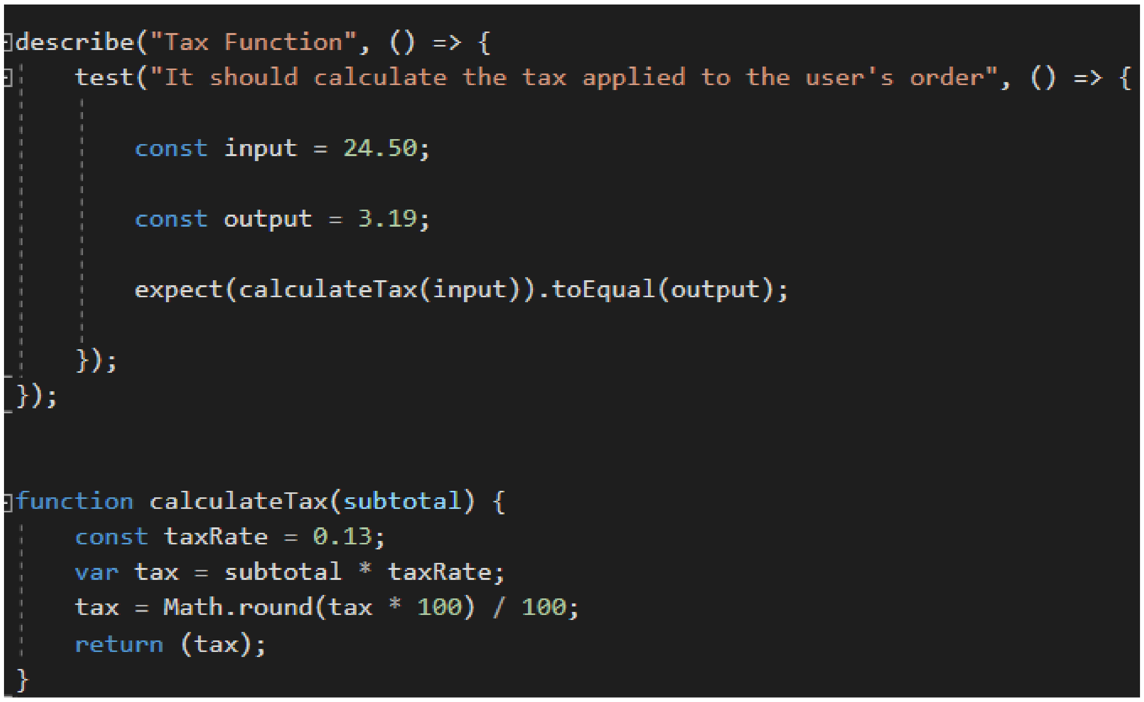
F. Test Cases

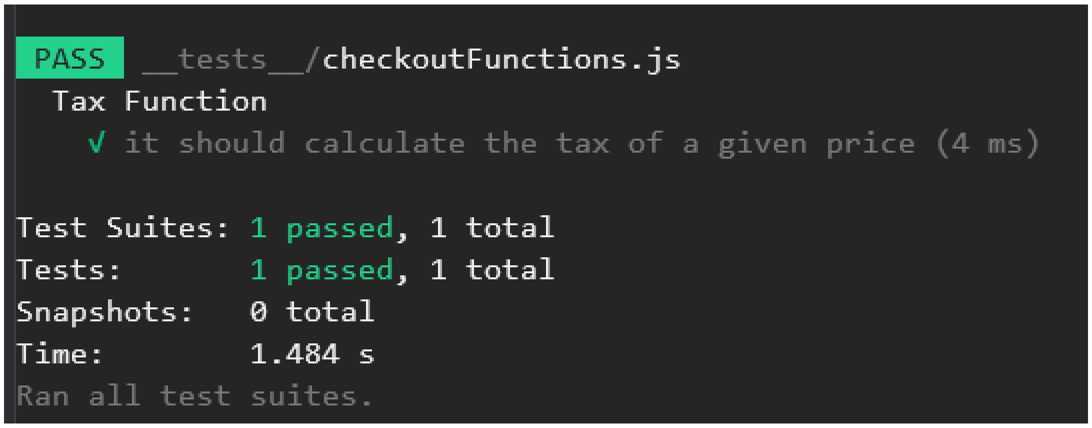
F. 1. Unit Testing with Jest

A main component of this project was the use of javascript to determine all areas of cost when the user places an order. These values were calculated within the payment.js file. Tests were performed using Jest and Node.js to determine if these functions were producing correct results. This specific area of testing used Unit Testing since each function was being tested independent of the other functions.

F.1.1. Testing the Calculation of the Tax Value

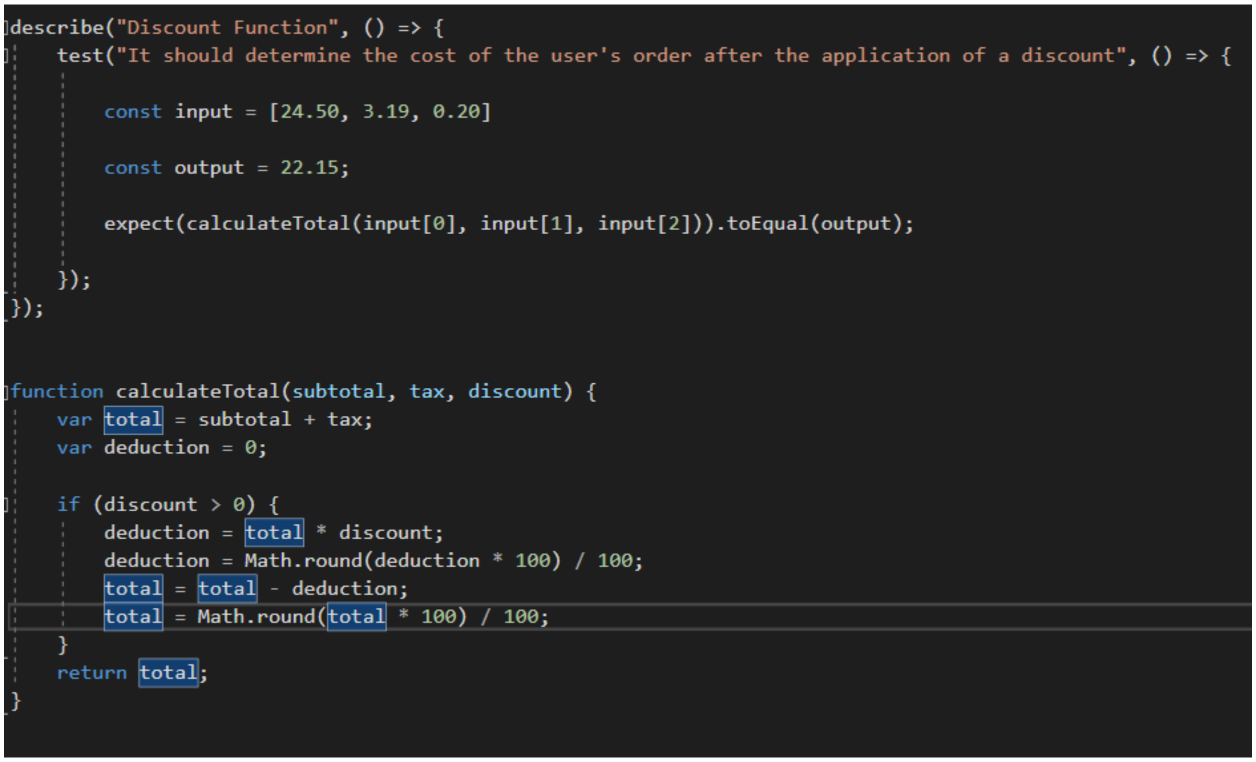


```
describe("Tax Function", () => {
    test("It should calculate the tax applied to the user's order", () => {

        const input = 24.50;

        const output = 3.19;

        expect(calculateTax(input)).toEqual(output);

    });
});


function calculateTax(subtotal) {
    const taxRate = 0.13;
    var tax = subtotal * taxRate;
    tax = Math.round(tax * 100) / 100;
    return (tax);
}
```

