Chat Blog Social Media WebApp

A MINI PROJECT REPORT

[CASE STUDY]

Submitted by

NIPUN DHIMAN [RegNo: RA2211003010626]

SHIVANSH SHARMA [RegNo: RA2211003010627]

Under the Guidance of

DR. RAJALAKSHMI M

Assistant Professor, Department of Computing Technologies

DEPARTMENT OF COMPUTING TECHNOLOGIES

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

NOV 2024

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that **21CSE354T – FULL STACK WEB DEVELOPMENT - Mini project report** titled **Chat Blog Social Media App** is the bonafide work of **Nipun Dhiman [RA2211003010626]** and **Shivansh Sharma [RA2211003010627]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation.

[illegible] 6/11/24

SIGNATURE

DR. RAJALAKSHMI M

COURSE FACULTY

Assistant Professor

Department of Computing Technologies

[illegible]

Department of Computing Technologies Kattankulathur 603 203 SRM IST

SIGNATURE

Dr G. NIRANJANA

HEAD OF THE DEPARTMENT

Department of Computing Technologies

DEPARTMENT OF COMPUTING TECHNOLOGIES
SCHOOL OF COMPUTING
College of Engineering and Technology
SRM Institute of Science and Technology

MINI PROJECT REPORT

ODD Semester, 2024-2025

Subject code & \Sub Name	:	21CSE354T & Full stack Web Development
Year & Semester	:	III & V
Project Title	:	Chat Blog Social Media WebApp
Course Faculty Name	**:**	Dr. Rajalakshmi M
Team Members	:	Nipun Dhiman & Shivansh Sharma

Particulars	**Max. Marks**	**Marks Obtained**
FRONT END DEVELOPMENT	2.5	
BACK END DEVELOPMENT	2.5	
IMPLEMENTATION	3	
REPORT	**2**	

Date :

Staff Name :

Signature :

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to **Dean-CET**, SRM Institute of Science and Technology, Dr **T.V. Gopal**, for his invaluable support.

We wish to thank **Dr Revathi Venkataraman**, **Professor and Chairperson**, School of Computing and **Dr M. Pushpalatha, Associate chairperson,** Department of Computing Technologies, SRM Institute of Science and Technology for their invaluable support.

We are grateful to our **Head of the Department**, **Dr G. Niranjana, Professor**, Dept of CTECH, SRMIST, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our **Audit Professor**, **Dr R. K Pongiannan, Professor,** Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

Our inexpressible respect and thanks to our course faculty, **Dr. Rajalakshmi M**, Assistant Professor Department of CTECH, SRMIST, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

NIPUN DHIMAN [Reg. No:RA2211003010626]

SHIVANSH SHARMA [Reg. No: RA2211003010627]

TABLE OF CONTENTS

ABSTRACT

The Chat Blog social media WebApp is a full-stack social media platform designed to enhance communication and content sharing among users. It combines a robust backend with an interactive frontend to provide a seamless experience for social interactions. The application enables essential functionalities such as creating, retrieving, updating, and deleting posts, along with user authentication, profile management, and real-time chat. Each post can include text, media files, and comments, fostering a dynamic environment for user engagement. A key feature of Chat Blog is its WebSocket-based chat system, enabling real-time messaging to enhance social connectivity. The backend, powered by a Spring Boot REST API, handles data management, user authentication, and authorization, ensuring secure communication between frontend and backend components. The frontend, built using ReactJS, offers a user-friendly interface that simplifies navigation through posts, profiles, and chat features. Additionally, the application implements robust security measures, including JWT-based authentication, to protect user data. Chat Blog exemplifies a comprehensive social media solution with practical applications in networking, community building, and content sharing, demonstrating the integration of front-end and back-end technologies for a cohesive, secure, and engaging user experience.

1. INTRODUCTION

The Chat Blog social media WebApp is a full-stack social media platform designed to facilitate seamless interaction and content sharing among users through foundational functionalities such as creating, retrieving, updating, and deleting posts. Built using ReactJS, Spring Boot, and MySQL, Chat Blog exemplifies the integration of modern web technologies to deliver a dynamic, interactive user experience. Each user post includes content, media attachments, and comments, while profile management and authentication features enrich the social engagement. By implementing core features like user authentication, real-time chat, and post interactions, Chat Blog offers a versatile solution for social networking and community building. The subsequent sections delve into the application's key components and technological foundation:

1.1 Real-Time Messaging and Social Interactions:

One of the key highlights of Chat Blog is its real-time chat feature, which utilizes WebSocket technology to enable instant communication between users. This feature allows users to engage in private messaging, group chats, and live updates, creating a dynamic social environment. The backend, powered by Spring Boot, handles real-time data exchange and ensures low-latency interactions, while the frontend, built with ReactJS, delivers a responsive and intuitive interface for smooth communication. The chat system enhances the platform's interactivity, promoting active user engagement and fostering a vibrant online community.

1.2 User Authentication and Secure Access:

Chat Blog prioritizes user security through a robust authentication system. The application uses Spring Security and JWT (JSON Web Tokens) for secure user login and session management. This ensures that only authenticated users can access personalized content and features, safeguarding sensitive user data. The backend handles user authentication, registration, and authorization processes efficiently, providing a secure and scalable environment for social networking. This secure framework not only enhances data protection but also builds user trust, making the platform a reliable space for online interaction.

1.3 Responsive Full-Stack Architecture:

The frontend of Chat Blog is built using ReactJS, featuring React Router for seamless navigation in a single-page application (SPA) format. The user interface is designed with Bootstrap, HTML, and CSS, ensuring responsiveness across different devices and screen sizes. The backend, powered by Spring Boot, supports efficient data management and API handling, while MySQL serves as the reliable database for storing user information, posts, and messages. Axios is employed for smooth communication between the frontend and backend, ensuring efficient data transfer and synchronization. This full-stack approach delivers a cohesive user experience, enabling scalability and adaptability for future enhancements.

In summary, the Chat Blog social media WebApp is a modern social networking platform that combines secure user management, real-time communication, and responsive design. It stands out by offering a blend of social features that promote user engagement and community building. With its scalable architecture and user-centric design, Chat Blog serves as a robust template for developing interactive social media applications.

2. PROJECT OVERVIEW AND OBJECTIVES

Chat Blog Social Media WebApp is an innovative full-stack project designed to provide users with a comprehensive social media experience. Developed using ReactJS for an engaging frontend, Spring Boot for robust backend services, and MySQL for efficient data management, Chat Blog is built to enhance social interactions through features like real-time messaging, post sharing, and user authentication. This application leverages best practices in full-stack development, offering a scalable and interactive solution for social networking and community building.

2.1 Objectives:

2.1.1 Facilitate Real-Time Communication and Social Interactions

The primary objective of the ChatBlog Social Media WebApp is to provide users with a seamless platform for managing social media profiles and fostering real-time communication. Each user profile and post is structured with unique attributes such as user ID, username, email, content, timestamps, and interaction metrics, ensuring organized and efficient data management. The backend of the application, built using Spring Boot and Spring MVC, exposes a RESTful API that enables secure and efficient handling of user accounts, posts, and social interactions.

2.1.2 Demonstrate a Scalable Full-Stack Architecture

A critical objective of Chat Blog is to prioritize user security and data protection through a robust authentication system. Utilizing Spring Security and JWT (JSON Web Tokens), the application provides secure login, registration, and session management. This ensures that user data is protected and accessible only to authorized individuals. The secure authentication framework not only enhances data privacy but also builds trust, encouraging users to actively participate in the platform. By implementing best practices in security, Chat Blog demonstrates how social media applications can maintain user safety while offering rich interactive features.

Through Chat Blog, users and developers can explore the integration of real-time communication, secure user management, and responsive design in a social media context. The project serves as both a learning tool and a practical template for developing similar full-stack applications that prioritize user engagement, security, and scalability.

2.1.3 Demonstrate an Intuitive Full-Stack Architecture

The ChatBlog Social Media WebApp exemplifies a well-integrated full-stack architecture, showcasing best practices in modern web development. The frontend is developed using ReactJS along with React Router v6, creating a single-page application (SPA) that delivers a smooth and responsive user experience. The use of Bootstrap 5, combined with HTML, CSS, and Sass, ensures a visually appealing and adaptive interface that looks consistent across various devices.

On the backend, Spring Boot and Spring MVC are employed to handle the business logic, with MySQL as the database for robust data management. The backend API enables efficient communication between the client and server, leveraging Axios for secure HTTP requests. This structure allows for smooth interactions between the frontend and backend, ensuring that data related to user profiles, posts, and social interactions is managed effectively and securely.

The application's architecture not only focuses on seamless integration of technologies but also prioritizes security and scalability. Features like role-based access control and validation mechanisms help secure the platform while promoting an optimal user experience.

By utilizing a full-stack approach, Chat Blog serves as an effective model for building modern social media applications, demonstrating how to cohesively connect frontend and backend technologies. This project provides valuable insights into developing scalable, efficient, and user-centric web applications, making it a practical template for developers working on similar full-stack projects.

3. ARCHITECTURE DIAGRAM AND TECHNOLOGIES USED

The architecture of the Chat Blog social media WebApp is designed to deliver a seamless and engaging user experience by integrating frontend and backend technologies in a cohesive manner. This section outlines the architecture diagram and describes the technologies utilized at each layer of the application to ensure efficient social interactions, data management, and real-time communication.

