CS5500 - Food Delivery Service Design and Implementation

Team members: Yu Fan, Hancong Wang, Linxin Wen

1. Overview

This doc describes the design and implementation of the course project of CS5500 - a food delivery service.

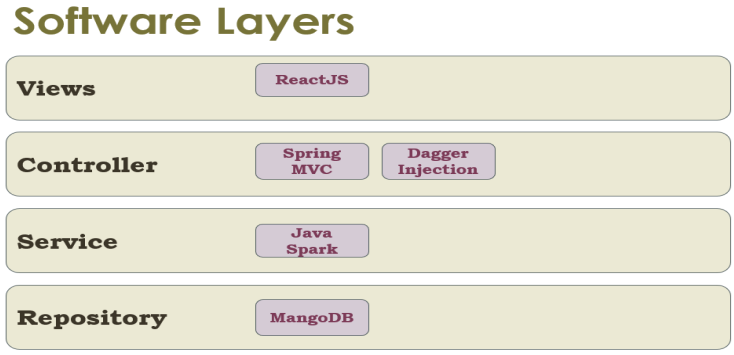
2. Functional Requirements

The food delivery service is designed and implemented to simulate the following real-world user cases:

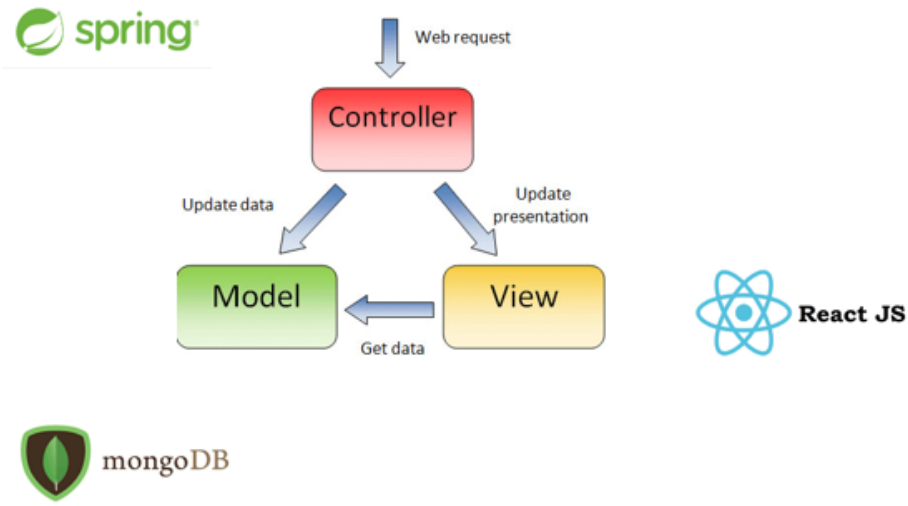
- It supports basic operations for customers, restaurants and couriers;
- From a customer's perspective:
  - The front-end website should display available restaurants info by default;
  - Customer has the ability to sign up, login and log out;
  - Customer has the ability to select meals;
  - Customer has the ability to check out and place an order;
  - Customer has the ability to track the status of the order;
- From a courier's perspective:
  - Courier has the ability to sign up, login and log out;
  - Courier has the ability to pick up an order based on the orderId;
  - Courier has the ability to deliver an order based on the orderId;
- From a restaurant's perspective:
  - The restaurant has the ability to sign up, login and log out;
  - The restaurant needs to ability to change the status of an order;

3. Architecture

From the aspect of software, the end-to-end food delivery service is constituted of four layers as illustrated below in the graph:



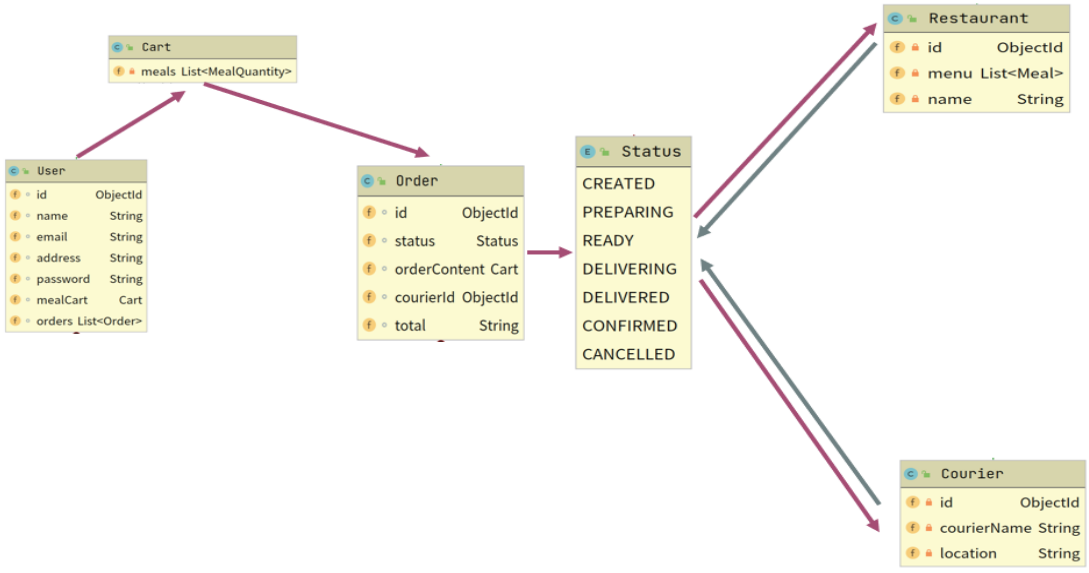
From the aspect of the architecture, there are mainly three components - the front-end website, the backend microservice and the data storage layer.



The details of each layer are covered in the following sections.

4. Database Schema

To support the use cases described in section 2, we have designed the following database tables: a User table to capture the customer information, a Restaurant table to record the restaurant information, an Order table to capture the status of an order and a Courier table to describe courier's information. The entity-relationship diagram is as the following graph:



5. Database

It uses an in-memory hashmap as a datastore when developing the service locally, and it can be easily switched to support other persistent data stores by changing the repository module. We have considered choices like AWS data storage services and MySQL, and we chose MongoDB as the backend data store at the end as it's easy to configure and implement.

We created a MongoDB cluster using mLab which is the default cluster management tool provided by MongoDB. We also utilized MongoDB Compass for manipulating the database data when testing the changes.

6. Frontend

The frontend is implemented in ReactJS and it mainly contains the following pages with the listed implemented functionalities:

6.1. Boarding Page

The boarding page supports the following functionalities:

- **User Login**: On the login page, users can log into their existing account.
- **User Registe**r: For new users, they are directed to the registration page for registering a new account for this web application.
- **User Filtering:** Register and route three types of users: Users who claim a customer, or a restaurant, or a courier will register through different channels and route to different pages.

6.2. Menu Page

The menu page supports the following functionalities:

- **Select Restaurant**: The info and links of the restaurants are listed in an upper sidebar. Right now, there are only three restaurants listed on our web. In the future, it can be expanded with more restaurants.
- **Check Menu**: Once the user chooses a restaurant, the menu of the selected restaurant will be listed on a scrolling page.
- **Add Meals to Shopping Cart:** The user can add her/his favorite meals into the cart through an "add" button. This function supports customers adding meals from different restaurants.
- **Remove Meals from the Shopping Cart**: Since the shopping cart is floating in the right part of the page, it is easy to add and remove the meals from the cart, as well as check the order's cost clearly and conveniently.
- **Place An Order**: Once the meals are added to the cart, the general cost, along with the delivery fee, tips, packing fees will be adjusted according to the price of the meals

in your cart. Once you complete with shopping, you can make an order with one click.

