

ICS2608
Final Academic Project



Bountiful Basket

- A WEB APPLICATION -

Created By:

Group 3 – 2CSB

Articulo, Matthew Gabriel B.
De Leon, John Thomas P.
Macuja, Lorenzo Tzar H.
Robles, Joseph Isaac

Submitted To:

Mr. Lawrence G. Decamora III, LPT, OCPJP, MAEd

Table of Contents

Table of Contents	1
Introduction	2
The Web Application	3
Site Navigation Hierarchy	3
Landing Page	3
Error Pages	4
Responsive Web Design	4
Conditional Page Elements	4
Logging into the System	5
Shopping for Products	6
The Shop Page	6
Adding Products to the Cart	6
Checking Out	7
Confirming Orders	8

INTRODUCTION



Bountiful Basket

Bountiful Basket is a simple online shop application. Users can browse the shop's extensive list of products and place an order after logging into the system. The application can manage each user's cart efficiently and ensure that their orders are placed successfully.

The concept for Bountiful Basket came from the developers' desire to help, support, and pay tribute to the country's local farmers who have been in tight spots since the onset of the COVID-19 pandemic. As such, the main purpose of the web application is to provide a platform by which these farmers can sell their produce for a living.

THE WEB APPLICATION

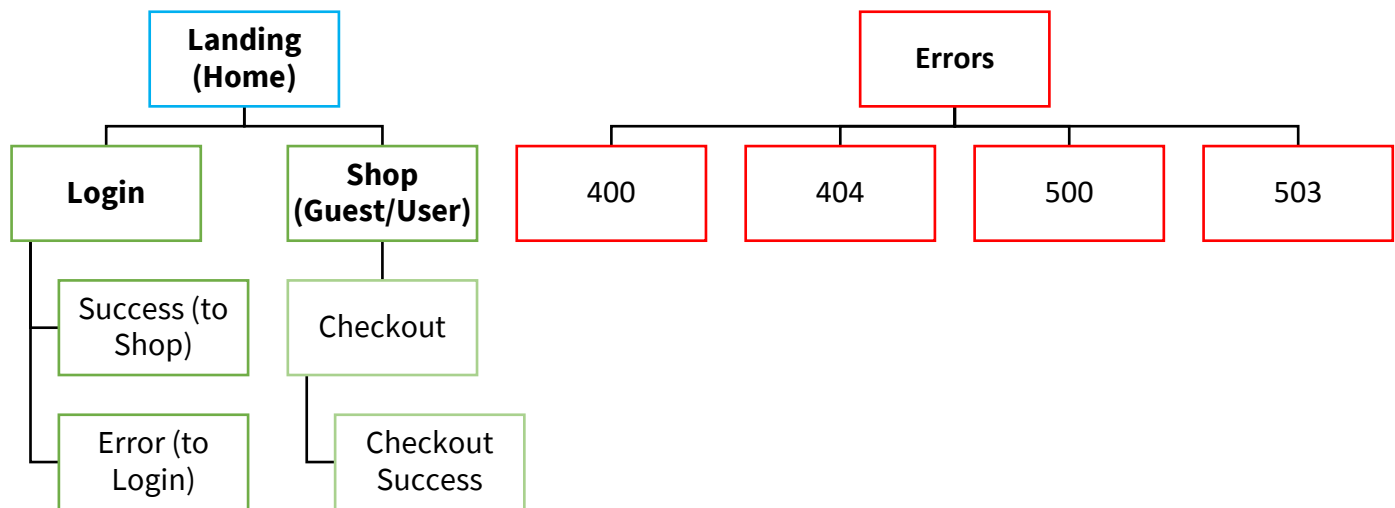


Diagram 1: Site Navigation Hierarchy

The web application consists of 10 different pages as JSPs. Users start from the **Landing** page and can navigate to either the **Login** or **Shop** pages. The Login page is responsible for handling the user's session. Users can still browse the Shop's selection of products on the Shop page, but its functionality is limited as they cannot place orders without logging in first. To go back to the landing page from either of these pages, users can click on the site's logo on the navigation bar.

