INTERNET PROGRAMMING ASSIGNMENT

CSYM019

WEATHER APPLICATION BASED ON API DATA

TASK 2

*BY*

*NAME: DAVID OLUWAROTIMI AFOLABI*

*STUDENT ID: 20417956*

*COURSE: MSC COMPUTING (SOFTWARE ENGINEERING)*

*MODULE CODE: CSYM019*

*TUTOR: DR MUAWYA ELDAW*

**Task 2 Code Explanation**

**JavaScript Code**

// create an object called citiesCategory which contains list of countries as keys and their corresponding cities as their values.

var citiesCategory = {

    england: [

        'Bath',

        'Bristol',

        'Birmingham',

        'Bradford',

        'Bournemouth',

        'Cambridge',

        'Canterbury',

        'Chester',

        'Derby',

        'Exeter',

        'Gloucester',

        'Lancaster',

        'Leeds',

        'Liverpool',

        'London',

        'Manchester',

        'Newcastle',

        'Norwich',

        'Nottingham',

        'Oxford',

        'Plymouth',

        'Ripon',

        'Salford',

        'Sheffield',

        'Wakefield',

        'Wolverhampton',

        'Worcester',

    ],

    scotland: ['Aberdeen', 'Dundee', 'Edinburgh', 'Glasgow', 'Inverness'],

    northernIreland: ['Armagh', 'Belfast', 'Londonderry', 'Lisburn', 'Newry'],

    wales: ['Bangor', 'Cardiff', 'Newport', 'Swansea'],

};

/\*The function getCities() is defined to help load data dynamically into the city dropdown created in the html file. In this case, when a country is selected, the corresponding cities automatically generates in another dropdown list

\*/

const getCities = (value) => {

    if (value.length == 0) {

        document.getElementById('cities').innerHTML = '<option></option>'; // if there are no cities in the list, the option is set to empty

    } else {

        var cityOptions = ''; // set an empty string for the city options so that no element is displayed in the options initially

        for (city in citiesCategory[value]) {

            cityOptions += '<option>' + citiesCategory[value][city] + '</option>'; // populates the city options with cities matching their corresponding country until the loop gets to the end of the list

        }

        document.getElementById('cities').innerHTML = cityOptions; // set the city options to the generated cities in the for loop

    }

};

//declare variables and access html elements created in the html file

var button = document.querySelector('.btn'); // button created in html file is accessed using query selector to get the class name and it is stored in a variable called button

var cityInfo = document.querySelector('#cities'); //select tag in the html contains an id called cities where list of cities will be displayed. The id is accessed with query selector and stored in a variable called cityInfo

var nameOfCity = document.querySelector('.cityName'); //this variable holds a selected city name and it is accessed by query selector using its class name.

var temp1 = document.querySelector('.temperatureInFarenheit'); //this variable holds the temperature value of a selected city measured in Fahrenheit

var temp2 = document.querySelector('.temperatureInCelcius'); //this variable holds the temperature value of a selected city measured in Celcius

var severeMsg = document.getElementById('severe-weather-condition');//this variable stores the severe warning message content and it is accessed by it’s  id

var weatherCondition = document.querySelector('.weatherCondition'); //this variable holds the weather condition string value of a selected city

var windSpeed1 = document.querySelector('.windSpeedInMPH'); //this variable holds the wind speed value of a selected city measured in miles per hour

var windSpeed2 = document.querySelector('.windSpeedInKMH'); //this variable holds the wind speed value of a selected city measured in kilometre per hour

var windDirection = document.querySelector('.windDirection'); //this variable holds the wind direction value of a selected city and its unit is in degrees

var date = document.querySelector('.date'); //this variable holds the current date value when a city is selected

var icons = document.getElementById('weatherIcons'); //this variable stores the weather icons depending on the weather description

var pres = document.getElementById('pressure'); //this variable holds the pressure value of a selected city and its unit is in hpa

/\*

This function is used to convert miles per hour to kilometre per hour. This function is needed to convert the wind speed value from miles per hour to kilometre per hour

\*/

const milesToKmConverter = (val) => {

    return (val \* 1.60934).toFixed(2) + 'km/h'; //this will string concatenate the kilometre value with its unit km/h after conversion from miles per hour

};

/\*

This function is used to convert Fahrenheit to Celsius. This function is needed to convert the temperature value from Fahrenheit to Celsius.

\*/

const fahrenheitToCelsius = (val) => {

    return ((val - 32) \* (5 / 9)).toFixed(0) + '°C'; //this will string concatenate the Celsius value with its unit degree Celsius after conversion from Fahrenheit.

