HTML – Web APIs

What is an API?

- API: Application Programming Interface
- Interfaceable code that provides easy access to complex functionality
  - Code abstraction
- Made available in Programming Languages

- APIs work like electrical plug sockets
  - Simple to connect and use
  - Provide acccess to complex (or dangerous) resources

APIs in Web development

- Many categories of APIs available
- Common JS Web APIs
  - Document Object Model (DOM) API
  - Browser Object Model (BOM) API
- Web Browser APIs
  - Forms (validation)
  - Storage (persisting data)
  - Drag & Drop (user interface features)
  - Geolocation (device position)
- Third Party
  - Google Maps, Paypal, etc
  - https://www.postman.com/explore/most-popular-apis-this-year

Web Browser APIs

https://developer.mozilla.org/en-US/docs/Web/API

- Browser APIs we have used so far:
  - Document: Provides access to the DOM
  - Event: Provides ways to invoke and respond to actions
  - HTML Element: Access to use and interact with elements
  - XMLDocument, XMLHttpRequest: XML file functionality

- New APIs we will use today:
  - Drag & Drop
  - Storage
  - Geolocation

Browser API: Drag & Drop

https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API

HTML Drag and Drop interfaces enable applications to use drag-and-drop features in browsers.

The user may select *draggable* elements with a mouse, drag those elements to a *droppable* element, and drop them by releasing the mouse button. A translucent representation of the *draggable* elements follows the pointer during the drag operation.

During drag operations, several event types are fired, and some events might fire many times, such as the **drag** and **dragover** events.

Browser API: Drag & Drop

Lifecycle Events:

Event	Fires when...
`drag`	...a *dragged item* (element or text selection) is dragged.
`dragend`	...a drag operation ends (such as releasing a mouse button or hitting the Esc key; see Finishing a Drag.)
`dragenter`	...a dragged item enters a valid drop target. (See Specifying Drop Targets.)
`dragleave`	...a dragged item leaves a valid drop target.
`dragover`	...a dragged item is being dragged over a valid drop target, every few hundred milliseconds.
`dragstart`	...the user starts dragging an item. (See Starting a Drag Operation.)
`drop`	...an item is dropped on a valid drop target. (See Performing a Drop.)

<*element* draggable="true|false|auto">

Definition and Usage

- The draggable attribute specifies whether an element is draggable or not.
- Links and images are draggable by default.
  - (explicit declaration is recommended for clarity)
- The draggable attribute is often used in drag and drop operations.

true	Specifies that the element is draggable
false	Specifies that the element in not draggable
auto	Uses the default behaviour of the browser

HTML draggable Attribute

- To make an element draggable, set the draggable attribute to true:

HTML:

```
<div id="myDiv" draggable="true">Draggable Div</div>
```

JavaScript

```
const myDiv = document.getElementById('myDiv');
myDiv.setAttribute("draggable", "true");
```

jQuery

```
$('#myDiv').attr("draggable", "true");
```

```
window.addEventListener("dragover", function (e) {
    e = e || event;
    e.preventDefault();
}, false);
window.addEventListener("drop", function (e) {
    e = e || event;
    e.preventDefault();
}, false);

function drag(e) {
    e.dataTransfer.setData("text", myData);
}

function drop(e) {
    e.preventDefault();
    let data = e.dataTransfer.getData("text");
    doSomethingWith(data);
}

const draggable = document.getElementById('draggable');
draggable.addEventListener("dragstart", drag);


const destination = document.getElementById('dropzone');
destination.addEventListener("drop", drop);
```

What to Drag - ondragstart and setData()

- Specify what should happen when the element is dragged.
- In the example above, the ondragstart attribute calls a function, drag(event), that specifies what data to be dragged.
- dataTransfer.setData(type, value) sets the data type and the value of the dragged data:

```
function drag(ev) {
    ev.dataTransfer.setData("Text", ev.target.id);
}
```

- In this case, the data type is "Text" and the value is the id of the draggable element ("myDiv").

