OBJECT ORIENTED ANALYSIS,

DESIGN AND PROGRAMMING

Ecommerce Application

Group: 5

Requirement Analysis

PROJECT MEMBERS:

Snehal Nikam- W1650226

Saurabh Deshmukh- W1648445

Shreya Krishnamoorthy- W1651153

Table of Contents

1. APPLICATION REQUIREMENTS

1.1 OVERVIEW

The objective of this project is to design and develop an E-commerce platform which performs various business functions that occur in online shopping such as user login, products display, shopping cart, order placement, Admin/Seller portal etc.

1.2 REQUIREMENTS

Develop a User Interface which allows the admin to:

- Login through admin credentials
- Add/Remove/Edit categories and products

Develop a User Interface which allows the guest users to:

- Register an account for the first time
- Browse through the products

Using the same user interface, registered users must be allowed to login and perform the following tasks:

- Browse through the products
- Add/Remove products from wish list
- Add/Remove products from cart
- Checkout from cart
- Make the payment
- View Order History
- Logout

The application should authenticate the admin and user credentials by verifying them in the database.

The application should have a user friendly interface with a robust and extensible backend.

1.3 FUNCTIONAL WORKFLOW

- The application is built with a user friendly interface which allows users to browse and buy products with ease.
- When the user accesses the portal, the home page of the application appears.
- If the user is unregistered, the user can register an account and login to continue shopping or the user can still browse the products without registration.
- Registered users can login by entering the user ID and password.
- After login, the users browse through the products or categories that appear on the screen.
- The users can add the products they like to the wish list or shopping cart.
- Once the items are in the cart, the users can Checkout to make the payment through the payment portal.
- The order is then placed and the user can view the order history.
- Once the shopping is completed, the user can log out.
- Other than the above functionalities for the user, there is a portal for Admin login which facilitates the Admin/Seller to perform relevant operations.
- Through the Admin login, the Admin/Seller can enter the credentials which is verified by the database and the Admin/Seller portal appears.
- The interface allows the Admin/Seller to add categories and add products to the appropriate categories.
- Through the portal, the history of orders by different users is also visible.
- The Admin/Seller can change the order status which would be visible to the user, depending on the status of the order delivery.
- The Admin/Seller can log out once the activities are performed.

2. OBJECT ORIENTED CONCEPTS

The entire application is designed with strict compliance with the object oriented principles to ensure a good and organized coding standard, code reusability, extensibility and increased security. Java (Spring Boot framework) is used for the backend with the Model-View-

Controller pattern, which facilitates Object Oriented design. All code is organized into classes and objects, which each class containing the appropriate state variables and member functions. Inheritance is used for all is-a class relationships where classes are extended to pass on specific functionalities to the derived classes. The objects interact with each other according to the requirement and with the required access level ensuring further security and avoiding accidental code changes. By employing Object Oriented concepts, the classes are well structured into modules, improving the readability of the complex programs involved. Appropriate data structures like Lists and ArrayLists are used in different places in the code.

2.1 Inheritance

Inheritance allows a class to get extended to a derived class to add additional functionality. Through inheritance, the child class/derived class inherits the attributes and methods of the base class. This allows the creation of hierarchy in classes and the extension of properties and behaviors from the parent class to the child class.

In this application, Inheritance is used in all cases of is-a relationships. For example, we extend the classes- CartRepository, CategoryRepository, ProductRepository, TokenRepository, UserRepository, WishListRepository from the JpaRepository which facilitates connectivity to the database.

2.2 Polymorphism

Polymorphism is closely related with inheritance. It means “many forms”. Polymorphism allows components of the program to exhibit different forms based on the requirement. Polymorphism in Object Oriented Programming is achieved through Overloading (Compile time) or Overriding (Run time). This greatly increases reusability and allows extension of code by adding different forms.

In this application, both Overloading and Overriding are used in code. More specifically, constructors are overloaded in most cases allowing to instantiate objects with different types of parameters. For example, in the Category class, the Admin/Seller may or may not chose to create a category with an image background. Similarly, overriding is used to override methods of a base class to ensure

implementation of the derived class method with the same signature. This can be seen in the implementations of Category, Product and User.

2.3 Encapsulation

Encapsulation allows to hid the implementation of data through access restrictions. This is a key component in ensuring security. Encapsulation is achieved by binding together the data members and member functions within a class and allowing specific accesses based on the required access level.

In this application, we have achieved encapsulation by ensuring that all code is structured into organized classes and all key data memebers, especially the user information such as userID, email, password, first name, last name etc are set to private to ensure access only through the object of the class to ensure security. Also, other data such as productID, description, price, unitsInStock etc are also private to eliminate mishandling of product information.

2.4 Abstraction

Abstraction refers to hiding the details of implementation of the code and only showing the outer functionality to the user. This is generally achierved through abstract classes, interfaces and methods.

In this application, abstraction is achieved by structuring the code into distinct directories. The use of the Model-View-Controller further facilitates Abstraction by creating a distinction between what is viewed and how it works. Abstraction is also achieved by the use of interfaces in various sections of the code such as ProductRepository, UserRepository etc, to act as an intermediate between the difference classes.

3. Design Patterns

Design patterns are also used to ensure a clearer implementation of the intended design and to accommodate changes in the future.

3.1 Decorator Pattern

The decorator pattern is also known as “Wrapper”. This structural design pattern involves linking a wrapper object to a target object. To an object, an additional functionality is added before it is passed on to the target. The object can be combined in multiple wrappers which adds the combined behavior of all the wrappers to the object.

In this application, wrapper classes are used to add additional behavior to target objects. The Wrappers directory contains the Wrapper classes for Cart- AddToCartWrapper.java, CartItemWrapper.java, CartWrapper.java; Product- ProductWrapper.java and User- SignInResponseWrapper.java, SignInWrapper.java, SignupWrapper.java. For example, in the ProductService.java file, the function createProductWrapper() is used to create a wrapper over the Product class object. This is used to set the data members of the ProductWrapper object using the data members of Product object and the set values of the Wrapper object are used for further processing.

4. MODEL-VIEW-CONTROLLER

The Model-View-Controller architectural pattern is used in this application to ensure organized units and security. The application is divided into three main logical units- the model, the view and the controller. Each of these components handle specific aspects of the application development.

4.1 MODEL:

The Model deals with the actual data related logic behind what the user works with. This can be any business logic data. Generally, the model is the closest logical component to the database in the backend. It contains the main backend core logic code and also retrieves from and sends data to the database.

In this application, the Model directory contains AuthenticationToken.java, Cart.java, Category.java, Product.java, User.java and Wishlist.java which contain the core logic for these components.

4.2 VIEW:

The View deals with all of the User Interface Logic. This includes the data logic closely related with the what is visible to the user. In other words, this is generally the part of the backend application which is logically closest to the Front-end of the application.

In this application, the Service directory contains the View classes such as AuthenticationService.java, CartService.java, CategoryService.java, ProductService.java, UserService.java and WishlistService.java. These classes contain the direct services offered to the users.

4.3 CONTROLLER:

Controller is the interface between the Model and the View. This is used to coordinate for receiving requests from the View, processing the data or performing the operation using the core logic in the Model which also communicates with the database and rendering the final output back to the View.

In this application, the Controller directory contains CartController.java, CategoryController.java, ProductController.java, UserController.java, WishlistController.java which act as the interfaces between the respective Model and View classes.