Ethan Hanlon

|  |  |
| --- | --- |
| GitHub: OccultSlolem  Group: The Beer Men  Teammates:  Pablo Martinez  Jake Swenson  Eric Thompson | ID: 921256970  CSC317 Intro to Web Software Development  922270801  918356067  922361944 |

Group Project 3 – Storefront (Final)

**Description**

Our assignment this semester was to create a storefront using several different technologies of the web, including HTML, CSS, JavaScript, Node.JS, and SQL. We chose to create a site called RoboStore – your one-shop stop for all your robotics needs! This final version of our writeup has parts from all the past writeups.

The first part of the assignment was to create a storefront in pure HTML only. Requirements included navigation between all pages, a home page, product pages, and intermediate pages. Files needed to be organized under the var/www/html directory, and images and product pages needed to be in their own directories. In addition, we had to present our webpage in class.

The second part of the assignment was to build upon the storefront we had built with HTML and add CSS. Requirements included navigation between all pages, a home page, product pages, and intermediate pages, as well as incorporating CSS elements in a visually pleasing manner. Files needed to be organized under the var/www/html directory, and images, product pages, and stylesheets needed to be in their own directories. In addition, we had to present our webpage in class

This final part of the assignment required us to make our storefront into a database-driven system, using SQL as well as either PHP or Node.js.

**Approach / What we did**

We chose to create a website named RoboStore to sell robotics equipment, including microcontrollers, tools, sensors, and motors. We used a consistent navbar across every website, the HTML for which we stored in navbar.html. The navbar was a 100% width table with a yellow background, an h1 with our site’s title on it, navigation icons, and category shortcuts. We also added a settings page where a user could change various account and profile settings.

For the CSS, we agreed upon a consistent color scheme and some design constraints: all pages would have the same navbar with yellowgold background color, the background of each page would be aliceblue to offset it from plain white just a little, and we would use high contrast blacks and whites to aid readability. Pablo and Ethan contributed to the stylesheets for much of the project. Eric added a footer which was used on every page.

During that phase of the project, it was somewhat complicated to get all the UI elements uniformly styled, especially in the product pages. All product pages had to be the same except for the product related titles and files, so this involved a lot of copying and pasting. Also, importing and getting the stylesheets working took a while to get working, and when they did work, there were conflicts with the previous html code. This required us to occasionally delete and rewrite parts of our HTML.

Next, we had to add JavaScript to the project. JavaScript is the functional language of the internet, so it lets us do things like simulate a cart and login system even without having a database to run it in.

During the JavaScript phase of the project, we were able to ameliorate the problem of repeating ourselves for certain UI elements quite a lot. In particular, the navbar no longer had to be retyped across every single page: Ethan created a script that found a “navbar-container” div to insert the HTML into rather than having to retype it in every file. This made any changes to the navbar take far less time than before.

Authentication was probably the simplest JS aspect to implement. When the user presses sign in or register, whatever is in the username box is recorded to localstorage. The username is read in the settings page, and when a user presses to sign out the token is cleared.

Searching was set up by Ethan out by storing all the items for sale in an array. When enter is pressed, if the index of any item in the list is greater than zero the user is redirected to that page. An improvement to be made in future versions would be to list all similar objects in a search page.

The filter was implemented by Ethan with a Javascript Object – the key is the name of the product, and the value is an array containing the rating, price, manufacturer, and preview image source. When the filter button is pressed, it iterates through all entries, and if the rating, price, or manufacturer matches, the items are written to localstorage and the user is redirected to filter.html. On this page, a for loop runs that adds the product card HTML for all the items onto the page.

For the cart, Pablo used localstorage to store the items that the user selects, and then they were displayed in the cart, and then the checkout page. Three .js files were created for the products, the cart, and the checkout page. In the cart, the price updates as the user changes the quantities, and items can be removed from it.

