**E-Commerce Web Application**

This web application titled ***Shopee*** uses vanilla Javascipt (no additional JS libraries) to create a dynamic e-commerce mock shopping experience.

For styling, in combination with custom CSS, it uses the popular styling library **Bootstrap** to style and structure the application visually.

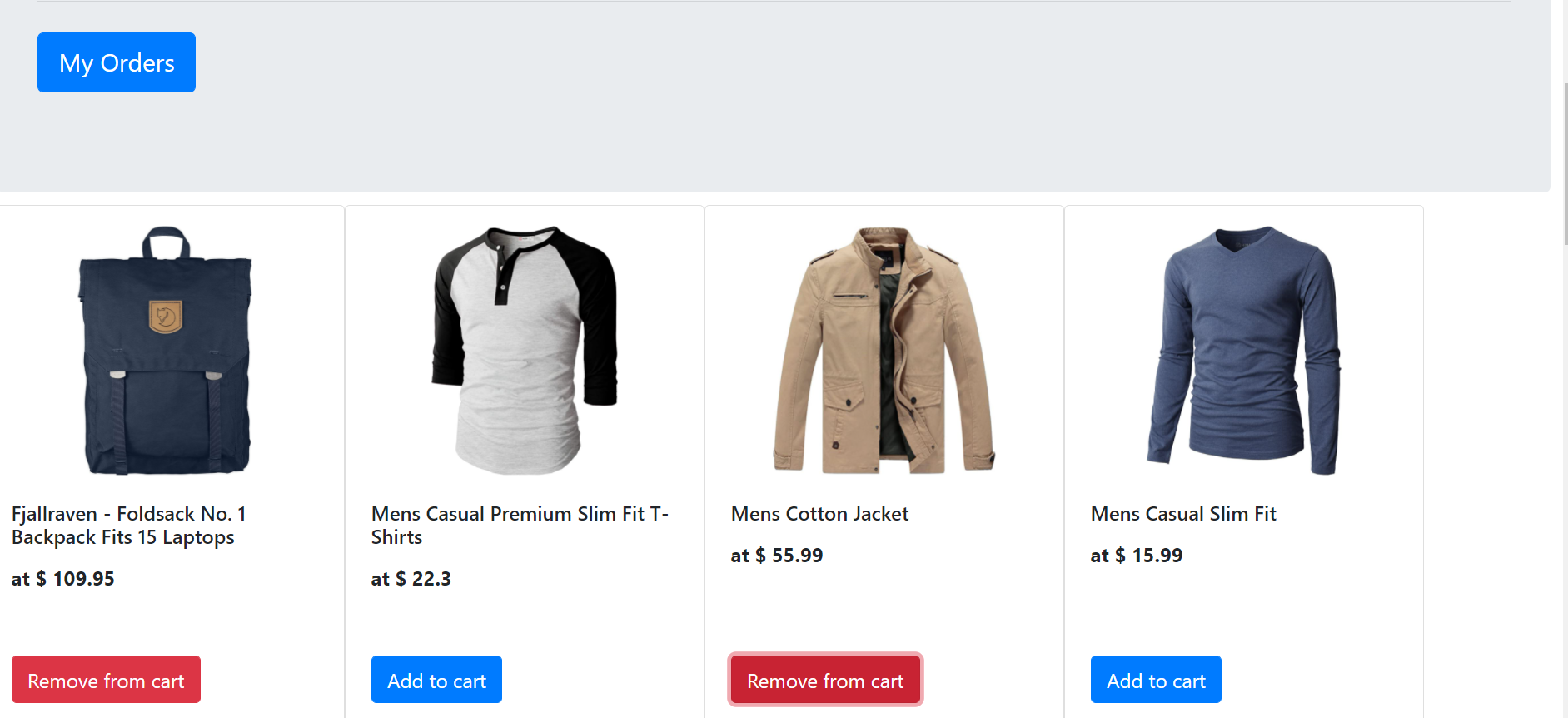
For file and folder structure and relations of the application, each *.html* file is mapped to a *.js* file, each with it’s own pertaining Javascript to manipulate the DOM to create a dynamic user experience. With regards to CSS, all *.html* files share the same *main.css* file as their styling source.