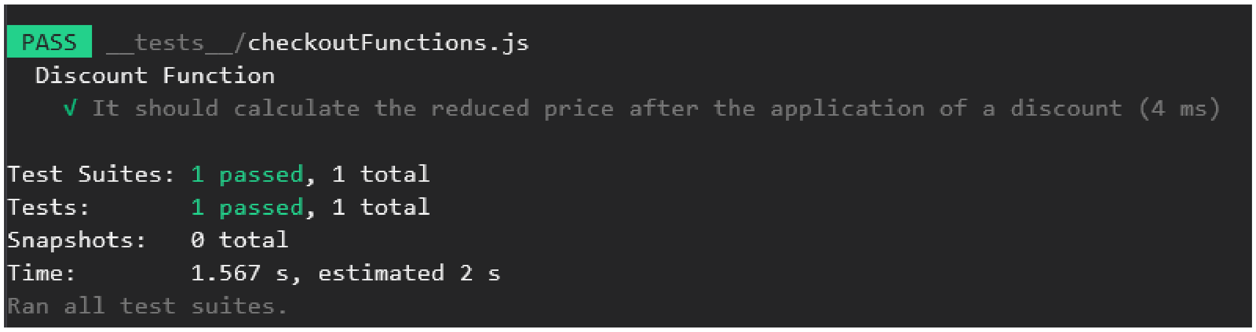


```
 PASS  __tests__/checkoutFunctions.js
  Tax Function
    √ it should calculate the tax of a given price (4 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.484 s
Ran all test suites.
```

F.1.2. Testing the Discount Function



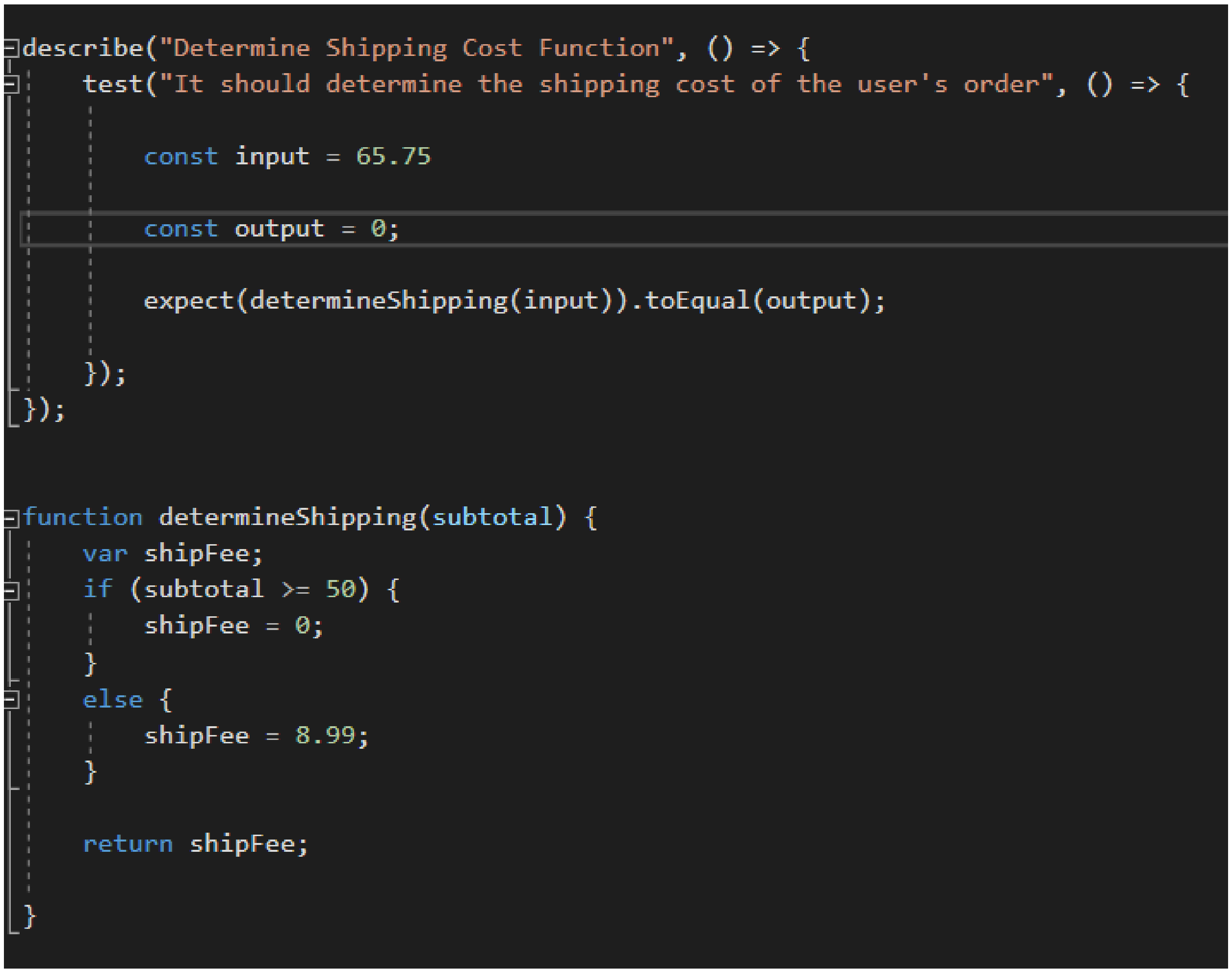
```
describe("Discount Function", () => {
    test("It should determine the cost of the user's order after the application of a discount", () => {

        const input = [24.50, 3.19, 0.20]

        const output = 22.15;

        expect(calculateTotal(input[0], input[1], input[2])).toEqual(output);

    });
});


function calculateTotal(subtotal, tax, discount) {
    var total = subtotal + tax;
    var deduction = 0;

    if (discount > 0) {
        deduction = total * discount;
        deduction = Math.round(deduction * 100) / 100;
        total = total - deduction;
        total = Math.round(total * 100) / 100;
    }
    return total;
}
```



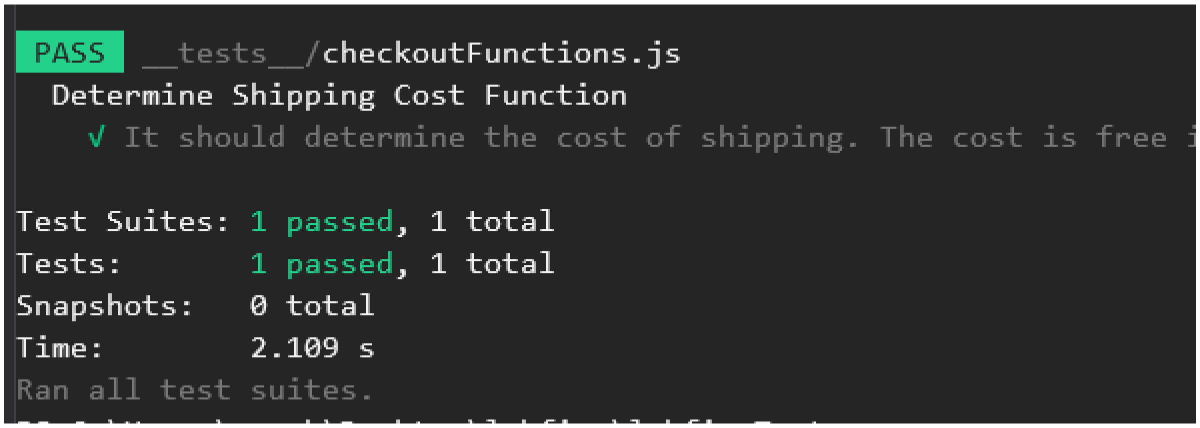
```
PASS  __tests__/checkoutFunctions.js
  Discount Function
    √ It should calculate the reduced price after the application of a discount (4 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.567 s, estimated 2 s
Ran all test suites.
```

F.1.3. Testing the Determination of Shipping Cost Function

For this function, the user should receive a free shipping cost whenever their subtotal is greater than or equal to fifty dollars and pay eight dollars and ninety-nine cents otherwise.

