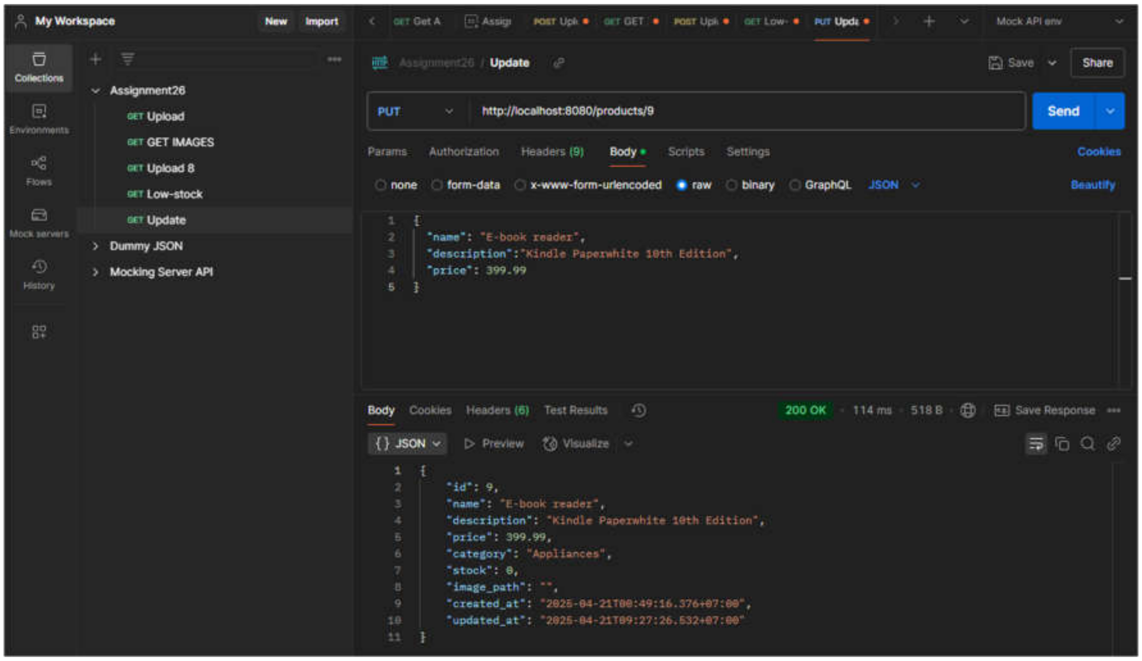
PUT method: ./products/9 (from blender to kindle): 200 OK



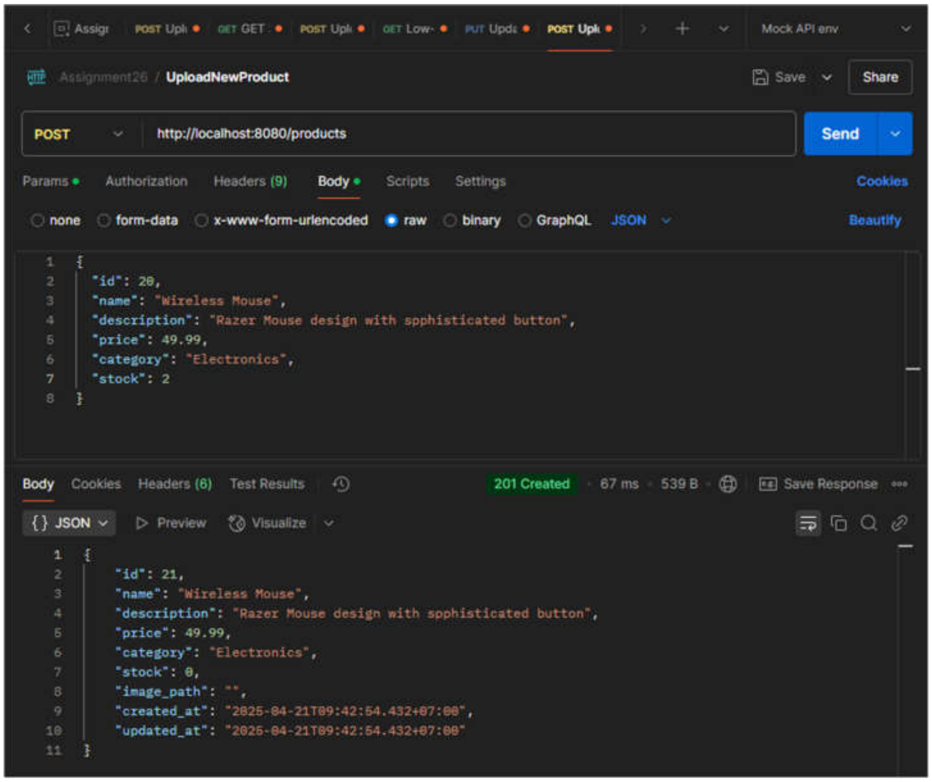
PUT http://localhost:8080/products/9

```
{
  "name": "E-book reader",
  "description":"Kindle Paperwhite 10th Edition",
  "price": 399.99
}
```

