Assignment Guidance: Building API CRUD

Golang Bootcamp Batch 3

Periode Pembelajaran

API Development with Golang: Implementing Update and Delete Operations
File Handling and Storage API

Objectives

1. Mampu merancang dan melakukan query pada database untuk mengelola produk, inventaris, dan pesanan.
2. Memahami RESTful API untuk berinteraksi dengan database.
3. Menghubungkan backend API untuk melakukan operasi CRUD.
4. Membuat endpoint untuk membuat, membaca, memperbarui, dan menghapus data inventaris.
5. Mengimplementasikan unggah/unduh file untuk gambar produk.
6. Menyusun route dan menyajikan API menggunakan framework Gin di Go.

Deskripsi Assignment

Assignment ini merupakan kelanjutan dari tugas sebelumnya "Web Server dan Golang Route". Pada tugas sebelumnya, peserta telah membangun sistem backend dasar dengan CRUD API menggunakan Golang dan framework Gin. Sekarang, tugas ini berfokus pada implementasi file handling dan storage API, yang akan memungkinkan pengguna untuk mengunggah dan mengunduh file gambar produk dengan aman dan efisien.

Peserta akan mengembangkan sistem backend dengan fitur:

- Unggah dan unduh gambar produk untuk memperkaya data inventaris.
- Penyimpanan file di sistem lokal atau layanan cloud seperti AWS S3 atau Firebase.
- Validasi file, error handling, dan optimasi pengelolaan file dalam API.

Dengan menyelesaikan tugas ini, peserta akan memahami bagaimana mengelola file dalam sistem backend serta menerapkan konsep RESTful API dalam file handling.

Detail Assignment

Tugas ini merupakan kelanjutan dari assignment sebelumnya "Web Server dan Golang Route". Pada tugas ini, peserta akan memperluas fitur backend yang telah dibuat sebelumnya dengan menambahkan file handling dan storage API untuk mendukung unggah dan unduh file gambar produk.

Tugas ini berfokus pada implementasi file handling dan storage API. Peserta harus mengembangkan fitur berikut:

1. File Handling and Storage

   Tambahan utama dalam tugas ini adalah unggah dan unduh gambar produk, dengan spesifikasi berikut:

   - Endpoint Upload: Mengunggah gambar produk ke penyimpanan lokal
   - Endpoint Download: Mengunduh gambar produk berdasarkan ID produk.
   - Penyimpanan: Bisa menggunakan folder lokal
   - Validasi File: Pastikan hanya format gambar tertentu yang diterima (misal: PNG, JPG, JPEG) dengan ukuran maksimum yang ditentukan.
   - Error Handling: Tangani skenario kesalahan seperti file berukuran terlalu besar atau format yang tidak sesuai.

2. Web Server and Routing

   - Gunakan framework Gin untuk menyusun route API.
   - Pastikan endpoint menangani metode HTTP POST dan GET dengan benar.

3. API Integration and Testing

   - Uji endpoint unggah dan unduh file menggunakan Postman.
   - Pastikan validasi input dan error handling diterapkan dengan baik.

Tools

MySQL dan bebas menggunakan pustaka apapun untuk penanganan file, pengujian API, dan konektivitas database.

Pengumpulan Assignment

Deadline :
Maksimal H+7 Kelas (Pukul 23.30 WIB)
Details :
Kirimkan tugas ini sebagai link repository GitHub, secara individu, melalui LMS. Sertakan file README.md dalam proyek yang berisi petunjuk tentang cara mengatur database dan menjalankan proyek.

Indikator Penilaian

No.	Aspek Penilaian	Parameter	Bobot Maksimal
1	File Handling & Storage Implementation	- Endpoint Upload: Gambar produk berhasil diunggah dan disimpan dengan benar. - Endpoint Download: Gambar produk dapat diakses dan diunduh berdasarkan ID produk. - Validasi File: Hanya menerima format gambar yang diperbolehkan (PNG, JPG, JPEG) dengan batasan ukuran maksimal yang ditentukan. - Error Handling: API menangani kasus kesalahan seperti ukuran file terlalu besar atau format tidak sesuai dengan baik. - Penyimpanan File: File disimpan di direktori lokal atau layanan cloud dengan struktur yang rapi dan efisien.	50
2	API Routing & Functionality	- Endpoint CRUD (Create, Read, Update, Delete) dari tugas sebelumnya tetap berfungsi dengan baik dan kompatibel dengan fitur baru. - Endpoint upload/download file menangani metode HTTP POST dan GET dengan benar. - API mengembalikan response yang sesuai dengan standar RESTful API (status code, JSON response).	20
3	Code Structure & Best Practices	- Kode terstruktur dengan modular, mengikuti praktik terbaik Golang. - Penggunaan framework Gin untuk routing API dilakukan dengan efisien. - Variabel dan fungsi diberi nama dengan deskriptif. - Error handling diterapkan di seluruh kode untuk menghindari crash dan bug yang tidak diharapkan.	20
4	Documentation & Testing	- File README.md: Memuat panduan pengaturan database dan cara menjalankan proyek. - Dokumentasi API: Termasuk daftar endpoint (upload, download), metode HTTP yang digunakan, dan contoh respons JSON. - Testing dengan Postman: Pengujian berhasil dilakukan dan mencakup berbagai skenario (sukses dan error).	10

Referensi/Template

Ketentuan Pencapaian Nilai:
Nilai minimum Lulus Penyaluran Kerja: 75
Nilai minimum Lulus Bootcamp: 65