Where to Drop - ondragover

- The ondragover event specifies the response when an element is dragged over.
- By default, data cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.
- This is done by calling the event.preventDefault() method for the ondragover event:
- *event*.preventDefault()

Default Behaviour

- Modern web browsers support opening common file types:
  - Documents: PDF, CSV, TXT
  - Images: JPG, PNG, WEBP, GIF
  - Markup: XML, SVG, HTML
- When these files types are dropped on a browser window, the browser will attempt to open the file in a new Tab.
- This can be prevented to allow a website to handle the file instead:

```
window.addEventListener("dragover", function (e) {
    e = e || event;
    e.preventDefault();
}, false);
window.addEventListener("drop", function (e) {
    e = e || event;
    e.preventDefault();
}, false);
```

Do the Drop - ondrop

- When the dragged data is dropped, a drop event occurs.
- The ondrop callback calls a function, drop(event):

```
function drop(e) {
    e.preventDefault();
    var data = e.dataTransfer.getData("Text");
    e.target.appendChild(document.getElementById(data));
}
```

Example: Move Element

```
function drag(e) {
    e.dataTransfer.setData("text", e.target.id);
}

function drop(e) {
    e.preventDefault();
    let data = e.dataTransfer.getData("text");
    let dragElement = document.getElementById(data);
    e.target.appendChild(dragElement);
}
```

Code explained:

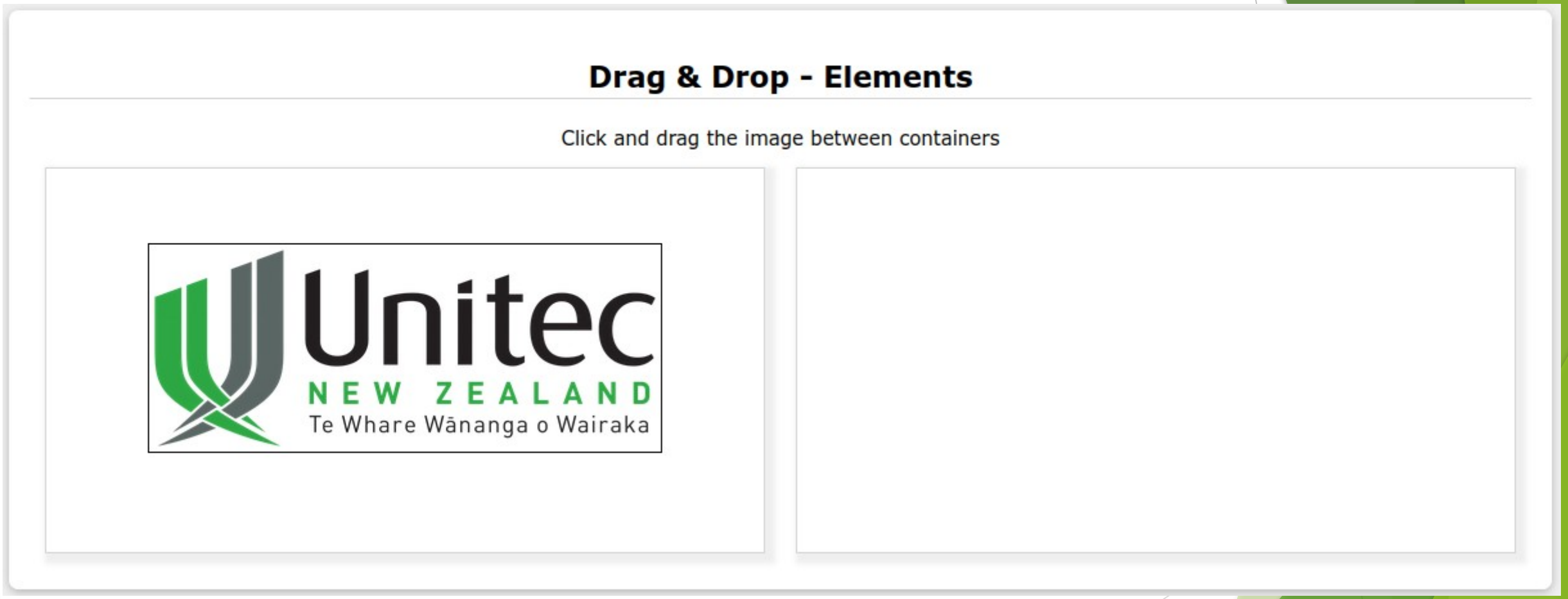
- Call preventDefault() to prevent the browser default handling of the data (default is open as link on drop)
- Get the dragged data with the dataTransfer.getData("Text") method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("myDiv")
- Append the dragged element to the drop element

