Weather Web App

You will build a weather app that allows you to search the weather by city, and get a daily or five day forecast.

See a sample version of the app here: http://jQuery-weather.surge.sh

Starter code

Each project comes with some starter code. In each starter code folder you'll find three files:

- `index.html` - This is where you will be writing all of your HTML code for your project.
- `styles.css` - This is where you will be writing all of your CSS code for your project.
- `app.js` - This is where you will be writing all of your JavaScript and jQuery code for your project.

To download your starter code go to:
https://github.com/lighthouse-labs/front-end-fundamentals-js-starter-files

Then, click the **Clone or download** button, and then click **Download ZIP**. Then, unzip your files, and open them in a text editor.

Before we start writing JavaScript in our new jQuery projects, we need to discuss how to add JavaScript to our HTML.

Adding jQuery to your project

In the case of adding the jQuery library, there are two common options.

1. Download jQuery

The entire jQuery library can be downloaded as a single JavaScript file. To include jQuery in your project this way, you download jQuery, add the file to your project folder, and then reference it in a script tag in your HTML document.

```
<script type="text/javascript" src="./jquery.js"></script>
```

We're going to add jQuery to the bottom of the body. In other, words, the `<script>` tag in the previous example should go just **above** the `</body>` tag.

2. Use a CDN

Alternatively, you can you a CDN (Content Delivery Network). This means that when a user visits your website, the browser will load jQuery from a source other than your computer or the server hosting your website.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
```

We're going to add jQuery to the bottom of the body. In other, words, the `<script>` tag in the previous example should go just **above** the `</body>` tag.

Adding your own JavaScript code to your project

Once you have jQuery loading, you will then need to add a link to your JavaScript file in your HTML document:

```
<script type="text/javascript" src="./index.js"></script>
```

We're going to add our JavaScript to the bottom of the body, but after jQuery. In other, the `<script>` tag in the previous example should go just **below** the `<script>` that loads jQuery, and **above** the `</body>` tag.

Tasks

- Add a `<script>` tag to your `index.html` file to include jQuery in your project
- Add a `<script>` tag to your `index.html` file to include your `index.js` file
- Add code that `console.log`s a string (like "Hello, world!") in your `index.js` file
- Open up your `index.html` page in the browser
- Open up your Dev Tools console
- Confirm that the string you passed to the `console.log` is visible in your console – this means your JavaScript file is loading properly!

Form Submission

Right now, when you click **Search** or press the `Enter` key on your form, you'll notice that the page refreshes. When we submit a form, our browser will gather up the information that has been inputted into it, send it to a specified server, and then refresh the page. In our case we don't want this to happen, so we're going to override the default behaviour and send the data to a server ourselves, using jQuery.
To do this we can use the `preventDefault` function. The `preventDefault` function allows us to cancel events or to prevent the browser from taking a default action. For example, we could also use it to prevent the browser from navigating to a new URL when a link is clicked.

preventDefault

Call the `preventDefault` function on the event that is returned as the first argument to the function on your event listener.

```
$('form').on('submit', function(event) {
  event.preventDefault()
})
```

One of the things you can do with jQuery is hide and show elements on your page.

Hide and Show Elements

Tasks

- Add code to hide your search input and button after the form has been submitted
- Add a placeholder attribute to your text input by default
- Add a jQuery animation to the elements as they are hidden and shown on your page
- Add some custom CSS and personalize your project

jQuery reference

click
Listens/waits for clicks on the selected elements.

```
$('p').on('click', function() {
  console.log("A paragraph was clicked!")
})
```

fadeOut
Will cause an element to slowly fade away.

```
$('.box').fadeOut(2000)
```

slideDown
Will cause an element to slide down and into view.

```
$('.box').slideDown(3000)
```

slideUp
Will cause an element to slide up and out of view.

```
$('.box').slideUp(5000)
```

addClass
Will add a class to the selected HTML element(s).

