GET STARTED WITH STOCK REST API

Create Database

```
CREATE SCHEMA StockSalesDB;

CREATE USER 'admin'@'localhost' IDENTIFIED WITH mysql_native_password BY 'P@ssw0rd';

ALTER USER 'admin'@'localhost' IDENTIFIED WITH mysql_native_password BY 'P@ssw0rd';

GRANT ALL PRIVILEGES ON StockSalesDB.* TO 'admin'@'localhost';
```

Postman documentation you can use to run endpoints:
There are 3 folders, 1 for the Registered user role, 2 for Guest users, and 3 for Admin users.

Every Folder needed its token, only guest users should not.

```
https://documenter.getpostman.com/view/22916278/2s93sdYs2P
```

Clone the repo on your machine

```
git clone https://github.com/NORZACO/Mwamuzi_Shadrick_EP_CA_AUG22FT.git
```

Go inside the app directory and install packages by

```
npm install
```

I have made it easy for you regarding the .env variables: This command provides all variables you need

```
npm run token
```

looks something like this below:

```
#DATABASE CONNECTION VARIABLES
HOST=127.0.0.1
ADMIN_USERNAME=admin
ADMIN_PASSWORD=P@ssw0rd
DATABASE_NAME=StockSalesDB
DIALECT=mysql
PORT=3000



#DROP_ALL_DATA_OR_SYNCHRONISE=true will drop all the data from the database if it is true or just synchronise th
DROP_ALL_DATA_OR_SYNCHRONISE=true
#guest id that will be added in database role id for guest user
ACCESS_GUEST_ROLE=guest-user-a098fd18597ee00



#Showing data for admin
ACCESS_Admin_TOKEN=Admin
```

```
#Showing data dor registered user
ACCESS_Register_TOKEN=Register
#Showing data for guest
ACCESS_Guest_TOKEN=Guest



#THIS IS OUR SECRET GENERATED ACCESS TOKEN
ACCESS_TOKEN_SECRET=7145cfa9723656fed1de4885053ad0e13ae4d29425c953924a9510eccd1760c1604f447b0ae20fc416a85cffa463

```

when viewing items endpoints you may see the name in like `Admin` Product. `Registered` Product and `Guest` Product. this is just to make it easy for you to see which user is requesting.

Running on Mac. you need to have nodemon installed on your machine to use npm run dev (easy for debugging).
if you do not want to use nodemon replace `npm run dev` with `npm start`

```
 DEBUG=Mwamuzi_Shadrick_EP_CA_AUG22FT:* npm run dev
```

