Techademy

Capstone Assessment-

Spring Boot/Angular

Mini ECommerce Application

Contents

[1 Problem Statement 2](#_Toc195871645)

[2 Screen Flows 4](#_Toc195871646)

[3 Technical Details 11](#_Toc195871647)

# Problem Statement

**Develop a mini Ecommerce application using ANGULAR and SpringBoot and MySQL.**

The system will consist of three independent microservices:

1. **User Management Microservice**: handles user-related operations
2. **Product Management Microservice**: manages product catalogues and inventory
3. **Order Management Microservice**: processes and manages orders

**Primary Actors:**

1. Admin: responsible for managing the product catalogue and inventory
2. Customer: browses and purchases products from the catalogue

Basic Flow:

Product Management:

1. Admin adds a new product:
   * Admin submits a new product request with details (name, description, price, quantity)
   * Product service validates the request and creates a new product entity
   * Product service returns a success response to the admin
2. Admin updates an existing product:
   * Admin submits an update product request with updated details (name, description, price, quantity)
   * Product service validates the request and updates the product entity
   * Product service returns a success response to the admin
3. Admin deletes a product:
   * Admin submits a delete product request
   * Product service validates the request and deletes the product entity
   * Product service returns a success response to the admin

User Management:

1. User registration: -**admin user creation will be done from back end**
   * User submits a registration request with details (username, email, password)
   * User service validates the request and creates a new user entity
2. User profile update:
   * User submits an update profile request with updated details (name, email, address)
   * User service validates the request and updates the user entity

Order Management:

1. Customer places an order:
   * Customer submits an order request with product details and quantity
   * Order service validates the request and communicates with the product service to check available quantity
   * If the requested quantity is available order is created and the quantity for that product will be reduced through Product service.
   * If the requested quantity is not available a message will be sent back to the user saying quantity is not available.
2. Order cancellation:

* Customer submits an order cancellation request
* Order service changes the status of the order to cancelled and updates the product quantity accordingly
* Order service will return a success message to the customer.

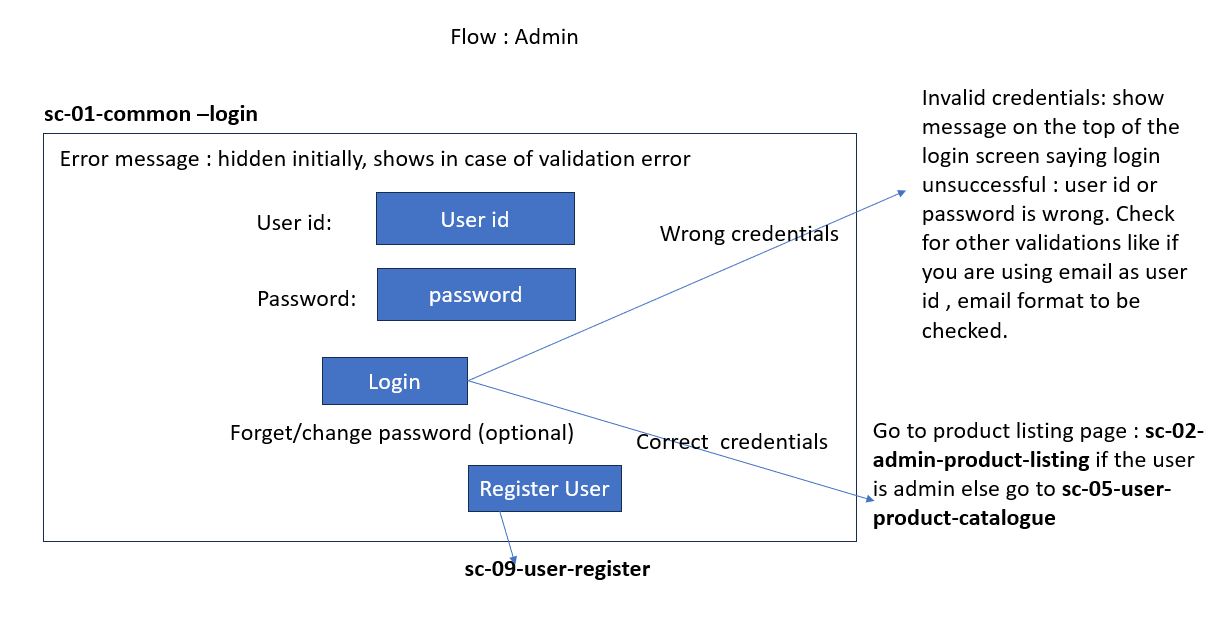
Role based access management needs to be implemented so that

