

# Assignment Guidance: Case Study

## Running web server dan route in golang

Golang Backend Development Bootcamp

### Periode Pembelajaran

Introduction to API

API Integration

API Development with Golang: Creating CRUD Operations

Web server dan golang route

### Objectives

1. Student mampu merancang dan melakukan query pada database untuk mengelola produk, inventaris, dan pesanan.
2. Student mampu memahami RESTful API untuk berinteraksi dengan database.
3. Student mampu menghubungkan backend API untuk melakukan operasi CRUD.
4. Student mampu membuat endpoint untuk membuat, membaca, memperbarui, dan menghapus data inventaris.
5. Student mampu mengimplementasikan unggah/unduh file untuk gambar produk.
6. Student mampu menyusun route dan menyajikan API menggunakan framework Gin di Go.

### Deskripsi Assignment

Dalam tugas ini, student akan membangun sistem backend untuk mengelola aplikasi manajemen inventaris. Proyek ini akan melibatkan perancangan dan query pada database relasional untuk mengelola produk, inventaris, dan pesanan. Student akan membuat dan mengintegrasikan RESTful API untuk berinteraksi dengan database, memungkinkan operasi CRUD untuk data inventaris menggunakan Golang.

Selain itu, student akan mengimplementasikan penanganan file untuk mengunggah dan mengunduh gambar produk, serta menyusun route menggunakan framework Gin untuk menyajikan API secara efektif. Dengan menyelesaikan tugas ini, student akan mendapatkan pengalaman langsung dalam pengembangan backend, integrasi API, dan penanganan penyimpanan file, serta menerapkan konsep pemrograman inti dalam skenario dunia nyata.

## Detail Assignment

### 1. Database Design and Queries

- a. Desain sebuah database relasional dengan setidaknya tabel-tabel berikut:
  - Produk: Menyimpan detail produk (ID, nama, deskripsi, harga, dan kategori).
  - Inventaris: Melacak tingkat stok dan lokasi produk (ID produk, jumlah, dan lokasi).
  - Pesanan: Mencatat pesanan pelanggan (ID pesanan, ID produk, jumlah, dan tanggal pesanan).
- b. Write SQL scripts to:
  - Memasukkan data sampel ke dalam tabel-tabel.
  - Melakukan query untuk produk, inventaris, dan detail pesanan.
  - Melakukan agregasi seperti total pesanan untuk suatu produk atau tingkat stok di lokasi tertentu.

### 2. RESTful API Development

Buat API untuk hal-hal berikut:

- Produk:
  - Menambahkan, melihat, memperbarui, dan menghapus produk.
  - Melihat detail produk berdasarkan ID atau kategori.
- Inventaris:
  - Melihat tingkat stok untuk suatu produk.
  - Memperbarui tingkat stok (menambah atau mengurangi stok).
- Pesanan:
  - Membuat pesanan baru.
  - Mengambil detail pesanan berdasarkan ID.

### 3. API Integration

- Uji API untuk memastikan integrasi dan fungsionalitas berjalan lancar.
- Tulis skrip atau gunakan alat (misalnya, Postman, Curl) untuk menguji semua endpoint.

### 4. Web Server and Routing

Gunakan framework Gin di Golang untuk:

- Menyusun route untuk semua endpoint API (misalnya, /products, /inventory, /orders).
- Menangani metode HTTP seperti GET, POST, PUT, dan DELETE.

## 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	Database design and query accuracy.	<ul style="list-style-type: none"><li>- Database dirancang dengan relasi yang sesuai, tipe data yang tepat, dan normalisasi.</li><li>- Query SQL dapat digunakan untuk menyisipkan, mengambil, memperbarui, dan menghapus data dengan benar.</li><li>- Agregasi seperti jumlah pesanan berjalan dengan baik.</li><li>- Implementasi GORM</li></ul>	30
2	API functionality and correctness.	<ul style="list-style-type: none"><li>- Endpoint CRUD (Create, Read, Update, Delete) untuk produk, inventaris, dan pesanan berjalan sesuai spesifikasi.</li><li>- API menangani input tidak valid dengan baik (error handling).</li><li>- Mengembalikan status code dan respons yang sesuai.</li></ul>	40
3	Code structure, readability, and use of Golang best practices.	<ul style="list-style-type: none"><li>- Kode terstruktur dengan modular, mengikuti praktik terbaik Golang.</li><li>- Variabel dan fungsi diberi nama dengan deskriptif.</li><li>- Error handling dilakukan di seluruh kode untuk meminimalkan bug.</li></ul>	20
4	Clear and concise documentation.	<ul style="list-style-type: none"><li>- File README.md mencakup instruksi yang jelas untuk setup database dan menjalankan proyek.</li><li>- Dokumentasi API (endpoint, method, dan contoh respons) dibuat secara ringkas dan jelas.</li></ul>	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.

