
Report

Assignment 2: EE417 Web Application Development

Name: Killian DECHAMPS

Email: killian.dechamps2@mail.dcu.ie

Id: 22107223

Programme: ESCAO

Dublin, 24/10/2022

About my web application:

The web application that I wanted to create for these different tasks is about sports cars. More precisely, it is a web application directed towards the rental of sports and luxury cars.

To do this, different pages have been created with different functions and goals. First, there is the creation and connection to an account via entries that the user must fill in. Then, as we will see later, the main part of this web application is directed by the navigation bar on the left side of the application. Thanks to this navigation bar, it is possible to navigate between the different pages. The rental pages concerning the cars available for hire are shown.

Each car page is accompanied by photos, text description and a table of characteristics. It is then possible, from these pages, to open a second navigation bar on the right-hand side. The second one allows the user to go to the pages containing the rental and quote forms.

1. Dynamic Menu Bar

The first objective of this assignment is to implement, using the web application we started in the previous assignment, a dynamic menu bar in the top or side panel of our application.

Previously, we had made a navigation bar using HTML and CSS with the style shown in the picture below:

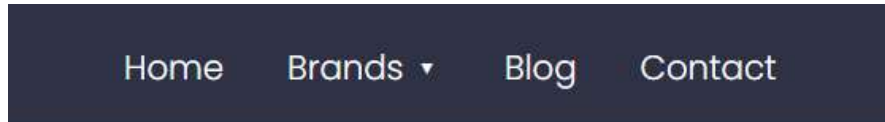


Figure 1: Navigation bar in the previous assignment

Our learning of the JavaScript language during the courses allows us to change the navigation bar to make it more dynamic. The use of JavaScript is also intended to allow the nav bar to fit in better with the style of our web application. Thus, we seek to create the later which will appear only after the capture of an event on the user side (a click, a mouse hover, ...) and in the same idea will disappear only after the same kind of event.

The first step in the design of the new navigation bar was to create the button that the user would press to open it. To do this, we used the "Font Awesome" library, which allows us to add logos to our application using a line of HTML code. The pictures below represent the logo we have chosen and the line of code we have implemented in our HTML part:



Figure 2: Button picture (web application)

```
<div class="btn">
  <span class="fas fa-bars"></span>
</div>
```

Figure 3: HTML line to implement button logo (all HTML file, I54/56 index.html)

However, in order to make the logo look exactly like the one in the picture above, we had to add a CSS part to style it. This CSS part also allowed us to implement the general style of the navigation bar. The pictures below show different parts of the code we implemented to style the whole navigation bar.

```
305 /*Button wich allow to have the transition of the nav bar*/
306 .btn{
307   position: absolute;
308   top: 135px;
309   height: 45px;
310   width: 45px;
311   text-align: center;
312   background: #2E3244;
313   border-radius: 3px;
314   cursor: pointer;
315   transition: left 1s ease;
316 }
```

Figure 4: Button style (Style.css)

```
332 /*Navbar style*/
333 .sidebar{
334   position: absolute;
335   width: 230px;
336   height: 92%;
337   left: -250px; /*when the button is not click*/
338   background: #2E3244;
339   transition: left 1s ease; /*navbar transition*/
340 }
```

Figure 5: Navigation bar style (Style.css)

With the CSS part added to our code, the picture below shows the final rendering of our navigation bar:

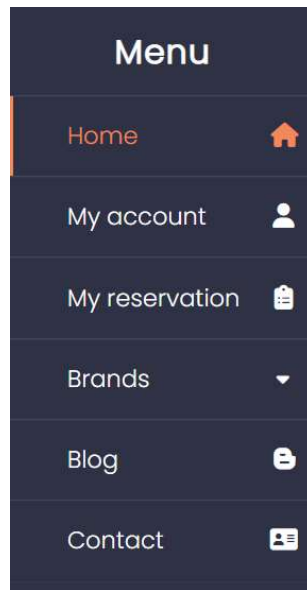


Figure 6: New navigation bar picture (web application)

Then, once the desired information was entered in the HTML part and the style was added via the CSS part, we turned to the JavaScript part. The objective was to hide the navigation bar in normal time, and to make it appear with a scrolling style when the button is pressed by the user.