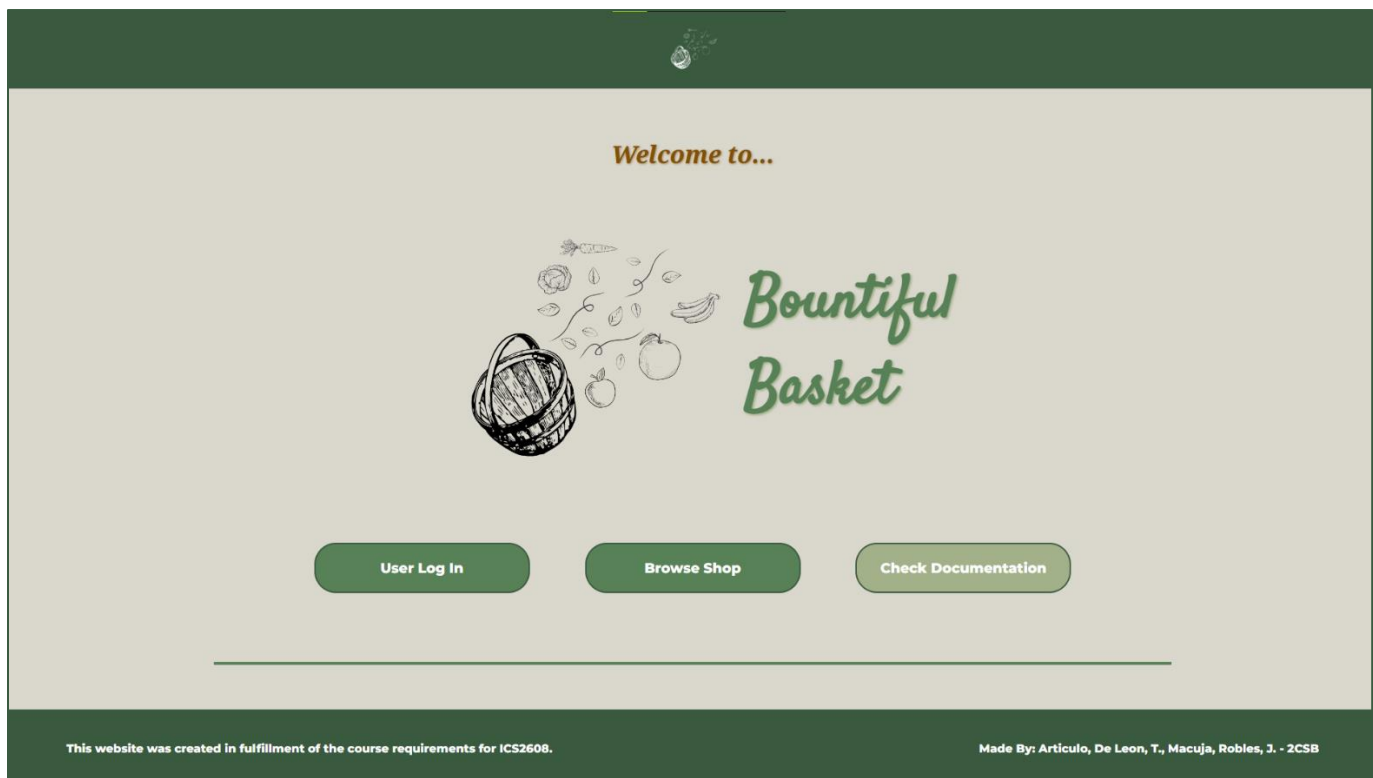


Figure 1: The Landing (Home) Page

Bountiful Basket – Documentation

The application also features customized **Error** pages which are automatically accessed when the application does not function properly, although the chances of this are very rare unless deliberate actions were taken to trigger them to appear.

