Sign Up Page

In this tutorial, we'll create a Sign-Up page using JavaScript and external Web APIs. Upon typing a desired username, a Web API will be accessed via a "fetch" HTTP Request and the user will be notified whether the username is available or not. Also, upon selecting a state, another Web API will return the list of all counties for that specific state. A drop down menu will be automatically updated with the list of counties.

Here is an incomplete version of the sign-up page. Try entering a zip code, the corresponding city should be displayed. When selecting a state, all of its counties are listed.

As we work on this project, you will practice using:

- Events and Handlers
- JavaScript Objects
- HTML Form elements
- JavaScript validation
- Web APIs
- Fetch HTTP Request
- Await and Async keywords

Optionally, you can refer to these video tutorials as a reference:
Lessons 1 to 2 Video Tutorial
Lessons 3 to 5 Video Tutorial

Planning!

Take a moment to think about the goals for this project and how we would go about achieving them.

a) What kind of HTML elements will be needed?

b) What JavaScript events should be included?

c) Why do we need to use external Web APIs?

Lab Web APIs

Here are the five Web APIs we'll use in this lab and a brief explanation. Try clicking on each of them and changing the parameter values (in red) to become familiar with the output.

1. API to get the city, latitude and longitude based on a zip code:
   https://csumb.space/api/cityInfoAPI.php?**zip**=93955

2. API to get all counties based on the two-letter abbreviation of a state:
   https://csumb.space/api/countyListAPI.php?**state**=ca

3. API to check whether a username is available or not
   (usernames already taken are: eeny, meeny, miny, maria, and john)
   https://csumb.space/api/usernamesAPI.php?**username**=eeny

4. API to get all US states along with their two-letter abbreviation
   https://csumb.space/api/allStatesAPI.php

5. API to generate a suggested password of a specific length
   https://csumb.space/api/suggestedPassword.php?**length**=8

Lesson 1: Create the HTML Form

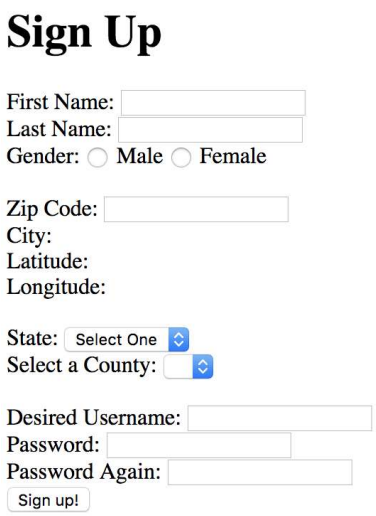
1.1 Create a new project

1.2 Open the **index.html** file and copy/paste the following HTML code

```
<!DOCTYPE html>
<html>
<head>
  <title> Sign Up Page </title>
</head>

<body>
   <h1> Sign Up </h1>
    First Name:<input type="text" name="fName"><br>
    Last Name: <input type="text" name="lName"><br>
    Gender:
       <input type="radio" name="gender" value="m">Male<br>
       <input type="radio" name="gender" value="f">Female<br>
     Zip Code:  <input type="text" id="zip" name="zip"><br>
     City:   <span id="city"></span> <br>
     Latitude:  <br>
     Longitude: <br><br>

     State:
     <select id="state" name="state">
          <option>Select One</option>
          <option value="ca">California</option>
          <option value="ny">New York</option>
           <option value="tx">Texas</option>
      </select> <br>

      Select a County: <select id="county"></select><br>

      Desired Username:
      <input type="text" id="username" ><br><br>

       Password: <input type="password">
<span id="suggestedPwd"></span> <br>
      Type Password Again: <input type="password"><br>

<span id="passwordError"></span>

       <input type="submit" value="Sign up!">
```

```
</body>
</html>
```

1.2 Preview the page. You should see something like the screenshot below:



As you can see, the form is using five different types of form elements:
text, radio buttons, select (dropdown menu), password, and a submit button.

Notes:

1) Observe that the <**select**> tag (the dropdown menu) does NOT use "input type" and each item is included within the **<option></option>** tags.

