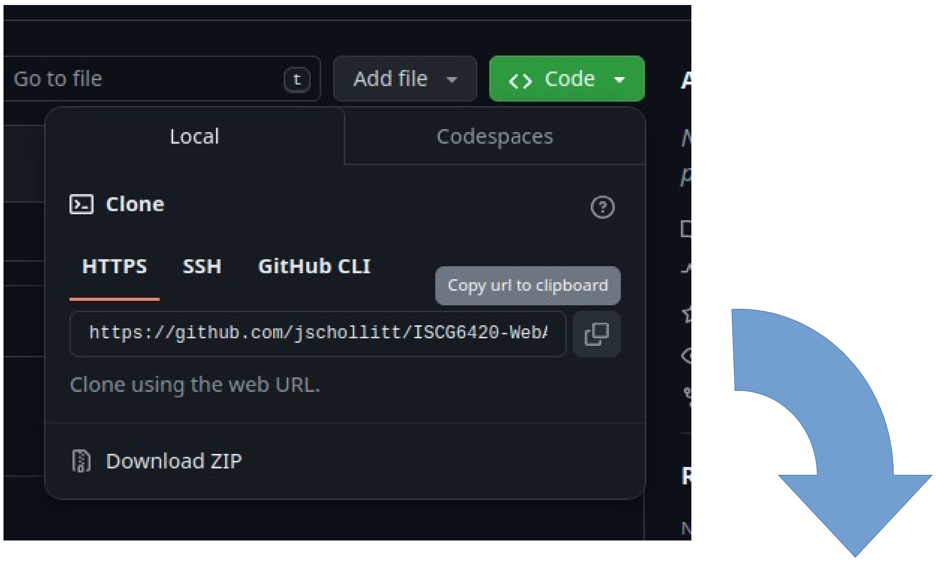
ISCG6420 Semester 1 2024

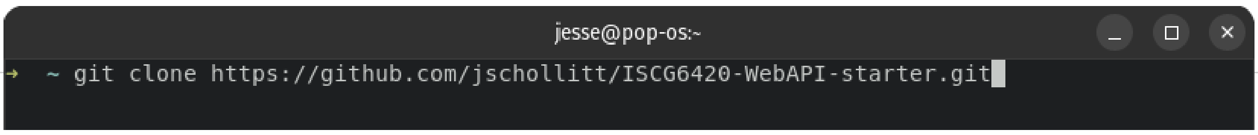
Exercise: Web APIs

These instructions will take you through implementing website features utilising the Drag & Drop, Storage, and Geolocation Web APIs.

Open Exercise Starter

Clone the content from the starter project repository:





```
jesse@pop-os:~
→ ~ git clone https://github.com/jschollitt/ISCG6420-WebAPI-starter.git
```

Open the project folder in VSCode, and open the index.html file for editing.

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Exercise - Web APIs</title>

    <!-- Link external CSS file -->

    <!-- Link Leaflet library CSS file -->

    <!-- Link Leaflet library JS file -->

</head>
```

This exercise will use the Leaflet.js library, which requires external CSS and JS resources.

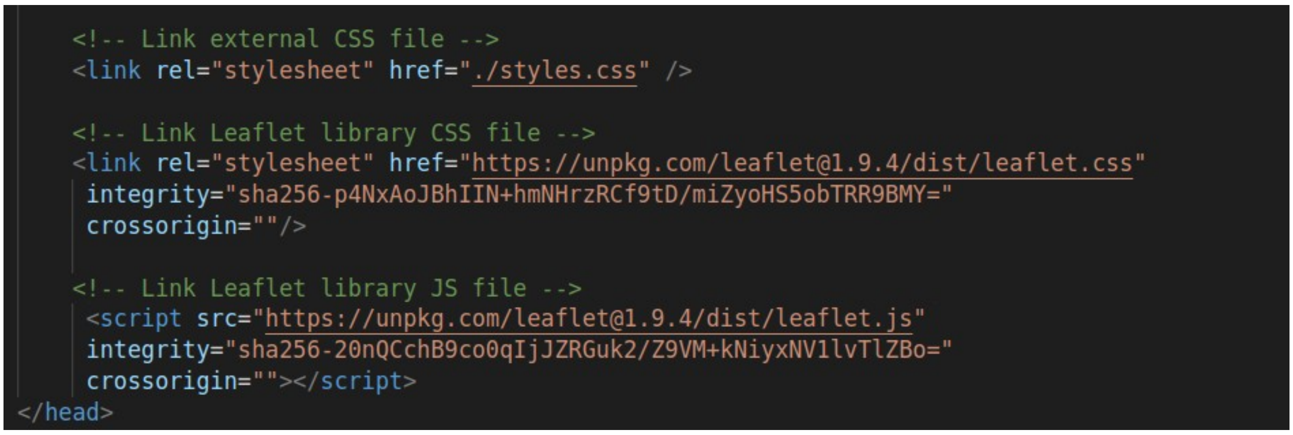
- Link the provided **styles.css** file as an external stylesheet.

- Link the Leaflet CSS file as an external stylesheet using the following snippet:
  ```
  <link rel="stylesheet"
  href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
  integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY="
  crossorigin=""/>
  ```

  or from the library website:
  https://leafletjs.com/examples/quick-start/#preparing-your-page

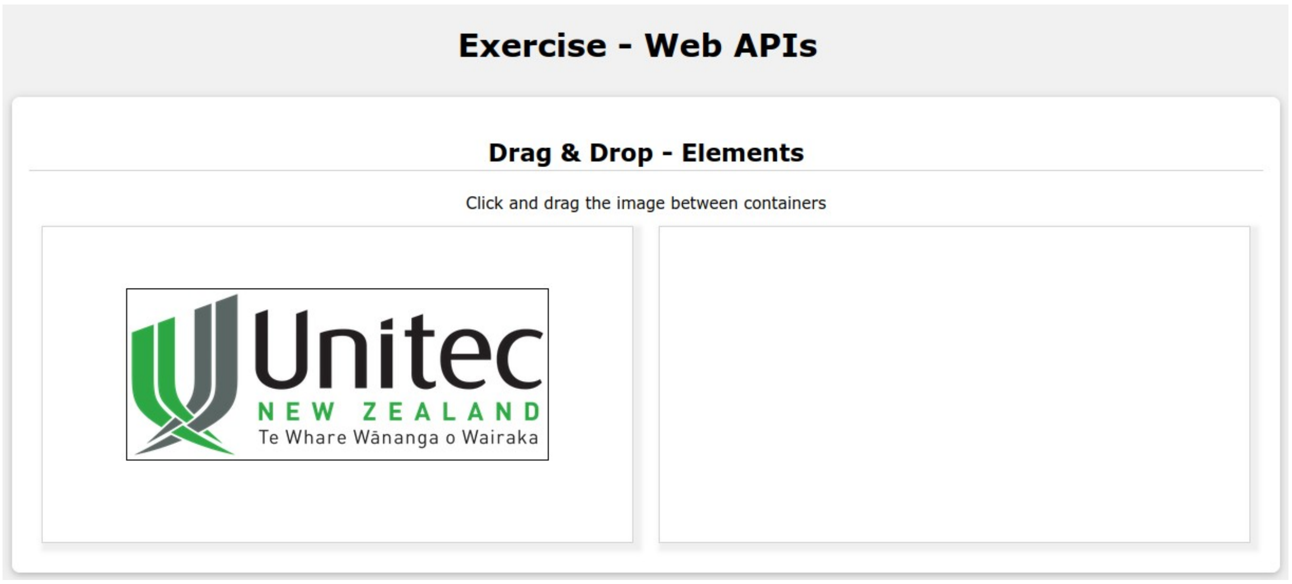
- Link the Leaflet JS file as an external script using the following snippet:
  ```
  <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
  integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
  crossorigin=""></script>
  ```

  or from the library website linked above.