6.3. Order Status Tracking Page

The order info page supports the following functionalities:

- **The Restaurant Prepares the Order:** Once the order being made by the customer, the order number generated. The restaurant users can get the order by searching the order number in their system, and then change the order status to “preparing”. Once the food is ready, the restaurant users can change the order status to “ready”.
- **The Courier Picks Up the Order:** Once the order status has changed to “ready”, the courier user can search the order through its order ID in the courier system, and then pick up the order by changing the order status into “delivering”. The order status can be changed to "delivered" once the courier delivers the order.
- **Customer can Track the Order Status**: Customers can get as nearly real-time updates of orders on the order page.

7. Backend

The backend is basically a microservice that supports Restful APIs utilizing Spark and following the MVC model. All the APIs are basically implemented in a similar way - it performs certain CRUD operations on the corresponding table in the database. See the details in the controller implementation of each model. To support the user cases described in section 2, we have developed the following set of APIs.

7.1. User APIs

The user APIs support functionalities to manipulate the user data, e.g. get a user, get the collection of users, update a user and delete a user. It has the following implemented endpoints:

- GetUsers: Get the collection of users (GET /user)
- AddUser: Add a user to the collection (POST /user)
- UpdateUser: Update the userinfo including name, email, address and password (PUT /user)
- DeleterUser: Delete a user by providing id (DELETE /user)
- UpdateCart: UpdateCart can be used for adding or removing a meal from a cart. The operation only accepts either ADD or REMOVE (PUT /user/cart)
- GetUserById: Get a user (GET /user/{:id})

7.2. Restaurant APIs

The restaurant APIs support functionalities to manipulate the restaurant data, e.g. add a restaurant, get a restaurant, delete a restaurant, update a restaurant, make food. It has the following implemented endpoints:

- DeleteRestaurant: Delete a restaurant with a given the restaurantId(DELETE /restaurant)
- GetAllRestaurants: Get all restaurants in collections (GET /restaurant)
- AddRestaurant: Create a new restaurant (POST /restaurant)
- UpdateRestaurant: Update restaurant name, restaurant menu or both (PUT /restaurant)
- GetRestaurantById: Get a restaurant from the collection by Id (GET /restaurant/{:id})

7.3. Order APIs

The order APIs support functionalities to manipulate the order data, e.g. place an order, get an order, delete an order, update an order etc. It has the following implemented endpoints:

- DeleteOrder: Delete an order (DELETE /order)
- GetAllOrders: Get all the orders in the collection (GET /order)
- PlaceOrder: Place order based on the user cart (POST /order)
- UpdateOrderStatus: Update the status of an order (PUT /order/status)
- GetOrderById: Get order by providing order id (GET order/{:id})

7.4. Courier APIs

The courier APIs support functionalities to manipulate the courier data, e.g. add a courier, get a courier, get the collection of couriers, delete a courier etc. It has the following implemented endpoints:

- DeleteCourier: Delete a courier (DELETE /courier)
- GetCouriers: Get a collection of couriers (GET /courier)
- AddCourier: Add a courier (POST /courier)
- UpdateCourier: Can be used for pickup the order or deliver the order (PUT /courier)
- GetCouriers: Get a courier by providing id (GET /courier/{:id})

8. Deployment

8.1. Local Run

Backend Repository: https://github.com/CS5500-Awesome404/Food-Delivery.git
To start the backend service locally, in the repo of the backend, run:
$ ./gradlew run

We can test the backend GET APIs listed in Section 6 using this endpoint. For the other types of APIs like POST, DELETE and PUT, we could utilize tools like Postman to perform manual testing against this endpoint.

UI Repository:https://git.heroku.com/quiet-thicket-01896.git
To start the frontend service locally, in the repo of the frontend, run:
$ npm run
We can navigate through different pages mentioned in section 5 to perform tests.

