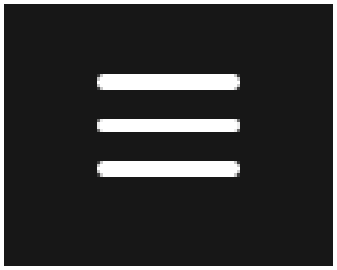
Assignment 2	
Student Name	ARSHAD HUSSAIN ASMATH
Student ID No.	18104550
Module Code	EE417
Degree	BACHELOR OF ENGINEERING
Programme	ELECTRONIC AND COMPUTER ENGINEERING(ECE)
Year	4
Date	15/2/2022

Dynamic Menu Bar

A JavaScript file called “app.js” was added to implement the dynamic menu bar. Three bars were used to replicate a small clickable menu which would expand and show all the different options available to the user.

```html
<!---Dynamic Navigation Bar--->
<a href="#" class="icon">
    <span class="bar"></span>
    <span class="bar"></span>
    <span class="bar"></span>
</a>
```



The styles used for this bar set the height, the width and radius of the bar while also setting the colour to white.

```css
.icon .bar{
    height: 3px;
    width: 100%;
    background-color: white;
    border-radius: 10px;
}
```

A different style was also added for when the bar was open (otherwise known as active). This would set the middle bar to have 0 opacity while the first and last bars would cross each other to form an “X” mark. These 2 bars were set to red colour indicating the way to close the dynamic bar and go back to the previous menu.

```css
/*Middle bar is removed/transparent*/
.icon.active .bar:nth-child(2)
{
    opacity: 0;
}
/*top bar is crossed and changed colour*/
.icon.active .bar:nth-child(1)
{
    transform:translateY(8.75px) rotate(45deg);
    background-color: #f44336;
}
/*bottom bar is crossed and changed colour*/
.icon.active .bar:nth-child(3)
{
    transform:translateY(-8.75px) rotate(-45deg);
    background-color: #f44336;
}
```

The JavaScript file was used to toggle or change the class for the navigation bar to be visible. Initially the first set of style for the navbar was set so that the contents would not be shown. The second set of active style was set that the content would be visible.

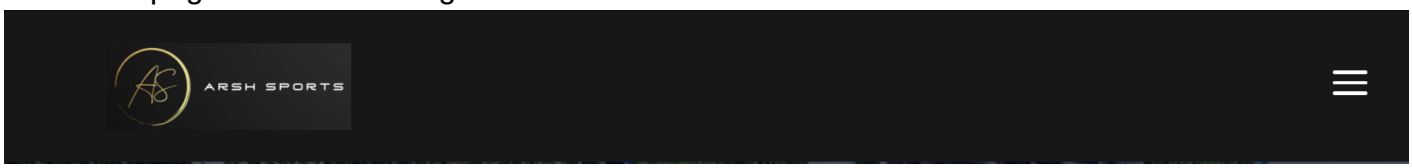
```
/*This aligns the segments under this class to the right which is the index on the navbar*/
.nav-links{
    flex: 1;
    text-align: right;
    display: none;
}
/* The contents are displayed */
.nav-links.active{
    display:contents;
}
```

The JavaScript file uses “query selector” to obtain the classes and then using an “add Event Listener” the classes of the selected queries are toggled to display the dynamic navigation bar whenever a user clicks the bar icon.

```
// Getting the selected queries and storing them in variables
const icon = document.querySelector('.icon');
const nav = document.querySelector('.nav-links');

//event listener toggles with queries when the event occurs
icon.addEventListener('click', ()=>{
    icon.classList.toggle('active')
    nav.classList.toggle('active')
})
```

When the page is loaded during start:



When the bar icon is clicked:

When the red “X” sign is clicked the menu bar reverts back to the first image.

Transition effects were added to the menu bar to showcase understanding of css. When the cursor is hovered over the different options in the navbar. A transitioning red underline was added in the css. In the picture below the cursor was hovered over “ARTICLES”.