3.1 Architecture Diagram

The architecture diagram of Chat Blog showcases the interaction flow between client-side and server-side components, illustrating how users engage with the platform. The architecture is structured as follows:

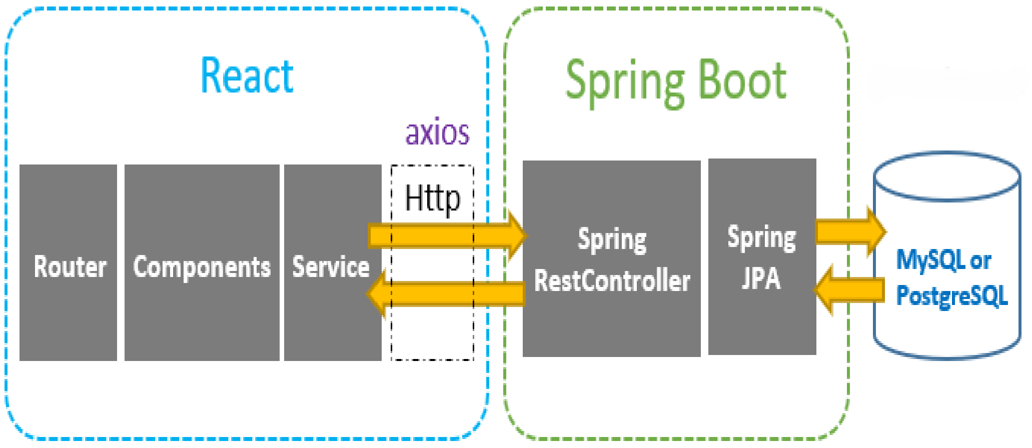


Fig 3.1 Architecture Diagram

3.1.1 Client-Side (Frontend):

The client-side interface of Chat Blog, developed with HTML, CSS, JavaScript, Node.js, and SASS, provides an interactive user experience for managing social posts, profiles, and chats. Users can create new posts by filling out forms that capture essential fields like content, images, and hashtags, ensuring that all required information is submitted for engaging social interactions. Existing posts are displayed in an organized feed, allowing users to view content, like, comment, and share posts with ease, along with quick access to edit or delete their own posts. The search functionality further enhances usability by enabling users to filter posts based on keywords, titles, or hashtags, which helps in quickly finding specific content within a large dataset.

The frontend communicates with the backend using the fetch API, which handles HTTP requests that follow RESTful principles. Each user action—such as creating a new post or searching for users—triggers an HTTP request, prompting the backend to process the request and return the relevant data, ensuring a seamless and dynamic user experience.

3.1.2 Server-Side (Backend):

The backend of Chat Blog, built on Spring Boot with Spring MVC, serves as the core processing engine by organizing the application into controllers, services, and repositories, ensuring a clean separation of concerns. Spring MVC's Rest Controller defines RESTful API endpoints that manage operations for posts, comments, messages, and user profiles, converting client requests into actions such as creating, retrieving, updating, and deleting data. For example, a POST request to /api/posts creates a new post, while a GET request to /api/users/{id} fetches user profile details. The backend enforces business logic, ensuring data integrity and validation—such as preventing duplicate usernames or invalid email formats. This logic also includes error handling, security measures using JWT for authentication, and user authorization, making sure only authenticated users can access certain features. The backend acts as a bridge between the frontend and database, simplifying the interface for the client while managing data processing complexity.

3.1.3 Database:

MySQL serves as the database solution for Chat Blog, providing reliable storage for all social media records, including users, posts, comments, and messages. It stores attributes like user_id, post_id, comment_id, and more in structured tables. The backend leverages Spring Data JPA to interact with MySQL, enabling efficient management of operations. For instance, when a user creates a post, the backend formats the data and saves it into the database table with a unique post_id. Each entry can be precisely accessed, updated, or deleted using its unique identifier. The database also supports search functionalities, where queries retrieve only the matching records from the database, optimizing performance. MySQL ensures data persistence, guaranteeing that all social interactions and content remain intact across sessions and application restarts, thus supporting data integrity and reliability.

3.2 Frontend Design

The frontend of our project is built using React 17/18 and designed to provide a responsive and intuitive user experience. The following elements characterize the frontend design:

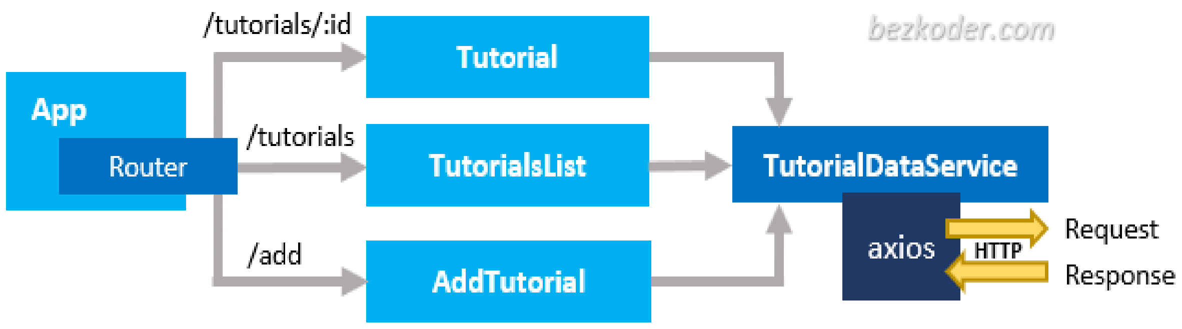


Fig 3.2 Frontend

3.2.1 Component-Based Architecture:

The component-based architecture of Chat Blog forms the core of its design. Using JavaScript and Node.js, the application is structured into reusable components that encapsulate distinct functionalities. For instance, components like PostCard manage the display of individual posts, while the CommentBox handles user input for commenting. Each component maintains its own state and behavior, keeping it independent and modular. This design ensures that elements like buttons for actions (e.g., “Like,” “Delete”) and lists displaying posts or messages are self-contained, enhancing scalability and maintainability. Developers can reuse these components throughout the app, reducing code redundancy and simplifying updates, which supports the addition of new features without impacting existing ones.

3.2.2 Routing with React Router:

Chat Blog utilizes a custom JavaScript-based routing system to provide a smooth Single-Page Application (SPA) experience. Instead of traditional page reloads, the app dynamically updates views, enabling seamless transitions between different sections like the home feed, chat interface, and user profiles. When users navigate from their feed to a specific post or a chat, the application changes only the relevant view, preserving the overall app state without a full reload. This setup ensures faster navigation and a consistent user experience. The routing logic manages various paths, such as /posts, /chat, and /profile/{id}, allowing efficient handling of user interactions within a single page structure.

3.2.3 Styling and Responsiveness:

Styling in Chat Blog leverages SASS alongside traditional CSS to create a modern, responsive design. The use of SASS allows for advanced styling capabilities like variables, nested rules, and mixins, promoting maintainable and scalable stylesheets. The application ensures responsiveness through a combination of SASS and CSS media queries, adapting layouts to different devices. For example, the feed may display in a grid format on larger screens but switch to a single-column view on mobile for better readability. This layered styling approach, along with custom HTML elements, ensures that the interface is both visually appealing and highly functional, delivering a consistent experience across all screen sizes. The use of Bootstrap components further enhances the design with pre-built UI elements like buttons and modals, speeding up development while maintaining a cohesive look. Overall, this approach ensures that Chat Blog is user-friendly, accessible, and engaging across various devices.

3.3 Backend Design

The backend of the Chat Blog application is built using Spring MVC, which serves as the foundation for handling server-side logic, managing data processing, and ensuring efficient communication between the frontend and the database. The backend design focuses on scalability, security, and seamless integration, making it suitable for a social media platform.

3.3.1 RESTful API Development:

The backend of Chat Blog exposes a comprehensive set of RESTful APIs to handle various operations like user management, posts, comments, and real-time chat functionalities. Each API endpoint is designed following REST principles, providing a structured way for the frontend to interact with the server using HTTP methods such as GET, POST, PUT, and DELETE. For instance:

- GET requests are used to fetch user profiles, posts, and comments.
- POST requests handle creating new user accounts, posts, or chat messages.
- PUT requests enable updating user information or modifying existing posts.
- DELETE requests allow for removing posts, comments, or deactivating user accounts.

This RESTful approach ensures clear communication between the client and server, making it easy for the frontend to perform operations. The separation of endpoints for different resources (like /users, /posts, /comments, and /messages) supports modularity and scalability, allowing the application to handle increased loads efficiently.

3.3.2 Business Logic and Validation:

The backend of the Chat Blog application implements critical business logic to ensure the integrity and reliability of data. Before any data is persisted to the database, the backend performs rigorous validation checks. These checks include ensuring that required fields such as username, email, and post content are correctly filled out, and validating unique constraints for fields like email and username to prevent duplicates. The backend also enforces security measures by integrating Spring Security with JWT tokens for user authentication and authorization. This ensures that only authenticated users can perform sensitive actions, such as posting, commenting, or editing content. Additionally, the backend includes error handling mechanisms that return meaningful responses in case of failures, improving the user experience and making the application more robust.

