Your blog starter project has a well-structured set of features and requirements. To implement this in a scalable, secure, and maintainable way, we’ll follow Clean Architecture principles, ensure proper version control, and focus on production-ready practices. Here's a step-by-step guide for your project:

### Step 1: Project Setup

1. \*\*Version Control\*\*:

- \*\*Git\*\*: Initialize a Git repository for version control.

- Use platforms like GitHub or GitLab for repository hosting.

- Create a `.gitignore` file (to ignore unnecessary files like dependencies, build files, etc.).

2. \*\*Directory Structure\*\* (following Clean Architecture):

```bash

/blog-api

├── /cmd # Entry point of the application

├── /domain # Entities and business logic (Blog, User)

├── /usecase # Application-specific business rules (User management, blog creation)

├── /controller # API controllers to handle HTTP requests

├── /infrastructure # External services (database, external APIs)

├── /repository # Database access layer

├── /dto # Data Transfer Objects (structs for requests and responses)

├── /middleware # JWT authentication, logging, and RBAC

├── /router # API routes

├── /config # Configuration settings (environment variables)

├── /docs # API documentation (Postman collections, OpenAPI/Swagger)

└── /test # Unit and integration tests

```

### Step 2: Core Technology Stack

1. \*\*Programming Language\*\*: Go

2. \*\*Framework\*\*:

- \*\*Gin\*\*: Fast HTTP router for creating the RESTful API.

- \*\*GORM or MongoDB\*\*: Choose between an SQL (e.g., PostgreSQL) or NoSQL (e.g., MongoDB) database, depending on your scalability needs.

- \*\*JWT\*\*: For stateless authentication.

- \*\*bcrypt\*\*: For secure password hashing.

- \*\*Mailtrap\*\*: For handling email features like account verification and password reset.

- \*\*Goroutines\*\*: For handling concurrency where needed.

### Step 3: Key Features Implementation

\*\*User Management:\*\*

- \*\*Registration\*\*: Create a route to accept user details, validate input, hash the password, and store the user.

- \*\*Login\*\*: Validate credentials, generate access and refresh tokens.

- \*\*JWT Middleware\*\*: Implement middleware to verify tokens for protected routes.

- \*\*Forgot Password\*\*: Implement email-based password reset using tokens.

- \*\*User Roles\*\*: Use RBAC middleware to manage admin and user permissions.

\*\*Blog Management:\*\*

- \*\*CRUD\*\*: Implement the CRUD operations for blog posts.

- \*\*Search & Filter\*\*: Use indexing for efficient search and implement filters based on tags, popularity, etc.

- \*\*Pagination\*\*: Ensure efficient pagination for fetching large datasets.

- \*\*AI Integration\*\*: Use an external API or a basic GPT-based model for blog suggestions.

### Step 4: Database Setup

1. \*\*MongoDB\*\* or \*\*PostgreSQL\*\*:

- For blog content, tags, users, and other relational data, consider using MongoDB for flexibility or PostgreSQL for structured data.

- Create models and repository interfaces in the `domain` and `repository` layers.

2. \*\*Connection\*\*:

- Use environment variables to store sensitive information like database credentials.

- Connect to MongoDB/PostgreSQL from the `infrastructure` layer and ensure proper error handling.

### Step 5: Authentication & Authorization

1. \*\*JWT Token\*\*:

- Implement JWT generation on login and refresh tokens for session continuation.

- Use middleware to protect routes and roles (admin vs. user).

2. \*\*Secure Password Handling\*\*:

- Use \*\*bcrypt\*\* to hash passwords before storing them in the database.

- Ensure tokens are stored securely.

3. \*\*Role-Based Access Control (RBAC)\*\*:

- Define roles (e.g., Admin, User) in the `domain` layer.

- Protect sensitive routes (e.g., promoting/demoting users, deleting blogs) with RBAC checks in middleware.

### Step 6: AI Integration

1. Use GPT-based APIs (like OpenAI or Hugging Face) to generate content suggestions.

2. Integrate AI functionality for post enhancements as a separate service.

### Step 7: API Documentation

1. \*\*Postman\*\*:

- Document all API endpoints with example requests, responses, and error codes.

- Generate a Postman collection for testing and sharing.

2. \*\*OpenAPI/Swagger\*\*:

- Use Swagger to generate interactive API documentation.

### Step 8: Testing

1. \*\*Unit Tests\*\*:

- Write tests for each use case (e.g., blog creation, user registration).

- Use Go’s testing framework with Testify for assertions and mocks.

2. \*\*Integration Tests\*\*:

- Test how your components interact with external systems (e.g., database, third-party APIs).

### Step 9: Performance Optimization

1. \*\*Concurrency\*\*:

- Use \*\*goroutines\*\* and channels for concurrent operations, such as background tasks (e.g., sending emails).

2. \*\*Caching\*\*:

- Cache expensive queries (like blog popularity metrics) to reduce database load.

3. \*\*Rate Limiting\*\*:

- Implement rate limiting to avoid misuse of your API.

### Step 10: Deployment

1. \*\*Containerization\*\*:

- Use Docker to containerize your app for easy deployment across different environments.

2. \*\*CI/CD\*\*:

- Set up continuous integration (CI) to automate testing and deployment.

3. \*\*Hosting\*\*:

- Deploy on platforms like AWS, Heroku, or Google Cloud.

- Use Kubernetes for managing microservices and scaling.

### Step 11: Version Control Workflow

1. \*\*Branching Strategy\*\*:

- Use Git branches for feature development (`feature/user-auth`, `feature/blog-crud`).

- Merge into the `main` branch after code review and testing.

2. \*\*Tagging & Releases\*\*:

- Use Git tags to mark releases (`v1.0`, `v1.1`).

- Maintain version control for easier rollback if issues arise.

---

This roadmap ensures your backend project follows best practices for Clean Architecture, scalability, security, and maintainability. Each feature aligns with your PRD, and version control (Git) will help manage and track progress.