Object Oriented Design and Development

COM528 / 504

Individual Assignment – Catalogue Shopping Cart System

By Guillermo Caballero Massa

**Documentation**

This project is a web application developed in java, using spring boot and JPA that simulates an online store, which contains a shopping cart that we can fill with shopping items and generate an invoice when buying them (once we have registered).

At the same time, we can log in as administrators and manage both the users, the invoices, and the catalogue (adding, deleting, and editing the available products).

To start this project, it is necessary to execute the command "mvn clean install" in the base of the project to build it.

Then we execute "cd web" and once inside the web module we execute the command "mvn spring-boot run".

You will be able to see the application at http://localhost:8080/shoppingCartApplication/

On the other hand, we can run it with Netbeans and a built-in Tomcat server, select 'run' and netbeans will activate a Tomcat instance and launch the application.

The architecture used is MVC, which means that we will organize the code in Models (access to the database and obtaining data from it), Views (the pages in charge of displaying the requested information) and Controllers (in charge of the managing URLs and connecting views with models).

The database used in this project is a relational database in SQL that is stored in memory.

To access the information in the database just mentioned, JPA is used, which allows us to use a repository for each entity that we have to manage the data and once injected into our services, these will be fully functional.

Once we have started the application and we go to home we will find the home.jsp view

Captura de pantalla de un celular

Descripción generada automáticamente

That has been loaded thanks to the following code in the MVCController:

Imagen que contiene Texto

Descripción generada automáticamenteImagen que contiene Texto

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

From line 57 to 63 the route is established and the possible values that we have been able to receive by parameter are declared.

Later we have the code that allows us to add and remove items from the cart (thanks to shoppingCartService); and that loads us the shoppingItems available from the catalog in an array that later we load inside the model

At the same time, ShoppingService and ShoppingCart are the services that contain the functions that contain the application logic.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

These are the functions that the services include and have to inject the necessary repositories into their respective implementations

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Same logic is used with users and invoices

Once we have selected the items we want to buy and click on "proceed to checkout" the following screen will appear

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Once we have entered our bank details, the following code will be executed (responsible for making the payment and creating the invoice if they are accepted by the bank)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza baja

This code will send the payment data to the purhcaseITems function within shoppingService and upon receiving the response there are two options:

1- in case of having been accepted, the invoice will be created and the money will be transferred from one account to another.

And a success message will appear and we will return to the home screen with and empty cart.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto, Correo electrónico

Descripción generada automáticamente

2-If it has failed, an error message will appear and you will return to the same payment screen

Interfaz de usuario gráfica, Texto, Aplicación

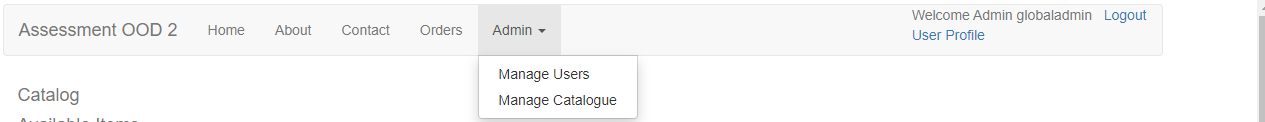
Descripción generada automáticamente

The last step as clients is to see our invoice when we go to the orders section.

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

On the other hand, if we register as administrators, an extra section will appear in the header.



We will have access to manage and see all registered users

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

And products in the catalogue

Captura de pantalla de computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Once we add a new Item it will be part of the catalogue.

Captura de pantalla de un celular

Descripción generada automáticamente

And if we access the invoices part as administrator we will be able to see all the invoices and the name and ID of the buyer

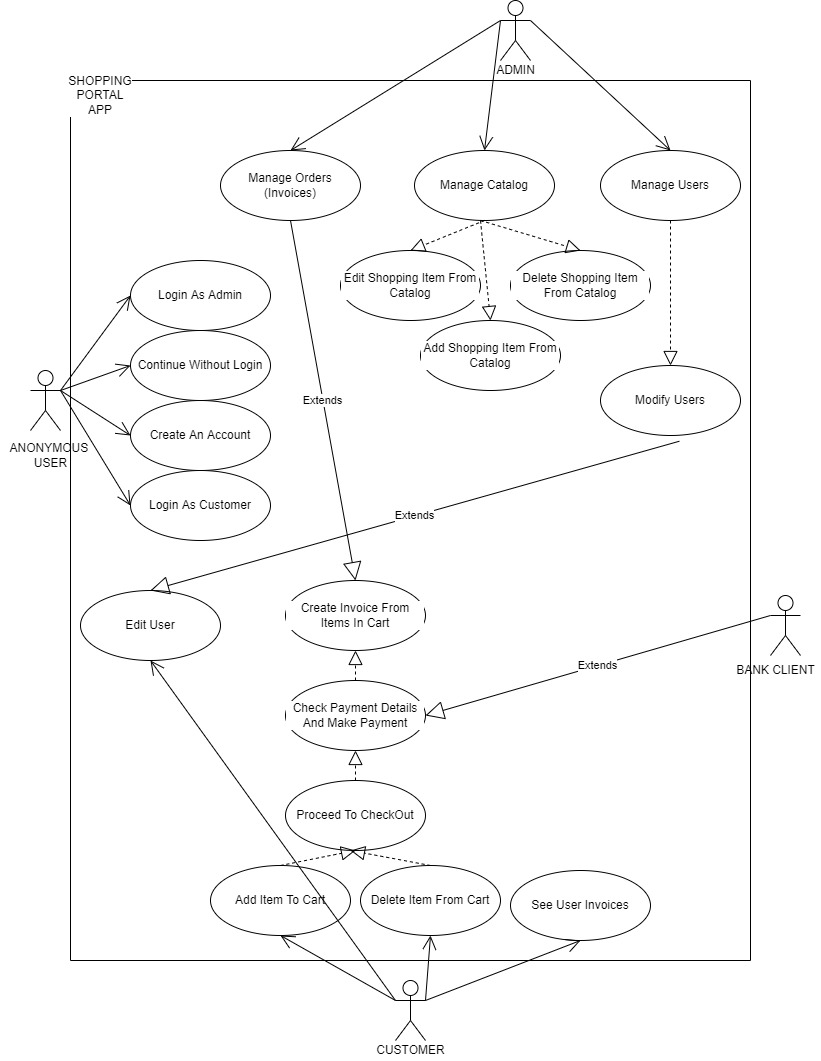
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. **Use cases**

