API Challenge

Challenge Overview

Build a simple warehouse management API using Golang, PostgreSQL, Gorm, and Echo. The API will include user authentication and basic CRUD operations for managing warehouse items and inventory. The project should follow the MVC (Model-View-Controller) architecture.

Requirements

1. **Features:**
    - User registration and login.
    - CRUD operations for items and warehouses.
    - CRUD operations for inventory management.
    - Bulk Insert for inventory management.

2. **Tables:**
    - **Users:** to store user data and authentication.
    - **Items:** to store item details.
    - **Warehouses:** to store warehouse details.
    - **Inventories:** to keep track of **items** in different **warehouses.**

Detailed Specifications

1. **Database Schema**
    - **Users Table**

Column Name	Data Type	Default Value	Description
id	VARCHAR(255)	uuid.new()	unique & not null
created_at	TIMESTAMP	created date in timestamp	not null
updated_at	TIMESTAMP	NULL	
deleted_at	TIMESTAMP	NULL	
username	VARCHAR(255)		unique & not null
email	VARCHAR(255)		unique & not null
password	VARCHAR(255)		hashed password & can't be null

- **Items Table**

Column Name	Data Type	Default Value	Description
id	VARCHAR(255)	uuid.new()	unique & not null
created_at	TIMESTAMP	created date in timestamp	not null
updated_at	TIMESTAMP	NULL	
deleted_at	TIMESTAMP	NULL	
name	VARCHAR(255)		not null
price	NUMERIC	0	float64 in golang
description	TEXT	NULL	

- **Warehouses Table:**

Column Name	Data Type	Default Value	Description
id	VARCHAR(255)	uuid.new()	unique & not null
created_at	TIMESTAMP	created date in timestamp	not null
updated_at	TIMESTAMP	NULL	
deleted_at	TIMESTAMP	NULL	
name	VARCHAR(255)		not null
address	TEXT		not null

- **Inventories Table:**

Column Name	Data Type	Default Value	Description
id	VARCHAR(255)	uuid.new()	unique & not null
created_at	TIMESTAMP	created date in timestamp	not null
updated_at	TIMESTAMP	NULL	
deleted_at	TIMESTAMP	NULL	
item_id	VARCHAR(255)		not null
warehouse_id	VARCHAR(255)		not null
quantity	NUMERIC	0	float64 in golang

2. **Project Structure**

   Since we a are utilizing the MVC (Model-View-Controller) architecture, please create these three modules:

   - **Routes:** Handles API routing
   - **Controllers:** Manages all business logic
   - **Models:** Manages database interactions (create, update, delete, etc.)

   Those are 3 main modules for this challenge, you are free to add more modules to help you finish this task (i.e. utility, middlewares, configs, etc.).

Endpoint

This is the list of endpoints for this challenge. Please use the appropriate http method for each endpoint. For example, use POST to create, GET to fetch data, etc. The details about http method could be read here https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

For all endpoint response code, please use the appropriate response code. For example use 201 on create success, 200 on success, 400 on bad request, 500 on internal server error, etc. The details about response code could be read here https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

User Endpoints

1. **Register**
   - URL: /register
   - Request Body:

```
{
    "username": "string", // Required
    "email": "string", // Required
    "password": "string" // Required
}
```

2. **Login**
   - URL: /login
   - Request Body:

```
{
    "username": "string", // Required
    "password": "string" // Required
}
```

   - Response: 200 ok with JWT token on success.

Item Endpoints

3. **Fetch Items**
   - URL to Fetch All: /items
   - URL to Fetch Single Item: /items/:id
   - Request Body:

```
{
  // filter value can be empty
  "filter": {
        "keyword": "string", // to filter item name
        "price_min": "float64",
        "price_max": "float64"
  }
}
```

   - Note: only fetch data that is not deleted. Since we use soft delete, show data which deleted_at value is NULL.