2) Observe that the "**radio**" elements MUST use the "**name**" attribute as part of the tag to be grouped. Otherwise, both of them would be marked as checked, and it wouldn't be possible to uncheck them. Other Form elements also have the "**name**" attribute but it's not really needed. This attribute is mainly used **when submitting data to a web server.**

3) The <input type="**submit**"> could have been been replaced by:
<button> Sign Up! **</button>**

4) The most common HTML Form elements are:

text	**radio**	**select**	**submit**
textarea	**checkbox**	**password**	**button**

For more information about HTML Form elements, refer to:
https://www.w3schools.com/html/html_form_elements.asp

Lesson 2. Get Data Based on Zip Code

Upon entering a Zip Code, the web page will be updated automatically to display the City, Latitude, and Longitude corresponding to the Zip Code entered. So, we'll need to add JavaScript code to create an event listener associated with the zip code text box.
Let us start by creating a file for JavaScript and embedding it within the **index.html** file.

2.1 Create a "**js**" folder and within it create a **script.js** file.

2.2 Embed the "**script.js**" file from the **index.html** file by adding the following line of code right before the closing </body> tag

```
39    <script src="js/script.js"></script>
40
41  </body>
```

2.3 Add an event listener to the zip code text box.
Observe that the zip code text box already has an id. We'll use it to add the event listener.

```
18      Zip Code:   <input type="text" id="zip" na
19      City:       <span id="city"></span><br>
```

Type the following code to add an event listener in the **script.js** file:

js/script.js

```
1  //event listeners
2  document.querySelector("#zip").addEventListener("change",displayCity);
3
4
5  //functions
6
7  //Displaying city from Web API after entering a zip code
8  function displayCity() {
9
10   alert(document.querySelector("#zip").value);
11
12 }
```

Test the code. After typing any zip code and clicking on any place within the page, you should see an Alert window that displays the value entered.

Why do you think we are using the "**change**" event instead of "**click**"?

Try changing "**change**" by "**click**" for the event listener, test it, and you'll find out!

We'll use the following Web API to get the city information based on the zip code entered. Notice that the output of the Web API uses JSON format (JS objects).
https://csumb.space/api/cityInfoAPI.php?zip=93955

Feel free to test it by replacing "93955" with any other zip code.

2.4 Update the displayCity() function. Comment out the alert() line and add the following code which uses the "fetch" API to retrieve data from an external Web API to get data about the zip code entered.

```
7  //Displaying city from Web API after entering a zip code
8  async function displayCity(){
9    let zipCode = document.querySelector("#zip").value;
10   //console.log(zipCode);
11   let url = `https://csumb.space/api/cityInfoAPI.php?zip=${zipCode}`;
12   let response = await fetch(url);
13   let data = await response.json();
14   console.log(data);
15 }
16
```

Observe that in **line 8**, we have added the keyword "**async**" right before **function**
The keyword "**async**" must be used in any function that uses the keyword "**await**", which is used in lines 12 and 13. **Await** and **Async** are keywords used to deal with the asynchronous behavior of JavaScript when getting remote data.

In **line 9**, we're assigning the zip code entered by the user to the "**zipCode**" variable.

In **line 11**, we're using template literals to concatenate the url to the external Web API and the zipCode variable. Observe that template literals use the backtick character (the inverted single quote is the key usually located to the left of number 1).

Here is the actual line of code:

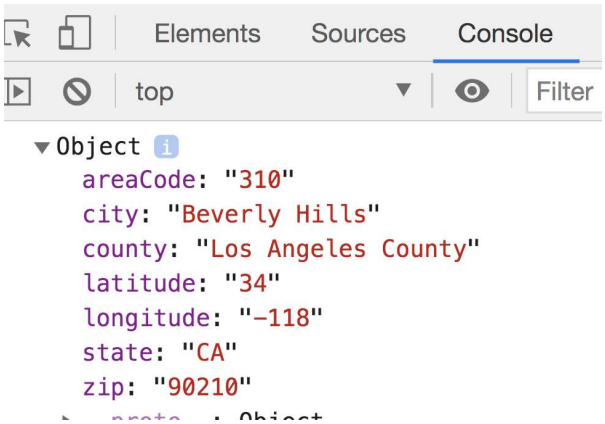
```
let url = `https://csumb.space/api/cityInfoAPI.php?zip=${zipCode}`;
```

In **line 12**, we're using the **fetch** API to retrieve data from the url and we're assigning the data to the **response** variable. Given that retrieving external data takes an unknown amount of time, we are using the "**await**" keyword to prevent JavaScript from skipping this line and moving on without getting the data.