Ketentuan Penilaian:
Mengumpulkan Assignment tepat waktu: Sesuai dengan nilai yang diberikan mentor
Mengumpulkan Assignment 12 jam setelah deadline: - 3 dari nilai yang diberikan mentor
Mengumpulkan Assignment 1 x 24 Jam setelah deadline: - 6 dari nilai yang diberikan mentor
Mengumpulkan Assignment 2 x 24 Jam setelah deadline: - 12 dari nilai yang diberikan mentor
Mengumpulkan Assignment 3 x 24 Jam setelah deadline: - 18 dari nilai yang diberikan mentor
Mengumpulkan Assignment 4 x 24 Jam setelah deadline: - 24 dari nilai yang diberikan mentor
Mengumpulkan Assignment 5 x 24 Jam setelah deadline: - 30 dari nilai yang diberikan mentor
Mengumpulkan Assignment 6 x 24 Jam setelah deadline: - 36 dari nilai yang diberikan mentor
Mengumpulkan Assignment 7 x 24 Jam setelah deadline: - 42 dari nilai yang diberikan mentor

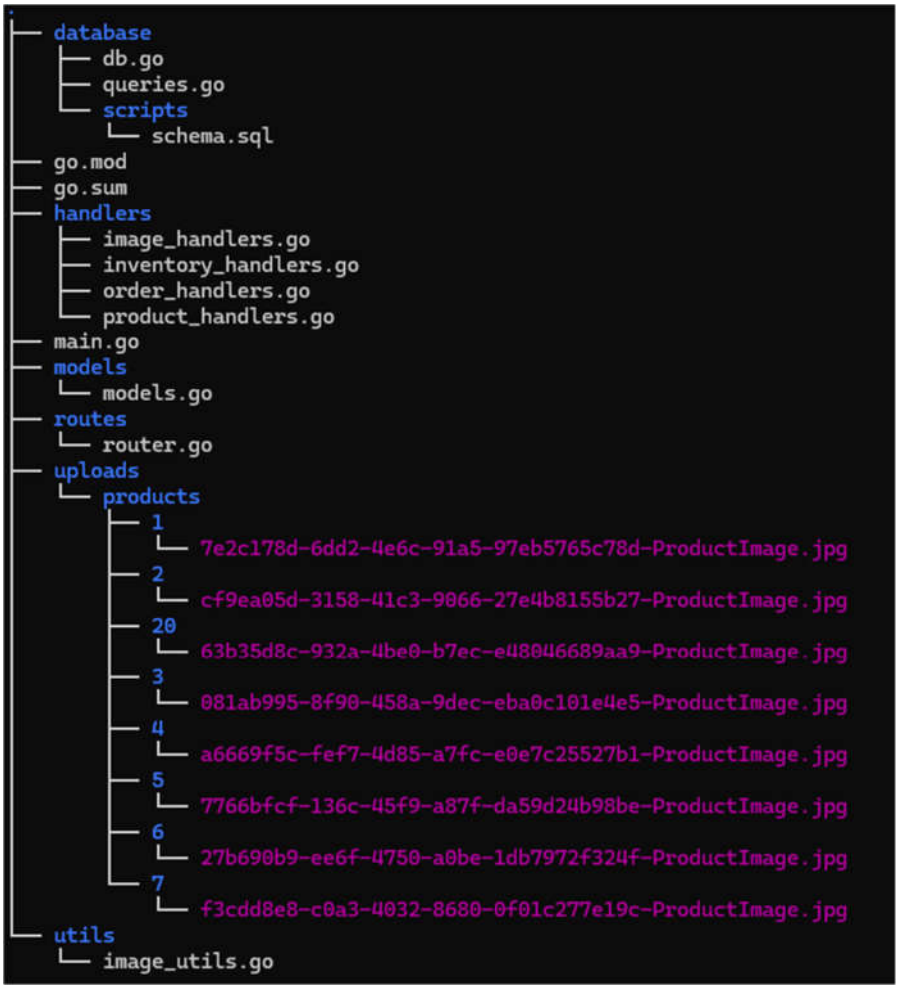
File README.md mencakup instruksi yang jelas untuk setup database dan menjalankan proyek.
Dokumentasi API (endpoint, method, dan contoh respons) dibuat secara ringkas dan jelas.

Assignment Day 26: Building API CRUD

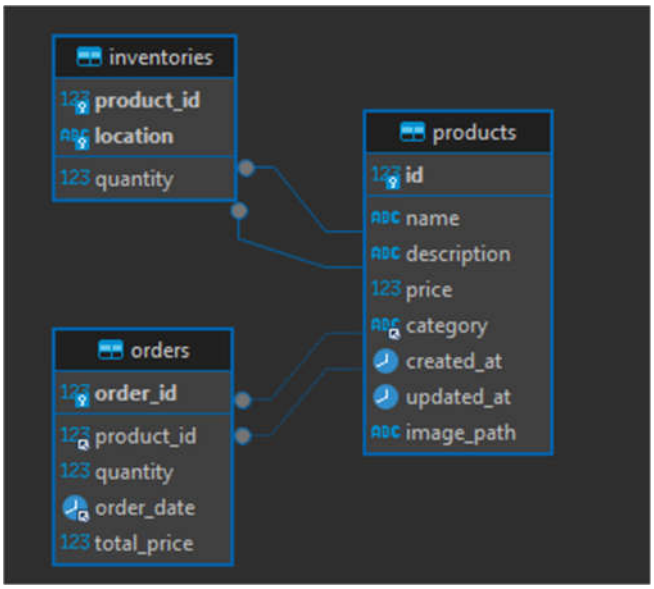
Golang Bootcamp Batch 3
By: Muhammad Ziad

Documentation

Folder Structure:

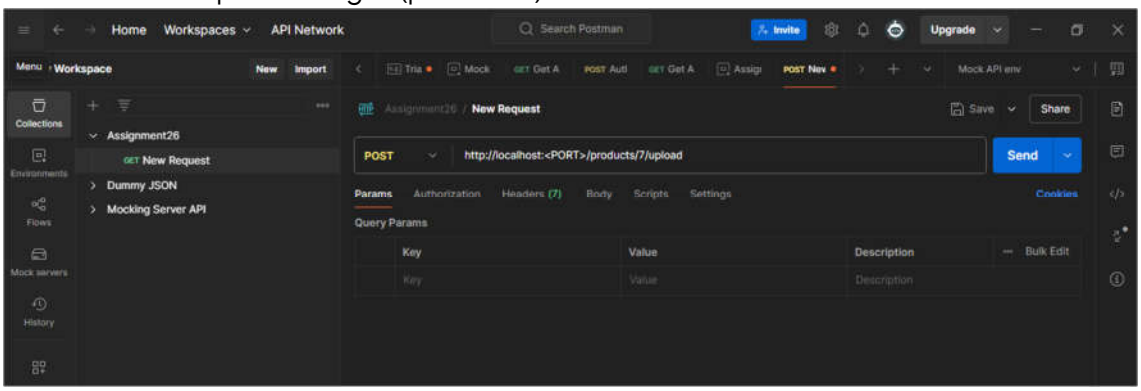


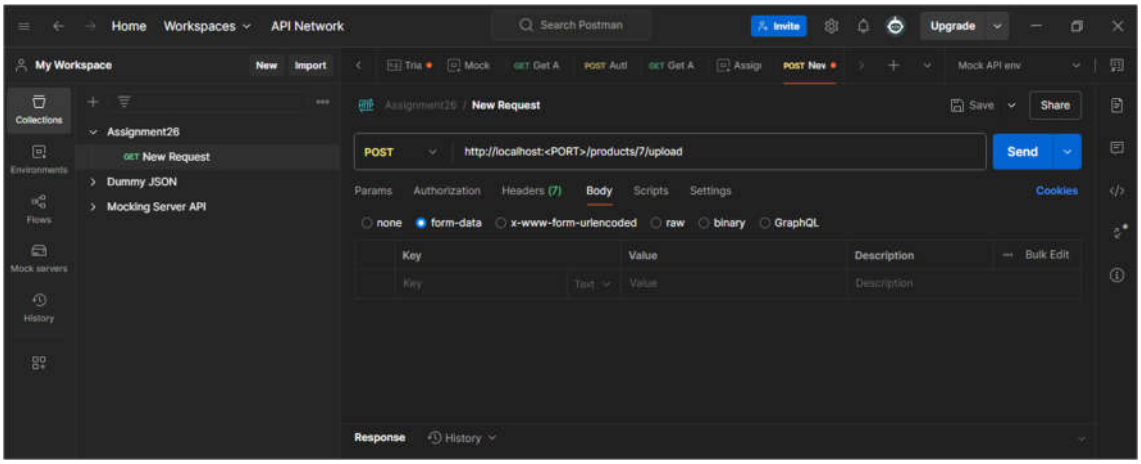
Database Design:



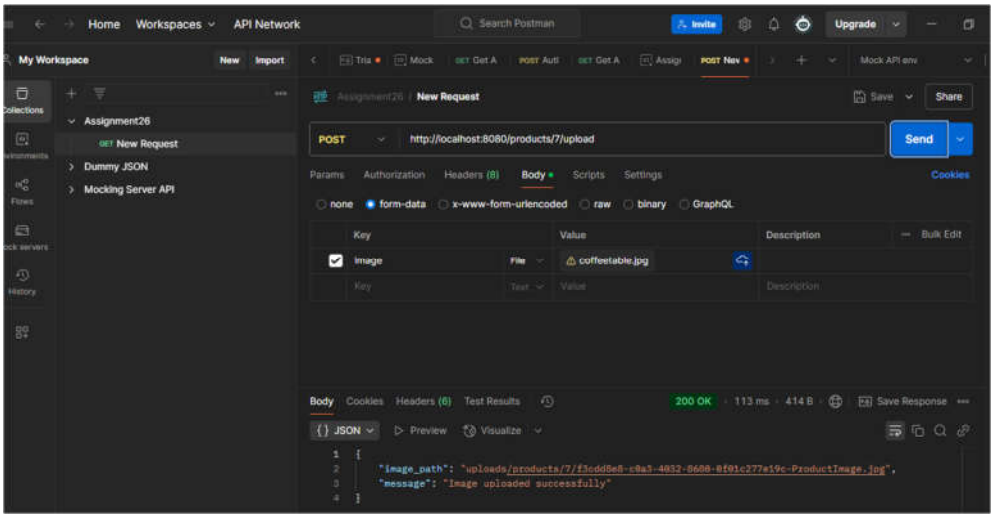
Postman Docs:

POST method: upload images (product 7)