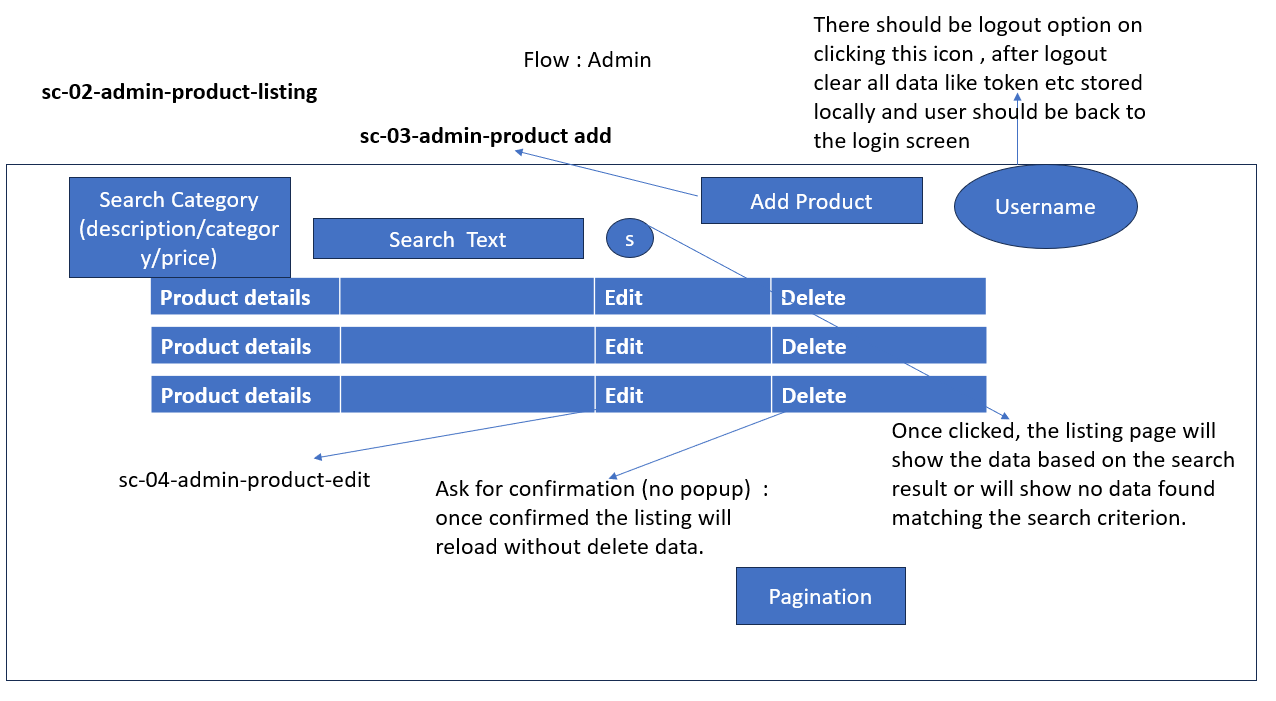
1. Only admin user can have access to the Product Management functionality.
2. Customer can only have access to the orders they have placed.

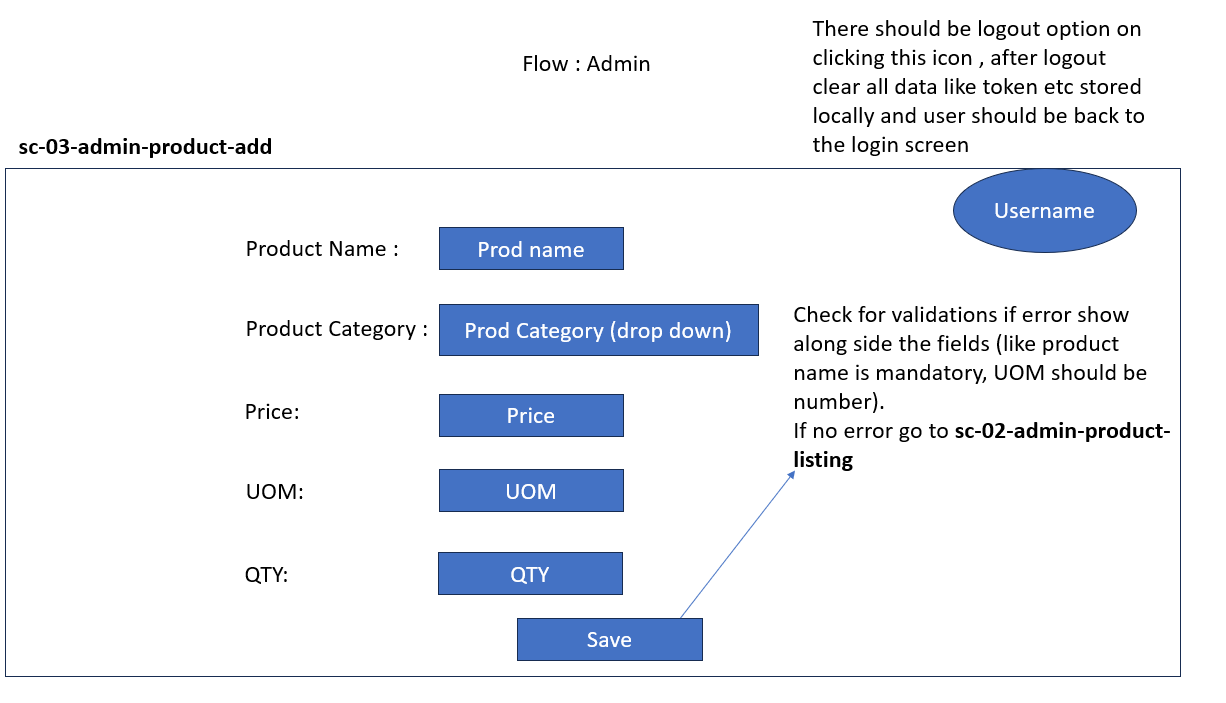
Scope of the project will be to :