run on Window

```
SET DEBUG=Mwamuzi_Shadrick_EP_CA_AUG22FT:* & npm run dev
```

Create all carts VIEWS. Run this SQL code on your MySQL workbench

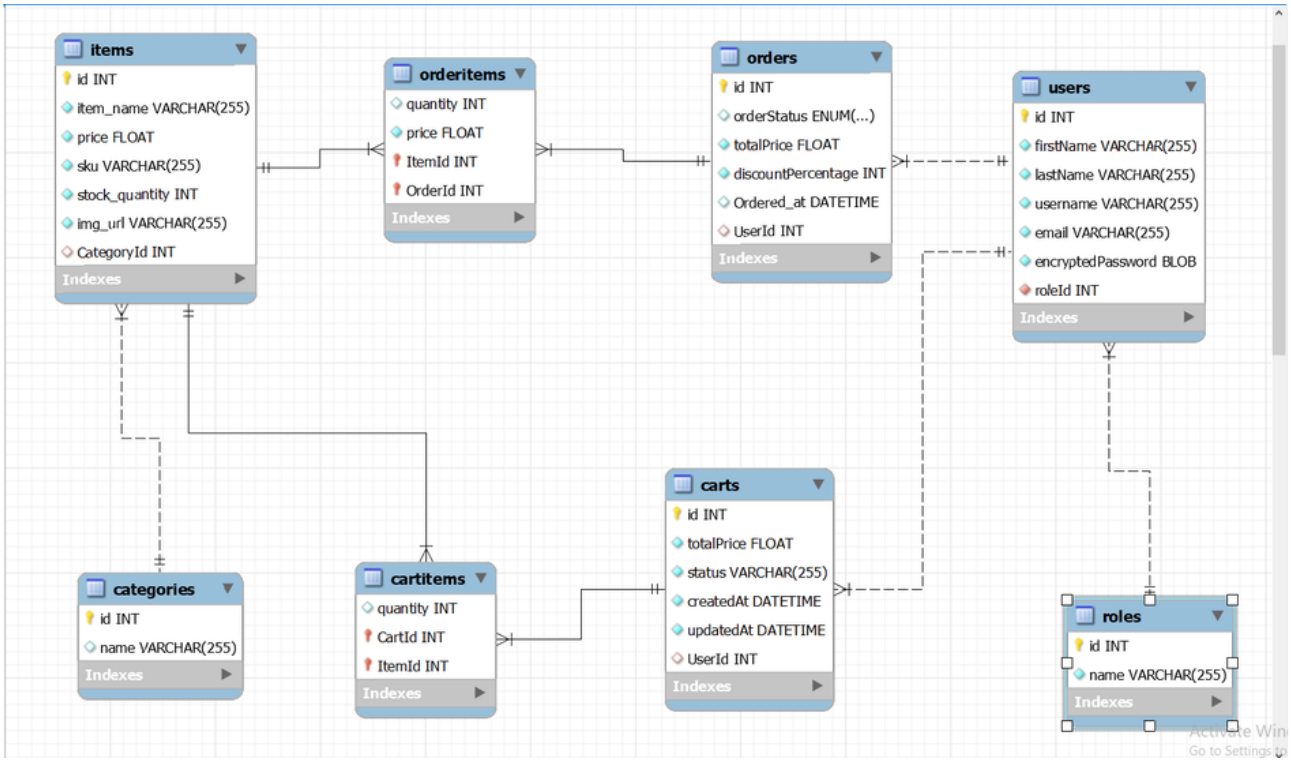
```
-- all carts views'
CREATE VIEW
        allcarts AS
SELECT
        Carts.id,
        Carts.UserId,
        Carts.totalPrice,
        -- Users.firstName,
        -- Users.lastName,
        CONCAT(Users.firstName, ' ' ,Users.lastName) AS FullName,
        Items.id AS item_id,
        Items.item_name,
        Items.price,
        Items.sku,
        Items.stock_quantity,
        CartItems.quantity
FROM
        Carts
        INNER JOIN Users ON Carts.UserId = Users.id
        INNER JOIN CartItems ON Carts.id = CartItems.CartId
        INNER JOIN Items ON CartItems.ItemId = Items.id;
```

Create all orders VIEWS. Run this SQL code on your MySQL workbench

```
-- all order views
CREATE VIEW
```

```
        allorders AS
SELECT
        Orders.id,
        Orders.UserId,
        Orders.totalPrice,
        Orders.orderStatus,
        CONCAT(Users.firstName, ' ' ,Users.lastName) AS FullName,
        Items.id AS item_id,
        Items.item_name,
        Items.price,
        Items.sku,
        Items.stock_quantity,
        OrderItems.quantity
FROM
        Orders
        INNER JOIN Users ON Orders.UserId = Users.id
        INNER JOIN OrderItems ON Orders.id = OrderItems.OrderId
        INNER JOIN Items ON OrderItems.ItemId = Items.id;
```

YOUR DATABASE RELATIONSHIP LOOKS LIKE THIS



The database name is `stocksalesdb`. Here's a little summery of the tables above.

1. `carts`: This table seems to represent shopping carts. Each cart has an `id`, `totalPrice`, `status`, `createdAt`, `updatedAt`, and `UserId`.
2. `cartitems`: This table represents the items in each cart. Each row has a `quantity`, `CartId`, and `ItemId`.
3. `items`: This table represents the items available for purchase. Each item has an `id`, `item_name`, `price`, `sku`, `stock_quantity`, `img_url`, and `CategoryId`.

