Université Abdelmalek Essaadi
Faculté ses Sciences et techniques de Tanger
Département Génie Informatique
Cycle Master : SIM
Applications distribuées
Pr . ELAACHAk LOTFI

Application web basée sur JSF et JPA

Réaliser Par : ELHoumaini Karim / P139481476

Summary

This document provides a comprehensive overview of an E-Commerce Application based on JSF (JavaServer Faces) and JPA (Java Persistence API). The report covers various aspects of the application, including its introduction, modelization, entities, beans, and entity diagram.

The introduction section outlines the objective of the project, which is to create a functional e-commerce platform for online product management, user interactions, and order processing. It highlights the importance of understanding the project's structure and features.

The modelization section focuses on the class diagram, which defines the essential entities, attributes, and associations within the E-Commerce Application. It describes the relationships between classes such as Product, User, ShoppingCart, CartItem, Order, OrderItem, Category, and Payment.

The entities section delves into the definition and purpose of each entity in the application. It provides detailed explanations of entities such as Product, User, ShoppingCart, CartItem, Order, OrderItem, Category, and Payment. Each entity's attributes, relationships, and significance within the application are discussed.

The entity diagram visually represents the interconnections between entities in the E-commerce application. It offers a clear overview of how entities are related and how they contribute to the application's functionality.

The beans section focuses on the definition and purpose of managed bean classes in the application. It explains the role of beans such as ProductBean, ShoppingCartBean, and CategoryBean in managing specific functionality within the application. The purpose of each bean and its interaction with the user interface and database are discussed.

In conclusion, this report provides a comprehensive understanding of the design and implementation of an E-Commerce Application based on JSF and JPA. It covers the modelization, entities, beans, and entity diagram, offering insights into the project's structure and features.

Introduction:

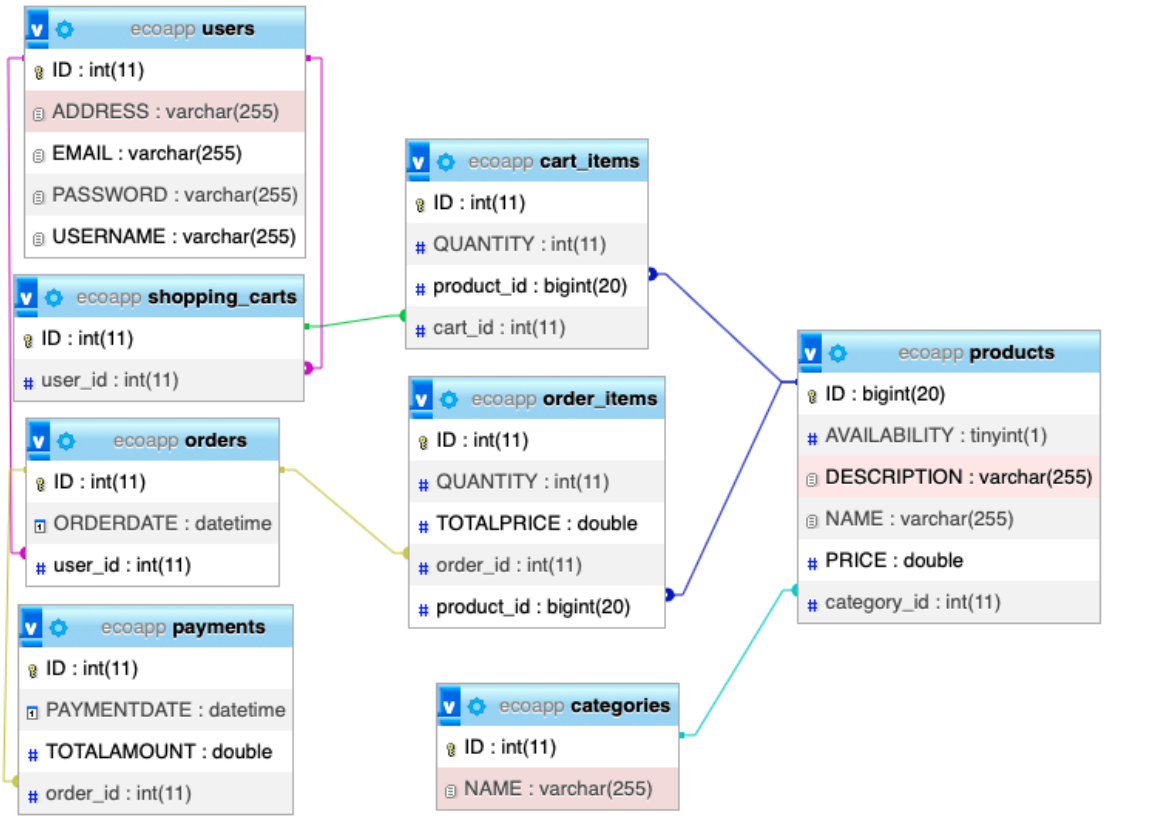
The report below provides a detailed overview of the design and implementation of an E-Commerce Application. The objective of this project was to create a functional e-commerce platform that enables online product management, user interactions, and order processing. The report covers the modeling, entity classes, bean classes, and the JavaServer Faces (JSF) view, offering a thorough understanding of the project's structure and features.