In **line 13**, we're converting the data stored in the **response** variable to JSON format (JS Object Notation) to be able to access the values of any property.

In **line 14**, we're displaying the data retrieved from the web API in the console.

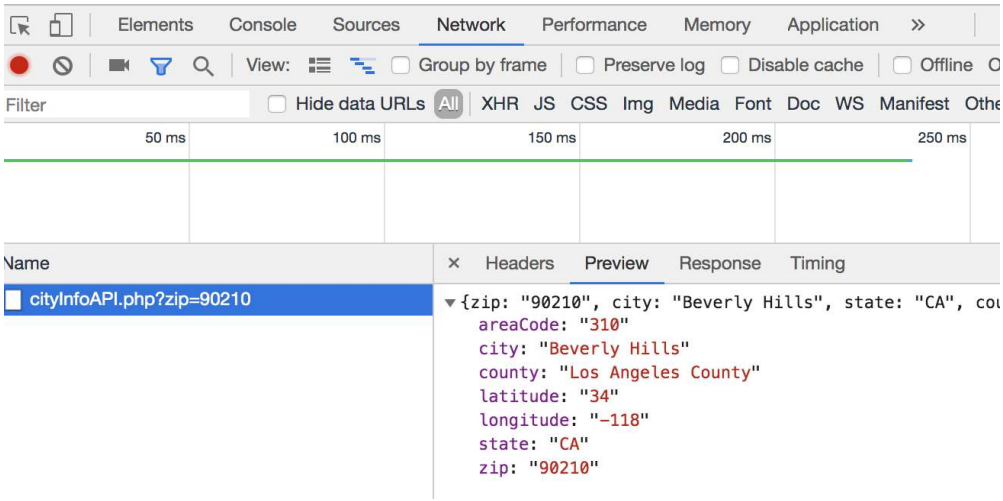
2.5 Save and test the file. Reload the page for the new code to take effect. Open the "**Inspect Tool**" in your browser to access the console.



Upon entering a zip code and clicking on any part of the page you should see the data retrieved from the Web API in the console.

Observe that a value of "**false**" is displayed in the console when entering a non-existing zip code such as "77777"

2.6 Inspect the data from the Web API. Most browsers provide a feature to see the data that is sent from an API. From the "Inspect Tool", click on the "**Network**" tab. Go back to the **index.html** file and type a new zip code. You should see the API endpoint listed in the Network tab. If you click on the endpoint, you'll see all of the information sent by the API.



2.6 Display the city name next to the “City:” label.

We already have a tag with an id in line 19 of the **index.html** file:

```
18      Zip Code:   <input type="text" id="zip" nam
19      City:       <span id="city"></span><br>
20      Latitude:   <br>
```

. Let’s display the city name within the span tag using JavaScript in the **script.js** file:

```
13    let data = await response.json();
14    //console.log(data);
15    document.querySelector("#city").innerHTML = data.city;
16  }
17
```

Save it and try it. Use different zip codes. The corresponding city should be displayed properly on the page next to the “City” label.

It’s Your Turn!

Display the Latitude and Longitude values from the API.

You’ll need to add an HTML tag next to their labels and include the “id” attribute to access them using JavaScript, something like:

```
Latitude: <span id="latitude"></span>
```

Lesson 3: Populate County Menu

Currently, the “Select County” menu is empty. This menu should be updated based on the State selected. We’ll use the following Web API to get the list of counties per state: https://csumb.space/api/countyListAPI.php?state=ca

Observe that the Web API is expecting the two-letter abbreviation of the state as a parameter instead of the full name. If you use the full name, nothing is returned.

