Toko API Documentation

Requirements:

- Go version v1.21++. For more detailed information related to Go installation, you can open this link.
- PostgreSQL version 14.10. For more detailed information related to PostgreSQL installation, you can open this link.
- (If you choose to use Docker), Docker engine version 19.03.0+ because I use docker-compose version 3.8. For more detailed information related to Docker installation, you can open this link.

Assumptions:

1. For products's items column, I assumed it it foreign key to product's id column

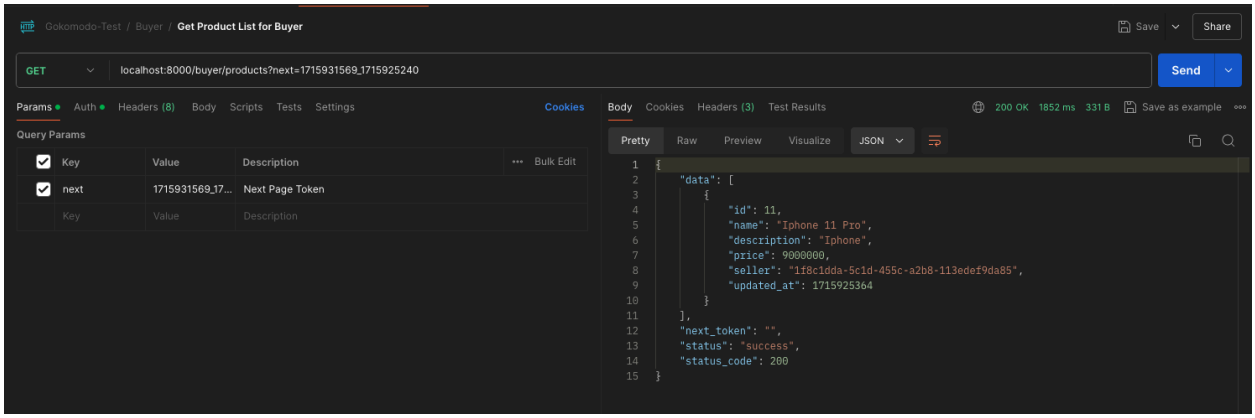
API's Endpoints

1. Health Check: GET /ping
2. Login for Seller: POST /login/seller
    - JSON request body required
3. Get Product List for Seller: GET /seller/products
    - Authentication: Bearer token (JWT)
    - Role Authorization: "Seller" (only role "Seller" can access this endpoint)
    - URL query param **next** for next page token because this endpoint only return 10 data per call (I used pagination feature)
4. Get Order List for Seller: GET /seller/orders
    - Authentication: Bearer token (JWT)
    - Role Authorization: "Seller" (only role "Seller" can access this endpoint)
    - URL query param **next** for next page token because this endpoint only return 10 data per call (I use pagination feature)

5. Accept Order: PUT /seller/orders/{order_id}
    - Authentication: Bearer token (JWT)
    - Role Authorization: “Seller” (only role “Seller” can access this endpoint)
6. Create Product: POST /seller/products
    - Authentication: Bearer token (JWT)
    - Role Authorization: “Seller” (only role “Seller” can access this endpoint)
    - JSON request body required
7. Login for Buyer: POST /login/buyer
    - JSON request body required
8. Get Product List for Buyer: GET /buyer/products
    - Authentication: Bearer token (JWT)
    - Role Authorization: “Buyer” (only role “Buyer” can access this endpoint)
    - URL query param **next** for next page token because this endpoint only return 10 data per call (I used pagination feature)
9. Get Order List for Buyer: GET /buyer/orders
    - Authentication: Bearer token (JWT)
    - Role Authorization: “Buyer” (only role “Buyer” can access this endpoint)
    - URL query param **next** for next page token because this endpoint only return 10 data per call (I used pagination feature)
10. Create Order: POST /buyer/orders
    - Authentication: Bearer token (JWT)
    - Role Authorization: “Buyer” (only role “Buyer” can access this endpoint)
    - JSON request body required

Notes:

For more detailed information related API spec (Each case examples for each endpoints with request & response body examples and etc), you can import this TOKO API Postman collections from **documents/Gokomodo-Test.postman_collection.json** file to your Postman application/workspace.



Unit Testing

Notes:

- I used the stretchr/testify library to help unit test preparation and to compare the expected output with actual output using the assert package.
- For mocking, I used gomock to mock methods in the repository layer only since I only created unit tests in the service & repository layer (the handlers do not have any logic). For call to database mock, I used sqlmock.
- I created all unit tests repository layers. However, for service layers I only created unit tests for some methods especially for methods that don't use data from JWT Custom Claims in the Gin Context (I injected the Claims to Gin Context after the request successfully authenticated by auth middleware). Other than that, I couldn't create the unit tests because the mock Gin Context can't be injected with the Claims and I've tried multiple times with many solutions.
- I tried to test every case (success & error) in every method by comparing the expected output with actual output (returned data and error) except for the service layer, I only checked the status code.
- You can use the command `go test -coverprofile=coverage.out ./... ; go tool cover -func=coverage.out` to run the unit test and display the unit test coverage result.

Additional info for Docker

For this part, I assumed you guys are already setting up Docker and Go.

- Run API server in local

  Before you run the API server, you need to create a **.env** file by copying **.env.sample** files and fill the env values based on your needs. Then you need to set up the PostgreSQL database first. You can use your local PostgreSQL database or you can use Docker to set up PostgreSQL using **postgredb.dockerfile** file (if you only want to use Docker for setup database only). You can use one of these commands to run the API server in local:

  - Go run main.go
  - Go build && ./gokomodo-assignment

- Run connected PostgreSQL & API server

  This point can be done by using the **docker-compose.yml** file located in the root folder of this project. For PostgreSQL database configuration in docker-compose, you can set the environment like (db name, username, password) in file **postgresdb.dockerfile** and set the port in docker-compose file. For API config, you can just edit the env value in the **.env** file because when the API image is being built by docker, it will copy the **.env** file value to the container. If you are already done with the configuration, you can run this command to build the docker images and run the containers **docker-compose up -d --build**.