Go Language Rest API Exercises

(Intermediate to Advanced Concepts**)**

Case Study 1: Building a Blog Management System Using Go HTTP Server and SQLite

Objective:

Develop a lightweight blog management system with features like creating, reading, updating, and deleting blog posts using Go. The focus is on scalability, simplicity, and performance.

Background:

A small blogging startup wants to build a cost-effective, fast, and scalable blogging platform. They prefer Go for its performance and concurrency capabilities. SQLite is chosen as the database because it is lightweight and requires no separate server setup.

Solution:

1. **Go HTTP Server:**
    - A basic HTTP server is set up using Go's built-in *net/http* package.
    - **Routes include:**
        - POST /blog: Create a new blog post.
        - GET /blog/{id}: Fetch a specific blog post by ID.
        - GET /blogs: Fetch all blog posts.
        - PUT /blog/{id}: Update a blog post by ID.
        - DELETE /blog/{id}: Delete a blog post by ID.
2. **Go REST API:**
    - RESTful principles are followed to ensure clean and predictable API endpoints.
    - JSON is used for request and response data formats.
3. **Go Air:**

- Go Air is integrated into the project for live reloading during development, enhancing developer productivity.
- As the server code is modified, Go Air automatically restarts the server, reflecting the changes instantly.

4. **SQLite Connectivity:**
    - SQLite is integrated using the *github.com/mattn/go-sqlite3* driver.
    - Database schema includes a blogs table with fields like id, title, content, author, and timestamp.
    - **Query examples:**
        - Create Blog: INSERT INTO blogs (title, content, author, timestamp) VALUES (?, ?, ?, ?)
        - Fetch Blogs: SELECT * FROM blogs ORDER BY timestamp DESC.
5. **Middleware:**
    - **Middleware functions are implemented for:**
        - Logging incoming requests.
        - Validating JSON payloads in requests.
        - Authentication (e.g., validating API keys).
        - Error handling for proper JSON responses when an error occurs.

Outcome:

The blog management system is deployed successfully. Key benefits include:

- Efficient request handling using Go's lightweight HTTP server.
- Cost-effective deployment since SQLite requires no dedicated server.
- Simplified development using Go Air.
- Easy extensibility with middleware to add features like rate limiting and advanced logging.

Case Study 2: Building an E-Commerce Microservice Using Go Rest API and Middleware

Objective:

Develop a standalone Go-based microservice for managing product inventory in an e-commerce platform, emphasizing high performance and modularity.

Background:

An e-commerce company is transitioning to a microservices architecture. They need an inventory service to manage product stock, including CRUD operations and real-time updates. The service must integrate with other microservices like the order service and user service.

Solution:

1. **Go HTTP Server**:
    - A Go HTTP server handles incoming requests for the inventory microservice.
    - Routes include:
        - POST /product: Add a new product.
        - GET /product/{id}: Fetch product details by ID.
        - PUT /product/{id}: Update stock details for a product.
        - DELETE /product/{id}: Remove a product from the inventory.
2. **Go REST API**:
    - RESTful API design ensures clear and consistent interactions with the microservice.
    - Pagination is implemented for fetching large inventories using query parameters (?page=1&limit=10).
3. **SQLite Connectivity**:
    - SQLite is used for prototyping and small-scale deployments.
    - Database schema includes:
        - products table with fields like id, name, description, price, stock, and category_id.
    - SQL Queries:
        - **Add Product**: INSERT INTO products (name, description, price, stock, category_id) VALUES (?, ?, ?, ?, ?)

        - **Update Stock**: UPDATE products SET stock = ? WHERE id = ?.

4. **Middleware**:
    - Custom middleware for:
        - **Request Authentication**: JWT-based token validation.
        - **Input Validation**: Ensure required fields are present in the JSON payload.
        - **Rate Limiting**: Prevent abuse of API endpoints.
        - **Logging**: Log API requests and responses with metadata (IP address, timestamp, method).
5. **Go Air Integration**:
    - Go Air enhances development by providing instant reloading during iterative testing.
6. **Inter-Service Communication**:
    - The service communicates with other microservices via REST APIs or message queues like RabbitMQ. For example:
        - An order service calls the inventory service to deduct stock when an order is placed.

Outcome:

The inventory microservice is successfully integrated into the e-commerce platform, offering:

- High performance due to Go's lightweight concurrency model.
- Scalability and modularity, enabling independent deployment and updates.
- Reduced development and debugging time using Go Air.
- Enhanced security and reliability with robust middleware implementations.