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 1.0

10 March 2025

Document Revision History

Version	Revised by	Revision Date	Comments
0.1	ZENG Bai Chuan	3 March 2025	Initial framework.
0.2	ZENG Bai Chuan	4 March 2025	Updates to all sections with specifics.
0.3	Wong Kwok Kam	6 March 2025	Updates to all sections with more details
0.4	LI Chun Leung	9 March 2025	Updates to a specific section with details
0.5	ZHANG Ka Sing	10 March 2025	Add some UML diagrams and optimize file layout
1.0	Wong Kwok Kam, ZENG Bai Chuan,	10 March 2025	Final modification and checking

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 and image-based 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.

1.1Requirements Overview

Main features

- User Authentication and Management (user registration, login, etc)
- Content Creation and Management (post creation, anonymous posting, etc)
- Social Interaction (likes, following, comments, repost)
- Content Discovery (search, categorization)
- Platform Moderation and Security (admin control, reporting system)

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

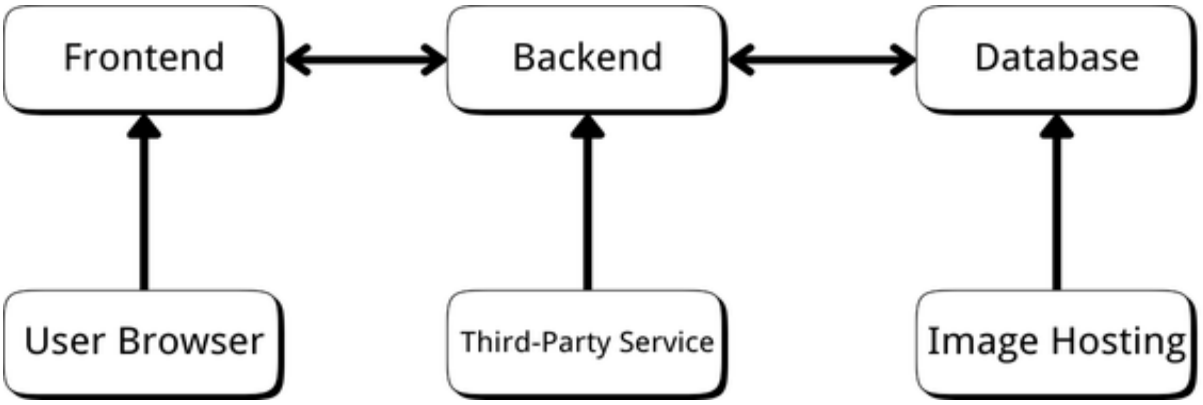
Role/Name	**Goal/Boundaries**
Developers	● Implement and maintain Buzznet according to requirements ● Follow established coding standards and architectural patterns ● Create maintainable, secure, and scalable code ● Perform testing at unit, integration, and system levels ● Collaborate with stakeholders 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) ● Follow other users and discover relevant content ● Report inappropriate content to administrators ● Maintain privacy through anonymous posting option ● Search for content using keywords and categories ● Manage their own posts and comments (create, edit, delete) ● Receive notifications about relevant activity ● Cannot access administrative functions or other users' private data
Administrators	● Moderate content across the platform ● Review and act on reported posts/comments ● Delete inappropriate content that violates guidelines ● Access audit logs for security and compliance purposes ● Monitor system performance and usage metrics ● Cannot modify or delete user content 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

	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 development	The application should be compilable 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.
TC5	Version Control and Collaboration	All code must be maintained in a shared Git repository with clear branching strategy and pull request workflow.
TC6	Testing Requirements	Core functionality must include automated tests with minimum 60% 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, interact with posts, and manage their accounts.
- **Administrators:** Users with elevated privileges for content moderation, user management, and system configuration.
- **External Systems:** (Future consideration) Integration with third-party services for image hosting, content moderation, or social sharing.

