THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

DEPARTMENT OF
COMPUTER SCIENCE
Te Tari Rorohiko

2024

Contributors

J. Turner

R. Mercado

V. Moxham-Bettridge

A. Hinze

C. Pilbrow

N. Kanji

T. Elphick

S. Cunningham

INTRODUCTION TO WEB DEVELOPMENT

Last time we looked at an introduction to Mobile Computing using Android Studio for smartphone application development. This week we will look at web development, with a focus on web sites rather than web applications.

Website vs. Web Application

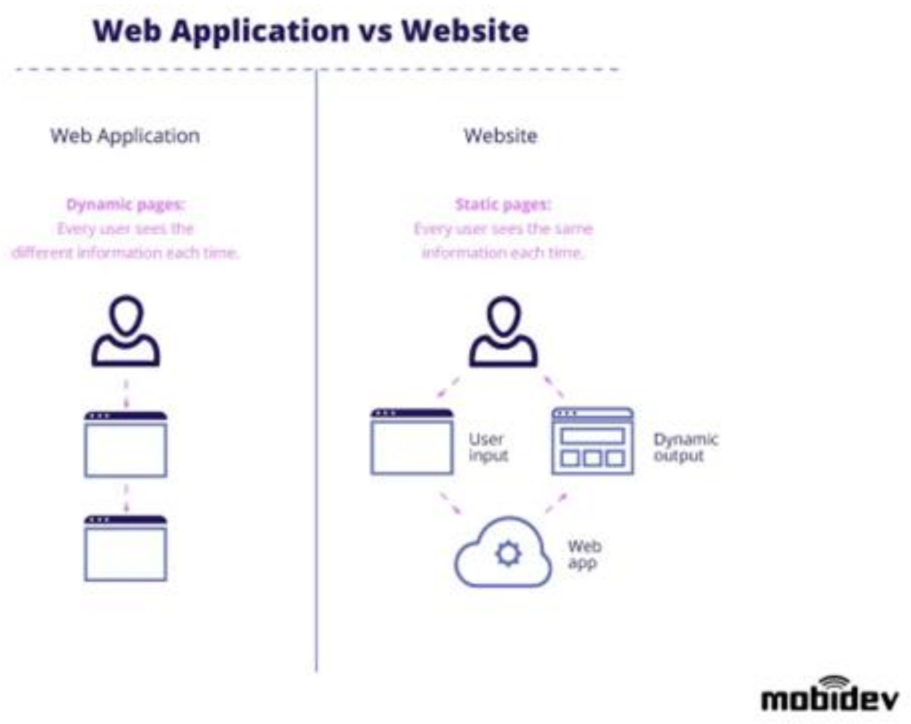

Web Application vs Website
Web Application
Website
Dynamic pages:
Every user sees the different information each time.
Static pages:
Every user sees the same information each time.
User input
Dynamic output
Web app
mobidev

Figure 49: Web Application vs. Website
Retrieved from: https://mobidev.biz/blog/web-application-architecture-types

In Figure 49 a standard definition of the difference between a website and web application is given. A website is static, in that the user sees the same information each time they go to a particular web address, while in contrast a web application is dynamic, in that the content changes for every user. For example, consider tonystyreservice.co.nz, no matter which user visits the web page the pages are static (i.e. the same). Contrast this with google documents as an example, each user has their own set of documents and can modify/change them. There are benefits to being able to write both types of systems, but for this session we will focus on websites.

What is a website?

A website is made up of different components and languages which interact with each other in order to display information in a browser. Consider Figure 50 which depicts a simplified web architecture. The typical components of the software architecture we “see” are the user interface displayed in a browser. However, behind the scenes we have code to manage the web server which hosts the website; code for handling the front end and back end systems; code for managing the database and lastly, actual data. Note that this structure does not consider security, which is yet another set of components for the application which protect against security threats.

