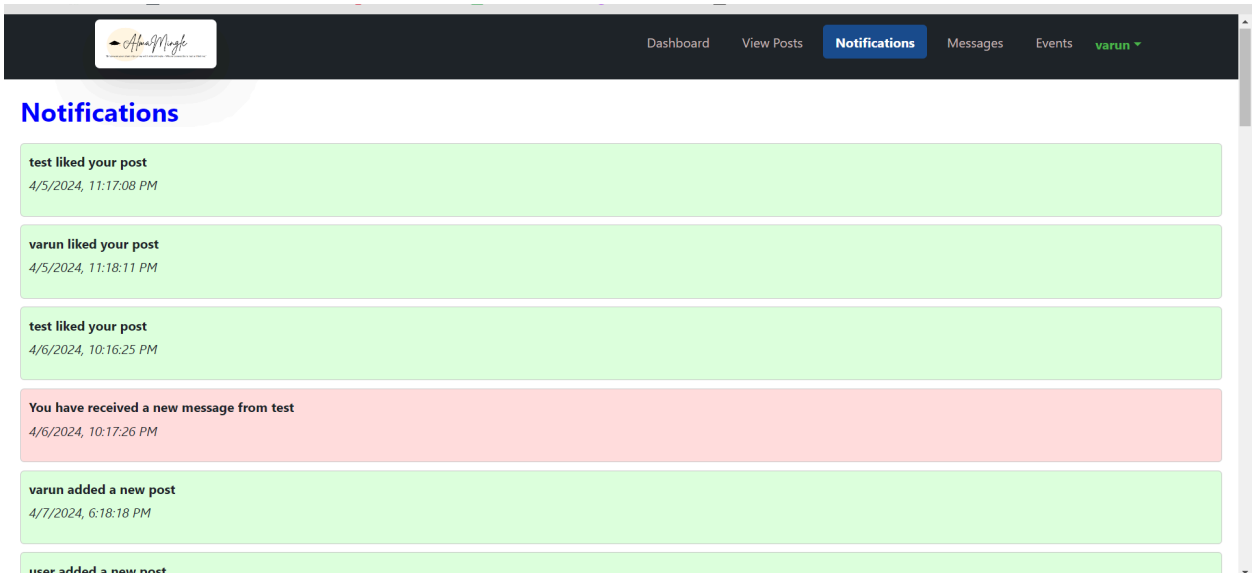


Fig: C.12.18 Notifications

Figure C.12.18 shows Notification pages that are received to the user. Notifications like when the post is liked, when the post is commented, when the user receives a message from user, admin and when the user's post is deleted by admin.

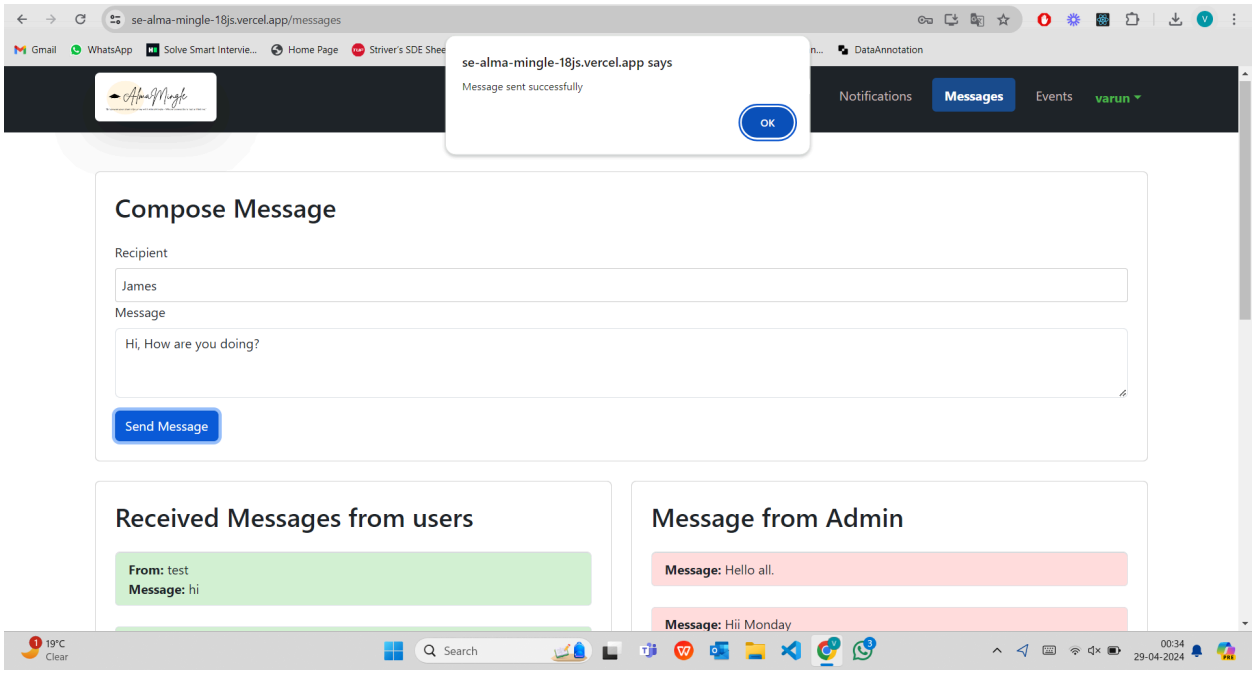


Fig: C.12.17 Messages

Figure C.12.17 shows the Message page, the user can send messages to other users. The figure shows a pop up message confirming to the user regarding the message sent confirmation.

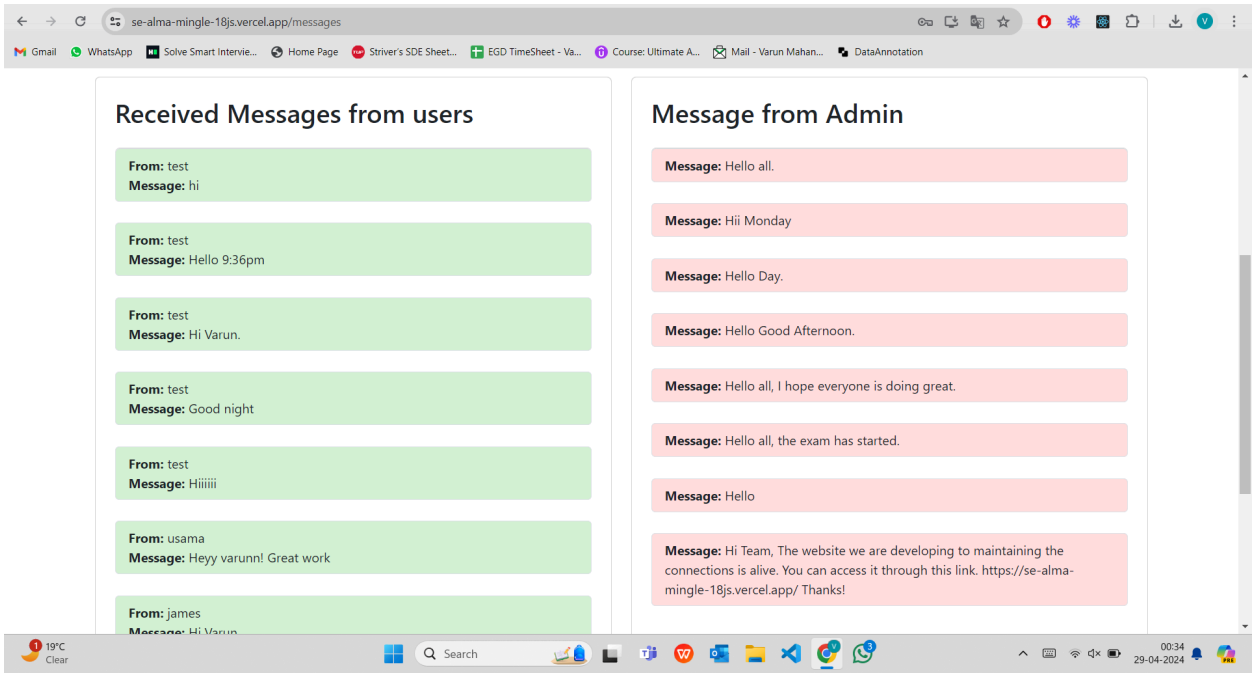


Fig: C.12.18 Messages

To summarize, we have tested the sanity for all the different sections present in the user perspective such as **View Profile, Posts, Events, Notifications, Messages and Donate.** And also, tested the Navigation Bar functionalities.

D. USER MANUAL

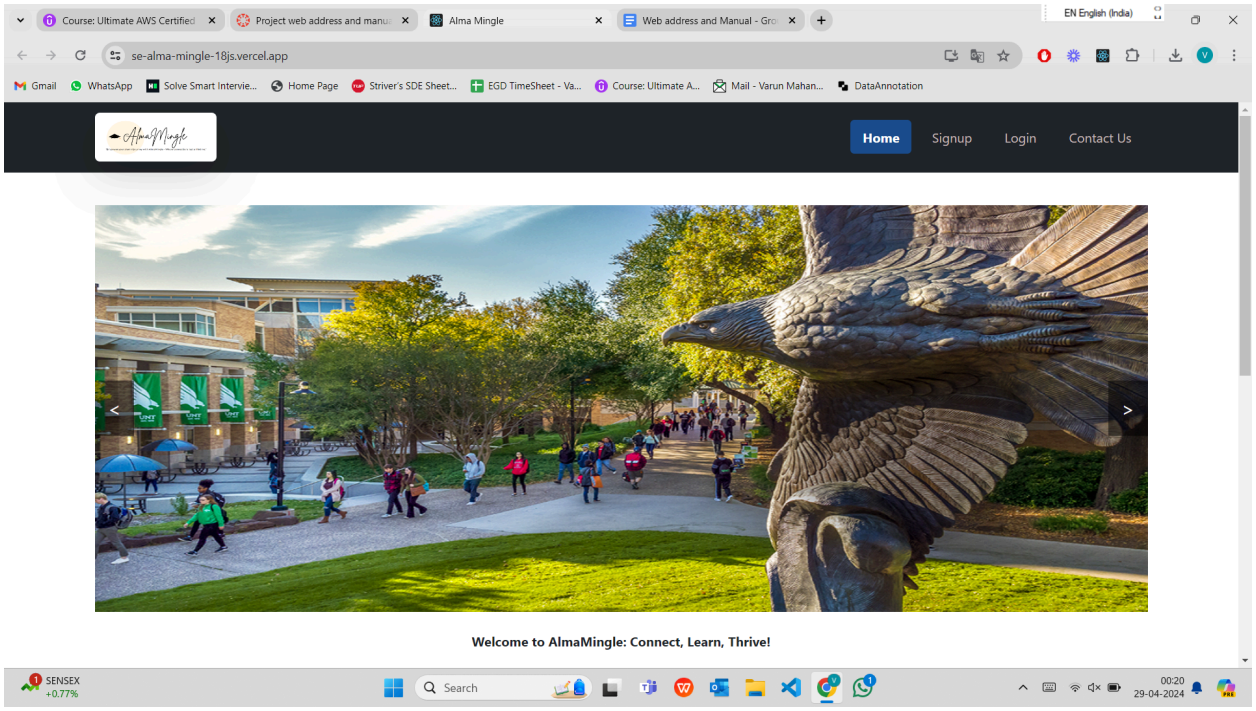


Figure D.1

The above figure D.1 is the home page of our application "**Alma Mingle**". The home page consists of Images of the UNT and description regarding the webpage.

