G6 Blog Starter Project - Mini PRD

1. Introduction

Overview

The goal of this project is to develop a backend API for a blog platform. The platform will allow users to create, read, update, and delete blog posts, manage user profiles, and perform advanced search and filtering operations. The project will also include features like user authentication and authorization, different user roles (e.g., Admin and User), and AI integration for content suggestions or enhancements.

2. Product Goals and Objectives

- Develop a RESTful API with clear and intuitive endpoints.
- Implement core CRUD operations for blog posts (create, read, update, delete).
- Integrate user authentication and authorization mechanisms.
- Provide functionalities for blog tags, filtration, and search.
- Incorporate AI capabilities for generating blog content based on user input.
- Ensure high performance, reliability, and scalability.

3 Functional Requirements

3.1 User Management

3.1.1 User Registration:

- ➔ Description: Users can register on the platform with their email, password, and profile details.
- ➔ Actors: Visitor
- ➔ Pre-conditions: None
- ➔ Flow of Events:

- The user submits necessary details (username, email, password, etc.) to the system.
- The system checks if the provided information is valid:
  - email format, password strength
  - verifies if a user with the same username or email already exists.
- If validation and uniqueness checks pass, the system creates a new user account in the database and returns a confirmation message.
- If required, the system sends an activation email or link to the user. (e.g: Mailtrap, OTP …)

3.1.2 Login:

➔ Description: Registered users can log in to access the platform's features

➔ Actors: User & Admin

➔ Pre-conditions: The user must be registered

➔ Flow of Events:

- The user enters their username (or email) and password.
- The system checks if the provided information is in the correct format.
- The system checks if the provided username (or email) and password match an existing user record in the database.
- If the credentials are valid, the system grants access and creates a session with an access token and a refresh token, each with specified expiration dates.
  - The system should store the generated tokens in a database.
- If credentials are invalid, the system displays an error message and prompts the user to try again.

3.1.3 Authentication:

- ➔ Description: As a user, I want to access the system without needing to log in every time.
- ➔ Actors: User & Admin
- ➔ Pre-conditions: Users must be logged in.
- ➔ Flow of Events:
  - Upon request, the user should send the access token to the backend.
  - The server decodes the access token and verifies its signature, expiration, and other claims.
  - If the access token is expired, the client (intercepter) sends a refresh request to the server.
  - The server validates the refresh token against the stored token.
  - If the refresh token is expired, the system deletes the tokens from the database and the user is logged out.
  - If the refresh token is valid, the server generates a new access token and returns it to the client.
    - The new access token is stored in a database.
    - The client (intercepter) resumes the initial request.

3.1.3 Forgot Password:

- ➔ Description: As a user, I want to reset my password if I forget it.
- ➔ Actors: User & Admin
- ➔ Pre-conditions: Users must be registered.
- ➔ Flow of Events:
  - ◆ Users request a password reset by providing their registered email.
  - ◆ System sends a password reset link/token via email.
  - ◆ User resets the password using the provided link.
  - ◆ System updates the password and confirms the change.

3.1.4 Logout:

- ➔ Description: As a user, I want to logout or be forced to logout.
- ➔ Actors: User & Admin
- ➔ Pre-conditions: Users must be logged in.
- ➔ Flow of Events:
    - ◆ The user explicitly requests to log out or the system identifies that the cookies (the access token and refresh token) expire upon requests or actions.
    - ◆ System invalidates the access token and refresh token and deletes the invalidated tokens from the database to prevent any further use.
    - ◆ System sends a confirmation response to the user, acknowledging that the logout process has been successfully completed.

3.1.5 User Promotion and Demotion:

- ➔ Description: As an admin, I want to promote a user to admin and demote them back to a regular user.
- ➔ Actors: Admin
- ➔ Pre-conditions:. None
- ➔ Flow of Events:
    - ◆ The client sends a request to promote or demote a user, possibly including the ID of the user to be promoted or demoted.
    - ◆ The system confirms that the admin is authenticated and authorized to demote or promote a user
    - ◆ The system demotes or promotes the specified user and sends a response to the client.

3.2 Blog

3.2.1 Blog Creation:

- ➔ Description: Users can create new blog posts.
- ➔ Actors: User & Admin
- ➔ Pre-conditions: The user must be authenticated
- ➔ Flow of Events:
  - The user sends a request to the blog creation endpoint with the necessary blog details (title, content, User, tags, date).
  - system checks if all required fields (title, content, User, tags) are present and meet the specified format and length constraints.
  - The system confirms that the user is authenticated and authorized to create a blog (e.g., checks if the user is logged in and has the necessary permissions).
  - If validation and authorization are successful, the system inserts a new blog record into the database, including the provided details and a timestamp.
  - The system returns a success message with the newly created blog post details.

3.2.2 Blog Retrieval:

- ➔ Description: As a user, I want to view all blog posts with the ability to navigate through them using pagination and see their popularity metrics.
- ➔ Actors: User & Admin
- ➔ Pre-conditions: None
- ➔ Flow of Events:
  - The user sends a request to retrieve blog posts, optionally specifying the desired page number and the number of posts per page, and any sorting preferences (e.g., by popularity, most recent).

