CIT 215 Final Project

This document will explain the general purpose of the project and what the user is able to do followed by the explanation of the “filter” functionality as it was not in the rubric, and finally c

My final project presents an array of cars and their information on a webpage. The nav bar contains buttons to create a new car, toggle a div showing user activity, and filter cars based on criteria. The body of the document contains cards. Each card contains the car name, image relevant to the type of car, and three buttons. These buttons allow the user to read car information, update car information, or delete the car.

Filtering Function Explanation

The filter functionality was not specified in the rubric so I will explain it here before getting into the rubric. Once clicked, the filter button will rewrite cars Wrapper div to contain text fields similar to what update and create functions do. When user presses submit, another function is started within the scope of that function (so everything is easily accessed) that takes values from user input and parses them to be integers before doing a for each that looks at each car’s type, year, miles and price to see if it is within what the user wants (i.e.: year 2007 and later, or 50000 miles and lower). It knows whether it is looking for it to be above or below from value of drop-down list after user input. By default, all checks are set to true at beginning of iteration and each “if” statement containing a check only occurs if user input is above 0, otherwise remaining true. This way if any weird inputs are made the loop defaults to letting cars through. This also means that any filtering fields not filled out will not affect filtering. Only if a car satisfies all checks will it be added to temp array. I then call upon an entirely separate function using temp array as a parameter, which displays the contents of the array very similar to load cars function except in place of update or delete button there is return button.

Rubric Requirements

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

I created a cars array stored within the info.js file. I create a function whose only purpose is to retrieve the cars array and make a variable ‘array’ equal to the returned cars array. I also make another variable equal to ‘array’ for filtered cars which will be further explained in the read function. Every time I want to use the cars array outside of read function I use ‘array’ variable instead of cars. Note that I created the attributes I wanted to use for the cars and what variables I wanted to be strings or integers, but I told ai to give me 30 cars using the given attributes.

```
const getCars = () => {
  return cars;
};

// Because of the filtering fu
//  Instead I will have to rea
//    then I can use same read
let array = getCars();
let filteredCars = array;
```

Reflect the objects in the display area, using jQuery to create the displayed items dynamically

The loadCars function will cycle through all objects in the cars array and display info accordingly. For each car we create an individual div for car make and model, car image, read, update and delete button, and information to be read. Divs are populated with information according to array index, and the buttons are given an id in the form of incrementing index variable, which we can use when buttons are clicked. Below shows a portion of the for each within load cars function, including an example of div being populated with info according to current car, and the index variable used to id the created buttons. When cars are deleted or updated, the loadCars function is called which will "refresh" the page.

```
let carIndex = 0;

// iterates through all cars and makes special container for each
array.forEach((car) => {
  // carCard will contain ALL relevant car information that will be appended to it
  let carCard = $(`<div class = "carCard"></div>`);

  // nameLine contains the make and model or 'name' of the car
  let nameLine = $(`<div class = "nameLine" style = "text-align: center;">
      ${car.carMake} ${car.carModel}
    </div>`);

  // carImage contains the image for the type of car
  //  image based off type of car (ie: luxury, sedan, suv, etc) so that created ent
  //  I use a generic silhouette image so that it can be applied to all entries
  let carImage = $(`<div class = "carImage">
      <img src="./images/${car.carType}.jpg" alt="${car.carType}"/>
    </div>`);

  // buttonBar contains the buttons for reading, updating, and deleting
  let buttonBar =
    $(`<div class = "buttonBar">
        <button class = "readCar" id = ${carIndex}>Read</button>
        <button class = "updateCar" id = ${carIndex}>Update</button>
        <button class = "deleteCar" id = ${carIndex}>Delete</button>
      </div>`);
```

The end of each iteration appends all divs to the carCard div which is then appended to main wrapper, before incrementing index used for buttons ids.

```
    // will be updated to have info with "Read" button
    let infoBar = $(`<div class = "infoBar"></div>`);

    // end of iteration, append all stuff to the card i
    carCard.append(nameLine);
    carCard.append(carImage);
    carCard.append(buttonBar);
    carCard.append(infoBar);
    carsWrapper.append(carCard);

    // carsWrapper is already put within parentContaine
    // $(".parentContainer").append(carsWrapper);

    carIndex++;
  });
```

