**ZOCKET BACKEND ASSINGMENT**

* **Krithik Viswanath S**

**Project Overview :**

The Product Management System is a backend service for managing products, including their details, images, and compressed images. The application is developed using **Go (Golang)** with **PostgreSQL** as the database. It provides APIs to create, retrieve, and list products, ensuring data integrity and scalability.

**Installation Steps**

**1.Folder Structure (Planned) :**

product-management/

├── api/

│ ├── handlers/

│ │ ├── products.go

├── async/

│ ├── image\_processor.go

├── config/

│ ├── config.go

├── database/

│ ├── migrations/

│ │ ├── create\_tables.sql

├── logging/

│ ├── logger.go

├── cache/

│ ├── redis.go

├── queue/

│ ├── rabbitmq.go

├── utils/

│ ├── response.go

├── tests/

│ ├── integration/

│ ├── unit/

├── main.go

├── go.mod

**2.Folder Structure (Implemented) :**

product-management/

├── main.go # Main application file

├── database.go # Database connection logic

├── .env # Environment variable file

├── go.mod # Module file for dependencies

├── go.sum # Dependency checksums

└── README.md # Documentation (optional)

**3. Setup Environment**

* Install **Go (Golang)** (version 1.17+).
* Install **PostgreSQL** (version 12+).
* Install **Git** for version control.
* Install **POSTMAN** to test APIs

**4. Install Required Libraries**

Run the following commands to install the required Go packages:

* go get github.com/gin-gonic/gin
* go get github.com/lib/pq
* go get github.com/joho/godotenv

**5. Setup .env File**

Create a .env file in the project root to store sensitive database credentials:

DB\_HOST=localhost

DB\_PORT=5433

DB\_USER=postgres

DB\_PASSWORD=assassinkv619

DB\_NAME=product\_management

**6. Database Setup**

1. Create a PostgreSQL database named product\_db.
2. Create a products table using the following SQL command:

CREATE TABLE products (

id SERIAL PRIMARY KEY,

user\_id INT NOT NULL,

product\_name TEXT NOT NULL,

product\_description TEXT,

product\_images TEXT[],

compressed\_product\_images TEXT[],

product\_price FLOAT

);

**How to Run the Application :**

**1. Run the Database**

Ensure PostgreSQL is running, and the product\_db database has been set up with the required table schema.

**2. Run the Application**

To start the application

go run database.go

go run main.go

The application will start on[**http://localhost:8080**](http://localhost:8080).

**API Endpoints**

1. **POST /products**

* **Description**: Create a new product.
* **Response**:

A screenshot of a computer

Description automatically generated

From the above image we can see the POST Request was successfully made.

**2. GET /products/**

* **Description**: Retrieve a product by its ID.
* **Response**:

A screenshot of a computer

Description automatically generated

From the above image we can see the GET Request was successfully made.

**Tasks Completed**

**1. API Development**

* **Implemented**:
  + POST /products to create a new product.
  + GET /products/:id to retrieve a product by ID.
  + GET /products to list all products.

**Current Progress**

**Completed Tasks :**

* Backend application setup with database integration.
* Development of product management APIs.

**Outstanding Issues :**

* The GET /products/:id endpoint does not return compressed\_product\_images in the response, despite the data being present in the database.

**Next Steps :**

* Investigate the Scan logic used in the GET /products/:id endpoint to ensure correct retrieval of the compressed\_product\_images field.
* Conduct further testing with different product IDs to isolate the bug.

**Conclusion :**

I sincerely apologize for not completing the project within the planned timeline of 5 days. Despite my best efforts, there were challenges, particularly with the retrieval of compressed\_product\_images, which caused some delays. I am actively working on debugging and resolving this issue and have made significant progress in implementing and testing the essential functionalities. I am committed to addressing this remaining task promptly to ensure the project is fully functional and meets expectations.