8.2. Heroku

For this project, we utilize Heroku to deploy both the frontend and backend services. Both of the frontend and backend repos are connected with Heroku so that automatic deployments are configured with new changes.

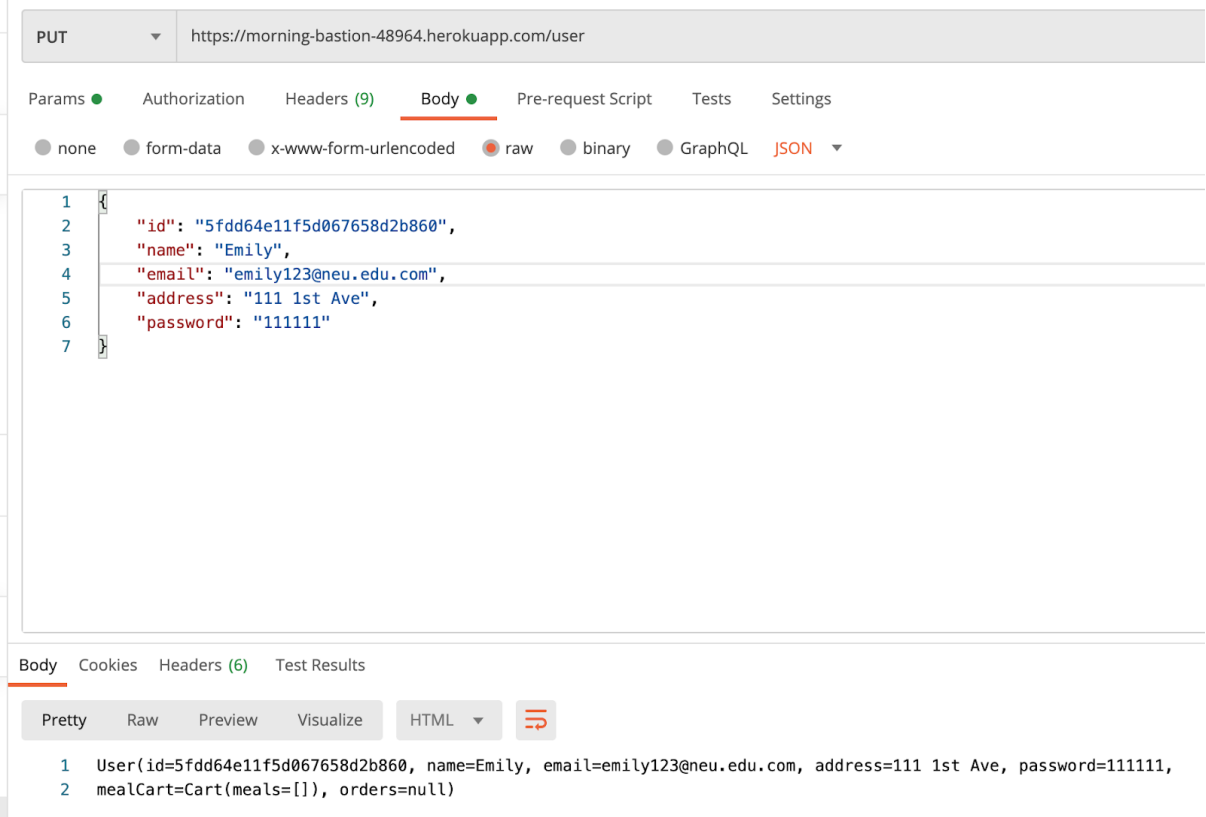
We have deployed both the backend and frontend to the following endpoints.

- Frontend url: https://quiet-thicket-01896.herokuapp.com/login
- Backend url: https://morning-bastion-48964.herokuapp.com/restaurant

9. Testing

Unit Test:
We used unit tests to check the pieces of code in the controller part. Specifically, we used JUnit by adding annotations like @Before and @Test to check the code logics and functionalities.