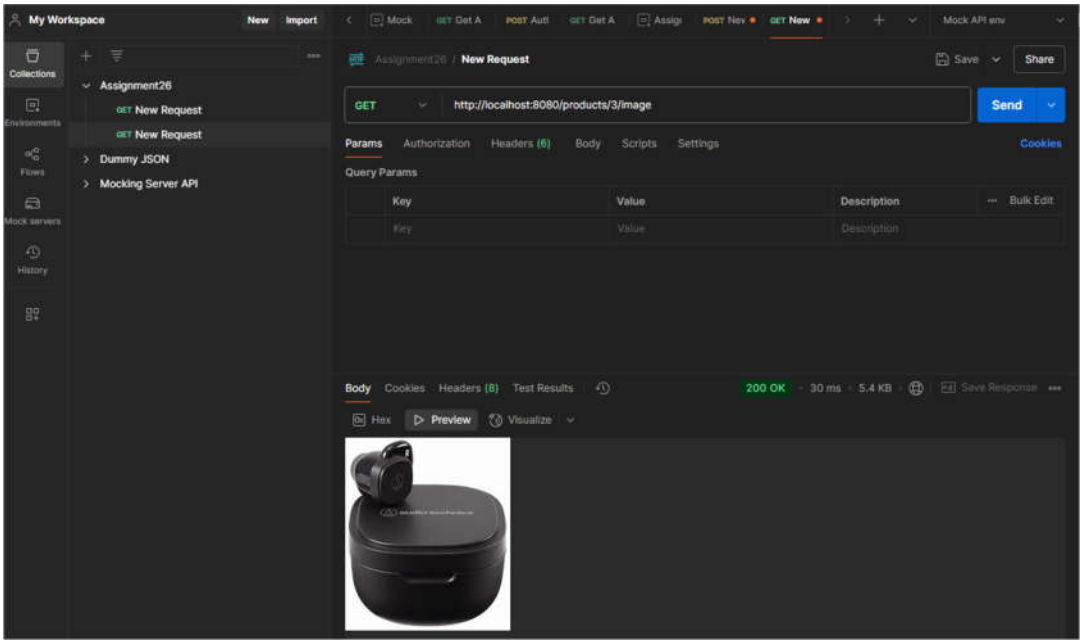


http 200 on POST: upload

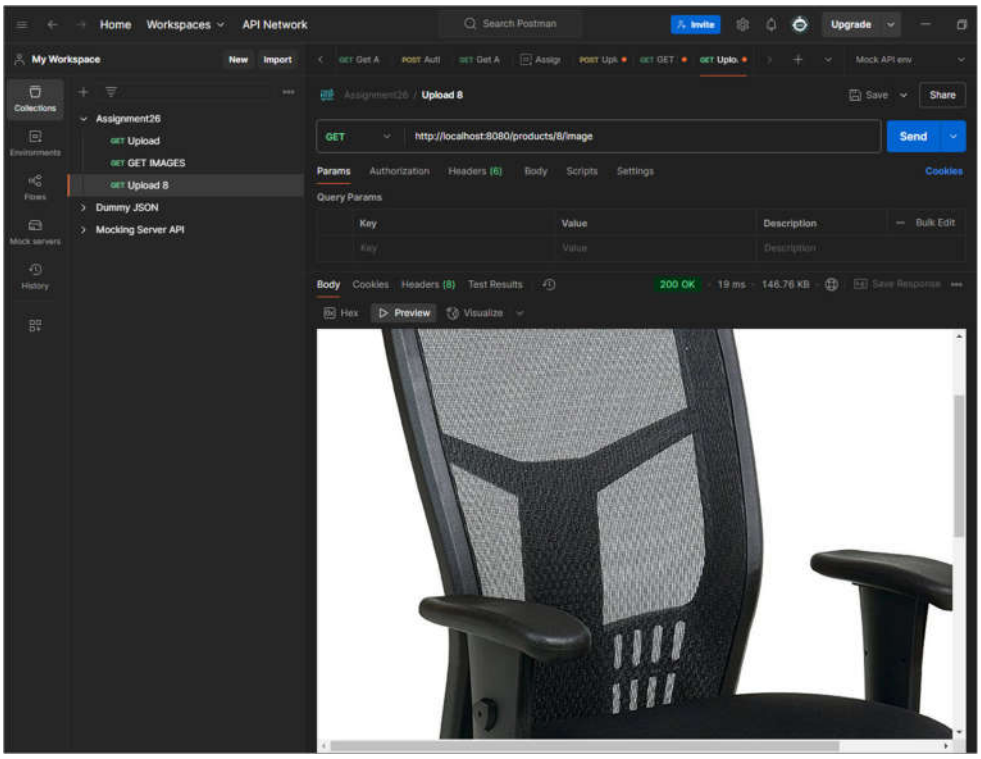


--

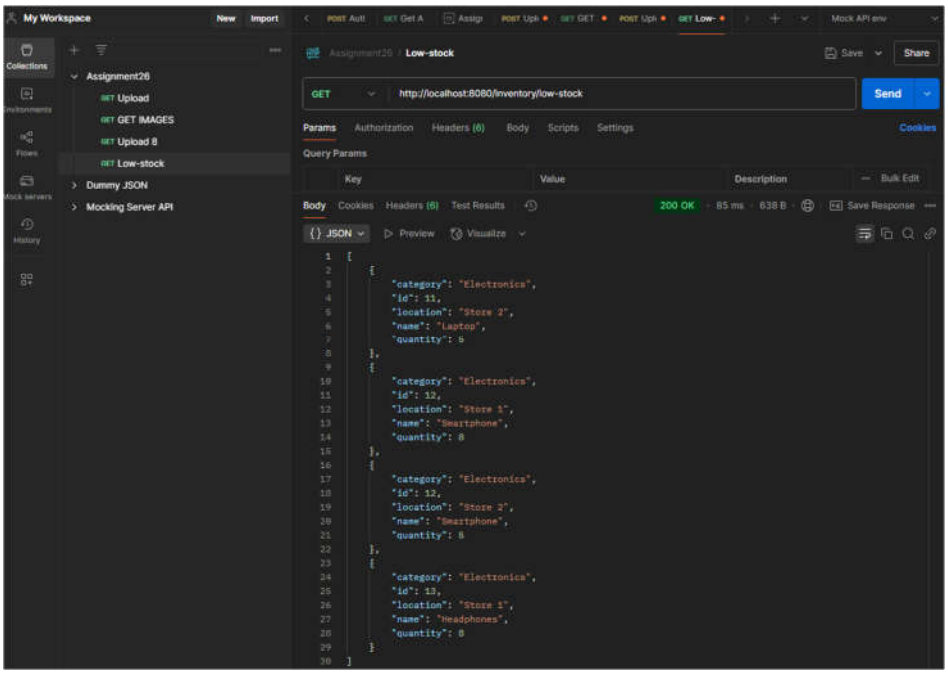
GET: images (products 3): http 200



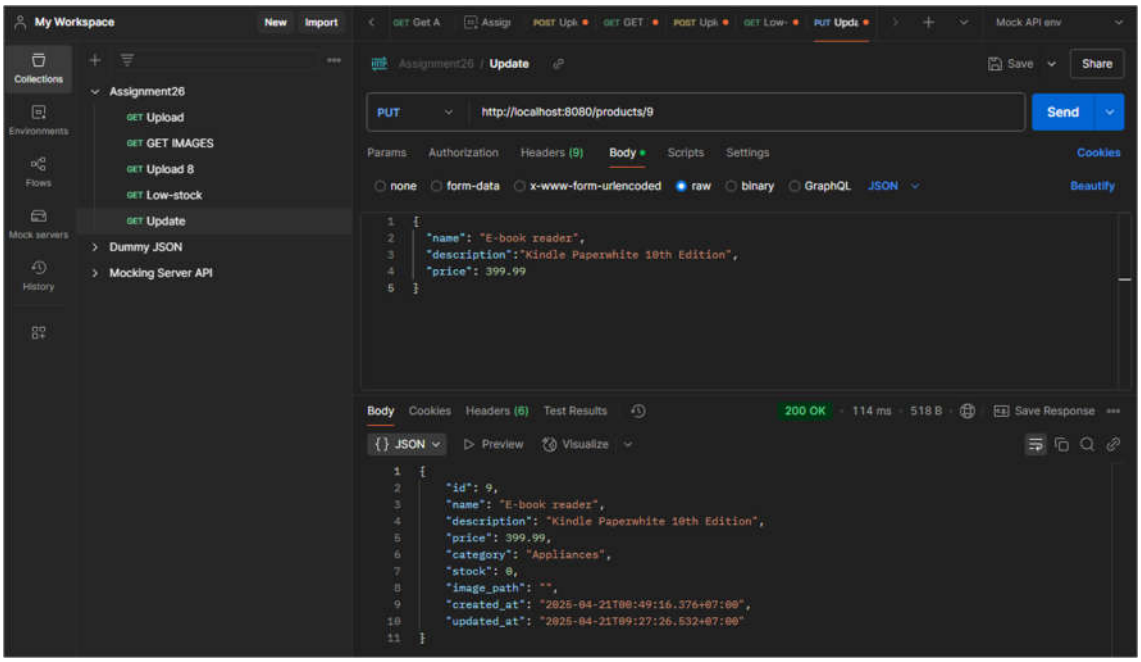