4. **Create Item**
   - URL: /items
   - Request Body:

```
{
    "name": "string", // Required
    "description": "string",
    "price": "float" // Required
}
```

   - Response: 201-created with the created data.

5. **Create Multiple Items**
   - URL: /items/bulk
   - Request Body:

```
[
  {
        "name": "string", // Required
        "description": "string",
        "price": "float" // Required
  },
  {
        "name": "string", // Required
        "description": "string",
        "price": "float" // Required
  },
  {
        "name": "string", // Required
        "description": "string",
        "price": "float" // Required
  },
]
```

- Response: 201-created with the created data.

6. **Update Item**
   - URL: /items/:id
   - Request Body:

```
{
    "name": "string",
    "description": "string",
    "price": "float"
}
```

   - Response: 200-ok with the updated data.

7. **Delete Item**
   - URL: /items/:id
   - Note: there's no hard delete on this challenge, use soft delete by changing the value of **deleted_at** from NULL to current time.

Warehouse Endpoints

8. **Fetch All Warehouses**
   - URL to Fetch All: /warehouses
   - URL to Fetch Single Item: /warehouses/:id
   - Request Body:

```
{

    // filter value can be empty
    "filter": {
        "keyword": "string", // to filter warehouse name
    }
}
```

   - Note: only fetch data that is not deleted. Since we use soft delete, show data which deleted_at value is NULL.

9. **Create Warehouse**
   - URL: /warehouses
   - Request Body:

```
{
    "name": "string", // Required
    "address": "string",
}
```

   - Response: 201-created with the created data.

10. Update Warehouse

- URL: /warehouses/:id
- Request Body:

```
{
    "name": "string",
    "description": "string",
    "price": "float"
}
```

- Response: 200-ok with the updated data.

11. Delete Warehouse

- URL: /warehouses/:id
- Note: there's no hard delete on this challenge, use soft delete by changing the value of **deleted_at** from NULL to current time.

Inventory Endpoints

12. Get All Inventories

- URL to Fetch All: /inventories
- Request Body:

```
{
// filter value can be empty
    "filter": {
        "item_id": "string",
        "warehouse_id": "string"
    }
}
```

- Note: only fetch data that is not deleted. Since we use soft delete, show data which deleted_at value is NULL.

13. Create Inventory

- URL: /inventories
- Request Body:

```
{
        "warehouse_id": "string",
        "items": [
            {
                "item_id": "string", // required
                "quantity": "float"
            },
            {
                "item_id": "string", // required
                "quantity": "float"
            }
        ]
}
```

- Response: 201-created with the created data.
- Note: if one of the item is already available on the warehouse, send duplicated data error.

14. Update Inventory

- URL: /inventories/:id
- Request Body:

```
{
  "warehouse_id": "string", // required
  "items": [
        {
              "item_id": "string", // required
              "quantity": "float" // required
        },
        {
              "item_id": "string", // required
              "quantity": "float" // required
        },
  ]
}
```

- Response: 200-ok with the updated data.
- Note: if one of the items is not available on the warehouse, send error.
- Note: quantity could be positive or negative.

15. Delete Inventory

- URL: /inventories/:id
- Note: there's no hard delete on this challenge, use soft delete by changing the value of **deleted_at** from NULL to current time.

16. Move Stock Inventory

- Overview: to move stock inventory from one warehouse to another warehouse.
- URL: /inventories/move-stock

- Request Body:

```
{
	"origin_warehouse_id": "string",
	"destination_warehouse_id": "string",
	"items": [
		{
			"item_id": "string",
			"quantity": "float"
		},
		{
			"item_id": "string",
			"quantity": "float"
		},
	]
}
```

- Response: 200-ok without the updated data.
- Note: Use SQL transaction to update the data.

Rule

- ChatGPT, gemini AI, or any other generative AI tools were not allowed.
- All built-in methods are allowed to be used.
- Please use golang version 1.19.