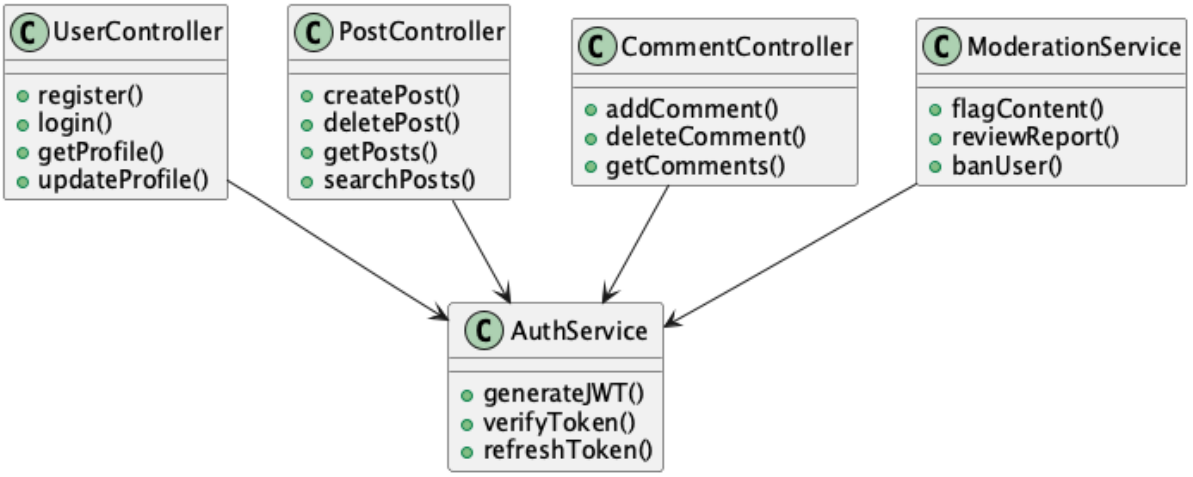
4. Building Block View

4.1 Frontend

- **Components:**
    - PostList: Displays a list of posts.
    - PostForm: Allows users to create or edit new posts.
    - User interaction: Allow users to like 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 and allows users to edit and manage their profiles.
    - SearchBar: Allows users to search posts by keywords or categories
    - CategoryBar: Displays and filters content by categories
- **Technology:** HTML, React, Tailwind CSS, and potentially a state management library like Redux or Context API.

4.2 Backend

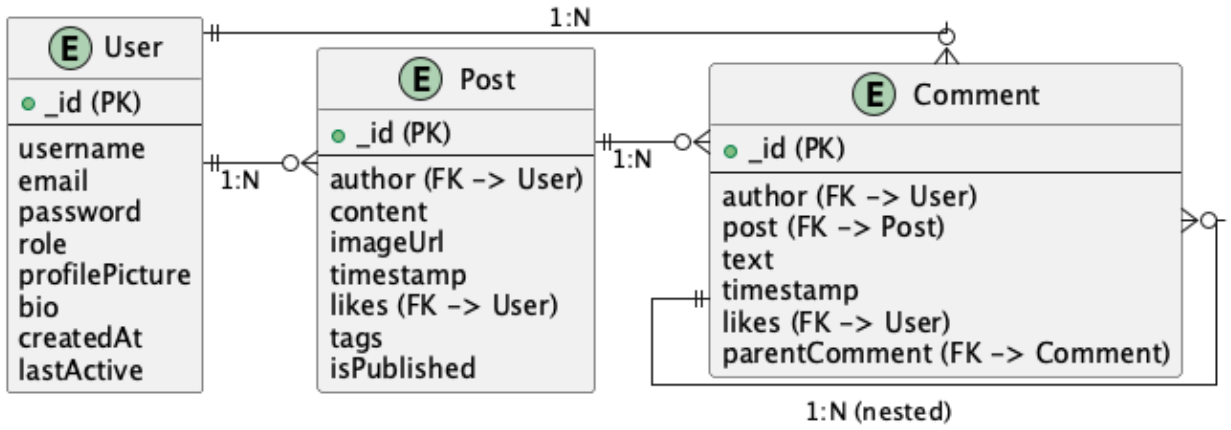
- **Class Diagram (core business logic):**



- **Modules:**
    - User Management: Handles user authentication, registration, and profile management.
    - Post Management: Manages creation, retrieval, modification, and deletion of posts.
    - Comment Management: Manages the creation, retrieval, modification, and deletion of comments.
    - Content Moderation: Provides tools for administrators for content review and user management.
    - Search Service: Processes search queries and returns relevant results
    - Category Management: Organizes and maintains content categories
    - 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:



- **Collections:**
  - users: Stores user information (username, email, password, etc.).
  - posts: Stores post data (text, images, author, timestamps, etc.).
  - comments: Stores comment data (text, author, post ID, timestamps, etc.).
  - categories: Maintains category definitions and relationships
  - notifications: Tracks user notifications and their status
- **Database Design**: MongoDB with appropriate indexes for query optimization and data validation rules