Section 1: Modelization

Class Diagram:

The modelization phase of the project involved the creation of a class diagram that defined the essential entities, their attributes, and associations within the E-Commerce Application. The key classes and associations in the class diagram include:

1. **Product:** Represents products available in the e-commerce platform. Each product has attributes such as name, description, price, category, and availability.
2. **User:** Represents users who interact with the application. User attributes include username, password, email, and address.
3. **ShoppingCart:** Represents a user's shopping cart, containing items to be purchased.
4. **CartItem:** Represents an item within a shopping cart and includes the associated product and quantity.
5. **Order:** Represents an order placed by a user, including the associated user, items, and order date.
6. **OrderItem:** Represents an item within an order and includes product details, quantity, and total price.
7. **Category:** Represents product categories to which products can belong.
8. **Payment:** Represents a payment made for an order, including details like payment date, total amount, and payment method.



Product and Category:

- **Product** and **Category** have a one-to-many relationship, where one **Category** can contain multiple **Products**. This relationship is established through the "category" attribute in the **Product** class, which refers to the **Category** to which the product belongs. Each product can be associated with a single category, but each category can have multiple products.

User and ShoppingCart:

- **User** and **ShoppingCart** are associated in a one-to-many relationship. Each user can have multiple shopping carts(holiday shopping cart, weekend shopping carts…), but each shopping cart belongs to a single user. This relationship is modeled through the "user" attribute in the **ShoppingCart** class, referencing the **User** who owns the cart

User and Order:

- **User** and **Order** are connected in a one-to-many relationship. Each user can place multiple orders, but each order is associated with a single user. This relationship is expressed through the "user" attribute in the **Order** class, which points to the **User** who placed the order.

Product and CartItem:

- **Product** and **CartItem** have a one-to-many relationship, where one **Product** can be part of multiple **CartItems**, but each **CartItem** references a single product. The "product" attribute in the **CartItem** class is used to establish this relationship.

Product and OrderItem:

- **Product** and **OrderItem** are associated in a one-to-many relationship, similar to the relationship between **Product** and **CartItem**. One **Product** can be part of multiple **OrderItems**, while each **OrderItem** corresponds to a single product. The "product" attribute in the **OrderItem** class signifies this connection.

ShoppingCart and CartItem:

- **ShoppingCart** and **CartItem** are associated in a one-to-many relationship. Each shopping cart can contain multiple cart items (e.g., various products added to the cart), while each cart item belongs to a single shopping cart. This relationship is modeled through the "items" attribute in the **ShoppingCart** class, which holds a collection of **CartItem** objects, representing the products added to the shopping cart.

Order and OrderItem:

- **Order** and **OrderItem** are linked in a one-to-many relationship. Each order can contain multiple order items, and each order item is associated with a specific order. This relationship is defined through the "items" attribute in the **Order** class, representing the list of **OrderItems** in an order.

Order and Payment:

- **Order** and **Payment** share a one-to-one relationship. Each order is connected to a single payment, and each payment corresponds to a specific order. The "order" attribute in the **Payment** class establishes this one-to-one relationship, referring to the order for which the payment was made.

Section 2: Entities in the E-commerce Application

2.1 Definition of Entities

In the context of Java Persistence API (JPA), entities represent the persistent data objects that are stored in a database. An entity is a Java class that is annotated with the `@Entity` annotation, indicating that it corresponds to a table in the database. Each instance of an entity class represents a row in the table.