200 OK · 114 ms · 518 B

```
{
    "id": 9,
    "name": "E-book reader",
    "description": "Kindle Paperwhite 10th Edition",
    "price": 399.99,
    "category": "Appliances",
    "stock": 0,
    "image_path": "",
    "created_at": "2025-04-21T08:49:16.376+07:00",
    "updated_at": "2025-04-21T09:27:26.532+07:00"
}
```

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



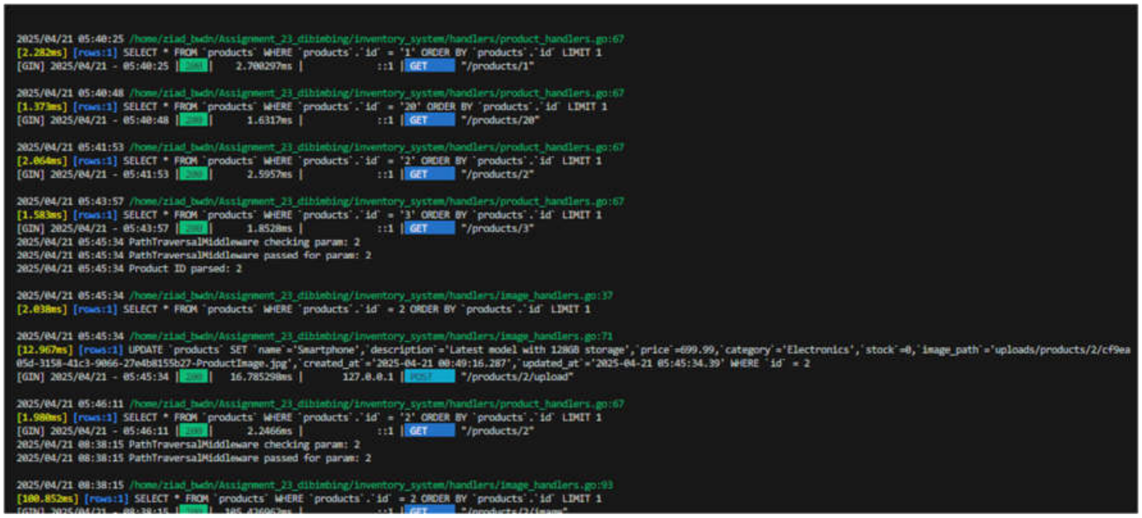
POST http://localhost:8080/products

```
{
  "id": 20,
  "name": "Wireless Mouse",
  "description": "Razer Mouse design with spphisticated button",
  "price": 49.99,
  "category": "Electronics",
  "stock": 2
}
```