POST method: upload images (product 8): 200



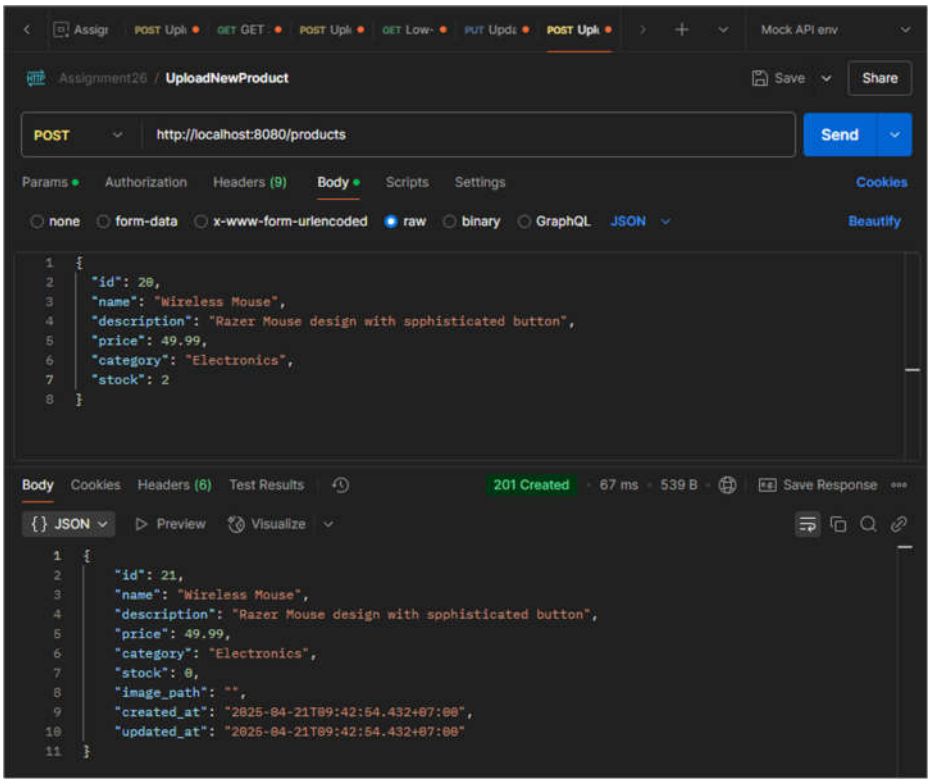
GET method: ./inventory/low-stock: 200



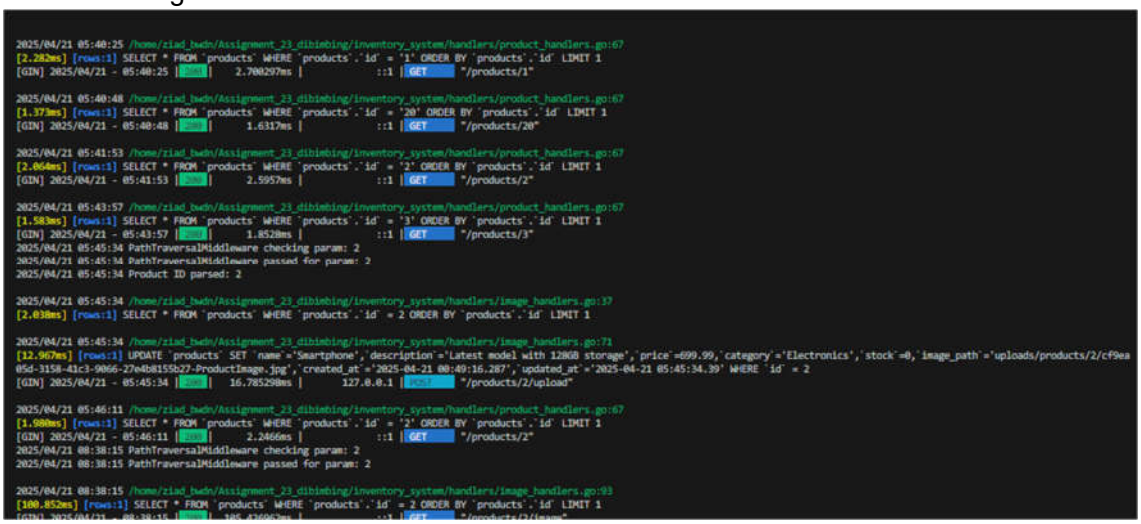
PUT method: ./products/9 (from blender to kindle): 200 OK