Entities in JPA are used to model the data and define the structure of the database tables. They encapsulate the state and behavior of the data, allowing developers to perform CRUD (Create, Read, Update, Delete) operations on the database using object-oriented programming techniques.

Entities are typically mapped to database tables, with each attribute of the entity class corresponding to a column in the table. Relationships between entities are established using annotations such as `@OneToOne`, `@OneToMany`, and `@ManyToOne`, which define the associations between the tables.

JPA provides a set of APIs and annotations that enable developers to interact with the database using entities. These APIs include the EntityManager, which is responsible for managing entity instances and performing database operations. Entities can be persisted, retrieved, updated, and deleted using the EntityManager API.

By using entities in JPA, developers can seamlessly integrate object-oriented programming with relational databases, allowing for efficient and flexible data management in Java applications.

2.2 Purpose of Each Entity

2.2.1 Product Entity

- **Purpose:** The `Product` entity represents the core element of our E-commerce application, allowing us to store information about individual products available for sale.
- **Attributes:** The entity includes attributes such as `id`, which serves as a unique identifier for each product, `name` for the product's name, `description` to provide a brief overview of the product, `price` for the product's price, and `category` to categorize the product within specific product categories.
- **Relationships:** Each product is associated with a `Category` entity to categorize it correctly.

2.2.2 User Entity

- **Purpose:** The `User` entity plays a pivotal role in managing user-related aspects of the application, allowing users to register, log in, and maintain their profiles.
- **Attributes:** Key attributes of the `User` entity include `id` for user identification, `username` for unique user names, `password` for secure authentication, `email` for user contact, and `address` for specifying the user's location.
- **Relationships:** Users can have one or more shopping carts through a one-to-many relationship with the `ShoppingCart` entity.

2.2.3 ShoppingCart Entity

- **Purpose:** The `ShoppingCart` entity represents a user's shopping cart, where they can add products they intend to purchase. It ensures a convenient shopping experience.
- **Attributes:** The `ShoppingCart` entity includes attributes like `id` for cart identification and `user` to link the cart to a specific user.
- **Relationships:** It is related to `CartItem` through a one-to-many relationship.

2.2.4 CartItem Entity

- **Purpose:** The `CartItem` entity complements the `ShoppingCart` by representing individual items added to the cart. It allows users to specify quantities of products to buy.
- **Attributes:** Attributes of the `CartItem` entity comprise `id` for item identification, `product` for associating the item with a specific product, and `quantity` for specifying the quantity of the product in the cart.
- **Relationships:** The `CartItem` entity is connected to the `Product` entity through a one-to-many relationship.

2.2.5 Order Entity

- **Purpose:** The `Order` entity is fundamental for tracking and managing customer orders. It represents an order placed by a user, which includes the list of products they intend to purchase.
- **Attributes:** Key attributes of the `Order` entity are `id` for order identification, `user` for linking the order to a specific user, and `orderDate` for recording the date when the order was placed.
- **Relationships:** An order is associated with multiple `OrderItem` entities, forming a one-to-many relationship.

2.2.6 OrderItem Entity

- **Purpose:** The `OrderItem` entity plays a vital role in tracking the products included in a specific order. It maintains the quantity and total price of each product in the order.
- **Attributes:** Attributes of the `OrderItem` entity consist of `id` for item identification, `product` to link it with a specific product, `quantity` for specifying the quantity of the product in the order, and `totalPrice` for the total cost of the items in the order.
- **Relationships:** The `OrderItem` entity is associated with the `Product` entity and `Order` entity in a one-to-many relationship.

2.2.7 Category Entity

- **Purpose:** The `Category` entity is responsible for categorizing products into specific groups. It helps users browse and filter products more efficiently.
- **Attributes:** The primary attribute in the `Category` entity is `name`, which provides a name for the product category.
- **Relationships:** The `Category` entity has a one-to-many relationship with the `Product` entity, allowing multiple products to belong to a single category.