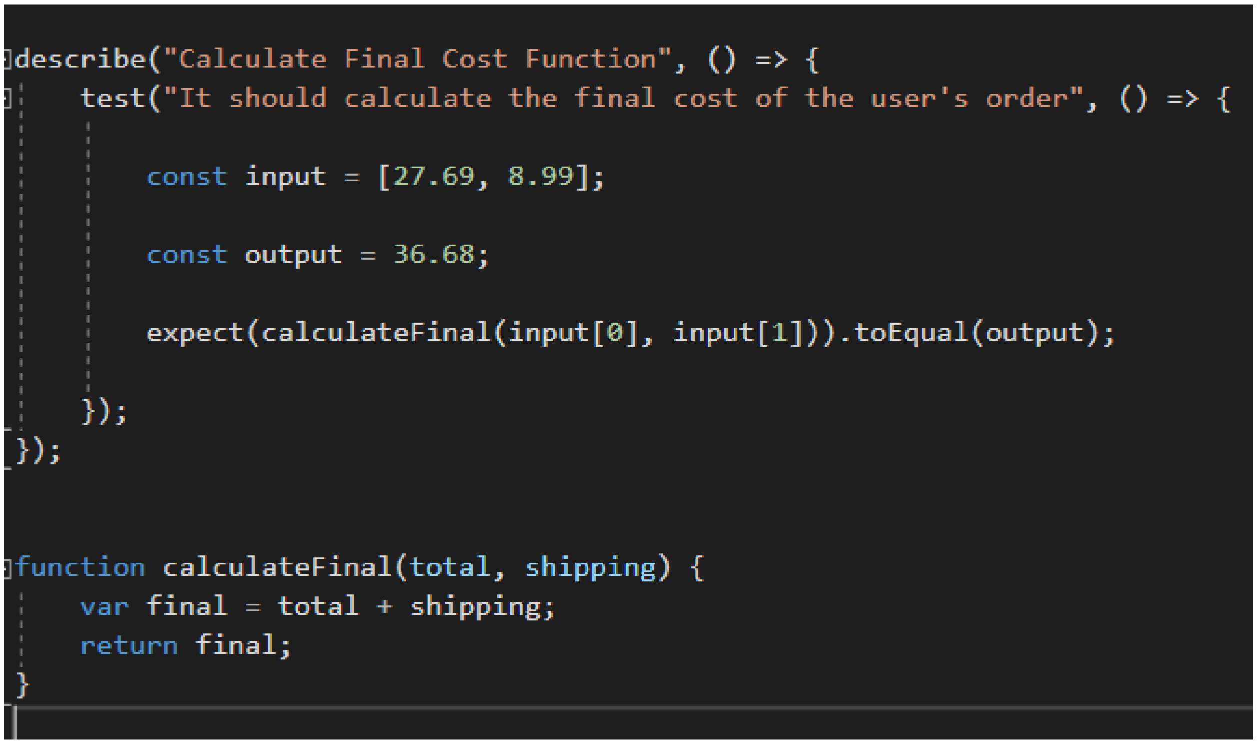


```
describe("Determine Shipping Cost Function", () => {
    test("It should determine the shipping cost of the user's order", () => {

        const input = 65.75

        const output = 0;

        expect(determineShipping(input)).toEqual(output);

    });
});


function determineShipping(subtotal) {
    var shipFee;
    if (subtotal >= 50) {
        shipFee = 0;
    }
    else {
        shipFee = 8.99;
    }

    return shipFee;

}
```

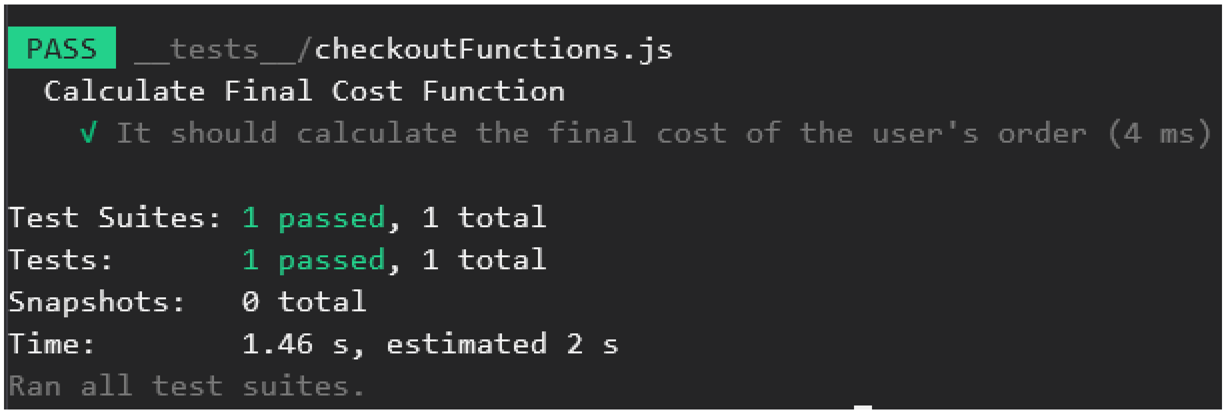


```
PASS  __tests__/checkoutFunctions.js
  Determine Shipping Cost Function
    √ It should determine the cost of shipping. The cost is free i

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.109 s
Ran all test suites.
```

F.1.4. Testing the Calculation of Final Cost Function



```
describe("Calculate Final Cost Function", () => {
    test("It should calculate the final cost of the user's order", () => {

        const input = [27.69, 8.99];

        const output = 36.68;

        expect(calculateFinal(input[0], input[1])).toEqual(output);

    });
});


function calculateFinal(total, shipping) {
    var final = total + shipping;
    return final;
}
```

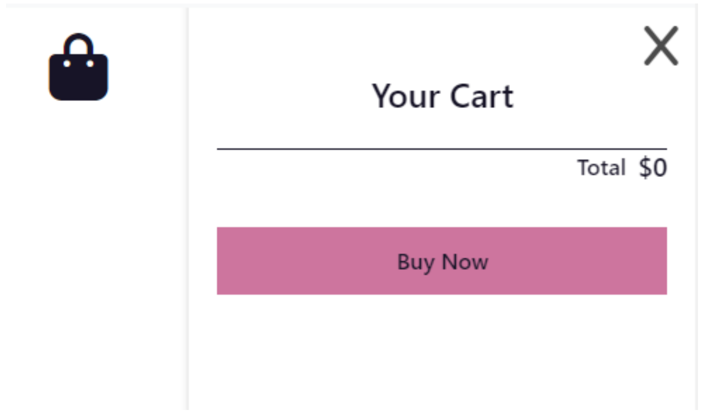


```
 PASS  __tests__/checkoutFunctions.js
  Calculate Final Cost Function
    √ It should calculate the final cost of the user's order (4 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.46 s, estimated 2 s
Ran all test suites.
```

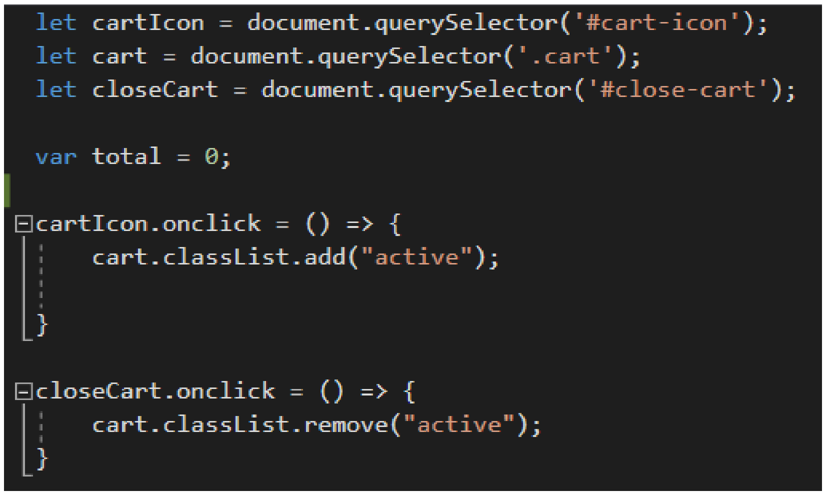
F. 2. Manual Integration Testing

Testing the project through Integration was done manually throughout the development of the website. Many components of the website relied on values obtained from other components through the use of functions. The components were tested by determining the expected results of the main interactions within the website and checking if the code could produce them.

F.2.1. Functionality of Cart



The cart section of the web page uses javascript to show or hide its contents. When the "X" icon is clicked, the cart will not be shown on the page. When the shopping bag icon is clicked, the cart will merge back into view. The code which controls the functionality of the cart is in the main.js file.



```
let cartIcon = document.querySelector('#cart-icon');
let cart = document.querySelector('.cart');
let closeCart = document.querySelector('#close-cart');

var total = 0;

cartIcon.onclick = () => {
    cart.classList.add("active");

}

closeCart.onclick = () => {
    cart.classList.remove("active");
}
```

The cart icon and close cart icons are identified from the cart.html file with the use of the querySelector function.