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 },
  role: { type: String, enum: ['user', 'admin'], default: 'user' },
  profilePicture: { type: String },
  bio: { type: String },
  createdAt: { type: Date, default: Date.now },
  lastActive: { type: Date }
});



// Post Schema
const postSchema = new mongoose.Schema({
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  content: { type: String, required: true },
  imageUrl: { type: String },
  timestamp: { type: Date, default: Date.now },
  likes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
```

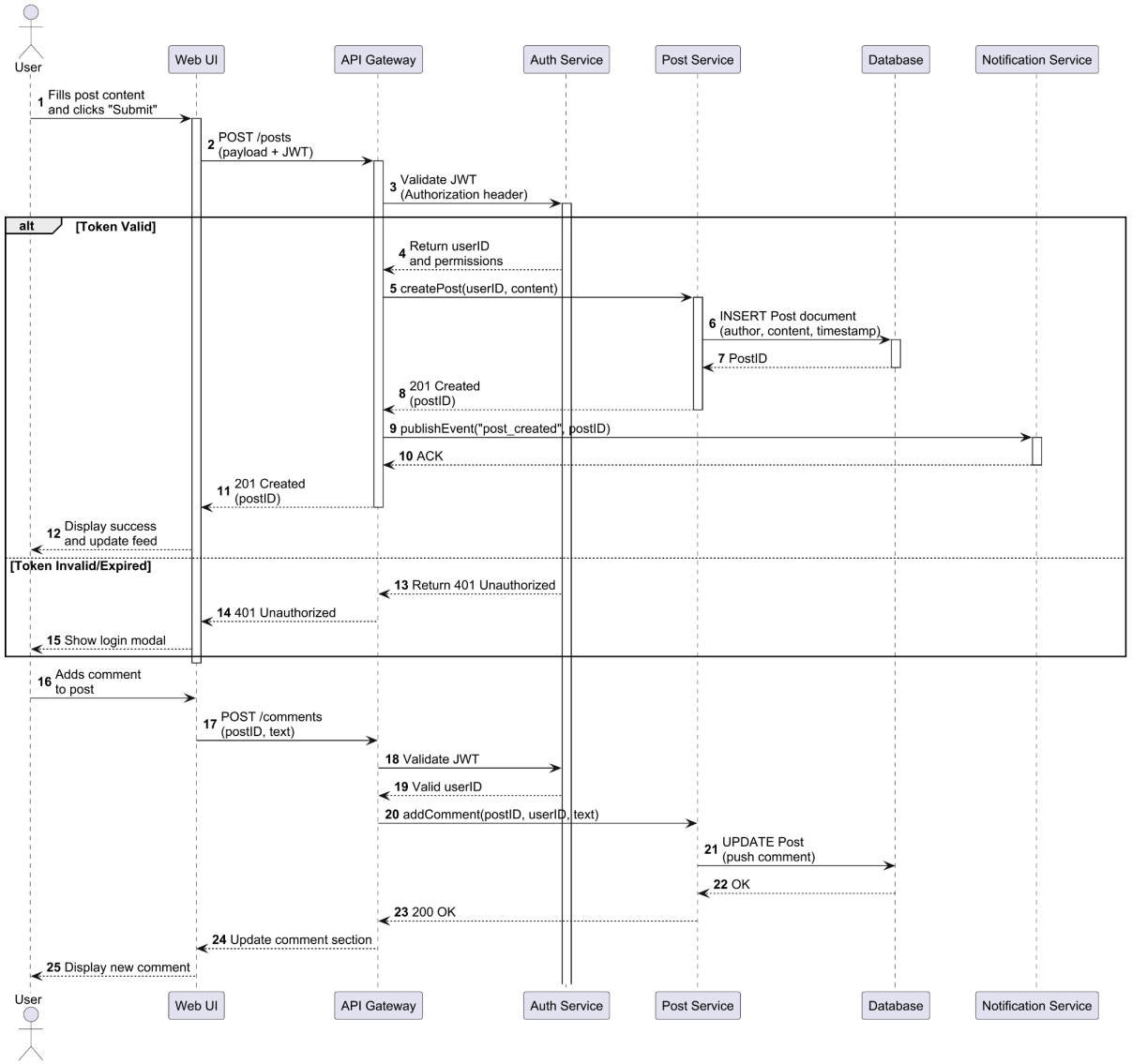
```
  tags: [{ type: String }],
  isPublished: { type: Boolean, default: true }
});