```
    <!-- Link external CSS file -->
    <link rel="stylesheet" href="./styles.css" />

    <!-- Link Leaflet library CSS file -->
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
      integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY="
      crossorigin=""/>

    <!-- Link Leaflet library JS file -->
      <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
      integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
      crossorigin=""></script>
  </head>
```

Save your work and test the output. Your website should look like this:



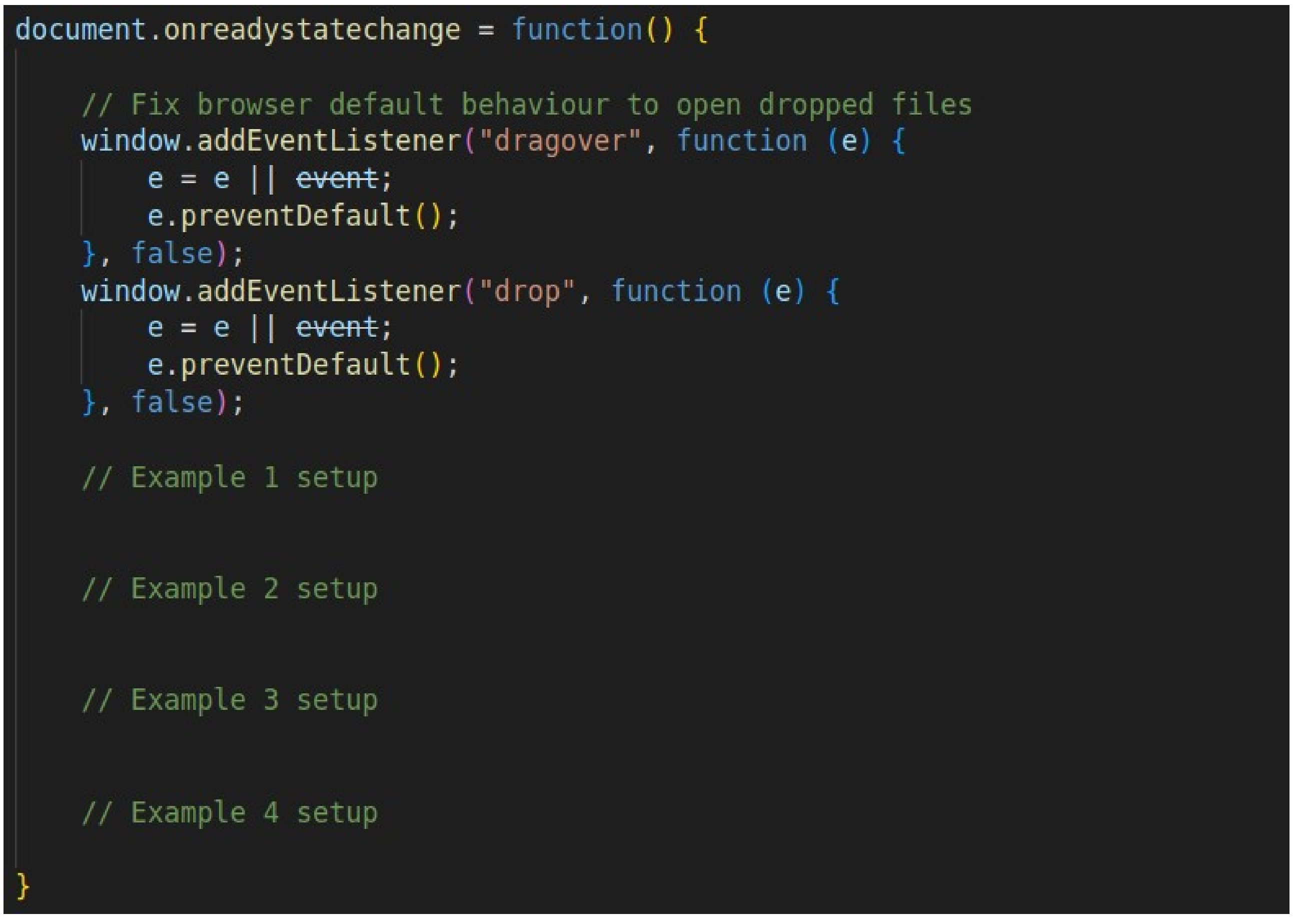
Exercise - Web APIs

Drag & Drop - Elements

Click and drag the image between containers

Document Ready

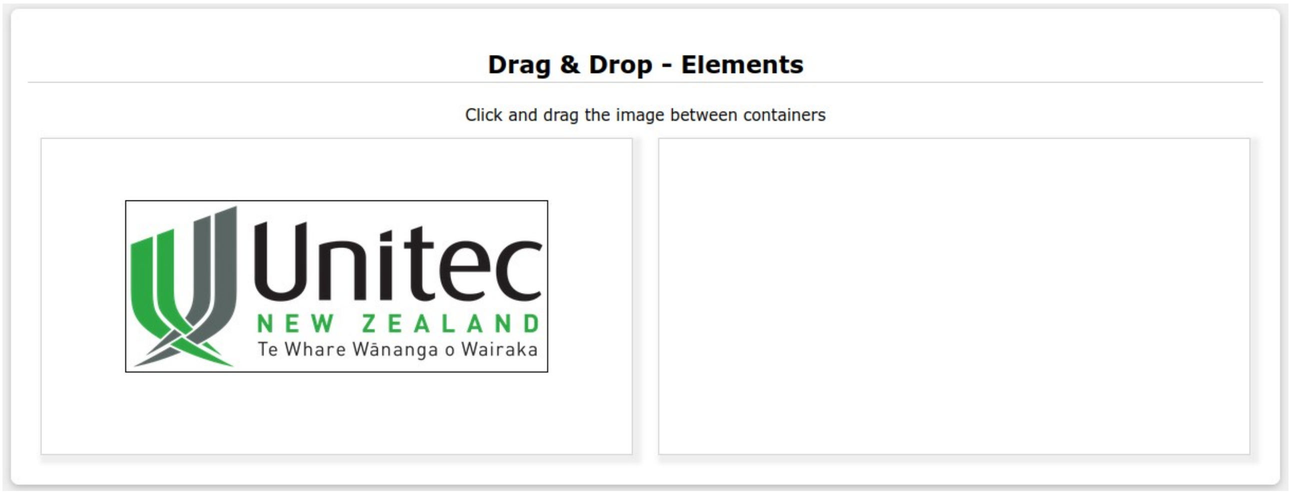
Open the provided **app.js** file for editing. Inside you will find the Document Ready function declaration. This function is run once the web page has completed loading synchronous content and is ready to begin running code. Inside is a section of code, and commented sections for each of the exercises in this activity:



```
document.onreadystatechange = function() {

    // Fix browser default behaviour to open dropped files
    window.addEventListener("dragover", function (e) {
        e = e || ~~event~~;
        e.preventDefault();
    }, false);
    window.addEventListener("drop", function (e) {
        e = e || ~~event~~;
        e.preventDefault();
    }, false);

    // Example 1 setup


    // Example 2 setup


    // Example 3 setup


    // Example 4 setup

}
```

The section of code adds two event listeners to the window element for the **Drag Over** and **Drop** events respectively. Each listener callback function is set to run before the bubbling phase (paramater 3: false), and to stop the web browser's default behaviour from occurring when files are dragged and dropped onto the browser. Most web browsers will attempt to open files that are dragged & dropped. If we wish to capture these files in our website we must prevent the default behaviour occurring.

Drag & Drop Elements


Drag & Drop - Elements
Click and drag the image between containers
Unitec
NEW ZEALAND
Te Whare Wānanga o Wairaka

Exercise 1 demonstrates drag & drop events for HTML elements. When the image in the left container is dragged with the cursor, it will separate from the parent element and be able to be dropped on any containers (called dropzones) that support receiving the dragged element. When this happens, the dropzones will change appearance to visually indicate their support.

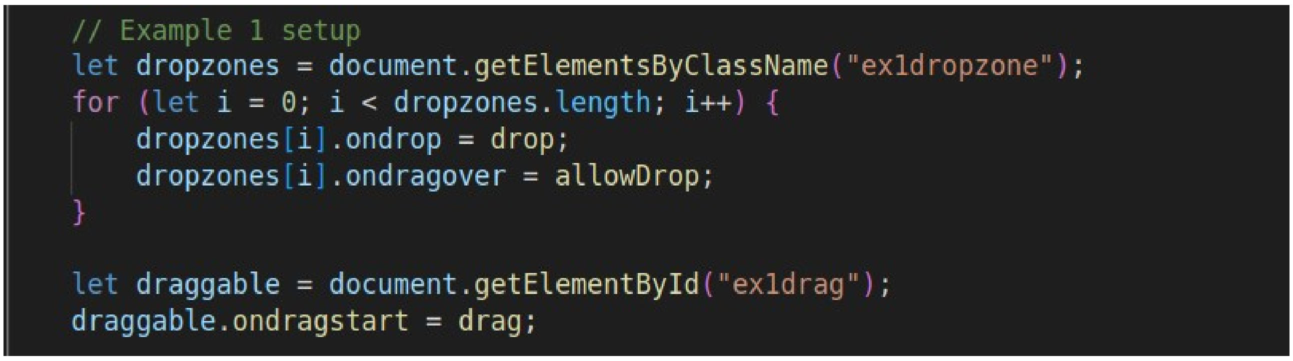
The draggable image element values:

- ID: "**ex1drag**"
- draggable: **true**

The dropzones:

- Div: All **<div>** tags that contain class "**ex1dropzone**".
- Class: "**ex1dropzone**"
- ID: none

Select all dropzones using the class name selector. For each dropzone, set the **ondrop** event to a function called "**drop**".
Select the draggable image and set the **ondragstart** event to a function called "**drag**":



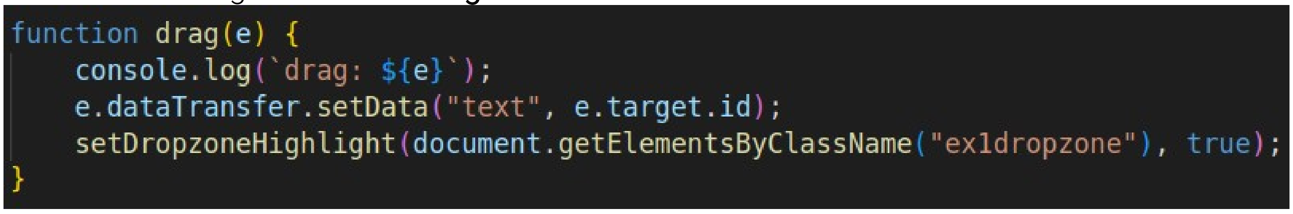
```
// Example 1 setup
let dropzones = document.getElementsByClassName("ex1dropzone");
for (let i = 0; i < dropzones.length; i++) {
    dropzones[i].ondrop = drop;
    dropzones[i].ondragover = allowDrop;
}

let draggable = document.getElementById("ex1drag");
draggable.ondragstart = drag;
```

Below the **document ready** function are three functions used in this exercise:

- drag(e)
- drop(e)
- setDropzoneHighlight(dropzones, state)

The **drag** function is responsible for capturing the data that will be transferred to the destination of the drag & drop. In this exercise we want the image to be transferred to the destination.

Add the following code to the **drag** event function:

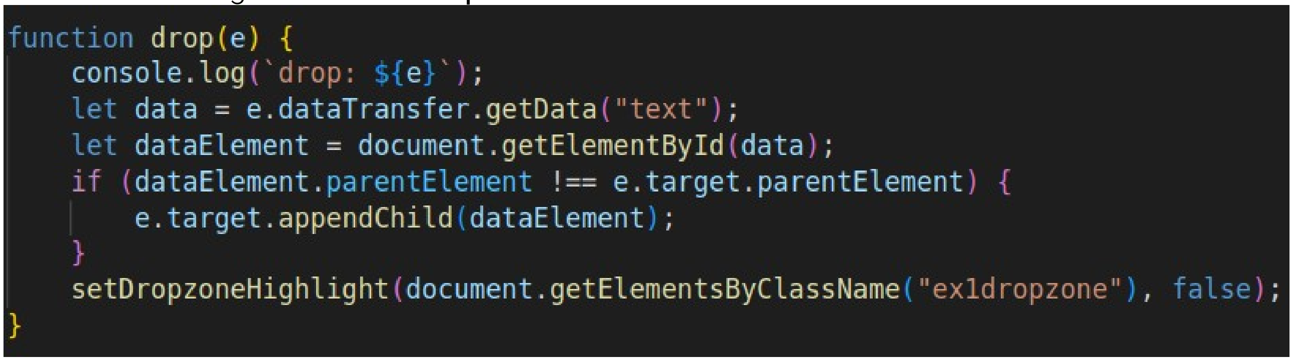


