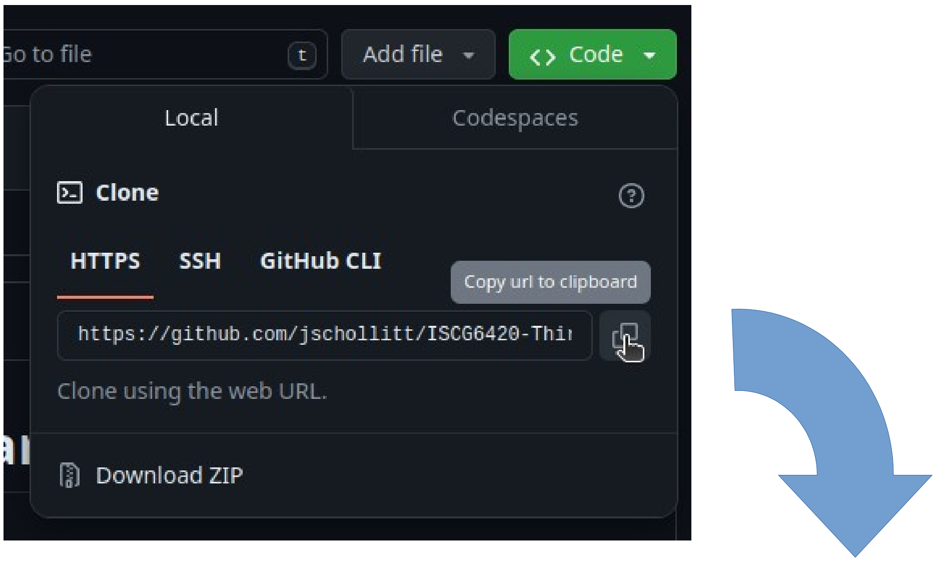
ISCG6420 Semester 1 2024

Exercise: Web APIs

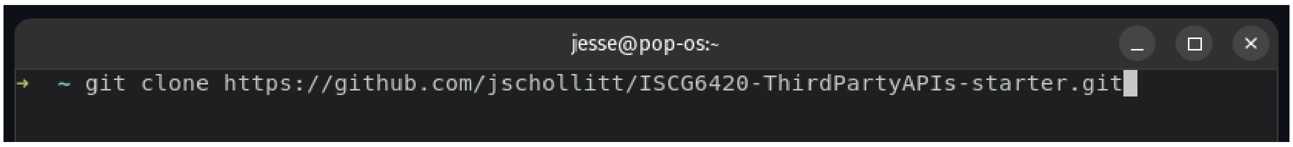
These instructions will take you through implementing website features utilising third party API services, and demonstrating JS <-> JSON conversion for client-server communication.

Open Exercise Starter

Clone the content from the starter project repository:

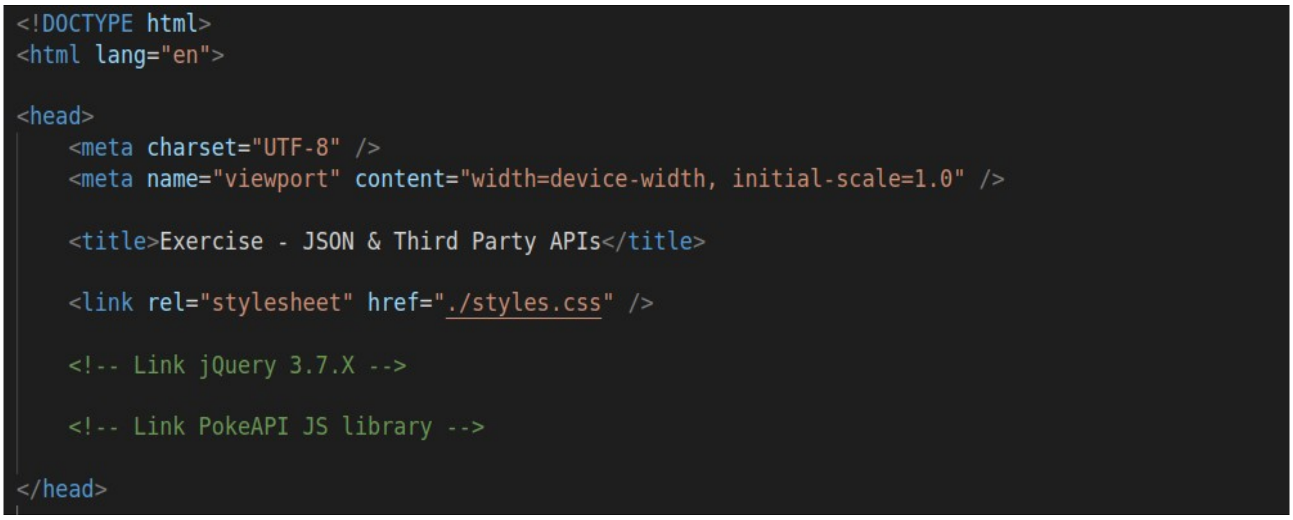


```
https://github.com/jschollitt/ISCG6420-Thi
```



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

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

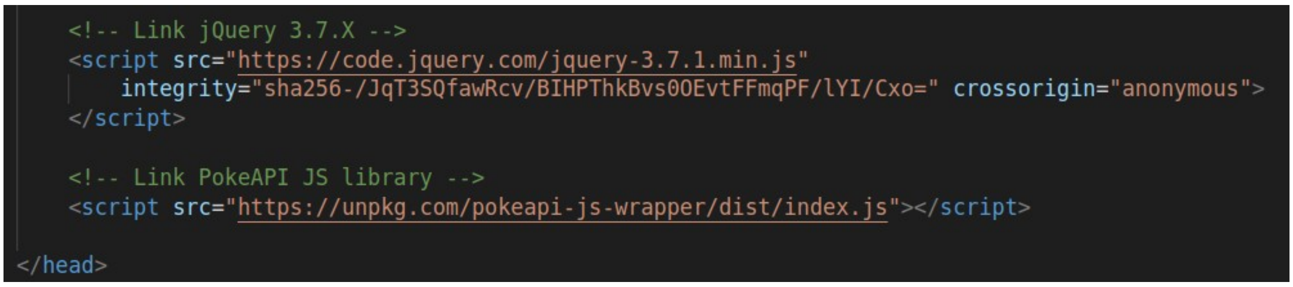


```
<!DOCTYPE html>
<html lang="en">

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

    <title>Exercise - JSON & Third Party APIs</title>

    <link rel="stylesheet" href="./styles.css" />

    <!-- Link jQuery 3.7.X -->

    <!-- Link PokeAPI JS library -->

</head>
```

Some exercises will use jQuery and PokeAPI via a library, which require external JS resources.
jQuery library website: https://releases.jquery.com/
API wrapper library project: https://github.com/PokeAPI/pokeapi-js-wrapper

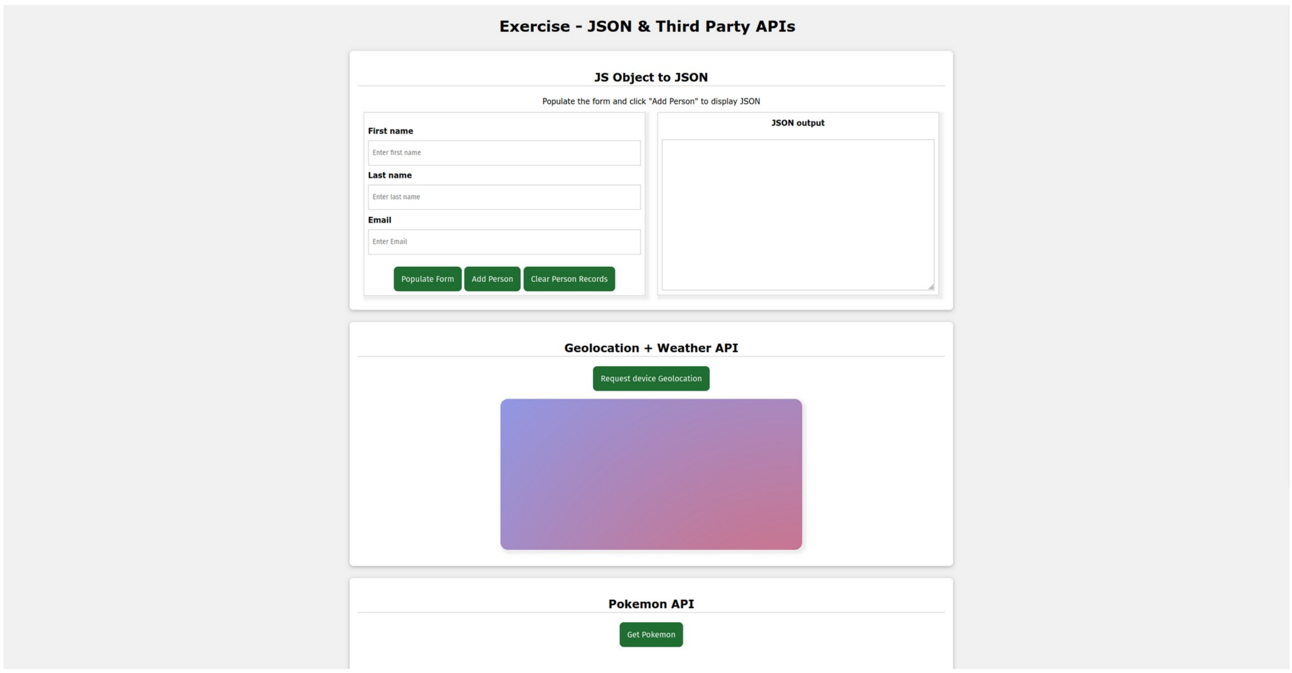
- Link the jQuery JS file as an external script using the following snippet:
```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"
        integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>
```
- Link the Leaflet CSS file as an external stylesheet using the following snippet:
```
<script src="https://unpkg.com/pokeapi-js-wrapper/dist/index.js"></script>
```