Postman API Testing:
We used Postman as our API testing tool. Specifically, we provide the urls and request body, and check if we get the corresponding response. More details can be found in the following screenshots:



▸ GetUserById ⚠ 2 issues

GET https://morning-bastion-48964.herokuapp.com/user/5fdd63111f5d067658d2b85e

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

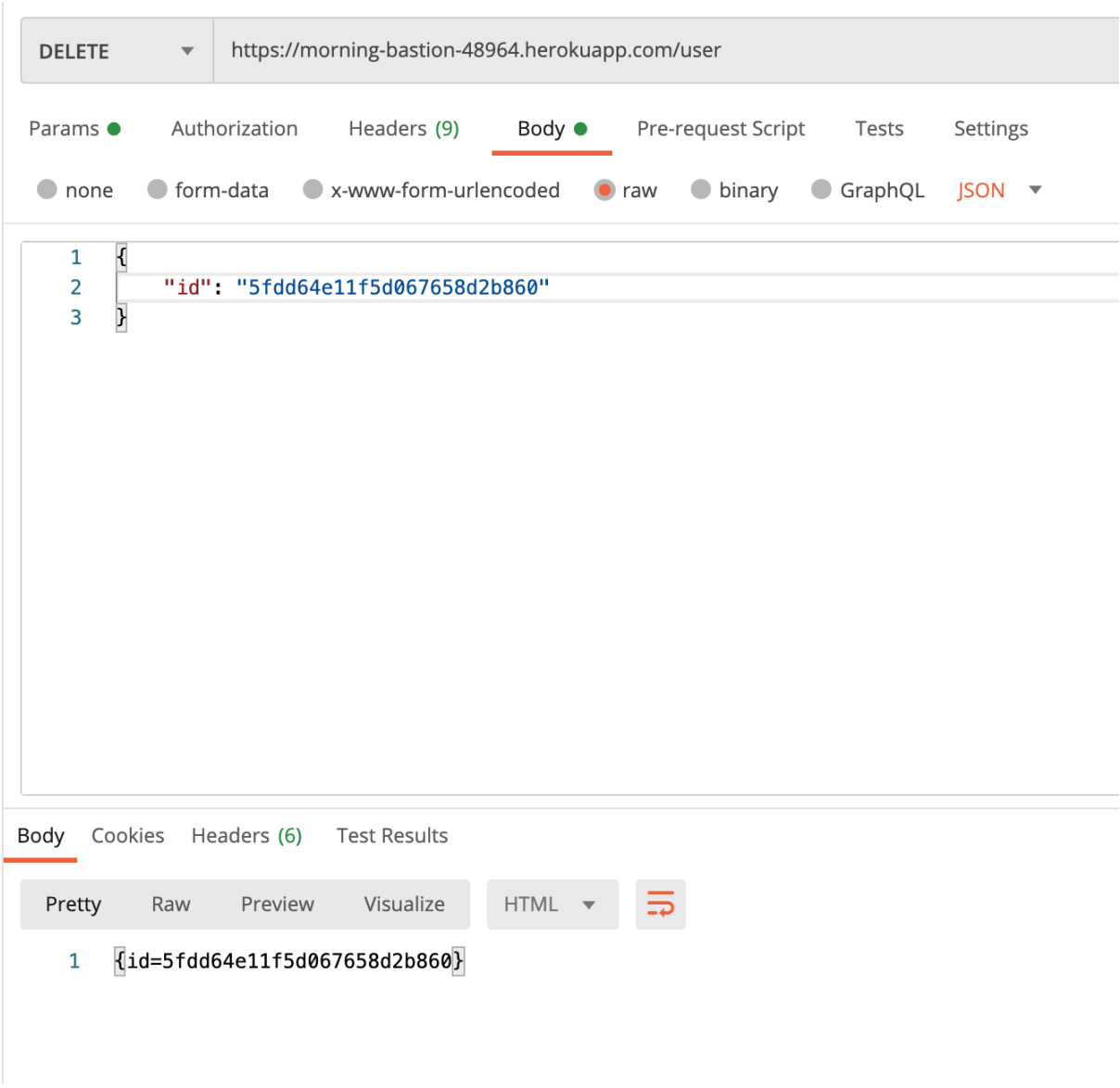
none form-data x-www-form-urlencoded raw binary GraphQL

