**CMPS 350 Web Development Course Project Phase 1:**

**E-Commerce Platform**

**Student Names:**

**Noora Al-Thani 201901903**

**Noor EAhmednooh 201901449**

**Maryam Alyafei 201901909**

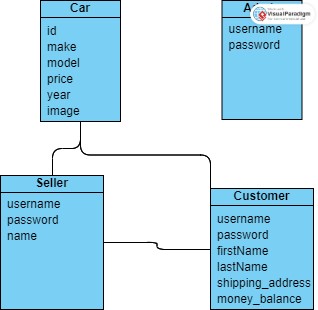
**Ecommerce website name: AutoAlley**

**Group: 1 - NNM**

1. **Design the App Web UI and navigation.**

In out project we got together and started designing our web interface, so this is it hand drawn. We did this when we first established our work, and going forward minor designs may have changed but overall, it stayed the same.

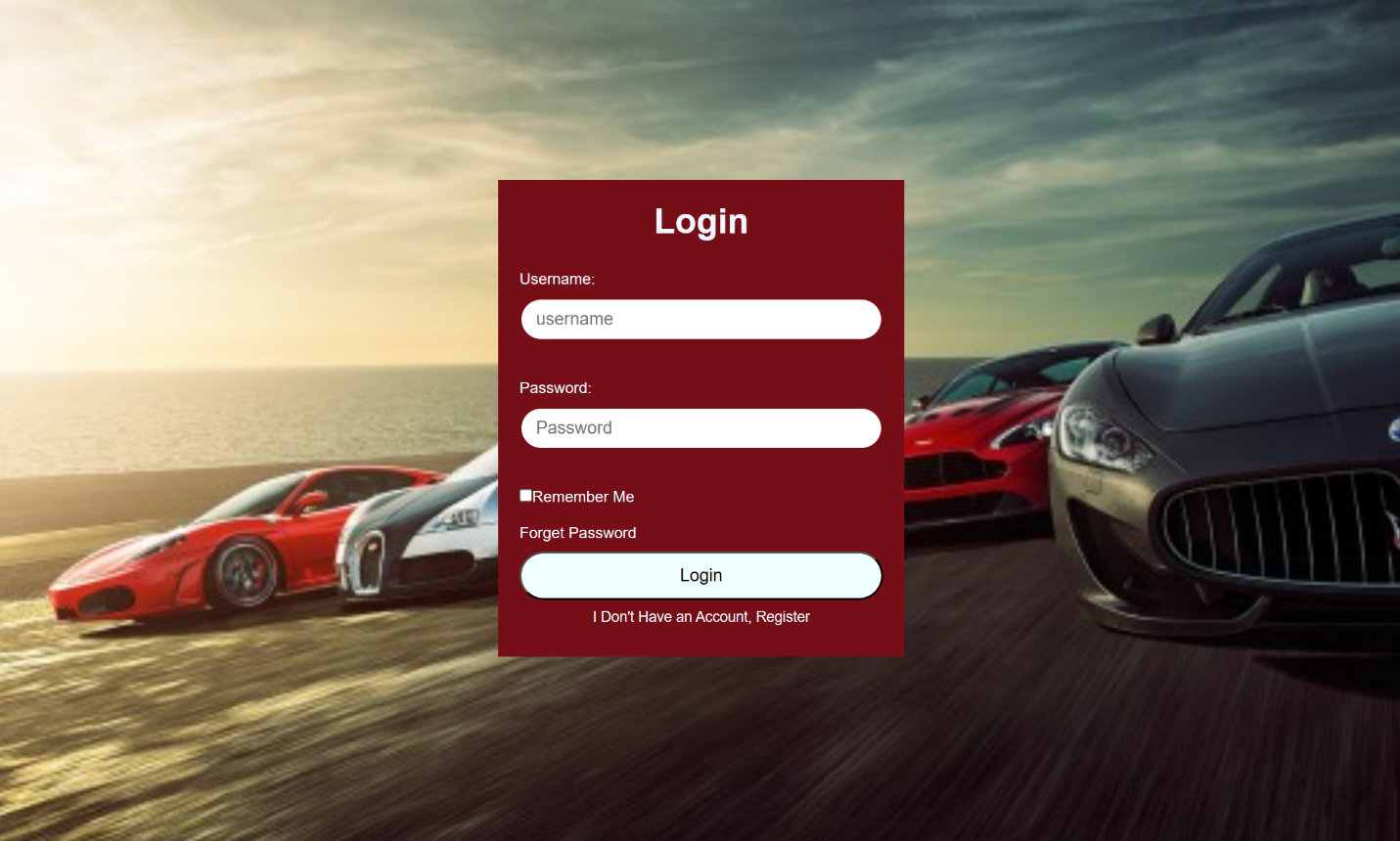


We also created a class diagram before starting to set our understanding to start JavaScript coding.

1. **Implement the app UI and navigation using HTML, CSS and JavaScript. (screenshots of pages)**
2. **Read/write to simple JSON files that you need to create and initialize with some sample data. (screenshots of json)**
3. **Application design documentation should include the Entities, Repositories and Web API class diagrams. (screenshots of javascript classes)**
4. **Document the app testing using screen shots illustrating the results of testing.**
5. **Student Contribution**

|  |  |  |  |
| --- | --- | --- | --- |
| **Names/usecases** | **Noora Al-thani** | **Noor Eahmednooh** | **Maryam Alyafei** |
| **Usecases** | **(5) View available items on sale and sale history** | **(3) Purchase an Item** | **(1)login /out** |
| **(6)upload an item to be solved** | **(4) View the purchase history** | **(2)Search Available Items** |
| **Other** | **JS classes + Cars JSON** |  | **Users JSON** |

* **Use Case 1: Login**
* **First we create a HTML and a CSS to design the login page to be like this :**



* **Then we create the users.json to put the users inside it .We wrote 9 users the first 3 are Customers, the 3 after are Seller and the last 3 are the Admin .**