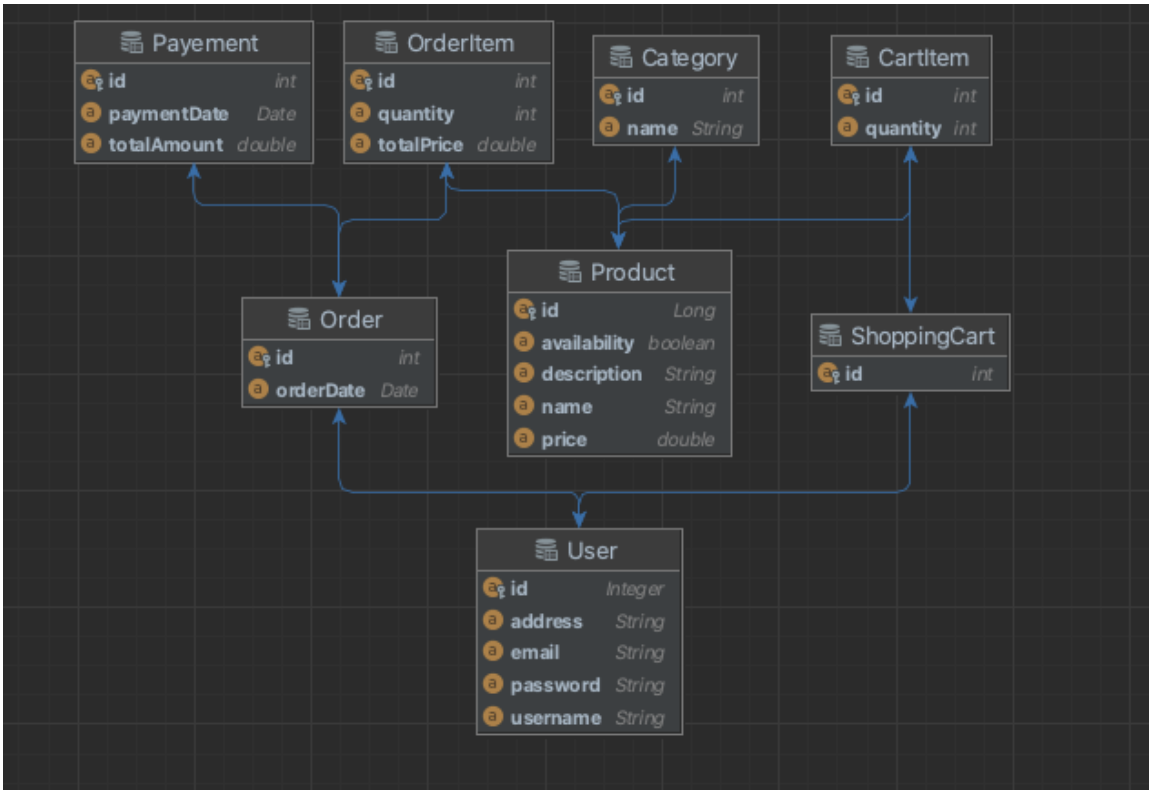
2.2.8 Payment Entity

- **Purpose:** The `Payment` entity records payment details for a specific order, including payment date, amount, and the payment method used by the customer.
- **Attributes:** The `Payment` entity comprises `id` for payment identification, `order` for linking the payment to a particular order, `paymentDate` for recording the date when the payment was made, `totalAmount` for the total cost of the order, and `paymentMethod` to specify the payment method chosen by the customer.
- **Relationships:** The `Payment` entity is linked to the `Order` entity through a one-to-one relationship, ensuring that each order has a corresponding payment.

These entities collectively contribute to the core functionality of our E-commerce application, enabling users to browse, purchase, and manage products efficiently. The relationships between these entities define the database structure and ensure data integrity throughout the application. This robust entity model forms the foundation for the system's capabilities, providing a seamless shopping experience for users.

2.3 Entity Diagram

This Entity-Relationship (ER) diagra provides a clear overview of how the entities are interconnected in our E-commerce application, allowing for a more intuitive understanding of the database structure.