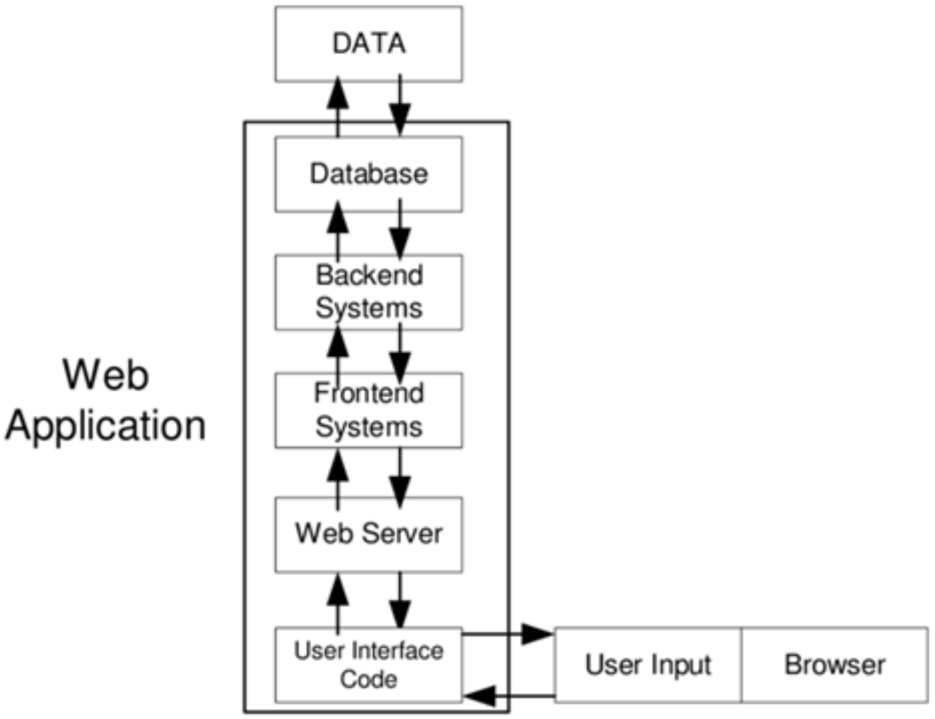


Figure 50: Web Architecture Components
Retrieved from: https://www.researchgate.net/figure/Web-Application-Components_fig1_228598161

While there are different types of architecture styles for a website, most still use the 3-tier system (see Figure 51). This simple architecture has three different layers, each of which may be implemented using different languages.

The presentation layer (or frontend) uses HTML to display information in a browser while CSS is used to define how it is displayed. Javascript (not to be confused with Java!) is used in the presentation layer to dynamically change the appearance of a webpage.

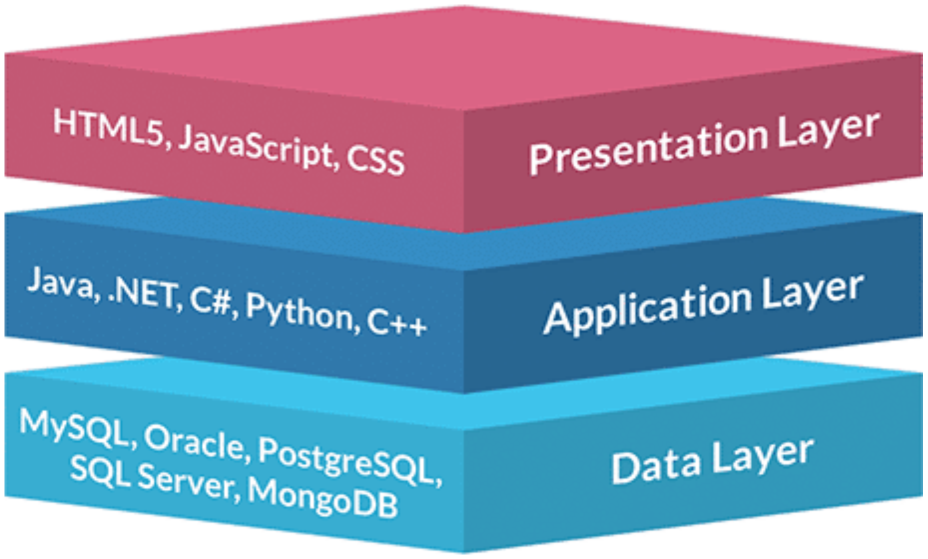

HTML5, JavaScript, CSS
Presentation Layer
Java, .NET, C#, Python, C++
Application Layer
MySQL, Oracle, PostgreSQL, SQL Server, MongoDB
Data Layer

Figure 51: 3-Tier Architecture in Web Development
Retrieved from: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/

The application layer handles the functionality or business logic of the application, this can be implemented in any programming language but common examples include Java, .NET, C#, Python and/or C++. It drives the core capabilities of the website.

Finally the data tier stores all the necessary data for the application. This includes the database management system as well as the access to that data. Data is usually accessed via an Application Programming Interface (API) which allows us to only expose the information from a database that is allowed to be accessed.