  Alternative library install / use options are available from their respective websites.



```
    <!-- Link jQuery 3.7.X -->
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"
      integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo=" crossorigin="anonymous">
    </script>

    <!-- Link PokeAPI JS library -->
    <script src="https://unpkg.com/pokeapi-js-wrapper/dist/index.js"></script>

</head>
```

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



Exercise - JSON & Third Party APIs

JS Object to JSON

Populate the form and click "Add Person" to display JSON

First name: Enter first name
Last name: Enter last name
Email: Enter Email

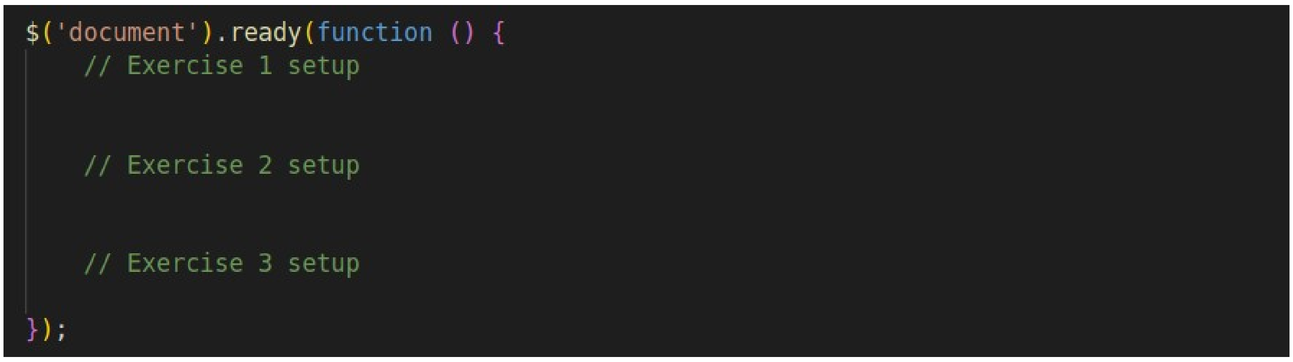
JSON output

Geolocation + Weather API

Pokemon API

Document Ready

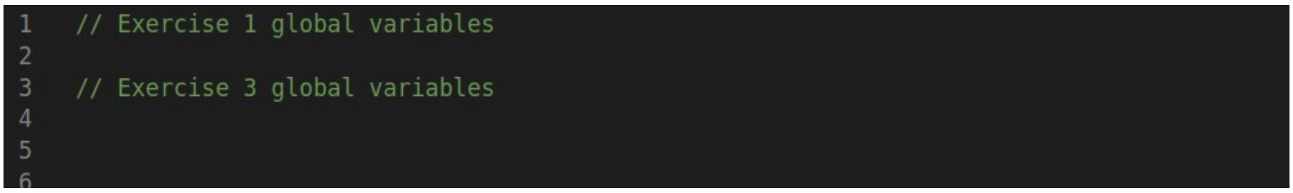
Open the provided **app.js** file for editing. Inside you will find the Document Ready function declaration. This function is run once the web page has completed loading synchronous content and is ready to begin running code. Inside are commented sections for each of the exercises in this activity:



```
$('document').ready(function () {
    // Exercise 1 setup


    // Exercise 2 setup


    // Exercise 3 setup

});
```

Global Variables

At the top of the **app.js** file, above the Document Ready function is where the global variables will be declared.



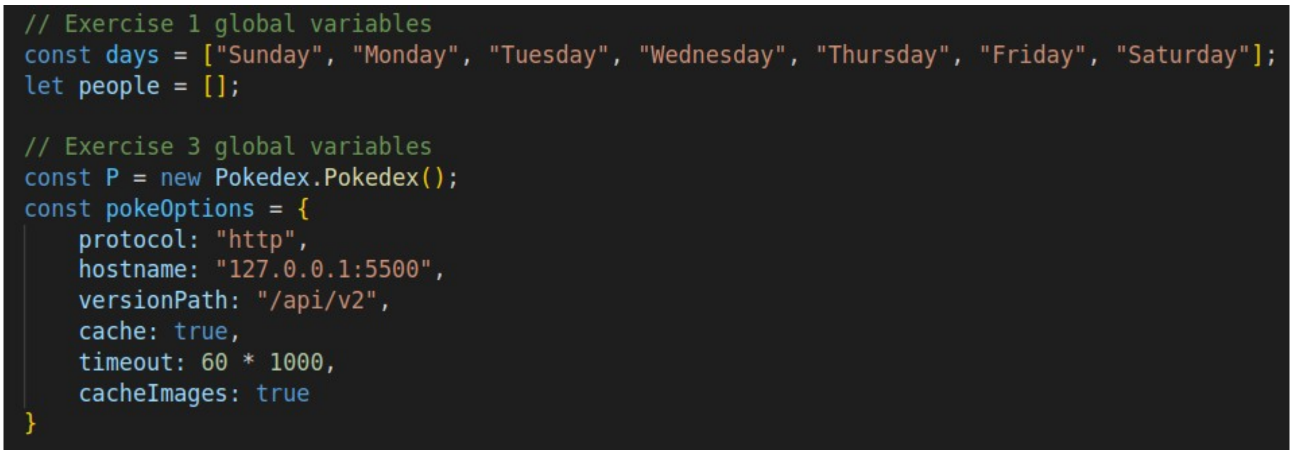
```
// Exercise 1 global variables

// Exercise 3 global variables



```

For exercise 1:

- Create a constant array called **days** and set its contents to the days of the week as strings.
- Underneath, create an array called **people** and set it empty.

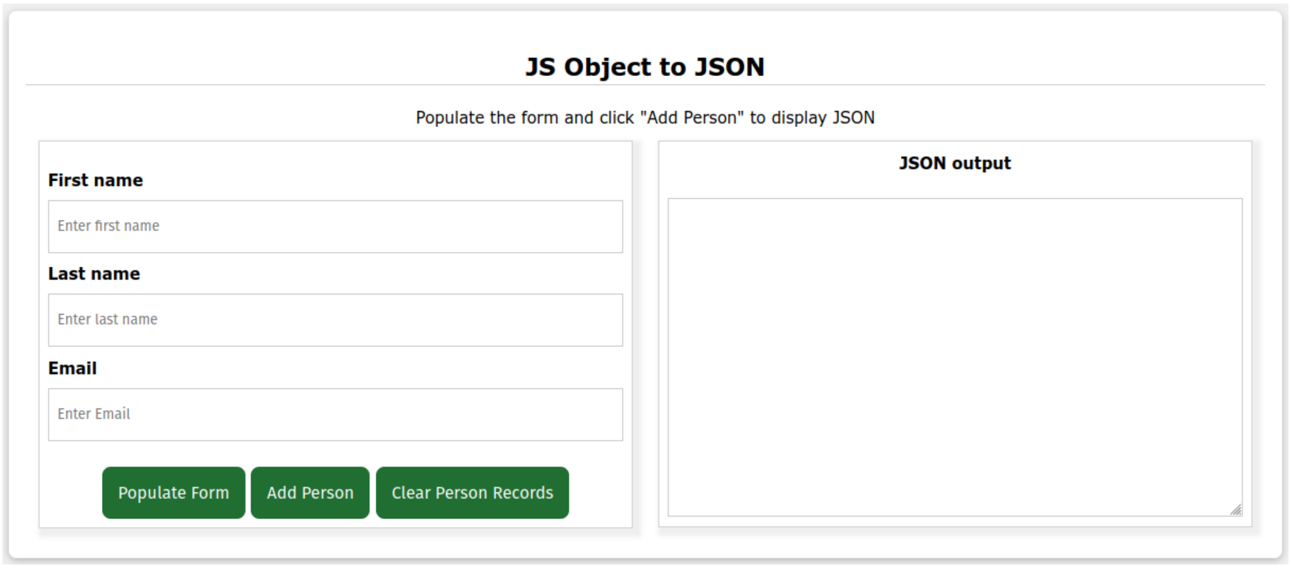
For exercise 3:

- Create a constant variable called **P** and assign a new Pokedex object to it.
- Create a constant object called **pokeOptions** and assign the properties below:



```
// Exercise 1 global variables
const days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];
let people = [];

// Exercise 3 global variables
const P = new Pokedex.Pokedex();
const pokeOptions = {
    protocol: "http",
    hostname: "127.0.0.1:5500",
    versionPath: "/api/v2",
    cache: true,
    timeout: 60 * 1000,
    cacheImages: true
}
```

Exercise 1: JS Objects and JSON



JS Object to JSON

Populate the form and click "Add Person" to display JSON

Exercise 1 demonstrates the transformation of JS objects into JSON, and the use of an API for fetching data. When data is entered into the form fields (manually or by clicking the Populate Form button) and the Add Person button is clicked, the form data is copied into a JS object, added to a collection of objects, converted to JSON, and displayed in the output box on the right. The Clear Person Records button removes all JS objects and JSON.

The form field elements:

- First Name: ID = “fname”
- Last Name: ID = “lname”
- Email: ID = “email”

The button elements:

- Populate Form: ID = “ex1randomPerson”
- Add Person: ID = “ex1addPerson”
- Clear Person Records: ID = “ex1clearData”

Other elements:

- Output JSON element: ID = “ex1json”
- Loading spinner: ID = “ex1loader”

0. Add the global variables for exercise one (listed in the Global Variables section above).
1. For each of the exercise 1 buttons, create a button click event callback in the ready() event:

```
$('document').ready(function () {
    // Exercise 1 setup
    $('#ex1randomPerson').click(function (e) {

    });

    $('#ex1addPerson').click(function (e) {

    });

    $('#ex1clearData').click(function (e) {

    });
```

2. In the ex1randomPerson function, call the getRandomPersonData() function.

3. In the **ex1addPerson** function, call the **addPerson()** function with the array elements of the **getFormFields()** function.

4. In the **ex1clearData** function, call the **clearPeople()** function.

```
$('document').ready(function () {
    // Exercise 1 setup
    $('#ex1randomPerson').click(function (e) {
        getRandomPersonData();
    });

    $('#ex1addPerson').click(function (e) {
        addPerson(...getFormFields());
    });

    $('#ex1clearData').click(function (e) {
        clearPeople();
    });
```

*NOTE: the **getFormFields()** function returns an array. The **addPerson()** function accepts the array elements separately. Instead of saving the array to a variable and referencing each element, we can use the **spread operator (...)** at the beginning of the function name. The spread operator expands iterables (like an array) in place.*

```
        // this
        addPerson(...getFormFields());

        // is the same as this
        let arr = getFormFields();
        addPerson(arr[0], arr[1], arr[2]);
```

5. Locate the function called getFormFields(). Inside the function add the following code:

```
function getFormFields() {
    console.log("Get form fields");
    return [
        $('#fname').val(),
        $('#lname').val(),
        $('#email').val()
    ];
}
```

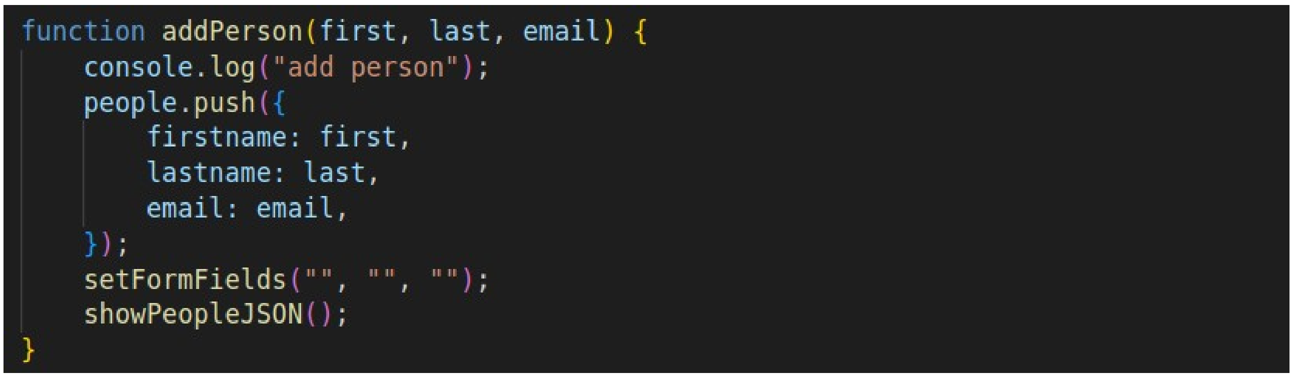
This function creates an array, populates it with the value of each input field using a jQuery selector, and returns it. It provides a quick and simple way to get all form data.

6. Locate the function called **setFormFields()**. Inside the function add the following code:

```
function setFormFields(fname, lname, email) {
    console.log("set form fields");
    $('#fname').val(fname);
    $('#lname').val(lname);
    $('#email').val(email);
}
```

This function is the get-set twin of the previous function. It takes three strings and sets the input field values to these strings.

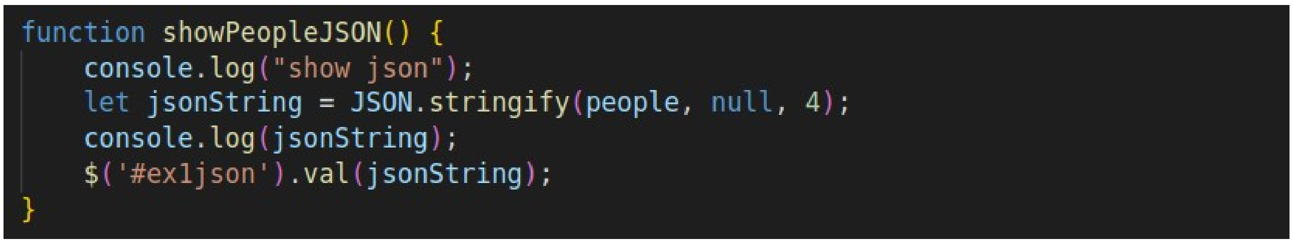
7. Locate the function called **addPerson()**. Inside the function add the following code:



```
function addPerson(first, last, email) {
    console.log("add person");
    people.push({
        firstname: first,
        lastname: last,
        email: email,
    });
    setFormFields("", "", "");
    showPeopleJSON();
}
```

This function takes three values and creates a Person object containing these values. The object is then appended to the global **people** array using **array.push()**. After this, the form fields are cleared using **setFormFields()** and the **showPeopleJSON()** function is called to handle the next part of functionality.

8. Locate the function called **showPeopleJSON()**. Inside the function add the following code:

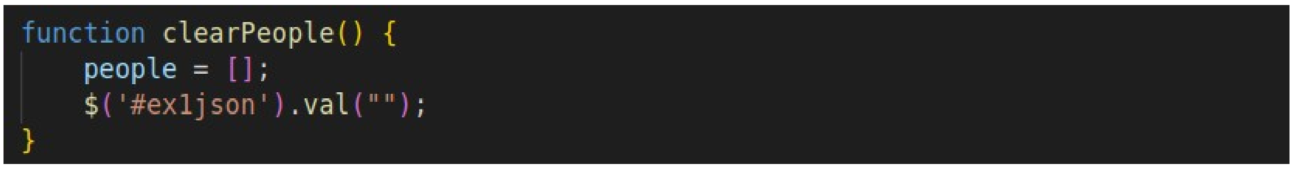


```
function showPeopleJSON() {
    console.log("show json");
    let jsonString = JSON.stringify(people, null, 4);
    console.log(jsonString);
    $('#ex1json').val(jsonString);
}
```

This function converts the **people** array to a JSON string, and sets the contents of element **ex1json** to the json string. Optional parameters are added with **null** for the replacer array, and **4** as the number of spaces to use for indentation.