Example of Product Entity class code :

```
@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(nullable = false)
    private Long id;

    private String name;
    private String description;
    private double price;

    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "category_id")
    private Category category;
    private boolean availability;

    @OneToMany(mappedBy = "product")
    private List<CartItem> cartItems;

    // One-to-Many Relationship with OrderItem
    @OneToMany(mappedBy = "product")
    private List<OrderItem> orderItems;

    public Product() {
    }

    public Product(Long id, String name, String description, double price, boolean availability) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.price = price;
        this.availability = availability;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
```

```
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    public boolean isAvailability() {
        return availability;
    }

    public void setAvailability(boolean availability) {
        this.availability = availability;
    }

    public List<CartItem> getCartItems() {
        return cartItems;
    }

    public void setCartItems(List<CartItem> cartItems) {
        this.cartItems = cartItems;
    }

    public List<OrderItem> getOrderItems() {
        return orderItems;
    }

    public void setOrderItems(List<OrderItem> orderItems) {
        this.orderItems = orderItems;
    }
}
```

The `Product` entity in the E-commerce application serves as the cornerstone for product management. Configured with a primary key, it maps to a database table named "products." This entity captures crucial product details, including name, description, price, and availability. It establishes a many-to-one relationship with the `Category` entity to categorize products efficiently. Additionally, it participates in one-to-many relationships with `CartItem` and `OrderItem` entities, facilitating product tracking within user shopping carts and customer orders. The entity's two constructors, including a parameterized option, offer flexibility when initializing product objects. Overall, the `Product` entity plays a pivotal role in managing and organizing product-related data within the E-commerce application

Section 3: Beans in the E-commerce Applicatio

3.1 Definition of Beans

In the context of Java Persistence API (JPA) and JavaServer Faces (JSF), bean classes serve as managed components that handle specific functionality within an application.

In the provided application, there are several bean classes used in the E-commerce application:

- **ProductBean**: This bean manages product-related data and interactions. It handles operations such as creating new products, retrieving products by ID, updating products, and deleting products. It acts as a bridge between the user interface and the database, ensuring seamless management of product data.
- **ShoppingCartBean**: This bean manages the user's shopping cart and its associated operations. It allows users to add products to the cart, update quantities, and remove items. It also handles the checkout process and order creation.
- **CategoryBean**: This bean handles product category-related operations. It provides functionality to retrieve all categories, create new categories, and associate products with specific categories.

These bean classes utilize JPA annotations to map the data to the database tables and JSF annotations to manage the user interface interactions. They play a crucial role in maintaining the application's state, processing user input, and interacting with the underlying data storage.

3.2 Purpose of Each Bean

In this section, we will take a closer look at the role and functionality of each managed bean inthe E-commerce application. To illustrate, we will use the `ProductBean` as an example. Similar explanations will apply to other beans used in the application.

ProductBean - Managing Product Data and More

The `ProductBean` is a fundamental component in our E-commerce application, serving as a "thin controller." This bean is responsible for managing products' data and orchestrating interactions between the user interface and the database. Here's how it fulfills its role:

- **Data Management**: The `ProductBean` manages various attributes of a product, such as its `name`, `price`, `description`, and the associated `category`.
- **Create New Product**: The `createProduct()` method takes center stage when a new product needs to be added. Upon invocation, it initializes an `EntityManager` and creates a new `Product` entity. The attributes of the product entity are populated with the data entered by the user in the JSF form. Once the product is successfully persisted in the database, the `EntityManager` is closed.

- **Show/Hide Create Form**: To enable a user-friendly product creation process, the `showCreateForm` attribute controls the visibility of the product creation form in the JSF view. The `showCreateForm()` method sets this attribute to `true`, effectively displaying the form.
- **Retrieve Product by ID**: When a specific product is to be retrieved, the `getProductById(Long productId)` method comes into play. This method initializes an `EntityManager`, conducts a database query based on the product's unique ID, and retrieves the product. This functionality is vital for searching and displaying individual products.
- **Retrieve All Products**: All products available in the system are obtained using the `getProducts()` method. By establishing an `EntityManager`, constructing a query to fetch products, collecting the result list, and then closing the `EntityManager, it ensures the data for the entire product inventory is available for display.
- **Update Product**: The `updateProduct(Product product)` method allows users to make modifications to an existing product. It merges the changes made to the product object with the database using the `EntityManager`.
- **Delete Product**: For product removal, the `deleteProduct(Product product)` method comes into play. It locates the managed product entity based on its ID and removes it from the database.
- **Search Product by ID**: The `searchProductById(Long productIdToSearch)` method is indispensable for users seeking a specific product by its unique ID. The searched product is set as the `searchedProduct` attribute, making it available for display in the JSF view.