Example: add message on event

```
<div id="divMsg1">I appear on enter  </div>
<div id="divMsg2">I appear on drop </div>

<div id="div1"></div>

div1.addEventListener(“drop”,drop);
div1.addEventListener(“dragover”, allowDrop);
div1.addEventListener(“dragenter”, enter);
```

```
function drag(e) {
    e.dataTransfer.setData("Text", e.target.id);
    document.getElementById("divMsg2").style.visibility="hidden";
    document.getElementById("divMsg1").style.visibility="hidden";
}

function drop(e) {
    e.preventDefault();
    var data = e.dataTransfer.getData("Text");
    e.target.appendChild(document.getElementById(data));
    document.getElementById("divMsg2").style.visibility="visible";

}
function enter(e) {
    document.getElementById("divMsg1").style.visibility="visible";
    return true;
}
```

Example:



https://jschollitt.github.io/week11/week11.html

Web API: Storage

Web Storage Types

Technology	Description
Cookies	An HTTP cookie is a small piece of data that the web server and browser send each other to remember stateful information across page navigation.
Web Storage	The Web Storage API provides mechanisms for webpages to store string-only key/value pairs, including `localStorage` and `sessionStorage`.
IndexedDB	IndexedDB is a Web API for storing large data structures in the browser and indexing them for high-performance searching.
Cache API	The Cache API provides a persistent storage mechanism for HTTP request and response object pairs that's used to make webpages load faster.
Origin Private File System (OPFS)	OPFS provides a file system that's private to the origin of the page and can be used to read and write directories and files.

Browser API: Storage

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

The **Web Storage API** provides mechanisms by which browsers can store key/value pairs, in a much more intuitive fashion than using cookies.

The two mechanisms within Web Storage are as follows:

sessionStorage maintains a separate storage area for each given origin that's available for the duration of the page session (as long as the browser tab is open, including page reloads and restores).

localStorage does the same thing, but persists even when the browser is closed and reopened.

Web Storage behaviour:

- **SessionStorage** and **LocalStorage** are synchronous.
  - Storage interactions are blocking (other code must wait until blocking operations are complete).
  - Performance can be affected by large operations.
- Developers are recommended to minimise storage operation size where possible, or utilise **asynchronous** alternatives.
  - **IndexedDB** is a low-level API providing access to a **transactional database system** (SQL-like).
  - Designed for large datasets.

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

Web Storage limits:

- **Storage** technologies have limits on their usability
  - Size / capacity
  - Data type
  - Accessibility
- Enforcement of these contraints differs between browsers.
  - Cookie size and count per origin varies
  - LocalStorage and sessionStorage is limited to 10MB
  - Firefox limits max storage per origin at 50% disk space or 8TiB
  - Chrome limits to 60% disk space

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

Web Storage limits:

- When the storage limit is exceeded, the attempt to store data will fail
  - **QuotaExceededError** exception
- **Data Eviction** is a browser process that deletes an origin's stored data.
  - **Storage Pressure eviction** occurs when a device is running low on storage space and the browser has less space than needed to store all origin stored data.
  - **Maximum Storage exceeded eviction** occurs when storage size for all origin data exceeds a browser-imposed limit.
- Data eviction will delete **all** data for an origin, across all storage types.

https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Storage_quotas_and_eviction_criteria

Web Storage persistence:

- **Local Storage** data has no expiration. Data stored will remain as long as the web browser's storage is not cleared, and no **eviction** event occurs.
- **Session Storage** data expires when the browser session is terminated. This occurs when a browser tab or window is closed.

- Exception to persistence rules: **Private** or **Incognito** sessions.
  - All data cleared on session termination.