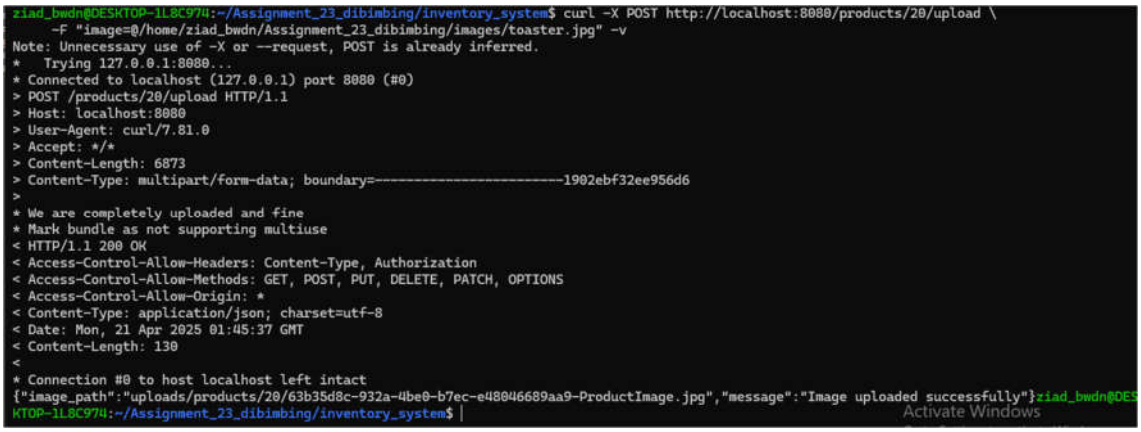
POST method: ./products/21 (Wireless Mouse): 200 OK (id input should be 21 but OK)