**UI Overview**

There are three pages within the application:

* Commodities page
* Order page
* Payment page

**Commodities page**

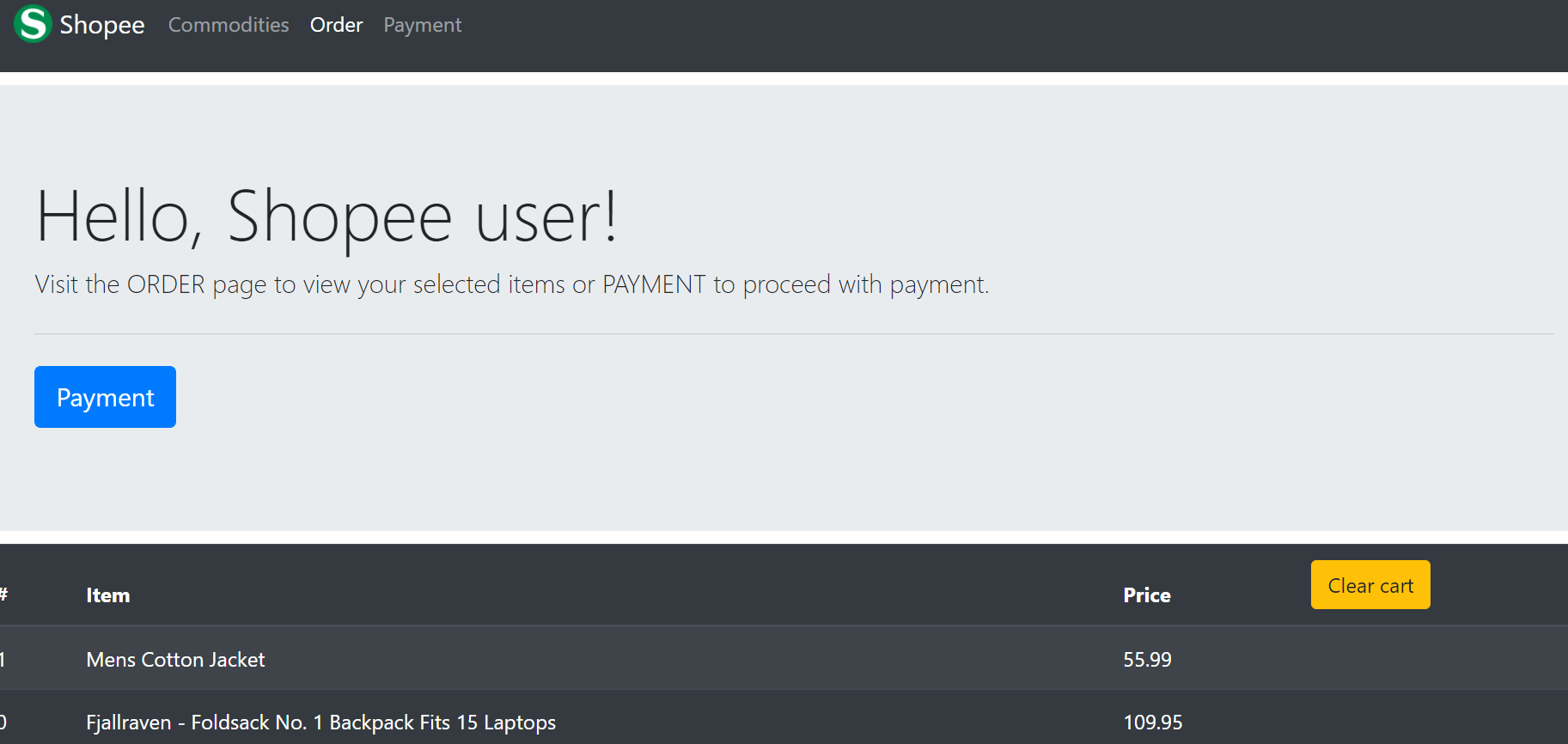


This page, **commo.html** is the entry point of the application.

It displays the commodities that are available for buying or adding to the user’s cart.

Each item has a “Add to Cart” which adds the item to the cart. After an item is added, the button will change to show “Remove to Cart” instead, which when pressed will remove the item from cart.