Web Storage usage:

- Using **local storage** and **session storage** is done through the **storage web API**

- **local Storage**

Store data
```
localStorage.setItem("my key", "my data");
```

Retrieve data
```
let mydata = localStorage.getItem("my key");
```

Delete data
```
localStorage.removeItem("my key");
```

Delete all
```
localStorage.clear();
```

Web Storage usage:

- Using **local storage** and **session storage** is done through the **storage web API**

- **Session Storage**

Store data
```
sessionStorage.setItem("my key", "my data");
```

Retrieve data
```
let mydata = sessionStorage.getItem("my key");
```

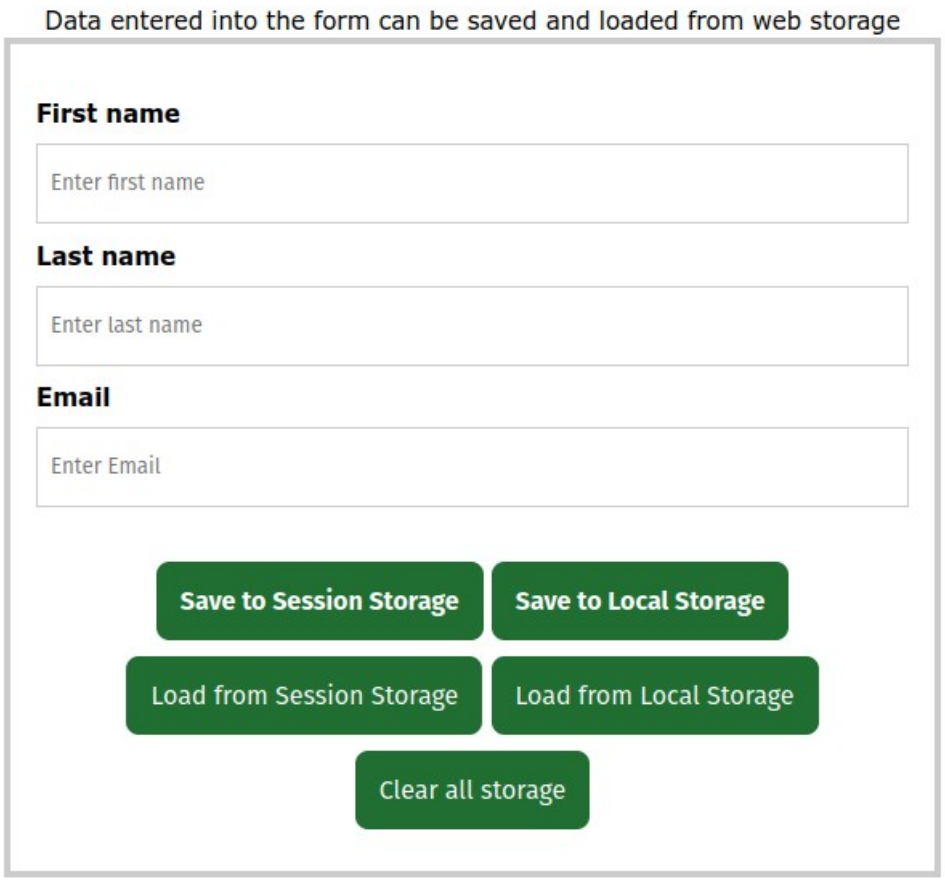
Delete data
```
sessionStorage.removeItem("my key");
```

Delete all
```
sessionStorage.clear();
```

Example:

```
function saveData(key, data) {
    if (storageAvailable("localStorage")) {
        localStorage.setItem(key, data);
    }
    else {
        alert("Unable to store data in localStorage.");
    }
}

function loadData(key) {
    let data = localStorage.getItem(key);
    if (data === null) {
        alert(`Unable to retrieve data for key ${key}. No data found.`);
    }
    return data ?? "";
}
```

Example:



https://jschollitt.github.io/week11/week11.html

Web API: Geolocation

Browser API: Geolocation

https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

The **Geolocation API** provides functionality to retrieve global positioning information about the user device. For privacy reasons, the user is asked for permission to access this information upon request.