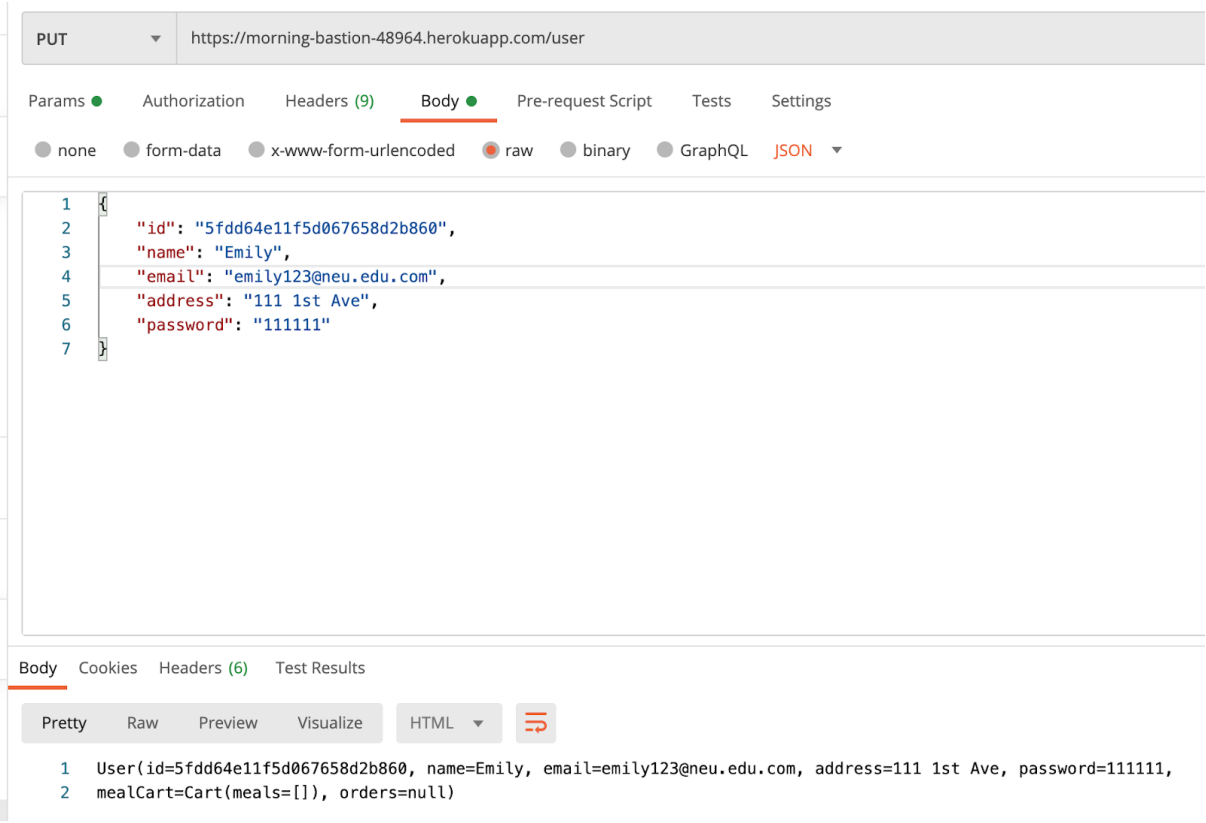
This

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "id": "5fdd63111f5d067658d2b85e",
    "name": "lllllll",
    "email": "llllll",
    "address": "llllll",
    "password": "llllll",
    "mealCart": {
        "meals": [
            {
                "meal": {
                    "mealName": "Meal 1",
                    "mealId": "5fdd30e8ca5fae5383e48dae",
                    "mealPrice": 11.059999999999999,
                    "pictureUrl": null
                },
                "quantity": 3
            }
```


DELETE
https://morning-bastion-48964.herokuapp.com/user
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
1 {
2 "id": "5fdd64e11f5d067658d2b860"
3 }
Body
Cookies
Headers (6)
Test Results
Pretty
Raw
Preview
Visualize
HTML
1 {id=5fdd64e11f5d067658d2b860}


PUT
https://morning-bastion-48964.herokuapp.com/user
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"id": "5fdd64e11f5d067658d2b860",
"name": "Emily",
"email": "emily123@neu.edu.com",
"address": "111 1st Ave",
"password": "111111"
}

Body
Cookies
Headers (6)
Test Results
Pretty
Raw
Preview
Visualize
HTML
User(id=5fdd64e11f5d067658d2b860, name=Emily, email=emily123@neu.edu.com, address=111 1st Ave, password=111111,
mealCart=Cart(meals=[]), orders=null)

POST https://morning-bastion-48964.herokuapp.com/user

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "id": "5fae04aaa40ba56ebdddf8a",
    "name": "Emily",
    "email": "emily@gmail.com",
    "address": "1111 1st Ave",
    "password": "111111"
}
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "id": "5fdd755d1f5d067658d2b862",
    "name": "Emily",
    "email": "emily@gmail.com",
    "address": "1111 1st Ave",
    "password": "111111",
    "mealCart": {
        "meals": []
    },
    "orders": null
}
```

GET https://morning-bastion-48964.herokuapp.com/user