};

/\*

This function getWindDirection() is defined to add textual description to the wind direction. This is achieved with the help of the four cardinal points using if statement to check different coordinates on the cardinal point.

\*/

const getWindDirection = (val) => {

    if (val == 0 || val == 360) {

        return 'N'; // this will add string N (North) to the wind direction value.

    } else if (val > 0 && val < 90) {

        return 'NE'; // this will add string NE (North East) to the wind direction value.

    } else if (val == 90) {

        return 'E'; // this will add string E (East) to the wind direction value.

    } else if (val > 90 && val < 180) {

        return 'SE'; // this will add string SE (South East) to the wind direction value.

    } else if (val == 180) {

        return 'S'; // this will add string S (South) to the wind direction value.

    } else if (val > 180 && val < 270) {

        return 'SW'; // this will add string SW (South West) to the wind direction value.

    } else if (val == 270) {

        return 'W'; // this will add string W () to the wind direction value.

    } else if(val>270 && val<360) {

        return 'NW'; // this will add string NW (North West) to the wind direction value.

    }

};

/\*

The function getGeneralCondition() checks if the temperature is within some defined range otherwise a severe weather warning is indicated. It takes in two parameters which are the temperature and wind speed parameters.

\*/

const getGeneralCondition = (val\_temp, val\_wind) => {

    val\_temp = Number(val\_temp); // wrap temperature val using Number wrapper class and store in a variable called val\_temp

    val\_wind = Number(val\_wind); // wrap wind speed val using Number wrapper class and store in a variable called val\_wind

/\*The if statement is used to check if temperature value is below -5 or above 35 degree Celsius or if the wind speed is greater than 50 miles per hour so as to display a warning message\*/

    if (val\_temp > 35 || val\_temp < -5 || val\_wind > 50) {

        return 'Severe Weather!'; // returns a string indicating there is a sever weather.

    } else {

        return ''; //returns an empty string.

    }

};

/\*

This function weatherData() helps to dynamically load specified weather parameters by accessing their html contents and updating the contents with the parameters that will be retrieved from the weather API.

\*/

const weatherData = (data) => {

    var nameVal = data.name; //variable that stores the name of a selected city

    var tempVal = data.main.temp; //variable that stores the temperature of a selected city

    var presVal = data.main.pressure; //variable that stores the pressure of a selected city

    var weatherConditionVal = data.weather[0].description; //variable that stores the weather description of a selected city

    var windSpeedVal = data.wind.speed; //variable that stores the windspeed value of a selected city

    var windDirectionVal = data.wind.deg; //variable that stores the wind direction value of a selected city

    var cloudICons = data.weather[0].icon; //variable that stores the cloud icon of a selected city

    //set inner html values

    nameOfCity.innerHTML = nameVal; //the html element holding the name of a selected city is updated to the name of the selected city as retrieved from the API

    temp1.innerHTML = tempVal.toFixed(0) + '°F'; //the html element holding the temperature of a selected city is updated to the temperature of the selected city as retrieved from the API and concatenated with a string “°F” which is the unit of measurement.

    temp2.innerHTML = fahrenheitToCelsius (tempVal); //the html element holding the Fahrenheit temperature of a selected city is updated and the temperature value gotten from the API which was originally in Fahrenheit is parsed into the farenheitToCelcius() method to convert the temperature value from degree Fahrenheit to degree Celsius.

    severeMsg.innerHTML = getGeneralCondition(fahrenheitToCelsius (tempVal), windSpeedVal);//the html element holding the severe message is updated. The getGeneralCondition() method is called here and the temperature value in Celsius and wind speed value in miles per hour is parsed into the method.

    weatherCondition.innerHTML = weatherConditionVal; //the html element holding the weather description of a selected city is updated to the weather description of the selected city as retrieved from the API

    date.innerHTML = new Date().toDateString(); //the html element holding the current date when the city was accessed is updated using the new Date() method and it is displayed in string format using the toString method.

    windSpeed1.innerHTML = windSpeedVal.toFixed(2) + 'mph'; //this concatenates the windspeed value to its unit which is miles per hour. toFixed() method is used along with the wind speed value to round up the number to two decimal places

    windSpeed2.innerHTML = milesToKmConverter(windSpeedVal); // wind speed that is retrieved from the API is converted from miles per hour to kilometre per hour using the milesToKmConverter() method and it is updated in the html element.