The two ways location information can be retrieved:

getCurrentPosition() initiates a quick-response asynchronous request to detect the user's position. Devices with GPS will often report their IP location or Wi-Fi location instead due to the time requirements of getting a GPS fix.

watchPosition() does the same thing, but continues to callback at intervals until **clearWatch()** is called.

Geolocation usage:

- Using **geolocation** is done through the **navigator**

```
navigator.geolocation.getCurrentPosition(function (position)
{
    let where = position.coords;
    let when = position.timestamp;
});

navigator.geolocation.watchPosition(function (position) {
    let where = position.coords;
    let when = position.timestamp;
});
```

Geolocation usage:

- Request for position returns with:
  - On success:
    - GeolocationPosition
      - GeolocationCoordinates
        - Latitude, Longitude, Altitude, Accuracy, AltitudeAccuracy, Heading, Speed
      - Timestamp (unix time in milliseconds)
  - On failure:
    - GeolocationPositionError
      - Error code
        - 1 = permission denied, 2 = position unavailable, 3 = timeout

Geolocation options:

- Position requests can take an optional parameter object which can specify three options:
  - MaximumAge. Positive **long** value indicating acceptable age of position.
  - Timeout. Positive **long** value indicating maximum time allowed to process position request.
  - EnableHighAccuracy. **Boolean** value indicating priority of accuracy request. A value of **true** will request more resource-intensive means of measuring user position, often using GPS.

Geolocation notes:

- Position request data often relies on Google services for device or Wi-Fi location. In regions without Google, alternative third party positioning services may be used. Such as: Baidu, Autonavi, Tencent

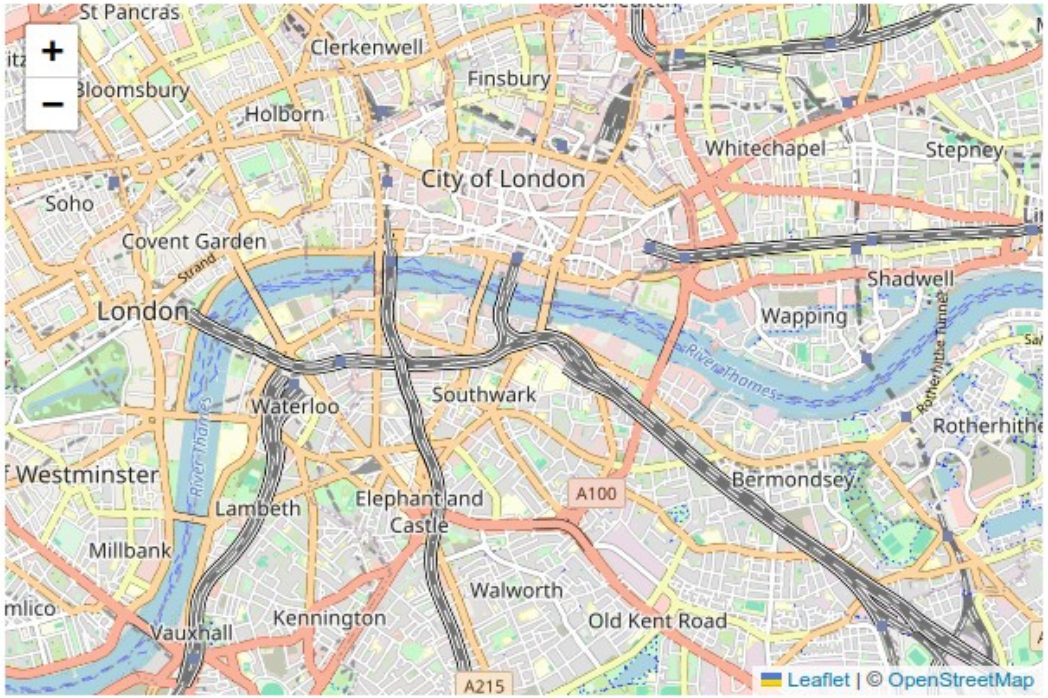
- Geolocation is only available on secure connections using https
  - OR localhost if supported by the browser for development purposes