```
$('p').addClass('highlight')
```

removeClass
Will remove a class from the selected HTML element(s).

```
$('p').removeClass('highlight')
```

toggleClass

This works kind of like a light switch. If the selected element(s) has the class, it will remove it. If the element(s) doesn't have the class, it will add it!

```
$('p').toggleClass('highlight')
```

CSS reference

display

The display property can be set to:

- `block`
- `inline`
- `inline-block`
- `flex`
- `none`

transition

The CSS transition property allows you to customize how a value handles a changing value on a specific CSS property.

```
p {
  color: blue;
  transition: color 2s linear 3s;
}
```

HTML reference

input

Input elements can have several attributes:

- `placeholder`
- `type`
- `name`
- `value

```
<input name="" placeholder="" value="" type="" />
```

Show Results

Tasks

- Add a div to your HTML that will hold the results of your search
- Add code to show your new results div when your form is submitted
- Add animations to your elements that are being shown and hidden

jQuery reference

hide
Will set the `display` property of an element to `none`, hiding it from the user.

```
$('.box').hide(500)
```

show
Will make an element visible by restoring its `display` property to its original value.

```
$('.box').show(500)
```

jQuery animations
jQuery provides us with some methods that let us animate an element being shown and hidden on the page. Some common ones are:

- slideUp
- slideDown
- fadeIn
- fadeOut

Listen for Focus

Tasks

- Add code so that when a user clicks on an input element, the border of this input element changes colour
- Use CSS to style the text input and submit button of your form

jQuery reference

focus
The focus event is triggered when a user focuses on an element, for example an input element.

```
$('form input').on('focus', function() {
  // add code here
})
```

blur
The blur event is triggered when a user is no longer focused on an element, or, in other words, when they "focus out" of an element. They can do this in several ways, for example by clicking elsewhere on the page, or by moving to the next input in the form using the `Tab` key on their keyboard.

```
$('form input').on('blur', function() {
  // add code here
})
```

change
The change event is triggered when the user changes the value of the input element. When this occurs depends on the type of element the user is interacting with, but generally speaking this event will be triggered *after* the user has made a change.

```
$('form input').on('change', function() {
  // add code here
})
```

CSS reference

:focus (pseudo selector)
Will add styles to an HTML element when its in a focus state.

```
.example:focus {
  background-color: green;
```

```
}
```

border-radius
Will add rounded corners to an element.

```
div {
  border-radius: 4px;
}
```

padding
Will add space between the content and the border of an element.

```
div {
  padding: 10px;
}
```

outline
The outline property is a line that is drawn **around** elements (outside of the border). It's **not** added to an element's dimensions.

```
div {
  outline: [width] [style] [color];
}
```

Just like with to the `border` CSS property, you can specify the `width`, `style` and `color` of the outline.

```
.text {
  outline: 1px dotted red;
}
```

Style Your Page

So far, our page is fairly basic. We have a form and a results div that is shown and hidden. Take a moment to start filling out the rest of the page. It's helpful to make a quick sketch of what you would like your page to look like before you start writing code. Think back to the project planning you did for your HTML and CSS project, the same process will be helpful here.

Tasks

- Add a Header with the title of your page
- Add a Footer
- Add a custom font (https://fonts.google.com/)
- Start adding a colour scheme to your project (https://coolors.co/)
- Adjust the layout using your Flexbox skills

Trigger a Transition With JQuery

We've learned about CSS animations and transitions, which we triggered with a hover. We can also trigger transitions and animations with jQuery. One way to do this is by adding and removing classes from our HTML elements.
Consider the following example:
HTML:

```
<div class="circle"></div>
<button>Change Colour</button>
```

CSS:

```
.circle {
  height: 200px;
  width: 200px;
  background-color: pink;
  border-radius: 50%;
  transition-property: background-color;
  transition-duration: 2s;
  transition-timing-function: ease-in;
}

.green {
  background-color: green;
}
```

JavaScript:

```
$('button').on('click', function() {
  $('.circle').toggleClass('green');
})
```

Now, when the button is clicked, the `green` class will be added and removed from the circle. This sets its background colour to green or pink, respectively.

Tasks

- Play around with triggering a CSS transition by adding or removing a class from an element in your jQuery project

jQuery reference

addClass
Will add a class to the selected HTML element(s).

