Deployment & Installation Guide

Prerequisites:

- Docker installed
- Postman installed
- Must call authorization API to obtain JWT token, then include the token as JWT Bearer in request header for subsequent API calls

1. **Docker Container Deployment**
    a. Go to the root directory(directory that stores docker-compose.yml) of the project using Terminal

    b. Execute bash command: docker-compose up --build

    c. After containers deployment, the following ports will be exposed:

        - localhost:8081 (restaurant-api)
        - localhost:5432 (postgresql)

    # **Remark**: First time deployment/installation might acquires longer time

2. **Postman API**
    a. Look up for the *Restaurant-API.postman_collection.json* in the project root directory(directory that stores docker-compose.yml)

    b. Import into Postman collection

    c. Trigger each API call via the template prepared [Must ensure containers are running]

3. **Swagger**

    a. Visit localhost:8081

    b. The API service comes with Swagger OpenAPI Documentation

    Trigger each API call via the swagger UI

1. Objective

To create a set of RESTful API services using **Java**

2. Requirements

2.1. To create an API that display a list of restaurants with the following information

- Name
- Category
- Picture
- Ratings (stars)
- Reviews

2.2. To create an API that allows searching for a particular restaurant name, and/or category

2.3. To create an API that allows creating a new restaurant to the existing list by inputting its name, category(allows selection) and picture

2.4. To create an API that allows adding a new rating and review to an existing restaurant
[Remarks: 1 restaurant can have many reviews]

2.5. To create an API that allows calculating and displaying the restaurant's rating based on the average rating of its reviews

3. API Formatting

3.1. Success Response

```
{
 "data": <T>,
 "error": null
}
```

3.2. Error Response

```
{
 "data": null,
 "error": {
   "code": "404",
   "message": "Error message…"
 }
}
```

4. APIs Details

4.1. POST localhost:8081/login

Input Parameters

```
{
 "username": "user",
 "password": "user"
}
```

Expected Output

```
{
   "data": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiZXhwIjoxNzQ4MTU0MDkzfQ.j02Jta8C7c9rfeD_wTqqEzDPBZdDEZUk4FynA1DEkFo",
   "error": null
}
```

4.2. GET localhost:8081/restaurant/all

Input Parameters

Auth Type - Bearer Token

Expected Output

```
{
   "data": [
      {
         "restaurantId": 1,
         "restaurantName": "Nasi Lemak Restaurant",
         "restaurantCategory": 1,
         "restaurantPictureUrl": "/app/sources/static/uploads/Screenshot 2024-08-07 122810.png"
      },
      {
         "restaurantId": 2,
         "restaurantName": "No Idea Restaurant",
         "restaurantCategory": 2,
         "restaurantPictureUrl": "/app/sources/static/uploads/Screenshot 2024-08-07 122810.png"
      }
   ],
   "error": null
}
```

4.3. GET localhost:8081/restaurant/get

Input Parameters	localhost:8081/restaurant/get?name=Restaurant&category=1 localhost:8081/restaurant/get?category=1 localhost:8081/restaurant/get?name=Restaurant

Expected Output	{
	"data": [
	{
	"restaurantId": 1,
	"restaurantName": "Nasi Lemak Restaurant",
	"restaurantCategory": 1,
	"restaurantPictureUrl": "/app/sources/static/uploads/Screenshot 2024-08-07 122810.png"
	},
	{
	"restaurantId": 2,
	"restaurantName": "No Idea Restaurant",
	"restaurantCategory": 2,
	"restaurantPictureUrl": "/app/sources/static/uploads/Screenshot 2024-08-07 122810.png"
	}
	],
	"error": null
	}

4.4. POST localhost:8081/restaurant/add

Input Parameters	restaurant: {"restaurantName":"No Idea Restaurant","restaurantCategory": 2} picture: images file

Expected Output	{
	"data": {
	"restaurantId": 2,
	"restaurantName": "No Idea Restaurant",
	"restaurantCategory": 2,
	"restaurantPictureUrl": "/app/sources/static/uploads/Screenshot 2024-08-07 122810.png"
	},
	"error": null
	}

4.5. POST localhost:8081/restaurant/rate

Input Parameters	`{ "rating": 1, "review": "Very Good", "restaurantId": 1 }`
Expected Output	`{ "data": true, "error": null }`