  {

        "user\_id":1,

        "role":"Customer",

        "username":"user1",

        "password":"user1",

        "name":"Noora",

        "surname":"Aziz",

        "balance":10000,

        "shipping address":{

            "counrty":"Qatar",

            "city":"Doha",

            "zone":45,

            "str name":71

        }

    },

{

"user\_id":4,

"role":"Seller",

"username":"user4",

"password":"user4",

"name":"Ahamad",

"surname":"Aziz",

"balance":1000,

"shipping address":{

"counrty":"Qatar",

"city":"Alsad",

"zone":80,

"str name":11

}

}

{

"user\_id":7,

"role":"Admin",

"username":"user7",

"password":"user7",

"name":"Mona",

"surname":"Ali",

"balance":3345100,

"shipping address":{

"counrty":"Jordan",

"city":"Amman",

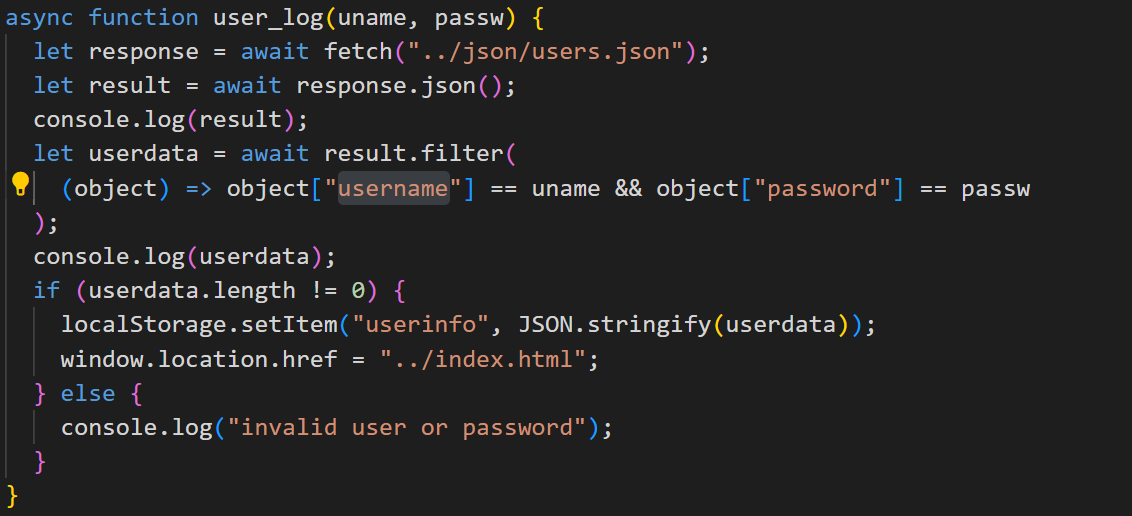
"zone":7,

"str name":781

}

}

**- Then we started with the login function :**



**We wrote an attributes to bring the information from the users.json then we convert it to json so it be readable. In the userdata it will filter the username and password after that we will have the if statement to check if the length of the userdata is not equal to one then it will be added to the local storage by (localstorage.setItem) and will redirect to the index.HTML page if otherwise it will show the error massage**

function checkLogStatus() {

let user\_data = localStorage.getItem("userinfo");

//console.log(user\_data);

if (user\_data != null) {

let ele = document.getElementById("user\_profile");

user\_data = JSON.parse(localStorage.getItem("userinfo"));

ele.innerHTML = `

<a href="#">Welcome, ${user\_data[0]["username"]}</a>

<button onclick="logoutUser()">Logout</button>

`;

}

if (user\_data[0]["role"] == "Customer") {

let ele = document.getElementById("top-nav");

ele.innerHTML = `

<a href="index.html">

<span>Home</span>

</a>

<a href="coustomer\_orders.html">

<span>Order History</span>

</a>

`;

return;

}

if (user\_data[0]["role"] == "Seller") {

let ele = document.getElementById("top-nav");

ele.innerHTML = `

<a href="index.html">

<span>Home</span>

</a>

<a href="seller/index.html">

<span>Add New Car</span>

</a>

<a href="seller/order\_history.html">

<span>Order History</span>

</a>

`;

return;

}

if (user\_data[0]["role"] == "Admin") {

let ele = document.getElementById("top-nav");

ele.innerHTML = `

<a href="index.html">