To do this, we use the "jquery-3.4.1" library. This library allows us to call the functions we want in JavaScript. Indeed, the first one is to be able to open and close the navigation bar. We also tried to create a drop-down menu inside this navigation bar, also coded in JavaScript. That is why, when you press the "Brands" button that you can see on the picture above, a drop-down menu will appear. All these JavaScript functionalities are called using the "Script" part present in all our pages at the end of each code as the following picture shows:

```

137      <!-- Calls javascript functions present in the jquery library for
138           the navigation bar -->
139      <script>
140          //When we click on the button
141          $(' .btn').click(function(){
142              $(this).toggleClass("click");
143              //We show the sidebar
144              $(' .sidebar').toggleClass("show");
145          });
146          //To show the dropdown menu
147          $(' .feat-btn').click(function(){
148              $(' nav ul .feat-show').toggleClass("show");
149          });
150      </script>
    
```

Figure 7: JavaScript part of the navigation bar (index.html)

Once this part of the code is implemented, we can show you the entire navigation bar:

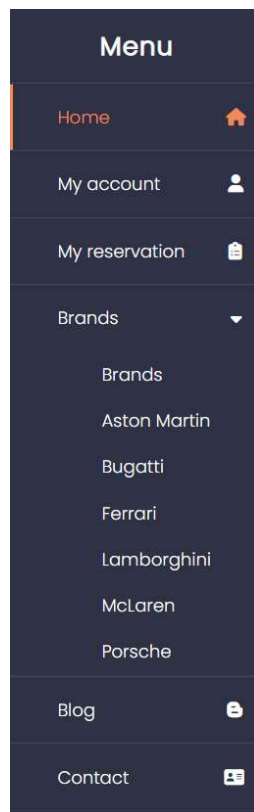


Figure 8: New complete navigation bar (web application)

For your information, another navigation bar is present in the web application. Indeed, if you go to one of page car via the "Brands" drop-down menu, you will see a second navigation bar at the top right (under the Instagram and YouTube logos) with the "Rent/Quote" button. This one works in the same way as the previous one, using the same JavaScript functions and keeping a common style with the main navigation bar. Below is a picture of the second navigation bar :

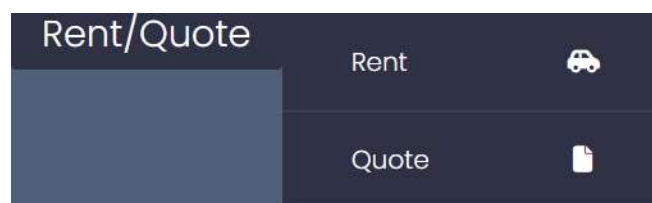


Figure 9: Other navigation bar (web application)

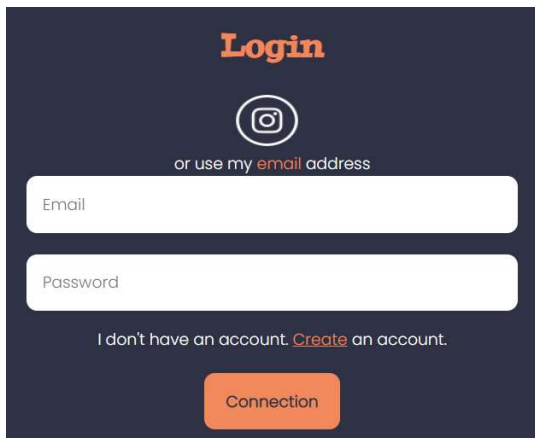
2. HTML forms

The second objective of this assignment is to create an HTML form to allow the user to enter information.

In order to make this consistent with our web application, we first created two forms to allow the user to create or login to an account. You can access to it on each page on the top right of your screen.

In addition, as our site is based on car rental, two other forms were created to allow the user to request a quote or make a car rental reservation. To access these forms, 'Rent' and 'Quote', you will need to be on the page of a vehicle (any vehicle). The menu bar that we introduced in the previous point is on the right-hand side of your screen just below the Instagram and YouTube logos. Simply open this navigation bar and click on one of the two links to be taken to the forms page.