4.6. GET localhost:8081/restaurant/getAvgRate

Input Parameters	localhost:8081/restaurant/getAvgRate?restaurantId=1
Expected Output	`{ "data": 4.25, "error": null }`

5. Security Implementation

5.1. Authentication with JWT

- Users require to authenticate via /login API to obtain JWT token in order to call other APIs that manipulate data

5.2. Password Encryption

- Password will never be stored in plain text
- Use hashing algorithm such as BCrypt to encrypt password before storing into the database

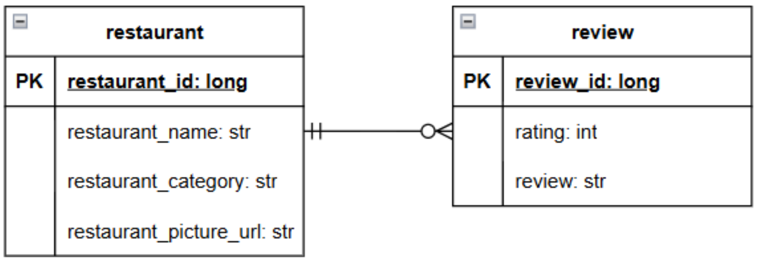
6. Database Configuration

6.1. Database Information

- This credentials is only applied for DEV and DEBUG
- H2 In-Memory database is used for unit test

Type	PostgreSQL
Database Name	restaurantdb
Username	admin
Password	admin
Port	5432

6.2. Database Schema



Remark: Category can be a single table in the database too[appendix 1], but in this test, enum be used.

7. Exceptions

Exception(s)	Description
InvalidCategoryException	Given input category does not exist
InvalidCredentialsException	Given input credentials does not exist
NoRatingException	Given input restaurant does not have rating
RestaurantNotFoundException	Given input restaurant does not exist

8. Unit Testing

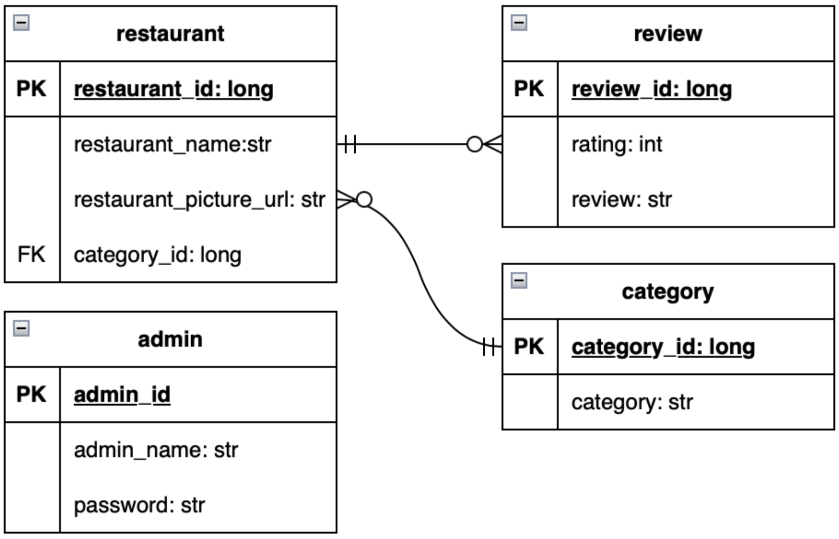
Unit Testing in this project is implemented in different environment, using:

- H2 Memory database during test execution
- It resets itself every time a test run starts, meaning no previous test data is remembered

Test Methods	Description
testAuthorize()	Tests user login and token generation with valid credentials
testAddRestaurant	Tests adding a new restaurant with valid data and image upload
testGetAllRestaurants()	Tests retrieval of all restaurants from the database
testSearchRestaurants()	Tests restaurant search by name or category
testAddReview()	Tests adding a review to an existing restaurant
testCalculateAverageRate()	Tests calculation of a restaurant’s average review rating

Appendix

Appendix 1: Alternative implementation for category



restaurant
PK
restaurant_id: long
restaurant_name:str
restaurant_picture_url: str
FK
category_id: long
review
PK
review_id: long
rating: int
review: str
category
PK
category_id: long
category: str
admin
PK
admin_id
admin_name: str
password: str