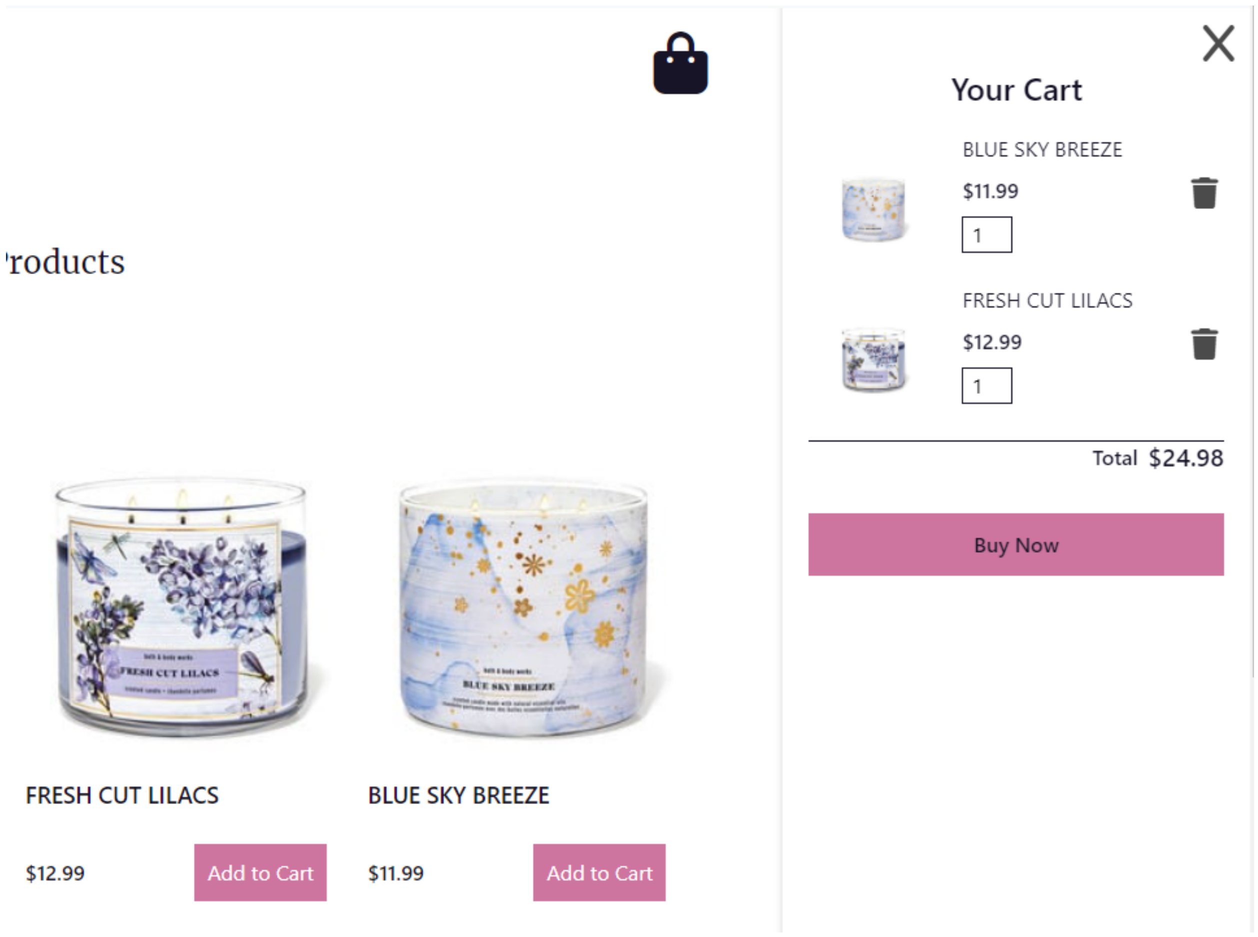
```html
<i class="fa-solid fa-bag-shopping" id="cart-icon"></i>
```

```html
<i class="fa-solid fa-x" id="close-cart"></i>
```

As seen in figure above, when the cart icon is clicked, the cart class in the html file will be put into an active state. The opposite occurs when the close cart icon is clicked, since the cart class will be inactive.

These functions were tested by interacting with the web page and confirming that the cart opens or closes depending on which icon is clicked.

F.2.2. Functionality of Cart with Products List



'roducts

FRESH CUT LILACS
$12.99
Add to Cart

BLUE SKY BREEZE
$11.99
Add to Cart

Your Cart

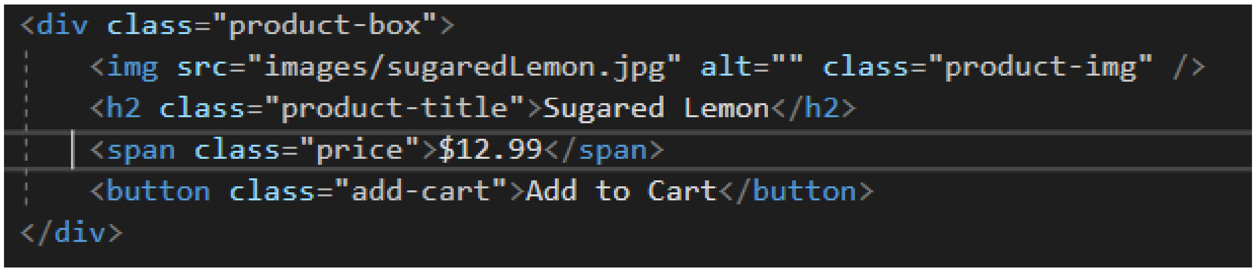
BLUE SKY BREEZE
$11.99
1

FRESH CUT LILACS
$12.99
1

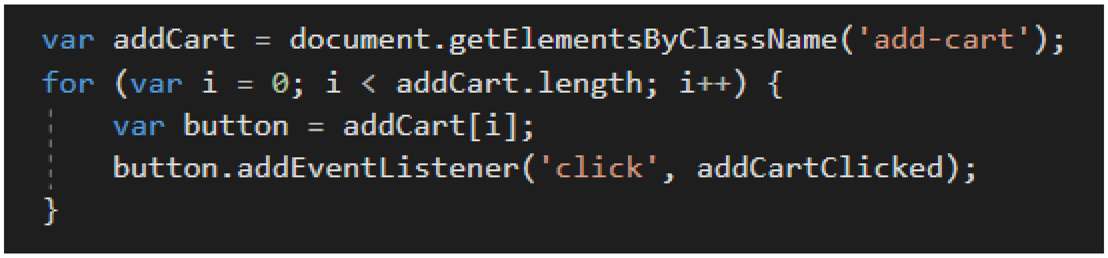
Total $24.98

Buy Now

The cart also needs to interact with the listed products. Whenever the “Add to Cart” button for a specific product is clicked, that product should appear as a new item in the cart. Each product is created in a divider element and given a title, price, image, and “Add to Cart” button.

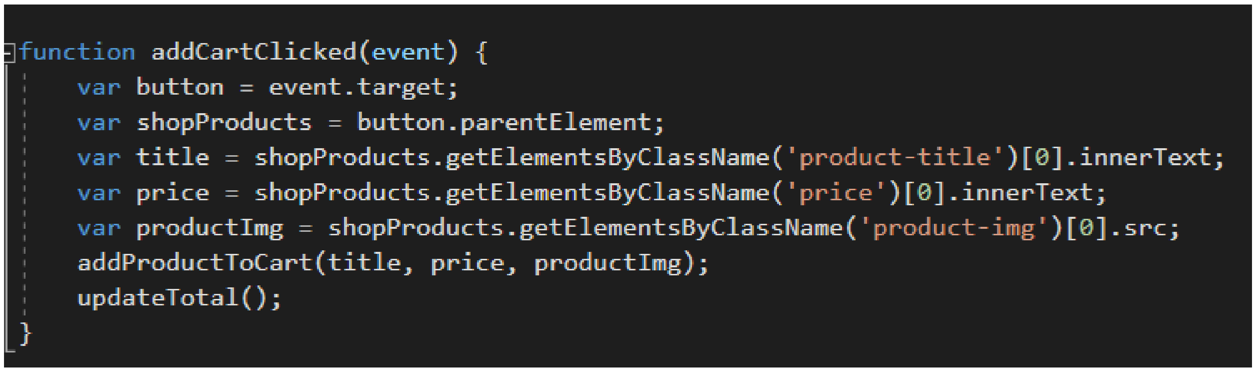


```html
<div class="product-box">
    <img src="images/sugaredLemon.jpg" alt="" class="product-img" />
    <h2 class="product-title">Sugared Lemon</h2>
    <span class="price">$12.99</span>
    <button class="add-cart">Add to Cart</button>
</div>
```