Params Authorization Headers (6) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": "5fdd29afc670280614bfc461",
        "name": "123",
        "email": "123",
        "address": "123",
        "password": "123",
        "mealCart": {
            "meals": [
                {
                    "meal": {
                        "mealName": "Meal 0",
                        "mealId": "5fdd281473cd161496b3b595",
                        "mealPrice": 1.46,
                        "pictureUrl": null
                    },
                    "quantity": 1
                }
            ]
        },
        "orders": null
    },
    {
```

▸ **UpdateOrderStatus**

PUT https://morning-bastion-48964.herokuapp.com/order

Params Authorization Headers (9) Body ● Pre-request Script Tests Settin

○ none ○ form-data ○ x-www-form-urlencoded ◉ raw ○ binary ○ GraphQL JSC

```
1  {
2      "id": "5fdd35e6812e52637fe2262f",
3      "status": "DELIVERED"
4  }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

```
1  ""
```

GetOrderById 2 issues

GET https://morning-bastion-48964.herokuapp.com/order/5fdd35e6812e52637fe2262f

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "id": "5fdd35e6812e52637fe2262f",
    "status": "CREATED",
    "orderContent": {
        "meals": [
            {
                "meal": {
                    "mealName": "Meal 2",
                    "mealId": "5fdd30e8ca5fae5383e48daf",
                    "mealPrice": 9.07,
                    "pictureUrl": null
                },
                "quantity": 1
            },
            {
                "meal": {
```

DeleteOrder 1 issue

DELETE https://morning-bastion-48964.herokuapp.com/order

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "id": "5fdd2a25c670280614bfc464"
3 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