In essence, the `ProductBean` acts as a vital bridge between the user interface and the database. Its functionality extends to managing products and executing CRUD operations seamlessly, offering an efficient, "thin controller" approac

```
@Named
@RequestScoped
public class ProductBean {

    private String name;
    private double price;
    private String description;
    private Category category;
    private List<Product> products;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    private boolean showCreateForm;

    public boolean isShowCreateForm() {
        return showCreateForm;
    }

    public void setShowCreateForm(boolean showCreateForm) {
        this.showCreateForm = showCreateForm;
    }

    public String showCreateForm() {
        showCreateForm = true;
        return null;
    }

    public void createProduct() {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
        EntityManager em = emf.createEntityManager();

        Product product = new Product();
        product.setName(name);
        product.setPrice(price);
        product.setDescription(description);
        product.setCategory(category);

        em.getTransaction().begin();
        em.persist(product);
        em.getTransaction().commit();

        em.close();
        emf.close();

        showCreateForm = false;
    }

    public Product getProductById(Long productId) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
```

```
        EntityManager em = emf.createEntityManager();

        em.getTransaction().begin();

        Product product = em.find(Product.class, productId);

        em.getTransaction().commit();

        em.close();
        emf.close();

        return product;
    }

    public List<Product> getProducts() {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
        EntityManager em = emf.createEntityManager();

        em.getTransaction().begin();

        Query query = em.createQuery("select p from Product p", Product.class);
        products = query.getResultList(); // Set the products field

        em.close();
        emf.close();

        return products;
    }

    public void updateProduct(Product product) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
        EntityManager em = emf.createEntityManager();

        em.getTransaction().begin();
        em.merge(product);
        em.getTransaction().commit();

        em.close();
        emf.close();
    }

    public void deleteProduct(Product product) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
        EntityManager em = emf.createEntityManager();

        em.getTransaction().begin();
        Product managedProduct = em.find(Product.class, product.getId());
        if (managedProduct != null) {
            em.remove(managedProduct);
        }
        em.getTransaction().commit();

        em.close();
        emf.close();
    }

    private Long productIdToSearch;
    private Product searchedProduct;

    public Long getProductIdToSearch() {
        return productIdToSearch;
```

```
    }

    public void setProductIdToSearch(Long productIdToSearch) {
        this.productIdToSearch = productIdToSearch;
    }

    public Product getSearchedProduct() {
        return searchedProduct;
    }

    public void setSearchedProduct(Product searchedProduct) {
        this.searchedProduct = searchedProduct;
    }

    public String searchProductById(Long productIdToSearch) {
        System.out.println("Searching for product with ID: " + productIdToSearch);
        searchedProduct = getProductById(productIdToSearch);
        return null;
    }

}
```

Section 4. JavaServer Faces (JSF) in the E-commerce Application

In the context of our E-commerce application, **JavaServer Faces (JSF)** acts as the primary framework for constructing the web-based user interface. JSF, a robust Java web framework, facilitates the creation of dynamic, component-based web applications.

4.1 Introduction to JavaServer Faces (JSF)

JavaServer Faces (JSF) is a Java-based web application framework that empowers developers to construct web applications with reusable UI components. It provides the essential tools for creating dynamic, interactive, and user-friendly web interfaces. JSF follows the MVC (Model-View-Controller) architectural pattern.

4.2 Product List Example

Let's delve into an example to understand how JSF is applied in our E-commerce application. We will utilize the "Product List" as a demonstration:

- **Search Functionality**: The `<h:inputText>` component and `<h:commandButton>` enable users to search for a specific product. When the "Search" button is clicked, the `searchProductById` method in the `ProductBean` is invoked.
- **Create New Product**: Another `<h:commandButton>` labeled "Add New Product" allows users to display the form for creating a new product. This form is conditionally shown using the `<h:panelGroup>` component with the `rendered` attribute controlled by

`productBean.showCreateForm`. After filling out the details and clicking "Save," the `createProduct` method in the `ProductBean` handles product creation.

- **Search Result**: A search result is presented using the `<h:outputText>` component, which is only displayed when a valid product is found. The product's details are retrieved from the `productBean.searchedProduct`.
- **Product List Table**: The `<h:dataTable>` component is employed to list all available products. It iterates through the collection of products stored in `productBean.products`. Each product's attributes, such as `id`, `name`, `price`, `description`, and `category`, are displayed in respective columns.
- **Actions**: For each product, two buttons, "Edit" and "Delete," are accessible using `<h:commandButton>`. Clicking "Edit" invokes the `updateProduct` method, allowing users to make modifications. The "Delete" button triggers the `deleteProduct` method, enabling product removal.