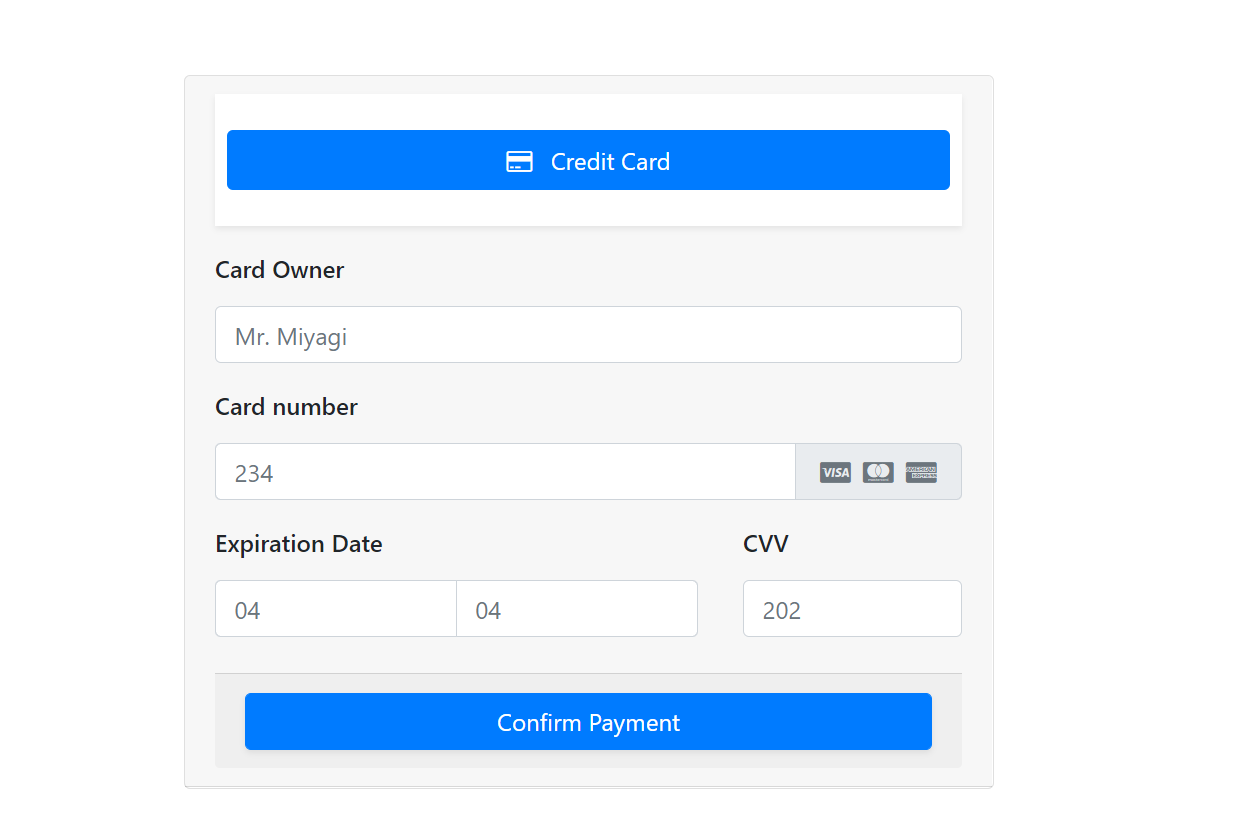
**Order page**



This is the page where the user can view what items they have added to the application.

Here they have the option to clear the cart entirely and start over again.