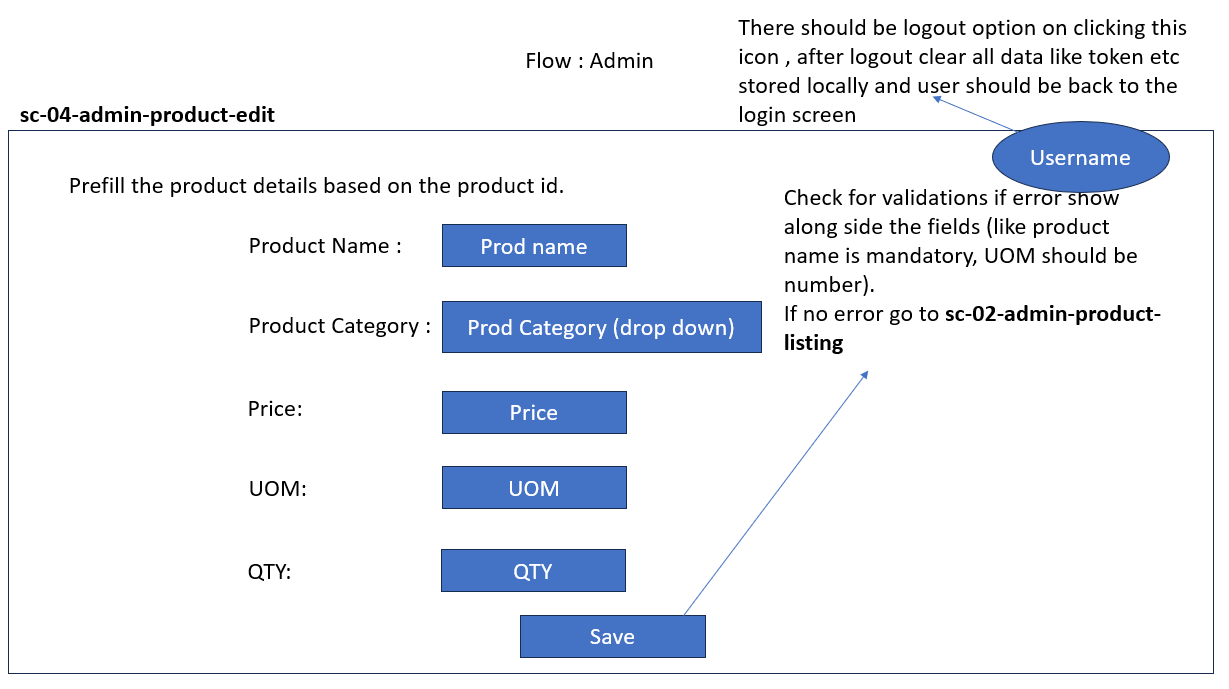
1. Analyze the requirement.
2. UI Design.
3. Database design.
4. REST endpoint design (i.e designing of REST urls, methods and input data).