Full web development requires knowledge of different programming languages, software architecture, database management, servers, networking, security, user interface design, user experience, human computer interaction, algorithm design, optimisation and so on. Fortunately, these systems are not all managed by one person, hence the need for a diverse team of developers in order to build these types of systems.

Single Page Application (SPA) Architecture

For today's exercise we are going to look at using an SPA structure (see figure 52). The presentation layer is considered to be a part of the client side of the application which allows for quick rendering of a website (i.e. the website loads quickly in the browser), while the application (business) layer and data layer are managed by the server side.

We will use HTML, CSS and Javascript to implement the presentation layer i.e. front-end of the application.

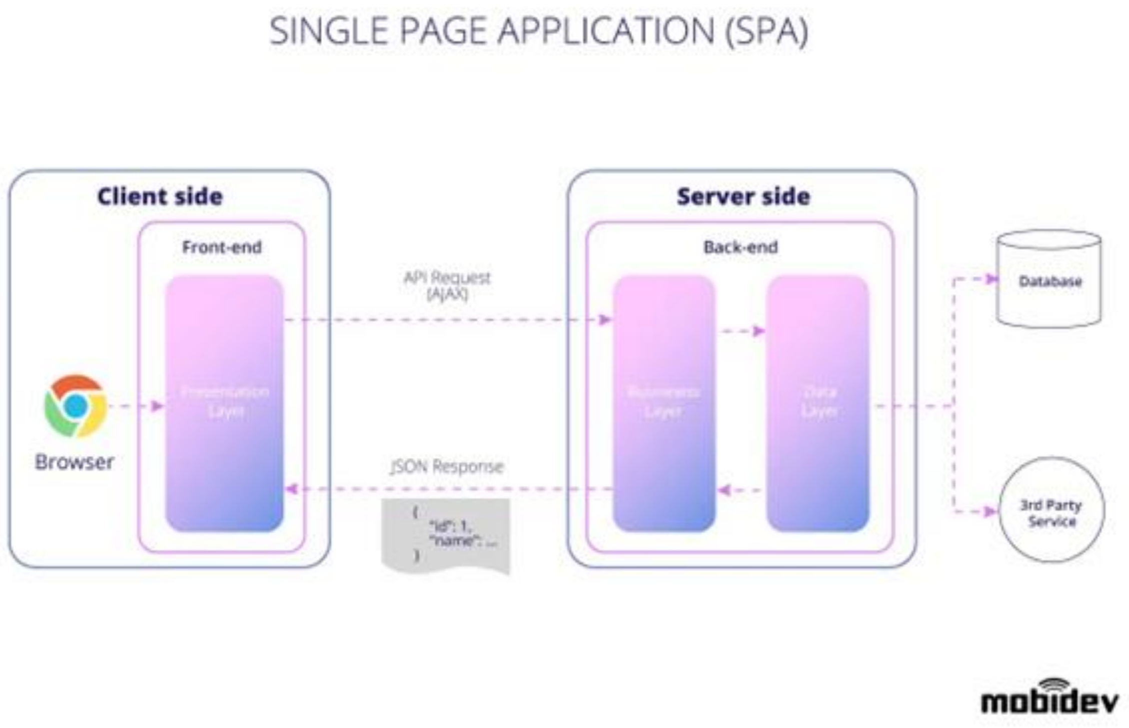


Figure 52: SPA Architecture

The Weather Monitor

For today's exercise we are going to implement a website which displays basic weather information for cities in New Zealand. It will also display webcam images of the weather so that users can see an illustration of what the weather looks like outside (see figure 53).

