

Design and Development of a Social Media Application Using the Mern Stack

Abhinav Singh¹, Amit Kumar Yadav², Ajeet Kumar³, Aditya Kumar⁴, Kokila S⁵

^{1,2,3,4} UG Scholar, Dept of CSE(DS), Sri Venkateswara College of Engineering and Technology, Chittoor, Andhra Pradesh, India.

⁵ Assistant Professor, Dept of CSE(DS), Sri Venkateswara College of Engineering and Technology, Chittoor, Andhra Pradesh, India.

To Cite this Article: Abhinav Singh¹, Amit Kumar Yadav², Ajeet Kumar³, Aditya Kumar⁴, Kokila S⁵, "Design And Development of a Social Media Application Using the Mern Stack", International Journal of Scientific Research in Engineering & Technology, Volume 06, Issue 05, September-October 2026, PP:173-180.



Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the [Creative Commons Attribution License](#); Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: This article presents the design and development of "Vibe," a web-based social media application built with the MERN stack: MongoDB, Express.js, React.js, and Node.js. The application is intended to support user registration and login, profile management, text/image posts, likes, comments, friend/follow relationships, notifications, and optional messaging and administration functions. The source project specifies JWT-based authentication, input validation, secure backend development, documented RESTful APIs, responsive user interfaces, and maintainability as core objectives.

The project report describes a modular architecture and includes use-case, class, sequence, and authentication-workflow diagrams. It also documents system requirements, a functional algorithm, unit and system testing approaches, eight representative test cases, application screenshots, an error graph, and a classification-accuracy formulation. The report states that the main user functions were implemented and tested. However, it does not provide a numerical dataset, measured latency, throughput, deployment scale, confusion-matrix counts, or a reproducible benchmark protocol. Accordingly, this article preserves reported functional outcomes while distinguishing them from performance claims that require further measurement.

Keywords MERN stack; social media application; MongoDB; Express.js; React.js; Node.js; JWT authentication; REST API; web application testing; user interaction.

I. INTRODUCTION

Social media applications enable users to publish information, maintain profiles, discover other users, and interact through reactions, comments, connections, and messages. A contemporary application must coordinate a responsive client interface, application services, persistent storage, authentication, and reliable handling of user-generated content. The project "Vibe" addresses this problem through a JavaScript-based full-stack architecture.

The project presentation identifies limitations associated with traditional or monolithic systems, including difficult maintenance, scaling challenges as user populations grow, slower performance, and limited flexibility for modern interaction. These are presented as general design concerns rather than as results of a controlled comparison with a named platform. The proposed solution uses the MERN stack to organize the client, server, API, and database responsibilities.

The contribution of this article is a consolidated, research-style account of the supplied project material. It brings together the functional requirements, module descriptions, system diagrams, implementation workflow, test cases, and visual evidence in a structured format. The paper does not claim new algorithms or validated performance improvements beyond the evidence in the source report.

1.1 Objectives

- Implement a secure backend with data encryption and input validation.
- Develop JWT-based authentication for user registration and login.
- Design scalable RESTful APIs with appropriate documentation.
- Create a responsive user interface that works across device sizes.
- Support maintainable, production-oriented application organization.
- Enable content creation and interaction through posts, likes, comments, and user connections.
-

II. EXISTING SYSTEM AND PROBLEM STATEMENT

The source report describes existing social media systems as supporting account creation, content sharing, likes, comments, and interaction through web or mobile interfaces, typically backed by centralized databases. It identifies two broad shortcomings: architecture that becomes difficult to maintain or scale, and limited responsiveness for activities such as updates, notifications, and communication.

Aspect	Concern identified in the project report	Design response in Vibe
Architecture	Traditional/monolithic structures can be difficult to maintain as usage grows.	Separate React client, Express/Node server, and MongoDB persistence.
Responsiveness	Delays may affect updates, notifications, and communication.	Responsive interface and interaction modules; real-time functions are described as intended capabilities.
Flexibility	Integrating modern web technologies may be difficult in less modular systems.	Use a unified JavaScript development stack.
User interaction	Users expect posts, reactions, comments, and connections.	Dedicated post, like/comment, and friend/follow modules.
Security	The project identifies authentication, validation, and secure data handling as requirements.	JWT-oriented login flow, authorization, and input validation are stated objectives.