    windDirection.innerHTML =

        windDirectionVal + '° ' + getWindDirection(windDirectionVal); //wind direction retrieved from the API is concatenated with its unit and the textual description of the wind direction using the getWindDirection() method and parsing the wind direction value into the method. This is then updated in the html element holding the wind direction value.

    pres.innerHTML = presVal + ' hPa'; //this string concatenates the pressure value from the API to its unit and then gets updated in the html page

    icons.setAttribute(

        'src',

        `http://openweathermap.org/img/wn/${cloudICons}@2x.png`

    ); //this is used to set the source for the cloud icon using the setAttribute() method which was originally set in the html page as an empty source. The source is gotten from open weather map icon API. Cloud icons is parsed template literals.

    document.getElementById('other-details').style.visibility = 'visible'; // this will make the div containing other information about the weather condition that was created in the html page to become visible. It was initially set hidden using CSS.

    console.log(data); //display the data retrieved from the API to the console.

};

/\*The function getCityWeather() is used  to fetch weather details from open weather map API.

The fetch method is used here to access the resources in the json file.

 The fetch method handles request and response to the web browser.

 First it makes a request to the server, then it returns a promise that resolves into a response object.

 The then function handles the data while the catch function handles any error that may occur while trying to load data.

  \*/

const getCityWeather = () => {

    const apiUrl =

        'https://api.openweathermap.org/data/2.5/weather?q=' +

        cityInfo.value +

        '&appid=7e4547eeb4a8dbf3ed8cb577e5996225&units=imperial'; //the API is called here and stored in a constant variable called apiUrl. cityInfo.value  is concatenated in the API so as to dynamically load the weather conditions for each city in the list. The unit of the measurements for the parameters in API is imperial.

//the fetch method takes in a compulsory argument which is the apiUrl in this case

fetch(apiUrl).then((response) => {

//the response is accessed here and it is important to ensure that the response status is 200 using an if statement

            if (response.status !== 200) {

                console.log("Error occured: " + response.status); // display error on the console if response status is not 200

            } else {

                return response.json(); // return the json object of the output if response status is satisfied

            }

        })

        .then((data) =>

/\*The text in the reponse is being examined here.

The weatherData() method is called here and takes in data as an argument

The data displayed consists of the defined attributes in the weatherData() method and it is matched with attributes retrieved from the API.

\*/

        weatherData(data) // this function displays the weather details of cities)

        .catch((error) => console.log('Error loading data' + error)))

    } //display error message to the console };

/\*Button function

The button handles a click event using the addEventListener() method and takes in the getCityWeather() function as a call-back function which displays the weather details of a selected city.

\*/

button.addEventListener('click', getCityWeather)

**HTML Code**

<!DOCTYPE html> <!--html DOCTYPE declaration which informs the browser what kind of document it is expecting.-->

<html lang="en"> <!--this indicates the language of the contents in the html. It is English in the case.-->

<!--Head tag-->

<head>

<!--meta tags are used in the head tags to describe the page’s content-->

    <meta charset="UTF-8" /> <!--UTF-8 character encoding is used here.-->

    <meta http-equiv="X-UA-Compatible" content="IE=edge" /> <!--this is used to select the IE=edge as the the version of internet explorer when the html content will be rendered.-->

    <meta name="viewport" content="width=device-width, initial-scale=1.0" /> <!--this is used to set the page width depending on the device on which it is loaded.-->

    <title>Uk Weather</title> <!--set page title -->

    <link rel="stylesheet" href="style.css" /> <!--this links the style.css file to the html page-->

</head>

<!--body tag-->

<body>

    <!--heading-->

    <div class="nav-bar"> <!--div to handle the navbar with class name set as nav-bar-->

        <h1>

            <a href="apiweather.html"> <!--this link reloads the html page when clicked-->

                <h1>UK weather forecast</h1> <!--nav-bar text set with an h1 tag-->

            </a>

        </h1>

    </div>

    <!-- end of heading-->

    <!--dropdown list of countries-->

    <div id="countries-container"> <!--div element that containing the dropdown list of countries-->

        <select name="countries" id="countries" onchange="getCities(this.value)"> <!--onchange() function is called here to automatically load the cities matching a selected country.-->

        <option selected disabled>Select a Country</option><!--this serves as a placeholder that cannot actually be selected using the ‘selected disabled’ property-->

        <option value="england">England</option> <!--set England as an option in the dropdown list with its value-->

        <option value="wales">Wales</option><!--set Wales as an option in the dropdown list with its value-->

        <option value="scotland">Scotland</option> <!--set Scotland as an option in the dropdown list with its value-->

