GroupE4 Design and Implementation

BuzzNet

Group E4

Group Members:

LI Chun Leung 1155193164
PENG Minqi 1155191548
WONG Kwok Kam 1155192018
ZENG Bai Chuan 1155193167
ZHANG Ka Sing 1155194769

Department of Computer Science and Engineering

Version 1.3

11 May 2025

Document Revision History

Version	Revised by	Revision Date	Comments
0.1	ZENG Bai Chuan	3 Mar 2025	Initial framework.
0.2	ZENG Bai Chuan	4 Mar 2025	Updates to all sections with specifics.
0.3	Wong Kwok Kam	6 Mar 2025	Updates to all sections with more details.
0.4	LI Chun Leung	9 Mar 2025	Updates to a specific section with details.
0.5	ZHANG Ka Sing	10 Mar 2025	Add some UML diagrams and optimize file layout.
1.0	Wong Kwok Kam, ZENG Bai Chuan,	10 Mar 2025	Modification and checking.
1.1	Wong Kwok Kam	27 Mar 2025	Major updates based on professor's suggestions.
1.2	Zeng Bai Chuan	12 April 2025	Updates based on discussion.
1.3	Wong Kwok Kam	11 May 2025	Finalize the document.

1 Introduction and Goals

This document outlines the design and implementation details for ***BuzzNet***, a public social media platform designed for users to share text content. It serves as a comprehensive guide for developers and stakeholders, providing a clear understanding of the system's architecture, components, and functionalities.

This Design and Implementation document aims for clarity and conciseness, focusing on the essential aspects of BuzzNet's design.

1.1 Requirements Overview

Main features

- User Authentication and Management (user registration, login, etc)
- Content Creation and Management (post creation and deletion, etc)
- Social Interaction (likes, dislikes, comments)
- Platform Moderation and Security (admin control)

Goals

- **Comprehensive Design:** To provide a detailed blueprint for the development of BuzzNet.
- **Clear Communication:** To ensure all stakeholders understand the system's architecture and functionality.
- **Maintainability:** To create a design that is easy to understand, modify, and extend.

1.2 Quality Goals

Nr.	Quality	Motivation
1	Performance	The system should be responsive and provide a smooth user experience.
2	Scalability	The architecture should be scalable to handle a large number of users and posts.
3	Security	The system should be secure and protect user data from unauthorized access.
4	Testability	The codebase should be well-tested to ensure its quality and reliability.
5	Maintainability	The code should be well-organized and easy to understand and modify.
6	Availability	The system should be highly available with minimal downtime.

