CS 457 group 3 project
Josiah Conely, Daniel Oliveros Guerra, Dalton Gilmore

The brief of this project was to build a webstore that could be used by shops and stores who desire to have a web presence. For this project, group 3 decided to build the front end of a standard client-server architecture with a dynamically generated front end. We chose this architecture because we wanted to have a flexible front end that would be populated by a simple API call, as opposed to hard coding the products and product categories.

Store:

Our store is a basic drug store, like a CVS, and has all the same products. Our product categories include food, drink, medicine, cosmetic, toys, candy, snacks, and hygiene products. Our store offers users the ability to browse, select and purchase the products with a standard shopping cart tool.

Architecture:

The architecture of this web app is that of a basic client-server, with an API relationship between the two. The frontend is dynamically generated by the front end, based on the data that it receives from an API call. For the purposes of this demonstration, we did hardcode a JSON object in the file products.js, however, the JSON object could be easily replaced with an HTTP get request as soon as the back end is completed. We chose this architecture because we wanted to be able to easily change the products/categories listed on the site, without having to change any code. As long as the database that holds the products is updated correctly, the front end will be able to display the product correctly. As such there is no need to keep track of positional correspondence across multiple arrays when adding or removing categories as is the case in the storeN.pl file that was supplied to us. The separation of concerns between data on the back end, and rendering on the front end, offers a number of benefits that include, ease of maintenance, code clarity, and the ability to change products/categories quickly.

CS 457 group 3 project
Josiah Conely, Daniel Oliveros Guerra, Dalton Gilmore

Layout of Store:

1. On the top of the page there is a basic header with our store name.
2. On the left hand side, there is an element that allows users to select the category that they want to see.
3. Below the header is a “product in focus” element that displays details about a product that the user wants to view and includes an add-to-cart button.
4. Below the “product in focus” there is the “products display” element that displays all the products in a specific category.
5. On the right side of the page, there is a shopping cart element that displays the products in the cart, the total cost of the items in the cart, and some tools to modify the cart
6. If the user clicks “check out” a hidden div appears as a popup, allowing the user to input card information, and pay.

Code:

Index.html : holds the basic html of the site including all the template html for the dynamically generated elements. The html has a number of empty divs that are filled in by loadstore.js and cart.js.

Products.js: Products.js holds the data of the products available for purchase in our store. Every product has a productID, name, category, price, an image to display the product, and a description. The productID is unique, and no product can have the same ID. Names can have similarities. For example, we would like to provide cough medicine from different brands. Category is a string that is used to sort the products based on the type of product. Products can have similar prices, and they’re not limited to a price range. Images must have unique names to avoid displaying the wrong images. Description, a string type, will contain a longer description of the product that will be displayed when the user selects a product in the product grid.

CS 457 group 3 project
Josiah Conely, Daniel Oliveros Guerra, Dalton Gilmore

htmlMaker.js: This module handles generating the dynamic html elements. The function htmlMaker(template) receives a html template. Once an htmlMaker object is created, it can then generate html when it is passed a json object like this: htmlMaker.getHTML(values). This function combines the template with the data by looking for key works that are marked like this: {{keyword}}. Any elements in the JSON that have the name of the keyword will replace the {{keyword}} with the values[keyword].value.

Loadstore.js: This module handles passing the templates and the data to the htmlMaker. It then loads the store by applying the html generated by htmlMaker.js module to the divs on the index.html.

Cart.js: handles adding, removing and clearing cart contents. It contains a variable cart, which contains items ids. This array is then used to generate html for each item in the cart.

Conclusion: this is a dynamic web store that can be changed by simply sending it different product data. In future iterations, the category buttons will also be dynamically generated by parsing the categories from the products.js data. This will further enhance the dynamic nature of the site, making it truly plug and play.