Friend Management API Specification

Contents

1. TECHNICAL STACK 3

1.1. Summary 3

1.2. Technical details 3

2. SOURCE CODE 4

2.1. Git 4

2.2. Project structure 4

2.3. Deployment 6

3. API DETAILS 7

3.1. API to create user email 7

3.1.1. Description 7

3.1.2. Request 7

3.1.3. Response 7

3.2. API to retrieve all user email addresses in database 8

3.2.1. Description 8

3.2.2. Request 8

3.2.3. Response 8

3.3. API to create a friend connection between two email addresses 9

3.3.1. Description 9

3.3.2. Request 9

3.3.3. Response 10

3.4. API to retrieve the friends list for an email address 11

3.4.1. Description 11

3.4.2. Request 11

3.4.3. Response 11

3.5. API to retrieve the common friends list between two email addresses 12

3.5.1. Description 12

3.5.2. Request 13

3.5.3. Response 13

3.6. API to subscribe to updates from an email address ... 14
3.6.1. Description ... 14
3.6.2. Request ... 14
3.6.3. Response ... 15
3.7. API to block updates from an email address ... 16
3.7.1. Description ... 16
3.7.2. Request ... 16
3.7.3. Response ... 16
3.8. API to retrieve all email addresses that can receive updates from an email address ... 17
3.8.1. Description ... 17
3.8.2. Request ... 18
3.8.3. Response ... 18

1. TECHNICAL STACK

This Friend Management API is a RESTful API built on Golang language.

1.1. Summary

- Programming Language: Golang
- Packages:
  + database/sql, github.com/go-sql-driver/mysql
  + net/http, github.com/gorilla/mux
  + github.com/swaggo/http-swagger
  + github.com/stretchr/testify/assert, github.com/stretchr/testify/mock, testing
- Database: MySQL
- Deployment: Linux, Docker
- Tools: Goland, Git

1.2. Technical details

- **database/sql:** provides light-weight interface to connect with the databases.
- **github.com/go-sql-driver/mysql:** it use for connecting with mysql database.
- **net/http:** provides HTTP client and server implementations
- **github.com/gorilla/mux:** implements a request router and dispatcher.
- **github.com/swaggo/http-swagger:** is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs.
- **github.com/stretchr/testify/assert:** set of packages that provide many tools for testifying that your code will behave as you intend.
- **github.com/stretchr/testify/mock:** provides a system by which it is possible to mock your objects and verify calls are happening as expected.
- **testing:** provides support for automated testing of Go packages. It is intended to be used in concert with the "go test" command, which automates execution of any function of the form.