```
$('p').addClass('highlight')
```

removeClass
Will remove a class from the selected HTML element(s).

```
$('p').removeClass('highlight')
```

toggleClass

This works kind of like a light switch. If the element(s) has the class, it will remove it. If the element(s) doesn't have the class, it will add it!

```
$(‘p').toggleClass('highlight')
```

JSON Exercise

The App lets users search for some information from an outside source. Next class, we'll learn all about how to implement this in jQuery.

For today, we've created a JavaScript object that mimics the responses you will get from the APIs.

Go through this sample response, and take note of the information provided. What information would you like to display to your users?

Weather app response


```
{
  base: "stations",
  clouds: {
    all: 1
  },
  coord: {
    lat: 43.65,
    lon: -79.38
  },
  dt: 1507510380,
  id: 6167863,
  main: {
    humidity: 77,
    pressure: 1014,
    temp: 17.99,
    temp_max: 20,
    temp_min: 16
  },
  name: 'Downtown Toronto',
  sys: {
    type: 1,
    id: 2117,
    message: 0.0041,
    country: 'CA',
    sunrise: 1507548290,
    sunset: 1507589027,
    type: 1
  },
  visibility: 16093,
  weather: [
    {
      description: 'clear sky',
      icon: '01n',
      id: 800,
      main: "Clear"
    }
```

```
  ],
  wind: {
    deg: 170,
    speed: 1.5
  }

}
```

- Copy and paste the object for your project to the **top** of your `index.js` file
-

You will use this data in today's exercises.

Tips & Tricks

Objects Review

Objects are made up of key value pairs. The values can be set to any data type, even other objects.

```
var order = {
  items: [
    {
      name: 't-shirt',
      price: 10
    },
    {
      name: 'shoes',
      price: 30
    }
  ],
  orderId: 12345,
  shipping: {
    address: '46 Spadina Road',
    name: 'Larry Ducksworth',
    city: 'Toronto',
    province: 'ON'
  }
}
```

If we wanted to access the items in this object we could write:
`order.items`
If we wanted to get the **second** item, we could write:
`order.items[1]`
We can also access object values using square brackets notation:
`order['items'][0]`
It may be helpful to play around with accessing different elements in your sample response object before you start adding the information to your page.

Remember that the `debugger` keyword can be used to create a breakpoint, and allow you to stop code as it's running, and investigate what's happening at that moment in the program's execution.

Creating HTML

When we get a response from our API, we want to display that information to our user. We've already added code to show our results `div`, but we'll need to make new elements to hold this new information.

Tasks

Weather app

- Write a function that takes the response object as an argument, and appends an HTML element containing the value of the `name` key to your page.
- Add code so this function is called when a user submits the search form
-

Tip: When you add your new HTML element to your page, make sure it goes *inside* your results div.

Add More Elements

Now that we've gotten started using our example JSON responses, it's time to decide what information we would like to display to our users.

Tasks

- Decide on which pieces of information you would like to display to your users
- Add more code to the function you created in the previous exercise (Creating HTML Elements) that displays all of the information you want to display to your users

Tips & Tricks

- Keep your console open as you code
- Go slowly, break the task down into small pieces. No need to add all the information from your example JSON response object all at once, one at a time is fine
- Write a little bit of code, test it, and repeat

jQuery and JavaScript reference

append
Will add a new element as the **last** child of your selected element(s).
```
$('article').append("<p>I'll be the last paragraph in the article</p>")
```

prepend
Will add a new element as the **first** child of your selected elements(s).
```
$('article').prepend("<p>I'll be the first paragraph in the article</p>")
```

replaceWith
Will replace the content of your selected element(s) with the new content.
```
$('li').replaceWith('Hello')
```

html
The `html` function will do one of two things. If the function is called with no parameters, it will get the HTML contents of the **first** element that matches its selector. If the function is called with a parameter, it will set the content of all of

the HTML elements that match its selector (and the new content will be the parameter's value).
In this first example, the `html` function will return the content of the first `<h2>` element on the page.