3.3.3 Database Integration with MySQL:

The backend integrates seamlessly with a MySQL database using Spring Data JPA, which abstracts complex SQL queries into simple repository method calls, enhancing efficiency and ease of use. This integration supports various database operations, such as saving user profiles, posts, and messages, fetching posts and comments with pagination for optimized performance, and updating or deleting records based on specific conditions. For example, when a user posts a new message, the backend uses JPA repositories to persist the data in the database, ensuring data consistency and integrity. When users request to view their chat history, the backend retrieves the data efficiently through optimized SQL queries generated by Spring Data JPA. Additionally, during development and testing, the use of an H2 in-memory database enables quick data validation without affecting the production database.

4. PROJECT PLANNING

Project planning for the Chat Blog social media WebApp is essential for ensuring smooth development and successful deployment. It involves defining clear requirements, designing a robust database structure, and organizing the development process to meet stakeholder expectations. The planning phase focuses on gathering user needs, establishing core functionalities like account creation, posting, commenting, and social media interactions, as well as defining user roles and security requirements. A well-executed project plan helps guide the development team through each phase, ensuring timely delivery of a secure, scalable, and user-friendly application.

4.1 Requirements

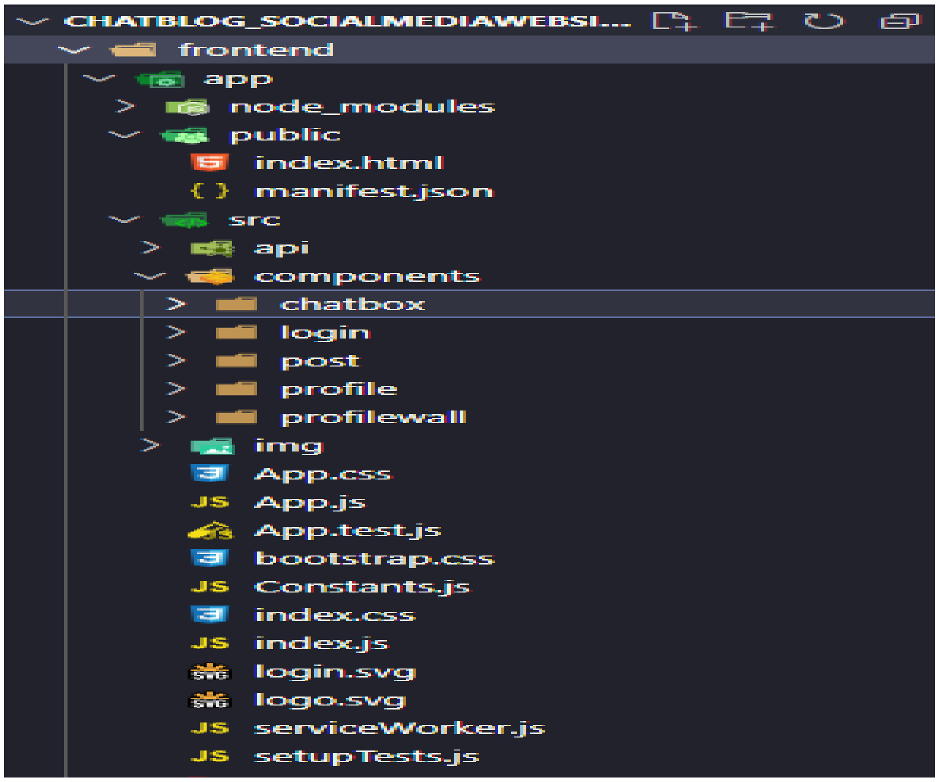
In the requirements phase, the core features of the Chat Blog social media WebApp are defined to meet the needs of users interacting in a social media environment. These features include account creation, posting, commenting, and following users, with functionalities to allow users to engage in discussions, share posts, and receive real-time notifications. The system should allow users to register with essential details like username, email, and password, and offer the ability to post content, comment on posts, and interact with others. Additionally, key requirements include implementing user authentication and authorization with JWT tokens, ensuring that only authenticated users can post or edit content. The app should also feature social interaction mechanisms like following other users and receiving notifications about new posts or comments. Furthermore, validation rules will be implemented to ensure that users provide valid, unique, and required information when registering or posting content.

4.2 Database Design

The database design for the Chat Blog social media WebApp is structured using MySQL. Key entities such as users, posts, comments, and followers are stored in separate tables. The user table stores basic user information like username, email, and password, with unique constraints to prevent duplicates. The posts table contains details like post content and the user who created it. Comments are stored in a separate table linked to posts, and a relationships table tracks users following other users. By utilizing Spring Data JPA, the backend interacts with the database to perform operations on these entities efficiently. This design ensures scalability and reliability as the app grows, providing smooth and secure data management.

5. FRONTEND DEVELOPMENT

Frontend development for the Chat Blog social media WebApp focuses on creating an engaging, intuitive, and responsive user interface that enhances the overall user experience. This involves building a solid HTML/CSS structure to design essential elements like forms, buttons, and navigation components, ensuring they are visually appealing and easy to use. By integrating JavaScript and React JS, the application becomes dynamic, enabling real-time updates and smooth interactions, which contribute to a seamless user experience.



```
CHATBLOG_SOCIALMEDIAWEBSI...
  frontend
    app
      node_modules
      public
        index.html
        manifest.json
      src
        api
        components
          chatbox
          login
          post
          profile
          profilewall
        img
        App.css
        App.js
        App.test.js
        bootstrap.css
        Constants.js
        index.css
        index.js
        login.svg
        logo.svg
        serviceWorker.js
        setupTests.js
```

Fig 5.1 Front-end Directory

5.1 HTML/CSS Structure:

The foundation of the Chat Blog application lies in a well-organized HTML/CSS structure. HTML (HyperText Markup Language) is used to create the basic skeleton of the app, including elements such as user registration forms, post creation forms, buttons for interactions like submitting or deleting content, and lists for displaying posts and comments.

Each HTML element is structured with semantic correctness, improving both accessibility and SEO. CSS (Cascading Style Sheets) is used to style these elements, providing an aesthetically pleasing and user-friendly interface. By implementing a responsive design with CSS frameworks like Bootstrap or Tailwind CSS, the application ensures compatibility across various screen sizes and devices, from desktops to mobile phones. This focus on responsive design ensures that users can enjoy an optimal experience regardless of the device they use. The structured HTML/CSS approach enhances both the visual appeal and usability of the application, guiding users through their interactions smoothly and efficiently

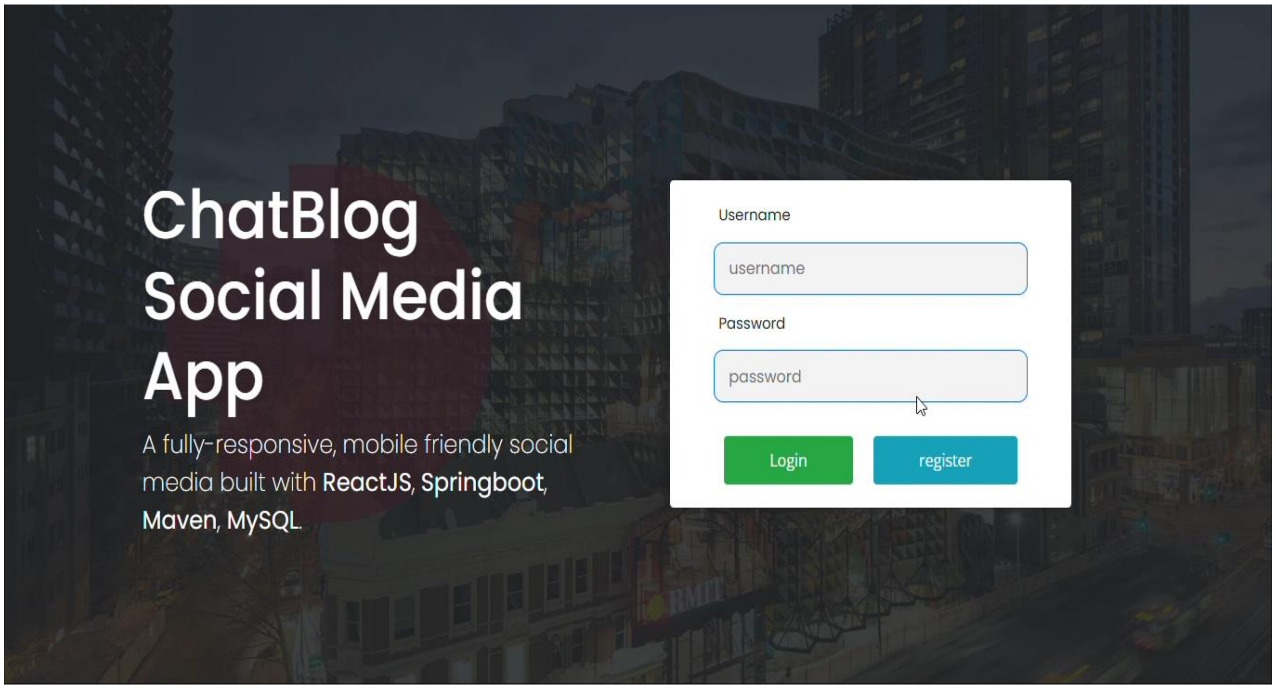
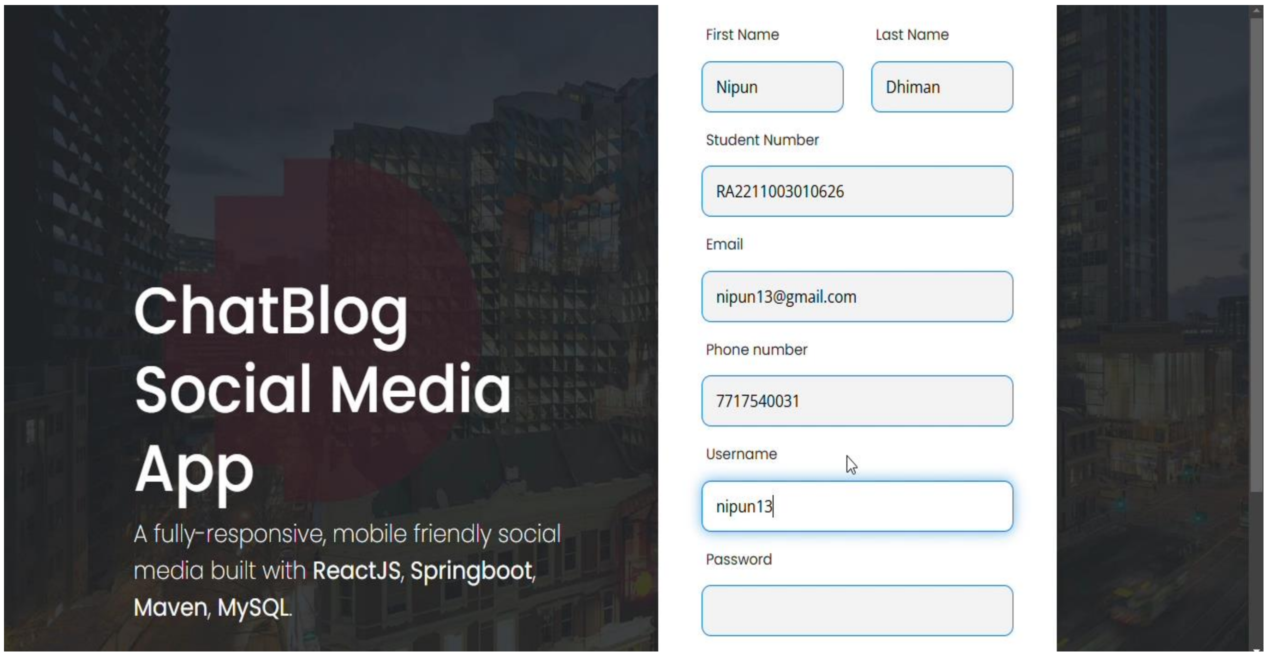