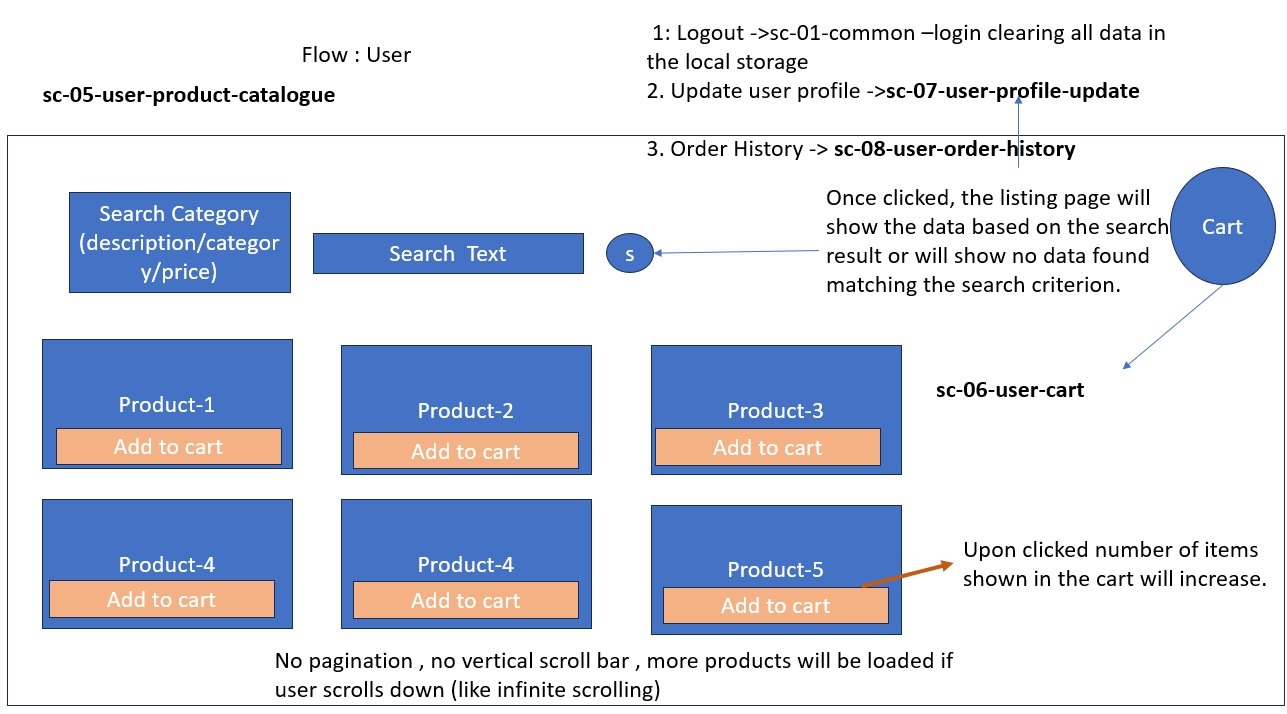
# Screen Flows

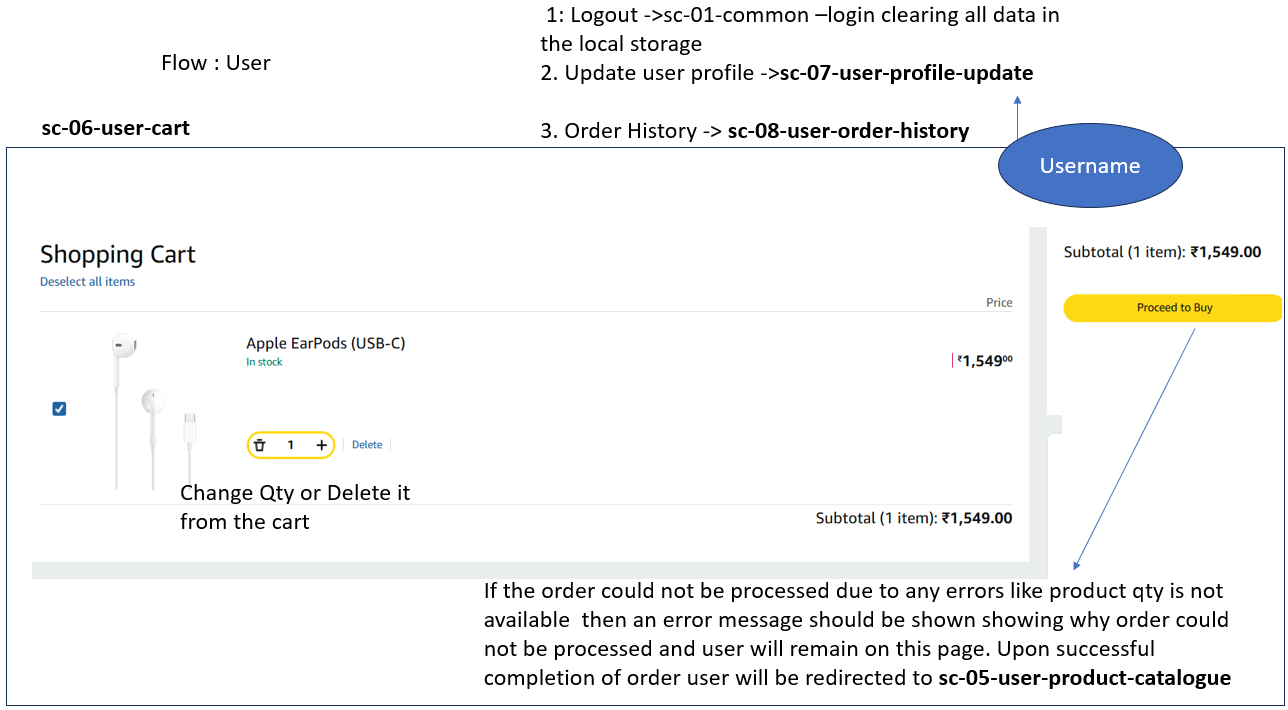


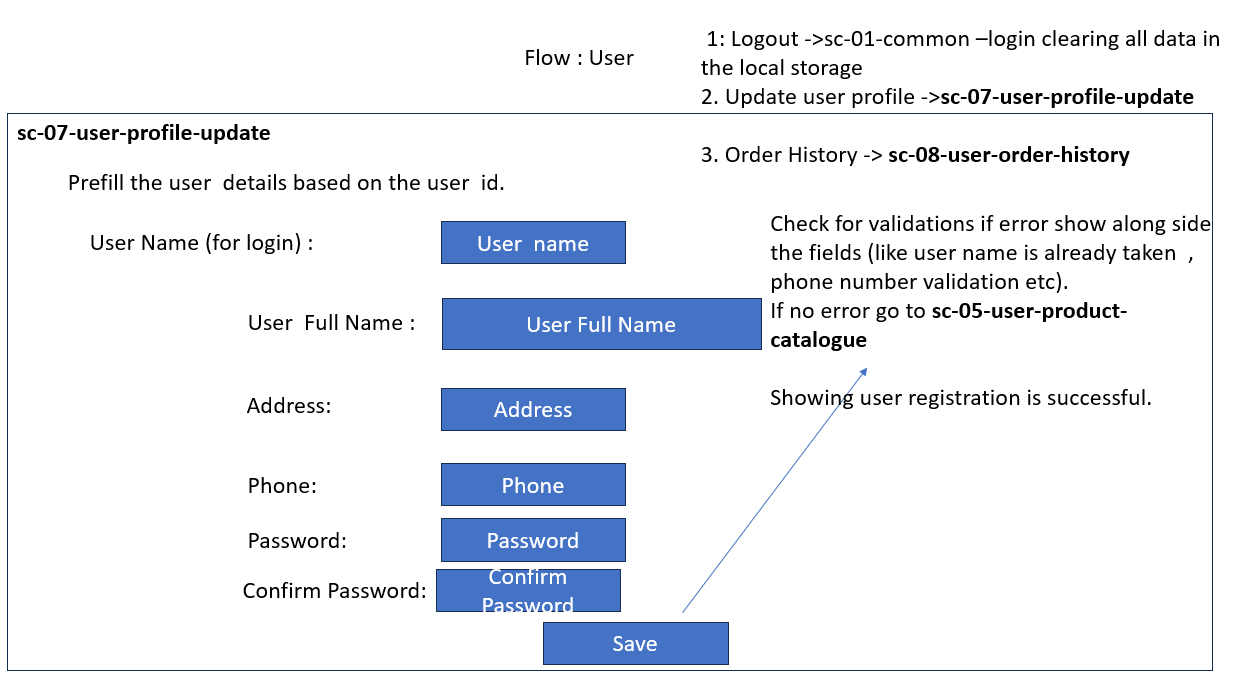


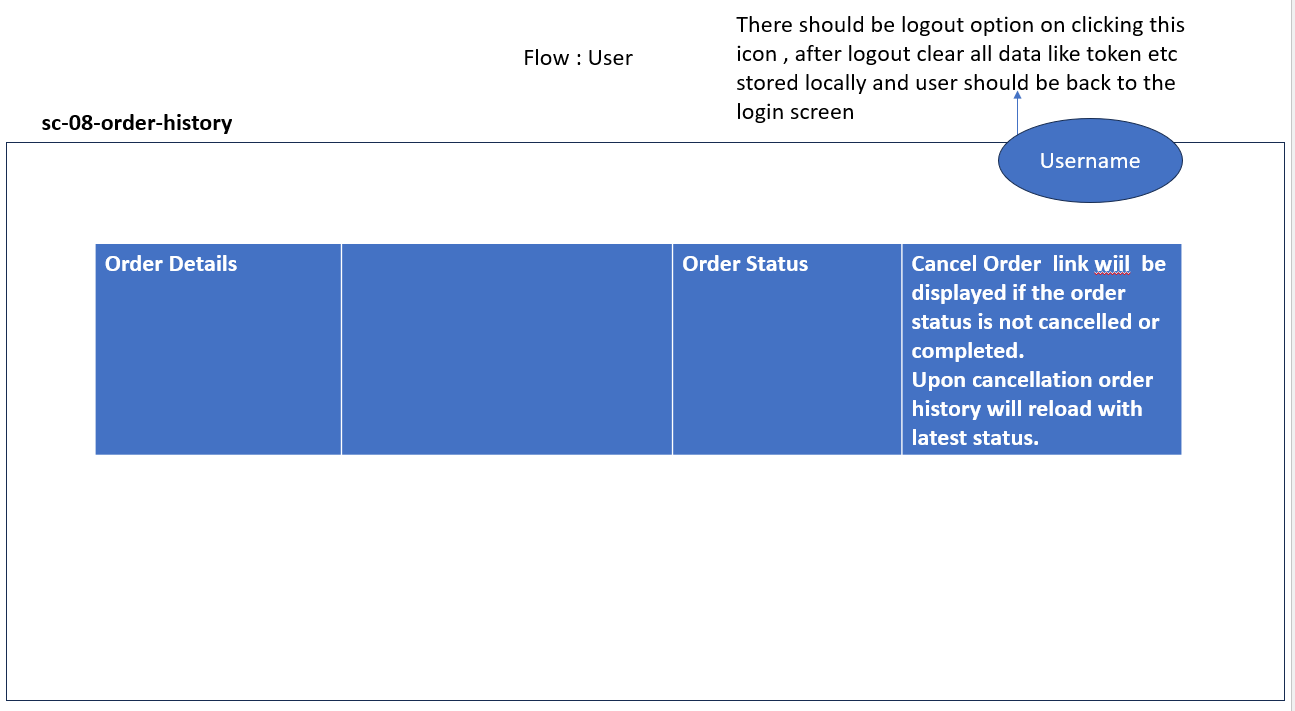


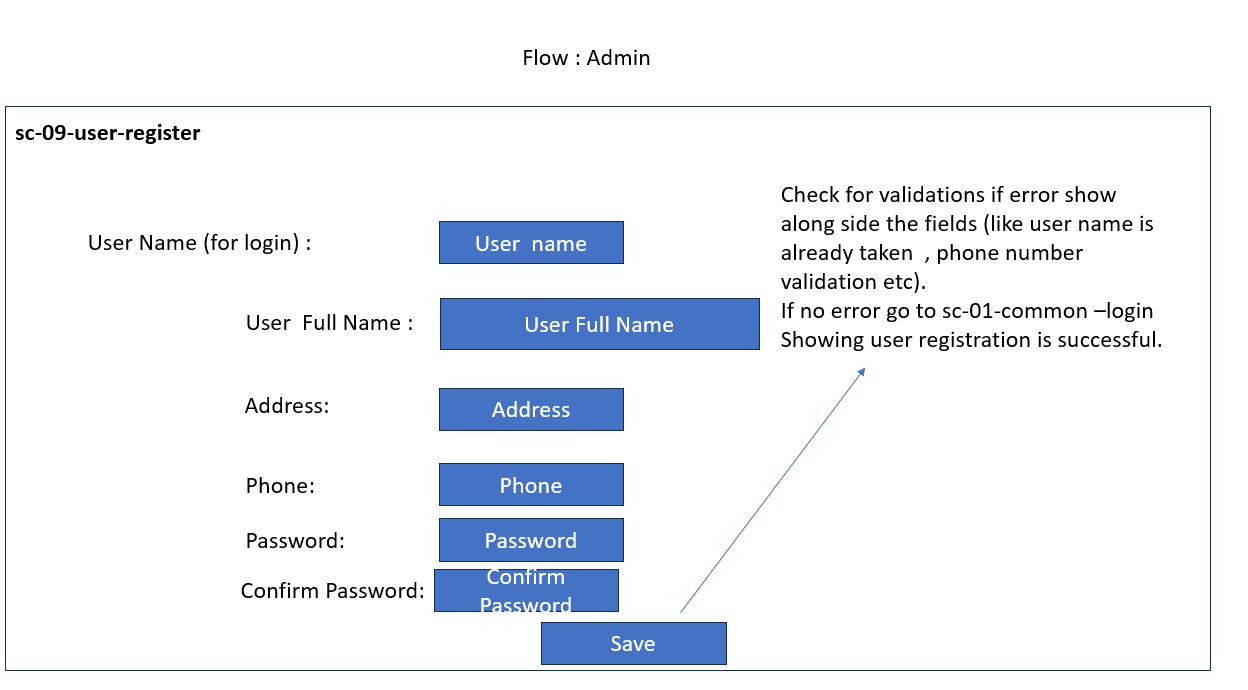












# API End Points

**User Management MS:**

1. Creation /user (POST)
2. Change user data : /user (PUT)
3. User deletion :/user (DELETE)
4. User List : /user (GET)
5. User detail :/user/{id} (GET)
6. Menu : /user/menu/{id}
7. /user/login (POST)
8. /user/logout (POST)

**Product Management MS:**

1. Creation /product (POST)
2. Change product data : / product (PUT)
3. Product deletion :/product (DELETE)
4. Product List : /product (GET)
5. Product detail :/ product /{id} (GET)

**Order Management (Customer) MS:**

There will be two modules in the order management microservice.

**Cart :**

1. Add Product to Cart : /cart/addProd (User ID/Product and Qty) ->Post
2. Delete products from the Cart (cart item id) ->/cart/deleteProd/{itemid} (Delete)
3. Change the Qty -> /cart/update (Product and Qty) -> PUT
4. View cart ->/cart/{userId} (Get)