For that reason, the “State” dropdown menu is already using the two-letter abbreviation as the values for each state:

```
<select id="state" name="state">
        <option value="">Select One</option>
        <option value="ca">California</option>
        <option value="ny">New York</option>
        <option value="tx">Texas</option>
</select><br />
```

Click on the API link and let us analyze the output of the Web API:

[{"county":"Alameda County"},{"county":"Alpine County"},{"county":"Amador County"},{
Costa County"},{"county":"Del Norte County"},{"county":"El Dorado County"},{"county":"
{"county":"Inyo County"},{"county":"Kern County"},{"county":"Kings County"},{"county"

You’ll notice that the output starts with a square bracket [which means that we’re getting an **array** of objects. Each object has a “county” property.

3.1 Let’s add an event to the State dropdown menu. Upon selecting a state, the event will be triggered. The Select element already has an id with the value of “state” (see screenshot above). We’ll use this id to add our second event listener in the **script.js** file:

```
1  //event listeners
2  document.querySelector("#zip").addEventListener("change",displayCity);
3  document.querySelector("#state").addEventListener("change",displayCounties);
4
```

3.2 Implement the “displayCounties()” function. Write the code below.

```
18  //Displaying counties from Web API based on the two-letter
    abbreviation of a state
19  async function displayCounties(){
20    let state = document.querySelector("#state").value;
21    let url = `https://csumb.space/api/countyListAPI.php?state=${state}`;
22    let response = await fetch(url);
23    let data = await response.json();
24    let countyList = document.querySelector("#county");
25    for (let i=0; i < data.length; i++) {
26       countyList.innerHTML += `<option> ${data[i].county} </option>`;
27    }
28  }
```

Notice that we’re declaring the function as “**async**” (line 19) because it uses the “fetch” API.

In line 25, we’re looping through all “county” objects retrieved from the API and displaying them within the “county” dropdown menu, using the <option> tags.

Observe that in line 26 we are using **+=** instead of just **=**
We need to use **+=** because the counties are being appended. Try removing the + symbol to see what happens. You’ll notice that only the last county in the list is displayed.

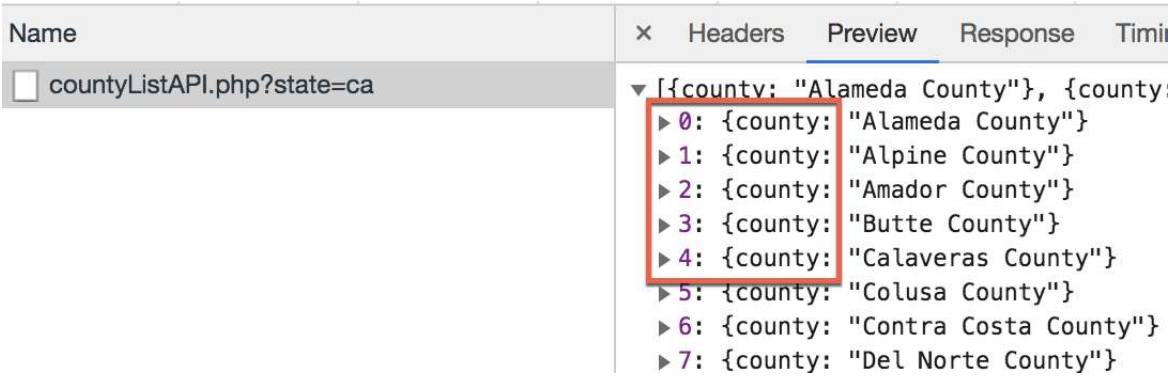
Reload your file and try selecting a state. You should see the list of all counties for that specific state within the “County” dropdown menu. However, try selecting a new state and then check the “County” dropdown menu. What is happening? The list of counties keeps increasing as a new state is selected. This is due to the use of **+=** in line 26.