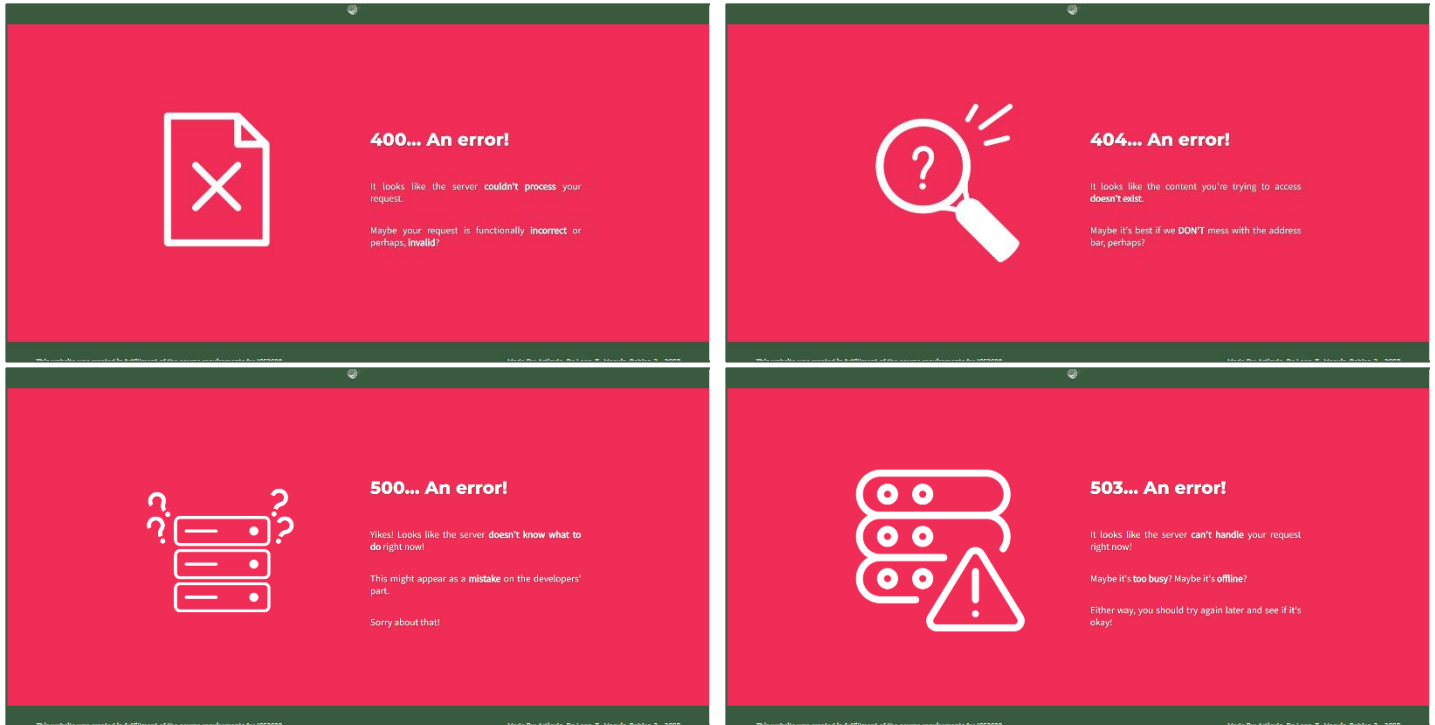


Figure 2: All Customized Error Pages

The site also features a **customized, responsive design** using CSS that adapts to any **computer** screen size, automatically scaling its content as needed. This ensures that elements do not clip nor are positioned weirdly when the screen shrinks/grows to any size.