4. `orders`: This table represents the orders placed by users. Each order has an `id`, `orderStatus`, `totalPrice`, `discountPercentage`, `Ordered_at`, and `UserId`.
5. `orderitems`: This table represents the items in each order. Each row has a `quantity`, `price`, `ItemId`, and `OrderId`.
6. `users`: This table represents the users of the app. Each user has an `id`, `firstName`, `lastName`, `username`, `email`, `encryptedPassword`, and `roleId`.
7. `roles`: This table represents the roles that users can have. Each role has an `id` and `name`.
   this is very important because it has admin, guest, and registered roles that give the user access to endpoints.
8. `categories`: This table represents the categories that items can belong to. Each category has an `id` and `name`.
9. `allcarts`: This is a view that joins the `carts`, `users`, `cartitems`, and `items` tables to provide a comprehensive view of all carts, including the items in them and the users they belong to.
10. `allorders`: This is a view that joins the `orders`, `users`, `orderitems`, and `items` tables to provide a comprehensive view of all orders, including the items in them and the users they belong to.

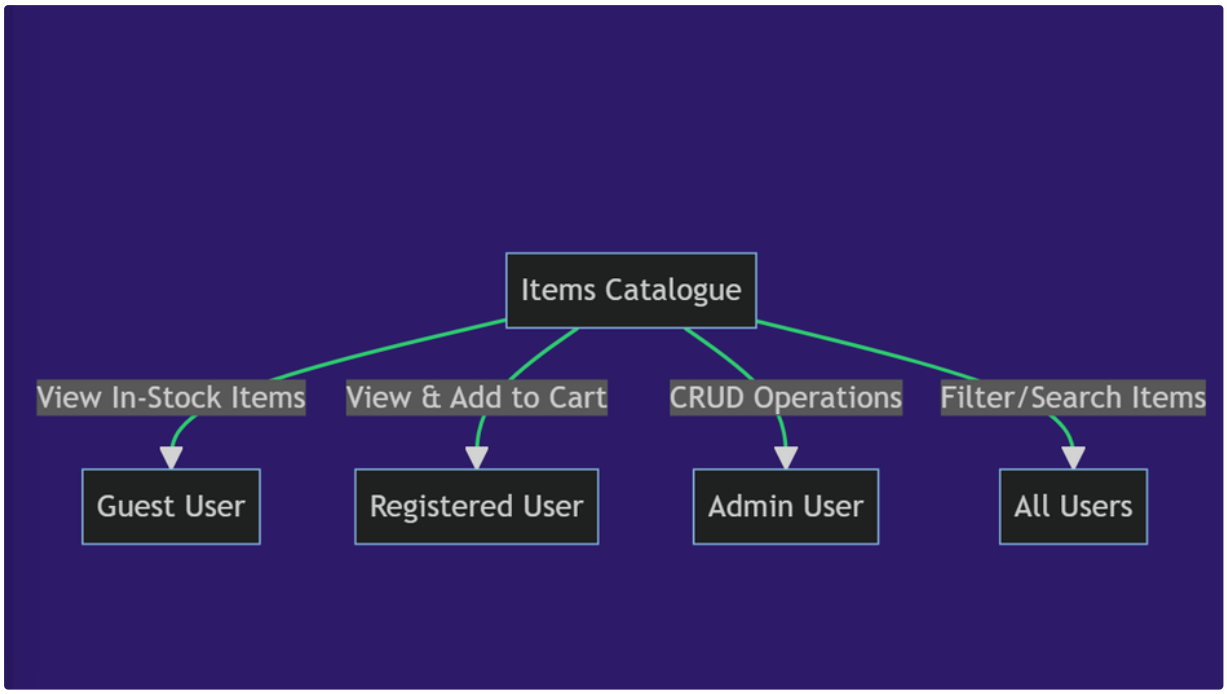
- **In-stock** means that stock quantity is greater than zero.
- Out-of-stock means that stock quantity is equal to zero.

Authentication:

Your back-end should implement Basic authentication, having multiple User types:

1. **Guest User - without Logging-in user**
2. **Registered User - Logged-in user**
3. **Admin User -** /setup API endpoint. I.e., Cannot be registered, 1x Admin User with the 'Admin' role.

diagram created on About Mermaid | Mermaid



Login/Registration:

unique username, password, and email address

4x different users can register with the same email address

Login/Registration:

- [ ] **Run set up, then. In order for the API to work correctly, you must first run setup.**
  This is also very important. In order for the RESP API to run properly as indicated

```
// POST /setup
```

