Cart Management API Documentation

Introduction

This documentation serves to help the user how this API works. It contains information about the pre-requisites, endpoints, connections as well why this API is developed. All technical documentation at docs folder inside the root folder.

Version history

Version	Author	Notes
1.0	Victor Barros Bologna	Initial version

Prerequisites

Make sure that you have the following installed in your system before use:

- Java 11 SDK;
- Docker;
- Maven.

General purpose

This API is a REST based application whose purpose is to store items from carts in the database as JSON. All carts must be validated through the Supermarket API (Mocked with Mockoon) before being inserted into a new database, so the user doesn't insert any cart that doesn’t exist in the Supermarket API.

How to run the project:

First you need to download the API, which you can find it here :victor-bologna/cart (github.com).There's two ways to start the application, either via Command line or Docker. In either way the application requires two images to be running as shown in the Dependencies section.

Dependencies:

First, in order to make this application to work, user must use Docker to download and configure MySQL and Mockoon:

1. Create a new network to be used for MySQL and your application:

`docker network create supermarket`

2. The first service is a MySQL database service. To start it, run the following command line at Windows PowerShell:

`docker container run --name mysql --network=supermarket -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=cartdb -p 3306:3306 -p 33060:33060 -d mysql`

3. The second is the Supermarket API, in this case we are mocking with Mockoon, so let's build an image with the following command in Windows Powershell:

`docker run --name supermarket-mockoon --network=supermarket -d -p 8080:8080 mockoon/cli:latest -d https://raw.githubusercontent.com/victor-bologna/cart/main/mockoon/Supermarket%20API.json -p 8080`

4. With Windows Powershell still opened, type the following:

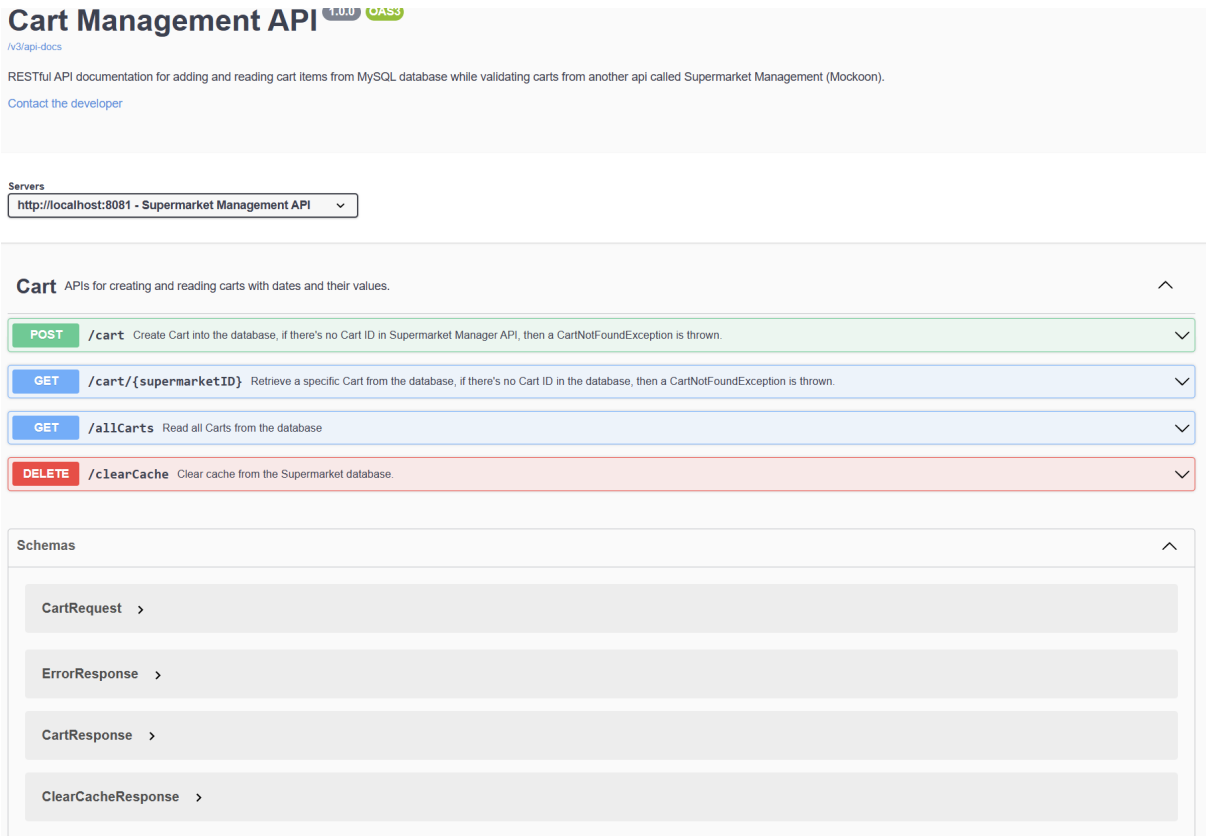
`docker container run --name cart-management --network=supermarket -p 8081:8081 -d wricked/cart-management;`