III. PROPOSED SYSTEM AND RESEARCH SCOPE

Vibe is proposed as a browser-based social platform built using MongoDB, Express.js, React.js, and Node.js. Users can register and log in, maintain profiles, create or remove posts, interact with other users' content, and manage relationships. The presentation also describes notifications, optional real-time chat, and optional administrative moderation. The source states that the MERN stack is expected to provide flexibility, maintainability, and the ability to scale. Those are architectural motivations; the supplied material does not include comparative load tests or a quantified scalability experiment. Therefore, this article treats scalability and performance as design goals rather than experimentally proven outcomes.

3.1 Principal features

Feature	Functionality described
Account and profile	Register, log in, update profile information, change passwords, and upload a profile picture.
Posts	Create, edit, delete, and view text or image posts in a feed.
Likes and comments	Record reactions and comments on posts and maintain interaction counts.
Friend/follow	Send, accept, or reject friend requests; follow or unfollow users; maintain connection lists.
Notifications	Notify users about likes, comments, and friend requests.
Messaging (optional)	Support text, image, and emoji messages using WebSockets or a similar mechanism.
Administration (optional)	Manage users, remove posts, monitor activity, and moderate content.

IV. TECHNOLOGY STACK AND SYSTEM REQUIREMENTS

The source project specifies React.js for the frontend, Node.js and Express.js for backend services, and MongoDB for data storage. Visual Studio Code and Postman are listed as development and API-testing tools; Git and npm support version control and package management.

Category	Requirement stated in source
Processor	Intel Core i3 or higher
Memory	Minimum 4 GB RAM; 8 GB recommended for smoother performance
Storage	Minimum 250 GB free disk space
Display	14-inch or larger; 1366 × 768 resolution
Connectivity	Internet connection required for real-time features and API calls
Operating system	Windows 10/11, Linux, or macOS
Frontend	React.js
Backend	Node.js and Express.js
Database	MongoDB
Development and testing	Visual Studio Code; Postman
Browser	Current versions of Chrome, Firefox, or Edge
Version/package tools	Git and Node Package Manager (npm)

4.1 Rationale for the MERN stack

Using JavaScript across the client and server can reduce the number of language ecosystems a team must maintain. React supports component-oriented interface construction, Express provides a routing layer for HTTP APIs, Node.js supplies the server runtime, and MongoDB offers document-oriented persistence. These characteristics align with the project's aim of implementing user profiles, posts, and interaction records. The source does not provide version numbers, hosting details, or a database schema; these should be specified in a deployment-ready technical report.

V. SYSTEM ARCHITECTURE AND AUTHENTICATION

The architecture can be understood as a client–server application. A user interacts with the client application; the client sends requests to the server; the server validates the request and coordinates data operations with the database. For authenticated requests, the project describes a JSON Web Token (JWT) flow in which the server generates a token after login and the client includes the token in subsequent requests.

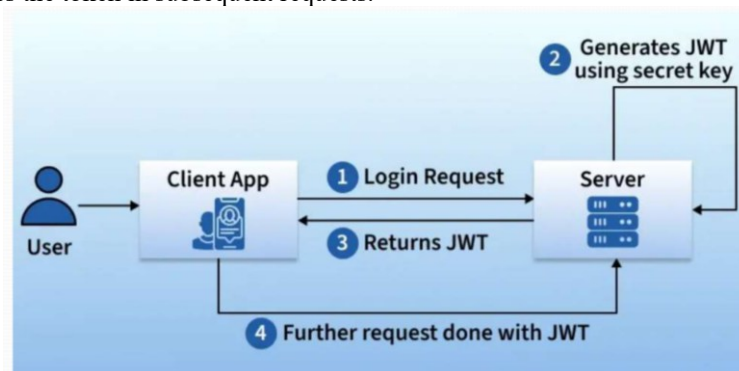


Figure 1. Authentication workflow reproduced from the project presentation

5.1 JWT-oriented workflow

1. The user submits valid registration or login credentials through the client.
2. The server validates the credentials and, following successful authentication, generates a JWT using a secret key.
3. The server returns the token to the client.
4. The client includes the JWT with subsequent requests to protected services.
5. The server validates the token and applies authorization checks before allowing protected operations.