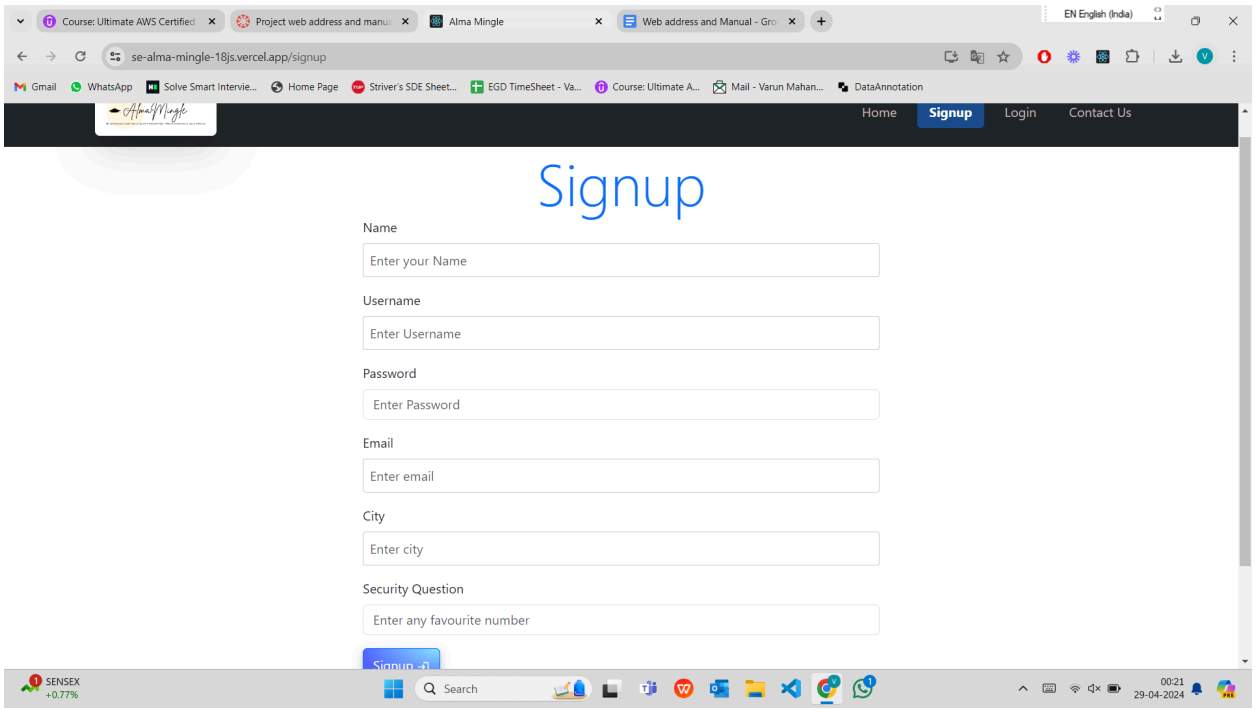


Figure D.2

The above figure D.2 shows the signup page, where the User can register their account only with their Unt mail id.

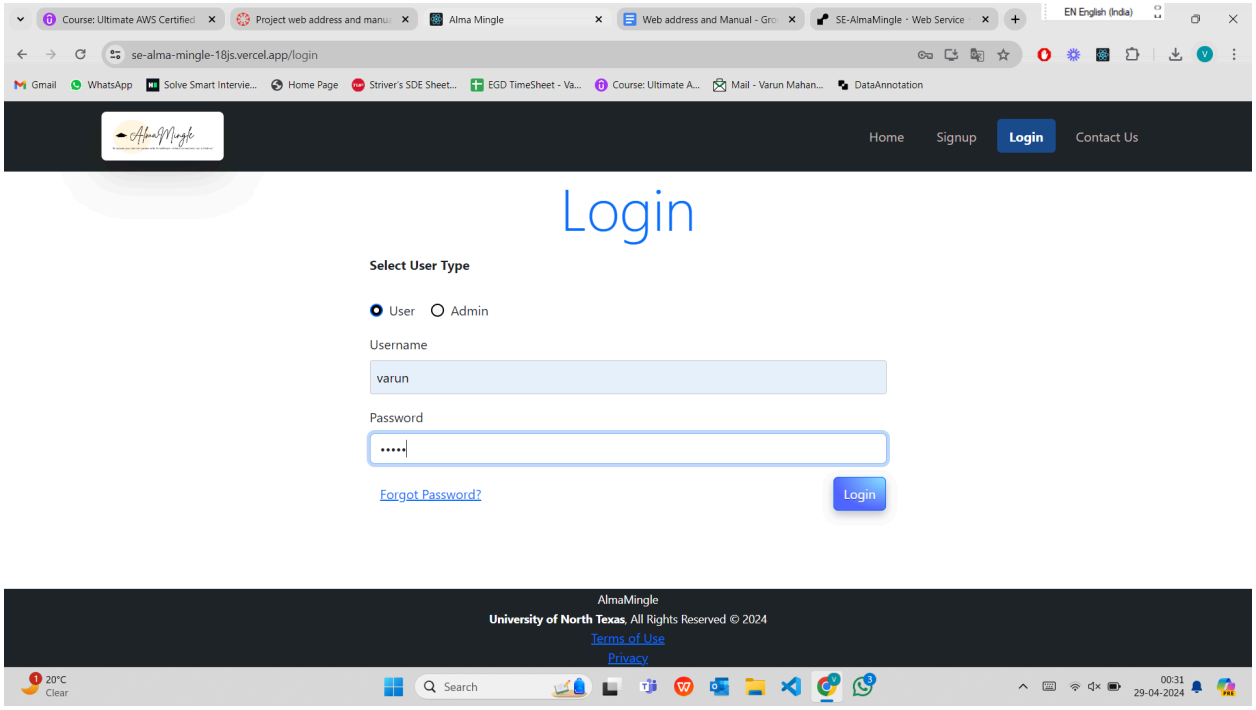


Figure D.3

The above figure D.3 shows the Login page, where the User needs to select User checkbox and can Login to their account by entering correct credentials.

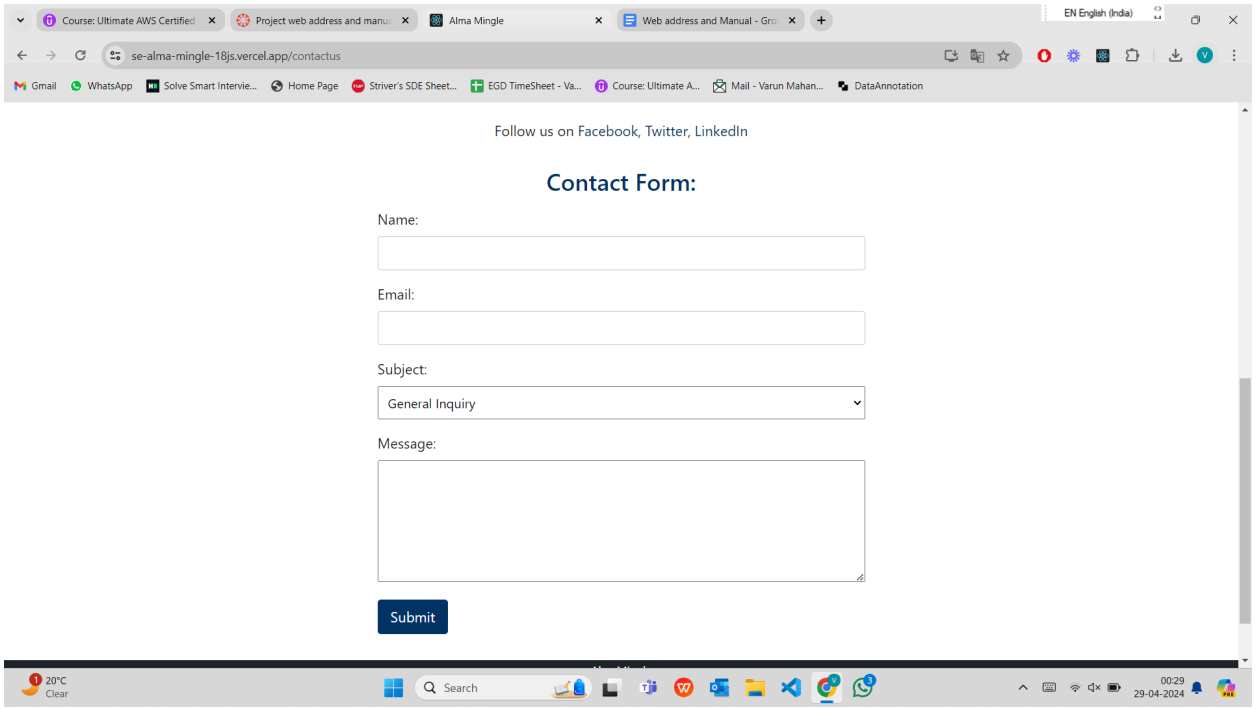


Figure D.4

The above figure D.4 shows the Contact Us page which is present to the right side of the Login navbar. The user can enter the above form for inquiries and the admin receives all these inquires and can contact to the user.

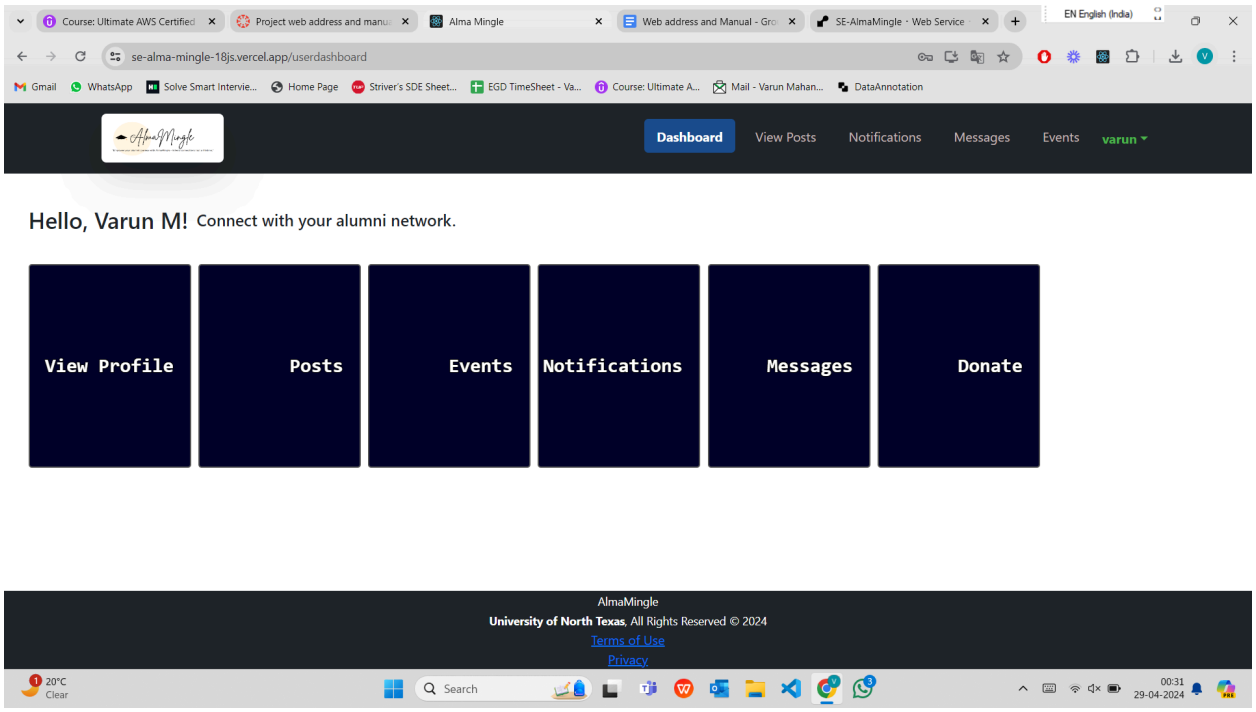


Figure D.5

The figure D.5 shows the Userdashboard after the user logins to their account. He can see all the features, add/view posts, Notifications, messages, Events, View Profile, Donation, Change Password.

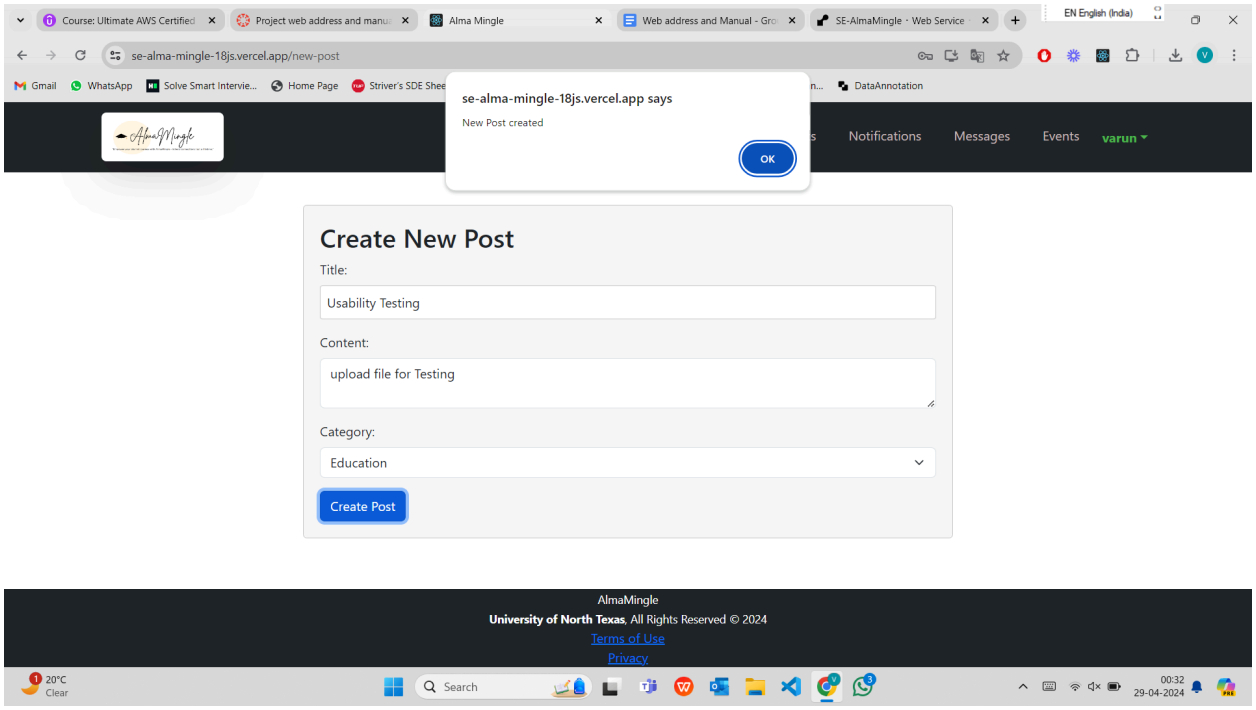


Figure D.6

After user creates a new post they get a pop up message confirming the post creation as shown in the figure D.6