Geolocation usage:

```
const options = {
    enableHighAccuracy: true,
    timeout: 5000,
    maximumAge: 0,
};

function success(position) {
    console.log(position.coords);
}

function failure(error) {
    console.warn(error);
}

navigator.geolocation.getCurrentPosition(success, failure, options);
```

Leaflet

- To simplify the display of position data we can use a library like **leaflet.js** to show a map of the user position.



https://leafletjs.com/examples/quick-start/#preparing-your-page

Leaflet.js

- Using leaflet requires these steps:
  - Inlude the Leaflet CSS and JS files
  - Add a with ID: “map” to the web page
  - Set a fixed map height in CSS
  - Create a map object in JS

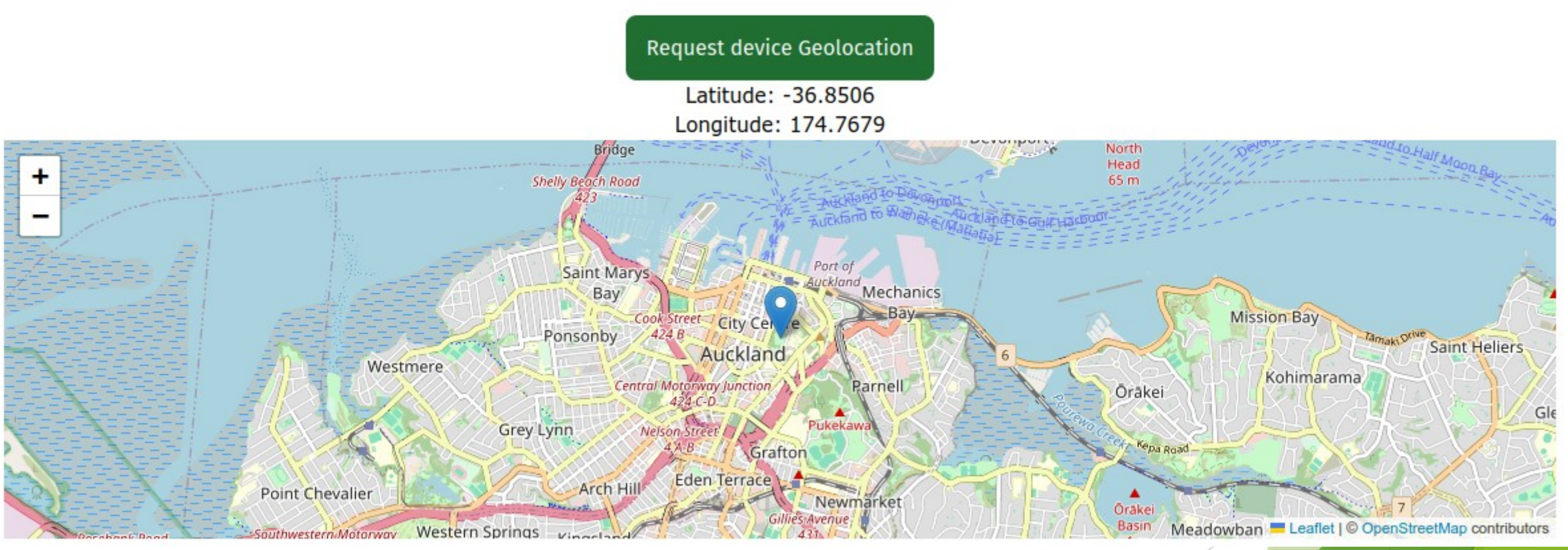
```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css"
integrity="sha256-p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY="
crossorigin=""/>

<!-- Make sure you put this AFTER Leaflet's CSS -->
<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
    integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo="
    crossorigin=""></script>
```

```
<div id="map"></div>
```

```
#map {
    height: 300px;
}
```

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

Example:



https://jschollitt.github.io/week11/week11.html

Exercise

- Download / clone the github repo linked from Moodle: Week 11
- Download the instructions document from Moodle

- Assessment 3 Check Point 2
  - Lecturer will check in with each team