NOTE: The optional parameters can be excluded as they only benefit the visual presentation of the JSON string.

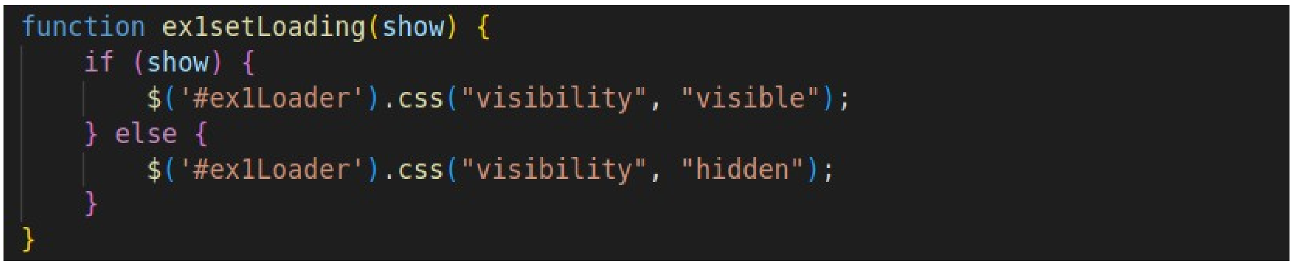
9. Locate the function called **clearPeople().** Inside the function add the following code:



```
function clearPeople() {
    people = [];
    $('#ex1json').val("");
}
```

This function clears the contents of the **people** array and the display textarea.

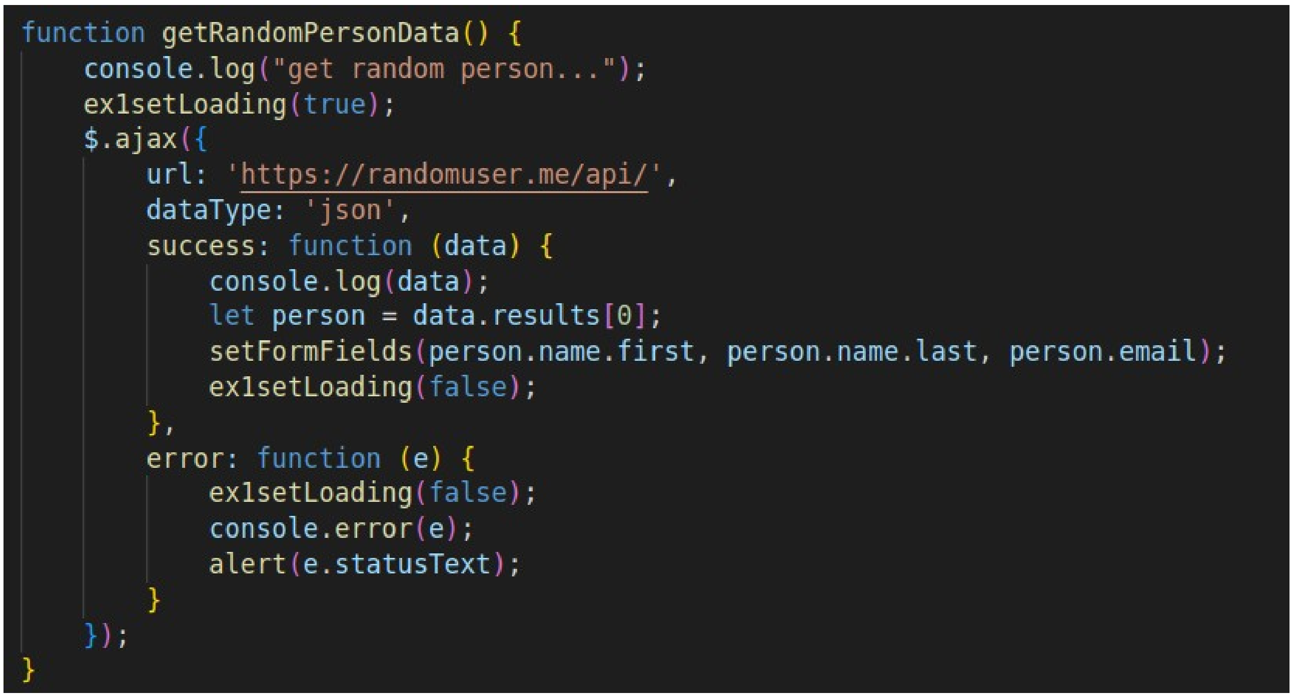
10. Locate the function called **ex1setLoading()**. Inside the function add the following code:



```
function ex1setLoading(show) {
    if (show) {
        $('#ex1Loader').css("visibility", "visible");
    } else {
        $('#ex1Loader').css("visibility", "hidden");
    }
}
```

This function shows and hides a loading spinner overlay that covers the form. When asynchronous code runs it indicates to the user that an action is happening while they wait. Once async code is complete, the spinner can be hidden again.

11. Locate the function called **getRandomPersonData()**. Inside the function add the following code:



```
function getRandomPersonData() {
    console.log("get random person...");
    ex1setLoading(true);
    $.ajax({
        url: 'https://randomuser.me/api/',
        dataType: 'json',
        success: function (data) {
            console.log(data);
            let person = data.results[0];
            setFormFields(person.name.first, person.name.last, person.email);
            ex1setLoading(false);
        },
        error: function (e) {
            ex1setLoading(false);
            console.error(e);
            alert(e.statusText);
        }
    });
}
```

This function calls a third party API to get random person data to populate the form. It displays the loading spinner, makes an asynchronous GET request to an API endpoint, and provides callback functions for success and error results. The success function populates the form, and the error function shows an alert. Both callback functions hide the spinner.

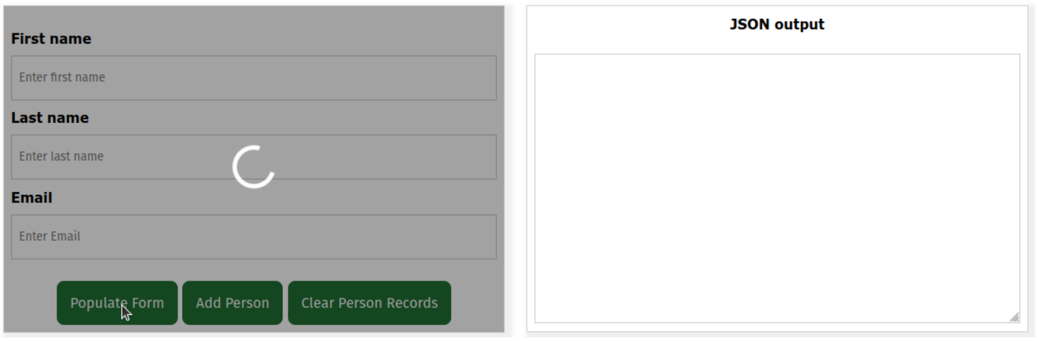
12. Save your work and test the output.

Clicking the **Populate Form** button should temporarily show the loading spinner, then hide and display the form with random data.

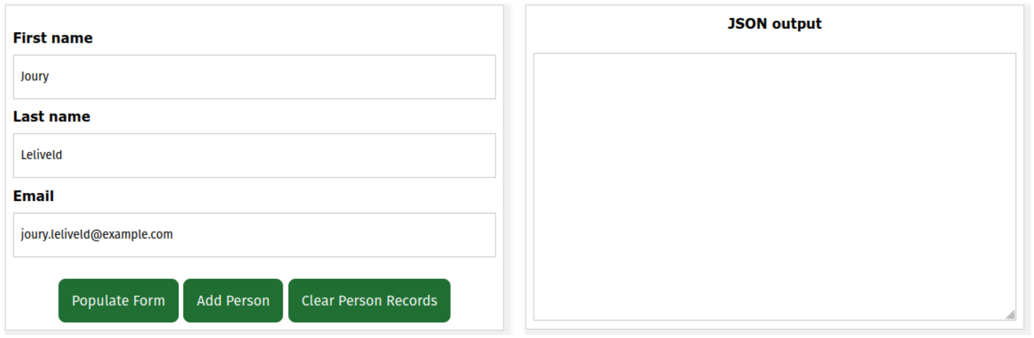
Clicking the **Add Person** button should submit the form data to appear in JSON string format in the textarea.

Clicking the **Clear Person Records** should remove all person data from the textarea.

