WriteUp - Technical Design Document

Table of Contents

Introduction and System Overview

Introduction

WriteUp is a modern Blogging Platform designed to enable users to create blogs, read and interact with people’s blogs. The platform offers a seamless user experience with features updates, user authentication.

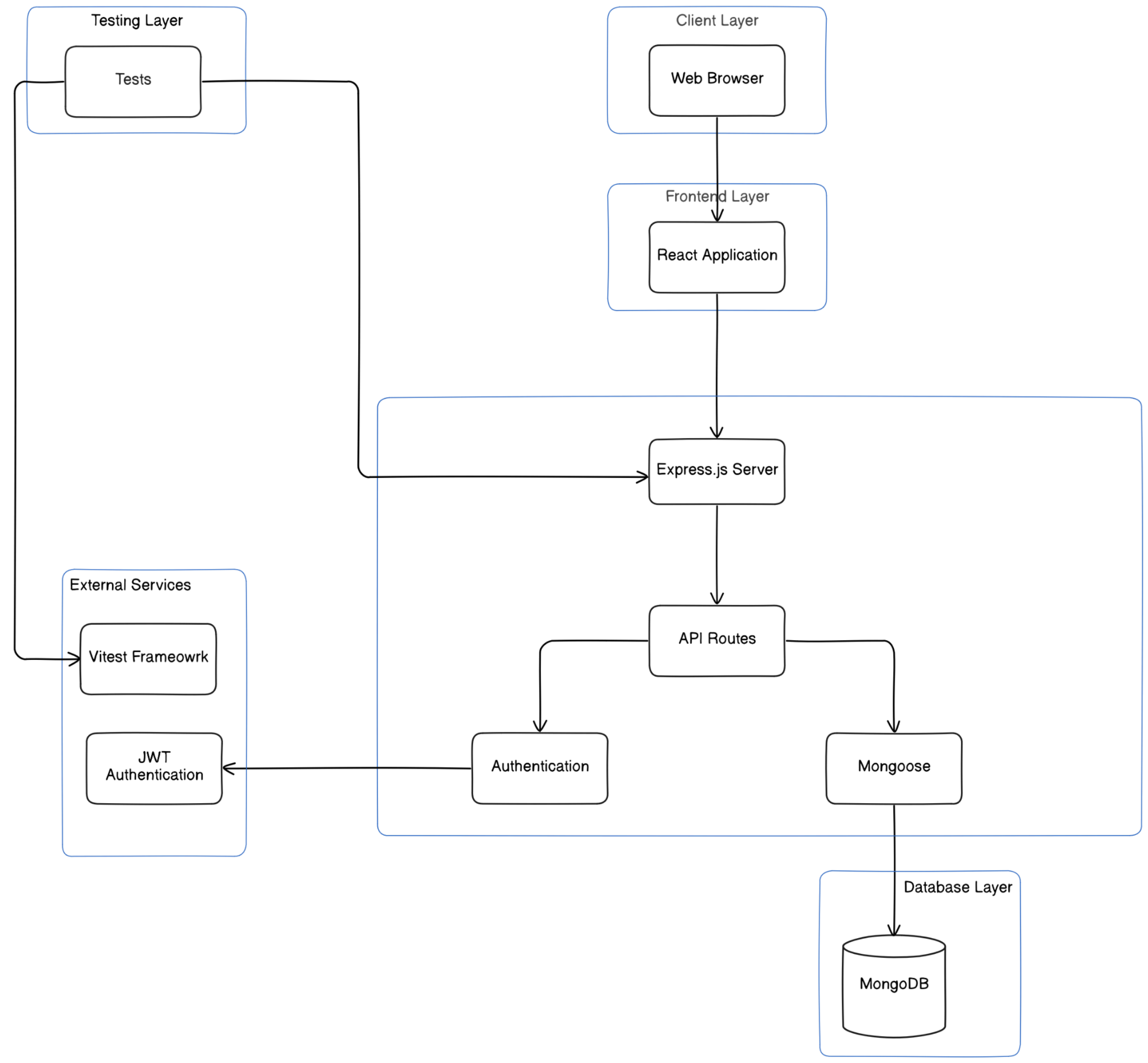
This design document provides a comprehensive overview of the project's architecture, components, technology stack, and design decisions. It aims to serve as a guide for developers, stakeholders, and contributors.