For the final phase of the project, we chose to go with Node.js, as we had already been using Javascript and wanted to keep our project in one language. To serve the website, we used a package for node called Express, which allows you to map certain actions and dynamic files onto different endpoints on your server. For example, on the simplest end I could say that a GET request to the endpoint /about would return the static about.html page. On the other hand, several tasks required information that was to be stored on the database:

* Displaying what products were on sale, including their names, preview images, prices, and ratings.
* Telling the user what products to display, especially for filtering and searching for products.
* Keeping track of what items were in each user’s cart.
* Authorizing and prohibiting access to different parts of the website.

Pablo started by creating the basic layout of what the SQL table looks like. Our database holds three tables. The first is a users table holding usernames and passwords. The second is a products table holding product names, prices, images, and ratings. The final table is called relation, where every user’s cart was stored. We were initially thinking of storing the cart inside each user but realized that with SQL’s ability to use foreign keys to link objects from two tables together, it would make queries to the user’s cart much simpler if we could simply store those in a separate table.

We were also able to search for products. Using the SQL LIKE keyword Ethan was able to match strings that are roughly similar, producing a working search function. In fact, using SQL cut down on the amount of work to do considerably, since we could just load one same template file and fill it with new information, rather than constantly copy and paste all the HTML.

Ethan also did the login, filter, product page, and category pages as well. Category, product, and filter pages could largely use the same template, simply adjusting the query to match for each route. For example, the filter route would look like this:

**Text

Description automatically generated**

This corresponds to the filter route in our server.js file. Each forward slash after filter separates an argument to the function. The first one (‘Arduino’) corresponds to the manufacturer. The second one (‘4’) corresponds to the minimum rating. The final one (‘60’) is the maximum price. This returns us two items from our inventory:

**Graphical user interface

Description automatically generated**

This formula was similar for the product pages: by loading the data in though a SQL query you could load any available product in the database with the same template page.

**Issues and Resolutions**

During the JS phase of the project, there were many more difficulties than the other phases.

One of the issues we faced was adding functionality to the navbar. Changing searchbar behavior on every single page was a major nuisance, so I wrote it out into a separate script that adds the HTML into the “navbar-container” div tag. I wrote similar behavior that adds the footer to the “footer-container” div.

The filtering was an especially complex project, although it worked surprisingly quickly for a 168 SLOC segment. The hardest part was displaying the results correctly. For some reason, I could not get JavaScript to write the td tag into the HTML, so I instead applied the requisite CSS styles to the a tag contained within and playing with the margin properties until it looked the same as other pages.

Just getting the cart to work took hours, and due to lack of experience of JS caused problems. Getting the total price in the cart to change when the products were modified was a challenge to get working correctly. Also, when adding a product to the cart, working with local storage was a challenge, as the data had to be stored in a complicated way, and inserting and extracting information was tough. It was a challenge to display the products correctly.

**Screenshots**

*Home page*

*Graphical user interface, website

Description automatically generated*

*Graphical user interface, website

Description automatically generated*

*Product Page*

*Graphical user interface, website

Description automatically generated*

*Graphical user interface, website

Description automatically generated*

*Category Page*

*Graphical user interface, application, website

Description automatically generated*

*Settings Page*

*Graphical user interface

Description automatically generated*

*Login page*

*Graphical user interface, website

Description automatically generated*

*Cart page (Zoomed out to 67%)*

*Graphical user interface, application

Description automatically generated*

*Checkout page (Zoomed to 67%)*

*Graphical user interface

Description automatically generated*

***Graphical user interface, website

Description automatically generated***

*About us*

*Graphical user interface, text, website

Description automatically generated*

*FAQGraphical user interface, text

Description automatically generated*

*Filter, JavaScript version (Manufacturer: Raspberry Pi)*

*Graphical user interface, website

Description automatically generated*

*Filter, Node.js version (Manufacturer: Arduino, $60 max price, >= 4 star rating)*

*Graphical user interface, website

Description automatically generated*

*Category (Motors)*

*Graphical user interface, website

Description automatically generated*