Populate form click:



Success callback:

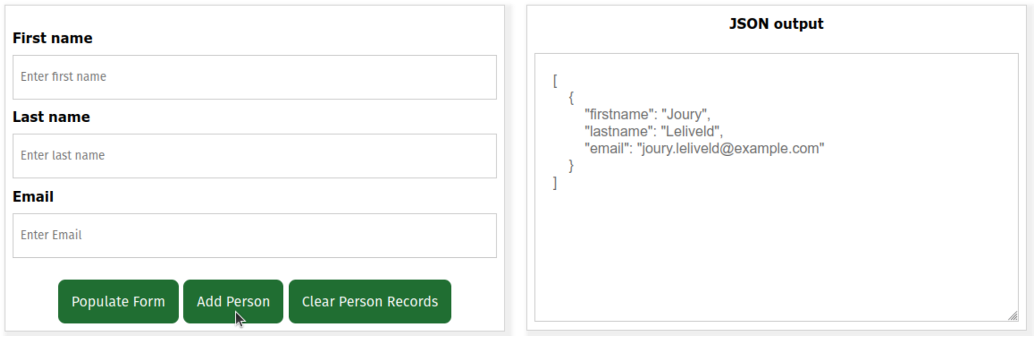


First name: Joury

Last name: Leliveld

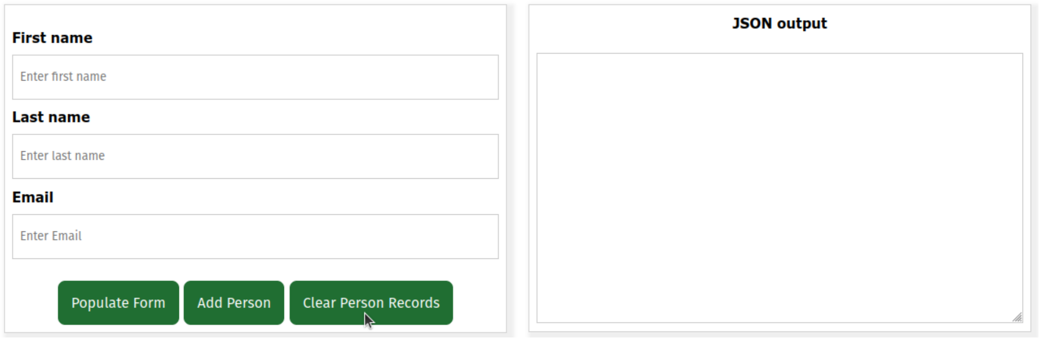
Email: joury.leliveld@example.com

Add Person click:



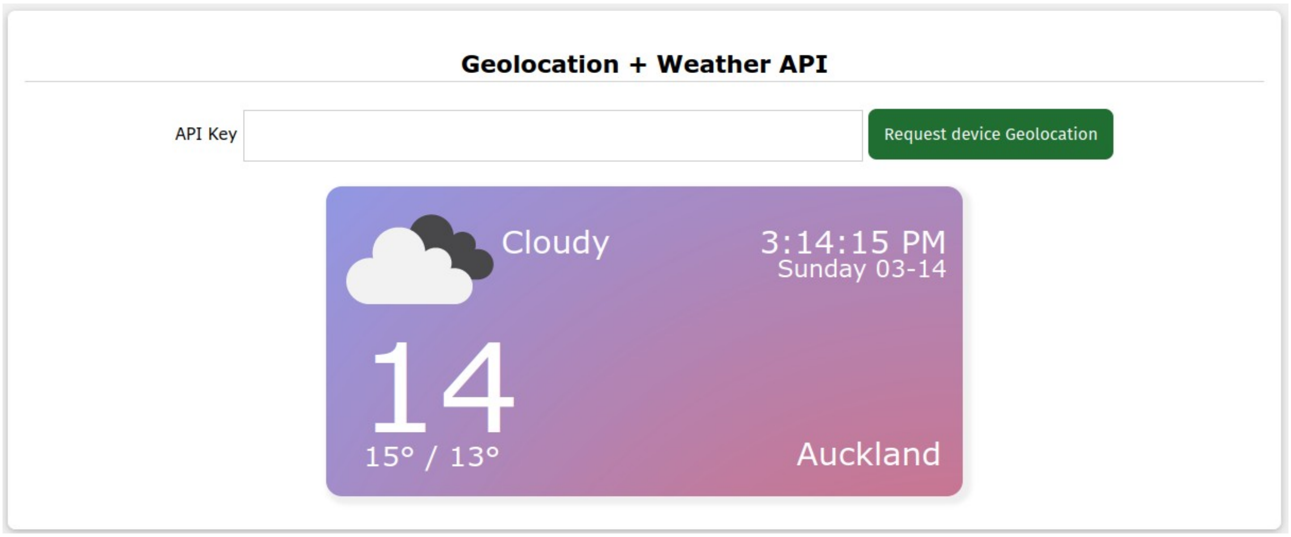
```
[
    {
        "firstname": "Joury",
        "lastname": "Leliveld",
        "email": "joury.leliveld@example.com"
    }
]
```

Clear Person click:



Exercise 1 complete.

Exercise 2: Geolocation + Weather API



Exercise 2 demonstrates the use of geolocation data to request data and display a custom widget with weather information about the user location.

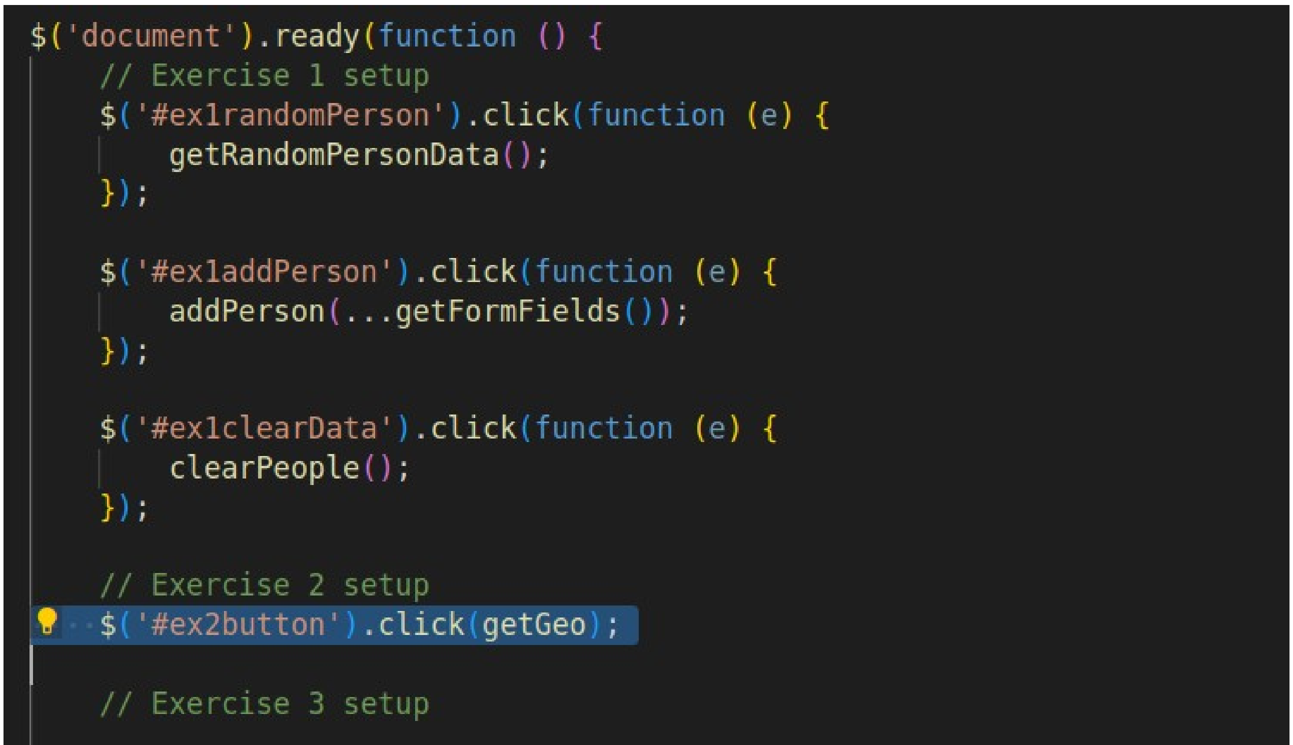
The widget elements:

- Weather Panel: class = "**weather-panel**"
- Loading spinner: ID = "**ex2loader**"
- Weather icon: ID = "**weather-icon**"
- Weather description: ID = "**weather-description**"
- Weather temperature: ID = "**weather-temp**"
- Weather temperature range: ID = "**weather-temp-range**"
- Weather time: ID = "**weather-time**"
- Weather date: ID = "**weather-date**"
- Weather location: ID = "**weather-location**"

The other elements:

- Button - Request Device Location: ID = "**ex2button**"
- Text Input – API Key: ID = "**apikey**"

1. In the **document.ready()** event function, assign the **getGeo()** function to the **ex2button** click event:



```
$('document').ready(function () {
    // Exercise 1 setup
    $('#ex1randomPerson').click(function (e) {
        getRandomPersonData();
    });

    $('#ex1addPerson').click(function (e) {
        addPerson(...getFormFields());
    });

    $('#ex1clearData').click(function (e) {
        clearPeople();
    });

    // Exercise 2 setup
    $('#ex2button').click(getGeo);

    // Exercise 3 setup
```

2. Locate the function called **getGeo()**. Inside the function, add the following code:

```
function getGeo() {
    if (!navigator.geolocation) {
        alert("Geolocation services not available");
        return;
    }
    if ($('#apikey').val() === "" || $('#apikey').val() === null) {
        alert("Weather request requires API key. Please enter a valid key.");
        return;
    }
    navigator.geolocation.getCurrentPosition(function (position) {
        let lat = position.coords.latitude;
        let long = position.coords.longitude;
        let apikey = $('#apikey').val();
        setLoading('#ex2loader', true);
        $.ajax({
            url: `https://api.openweathermap.org/data/2.5/weather?lat=${lat}&
            lon=${long}&units=metric&appid=${apikey}`,
            dataType: 'json',
            success: function (data) {
                console.log(data);
                setWeather(data);
                setLoading('#ex2loader', false);
            },
            error: function (e) {
                console.error(e);
                setLoading('#ex2loader', false);
                alert(e.statusText);
            }
        });
    });
}
```

Copyable URL string:
`` `https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${long}&units=metric&appid=${apikey}` ``
The API key is on Moodle, or you can create an account at https://openweathermap.org

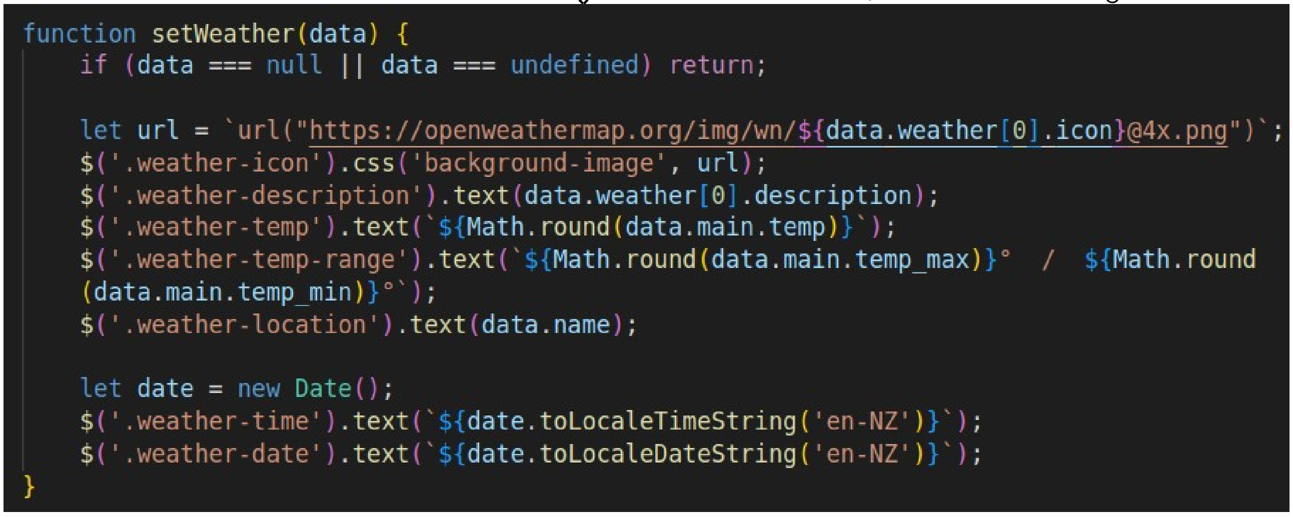
This function uses the Geolocation Web API to fetch user position, then calls the OpenWeatherMap API with the Latitude and Longitude as parameters. Two callback functions are provided for success and error. Success receives a data object and passes it to the **setWeather()** function for processing. Error shows an alert to the user of the problem. A loading spinner is shown during the asynchronous operation, and hidden in both callbacks.

3. Locate the function called **ex2setLoading()**. Inside the function, add the following code:

```
function ex2setLoading(show) {
    if (show) {
        $('#ex2Loader').css("visibility", "visible");

    } else {
        $('#ex2Loader').css("visibility", "hidden");
    }
}
```

This function operates the same way as the spinner function in exercise 1. Both functions can be consolidated into one function by passing the selected element as an input parameter.

4. Locate the function called **setWeather()**. Inside the function, add the following code:



```
function setWeather(data) {
    if (data === null || data === undefined) return;

    let url = `url("https://openweathermap.org/img/wn/${data.weather[0].icon}@4x.png")`;
    $('.weather-icon').css('background-image', url);
    $('.weather-description').text(data.weather[0].description);
    $('.weather-temp').text(`${Math.round(data.main.temp)}`);
    $('.weather-temp-range').text(`${Math.round(data.main.temp_max)}°  /  ${Math.round
    (data.main.temp_min)}°`);
    $('.weather-location').text(data.name);

    let date = new Date();
    $('.weather-time').text(`${date.toLocaleTimeString('en-NZ')}`);
    $('.weather-date').text(`${date.toLocaleDateString('en-NZ')}`);
}
```

This function takes the weather data from the API, finds all the data needed for the weather widget, and applies the values to the widget elements.

While the weather API provides string and number data, it does not provide a link to a relevent image or icon. Instead, the API provides the name of the icon file that can be accessed with the constructed URL.

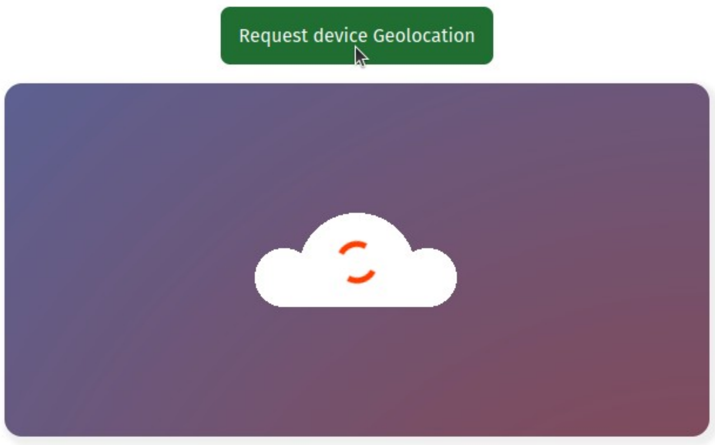
Copyable code:

```
function setWeather(data) {
    if (data === null || data === undefined) return;

    let url = `url("https://openweathermap.org/img/wn/$
{data.weather[0].icon}@4x.png")`;
    $('.weather-icon').css('background-image', url);
    $('.weather-description').text(data.weather[0].description);
    $('.weather-temp').text(`${Math.round(data.main.temp)}`);
    $('.weather-temp-range').text(`$
{Math.round(data.main.temp_max)}°  /  $
{Math.round(data.main.temp_min)}°`);
    $('.weather-location').text(data.name);

    let date = new Date();
    $('.weather-time').text(`${date.toLocaleTimeString('en-
NZ')}`);
    $('.weather-date').text(`${date.toLocaleDateString('en-
NZ')}`);
}
```

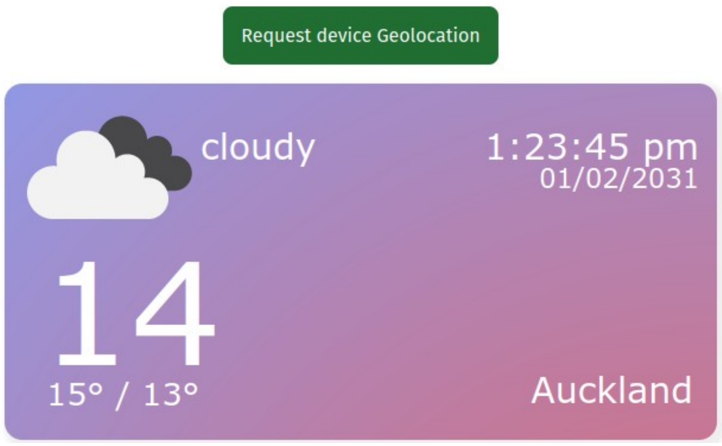
5. Save your work and test the output.