The source diagram specifically illustrates login request, token generation using a secret key, token return, and subsequent requests carrying the JWT. It does not specify token expiry, refresh-token handling, secret rotation, password-hashing algorithm, or storage location. These implementation details should be documented and security-tested before production deployment.

5.2 Logical component responsibilities

Layer	Responsibility	Example operations
React client	Present screens and capture user actions.	Registration form, login, feed, profile, discover users, messages.
Express API	Expose endpoints and validate incoming requests.	Authentication, posts, likes, comments, follows, notifications.
Node.js runtime	Execute application logic and coordinate services.	Request processing and business rules.
MongoDB	Persist application records.	Users, posts, comments, follows, messages, notifications.
Authentication/authorization	Protect routes and restrict actions.	JWT verification and ownership checks.

VI. FUNCTIONAL MODULES

6.1 User module

The user module supports registration, login, profile management, updates to personal details, password changes, and profile-picture uploads. Authentication and authorization are identified as controls for protecting user information. A complete implementation should ensure that users can update only their own profiles unless an authorized administrator acts.

6.2 Post module

The post module supports creating, editing, deleting, and viewing text or image posts. Posts are stored and retrieved from the database and appear in a user feed for friends or followers, as described in the project. Ownership checks and validation of text and media metadata are important safeguards.

6.3 Like and comment module

The like/comment module allows users to react to posts and add comments. It maintains interaction records and displays counts. Tests should verify that duplicate likes are handled consistently, comments are associated with the correct post and author, and deleted content does not leave inconsistent interaction data.

6.4 Friend/follow module

Users can send, accept, or reject friend requests and follow or unfollow other users. The module maintains follower/following lists and connection state. The supplied material does not fully define whether friend and follow relationships are mutually exclusive or how privacy settings affect feed visibility; these are design decisions that should be formalized.

6.5 Notifications, chat, and administration

The notification module reports likes, comments, and friend requests. Messaging is marked optional and may use WebSockets or a similar technology to support real-time text, image, and emoji communication. The optional administrator

module is intended to manage users, remove posts, monitor activity, and moderate content. These optional functions should be clearly separated from features verified in the baseline build.

Module	Inputs / trigger	Expected outcome	Status in source
User	Registration or login details	Account/session established; profile can be managed	Core feature
Post	Text or image content; edit/delete action	Post appears, updates, or is removed	Core feature
Like/comment	Like action or comment text	Interaction recorded and visible	Core feature
Friend/follow	Request, acceptance, follow/unfollow	Connection list updated	Core feature
Notifications	Like, comment, or connection event	Relevant user is notified	Described feature
Chat/message	Message payload	Message delivered in real time	Optional
Admin	Moderation or management action	Users/content/activity managed	Optional

VII. SYSTEM DESIGN MODELS

7.1 Use-case model

The use-case diagram identifies the User and an optional Admin actor. User actions include registration/login, profile management, post upload, liking/commenting, uploading and viewing stories, following/unfollowing users, and viewing the feed. Admin actions include login, user management, post removal, and activity monitoring.

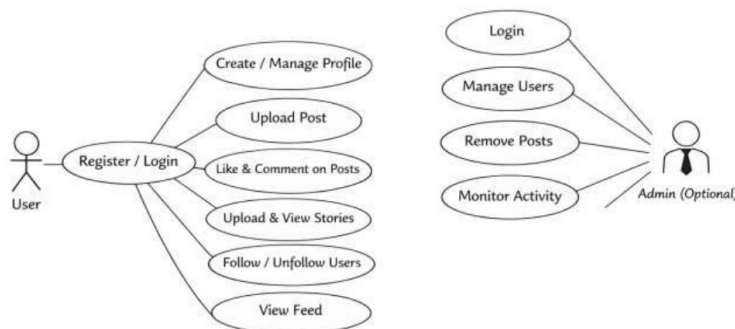


Figure 2. Use-case diagram for the Vibe social media application

7.2 Class model

The class diagram shows a User entity linked to content and interaction entities. The visible classes include Post, Story, Comment, Message, Message/Chat, Follow, and an optional Admin. Their listed attributes and operations represent the main domain concepts, including post content and media, timestamps, comments, follower/following identifiers, and sending or receiving messages.