This demonstration showcases how JSF components and managed beans are interwoven to create a dynamic and interactive user interface for managing products in the E-commerce application.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<f:view>
    <h:form>
        <h1>Product List</h1>

        <!-- Search input field and button -->
        <h:inputText value="#{productBean.productIdToSearch}" label="Product ID" />
        <h:commandButton action="#{productBean.searchProductById}" value="Search" />


        <h:commandButton action="#{productBean.showCreateForm}" value="Add New Product" />

        <!-- Add a form for creating a new product -->
        <h:panelGroup rendered="#{productBean.showCreateForm}">
            <h3>Create New Product</h3>
            <h:inputText value="#{productBean.name}" label="Name" />
            <h:inputText value="#{productBean.price}" label="Price" />
            <h:inputText value="#{productBean.description}" label="Description" />
            <h:inputText value="#{productBean.category}" label="Category" />
            <h:commandButton action="#{productBean.createProduct}" value="Save" />
        </h:panelGroup>

        <!-- Display the search result -->
        <h:outputText rendered="#{not empty productBean.searchedProduct}">
            <h1>Search Result</h1>
            <p>ID: #{productBean.searchedProduct.id}</p>
```

```
            <p>Name: #{productBean.searchedProduct.name}</p>
            <p>Price: #{productBean.searchedProduct.price}</p>
            <p>Description: #{productBean.searchedProduct.description}</p>
            <p>Category: #{productBean.searchedProduct.category.name}</p>
        </h:outputText>


        <h:dataTable value="#{productBean.products}" var="product" border="1">
            <h:column>
                <f:facet name="header">ID</f:facet>
                #{product.id}
            </h:column>
            <h:column>
                <f:facet name="header">Name</f:facet>
                #{product.name}
            </h:column>
            <h:column>
                <f:facet name="header">Price</f:facet>
                #{product.price}
            </h:column>
            <h:column>
                <f:facet name="header">Description</f:facet>
                #{product.description}
            </h:column>
            <h:column>
                <f:facet name="header">Category</f:facet>
                #{product.category.name}
            </h:column>
            <h:column>
                <f:facet name="header">Actions</f:facet>
                <!-- Edit Button -->
                <h:commandButton action="#{productBean.updateProduct(product)}" value="Edit" />
                <!-- Delete Button -->
                <h:commandButton action="#{productBean.deleteProduct(product)}" value="Delete" />
            </h:column>
        </h:dataTable>


    </h:form>
</f:view>
</html>
```

Demonstration Image: "Product List Interface"

Below is a visual representation of the product interface within the E-commerce application discussed in this report. The image illustrates a specific aspect of the application's user interface, providing an example of how product information is presented and managed.

Conclusion

In this report, we delved into the modeling, implementation, and evaluation of an E-commerce application designed to streamline the management and sale of products. By adopting a well-defined set of entities, bean classes, and JavaServer Faces (JSF) views..

We initiated our journey by presenting an overview of the essential entities, providing a comprehensive understanding of the structure and relationships that underpin the application. The entities encompassed product, user, shopping cart, order, and more, each playing a unique role in realizing the application's functionality.

Next, we turned our attention to the bean classes, which act as the thin controllers and data access layers of the application. Using the ProductBean as an example, we observed how beans facilitate the seamless interaction between the application's views and the underlying data. Through these beans, critical operations, such as creating, updating, and searching for products, are managed with precision.

The JSF views represent the final piece of the puzzle, as they provide a rich and interactive user interface for both customers and administrators. Taking the product list as a prime example, we uncovered how JSF empowers the presentation layer with its dynamic components. The ability to search, create, and manipulate product information through a user-friendly interface was demonstrated.

In summary, this E-commerce application, with its meticulously designed entities, efficient bean classes, and interactive JSF views, signifies a robust solution for managing and selling products. It brings together the power of data modeling, seamless data access, and user-friendly presentation, resulting in an application that is not only effective but also a delight to use. The combination of strong structural foundations and intuitive user experiences positions this application as a valuable tool for businesses seeking to establish a prominent online presence in the world of E-commerce.