7	Usability	The interface should be intuitive and accessible to users of all abilities.

1.3 Stakeholders

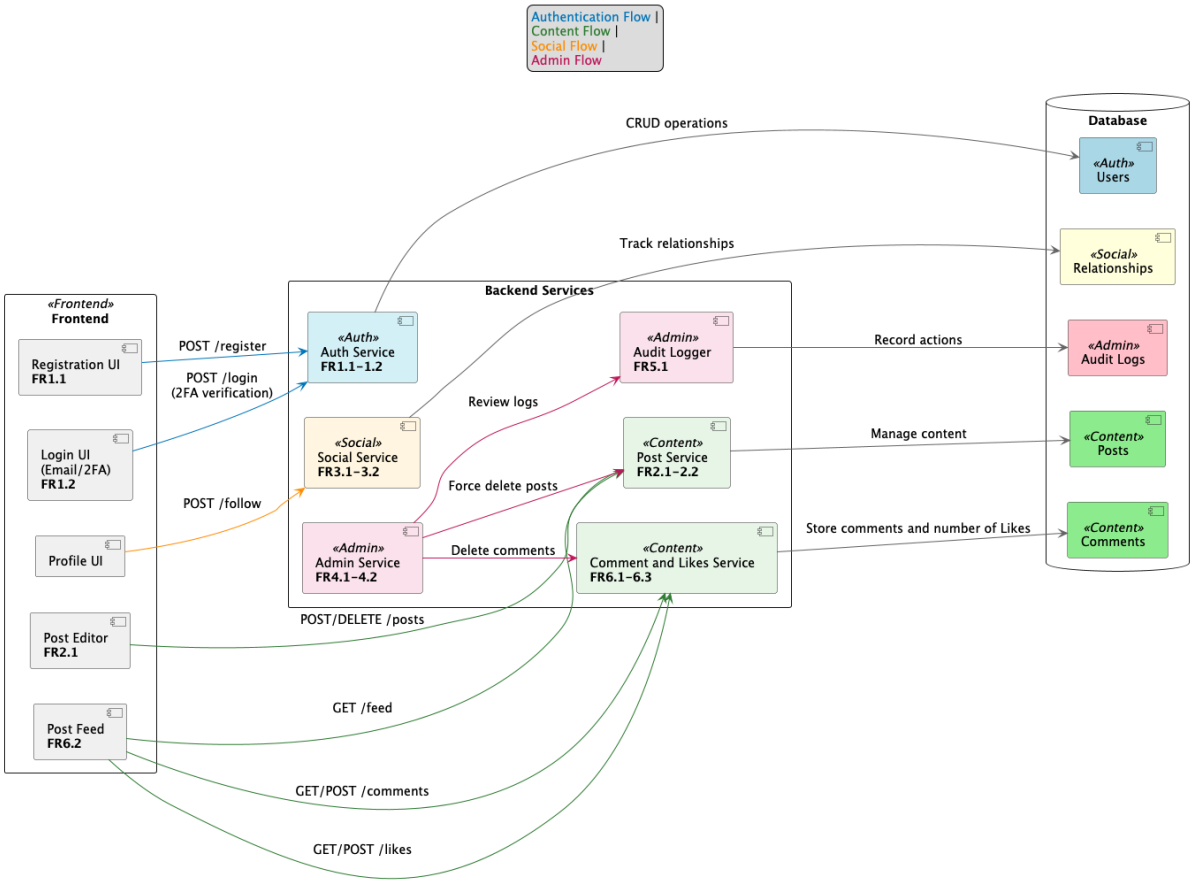
Role/Name	Goal/Boundaries
Developers	● Implement and maintain BuzzNet according to requirements ● Follow established coding standards and architectural patterns ● Create maintainable, secure, and scalable code ● Perform testing (at unit, integration, and system levels) ● Collaborate with stakeholders to resolve technical issues ● Document code and system architecture ● Participate in code reviews and quality assurance ● Implement security best practices throughout the codebase ● Optimize performance and address technical debt
Users	● Register and maintain personal accounts on the platform ● Create, view, and interact with content (posts, comments, likes) ● Manage their own posts and comments (create and delete) ● Cannot access administrative functions (delete posts)
Administrators	● Moderate content across the platform ● Delete inappropriate content that violates guidelines ● Access audit logs for security and compliance purposes ● Cannot delete user posts without justification ● Must adhere to privacy policies when accessing user data ● Responsible for maintaining platform integrity

2 Architecture

2.1 Architecture Overview

Buzznet adopts a client-server architecture, separating the frontend (client) and backend (server) for scalability and maintainability.

- **Frontend:** A web-based application built with React, providing the user interface for interacting with BuzzNet.
- **Backend:** A RESTful API built with Node.js and Express, handling data storage, business logic, and security.
- **Database:** MongoDB, a NoSQL database, is used for storing user data, posts, comments, and other relevant information.



2.2 Technical Constraints

No.	Constraint	Background and/or motivation
TC1	Implemented with modern JavaScript and language-agnostic API	The application should be built with JavaScript (ES6+) and Node.js. The API should be language and framework agnostic, allowing clients to be implemented in any programming language.
TC2	All third-party software must be available under compatible open source licenses and installable via a package manager.	Standardized dependency management through npm ensures proper versioning and easy installation across development environments. Open source licensing protects all stakeholders by clearly defining usage rights and obligations.
TC3	OS independent	The application should be runnable on all 3 major operating systems (Mac OS, Linux and Windows).
TC4	Limited Resource Consumption	The application must operate efficiently on

		standard university lab computers and student laptops without requiring specialized hardware such as discrete graphic cards.
TC5	Version Control and Collaboration	All code must be maintained in a shared GitHub repository with clear branching strategy and pull request workflow.
TC6	Testing Requirements	Core functionality must include automated tests with minimum 70% code coverage.
TC7	Security Baseline	The application must implement basic security measures including input validation, authentication, and protection against common web vulnerabilities.