201 Created · 67 ms · 539 B

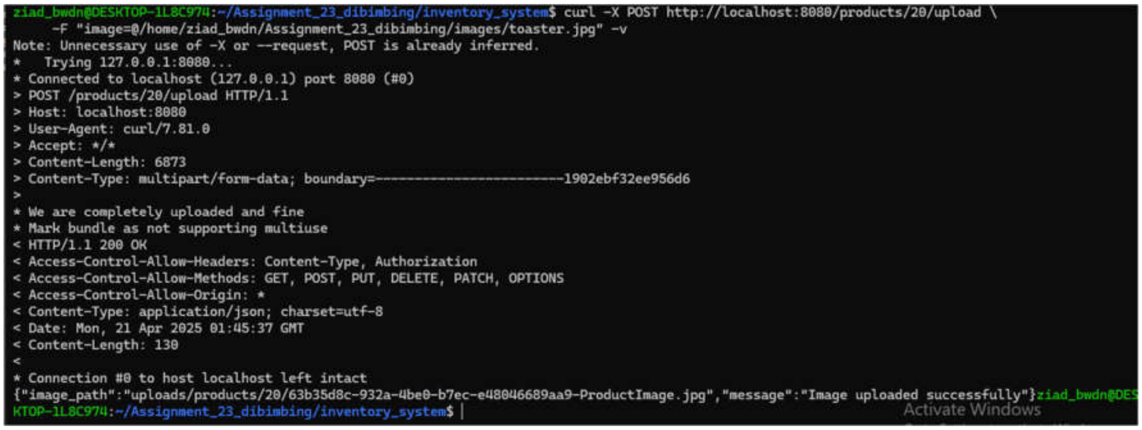
```
{
    "id": 21,
    "name": "Wireless Mouse",
    "description": "Razer Mouse design with spphisticated button",
    "price": 49.99,
    "category": "Electronics",
    "stock": 0,
    "image_path": "",
    "created_at": "2025-04-21T09:42:54.432+07:00",
    "updated_at": "2025-04-21T09:42:54.432+07:00"
}
```