```
$('h2').html()
```

In this second example, the `html` function will change the content of **all** of the `<p>` tags on the page to be "**All** of the paragraph tags have this content".

```
$('p').html("<strong>All</strong> of the paragraph tags have this content")
```

For loops
For loops are used for looping over elements in an array.

```
var students = [{ name: 'Bob', grade: 'A-' }, { name: 'Dylan', grade: 'B+' }]

for (var i = 0; i < students.length; i++) {
  console.log(students[i])
}
```

Capturing User Input

Next class we'll be taking the user input and using it to send a request to a server. To prepare for that, we're going to start by logging out the user input to our console.

Tasks

- Add code so that when the user performs a search, the content of the search input is logged to the console

jQuery and JavaScript reference

val
Will return the value of an element:
```
$('p').val()
```

console.log
Will print a value to the console:
```
console.log("Hello World”)
```

Add a Search Again Button

So you've got some elements appearing on your page when you submit your search form – great! Now, let's add a **Search Again** button that allows your users to see the search bar again, so they may find new information.

Tasks

- Add a **Search Again** button to your Header
- When the **Search Again** button is clicked, hide the previous results and show the search bar
- Allow users to search again and see the correct results
- Animate elements as they are shown and hidden

Note: You might notice when you "search again" that you suddenly have twice the amount of information than you expected in your results div. If this is the case, try to remove the old search results before you add the new ones.

jQuery reference

val
This will fetch the value of the input:
```
$('input').val()
```
This will reset the value of the input:
```
$('input').val('Hello there')
```

hide
```
  $('.box').hide(500);
```

show
```
  $('.box').show(500);
```

replaceWith
Will replace the content of your selected element(s) with the new content.
```
$('li').replaceWith('Hello')
```

remove
Will remove the selected element(s) from the page.
```
$(‘li').remove()
```

Get Your API Key

In order to use some APIs, you first need to get an API key. This API key is associated with a particular account, and basically tells the API server you're sending requests to: "Hey, I'm allowed to send you requests".
To use the Weather API we've picked out for you, you'll have to sign up for an account and find your API key in your dashboard.

Tasks

- Create an account at https://home.openweathermap.org
- Under the **My Home** section, click the **API Keys** tab
- Copy your API key
- Declare a variable at the top of your `index.js` file and assign it to your API key

Send a Get Request

Sending AJAX GET requests using jQuery

Helpfully, jQuery has a built in method for sending GET requests. The `$.get` jQuery method takes two arguments, the URL we're sending the GET request to, and the function that is called when the response comes back from the server.

