Microservices Development

Designing and Implementing a set of microservices for managing customer orders within an e-commerce system.

Data Models:

1.**Customer Table**: - Fields: Customer ID, Name, Email, Address, Phone, Registration Date - Represents information about registered customers in the system.

```
Entity:
public class Customer {
   private Long customerId;
   private String customerName;
   private String customerEmail;
   private String customerAddress;
   private Long customerPhone;
   private Instant customerRegistrationDate;
}

Request Body:
public class CustomerRequest {
    private String name;
    private String email;
    private String address;
    private Long phone;
}

Response Body:
public class CustomerResponse {
    private Long customerId;
    private String customerName;
    private String customerEmail;
    private String customerAddress;
    private Long customerPhone;
    private Instant customerRegistrationDate;
}
```

2.**Vendor Table**: - Fields: Vendor ID, Name, Email, Address, Phone, Registration Date - Represents information about vendors who supply products to the platform.

```
Entity:
public class Vendor {
    private Long vendorId;
    private String vendorName;
    private String vendorEmail;
    private String vendorAddress;
    private Long vendorPhone;
    private Instant vendorRegistrationDate;
}

Request Body:
public class VendorRequest {
    private String name;
    private String email;
    private String address;
    private Long phone;
}

Response Body:
public class VendorResponse {
    private Long vendorId;
    private String vendorName;
    private String vendorEmail;
    private String vendorAddress;
    private Long vendorPhone;
    private Instant vendorRegistrationDate;
}
```

3. **Order Table**: - Fields: Order ID, Customer ID, Vendor ID, Order Date, Total Amount, Status - Contains details about customer orders, including which customer placed the order and which vendor fulfills it.

```
Entity:
public class Order {
    private Long orderId;
    private Long customerId;
    private Long vendorId;
    private Instant orderDate;
    private Long orderTotalAmount;
    private String orderStatus;

}

Request Body:
public class OrderRequest {
    private Long customerId;
    private Long vendorId;
    private Long orderTotalAmount;
}

Response Body:
public class OrderResponse {
    private Long orderId;
    private Long customerId;
    private Long vendorId;
    private Instant orderDate;
    private Long orderTotalAmount;
    private String orderStatus;
}
```

4. **Item Table**: - Fields: Item ID, Vendor ID, Name, Description, Price, Stock Quantity - Represents individual items/products available for purchase from various vendors.

```
Entity:
public class Item {
    private Long itemId;
    private Long vendorId;
    private String itemName;
    private String itemDescription;
    private Long itemPrice;
    private Long itemStockQuantity;

}

Request Body:
public class ItemRequest {
    private Long vendorId;
    private String name;
    private String description;
    private Long price;
    private Long stockQuantity;
}

Response Body:
public class ItemResponse {
    private Long itemId;
    private Long vendorId;
    private String itemName;
    private String itemDescription;
    private Long itemPrice;
    private Long itemStockQuantity;
}
```

Microservice Requirements:

1.**Customer Microservice**: - Implemented CRUD (Create, Read, Update, Delete) operations for managing customer information. - Allow customers to register, update their profile, and view their information.
2.**Vendor Microservice**: - Implemented CRUD operations for managing vendor details. - Allow vendors to register, update their profile, and view their information.

3.**Order Microservice**: - Implemented operations to create new orders, retrieve order details, and update order status.
4.**Item Microservice**: - Implemented operations to manage items available for sale. - Allow vendors to add/update/delete their products.

API End Points :

To Access Eureka Server Use Link : http://localhost:8761/

To Access Secure End Points :

1. Go to Okta Auth Link → http://localhost:9090/authenticate/login
2. Enter Username: *201514@juitsolan.in*
3. Enter Password: Vasu123@
4. Now Use `"accessToken"` to access secure end points

Customer Service:

Swagger Link For Testing API : http://localhost:8080/swagger-ui/#/

- **POST** /api/customers : Creates a new customer.

  Required Request Body :

```
{
  "address":"string",
  "email":"string",
  "name":"string",
  "phone":0
}
```

- **GET** /api/customers/{id} : Returns details of a customer by its id.

- **PUT** /api/customers/{id}: Updates an existing customer by its id.

  Required Request Body :

```
{
  "address":"string",
  "email":"string",
  "name":"string",
  "phone":0
}
```

- **DELETE** /api/customers/{id} : Deletes an existing customer by its id.

Vendor Service:

Swagger Link For Testing API : http://localhost:8082/swagger-ui/#/

- **POST** /api/vendors : Creates a new vendor.

Required Request Body :

```
{
  "address":"string",
  "email":"string",
  "name":"string",
  "phone":0
}
```

- **GET** /api/vendors/{id} : Returns details of a vendor by its id.

- **PUT** /api/vendors/{id} : Updates an existing vendor by its id.

  Required Request Body :

```
{
  "address":"string",
  "email":"string",
  "name":"string",
  "phone":0
}
```

- **DELETE** /api/vendors/{id} : Deletes an existing vendor by its id.

Order Service:

- **POST** /api/orders : Creates a new order. This is a secure End Point and can only be accessed by admin authority.

  Required Request Body :

```
{
    "customerId": 0,
    "vendorId": 0,
    "orderTotalAmount": 0
}
```

- **GET** /api/orders/{id} : Returns details of an order by its id.

- **GET** /api/orders/customers/{id} : Returns details of an order placed by a specific customer by its id. This is a secure End Point and can only be accessed by admin authority.

- **PUT** /api/orders/vendors/{id} : Updates status of an order by vendor id. This is a secure End Point and can only be accessed by admin authority.

Item Service:

- **POST** /api/items : Creates a new item. This is a secure End Point and can only be accessed by admin authority.

  Required Request Body :

```
{
    "vendorId": 0,
    "name": "string",
    "description": "string",,
    "price": 0,
    "stockQuantity": 0
}
```

- G**ET** /api/items : Returns a list of items with same name. Takes itemName as an input parameter.

- G**ET** /api/items /{id} : Returns details of an item by its id.

- G**ET** /api/items/vendors/{id} : Returns a list of items sold by a vendor. Takes vendorId as an input parameter. This is a secure End Point and can only be accessed by admin authority.

- **PUT** /api/items/{id} : Updates an existing item by its id. This is a secure End Point and can only be accessed by admin authority.

  Required Request Body :

  ```
  {
      "vendorId": 5,
      "name": "string",
      "description": "string",
      "stockQuantity": 5
  }
  ```

- **DELETE** /api/items/{id} : Deletes an existing item by its id.

Tech Stack :

Java, Spring / Spring Boot, JPA, Hibernate, MySQL, Spring Security, Okta Security, JWT, Maven, JUnit, Spring Cloud, Eureka, Spring Cloud Config Server, Spring Web.

How To Run Application :

- Open The Project on your compiler.
- Configure application.yml files of the services:
  - ItemService
  - VendorService
  - CustomerService
  - OrderService
- Build the application.
- First run service-registry service it is eureka server.
- Then run ConfigServer service it has common configurations for services, To run this make sure you are connected to internet.
- Now you can run CloudGateway service it is API gateway for internal services. It runs on `http://localhost:9090`
- You are ready to run internal services.