- [ ] Register User + No duplicate usernames. + error handler.
- [ ] Create a role name ‘`Registered'`
  after creating a role, you will receive the role’s id
  - [x] when creating a user, refer to the role name ‘`Registered'`
    if you forget a role id, go to ‘{{BaseUrl}}/roles’ and check all role names and their ids.

```
// POST /signup
{
    "firstName": "Alice",
    "lastName": "Smith",
    "username": "alice_smith",
    "email": "alice.smith@example.com",
    "password": "AlicePassword1#",
    "roleId": "ed4cc590-aeb5-455e-8c64-4f51a7359583"
}
```

```
// POST /signup
{
    "firstName": "Bob",
    "lastName": "Johnson",
    "username": "bob_johnson",
    "email": "bob.johnson@example.com",
    "password": "BobPassword2#",
    "roleId": "ed4cc590-aeb5-455e-8c64-4f51a7359583"
}
```

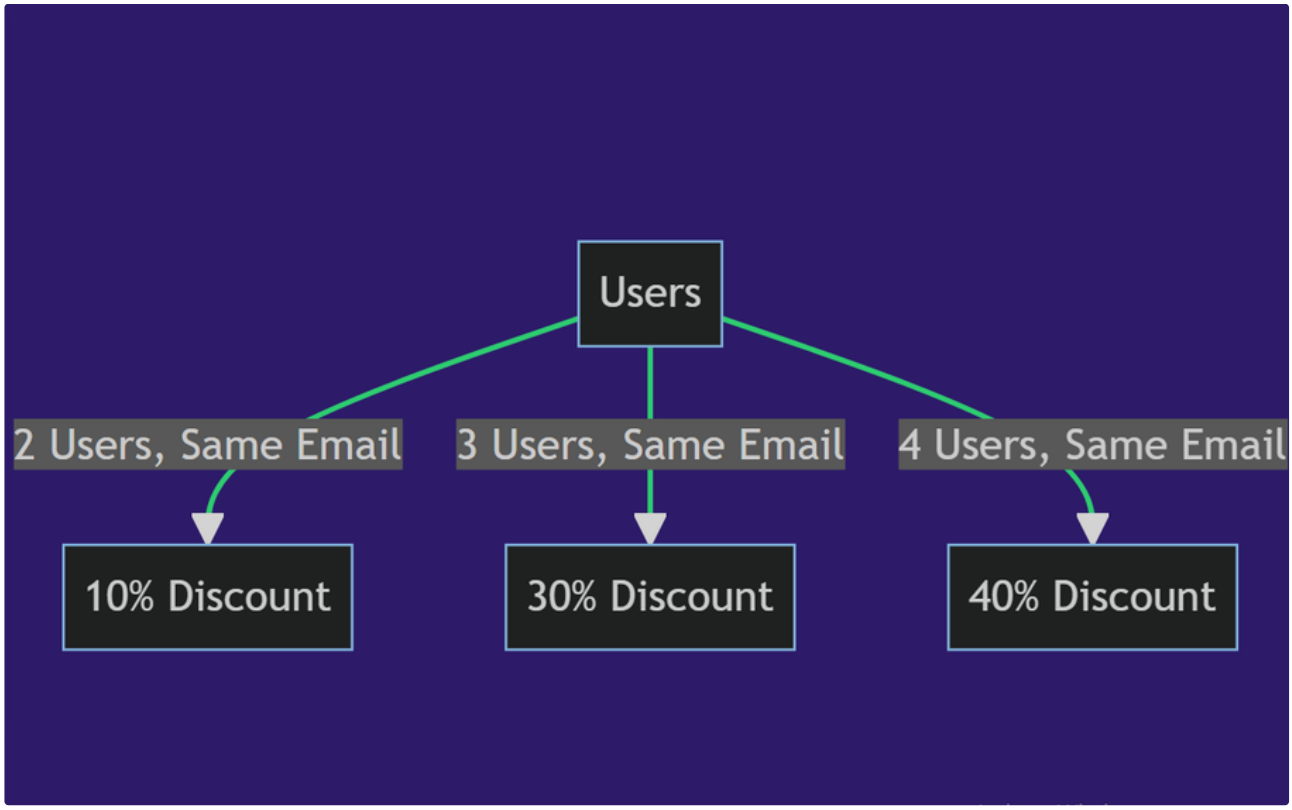
```
// POST /signup
{
    "firstName": "Charlie",
    "lastName": "Brown",
    "username": "charlie_brown",
    "email": "charlie.brown@example.com",
    "password": "CharliePassword3#",
    "roleId": "ed4cc590-aeb5-455e-8c64-4f51a7359583"
}

```

- [ ] Admin User populated during setup initial data

```
POST /setup
{
    "firstName": "Admin",
    "lastName": "User",
    "username": "admin_user",
    "email": "admin.user@example.com",
    "password": "AdminPassword#",
    "roleId": admin role id
}
```

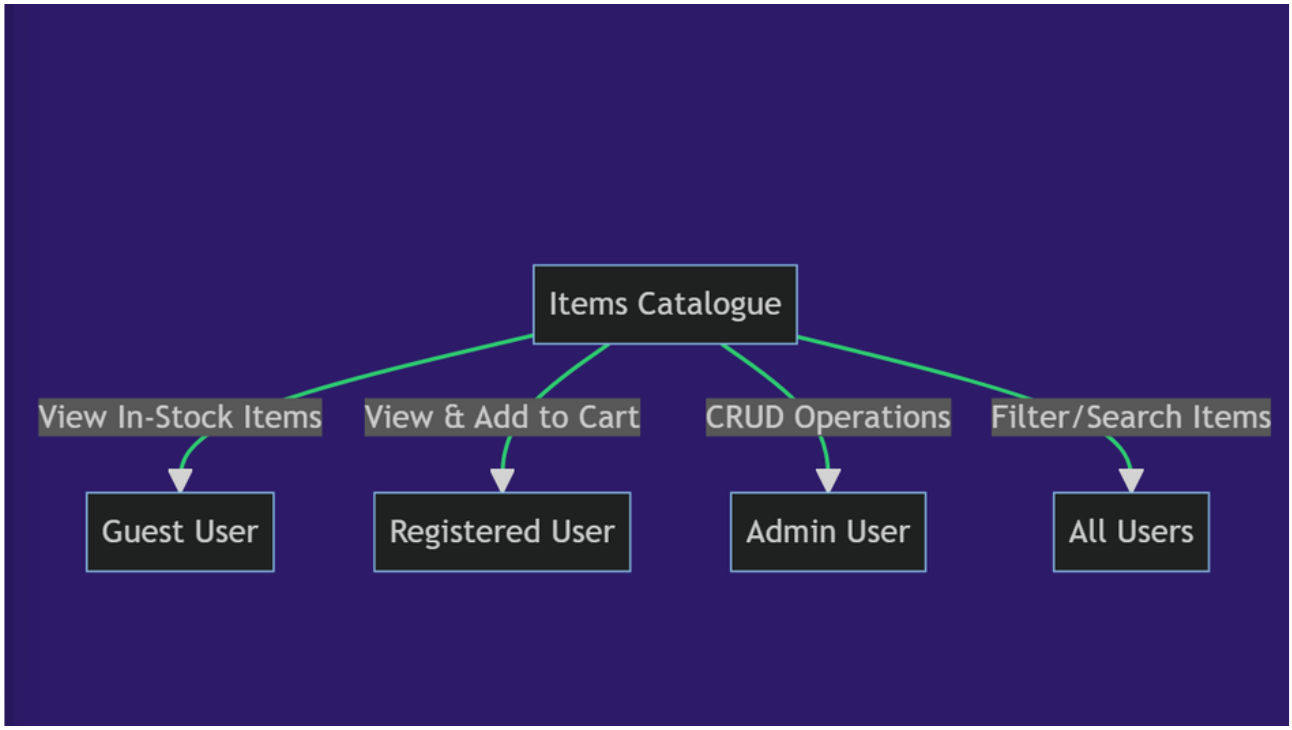
```
10
```

FAMILY WITH THE SAME EMAIL ADDRESS



- [ ] **Items Catalogue:**

items for sale are shown, with their name, price, and stock levels.
Registered: + check out



items for sale are shown, with their name, price, and stock levels.

Registered: + check out

Authentication:

Authentication should be implemented with JWT. The JWT token should expire in 2 hours. Other than Logging-in or Registering, **no User credentials should be sent in the URL Path of any API endpoint**.