Server running



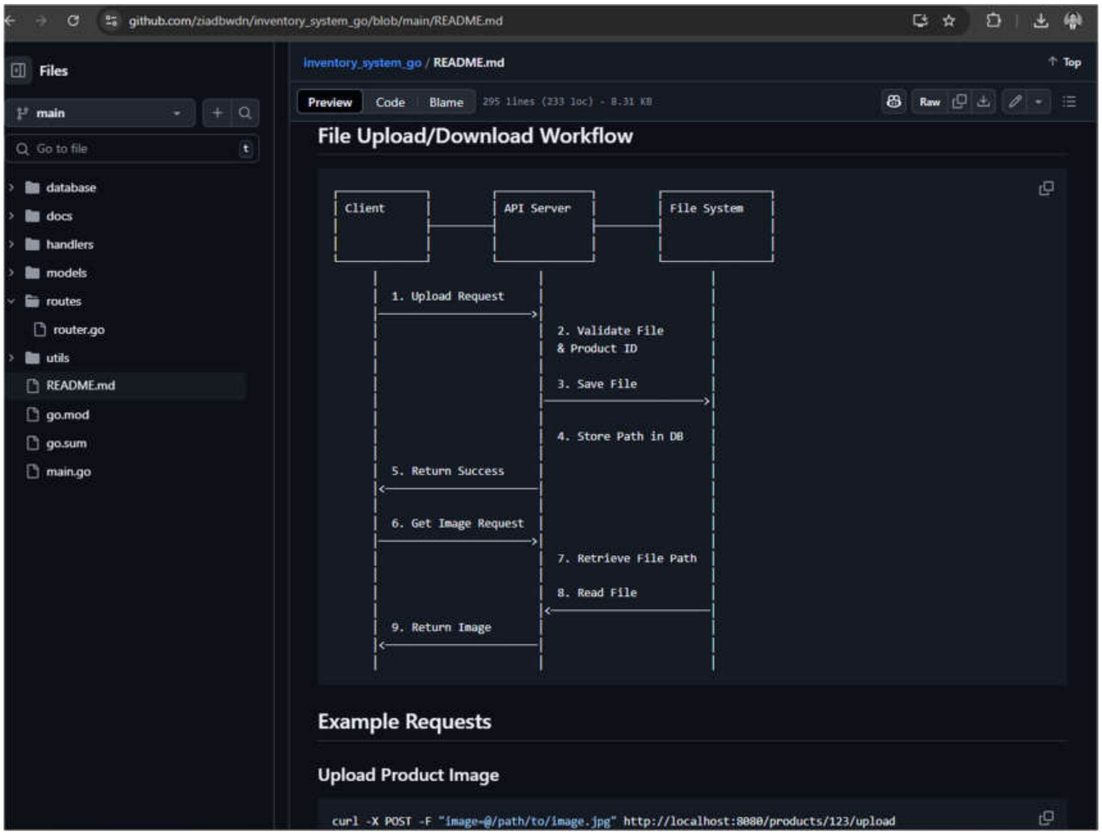
```
2025/04/21 05:40:25 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:67
[2.282ms] [rows:1] SELECT * FROM `products` WHERE `products`.`id` = '1' ORDER BY `products`.`id` LIMIT 1
[GIN] 2025/04/21 - 05:40:25 | 200 |    2.700297ms |             ::1 | GET      "/products/1"

2025/04/21 05:40:48 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:67
[1.373ms] [rows:1] SELECT * FROM `products` WHERE `products`.`id` = '20' ORDER BY `products`.`id` LIMIT 1
[GIN] 2025/04/21 - 05:40:48 | 200 |     1.6317ms |             ::1 | GET      "/products/20"

2025/04/21 05:41:53 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:67
[2.064ms] [rows:1] SELECT * FROM `products` WHERE `products`.`id` = '2' ORDER BY `products`.`id` LIMIT 1
[GIN] 2025/04/21 - 05:41:53 | 200 |     2.5957ms |             ::1 | GET      "/products/2"

2025/04/21 05:43:57 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:67
[1.583ms] [rows:1] SELECT * FROM `products` WHERE `products`.`id` = '3' ORDER BY `products`.`id` LIMIT 1
[GIN] 2025/04/21 - 05:43:57 | 200 |     1.8528ms |             ::1 | GET      "/products/3"
2025/04/21 05:45:34 PathTraversalMiddleware checking param: 2
2025/04/21 05:45:34 PathTraversalMiddleware passed for param: 2
2025/04/21 05:45:34 Product ID parsed: 2

2025/04/21 05:45:34 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/image_handlers.go:37
[2.038ms] [rows:1] SELECT * FROM `products` WHERE `products`.`id` = 2 ORDER BY `products`.`id` LIMIT 1

2025/04/21 05:45:34 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/image_handlers.go:71
[12.967ms] [rows:1] UPDATE `products` SET `name`='Smartphone',`description`='Latest model with 128GB storage',`price`=699.99,`category`='Electronics',`stock`=0,`image_path`='uploads/products/2/cf9ea
05d-3158-41c3-9066-27e4b8155b27-ProductImage.jpg',`created_at`='2025-04-21 00:49:16.287',`updated_at`='2025-04-21 05:45:34.39' WHERE `id` = 2
[GIN] 2025/04/21 - 05:45:34 | 200 |  16.785298ms |       127.0.0.1 | POST     "/products/2/upload"

2025/04/21 05:46:11 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/product_handlers.go:67
[1.980ms] [rows:1] SELECT * FROM `products` WHERE `products`.`id` = '2' ORDER BY `products`.`id` LIMIT 1
[GIN] 2025/04/21 - 05:46:11 | 200 |     2.2466ms |             ::1 | GET      "/products/2"
2025/04/21 08:38:15 PathTraversalMiddleware checking param: 2
2025/04/21 08:38:15 PathTraversalMiddleware passed for param: 2

2025/04/21 08:38:15 /home/ziad_bwdn/Assignment_23_dibimbing/inventory_system/handlers/image_handlers.go:93
[100.852ms] [rows:1] SELECT * FROM `products` WHERE `products`.`id` = 2 ORDER BY `products`.`id` LIMIT 1
[GIN] 2025/04/21 - 08:38:15 | 200 | 105.436962ms |             ::1 | GET      "/products/2/image"
```