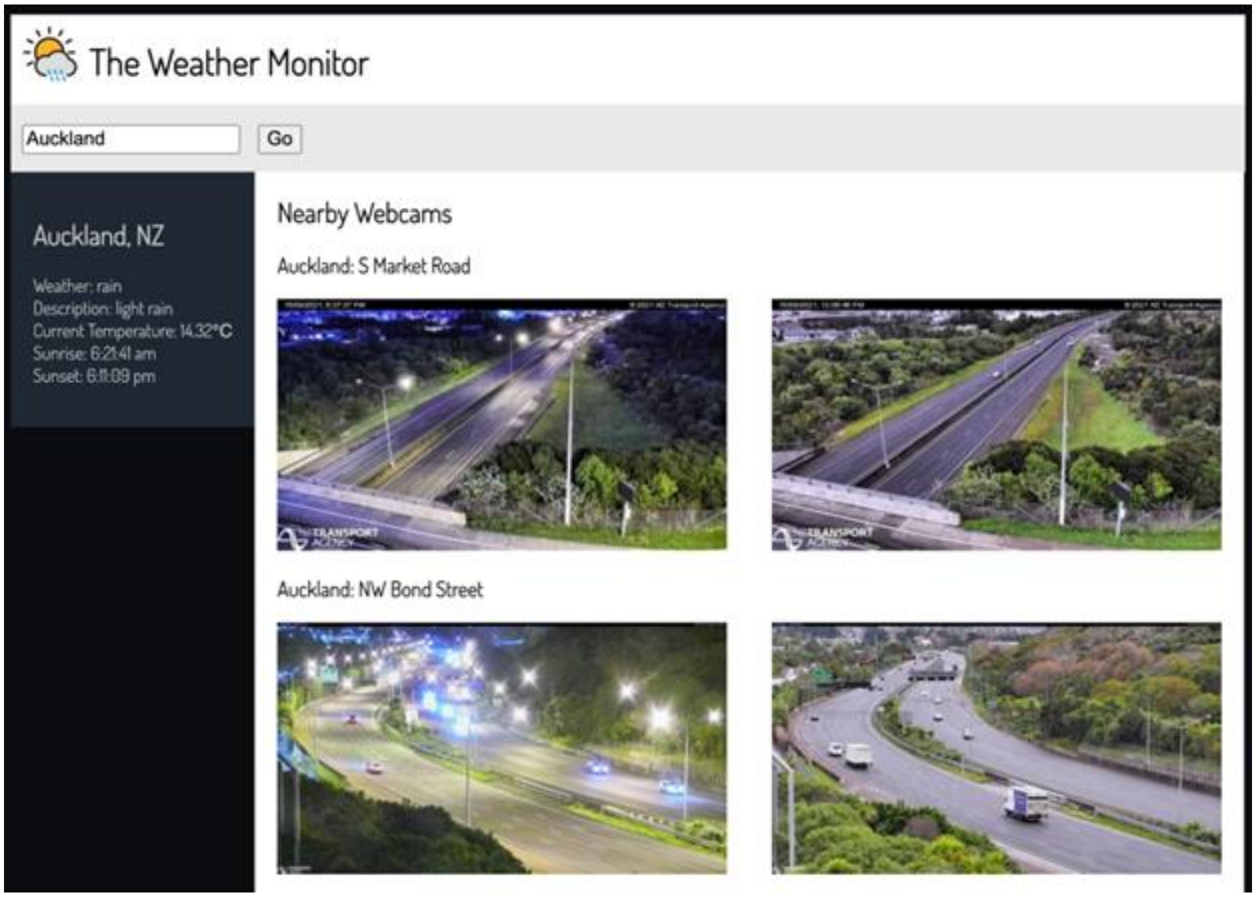
This application uses two REST API's to collect information based on a city search and display that information. A REST API transfers information to an entity (in this case a website) in a recognisable format, for example JSON or XML[8].

To do this we will make an AJAX request to the REST API. An AJAX request simply bundles our query in a specified format so that it can be understood by the API. We will then process the response from the request and dynamically build the display for the information provided.

Exercises:

1. Download the template from the Slack channel and familiarize yourself with each file. Can you work out how the CSS and Javascript files are linked to the HTML webpage?

2. Read through the Javascript file and ensure you understand what is happening.



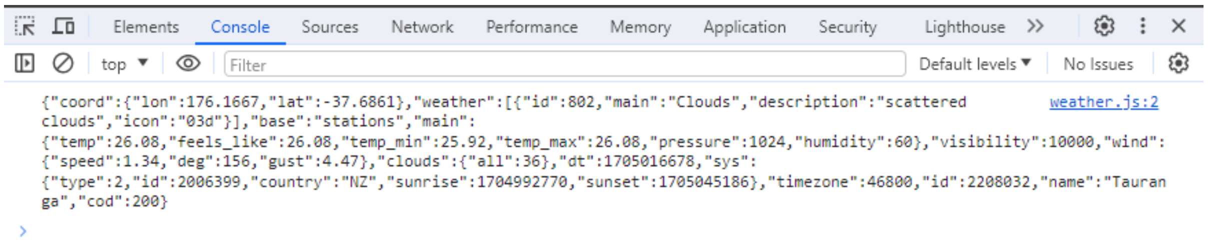
Auckland, NZ

Weather: rain
Description: light rain
Current Temperature: 14.32°C
Sunrise: 6:21:41 am
Sunset: 6:11:09 pm

Figure 53: The Weather Monitor

While the template handles the AJAX request for us we need to fill in the code to handle the response. The good news is that both requests return information in a JSON format and Javascript has a standard way of processing JSON.