        <option value="northernIreland">Northern Ireland</option> <!--set Northern Ireland as an option in the dropdown list with its value-->

      </select>

    </div>

    <!-- end of dropdown list of countries-->

    <!--dropdown list of cities-->

    <div id="cities-container"> <!--div element holding the dropdown list of cities-->

        <select name="cities" id="cities"> <!--select tag is given id and name-->

        <option selected disabled>Select a City</option> <!--serves as a placeholder that cannot be selected using the ‘selected disabled’ property.-->

        <option value="" id="list-of-cities"></option> <!--set options empty which will be dynamically loaded using the id in the javascript-->

      </select>

    </div>

<!--button text is set within the button tag and the button has an onclick function which when clicked, generates the weather details-->

    <button type="submit" onclick="getCityWeather()" class="btn">

      Get Weather Details

    </button>

    <!--end of dropdown list of cities-->

    <!--weather information-->

    <div id="weather-info"> <!--div element that holds the weather information generated from the API-->

        <h4>

            <span id="severe-weather-condition" style="color:rgb(131, 9, 9)"></span> <!--span tag which has been styled using inline CSS to display severe weather warning message and will be dynamically generated using JavaScript-->

        </h4>

        <h4 class="cityName"></h4> <!--h4 element to display city name-->

        <h1 class="temperatureInCelcius" id="temp-in-celcius"></h1> <!--h1 element to display temperature in Celsius.-->

        <img id="weatherIcons" src="" alt="" /> <!--image element to display weather icons which is dynamically loaded in the JavaScript file-->

        <p class="date"></p>

    </div>

    <div id="other-details"> <!--div element containing other weather information which is set hidden using CSS and set visible using JavaScript-->

        <h4 class="temperatureInFarenheit"></h4> <!--h4 element to display temperature in Fahrenheit-->

        <p class="weatherCondition"></p> <!--paragraph element to display weather description-->

        <p class="windSpeedInMPH"></p> <!--paragraph element to display windspeed in miles/hour-->

        <p class="windSpeedInKMH"></p> <!--paragraph element to display windspeed in km/hour-->

        <p class="windDirection"></p> <!--paragraph element to display wind direction-->

        <p id="pressure"></p> <!--paragraph element to display pressure-->

    </div>

    <!--end of weather information-->

    <script src="apiweather.js"></script> <!--script tag that links the JavaScript file to the HTML-->

</body>

<!--end of html tag-->

</html> <!--end of body tag-->

**CSS code**

/\*This is used to style the body of the webpage including background pictures, colours, font styles and margins\*/

body {

    background: url("../img/background.png");

    background-size: cover;

    background-repeat: no-repeat;

    margin: 0;

    --white: #ffffff;

    --red: rgb(190, 29, 29);

    --blue: #272759;

    --dark: #333;

    --gray: lightgray;

    font-family: 'Poppins', sans-serif;

    text-transform: capitalize;

}

/\*This is used to style the navbar of the webpage\*/

.nav-bar {

    overflow: hidden;

    background-color: var(--dark);

    color: var(--white);

}

/\*This is used to style the link text of the navbar of the webpage\*/

.nav-bar h1 a {

    color: var(--white);

    text-decoration: none;

    text-align: center;

    font-size: 1.2rem;

}

/\*This is used to style the hover effect of the navbar text of the webpage\*/

.nav-bar h1 a:hover {

    font-style: italic;

    text-decoration: underline;

}

/\*This is used to style the select tag of the countries\*/

#countries-container select {

    left: 39.24%;

    right: 50.69;

    top: 20%;

    bottom: 78.56%;

    background: #000000;

    border-radius: 15px;

    height: 50px;

}

/\*This is used to style the select tag of the cities\*/

#cities-container select {

    left: 51%;

    right: 39.17%;

    top: 20%;

    bottom: 78.56%;

    background: #000000;

    border-radius: 15px;

    height: 50px;

}

/\*This is used to set the width and position of the options in both the cities and countries dropdown list\*/

#countries-container select,

#cities-container select,

option {

    position: absolute;

    color: var(--white);

    width: 150px;

}

/\*This is used to style the button\*/

button {

    position: absolute;

    left: 39.24%;

    right: 39.17%;

    top: 30%;

    bottom: 70.44%;

    background: #c5940e;

    border-radius: 15px;

    padding-top: 20px;

    padding-bottom: 30px;

    font-weight: bold;

    width: 21rem;

    text-align: center;

    border: none;

}

/\*This is used to style the second child(city name element) of the weather-info div\*/