ChatBlog
Social Media
App
A fully-responsive, mobile friendly social media built with ReactJS, Springboot, Maven, MySQL.
Username
username
Password
password
Login
register

Fig 5.2 Main page

5.2 JavaScript and Frontend Frameworks (React JS):

To add interactivity and dynamic functionality to the Chat Blog, JavaScript plays a critical role. React JS, a powerful frontend library developed by Facebook, is employed to create a modern and efficient user interface. React’s component-based architecture enables developers to build reusable UI components, which encapsulate both logic and presentation. This modular structure simplifies the development process and makes it easier to manage, update, and scale different parts of the application.

React’s state management allows for real-time updates, ensuring that any changes made to data, such as a new post or comment, are instantly reflected in the UI. For example, when a user posts a comment, React automatically updates the display without requiring a full-page reload. This is achieved through React’s virtual DOM, which optimizes the rendering process by updating only the components that have changed, leading to improved performance and a smoother user experience. Additionally, React integrates seamlessly with tools like Redux or the Context API, enabling centralized state management that streamlines data flow across the entire application.


ChatBlog Social Media App
A fully-responsive, mobile friendly social media built with ReactJS, Springboot, Maven, MySQL.

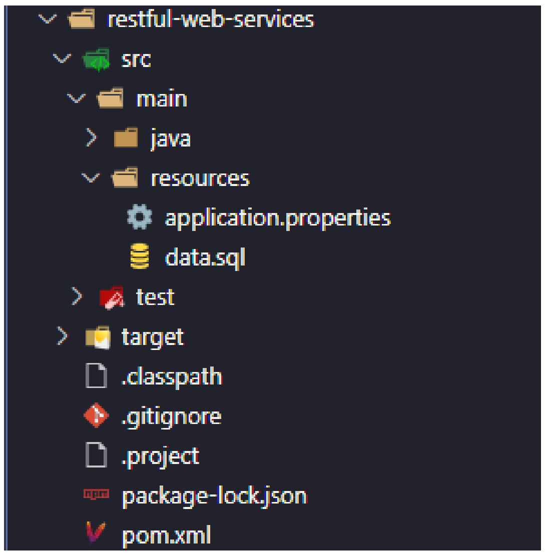
First Name: Nipun
Last Name: Dhiman
Student Number: RA2211003010626
Email: nipun13@gmail.com
Phone number: 7717540031
Username: nipun13
Password:

Fig 5.3 Editing

Furthermore, JavaScript is utilized to handle asynchronous operations, such as fetching user data, posts, and comments from the backend API using Axios or the Fetch API. This ensures smooth communication between the frontend and backend, allowing users to create, read, update, and delete content effortlessly. By combining HTML/CSS for layout and styling with React for dynamic functionality, the frontend development of the Chat Blog Social Media WebApp results in a responsive, engaging, and user-friendly application that enhances the overall user experience.

6. BACKEND DEVELOPMENT

Backend development for the Chat Blog Social Media WebApp focuses on creating a robust API that facilitates seamless communication between the frontend and server while ensuring data security and integrity. Utilizing Spring Boot and Spring Data JPA, the application is designed to handle various operations efficiently. Key features include authentication for secure access, structured database connectivity, and a well-defined repository pattern, all contributing to a scalable and maintainable backend architecture that supports the application's functionality.



```
restful-web-services
  src
    main
      java
      resources
        application.properties
        data.sql
    test
  target
  .classpath
  .gitignore
  .project
  package-lock.json
  pom.xml
```

Fig 6.1 Back-end Directory

6.1 API Design, Authentication, and Database Connectivity:

The backend of the Chat Blog social media WebApp is fundamentally centered on the development of a well-structured API that acts as a bridge between the frontend and the server. Employing RESTful API design principles, the application defines clear and concise endpoints for core functionalities such as managing user profiles, posts, and comments.

For example, endpoints are established for creating a new post (POST /api/posts), retrieving all posts (GET /api/posts), updating a post (PUT /api/posts/{id}), and deleting a post (DELETE /api/posts/{id}). Each endpoint is carefully crafted to handle specific HTTP methods and return appropriate status codes, ensuring smooth interactions and clarity in client-server communication..

Authentication is a crucial aspect of API security, implemented through mechanisms such as JSON Web Tokens (JWT). When a user logs in, they receive a token that must be included in the headers of subsequent requests. This approach ensures that only authenticated users can access protected routes, thus safeguarding sensitive operations such as creating or modifying posts. Additionally, implementing role-based access control can further enhance security by allowing different user roles (e.g., admin, regular user) to perform specific actions based on their permissions.

Database connectivity is established using Spring Data JPA, which abstracts the complexities of database interactions. The configuration includes setting up a connection to a MySQL database, allowing the application to execute SQL queries and manage data effectively. Spring Boot simplifies this setup with its auto-configuration capabilities, enabling developers to focus on business logic rather than boilerplate code. By leveraging connection pooling, the application can handle multiple database connections efficiently, optimizing performance and resource utilization.

6.2 Database Management – Spring Boot Concepts:

Spring Boot provides a powerful framework for managing database interactions and operations through its integration with Spring Data JPA. In this setup, each entity (e.g., users, posts, comments) is represented by an entity class, which is annotated with JPA annotations to map it to a corresponding table in the database. For example, the @Entity annotation designates the class as a JPA entity, while the @Table annotation specifies the database table name. Attributes of the class, such as id, title, content, and createdAt, are mapped to columns in the database, facilitating automatic synchronization between Java objects and relational database entries.

The repository layer, typically defined as an interface extending JpaRepository, encapsulates data access logic. This pattern promotes separation of concerns and provides a clean API for the service layer. For instance, the PostRepository interface may include methods like findAll(), save(Post post), and deleteById(Long id), allowing the service layer to perform data operations.

The service layer orchestrates calls to the repository, handling business logic and transaction management. Each operation is implemented as a method within a service class, promoting reusability and maintainability. For example, a method to create a new post might involve validation checks before saving the post object to the database. Spring Boot’s built-in transaction management ensures that operations are atomic, meaning that if an error occurs during a transaction (e.g., a failure while updating a record), all changes can be rolled back, maintaining data integrity.

Furthermore, Spring Data JPA supports advanced features such as pagination and sorting out of the box. By extending the repository interface with additional methods like findAll(Pageable pageable) or findByTitleContaining(String title, Pageable pageable), developers can easily implement these functionalities without extensive custom queries, enhancing data retrieval processes and providing a better user experience when dealing with large datasets.

Overall, backend development for the Chat Blog Social Media WebApp emphasizes a structured, secure, and efficient approach to API design, authentication, and database management. By leveraging Spring Boot’s capabilities, developers can create a robust backend that not only meets functional requirements but also scales effectively as the application grows.

7. TESTING AND DEPLOYMENT

For the Chat Blog Social Media WebApp project, I executed a comprehensive testing strategy to ensure the application functions effectively and meets user expectations. This process involved several critical stages, including unit testing, integration testing, and user acceptance testing (UAT), each designed to identify potential issues and confirm that all components work seamlessly together.