<span>Home</span>

</a>

<a href="admin/index.html">

<span>Admin Controls</span>

</a>

`;

return;

}

}

checkLogStatus();

**Then in each page except the login.html we have the logout and checkLogStatus function :**

**The checkLogStatus dose check if the user logged in or not if the user logged in, it will start checking in the role for example if it is customer it will chand the page style like change the navigation it will add to it the order history and will redirect it to the home page if it was seller it will redirect it to the home page and the add new car page and the order history page ‘this order history is different from the customer order history. For the customer it will show the customer orders but for the seller it will Showen the order history).**

function logoutUser(url = "") {

localStorage.removeItem("userinfo");

if (url == "") {

let ele = document.getElementById("user\_profile");

ele.innerHTML = `

<a href="login.html"> Login </a>

`;

window.location.href = "index.html";

}

}

**The logout function: is to take the user to the index.html page so if url was empty it will take the user to the login page that means that the customer is not logged in otherwise it will return the user to the home page**

* **Use Case 2: Search available items**

function searchbumodel(model\_name) {

let carlist = JSON.parse(localStorage.getItem("cars\_store"));

let result = carlist.filter((car) =>

car["model"].toLowerCase().includes(model\_name)

);

showcars(result);

}

**For this function we want to search for the car by model. It keeps the Carlist that is in the local storage . Then filter the result that has the car list that is in the storage then check what the user wrote in the search input box and after that check partially identical to what the User enters if there is a car model with this letters it will be shown**

function showcars(carlist) {

if (carlist.length == 0) {

listings.innerHTML = "there are no models with this name";

} else {

listings.innerHTML = "";

carlist.forEach((car) => {

listings.innerHTML += `

<div class="car\_card">

<img src="${car.image}" alt ="${car.model}" loading="lazy">

<p>Make: <span class="make">${car.make}</span></p>

<p>Model: <span class="model">${car.model}</span></p>

<p>Price: ${car.price}</p>

<button class="purchase-button" onclick="car\_details('${car["id"]}')">Purchase</button>