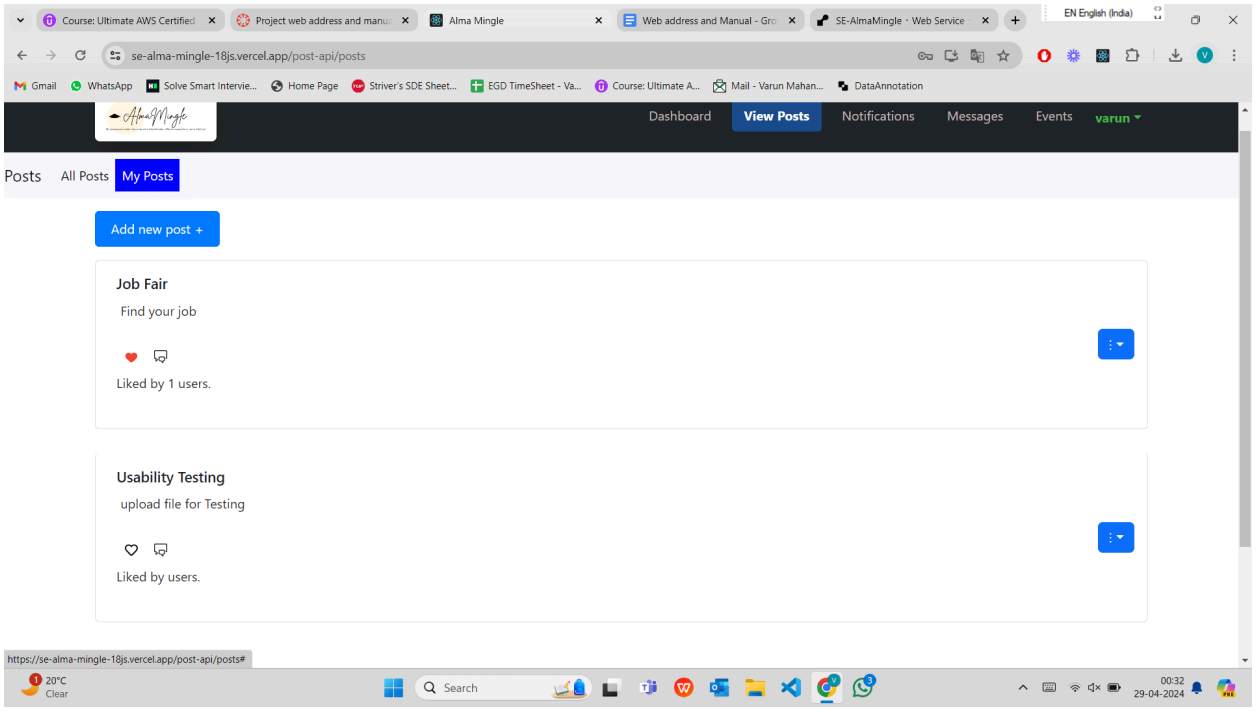


Figure D.7

In the figure 7, in the My Posts section, the user sees the posts that are created only by him. Users can edit or delete the post also if they want by clicking on the blue dropdown to the right side of the post.

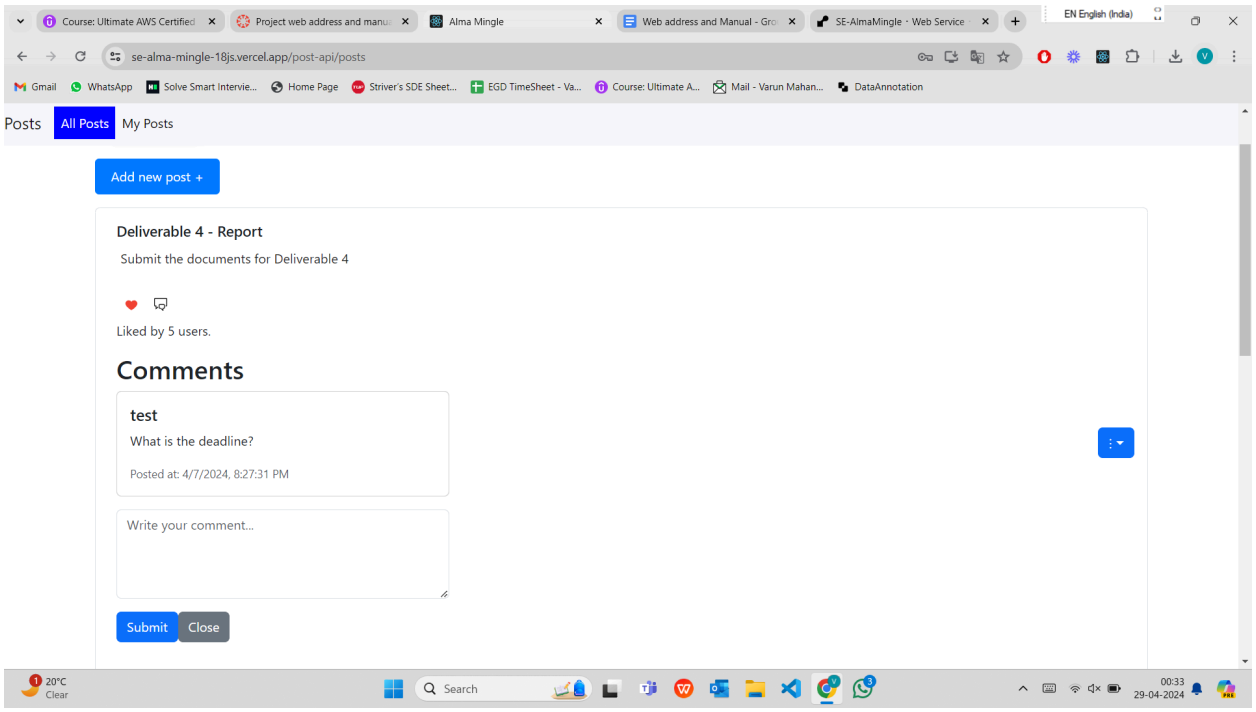


Figure D.8

In the figure 8 above, in the All Posts section all the posts that are created by other users are visible. The other users can Like the post, Comment the post and Report the post if they find it inappropriate.

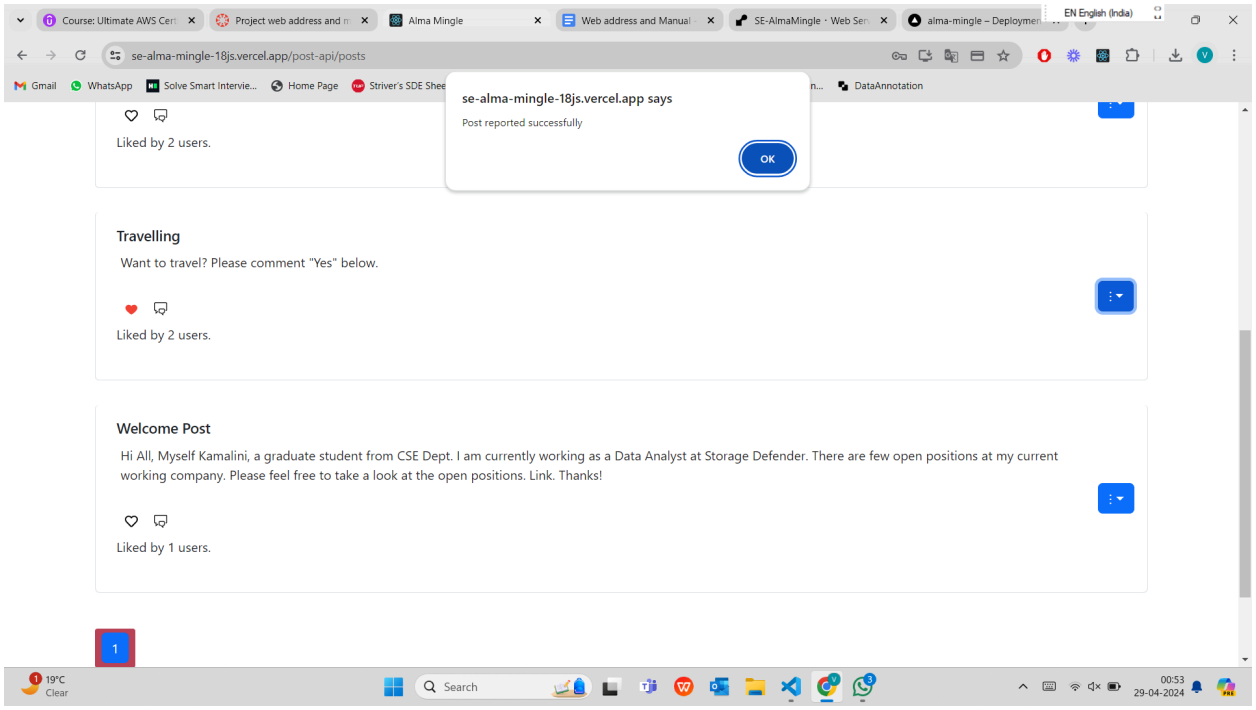


Figure D.9

In figure D.9 above shows the popup message confirming the user after he reported the post.

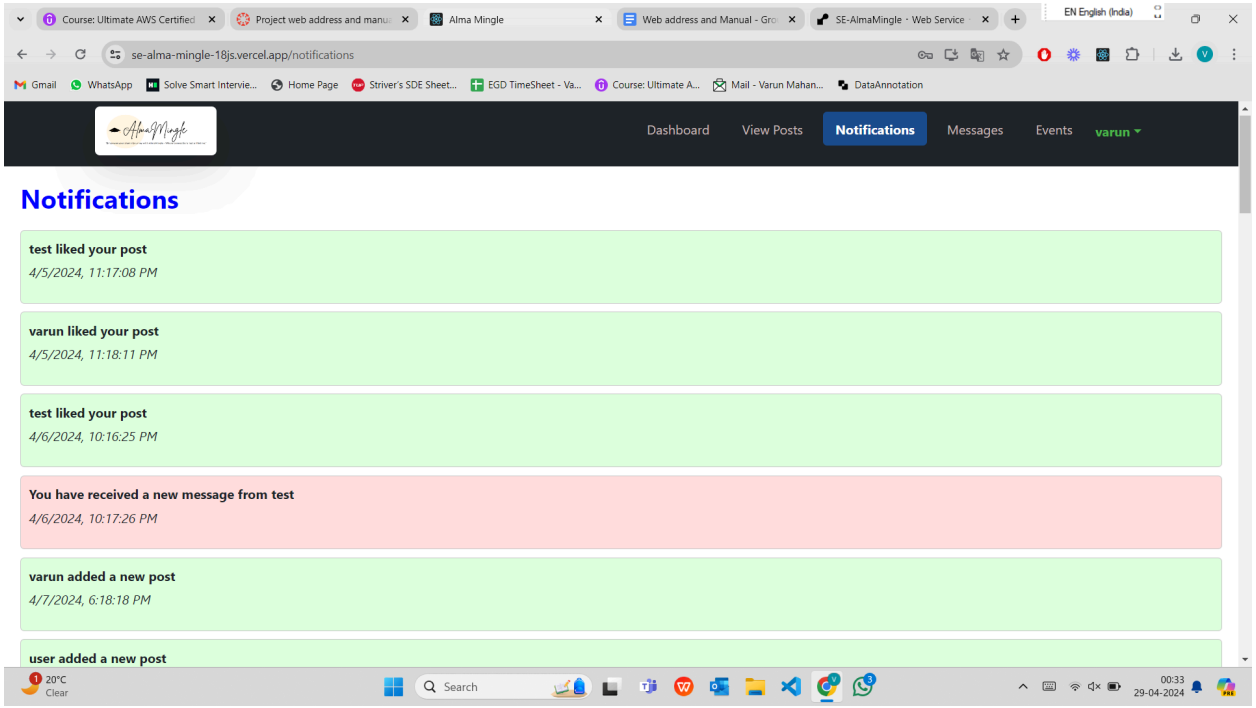


Figure D.10

Figure D.10 shows Notification page that are received to the user. Notifications like when the post is liked, when the post is commented, when the user receives a message from user, admin and when the users post is deleted by admin.