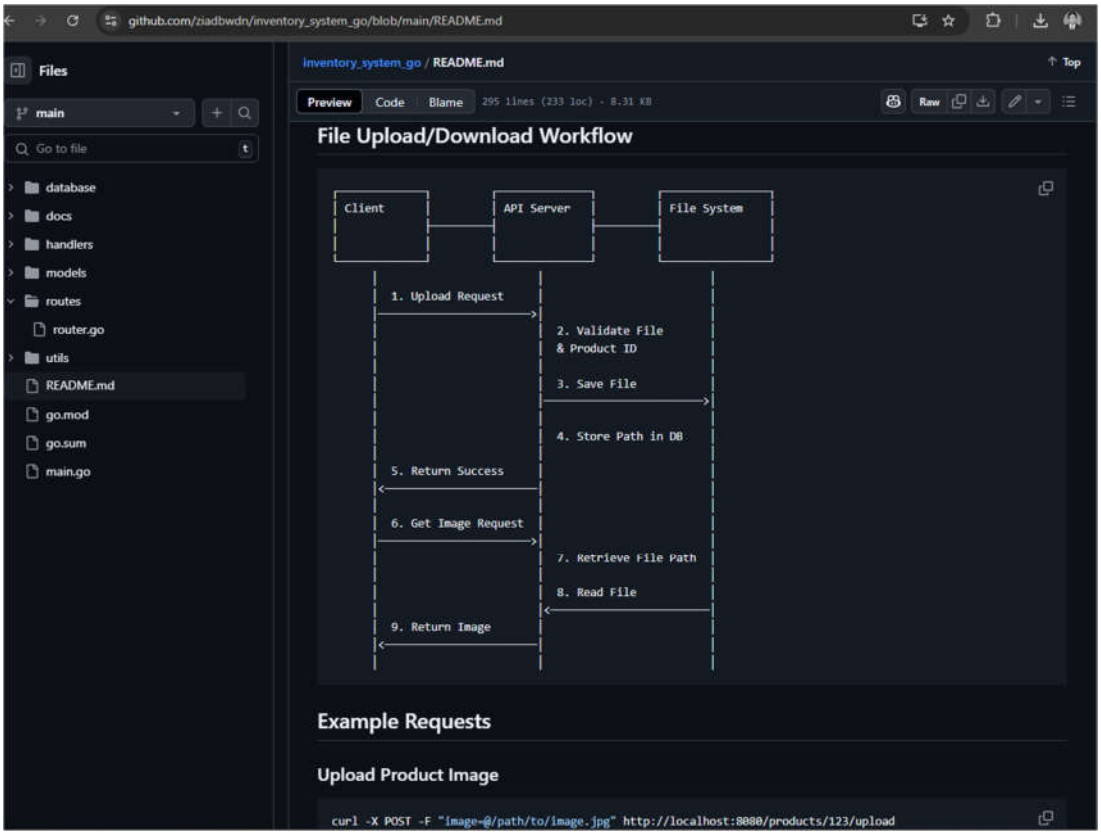
Server running



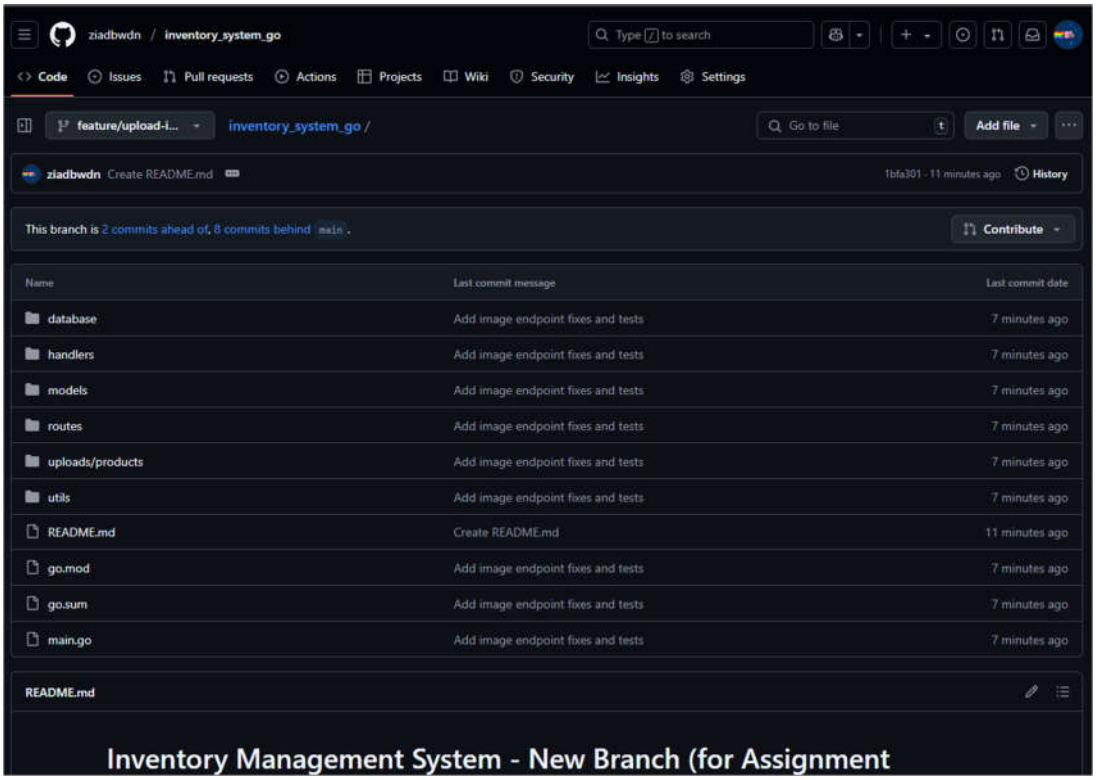
Using Curl Command



Readme.md (updated on GitHub directly)



Github Update on new Branch



README.md

Inventory Management System - New Branch (for Assignment Update Purposes)

A RESTful API for inventory management built with Go, Gin framework, and MySQL.

Features

- Product management (CRUD operations)
- Inventory tracking across multiple locations
- Order processing with automatic inventory updates
- File upload and serving for product images
- Comprehensive data reporting

Prerequisites

- Go 1.19 or higher
- MySQL 8.0 or higher

Project Structure

```
/inventory_system_go
├── database/       # Database configuration and queries
│   ├── db.go
│   ├── queries.go
│   └── scripts/
│     └── schema.sql
├── docs/
│   └── documentation.pdf
├── handlers/       # Gin route logic
│   ├── inventory_handlers.go
│   ├── order_handlers.go
│   ├── product_handlers.go
│   └── image_handlers.go
├── main.go
├── models/        # Structs (Product, Inventory, Order)
│   └── models.go
├── routes/        # Gin router groups
│   └── router.go
├── uploads/       # Product images
│   └── products/
```

```
├── utils/          # Utility functions
│   └── file_utils.go
├── go.mod          # Dependencies (Gin, GORM, MySQL driver)
├── go.sum
└── README.md       # This file
```

Database Setup

Method 1: Run the Schema SQL Script

1. Log in to MySQL:
   ```bash
   mysql -u root -p
   ```

2. Run the schema script:
   ```bash
   mysql -u root -p < database/scripts/schema.sql
   ```

Method 2: Let the Application Handle It

1. Set environment variables for database connection:
   ```bash
   export DB_USER=root
   export DB_PASSWORD=your_password
   export DB_HOST=localhost
   export DB_PORT=3306
   export DB_NAME=inventory
   ```

2. Run the application, which will create the database schema automatically using GORM AutoMigrate.

Getting Started

Installation

1. Clone the repository:
   ```bash
   git clone https://github.com/yourusername/inventory-system.git
   cd inventory-system
   ```

```

2. Install dependencies:
```bash
go mod download
```

3. Set up environment variables:
```bash
export DB_USER=root
export DB_PASSWORD=your_password
export DB_HOST=localhost
export DB_PORT=3306
export DB_NAME=inventory
```

4. Run the application:
```bash
go run main.go
```

5. The server will start at http://localhost:8080

API Documentation

Products

Get all products
```bash
curl -X GET http://localhost:8080/products
```

Get products by category
```bash
curl -X GET "http://localhost:8080/products?category=Electronics"
```

Get products by price range
```bash
curl -X GET "http://localhost:8080/products?min_price=50&max_price=200"
```

Get a specific product
```

```bash
curl -X GET http://localhost:8080/products/1
```

Create a new product
```bash
curl -X POST http://localhost:8080/products \
 -H "Content-Type: application/json" \
 -d '{"name":"Wireless Mouse","description":"Ergonomic wireless
mouse","price":29.99,"category":"Electronics"}'
```

Update a product
```bash
curl -X PUT http://localhost:8080/products/1 \
 -H "Content-Type: application/json" \
 -d '{"name":"Updated Product Name","price":39.99}'
```

Upload a product image
```bash
curl -X POST http://localhost:8080/products/1/upload \
 -F "image=@/path/to/image.jpg"
```

Get a product image
```bash
curl -X GET http://localhost:8080/products/1/image
```

Inventory

Get all inventory
```bash
curl -X GET http://localhost:8080/inventory
```

Get inventory by product
```bash
curl -X GET "http://localhost:8080/inventory?product_id=1"
```

Adjust stock

```bash
curl -X PATCH "http://localhost:8080/inventory/1?location=Warehouse%20A" \
 -H "Content-Type: application/json" \
 -d '{"action":"add","value":10}'
```

Get inventory by location
```bash
curl -X GET http://localhost:8080/inventory/locations
```