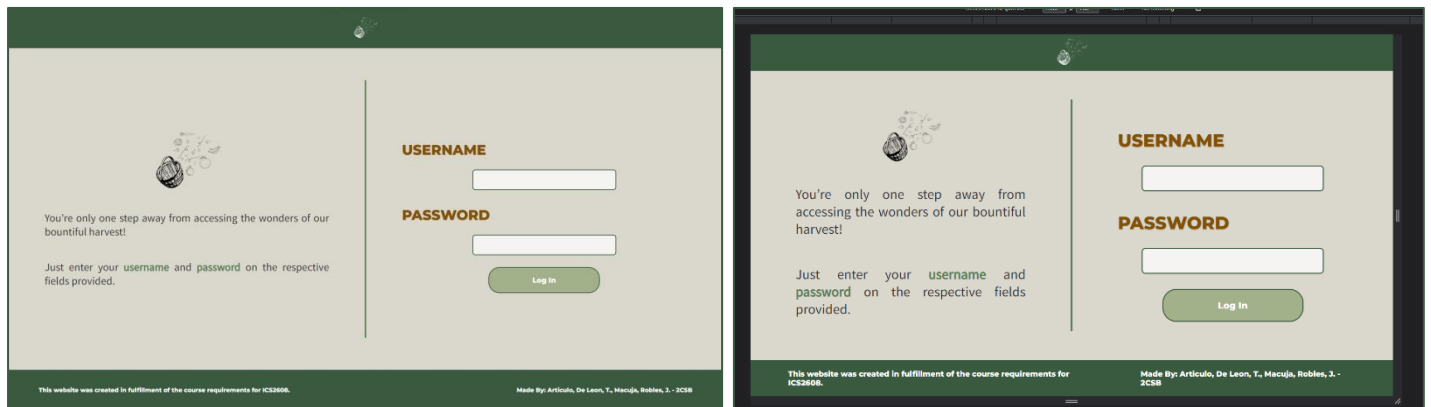


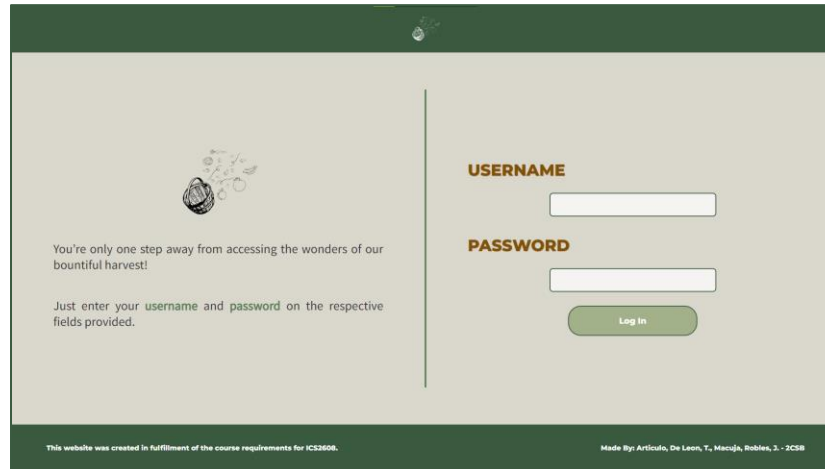
Figure 3: Adaptation based on Screen Size

Aside from that, some elements of each webpage are **conditional** in nature and may only appear when certain conditions, such as the user being logged in, are fulfilled using scriptlets.



Figure 4: Conditional Page Elements

LOGGING INTO THE SYSTEM



The login page features a dark green header with a small logo. The main content area is light beige. On the left, there is a small illustration of a basket with fruit and a text block: "You're only one step away from accessing the wonders of our bountiful harvest! Just enter your username and password on the respective fields provided." On the right, there are two input fields labeled "USERNAME" and "PASSWORD" in orange. Below the password field is a green "Log In" button. At the bottom, there is a dark green footer with small white text: "This website was created in fulfillment of the course requirements for IC33606." and "Made By: Articulo, De Leon, T., Macuja, Robles, J. - 2CSB".

Figure 5: Normal Login Page

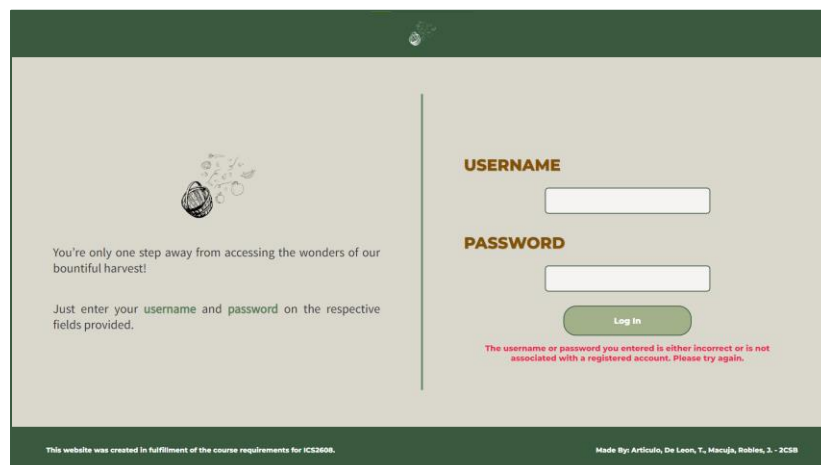
The **Login** page is where users can enter their login credentials to be granted further access to the system and its features. Requests are handled by its dedicated servlet (`LoginServlet`) and is responsible for initializing each user's session data.