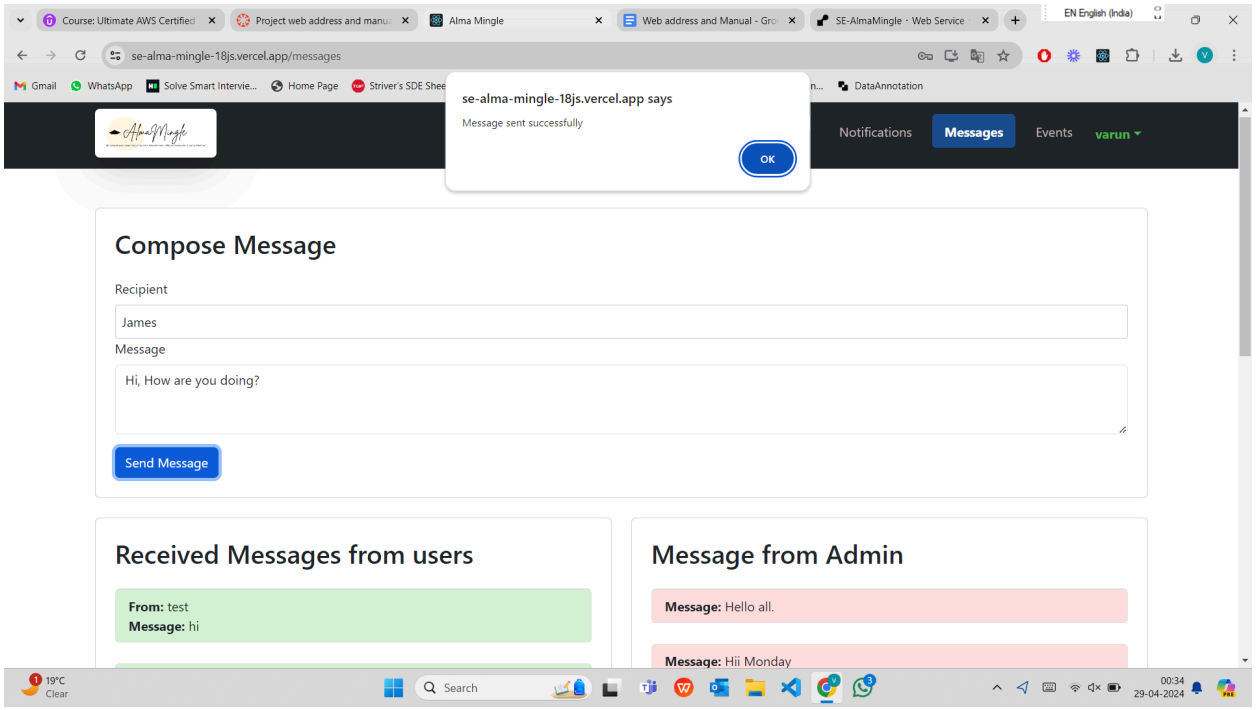


Figure D.11

Figure D.11 shows the Message page, the user can send messages to other users. The figure shows a pop up message confirming to the user regarding the message sent confirmation.

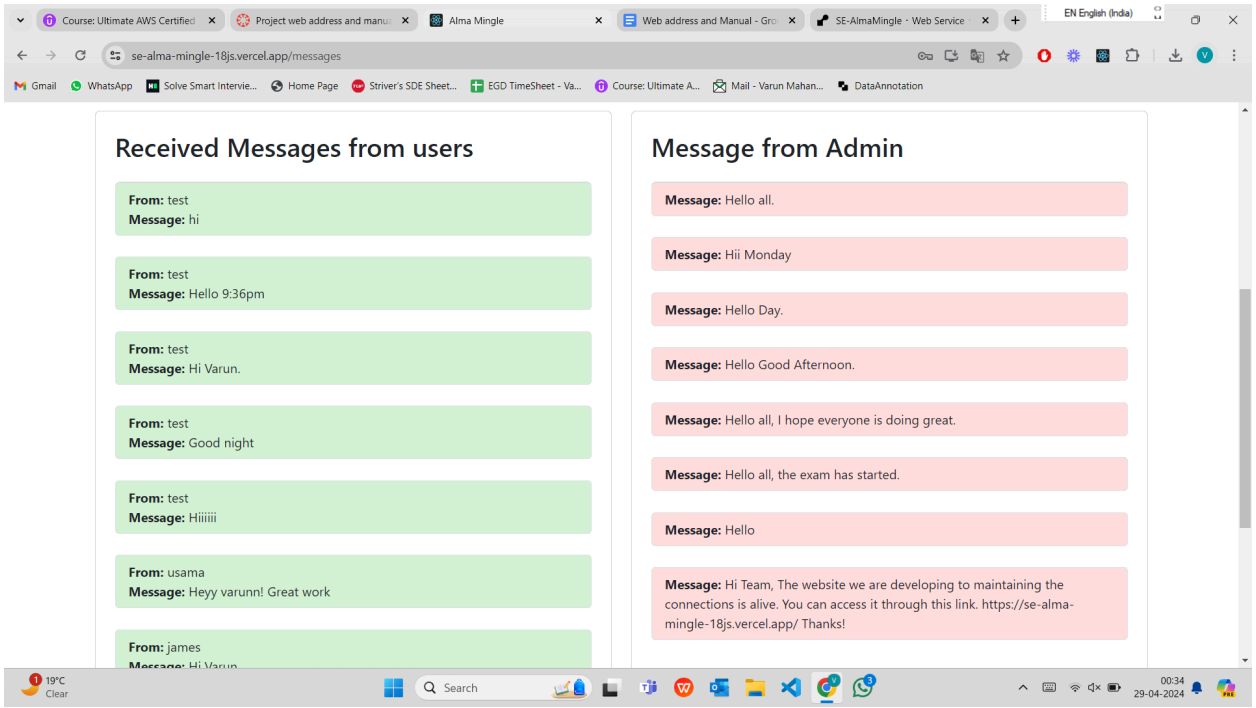


Figure D.12

The figure D.12 shows the messages received from other users to the left side and the messages received from admin to the right side.

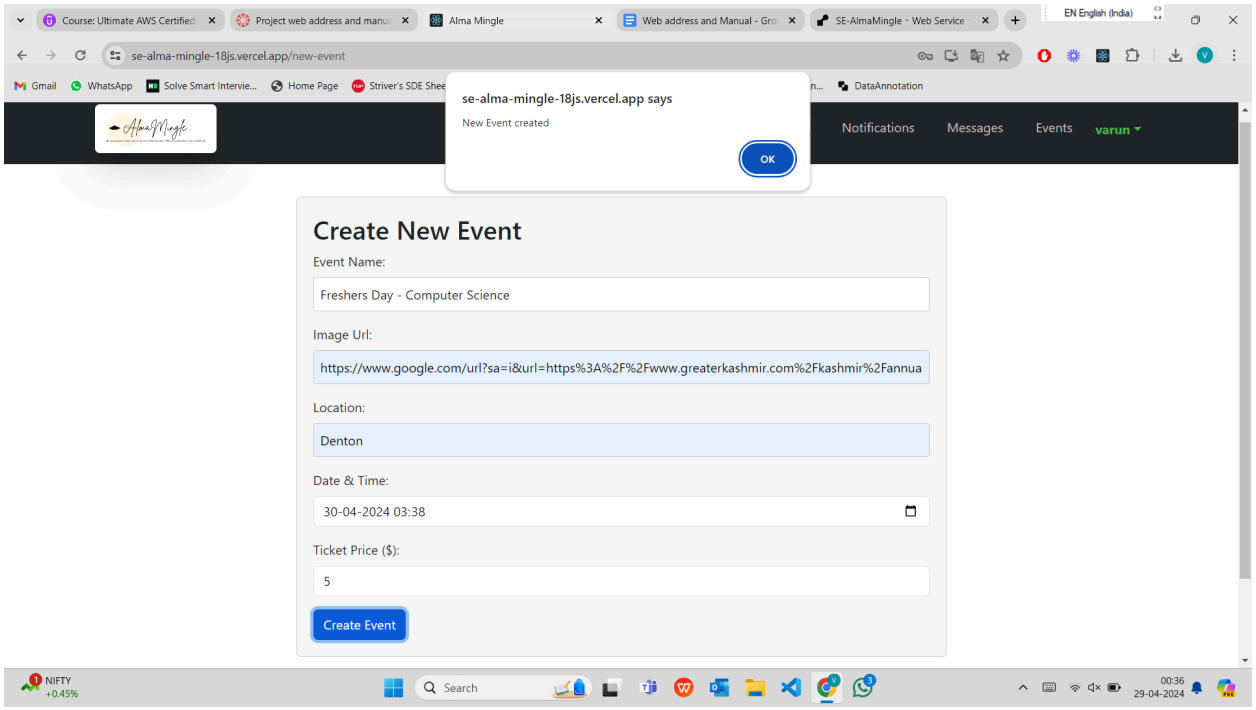


Figure D.13

Figure D.13 shows the pop up message confirming the user after creation of new event by entering the details.

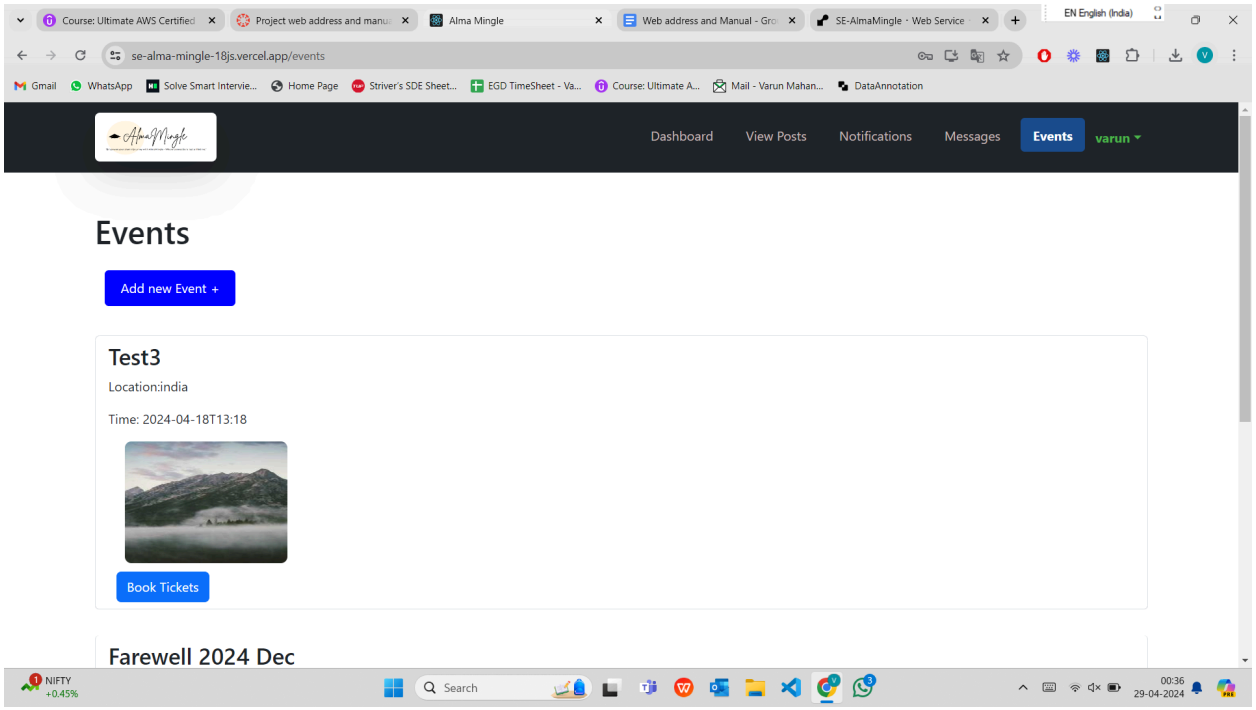


Figure D.14

Figure D.14 shows all the events created by other users. Other users can register to the event by booking tickets.