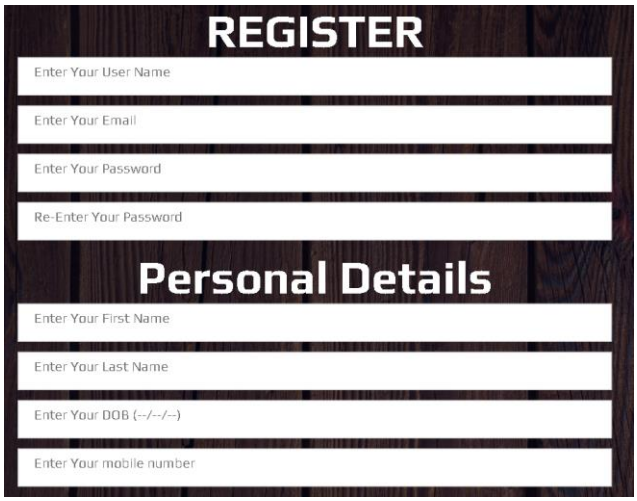
The style was also added so that when the mouse hovers over the index the cursor turns into a pointer. The styles used for this are shown below:

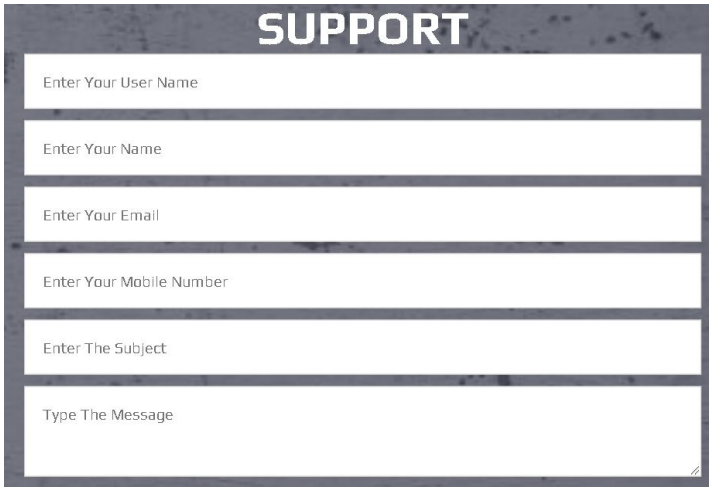
```
/*this style is for the effect on the list items when the cursor is hovered over it*/
/*the color is set to red and transition time is 0.5s*/
.nav-links ul li::after{
    content: '';
    width: 0%;
    height: 2px;
    background: #f44336;
    display: block;
    margin: auto;
    transition: 0.5s;
}
/*this takes the width of the transition to 100%*/
.nav-links ul li:hover::after{
    width: 100%;
}
```

Form Data Validation

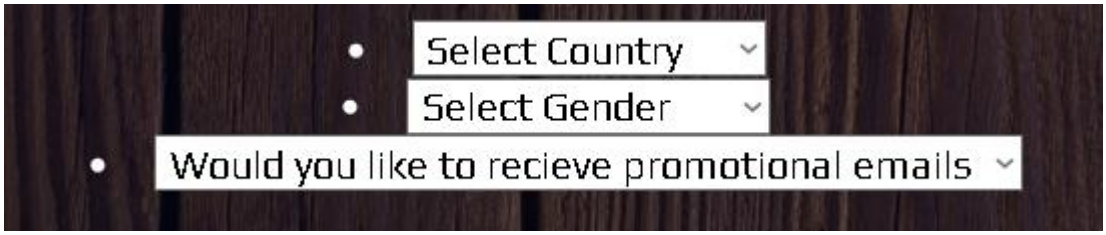
The form data validation was implemented in the 3 pages: Registration, Support and Log In. All 3 required JavaScript files of their own and implemented the same ideology.