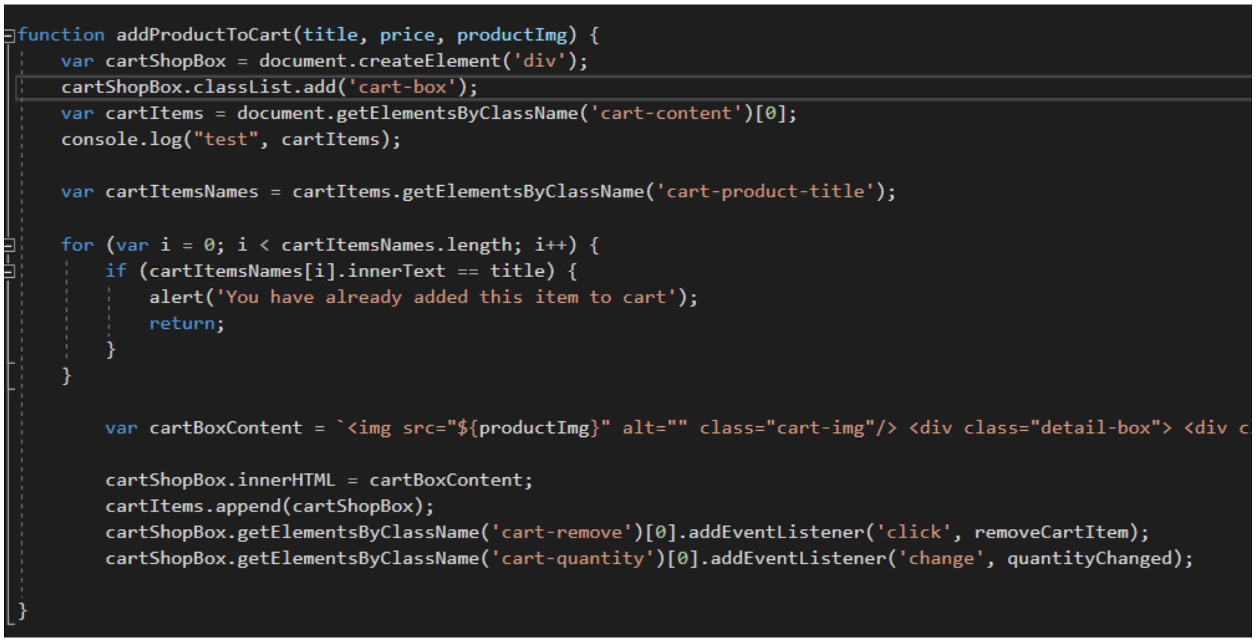
```javascript
var addCart = document.getElementsByClassName('add-cart');
for (var i = 0; i < addCart.length; i++) {
    var button = addCart[i];
    button.addEventListener('click', addCartClicked);
}
```

When the button is clicked for any item, an event listener is activated and the addCartClicked method is called.



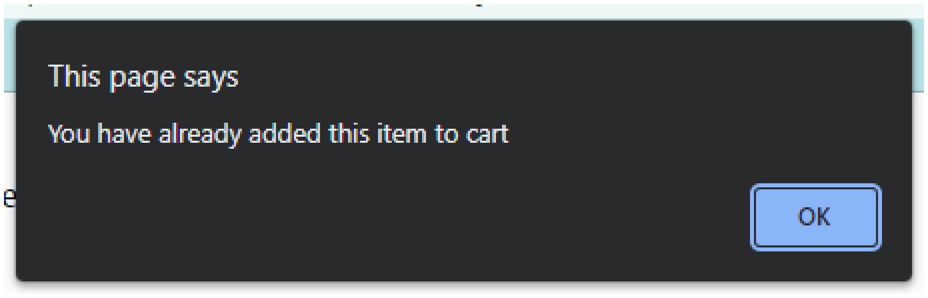
```javascript
function addCartClicked(event) {
    var button = event.target;
    var shopProducts = button.parentElement;
    var title = shopProducts.getElementsByClassName('product-title')[0].innerText;
    var price = shopProducts.getElementsByClassName('price')[0].innerText;
    var productImg = shopProducts.getElementsByClassName('product-img')[0].src;
    addProductToCart(title, price, productImg);
    updateTotal();
}
```

The addCartClicked method determines which product triggered the event and obtains its corresponding title, price, and image. It then calls the addProductToCart method to display the product in the cart section of the page. The updateTotal method is called to change the cart total to include the price of the newly added product.



```
function addProductToCart(title, price, productImg) {
    var cartShopBox = document.createElement('div');
    cartShopBox.classList.add('cart-box');
    var cartItems = document.getElementsByClassName('cart-content')[0];
    console.log("test", cartItems);

    var cartItemsNames = cartItems.getElementsByClassName('cart-product-title');

    for (var i = 0; i < cartItemsNames.length; i++) {
        if (cartItemsNames[i].innerText == title) {
            alert('You have already added this item to cart');
            return;
        }
    }

        var cartBoxContent = `<img src="${productImg}" alt="" class="cart-img"/> <div class="detail-box"> <div c

        cartShopBox.innerHTML = cartBoxContent;
        cartItems.append(cartShopBox);
        cartShopBox.getElementsByClassName('cart-remove')[0].addEventListener('click', removeCartItem);
        cartShopBox.getElementsByClassName('cart-quantity')[0].addEventListener('change', quantityChanged);

}
```

The addProductToCart function creates a new divider element in the cart.html file and tracks items in the cart through the use of array. If the current product is already in the cart, then an alert is sent to the user that they have already added the item to their cart. Then the product details are written to the html page using the innerHTML function. A quantity select element and remove icon are also added to the cart to allow the user to select the amount of items or to remove items.



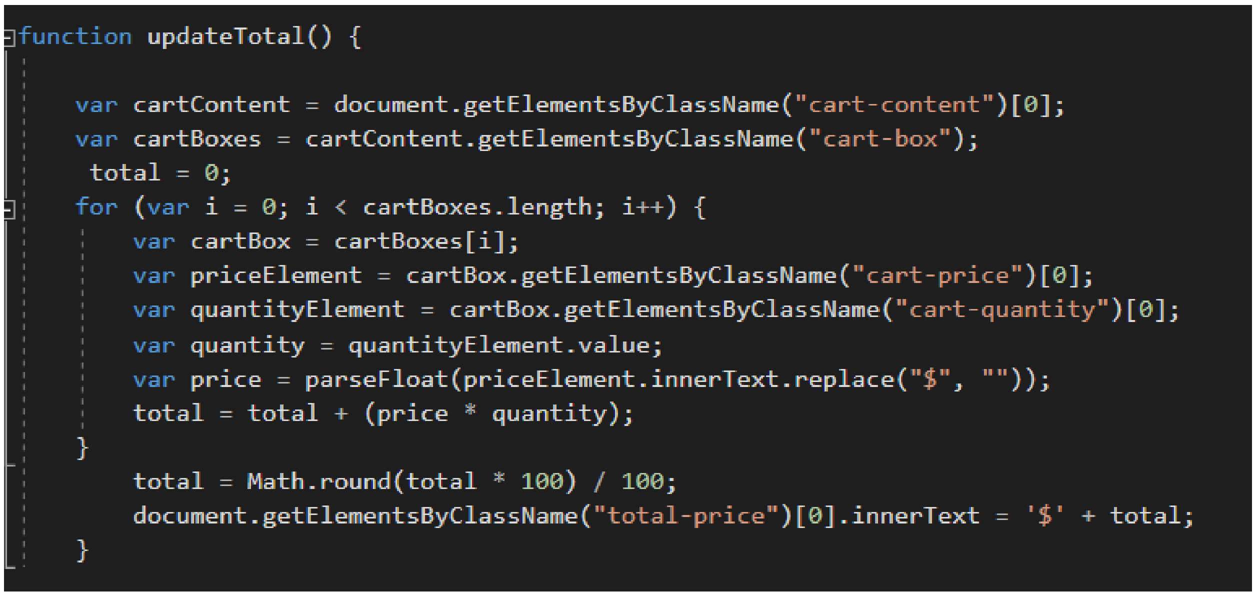
This page says

You have already added this item to cart

This requirement was texted by selecting the "Add to Cart" button of various products and seeing if they appear in the cart. The remove icon was tested to see if it removes the product divider element from the cart when clicked. The action of adding, removing, or changing the quantity of an element in the cart was also tested with the updateTotal function to ensure the cart total responds to changes.

F.2.3. Response of Total Price to Changes

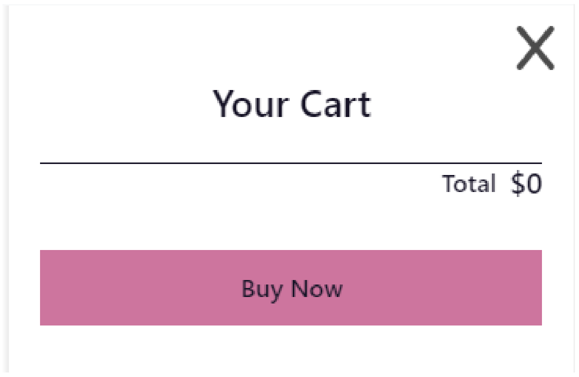
The updateTotal method is used to update the cart total whenever a change is made to its contents.