Optimally, login data should be obtained from an external source. But for the purpose of showcasing the functionality of the web application, login credentials are hard coded into the system in the `LoginServlet`'s `loadLoginInfo()` method, which reads a pre-defined `HashMap` (`java.util.HashMap`) of user login information data as shown below (Note that these are case-sensitive):

Username	Password
ryuuKur0ki	12345
testuser1	aaaaa
admin1	qwerty
asdad	11111

Diagram 2: Application Login Information

When the user enters incorrect login information, the handling servlet sends a redirect to the `login-error` page which informs them of their mistake and asks them to try again. Otherwise, when the user enters valid login information, the servlet initializes their user ID (as the username associated with their login), shopping cart (as an empty list), and the total cost of their cart (which is initially zero) then redirects them to the shop page.



The login error page is identical to the normal login page, but it includes an error message in red text below the "Log In" button: "The username or password you entered is either incorrect or is not associated with a registered account. Please try again." The footer text remains the same as in Figure 5.

Figure 6: Login Error Page

SHOPPING FOR PRODUCTS

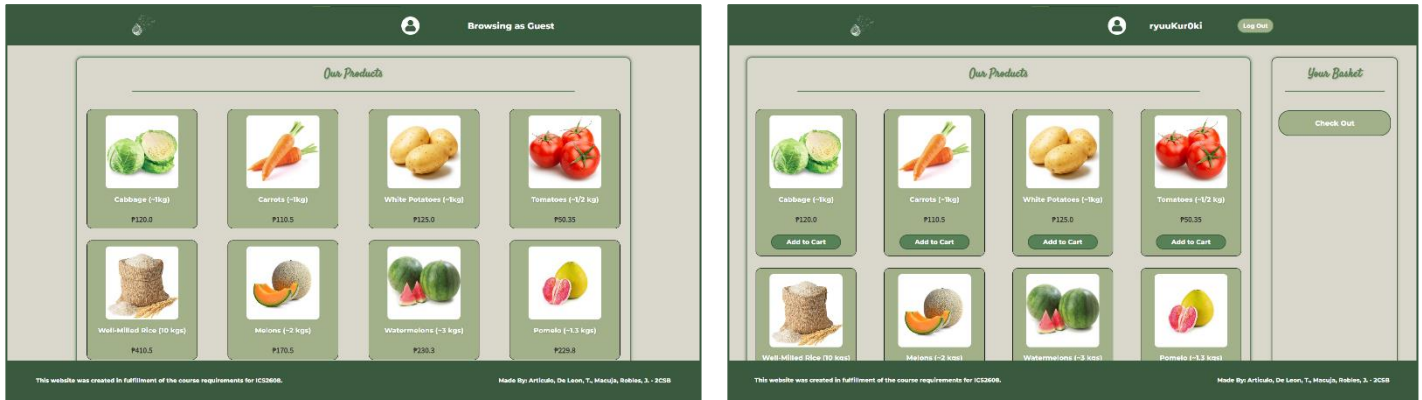


Figure 7: Shop Pages – for Guests (Left) and Users (Right)

The **Shop** page is the main feature of the application and is where users can place orders for various products. Stock items are defined as custom `ShopItem` objects with certain defined properties. The list of stock items is initialized by the application's `ShopInitializer` utility class. This list then dynamically loaded into the shop page to be displayed to the user.

As mentioned beforehand, some pages of the application contain conditional elements. Here, some of these elements can be seen on the page's header and its main contents. When browsing the shop as a guest, the log out button (which sends a request to the application's `LogoutServlet` to handle session invalidation) does not appear, and the text beside displays "Browsing as Guest" instead of the user's ID. As guests do not have a cart to begin with, they cannot add items to it nor can they see the section of the page that displays its contents.