3. First, try simply logging the output of the response using "`console.log(response);`" inside the displayWeather function. You can view the console output by right clicking on the HTML page in a web browser, selecting "Inspect" and the console option (see figure 54).



```
{"coord":{"lon":176.1667,"lat":-37.6861},"weather":[{"id":802,"main":"Clouds","description":"scattered
clouds","icon":"03d"}],"base":"stations","main":
{"temp":26.08,"feels_like":26.08,"temp_min":25.92,"temp_max":26.08,"pressure":1024,"humidity":60},"visibility":10000,"wind":
{"speed":1.34,"deg":156,"gust":4.47},"clouds":{"all":36},"dt":1705016678,"sys":
{"type":2,"id":2006399,"country":"NZ","sunrise":1704992770,"sunset":1705045186},"timezone":46800,"id":2208032,"name":"Tauran
ga","cod":200}
```

Figure 54: Console Output of API Call

Notice the structure of the response, there is a "name" at the top level of the JSON response (it looks like it's at the bottom of the response but it actually isn't), while to get access to a "country" we have to first access "sys". This translates directly to how we access the information in Javascript. Also note square brackets "[]" depict a JSON array.

4. Parse the JSON response:
   `var data = JSON.parse(response);`

5. Get the information from the response, for example, to get the name use "`var city = data.name;`". Can you work out how to get the other information? We will need the country, weather main, weather description, temperature, sunrise, sunset, latitude and longitude.

6. Inside the HTML you should have seen two divs, "weather-container-1" and "weather-container-2", we will populate these with our information. To get the container use the following:
   ```
   let container_element = document.getElementById("weather-
   container-1");
   ```

7. Clear any potential information in the div:
   `container_element.innerHTML = "";`

8. Create a header element at level 2 (h2):
   `let header = document.createElement("h2");`

9. Set the header to display the name of the city and the country i.e. Tauranga, NZ:
   `header.innerHTML = city + "," + country;`

10. Append the header to the container_element so that it will be displayed:
    `container_element.appendChild(header);`

11. Try opening the website in a browser and doing a search for Tauranga, do you see the expected information displayed?

12. Can you work out how to display the rest of the information? Each time you add a new child check the website in the browser.

13. When you have all the information displayed call the getWebcams function: `getWebcams(lat,long);`

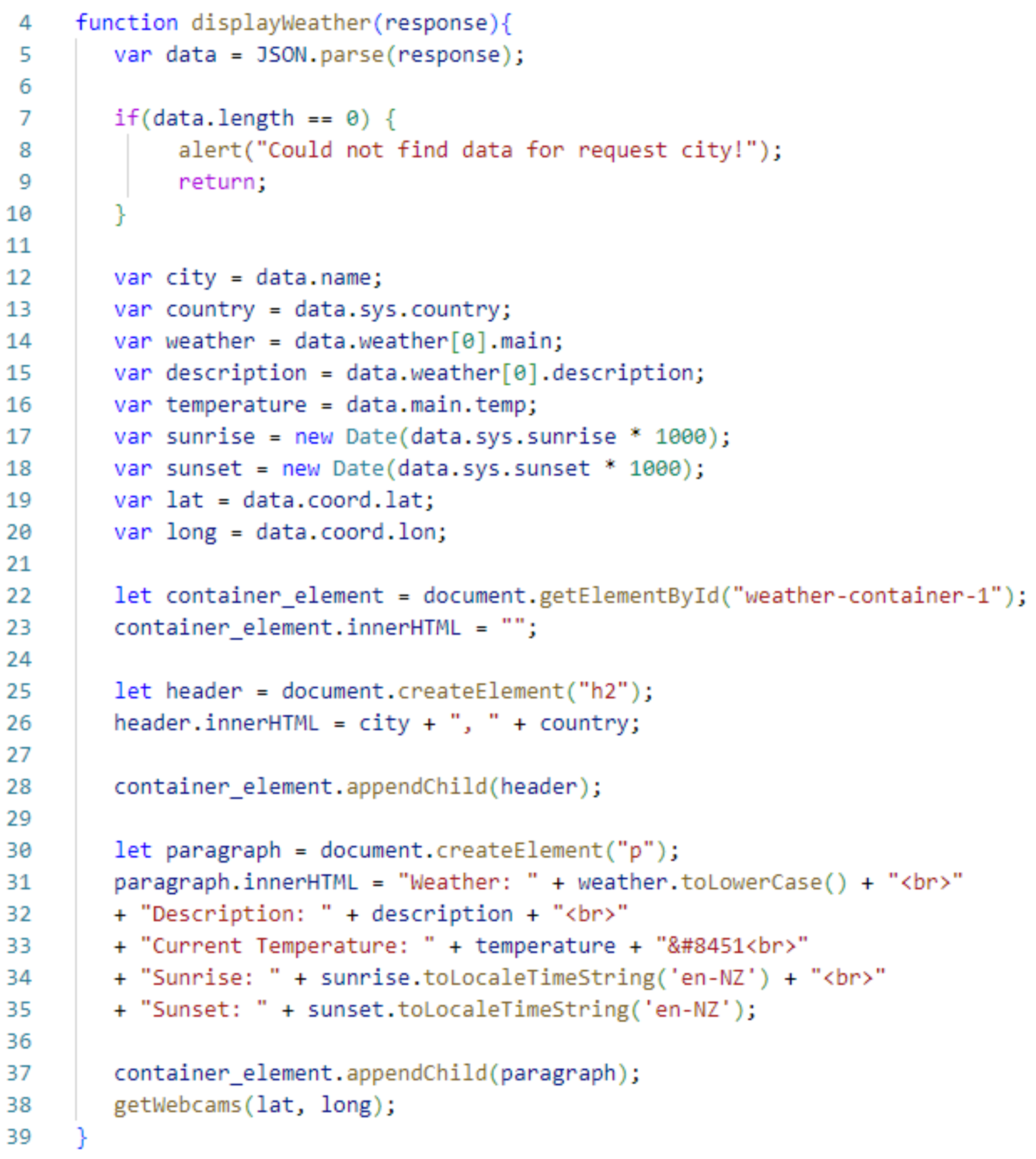
When you are finished your function should look like code shown in figure 55. The `displayWebcams` function utilises similar functions to the above, however, you will need to create "img" elements and set their "src" attribute, for example:

```
var img_element = document.createElement("img");
img_element.src = image;
```