Creating Object

Nav bar has a button to create car. When clicked will rewrite carsWrapper div to contain various text fields for user to insert information into. When the submit button is clicked it activates a function (within the scope of create function for easy access to info) that creates an object with attributes of what user input before pushing it to the cars array and refreshing the page by calling loadCars function. The image below shows what happens once the button is clicked.

```
// another function I am choosing to make anonymous for efficiency sake and it s
$("#createButton").on("click", () => {
  // make variable for the object that will be pushed, containing all informatio
  let userSubmittedCar = {
    carMake: $("#CarMake").val(),
    carModel: $("#CarModel").val(),
    carType: $("#typeID").val(),
    carColor: $("#CarColor").val(),
    carYear: $("#CarYear").val(),
    carMiles: $("#CarMiles").val(),
    carPrice: $("#CarPrice").val(),
  };

  // log the object
  console.log(userSubmittedCar);

  // push the created car to the cars array before checking to see if it worked
  array.push(userSubmittedCar);
  console.log(array[cars.length - 1]);
```

Indication of Object Created/Updated/Deleted

The nav bar has a button for toggling user activity. When the button is clicked, the visibility of div containing activity history is toggled. I did something similar for the chocolate lab, but the function itself is really short shown below.

```
const toggleUserActivity = () => {
  // Did this same toggle feature in my chocolate lab assignment, so familiar with it.
  console.log(`now in user activity function`);
  if (showActivity) {
    showActivity = false;
    $(".activityContainer").hide();
    console.log("showing user activity is now " + showActivity);
  } else if (!showActivity) {
    showActivity = true;
    $(".activityContainer").show();
    console.log("showing user activity is now " + showActivity);
  }
};
```

I wanted to start the webpage hiding it, but I needed to wait until document was ready before doing it so I set up the following to happen as soon as document was ready, right before loadCars happens

```
let showActivity = false;
const startUp = () => {
  // we need to wait to hide activity con
  $(".activityContainer").hide();
};

$(document).ready(startUp);
// by starting loadCars we are
$(document).ready(loadCars);
```

The actual adding of content is done within the create, update, delete, and filter functions. The following image is an example of adding message in create function. The list can get long, so I prepend it so that the user doesn't have to scroll to see most recent activity.

```
let userActivity = `<div class = "userActivity">
    Added ${userSubmittedCar.carColor} ${userSubmittedCar.carYear}
  </div>`;
  $(".listItems").prepend(userActivity);
```

Reading Object

Each carCard has a read button, when that button is clicked the readContent function is called on. The infoBar div is left blank when objects are created in loadCars, but now we update it with relevant information. jQuery can automatically obtain the event information. Recall that when buttons are made they are given id equal to indexing variable I created. This index corresponds with the index position of object in array that card is displaying, so read button with id of 0 is inside carCard that displays information for first car in array. I use this variable to determine what car from array I need to read. I also use this index to determine which InfoBar to append to. These events are shown in the function below.

```
const readContent = (event) => {
  // series of logs to claify that we are in function and what we are doing
  console.log(`Now carrying out the 'readContent' function`);
  index = event.target.id;
  console.log(`The index we are working with is ${index}`);
  console.log(
    `We are working with the ${array[index].carMake} ${array[index].carModel}`
  );

  // set what the text content is going to be, use the imported index to ensure correct c
  // at the end after a line break we also add a close button
  infoBarContent = `<div class = "infoBarContent" style="font-size: x-large;">
    This ${array[index].carYear} ${array[index].carMake} ${array[index].carModel} is a ${
    <br> <br>
    <button class = "close" style = margin-left: 0px; margin-right: 0px;>Close</button>`;
  console.log(infoBarContent);

  // Every carCard has a ".moreInfo" div, we need to make sure we're appending to the rig
  // we know that something jQuery does is automatically assign an index to objects creat
  // so we can make variable equal to the ".infoBar" div that has an index equal to the i
  let correspondingDiv = $(".infoBar:eq(" + index + ")");
```

In addition to car information, a close button is also appended. When this button is clicked a simple .html is used to overwrite the infoBar to be blank

```
$(".close").on("click", () => {
  $(correspondingDiv).html(" ");
});
```

Another feature added is a warning in the case of empty information. If any of the car's attributes are empty, I append a warning before the actual car information, otherwise I only append car information

```
  let missingContentWarning =
    $(`<div class="noContentWarning" style="font-s
    There is missing info that needs to be updated
  </div>`);

  // if there is any missing information at all we
  if (
    !array[index].carYear ||
    !array[index].carMake ||
    !array[index].carModel ||
    !array[index].carColor ||
    !array[index].carType ||
    !array[index].carMiles ||
    !array[index].carPrice
  ) {
    correspondingDiv.html(missingContentWarning);
    correspondingDiv.append(infoBarContent);
  } else {
    // once we have locked onto the correct one, w
    correspondingDiv.html(infoBarContent);
  }
```

Note that when reading from the filtered cars, it calls upon a separate function that does the same thing but everywhere that uses ‘array’ instead uses ‘filteredCars’. I had thought there were no issues from using filteredCars for both but that turned out to not be the case after testing more, so they use different read functions and draw from different arrays.