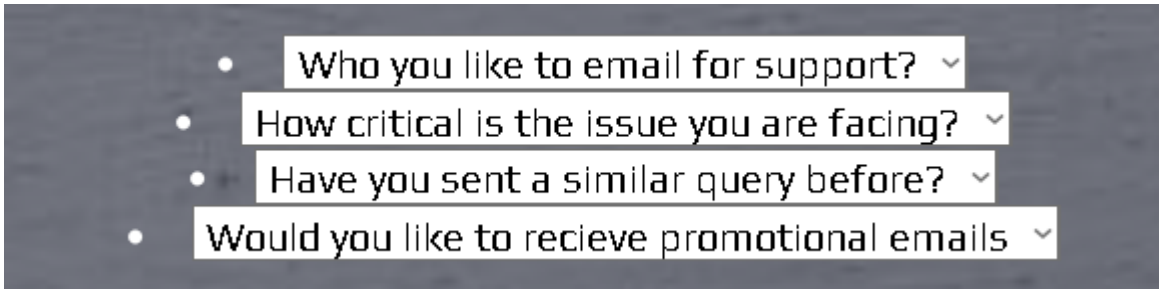
The Register page and the Support page took 10 inputs each from the user and each of these inputs was run through a validation process to check and ensure that the inputs were valid. Input boxes were used throughout except the message box in support which was used with the help of a text box and can be expandable if the message entered by the user is very long





3 drop down options were used in the registration portal and 4 drop down boxes were used in the support portal where the user could click on what they wanted.





These drop down boxes had multiple inputs ranging from 2 to 7.

Both the register and the submission page had a submit button at the end and a normal button. The submit button would submit all the input from the user if they were valid and the normal button would add the inputs from the user to a table of all the different valid submissions.
In both scenarios the validation check is carried out.

In the JavaScript file for both the panels first the values from the different text boxes were saved in local variables.

```
function formValidation(){
    var uid = document.getElementById('userName');
    var uemail = document.getElementById('email');
    var upass = document.getElementById('password');
    var upass2 = document.getElementById('password2');
    var ufname = document.getElementById('fname');
    var ulname = document.getElementById('lname');
    var udob = document.getElementById('dob');
    var umobile = document.getElementById('mobile');
    var ucountry = document.getElementById('country');
    var ugender = document.getElementById('gender');
    var upromotion = document.getElementById('promotion');
```

These local variables were then passed through functions that are specifically designed to check if the particular submission is valid or not.
The name validator, Password validator: would check if the submitted values by the user is between the range of 6 to 12 characters. If not, an error message was printed.

```
function userid_validation(uid,max,min){
    var uid_len = uid.value.length;
    if (uid_len == 0 || uid_len >= min || uid_len < max){
        alert("User ID must be between "+max+" to "+min);
        return false;
    }
    return true;
}
```

The email validator: Checks if the mail format matches the format found using online resources for emails.

```
function ValidateEmail(uemail){
    var mailformat = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;
    if(uemail.value.match(mailformat)){
        return true;
    }
    else{
        alert("Invalid Email Address");
        return false;
    }
}
```

The password validation check: Checks if password 1 and password 2 are identical or not.

```
/* Function for confirming matching passwords */
function matchPassword(upass, upass2) {
    if(upass2.value.match(upass)) {
        return true;
    }
    else {
        alert("Password did not match");
        return false;
    }
}
```

The first name and last name validation check: Checks if the names provided only have alphabetical characters.

```
function allFLetter(ufname){
    var letters = /^[A-Za-z]+$/;
    if(ufname.value.match(letters)){
        return true;
    }
    else{
        alert('First name must have alphabet characters only');
        return false;
    }
}
```

The mobile number validator: Checks if the numbers provided are above 8 to be valid.

```
function allNumber(umobile){
    var numbers = /^[0-9]+$/;
    if(umobile.value.match(numbers)){
        return true;
    }
    else{
        alert('Mobile number must have a minimum of 7 numbers');
        return false;
    }
}
```