// Comment Schema
const commentSchema = new mongoose.Schema({
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  post: { type: mongoose.Schema.Types.ObjectId, ref: 'Post', required: true },
  text: { type: String, required: true },
  timestamp: { type: Date, default: Date.now },
  likes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
  parentComment: { type: mongoose.Schema.Types.ObjectId, ref: 'Comment' } // For nested
comments
});
```

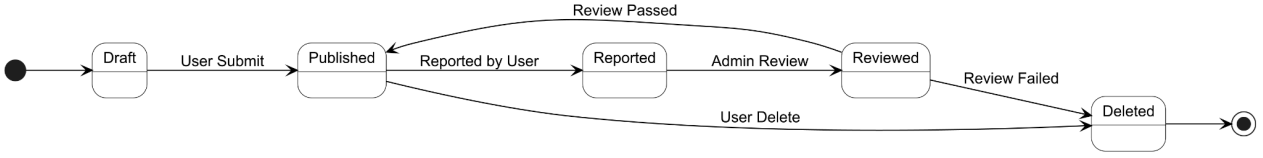
Rollback Mechanism: All schema changes support up/down methods for reversibility.

5 Runtime View

5.1 Sequence Diagram: User Creates Post with Comment Interaction



5.2 State Diagram (Post Lifecycle)



6 API Design

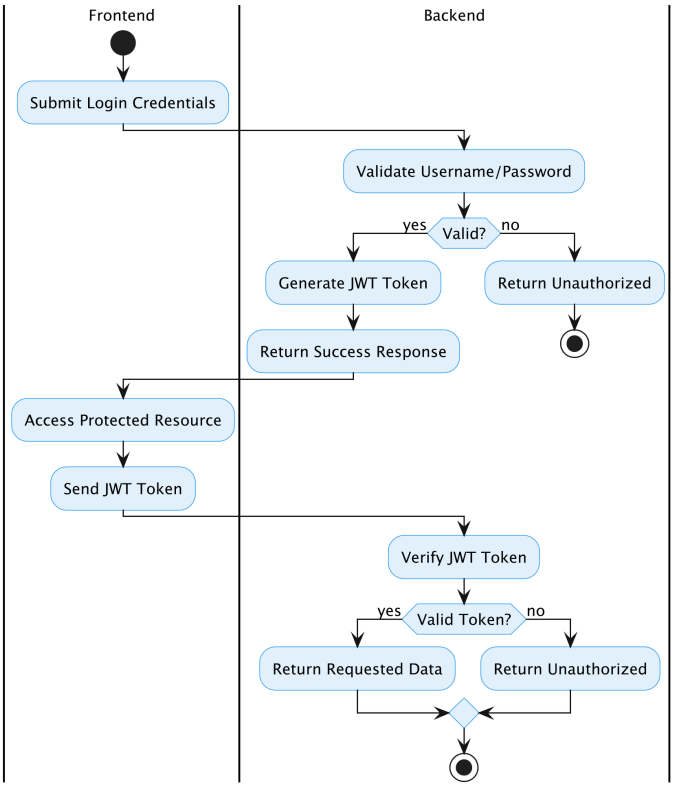
Buzznet exposes a RESTful API for communication between the frontend and backend. API endpoints should follow a logical and consistent structure.

- **User management endpoints**
  - POST /users/register: Register a new user
  - POST /users/login: Log in an existing user
  - GET /users/{id}: Get user profile information
  - PUT /users/{id}: Update user profile information
  - GET /users/{id}/posts: Get all posts by a specific user
  - PUT /users/{id}/profilePicture: Update user profile picture
- **Post management endpoints**
  - GET /posts: Get all posts (with pagination)
  - POST /posts: Create a new post
  - GET /posts/{id}: Get a specific post
  - PUT /posts/{id}: Update a post
  - DELETE /posts/{id}: Delete a post
  - PUT /posts/{id}/like: Like/unlike a post
  - GET /posts/tags/{tag}: Get posts by tag
  - GET /posts/search?keyword={searchTerm}: Search posts by keyword
- **Comment management endpoints**
  - GET /posts/{postId}/comments: Get all comments for a post
  - POST /posts/{postId}/comments: Add a new comment to a post
  - GET /comments/{id}: Get a specific comment
  - PUT /comments/{id}: Update a comment
  - DELETE /comments/{id}: Delete a comment
  - POST /comments/{id}/replies: Add a reply to a comment
  - PUT /comments/{id}/like: Like/unlike a comment
- **Authentication endpoints**
  - POST /auth/login: Authenticate user and issue JWT
  - POST /auth/refresh: Refresh an existing JWT before expiration
  - POST /auth/logout: Invalidate current JWT
  - GET /auth/me: Get current authenticated user details