3. System Scope and Context

- **Users:** Registered users who can create content and interact with posts.
- **Administrators:** Users with elevated privileges for content moderation.
- **External Systems:** (Future consideration) Integration with third-party services for image hosting, content moderation, or social sharing.

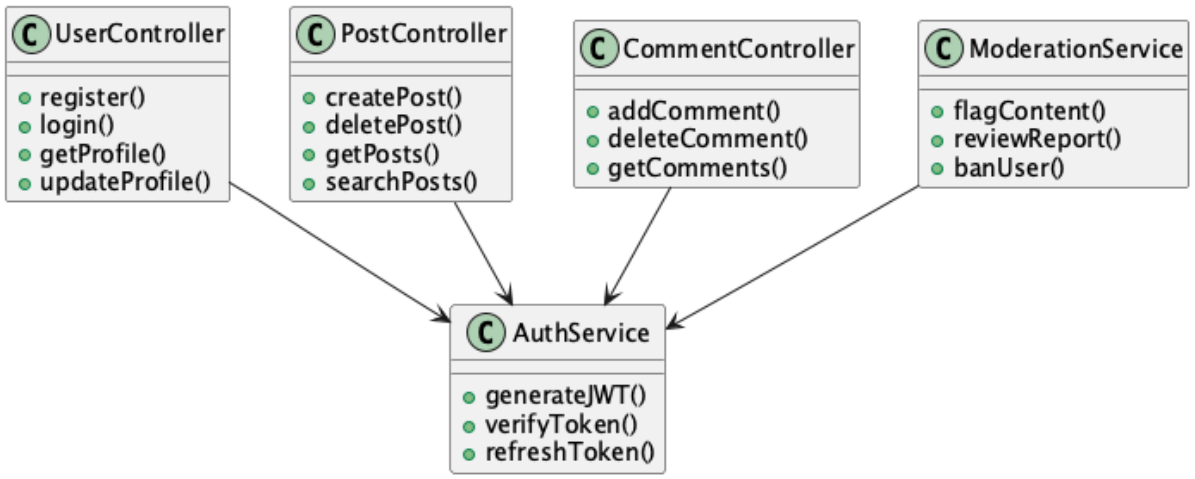
4. Building Block View

4.1 Frontend

- **Components:**
    - PostList: Displays a list of posts.
    - PostForm: Allows users to create or edit posts.
    - User interaction: Allow users to like/dislike the posts.
    - CommentSection: Displays comments for a specific post and allows users to add new comments.
    - UserAuthentication: Handles user registration, login, and logout.
    - UserProfile: Displays user information.
- **Technology:** HTML, React, Vite and Tailwind CSS.

