S3CORP

golang-fresher

Loc Dang

Mai nguyen

Table of Contents

[Introduction 2](#_Toc110590887)

[Project structure 3](#_Toc110590888)

[Database schema 5](#_Toc110590889)

[Technology 7](#_Toc110590890)

[How to run? 8](#_Toc110590891)

[API documents 9](#_Toc110590892)

# Introduction

This is a sample project to learn how to develop a web application with the Go language.

# Project structure

1. **/cmd**

This folder contains the main application entry point files for the project.

1. **/internal**

This package holds the private library code, it is specific to the function of the service and not shared with other services. It includes handler, service, repository and model packages.

* 1. /handler

This folder contains all the handler functions that will process requests, call functions from service and return response. Includes RESTful and GraphQL.

* 1. /service

This folder contains all the service functions that will take parameters, perform operations, and return results.

* 1. /repository

This folder contains all the functions for performing database queries.

* 1. /model

This package holds all the ORM models.

1. **/pkg**

This folder contains code which is OK for other services to consume, this includes utility functions.

1. **/data**

This folder contains data files such as migration files.

1. **Makefile**

This is special file contains shell commands, such as “setup” and “run” commands.

1. **Dockerfile**

This is a text document that contains all the commands a user could call on the command line to assemble an image.

1. **docker-compose.yml**

The Compose file provides a way to document and configure all of the application's service dependencies (databases, queues, caches, web service APIs, etc).

1. **go.mod & go.sum**

The go.mod file is the root of dependency management in Go language. All the modules which are needed or to be used in the project are maintained in go.mod file.

# Database schema

1. Users schema:

|  |  |  |  |
| --- | --- | --- | --- |
| **Fields** | **Datatype** | **Description** | **Refer to** |
| id | INT NOT NULL | PRIMARY KEY, AUTO\_INCREMENT |  |
| name | TEXT NOT NULL |  |  |
| email | TEXT NOT NULL | UNIQUE |  |
| phone | TEXT NOT NULL | UNIQUE |  |
| password | TEXT NOT NULL |  |  |
| role | TEXT NOT NULL | ADMIN or GUEST |  |
| is\_active | BOOL NOT NULL |  |  |
| created\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |
| updated\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |

1. Products schema:

|  |  |  |  |
| --- | --- | --- | --- |
| **Fields** | **Datatype** | **Description** | **Refer to** |
| id | SERIAL NOT NULL | PRIMARY KEY, AUTO\_INCREMENT |  |
| title | TEXT NOT NULL |  |  |
| code | TEXT NOT NULL | UNIQUE |  |
| description | TEXT NOT NULL | DEFAULT ''” |  |
| price | FLOAT NOT NULL |  |  |
| amount | INT NOT NULL | DEFAULT 0 |  |
| is\_active | BOOL NOT NULL | DEFAULT TRUE |  |
| user\_id | INT NOT NULL | FOREIGN KEY | users(id) |
| created\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |
| updated\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |

1. Order schema:

|  |  |  |  |
| --- | --- | --- | --- |
| **Fields** | **Datatype** | **Description** | **Refer to** |
| id | SERIAL NOT NULL | PRIMARY KEY, AUTO\_INCREMENT |  |
| order\_number | TEXT NOT NULL | UNIQUE |  |
| order\_date | TIMESTAMP WITH TIME ZONE NOT NULL | DEFAULT NOW() |  |
| user\_id | INT NOT NULL | FOREIGN KEY | users(id) |
| note | TEXT NOT NULL | DEFAULT "" |  |
| status | TEXT NOT NULL | NEW, PENDING, SUCCESS, FAILED |  |
| created\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |
| updated\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |

1. Order items schema:

|  |  |  |  |
| --- | --- | --- | --- |
| **Fields** | **Datatype** | **Description** | **Refer to** |
| id | SERIAL NOT NULL | PRIMARY KEY, AUTO\_INCREMENT |  |
| order\_id | INT NOT NULL | FOREIGN KEY | orders(id) |
| product\_id | INT NOT NULL | FOREIGN KEY | products(id) |
| product\_price | FLOAT NOT NULL |  |  |
| product\_name | TEXT NOT NULL |  |  |
| quantity | INT NOT NULL | DEFAULT 0 |  |
| discount | FLOAT NOT NULL | DEFAULT 0 |  |
| note | TEXT NOT NULL | DEFAULT "" |  |
| created\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |
| updated\_at | TIMESTAMP WITH TIME ZONE | DEFAULT NOW() |  |

# Technology

* Golang 1.18
* Go-chi v5.0.7
* SQLBoiler v4.11.0
* Docker
* Makefile
* Gqlgen
* Postgres 11
* Testify v1.8.0

# How to run?

Open your terminal and type “make setup”, This will start PostgreSQL, migrate the database, build the app to docker image, and start the app.