  - POST /auth/password/reset-request: Request password reset
  - POST /auth/password/reset: Reset password with token
- **Administration endpoints (restricted)**
  - GET /admin/users: Get all users (admin only)
  - PUT /admin/users/{id}/role: Update user role (admin only)
  - GET /admin/posts/reported: Get reported posts (admin only)

7 Security

6.1 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

6.2 Authorization

Role-Based Access Control (RBAC):

- Three primary roles: USER, MODERATOR, and ADMIN
- Each role has specific permissions:
    - USER: Basic platform access and content creation
    - MODERATOR: Content moderation capabilities
    - ADMIN: Full system access and user management

- API endpoints verify user roles before processing requests
- Frontend conditionally displays features based on user role

6.3 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
- File uploads validated for type, size, and content

6.4 Password Security

Password Protection:

- Bcrypt algorithm used for secure password hashing
- Unique salt generated for each password
- Work factor of 10 balances security and performance
- Password requirements:
    - Minimum 8 characters
    - Mix of uppercase, lowercase, numbers, and symbols
- Account lockout after multiple failed attempts
- Password strength Awareness: Password strength bar will be displayed to remain users creating a stronger password.
- Forgot Password: Password could be reset via a link sent to users' registered email.

6.5 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)

7.1 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.

7.2 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.

7.3 Internationalization & Localization

- Multi-language Support: Interface adaptability for different languages.
- Cultural Considerations: Design elements that respect cultural differences and preferences.
- Accessibility: The UI is designed to be accessible to users with disabilities, following accessibility guidelines (WCAG).

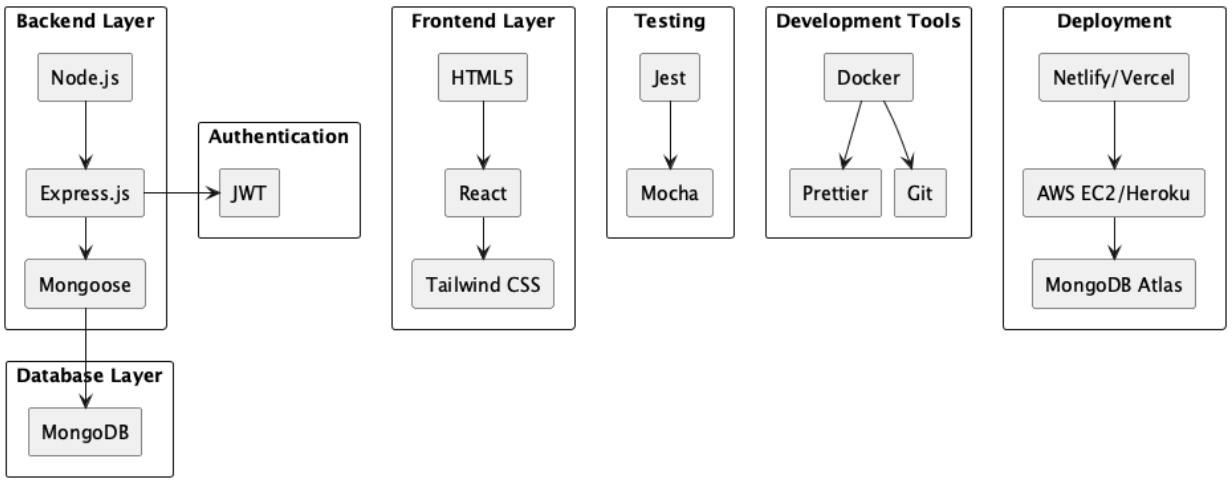
7.4 Documentation

- 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, Mocha
- **Development environment:** Docker
- **Code formatting:** Prettier
- **Version Control:** Git

The layered technology stack diagram (ver.1):



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.