7.1 Testing Strategy

The testing process began with unit testing, focusing on verifying the functionality of individual components within both the frontend and backend. On the frontend side, using ReactJS, Jest, and React Testing Library, I wrote tests to validate component behavior in isolation. These tests ensured that form inputs correctly handled user data, buttons triggered expected actions, and API calls returned anticipated responses. For example, the user registration form was tested to confirm it captured user inputs accurately, validated fields properly, and triggered the correct API call to create a new user profile in the backend.

For the backend, developed with Spring Boot, JUnit and Mockito were utilized to test the service and repository layers. The testing efforts were concentrated on the core business logic, ensuring user profiles, posts, and comments were correctly saved to the database, accurately retrieved, and handled errors effectively when dealing with invalid data. Tests included scenarios like verifying that creating a post with invalid fields triggered appropriate validation errors, and checking that retrieving posts returned the correct datasets. This early identification of issues allowed for quick fixes, streamlining the development process.

Following unit testing, integration testing was conducted to evaluate the interactions between the frontend and backend systems. This phase ensured that the components communicated effectively and that data flowed as expected. Using Postman and JUnit, API testing was performed to simulate requests and verify that the responses were correct. The core operations for user accounts, posts, and comments were tested by sending HTTP requests to the Spring Boot REST API endpoints. For instance, tests confirmed that a POST request to create a new post returned a 201-status code along with the correct post object.

Similarly, GET requests were validated to retrieve lists of posts and user profiles, while DELETE requests were checked to ensure the accurate removal of specified posts from the database. These integration tests solidified the reliability of client-server communication.

User Acceptance Testing (UAT) was the final testing phase, where selected users interacted with the live application to provide feedback on usability, functionality, and performance. This feedback was instrumental in making final adjustments and refinements before deployment.

7.2 Deployment Process

With testing complete and all identified issues resolved, the deployment phase was initiated for the Chat Blog social media WebApp. The application was deployed using Heroku for the backend and GitHub Pages for the frontend.

For the backend, the Spring Boot application was prepared for production by configuring environment variables and optimizing database connection settings. A new Heroku app was created using the Heroku CLI, where environment variables like MySQL database credentials were set up. The backend code was then pushed to Heroku, and after a brief build process, the API became publicly accessible through a Heroku-provided URL.

For the frontend deployment, the React application was built using the npm run build command, which compiled the project into optimized static files, generating a build folder containing all necessary HTML, CSS, and JavaScript files. A new branch named gh-pages was created in the GitHub repository, dedicated to deploying the application. The contents of the build folder were pushed to this branch, enabling GitHub Pages to host the frontend. In the repository settings, GitHub Pages was enabled for the gh-pages branch, providing a live URL for users to access the frontend application.

After deployment, the frontend was configured to interact with the live Heroku API endpoint, ensuring full functionality of operations, authentication, and data retrieval in the production environment. This seamless integration between the frontend and backend ensured that the Chat Blog social media WebApp delivered a smooth and reliable user experience.

By implementing a thorough testing strategy and a streamlined deployment process, the application is now live and fully operational, offering a secure and efficient platform for users to interact and share content.

8. RESULTS AND CONCLUSION

The Chat Blog Social Media WebApp project successfully achieved its primary objectives of creating a user-friendly social media platform for content sharing and interaction. Throughout the development process, robust functionalities were implemented, allowing users to create, view, update, and delete posts, profiles, and comments efficiently. The application leverages a solid tech stack, using ReactJS for the frontend, Spring Boot for the backend, and MySQL for database management, ensuring scalability and high performance.

8.1 Conclusion

The Chat Blog Social Media WebApp project successfully achieved its primary objectives of creating a user-friendly social media platform for content sharing and interaction. Throughout the development process, robust functionalities were implemented, allowing users to create, view, update, and delete posts, profiles, and comments efficiently. The application leverages a solid tech stack, using ReactJS for the frontend, Spring Boot for the backend, and MySQL for database management, ensuring scalability and high performance. The project not only met all initial requirements but also provided valuable insights into the complete software development lifecycle. It significantly enhanced my skills in key areas such as frontend and backend integration, API development, testing, debugging, and deployment. Overall, the Chat Blog social media WebApp serves as a testament to the potential of combining modern web frameworks to deliver a cohesive and responsive application.

8.2 Future Scope

Looking ahead, there are several promising opportunities for enhancing the Chat Blog Social Media WebApp:

User Authentication and Enhanced Security: Expanding the authentication system to include social login options (e.g., Google, Facebook) and implementing multi-factor authentication (MFA) can enhance user security. Role-based access control could further restrict access to admin-specific functionalities.

Advanced Search and Filtering: Improving the search functionality by incorporating advanced filtering options (such as filtering posts by date, category, or popularity) can enhance user experience, allowing users to discover content more efficiently.

Responsive Design Enhancements: Although the application is already responsive, further refining the user interface for various screen sizes, particularly tablets and mobile devices, could improve the overall user experience. Implementing a mobile-first approach would ensure consistency across all devices.

Real-Time Features and Notifications: Introducing real-time capabilities like push notifications, live comment updates, and chat functionality would enhance user engagement. Technologies such as WebSockets or Firebase could be integrated to support these features.

Analytics Dashboard: Developing an analytics dashboard for users to track post engagement (e.g., views, likes, comments) would provide valuable insights. Admins could also benefit from monitoring user activity, registration trends, and content popularity, aiding in data-driven decision-making.

Deployment on Cloud Platforms: Migrating the backend to cloud platforms like AWS, Azure, or Google Cloud would offer enhanced scalability, security, and performance. Leveraging services like AWS RDS for database management or Azure’s App Services for load balancing could improve the platform’s reliability.

Integration of a Rich Text Editor: Implementing a rich text editor for creating posts would allow users to format their content more effectively, adding features like text styling, image uploads, and embedding links, which would enrich the content creation process.

Content Moderation and Reporting System: Adding features for content moderation, such as automated detection of inappropriate content using machine learning, and enabling users to report posts or comments, would help maintain a positive community environment.

By focusing on these areas for future development, the Chat Blog Social Media WebApp can evolve into a more comprehensive and versatile platform, catering to a broader user base and expanding its utility for both personal and professional social networking.

REFERENCES

[1] Altamira Team, 2020. *Best Web Application Technology Stack in 2022*. Available at: https://www.altamira.ai/blog/how-to-choose-your-technology-stack/. Accessed 12 February 2023.

[2] Bennison, J., 2022. *What is module in Node.js*. Available at: https://medium.com/yavar/what-is-the-node-module-in-node-js-19ef41820af8. Accessed 13 February 2023.

[3] Ojo, M., 2022. *How to automate API tests with Postman*. Available at: https://blog.logrocket.com/how-automate-api-tests-postman/. Accessed 14 February 2023.

[4] Bormon, M., 2022. *What is Mongoose?* Available at: https://medium.com/@monibbormon14/what-is-mongoose-c1bc3031cc08. Accessed 13 February 2023.

[5] Chaddha, M., 2022. *Routing in Node.js*. Available at: https://www.codingninjas.com/codestudio/library/routing-in-node-js. Accessed 13 February 2023.

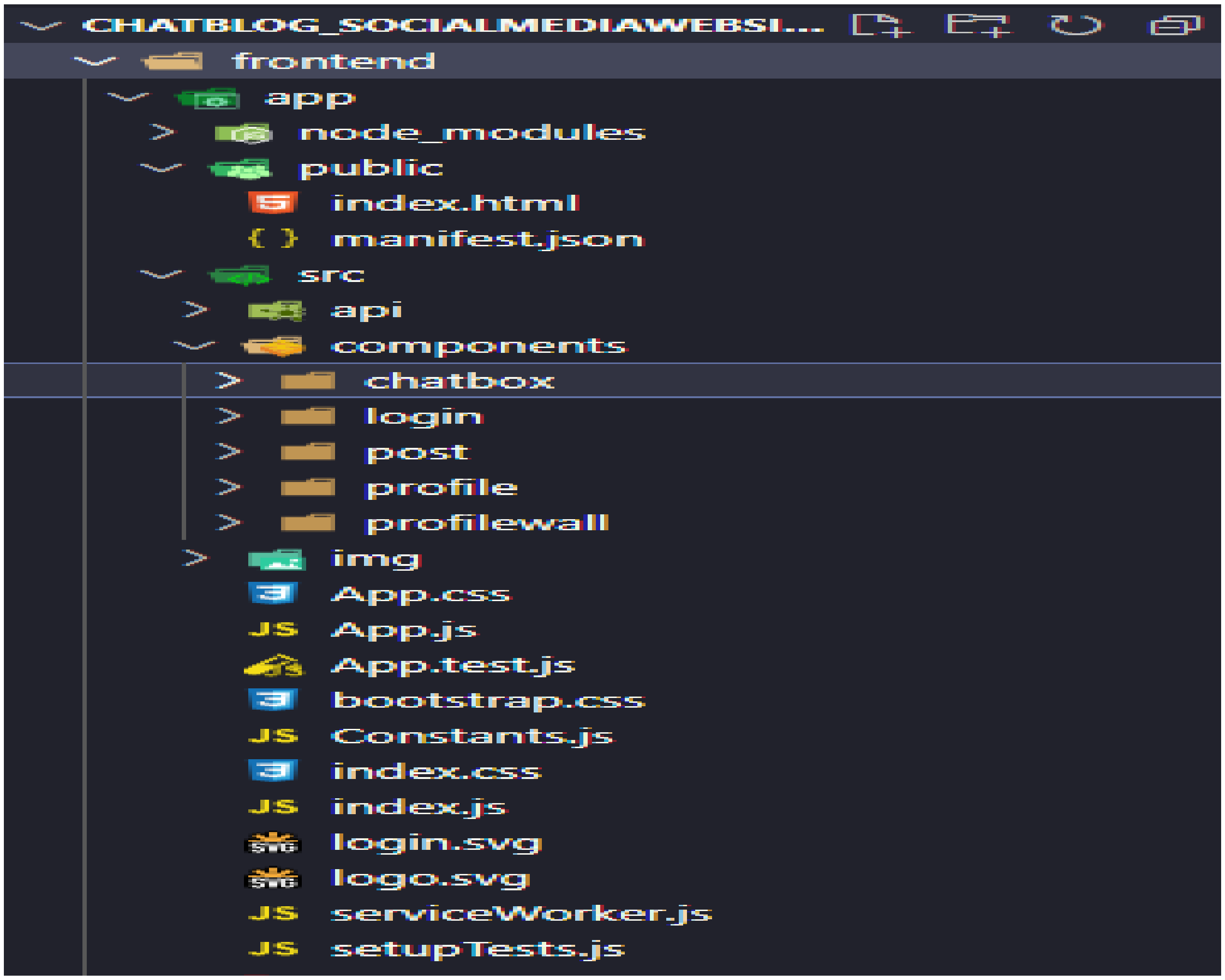
[6] Educative, Inc., 2022. *What is MERN Stack*. Available at: https://www.educative.io/answers/what-is-mern-stack. Accessed 12 February 2023.