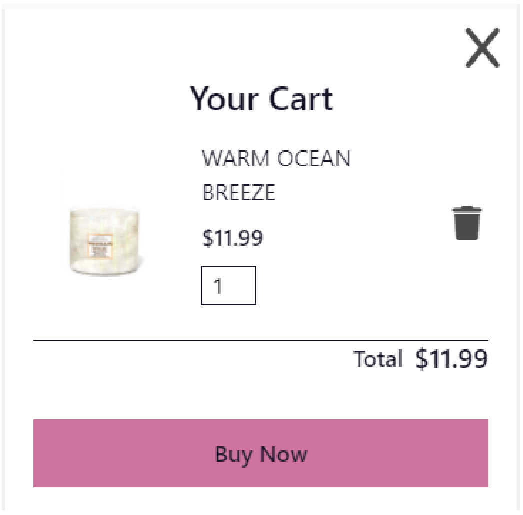


```
function updateTotal() {

    var cartContent = document.getElementsByClassName("cart-content")[0];
    var cartBoxes = cartContent.getElementsByClassName("cart-box");
     total = 0;
    for (var i = 0; i < cartBoxes.length; i++) {
        var cartBox = cartBoxes[i];
        var priceElement = cartBox.getElementsByClassName("cart-price")[0];
        var quantityElement = cartBox.getElementsByClassName("cart-quantity")[0];
        var quantity = quantityElement.value;
        var price = parseFloat(priceElement.innerText.replace("$", ""));
        total = total + (price * quantity);
    }
        total = Math.round(total * 100) / 100;
        document.getElementsByClassName("total-price")[0].innerText = '$' + total;
    }
```

Three tests were performed to determine the cart was functioning correctly.

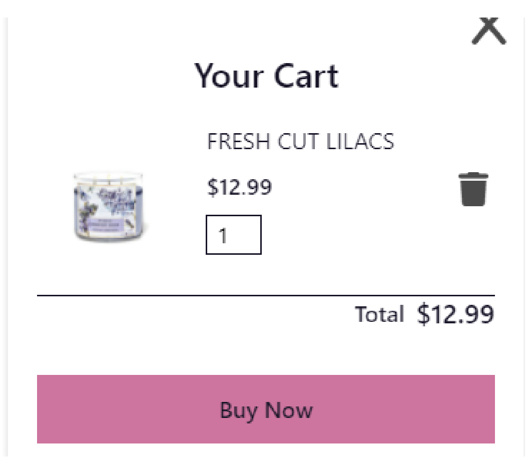
Test 1: Addition of Item





Adding an item to the cart should update the total price to include the price of the item.

Test 2: Quantity Change



X

Your Cart

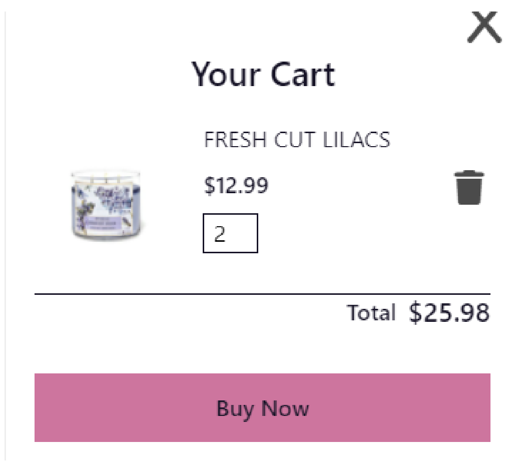
FRESH CUT LILACS

$12.99

1

Total $12.99

Buy Now



X

Your Cart

FRESH CUT LILACS

$12.99

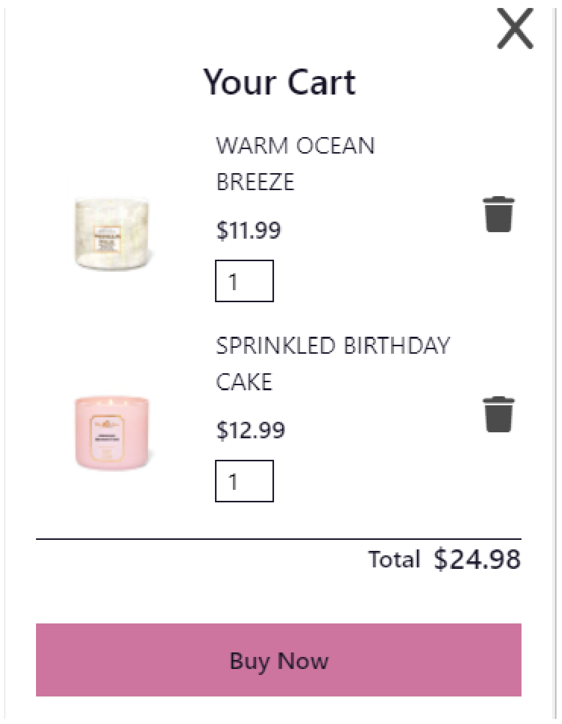
2

Total $25.98

Buy Now

A change in the quantity of an item should update the total price.

Test 3: Removal of Item



X

Your Cart

WARM OCEAN BREEZE

$11.99

1

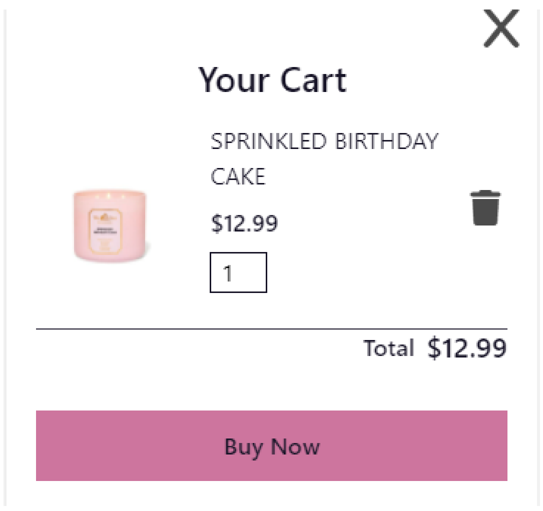
SPRINKLED BIRTHDAY CAKE

$12.99

1

Total $24.98

Buy Now



X

Your Cart

SPRINKLED BIRTHDAY CAKE

$12.99

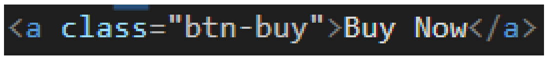
1

Total $12.99

Buy Now

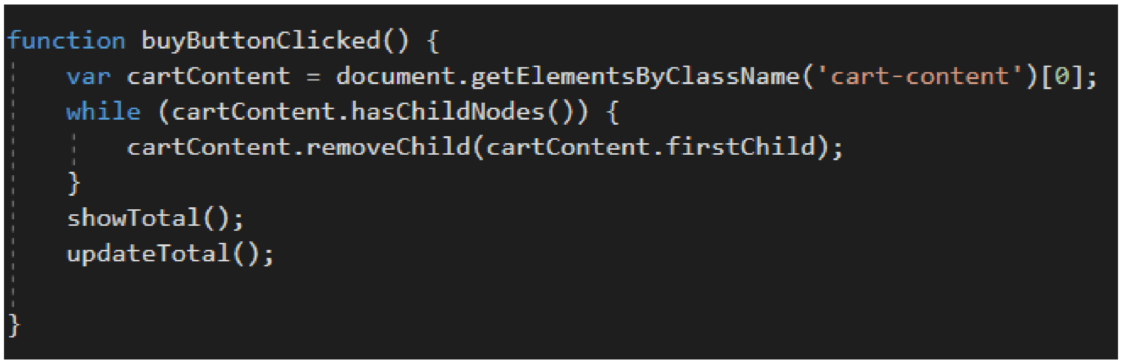
Removal of an item should deduct that item's cost from the total cost of the cart.

F.2.4. Functionality of the Buy Now Button

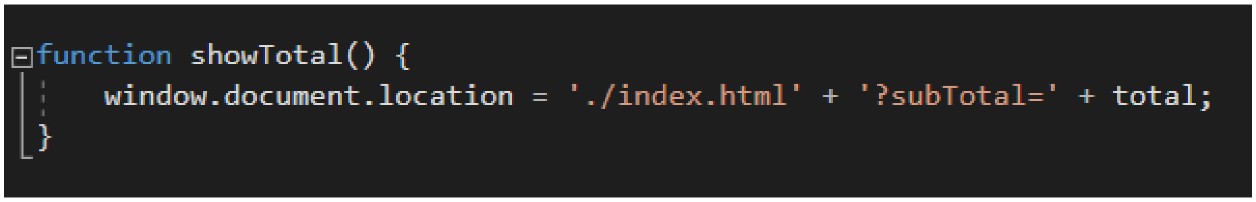


```
<a class="btn-buy">Buy Now</a>
```

When selected, the “Buy Now” button triggers an event which calls the buyButtonClicked method. This method calls the showTotal method to redirect the user to the checkout page.