</div>`;

});

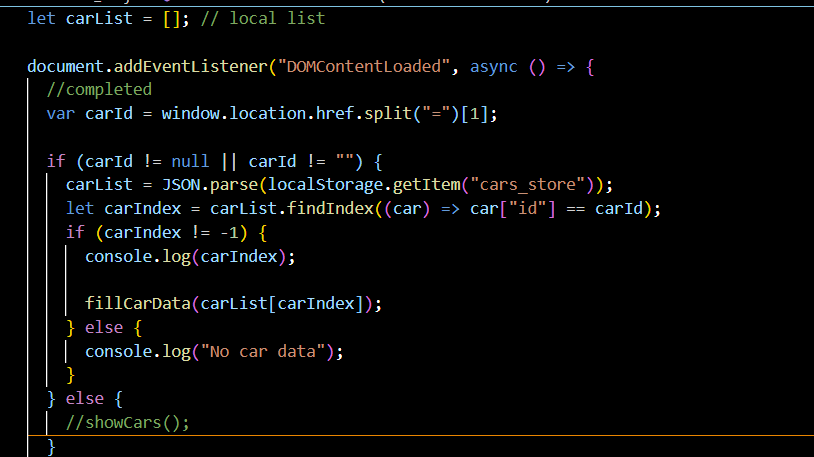
}

}

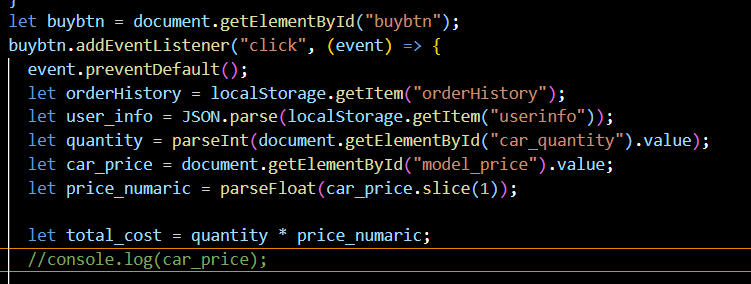
**This function is to show what is the items that the user are try to find without this function we will not have any shown result the result will be in the console only**

* **Use Case 3: order car:**

**After the user select a car by pressing the purchase button, they are transferred to a new page where they review their selected car and proceed with the purchase. The user will see the "car ID", "Model Name" and "Model Price" fields that are read only on this new page, which means they can not be modified directly. The information on the chosen car can be seen in these fields. You can enter the number of cars you want to buy in the "Quantity" field. Input fields for country, city, zone and street address are included in the "Shipping Address" box. This is where the user may enter their shipping details when placing an order for a vehicle.**



**The DOMContentLoaded event, which is triggered when the HTML document is fully loaded, is monitored by this function. When this event occurs, it extracts the car ID from the URL and retrieves information about the vehicle in local storage based on that identification as well as populates a page with relevant details if found. If no vehicle with the requested ID is found, it will record a message suggesting that car data has not been entered.**

****

**The user can activate this function by clicking on the "buy" button. It will cancel the default action to submit a form, retrieve required data such as order history and user information from local storage, calculate the overall cost of an order on the basis of quantity and price of selected vehicle, store it in Local Storage.**

**A computer screen shot of code

Description automatically generated**

**This function creates an object orderData representation of the order, which includes a car ID, user name, number and total cost. If there is any existing order history in local storage, it will be checked. If not, it will initialize a single array for the Order History and assign an order ID as 1, add ordering information to this history in Local Data Storage. If an order history exists, it will be retrieved from local storage and assigned the next order ID, add new orders data to this history as well as update your order history in Local Storage.**

****

**The HTML input fields are filled with the user's vehicle data and the user's shipping address by this function.**

**A computer screen with colorful text

Description automatically generated**

**When the user amount of user less of the amount of car , the user can’t order the car.**

* **Use Case 4: order History:**

**The car will be added to the Order History when a user has placed an order, where all previous orders are retained. This will allow the user to monitor and view their previous purchases later. When the user purchases an item the it will check for two things: (1) if the user balance is sufficient to perchase that item (2) if the quantity ordered is enough. If those two checks are passed it will then proceed to remove from the storage quantity and user balance.**