[7] Flexible, 2023. *An Intro to ExpressJS*. Available at: https://flexiple.com/express-js/deep-dive/. Accessed 12 February 2023.

[8] Hamedani, M., 2018. *React Functional Components*. Available at: https://programmingwithmosh.com/react/react-functional-components/. Accessed 12 February 2023.

[9] Heller, M., 2022. *What is Node.js? The JavaScript Runtime Explained*. Available at: https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html. Accessed 13 February 2023.

APPENDIX 1 – CODE



```
CHATBLOG_SOCIALMEDIAWEBSI...
  frontend
    app
      node_modules
      public
        index.html
        manifest.json
      src
        api
        components
          chatbox
          login
          post
          profile
          profilewall
        img
        App.css
        App.js
        App.test.js
        bootstrap.css
        Constants.js
        index.css
        index.js
        login.svg
        logo.svg
        serviceWorker.js
        setupTests.js
```

Front-end directory

Common/add-tutorial-component.js

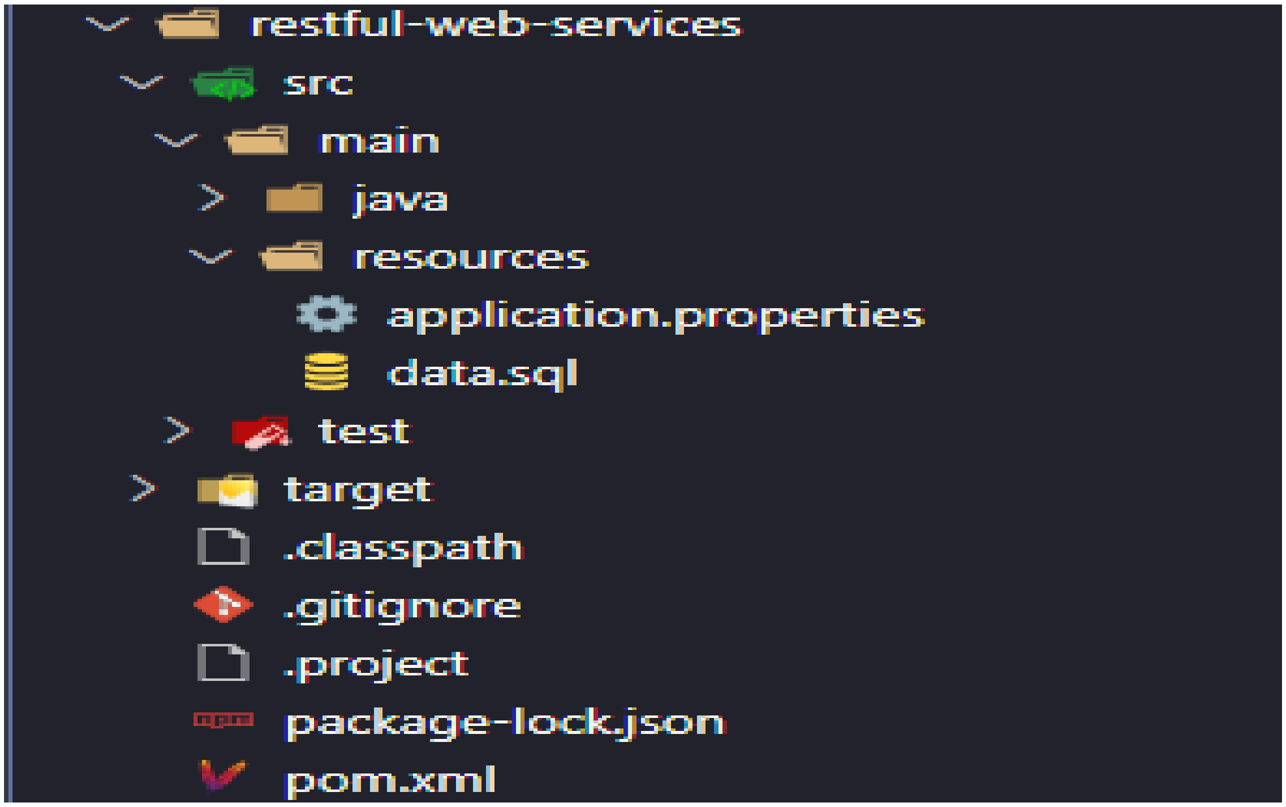
```
import React from "react";
import loginImg from "../../login.svg";
import { Formik, Form, Field, ErrorMessage } from 'formik';
import AccountProfileService from "../../api/main/AccountProfileService";
import OverlayTrigger from "react-bootstrap/OverlayTrigger";
import Tooltip from "react-bootstrap/Tooltip";
export class Register extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      firstname: this.props.firstname,
      lastname: this.props.lastname,
      studentnumber: this.props.studentnumber,
      email: this.props.email,
      phonenumber: this.props.phonenumber,
      aboutme: this.props.aboutme,
      usernamecheck: false,
      isStudentnumberDuplicate: false,
      isEmailDuplicate: false,
      isPhonenumberDuplicate: false
    }
    this.validate = this.validate.bind(this)
  }

  checkDuplicateUser(username) {
    if (username != null) {
      AccountProfileService.checkDuplicateUsername(username)
        .then(response => {

          if (response.data == true) {
            this.setState({
              usernamecheck: true
            });
          }
          else if (response.data == false) {
            this.setState({
              usernamecheck: false
            });
          }

        })
    }
  }

  checkDuplicateStudentnumber(studentnumber) {

    if (studentnumber != null) {
      AccountProfileService.checkDuplicateStudentnumber(studentnumber)
        .then(response => {

          if (response.data == true) {
            this.setState({
              isStudentnumberDuplicate: true
            });
          }
          else if (response.data == false) {
            this.setState({
              isStudentnumberDuplicate: false
            });
          }

        })
    }
  }

  checkDuplicateEmail(email) {
    if (email != null) {
      AccountProfileService.checkDuplicateEmail(email)
        .then(response => {
          if (response.data == true) {
            this.setState({
              isEmailDuplicate: true
            });
          }
          else if (response.data == false) {
            this.setState({
              isEmailDuplicate: false
            });
```

```
          });
```

```
        }
      })
  }
}

checkDuplicatePhonenumber(phonenumber) {
  if (phonenumber != null) {
    AccountProfileService.checkDuplicatePhonenumber(phonenumber)
      .then(response => {

        if (response.data == true) {
          this.setState({
            isPhonenumberDuplicate: true
          });
        }
        else if (response.data == false) {
          this.setState({
            isPhonenumberDuplicate: false
          });
        }

      })
  }
}

validate(values) {
  let errors = {}

  const usernameRegex = /^[a-zA-Z0-9._-]*$/
  const nameCheck = /^[a-zA-Z\s]*$/
  const phoneCheck = /^[6-9]\d{9}$/
  const emailCheck = /^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$/
  const studentnumberCheck = /^RA\d{13}$/
  const passwordCheck = /^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.{8,})/

  this.checkDuplicateUser(values.username);
  this.checkDuplicateEmail(values.email);
  this.checkDuplicateStudentnumber(values.studentnumber);
  this.checkDuplicatePhonenumber(values.phonenumber);

  if (values.username == null) {
    errors.username = "Please enter your username"
  } else if (!usernameRegex.test(values.username)) {
    errors.username = 'Please enter a valid username'
  }

  else if (this.state.usernamecheck == true) {

    errors.username = "This username already exists";
  }


  if (!passwordCheck.test(values.password)) {
    errors.password = "Minumum 8 characters long, 1 lower and upper case, 1 number"
  }
```

```
if (values.retypepassword != values.password) {
  errors.retypepassword = "Password doesn't match"
}

if (values.firstname == null) {
  errors.firstname = 'Enter your first name'
} else if (!nameCheck.test(values.firstname)) {
  errors.firstname = 'Please enter a valid name'
}

if (values.lastname == null) {
  errors.lastname = 'Enter your last name'
} else if (!nameCheck.test(values.lastname)) {
  errors.lastname = 'Please enter a valid name'
}

if (values.phonenumber == null) {
  errors.phonenumber = 'Enter your phone number'
} else if (!phoneCheck.test(values.phonenumber)) {
  errors.phonenumber = 'Please enter a valid phone number'
} else if (this.state.isPhonenumberDuplicate === true) {
  errors.phonenumber = 'This phone number is already in use'
}

if (values.email == null) {
  errors.email = 'Enter your email'
} else if (!emailCheck.test(values.email)) {
  errors.email = 'Please enter a valid email'
} else if (this.state.isEmailDuplicate == true) {
  errors.email = 'This email is already in use'
}

if (values.studentnumber == null) {
  errors.studentnumber = 'Enter your student number'
} else if (!studentnumberCheck.test(values.studentnumber)) {
  errors.studentnumber = 'Please enter a valid student number (includes the s)'
} else if (this.state.isStudentnumberDuplicate == true) {
  errors.studentnumber = 'This student number is already in use'
}

return errors

}

render() {
  return (
    <div className="base-container">
      <div className="content">
        <Formik className="form form-row"

          onSubmit={this.props.handleRegister}
          validateOnChange={this.validate}
          validateOnBlur={this.validate}
          validateOnSubmit={this.validate}
          validate={this.validate}
          enableReinitialize={true}
        >
```

```
            {
              (props) => (
                <Form>
                  <div className="row">
                      <div className="col">
                        <fieldset className="form-group">
                          <label htmlFor="firstname">First Name</label>
                          <Field className="field" type="text" name="firstname" />
                          <ErrorMessage name="firstname" component="div"
                                        className="checkError" />
                        </fieldset>
                      </div>
                      <div className="col">
                        <fieldset className="form-group">
                          <label htmlFor="lastname">Last Name</label>
                          <Field className="field" type="text" name="lastname" />
                          <ErrorMessage name="lastname" component="div"
                                        className="checkError" />
                        </fieldset>
                      </div>
                  </div>
                  <fieldset className="form-group">
                    <label htmlFor="studentnumber">Student Number</label>
                    <OverlayTrigger placement={"bottom"} overlay={<Tooltip id={"tooltip-bottom"}>Please include
's'</Tooltip>}>
                      <Field className="field" type="text" name="studentnumber" />
                    </OverlayTrigger>
                    <ErrorMessage name="studentnumber" component="div"
                                  className="checkError" />
                  </fieldset>

                  <fieldset className="form-group">
                    <label htmlFor="email">Email</label>
                    <Field className="field" type="text" name="email" />
                    <ErrorMessage name="email" component="div"
                                  className="checkError" />
                  </fieldset>

                  <fieldset className="form-group">
                    <label htmlFor="phonenumber">Phone number</label>
                    <OverlayTrigger placement={"bottom"} overlay={<Tooltip id={"tooltip-bottom"}>Start with 0 or
'+'</Tooltip>}>
                      <Field className="field" type="text" name="phonenumber" />
                    </OverlayTrigger>
                    <ErrorMessage name="phonenumber" component="div"
                                  className="checkError" />
                  </fieldset>

                  <fieldset className="form-group">
                    <label htmlFor="username">Username</label>
                    <Field className="field" type="text" name="username" />
                    <ErrorMessage name="username" component="div"
                      className="checkError" />
                  </fieldset>

                  <fieldset className="form-group">
<label htmlFor="password">Password</label>
                    <OverlayTrigger placement={"bottom"} overlay={<Tooltip id={"tooltip-bottom"}>More than 8 chars, 1
lower and upper case, 1 number</Tooltip>}>
                      <Field className="field" type="password" name="password" />
                    </OverlayTrigger>
                    <ErrorMessage name="password" component="div"
                      className="checkError" />
                  </fieldset>

                  <fieldset className="form-group">
                    <label htmlFor="retypepassword">Retype Password</label>
                      <Field className="field" type="password" name="retypepassword" />
                    <ErrorMessage name="retypepassword" component="div"
                      className="checkError" />
                  </fieldset>

                  <div className={"footerBtn"}>
                    <button type="submit" className="btn text-center btn-info center" name={"register"}>
                    Register
                    </button>
                  </div>

                  <p className={"loginLink"}>
                    <a href="#" onClick={this.props.changeState}>Wait, I already have an account ;)</a>
                  </p>
                </Form>
              )
            }
          </Formik>
        </div>
      </div>

    );
  }
                              }}
```



```
restful-web-services
  src
    main
      java
      resources
        application.properties
        data.sql
    test
  target
  .classpath
  .gitignore
  .project
  package-lock.json
  pom.xml
```