3.3 Reset the “County” dropdown menu. To reset the County dropdown menu and avoid displaying counties from other states, add the following line of code before the for loop.

```
let countyList = document.querySelector("#county");
countyList.innerHTML = "<option> Select County </option>";
for (let i=0; i < data.length; i++) {
    countyList.innerHTML += `<option> ${data[i].county} </option>`;
}
```

Notice that **innerHTML =** overrides previous content within the HTML element.

Reload the page in the browser and test it. Be patient the first time you select a state, it might take a few seconds to populate the list of counties. If the list of counties is not updated, open the “Inspect” tool to debug your code. You could check whether the API is returning data by using the “Network” tab of the “Inspect Tool”. It should return an array of objects:



Note:

There are several ways to loop through an array in JavaScript. We are using the regular ‘for loop’ since most likely you’re already familiar with it. However, a most efficient method is to use the newer **for…of** loop (more info here)

```
for (let i of data) {
  countyList.innerHTML += `<option>${i.county}</option>`;
}
```

Lesson 4: Check Username Availability

In this section, we'll check whether the username entered is already taken or is available. The corresponding message will be displayed next to the username text box.
If possible, duplicate the current "index.html" file and try to implement this functionality by yourself, referring to the previous lessons. Then compare your code with the code below.

The API endpoint we'll use is:
https://csumb.space/api/usernamesAPI.php?username=eeny

The usernames already taken are: eeny, meeny, miny, maria, and john.

4.1 Let's start by adding an HTML element as a placeholder to display an error message if the username is already taken. We didn't have to do this step as part of the Zip code event because we already had HTML elements to display the city. We also had an HTML element to display the list of counties. But the current code doesn't have any HTML element to display an error message next to the username textbox. Let's add a element with "usernameError" as the id.

```
33      Desired Username: <input type="text" id="username" name="username"><br>
34                        <span id="usernameError"></span><br>
35      Password:         <input type="password" id="password" name="password"><
```

4.2 Add an event listener to the username text box. Upon typing a username, the event will be triggered. The username textbox already has an id ("username"). We'll use this id to add the event listener.

```
1  //event listeners
2  document.querySelector("#zip").addEventListener("change",displayCity);
3  document.querySelector("#state").addEventListener("change",displayCounties);
4  document.querySelector("#username").addEventListener("change",checkUsername);
5
```

4.3 Implement the **checkUsername()** function.

```
32 // checking whether the username is available
33 async function checkUsername() {
34     let username = document.querySelector("#username").value;
35     let url = `https://csumb.space/api/usernamesAPI.php?username=${username}`;
36     let response = await fetch(url);
37     let data = await response.json();
38     let usernameError = document.querySelector("#usernameError")
39     if (data.available) {
40         usernameError.innerHTML = " Username available!";
41         usernameError.style.color = "green";
42     }
43     else {
44         usernameError.innerHTML = " Username taken";
45         usernameError.style.color = "red";
46     }
47 }
```

Access the API endpoint directly and notice that returns either
`{"available":false}` or `{"available":true}`

In line 39 we're using the condition `if(data.available)` to display a message indicating whether the username is available or not.

Reload the page, and test it using several usernames.
Debug your code using the Inspect tool.

Lesson 5. Submit the Form

Currently, nothing is happening when clicking on the “Sign Up” button. If this was a real sign up page, the data should be submitted to a web server, added to a database, and then a new page should show up informing users that their account was created.
We’ll skip processing the data in the web server, we’ll just submit it to the server and display a new page.

5.1 Add the **<form>** tag. When form data needs to be submitted to a web server or you need to display a new page after the submission, then all form elements must be included within the **<form></form>** tags.

```
12
13      <form>
14          First Name: <input type="text"
15          Last Name:  <input type="text"
16          Gender:     <input type="radio"
:
39          <input type="submit" value="Sign up!">
40
41      </form>
```

Save the file, reload the page, fill out the form, and click on the “Sign up!” button… pay attention to the URL. By adding the <form> tags, the form is submitted to the web server and **the page is reloaded**. A question mark with the list of parameters and corresponding values is displayed in the URL.

Fill out the Signup form and submit it… Do you notice anything wrong with the URL?