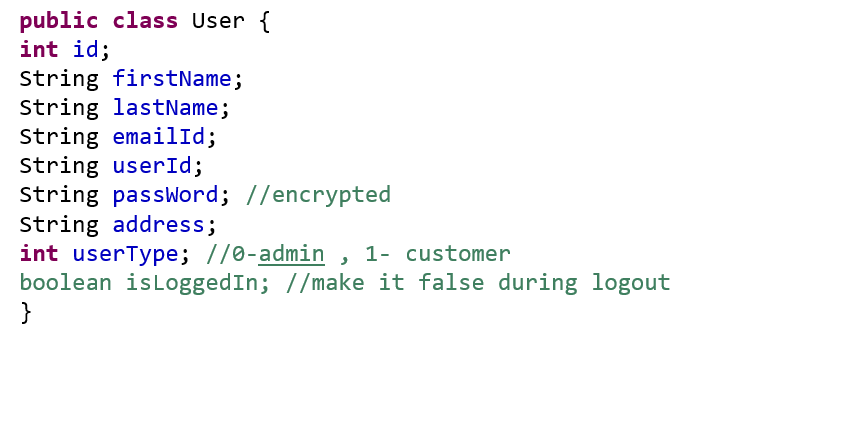
**Order:**

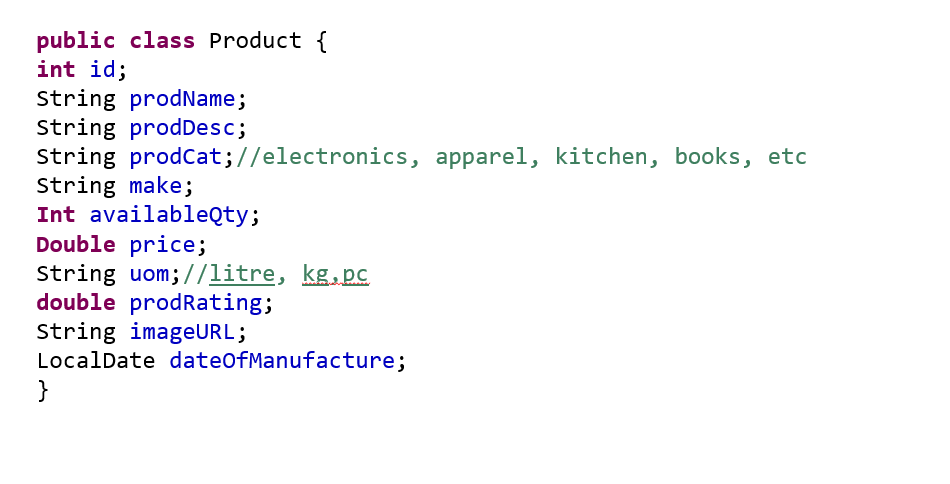
1. Create order /order ( Order Details -> get calling cart details ) – POST
2. Cancel order /order/{ordeId} –PUT update status field of the order
3. No delete order feature
4. View order /order ->Get list of all orders for all users
5. List /order/{userId} -> List of orders for a particular user ->GET
6. Order details /order/{orderId} ->GET

Note : Add additional API end points if required to cater to all requirements.

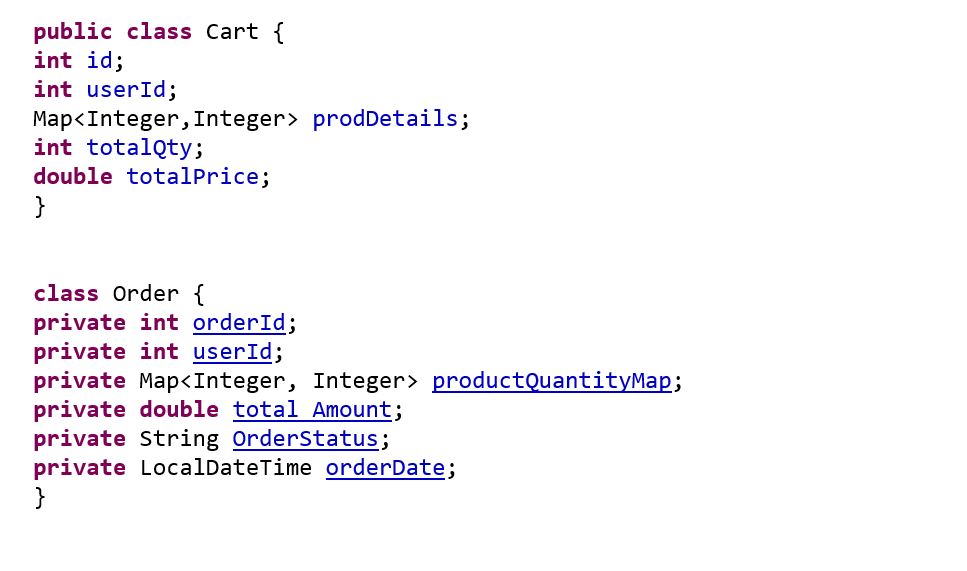
# POJO/ENTITY CLASSES

**Order MS:**



**Product MS:**

**Order MS:**



Note: Add additional classes if required to cater to all requirements.