4.2 Backend

- **Class Diagram (core business logic):**

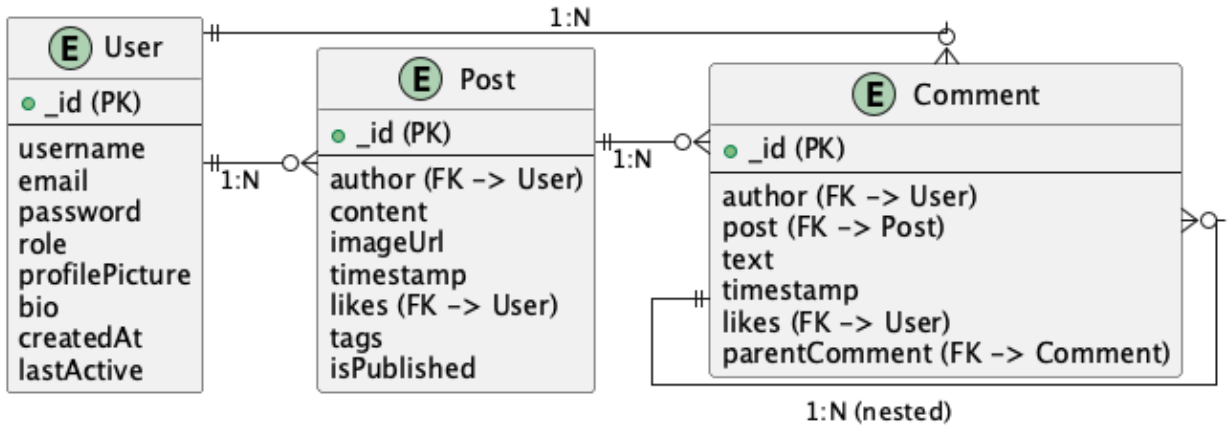


(Note that *ModerationService* and *searchPost()* will not be implemented at the early stage.)

- **Modules:**
  - User Management: Handles user authentication, registration, and profile viewing.
  - Post Management: Manages creation, retrieval, and deletion of posts.
  - Comment Management: Manages the creation, retrieval, and deletion of comments.
  - API Layer: Exposes endpoints for frontend communication.
- **Technology:** Node.js, Express, Mongoose (for MongoDB interaction), and JSON Web Token (JWT) for authentication.

4.3 Database

ER Diagram (for reference only):



(Note that *nested comment* may not implemented)

- **Collections:**
    - users: Stores user information (username, email, hashed password, etc.).
    - posts: Stores post data (content, author, timestamps, etc.).
    - comments: Stores comment data (text, author, post ID, timestamps, etc.).
- **Database Design**: MongoDB with data validation rules, and potentially connection pool.