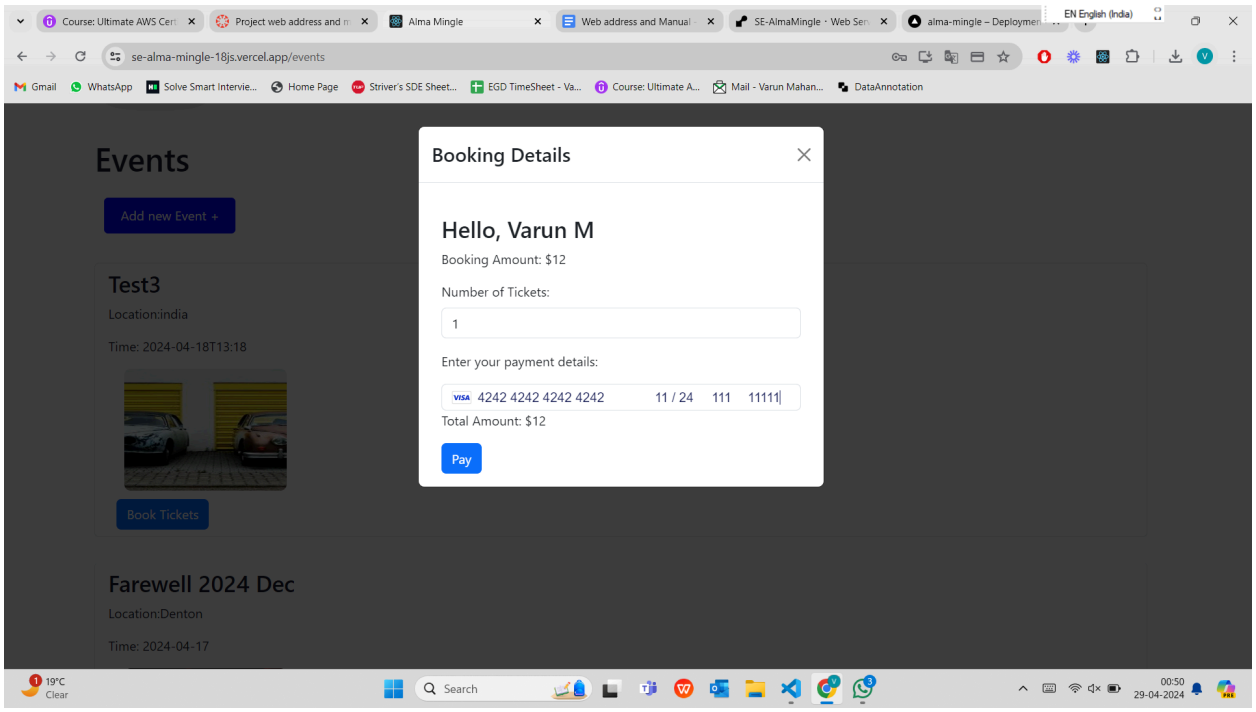


Figure D.15

Figure D.15 shows the modal when the “Book Tickets” button is clicked. The user can enter details of the card and book the number of tickets they wants.

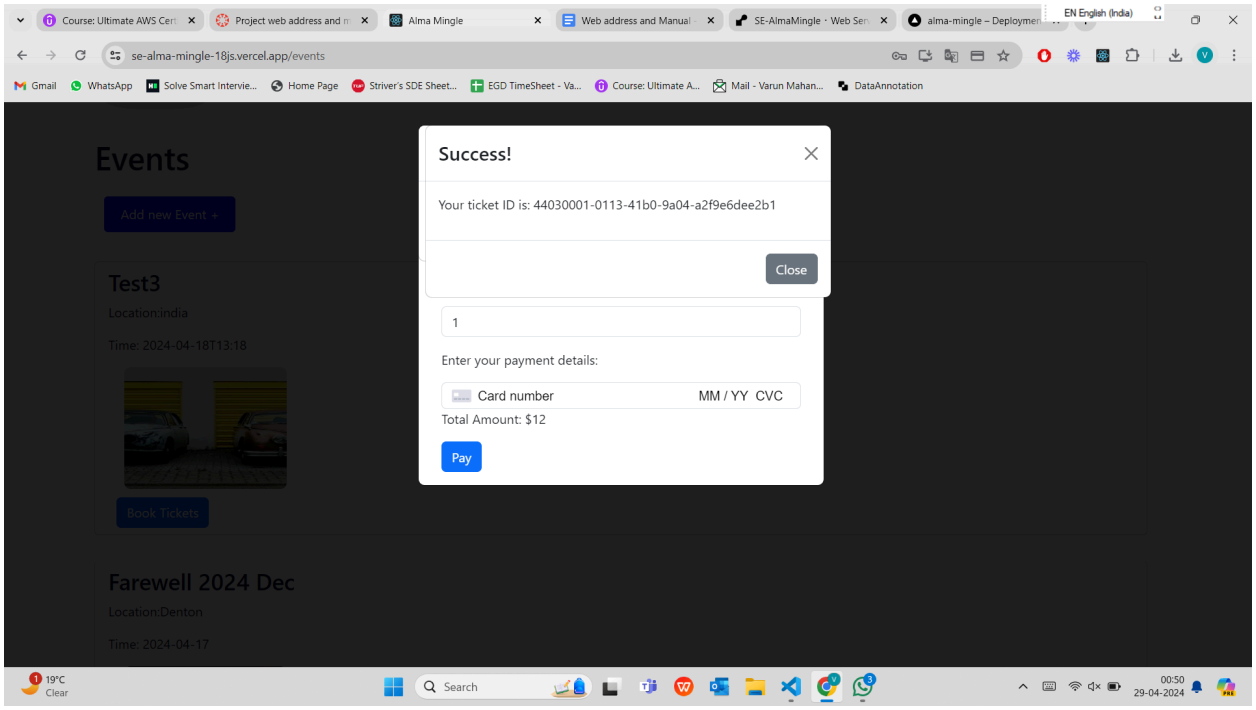


Figure D.16

Figure D.16 shows the Ticket ID sent to user after the payment is success.

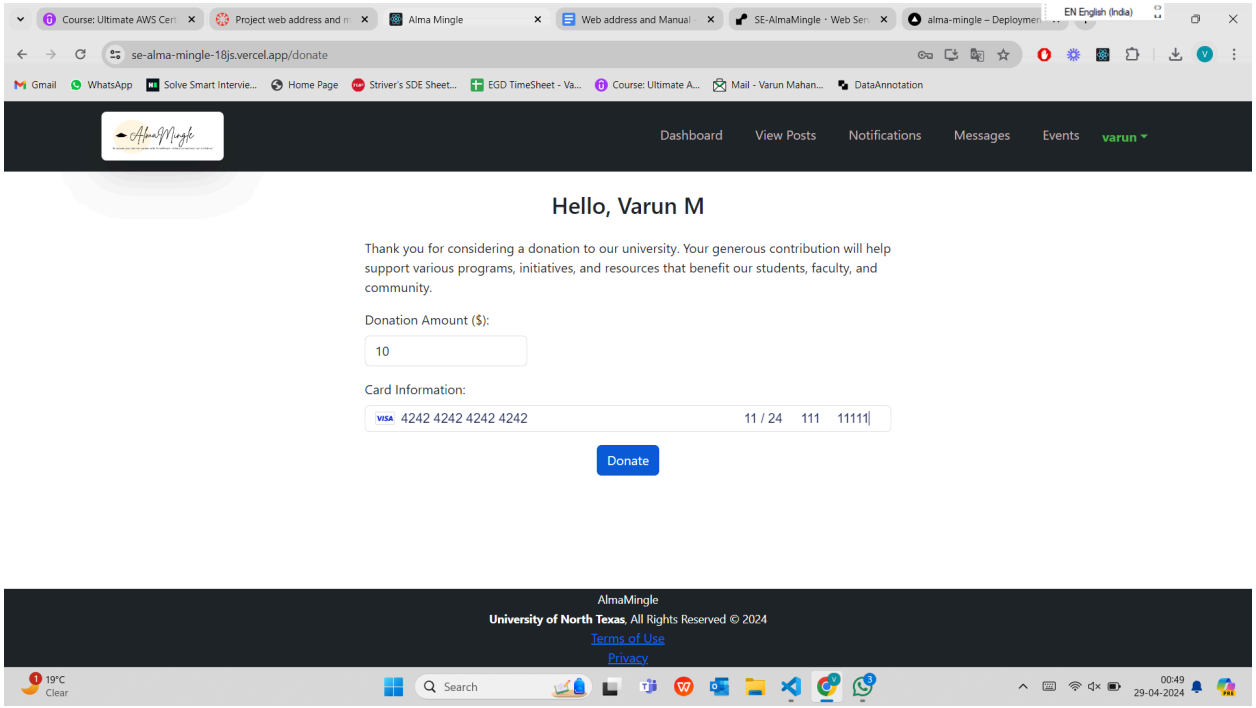


Figure D.17

Figure D.17 is the Donation page which is available on the userdashboard. The user can enter the amount they want to donate and enter the card details and click donate.

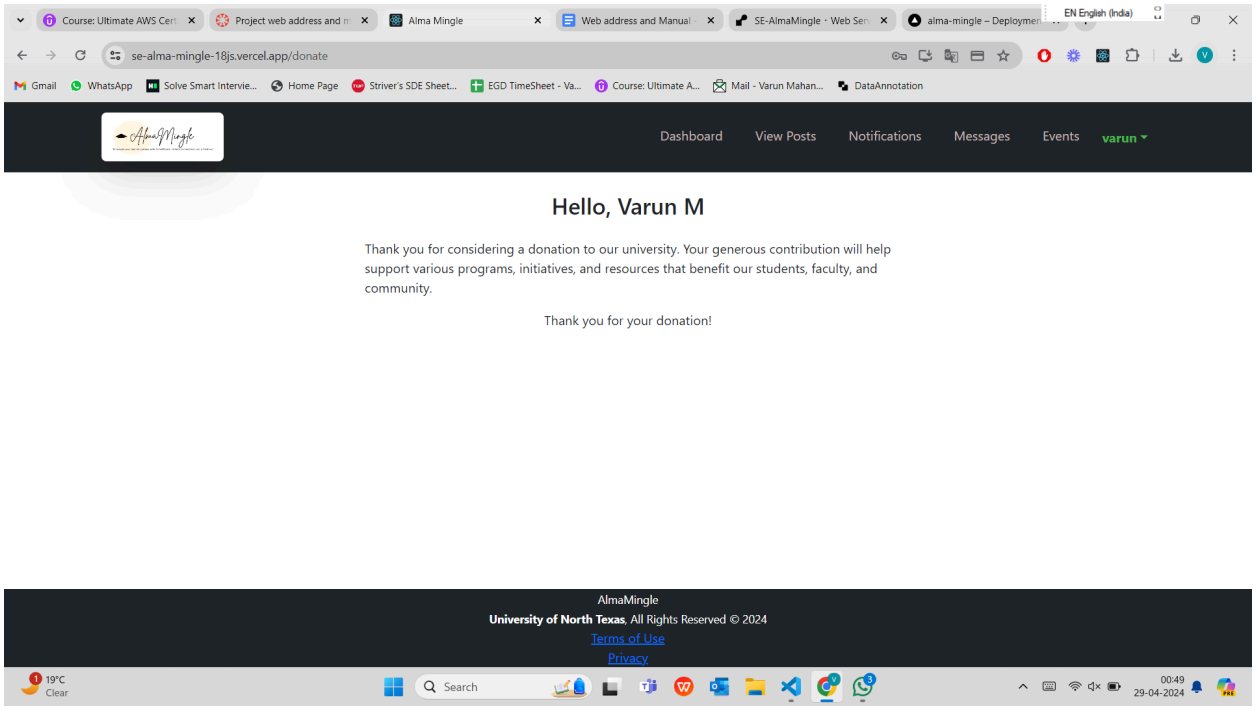


Figure D.18

Figure D.18 shows the Confirming page to the user after the donation payment is success full.