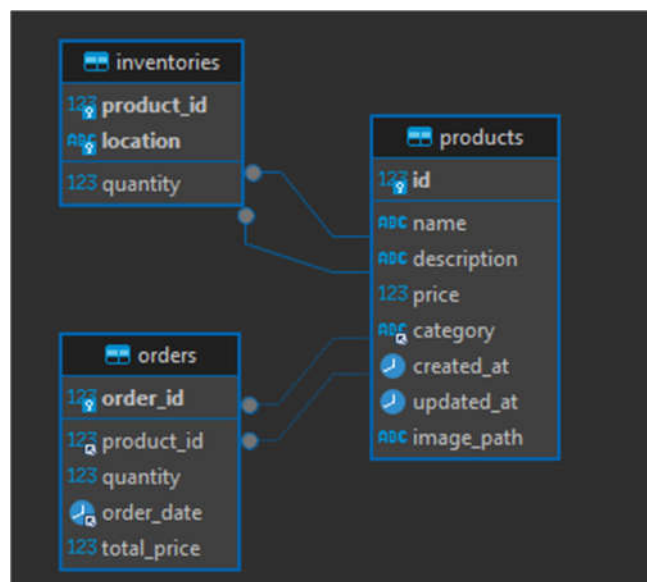
Dokumentasi API (endpoint, method, dan contoh respons) dibuat secara ringkas dan jelas

# Documentation

Folder Structure:

```
ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23_dibimbing/inventory_system$ tree
.
├── database
│   ├── db.go
│   ├── queries.go
│   └── scripts
│       └── schema.sql
├── go.mod
├── go.sum
├── handlers
│   ├── inventory_handlers.go
│   ├── order_handlers.go
│   └── product_handlers.go
├── main.go
├── models
│   └── models.go
├── routes
│   └── router.go
└── uploads
```

Database Design:



## SQL Queries

```
-- database/scripts/schema.sql
-- Run this script to set up the database from scratch

-- Create database if it doesn't exist
CREATE DATABASE IF NOT EXISTS inventory;
USE inventory;

-- Products table
CREATE TABLE IF NOT EXISTS products (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  description TEXT,
  price DECIMAL(10, 2) NOT NULL CHECK (price >= 0),
  category VARCHAR(50) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  image_path VARCHAR(255)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Create index on category for faster filtering
CREATE INDEX idx_products_category ON products(category);

-- Inventory table
CREATE TABLE IF NOT EXISTS inventories (
  product_id INT UNSIGNED NOT NULL,
  location VARCHAR(100) NOT NULL,
  quantity INT NOT NULL DEFAULT 0,
  PRIMARY KEY (product_id, location),
  FOREIGN KEY (product_id) REFERENCES products(id) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Create index on location for faster queries
CREATE INDEX idx_inventories_location ON inventories(location);

-- Orders table
CREATE TABLE IF NOT EXISTS orders (
  order_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  product_id INT UNSIGNED NOT NULL,
  quantity INT NOT NULL CHECK (quantity > 0),
  order_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  total_price DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (product_id) REFERENCES products(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Create index on order_date for faster date range queries
CREATE INDEX idx_orders_date ON orders(order_date);

-- Sample data insertion
-- See seedData() function in Go code for implementation
```