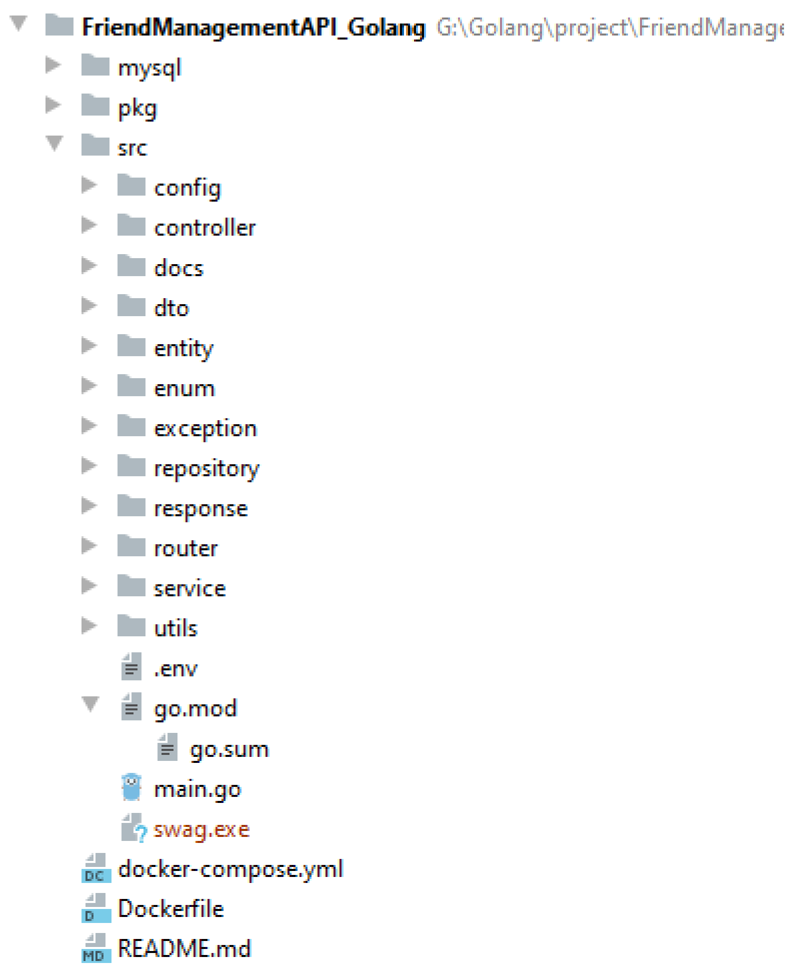
2. SOURCE CODE

2.1. Git

Link source code: https://github.com/toannx95/FriendManagementAPI_Golang

2.2. Project structure

This is structure of this project:



```
▼ FriendManagementAPI_Golang G:\Golang\project\FriendManage
  ▶ mysql
  ▶ pkg
  ▼ src
    ▶ config
    ▶ controller
    ▶ docs
    ▶ dto
    ▶ entity
    ▶ enum
    ▶ exception
    ▶ repository
    ▶ response
    ▶ router
    ▶ service
    ▶ utils
      .env
    ▼ go.mod
        go.sum
      main.go
      swag.exe
    docker-compose.yml
    Dockerfile
    README.md
```

There are 2 main folders and many packages for each folder:

- **mysql:** contains query to init database.
- **src:**
  - config: contains configuration classes
  - controller: contains controller classes
  - dto: contains the data transfer objects classes
  - entity: contains the entity classes to map with database
  - enum: contains enum classes
  - exception: contains exception classes
  - repository: contains function manipulating with database
  - response: contains response classes
  - router: contains main router classes
  - service: contains service classes
  - utils: contains util classes
- **other files:**
  - .env: contain the configuration for database.
  - Dockerfile file: contains all the commands a user could call on the command line to assemble an image.
  - Docker-compose.yml file: defining services, networks, volumes, sql database, and other configuration.

2.3. Deployment

This project was deployed by Docker to Linux server at: http://localhost:8081/

OpenAPI Specification by the Swagger framework: http://localhost:8081/swagger/index.html#/

❖ **Prerequisites:**

- Docker
- Docker Compose
- MySQL 5.7 for docker

❖ **Here are the steps to deploy:**

# Quick Run Project

```
git clone https://github.com/toannx95/FriendManagementAPI_Golang

docker-compose up -d
```

# Run test

```
cd src

go test ./...
```

# Run test with coverage

```
cd src

go test -cover ./...
```

3. API DETAILS

3.1. API to create user email

3.1.1. Description

- ❖ The API is used to create new user email. Need create user email before testing others API.
- ❖ Conditions for creating user email are:
  - *Valid email format*
  - *Email does not exist in database yet*
- ❖ The URL for accessing the API is:

  http://localhost:8080/api/emails/create-user

- ❖ HTTP method supported is: **POST**

3.1.2. Request

Note that this request only accepts HTTP POST.

3.1.2.1. Request Example

- ❖ URL: http://localhost:8080/api/emails/create-user
- ❖ JSON request:

```
{
    "email": "andy@example.com"
}
```

3.1.3. Response

Note that this response returns only JSON format.

