Documentation for Restaurant Menu Manager

RestaurantMenuManager

Technologies and Tools Used:

- **Spring Boot 3.2.2:** Main framework for building the application.
- **Spring Data JPA:** For database interaction and CRUD operations.
- **H2 Database:** Used for in-memory database testing.
- **JUnit 5:** For unit testing.

Introduction:

The Restaurant Menu Manager system is a RESTful API designed to manage the menu of a restaurant. It allows users to add, retrieve, update, and delete menu items. Built using **Spring Boot**, the application interacts with a database to persist data related to menu items, such as the dish name, price, and description.

Detailed Explanation of the Project:

1. Entry Point: `RestaurantMenuApplication.java`

This is the main class of the application that starts the Spring Boot server and initializes the application context.

2. Model: `MenuItem.java`

Represents a dish or item in the restaurant's menu. Each menu item includes fields such as `name`, `price`, and `description`. The class uses JPA annotations to map to the database.

- **Key Fields**:
  - `name`: Name of the dish.
  - `price`: The price of the dish.
  - `description`: A short description of the dish.
- **Annotations**:
  - `@Entity`: Specifies the class as an entity.
  - `@Id`: Identifies the primary key.
  - `@GeneratedValue`: Specifies that the ID is auto-generated.

3. Persistence Layer: `MenuItemRepository.java`

This interface extends **JpaRepository** and provides CRUD operations for `MenuItem` objects, allowing for easy database interactions.

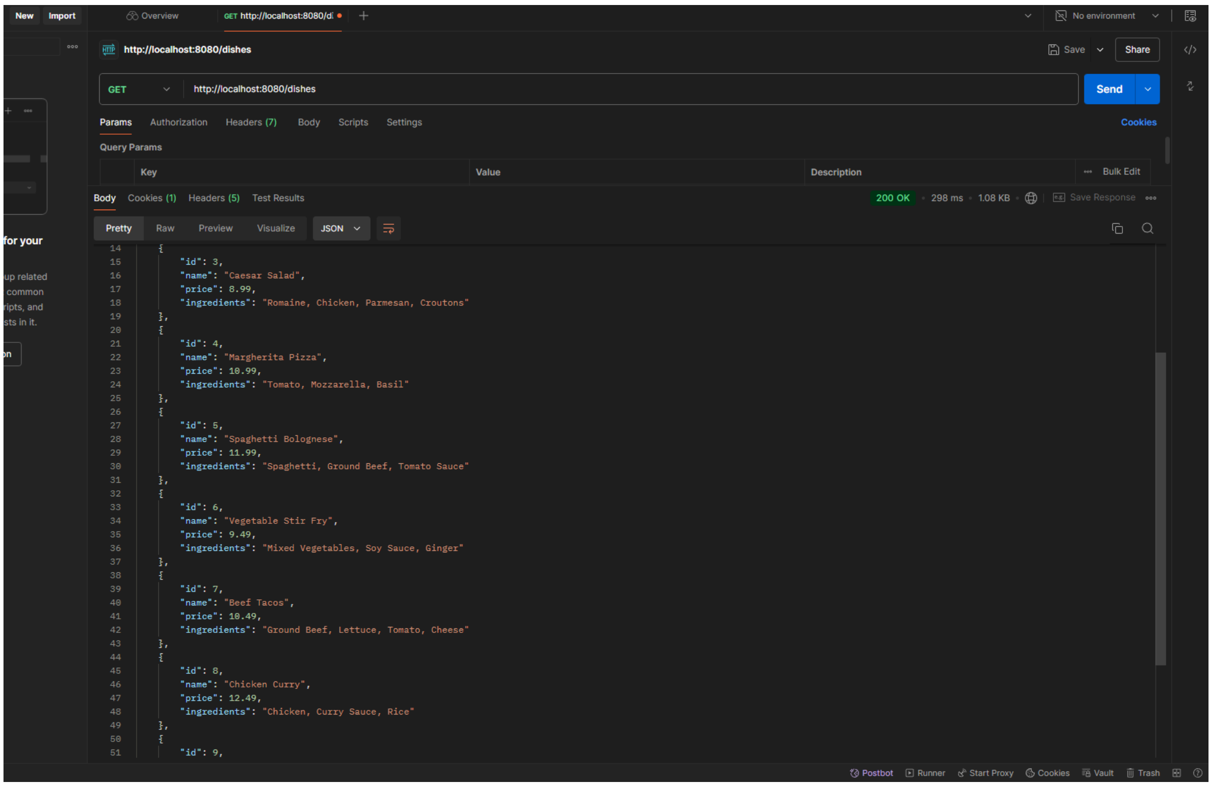
4. Service Layer: `MenuItemService.java`

The service contains the business logic related to menu items. It interacts with the `MenuItemRepository` to fetch, add, update, or delete items from the menu.

5. Controller Layer: `MenuItemController.java`

Handles incoming HTTP requests to manage menu items.

- **Endpoints**:
  - `GET http://localhost:8080/dishes/{id}` Retrieves specific item from the menu.
  - `GET http://localhost:8080/dishes` Retrieves all items from the menu.



```
    {
        "id": 3,
        "name": "Caesar Salad",
        "price": 8.99,
        "ingredients": "Romaine, Chicken, Parmesan, Croutons"
    },
    {
        "id": 4,
        "name": "Margherita Pizza",
        "price": 10.99,
        "ingredients": "Tomato, Mozzarella, Basil"
    },
    {
        "id": 5,
        "name": "Spaghetti Bolognese",
        "price": 11.99,
        "ingredients": "Spaghetti, Ground Beef, Tomato Sauce"
    },
    {
        "id": 6,
        "name": "Vegetable Stir Fry",
        "price": 9.49,
        "ingredients": "Mixed Vegetables, Soy Sauce, Ginger"
    },
    {
        "id": 7,
        "name": "Beef Tacos",
        "price": 10.49,
        "ingredients": "Ground Beef, Lettuce, Tomato, Cheese"
    },
    {
        "id": 8,
        "name": "Chicken Curry",
        "price": 12.49,
        "ingredients": "Chicken, Curry Sauce, Rice"
    },
    {
        "id": 9,
```

Database Configuration: `application.properties`

The database connection and JPA settings are configured to persist menu items. It uses either MySQL, PostgreSQL, or an in-memory H2 database for development:

properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/restaurant_db

spring.datasource.username=root

spring.datasource.password=your_password_here

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true
```

Unit Testing with JUnit/Mockito

Unit tests for the service and controller layers are written using **JUnit** and **Mockito**. The tests mock the behavior of the repository layer to ensure that the business logic and controllers function correctly without hitting a real database.