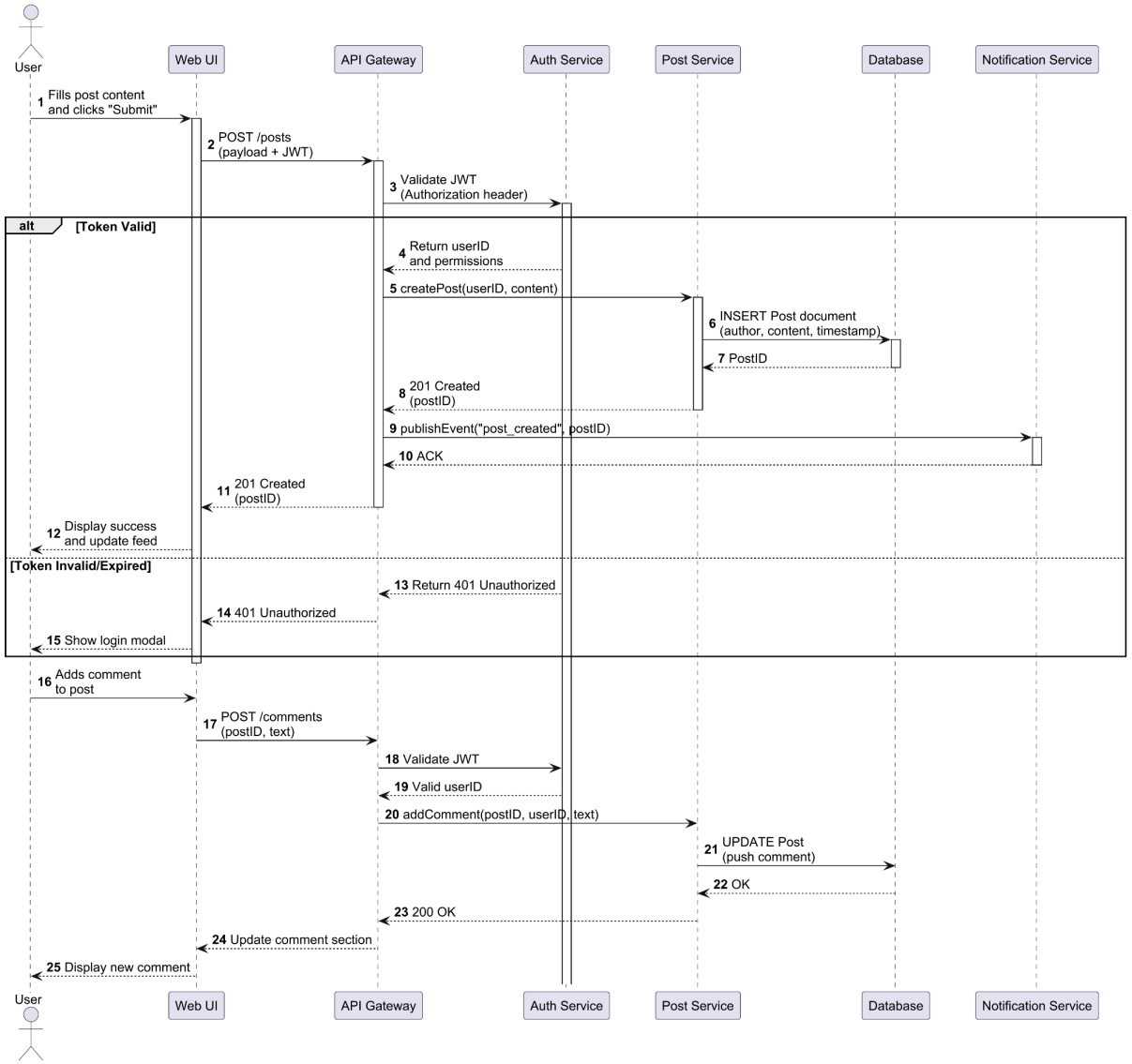
Schema Design

JavaScript

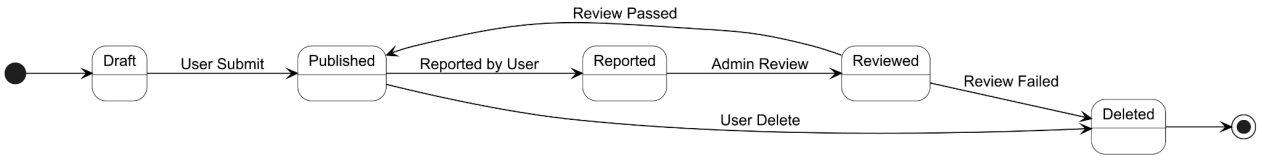
```
// User Schema
const userSchema = new mongoose.Schema({
   username: { type: String, required: true, unique: true },
   email: { type: String, required: true, unique: true },
   password: { type: String, required: true }, // Store hashed password
   role: { type: String, default: 'user' }, // e.g., user, admin
  },
  {
   timestamp: true,
});

// Post Schema
const postSchema = new mongoose.Schema({
   author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
   content: { type: String, required: true },
   likes: { type: Number, default: 0, min: 0 },
   likedBy: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User', }],
   dislikes: { type: Number, default: 0, min: 0 },
   dislikedBy: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
   comments: [commentSchema],
   commentCount: { type: Number, default: 0, min: 0 }
  },
  {
   timestamp: true,
});

// Comment Schema
const commentSchema = new mongoose.Schema({
   author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
   username: { type: String, required: true, },
   text: { type: String, required: true },
   likes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
   dislikes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
  },
  {
   timestamp: true,
});
```

5 Runtime View

5.1 **Sequence Diagram**: User Creates Post with Comment Interaction



5.2 State Diagram (Post Lifecycle)



*(Note that diagrams are **for reference only**)*

6 API Design

BuzzNet exposes a RESTful API for communication between the frontend and backend. API endpoints should follow a logical and consistent structure.

6.1 User Management Endpoints

- POST /api/users/register: Register a new user
- POST /api/users/login: Authenticate a user (login)
- GET /api/users/profile: Get user profile (of the logged-in user) (Protected)