3.1.3.1. Success Response Example

- ❖ Code: 201
- ❖ JSON response:

```
{
    "success": true
}
```

3.1.3.2. Error Response Example

- Code: 500
- JSON response:

```
{
    "error": "Email aldready existes!"
}
```

- Code: 400
- JSON response:

```
{
    "error": "Wrong email format!"
}
```

3.2. API to retrieve all user email addresses in database

3.2.1. Description

- The API is used to retrieve all user email addresses in the database.
- The URL for accessing the API is:

  http://localhost:8080/api/users

- HTTP method supported is: **GET**

3.2.2. Request

Note that this request only accepts HTTP GET.

3.2.2.1. Request Example

- URL: http://localhost:8080/api/emails

3.2.3. Response

Note that this response returns only JSON format.

3.3.3.1. Success Response Example

- Code: 200
- JSON response:

```
{
    "success": true,
    "friends": [
        "andy@example.com",
        "john@example.com"
    ],
    "count": 2
}
```

3.3.3.2. Error Response Example

3.3. API to create a friend connection between two email addresses

3.3.1. Description

- The API is used to create friend connection between two email addresses.
- Conditions for creating friend connection are:
    - *Valid email format*
    - *Both emails already existed in the database*
    - *Both emails did not block each other*
- The URL for accessing the API is:

  http://localhost:8080/api/friends/create-friend

- HTTP method supported is: **POST**

3.3.2. Request

Note that this request only accepts HTTP POST.

3.3.2.1. Request Example

- ❖ URL: http://localhost:8080/api/friends/create-friend
- ❖ JSON request:

```
{
    "friends": [
        "andy@example.com",
        "john@example.com"
    ]
}
```

3.3.3. Response

Note that this response returns only JSON format.

3.3.3.1. Success Response Example

- ❖ Code: 200
- ❖ JSON response:

```
{
    "success": true
}
```

3.3.3.2. Error Response Example

- ❖ Code: 500
- ❖ JSON response:

```
{
    "error": "Can not make friend!"
}
```

- ❖ Code: 400
- ❖ JSON response:

```
{
    "error": "Wrong email format!"
}
```

- Code: 404
- JSON response:

```
{
    "error": "Email not found with email: 'andyee@example.com'"
}
```

3.4. API to retrieve the friends list for an email address

3.4.1. Description

- The API is used to retrieve the friends list for an email address.
- The URL for accessing the API is:

  http://localhost:8080/api/friends/get-friends-list

- HTTP method supported is: **POST**

3.4.2. Request

Note that this request only accepts HTTP POST.

3.4.2.1. Request Example

- URL: http://localhost:8080/api/friends/get-friends-list
- JSON request:

```
{

    "email": "andy@example.com"

}
```

3.4.3. Response

Note that this response returns only JSON format.

3.4.3.1. Success Response Example

- Code: 200
- JSON response:

```
{
    "success": true,
    "friends": [
        "john@example.com"
    ],
    "count": 1
}
```

3.4.3.2. Error Response Example

- Code: 400
- JSON response:

```
{
    "error": "Wrong email format!"
}
```

- Code: 404
- JSON response:

```
{
    "error": "Email not found with email: 'andy@1example.com'"
}
```

3.5. API to retrieve the common friends list between two email addresses

3.5.1. Description

- The API is used to retrieve the common friends list
- The URL for accessing the API is:

  http://localhost:8080/api/friends/get-common-friends-list

- HTTP method supported is: **POST**

3.5.2. Request

Note that this request only accepts HTTP POST.

3.5.2.1. Request Example

- URL: http://localhost:8080/api/friends/get-common-friends-list
- JSON request:

```
{
    "friends": [
        "andy@example.com",
        "john@example.com"
    ]
}
```

3.5.3. Response

Note that this response returns only JSON format.