Get low stock products
```bash
curl -X GET "http://localhost:8080/inventory/low-stock?threshold=15"
```

Orders

Get all orders
```bash
curl -X GET http://localhost:8080/orders
```

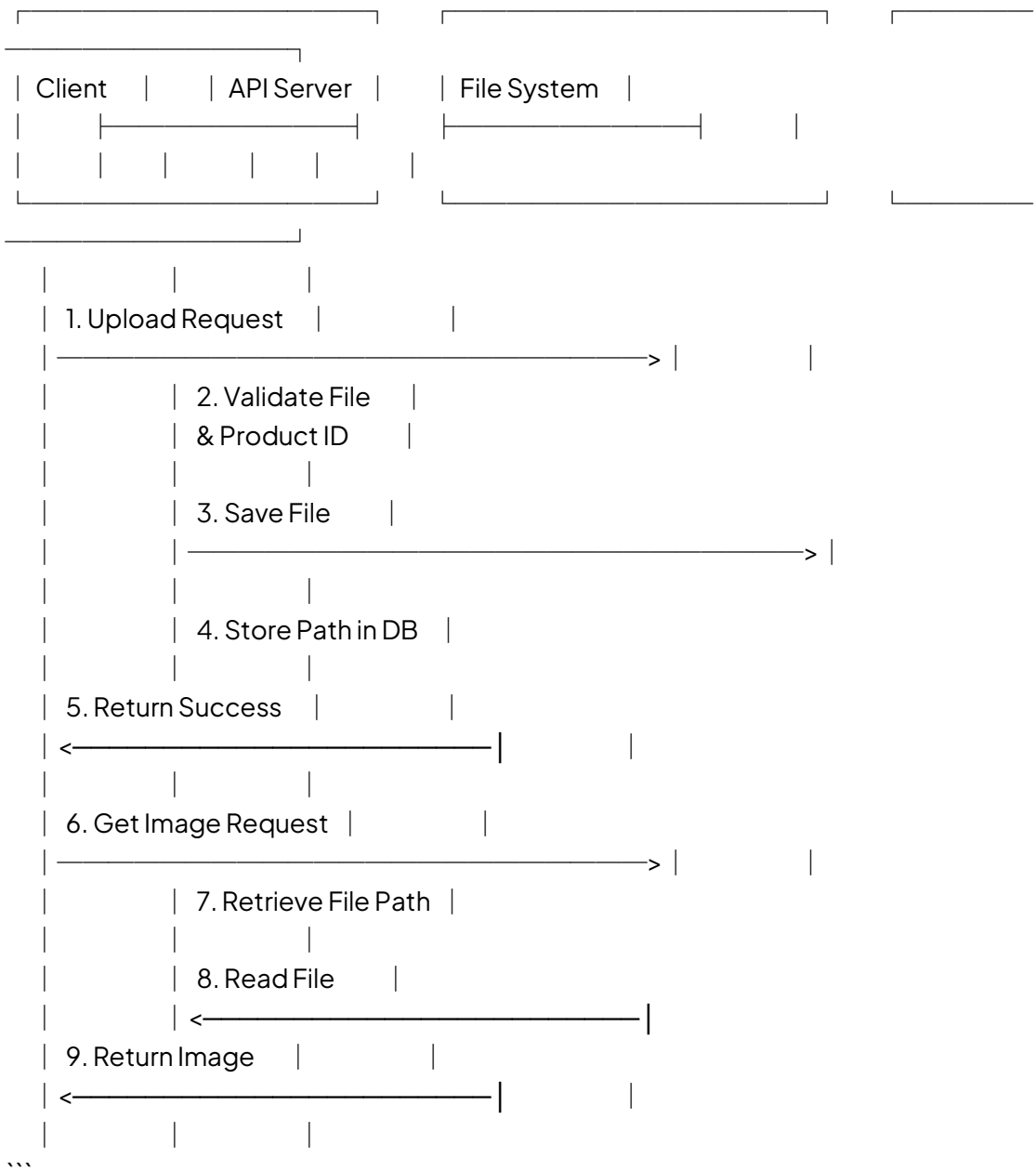
Get a specific order
```bash
curl -X GET http://localhost:8080/orders/1
```

Create a new order
```bash
curl -X POST http://localhost:8080/orders \
 -H "Content-Type: application/json" \
 -d '{"product_id":1,"quantity":2}'
```

Get revenue by category
```bash
curl -X GET http://localhost:8080/orders/revenue
```

File Upload/Download Workflow

```
```



Example Requests

Upload Product Image

```bash
curl -X POST -F "image=@/path/to/image.jpg"
http://localhost:8080/products/123/upload
```

Get Product Image

```bash
curl -X GET http://localhost:8080/products/123/image -o product_image.jpg
```

Security Features

- MIME type validation (not just file extension)
- File size limitation (max 5MB)
- Path traversal prevention
- Secure file storage structure
- Automatic cleanup of orphaned images

Error Handling

The API returns appropriate HTTP status codes and JSON error messages:

- 400 Bad Request: Invalid input or file type
- 404 Not Found: Product not found
- 413 Request Entity Too Large: File too large
- 500 Internal Server Error: Server-side issues

Dependencies

- Gin Web Framework
- GORM ORM
- UUID Generator
- MySQL Driver (or your chosen database)

Database Optimization

The project implements several MySQL-specific optimizations:

1. **Proper Indexing**: Indexes on frequently queried columns
2. **Efficient JOINs**: Using appropriate JOIN types and index hints
3. **Connection Pooling**: Configured for optimal performance
4. **Query Optimization**: Structured queries to take advantage of MySQL's query optimizer

Scaling Considerations

For high-volume applications, consider:

1. **Read Replicas**: Set up MySQL read replicas to distribute query load
2. **Load Balancing**: Deploy multiple API instances behind a load balancer
3. **Caching**: Implement Redis caching for frequently accessed data
4. **Partitioning**: Consider table partitioning for very large datasets