Back-end Directory

src/main/java/controller/TutorialController.java

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>


<groupId>com.sept.rest.webservices</groupId>

<artifactId>restful-web-services</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>


<name>restful-web-services</name>

<description>Demo project for Spring Boot</description>

<parent>
```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.1.0.RELEASE</version>
<relativePath /> <!-- lookup parent from repository -->
</parent>

<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
<java.version>1.8</java.version>
</properties>

<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-websocket</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-reactor-netty</artifactId>
</dependency>



<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
```

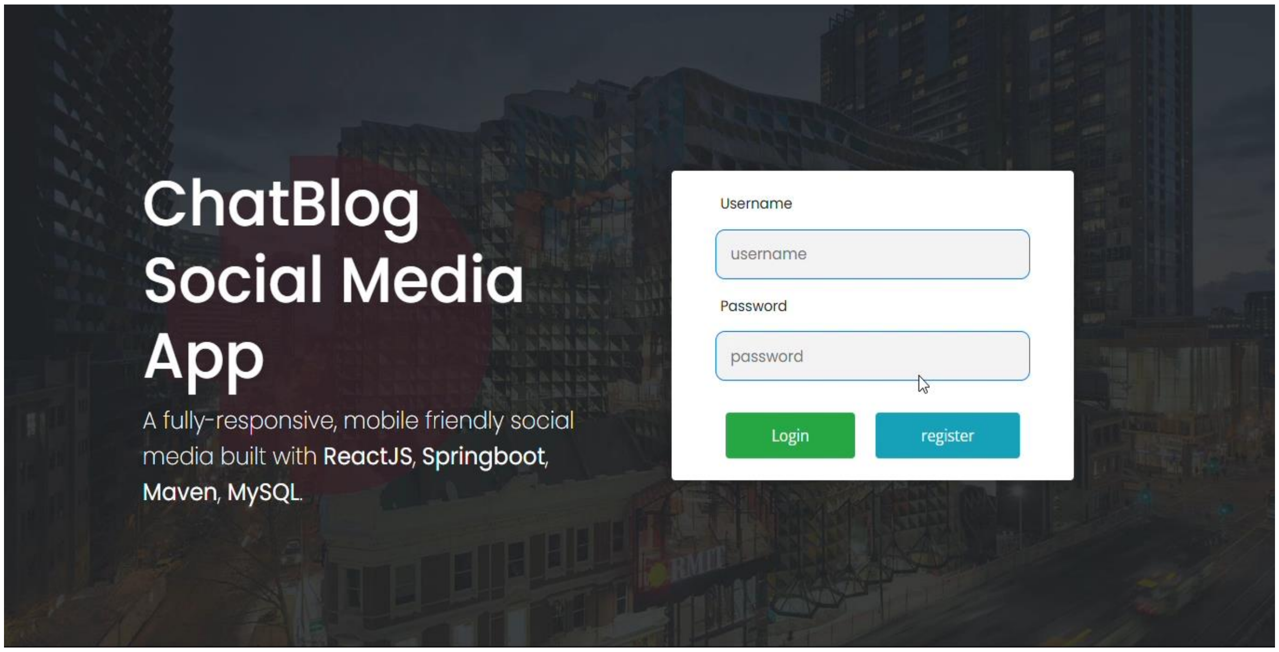
```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt</artifactId>
<version>0.9.1</version>

</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
</dependency>

<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<scope>runtime</scope>
</dependency>

<dependency>
```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>

<dependency>
<groupId>io.rest-assured</groupId>
<artifactId>rest-assured</artifactId>
<version>3.2.0</version>
<scope>test</scope>
</dependency>

<!-- Comment out the following because of deployment to GCP
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
</dependency> -->
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>