- The system queries the database and retrieves a specified number of the most recent blog posts (e.g., 20) based on the provided pagination parameters and sorting preferences.
- The system includes popularity metrics (view count, likes, comments) with each retrieved blog post.
- The system returns the retrieved blog posts to the user interface for display, along with pagination metadata such as total pages, current page, and total number of posts.
- As the user navigates through pages (scrolls and reaches the end of displayed posts), the system retrieves the corresponding blog posts for the specified page, with updated poplarity data.

3.2.3 Blog Update:

- ➔ Description: As a user, I want to update the details of an existing blog.
- ➔ Actors: User
- ➔ Pre-conditions: User must be the author of the post.
- ➔ Flow of Events:
  - User sends a request with updated details
  - The system validates the input data to ensure it meets the required format and constraints.
  - If the validation is successful, the system updates the retrieved blog record with the validated new data.
  - The system returns a success message or the updated blog data to the user as confirmation.

3.2.3 Blog Delete:

- ➔ Description: Users or Admins can delete blog posts.
- ➔ Actors: Admin & User

- ➔ Pre-conditions: The user must be authenticated and either own the blog post (for Authors) or have Admin privileges.
- ➔ Flow of Events:
  - User sends a request to delete a blog post.
  - System deletes the blog post and returns a confirmation message

3.2.4 Blog Search:

- ➔ Description: Users can search for blog posts based on titles or Author name (or both).
- ➔ Actors: Any User
- ➔ Pre-conditions: None
- ➔ Flow of Events:
  - The user sends a request to the blog search endpoint with the specified search criteria (e.g: title, Author name).
  - The system validates the request
  - The system fetches blog data from the database based on the search criteria.
  - If additional filters options are specified by the user, the system applies them to the retrieved blog data.
  - The system sends the formatted search results to the user.

3.2.5 Blog Popularity Tracking:

- ➔ Description: Track the popularity of each blog post.
- ➔ Actors: System
- ➔ Pre-conditions: None
- ➔ Flow of Events:
  - System tracks metrics like view count, likes (dislikes), and comments.

    - When a user interacts with a blog post (e.g., likes, or dislikes), the system checks if the user has already performed the same action (e.g., liked the post).
    - If the user has already liked or disliked the post, the system prevents duplicate actions to ensure accurate tracking.
    - System updates popularity data whenever users interact with blog posts.

3.2.6 Blog Filtration:

- ➔ Description: Users can filter blog posts by tags, date, or popularity.
- ➔ Actors: Any User
- ➔ Pre-conditions: None
- ➔ Flow of Events:
    - The user sends a request to the blog filter endpoint with the specified tag(s), date, or popularity.
    - The system checks if the provided criteria are valid.
    - The system fetches blog data from the database.
    - If additional search options are specified by the user, the system applies them to the retrieved blog data.
    - The system filters the retrieved blog data based on the specified criteria.
    - The system sends the formatted filtered blog data to the user.

3.3 AI Integration

3.3.1 Generate blog content based on user-provided keywords or topics

- ➔ Description: The system suggests improvements or generates content ideas.
- ➔ Actors: Any User
- ➔ Pre-conditions: User should be registered

- ➔ Flow of Events:
  - Integrate simple AI chat feature
  - User requests content suggestions for a blog post.
  - System processes the blog content and returns AI-generated suggestions or enhancements.

3.4 Profile Management

3.4.1 User Profile Update:

- ➔ Description: Users can update their profile details such as bio, profile picture, and contact information.
- ➔ Actors: Registered User
- ➔ Pre-conditions: The user must be authenticated
- ➔ Flow of Events:
  - User sends a request to update their profile information.
  - System validates the input and updates the user's profile.
  - System returns the updated profile details.

4. API Documentation

- Document all API endpoints clearly using Postman. Include example requests and responses, error codes, and descriptions for each endpoint. Ensure that the documentation is kept up to date with any changes made to the API during development. This will help both developers and users understand how to interact with the API effectively.

5. Non Functional Requirements

- **Scalability**: The API should be designed to handle multiple users and requests simultaneously.

    - Concurrency with Goroutines: Leverage Go’s lightweight goroutines to handle multiple tasks concurrently. Use channels and `` `sync` `` primitives to manage synchronization and communication between goroutines.
- **Security**:
    - Secure Password Handling:
        - Encryption: Use a secure hashing algorithm like `bcrypt` for storing passwords. Avoid storing plain-text passwords.
    - Authentication & Authorization:
        - JWT Tokens: Implement JWT (JSON Web Tokens) for stateless authentication. Ensure tokens are signed and verified correctly.
        - OAuth2: For more advanced use cases, consider implementing OAuth2 for third-party authentication.
    - Role-Based Access Control (RBAC):
        - Roles & Permissions: Define user roles (e.g., Admin, User) and enforce permissions at the API level. Use middleware to check user roles before accessing certain endpoints.
- **Performance:** API response time should be optimized to ensure a smooth user experience.
    - API Response Optimization:
        - Pagination: Implement pagination to avoid loading large datasets at once. This reduces memory usage and improves response times.
        - Caching: Use caching to store the results of expensive operations (e.g., computed results or frequent database queries) to reduce processing time.