Using Curl Command



```
ziad_bwdn@DESKTOP-1L8C974:~/Assignment_23_dibimbing/inventory_system$ curl -X POST http://localhost:8080/products/20/upload \
     -F "image=@/home/ziad_bwdn/Assignment_23_dibimbing/images/toaster.jpg" -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST /products/20/upload HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Length: 6873
> Content-Type: multipart/form-data; boundary=------------------------1902ebf32ee956d6
>
* We are completely uploaded and fine
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Access-Control-Allow-Headers: Content-Type, Authorization
< Access-Control-Allow-Methods: GET, POST, PUT, DELETE, PATCH, OPTIONS
< Access-Control-Allow-Origin: *
< Content-Type: application/json; charset=utf-8
< Date: Mon, 21 Apr 2025 01:45:37 GMT
< Content-Length: 130
<
* Connection #0 to host localhost left intact
{"image_path":"uploads/products/20/63b35d8c-932a-4be0-b7ec-e48046689aa9-ProductImage.jpg","message":"Image uploaded successfully"}ziad_bwdn@DES
KTOP-1L8C974:~/Assignment_23_dibimbing/inventory_system$ |
```

Readme.md (updated on GitHub directly)


github.com/ziadbwdn/inventory_system_go/blob/main/README.md
Files
main
Go to file
database
docs
handlers
models
routes
router.go
utils
README.md
go.mod
go.sum
main.go
inventory_system_go / README.md
Top
Preview
Code
Blame
295 lines (233 loc) · 8.31 KB
Raw

## File Upload/Download Workflow

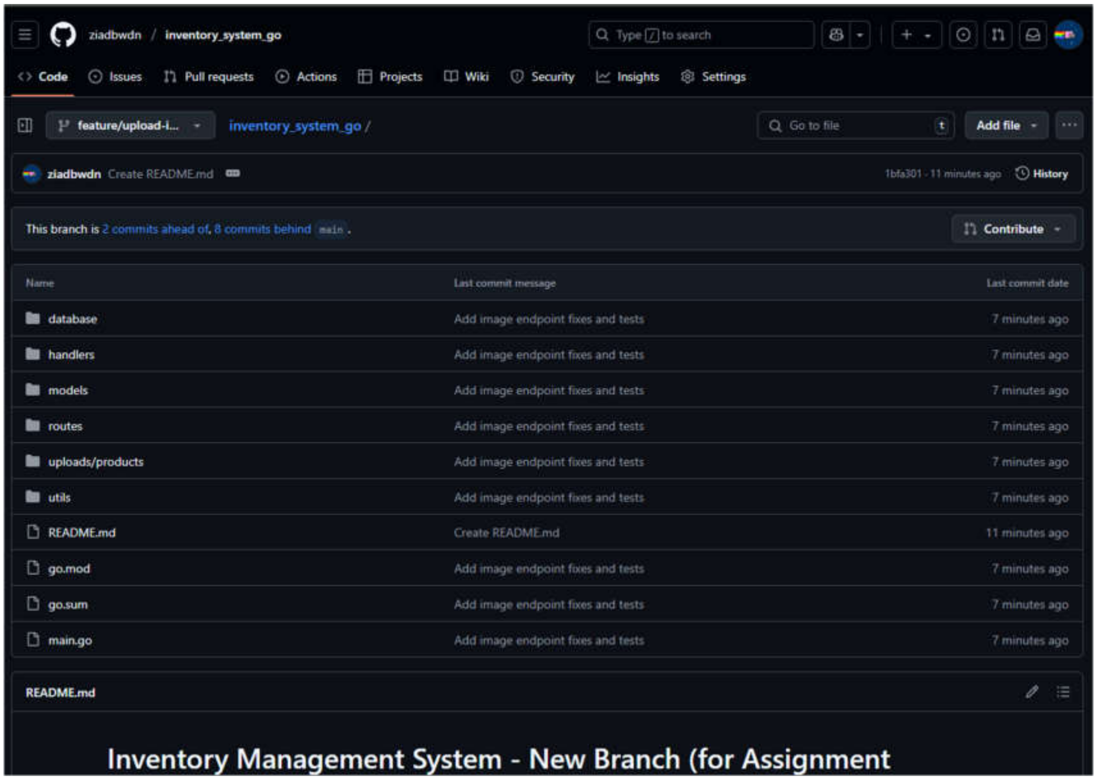
```
┌──────────┐     ┌──────────┐     ┌─────────────┐
│ Client   │     │API Server│     │ File System │
│          │─────│          │─────│             │
│          │     │          │     │             │
└──────────┘     └──────────┘     └─────────────┘
     │                 │                 │
     │ 1. Upload Request                 │
     │────────────────>│                 │
     │                 │ 2. Validate File│
     │                 │ & Product ID    │
     │                 │                 │
     │                 │ 3. Save File    │
     │                 │────────────────>│
     │                 │                 │
     │                 │ 4. Store Path in DB
     │                 │                 │
     │ 5. Return Success                 │
     │<────────────────│                 │
     │                 │                 │
     │ 6. Get Image Request              │
     │────────────────>│                 │
     │                 │ 7. Retrieve File Path
     │                 │                 │
     │                 │ 8. Read File    │
     │                 │<────────────────│
     │ 9. Return Image │                 │
     │<────────────────│                 │
     │                 │                 │
```