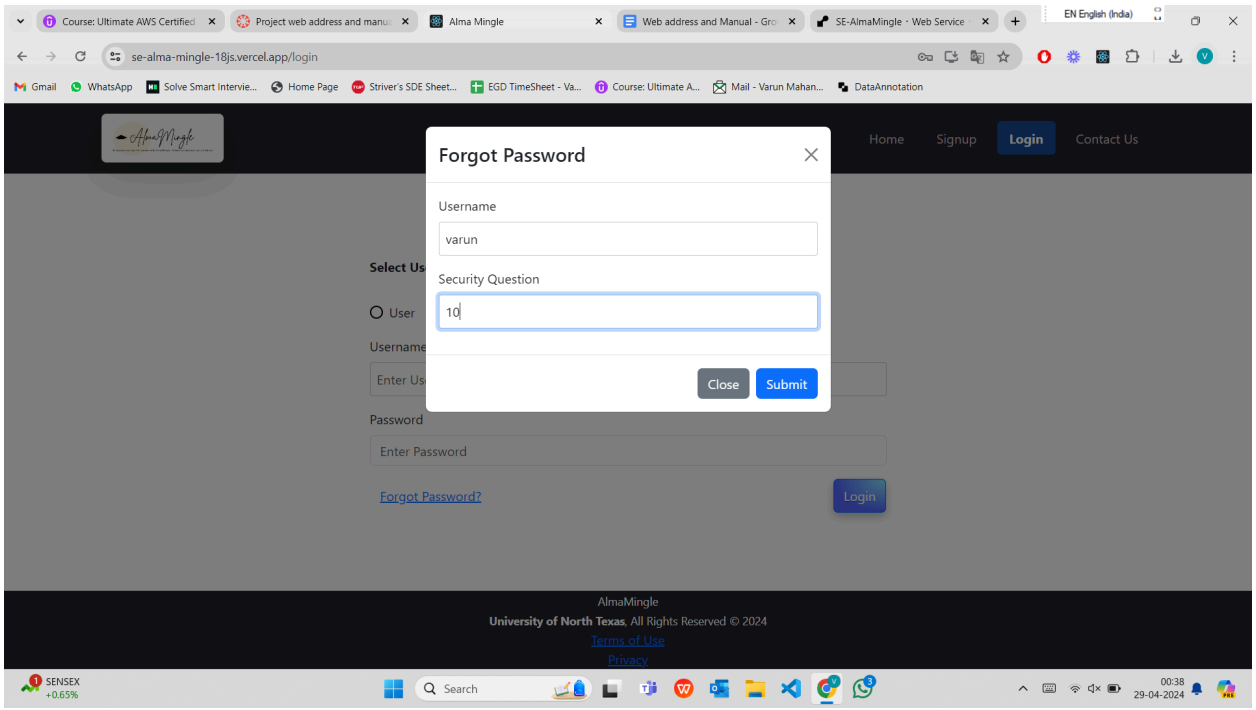


Figure D.19

Figure D.19 shows the forgot password modal which is available on the login page. If the user forgets password then he can click on forgot password and enter the questions asked. Even if the user forgets the security question, then they can contact admin on the contact us page by

filling the form. After the user enters the correct details and then clicks submit, the screen is navigated to Change Password screen as shown in the figure D.20.

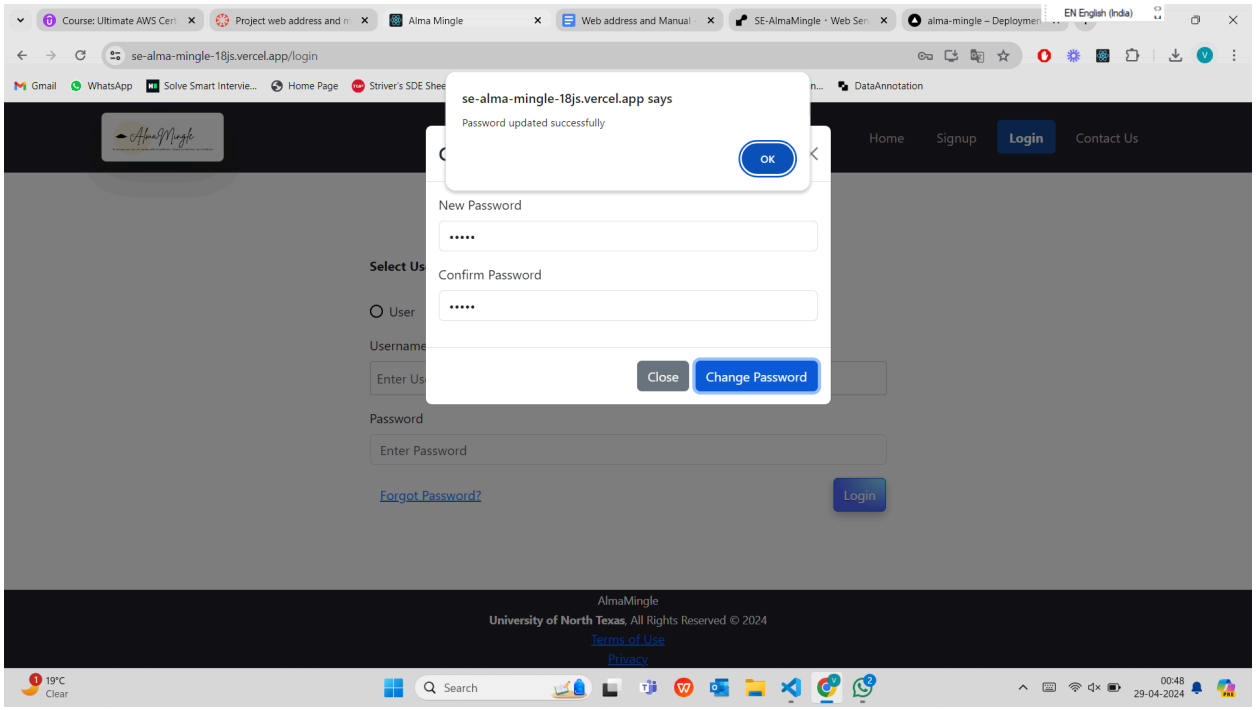


Figure D.20

In figure D.20, after user enters the new password and confirm password and clicks on change password, the pop up message confirming the Password updated successfully is appeared to the user.

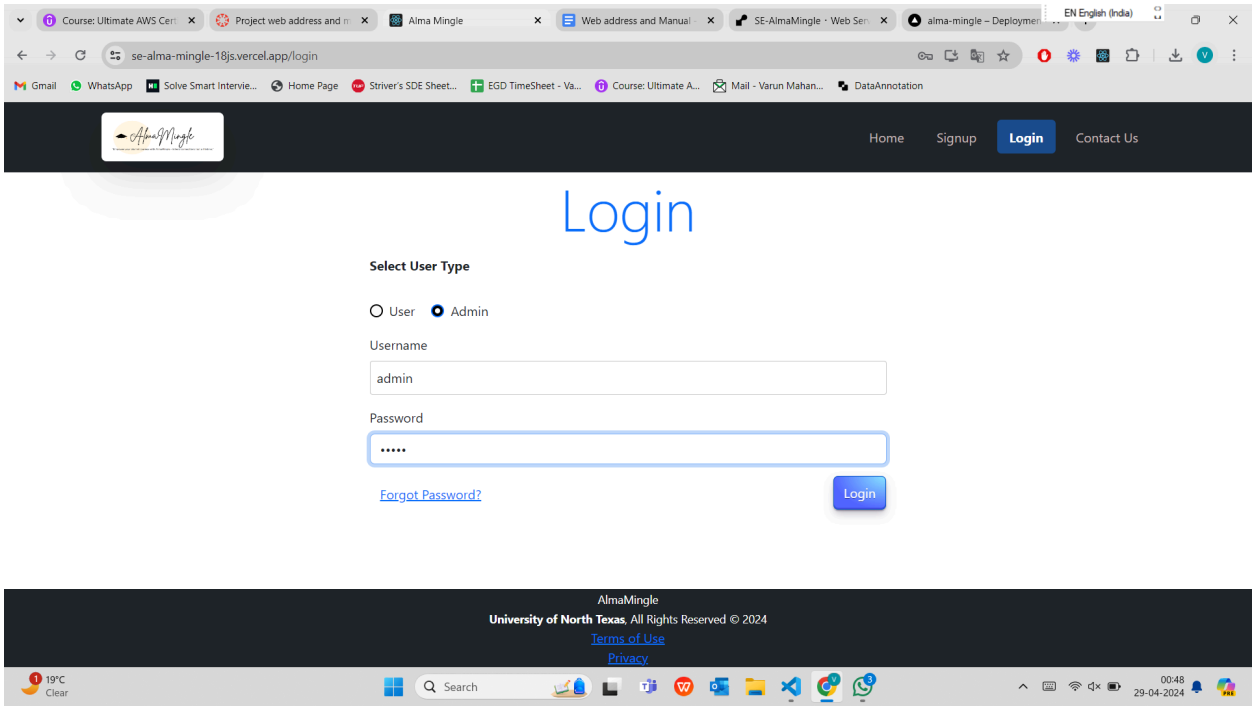


Figure D.21

Now the admin can login to the application by entering the admin details. Admin credentials are manually created in the database. One of the admins credentials are (Username : admin, Password : admin) for testing purposes.

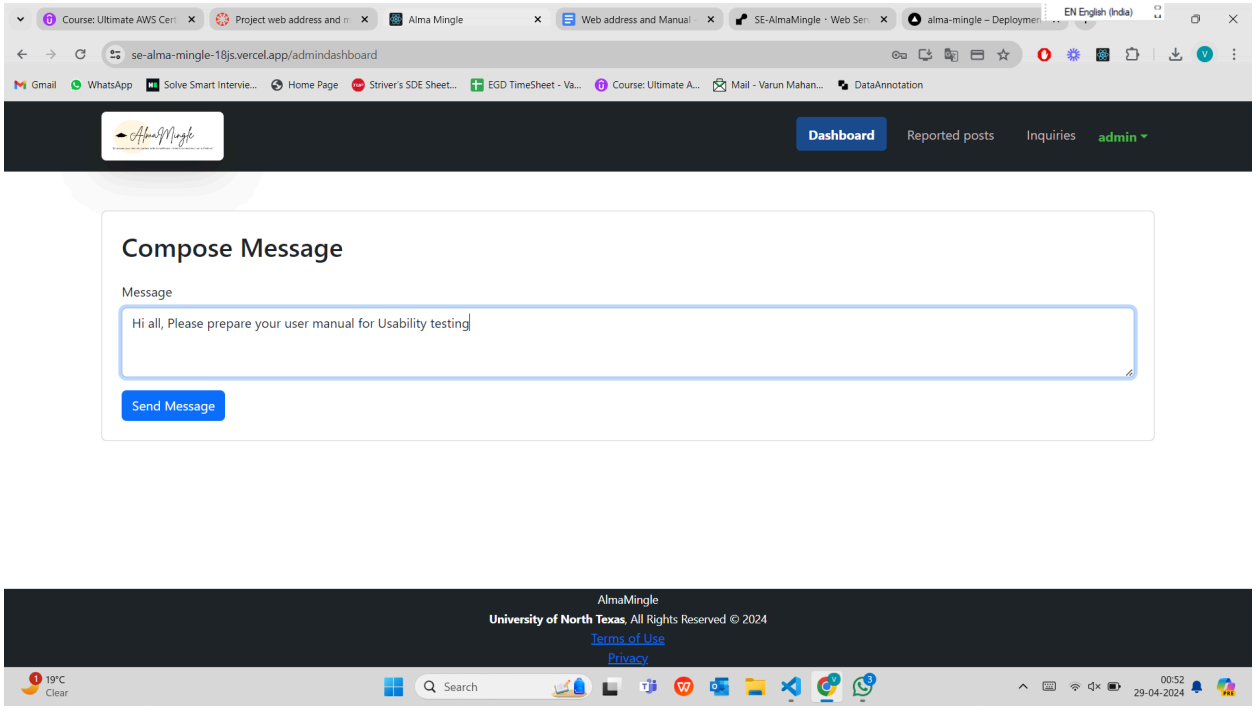


Figure D.22

Figure D.22 shows the Admin dashboard consisting of Composing messages to all the users. Seeing the reported posts and deleting them if required, Inquires.

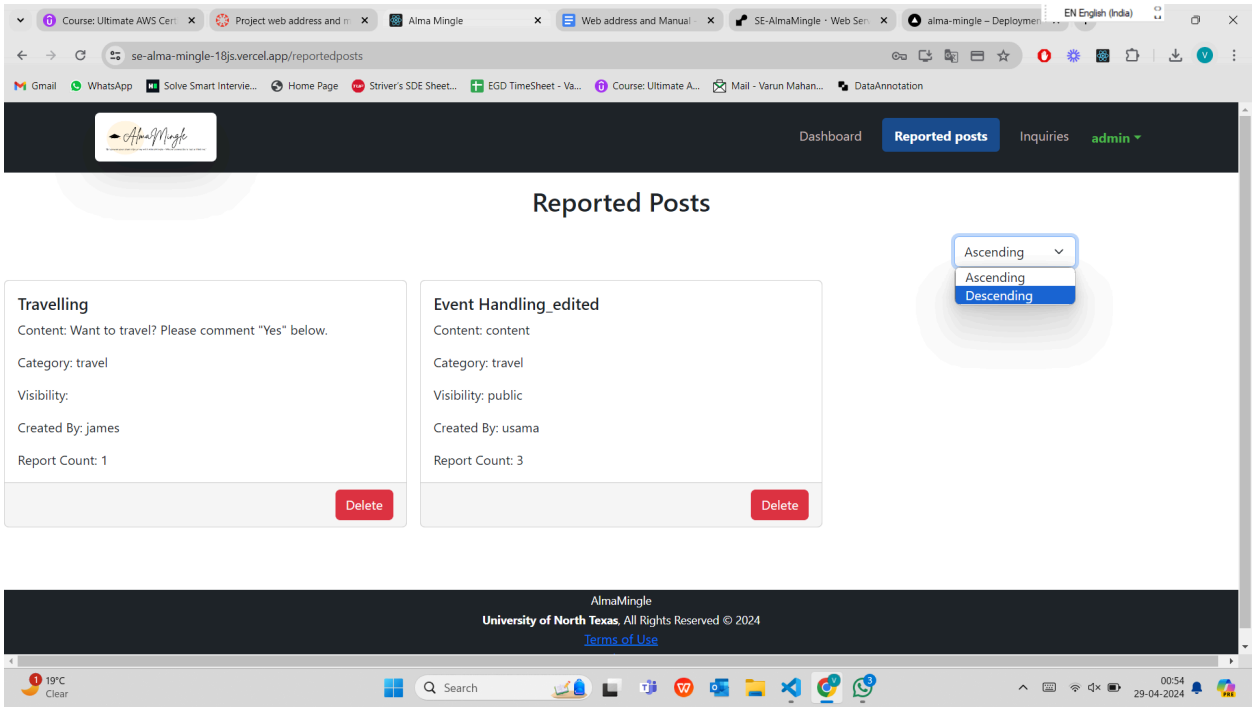


Figure D.23

Figure D.23 shows the reported posts that are reported by other users. If a particular posts is reported by 2 or more users then the Report Count is increased for that post. The posts can be ordered ascending or descending based on the report count.

