OLUWATOSIN OGUNSAN

TechCorp Blog API

Task 1: Database Creation

**User**:

We need a way to store information about users who register for the blog. So, we created a table named "User".

For each user, we need to keep track of their username, email, and password securely (using a password hash).

We also want to know when users joined, so we have "created\_at" and "updated\_at" fields to track the creation and modification timestamps.

Each user is uniquely identified by a user\_id, which is automatically generated and serves as the primary key.

**Post**:

Users should be able to create blog posts, so we have a table called "Post" to store them.

Each post needs to have a title, content, and reference to the user who authored it (user\_id).

We keep track of when posts were created and last updated using "created\_at" and "updated\_at" fields.

Each post has its own unique post\_id, serving as the primary key.

**Comment**:

Users can comment on posts, so we have a table named "Comment" to store these comments.

Each comment should be associated with the user who made it (user\_id) and the post it's commenting on (post\_id).

We store the content of each comment, along with creation and update timestamps.

Each comment is uniquely identified by a comment\_id, acting as the primary key.

**Considerations**:

Data Integrity: We used foreign keys (user\_id, post\_id) to maintain relationships between entities. This ensures that users, posts, and comments remain consistent and referentially intact.

Security: Passwords are stored securely using password hashing techniques to protect user accounts from unauthorized access.

Timestamps: We added "created\_at" and "updated\_at" fields to track when records were created or modified. This helps in auditing and understanding data changes over time.

Scalability: The schema is designed to support scaling the application as it grows. Relationships between entities are essential for maintaining data integrity and facilitating efficient querying.

Simplicity: The schema is kept relatively simple to maintain clarity and ease of understanding. While additional features could be added, we've focused on the core functionality needed for a basic blog API.

TASK 2:

I've designed endpoints like "/posts" to handle all actions related to blog posts. Additionally, I've included optional endpoints like "/posts/{postId}/comments" and "/dashboard" for extra features if needed.

To organize data, I've created structures like "Post" and "Comment", which contain attributes such as postId, title, and content and authorId to store relevant information.

My code is structured in a readable way, following best practices for maintainability and reusability. Error handling is implemented to handle exceptions gracefully and provide clear messages using appropriate HTTP status codes.

For security, I've focused on measures like authentication, and authorization to safeguard sensitive data and prevent unauthorized access.

Overall, my goal is to create a robust, secure, and user-friendly API for managing blog posts , with room for future enhancements and customization.