You will also need to loop through the webcam JSON array. For example:

```
var webcam_data = data.result.webcams;
for(var num in webcam_data) {
var webcam = webcam_data[num];
//get images
}
```

The images you want to display are the current preview and the daylight preview. You will also need to get access to the webcam's title.



```
function displayWeather(response){
    var data = JSON.parse(response);

    if(data.length == 0) {
        alert("Could not find data for request city!");
        return;
    }

    var city = data.name;
    var country = data.sys.country;
    var weather = data.weather[0].main;
    var description = data.weather[0].description;
    var temperature = data.main.temp;
    var sunrise = new Date(data.sys.sunrise * 1000);
    var sunset = new Date(data.sys.sunset * 1000);
    var lat = data.coord.lat;
    var long = data.coord.lon;

    let container_element = document.getElementById("weather-container-1");
    container_element.innerHTML = "";

    let header = document.createElement("h2");
    header.innerHTML = city + ", " + country;

    container_element.appendChild(header);

    let paragraph = document.createElement("p");
    paragraph.innerHTML = "Weather: " + weather.toLowerCase() + "<br>"
    + "Description: " + description + "<br>"
    + "Current Temperature: " + temperature + "&#8451<br>"
    + "Sunrise: " + sunrise.toLocaleTimeString('en-NZ') + "<br>"
    + "Sunset: " + sunset.toLocaleTimeString('en-NZ');

    container_element.appendChild(paragraph);
    getWebcams(lat, long);
}
```

Figure 55: displayWeather function

14. See if you can work out how to display the webcams information as shown in figure 53. Try incremental steps and always test in the browser as you go. Use the inspect element to let you know of any Javascript errors.

Summary

In this session we have given even more "hands-off" instructions to allow you to experiment with website development and spend more time thinking about how you will break down problems. For this reason there are no advanced exercises for this session. Hopefully this has provided some insight into how websites are created and gives you a better understanding of how these types of applications are coded. The technology used to build websites is vast and requires many different skills, check out some of the useful resources below for more information.

Useful Resources

- W3Schools: https://www.w3schools.com/
- RedHat IT Topics: https://www.redhat.com/en/topics
- Website Architecture: https://www.webfx.com/blog/web-design/website-architecture/
- OpenWeatherMap API Documentation: https://openweathermap.org/current#geo
- Windy API Documentation: https://api.windy.com/webcams/api/v3/docs#/Webcams%20API%20V3/PublicApiController_getWebcams
- Also see footnotes.