Gareth landy

C15735901@MYDIT.IE

Software Design Patterns

Contents

[Database Tables 2](#_Toc7194591)

[Database Columns 2](#_Toc7194592)

[Product 2](#_Toc7194593)

[User Account 2](#_Toc7194594)

[Shopping Cart 3](#_Toc7194595)

[Product Review 3](#_Toc7194596)

[Design Requirements 3](#_Toc7194597)

[Star Comments and Reviews 5](#_Toc7194598)

[Patterns 5](#_Toc7194599)

[UseCases 6](#_Toc7194600)

Link to GitHub:

<https://github.com/garethlandy/Paddy2019CA4>

## Database Tables

The structure of the database was kept simple as there are only four tables used in this assignment. There are reference (foreign) keys to other tables within this application but I concentrated on the use of SQL to get the data required from the tables to then display the results within the classes.

## Database Columns

Tables:

1. Product
2. User Account
3. Shopping Cart
4. Product Review

## Product

All items relating the actual product for sale are stored here:

Code: references the unique id of the product.

Name: Brand name of the product.

Price: Price in Euro stored as an integer value.

Manufacturer: Name of Manufacturer of the product.

Category: Assigned a specific Category for searching.

ImageFile: A reference to an image to attach to a product.

Stock: Number of items currently in stock

## User Account

All items relating the Users of the system including the administrator are stored here:

Firstname, Lastname Username and password.

Each User has their own current billing address which can be updated by both the admin and the Customer.

Admin: this column has a Boolean value which allows certain aspects of the system to remain hidden to the customer. This value runs across the application enabling the admin to update, edit and delete products while also maintaining the stock levels of each. This admin value is then hidden from the user. Before each URL is called the system will check the admin rights of the user that is currently logged in and will then redirect them to the correct page to view the requested data.

## Shopping Cart

The Shopping Cart table has three columns. Username, the product code and a column that has a Boolean value that is called paid that is updated once the payment confirmation has been processed.

## Product Review

The Product review table has a reference to the product table and also has a reference to the user table. Other colums include:

Meaasge: this is where the user creates a review of the product after they have purchased it.

Star Rating: A star rating (1-5) is then added to the purchased product.

## Design Requirements

My design of this application included:

MySQL Database

Servlets using html JavaScript and jQuery.

As there were two different users of the system it was important to keep the admin rights of each user separate. This reduced the need to have separate java classes when a user of the system logged. Each Servlet had a corresponding jsp page or pages to view the returned data. Once any user logged in, they were then stored in a HTTP session which carried the user throughout the application.

Cookies were used to reload pages if the application url was terminated by the user without logging out. This enabled the user to be brought back to the corresponding page when the URL was reloaded. The next Diagram shows what each user (Admin, non Admin) sees when the first login to the application.

Customer/Home

A screenshot of a computer

Description automatically generated

Admin/Home

A screenshot of a cell phone

Description automatically generated

Admin/ProductList

A screenshot of a social media post

Description automatically generated

Customer/ProductList

A screenshot of a computer

Description automatically generated

The biggest difference here is within the Product list link and the extra Shopping cart icon on the customer page. As an admin you have the rights to update a product in terms of stock level or price. These changes are then instantly available to be viewed by the customer.

As a customer you can add items to your shopping cart with the number of items currently in your cart displayed beside the icon where they can also be deleted from the cart when the customer views all their items in their carts.

The search bar at the top of the screen can look for items that are currently in the Categories file and the Name file. Partial matches are also available with the results displayed on the above screens. Each column can also be sorted in ascending or descending order by clicking the name of the column. This sorting feature runs throughout the application.

## Star Comments and Reviews

One a purchase has been made by the user the will then be able to leave a start rating of the product and enter in a review. These reviews are then enabled to be viewed by all customers.

A screenshot of a social media post

Description automatically generated

Each product can then be sorted by their product code and star rating in descending and ascending order.

## Patterns

Strategy:

The strategy pattern was used when the customer wished to purchase items that were in their cart. As a lot of the functionality was the same across the three different types of credit card that could be used. Each payment type was directed through an Interface that contained a single method to pay() that was passed as an argument. Each of three different credit card classes then extended the interface but had methods in their respective classes that were different to each.

Prototype:

The prototype pattern was used when I was trying to implement a payment history section of the application where users would only see their history and admins could view all payment history across all users. By having an Interface that could take in an argument to clone and return different payment history objects I tried to implement a scenario where both classes were identical, but the return values would differ slightly. I thought by using this pattern it would save on a great deal of code implementation, but I found that the SQL worked better to a certain degree.

## UseCases

A close up of a map

Description automatically generated