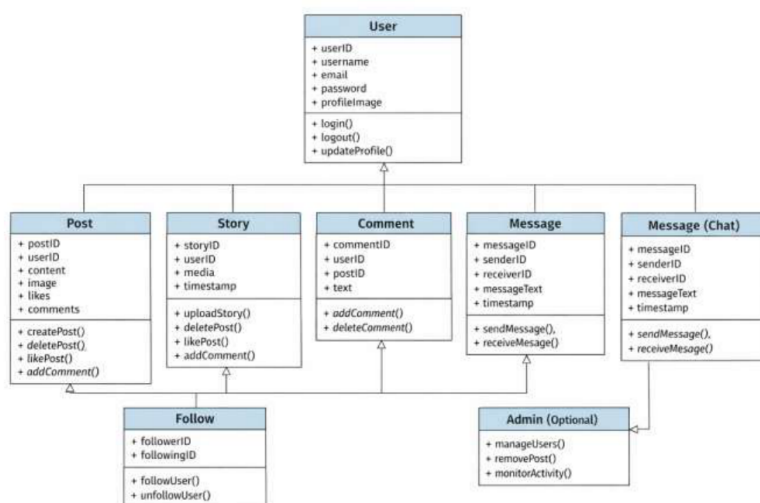


Figure 3. Class diagram showing the principal application entities

7.3 Registration sequence

The sequence diagram models communication between a user, web browser, web server, and database. The user enters registration details, submits a form, the server validates and processes registration, the user record is saved, and a response is returned to the browser for confirmation.

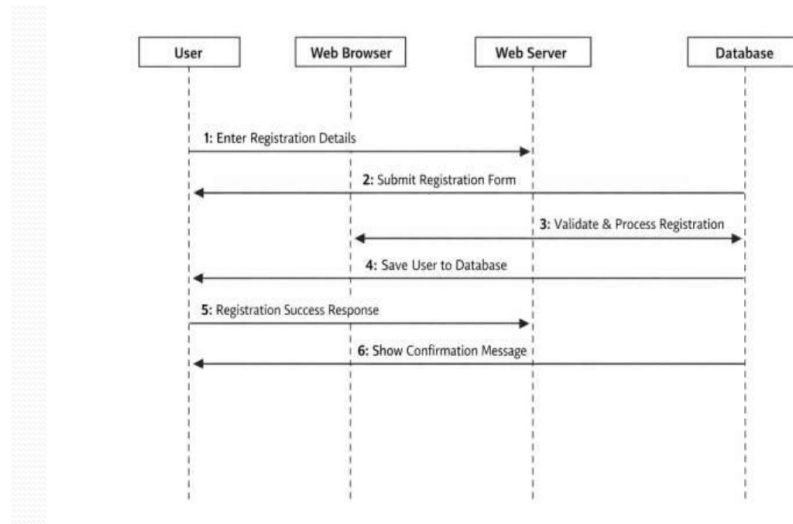


Figure 4. Registration sequence diagram

VIII. APPLICATION ALGORITHM AND DATA FLOW

The algorithm listed in the source report is a high-level interaction sequence rather than a low-level implementation algorithm. It describes the user lifecycle and the primary operations available after authentication.

6. Register or log in with valid credentials.
7. View and update the profile.
8. Create, edit, or delete posts; allow other users to like or comment.
9. Send or accept friend requests and follow other users.
10. Generate notifications for likes, comments, and requests.
11. Optionally exchange real-time messages.
12. Optionally allow administrators to manage users, posts, and content moderation.
13. Log out to end the session.