Yes! All the information you typed, **including the password**, is displayed in the URL! This can be avoided by using the “method” attribute with the value of “POST” in the form tag: <form method=”POST”>, however, **let’s not use it yet** in this lab. We’ll learn more about it in upcoming labs.

Important!

When using the **<form></form>** tags the form data is submitted to the server and the page is reloaded. This means that you won't be able to see anything that you might be displaying on the page using JavaScript.

If you did the "Guess a Number" lab, you can try surrounding the <input> elements with the <form> tags and then notice how the page is reloaded when clicking on a button, without letting you see any feedback.

5.3 Specify a different target file when submitting the form. By default, the same page is reloaded when submitting the form. However, you can specify a different URL. Let's send the users to another file called "**welcome.html**" when submitting the form by adding the **action** attribute within the form tag:

```
13   <form action="welcome.html">
14     First Name: <input type="text"><br>
15     Last Name: <input type="text"><br>
```

If you try submitting the form right now, it will show a "Page not found" error, since the **welcome.html** file doesn't exist.

Note: If this was a real sign-up page, the action would be the URL of a backend program that would create a new record in the database.

5.4 Create a "**welcome.html**" file using the HTML template and display any message, something like: "Congrats! Your account was created!"

Save and try submitting the form. You should be able to see the welcome message

Lesson 6. Validate Form Data

One of the most common uses of JavaScript is to validate form data. Data shouldn't be submitted to the web server if it is invalid or inaccurate. For instance, if you enter a username that exists in the database (e.g., eeny, meeny, miny), an error message will be displayed but nothing will prevent the form from being submitted.

In this section, we'll use JavaScript to validate that the username is available and that the password entered is the same as "password again", otherwise, the form won't get submitted (the content of the **welcome.html** file won't be displayed).

6.1 Add an id to the <form> tag. We'll use this id to add an "on submit" event listener to it.

```
<form id="signupForm" action="welcome.html">
  First Name: <input type="text"><br>
```

Note: The order of the tag attributes does NOT matter…. But it's helpful to follow a convention (e.g. ordering them alphabetically, or always starting with the id, etc.)

6.2 Add the " **submit**" event listener to the form:

```
1  //event listeners
2  document.querySelector("#zip").addEventListener("change",displayCity);
3  document.querySelector("#state").addEventListener("change",displayCounties);
4  document.querySelector("#username").addEventListener("change",checkUsername);
5  document.querySelector("#signupForm").addEventListener("submit", function(event) {
6    validateForm(event);
7  });
8
```

Notice that the event listener is a little bit different than the others. The functions used in the previous event listeners didn't need to receive any parameter. But the **validateForm()** function that we'll be implementing needs to receive a special parameter that includes all of the information associated with the actual event. The next step explains why we need it.

6.3 Create the **validateForm()** function with the code below.

```
50  //Validating form data
51  function validateForm(e){
52    e.preventDefault();
53  }
```

Reload the page and click on the button. The “welcome.html” page shouldn’t be displayed. Try commenting out line 52 and test it again, you’ll notice that e.preventDefault() is preventing the submission of the form.

> When any event is triggered, the first parameter passed in the function includes all the information and methods about the **event.** One of these methods is “**preventDefault()**” which prevents executing the default functionality of the event. In the case of the “submit” event, the default functionality is the form submission.

6.4 Modify the **validateForm** function to check that the username is not left blank by adding the following code:

```
50  //Validating form data
51  function validateForm(e){
52    let isValid = true;
53    let username = document.querySelector("#username").value;
54    if (username.length == 0) {
55      document.querySelector("#usernameError").innerHTML = "Username Required!";
56      isValid = false;
57    }
58
59    if (!isValid) {
60      e.preventDefault();
61    }
62  }
```

Important!

JavaScript validation can be bypassed by users.
Its main goal is to warn users about an invalid input as fast as possible, instead of having to wait to get the error message from the web server.
However, the actual validation must be done on the web server as well, to ensure no invalid data has been submitted.

It's Your Turn!

1) Validate that the password has at least 6 characters
2) Validate that the values of password and "retype password" are the same