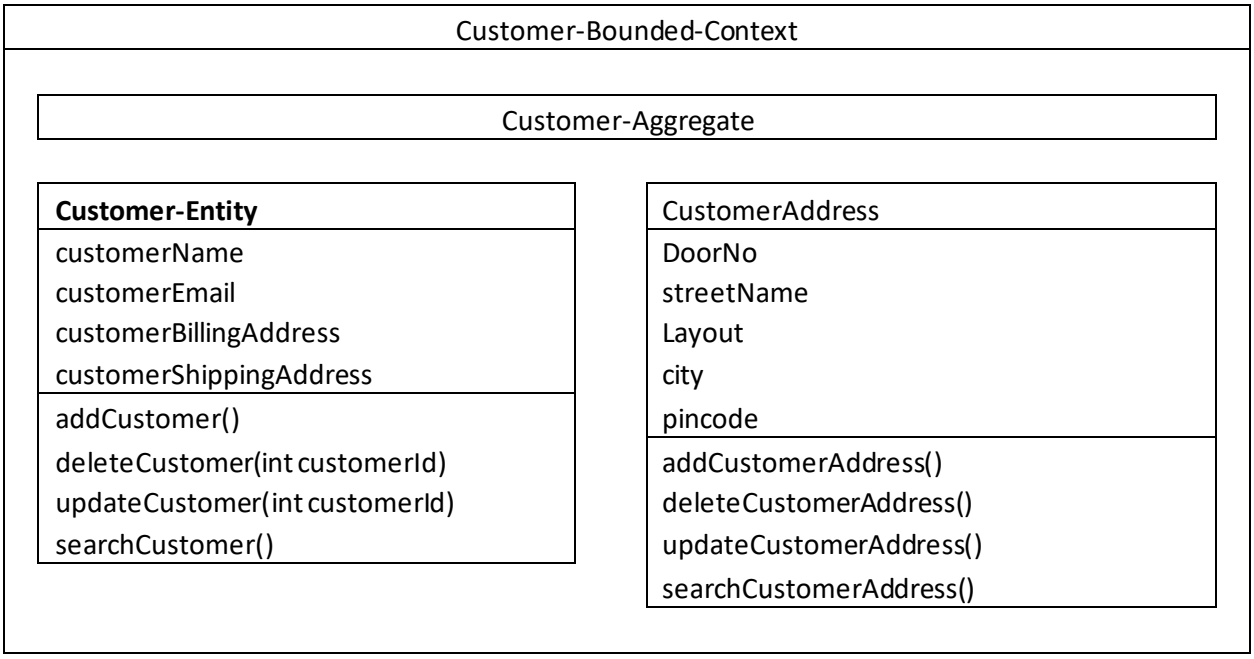
Online-Retail-Store

A popular online retail store wish to automate their business and decided to follow Domain driven design approach which involves colloboration with Business & Development teams. As part of the design, various subdomains are identified and "OrderService" is classified as core subdomain as this is their business differentiator & earns them the revenue.In this subdomain they have identified various bounded contexts as mentioned below:

1) Customer

Customer is the user of this subdomain. In order to seggregate the responsibilities of Customer a separate bounded context is maintained that captures information about the customer. The Domain model of the customer is as depicted below:

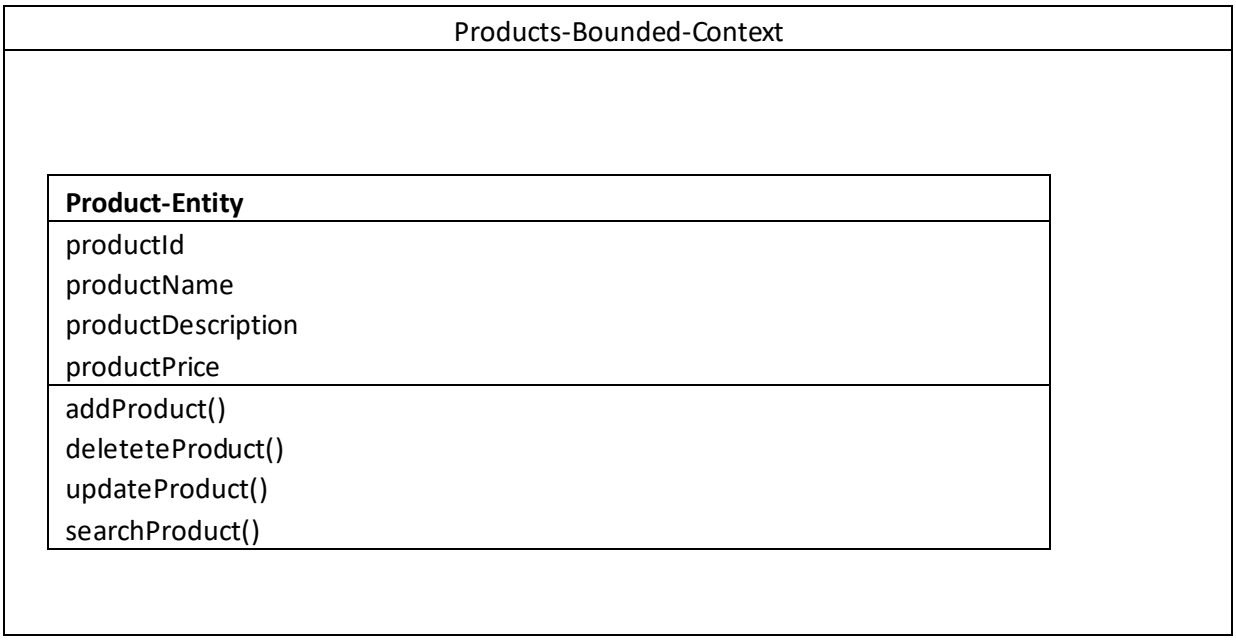


API description:

HttpMethod	MethodName	Input parameter	Description	Response
POST	/api/addCustomer()	Customer details	Creates a new customer with billing & shipping details. Customer id to be autogenerated	201
DELETE	/api/deleteteCustomer()	customerId	Deletes the Customer with the given id	200
PUT	/api/updateCustomer()	customerid & new customer detals in POST request	Updates the customer with given customer id with new details	200
GET	/api/searchCustomer()	customerId	Returns the customer details for the customerid.	200

2) Products-Catalogue

The attributes of the products which helps Customer to search & purchase the product is maintained as part of "Products Catalogue" bounded context. A customer would search for products in Products-Catalogue. The Domain model for this bounded context is as shown below:

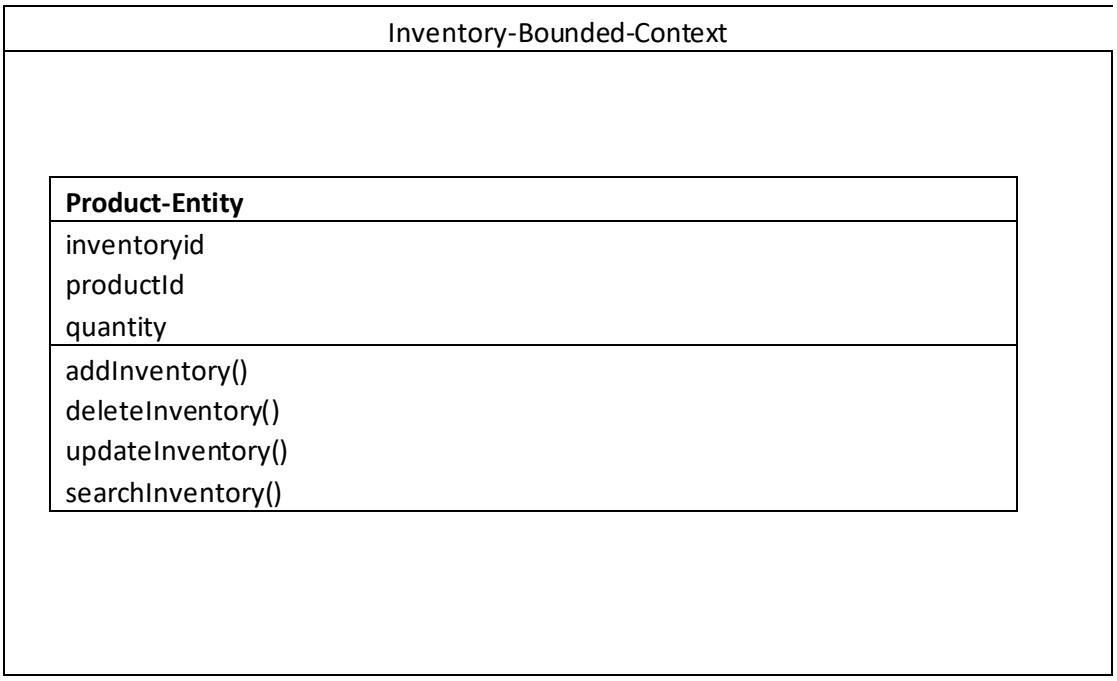


API Description:

HttpMethod	API	Input parameter	Description	Response
POST	/api/products/	Input : Product details. This api should be mapped to addProduct method	Creates a new Product and product id should be auto generated.	201
DELETE	/api/products/{id}	Input: productid . This should be mapped to deleteProduct() method.	Deletes the Product with the given id	200
PUT	/api/products/{id}	Input: productid and new productcetails This should be mapped to updateProduct() method.	Updates the product with given product id with new details	200
GET	/api/products/{id}	Input: productid . This should be mapped to getProduct() method.	Returns the product details for the productid	200

3) Inventory

This Microservice maintains the inventory information of the product which needs frequent updation. Here we would maintain only quantity corresponding to the product . Domain model is as represented below:

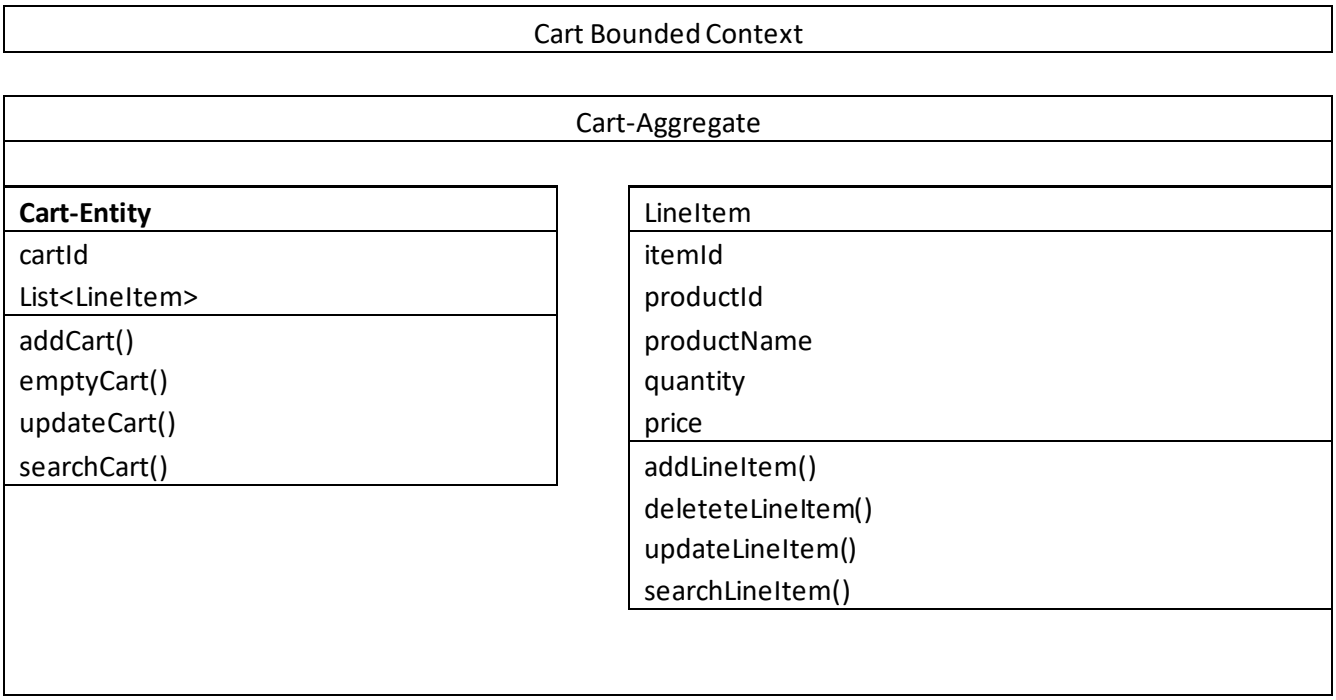


API Specification:

HttpMethod	API	Input parameter	Description	Response
POST	/api/inventory	Input : Product details(product id & quantity) This api should be mapped to addInventorymethod	Creates a new inventory (product id & quantity) and inventory id should be auto generated.	201
DELETE	/api/inventory/{id}	Input: productid . This should be mapped to deleteInventory() method.	Deletes the Inventory with the given id	200
PUT	/api/inventory/{id}	Input: productid and new productdetails This should be mapped to updateInventory() method.	Updates the Inventory with new details	200
GET	/api/inventory/{id}	Input: productid . This should be mapped to getInventory() method.	Returns the Inventory details for the productid	200

4) Cart

This Microservice represents the Shopping cart of the customer. This contains set of products customer has selected to purchase. Please note there should be only one cart for the customer and this cart should be made empty upon placing order for the products in the cart.
Domain Model for the cart is as shown below:



Please note we should maintain One-to-Many relationship between Cart & Line Item.