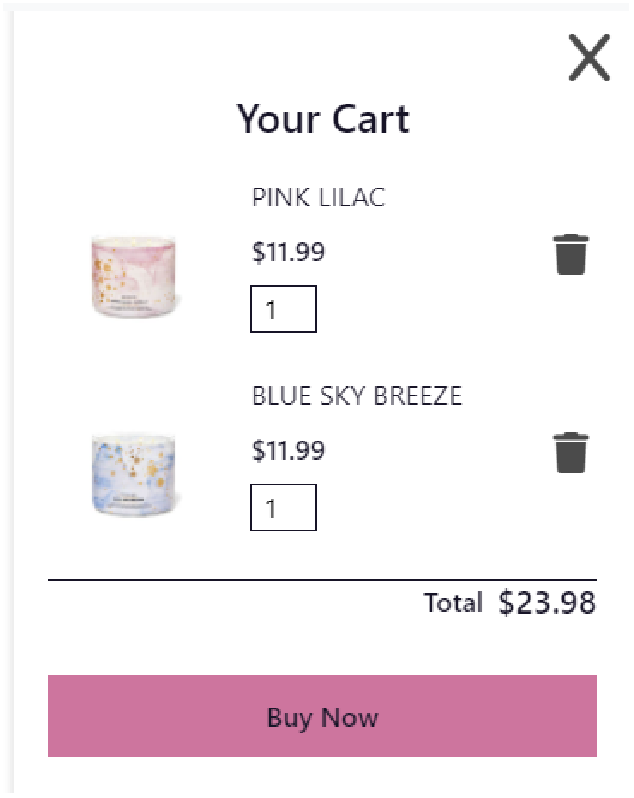


```
function buyButtonClicked() {
    var cartContent = document.getElementsByClassName('cart-content')[0];
    while (cartContent.hasChildNodes()) {
        cartContent.removeChild(cartContent.firstChild);
    }
    showTotal();
    updateTotal();

}
```



```
function showTotal() {
    window.document.location = './index.html' + '?subTotal=' + total;
}
```

The showTotal method also uses a query string to send the subtotal of the cart to the checkout page so the total cost of the order can be calculated.


Your Cart

PINK LILAC

$11.99

1

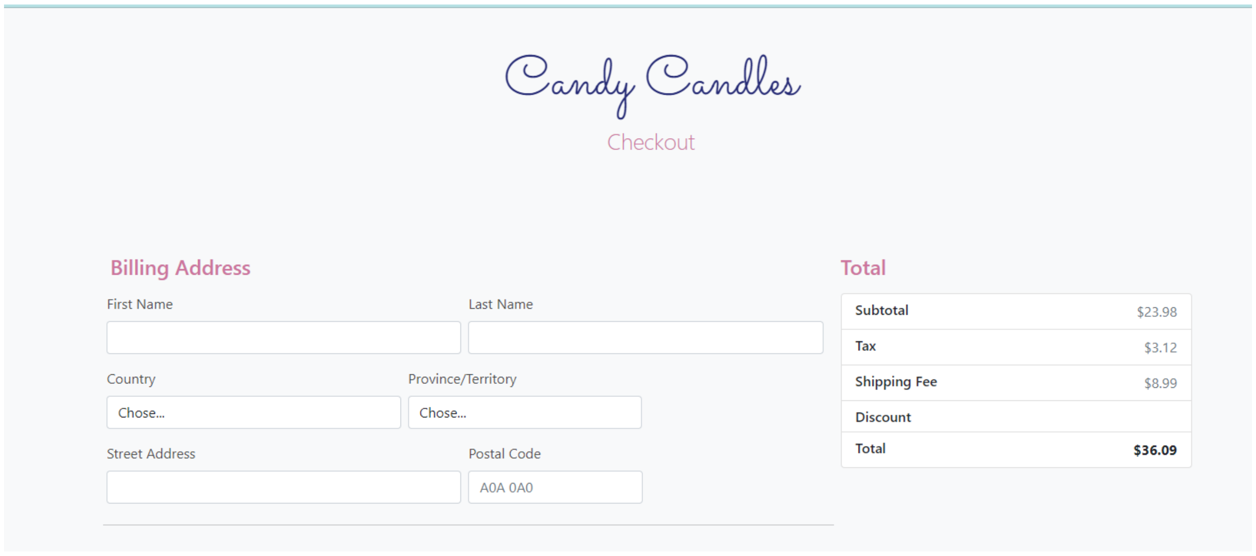
BLUE SKY BREEZE

$11.99

1

Total $23.98

The cost of the above order is $23.98. Once the “Buy Now” button is clicked, this data is sent to the checkout page below.


Candy Candles

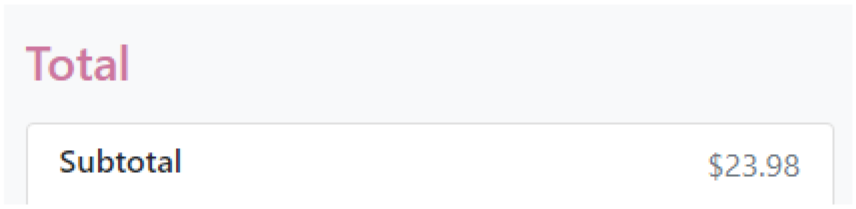
Checkout

Billing Address

First Name	Last Name
Country: Chose...	Province/Territory: Chose...
Street Address	Postal Code: A0A 0A0

Total

Subtotal	$23.98
Tax	$3.12
Shipping Fee	$8.99
Discount	
Total	$36.09



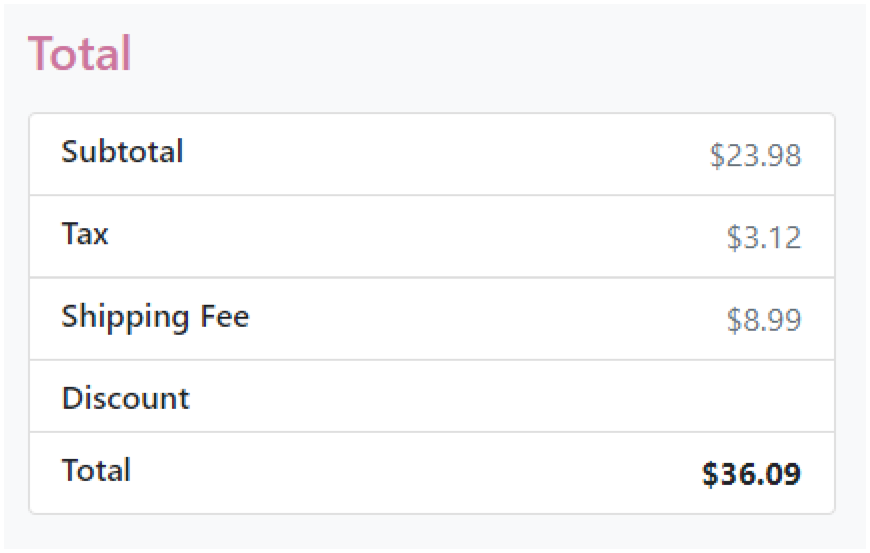
Total

Subtotal	$23.98

As seen above, the buy button directed the user to the checkout page where the subtotal is displayed. This subtotal matches the total cost of the user's order in the cart, which confirms that the website is working correctly.

F.2.5. Application of Discount

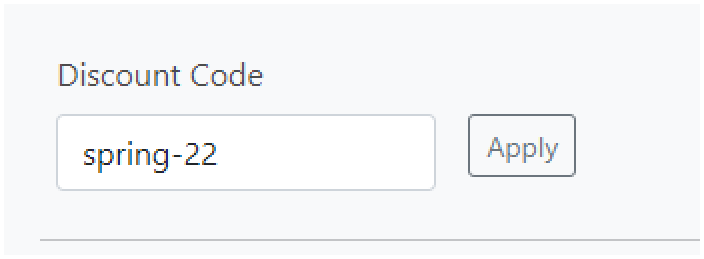
According to the previous unit tests, the cart accurately produces the subtotal, tax, and shipping fee with help from the payment.js file.