## Dokumentasi:

### Proses setup project

```
2025/04/14 01:14:20 /home/ziaad_buadi/Assignment_23_dibimbing/inventory_system/database/db.go:80
[7.654ms] [rows:1] SELECT count(*) FROM 'products'
2025/04/14 01:14:20 Database connection established successfully
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env: export GIN_MODE=release
- using code: gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /uploads/*filepath      --> github.com/gin-gonic/gin.(*RouterGroup).createStaticHandler.func1 (4 handlers)
[GIN-debug] HEAD    /uploads/*filepath      --> github.com/gin-gonic/gin.(*RouterGroup).createStaticHandler.func1 (4 handlers)
[GIN-debug] GET    /health                  --> inventory_system/routes.SetupRouter.func2 (4 handlers)
[GIN-debug] GET    /products                --> inventory_system/handlers.(*ProductHandler).GetProducts-fm (4 handlers)
[GIN-debug] GET    /products/:id            --> inventory_system/handlers.(*ProductHandler).GetProduct-fm (4 handlers)
[GIN-debug] POST   /products               --> inventory_system/handlers.(*ProductHandler).CreateProduct-fm (4 handlers)
[GIN-debug] PUT    /products/:id            --> inventory_system/handlers.(*ProductHandler).UpdateProduct-fm (4 handlers)
[GIN-debug] POST   /products/:id/upload     --> inventory_system/handlers.(*ProductHandler).UploadProductImage-fm (4 handlers)
[GIN-debug] GET    /products/:id/image      --> inventory_system/handlers.(*ProductHandler).GetProductImage-fm (4 handlers)
[GIN-debug] GET    /inventory               --> inventory_system/handlers.(*InventoryHandler).GetInventory-fm (4 handlers)
[GIN-debug] PATCH  /inventory/:product_id   --> inventory_system/handlers.(*InventoryHandler).AdjustStock-fm (4 handlers)
[GIN-debug] GET    /inventory/locations      --> inventory_system/handlers.(*InventoryHandler).GetInventoryByLocation-fm (4 handlers)
[GIN-debug] GET    /inventory/low-stock     --> inventory_system/handlers.(*InventoryHandler).GetLowStockProducts-fm (4 handlers)
[GIN-debug] GET    /orders                  --> inventory_system/handlers.(*OrderHandler).GetOrders-fm (4 handlers)
[GIN-debug] GET    /orders/:id              --> inventory_system/handlers.(*OrderHandler).GetOrder-fm (4 handlers)
[GIN-debug] POST   /orders                  --> inventory_system/handlers.(*OrderHandler).CreateOrder-fm (4 handlers)
[GIN-debug] GET    /orders/revenue          --> inventory_system/handlers.(*OrderHandler).GetRevenueByCategory-fm (4 handlers)

2025/04/14 01:14:20 Server starting on port 8080...
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :8080
[GIN] 2025/04/14 - 01:16:57 | 200 | 113.9µs | ::1 | GET | "/health"
[GIN] 2025/04/14 - 01:16:57 | 200 | 5µs | ::1 | GET | "/favicon.ico"

2025/04/14 01:17:08 /home/ziaad_buadi/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:54
[2.888ms] [rows:10] SELECT * FROM 'products'
[GIN] 2025/04/14 - 01:17:08 | 200 | 2.459199ms | ::1 | GET | "/products"

2025/04/14 01:17:41 /home/ziaad_buadi/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:68 Error 1064 (42000): You have an error in your SQL syntax; check the ma
nual that corresponds to your MySQL server version for the right syntax to use near 'product_id ORDER BY 'products'. 'id LIMIT 1' at line 1
[1.122ms] [rows:0] SELECT * FROM 'products' WHERE 'product_id ORDER BY 'products'. 'id LIMIT 1
[GIN] 2025/04/14 - 01:17:41 | 400 | 3.412ms | ::1 | GET | "/products/{product_id}"

2025/04/14 01:17:53 /home/ziaad_buadi/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:68
[12.530ms] [rows:1] SELECT * FROM 'products' WHERE 'products'. 'id' = '4' ORDER BY 'products'. 'id LIMIT 1
[GIN] 2025/04/14 - 01:17:53 | 200 | 13.016409ms | ::1 | GET | "/products/4"

Activate Windows
Go to Settings to activate Windows.
```