```
function drag(e) {
    console.log(`drag: ${e}`);
    e.dataTransfer.setData("text", e.target.id);
    setDropzoneHighlight(document.getElementsByClassName("ex1dropzone"), true);
}
```

To help with debugging we print the event to the console. Next we set the event data to be of type "text" and to contain the ID of the image element. The last line of the function will enable the highlighting of all supported dropzones when the drag begins.

The **drop** function is responsible for receiving the data from the drag & drop, and to handle processing it. Since we are dragging an image element, the handling will be moving the element location in the DOM tree by making it a child of the destination element.

Add the following code to the **drop** event function:

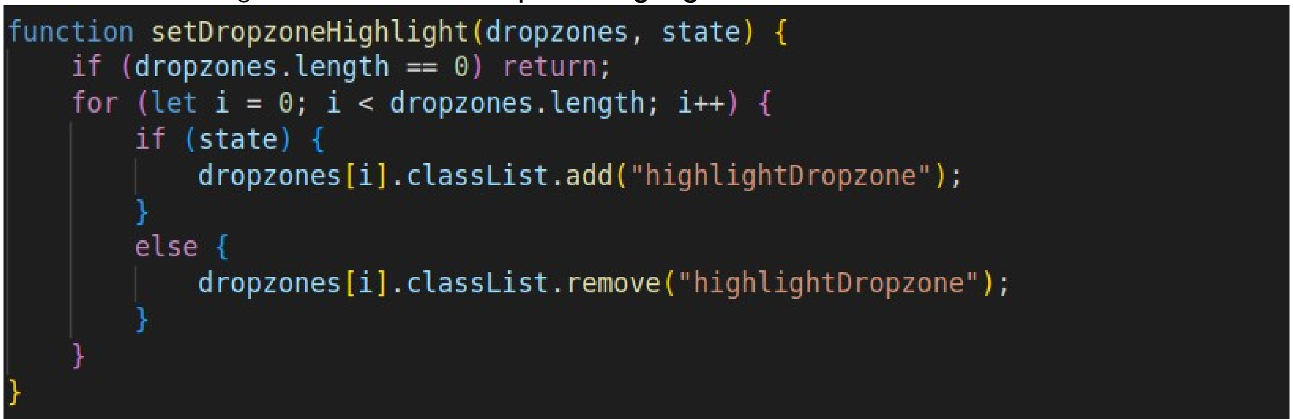


```
function drop(e) {
    console.log(`drop: ${e}`);
    let data = e.dataTransfer.getData("text");
    let dataElement = document.getElementById(data);
    if (dataElement.parentElement !== e.target.parentElement) {
        e.target.appendChild(dataElement);
    }
    setDropzoneHighlight(document.getElementsByClassName("ex1dropzone"), false);
}
```

Again, we print the event to the console for debugging. Next we get the image ID from the event data and select the element with it. We then check if the origin and destination are different elements, and if so, we append the image to the destination. The last line of the function turns off the dropzone highlighting.

The **setDropzoneHighlight** function is responsible for changing the look of the dropzones when a drag & drop event is underway. This is achieved by adding a CSS class to the dropzones when the drag event triggers, and removes the class when the drop event triggers.

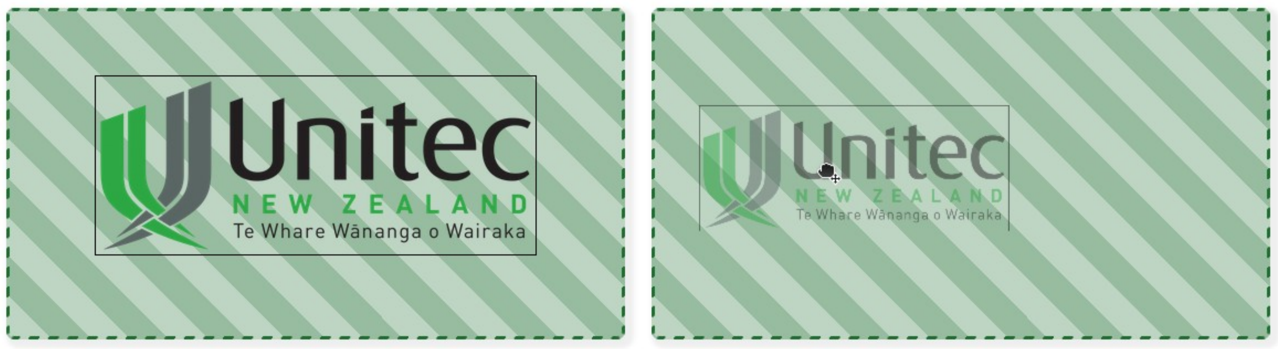
Add the following code to the **setDropzoneHighlight** function:



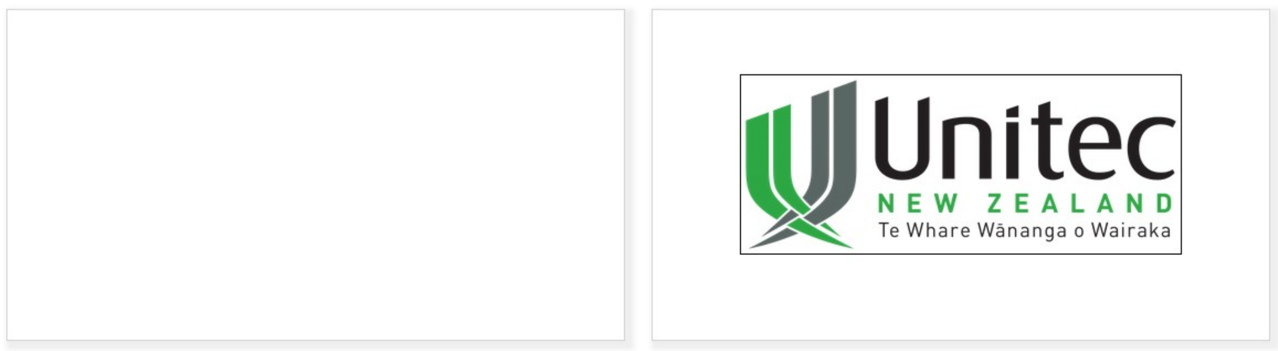
```
function setDropzoneHighlight(dropzones, state) {
    if (dropzones.length == 0) return;
    for (let i = 0; i < dropzones.length; i++) {
        if (state) {
            dropzones[i].classList.add("highlightDropzone");
        }
        else {
            dropzones[i].classList.remove("highlightDropzone");
        }
    }
}
```

For each dropzone element, check if the **state** parameter is true of false, and apply or remove the highlighting class respectively.

Save your work and test the output. When you drag the image on the left, all dropzones should become animated with a striped pattern. When the image is dropped on a dropzone, the animation should disappear and the image has moved to the destination element:

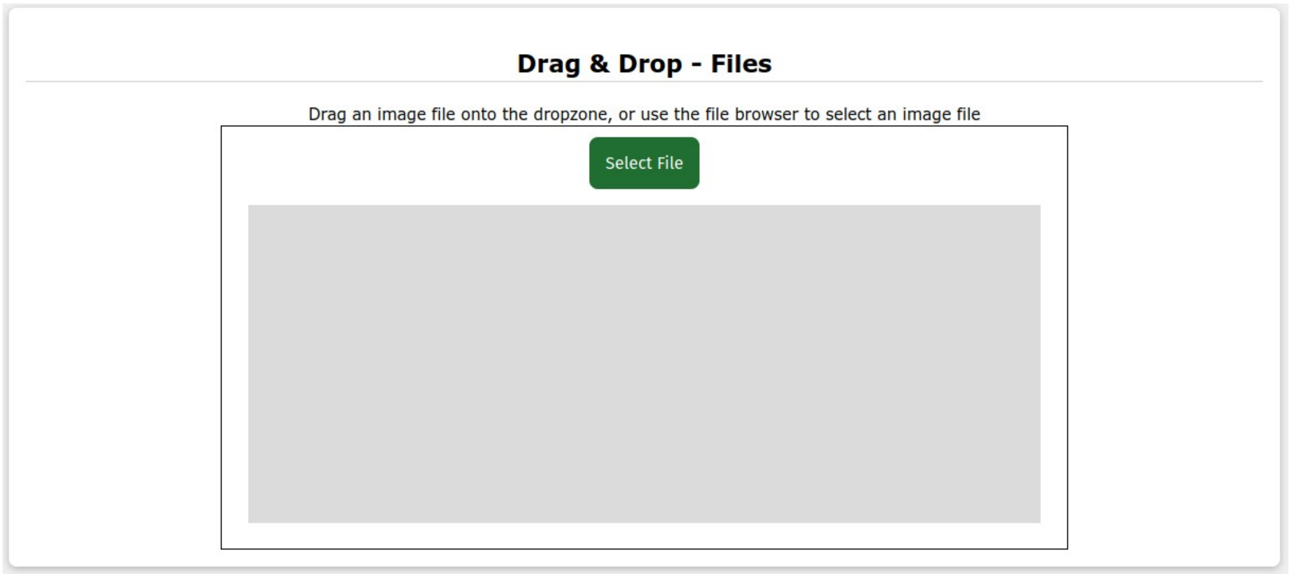


After the drag & drop:



Exercise 1 complete.

Drag & Drop Files


Drag & Drop - Files

Drag an image file onto the dropzone, or use the file browser to select an image file

Exercise 2 demonstrates drag & drop events involving external resources and the delivery of data to the website from the user's environment. This is achieved using three elements and five event functions:

Elements:

- <div> with ID: "**ex2dropzone**"
- <button> with ID: "**ex2button**"
- <input> with ID: "**ex2fileInput**" and type: "**file**" and accept filter: "**image/**"

In the **document ready** event function under the exercise 2 comment, set the fileReader variable to a new **FileReader** object and add a function to the **load** event called **loadImage**:

```
// Example 2 setup
fileReader = new FileReader();
fileReader.addEventListener("load", loadImage);
```

Add five more event listener functions for the dropzone drag events, the button click event, and the file input change event:

```
fileReader.addEventListener("load", loadImage);

document.getElementById("ex2dropzone").addEventListener("dragover", dragoverImage);
document.getElementById("ex2dropzone").addEventListener("dragleave", dragleaveImage);
document.getElementById("ex2dropzone").addEventListener("drop", dropImage);
document.getElementById("ex2button").addEventListener("click", fileExplorer);
document.getElementById("ex2fileInput").addEventListener("change", fileExplorerUpload);
```

Below the exercise 1 functions locate the function called **dragoverImage()**. Add the following code to the function:

```
function dragoverImage() {
    setDropzoneHighlight([document.getElementById('ex2dropzone')], true);
    return false;
}
```

This function reuses the highlighting from exercise 1 to highlight the file dropzone.

Add the following code to the **dragleaveImage()** function:

```
function dragleaveImage() {
    setDropzoneHighlight([document.getElementById('ex2dropzone')], false);
}
```

Add the following code to the **dropImage()** function:

```
function dropImage(e) {
    let file = e.dataTransfer.files[0];
    uploadImage(file);
    setDropzoneHighlight([document.getElementById('ex2dropzone')], false);
}
```

This function selects the first file in the drag & drop data and passes it to the **uploadImage()** function. It then disables the dropzone highlighting.

To provide a method of sending images on devices that do not support drag & drop, a button in the dropzone will trigger a file input browser to let the user select a file manually.

Add the following code to the **fileExplorer()** function:

```
function fileExplorer() {
    document.getElementById('ex2fileInput').click();
}
```

This will manually activate the hidden file input browser when the button is clicked.

Add the following code to the **fileExplorerUpload()** function:

```
function fileExplorerUpload() {
    let file = document.getElementById('ex2fileInput').files[0];
    uploadImage(file);
}
```

This function is called when the file selection by the user is confirmed (thus, changed). It selects the first file in the selection and passes it to the **uploadImage()** function.

Add the following code to the **uploadImage()** function:

```
function uploadImage(file) {
    if (!file.type.includes('image')) {
        return alert("Image file types only");
    }
    if (file.size > 10_000_000) {
        return alert("File too large, please try a smaller image");
    }