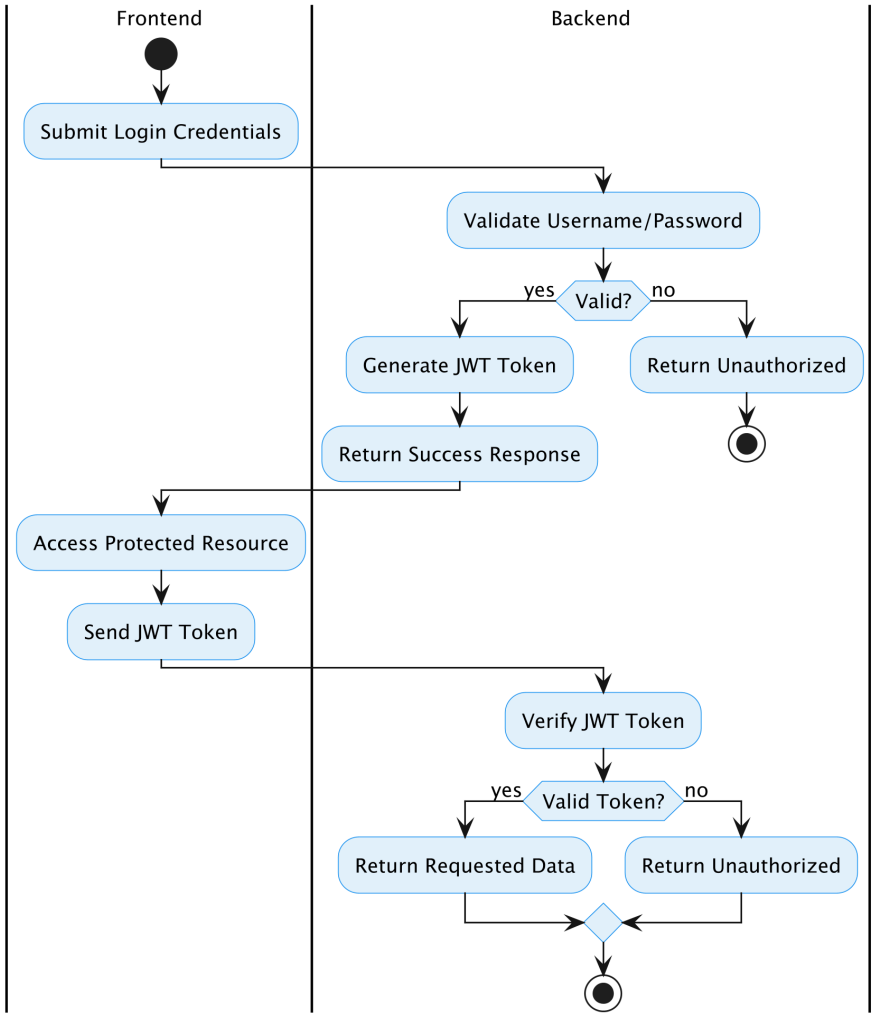
6.2 Post Management Endpoints

- GET /api/posts/: Get all posts
- POST /api/posts/: Create a new post (Protected)
- GET /api/posts/:id: Get single post by ID
- DELETE /api/posts/:id: Delete post by ID (Protected)
- PATCH /api/posts/:id/like: Like a post (Protected)
- PATCH /api/posts/:id/dislike: Dislike a post (Protected)
- POST /api/posts/:id/comments: Add a comment to a post (Protected)
- PATCH /api/posts/:id/comments/:commentId/like: Like a comment on a post (Protected)
- PATCH /api/posts/:id/comments/:commentId/dislike: Dislike a comment on a post (Protected)

7 Security

7.1 Authentication

Authentication & Authorization Flow:



JWT Implementation:

- JSON Web Tokens used for user authentication
- Tokens contain encoded user information and permissions
- Access tokens expire after 1 hour
- Refresh tokens allow obtaining new access tokens without re-login
- Tokens invalidated during logout (a token blacklist/database should be implemented)

7.2 Authorization

Role-Based Access Control (RBAC):

- Two primary roles: USER and ADMIN
- Each role has specific permissions:
    - USER: Basic platform access and content creation, interaction
    - ADMIN: Full system access and content management
- API endpoints verify user roles before processing requests
- Frontend conditionally displays features based on user role

7.3 Data Validation

Validation Strategy:

- Frontend validation provides immediate user feedback
- Backend validation ensures data integrity regardless of source
- Input sanitization prevents cross-site scripting (XSS)
- Parameterized queries prevent SQL injection

7.4 Password Security

Password Protection:

- Bcrypt algorithm with unique salt for each password
- Password requirements:
    - Minimum 8 characters
    - Mix of uppercase, lowercase and numbers
- Account lockout after multiple failed attempts
- Password Strength Indicator: Password strength bar will be displayed to remain users creating a stronger password.