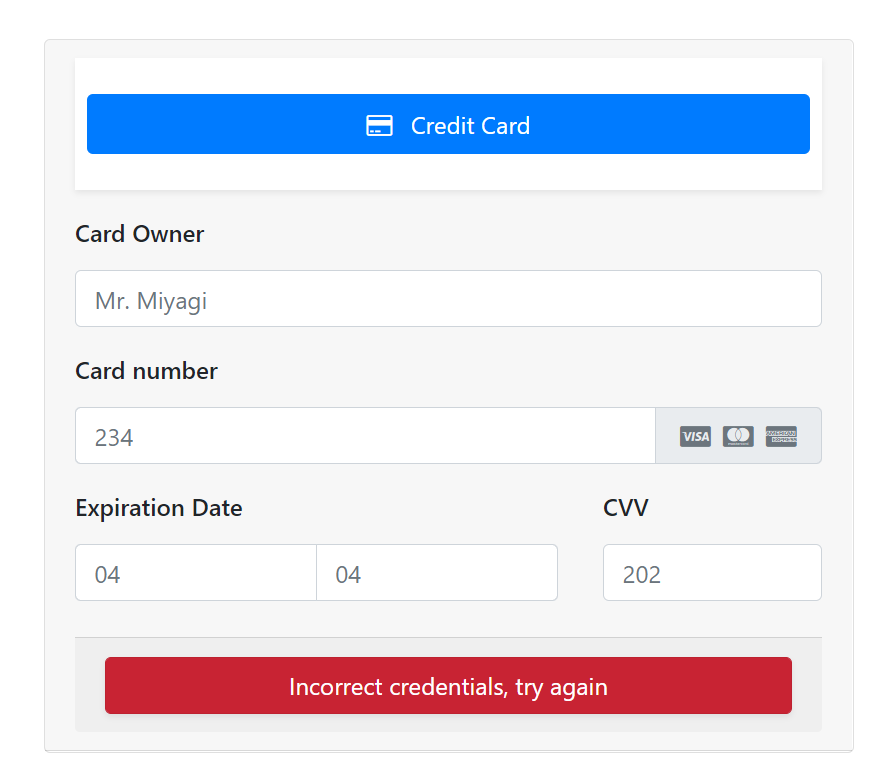
**Payment page**



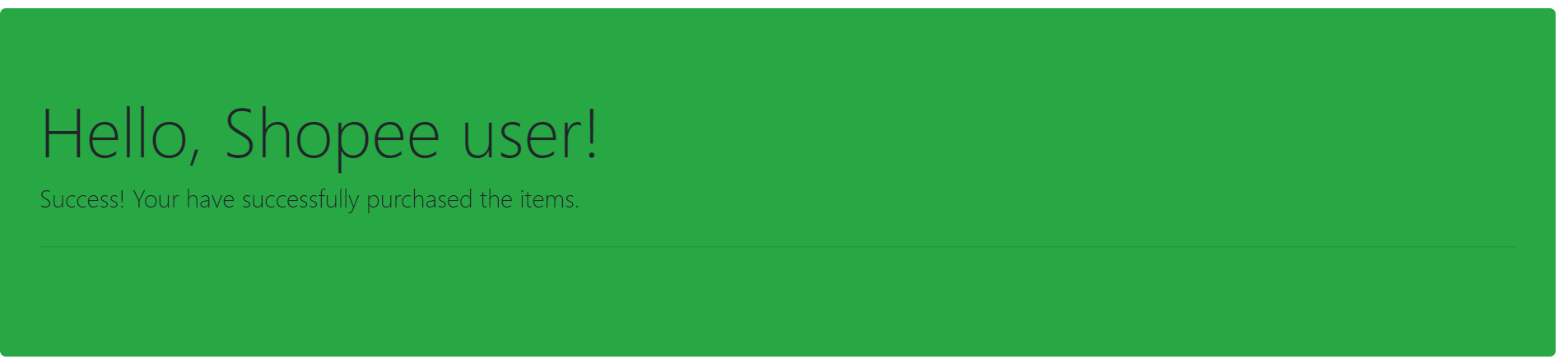
The payment page asks for pertinent information to proceed with paying for the items bought.

To test the feature, one can enter the placeholder information shown in the form or the test credentials can be found in the file **mockCredentials.txt** as well.

If wrong information or correct information is entered, appropriate response and visual representation is shown for each situation.



Or, when correct –



**Usage of Javascript**

* **Data.js**
  + This file is responsible for pulling in data from a mock e-commerce REST API.
* **Commo.js**
  + addCommodity function
    - This function takes in parameters: imageSource, title, price and adds a single commodity to the page.
    - If the commodity is in the cart already, it will change the button from “Add to Cart” to “Remove from Cart” appropriately.
  + displayCommodities function
    - This function calls upon addCommodity function to add all the commodities that were brought in by the **data.js** file
  + addToCart function
    - This function is an onclick event handler for when the user presses the “Add to Cart” button
    - It uses window.localStorage to store the cart information so that even when the user exits the browser, their cart items will still be retained
  + removeFromCart function
    - onclick event handler for when the user presses the “Remove from Cart” button
* **Order.js**
  + displayCart function
    - Displays all the information stored in the cart by access window.localStorage
  + clearCart function
    - Clears all the item in the cart so that the user can start over again
* **Payment.js**
  + pay function
    - This function checks whether the credentials submitted by the user match with the mock credentials in store. Either way, shows an appropriate visual representation to the user.

**Flow Charts and Diagrams**



Commo.js page uses the data gathered by Data.js to dynamically display the data in commo.html page



MOCK E-commerce REST API

**Figure 1**



User adds an item to the cart

addToCart function is called and dynamic UI changes as well as data layer changes are done.

The cart data is stored in window.localStorage so that shopping can be resumed whenever the user returns

localStorage

CART information stored as a string

User removes an item to the cart

removeFromCart function is called and dynamic UI changes as well as data layer changes are done.

**Figure 2**

Mock REST AhPI

Commo.html

User adds/remove items from cart

Commo.js

localStorage – for cart info

Payment.js

Order.js

Payment.html

Order.html

Data.js