Total

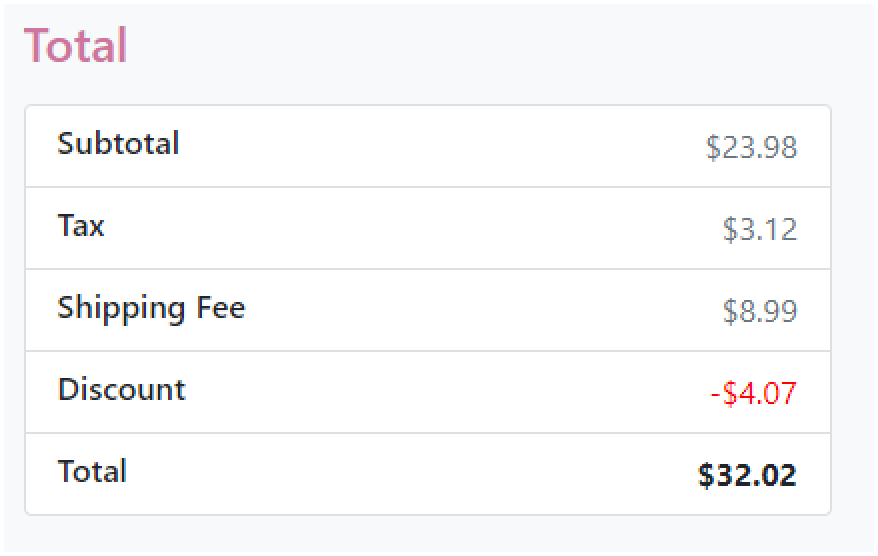
Subtotal	$23.98
Tax	$3.12
Shipping Fee	$8.99
Discount	
Total	**$36.09**

The discount is not applied until the user enters a discount code. When an accurate discount code is entered, the discount will be calculated, displayed, and deducted from the total cost.



Discount Code

spring-22

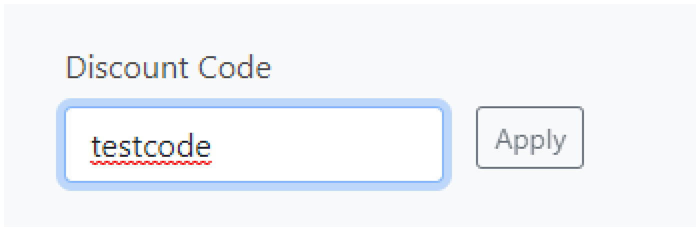


Total

Subtotal	$23.98
Tax	$3.12
Shipping Fee	$8.99
Discount	-$4.07
Total	**$32.02**

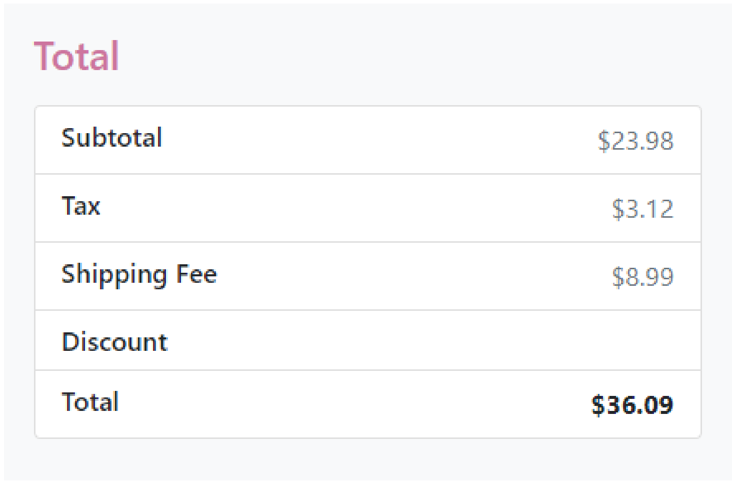
As seen above, a valid discount code, "spring-22" was entered. This triggered a discount value to be calculated from the subtotal. The discount applied was 15% and this value is stored within the javascript file. The value deducted from the subtotal was $4.07. The total is updated to reflect this.

If an incorrect code is entered, any previous discount is removed and the total cost is updated.



Discount Code

testcode

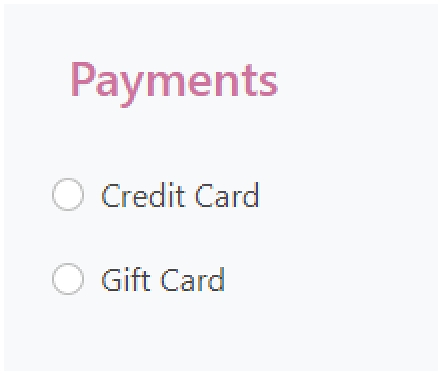


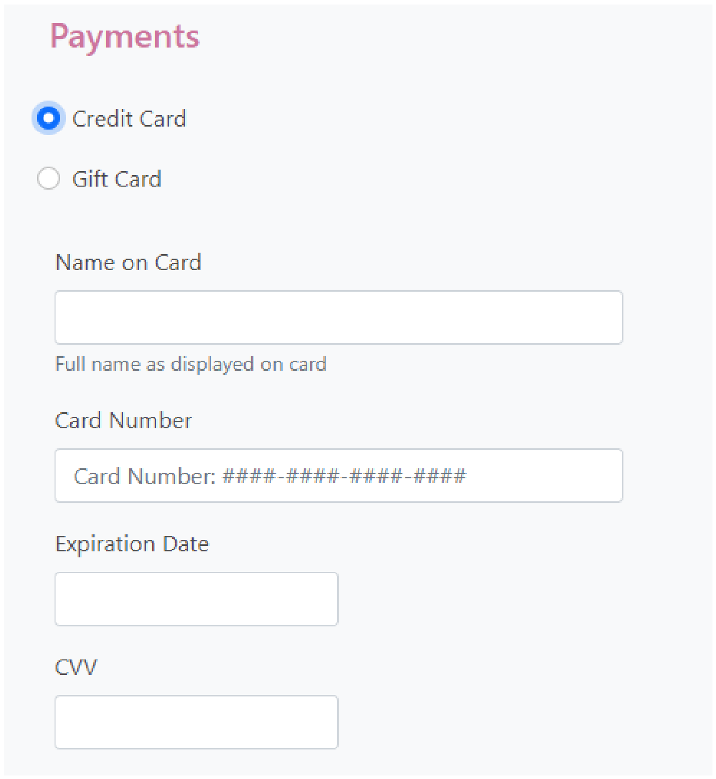
Total

Subtotal	$23.98
Tax	$3.12
Shipping Fee	$8.99
Discount	
Total	**$36.09**

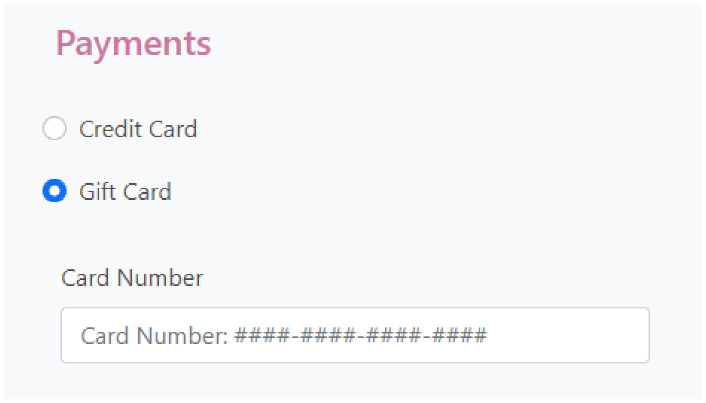
F.2.6. Selection of Payment Method

By default, no payment will be selected. If the user clicks the Credit Card radio button, various inputs corresponding to the credit card details will appear. On the other hand, if the user decides to select Gift Card, they will only be asked to enter the card number.





In the above image, the user selected Credit Card. The website now allows the user to enter in their credit card information.



Payments

Credit Card
[✓ Gift Card]

Card Number

Card Number: ####-####-####-####

Here the user selected the gift card option. Now the website allows the user to enter in the card number on their gift card.

F. 3. Acceptance Testing

The design process for the Candy Candles shopping system began by outlining use cases between the end user and the system. Requirements for the system were developed from the needs of the determined use cases. The code was created to reflect each of these requirements.

All of the previously mentioned tests confirm the functionality of the main components of the website, which indicates that the requirements of the user and customer were met. The website displays products, has a working cart, a checkout system, application of discount codes, options for payment method, and gathers all necessary information to charge the user and ship the order. The payment itself will be processed through the user's bank with use of the entered account information. The shipping details entered on the page as well as the cart contents will be put on file in the warehouse database so the orders can be shipped. Finally all of the information surrounding the purchase will be placed into the system database so any returns can be easily processed.

Various users were given the opportunity to test the application to confirm it met their needs. Also, the client tested the page to approve that the design fulfills their requirements.