    fileReader.readAsDataURL(file);
}
```

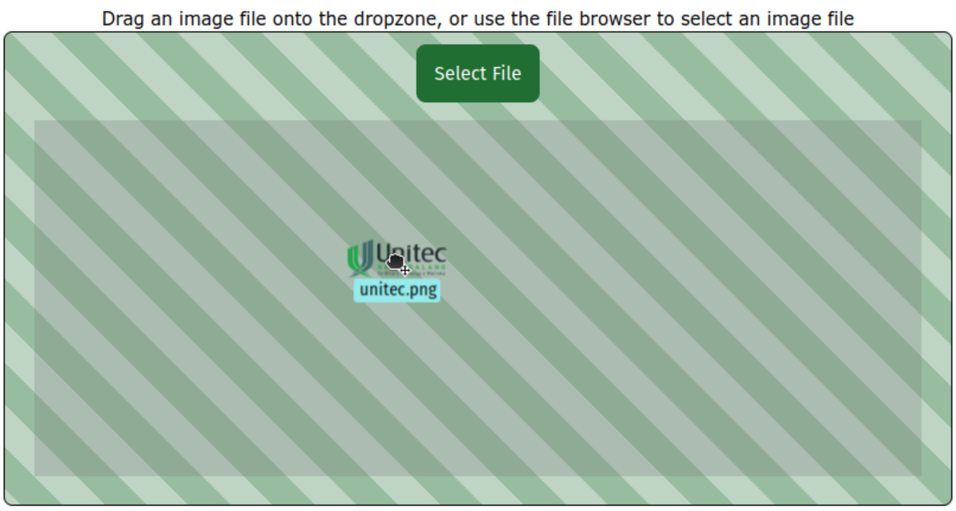
This function checks that the uploading file is an image and less than 10MB in size. If not, an alert is shown to the user. If it passes the checks the file is read into a website-usable format called a data URL. When this happens the **load** event will trigger the **loadImage()** function.

Add the following code to the **loadImage()** function:

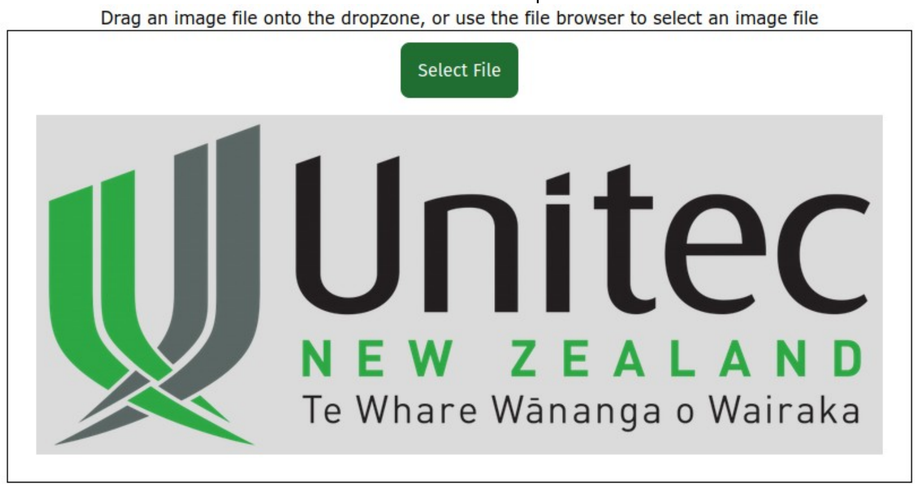
```
function loadImage(e) {
    const image = document.getElementById("ex2img");
    image.style.backgroundImage = `url(${e.target.result})`;
}
```

This function takes the data URL from the FileReader and sets it as the source of the background image for the "**ex2img**" element.

Save your work and test the output. You should be able to drag an image file from your system's file browser over the exercise 2 dropzone, which will highlight the dropzone. Upon dropping the image file the dropzone should change to disable the highlighting and display the dropped image. The same effect can be achieved using the manual **Select File** button:

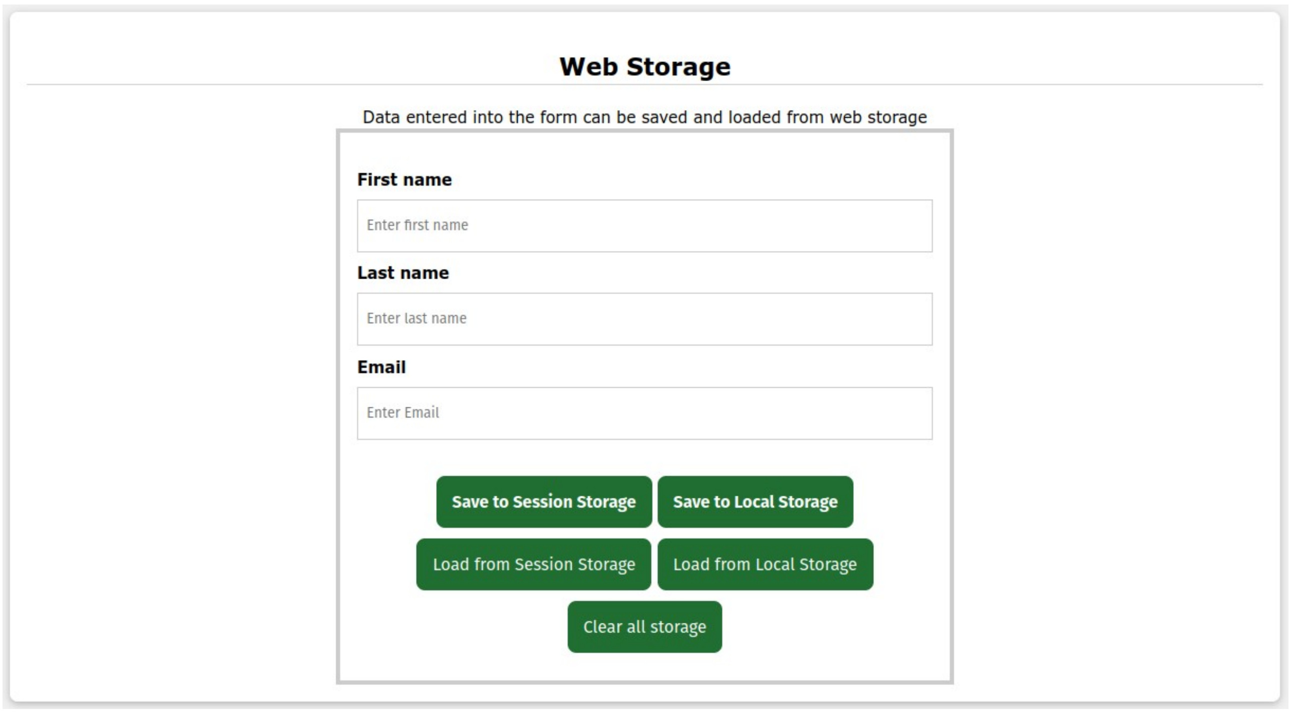

Drag an image file onto the dropzone, or use the file browser to select an image file

After the drop:


Drag an image file onto the dropzone, or use the file browser to select an image file

Exercise 2 complete.

Web Storage



Exercise 3 demonstrates the Web Storage API to access the Local and Session Storage. Form values are able to be saved and loaded from both storage types, and the user can clear all storage. This is achieved using three input elements, five buttons, and nine functions:

Elements:

- <input> with ID: "**fname**" and type: "**text**"
- <input> with ID: "**lname**" and type: "**text**"
- <input> with ID: "**email**" and type: "**text**"
- <button> with ID: "**ex3savesession**"
- <button> with ID: "**ex3savelocal**"
- <button> with ID: "**ex3loadsession**"
- <button> with ID: "**ex3loadlocal**"
- <button> with ID: "**ex3clear**"
- <span> with ID: "**ex3message**"

In the **document ready** event function under the exercise 3 comment, create a **click** event listener for the five buttons. Assign the buttons to the following functions:

- ex3savesession: saveSessionData
- ex3savelocal: saveLocalData
- ex3loadsession: loadSessionData
- ex3loadlocal: loadLocalData
- ex3clear: clearStorage

