Blog Application Documentation

Project Overview

The Blog Application is a full-stack web platform that allows users to create, manage, and share blog posts. It provides user authentication, Markdown-based blog writing, and a streamlined UI for a seamless blogging experience. Users can manage their own posts while maintaining security and access control through authentication and authorization mechanisms.

Tools & Technologies Used

Frontend:

- **React (Vite)** – For building the user interface.
- **Tailwind CSS** – For styling and responsive design.
- **React Icons** – For adding icons to the UI.
- **Appwrite** – Cloud storage for blog thumbnails (images).
- **React Markdown** – For rendering Markdown in blogs.
- **MDE Editor** – For writing blogs in Markdown format.
- **Prism React** – For syntax highlighting in code blocks.

Backend:

- **Node.js** – For server-side logic.
- **MongoDB Atlas NoSQL Database** – For storing user and blog data.
- **JWT (JSON Web Tokens)** – For authentication and security.

Features

User Authentication & Management

- User signup and login functionality.
- JWT-based authentication for secure access.
- Profile management with update options.

Blog Management

- Create, read, update, and delete (CRUD) blogs.
- Markdown support for blog writing.

- Syntax highlighting for code snippets.
- Image uploads via Appwrite storage.
- Blog search functionality.

Security Measures

- **JWT Authentication:** Ensures secure access to protected routes.
- **Authorization Controls:** Only the blog owner can edit or delete their posts.
- **Secure API Handling:** Data validation and error handling for API requests.
- **Client-Side JWT Storage:** Tokens are securely managed on the frontend and removed on logout.

Authentication & Authorization

Authentication:

- Users register and log in using their username and password.
- Upon login, a JWT token is generated and sent to the client for authentication.
- The client stores the JWT token and includes it in API requests for protected routes.
- Logout removes the token from the client.

Authorization:

- Only authenticated users can create, update, or delete their own blogs.
- Users can view all blogs but can only edit or delete their own.
- API routes verify the user’s identity before performing any action.

Database Schema Overview

The database uses **MongoDB Atlas** with **Mongoose** to define schemas for managing users and blogs.

User Schema

- Stores user details such as username, fullName, password, and joinedDate.
- Ensures username is unique and password is securely stored (hashed before saving).

Blog Schema

- Contains blog-related data like title, description, thumbnail, content, and timestamps (createdAt, updatedAt).
- References the User schema via username, linking blogs to their creators.
- Auto-updates updatedAt whenever a blog is modified.

API Endpoints

Authentication Routes (authRoutes)

1. User Registration (POST /register)

- **Logic:**
  - Validate required fields (username, fullName, password).
  - Check if the username already exists.
  - Hash password and store user in **MongoDB**.
- **Request Body:**

  { "username", "fullName", "password" }
- **Response:**
  - 201 Created: **User registered successfully**.
  - 409 Conflict: **Username already exists**.
  - 400 Bad Request: **Missing fields**.

2. User Login (POST /login)

- **Logic:**
  - Validate required fields (username, password).
  - Find user in **MongoDB**.
  - Compare password using **bcrypt**.
  - Generate and return **JWT token** if valid.
- **Request Body:**

  { "username", “password" }
- **Response:**
  - 200 OK: **Login successful, returns JWT**.

  - 401 Unauthorized: **Incorrect password**.
  - 404 Not Found: **User not found**.
  - 400 Bad Request: **Missing fields**.

User Routes

1. **Get User Profile (GET /user/profile)**
   - **Logic:**
     - Authenticate user using JWT.
     - Fetch username, fullName, and joinedDate from MongoDB.
   - **Request Headers:**

     { "Authorization": "Bearer <JWT>" }
   - **Response:**
     - 200 OK: Returns user details.
     - 404 Not Found: User does not exist.
     - 401 Unauthorized: Invalid or missing token.
2. **Get User Blogs (GET /user/blogs)**
   - **Logic:**
     - Authenticate user using JWT.
     - Fetch all blogs created by the authenticated user (thumbnail, title, description, createdAt, updatedAt).
   - **Request Headers:**

     { "Authorization": "Bearer <JWT>" }
   - **Response:**
     - 200 OK: Returns the list of user’s blogs.
     - 404 Not Found: No blogs found for the user.
     - 401 Unauthorized: Invalid or missing token.

Blog Routes

1. **Search Blogs (GET /search?query=<text>)**
   - **Logic:**

        - Authenticate user using JWT.
        - Perform a **text search** on title and description using MongoDB's **Atlas Search** with typo tolerance (fuzzy).
        - Sort results by **relevance**.
    - **Request Headers:**

      { "Authorization": "Bearer <JWT>" }
    - **Response:**
        - 200 OK: Returns matching blogs.
        - 400 Bad Request: Missing query parameter.
        - 500 Internal Server Error: Database error.

2. **Get Blog by ID (GET /blog/:id)**
    - **Logic:**
        - Authenticate user using JWT.
        - Fetch blog using MongoDB’s findById().
    - **Request Headers:**

      { "Authorization": "Bearer <JWT>" }
    - **Response:**
        - 200 OK: Returns the blog.
        - 400 Bad Request: Missing blog ID.
        - 401 Unauthorized: Blog not found.
3. **Create Blog (POST /create-blog)**
    - **Logic:**
        - Authenticate user using JWT.
        - Validate required fields (title, description, content).
        - Save new blog to MongoDB.
        - Return generated blog_id to client.
    - **Request Headers:**

      { "Authorization": "Bearer <JWT>" }
    - **Request Body:**

      { "title", "description", "thumbnail", "content" }

- **Response:**
  - 201 Created: Blog added successfully.
  - 400 Bad Request: Missing required fields.
  - 500 Internal Server Error: Database error.

4. **Delete Blog (DELETE /delete-blog)**
   - **Logic:**
     - Authenticate user using JWT.
     - Validate blogId.
     - Check if the blog belongs to the user.
     - Delete blog from MongoDB.
   - **Request Headers:**

     { "Authorization": "Bearer <JWT>" }
   - **Request Body:**

     { "blogId" }
   - **Response:**
     - 200 OK: Blog deleted successfully.
     - 400 Bad Request: Missing blog ID.
     - 403 Forbidden: User not authorized.
5. **Update Blog (PUT /update-blog)**
   - **Logic:**
     - Authenticate user using JWT.
     - Validate blogId.
     - Check if the blog belongs to the user.
     - Update fields if provided.
   - **Request Headers:**

     { "Authorization": "Bearer <JWT>" }
   - **Request Body:**

     { "blogId", "title", "description", "thumbnail", "content" }
   - **Response:**

- 200 OK: Blog updated successfully.
- 400 Bad Request: Missing blog ID.
- 403 Forbidden: User not authorized.