Updating Object

When the update button is clicked it calls upon updateContent function. It rewrites the carsWrapper to have text fields similar to createContent but each field has a default value of the pre-existing car attributes. We did not learn how to set default value of drop down list so I had to look that up, but there were otherwise no issues. I use the same index technique as in readContent to identify what car we are updating.

```
const updateContent = (event) => {
  // console commands to keep track
  console.log(`Now carrying out the
  index = event.target.id;
  index = parseInt(index);
```

Once the submit button is clicked the change is made and user activity is updated before ‘refreshing’ the page with loadCars function

```
$("#updateButton").on("click", () => {
  array[index].carMake = $("#CarMake").val();
  array[index].carModel = $("#CarModel").val();
  array[index].carType = $("#typeID").val();
  array[index].carColor = $("#CarColor").val();
  array[index].carYear = $("#CarYear").val();
  array[index].carMiles = $("#CarMiles").val();
  array[index].carPrice = $("#CarPrice").val();

  let userActivity = `<div class = "userActivity">
    Updated ${array[index].carColor} ${array[index].car
  </div>`;
  $(".listItems").prepend(userActivity);

  // now we just have to refresh the content so call up
  loadCars();
```

Deleting Object

When the user clicks the delete button the deleteContent function is called on. I get the index of the object I want to delete using the same index method as with update and read

```
const deleteContent = (event) => {
  // console commands to keep track of index we will use lat
  console.log(`Now carrying out the 'deleteContent' function
  index = event.target.id;
  index = parseInt(index);
  console.log(`The index we are working with is ${index}`);
```

Now all I have to do is update the user activity history and splice the array to remove the object before ‘refreshing’ the page.

```
let userActivity = `<div class = "userActivity">
  Deleted ${array[index].carColor} ${array[index].carYear}
</div>`;
$(".listItems").prepend(userActivity);

// actual deleting part
array.splice(index, 1);

// 'reload' cars
loadCars();
```

User Interface

In addition to static styling being used, jQuery is used to update the styling of carCards and buttons when hovered over

```
$(".carCard").on("mouseover", function () {
  $(this).css("border-top-left-radius", "22%")
  $(this).css("border-bottom-right-radius", "22%");
  $(this).css("border-top-right-radius", "0%")
  $(this).css("border-bottom-left-radius", "0%");
  $(this).find(".carImage img").css("border-radius", "100%")
  setTimeout(() => {
    $(this).find(".carImage img").css("border-top-right-radius", "100px")
    $(this).find(".carImage img").css("border-bottom-left-radius", "100px")
  }, 1000);

});
$(".carCard").on("mouseout", function () {
  $(this).css("border-top-left-radius", "0%")
  $(this).css("border-bottom-right-radius", "0%");
  $(this).css("border-top-right-radius", "22%")
  $(this).css("border-bottom-left-radius", "22%");
  $(this).find(".carImage img").css("border-radius", "0%")
  $(this).find(".carImage img").css("border-top-right-radius", "22%")
  $(this).find(".carImage img").css("border-bottom-left-radius", "22%")
});
```

```
$(".navbar button").on("mouseover", function () {
  $(this).css("background-color", "#bebebe");
  $(this).css("box-shadow", "3px 3px 3px #fc2e20");
  $(this).css("width", "130px");
  $(this).css("height", "35px");
  $(this).css("margin-left", "20px");
  $(this).css("margin-right", "20px");
});
$(".navbar button").on("mouseout", function () {
  $(this).css("box-shadow", "");
  $(this).css("width", "150px");
  $(this).css("height", "40px");
  $(this).css("margin", "10px");
  $(this).css("background-color", "");
  $(this).css("color", " ");
});
```

```
$("button").on("mouseover", function () {
  $(this).css("background-color", "#bebebe");
  $(this).css("box-shadow", "3px 3px 3px #fc2e20");
  $(this).css("width", "120px");
  $(this).css("height", "40px");
  $(this).css("margin-left", "17px");
  $(this).css("margin-right", "18px");
});
$("button").on("mouseout", function () {
  $(this).css("box-shadow", "");
  $(this).css("width", "135px");
  $(this).css("height", "50px");
  $(this).css("margin", "10px");
  $(this).css("background-color", "");
  $(this).css("color", " ");
});
```