3.5.3.1. Success Response Example

- Code: 200
- JSON response:

```
{
    "success": true,
    "friends": [
        "common@example.com"
    ],
    "count": 1
}
```

3.5.3.2. Error Response Example

- Code: 400
- JSON response:

```
{
    "error": "Wrong email format!"
}
```

- Code: 404
- JSON response:

```
{
    "error": "Email not found with email: 'andy313@example.com'"
}
```

3.6. API to subscribe to updates from an email address

3.6.1. Description

- The API is used to subscribe to updates from an email address.
- Conditions for creating subscribe are:
  - *Valid email format*
  - *Both emails already existed in the database*
  - *Requestor has not subscribed target yet*
  - *Both emails have not blocked each other*
- The URL for accessing the API is:

  http://localhost:8080/api/subscribe

- HTTP method supported is: **POST**

3.6.2. Request

Note that this request only accepts HTTP POST.

3.6.2.1. Request Example

- URL: http://localhost:8080/api/subscribe
- JSON request:

```
{
    "requestor": "lisa@example.com",

    "target": "john@example.com"

}
```

3.6.3. Response

Note that this response returns only JSON format.

3.6.3.1. Success Response Example

- Code: 200
- JSON response:

```
{
    "success": true
}
```

3.6.3.2. Error Response Example

- Code: 500
- JSON response:

```
{
    "error": "Can not subscribe!"
}
```

- Code: 400
- JSON response:

```
{
    "error": "Wrong email format!"
}
```

- Code: 404
- JSON response:

```
{
    "error": "Email not found with email: 'lisa@example.com'"
}
```

3.7. API to block updates from an email address

3.7.1. Description

- The API is used to block updates from an email address.
- Conditions for creating block are:
    - *Valid email format*
    - *Both emails already existed in the database*
    - *Both emails did not block each other*
- The URL for accessing the API is:

  http://localhost:8080/api/block

- HTTP method supported is: **POST**

3.7.2. Request

Note that this request only accepts HTTP POST.

3.7.2.1. Request Example

- URL: http://localhost:8080/api/block
- JSON request:

```
{
    "requestor": "andy@example.com",
    "target": "john@example.com"
}
```

3.7.3. Response

Note that this response returns only JSON format.

3.7.3.1. Success Response Example

- Code: 200
- JSON response:

```
{
    "success": true
}
```

3.7.3.2. Error Response Example

❖ Code: 500
❖ JSON response:

```
{
    "error": "Already blocked friends!"
}
```

❖ Code: 400
❖ JSON response:

```
{
    "error": "Wrong email format!"
}
```

❖ Code: 404
❖ JSON response:

```
{
    "error": "Email not found with email: 'andy121@example.com'"
}
```

3.8. API to retrieve all email addresses that can receive updates from an email address

3.8.1. Description

❖ The API is used to retrieve all email addresses that can recevie updates from an email address.
❖ Conditions for receiving updates from sender are:
  - *Has not blocked updates from sender and*
  - *At least one of the following:*
    - *has a friend connection with sender*
    - *has subscribed to updates from sender*
    - *has been @mentioned in the update*

❖ The URL for accessing the API is:

http://localhost:8080/api/friends/get-receivers-list

❖ HTTP method supported is: **POST**

3.8.2. Request

Note that this request only accepts HTTP POST.

3.8.2.1. Request Example

- URL: http://localhost:8080/api/friends/get-receivers-list
- JSON request:

```
{
    "sender": "john@example.com",
    "text": "Hello World! kate@example.com"
}
```

3.8.3. Response

Note that this response returns only JSON format.

3.8.3.1. Success Response Example

- Code: 200
- JSON response:

```
{
    "success": true,
    "friends": [
        "common@example.com",
        "lisa@example.com",
        "kate@example.com"
    ],
    "count": 3
}
```

3.8.3.2. Error Response Example

- Code: 400
- JSON response:

```
{
    "error": "Wrong email format!"
}
```

- Code: 404
- JSON response:

```
{
    "error": "Email not found with email: 'john12@example.com'"
}
```