System Overview

WriteUp is built using the MERN stack (MongoDB, Express.js, React, Node.js), utilizing modern web development practices. The application is structured to provide scalability, maintainability, and a responsive user experience across devices.

Architecture

High-Level Architecture



The system follows a three-tier architecture, comprising:

1. **Frontend**: Developed with React.js, responsible for the client-side user interface and interactions.
2. **Backend API**: Built with Express.js and Node.js, handling server-side logic, API endpoints, authentication, and business logic.
3. **Testing:** Developed using JS Testing library Vitest.
4. **Database**: Utilizes MongoDB for storing user data and related information.

Backend Design

Technologies Used

- **Node.js**: JavaScript runtime environment.
- **Express.js**: Web application framework for building APIs.
- **MongoDB:** NoSQL database for data storage.
- **Mongoose**: OBM (Object Document Model) library for MongoDB.
- **JWT**: JSON Web Tokens for authentication.

Project Structure

- **client/**:Client Side Logic of WriteUp.
- **server/:** Server Side Logic of WriteUp

Backend Design - API, Database Schema and Middlewares

API Design

The backend exposes RESTful API endpoints categorized under:

- **Auth User (`/api`)**
  - `POST /auth/signup`: User Signup.
  - `POST /auth/signin`: User login and JWT token issuance.
  - `GET /auth/me`: Retrieves User info from cookie.
  - `GET /auth/verify/:token`:Email based Verification.
  - `POST /auth/resend-verification`:Resend Email Verification.
  - `POST /auth/signout`:User Signout.
  - `GET /user/posts`: Get all user posts
- **Comment (`/api`)**
  - `POST /:postId/comment`: Create new Comment
  - `GET /:postId/comment`: Get all Comments
- **Admin (`/api/posts`)**
  - `GET /search`: Search Posts
  - `POST /`: Create new posts
  - `GET /:id` : Get post by Id
  - `PUT /:id` :Edit Post By Id.
  - `DELETE /:id` : Delete Post By Id
  - `POST /:postId/like :`Like a Post.
- **Admin (`/api/category`)**
  - `POST /`: Create Category
  - `GET /`: Get all Categories
  - `GET /:id` : Get Category by Id
  - `PUT /:id` :Edit Category by Id.
  - `DELETE /:id` : Delete Post By Id

Database Schema

User Model

Fields:

- **firstName**
- **lastName**
- **username**
- **email**
- **password**
- **bio**
- **website**
- **socialLinks**
- **role**
- **emailVerificationToken**
- **emailVerificationExpiry**
- **isEmailVerified**
- **lastLogin**
- **posts**
- **savedPosts**
- **createdAt**
- **UpdatedAt**

Post Model

Fields:

- **title**
- **content**
- **author**
- **categories**
- **image**
- **createdAt**
- **updatedAt**
- **views**
- **status**
- **likeCount**
- **likes**
- **comments**
- **commentsCount**

---

comments Model

Fields:

- **content**
- **post**
- **author**
- **createdAt**
- **updatedAt**
- **likeCount**
- **likes**

---

category Model

Fields:

- **name**
- **description**
- **createdAt**
- **updatedAt**

Middlewares

Authentication Middleware

- Validates JWT tokens sent in the `Authorization` header.
- Attaches the authenticated user's information to the request object.
- Protects routes that require authentication.

Image Upload Middleware

- Utilizes Multer to handle images
- Uses memory storage to temporarily store image and upload it to Cloudinary and retrieves link to be put in database

Email Verification Middleware

- Checks if user has completed email verification
- Marks the same in database emailVerification field

Frontend Design

Technologies Used

- **React.js**: JavaScript library for building user interfaces.
- **React Router DOM**: Handling client-side routing.
- **Tailwind CSS**: Utility-first CSS framework for styling.
- **Vite**: Build tool for faster development.
- **ESLint**: Linting utility to maintain code quality.
- **Radix-UI:** UI Component Library
- **Lucide:** React-Icons
- **zod:** validation