8.1 Data entities and relationships

Entity	Core information implied by the diagrams	Relationship / use
User	User ID, username, email, password, profile image	Owns profile and creates content.
Post	Post ID, user ID, content, media/image, likes, comments	Belongs to a user; receives reactions and comments.
Story	Story ID, user ID, media, timestamp	Temporary or feed-related user content; expiry is not specified.
Comment	Comment ID, user ID, post ID, text	Connects a user and a post through a comment.
Follow	Follower ID, following ID	Represents a directed user relationship.
Message / Chat	Message ID, sender ID, receiver ID, message text, timestamp	Supports user-to-user communication.
Admin	Management and moderation operations	Optional privileged actor for platform administration.

This table summarizes visible model concepts; it is not a substitute for a validated database schema. Field types, required constraints, indexes, media-storage strategy, retention rules, and referential-integrity behavior are not specified in the source material.

IX. TESTING METHODOLOGY

9.1 Unit testing

The project describes unit testing of registration, login, post creation, commenting, liking, and profile management. Valid and invalid inputs are to be used to confirm expected behavior and error handling. The report names Jest or JUnit as examples of testing tools, but does not provide test code, test-run logs, coverage percentages, or a test environment configuration.

9.2 System testing

System testing checks whether the application modules operate together. The stated coverage includes account creation, login, posting, likes, comments, profile management, invalid inputs, error handling, performance standards, and security needs. The source reports that the application was developed and tested, but does not include a detailed pass/fail execution record for each listed case.

9.3 Test cases transcribed from the source

ID	Module	Test objective	Input	Expected result
TC01	User registration	Register with valid details	Valid name, email, password	Account created successfully
TC02	User registration	Reject invalid email	Invalid email format	Error message displayed
TC03	Login	Accept correct credentials	Valid email and password	User logged in successfully
TC04	Login	Reject incorrect password	Valid email, wrong password	Login fails with error message
TC05	Post creation	Create a post	Text/image post	Post created and visible on feed
TC06	Like feature	Like a post	Click like button	Like count increases
TC07	Comment feature	Comment on a post	Comment text	Comment added successfully
TC08	Profile update	Update profile details	New profile information	Profile updated successfully

9.4 Recommended extensions to testing

To make the evaluation reproducible, future testing should report the software versions, test environment, test-data size, number of repeated runs, observed outcomes, and coverage. Additional cases should cover unauthorized requests, expired or malformed JWTs, password-handling controls, duplicate likes, empty or oversized posts, malformed uploads, authorization boundaries, concurrent updates, and recovery from database or network errors. These are recommendations for future validation, not results reported by the source.

X. RESULTS AND DISCUSSION

The project presentation states that users can register, log in, manage profiles, create/edit/delete posts, and like or comment on posts. It also describes friend/follow interactions and notifications as enabling smoother interaction. Chat/messaging and the admin dashboard are presented as optional capabilities. The application screenshots provide visual evidence of registration, login, a dark-themed application interface, discovery/follow controls, message screens, and a user profile.

