# Documentation for Thames Shopping Cart SPA

The initial premise behind the build is that the user has been on the shopping page on the Thames site and selected items that they would like to purchase. The items being Product A, B and C. The add to cart button has been clicked by the user and they have proceeded to checkout. The items have now been added to the shopping cart and can be viewed on the shopping cart checkout page with the quantity of products selected.

# Tools and Dependencies used:

This SPA is built in plain HTML, Css and JavaScript.

JavaScript: Because this is a relatively small build, I felt it unnecessary to involve any heavy JavaScript frameworks such as React or Vue and decided to code the application in vanilla JavaScript.

CSS: Although the incorporation of Tailwind.css would have made it easier to build out the user interface, there are a few properties of CSS that Tailwind has not yet incorporated into their codebase. Such as flex-grow proportions greater than 1 and the clip-path property, a property I learnt in the process of this build when trying to solve the background-image container changes shifting proportions as the browser window size changed. Although it is possible to both incorporate vanilla CSS and Tailwind using a CDN, I wanted to keep my codebase consistent and avoid bloat of the inline styles.

**Cross Browser Testing**

I developed the app on a windows operating system and have used Google Chrome and Edge browsers to test my applications responsiveness. The code works as expected in the various browsers according to the specific breakpoints and functionality.

**My Process of the Build**

Begin the build by treating it like any normal app focusing on healthy SEO practices using semantic html elements and meta tags to describe the page content, image tags and title. I always find it easier styling applications when the content of the page is already laid down. Because I was using flexbox for the basket and order-summary containers, I wanted to see the layout using a live view in the browser. As I mentioned earlier, the biggest challenge that I faced when it came to the UI design was the background-image responsiveness. I was unaware that clip path existed and took some struggling for a while using transform:skew properties to see if I could manipulate the image-container without the background-image resolution changing and becoming distorted. I then found the clip path property whilst searching for solutions online and a useful site(<https://bennettfeely.com/clippy/>) which I used to append the styling properties to the image. I built and styled the components one at a time in a mobile first media(smallest breakpoint) width environment and added the media queries responsiveness after I had finished styling the smaller

I know from the visual design that I will need to have some form of loop which iterates through each cart product, its specific price, and quantity before updating the totals table. So to do this I’ve placed each product in a row container(cart-items) which I will traverse using the Dom Selector properties and .children property. I have used the dataset (data-tax) attribute on each of the products to extract the tax percentage for each product that will be used in the final calculation of the tax, shipping and sub-total totals.

**JavaScript functionality integration:**

I have placed the JavaScript script after the Html content to ensure that the Html content is parsed first before the script runs. I used the window.onload event in the script to execute a function jsLoaded() once the web page has completely loaded all content. The function contains all functions required to make a functional shopping cart app.

I began building the Javascript functionality by first selecting and initialising the variables I would be using throughout the app, keeping them in the lexical scope available to other functions. I started by selecting the remove buttons of each product Item and added event listeners for the onclick function by users. I then wrote a corresponding function that removes the selected product from the page and executes the updateSummaryTotals function.

The updateSummaryTotals function took me the longest to build as it contains all the totals and DOM elements that will be updated after each user input change and is used widely across the app. I spent a bit of time trying to traverse through the product item container, selecting elements that would update in real time as the users remove them from the page.

I then used conditionals to calculate the shipping price based on the requirements given in the design brief as well as check to see if the total of the items in the cart are equal to 0. If the total items in the cart are equal to zero (all products removed from the cart) then an emptybasket function is executed.

I struggled a bit when it came to rounding down to two decimal places using the toFixed(2) method as I kept getting unrecognised errors from the console. I found a solution using the conversion string to number method Number() or parseInt() as the total tax was being converted to a string upon addition of the other totals.

The function then takes the updated items total, tax, shipping and sub-total totals from the variables and appends them to the DOM.

With the main function built, I selected and added event listeners to other user inputs on the page such as the product quantity changed and buy button selected. I then wrote the corresponding functions of the event listeners.

After every alteration to the basket table the UpdateSummaryTotal function runs. The function was used to loop through each row of products and looks for the quantity and price of the item changed or removed. This continually updates the totals as the user manipulates the user inputs.

Thank you for your consideration and the opportunity to build out this SPA application. I learnt a lot from the build and I look forward to hearing from you soon.

Regards

Nick Craze