Clicking the **Request Device Geolocation** button should request geolocation permission from the user, and if granted, show a loading spinner over the weather panel, load weather information, then hide the spinner.

Button click:

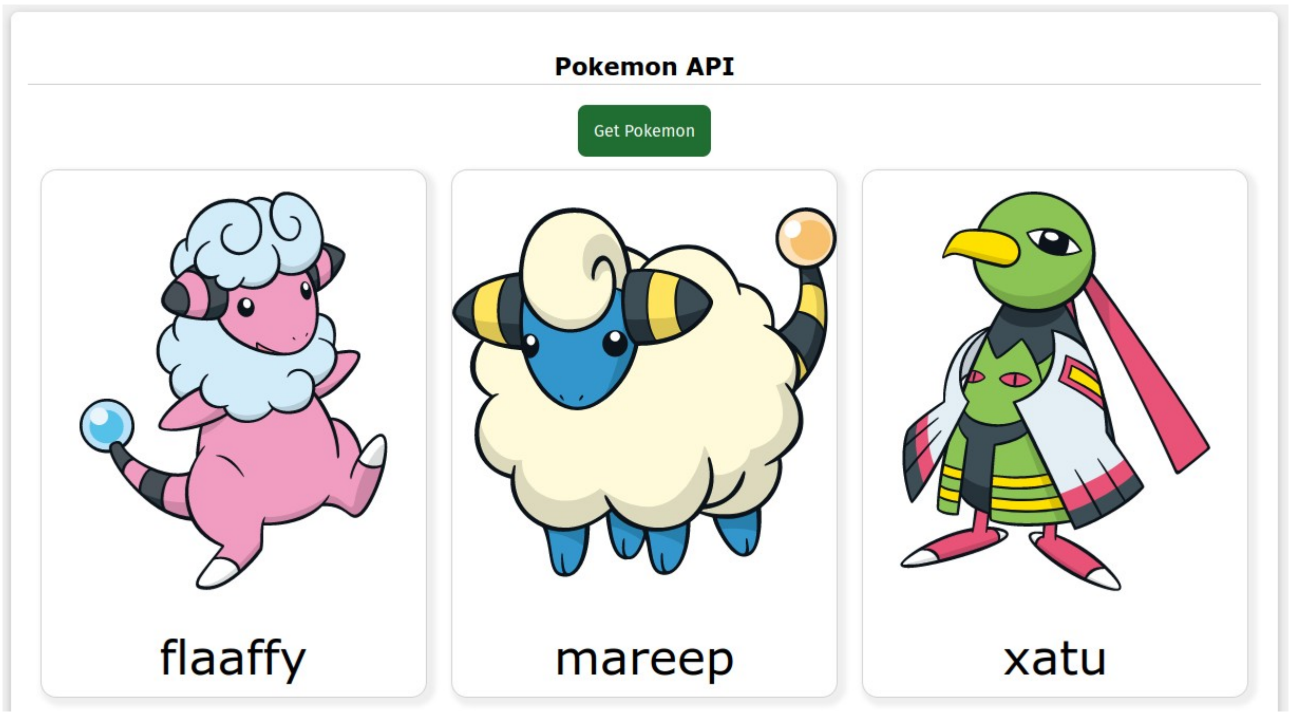


Result:



Exercise 2 complete.

Exercise 3: Pokemon API



Exercise 3 demonstrates the use of a third party API using a wrapper library to abstract connection and query complexity.

The widget elements:

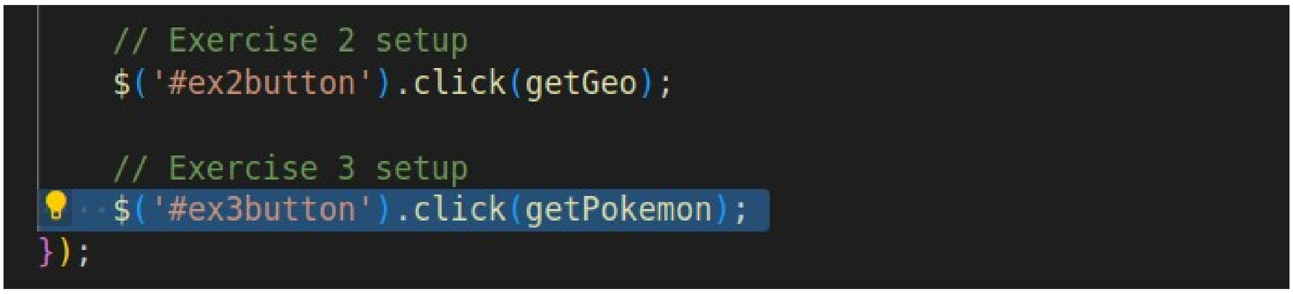
- Content container: ID = “**ex3content**”

The button elements:

- Get Pokemon: ID = “**ex3button**”

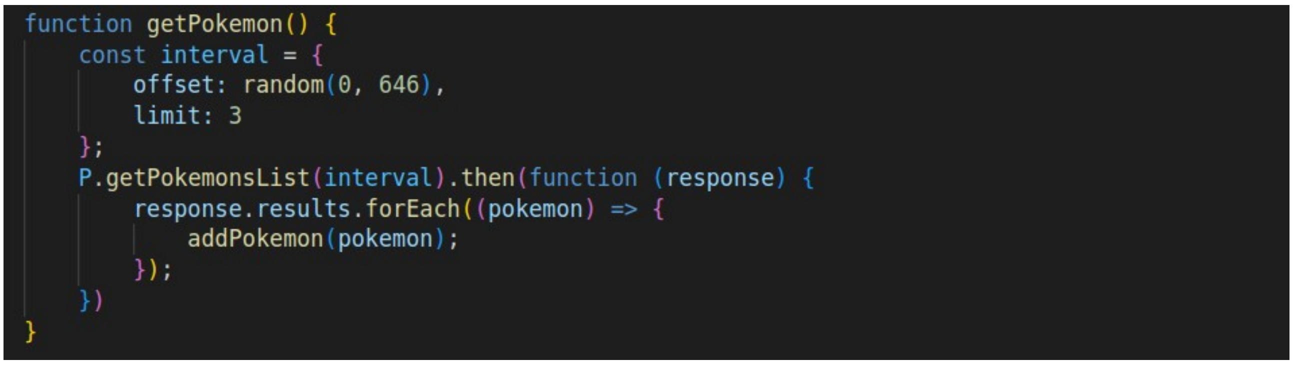
0. Add the global variables for exercise one (listed in the **Global Variables** section above).

1. In the **document.ready()** event function, assign the **getPokemon()** function to the **ex3button** click event:



```
    // Exercise 2 setup
    $('#ex2button').click(getGeo);

    // Exercise 3 setup
    $('#ex3button').click(getPokemon);
});
```