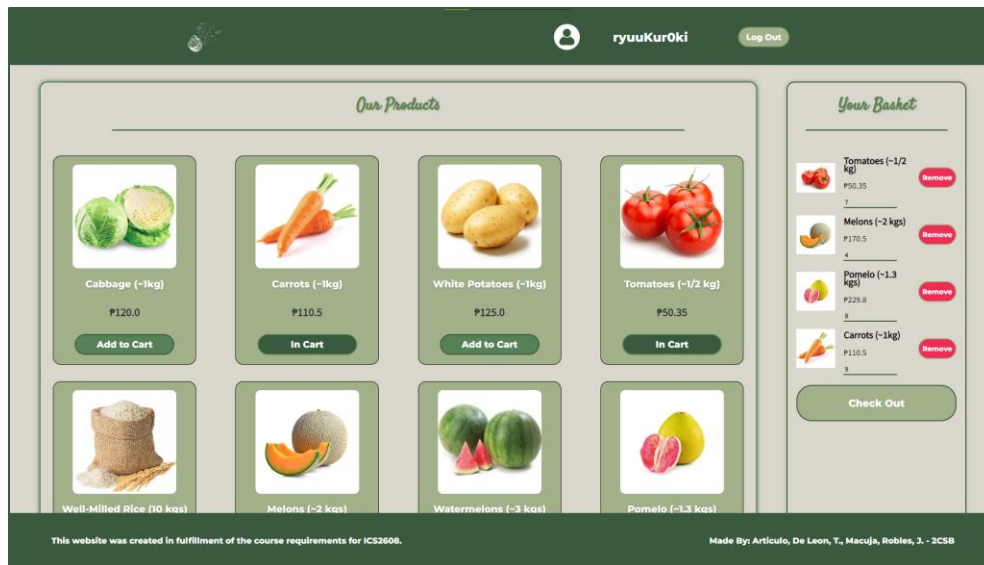


Figure 8: Adding Products to the Cart

Furthermore, when a type of product is in a user's cart, the corresponding **Add to Cart** button is replaced by a disabled **In Cart** button to indicate that it is already in the cart and prevent duplicate entries from being added to the cart (Initially, when this conditional script was not implemented yet, there was logic in the servlet to check if an item was already in the cart and simply add their quantities together. However, this was redundant and was subsequently removed). When an item in the cart is removed from the cart by pressing its corresponding **Remove** button, the item's **Add to Cart** button in the stock list appears once more.

Bountiful Basket – Documentation

The shop's dedicated servlet (`CartServiceServlet`) handles requests to add and remove items to the user's cart. All **Add to Cart** buttons are contained within a single form element and their `value` attribute are set to the associated item's ID. The servlet uses the value sent and uses the `retrieveItemByID()` method from the application's `CartUtils` utility class to determine which item from the stock list is being added to the cart, then updates it accordingly before redirecting back to the shop page for its contents to be display dynamically.

Removing an item from the user's cart works in a similar manner. Whenever a **Remove** button is pressed, the quantities of items in the cart set by the users using its associated number box is saved by passing its current `value` as attributes to the request. The servlet then determines which item to remove from the user's cart by using the same `retrieveItemByID()` method and updates the cart accordingly before redirecting back to the shop page for its contents to be display dynamically.

Shown below are diagrams of these processes when the application is in use:

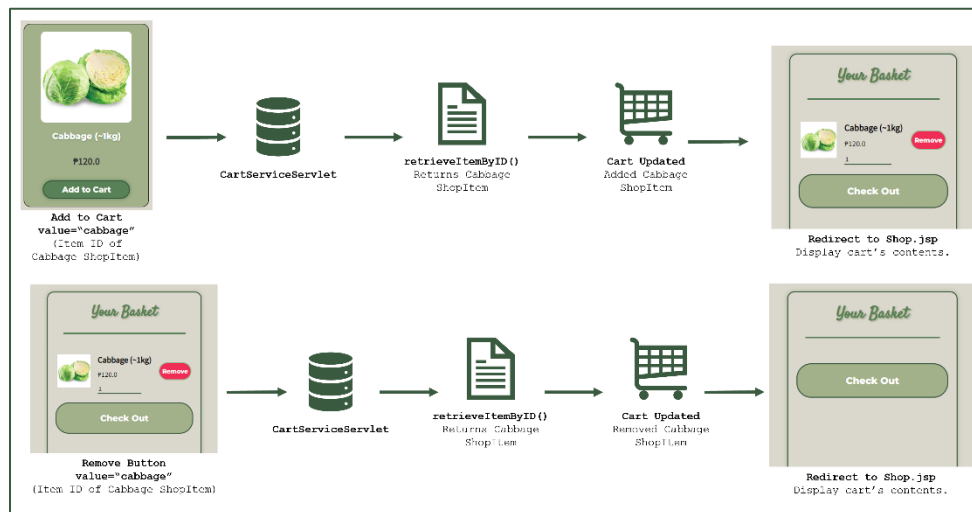


Diagram 3: Add to/Remove from Cart Process

Checking out is a relatively simple process. Any changes made to the quantities of the items of the cart are saved and the request is forwarded to a dedicated servlet handling checkout, which essentially computes the subtotal of each item in the cart ($\text{quantity} * \text{itemPrice}$) and the total cost of the order (summing up all subtotals) and saves it as the corresponding session attribute before redirecting to the checkout page to display this information.