```
1 {
2 "id" : "5fdd2a25c670280614bfc464"
3 }
```

▸ **PlaceOrder**

POST https://morning-bastion-48964.herokuapp.com/order

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

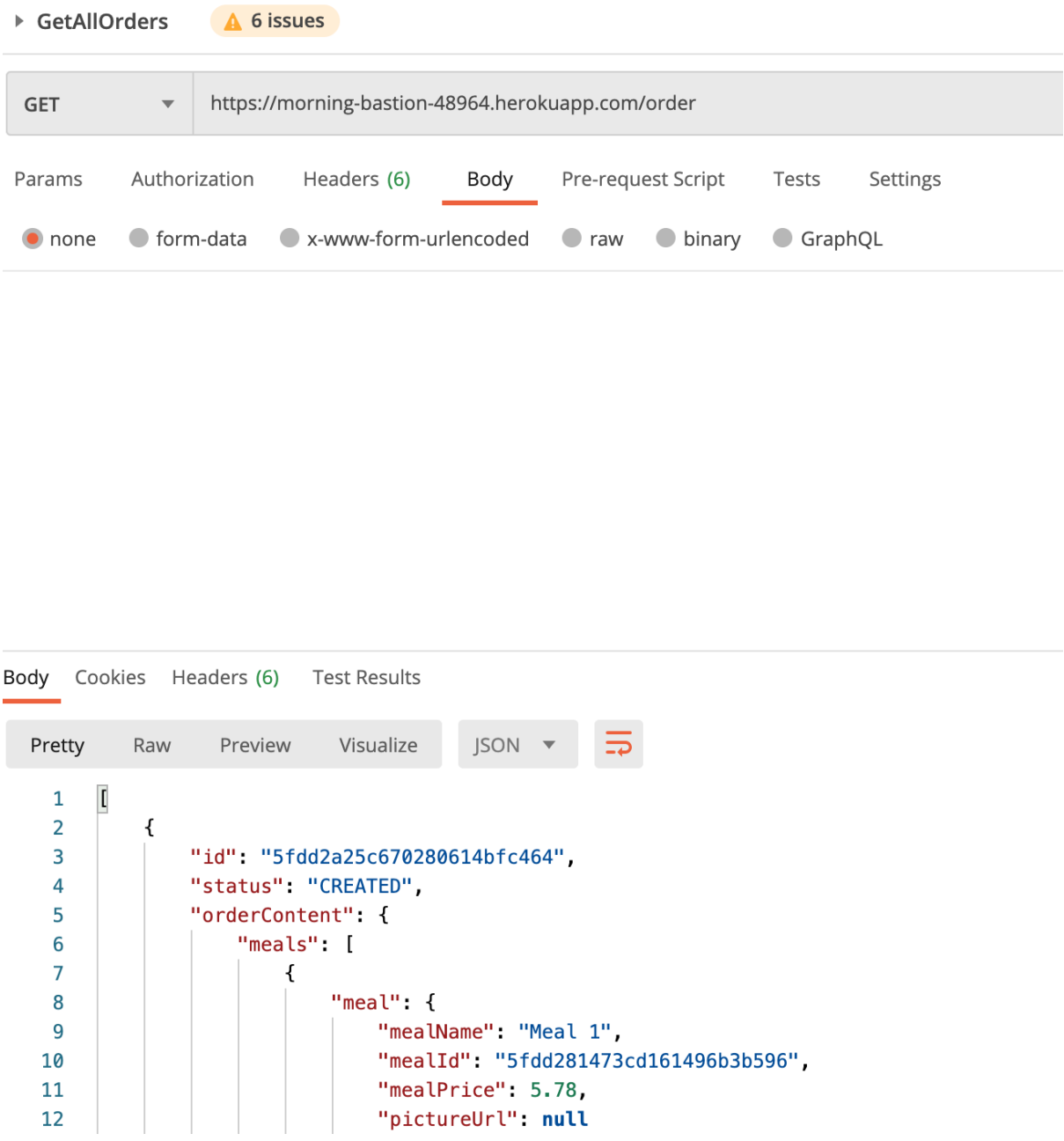
○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
1  {
2      "userId": "5fdd2a25c670280614bfc466"
3  }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize HTML

```
1
```



10. Future Work

We currently only implemented the basic functionalities to support CRUD operations against the designed backend database tables. There are tons of cool features that could be implemented in the future, for example, to support payment, to support promotion of deals, to support drive location tracking etc.