```
// Example 3 setup
document.getElementById('ex3savesession').addEventListener("click", saveSessionData);
document.getElementById('ex3savelocal').addEventListener("click", saveLocalData);
document.getElementById('ex3loadsession').addEventListener("click", loadSessionData);
document.getElementById('ex3loadlocal').addEventListener("click", loadLocalData);
document.getElementById('ex3clear').addEventListener("click", clearStorage);
```

Below the exercise 2 functions locate the function called **getFormFields()**. Add the following code to the function:

```
function getFormFields() {
    return [
        document.getElementById('fname').value,
        document.getElementById('lname').value,
        document.getElementById('email').value
    ];
}
```

This is a helper function to easily get the data from the input fields as an array.

Locate the function called **setFormFields()**. Add the following code to the function:

```
function setFormFields(fname, lname, email) {
    document.getElementById('fname').value = fname;
    document.getElementById('lname').value = lname;
    document.getElementById('email').value = email;
}
```

This is a helper function to easily populate the input fields from three string values.

Locate the function called **storageAvailable()**. It contains code from Mozilla to detect support for storage types in the browser. A description of the code and its use can be found here:
https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API#feature-detecting_localstorage

No changes to this function are required.

Locate the function called **saveSessionData()**. Add the following code to the function:

```
function saveSessionData() {
    if (!storageAvailable("sessionStorage")) {
        setMessage("Session Storage is not available in this browser", "red", 5000);
        return;
    };

    let data = getFormFields();
    sessionStorage.setItem("firstname", data[0]);
    sessionStorage.setItem("lastname", data[1]);
    sessionStorage.setItem("email", data[2]);
    setMessage("Data saved in Session Storage", "#55ff55", 3000);
}
```

This function checks if Session Storage is available and if not, shows an error message to the user with **setMessage()**. If available, it fetches the input field values with **getFormFields()** and saves them in Session Storage, and shows a success message to the user.

Locate the function called **saveLocalData()**. Add the following code to the function:

```
function saveLocalData() {
    if (!storageAvailable("localStorage")) {
        setMessage("Local Storage is not available in this browser", "red", 5000);
        return;
    }

    let data = getFormFields();
    localStorage.setItem("firstname", data[0]);
    localStorage.setItem("lastname", data[1]);
    localStorage.setItem("email", data[2]);
    setMessage("Data saved in Local Storage", "#55ff55", 3000);
}
```

This function is similar to the previous function, except it utilises Local Storage instead of Session Storage.

Locate the function called **loadSessionData()**. Add the following code to the function:

```
function loadSessionData() {
    let fname = sessionStorage.getItem("firstname") ?? "";
    let lname = sessionStorage.getItem("lastname") ?? "";
    let email = sessionStorage.getItem("email") ?? "";
    setFormFields(fname, lname, email);
    setMessage("Data loaded from Session Storage", "#55ff55", 3000);
}
```

This function attempts to fetch the stored data from Session Storage with the keys used to store the data. If a value fetch is unsuccessful, an empty string is fetched instead. This is done using the **nullish coalescing operator (??)**. This operator offers a fallback value in case the first value is null. More information about this operator can be found on MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing

Locate the function called **loadLocalData()**. Add the following code to the function:

```
function loadLocalData() {
    let fname = localStorage.getItem("firstname") ?? "";
    let lname = localStorage.getItem("lastname") ?? "";
    let email = localStorage.getItem("email") ?? "";
    setFormFields(fname, lname, email);
    setMessage("Data loaded from Local Storage", "#55ff55", 3000);
}
```

As with the data saving functions, the loading functions are similar except for the type of storage being accessed.

Locate the function called **clearStorage()**. Add the following code to the function:

```
function clearStorage() {
    sessionStorage.clear();
    localStorage.clear();
}
```

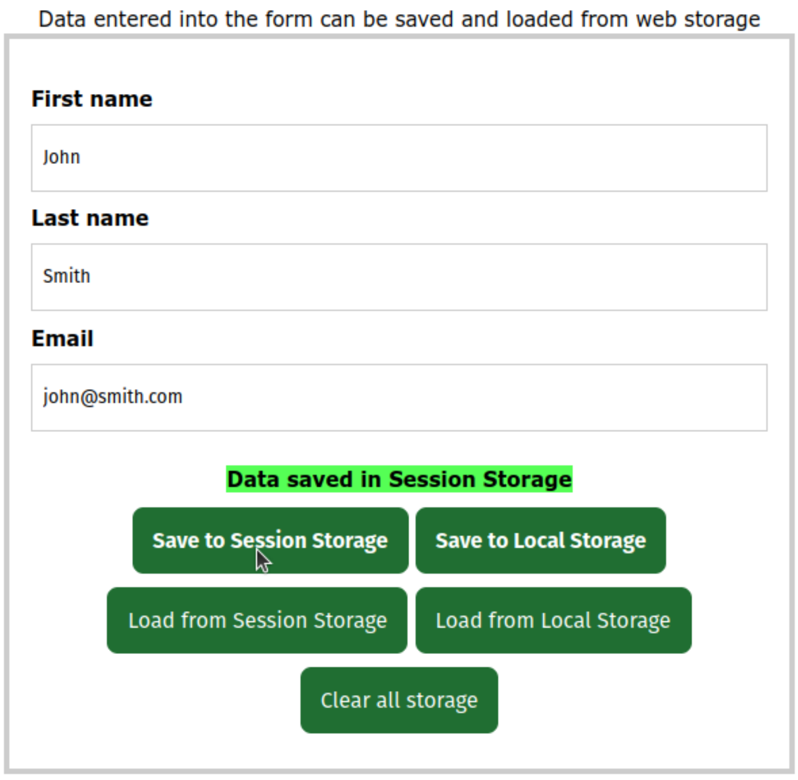
This function is for debugging and will clear all Session and Local storage.

Locate the function called **setMessage()**. Add the following code to the function:

```
function setMessage(message, colour, time) {
    let msg = document.getElementById('ex3message');
    msg.innerText = message;
    msg.style.backgroundColor = colour;
    msg.style.visibility = "visible";

    setTimeout(function () {
        msg.style.visibility = "hidden";
    }, time);
}
```

This function takes a string message, a colour, and an amount of time, and displays a message to the user with the provided properties. The message is displayed in the hidden **<span>** element which is made visible for the allocated time. The timing action is achieved using **setTimeout** which is similar to **setInterval** except it only triggers once.

Save your work and test the output. You should be able to populate the text fields and save the contents to either storage types. The text field values can then be modified but restored to the saved values using the load buttons. Each storage-related action should display a succcess or failure message which will appear for a limited time.


Data entered into the form can be saved and loaded from web storage

First name

John

Last name

Smith

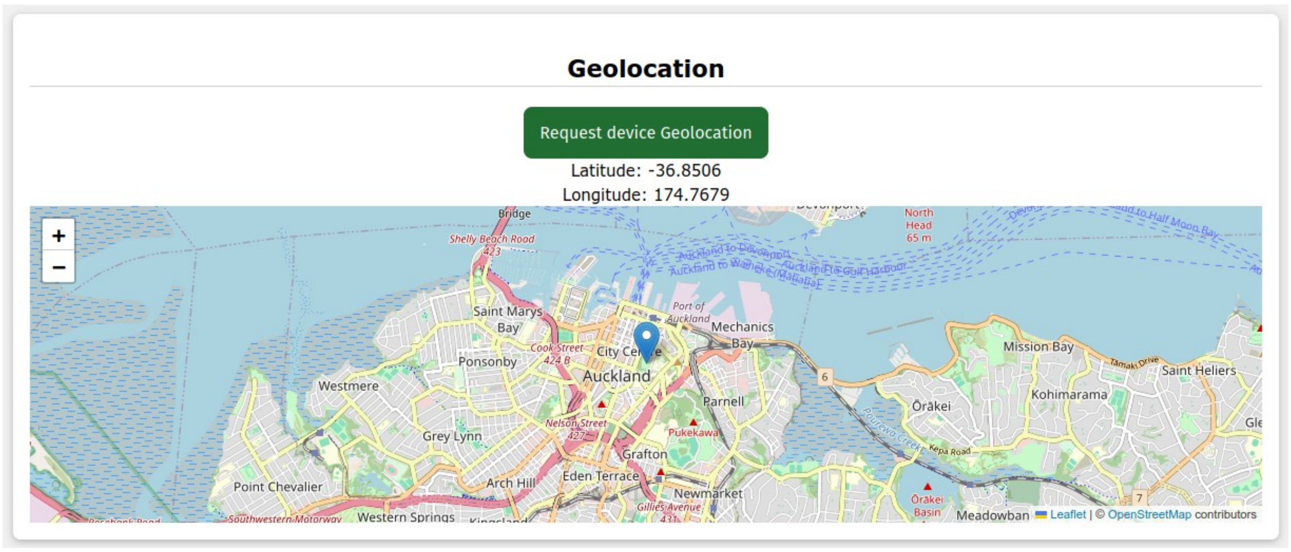
Email

john@smith.com

Data saved in Session Storage

Exercise 3 complete.

Geolocation


Geolocation

Latitude: -36.8506
Longitude: 174.7679

Exercise 4 demonstrates how to use the Geolocation API to access user location and display location information in text and map form. This exercise uses the Leaflet.js library, which provides a simple way to display OpenStreetMap map components on web pages.

In the **document ready** event function under the exercise 4 comment, create a **click** event listener for the button with ID: "**ex4button**" and assign the **getGeo** function:

```
// Example 4 setup
document.getElementById('ex4button').addEventListener("click", getGeo);
```

Under the exercise 3 functions, locate the function called **getGeo()**. Add the following code to the function:

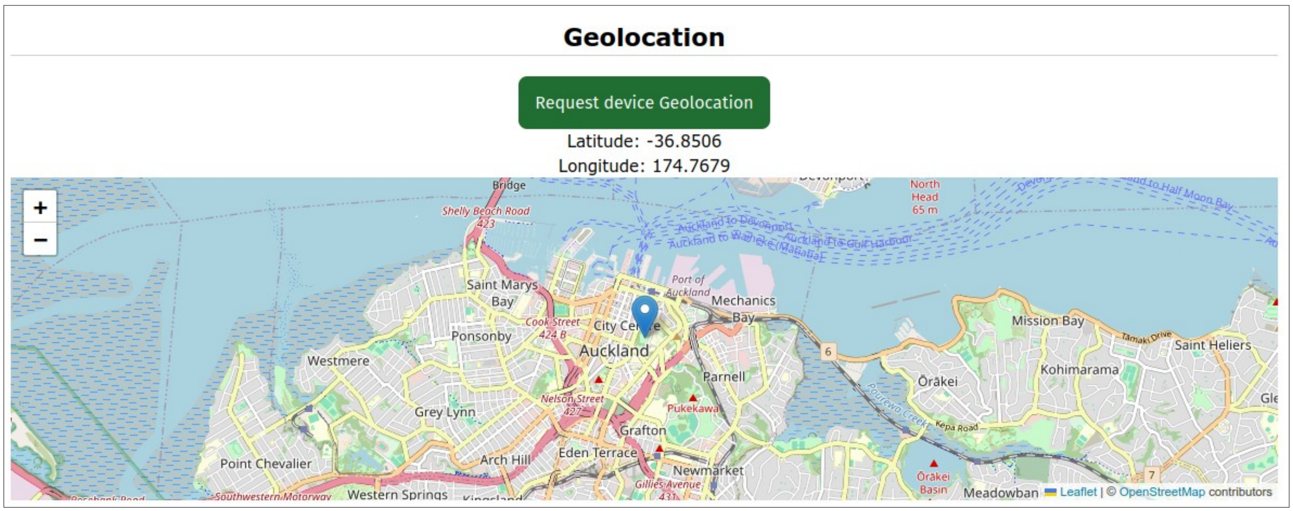
```
function getGeo() {
    if (!navigator.geolocation) {
        document.getElementById('ex4lat').innerText = "Geolocation permission denied";
        document.getElementById('ex4long').innerText = "Geolocation permission denied";
        return;
    }
    navigator.geolocation.getCurrentPosition(function (position) {
        let lat = position.coords.latitude;
        let long = position.coords.longitude;
        document.getElementById('ex4lat').innerText = lat;
        document.getElementById('ex4long').innerText = long;

        var map = L.map('map').setView([lat, long], 13);

        L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
            attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        }).addTo(map);

        L.marker([lat, long]).addTo(map);
    });
}
```

Copyable Leaflet-specific code:

```
var map = L.map('map').setView([lat, long], 13);

L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
}).addTo(map);

L.marker([lat, long]).addTo(map);
```

This function checks if the browser supports geolocation and displays error messages if not. If supported, it then attempts to fetch the current location (which will request user permission if not already granted) and display the latitude and longitude as text. It then uses the Leaflet library to create a map centred on the latitude and longitude, adds the attribution overlay, and adds a pin marker to the exact geolocation position on the map.

Save your work and test the output. You should be able to click the **Request device Geolocation** button and see your device latitude, longitude and map position displayed:


Geolocation
Request device Geolocation
Auckland
Leaflet | © OpenStreetMap contributors

Latitude: -36.8506
Longitude: 174.7679

Exercise 4 complete.

Next Steps

- Modify Exercise 2 to accept text files and display their contents in a <p> element.
- Explore the persistence and possible uses of Session and Local storage.
- Incorporate these Web API features into your Assessment 3 Part 1 project.

Exercises complete.