2. Locate the function called **getPokemon()**. Inside the function, add the following code:

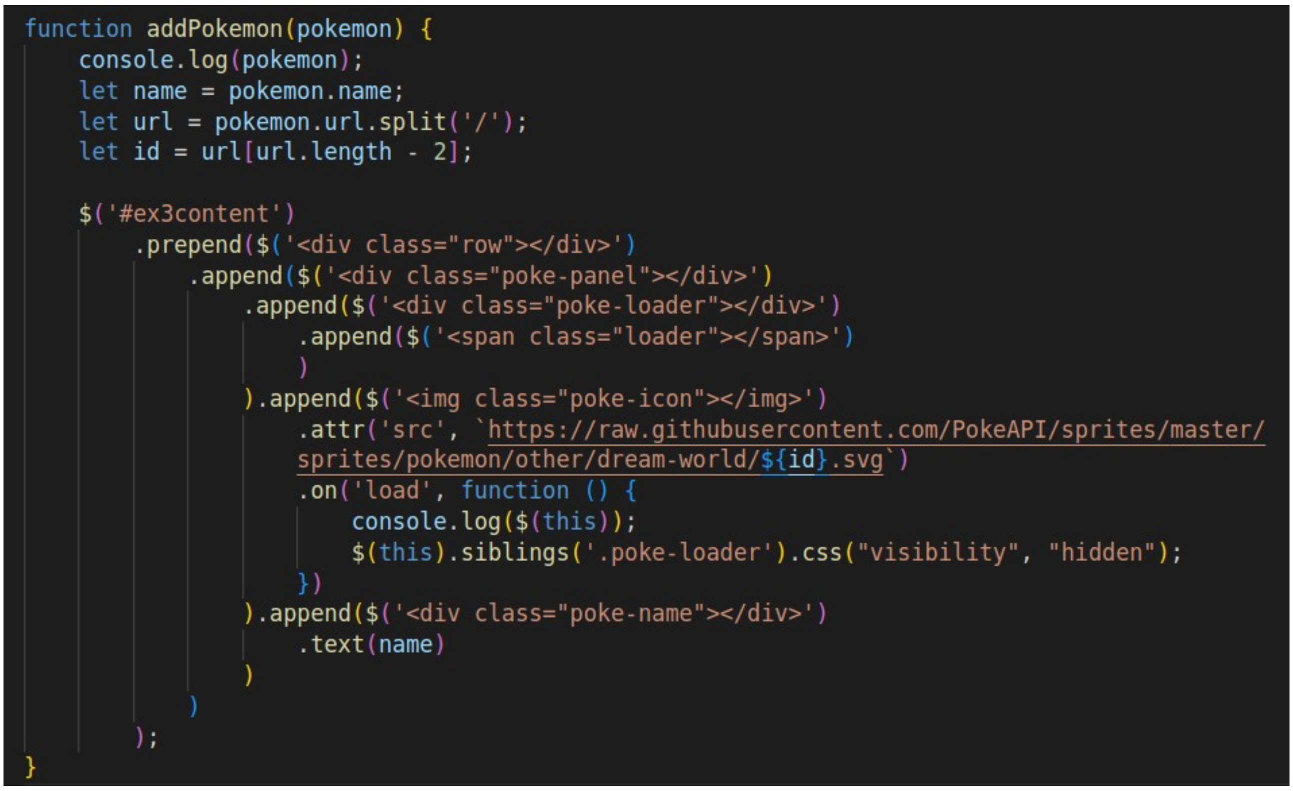


```
function getPokemon() {
    const interval = {
        offset: random(0, 646),
        limit: 3
    };
    P.getPokemonsList(interval).then(function (response) {
        response.results.forEach((pokemon) => {
            addPokemon(pokemon);
        });
    })
}
```

This function uses **P** to access the Poke API wrapper library and call the **getPokemonsList()** function. Using **.then** to provide an asynchronous callback function, the response object results are looped over passed to the **addPokemon()** function for processing.

The optional parameters are passed in to limit the impact of the API call.
Offset: sets the starting index to request Pokemon data from.
Limit: set the number of pokemon to request data for.

3. Locate the function called **addPokemon()**. Inside the function, add the following code:



```
function addPokemon(pokemon) {
    console.log(pokemon);
    let name = pokemon.name;
    let url = pokemon.url.split('/');
    let id = url[url.length - 2];

    $('#ex3content')
        .prepend($('<div class="row"></div>')
            .append($('<div class="poke-panel"></div>')
                .append($('<div class="poke-loader"></div>')
                    .append($('<span class="loader"></span>')
                    )
                ).append($('<img class="poke-icon"></img>')
                    .attr('src', `https://raw.githubusercontent.com/PokeAPI/sprites/master/
                    sprites/pokemon/other/dream-world/${id}.svg`)
                    .on('load', function () {
                        console.log($(this));
                        $(this).siblings('.poke-loader').css("visibility", "hidden");
                    })
                ).append($('<div class="poke-name"></div>')
                    .text(name)
                )
            )
        );
}
```

This function gets the pokemon name and ID from the input data, constructs a display panel with jQuery, and populates the panel with data about the pokemon.
The ID is found by splitting the URL string at each forward slash and taking the second-last substring.

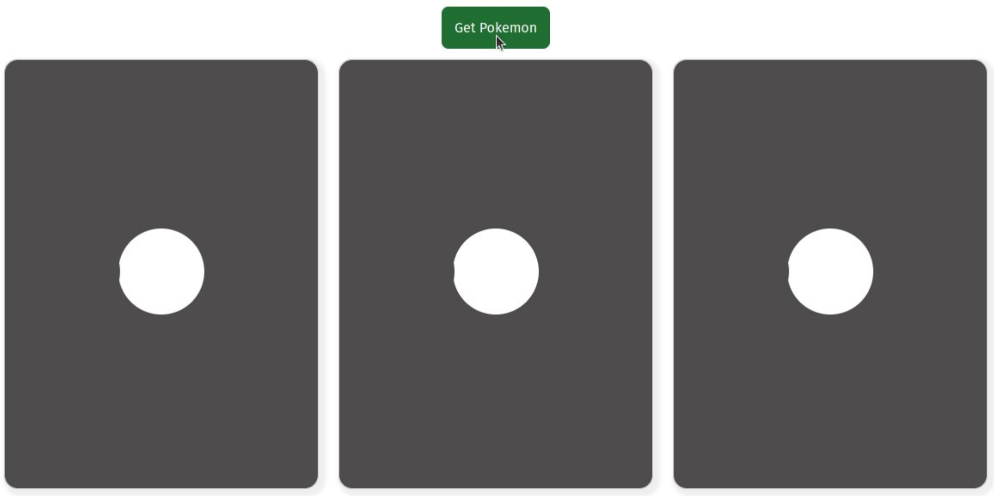
URL string: `"https://pokeapi.co/api/v2/pokemon/369/"`
URL split: `["https:","","pokeapi.co","api","v2","pokemon","369","" ]`

Copyable code for jQuery panel construction:
```
$('#ex3content')
        .prepend($('<div class="row"></div>')
            .append($('<div class="poke-panel"></div>')
                .append($('<div class="poke-loader"></div>')
                    .append($('<span class="loader"></span>')
                    )
                ).append($('<img class="poke-icon"></img>')
                    .attr('src', `https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/
pokemon/other/dream-world/${id}.svg`)
                    .on('load', function () {
                        console.log($(this));
                        $(this).siblings('.poke-loader').css("visibility", "hidden");
                    })
                ).append($('<div class="poke-name"></div>')
                    .text(name)
                )
            )
        );
```

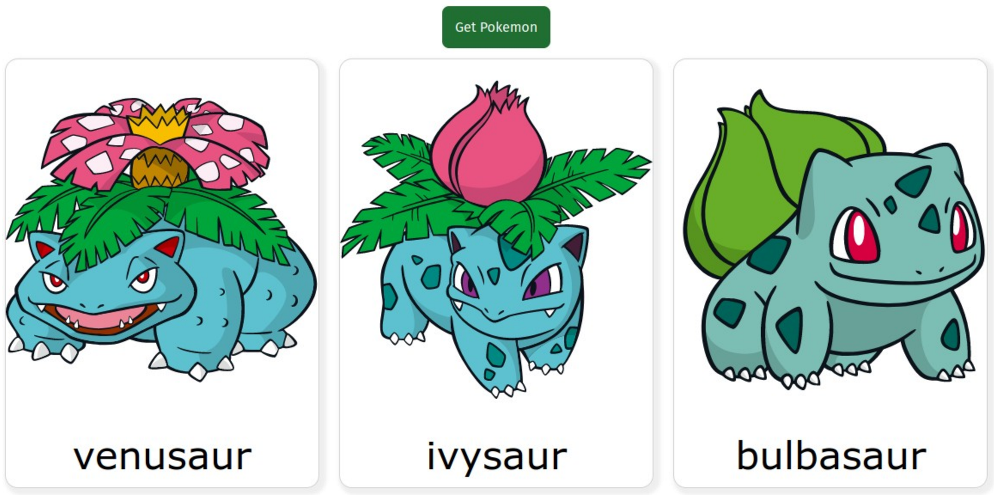
4. Save your work and test the output.

Clicking the **Get Pokemon** button should create three panels with loading spinners visible on each. Once the data has loaded, the spinners should disappear to show the image and name of random pokemon.

Button click:



Loaded:



Exercise 3 complete.

Next Steps

- Create or copy different loading spinners
  - https://cssloaders.github.io/
- Modify Exercise 2 to make use of more data available in the response object.
- Incorporate Web Storage API to save pokemon images locally.
- Explore other APIs and create proof-of-concepts for use in projects.
  - https://github.com/public-api-lists/public-api-lists
  - https://github.com/lakhassane/APIs-Inspiration
  - https://github.com/Vets-Who-Code/api-list

Exercises complete.