The DOB validation check only checks if the submitted value is empty or not

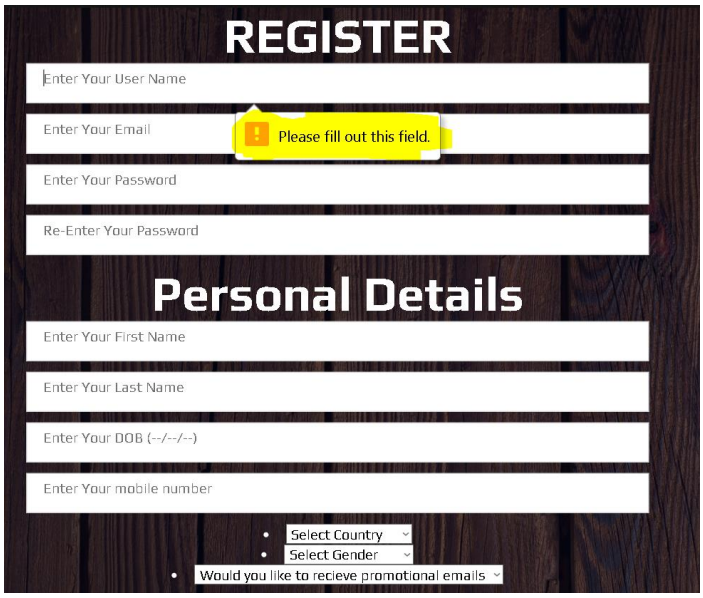
```
function dob_validation(udob){
    if(!udob.value){
        alert('Please enter a date of birth.');
        return false;
    }
    else{
        return true;
    }
}
```

The optional box validators (Country, Gender, Promotion) check if the user has selected anything different other than the default value before accepting the submission

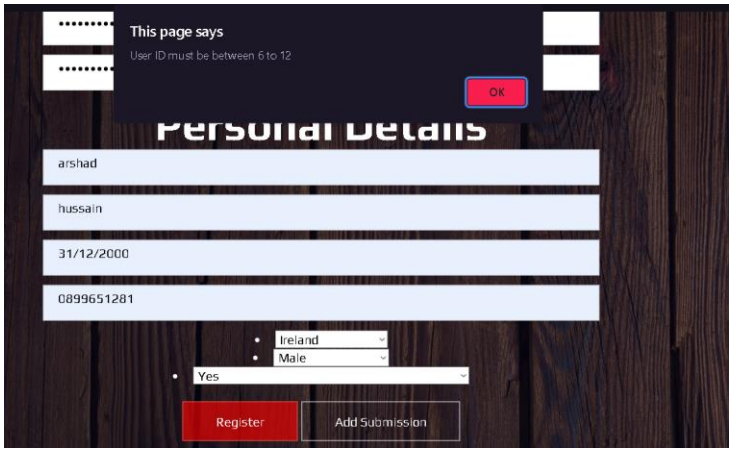
```
/* Function for validating promotion */
function promotionSelect(upromotion){
    if(upromotion.value == "Default"){
        alert('Select your option from the promotional list');
        return false;
    }
    else{
        return true;
    }
}
```

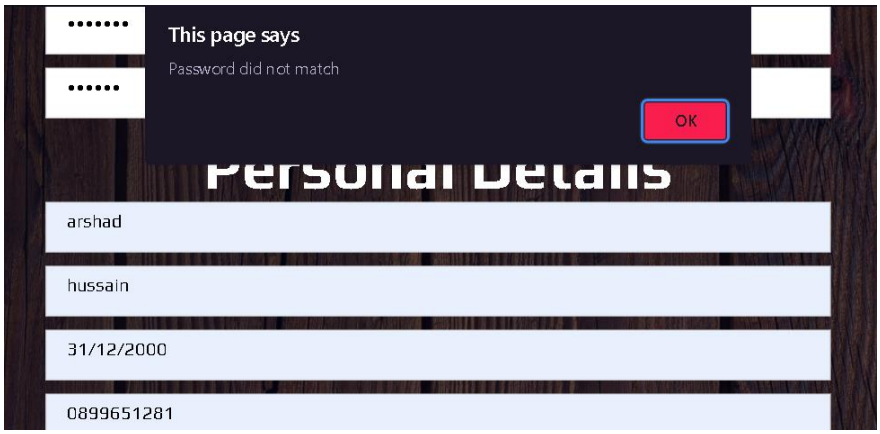
Similar types of validation checks are used in the support forms page and the log in page. When a particular value does not pass through the validation check. An alert message is displayed to the user indicating where they went wrong.

