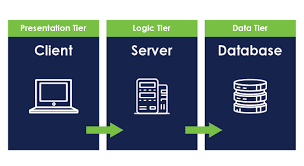
### **System Design for Agricultural Products Selling Store**

**Project Title: SPRING FOOD**  
**Technology Stack**: Java (Backend), Angular (Frontend), MySQL (Database)

### **1. System Architecture Overview**

The Agricultural Products Selling Store will follow a **3-tier architecture**:

1. **Frontend (Client-Side)**: Angular will be used to build the user interface that interacts with the users (both merchants and customers). It will be a Single Page Application (SPA) to ensure fast and responsive user experiences.
2. **Backend (Server-Side)**: The backend will be built with Java, providing RESTful APIs for communication with the frontend. The backend will handle business logic, authentication, authorization, and interaction with the database.
3. **Database (Data Layer)**: MySQL will be used for storing structured data such as user information, products, and orders. It will be responsible for ensuring data integrity and supporting complex queries.



### **2. System Components**

#### **2.1 Frontend (Angular)**

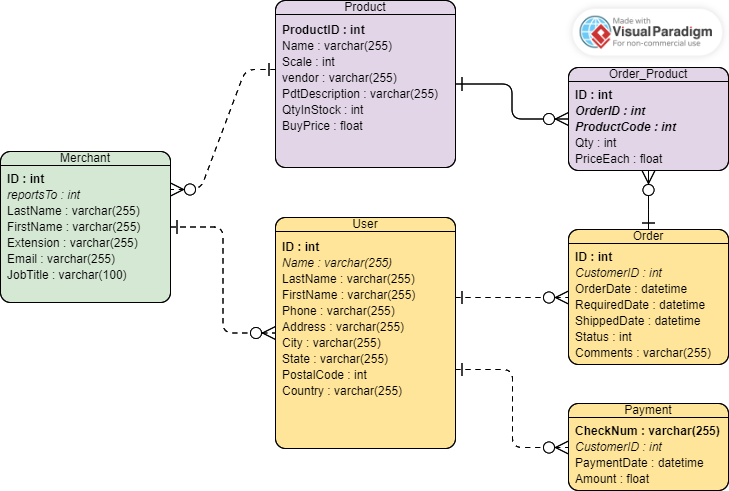
* **Purpose**: To provide an interactive and responsive user interface for merchants and customers.
* **Components**:
  + **Merchant Dashboard**: For product management and order handling.
  + **User Dashboard**: For browsing products, adding them to the cart, and tracking orders.
  + **Authentication**: Login, registration, and session management using JWT tokens.
* **Communication with Backend**: Uses HTTP/HTTPS to communicate with the Java backend through RESTful API calls.
* **Key Modules**:
  + **Product Management Module**: Displays product catalog, allows merchants to add/edit/delete products.
  + **Shopping Cart Module**: Allows users to add, view, and checkout products.
  + **Order Tracking Module**: Displays status updates on user orders.
  + **Authentication Module**: Manages login, registration, and access control based on user roles (merchant or user).

#### **2.2 Backend (Java + Spring Boot)**

* **Purpose**: To handle business logic, authentication, product management, and database interactions.
* **Key Components**:
  + **RESTful API Layer**: Handles all requests from the frontend, routing them to appropriate services. Uses Spring Boot to expose endpoints for merchants and users.
  + **Service Layer**: Contains business logic for managing users, products, and orders.
  + **Authentication/Authorization**: Implemented with JWT (JSON Web Token) for securing routes based on user roles.
  + **Persistence Layer**: Communicates with the MySQL database via JPA (Java Persistence API). It manages CRUD operations for products, orders, and users.
* **Core Services**:
  + **User Service**: Manages user registration, login, password management, and role-based access.
  + **Product Service**: Manages product creation, updating, deleting, and fetching details.
  + **Order Service**: Manages user orders, including order placement, status updates, and order history.