### Status HTTP 200

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env: export GIN_MODE=release
- using code: gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /uploads/*filepath      --> github.com/gin-gonic/gin.(*RouterGroup).createStaticHandler.func1 (4 handlers)
[GIN-debug] HEAD    /uploads/*filepath      --> github.com/gin-gonic/gin.(*RouterGroup).createStaticHandler.func1 (4 handlers)
[GIN-debug] GET    /health                  --> inventory_system/routes.SetupRouter.func2 (4 handlers)
[GIN-debug] GET    /products                --> inventory_system/handlers.(*ProductHandler).GetProducts-fm (4 handlers)
[GIN-debug] GET    /products/:id            --> inventory_system/handlers.(*ProductHandler).GetProduct-fm (4 handlers)
[GIN-debug] POST   /products               --> inventory_system/handlers.(*ProductHandler).CreateProduct-fm (4 handlers)
[GIN-debug] PUT    /products/:id            --> inventory_system/handlers.(*ProductHandler).UpdateProduct-fm (4 handlers)
[GIN-debug] POST   /products/:id/upload     --> inventory_system/handlers.(*ProductHandler).UploadProductImage-fm (4 handlers)
[GIN-debug] GET    /products/:id/image      --> inventory_system/handlers.(*ProductHandler).GetProductImage-fm (4 handlers)
[GIN-debug] GET    /inventory               --> inventory_system/handlers.(*InventoryHandler).GetInventory-fm (4 handlers)
[GIN-debug] PATCH  /inventory/:product_id   --> inventory_system/handlers.(*InventoryHandler).AdjustStock-fm (4 handlers)
[GIN-debug] GET    /inventory/locations      --> inventory_system/handlers.(*InventoryHandler).GetInventoryByLocation-fm (4 handlers)
[GIN-debug] GET    /inventory/low-stock     --> inventory_system/handlers.(*InventoryHandler).GetLowStockProducts-fm (4 handlers)
[GIN-debug] GET    /orders                  --> inventory_system/handlers.(*OrderHandler).GetOrders-fm (4 handlers)
[GIN-debug] GET    /orders/:id              --> inventory_system/handlers.(*OrderHandler).GetOrder-fm (4 handlers)
[GIN-debug] POST   /orders                  --> inventory_system/handlers.(*OrderHandler).CreateOrder-fm (4 handlers)
[GIN-debug] GET    /orders/revenue          --> inventory_system/handlers.(*OrderHandler).GetRevenueByCategory-fm (4 handlers)

2025/04/14 05:51:23 Server starting on port 8080...
[GIN-debug] [WARNING] You trusted all proxies, this is NOT safe. We recommend you to set a value.
Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.
[GIN-debug] Listening and serving HTTP on :8080

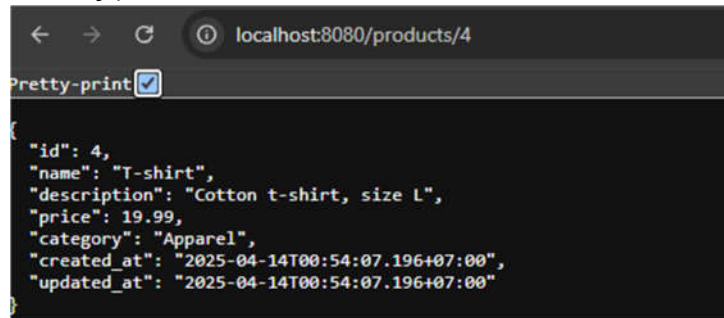
2025/04/14 05:53:58 /home/ziaad_buadi/Assignment_23_dibimbing/inventory_system/database/queries.go:34
[10.437ms] [rows:5] SELECT products.category, SUM(orders.total_price) as total_revenue, COUNT(orders.order_id) as order_count FROM 'orders' JOIN products ON order
s.product_id = products.id GROUP BY 'products'. 'category'
2025/04/14 05:53:58 DEBUG: GetRevenueByCategory Success. Rows found: 5
[GIN] 2025/04/14 - 05:53:58 | 200 | 11.2722ms | 127.0.0.1 | GET | "/orders/revenue"