#weather-info :nth-child(2) {

    position: absolute;

    color: var(--white);

    align-items: center;

    font-weight: 600;

    font-size: 76px;

    width: 299px;

    height: 115px;

    left: 670px;

    display: flex;

    top: 265px;

    padding-left: 150px;

    line-height: 172px;

    letter-spacing: -0.045em;

}

/\*This is used to style the third child (temperature in Celsius) of the weather-info div\*/

#weather-info :nth-child(3) {

    position: absolute;

    color: var(--white);

    align-items: center;

    font-size: 130px;

    width: 91px;

    height: 172px;

    left: 530px;

    display: flex;

    top: 270px;

    line-height: 172px;

    letter-spacing: -0.045em;

}

/\*This is used to style the fourth child (weather icons) of the weather-info div by setting margin positions\*/

#weather-info :nth-child(4) {

    margin-top: 220px;

    margin-left: 450px;

}

/\*This is used to style the last child (date) of the weather-info div\*/

#weather-info :last-child {

    position: absolute;

    font-family: 'Noto Sans JP', sans-serif;

    width: 200px;

    height: 29px;

    left: 850px;

    top: 445px;

    font-weight: 600;

    font-size: 24px;

    line-height: 29px;

    color: var(--white);

    align-items: center;

    display: flex;

    letter-spacing: -0.045em;

}

/\*This is used to style the other-details div which contains other weather parameters\*/

#other-details {

    visibility: hidden;

    color: var(--white);

    margin-top: 120px;

    margin-left: 540px;

    width: 32%;

    height: 120px;

    border-style: hidden;

    border-radius: 20px;

    background: linear-gradient(rgba(0, 0, 0, 0.6), rgb(0, 0, 0, 0.6));

}

/\*This is used to style the first child (temperature in Fahrenheit) of the other-details div\*/

#other-details :nth-child(1) {

    position: absolute;

    width: 60px;

    height: 29px;

    left: 580px;

    top: 530px;

    font-size: 30px;

    display: flex;

    align-items: center;

    letter-spacing: -0.02rem;

}

/\*This is used to style the second child (weather condition) of the other-details div\*/

#other-details :nth-child(2) {

    position: absolute;

    width: 89px;

    height: 29px;

    left: 700px;

    top: 600px;

    align-items: center;

    letter-spacing: -0.02rem;

}

/\*This is used to style the third child (wind speed in mils/hour) of the other-details div\*/

#other-details :nth-child(3) {

    position: absolute;

    width: 89px;

    height: 29px;

    left: 900px;

    top: 555px;

    align-items: center;

    letter-spacing: -0.02rem;

}

/\*This is used to style the fourth child (wind speed in km/hour) of the other-details div\*/

#other-details :nth-child(4) {

    position: absolute;

    width: 89px;

    height: 29px;

    left: 700px;

    top: 555px;

    align-items: center;

    letter-spacing: -0.02rem;

}

/\*This is used to style the fifith child (wind direction) of the other-details div\*/

#other-details :nth-child(5) {

    position: absolute;

    display: flex;

    height: 29px;

    left: 900px;

    top: 600px;

    align-items: center;

    letter-spacing: -0.02rem;

}

/\*This is used to style the last child (pressure) of the other-details div\*/

#other-details :last-child {

    text-transform: none;

    position: absolute;

    display: flex;

    height: 29px;

    left: 580px;

    top: 595px;

    align-items: center;

    letter-spacing: -0.02rem;

}

/\*The span tag for the severe weather message is styled here\*/

#weather-info>h4>span {

    margin-top: 71px;

    margin-left: -250px;

}

/\*This is used to handle bigger screens

The position of the button is handled to align properly on bigger screens

\*/

@media only screen and (max-width: 1920px) {

    button {

        position: absolute;

        left: 40.54%;

        right: 39.17%;

        top: 30%;

        bottom: 70.44%;

        background: #c5940e;

        border-radius: 15px;

        padding-top: 20px;

        padding-bottom: 30px;

        font-weight: bold;

        width: 21rem;

        text-align: center;

        border: none;

    }

}

/\*This is used to handle bigger screens

The position of the button is handled to align properly on laptop screens

\*/

@media only screen and (max-width: 1536px) {

    button {

        position: absolute;

        left: 39.24%;

        right: 39.17%;

        top: 30%;

        bottom: 70.44%;

        background: #c5940e;

        border-radius: 15px;

        padding-top: 20px;

        padding-bottom: 30px;

        font-weight: bold;

        width: 21rem;

        text-align: center;

        border: none;

    }

}