#### **2.3 Database (MySQL)**

* **Purpose**: To store persistent data for users, products, and orders.
* **Users Table**:
  + User ID (Primary Key)
  + Username
  + Password (Encrypted)
  + Role (Merchant or User)
  + Email
  + Contact Information
* **Products Table**:
  + Product ID (Primary Key)
  + Merchant ID (Foreign Key from Users Table)
  + Name
  + Description
  + Category
  + Price
  + Stock
* **Orders Table**:
  + Order ID (Primary Key)
  + User ID (Foreign Key from Users Table)
  + Total Price
  + Status (Pending, Shipped, Completed)
  + Order Date
* **Order Details Table**:
  + Order Details ID (Primary Key)
  + Order ID (Foreign Key from Orders Table)
  + Product ID (Foreign Key from Products Table)
  + Quantity
  + Price
* **Database Communication**: The backend server (Java) will use JPA for Object Relational Mapping (ORM) to interact with the MySQL database.

**ERD Diagram**

### **4. Data Flow**

#### **4.1 User Registration and Login**

1. **Frontend**: User fills out registration form → Angular sends a POST request to /auth/register.
2. **Backend**: Java processes the request, saves user data in the MySQL Users table, and returns a success response.
3. **Frontend**: On successful registration, users can log in with their credentials, and a JWT token is returned on successful login.

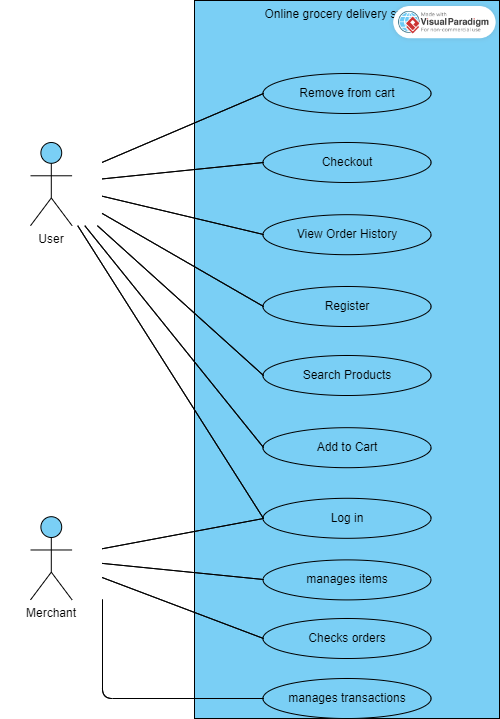
#### **4.2 Product Management (Merchant)**

1. **Frontend**: Merchant fills out a product form (add/update) → Angular sends a POST/PUT request to /merchant/product.
2. **Backend**: Java validates the request and persists the product data in the Products table of MySQL.
3. **Frontend**: Product is displayed on the merchant's product management page.

#### **4.3 Order Placement (User)**

1. **Frontend**: User adds products to their cart and places an order → Angular sends a POST request to /user/order.
2. **Backend**: Java processes the order, creates an entry in the Orders and Order Details tables, and returns the order confirmation.
3. **Frontend**: Order status and details are shown in the user's order history.

### **5. Use Case Diagram**



### **6. Security Considerations**

* **JWT Authentication**: Secure API endpoints by validating JWT tokens.
* **Role-Based Access Control (RBAC)**: Ensure that merchants and users only access permitted functionalities.
* **SQL Injection Protection**: Use prepared statements in MySQL queries.
* **Password Encryption**: Use hashing to store user passwords securely.

### **7. Scalability Considerations**

* **Load Balancing**: Use a load balancer for distributing traffic across multiple backend instances.
* **Database Optimization**: Index frequently queried fields and implement database caching for high-traffic queries.
* **Horizontal Scaling**: Deploy additional instances of the Angular frontend and Java backend services as traffic increases.

### **8. Conclusion**

This system design provides a robust, scalable, and secure architecture for an Agricultural Products Selling Store, meeting both functional and non-functional requirements for merchants and users. It ensures efficient communication between the frontend, backend, and database layers.