Take-home test for a backend engineer that focuses on Go, MongoDB, Redis, and authorization. The goal of this test is to assess the candidate's ability to build a basic API that involves CRUD operations, Redis caching, and JWT-based authorization.

Backend Engineer Take-Home Test

Objective

Build a RESTful API for a simple case management system that supports user authentication, task creation, task management, and caching. The API should be built using Go and should store the data in MongoDB, with Redis used for caching.

Requirements

1. Task Management API

- **Task model**:
  - `id`: unique identifier (UUID)
  - `title`: string
  - `description`: string
  - `status`: string (e.g., "todo", "in-progress", "done")
  - `created_at`: timestamp
  - `updated_at`: timestamp
- Implement the following endpoints:
  1. **Create a task**: `POST /tasks`
  2. **Get a task by ID**: `GET /tasks/{id}`
  3. **Update a task**: i`PUT /tasks/{id}`
  4. **Delete a task**: `DELETE /tasks/{id}`
  5. **List all tasks**: `GET /tasks`

2. Authorization

- Use JWT for authorization.
- Implement user roles (e.g., `admin` and `user`).
  - **Admin**: Can create, update, delete, and view all tasks.
  - **User**: Can only create and view their own tasks.
- Add an authentication layer that:
  1. Issues a JWT token upon login.
  2. Validates JWT tokens for all endpoints (except login).
  3. Extracts user roles and permissions from the token.
- Create an endpoint for user login that returns a JWT token: `POST /login`.

3. Redis Caching

- Cache the result of the **Get Task by ID** (`GET /tasks/{id}`) endpoint using Redis.
- If a task is cached, return the cached version. If not, fetch from MongoDB and cache the result in Redis for subsequent requests.
- Allow cache invalidation when a task is updated or deleted.

4. Database

- Store all tasks in MongoDB.
- Ensure the use of indexes where appropriate (e.g., `created_at`).

Deliverables

1. Codebase:
   - Your solution should include:
     - A fully functional API written in Go.
     - MongoDB integration for storing tasks.
     - Redis integration for caching.
     - JWT-based authentication and authorization.

2. Documentation:
   - Clear instructions on how to run the project locally (using Docker if possible).
   - Details about API endpoints and their expected request/response payloads.
   - Description of how JWT authorization is implemented.

Bonus (Optional)

- Use **Golang's standard library** as much as possible.
- Implement rate-limiting on the endpoints.
- Use **Docker** for setting up MongoDB, Redis, and the API server.
- Add unit tests for the core logic (task creation, JWT generation, etc.).

Submission Guidelines

- Submit your solution as a Git repository (GitHub, GitLab, etc.).
- Include a README file that describes how to set up and test the application.
- Deadline: [Insert deadline].

Assessment Criteria

- Code quality (readability, structure, and best practices).
- Correctness (meets the functional requirements).
- Understanding of Go, MongoDB, Redis, and JWT-based authorization.
- Usage of appropriate libraries and tools.
- Documentation and clarity.

Good luck! We look forward to reviewing your submission.