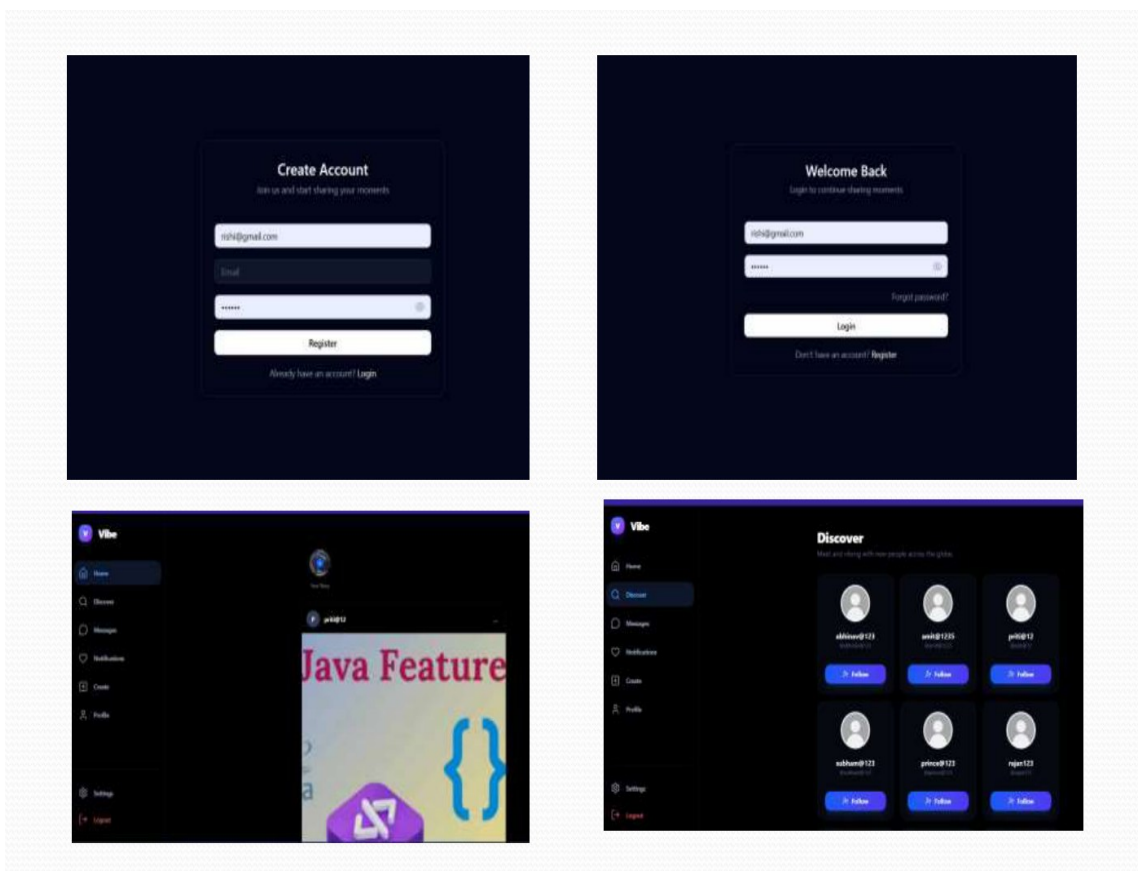


Figure 5. Application screenshots: account creation, login, feed/content view, and discovery interface