The Order Summary (Checkout) page displays the following information:

Item Name	Item Price	Quantity	Subtotal	Action
Tomatoes (~1/2 kg)	P50.35	11	P553.85	<button>Remove from Cart</button>
Watermelons (~3 kg)	P230.3	7	P1,612.10	<button>Remove from Cart</button>
Cabbage (~1kg)	P120.0	5	P600.00	<button>Remove from Cart</button>
Total Order Cost:			P2,765.95	

Buttons: Cancel Order Proceed and Place Order

Footer: This website was created in fulfillment of the course requirements for ICS2608. Made By: Articulo, De Leon, T., Macuja, Robles, J. - 2CS8

Figure 9: Order Summary (Checkout) Page

Bountiful Basket – Documentation

Additionally, users can still choose to remove items from their cart from this page. It works the same as removing an item in their cart on the shop page and even updates the total order cost accordingly. If users change their mind and decide to add more items to their cart, they can press the **Cancel Order** button to return to the shop page and any changes made on this page will be preserved.

Likewise, they can press the **Proceed and Place Order** button to complete their transaction. It will redirect to a confirmation page showing the value of their order and informing them that it was placed successfully. Users can place another order by pressing the **Continue Shopping** button which clears the contents of their cart and reset its total value before sending them back to the shop page. Likewise, users who want to stop shopping can press the **Log Out and Exit** button which invalidates the current session and redirects them back to the landing page.

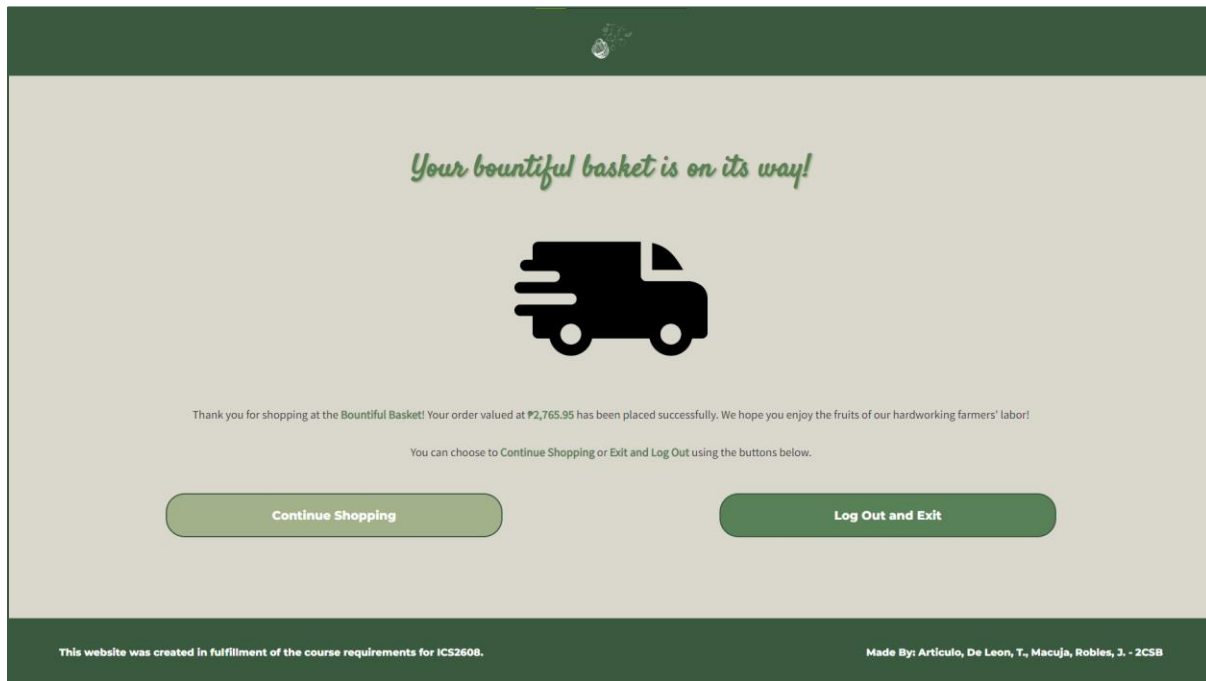


Figure 10: Checkout Success Page