Develop an “ONLINE PIZZA ORDERING APPLICATION” having problem statement :

To Create an Application for online Pizza Order System

Customer should be able to view the list of pizzas and can order pizza based on his/her desire by applying coupons.

Customer should be able to see the order history and can update, cancel orders.

Admin should be able to perform below operations:

1. Pizza Management

2.Order Management

3.Customer Management

4. Coupons Management

Application will be developed in following Sprints-

1.Core Java + JPA with hibernate

2.Sprint Boot +Rest Controller +JPA with Hibernate

3.React, HTML, CSS, JavaScript, Bootstrap for UI designing(Front End)

Implement the case study in incremental manner

MODULES:

1. Login, Registration Module (For both ,Customer Role and Admin Role)

User Management:

\* User Registration: Creating a new account with personal details (name, email, password),

\* User Login: Authenticating user credentials to access the application.

Profile Management: Updating personal information, managing shipping addresses, and payment

Methods

Customer View The Products, Add Product to the cart, remove product from the cart, customer can change the password, customer can buy the product

(Add to Cart: Adding selected products to the shopping cart.

Cart View: Reviewing items in the cart, updating quantities.

Cart Removal: Deleting items from the cart)

1. Pizza Ordering Module

Order Management

Order Tracking: Viewing order status and tracking shipment details.

Order History: Accessing past orders with details and re-ordering options

1. Pizza Management Module

Product Listing: Displaying available products with details like name, price, description, image

Product Search: Filtering products based on keywords, categories, brands, price range

Product Details: Viewing detailed product information, reviews, size/color options

1. Coupons Module

Apply coupons according to coupons, generate the final price of the Order to be paid by the customer.

1. System Management module :
2. (Payment Type such as Debit, UPI, Cash On Delivery,)
3. Previous Orders of What items customers purchased

**Admin**

-adminid.int

-adminUserName: String

-adminPassword:String

**Customer**

-customerld int

-customerName:String

-customerMobile long

-customerEmail String

-customerAddress:string

userName: String

-password: String

**PizzaOrder**

-bookingOrderld:int

-orderDateL:LcaleDate

-transactionMode: String

-quantity:int

-size:String

-totalCost double

-pizza Pizza

-order Order

-coupan:Coupan

**User**

-userid int

-userName:String

-password: String

**Order**

-orderld int

-order Type String

-orderDescription String

-order Customerld.int

**Pizza**

-pizzald int

pizza Type: String

-pizzaName String

-pizzaDescription String

-pizzaCost double

-pizzaCostAfter Coupan double

**Coupan**

-coupanid int

-coupanName: String

-coupan Type String

-coupanDescription: String

-coupanPizzald: int

**------------------------------------------------------------------------------**

**Service Interface:**

IUserService

+addNewUser(User user): User

+signin(User user): User

+signOut(User):User

forgot Password(String oldPassword, String newPassword):boolean

ICustomer Service

+addCustomer(Customer customer): Customer

+updateCustomer(Customer customer): Customer

+deleteCustomer(Integer customerld): Customer

+viewCustomer(): List<Customer> +viewCustomer(Integer customerid): Customer

ICouponService

+add ddCoupans (Coupan coupan): Coupan +editCoupans(Coupan coupan): Coupan +deleteCoupans(Integer coupanid):Coupan

+viewCoupans():List<Coupan>

IPizzaService

+addPizza(Pizza pizza):Pizza

+updatePizza(Pizza pizza):Pizza

+deletePizza(Integer pizzald): Pizza

+viewPizzaList():List<Pizza>

+viewPizza(Integer pizzald):Pizza

+viewPizzaList(double minCose, double maxCost):List<Pizza>

+viewPizzaList(String pizzaType):List<Pizza>

IPizza OrderService

+bookPizzaOrder(PizzaOrder order): PizzaOrder

+updatePizzaOrder(PizzaOrder order):PizzaOrder +cancelPizzaOrder(Integer bookingOrderld):PizzaOrder +viewPizzaOrder(Integer bookingOrderld):PizzaOrder

+viewOrdersList():List<PizzaOrder>

+viewOrdersList(LocaleDate date):List<PizzaOrder> +calculate Total(String size. Integer quantity):List<PizzaOrder>

Class Design(Repository):

**IUserRepository**

addNewUser(User user): User +signin(User user):User +signOut(User): User forgot Password(String oldPassword. String newPassword) boolean

**Customer Repository**

+addCustomer(Customer customer):Customer

updateCustomer(Customer custo ner customer): Customer -deleteCustomer(Integer customerld) Customer +viewCustomer() List<Customer> +viewCustomer(integer customerid) Customer

**Coupon Repository**

addCoupans(Coupan coupan): Coupan editCoupans(Coupan coupan): Coupan +deleteCoupans(integer coupanid): Coupan viewCoupans(): List<Coupan

**PizzaRepository**

+addPizza Pizza pizza): Pizza +update Pizza(Pizza pizza) Pizza +deletePizza(Integer pizzald):Pizza +viewPizzaList) List<Pizza +viewPizza(Integer pizzald) Pizza viewPizzaList(double minCose.double maxCost) List<Pizza> +viewPizzaList(String pizzaType) List<Pizza>

**Pizza OrderRepository**

bookPizza Order(PizzaOrder order): PizzaOrder updatePizzaOrder(PizzaOrder order): PizzaOrder cancelPizzaOrder(Integer booking Orderid): PizzaOrder +viewPizzaOrder(Integer booking Orderld): PizzaOrder viewOrdersList():List<PizzaOrder viewOrdersList(LocaleDate date) List<PizzaOrder calculate Total(String size integer quantity):List PizzaOrder

As we advance in our case study and implementation of Spring Boot microservices, outline the key aspects and guidelines to ensure we stay aligned and achieve our objectives efficiently:

Project Implementation Guidelines:

JWT Authentication:

Implement JWT-based authentication to secure your microservices.