## Example Requests

### Upload Product Image

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

Github Update on new Branch



ziadbwdn Create README.md 1bfa301 · 11 minutes ago

This branch is 2 commits ahead of, 8 commits behind `main`.

| Name | Last commit message | Last commit date |
|---|---|---|
| database | Add image endpoint fixes and tests | 7 minutes ago |
| handlers | Add image endpoint fixes and tests | 7 minutes ago |
| models | Add image endpoint fixes and tests | 7 minutes ago |
| routes | Add image endpoint fixes and tests | 7 minutes ago |
| uploads/products | Add image endpoint fixes and tests | 7 minutes ago |
| utils | Add image endpoint fixes and tests | 7 minutes ago |
| README.md | Create README.md | 11 minutes ago |
| go.mod | Add image endpoint fixes and tests | 7 minutes ago |
| go.sum | Add image endpoint fixes and tests | 7 minutes ago |
| main.go | Add image endpoint fixes and tests | 7 minutes ago |

README.md

# Inventory Management System - New Branch (for Assignment

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

```
```

```
┌──────────────────┐    ┌──────────────────┐    ┌───────
───────────────┐
│  Client    │    │  API Server  │    │  File System  │
│      ├──────────────┤    ├──────────────┤      │
│      │    │    │    │     │
└──────────────────┘    └──────────────────┘    └───────
───────────────┘
   │          │          │
   │  1. Upload Request  │          │
   │ ──────────────────────────────> │          │
   │          │  2. Validate File  │
   │          │  & Product ID   │
   │          │          │
   │          │  3. Save File   │
   │          │ ──────────────────────────────> │
   │          │          │
   │          │  4. Store Path in DB  │
   │          │          │
   │  5. Return Success  │          │
   │ <────────────────────┃          │
   │          │          │
   │  6. Get Image Request  │          │
   │ ──────────────────────────────> │          │
   │          │  7. Retrieve File Path  │
   │          │          │
   │          │  8. Read File   │
   │          │ <────────────────────┃
   │  9. Return Image   │          │
   │ <────────────────────┃          │
   │          │          │
```

## 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