# Technical Architecture

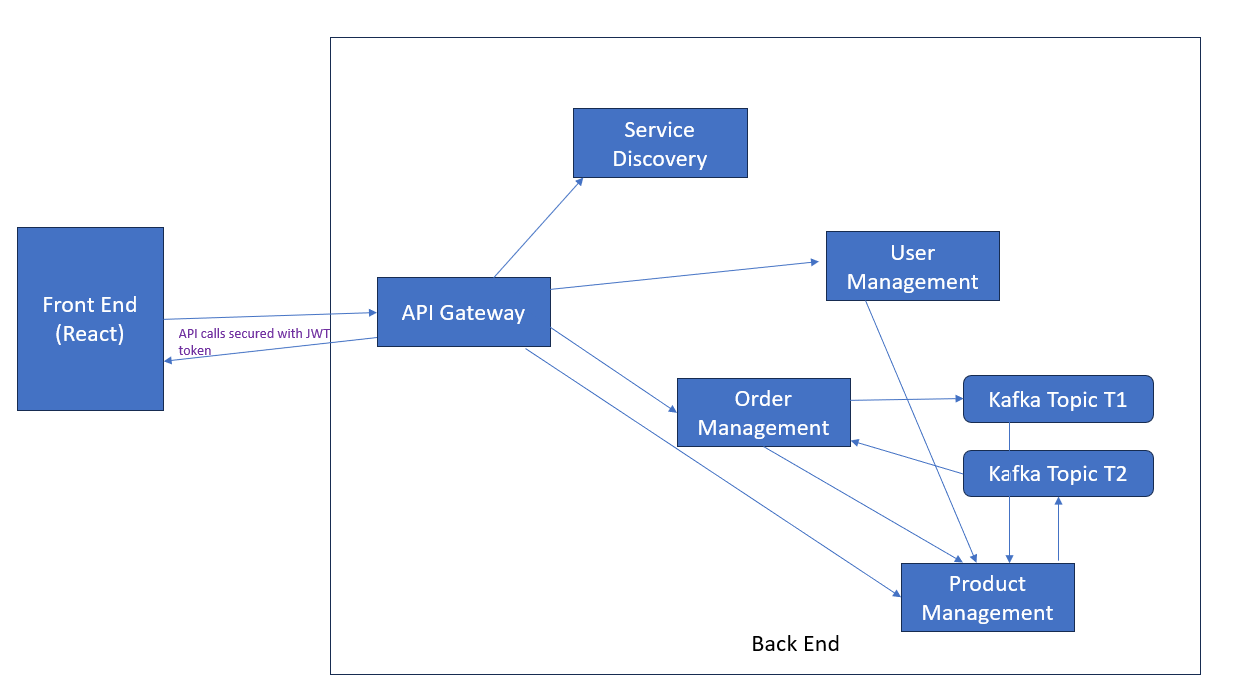


Figure Application Architecture

# Development Methodolgy

Sprint based development methodolgy will be followed as below :

**Sprint 1**: **Design Thinking and Backend Foundation :**

* Develop wireframes and user stories for the project.
* Set up a Spring Boot application.
* Create RESTful APIs for user management.
* Establish database schema and integrate it with the backend.

**Sprint 2**: **Backend Module:**

* Complete User Management API along with Role-Based Access
* Product Catalog API
* Order Management API
* Implement Kafka integration

**Sprint 3**: **Front end Module:**

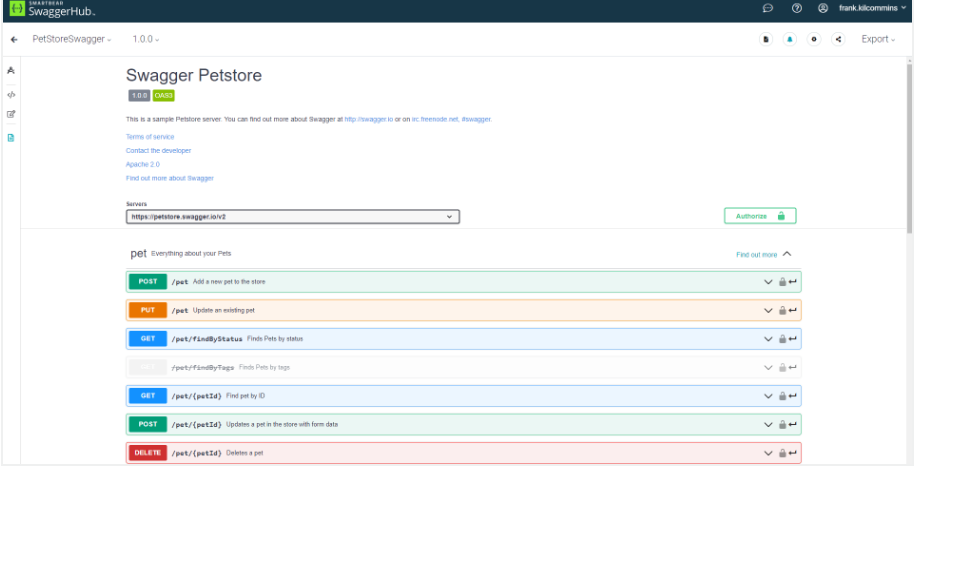
* Build dashboard and all forms for User Management Product Management and Order Management and login.

***Swagger Documentation Guideline:***

* Use @Tag , @Operation, ApiResponses and @ApiResponse annotations for swagger documentation. A sample class is provided below as reference:



The api documentation should be visible through swagger-ui as below (image is for reference only):



***UI/UX Guideline:***

***Core UX Guidelines for UI Development***

* + Consistency: Maintain consistent design elements (colors, fonts, buttons).
  + Clarity: Make UI elements clear and understandable.
  + User control: Give users control over interactions.
  + Feedback: Provide feedback for user actions (e.g., button clicks).
  + Clear messaging: Use clear, concise language for labels, buttons, errors.

***UI Design Best Practices***

* + Simple and intuitive: Minimize complexity in UI.
  + Use familiar patterns: Use common UI patterns users are familiar with.
  + Responsive design: Ensure UI works well across devices/screen size

General Guidelines:

* Use Java version 17/21.
* Use layering in Microservices.
* Exception handling to be done.
* Use proper package structure like com.wipro.ecom
* Use Swagger for API documentation (optional).
* Angular Routing should be used.
* Usage of Bootstrap is advisable.
* Proper resting to be done.
* Code should be maintained in a Git repository.

Core Technology:

* HTML/CSS/Angular/Spring Boot

Deliverables:

* Screen shots/Videos showing all operations and data in the database.
* The html , css and .js files as source code in proper folder structure.