2025/04/14 05:54:25 /home/ziaad_buadi/Assignment_23_dibimbing/inventory_system/database/queries.go:34
[10.437ms] [rows:5] SELECT products.category, SUM(orders.total_price) as total_revenue, COUNT(orders.order_id) as order_count FROM 'orders' JOIN products ON order
s.product_id = products.id GROUP BY 'products'. 'category'
2025/04/14 05:54:25 DEBUG: GetRevenueByCategory Success. Rows found: 5
[GIN] 2025/04/14 - 05:54:25 | 200 | 7.5872ms | ::1 | GET | "/orders/revenue"

2025/04/14 05:56:08 /home/ziaad_buadi/Assignment_23_dibimbing/inventory_system/database/queries.go:55
[11.019ms] [rows:4] SELECT location, SUM(quantity) as total_stock, COUNT(DISTINCT product_id) as product_count FROM 'inventories' GROUP BY 'location'
[GIN] 2025/04/14 - 05:56:08 | 200 | 13.0451ms | 127.0.0.1 | GET | "/inventory/locations"
```




GET (by products)



```
localhost:8080/products/4
Pretty-print ☒
{
  "id": 4,
  "name": "T-shirt",
  "description": "Cotton t-shirt, size L",
  "price": 19.99,
  "category": "Apparel",
  "created_at": "2025-04-14T00:54:07.196+07:00",
  "updated_at": "2025-04-14T00:54:07.196+07:00"
}
```

GET (by price range)



```
localhost:8080/products?min_price=50&max_price=200
Pretty-print ☒
[
  {
    "id": 3,
    "name": "Headphones",
    "description": "Wireless noise-cancelling headphones",
    "price": 199.99,
    "category": "Electronics",
    "created_at": "2025-04-14T00:54:07.181+07:00",
    "updated_at": "2025-04-14T00:54:07.181+07:00"
  },
  {
    "id": 6,
    "name": "Sneakers",
    "description": "Running shoes, size 10",
    "price": 89.99,
    "category": "Footwear",
    "created_at": "2025-04-14T00:54:07.223+07:00",
    "updated_at": "2025-04-14T00:54:07.223+07:00"
  },
  {
    "id": 7,
    "name": "Coffee Table",
    "description": "Wooden coffee table",
    "price": 149.99,
    "category": "Furniture",
    "created_at": "2025-04-14T00:54:07.24+07:00",
    "updated_at": "2025-04-14T00:54:07.24+07:00"
  },
  {
    "id": 8,
    "name": "Desk Chair",
    "description": "Ergonomic office chair",
    "price": 199.99,
    "category": "Furniture",
    "created_at": "2025-04-14T00:54:07.26+07:00",
    "updated_at": "2025-04-14T00:54:07.26+07:00"
  },
  {
    "id": 9,
    "name": "Blender",
    "description": "High-speed blender for smoothies",
    "price": 79.99,
    "category": "Appliances",
    "created_at": "2025-04-14T00:54:07.274+07:00",
    "updated_at": "2025-04-14T00:54:07.274+07:00"
  }
]
```



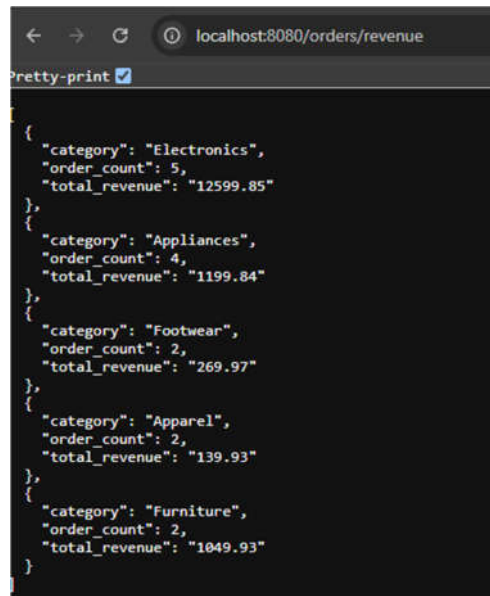
## POST