Exception Handling:

Develop a comprehensive exception handling strategy to ensure that meaningful and consistent error messages are returned.

Validations:

Ensure that all necessary data validations are in place to maintain the integrity and reliability of your application.

Naming Conventions:

Adhere to standard naming conventions for clarity and consistency across your codebase.

UML Diagrams:

Create and maintain UML diagrams to visually represent the architecture and design of your microservices.

Database:

Use PostgreSQL for database management. Ensure that your microservices are properly configured to interact with PostgreSQL.

API Gateway:

Implement an API Gateway to manage and route requests between microservices effectively.

Feign Client:

Utilize Feign Client for making HTTP reqijests to other microservices, simplifying communication between then.

Additional Tasks:

Code Management:

Regularly commit and push your code to our GitHub repository. Follow the branching strategy and naming conventions we've set.

Writing Test Cases:

Write thorough test cases using JUnit and Mockito. Include unit tests for individual components and integration tests where needed.

API Testing:

Use Swagger for API documentation and Postman for testing HTTP requests. Share Postman collections if necessary to demonstrate endpoint functionality.

Coding:

**1. Login, Registration Module**

**Entities**

**User**

java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int userId;

private String userName;

private String email;

private String password;

// Getters and Setters

}

**Customer**

java

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int customerId;

private String customerName;

private long customerMobile;

private String customerEmail;

private String customerAddress;

private String userName;

private String password;

// Getters and Setters

}

**Admin**

java

@Entity

public class Admin {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int adminId;

private String adminUserName;

private String adminPassword;

// Getters and Setters

}

**Repositories**

**UserRepository**

java

public interface UserRepository extends JpaRepository<User, Integer> {

User findByEmail(String email);

}

**CustomerRepository**

java

public interface CustomerRepository extends JpaRepository<Customer, Integer> {

}

**AdminRepository**

java

public interface AdminRepository extends JpaRepository<Admin, Integer> {

}

**2. Pizza Ordering Module**

**Entities**

**PizzaOrder**

java

@Entity

public class PizzaOrder {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int bookingOrderId;

private LocalDate orderDate;

private String transactionMode;

private int quantity;

private String size;

private double totalCost;

@ManyToOne

private Pizza pizza;

@ManyToOne

private Order order;

@ManyToOne

private Coupon coupon;

// Getters and Setters

}

**Order**

java

@Entity

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int orderId;

private String orderType;

private String orderDescription;

private int customerId;

// Getters and Setters

}

**Repositories**

**PizzaOrderRepository**

java

public interface PizzaOrderRepository extends JpaRepository<PizzaOrder, Integer> {

List<PizzaOrder> findByOrderDate(LocalDate orderDate);

}

**OrderRepository**

java

public interface OrderRepository extends JpaRepository<Order, Integer> {

}

**3. Pizza Management Module**

**Entities**

**Pizza**

java

@Entity

public class Pizza {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int pizzaId;

private String pizzaType;

private String pizzaName;

private String pizzaDescription;

private double pizzaCost;

private double pizzaCostAfterCoupon;

// Getters and Setters

}

**Repositories**

**PizzaRepository**

java

public interface PizzaRepository extends JpaRepository<Pizza, Integer> {

List<Pizza> findByPizzaType(String pizzaType);

List<Pizza> findByPizzaCostBetween(double minCost, double maxCost);

}

**4. Coupons Module**

**Entities**

**Coupon**

java

@Entity

public class Coupon {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int couponId;

private String couponName;

private String couponType;

private String couponDescription;

private int couponPizzaId;

// Getters and Setters

}

**Repositories**

**CouponRepository**

java

public interface CouponRepository extends JpaRepository<Coupon, Integer> {

}

1. **System Management Module**

**Entities**

**Payment**

java

@Entity

public class Payment {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int paymentId;

private String paymentType; // E.g., Debit, UPI, Cash On Delivery

private String transactionId;

private double amount;

private LocalDate paymentDate;

@ManyToOne

private PizzaOrder pizzaOrder;

// Getters and Setters

}

**PreviousOrder**

java

@Entity

public class PreviousOrder {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int previousOrderId;

private LocalDate orderDate;

private String orderDescription;

private double totalCost;

@ManyToOne

private Customer customer;

@OneToMany(mappedBy = "previousOrder")

private List<Pizza> pizzas;

// Getters and Setters

}

**Repositories**

**PaymentRepository**

java

public interface PaymentRepository extends JpaRepository<Payment, Integer> {

}

**PreviousOrderRepository**

java

public interface PreviousOrderRepository extends JpaRepository<PreviousOrder, Integer> {

List<PreviousOrder> findByCustomerId(int customerId);

}