```
$.get('http://www.example.com', function(data) {
  console.log(data)
})
```

It's important to note here that the request does not happen instantly. It is **asynchronous**.

Weather app api

`https://api.openweathermap.org/data/2.5/weather?units=metric&q=London`

Note: To use this API you will have to add your API key as a query param. (If you don't have an API key yet, see the previous activity). To do this, we'll add `&appid=1234` to the end of the URL.

`https://api.openweathermap.org/data/2.5/weather?units=metric&q=London&appid=1234`

Tasks

- Create a function that sends a GET request to your API and `console.log`s the response.
- Add code so that the GET request is only sent when the user submits the search form

Tips & Tricks

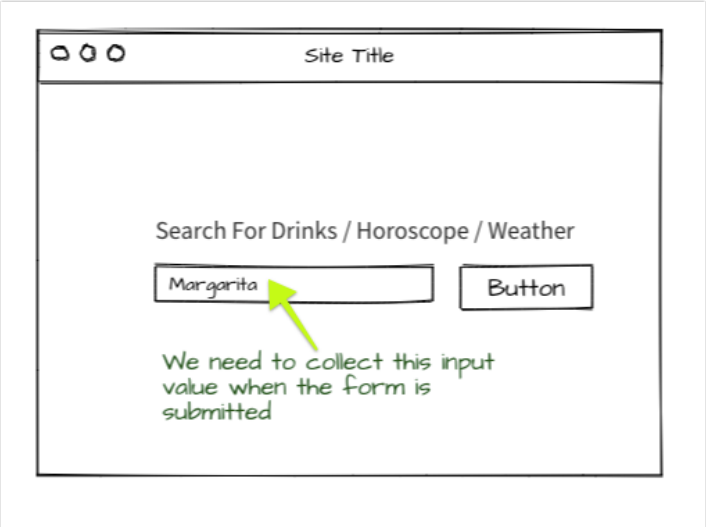
Query Params

Query params are pieces of information we can send through the URL to an API. To separate the query params from the rest of the URL we use `?` and to separate the query params themselves, we use `&`s.

`https://example.com?query_key=query_value&another_key=1234`

Search By User Input

Great! We're sending a GET request and logging the response to our console. Now, we need to take our user input, and use it in our GET request.
For example, when the form is submitted, we need to take the text that the user has submitted, and send it to the API we've chosen, so we get the correct response.



Tasks

- Collect the user input (or user selection), and save it to a variable
- Edit your code so your saved user input is added to the URL you're using in your GET request
- `console.log` the response and, using the Chrome Developer Tools, double check that the response matches the values you're inputting into the search bar

jQuery and JavaScript reference

Concatenating strings

We can concatenate strings together using the `+` operator:

```
"http://example.com?search_term=" + "margarita"
```

val

The `.val()` jQuery method is used to get or set the value of a form input element.

```
$('input').val()
```

Show Your Results

So we're sending GET requests to our API and getting responses, but our page is still displaying the information from our example responses from last week. That's not what we want; we want to display the information our users search for.

Tasks

- Delete the example response from the top of your `index.js` file
- Change your code so that instead of displaying information from your example JSON response, it displays the response from your GET request

Tips & Tricks

- Go slowly. Start by just displaying one property from the API response before building on it.
- Ask for help if and when you need it, that's what your mentors are here for.
- Keep an eye out for places where you might be repeating yourself (see if there's a way you can re-use a function, or another way to DRY up your code).

JavaScript and jQuery reference

Objects

Remember that the response from the API is a JavaScript **object** made up of **key-value pairs**. We can treat the response just like the objects we've defined in our own code.

Remember that we can access values in an object using **dot notation**.

```
var person = {
  name: 'Sherman',
  age: 5
}

// We can access the name ('Sherman') like this:
person.name
```

For loops

If some information in your response comes back as an array, you might want to loop over it in order to display each piece of information. A for loop is a great way to do this.

```
for (var i = 0; i < array.length; i++ ) {
  // your code here
}
```

append

The jQuery `append` function lets you add a new element as the last **child** of your selected element(s).

```
$('article').append("<p>I'll be the last paragraph in the article</p>")
```

prepend

The jQuery `prepend` function will add a new element as the **first** child of your selected element(s).

```
$('article').prepend("<p>I'll be the first paragraph in the article</p>")
```

after

Creates a new element as a **sibling** of your selected element(s). This element will appear **after** the selected element(s) on the page.

```
$('li').after("<li>A new sibling list item</li>")
```

before

Creates a new element as a **sibling** of your selected element(s). This new element will appear **before** the selected element(s) on the page.

```
$('li').before("<li>A new sibling list item</li>")
```