That would start the applications, to stop a specific container, first execute `[docker ps]` to get container's id. Then, execute `[docker stop {id}]` to stop the specific container. And use `[docker start {name}]` to restart the existing container. Also user can use Docker App to stop and run applications easily.

Endpoints

The API is accessible through URL address http://localhost:8081 and contains 4 endpoints:

Method	Endpoint	Observations
POST	/cart	Creates a cart with specified items and their values for a specific supermarket. This endpoint also automatically stores all Supermarket IDs on the first run as cache from Supermarket API and validates if requested payload's cart ID exists until cache is cleared.
GET	/cart/{supermarketID}	Get all items from a specific supermarket by ID.
GET	/allCarts	It automatically retrieves all supermarket's carts and items from the database.
DELETE	/clearCache	This endpoint is usable through the Supermarket API when a new supermarket is registered on that API.



More information about the endpoints is found here (requires application started)
http://localhost:8081/swagger-ui/index.html

Examples of Payload

POST /cart:

Request Payload:

```
{
   "supermarketID": "1",
   "items":
      {
      "Onions": 15.50
   }
}
```

Response - 201 Created:

```
{
   "id": "029b6ebc-4c1e-45e7-98d5-5e00ed670efd",
   "supermarketID": 1,
   "name": "Carrefour",
   "items": {
      "Onions": 15.5
   }
}
```

GET /cart/1:

Request Payload:

None

Response - 200 OK:

```
[
   {
      "id": "0287d96d-fbd8-4da3-8459-c925b1bff93d",
      "supermarketID": 1,
      "name": "Carrefour",
      "items": {
         "PC": 600.2,
         "Sauce": 5.5
      }
   },
   {
      "id": "029b6ebc-4c1e-45e7-98d5-5e00ed670efd",
      "supermarketID": 1,
      "name": "Carrefour",
      "items": {
         "Onions": 15.5
      }
   }
]
```

GET /allcarts:

Request Payload:
None
Response - 200 OK:


```
[
   {
      "id": "3ff7f91f-26ff-4609-9bb7-accc301a68bb",
      "supermarketID": 2,
      "name": "Makro",
      "items": {
         "Salt": 5.5,
         "Onion": 6.2
      }
   },
   {
      "id": "5afcc030-dd6d-4cd7-93e1-fdfd6c3f97f7",
      "supermarketID": 3,
      "name": "Assai",
      "items": {
         "PC": 500.2,
         "Sauce": 8.5,
         "Bicycle": 106.2
      }
   },
   {
      "id": "da58e3bc-3416-4076-944b-eda497bf460b",
      "supermarketID": 1,
      "name": "Carrefour",
      "items": {
         "Chicken Breasts": 10.5,
         "Bread": 5.3
      }
   }
]
```


DELETE /clearCache:

Request Payload:
None
Response - 200 OK:


```
{
   "message": "Cache cleared successfully."
}
```

How Cache works

In order to increase the performance, the API caches the data retrieved for the first time from Supermarket API in endpoint GET http://localhost:8080/supermarket and it will be only cleaned up when the user adds a new cart ID in Supermarket API manually, which results in Supermarket API to execute DELETE http://localhost:8081/clearCache in Supermarket API and clear the cache.

Connections

This API connects itself with MySQL server and Supermarket API both through Docker.

- To connect to MySQL to perform queries the Supermarket API uses CartService to send and retrieve cart data with their respectives Carts;
- Mockoon to use as Supermarket API to get all Supermarket data before inserting carts in.