To give you an idea of the forms we have created, here are some pictures of them:



Login

or use my [email address](#)

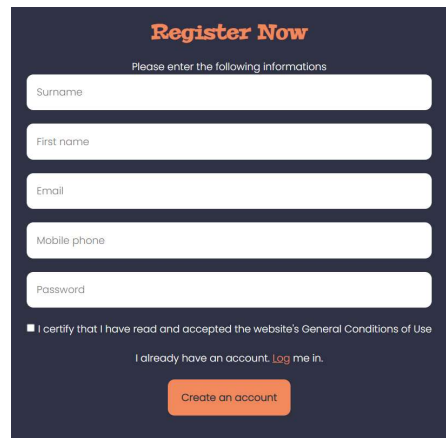
Email

Password

I don't have an account. [Create an account.](#)

Connection

Figure 10: Login form (web application)



Register Now

Please enter the following informations

Surname

First name

Email

Mobile phone

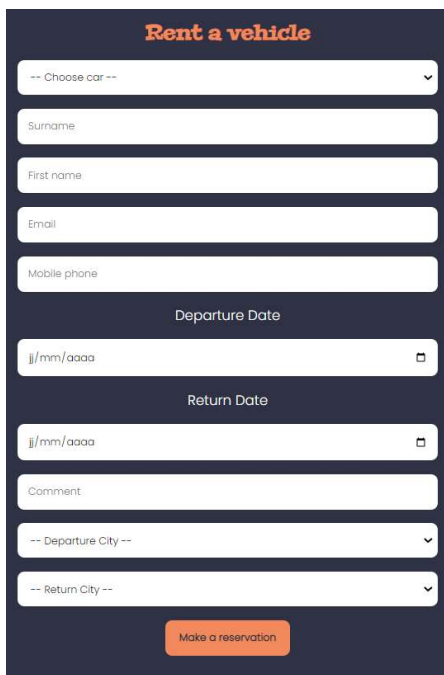
Password

☐ I certify that I have read and accepted the website's General Conditions of Use

I already have an account. [Log me in.](#)

Create an account

Figure 11: Register form (web application)



Rent a vehicle

-- Choose car --

Surname

First name

Email

Mobile phone

Departure Date

jj/mm/aaaa

Return Date

jj/mm/aaaa

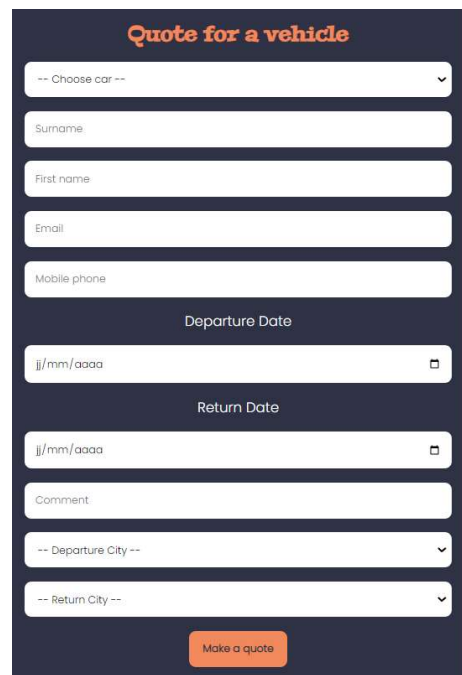
Comment

-- Departure City --

-- Return City --

Make a reservation

Figure 12: Rent form(web application)



Quote for a vehicle

-- Choose car --

Surname

First name

Email

Mobile phone

Departure Date

jj/mm/aaaa

Return Date

jj/mm/aaaa

Comment

-- Departure City --

-- Return City --

Make a quote

Figure 13: Quote form(web application)

As you can see from these pictures, we have used all three languages, HTML, CSS and JavaScript to create these forms.

Now, we will show you how we created these forms. Let's take the example of "Register Now".

First part of the creation, the HTML code. It is useful to know that in the HTML language there is a tag called "form" which has the exact purpose of what we are looking for. Then, we create different "inputs", or "selects" as we can find in the "Rent" and "Quote" forms, to allow the user to enter information. The HTML code for the "Register Now" form is shown in the photo below:

```

90 <!-- Register form -->
91 <form class="Reg">
92 <!-- Text -->
93 <h1 id="register" class="Register">Register Now</h1>
94 <p id="info" class="info">Please enter the following informations</p>
95 <!-- Inputs allowing the user to insert his information -->
96 <div id="inputs">
97 <!-- Input for text -->
98 <input type="Surname" id="surname" placeholder="Surname">
99 <input type="Firstname" id="firstname" placeholder="First name">
100 <input type="email" id="email" placeholder="Email">
101 <input type="Mobile" id="mobile" placeholder="Mobile phone">
102 <input type="password" id="password" placeholder="Password">
103 </div>
104 <!-- Input as a checkbox -->
105 <input type="checkbox" id="certify" >
106 <label for="certify">I certify that I have read and accepted the website's General Conditions of User</label>
107 </div>
108 <!-- Text with span to change the color of a word in the phrase -->
109 <p id="account2" class="Account">I already have an account. <span color="#F1895C"><a href="Login.html">Log</span></a> me in.</p>
110 <!-- Button in the center of the page to create an account (Javascript to record this information) -->
111 <div align="center">
112 <button class="create" id="bouton">Create an account</button>
113 </div>
114 <!-- Javascript allowing to make appear this information containing text and buttons. The text is dynamic with the information entered by the user. -->
115 <h1 id="h1"></h1>
116 <div align="center">
117 <!-- This button allows you to erase the user's registered information as if he/she was logging out -->
118 <button class="create" id="clear">Log out</button>
119 </div>
120 <div align="center">
121 <!-- Button to go to the MyAccount page to see the registered information -->
122 <button class="account" id="myaccount"><a href="..\..\HTML\English\MyAccount.html">My Account</a></button>
123 </div>
124 </form>
125

```

Figure 14: HTML code to implement a form (Register.html)

Once this was done, we could move on to the CSS part to style the form as you can see on the previous pictures. To do this, different lines of code allow us to change the background colour of the whole form, to enlarge the size of the "input", to modify the font and many other things. We then implemented this part in our CSS code:

```

/*General style for the register page*/
header .Reg{
  justify-content: center;
  align-items: center;
  background-color: #2E3244;
  min-width: 500px;
  min-height: 500px;
  margin: 50px 400px;
  padding: 20px;
}

/*Inputs style*/
header .Reg #inputs input[id="surname"],
padding: 15px;
border-radius: 10px;
border: none;
background-color: #FFFFFF;
margin-bottom: 20px;
outline: none;
font-size: 15px;
font-family: 'Poppins', sans-serif;
display: flex;
}

/*Button style*/
header .Reg .create{
  padding: 15px 20px;
  border: none;
  border-radius: 10px;
  color: #2E3244;
  font-size: 15px;
  font-family: 'Poppins', sans-serif;
  font-weight: bold;
  background-color: #F1895C;
  cursor: pointer;
  color: #2E3244;
}

```

Figure 15: CSS code to style the different point of a form (Style.css line 735)

Then the objective was to verify the correct entry of information in each form input. To do this, we used the JavaScript language to verify the user's data input. Our code works as follows.

The user enters the information. When the user presses the button at the bottom of the form, all the checks are performed. Taking the example of the "Register Now" form, it will be necessary to enter only letters with a minimum of 2 for the surname and first name, a valid email address with an like "xxxx@xxx.xxx" is required, a phone number with only numbers and a minimum of 8 is required, a password of at least 5 characters, as well as validating the "checkbox" of the general conditions. If the user presses the button when there is an error, the user will see an alert message on his screen asking him to make the necessary changes. This JavaScript part can be found in the "Script.js" file and has the following form:

```

97  /*Validate surname*/
98  function ValidateSurname(surname,my)
99  {
100     var letters = /^[A-Za-z]+$/; //variable including all the letters
101     var surname_len = surname.value.length; //variable with the length of the surname
102     if(!surname.value.match(letters))//if there are other character than letters in the surname
103     {
104         alert('Please input alphabet characters only');
105         surname.focus();
106         localStorage.clear();
107         return false;
108     }
109     else if(surname_len < my) //if the surname size is less than 2 letters
110     {
111         alert("It should not be empty / length be more than " +my);
112         surname.focus();
113         localStorage.clear();
114         return false;
115     }
116     else //if all is good
117     {
118         return true;
119     }
120 }

```

Figure 16: JavaScript validation form Surname (Script.js)

The function shown above is intended to validate the "Surname" entry. It is then broken down into three parts. Firstly, it is necessary that the user uses only letters, then there must be a minimum of letters. Finally, if both conditions are met then the function will return "true" in order to proceed to the next validation.

```

147  /*Validate email*/
148  function ValidateEmail(email)
149  {
150     var mailformat = /^[a-zA-Z0-9_\.\-]+@[a-zA-Z0-9_\.\-]+\.[a-zA-Z]{2,3}$/; //mail aspect request
151     if(email.value.match(mailformat)) //if the email format is respected
152     {
153         return true;
154     }
155     else if(!email) //if there is no value for the email
156     {
157         alert("Please input your email");
158         localStorage.clear();
159         email.focus();
160         return false;
161     }
162     else //if the email format is not respected
163     {
164         alert("You have entered an invalid email address!");
165         localStorage.clear();
166         email.focus();
167         return false;
168     }
169 }

```

Figure 17: JavaScript validation form Email (Script.js)

This second function is also broken down into three parts. Firstly, it checks whether the format entered by the user corresponds to that of an email. If so, the function returns the value "true" to move on to the next validation. Two other checks are made to see if the user has entered an email address or if the format is wrong.

```

213  /*Validate checkbox*/
214  function ValidateCertify(certify)
215  {
216     if(certify.checked) //if the checkbox is check
217     {
218         return true;
219     }
220     else //if the checkbox is not check
221     {
222         alert('Please read the General Conditions and validate the checkbox');
223         localStorage.clear();
224         certify.focus();
225         return false;
226     }
227 }

```

Figure 18: JavaScript validation form Certify (Script.js)

Finally, another interesting check to show you is the checkbox. Indeed, the aim was to be able to validate the form only if this checkbox was validated by the user. Thanks to this function, we can then check the state of the checkbox and return the value "true" if it is validated and the value "false" if it is not.

3. Dynamic input table

We have just seen in the previous question how we set up the creation of a form using HTML and CSS and how we verify the user's data input using Javascript code. The objective now is to retrieve all the information sent by the user in order to display it dynamically in a table.

The goal of this web application is to get as close as possible to reality. Thus, we have made sure that when the user fills in the "Register Now" form, he/she sees himself/herself as connected to his/her account. Indeed, as we do not have a database, we cannot use the "Login" form allowing the user to connect to his or her personal account.

Using JavaScript and local HTML storage, we have managed to get a person to create an account. When the account is created, a .txt file will be created in the user's "Download" folder with all the information entered in the form. At the same time, in the same page, the form will be replaced by a message. In addition, two new buttons will appear and will allow the user to perform different functions that we will see later.

We are now going to show you step by step how to create a reservation in the "Rent" page:

First step: go to the form page and enter the data

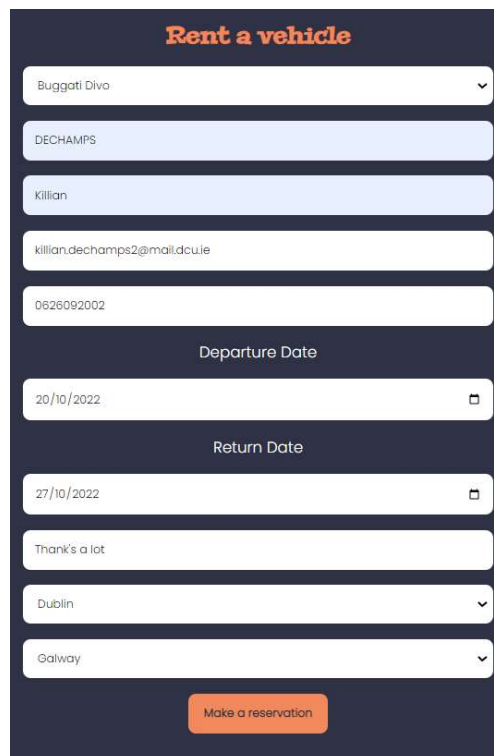


Figure 19: Picture of the Rent form (web application)

As we saw earlier, when the user presses the "Make a reservation" button, the form's check functions will run. If everything is valid, we can move on to the next point.
Second step: validate the form by pressing the "Make a reservation" button

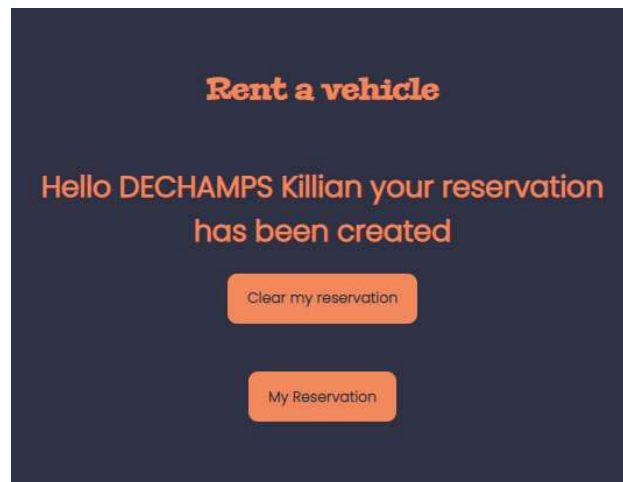


Figure 20: Rent form after pressing the button (web application)

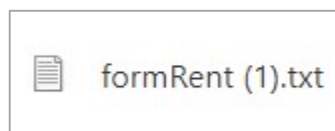


Figure 21: Creation of a text file

Third step: Press the "My reservation" button to see the dynamic table created and open the .txt file to see the information in a different way. You can also find the "My reservation" button at any time in the navigation bar on the left side.

My Reservation	
Car	BuggatiDivo
Surname	DECHAMPS
Firstname	Killian
Email	killian.dechamps2@mail.dcu.ie
Mobile	0626092002
Date Departure	2022-10-20
Date Return	2022-10-27
Comment	Thank's a lot
City Departure	Dublin
City Return	Galway
<div>Clear my reservation</div>	

Figure 22: Dynamic table (web application)

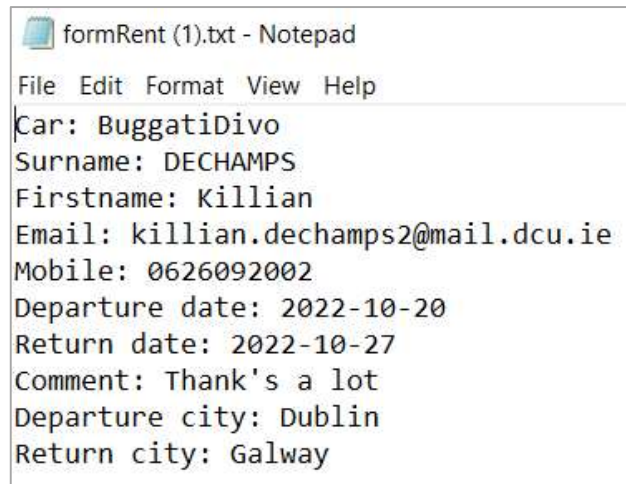


Figure 23: Information text file

Step 4: Delete the reservation

You don't have any reservation

Figure 24: Text when the form is not saved by the user (web application)

IMPORTANT NOTE: As we have seen before, we use local HTML storage to keep the values entered by the user in the various inputs. It is important to know that when you press the "Clear my reservation" or "Log out" button it will clear all the data. This means that if you are logged in to your account and you delete a reservation, you will be automatically logged out of your account. Using a database afterwards will be a good way to deal with this.

Now that we have seen the graphical aspect of saving data, let's look at the code we have created.

Firstly, we need to create a variable to store the information in local storage. Then we write in this variable all the data entered by the user in the various inputs of the form:

```

198      /*When the user click on the button*/
199      bouton.onclick = () => {
200          const user = {
201              /*We save all those value*/
202              carRent: carRent.value,
203              surnameRent: surnameRent.value,
204              firstnameRent: firstnameRent.value,
205              emailRent: emailRent.value,
206              mobileRent: mobileRent.value,
207              dateDepartureRent: dateDepartureRent.value,
208              dateReturnRent: dateReturnRent.value,
209              commentRent: commentRent.value,
210              cityDepartureRent: cityDepartureRent.value,
211              cityReturnRent: cityReturnRent.value
212          };
213          /*We set the item "user" with the value on the top*/
214          localStorage.setItem("userRent", JSON.stringify(user))
    
```

Figure 25: JavaScript function when we click on the button (Rent.html)

Once the information is recorded in the variable, we call the functions of validation:

```

216      /*We have the verification if the form is well filled*/
217      if (ValidateCar(carRent))
218      {
219          if (ValidateSurname(surnameRent, 2))
220          {
221              if (ValidateFirstname(firstnameRent, 2))
222              {
223                  if(ValidateEmail(emailRent))
224                  {
225                      if(ValidateMobile(mobileRent, 8))
226                      {
227                          if(ValidateDepartureDate(dateDepartureRent))
228                          {
229                              if(ValidateReturnDate(dateReturnRent))
230                              {
231                                  if(ValidateDepartureCity(cityDepartureRent))
232                                  {
233                                      if(ValidateReturnCity(cityReturnRent))
234                                      {

```

Figure 26: Validation function for the form (Rent.html)