The use case shows that it is an application where there are mainly 3 actors (4 if we consider the bank's client); they are the Anonymous User, the Registered User, and the Administrator.

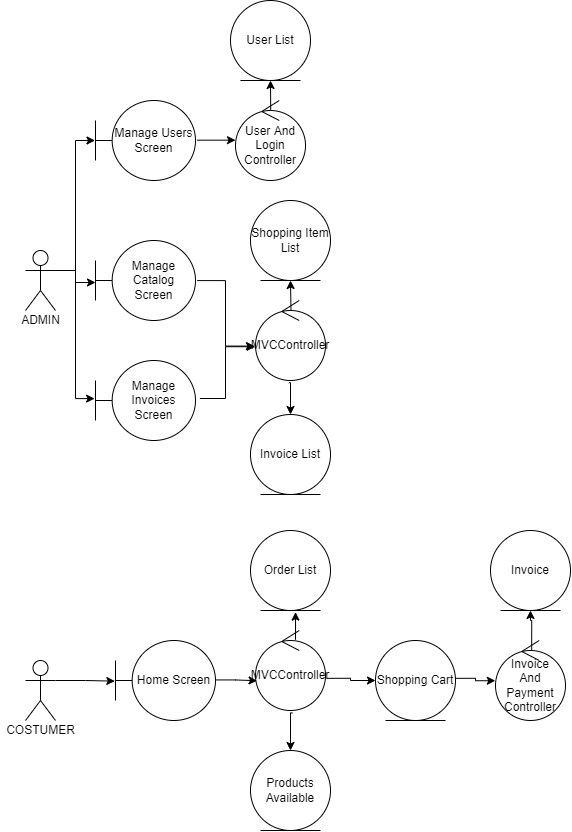
1. The anonymous user can:
   1. Log in as an Admin or Customer
   2. Create an account
   3. Add and remove shopping items to the shopping cart
2. The Client can:
   1. Add and remove shopping items to the shopping cart
   2. Enter payment details
   3. Pay and create invoice
   4. See previous orders (invoices)
   5. Modify user data
3. The Administrator can:
   1. Manage users and modify their data
   2. Manage invoices (see all invoices)
   3. Manage the shopping items catalogue (add, remove and modify products from the catalogue)
4. The Bank´s client:
   1. Checks the payment details entered by the user and transfers the money from one account to another (in case any problem occurs, the purchase process is aborted, and no invoice is created)



1. **Model**

The followed model in this project is a scheme where the code is divided into 2 controllers (MVCController and invoiceAndPaymentController), the 3 services used: ShoppingService, ShoppingCart and InvoiceService (including their respective interfaces and implementations, as well as repositories where all the necessary functions have been added to achieve a fully functional store) and several views that have been added to display the content requested to the user (as well as the view checkout, orders, catalog, etc). All this considering the use of an external API (bank).

1. **Robustness diagrams / Sequence Diagrams**

****

1. **Report**
2. Decisions you made when drawing up your domain model and use case texts

The main points to consider during the writing of the use case and the domain model, have been that it is an application used by 4 actors and that it must be able to handle and display data hosted in a database (in this case in memory). Because for this application

it is necessary to use 3 different entities (user, invoice and shoppingItem), it has resorted to 3 different repositories with access to the database (working as DAO); These 3 repositories

have allowed the 3 services to use the data dynamically (ShoppingServiceImp, InvoiceServiceImp and ShoppingCartImp)

1. Any decision made when drawing up your diagrams

The most complex part when drawing the diagrams has been identifying which parts of the project are connected to each other, to achieve an application that respects the integrity of the data.

In addition, it was decided to add the anonymous user as an actor because it has use cases and it can use certain functions of the application without registering

1. Detail on places where your code did not match your design, and why

One of the problems that took me the longest to identify and solve has been due to the declaration of database relationships, since in the Invoice model it is declared that the relationship with the purchaseItems is @OneToMany and this produced an error when trying to buy the same object more than once (due to the way the database is created, which joins the invoice and the set of objects purchased by the id of both).

This problem is solved by changing the relation to @ManyToMany (Line 63 in Invoice.java)

Texto

Descripción generada automáticamente con confianza media

1. Critical evaluation of your code and/or design

The main points to improve in this project lie in the possibility of having written a cleaner code (adding all the logical code of the application in the service and not in the controller and having created a controller for each service, so that the code would be More organized)

In addition, the ideal way to implement a middleware would suppose that the controller is not called until it has been verified that the required conditions are met, however in this project the conditions are checked (such as that we have logged in before seeing the invoices) in the first few controller code lines.

1. **Demonstration**