When a value is not added at all a message is displayed to the user. This is done by making all the fields as “required” in html:



Examples of alert messages:





Form Data Storage

Data Storage for the forms was done locally. A new section was added on the html side which would allow the user to display and delete the stored data. The data can be stored by clicking on the button next to submit as shown in previous photos. After undergoing the validation checks the data could be stored in the containers.

```
<section>
    <div class="submissions">
            <h1>Submissions</h1>
            <br>
            <!-- Button to view table -->
            <button type="submit" class="ytlink3" onclick="tableOfContents()">Click to View Subm
            <div id="content" class="content">
                <!-- table added with border stregnth and cell padding -->
                <table id="myTableData" className="myTableData" border="10" cellpadding="8">
                    <tr>
                        <!-- different values in the table -->
                        <td><b>Delete</b></td>
                        <td><b>Username</b></td>
                        <td><b>Password</b></td>
                        <td><b>Email</b></td>
                        <td><b>Mobile</b></td>
                    </tr>
                </table>
            </div>
        </div>
    </div>
</section>
```

This code helped in creating the table with the help of styles and centring as shown below:
The table was only visible after clicking the button present above it. The ideology for this is the same as the dynamic navigation bar where the content is first hidden and is then visible after clicking the button with the help of “Add action event listeners”.


```
function tableOfContents(){
    var tableid = document.getElementById("content");
    tableid.classList.toggle("active");
}
```

When the user clicks Add submission. The add row() function is called which first performs the validation check and then adds the submission to the table.

```
<button type="button" class="ytlink" onclick="addRow()">Add Submission</button>
```

The values are stored in local variables and then added to the inner html of the file:

```
function addRow(){
    if(formValidation()){
        var uid = document.getElementById('userName');
        var uemail = document.getElementById('email');
        var upass = document.getElementById('password');
        var upass2 = document.getElementById('password2');
        var ufname = document.getElementById('fname');
        var ulname = document.getElementById('lname');
        var udob = document.getElementById('dob');
        var umobile = document.getElementById('mobile');
        var ucountry = document.getElementById('country');
        var ugender = document.getElementById('gender');
        var upromotion = document.getElementById('promotion');

        var table = document.getElementById("myTableData");
        var rowCount = table.rows.length;
        var row = table.insertRow(rowCount);

        row.insertCell(0).innerHTML= '<input type="reset" class="reset" value = "Delete" onClick=
        row.insertCell(1).innerHTML= uid.value;
        row.insertCell(2).innerHTML= upass.value;
        row.insertCell(3).innerHTML= uemail.value;
        row.insertCell(4).innerHTML= umobile.value;
    }
}
```

Another function called delete row() helps to reset all the types, classes and the values. In doing so it also removes the values that are present in the table.

```
/* Function to delete a row of responses */
function deleteRow(obj) {
    var index = obj.parentNode.parentNode.rowIndex;
    var table = document.getElementById("myTableData");
    table.deleteRow(index);

}
```

The delete button is visible in the row when a submission is added.

When a submission is added:

Submissions

Click to View Submissions

Delete	Username	Name	Email	Mobile	Preferred Contact	Previous Contact	Critical
Delete	spdermon	Arshad	bubbycity@gmail.com	0899651281	Manager	critical	yes

When multiple submissions are added:

Submissions

Click to View Submissions

Delete	Username	Name	Email	Mobile	Preferred Contact	Previous Contact	Critical
Delete	spdermon	Arshad	bubbycity@gmail.com	0899651281	Manager	critical	yes
Delete	spdermon2	Arshad2	bubbycity2@gmail.com	0899651282	Technician	moderate	no
Delete	spdermon3	Arshad3	bubbycity3@gmail.com	0899651283	Help Desk	critical	yes