7.5 Additional Security Measures

API Protection:

- Rate limiting prevents brute force attacks
- HTTPS required for all API communication
- CORS policies restrict unauthorized domain access

Data Protection:

- Sensitive data encrypted in database
- All network traffic encrypted with TLS
- Regular security audits and updates

8 User Interface (UI) and User Experience (UX)

8.1 Design System

- Visual Consistency: Implementation of a cohesive design system with consistent typography, color palette, spacing, and component styling.
- Responsive Design: The UI is designed to be responsive and adapt to different screen sizes.
- Design Tokens: Documentation of reusable design elements (colors, fonts, spacing) to maintain consistency across the application.

8.2 User Interaction

- Intuitive Navigation: The application provides clear and intuitive navigation for users to easily find and access content.
- Feedback Mechanisms: Clear visual and possibly auditory feedback when users perform actions.
- Error Handling: User-friendly error messages and recovery paths.
- Loading States: Appropriate indicators for processing actions and loading content.

8.3 Accessibility

- Accessibility: The UI is designed to be accessible to users with disabilities, following accessibility guidelines (WCAG).

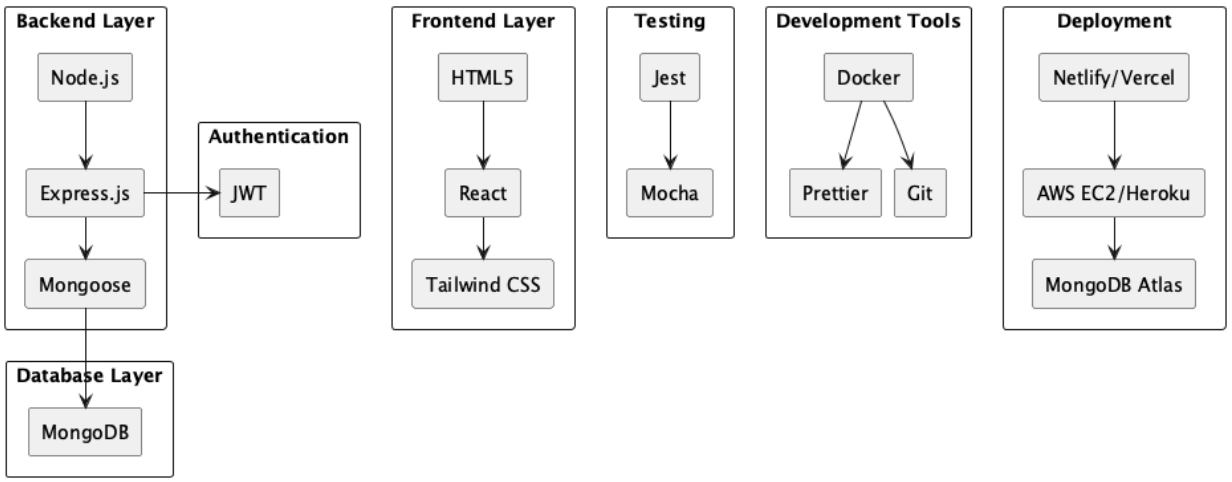
8.4 Documentation

- Design Guidelines: Documentation for developers and designers to maintain UI/UX consistency.

9 Technology Stack

- **Frontend:** HTML, React, Tailwind CSS
- **Backend:** Node.js, Express, Mongoose
- **Database:** MongoDB
- **Authentication:** JWT
- **Testing:** Jest
- **Development environment:** Docker, VS Code
- **Code formatting:** Prettier
- **Version Control:** Git

The layered technology stack diagram:



(For reference only)

10 Deployment

- The frontend can be deployed on a static hosting service like Netlify or Vercel.
- The backend can be deployed on a cloud platform like AWS, Google Cloud, or Azure.

11 Future Considerations

- Integrate Topics and Votes function.
- Implement real-time features using WebSockets for live updates and notifications.
- Integrate with third-party services for image hosting and content moderation.
- Add support for more media types (e.g., videos, audio).
- Provide a document for designers to install all the dependencies needed.