Then we add the creation of the .txt file in which all the information entered by the user beforehand will be displayed.

```

/*If yes, we create a variable for the .txt file*/
let data =
'Car: ' + carRent.value + '\r\n' +
'Surname: ' + surnameRent.value + '\r\n' +
'Firstname: ' + firstnameRent.value + '\r\n' +
'Email: ' + emailRent.value + '\r\n' +
'Mobile: ' + mobileRent.value + '\r\n' +
'Departure date: ' + dateDepartureRent.value + '\r\n' +
'Return date: ' + dateReturnRent.value + '\r\n' +
'Comment: ' + commentRent.value + '\r\n' +
'Departure city: ' + cityDepartureRent.value + '\r\n' +
'Return city: ' + cityReturnRent.value;

// Convert the text to BLOB.
const textToBLOB = new Blob([data], { type: 'text/plain' });
const sFileName = 'formRent.txt'; // The file to save the data.

let newLink = document.createElement("a");
newLink.download = sFileName;

if (window.webkitURL != null) {
    newLink.href = window.webkitURL.createObjectURL(textToBLOB);
}
else {
    newLink.href = window.URL.createObjectURL(textToBLOB);
    newLink.style.display = "none";
    document.body.appendChild(newLink);
}
newLink.click();
document.location.reload();

```

Figure 27: Creation of the text file (Rent.html line 236)

Once this part is completed, the user receives his .txt file. If there was an input error, a warning message will appear, the .txt file will not be created, and the local storage will be cleared. If the form is valid, the user can then change pages and see the information entered in a dynamic table:

The JavaScript code below is used to create this dynamic table. Indeed, to do this we check if the variable containing the information is null or not. If it is not, we retrieve all the information from the local HTML storage in order to insert them into variables in our code as shown in the picture below:

```

155  /*If there is something on the local storage*/
156  if(local2 != null)
157  {
158      /*We put all of those informations in the HTML page*/
159      carRent.textContent = `${local2.carRent}`
160      surnameRent.textContent = `${local2.surnameRent}`
161      firstnameRent.textContent = `${local2.firstnameRent}`
162      emailRent.textContent = `${local2.emailRent}`
163      mobileRent.textContent = `${local2.mobileRent}`
164      dateDepartureRent.textContent = `${local2.dateDepartureRent}`
165      dateReturnRent.textContent = `${local2.dateReturnRent}`
166      commentRent.textContent = `${local2.commentRent}`
167      cityDepartureRent.textContent = `${local2.cityDepartureRent}`
168      cityReturnRent.textContent = `${local2.cityReturnRent}`
169  }
170  }

```

Figure 28: Restore the value in the local storage (MyReservation.html)

Once the information has been retrieved by the JavaScript part, we go back to the HTML code. In this one, we create an array allowing to insert the values recovered in the local HTML storage. The user will then be able to see this table in the "My Account" or "My Reservation" page in the navigation bar.

```

<!-- Table containing the information of a car reservation -->
<table id="accountTable" class="account">
  <!-- Table title -->
  <caption id="titleTable" class="table-title">My Reservation
</caption>
  <tr>
    <th>Car</th>
    <td><h1 id="carRent"></h1></td>
  </tr>
  <tr>
    <th>Surname</th>
    <td><h1 id="surnameRent"></h1></td>
  </tr>
  <tr>
    <th>Firstname</th>
    <td><h1 id="firstnameRent"></h1></td>
  </tr>

```

Figure 29: Put the information in an HTML table (MyReservation.html line 97)

Once you are fictitiously connected to an account, it is also important to be able to delete it. The "clear" function allows you to erase all the variables present in the local HTML storage. When you press the button under the information table, it will delete the account if you are in the "My account" page or delete the reservation if you are in the "My reservation" page.

Furthermore, as we saw earlier, if the user deletes one of the two tables, automatically both tables will be deleted as there is no database.

```
182  /*Button to "erase a reservation", in reality to clear the local storage*/
183  clearReservation.onclick = () =>
184  {
185      localStorage.clear();
186      document.location.reload();
187  }
```

Figure 30: Clear the local storage when you click on the button (MyReservation.html)

Thus, we have just seen the different functionalities implemented during this assignment. The new JavaScript language allows our web application to be rendered more dynamically, as we have been able to do with the navigation bars.

Moreover, this language also allows us to start implementing user accounts with inputs allowing the creation of accounts or reservations in our case, which can then be considered in databases to save all this information in a more logical way.