**Documentation**

**Explanation of my process and code according to CRUD:   
  
C – Create**

I have an initial array created. This is meant to be a shopping list for the grocery, so all items within the array are what you would find at a grocery store.

In order to create, you must enter in all the item details in the form to the right side.

A screenshot of a form

Description automatically generated

My code will check to ensure that everything is entered properly. If something is incorrect, you will get an alert letting you know. If everything is correct, a new item will be created, the list will refresh, and you will get an alert letting you know the creation was successful.

A screen shot of a computer program

Description automatically generated

addItem is where this all occurs. The if checks to make sure there is a proper input on ALL input bars. If not, you will get an alert letting you know. The cost must be a number AND larger than 0 (nothing is expected to be free at the store.) A new item is then created using the inputs. The items is pushed to the end of the list/array using .push. The input bars are then cleared so they can be empty and the previous inputs don’t remain. The list and total cost refresh by calling list() and totalCost(). The user is then alerted of a successful creation.

**R – Read**

To view all the details of each item, the user can click on the “Show details” button at the bottom of the list.

A screenshot of a list

Description automatically generated

A computer screen shot of a program code

Description automatically generated

ShowDetails is called when the document is ready using $(“#showDetails”).click(showDetails) this means that once you click on the button, showDetails will run. listofItems clears the current list. Then, each item is added back onto the list but with its additional details. $.each is used in order to do this for each item in the array. Each paragraph contains either the name of the item, type, subtybe, or description. The listofItems is added to the proper area on the page using $(“#shoppingListContainer”).html(listofItems). Then, the text on the button is changed by selected the button and using .text. Lastly, the hideDetails function is called once the button is clicked again.

After clicking it again, the list expands and shows the details of each item. The button then reads “Hide Details.” Upon clicking it again, the list shrinks back to its original state and the button once again reads “Show Details.”   
  
A screenshot of a computer screen

Description automatically generated

A screen shot of a computer code

Description automatically generated

Originally, I had typed everything out again so it would be the same as the list function. However, I found that it was easier and quicker to simply call the list function again within the hideDetails function. This way, there isn’t extra lines. This also works because once the extra details are hidden, the list goes back to its original form. The button text is changes back to “show details” using the same method as above. The showDetails function is called so that once the user clicks on the button, it expands the list. showDetails and hideDetails essentially go back and forth.

**U – Update**

To update an item, the user must select the item based on the number it is on the list. If anything, other than a number or a number on the list is typed, the user gets an alert letting them know. If a correct number is entered, the user will proceed to updating the item.

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of code

Description automatically generated

itemToUpdate takes the user input and gets the value of it. indexToFind takes this number and converts it to an integer using parseInt. It then subtracts 1 since an array is 0 base. This gets the proper index of the item. The input is then checked to make sure it is a number, larger than or equal to 0, and no larger than the index of the last item in the array. If none of these conditions are met, the user is alerted that the number they entered is incorrect. If these conditions are all met, the user moves on to updating. For each thing that can be updated, a prompt is given. The user enters what they would like to update it to. The current name, type, subtype, description, and cost is also shown in the prompt text. If the user types in a name, the item is updated (example: The name is updated with currentItem.name = nameUpdate with nameUpdate being the input). If nothing is typed and the field is left blank, the item information will remain and the user will be given an alert of an invalid entry. (example: currentItem.name = currentItem.name) The if statement will check for invalid entries with (example) if (nameUpdate === “”) the === means it MUST be exactly that. The “” means the input is blank with nothing typed in. This process repeats for all bits of the item until the item is successfully updated (an alert comes up to let the user know.) the list and cost is then refreshed using the same list(); and totalCost(); the input bar is also cleared using the same method of .val(“”)

**D - Delete**

To delete an item, the user must input the number of the item on the list, the name of the item, the type, or subtype of the item. If using the type or subtype, you can delete all items of that type, so be careful! If you want only a specific item, use the number or name method instead.

A screenshot of a computer

Description automatically generated

A computer screen shot of a program code

Description automatically generated

Delete uses the same methods as the ones above. Using the same method, I converted the input to a value. itemFound is set to false and will change upon a successful find and deletion. indexToFind changes the input to an integer using parseInt and subtracts 1 in order to get the index of the item (arrays are 0 base, but the list is not.) If the input is a number, is larger than or equal to 0, and is not larger than the arrays index, it moves onto the deletion. If not, an alert will let the user know that the item was unable to be found. If deleting by number the item is simply spliced using indexToFind. ItemFound becomes true which means everything was successful.

If deleting by name, type, or subtype, a for loop is used. This will go through every item within the array. If the item is found to have a matching name, type, or subtype with the input, it will be deleted using the same splice method (instead of indexToFind, i is used instead as this is the number of the items index.) .toLowerCase converts the text to all lower case to make searching easier. === ensures that the input is the exact same as the array detail. || is used because it is an “or” (example: the input is the same as the item name OR the input is the same as the item type). Once the item (or items, since this is a loop, all items that match the input will be deleted) is found, the user will get an alert that deletion was successful. This alert will show for all deleted items. itemFound once again becomes true. After all the deletion, the list and totalcost will refresh.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sub-Task | Completion | Partially completed, no errors | Partially completed, errors | No attempt |
| **Define an array of objects**. Store this array in an external file (separate .js file). **Create a function to return the array**, and use this function in the calling code | 20 | 15 | 10 | 0 |
| **Reflect the objects in the display area, using jQuery** to create the displayed items **dynamically** | 20 | 15 | 10 | 0 |
| **Create:**Include a mechanism for creating a new  object, using the attributes entered by the user  **Add the object to the array** and **refresh** the display on the web page, using jQuery (hint, recreate the element content, in which the items are displayed) | 15 | 10 | 5 | 0 |
| Include a small but visible **indication** that the item has been created (hint, could be a message, could be a status bar update, like a div with text that is updated) | 5 | 3 | 1 | 0 |
| **Read:** Include a mechanism for listing a specific object details | 15 | 10 | 5 | 0 |
| **Update:** Provide a mechanism to **change a part of the object**, and to **display changed details** on web page - do not use a prompt (hint, think about creating text fields and retrieving content from these) | 15 | 10 | 5 | 0 |
| **Delete :**Provide a mechanism to **remove the object from the array** and **refresh the details on the web page** (hint, think about resetting the content of your list of items on the page) | 15 | 10 | 5 | 0 |
| **Include a small but visible indication that the item has been deleted** | 5 | 3 | 1 | 0 |
| **UI Design - UI is unique and professional**  Should not be a copy of an exemplar, should not be AI generated | 15 | 10 | 5 | 0 |
| **Functional links provided, GitHub repository and GitHub Pages** | 10 | 0 | 0 | 0 |
| **Documentation**  **Describe**the process involved and**explain the code**you used to implement**Read, Update, Delete and Create. Illustrate your code and technique**by including the code snippet that corresponds to the operation. | 25 | 15 | 10 | 0 |
| Totals | 160 |  |  |  |