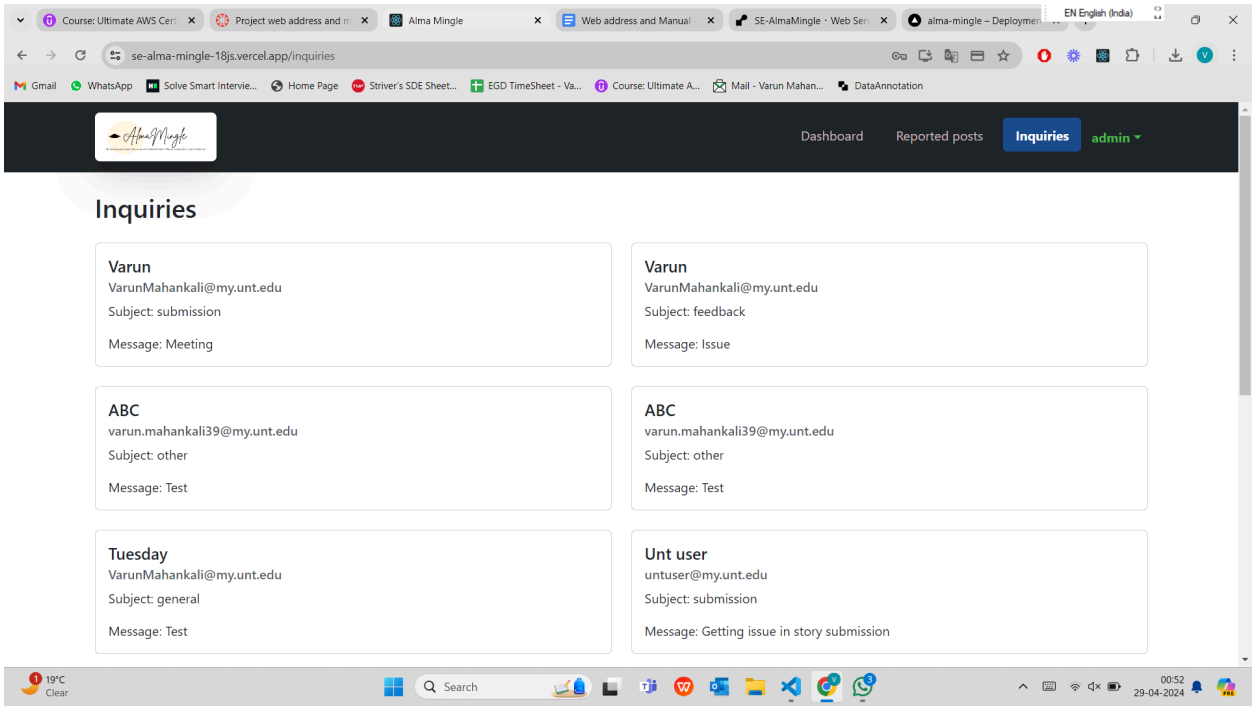


Figure D.24

Figure D.24 shows all the inquiries that are from the user. Admin can contact users through email and resolve their inquiries.

E. 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.

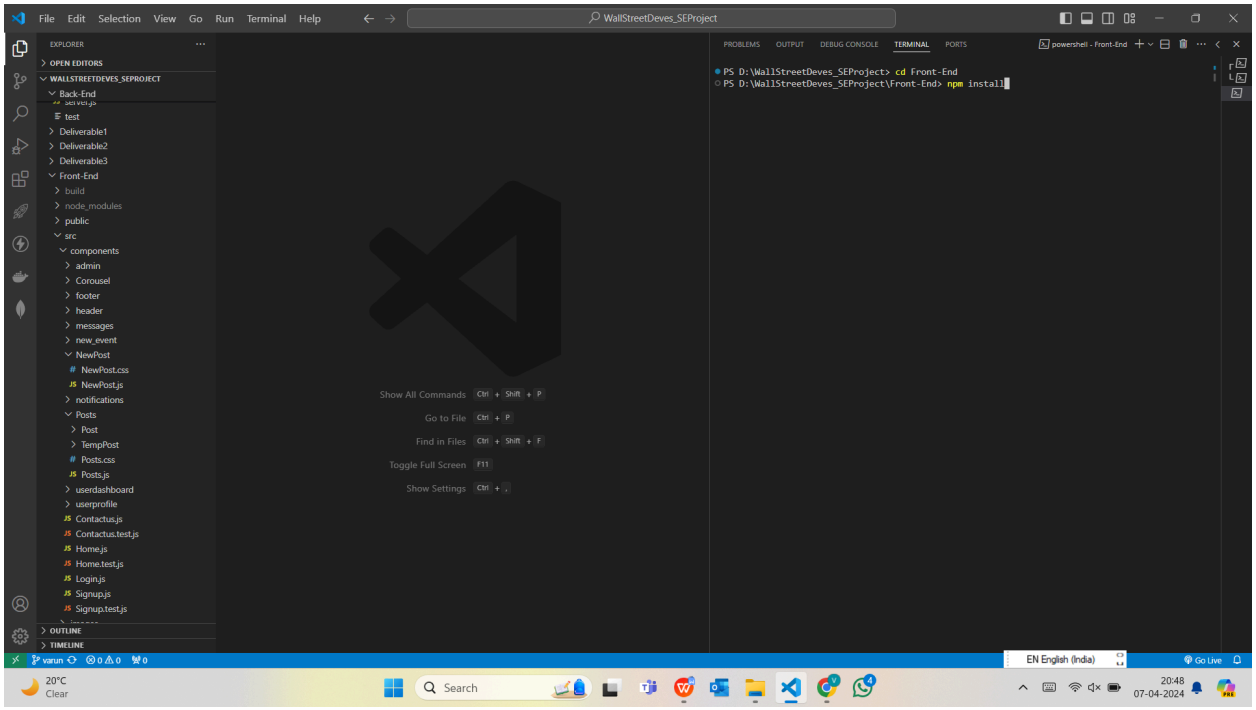


Fig E.1

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

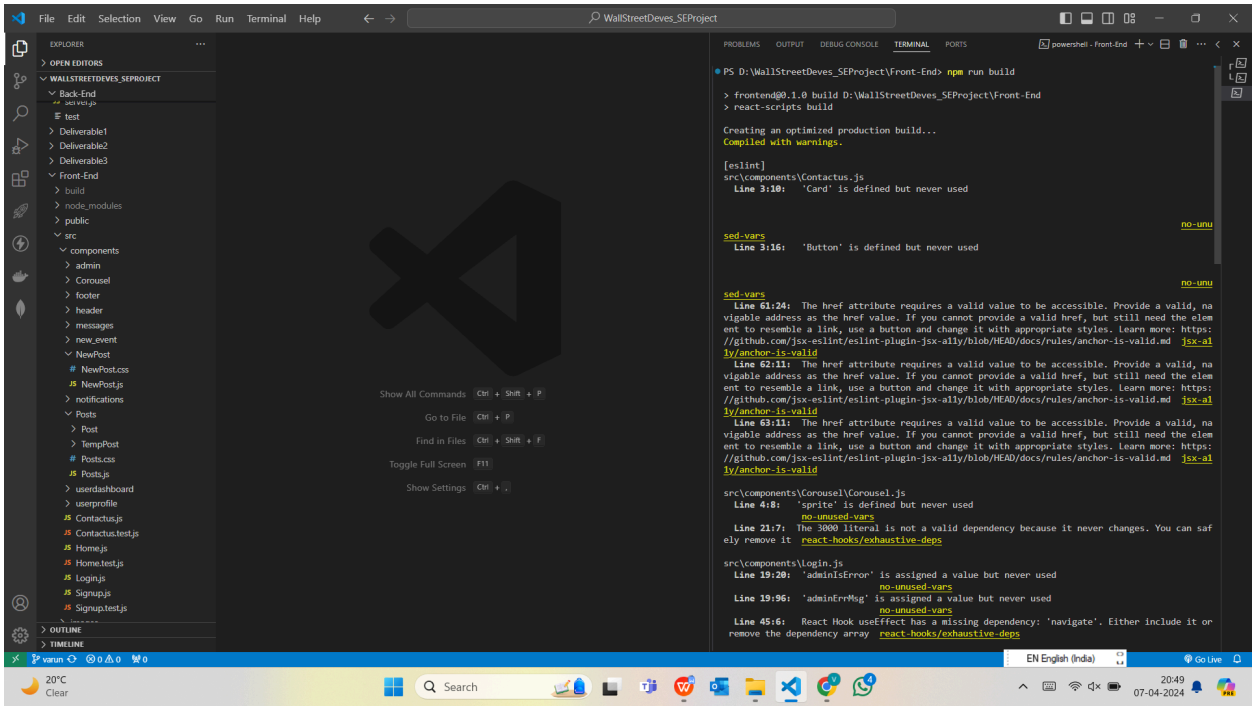


Fig E.2

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

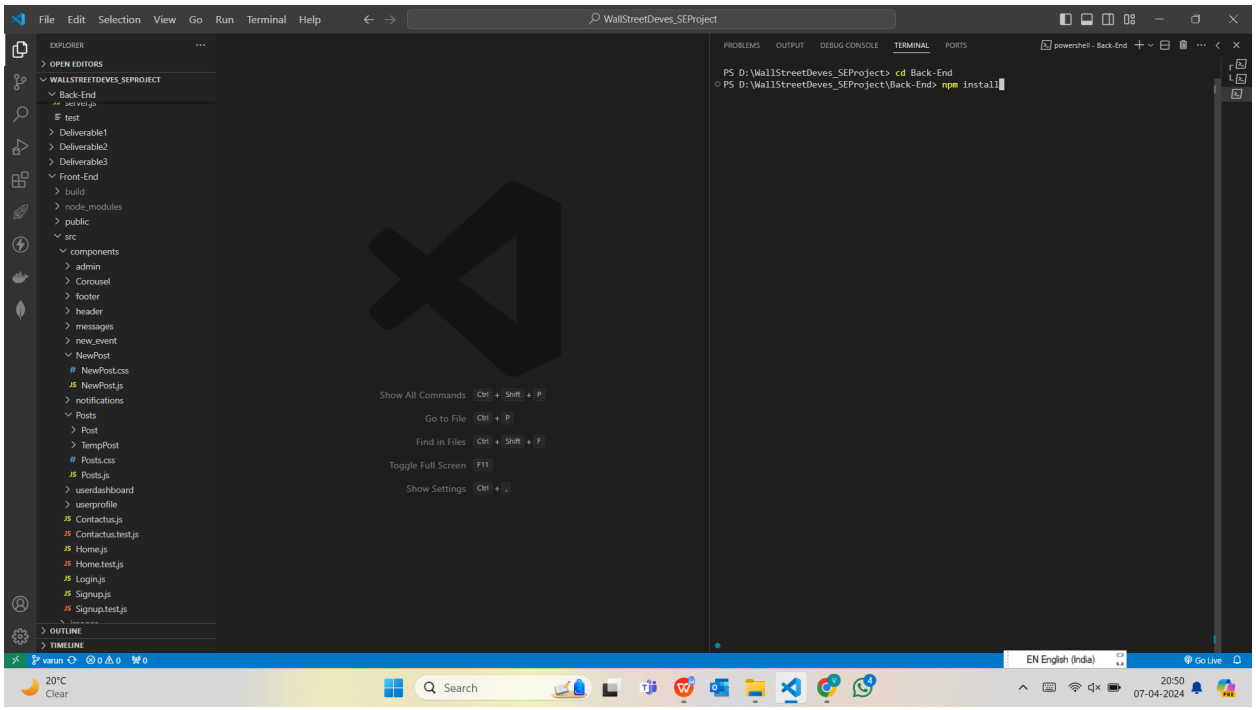


Fig E.3

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.

