Inhaltsverzeichnis

1 Design UML Diagrams for an E-Commerce Web Application

Objectives:
This practical exercise involves designing a system architecture for a distributed E-Commerce Web Application.
Your task is to create various UML diagrams, including:

- Use Case Diagram
- Class Diagram
- Object Diagram
- Component Diagram
- An overall MVC architecture
- etc.

The E-Commerce Application should include the following modules:

- Seller Module: Allows sellers to create products with details such as name, descrip tion, price, and image. Supports product variants (color, size, price).
- Customer Module: Enables customers to shop by adding items to a shopping cart.
- Invoice Module: Generates invoices containing information like invoice ID, date, customer name, list of items, and final price.
- Manager Module: For managing user and providing access to various services of the system.
- Cloud Module: For managing cloud infrastructures (storage, computing power) for easy deployment.
- Security Components: Ensures authentication and authorization.

Additionally, the application should follow an MVC architecture, incorporating fron tend, backend, service layers, databases, and all necessary systems and services to develop a fully functioning distributed system deployable on the cloud.

1. Use Case Diagram to design using Draw.io

   Link to Draw.io

   Requirements:

   - Identify all the classes required for the system.
   - Define the relationships between classes (association, inheritance, aggregation, composition).
   - Specify attributes and methods for each class.

   Design the Class Diagram for the following Tasks:

- Product class with attributes: productId, name, description, price, image. • ProductVariant class with attributes: variantId, color, size, price.
- Seller class with attributes: sellerId, name, email.
- Customer class with attributes: customerId, name, email.
- ShoppingCart class with attributes: cartId, customerId, items.
- Invoice class with attributes: invoiceId, date, customerId, items, finalPrice.

Pseuod Code how this may be implemented using Java. This code helps you identify what methods (actions) may be required for your app. But you must design the class diagram in Draw.io only. No coding is required!

```
c l a s s Product {
    i n t p roduc t Id ;
    S t ri n g name ;
    S t ri n g d e s c r i p t i o n ;
    f l o a t p r i c e ;
    S t ri n g image ;

    void c r ea t eP ro d u c t ( ) ;
    void updateProduct ( ) ;
    void d el e t eP ro d u c t ( ) ;
}

c l a s s P roduc tVa rian t {
    i n t v a ri a n t Id ;
    S t ri n g c o l o r ;
    S t ri n g s i z e ;
    f l o a t p r i c e ;
}

c l a s s S e l l e r {
    i n t s e l l e r Id ;
    S t ri n g name ;
    S t ri n g email ;

    void addProduct ( Product p roduc t ) ;
    void updateProduct ( Product p roduc t ) ;
    void d el e t eP ro d u c t ( Product p roduc t ) ;
}

c l a s s Customer {
    i n t cu s tome r Id ;
    S t ri n g name ;
    S t ri n g email ;

    void addToCart ( P roduc tVa rian t v a ri a n t ) ;
    void checkou t ( ) ;
```

```
        InvoiceviewInvoice(intinvoiceId);
    }
```



```
    class ShoppingCart {
        int cartId;
        int customerId;
        List <ProductVariant> items;

        void addItem(ProductVariant v a r i a n t);
        void removeItem(ProductVariant v a r i a n t);
        void checkout();
    }

    class Invoice{
        int invoiceId;
        Date date;
        int customerId;
        List <ProductVariant> items;
        float finalPrice;

        void generateInvoice();
        void viewInvoice();
    }
```