****

**This function loads data asynchronously from a JSON file containing vehicle-specific information. After that, it takes certain automobile manufactures out of the data and puts them in a dropdown menu. It appears from the commented-out code that there was also a plan to add unique models to a dropdown menu, but that feature is not operational now. Furthermore, an automobile class is imported from an external file.**

****

**This function will handle a new car being added by the user. It retrieves the necessary data, generates a unique ID for this new vehicle, creates a new car object and logs relevant information to perform debugging.**

****

**This function adds a new vehicle to the general vehicles list (carsList) as well as the seller's sales list (seller\_sales). It then reports to the console the modified seller's sales and general vehicle lists for testing purposes. In addition, in local storage, the seller's sales list and the updated car list are updated. The loadSellerSales() function will also be running to refresh the HTML display with updated sales for sellers.**

****

this function fetches sales data for a seller from local storage, displays it on the webpage, and updates sales counters accordingly.

**A screen shot of a computer program

Description automatically generated**

**This function will set up event listeners for the "Update" buttons on your car cards. When you click, it will retrieve relevant data including quantity and vehicle ID as well as user and sales information from the Local Storage to be processed or displayed in future.**

**A computer screen shot of a program

Description automatically generated**

This function makes sure that the amount of a certain car is updated in the local storage general cars list as well as the seller's sale list. It also logs the contents of both lists before updating or refreshing the seller sales display.

* **Use Case 6: Upload an Item to be Sold + Use Case 5: View Available Items on Sale and Sale History.**
* In this use cases when a seller is signed in, he can see a button that only sellers can access called **[Add New Car]** and **[Order History].** Theseller can ***upload new items*** to be sold ***and update the quantity of the items*** on sale. On the top left of the page the Seller can see his *Total sales, Sold Items, and On Sale Items.*

Page With no User Signed In

**A white truck parked on a dirt road

Description automatically generated**

Page With Seller Role Signed In

A white car on a dirt road

Description automatically generated

Page With no User Signed In

* **Add New Car (UC6) :** Allows seller to upload sale item (car) by providing its price, quantity, mode, year, and image as text and number inputs, and the model is a drop down that shows the models as options. The models are retrieved from the cars list from the local storage ‘cars\_store’ and displayed as options in the drop down. When the user selects everything a newCar object will be created using the car class. (see screen shots and more info below)

A screenshot of a computer

Description automatically generated

Add New Car

* In the same page a seller can view the added sale item (part of UC5).
* A Seller enters the vehicle information as mentioned above, they will be stored in an object (newCar -> made using Car class ) then this information will be stored in the local storage and it will be name after the sellers id.

A screenshot of a car

Description automatically generated

Item Added

* The new created car object is then accessed from the local storage and displayed in this view sale items section. However, the seller can update the quantity of the item whenever he sees fit. Only by changing the value of quantity input type text and pressing on the button Update info. The change will happen in three places: (1) the main cars store list in the local storage will be updated (2) the sellers list (named by his id) in the local storage will be updated (3) the global list in the code will also be updated. *List[‘quantity’] = new\_quantity.*
* **Order History (UC5):** Will open a page for the seller that has a table that contains the order object (order no, model, make, unit price, quantity, total price, user id).Items appearing in order history are object that was created in the previous use case when a user purchases an item, in our case both seller and customer can purchase items.

**A computer screen with text on it

Description automatically generated**

* However, when a seller clicks on order history, he can see all users who purchased their items, and when a customer clicks on order history he can only see his purchases.
* When a seller user is signed in, he will have these lists stored in the local storage:

1. Assuming he does not have an item on sale and no orders made: A screenshot of a computer

   Description automatically generated
2. Assuming he put items on sale: [he will have a list added with his id]

A screenshot of a computer

Description automatically generated

1. Assuming he also made a purchase: [order history list is added ]

A screenshot of a computer code

Description automatically generated

1. Assuming a customer purchased from his items: [purchase item will be added to the order history list]