When a submission is removed using the delete button:

Submissions

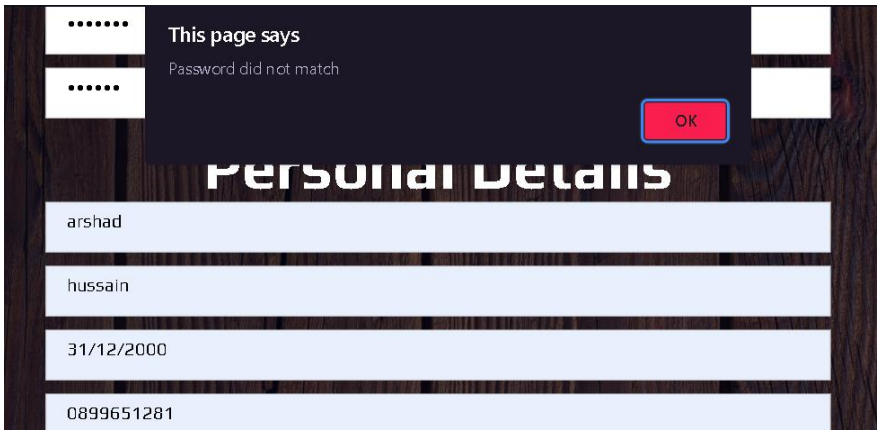
Click to View Submissions

Delete	Username	Name	Email	Mobile	Preferred Contact	Previous Contact	Critical
Delete	spdermon	Arshad	bubbycity@gmail.com	0899651281	Manager	critical	yes
Delete	spdermon3	Arshad3	bubbycity3@gmail.com	0899651283	Help Desk	critical	yes

Events Capturing

Event capturing has been used multiple times throughout the document in various places.

Alert: Alert messages have been used when creating the form data storage for errors in user input. These alert messages are displayed at the top of the html screen and can only be exited by clicking okay on them. They are triggered when a user clicks a button and form validation breaks down. Examples are in the form submission section.



EventActionListener: Event action listeners were used to change the style lists of particular elements which were hidden from the user. When a particular button is clicked with the help of event action listener and style list we are able to change the style sheet for a component. Example is the dynamic navigation bar used in the document.

```
//event listener toggles with queries when the event occurs
icon.addEventListener('click', ()=>{
    icon.classList.toggle('active')
    nav.classList.toggle('active')
})
```

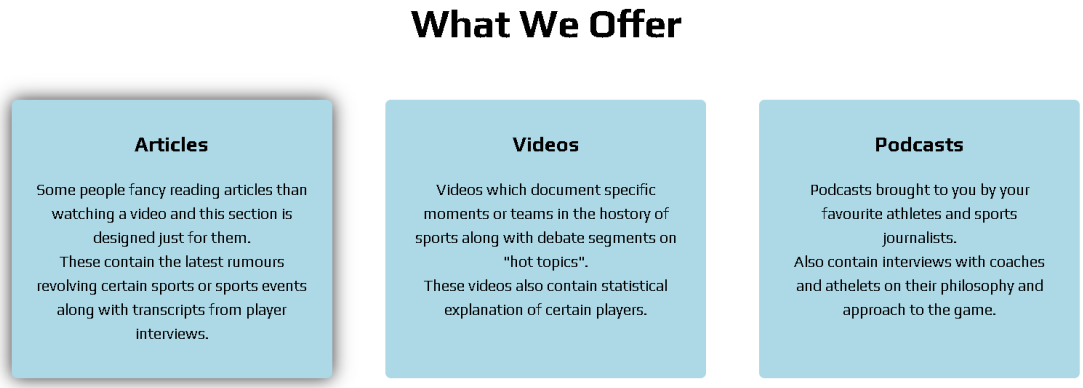
Onclick: On click elements have been used when the user can be directed to a different page upon clicking a box. The html of the page is linked on to the onclick.

```
<div class = "column" onclick="location.href = 'articles.html'">
    <h3>Articles</h3>
    <p>Some people fancy reading articles than watching a video and this sectio
        <br>These contain the latest rumours revolving certain sports or sports
    </p>
</div>
```