<plugin>
<groupId>com.google.cloud.tools</groupId>
<artifactId>appengine-maven-plugin</artifactId>
```

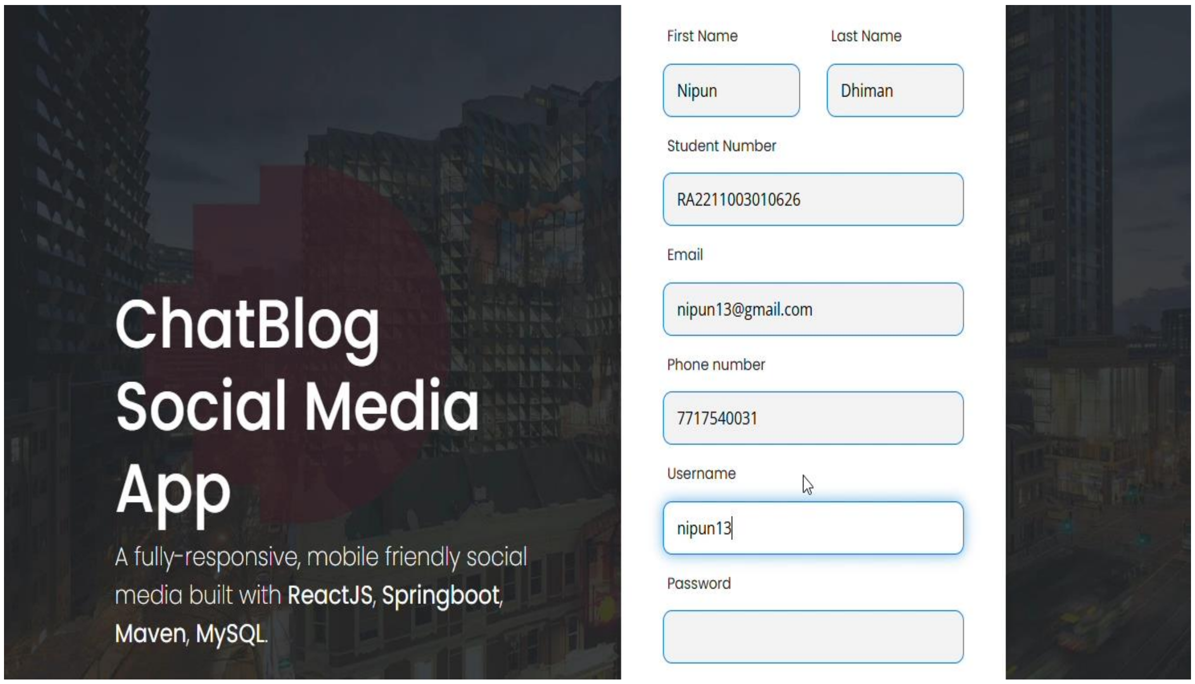
```
<version>1.3.1</version>

</plugin>

</plugins>

</build>


<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>


<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>
```

```
<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>
</project>
```

APPENDIX -2   SCREENSHOT OF MODULES

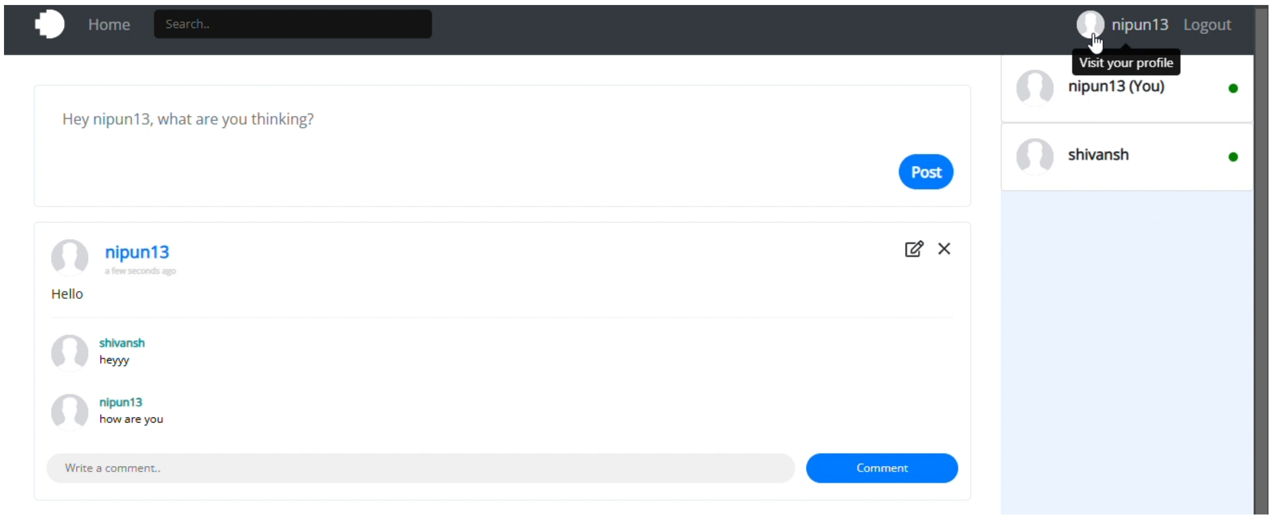

ChatBlog
Social Media
App
A fully-responsive, mobile friendly social media built with ReactJS, Springboot, Maven, MySQL.
Username
username
Password
password
Login
register

Main page


ChatBlog
Social Media
App
A fully-responsive, mobile friendly social media built with ReactJS, Springboot, Maven, MySQL.

First Name: Nipun
Last Name: Dhiman
Student Number: RA2211003010626
Email: nipun13@gmail.com
Phone number: 7717540031
Username: nipun13
Password:

Registering a new account



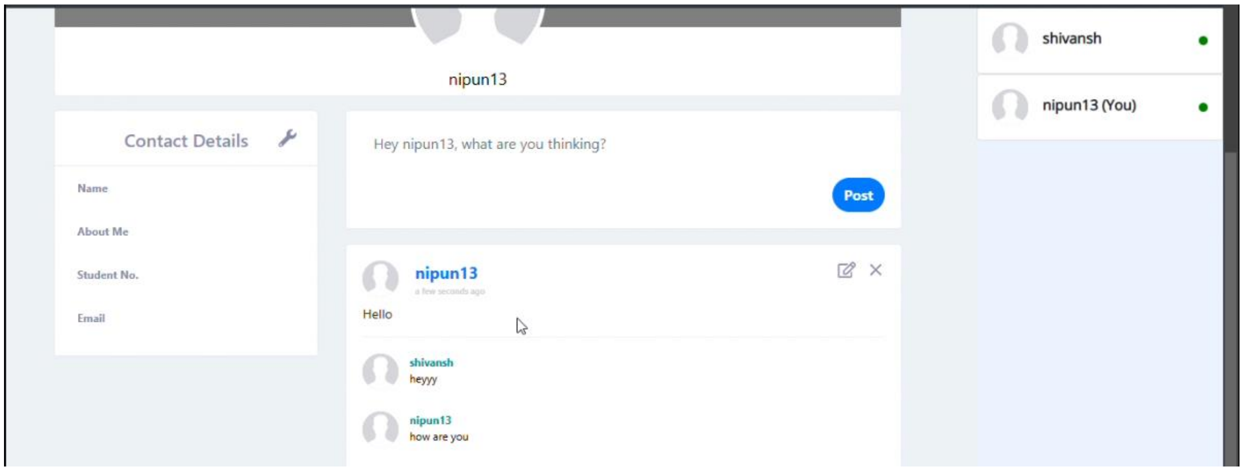
Hey nipun13, what are you thinking?

nipun13
a few seconds ago
Hello

shivansh
heyyy

nipun13
how are you

Home Page



nipun13

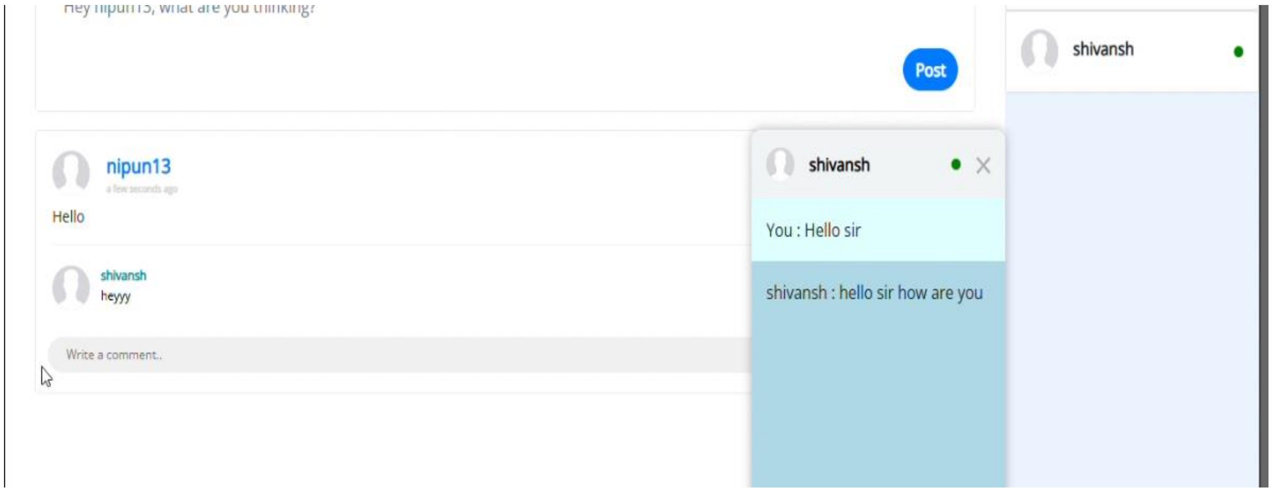
Hey nipun13, what are you thinking?

nipun13
a few seconds ago
Hello

shivansh
heyyy

nipun13
how are you

Profile Page



Hey nipun13, what are you thinking?

nipun13
a few seconds ago
Hello

shivansh
heyyy

shivansh

You : Hello sir

shivansh : hello sir how are you

DM (Direct Message) page