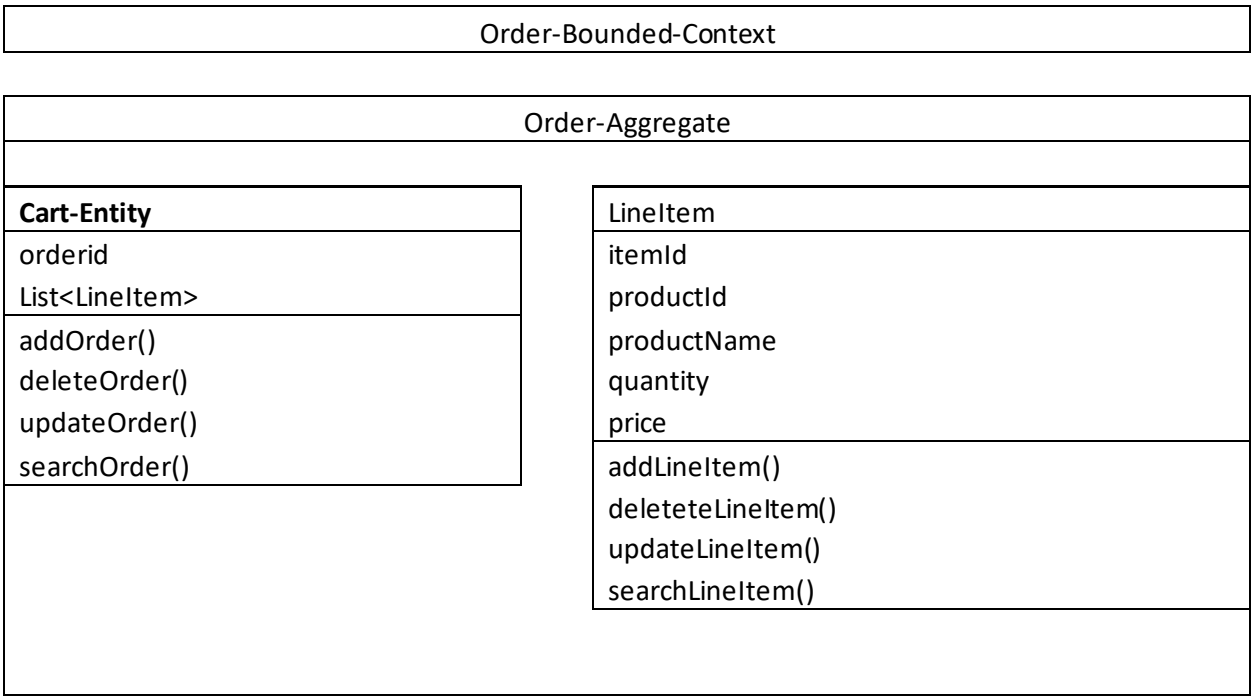
API Specification:

HttpMethod	API	Input parameter	Description	Response
POST	/api/cart	Input : Cart Details that can contain one or more line items This api should be mapped to addCart method	Creates a new cart for the customer with one or more products Cart id to be autogenerated	201
DELETE	/api/cart/{id}	Input: cartid. This should be mapped to emptyCart() method.	Deletes the cart with the given id	200
PUT	/api/cart/{id}	Input: cartid and cart details is passed as Request body. This should be mapped to updateCart() method.	Updates the cart with new cart details for a given cartid.	200
GET	/api/cart/{id}	Input: cartId . This should be mapped to searchCart() method.	Returns the cart details for the cartid.	200

5) Order

Customer can place order for all the items selected in the cart and Order Micro service is responsible for this functionality. Customer can place an order, view all his orders using this microservice.

The Domain model is as shown below.