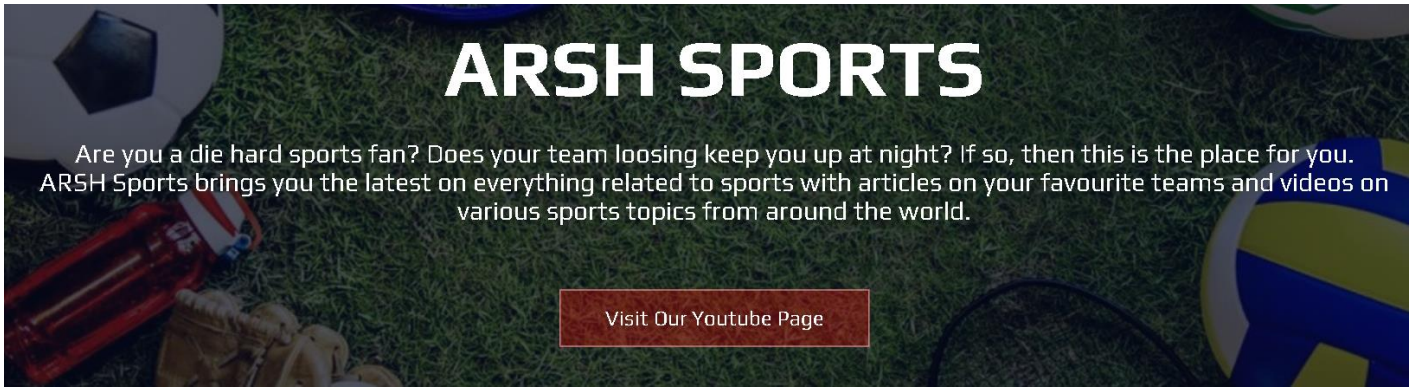
Click elements: Click elements have been used throughout the document where an action by the user triggers a particular event

```
<a href="https://theathletic.com/2823456/2021/09/14/jose-mourinho-from-game-1-to-game-1000/"
class="ytlink2">Read More on the Athletic</a>
```

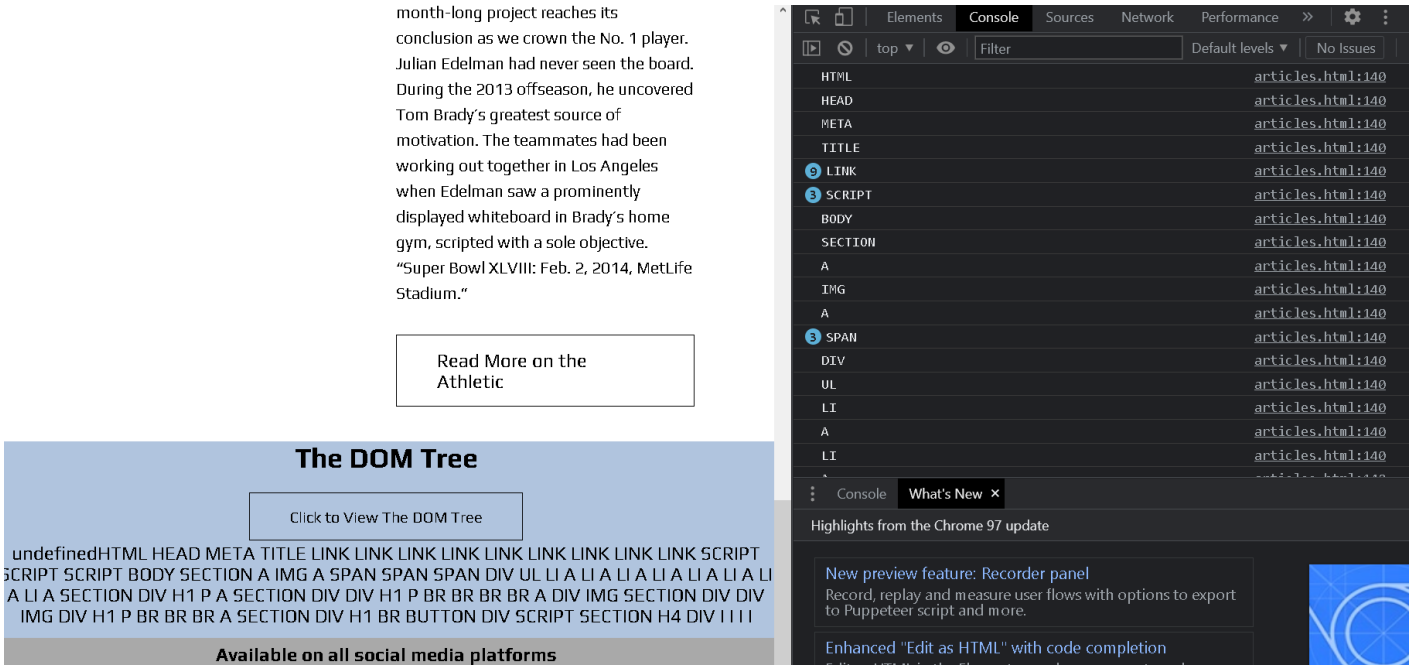
Mouse Over: Mouse over effects have also been used in this assignment. Whenever a mouse is dragged over a clickable object it changes from the default to a pointer which tells a user that they can click on the element. For example the boxed shadow here when the mouse is hovered over it.



