Zocket

SDE - 1 Backend Assignment

Advanced Coding Assignment: Product Management System with Asynchronous Image Processing

Objective: Develop a backend system in **Go** for a product management application, emphasizing architectural best practices, including asynchronous processing, caching, logging, and high scalability.
—

Project Overview

Create a RESTful API in Golang for managing products, focusing on asynchronous processing, caching, and high performance. Implement the following components:

1. **API Design**:
   - **POST /products**: Accepts product data with the following fields:
   - `user_id` (reference to the user table).
   - `product_name` (string).
   - `product_description` (text).
   - `product_images` (array of image URLs).
   - `product_price` (decimal).
   - **GET /products/:id**: Retrieves product details by ID, with image processing results.
   - **GET /products**: Returns all products for a specific `user_id`, with optional filtering by price range and product name.

2. **Data Storage**:
   - Use PostgreSQL for storing `users` and `products` data. Design schema similar to the example, with the following additional fields:
   - **Products Table**: Add a `compressed_product_images` column for storing processed images.
3. **Asynchronous Image Processing**:
   - After storing product details, add the `product_images` URLs to a message queue (RabbitMQ or Kafka).
   - Create an image processing microservice that consumes messages from the queue, downloads, compresses images, and stores compressed images in designated storage (e.g., S3). Update the `compressed_product_images` field in the database upon completion.
4. **Caching**:
   - Use Redis to cache product data retrieved by the `GET /products/:id` endpoint to reduce database load.
   - Implement cache invalidation to ensure that updates to the product data are reflected in real time.

5. **Enhanced Logging**:
   - Implement structured logging (using a library like `logrus` or `zap`) for all service components.
   - Log all requests with response times, API errors, and processing details. Additionally, log specific events in the image processing service (e.g., download success, compression failure).
6. **Error Handling:**
   - Implement robust error handling across all components, especially for asynchronous processing failures (e.g., queue retry mechanisms or dead-letter queues).
7. **Testing**:
   - Write unit tests for each API endpoint and core function.
   - Include integration tests to validate end-to-end functionality, particularly asynchronous processing and cache effectiveness.
   - Benchmark tests for the `GET /products/:id` endpoint, measuring response times with and without cache hits.

—

System Architecture Requirements

- **Modular Architecture**: Structure code to separate API, asynchronous tasks, caching, and logging modules.
- **Scalability**: Design with scalability in mind, including the ability to handle increased API load, distributed caching, and image processing services.
- **Transactional Consistency**: Ensure that data is consistent across the database, cache, and message queue, with retries and compensating transactions in case of failure.

Submission Requirements

1. **Codebase**:
   - Modular and organized, with clear instructions on setup and configuration.
   - Include all configuration files (e.g., database schema, environment files) and caching strategies.
2. **Testing Coverage**:
   - Comprehensive unit and integration tests with a minimum of 90% code coverage.
3. **Documentation**:
   - Detailed README explaining architectural choices, setup instructions, and assumptions.

Submit your assignment here - https://forms.gle/QXjvhr9g4SG8MMY46