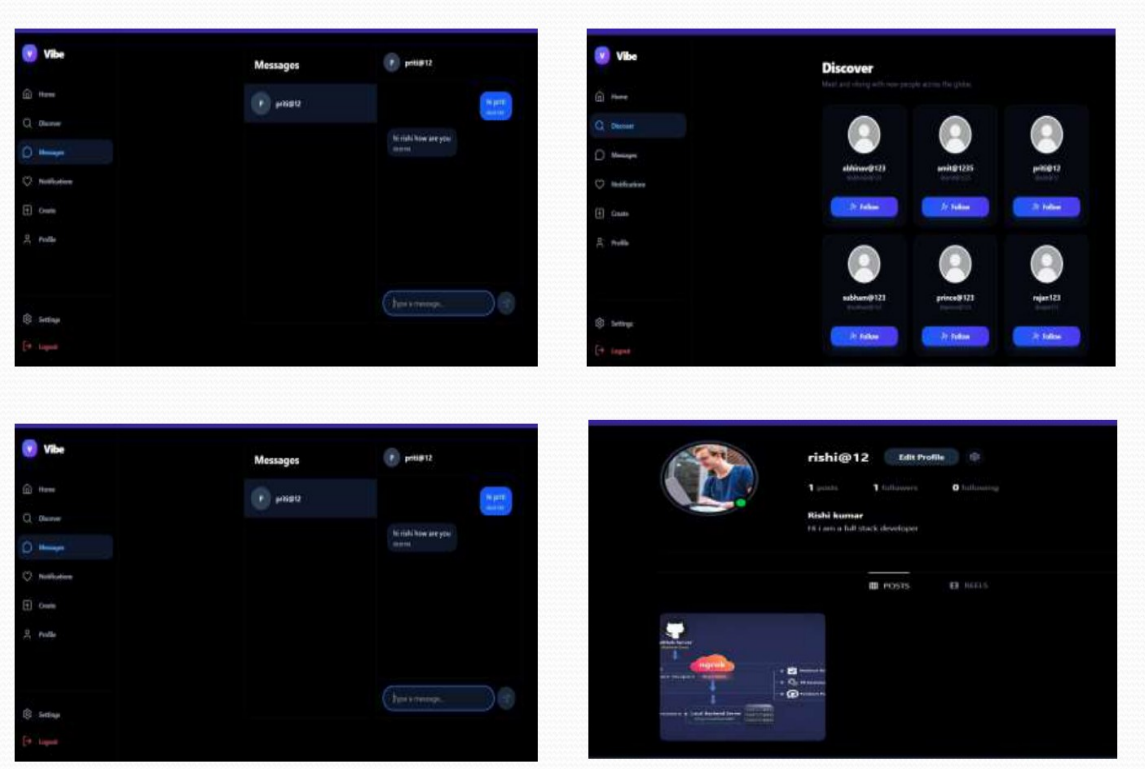


Figure 6. Additional interface screenshots: messages, user discovery, and profile view

These screenshots document interface states but do not by themselves establish usability, accessibility, security, or performance. The source includes an error graph with values visually declining from approximately 40 at the first plotted point to approximately 21 at the tenth, with a small increase at point 9. However, the graph does not identify the metric, unit, sample size, experimental conditions, or meaning of the error bars. It should therefore be treated as an unlabeled trend plot rather than as quantitative proof of improved performance.

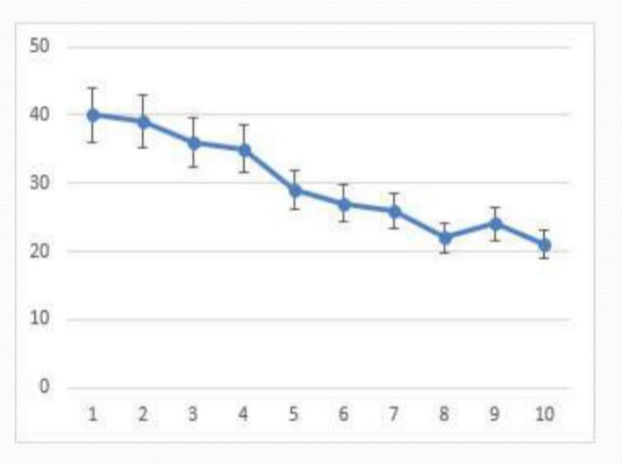


Figure 7. Error graph reproduced from the source; axis metric and experimental conditions are not identified

10.1 Classification accuracy formulation

The source describes a binary classification task for distinguishing valid content from spam or invalid content. It defines true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), and gives the standard accuracy formula:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Term	Definition in the project report
TP — True Positive	Valid content correctly identified as valid.
TN — True Negative	Spam or invalid content correctly identified as invalid.
FP — False Positive	Valid content incorrectly marked as spam/invalid.
FN — False Negative	Spam content not detected and treated as valid.

No TP, TN, FP, or FN counts are supplied, so a numerical accuracy percentage cannot be calculated from the provided report. Moreover, the project does not document a trained classifier, labeled dataset, split strategy, or evaluation protocol. Classification accuracy is therefore presented as a proposed evaluation measure, not as a measured result.

10.2 Summary of reported evidence

Evidence item	What the source supports	What remains unreported
Functional description	Core account, post, reaction, comment, and profile features are described as implemented/tested.	Detailed execution logs and acceptance criteria.
Screenshots	Visual states for registration, login, discovery, messaging, and profile screens.	Usability study, accessibility audit, device/browser matrix.
Error graph	A plotted trend decreases overall across ten points.	Metric definition, units, sample count, conditions, error-bar meaning.
Accuracy equation	Standard TP/TN/FP/FN formula is presented.	Confusion-matrix counts and actual accuracy.
Architecture diagrams	JWT flow, use cases, classes, and registration sequence are illustrated.	API specification, deployment topology, security configuration.