Project Structure

- **main.tsx**: Entry point of the React application.
- **App.tsx:** Main application component.
- **components/auth/* :**Login and Signup logic
- **components/editor/* :** Post editor Logic
- **components/email/***: Email Verification Logic
- **components/header/*:** Header Logic
- **components/post/*:** Complete post logi
- **context/* :** Auth and Theme Context Logic
- **helper/* :** Helper functions
- **pages/*:** Pages for the whole router

Frontend Design - Routing, State Management

Routing

Implemented using React Router:

- `/`: Landing Page
- `/posts`: Explore Page
- `/user/post`: Current user posts page
- `/create/post`: Create a new post page

State Management

- **Data Fetching**:
  - Utilizes `axios` API with `tanstack-query`.
- **Context API**:
  - Manage Global State Management

Security Considerations

Authentication

- **JWT Tokens**:
  - Securely generated and signed with a secret key.
  - Stored in the client's cookies.
- **Password Security**:
  - Passwords hashed using `bcrypt` before storing in the database.
  - Plain passwords are never stored or logged.
- **2FA**:
  - After Signup user must also verify email adress by clicking the link sent to email by the server

Authorization

- **Protected Routes**:
  - Backend routes require valid JWT tokens.
  - Frontend routes use Protected components to restrict access.

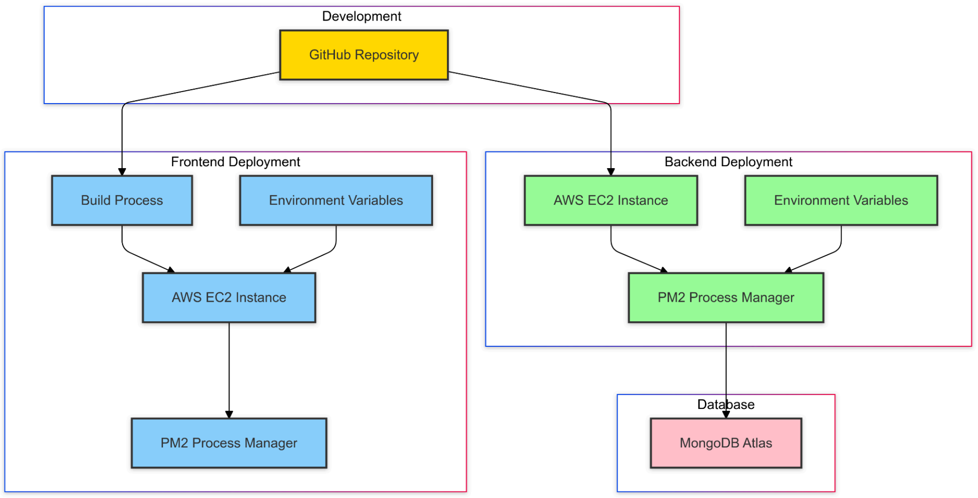
Deployment Plans

Environment Setup

- **Backend Environment Variables**:
  - `JWT_SECRET and JWT_PASSWORD`: Secret key for signing JWTs.
- **Frontend Environment Variables**:
  - `VITE_WS_URL`: Base URL for the WS Server.

Deployment Steps

1. **Backend Deployment**:
   - Host on platforms like Heroku, AWS EC2, or DigitalOcean.
   - Ensure environment variables are securely set.
2. **Frontend Deployment**:
   - Build the React application using `npm run build`.
   - Host static files on services like Vercel or AWS.
3. **Domain and SSL**:
   - Configure a custom domain.
   - Set up SSL certificates for secure HTTPS communication.



Future Enhancements

Technical Improvements

- **State Management Library**:
    - Implement Redux, Context API, Zustang for more complex state needs

Feature Enhancements

- **Notifications**:
    - Real-time notifications for interactions.
- **Admin Panel**:
    - Admin panel for easier admin tasks
- **Analytics Dashboard**:
    - Provide users with insights on Room info, visits and interactions.

Conclusion

The WriteUp project showcases an emerging concept in Virtual Blogging, focusing on creating an interactive, immersive experience. It integrates JavaScript for functionality, with a clear structure for future development. As the project evolves, further features and improvements could be added, expanding its potential applications in virtual spaces.