```
5-04-14T00:54:07.274+07:00}}ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23_dibimbing/inventory_system$ 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"}'
{"id":11,"name":"Wireless Mouse","description":"Ergonomic wireless mouse","price":29.99,"category":"Electronics","created_at":"2025-04-14T01:28:51.291+07:00","updated_at":"2025-04-14T01:28:51.291+07:00"}ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23_dibimbing/inventory_system$ tree
```

## PUT

```
5-04-14T00:54:07.274+07:00}}ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23_dibimbing/inventory_system$ curl -X PUT http://localhost:8080/products \
-H "Content-Type: application/json" \
-d '{"name":"Wireless Mouse","description":"Ergonomic wireless mouse","price":29.99,"category":"Electronics"}'
{"id":11,"name":"Wireless Mouse","description":"Ergonomic wireless mouse","price":29.99,"category":"Electronics","created_at":"2025-04-14T01:28:51.291+07:00","updated_at":"2025-04-14T01:28:51.291+07:00"}ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23_dibimbing/inventory_system$ tree
```

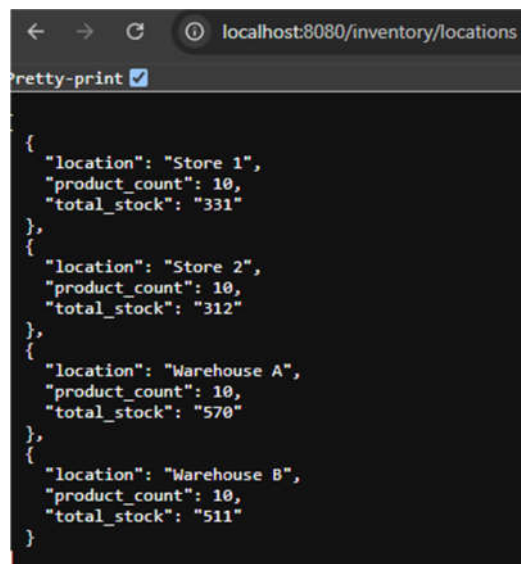
## GET (revenue)



```
localhost:8080/orders/revenue
Pretty-print ☒

[
  {
    "category": "Electronics",
    "order_count": 5,
    "total_revenue": "12599.85"
  },
  {
    "category": "Appliances",
    "order_count": 4,
    "total_revenue": "1199.84"
  },
  {
    "category": "Footwear",
    "order_count": 2,
    "total_revenue": "269.97"
  },
  {
    "category": "Apparel",
    "order_count": 2,
    "total_revenue": "139.93"
  },
  {
    "category": "Furniture",
    "order_count": 2,
    "total_revenue": "1049.93"
  }
]
```

## GET (location)



```
localhost:8080/inventory/locations
Pretty-print ☒

[
  {
    "location": "Store 1",
    "product_count": 10,
    "total_stock": "331"
  },
  {
    "location": "Store 2",
    "product_count": 10,
    "total_stock": "312"
  },
  {
    "location": "Warehouse A",
    "product_count": 10,
    "total_stock": "570"
  },
  {
    "location": "Warehouse B",
    "product_count": 10,
    "total_stock": "511"
  }
]
```

## README.md

### Sistem Manajemen Inventaris

RESTful API untuk manajemen inventaris yang dibangun dengan Go, kerangka kerja Gin, dan MySQL.

#### Fitur

- Manajemen produk (operasi CRUD)
- Pelacakan inventaris di berbagai lokasi
- Pemrosesan pesanan dengan pembaruan inventaris otomatis
- Pengunggahan dan penyajian file untuk gambar produk
- Pelaporan data yang komprehensif

#### Prasyarat

- Go 1.19 atau lebih tinggi
- MySQL 8.0 atau lebih tinggi

#### Struktur Project

/inventory-system

```
.
├── database/    # Database configuration and queries
│   └── scripts/ # SQL scripts for database setup
├── handlers/    # Gin route logic
├── models/      # Structs (Product, Inventory, Order)
├── routes/      # Gin router groups
├── uploads/     # Product images
├── go.mod       # Dependencies (Gin, GORM, MySQL driver)
└── README.md   # This file
```

#### Setup Database

Cara 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
```

## Cara 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

### Instalasi

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>

## Dokumentasi API

### 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
```

Optimisasi Database

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