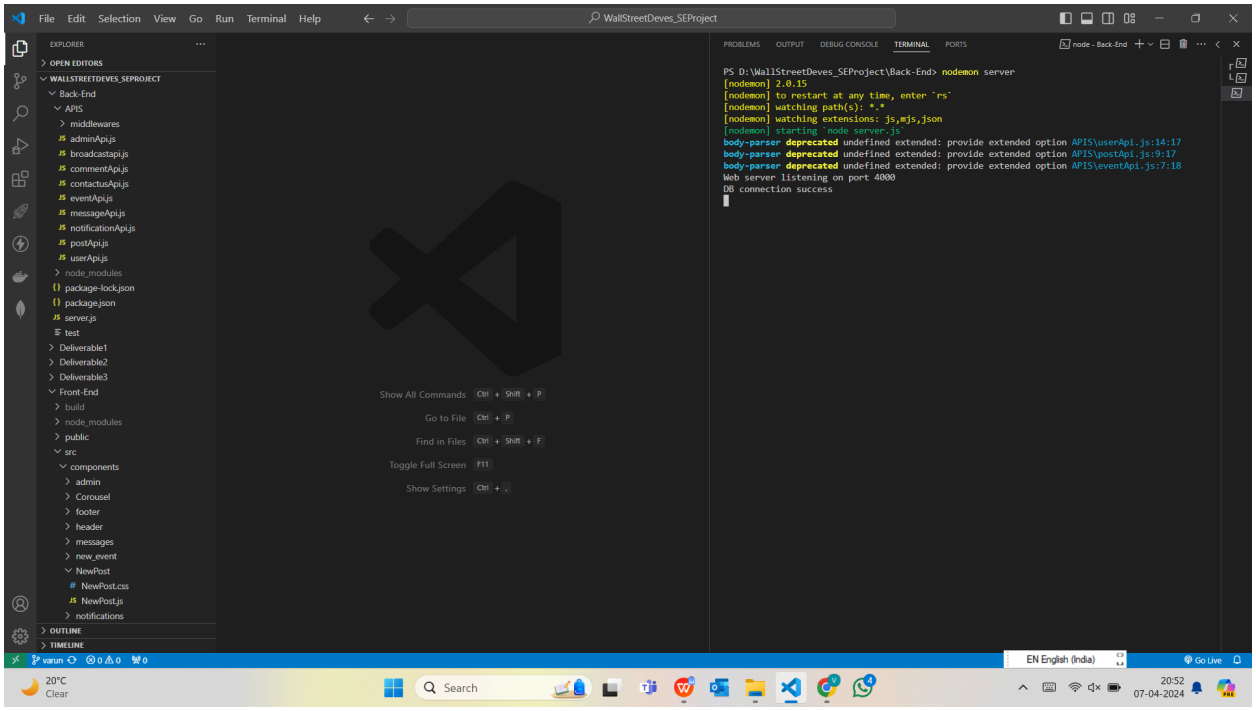


Fig E.4

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.

This website is directly accessible by using https://se-alma-mingle-18js.vercel.app/

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.

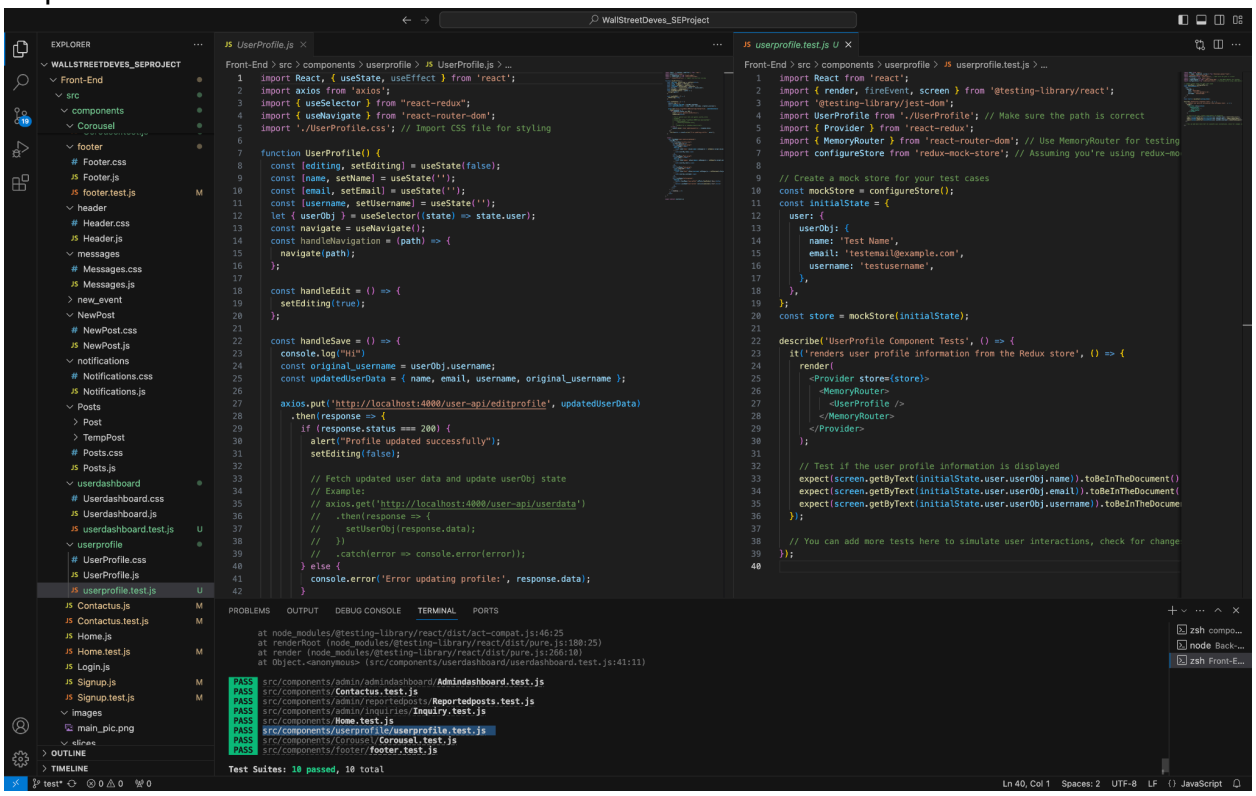


Fig E.5

F. FEEDBACK

We engaged in thorough discussions to enhance our testing framework, spurred by the feedback we incorporated more advanced features and broadened our testing scope to encompass backend components. This prompted the integration of Jest, a widely-used testing framework, to conduct rigorous assessments. The expansion aimed to fortify the resilience and reliability of all layers within our application, instilling a sense of confidence in its performance.

Furthermore, feedback from "Tiny Coders" prompted us to evaluate website accessibility through the public network space. Despite encountering a challenge in connecting to MongoDB initially, we overcame this hurdle and established an accessible link, prioritizing data integrity and security throughout the process. Rigorous data checks were implemented on the signup page, including validations for password length and email format, underscoring our commitment to robustness.

Conversations also revolved around enriching the user experience by introducing messaging and notification features. Looking ahead, if we were to continue this project into the next semester, we envision extending its deployment to other universities, leveraging the fully developed and usable platform tailored specifically for UNT.

Our journey toward developing a seamless communication and engagement platform was characterized by iterative development and user-centric refinements. We actively embraced user feedback to adapt and evolve the platform to meet ever-changing needs and expectations, shaping it into a more dynamic and effective tool.

The code inspection session proved to be an invaluable learning opportunity, guiding us towards continuous improvement and development. We are appreciative of the constructive criticism that influenced our path, using it as a catalyst to implement necessary changes and enhancements, ensuring the application's success and relevance in the long run.

G. REPORT ENDING FEATURE SUMMARY

Implemented features : All the features that were initially planned have been implemented in the project. The features include Sending messages between the users and admin, Notifications for the user, Adding/Deleting the post for the user, Like/Unlike of the post and comment on the post, report of the post, edit/delete to the post,Adding event section, feature to book tickets for the events, making the payment and also donation to out application, view/edit of the profile, changing the password inside the user dashboard, Forgot password if the user forgets it during the login time. For the admins, sending messages to all the users, deleting the inappropriate posts, contacting the users for their inquiries.
UnImplemented features : All the features are implemented except the visibility functionality for all the posts which we planned and kept as an optional.

Limitations :

- **Resource Constraints:** Limited availability of hardware resources (memory, processing power) impacting scalability.
- **Scalability Challenges:** Difficulty in handling increasing data volume or user traffic without performance degradation.
- **UX Limitations:** Design choices impacting user satisfaction and adoption rates.
- **Geographical Limitations:** Accessibility issues in remote areas with limited internet connectivity.
- **Budgetary Constraints:** Financial limitations affecting technology investment like selecting the deployment platforms. We need to make sure the platform we select for deployment will not cost anything.

Future Plans: If we were to continue this project into the next semester, we envision extending its deployment to other universities, leveraging the fully developed and usable platform tailored specifically for UNT. Along with this we can plan to add visibility to all the posts such that the posts are visible particularly to a group of people.

H. REFLECTION

Reflecting on our progress, we've accomplished the functional components outlined for the final phase and the overall project. Additionally, we successfully integrated optional functionalities such as "post comments", "post likes", forgot password options,”donation” and event ticket payment features. These enhancements significantly increased the project's overall value. During Phase 3, we maintained the effective approach from Phase 2, involving the sequential categorization of tasks, regular team meetings for updates, and weekly work reviews.

The functionalities implemented for this phase were extensively tested with defined test cases. We ensured each functionality, starting from successful and unsuccessful sign-ups, was rigorously vetted with appropriate test cases. The home button feature was tested to guarantee that the home page remained accessible to users. Similarly, the "contact us" feature and the reported posts feature were meticulously tested to ensure the platform could handle and delete posts appropriately. The admin dashboard was evaluated for its ability to send messages to users, while the user dashboard was tested for navigation and functionality, such as viewing the user profile and rendering profile information accurately. The carousel feature was examined for smooth transitions between slides, and the footer was checked to display the correct year in the privacy policy. Lastly, diagnostics were performed to ensure there were no errors across the entire platform.

We successfully added a few enhancements, which included implementing forgotten password functionality on the homepage, updating and deleting events, donation and enabling event ticket purchases. Significant UI enhancements were made, and the website code was finalized, contributing to a streamlined and user-friendly interface.

User Acceptance Testing (UAT) was initiated in Phase 3 to promptly identify and address any issues, ensuring a smooth transition to the live website. We believe that the functionalities met for this current phase and the project as a whole have created a more robust and user-friendly platform, fulfilling the evolving needs and expectations of our users.

In conclusion, the project's final phase witnessed the successful deployment of critical functionalities that meet the evolving needs of our users, marking a strong finish. Continuous improvement in our processes and testing strategies will be essential for future projects to enhance efficiency and effectiveness. To further develop our platform, we plan to expand our reach to more universities, aiming to enhance alumni connections for more students. We are confident that our project has fully served the UNT alumni community and is ready for full deployment and usage.

I. MEMBER CONTRIBUTION TABLE

S.No	Member Name	Contribution	Overall Contribution (%)
1	Kamalini Ponnuru	I focused on creating test cases for the front-end features, ensuring we covered everything. I spent time writing clear documentation for these tests, detailing what they do and what we expect from them. Then, I brought everything together into a final document. I also coordinated the team's tasks and made sure everyone knew what they were responsible for. In our meetings, we discussed project requirements and divided up the work among ourselves.	12.5%
2	Andre Sharp	Added additional test case functionality with rending and security. The applications were pinged with information to test responsiveness. Additional handling of data types were also tested.	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. Cross checked UI codes and helped coordinate all requirements that were executed in the deliverable. I also did the code inspection and reflections. Deliverable 5 document and ppt coordination.	12.5%
4	Shashank Verma	Implemented Booking Events functionality using stripe both in frontend and backend. Implemented a Donation page with the same stripe technology. Added the functionality of showing the edit and delete button to the user that created that event. Added env variable functionality in both frontend and backend.	12.5%
5	Usama Bin Faheem	Tried implementing the event update and event delete functionalities. Worked on the requirements documentation for the report.	12.5%

6	Suhaibuddin Ahmed	Contributed to the deliverable 5 draft. Handled phase 2 and requested feature documentation. Tested the website post-deployment, reviewed the deliverable and the requirements.	12.5%
7	Varun Mahankali	Completed the rest of the functionalities, Forgot Password, Event Edit, Bug Fixes. Finalized the code by making additional changes in the UI. Finished the document for Web address and Manual Submission. Added the user manual in Deliverable also. Researched and deployed the application. Discussed the requirements with the team and helped the teammates in completion of the tasks.	12.5%
8	Rahman Mehmood	Responsible for the creation of UML Diagrams, Tested the website's new features, Helped in delegating responsibilities for the deliverable and presentation, Reviewed the deliverable document.	12.5%