API Specification:

HttpMethod	API	Input parameter	Description	Response
POST	/api/order	Input : order Details that can contain one or more line items This api should be mapped to addorder method	Creates a new order for the customer with one or more products order id to be autogenerated	201
DELETE	/api/order/{id}	Input: orderid. This should be mapped to deleteorder() method.	Deletes the order with the given id	200
PUT	/api/order/{id}	Input: orderid and order details is passed as Request body. This should be mapped to updateorder() method.	Updates the order with new order details for a given orderid.	200
GET	/api/order/{id}	Input: orderId . This should be mapped to searchorder() method.	Returns the cart details for the cartid.	200

6) API Gateway and Composite Microservice

Above mentioned 5 business microservices should be abstracted from External world by API Gateway. In this case we would follow a variant of Database Aggregation pattern. Content Aggregation would be done by a composite microservice (ShippingService)

API Gateway exposes below APIs to the external world.

i) POST /api/shoppingservice/products

This API is used by external client to create a product

- API gateway would route the request to Composite Microservice

- Composite Microservice would invoke Product Microservice to create a product.

- Composite Microservice would invoke Inventory Micro Service with productid generated from previous call & updates the inventory information

Sample Request:

```
{
 "productName": "Samsung Galaxy",
 "productDescription": "Samsung Mobile",
 "productPrice": "10000",
 "quantity": "5"
}
```

ii) **POST** /api/shoppingservice/customer

This API is used by external client to create a customer.

- API gateway would route the request to Composite Microservice

- Composite Microservice would invoke Customer Microservice to create a customer.

- Composite Microservice would invoke Cart Service to create an empty cart for new customer.

- Composite Microservice would receive customerId & cartId from above calls and mapping between Customer & Cart is maintained in Customer-Cart table

customerId	cartid
1001	102
1002	103

In the above example, customer with customer id : 1001 has a cartid : 102. Similarly Customer with customer id :1002 has a cartid :103.

Sample Request:


```
{
 "customerName": "Suresh",
 "customerEmail": "suresh.kumar@wipro.com",
 "customerBillingAddress": [
  {
   "doorNo": "312",
   "streetName": "Raman Street",
   "layout": "Gandhi Nagar",
   "city": "Bangalore",
   "pincode": "560004"
  }
 ],
 "customerShippingAddress": [
  {
   "doorNo": "312",
   "streetName": "Raman Street",
   "layout": "Gandhi Nagar",
   "city": "Bangalore",
   "pincode": "560004"
  }
 ]
}
```


iii) **PUT /api/shoppingservice/customer/{customerId}/cart**

This API is used by external client to add products to the cart. This can be invoked multiple times to add different products to the cart.

- API gateway would route the request to Composite Microservice

- Composite Microservice would check "Customer-Cart" table to check if the customer with the customerId passed in the request exists & and fetches the cartid for the customer.

- Composite Microservice would invoke Cart Service to add the products to the cart.

Sample Request:


```
{
 "LineItems": [
  {
   "productId": "1001",
   "productName": "Samsung Galaxy",
   "quantity": "1"
  },
  {
   "productId": "1002",
   "productName": "LG Refrigerator",
```

```
    "quantity": "1"
  }
 ]
}
```

iv) **POST /api/shoppingservice/customer/{customerId}/order**

This API is used by external client to place order.

- API gateway would route the request to Composite Microservice

- Composite Microservice would check "Customer-Cart" table to check if the customer with the customerId passed in the request exists & and fetches the cartid for the customer.

- Composite Microservice would invoke order microservice and pass all the line items present in the cart.

- Composite microservice to invoke API to empty Cart once order placement is successful so that no duplicate order could be created.

- Composite microservice would invoke inventory microservice to update the inventory of products for which order is placed.

- Composite Microservice would maintain mapping between CustomerId & OrderId generated from placing the order in Customer-Order table.

Sample data is as shown below:

customerId	orderId
1001	5001
1001	5002

Here Customer 1001 has 2 orders: 5001,5002

Sample Request:

No Request body is sent. Composite microservice should get the products from Cart.

v) **GET /api/shoppingservice/customer/{customerId}/orders**

This API is used by external client to view all orders placed by customer.

- API gateway would route the request to Composite Microservice

- Composite Microservice would check “Customer-Order” table to check if the customer with the customerId passed in the request exists & and fetches the orderIds for the customer.

- Composite microservice would invoke Customer Service to get the customer details for the customer Id passed in the request.It then invokes Order Service to get all the orders for the customer. Composite Microservice would aggregate both the responses and send the final response to the client.

In addition to the Microservices explained, Discoveryserver, Configuration server & Hystrix should also be implemented & demonstrated.