Besides, we provide the following commands:

|  |  |
| --- | --- |
| **Command** | **Detail** |
| make setup | Setup the application (includes database, migration, build docker the image) and start it. |
| make run | Run the application. |
| make db | Run PostgreSQL on a docker container. |
| make db-migration | Run migration. |
| make docker-build-go-imange | Build the application to docker image. |
| make docker-run-go-image | Run the application on a docker container. |
| make down | Stop and remove all docker containers of project. |
| make test | Run test. |
| make gql-gen | Generate GraphQL models. |
| make vendor | Create folder named vendor in the root directory that contains copies of all packages needed to support builds and tests of packages in the main module. |

# API documents

1. Create User API

**Endpoint**: POST /api/v1/users

**Request body**

Content-type: application/json

* name: the name of user.
* email: the email of user.
* password: the account password.
* phone: the user phone number.
* role: the role name, it can be “ADMIN” or “GUEST”.
* is\_active: this account is activated or not.

**Sample request**

1. POST /api/v1/user
2. {
3. "name": "guest",
4. "email": "guest@appcom",
5. "password": "123456789",
6. "phone": "0987654321",
7. "role": "GUEST",
8. "is\_active": false
9. }
   1. Success:

**Attributes**

* status code: 201 (Created)
* id: the id of user.
* [user propertites]: other propertites of user.

**Sample response**

1. {
2. "id": 1,
3. [user properties]
4. }
5. 1. Error:

* Invalid email

1. {"status": 400,"code": "invalid\_email","desc": "email is invalid"}

* Invalid role

1. {"status": 400,"code": "invalid\_role","desc": "role is invalid"}

* Email duplicated

1. {"status": 400,"code": "email\_existed","desc": "email is adready exists"}
2. Get users API

**Endpoint**: GET /api/v1/users

**Request body**

Content-type: application/json

* id: the id of user.
* email: the email of user.
* name: the name of user.
* is\_active: the is\_active of user.
* role: the role of user.
* sort: the results to be sorted by, such as “name”, “email”, “created\_at”.
* pagination: the results to be pagination by “page” and “limit”.

**Sample request**

1. {
2. "is\_active": true,
3. "role": "ADMIN",
4. "sort": {
5. "name": "desc",
6. "created\_at": "asc"
7. }
8. }

**Note**

By default, pagination with page is 1 and limit is 20. It can also be used with filter and sorter on the same request.

* 1. Success:

**Attributes**

* status code: 200 (OK)
* users: list of users
* pagination: pagination parameters

**Sample response**

1. {
2. "users": [
3. {
4. "id": 10,
5. "name": "test1",
6. "email": "test1@example.com",
7. "password": "test",
8. "phone": "test",
9. "role": "ADMIN",
10. "is\_active": true,
11. "created\_at": "2022-08-05T03:34:50.545713Z",
12. "updated\_at": "2022-08-05T03:34:50.545713Z"
13. }
14. ],
15. "pagination": {
16. "current\_page": 1,
17. "limit": 20,
18. "total\_count": 1
19. }
20. }
    1. Error:

* Invalid id

1. {"status": 400,"code": "invalid\_id","desc": "id is invalid"}

* Invalid email

1. {"status":400,"code":"invalid\_email","desc":"email is invalid"}

* Invalid role

1. {"status":400,"code":"invalid\_role","desc":"role is invalid"}

* Invalid sort order type

1. {"status": 400,"code": "invalid\_sort\_type","desc": "sort type is invalid"}

* Invalid page

1. {"status": 400,"code": "invalid\_page","desc": "page is invalid"}

* Invalid limit

1. {"status": 400,"code": "invalid\_limit","desc": "limit is invalid"}
2. Get user

**Endpoint**: GET /api/v1/users/{id}

**Attributes**

* id: the id of user.

**Sample request**

1. GET /api/v1/users/10
   1. Success

**Attributes**

* status code: 200 (OK)
* [user properties]: properties of user

**Sample response**:

1. {
2. "id": 10,
3. "name": "test1",
4. "email": "test1@example.com",
5. "password": "test",
6. "phone": "test",
7. "role": "ADMIN",
8. "is\_active": true,
9. "created\_at": "2022-08-05T03:34:50.545713Z",
10. "updated\_at": "2022-08-05T03:34:50.545713Z"
11. }
    1. Error
12. {status": 400,"code":"user\_not\_found","desc":"user is not found"}
13. Update user

**Endpoint**: PUT /api/v1/users/{id}

**Attributes**

* id: the id of user

**Request body**

Content-type: application/json

* name: the name of user.
* email: the email of user.
* password: the account password.
* phone: the user phone number.
* role: the role name, it can be “ADMIN” or “GUEST”.
* is\_active: this account is activated or not.

**Sample request**

1. {
2. "name": "guest2",
3. "email": "test2@example.com",
4. "password": "test",
5. "phone": "test",
6. "role": "ADMIN",
7. "is\_active": false
8. }
   1. Success

**Attributes**

* status code: 200 (OK)

**Sample response**:

1. {"success":true,"msg":"Update user successfully"}
   1. Error
2. {"status":400,"code":"invalid\_user\_id","desc":"user ID is not a valid"}
3. Delete user

**Endpoint**: DELETE /api/v1/users/{id}

**Attributes**

* id: the id of user

**Sample request**

1. DELETE /api/v1/users/1
   1. Success

**Attributes**

* Status code: 200 (OK)

**Sample response**

1. {"success":true,"msg":"Delete user successfully"}
   1. Error
2. {"status":400,"code":"invalid\_user\_id","desc":" user ID is not a valid "}