XI. SECURITY, PRIVACY, AND RELIABILITY CONSIDERATIONS

Security is an explicit objective of the project, particularly through authentication, authorization, data encryption, and input validation. A social platform handles personal information and user-generated content; it should therefore apply least privilege, secure password storage, protected transport, careful token management, and validation at both the client and server. The source report does not establish that every control below was implemented; they are practical recommendations for strengthening the proposed system.

- Hash passwords with a modern password-hashing algorithm; never store plaintext passwords.
- Use HTTPS in deployment and keep JWT secrets outside source control.
- Set token expiry and document logout, refresh, and revocation behavior.
- Validate and normalize all user-controlled inputs on the server; enforce limits on post size and media uploads.
- Check ownership and role permissions for profile edits, post deletion, and moderation actions.
- Apply rate limiting to authentication and high-volume endpoints, and log security-relevant events without exposing secrets.
- Define privacy controls for profile visibility, follower relationships, messages, and user deletion.
- Back up data and test restoration, failure handling, and database access controls.

XII. LIMITATIONS AND FUTURE WORK

The source report is a project review presentation rather than a full experimental paper. It gives feature descriptions and diagrams but does not supply source code, API endpoint definitions, database indexes, deployment settings, numerical load-test outcomes, or a complete reproducible evaluation. The error graph is not labeled, and the classification-accuracy formula is not accompanied by measured counts. Literature survey entries are presented with abbreviated author names and titles but without publication years, venues, page ranges, or DOIs; they are preserved below as supplied rather than expanded with guessed bibliographic information.

The presentation identifies future enhancements including real-time messaging, video sharing, live streaming, AI-based content recommendations, spam detection, improved security, performance optimization, scalability, and mobile compatibility. These should be prioritized according to user needs and evaluated through explicit acceptance criteria.

Future direction	Suggested evaluation
Real-time messaging	Message delivery latency, delivery failures, concurrent connections, reconnection behavior.
Video sharing/live streaming	Upload success, startup delay, buffering ratio, bandwidth consumption, moderation controls.
AI-based recommendation	Offline ranking metrics plus user-centered relevance and diversity evaluation.
Spam detection	Precision, recall, F1-score, confusion matrix, and robustness across content types.
Security improvements	Threat modeling, authorization tests, dependency scanning, penetration testing.
Scalability/performance	Load tests at increasing concurrent-user levels; report throughput and p50/p95/p99 latency.
Mobile compatibility	Responsive layout checks across viewport sizes, devices, browsers, and accessibility criteria.

XIII. CONCLUSION

This article consolidated the supplied Vibe project report into a research-style presentation of a MERN-based social media application. The proposed design combines a React interface, Node.js/Express services, MongoDB persistence, JWT-oriented authentication, and modules for profiles, posts, likes, comments, connections, notifications, and optional messaging and administration. The source material provides requirements, system diagrams, an eight-case functional test table, screenshots, and a classification-accuracy formula.

The project report states that core functions were developed and tested. At the same time, it does not provide enough quantitative evidence to verify performance, scalability, or content-classification accuracy. Future work should include reproducible benchmarks, complete security testing, a documented database/API design, and measured evaluation of any AI-based recommendation or spam-detection features. These additions would strengthen the evidence base and make the system easier to assess and maintain.

References

- [1] Sommerville, I. Software Engineering, 10th ed. Pearson Education. (Publication year and edition metadata as supplied in the project presentation.)
- [2] Pressman, R. S. Software Engineering: A Practitioner's Approach. McGraw-Hill. (Further bibliographic details were not supplied.)
- [3] Mitchell, T. Machine Learning. McGraw-Hill Education. (Further bibliographic details were not supplied.)
- [4] Google Developers Documentation. <https://developers.google.com>
- [5] W3Schools. <https://www.w3schools.com>
- [6] IEEE Xplore: IEEE research papers on software testing and machine learning. <https://ieeexplore.ieee.org>