Transition effects: Transition elements have been used in the dynamic index as well as several buttons which boost the visual aesthetic of the assignment and also show understanding of how to use css to enable transition effects. For example the button here.



JQuery: the document ready function is also an example of an event capture as it captures the html data from the website when it is loading.



DOM Tree

The DOM tree was created using JQuery. A new section was added below the article, videos and podcasts pages. This section is where the DOM tree was implimented.
After implementing the header, a button was added to ask the user if they want to display the DOM tree.

```
<div class="dom">
    <h1>The DOM Tree</h1>
    <br>
    <!-- the button to display dom -->
    <button id="btn2" name="btn2" type="button" class="ytlink3" onclick="displayhtml()">Click t
    <div class="dom2" id="dom2">
    <!-- this is where dom is displayed -->
    </div>
</div>
```

“dom2” was used to display the DOM tree.
A script method was used to invoke JQuery. This script file scanned the entire html page and got the nodeName of the different values (the html code for components). This was then printed in the “dom2”

```
<script>
    // variable to store dom html
    var temp;
    $(document).ready(function () {
        $('*').each(function(index,element){
            // var licount = $('*').length
            // logs the html values in the console
            console.log( $(this)[0].nodeName + '\n' );
            // console.log($(this)[0].nodeName);
            // the html values are added to the temp file
            temp = temp + $(this)[0].nodeName + '\n';
            // console.log($(this));
            // temp = temp + $(this);
        });
        // the values are displayed in dom2
        $('#dom2').html(temp);
        // $("#reset").click(function (e) {
        //     location.reload();
        // });
    });
</script>
```

The style sheet used is shown below. A background color was given and the text were aligned to the center.

```
.dom{
    background-color: lightsteelblue;
    text-align: center;
}

.dom2{
    background-color: lightsteelblue;
    text-align: center;
    padding-bottom: 10px;
    position: relative;
    display:none;
}
.dom2.active{
    display: flex;
}
```

A JavaScript file called “article.js” was created to act as an event listener for the button to display the DOM tree.

```
// Getting the selected queries and storing them in variables
const icon = document.querySelector('.btn2');
const dom2 = document.querySelector('.dom2');

//event listener toggles with queries when the event occurs
icon.addEventListener('click', ()=>{
    dom2.classList.toggle('active')
})

function displayhtml(){
    var change = document.getElementById(dom2);
    change.classList.toggle(active)
}
```

The DOM tree printed is shown below and is similar in articles,videos and podcasts pages: