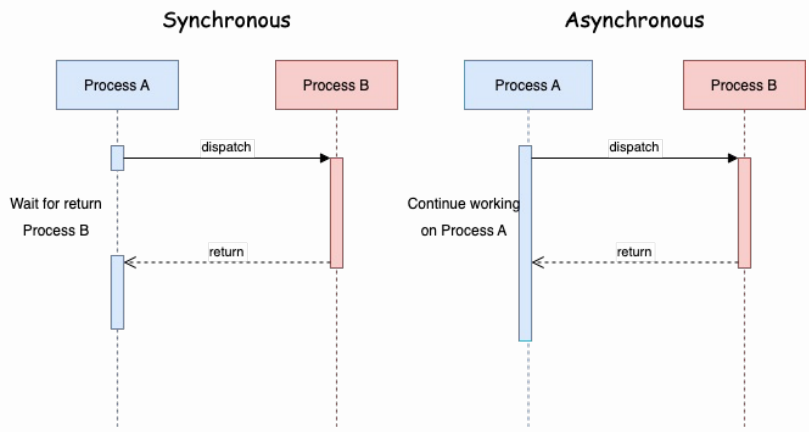
Synchronous & Asynchronous JavaScript



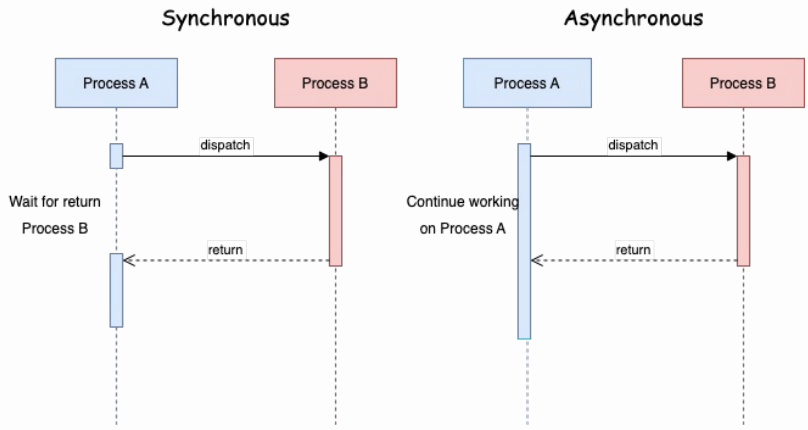
Goals
By the end of this lesson you will:
- Understand what Synchronous means
- Understand what Asynchronous means
- Build Asynchronous functions with JavaScript

Introduction

This lesson explains the difference between Synchronous and Asynchronous JavaScript and how to build these functions in JavaScript. The topics covered will help you understand how JavaScript works by way of Synchronous or Asynchronous loading.

JavaScript is a single-threaded synchronous language, which means that the code is executed in order, one piece at a time. However, in some situations, JavaScript may appear to be asynchronous. This is made possible through the use of functions that allow the code to be executed out of order, at a different time than when it was dispatched. This change in the order of execution is known as asynchronous execution.

What is Synchronous Execution?

> Synchronous execution in JavaScript refers to events that occur sequentially, one after another. It's like doing things in a straight line, with one thing happening at a time. This means that the code is executed in a single-threaded manner.
>
> 
>
>
> The figure above shows an example of synchronous execution, where Process A dispatches (calls) Process B. During this time, Process A pauses and waits until Process B runs to completion. Once Process B is complete, it returns to Process A, and Process A resumes its execution. This is because JavaScript is single-threaded / synchronous, which means that only one process can run at a time.

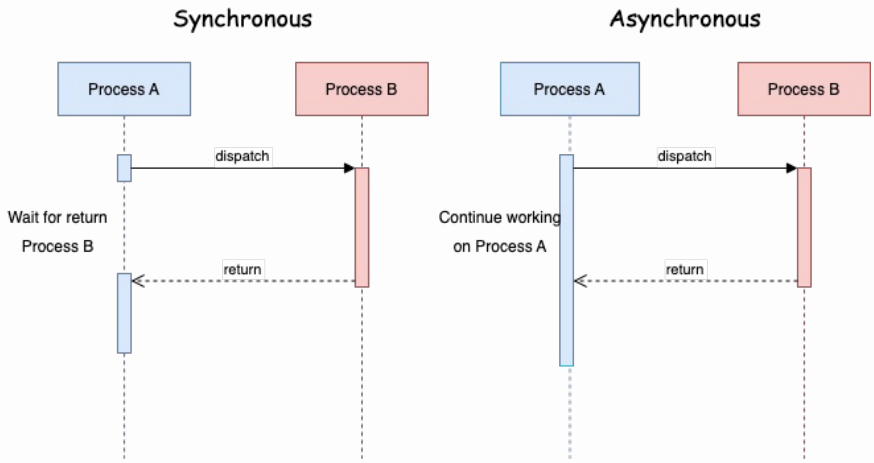
In this example, you will see how synchronous JavaScript works in practice:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <title>Synchronous</title>
</head>
<body>
   <div id="message"></div>
   <script>
       var msg = document.getElementById("message");

       function fun1() {
           msg.innerHTML += "<p>fun1 started :)</p>";
           fun2();
           msg.innerHTML += "<p>fun1 ended :)</p>";
       }
       function fun2() {
           msg.innerHTML += "<p>fun2 started :)</p>";
           msg.innerHTML += "<p>fun2 ended :)</p>";
       }
       fun1();
   </script>
</body>
</html>
```

Output

```
fun1 started :)
fun2 started :)
fun2 ended :)
fun1 ended :)
```

When dealing with a large amount of code, synchronous loading of JavaScript files can be time consuming. Since long load times are undesirable, we solve this problem with asynchronous loading.
>
What is Asynchronous Loading?
With asynchronous loading, events take place independent of the main process allowing for the execution of several processes in parallel.



>

Process A dispatches Process B. Process A continues to execute in parallel with Process B to completion without depending on one another. In this way, total execution time is reduced and a faster load time is achieved.

>

How to create an Asynchronous function in JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <title>Asynchronous</title>
</head>
<body>
   <div id="text"></div>
   <script>
       var msg = document.getElementById("text");

       function fun1() {
           setTimeout(function () {
              msg.innerHTML += "<p>fun1 started :)</p>";
              msg.innerHTML += "<p>fun1 ended :)</p>";
           },100);
       }
      function fun2() {
           msg.innerHTML += "<p>fun2 started :)</p>";
           fun1();
           msg.innerHTML += "<p>fun2 ended :)</p>";
       }

       fun2();
   </script>
</body>
</html>
```

> **Output**

```
fun2 started :)
fun2 ended :)
fun1 started :)
fun1 ended :)
```

Step by step of previous code:
1. The code calls function fun2()
2. The function fun2() shows text “fun2 started :)”

3. The function fun2() calls function fun1()
4. The function fun1() waits for 100 ms before outputting its text, but function fun2() does not wait for fun1() to be finished before moving on; fun2() jumps on the next statement immediately and shows the text “fun2 ended :)”
5. After of 100 ms, function fun1() executes its next statement showing text “fun1 started :)”
6. Function fun1() then shows text “fun1 ended :)”
>
Because the two functions can run in parallel, asynchronous JavaScript loads more quickly than synchronous JavaScript.
>
In the previous code example, we used the **setTimeout(Function, Time in milliseconds)** function to delay the execution of fun1() by 100ms. This function allows you to pause the execution of a piece of code for the amount of time specified in the function argument before resuming its execution.
>
Additionally, we can use the **setInterval(Function, Time in milliseconds)** function to repeatedly execute a piece of code at a specified time interval. This function is useful when we need to run a certain piece of code repeatedly, such as updating the user interface or making periodic requests to a server.
>
Let’s see it in action:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <title>Asynchronous</title>
</head>
<body>
   <div id="text"></div>
   <script>
       var msg = document.getElementById("text");

       function fun1() {
           setInterval(function () {
              msg.innerHTML += "<p>fun1 started :)</p>";
              msg.innerHTML += "<p>fun1 ended :)</p>";
           },60000);
       }
      function fun2() {
           msg.innerHTML += "<p>fun2 started :)</p>";
           fun1();
           msg.innerHTML += "<p>fun2 ended :)</p>";
       }

       fun2();
   </script>
</body>
</html>
```

Output

```
fun2 started :)
fun2 ended :)
fun1 started :)
fun1 ended :)
fun1 started :)
fun1 ended :)
…
fun1 started :)
fun1 ended :)
```

> Every 60 seconds, the function fun1() will be called.

Warning: the function **setInterval** could run infinitely without stopping. Be sure to tell it when to stop.

How to create an Asynchronous function in JavaScript part 2

Exercise 2:

Build a clock using the function setInterval()

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Clock</title>
</head>
<body>
  <div id="text"></div>
  <script>
       var msg = document.getElementById("text");
      function clock() {
          //code here
      }
      clock();
  </script>
</body>
</html>
```

Tip: Use the Date object, for more information here.

We can implement asynchronous functions using the `async`, `await`, and `Promise` keywords. The **async** keyword defines an asynchronous function, which allows us to write asynchronous code that looks and behaves like synchronous code.

By using the **async** keyword, we can make a function return a `Promise`, which can be resolved or rejected at a later time. The `await` keyword is used to wait for the completion of a `Promise`, which allows us to write code that looks like synchronous code but behaves asynchronously.

Syntax

```
async function FunctionName(){
    ...
}
```

await: The “async” function contains “await” which pauses execution of an “async” function. “await” is only valid inside the “async” function.

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

<head>
   <meta charset="utf-8" />
   <title>Asynchronous function using "Async"</title>
</head>

<body>
   <div id="text"></div>

   <script>
       var msg = document.getElementById("text");
       function fun1() {
           return new Promise(function (resolve, reject) {
               setTimeout(function () {
                   msg.innerHTML += "<p>fun1 started :)</p>";
                   msg.innerHTML += "<p>fun1 ended :)</p>";
                   resolve();
               }, 100);
           });
       }

       async function fun2() {
           msg.innerHTML += "<p>fun2 started :)</p>";
           await fun1();
           msg.innerHTML += "<p>fun2 ended :)</p>";
       }

       fun2();
   </script>

</body>

</html>
```

Output

```
fun2 started :)
fun1 started :)
fun1 ended :)
fun2 ended :)
```

In the above example, the engine waits for fun1() to finish its execution before executing the next line. The “await” stops the execution of that code segment until a Promise is received.
>
Why use `await` if `async` is faster?

`async` allows for faster code execution, but it can cause issues when order or dependencies matter. `await` can pause execution until a process completes to ensure correct code execution and avoid errors.

Conclusion & Takeaways

> - Asynchronous JavaScript is a powerful functionality that can be incredibly helpful in your journey as a web developer. By allowing for multiple processes to run simultaneously, asynchronous code can greatly improve the performance and functionality of your applications.
> - However, it's important to be careful when using the `setInterval` function, which can cause your code to run indefinitely without stopping. To avoid this issue, it's important to specify when the interval should stop running, either by using `clearInterval` or by implementing a stop condition within your code.

By using asynchronous JavaScript correctly and being mindful of potential issues, you can take full advantage of its benefits and build high-quality, efficient web applications.
>
Attribution
- Asynchronous JavaScript . (n.d.). https://www.w3schools.com/js/js_asynchronous.asp
>
- GeeksforGeeks. (2021, December 17). How to create an Asynchronous function in Javascript? https://www.geeksforgeeks.org/how-to-create-an-asynchronous-function-in-javascript/

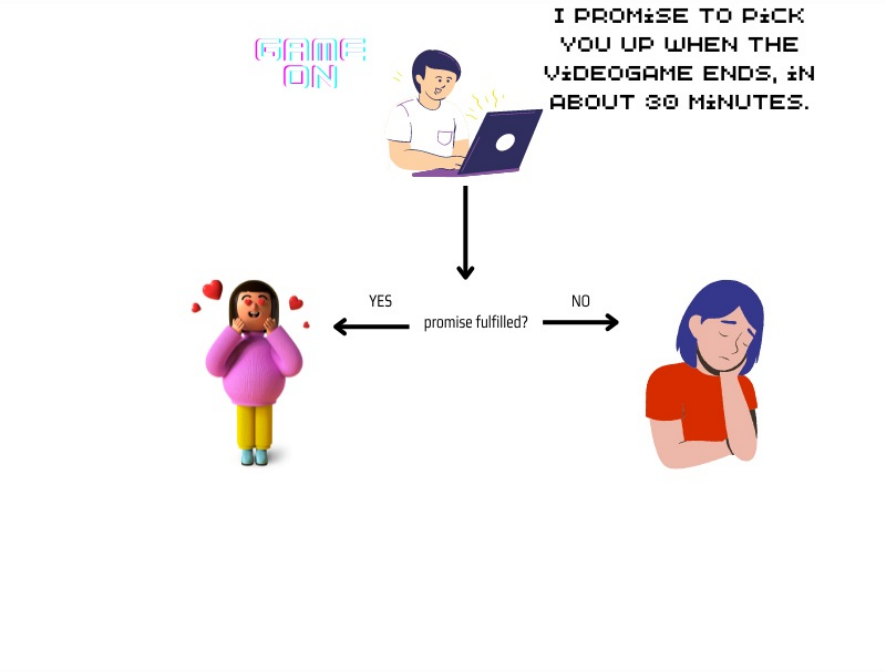
Javascript Promises

Goals
By the end of this lesson you will:
- Understand Javascript Promises
- Know how use Javascript Promises
- Know the benefits of Javascript Promises

Introduction

This lesson will help you understand JavaScript promises and its benefits. Promises in JavaScript function as they do in real life; you get a rejection (a negative result) until the promise is fulfilled. After the promise is fulfilled you get a resolve (a positive result). Therefore, if you make a promise you must try to fulfill it or else, you will only get negative results ;)

Promises



Promises
Technically, Promises will help to handle asynchronous operations easily. It can handle multiple asynchronous operations and error handling is easier to manage. Thanks to this, promises are perfect for handling multiple callbacks at the same time, thus avoiding undesired callbacks.
>
Benefits of Promises:
→ Handling of asynchronous operations
→ Code readability
→ Error handling
→ Flow of control definition in asynchronous logic
>
States promises:
> fulfilled: the promise succeeded
> rejected: the promise failed
> pending: Promise is neither fulfilled nor rejected
> settled: Promise is either fulfilled or rejected
>
Syntax

```
var promise = new Promise(function(resolve, reject){
     //do something
});
```

Promise receives an argument as a callback function and at the same time takes two arguments: resolve and reject. The code inside the callback function is performed and returns resolve if all is ok, otherwise it returns reject.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <title>Javascript Promises</title>
</head>
<body>
   <div id="text"></div>

   <script>
       var msg = document.getElementById("text");
       var promise = new Promise(function (resolve, reject) {
           const continuePlaying = "Y";
           if (continuePlaying === 'N') {
               resolve('Your friend is happy!');
           } else {
               reject('Your friend is boring!');
           }
       });

       promise.
           then(function (successMessage) {
               msg.innerHTML += "<p>" + successMessage + "</p>";
               console.log(successMessage);
           }).
           catch(function (errorMessage) {
               msg.innerHTML += "<p>" + errorMessage + "</p>";
               console.log(errorMessage);
           });
   </script>
</body>
</html>
```

Output

```
Your friend is boring!
```

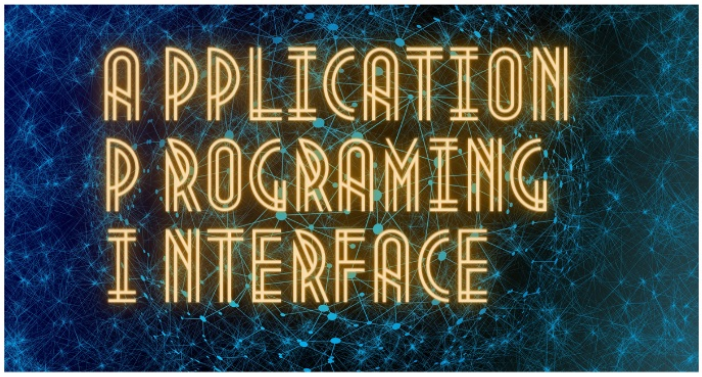
Promises Consumers

> Promises are taken by functions then and catch
>
> **Then()**
>
> Then is performed when the promise is: resolve or reject. This takes two functions as parameters, the first one (then) is executed when the promise ends as resolve and the second one (catch) is executed when the promise ends as reject and an error is received.
>
> **Catch()**
>
> Catch is performed when the promise ends as reject or experiences an execution error. Catch takes one function as a parameter, which is used to handle the error or rejection.

Conclusion & Takeaways

> - When working with JavaScript promises, it's important to be mindful of the results that they can produce, as these can be either positive or negative. This means that when making a promise, it's important to do so with care and to make every effort to fulfill it.
> - Despite these challenges, promises can be a powerful tool for handling multiple operations within your application. By allowing you to manage complex, asynchronous code more effectively, promises can help improve the performance, reliability, and functionality of your web applications.
> ### Attribution
> - GeeksforGeeks. (2021a, December 6). JavaScript | Promises. https://www.geeksforgeeks.org/javascript-promises/

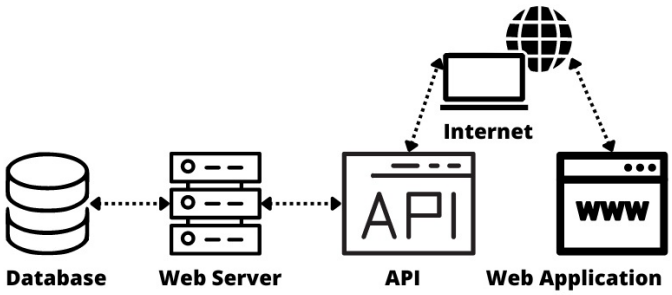
What is an API?



Goals
By the end of this lesson you will:
- Understand how an API works
- Know its different architectures
- Create your own API

Introduction

This lesson will teach you how to work with an API (Application Programming Interface), and its different types of architectures, and at the end of the lesson, you will be able to create your own API. An API allows you to execute code from a different server than where your frontend or backend is. This is of great benefit because it gives you access to several new functions and you will be able to use these in your code. APIs are used by big companies. Google, for example, has several APIs that you can use to get maps in your application and manipulate these according to your needs. Here you can get more information about the Google Maps API.



api diagram

Business Context

Most companies will include a map with a location pin on their contact page. This map can be made interactive using the Google maps API. Payment solutions on web stores also depend on APIs. For example, there is a PayPal Payments API.

What is an API?

> An API is a messenger that takes your requests, tells the system what you want to do, and gives the response back to you.
>
> Consider buying airline tickets. You put in the dates you're interested in for your flight, and you get back options for you to choose from.

Types of API

An API has a classification that defines its scope of use. The types of API are:
- **Private API**
A private API is of internal use within a company to connect internal applications
- **Public API**
A public API is available for public use and can be accessed by anyone. It may require some authorization and there may be an associated cost.
- **Partner API**
A partner API is accessible by an external agent who is a partner.
- **Composite API**
A composite API combines several types of API to build a more complex system.

How do APIs work?

In terms of client and server, the client sends a request to the server who gives a response.

There are four different ways an API can work:

- **SOAP**
  Simple Object Access Protocol: client and server exchange messages using XML.
- **REST**
  Representational State Transfer: this is for communication between different systems and it is the most popular API architecture currently used. Client and server exchange messages using a plain file. We will consider REST APIs in more detail later.
- **RPC**
  Remote Procedure Calls: the client executes a procedure on the server and the server responds with the output to the client.

- **WEBSOCKET**
  This supports two-way communication between client and server, the server can send requests to connected clients and clients can send requests to the server. This is different from the REST API where only the client can send requests to the server. The WEBSOCKET API is another modern API. It uses JSON format for communication.

REST API Part 1

> Representational State Transfer has the functions GET, PUT, DELETE and POST. The client can use these functions to access the server. Clients and servers exchange information using HTTP.
>
> REST API is stateless which means the server does not need to know anything about what state the client is in and vice versa. Client requests seem like URLs that you type in your browser.
>
> The implementation of the client and server are done independently. This is an important benefit because you can modify the code of the client without affecting the operation of the server.
>
> ### Communication between Client and Server

Making Requests
- A request generally consists of:
- an HTTP verb, which defines what kind of operation to perform
- a header, which allows the client to pass along information about the request
- a path to a resource
- an optional message body containing data
>
HTTP Verbs
- GET — retrieve a specific resource (by id) or a collection of resources
- POST — create a new resource
- PUT — update a specific resource (by id)
- DELETE — remove a specific resource by id
>

> ### Headers and Accept parameters

When the client sends a request in the header you should send a type of content that the client can receive from the server. This is called **Accept**; the goal is to ensure that the server sends data that the client can understand or process. The options for types of content are MIME Types, or Multipurpose Internet Mail Extensions, which you can read more about in the MDN Web Docs.

REST API Part 2

Types and commonly used subtypes:
>
- image — image/png, image/jpeg, image/gif
- audio — audio/wav, audio/mpeg
- video — video/mp4, video/ogg
- application — application/json, application/pdf, application/xml, application/octet-stream
- Text — text/html, text/plain, text/css
>
Example: A get request for id 4 in a **fellow** resource

```
GET /fellow/4
Accept: text/html, application/json,  text/plain
```

The Accept header field in this case is saying that the client will accept the content in text/html or application/json or text/plain.
>
> Paths

Requests must contain a path where the operation will be performed. For example:
`GET correlationone.com/fellows/4`
correlationone.com => It is the domain of the server
fellows => Accessing a fellows resource
4 => Identify the fellow
>
Paths should be designed to help the client know what is going on.

Sending Responses

Content Types
The server response sends a data payload to the client. This must include a content-type in the header with the goal of indicating what type of data is being sent. These content types are MIME Types, as seen previously.
>
Following a request, the server might send back content with the response header:

```
HTTP/1.1 200 (OK)
Content-Type: text/html
```

This would signify that the content requested is being returned in the response body with a content-type of text/html, which the client said it is able to accept.
>
In a later lesson we will see an API in operation.
>
Response Codes

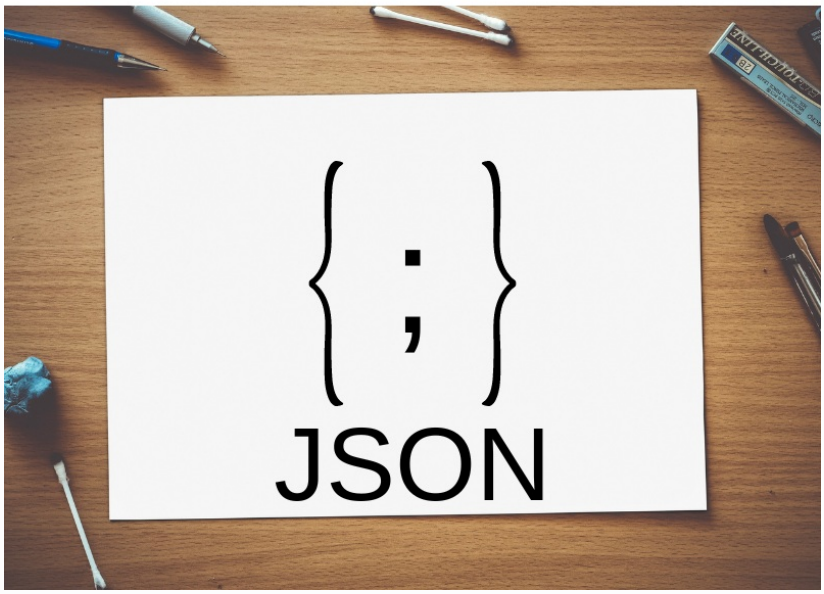
Status code	Meaning
200 (OK)	This is the standard response for successful HTTP requests.
201 (CREATED)	This is the standard response for an HTTP request that resulted in an item being successfully created.
204 (NO CONTENT)	This is the standard response for successful HTTP requests, where nothing is being returned in the response body.
400 (BAD REQUEST)	The request cannot be processed because of bad request syntax, excessive size, or another client error.
403 (FORBIDDEN)	The client does not have permission to access this resource.
404 (NOT FOUND)	The resource could not be found at this time. It is possible it was deleted, or does not exist yet.
500 (INTERNAL SERVER ERROR)	The generic answer for an unexpected failure if there is no more specific information available.

These codes are useful for knowing the status of an operation's success. The server responds to the client with some of these codes. As a developer, you do not need to know every status code, only the most common ones.
>
Every **HTTP Verb** has an expected status code that it should return to the server:
>
GET => return 200 (OK)
POST => return 201 (CREATED)
PUT => return 200 (OK)
DELETE => return 204 (NO CONTENT)
>
If the operation fails, the server will return the most specific status code possible corresponding to the problem that was encountered.

Conclusion & Takeaways

> - APIs are used to share data between server and client
> - Remember the types of request (HTTP verbs) GET, PUT, DELETE and POST each of which has its expected response code.
>
> ### Attribution
>
> - MuleSoft Videos. (2015b, June 19). What is an API? [Video]. YouTube. https://www.youtube.com/watch?v=s7wmiS2mSXY
> - Codecademy. (n.d.). What is REST? https://www.codecademy.com/article/what-is-rest
> - What is an API? - API Beginner's Guide - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/api/?nc1=h_ls

JSON format files



Goals
By the end of this lesson you will:
- Understand the usefulness of JSON files
- Know the structure of a JSON file
- Be able to build a JSON file

Introduction

JSON (JavaScript Object Notation) is an open standard file format supported by many modern programming languages for generation and parsing. JSON was based on Standard ECMA-262 3rd Edition—December 1999 which is a subset of JavaScript. Also, this file format is the most used for server to client communication.

What is a JSON file?

> JSON (JavaScript Object Notation) is a file format that is easily readable for a human and requires less formatting. It is used for sharing or transmitting data to store. The extension of this type of file is .JSON and it is supported by many modern programming languages.
>
> The media type used for JSON is application/json.
>
> ### JSON File Structure
>
> JSON is structured with key and value pairs. The key is unique and is a string surrounded by double quotation marks. The Value can be a Number, string (surrounded by double quotation marks), Array, Boolean, null and object. The key and value are separated by a colon(:) in the middle with the key on the left and the value on the right. Different key/value pairs are separated by a comma(,).

Examples by types values:
- Number
```
{"age" : 34}
```
- String
```
{"name" : "Joseph"}
```
- Array
```
{"fruits" : ["Apple","Banana","Strawberry"]}
```
- Boolean
```
{"Enable" : true}
```
- Null
```
{"genre" : null}
```
- Object
```
{"user" : {"name" : "Joseph", "age" : 34, "genre" : null}}
>
```

> ### Exercise 3
>
> Build a JSON file for the data below (hint: think of this as a user object).
> | Name | Id | Telephone number | Hobbies | Lessons |
> |———|———-|———-|———-|———|
> | Christopher | 7464839 | +1457938740 | Read, Sing, Dance | Name:HTML, ID:1; Name:Javascript, ID:2; Name:API; ID:3 |

Conclusion & Takeaways

- You should be mindful of the available resources of the computer; the transfer of data will be affected if the client browser has limited memory.
- JSON files are easily readable; remember that it has key and value pairs and its value types are: number, string, boolean, array, null and object.

Attribution

- GeeksforGeeks. (2021a, May 19). JSON | Data Types. https://www.geeksforgeeks.org/json-data-types/
- Chishti, M. A. (2020, April 12). JSON File Format - What is a JSON file? https://docs.fileformat.com/web/json/

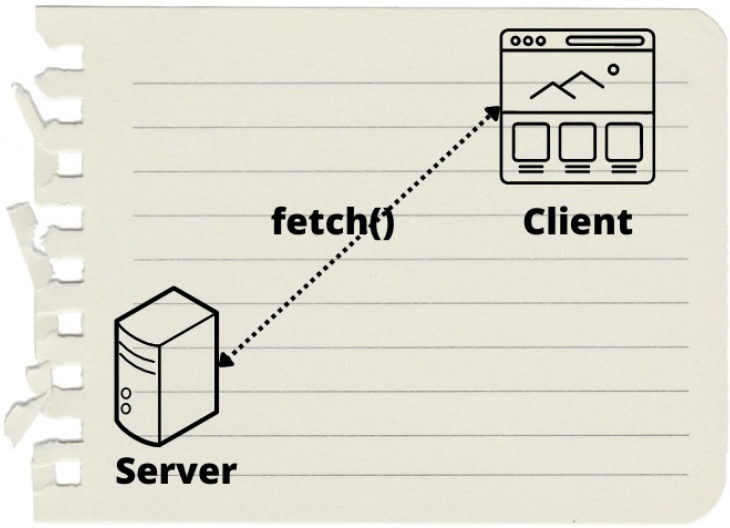
Accessing APIs with Fetch

Goals
By the end of this lesson you will:
- Know how to work with the fetch method
- Be able to get data using the fetch method
- Be able to execute a public free API

Introduction

The fetch() method easily connects the server to the client. You need only type the URL of the API and a second parameter for POST requests. With the fetch() method you can execute an API (if you have access) and begin to interact a little more with APIs to see an API in action. Also, the fetch() method allows you to get data and show it on your page.

fetch() Method in Javascript



fetch() Method in Javascript
The fetch() method is used to request data from a server for any type of API that returns the data in JSON or XML format.

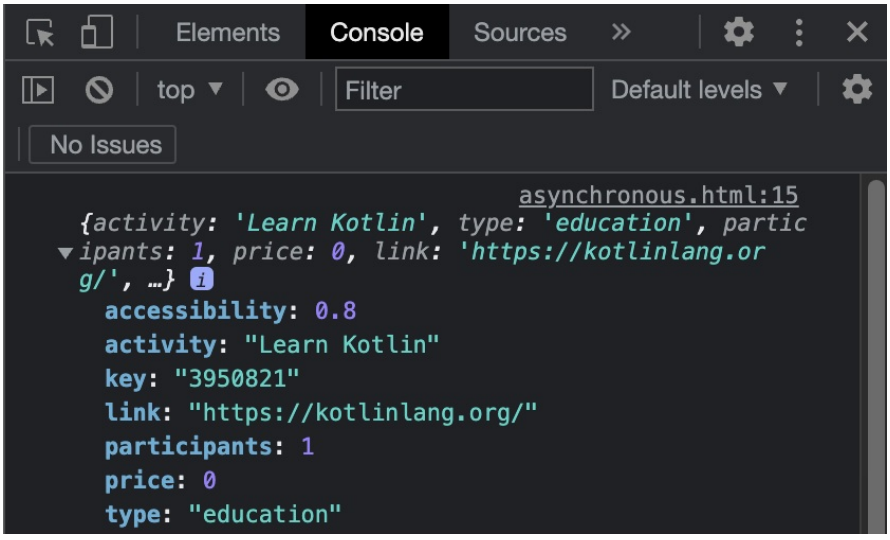
Syntax

```
fetch('url')
  .then(response => response.json())
  .then(data => console.log(data));
```

The fetch() method requires one parameter: the URL to request. It accepts two parameters and returns a promise. The second parameter is an array of properties.
>
For the next example we will use a free API found here.

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>C1 | JavaScript | fetch() Method</title>
</head>

<body>
    <script>
        let result =
fetch("https://www.boredapi.com/api/activity");
        result.then(res => res.json())
              .then(d => { console.log(d)});
    </script>
</body>

</html>
```

In the console, data in JSON will look like this:

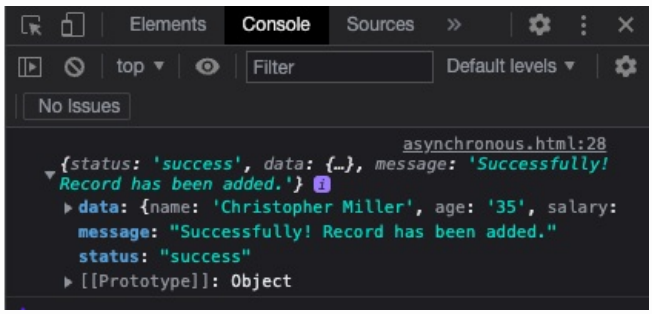


The return value is a promise whether it is resolved or not. The return data can be of the format JSON or XML. It can be an array of objects or simply a single object.

>

```
<!-- Fetching data with options: -->
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>C1 | JavaScript | fetch() Method with Options</title>
</head>

<body>
    <script>
        user = {
            "name": "Christopher Miller",
            "age": "35",
            "salary": "4321"
        };
        let options = {
            method: 'POST',
            headers: {
                'Content-Type':'application/json;charset=utf-8'
            },
            body: JSON.stringify(user)
        };
        // Rest API example
        let result =
fetch("https://dummy.restapiexample.com/api/v1/create",
options);
        result.then(res => res.json())
              .then(d => {console.log(d)});
    </script>
</body>

</html>
```

In the console, data in JSON will look like this:



json console image

Exercise 2:

Complete the code below using this API and embed the id “imageAPI” in the tag.
>
Tip: the API returns the url of an image in the “message” key in the JSON response.

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>C1 | JavaScript | fetch() Method Exercise</title>
</head>

<body>
    <script>
        //Completed code here
    </script>
    <img id="imageAPI" width="500" height="600">
</body>

</html>
```

Conclusion & Takeaways

- The fetch() method has two parameters, the URL of the API and a second parameter for options.
- The fetch() method is very easy to use, you can try to do several tests to see it in action, there are many free APIs on the web.

Attribution

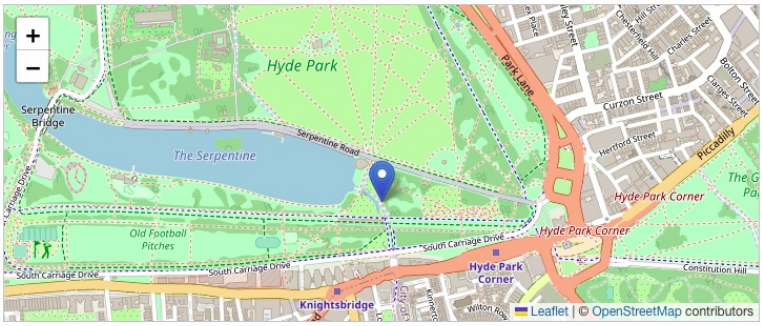
- Team, A. (2022, February 19). Free API – Huge List of Public APIs For Testing [No Key]. Apipheny. https://apipheny.io/free-api/
- GeeksforGeeks. (2022, October 19). JavaScript | fetch() Method. https://www.geeksforgeeks.org/javascript-fetch-method/
- GeeksforGeeks. (2021a, January 8). How to use JavaScript Fetch API to Get Data ? https://www.geeksforgeeks.org/how-to-use-javascript-fetch-api-to-get-data/?ref=rp

Leaflet

Goals
By the end of this lesson you will:
- Know how to work with Leaflet
- Implement a solution with maps using Leaflet
- Practice knowledge in Javascript

Introduction

Leaflet is a powerful tool for developing maps. The programming language used to develop Leaflet is JavaScript, whereby it is easy to learn and use. Leaflet is designed with simplicity, performance and usability in mind. It works efficiently across all major desktop and mobile platforms.



Leaflet map

Image from https://leafletjs.com/index.html

What is Leaflet?

> Leaflet is an open-source JavaScript library for interactive maps. It has all the mapping features most developers need. It is compatible with mobile platforms and all major desktops. It has a beautiful, easy to use and well-documented API.

In this link you can see all documentation of the APIs available in this library.
>
Preparing your page
1. Write the basic HTML code for a webpage or copy and paste in the following code in your editor:

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

<head>

    <title>Hello Leaflet</title>

</head>

<body>

    <div id="map" style="width: 600px; height: 400px;"></div>

</body>

</html>
```

2. Include Leaflet CSS file in the head section of your document:

```
<link rel="stylesheet"
href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css"
     integrity="sha256-
kLaT2GOSpHechhsozzB+flnD+zUyjE2LlfWPgU04xyI="
     crossorigin=""/>
```

3. Include Leaflet JavaScript file after Leaflet's CSS:

```
<script src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js"
     integrity="sha256-
WBkoXOwTeyKclOHuWtc+i2uENFpDZ9YPdf5Hf+D7ewM="
     crossorigin=""></script>
```

4. This div element with a certain id is where you want your map to be, so feel free to move it or add content around it. For now, we're just going to work with this map's functionality. In addition, note that we're writing everything right in the html, but you know how to create a separate javascript file for the code.

```
<div id="map" style="width: 600px; height: 400px;"></div>
```

Up until now, the page was blank.

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

<head>
    <title>Hello Leaflet</title>

    <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.9.3/dist/leaflet.css"
        integrity="sha256-
kLaT2GOSpHechhsozzB+flnD+zUyjE2LlfWPgU04xyI=" crossorigin="" />
    <script
src="https://unpkg.com/leaflet@1.9.3/dist/leaflet.js"
        integrity="sha256-
WBkoXOwTeyKclOHuWtc+i2uENFpDZ9YPdf5Hf+D7ewM=" crossorigin="">
</script>
</head>

<body>
    <div id="map" style="width: 600px; height: 400px;"></div>
</body>

</html>
```

Setting up the map

> Let's do our first map! Add the following code to the body of your HTML code, save and refresh your navigator.

```
<script>

        const map = L.map('map').setView([40.689, //latitude
            -74.044], //longitude
            13);    //zoom

        const tiles =
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
            maxZoom: 19,
            attribution: '&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
        }).addTo(map);


</script>
```

In the previous code we set up the coordinates of the "Statue of liberty" monument. We suggest that you use the page https://www.latlong.net/ for searching the latitude and longitude of a place.
>
Exercise 2:
Search the coordinates of Niagara falls, put in your code and refresh your navigator.

Answer:
It looks like this:

```
<script>

        const map = L.map('map').setView([43.096214, //latitude
            -79.037743], //longitude
            13);    //zoom

        const tiles =
L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
            maxZoom: 19,
            attribution: '&copy; <a
href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>'
        }).addTo(map);


</script>
```

As you have seen, the method tileLayer shows the map in the navigator. In this case it is an OpenStreetMap tile layer. More information on tileLayer can be found here.

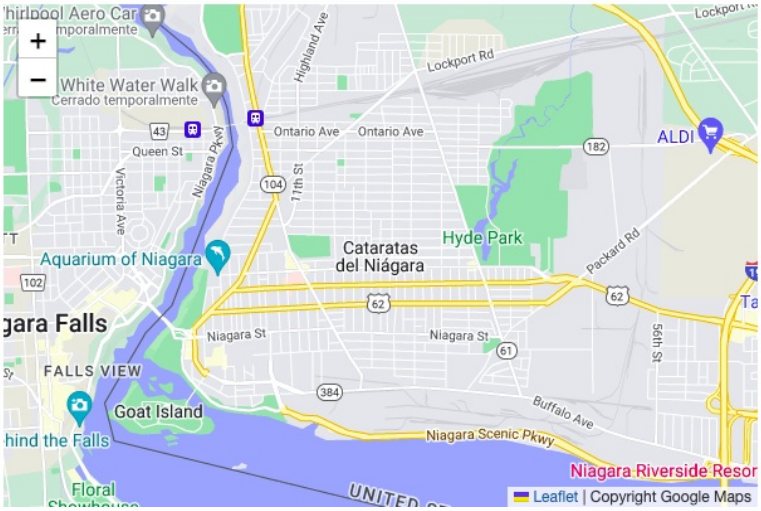
>

Leaflet is provider-agnostic, meaning that it doesn't enforce a particular choice of providers for tiles. In the next example we use Google Maps tiles; there are many providers available and you can choose the provider that best fits your needs.

>

```
<script>

        const map = L.map('map').setView([43.096214, //latitude
            -79.037743], //longitude
            13);    //zoom

        L.tileLayer('http://{s}.google.com/vt/lyrs=m&x={x}&y=
{y}&z={z}', {
            maxZoom: 20,
            subdomains: ['mt0', 'mt1', 'mt2', 'mt3'],
            attribution: 'Copyright Google Maps'
            }).addTo(map);

    </script>
```

It looks like this:

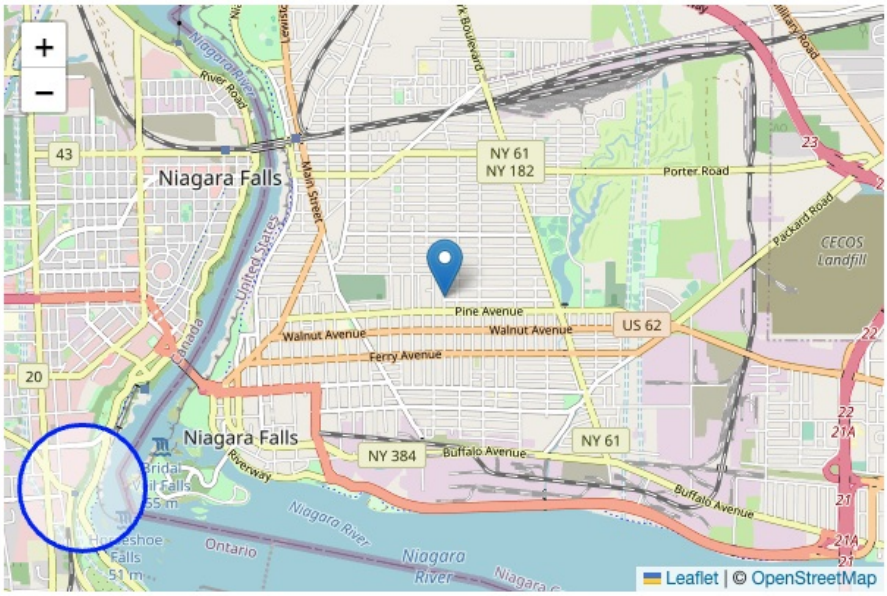


map image

Most tile providers such as OpenStreetMap and Google Maps require an attribution, please check the copyright notice of the tiles provider before publishing your page in production.

Markers, Circles, and Polygons

> Now, let's go to include makers, polylines, polygons, circles and popups:
> 1. Add marker to map:
> `var marker = L.marker([43.096214, -79.037743]).addTo(map);`
> 2. Add a circle to map: in this case you should indicate the radius of the circle in meter
> `var circle = L.circle([43.08, -79.08], {radius: 600}).addTo(map);`
>
> You can add more options for changing the design of the circle, polyline or polygon. For more information visit this link.
>
> With this code, your map should look like the one below:



Exercise 3:
Add a polygon with the following coordinates and options:

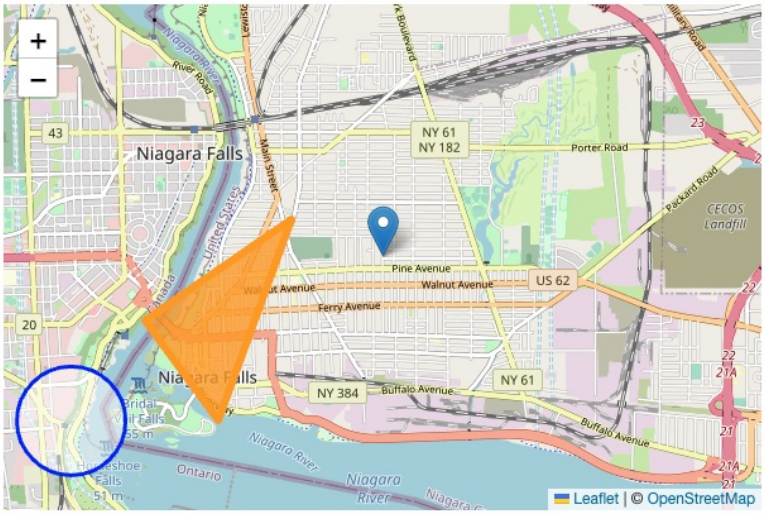
Tip: You can refer to Leaflet's polygon documentation.

Coordinates:
[43.09, -79.07]
[43.08, -79.06]
[43.1, -79.05]

Options:
Color: orange
fillOpacity: 0.8

weight: 6

Your map should now look like the one below.



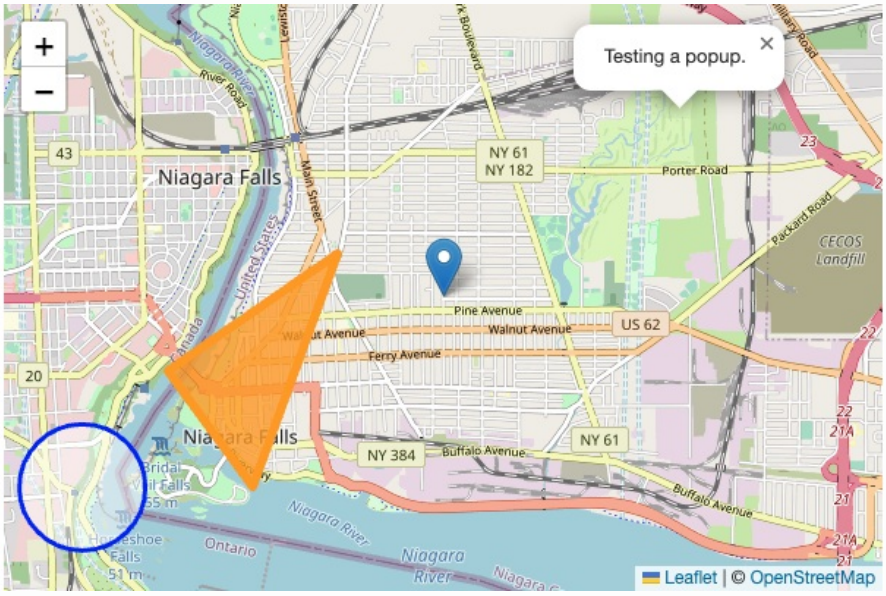
map with circle and triangle

Working with popups

> Sometimes you need to attach information in your map. With the next code you can add easily a popup:

```
var popup = L.popup()
            .setLatLng([43.1123, -79.01])
            .setContent("Testing a popup.")
            .openOn(map);
```

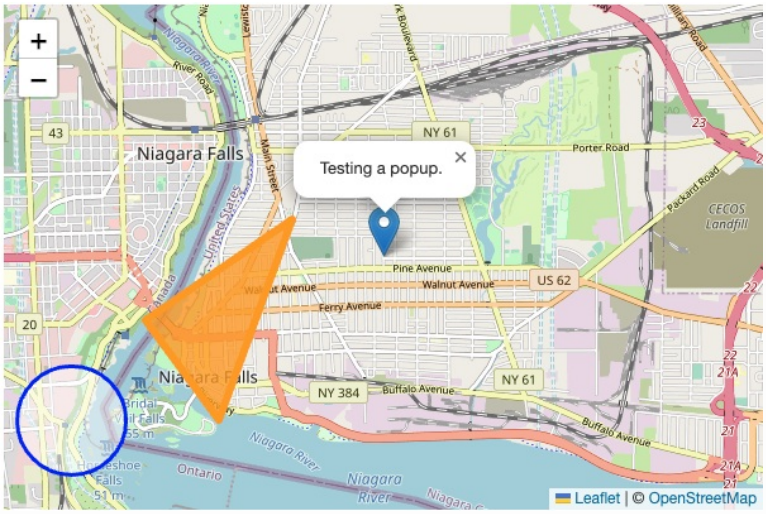
It looks like this:



We also can add a popup to our objects as makers, polygons, circles, etc. Add the following code to your page.

```
marker.bindPopup("Testing a popup.").openPopup();
circle.bindPopup("<b>Testing!</b>I am a circle.");
polygon.bindPopup("Testing a polygon.");
```

Refresh your page and try clicking on our objects. The popup appears when you click on the object and with method openPopup() you can see the popup immediately open the page (for makers only). See below.



map with circle, triangle, and labeled place marker

Dealing with events

> You can interact with the map using events (see documentation for details).

Every object has its corresponding event which can be used as a function. It allows you to react to user interaction. For example:

```
function onMapClick(e) {
        alert("You clicked the map at " + e.latlng);
}

map.on('click', onMapClick);
```

With this code, you can click anywhere on the map to view the coordinates of that point. It will show an alert like the one below:

alert image

Try to click on your page and see the alerts.

Conclusion & Takeaways

- map: used to create a map on a page and manipulate it
- tileLayer: used to load and display tile layers on the map
- Marker: used to display clickable/draggable icons on the map
- Polygon: A class for drawing polygon overlays on a map
- Circle: A class for drawing circle overlays on a map
- Leaflet allows easy interaction with and manipulation of maps.
- The documentation is very complete.
- With knowledge in JavaScript and Leaflet you can create big projects.

Attribution

- Quick Start Guide - Leaflet - a JavaScript library for interactive maps. (n.d.). [Video]. https://leafletjs.com/examples/quick-start/

Google Maps API - Part 1

Goals
By the end of this lesson you will:
- Know how to set up Google Cloud
- Create a project in Google Cloud
- Learn how to use Google Maps API
- Enable APIs in Google Cloud

Introduction

In this lesson we will set up your Google Cloud project to use Google Maps Platform APIs. Before starting with Google Maps Platform we should create a project, enable billing and enable APIs. Also we will introduce to you Google Maps and its usefulness.

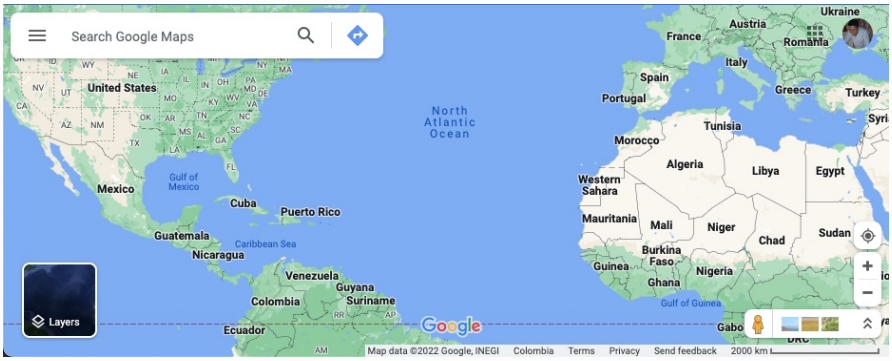


Image from https://maps.google.com/
>
For most personal projects, and even small business projects, leaflet is fine (and free). Google maps has extended functionality, so for large projects with heavy mapping needs, Google maps could offer more solutions.
>
Business Context

It is standard practice to include a map of the business location either on the contact page or in the footer of the website. Integrating Google Maps API makes the map interactive and provides the user easy access to directions in the event that they plan to visit the business in person.

What is Google Maps?

> Google Maps is a web mapping platform created by Google (Alphabet Inc) in the year 2005. It offers satellite imagery, aerial photography, street maps, 360° interactive panoramic views of streets (Street View), real-time traffic conditions, and route planning for traveling by foot, car, bike, air (in beta) and public transportation.
>
> Google Maps is available on all devices (smartphones, tablets, desktops, laptops and others) and also from a browser:

https://maps.google.com/.
>
Google Maps has enabled APIs that you can interact with and use to build solutions that require maps. For more details:

https://developers.google.com/maps/documentation
>
Get started with Google Maps Platform
Step 1: Create a project

Before you can use Google Maps Platform you should create a project to manage services, credentials, billing, APIs and SDKs. For each project a billing setup is required. You will only be charged if your project exceeds its quota of no-charge services. When developing for customers, it is recommended that you use the customer's credit card to activate billing. In this case the API would go through the customer's Google account.
>

warning

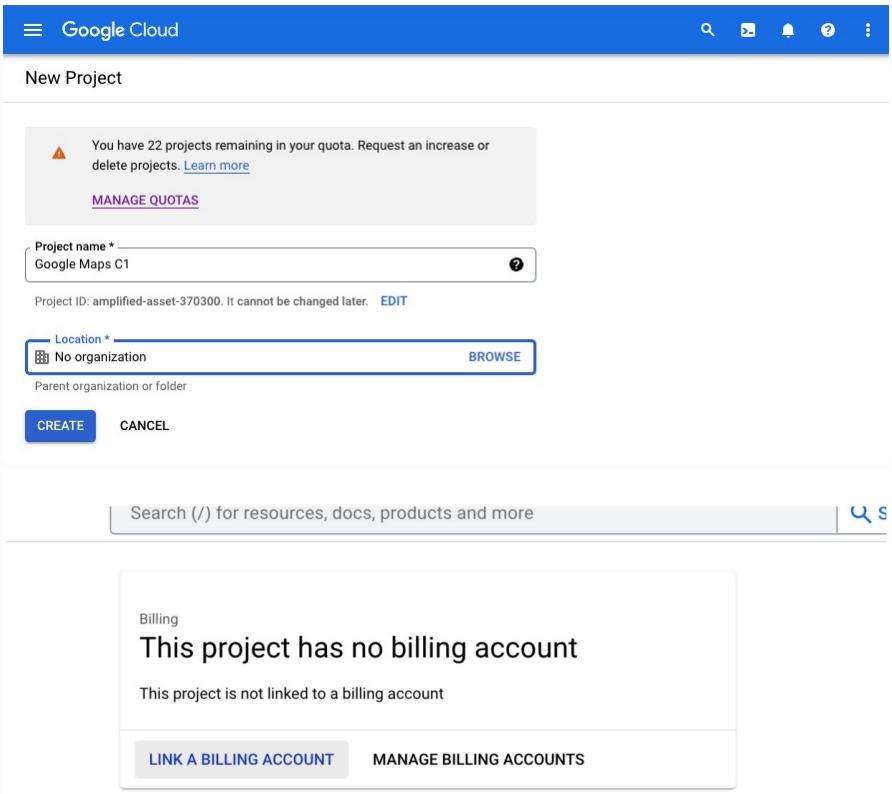
Prerequisites:

1. You should have a Google account
2. You should have a credit card

To create a Cloud project:

1. Create a new Google Cloud project in the Cloud Console, go to this direction

https://console.cloud.google.com/projectcreate



2. Enter your customized **Project name**
3. You can accept the **Project ID** or click EDIT and enter your customized ID
4. In **Location** in this case choose “No organization”
5. Click Create

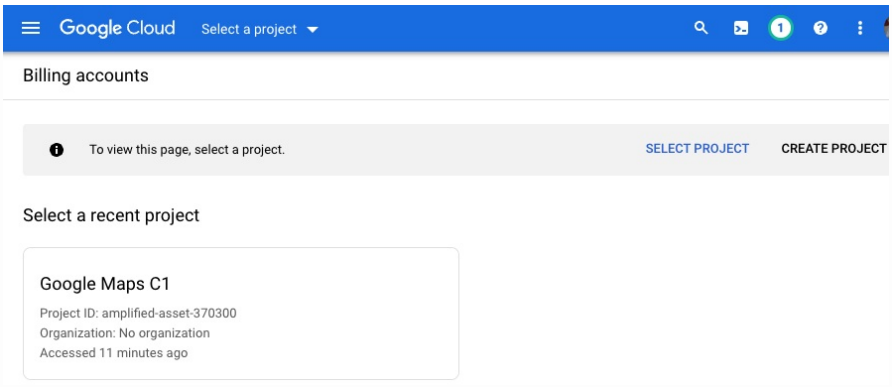
Step 2: Enable billing

It is very important that you enable billing to deploy your apps. Within your free quota your account will not be charged.

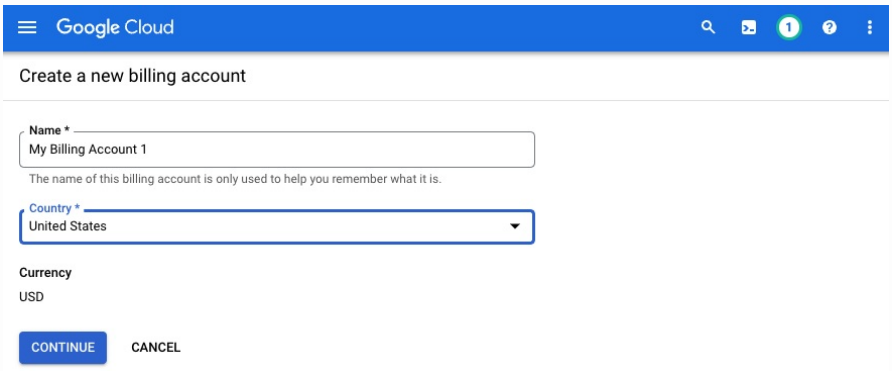
Be careful with exceeding your free quota to prevent headaches.
>
To enable billing:
1. Go to the billing page:

https://console.cloud.google.com/projectselector/billing

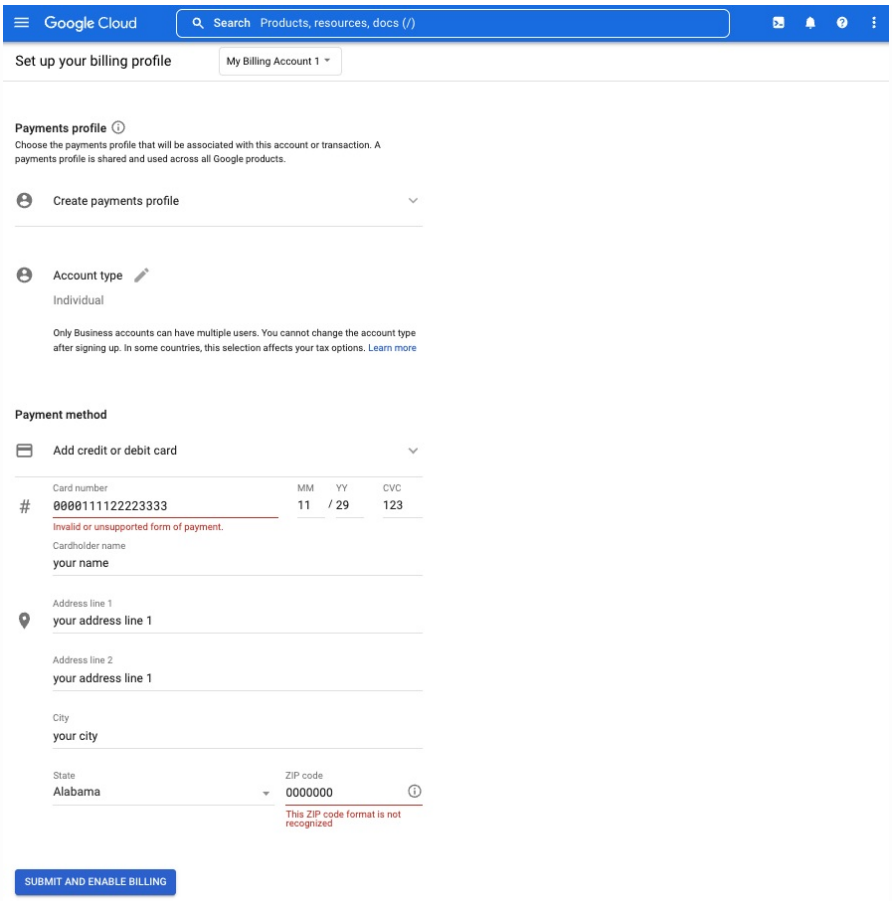
2. Select the project



3. Create a new billing account



4. Set up your billing profile



5. Submit and Enable billing

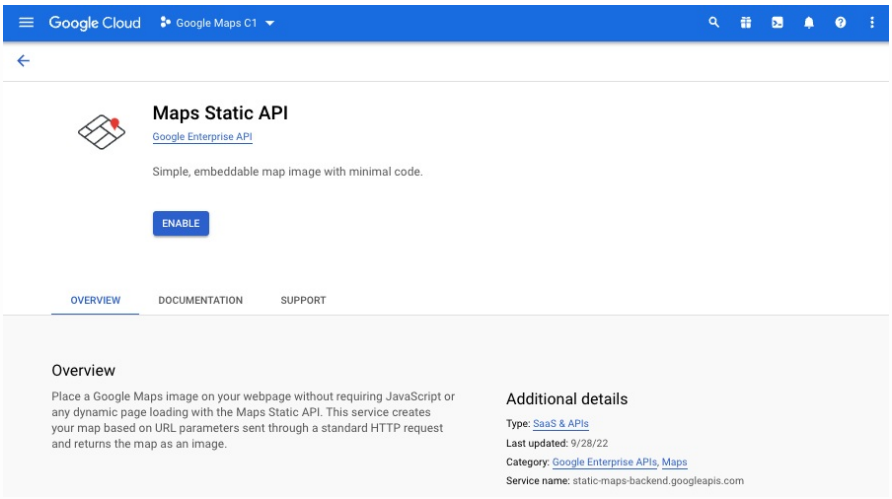
To gain more control over your costs, you can create a budget and set alerts. For more information, see Billing.
>

Step 3: Enable APIs

Now, let's go to enable APIs to use in your application, go to url

https://console.cloud.google.com/apis/library/static-maps-backend.googleapis.com

Click the button **Enable**.

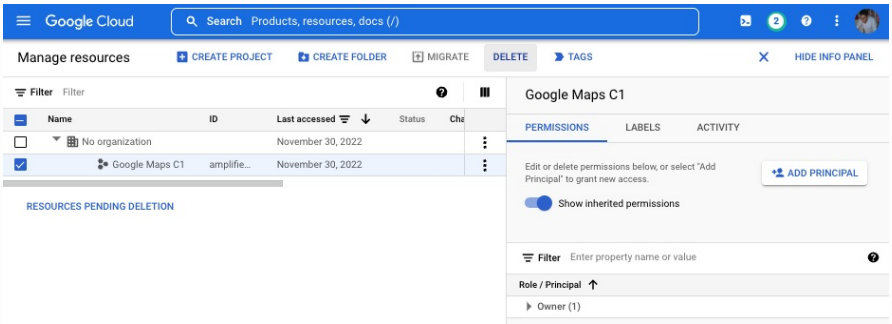


Bonus Step: Shutting down a project

> We suggest that you complete this additional step **after completing the program** if you intend to discontinue your use of the Google API.

1. Go to the Project page:

https://console.cloud.google.com/iam-admin/projects



2. Select the project to shut down and click **DELETE**. It is not recommended that you delete **the project at this time, as you will need to use the API in the upcoming lessons 3.2 and 3.3.**

Conclusion & Takeaways

- Google Maps is a powerful tool; it is used by many people and by developers worldwide.
- There is a great deal of documentation provided by Google for using all methods of these APIs.
- Note that a monthly recurring credit of 200 USD will be applied to your Billing Account. During the 90-day trial period, the 200 USD recurring credit will be used first, then anything further is deducted from the 300 USD trial credit. We HIGHLY doubt you will use 200 USD worth of Google Maps API credit for this program, so we encourage you to find other ways to use the 300 USD Cloud Billing credits, within 90-days, to further your learning! You can check any additional credit by selecting your billing account on the Billing page in the Cloud Console.

Attribution

- Google Maps Platform |. (n.d.). [Video]. Google Developers. https://developers.google.com/maps

Google Maps API - Part 2

> ### Goals
>
> By the end of this lesson you will:
> - Use Google API keys in your deploy
> - Learn how to restrict an API key
> - Develop some simple examples using Google Maps APIs
>
> ### Introduction
>
> In this second section on Google Maps you will learn more technical Google Maps skills. We will begin by setting up the API keys and then we will use them in our development. Also in this lesson, you will learn about APIs such as Geocoding and Directions services. Geocoding is important for getting the position and mark on the map, and Directions give an efficient path for rendering on the map.

Use API Keys

> Only the calls to the API that have proper authentication credentials can use the functions of Google Maps Platform. The API key is a unique alphanumeric string that associates your project with a specific API.

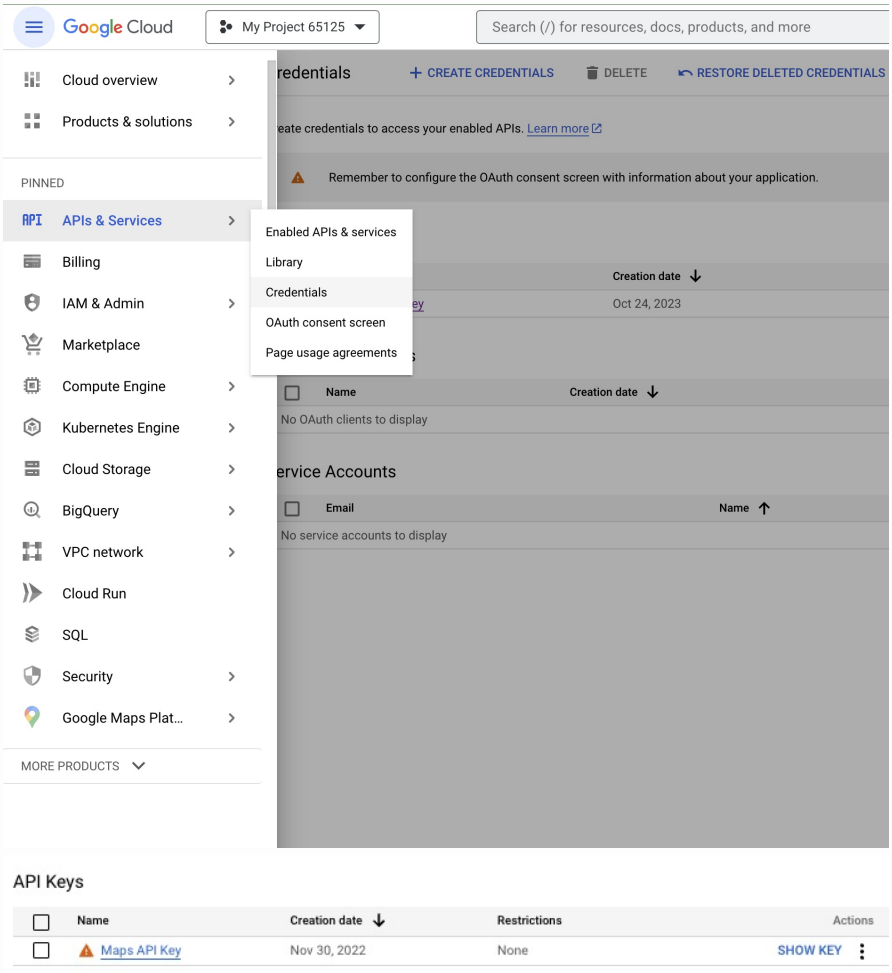
Create API keys

In the previous lesson (Google Maps Part 1) when you created the project, enabled the billing account, and enabled the API, it generated the API key.

You can check by clicking on

Navigation Menu > APIs & Services > Credentials

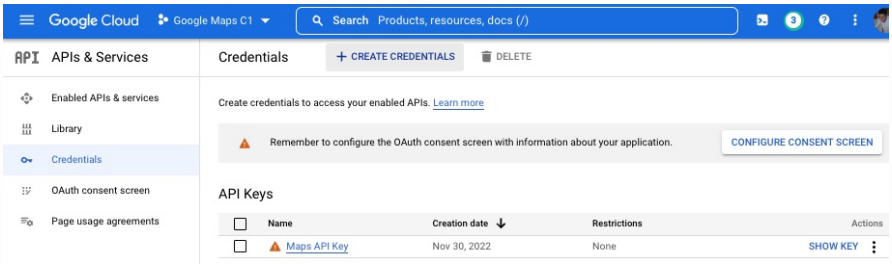
and then clicking on **SHOW KEY** if you have an API key in the list of API Keys.



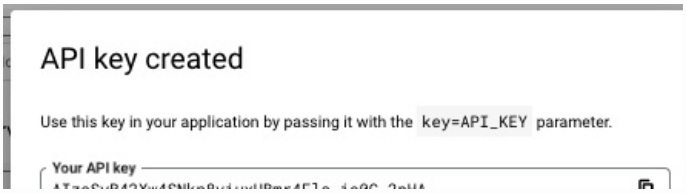
warning

In the case that you do not have API key, please follow the next steps:

1. Go to **Navigation Menu** > **APIs & Services** > **Credentials**
2. Click + **CREATE CREDENTIALS**



3. Chose API key from the list:
4. You will see a popup like this one:



5. Now you have your API key.

>

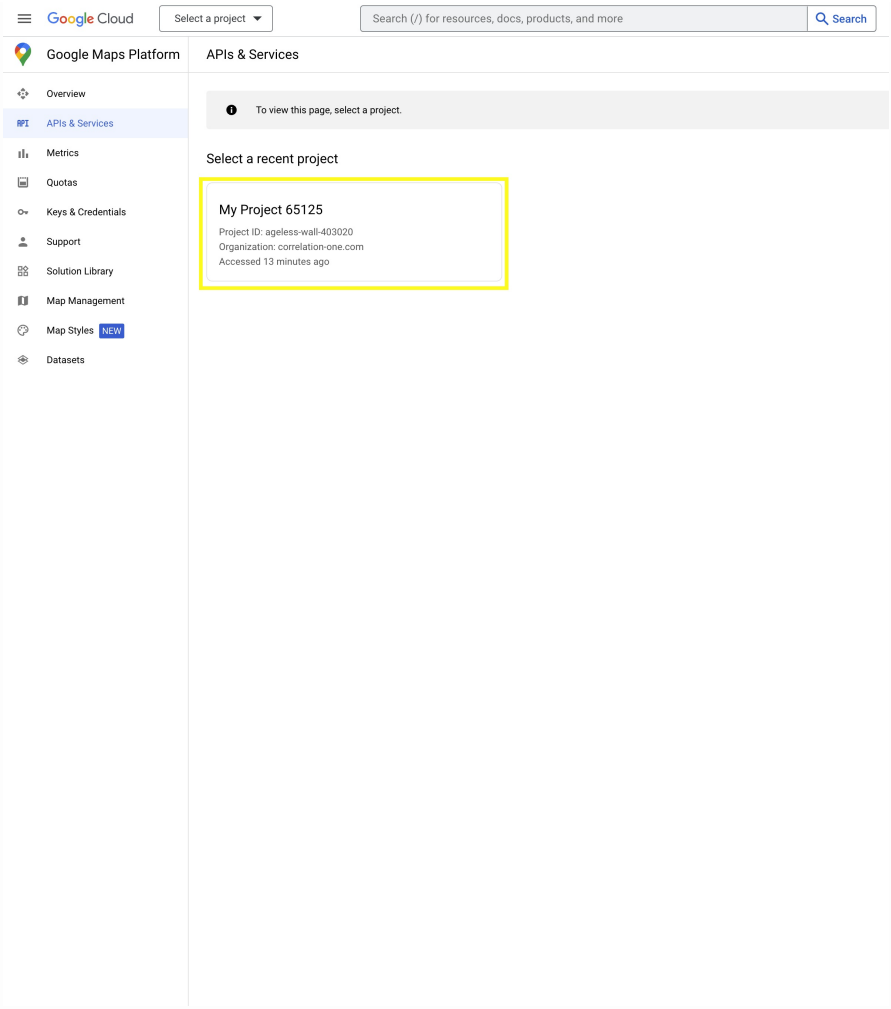
Enable the APIs or SDKs

In order to utilize the Google Maps Platform, we need to activate the APIs or SDKs that we intend to use for our project. Currently, we will need to **enable six APIs** essential for our upcoming examples and project.

1. Go to **Maps API Library Page**

https://console.cloud.google.com/project/_/google/maps-apis/api-list?utm_source=Docs_EnableAPIs&utm_content=Docs_Central

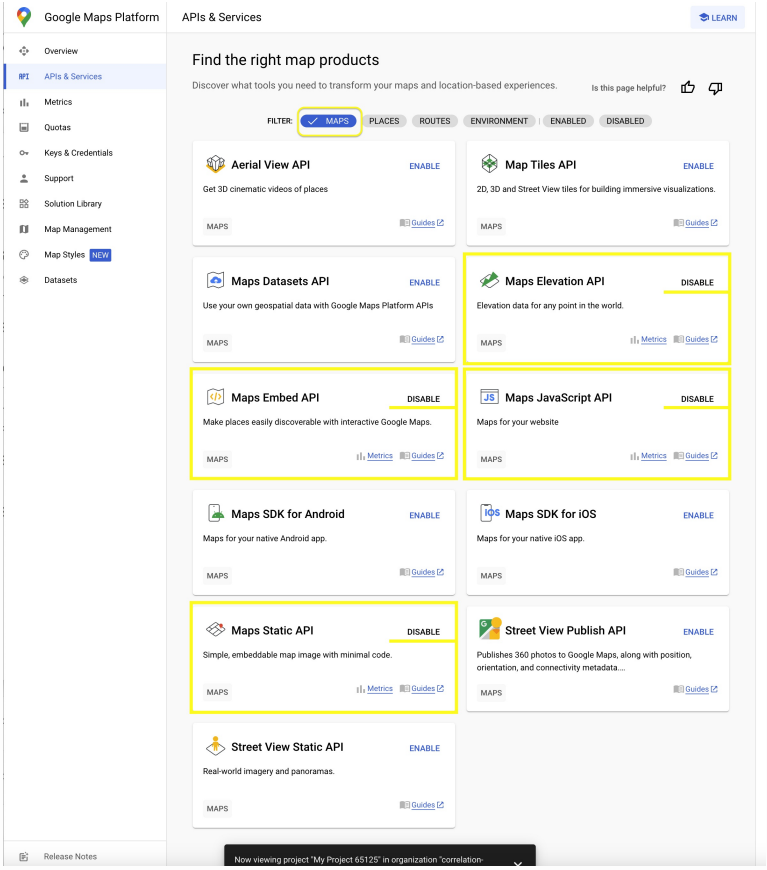
2. Select your **project**



3. Click on the **Maps** filter and **Enable** the following APIs:

topic

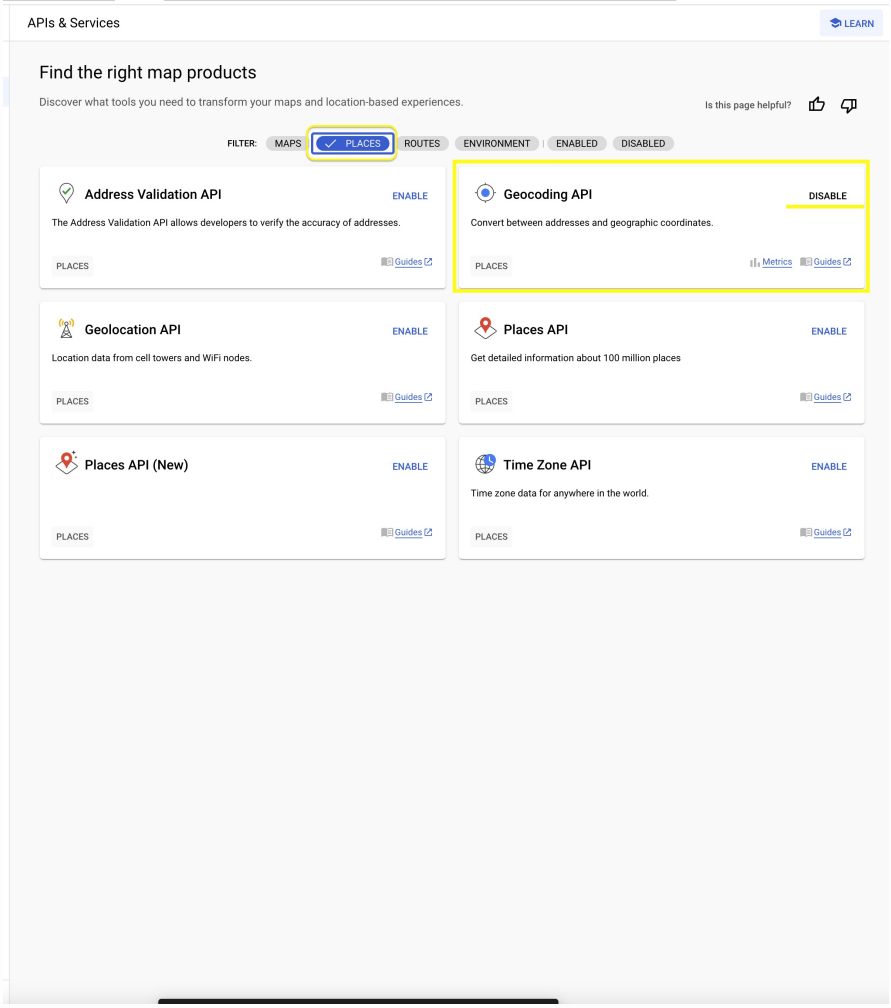
1. Maps Elevation API
2. Maps Embed API
3. Maps JavaScript API
4. Maps Static API



select maps APIs

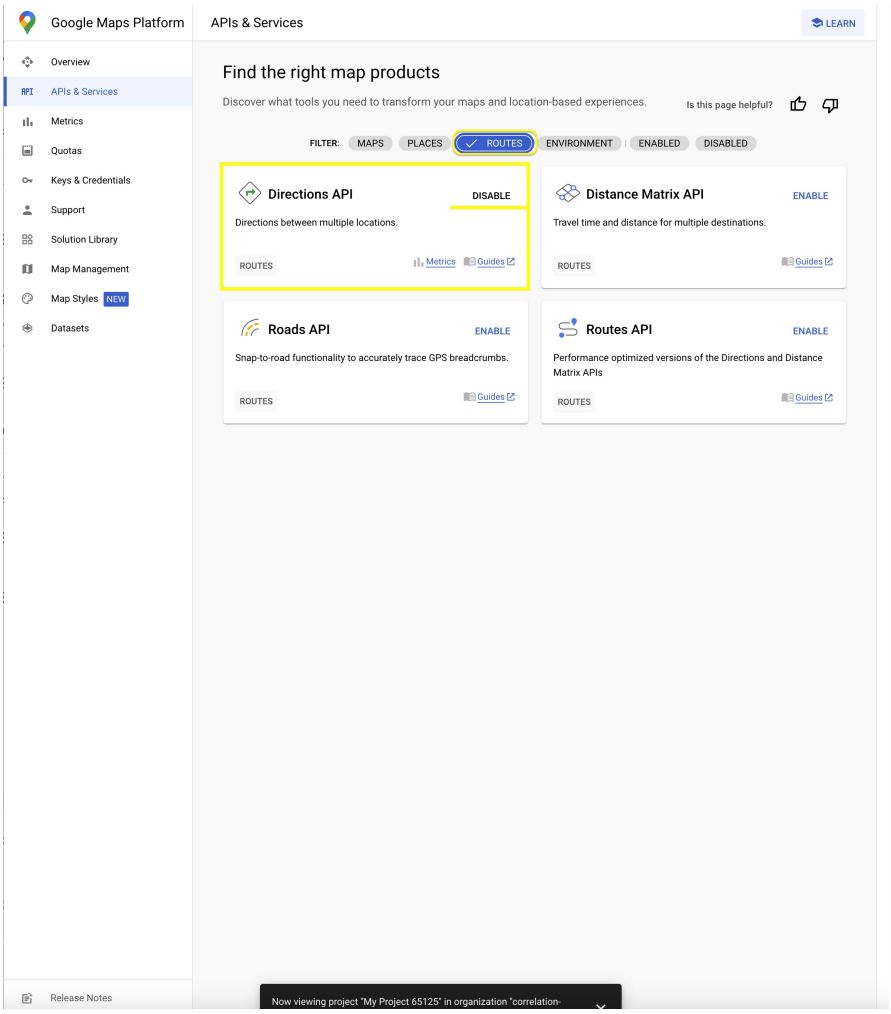
4. Click on the **Places** filter and **Enable** the following APIs:

topic
1. Geocoding API



5. Click on the **Routes** filter and **Enable** the following APIs:

topic 1. Directions API

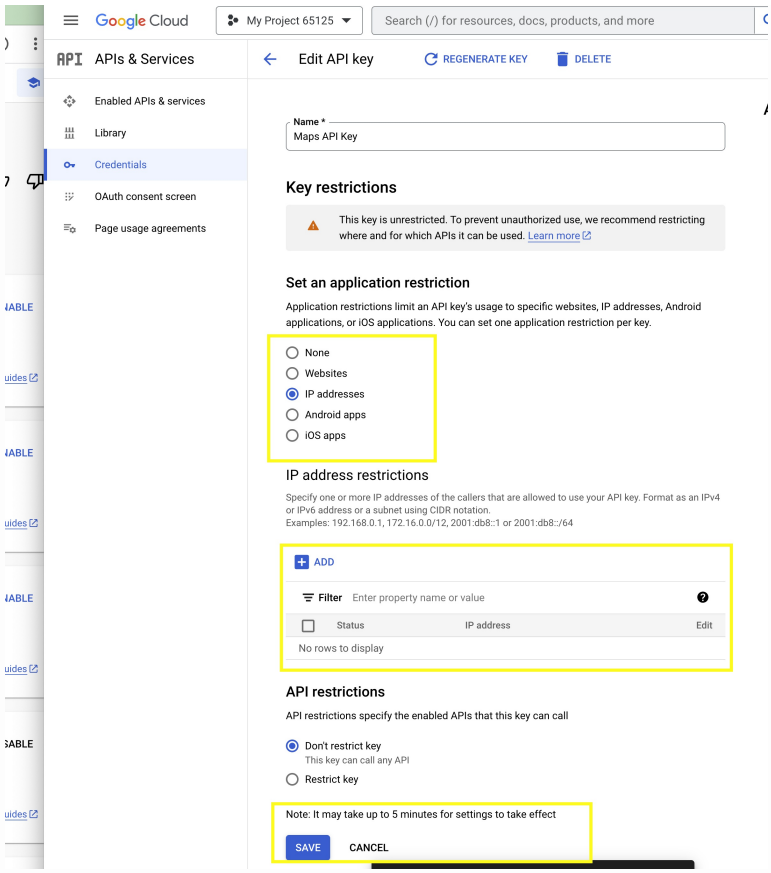


Restrict API keys

It is recommended that you restrict your API Keys to protect them from unwanted requests.

1. Go to **Navigation Menu** > **APIs & Services** > **Credentials**

2. Click on your **API key**.



restrict api keys

3. Select the option **IP addresses**
4. Add you IP address. You can locate your IP address by using this website.
   >
5. Leave the **API restrictions** as default - **Don't restrict key**
6. To finalize your changes, click **SAVE**

warning

IPv6 disabled/ Dynamic IP address

If you encounter an error after restricting your API key, please make sure you have both IPv4 and IPv6 added to the list. If your IPv6 is disabled or you have a dynamic IP address, you may have to temporarily remove any API restrictions for our upcoming examples.

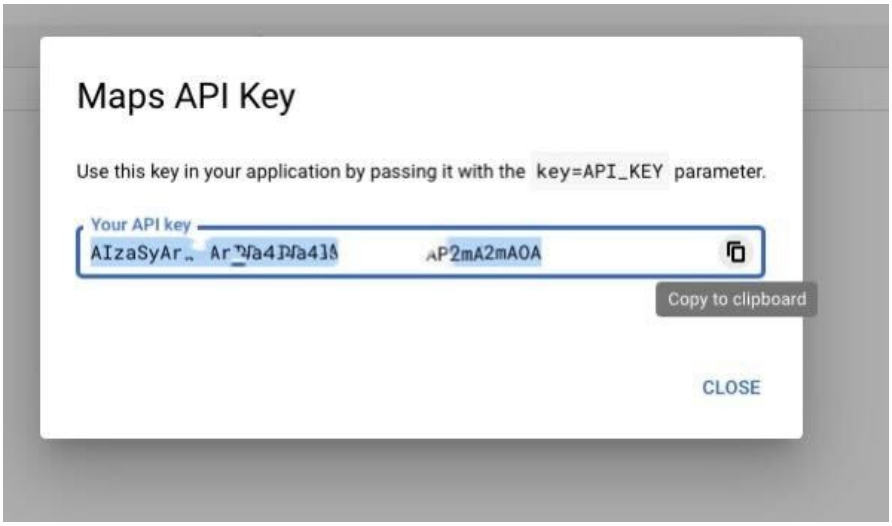
If you were unable to restrict your API key, it is crucial to protect your key and refrain from sharing it in a public GitHub repository or Codepen.

Add the API key to your request

> For you to use the API Key you must include the Google Maps web service. It looks like this:

```
https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly
```

Replace **YOUR_API_KEY** with your **API key**. HTTPS is required for requests that use an API key.



Part 1: Hello Google Maps
Now, let's deploy some methods of the APIs to see it in action:

Step 1: Preparing your page
1. Write the basic HTML code for a webpage or copy and paste the following code in your editor:

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

<head>

    <title>Hello Google Maps</title>

</head>

<body>

</body>

</html>
```

2. Include the Google Maps JavaScript file in the body. Make sure to replace **YOUR_API_KEY with your api key**.

```
<script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly"
      defer></script>
```

3. **Put a div element** with a certain id where you want your map to be:
   `<div id="map" style="width: 600px; height: 400px;"></div>`

Up to this point we have the following code which displays a blank page.

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

<head>
    <title>Hello Google Maps</title>
</head>

<body>
    <script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly" defer></script>

    <script>

    <!-- CODE GOES HERE  -->


    </script>

    <div id="map" style="width: 600px; height: 400px;"></div>
</body>

</html>
```

Step 2: Setting up the map

Let's create our first map. Add the following code between `<script> CODE GOES HERE </script>` tags and below the script that calls the Google API, then save and refresh your browser:

```
function initMap() {

            const coordinates = { lat: 40.689, lng: -74.044 };

            const map = new
google.maps.Map(document.getElementById("map"), {
                zoom: 13,
                center: coordinates,
            });

            <!-- const marker = new google.maps.Marker({
                position: coordinates,
                map: map,
            }); -->
        }
window.initMap = initMap;
```

For this example, we are utilizing the **coordinates of Statue of Liberty in NY, New York.**

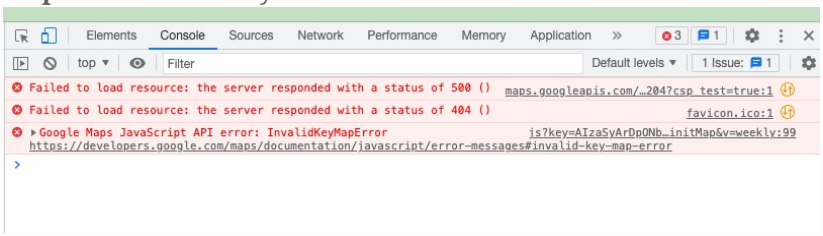
challenge

“lat” and “lng” values

Feel free to modify the “lat:” and “lng:” values to the coordinates of a place of your choice.

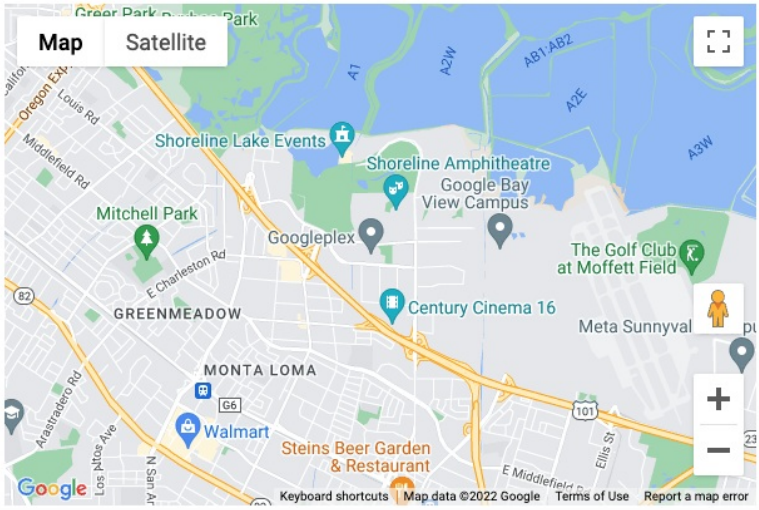
We recommend using the website https://www.latlong.net/ to find the latitude and longitude of a specific location.

> In the above code, the **callback** parameter executes the **initMap** function after the API loads. The async attribute allows the browser to continue to parse the remainder of your page while the API loads. Once it has loaded, the browser will pause and immediately execute the script.

> **Troubleshooting Tips**
>
> Here are some tips for troubleshooting if you're having trouble viewing an embedded map:
>
> 1. If you're unable to load your **Google Billing Account**, try accessing it through a different browser such as **Microsoft Edge** or **Firefox** to obtain your API key.
> 2. If you're experiencing issues with **expired** or **invalid keys**, ensure that your Google Console's **IP address restrictions** or **API restrictions settings** are properly **configured** with the correct IP address. Additionally, make sure that you're using the **correct Google Script Tag**.
> 3. If you see an error message that says **"Oops! Something went wrong. This page didn't load Google Maps correctly. See the JavaScript console for technical details."**, it could mean that your **API key is incorrect or your script isn't properly linked**.
> 4. If you encounter an error after r**estricting your API key**, please make sure you have **both IPv4 and IPv6 added to the list**. If your IPv6 is **disabled**, you may have to temporarily remove any API restrictions.
> 5. Please make sure you are **not connected to a VPN**.
>
> **Bonus Tip**: If you're unsure of what the error is, you can use the **Inspect tool** to check your **console** for more information.
>
> 
>

Part 2: Markers and circles

Search for the coordinates of Googleplex in Mountain View, CA, and put these **coordinates** in your code then refresh your browser.



google map embedded

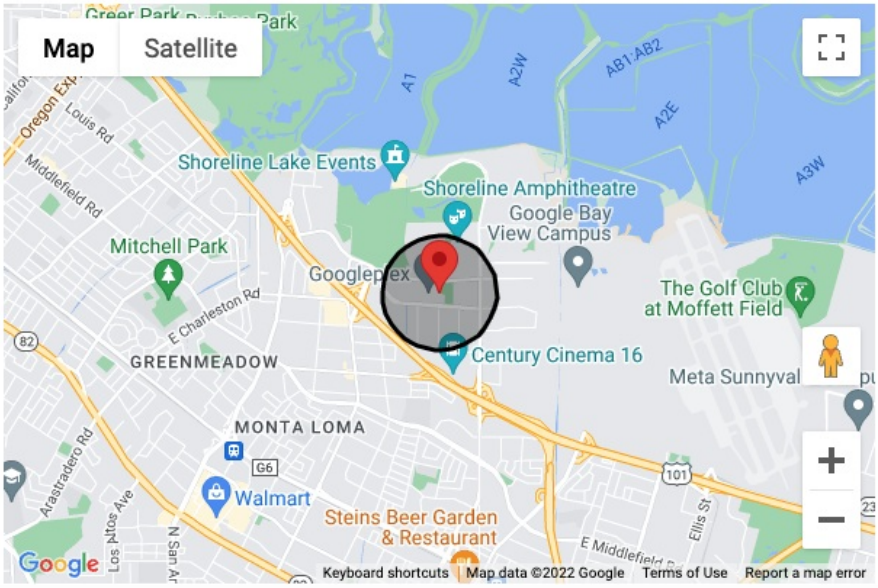
Now, let's add some markers and circles:

1. Add **marker** to map:

```
// inside the initMap() function

const marker = new google.maps.Marker({
               position: coordinates,
               map: map,
           });
```

2. Add a **circle** to map. In this case you should indicate the radius of the circle in meters:

```
// inside the initMap() function

const circle = new google.maps.Circle({
                map,
                center: coordinates,
                radius: 600,
            });
```



You can add more custom options for changing the design of the figure.

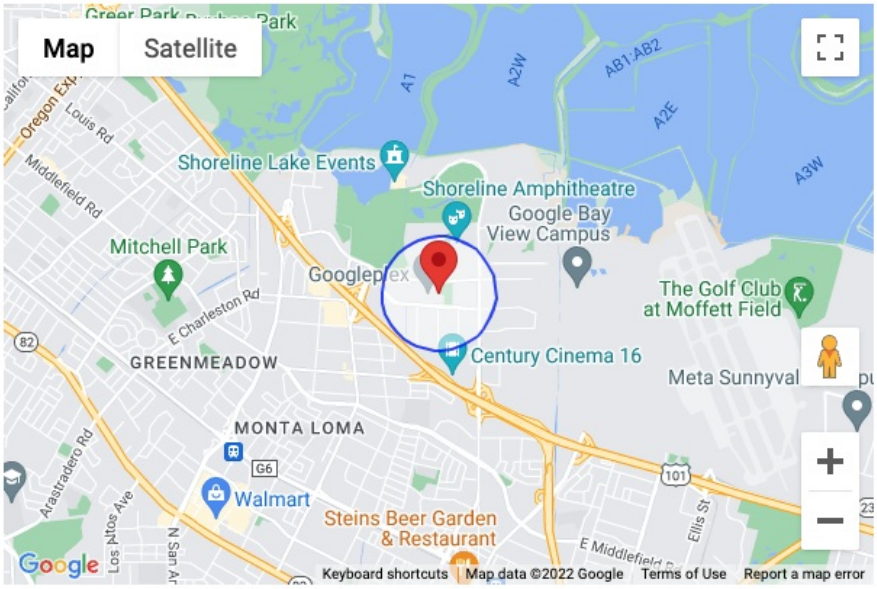
```
const circle = new google.maps.Circle({
                    strokeColor: "blue",
                    strokeOpacity: 0.8,
                    strokeWeight: 2,
                    fillColor: "#FFF",
                    fillOpacity: 0.5,
                    map,
                    center: coordinates,
                    radius: 600,
            });
```

Your code should look like this:

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

<head>

    <title>Hello Google Maps</title>

</head>

<body>
<script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly"
      defer></script>
      <div id="map" style="width: 600px; height: 400px;"></div>
<script>
```

```
    function initMap() {

            const coordinates = { lat: 37.422040, lng:
-122.082810 };

            const map = new
google.maps.Map(document.getElementById("map"), {
                zoom: 13,
                center: coordinates,
            });

const marker = new google.maps.Marker({
               position: coordinates,
               map: map,
           });
           const circle = new google.maps.Circle({
                    strokeColor: "blue",
                    strokeOpacity: 0.8,
                    strokeWeight: 2,
                    fillColor: "#FFF",
                    fillOpacity: 0.5,
                    map,
                    center: coordinates,
                    radius: 600,
            });
        }

window.initMap = initMap;


</script>
</body>

</html>
```

Now, your map should look like this:



Part 3: Rectangles

Add a rectangle with the following coordinates:

> **Tip**: You can find Google Maps API's documentation on rectangles here.

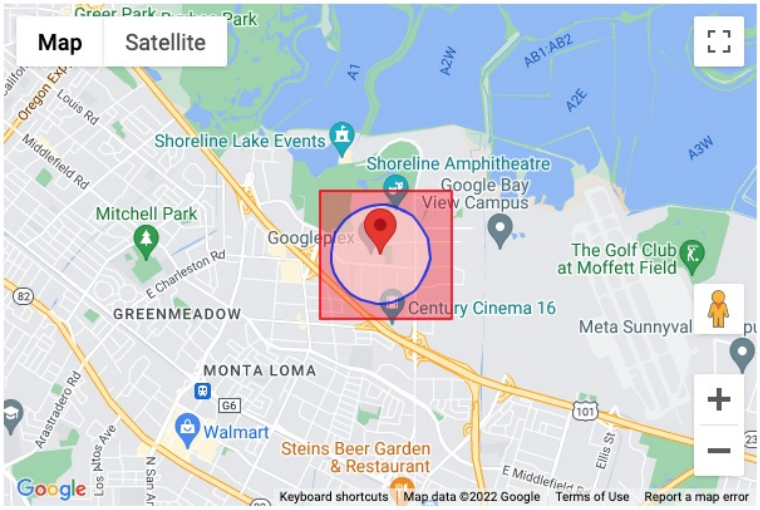
Also, make sure you write your code for the **rectangle** inside the `initMap` function and under the `const circle`.

Coordinates for the rectangle:

info

north: 37.429
south: 37.415
east: -122.073
west: -122.091

And now, your map should look like this:



circle, marker, and square on map

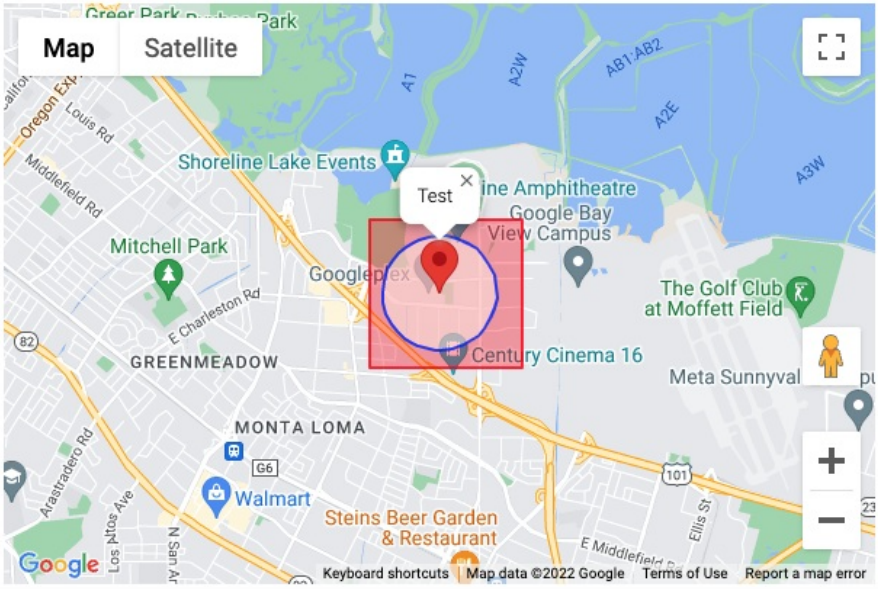
Working with popups in makers

Part 4: Popups

Sometimes you need to attach information to your map. With the following code you can easily add a popup.

```
// inside the initMap() function
let infowindow = new google.maps.InfoWindow();

            google.maps.event.addListener(marker, 'click',
function() {
                        infowindow.setContent("Test");
                        infowindow.open(map, marker);
            });
            markers.push(marker);
```

Click on the **red marker** and you will see the “Test” popup.



For more details see documentation.

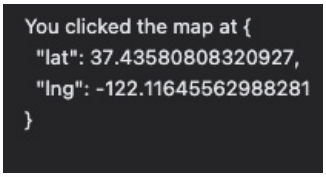
Part 5: Events

You can interact with the map using events (see documentation for details). Every object has its corresponding event and this can be used as a function. It allows you to react to user interactions. For example:

> **Tip**: Make sure to comment out or **delete the code** from our **prior popup example**.

```
// inside the initMap() function
map.addListener("click", (e) => {
                alert("You clicked the map at " +
JSON.stringify(e.latLng.toJSON(), null, 2));
            });
```

Now when you click anywhere on the map, you will see an alert box that includes the coordinates of the point you clicked on.



alert image from map

Try **clicking** on your map to see the **alerts**.
>
Geocoding services
This service helps to convert addresses into geographic coordinates; the result can be used for place makers or positions the map. The **GeocoderRequest** has the following structure:

```
{
 address: string,
 location: LatLng,
 placeId: string,
 bounds: LatLngBounds,
 componentRestrictions: GeocoderComponentRestrictions,
 region: string
}
```

Required parameters are address, location or placeID. You must supply one, and only one.
>
The **GeocoderResult** has the following structure:

```
results[]: {
 types[]: string,
 formatted_address: string,
 address_components[]: {
   short_name: string,
   long_name: string,
   postcode_localities[]: string,
   types[]: string
 },
 partial_match: boolean,
 place_id: string,
 postcode_localities[]: string,
 geometry: {
   location: LatLng,
   location_type: GeocoderLocationType
   viewport: LatLngBounds,
   bounds: LatLngBounds
 }
}
```

The status codes are:
- “OK”
- “ZERO_RESULTS”
- “OVER_QUERY_LIMIT”
- “REQUEST_DENIED”
- “INVALID_REQUEST”
- “UNKNOWN_ERROR”
- “ERROR”
>
Example of a Geocoding services:

The following code has input where you indicate the place to search and return a place maker in the map.

1. Initialize variable “**geocoder**” and **map** with the construct of Google Maps API. Let’s also add a little **form** under the map **for the user to input the address** they would like to search for:

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

<head>

    <title>Hello Google Maps</title>

</head>

<body>
  <div id="map" style="width: 600px; height: 400px;"></div>
    <div>
        <input id="address" type="textbox">
        <input type="button" value="Search Address"
onclick="getCoordinates()">
    </div>
<script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly"
      defer></script>
<script>
        let geocoder;
        let map;

        function initMap() {
            geocoder = new google.maps.Geocoder();
            const coordinates = { lat: 37.422040, lng:
-122.082810 };

            map = new
google.maps.Map(document.getElementById("map"), {
                zoom: 13,
                center: coordinates,
            });

        }
    </script>
</body>

</html>
```

2. **Add the function** responsible for getting the address of the front page and set to the center on your map.

> **Tip**: Make sure to **add the function inside the `script` tag and under the `initMap` function**

```
function getCoordinates() {
            let address =
document.getElementById('address').value;
            geocoder.geocode({ 'address': address }, function
(results, status) {
                if (status == 'OK') {
                    map.setCenter(results[0].geometry.location);
                    let marker = new google.maps.Marker({
                        map: map,
                        position: results[0].geometry.location
                    });
                } else {
                    alert('Geocode was not successful for the
following reason: ' + status);
                }
            });
        }
```

3. This is the result when you search for **Mountain View**:

>

Directions Services

This service helps to calculate directions (an efficient path) specifying an origin and a destination. Is important to indicate the method of transportation. Directions are displayed as a polyline drawing the route on a map and can return multi-part directions using a series of waypoints.

>

The DirectionsRequest object literal contains the following fields:

```
{
  origin: LatLng | String | google.maps.Place,
  destination: LatLng | String | google.maps.Place,
  travelMode: TravelMode,
  transitOptions: TransitOptions,
  drivingOptions: DrivingOptions,
  unitSystem: UnitSystem,
  waypoints[]: DirectionsWaypoint,
  optimizeWaypoints: Boolean,
  provideRouteAlternatives: Boolean,
  avoidFerries: Boolean,
  avoidHighways: Boolean,
  avoidTolls: Boolean,
  region: String
}
```

Required parameters are origin, destination and travelMode.

Travel Modes:
- DRIVING (Default)
- BICYCLING
- TRANSIT
- WALKING
>
The DirectionsResult contains the result of the directions query, which you may either handle yourself, or pass to a `code` directionsRenderer object, which can automatically handle displaying the result on a map.
The status codes are:
- OK
- NOT_FOUND
- ZERO_RESULTS
- MAX_WAYPOINTS_EXCEEDED
- MAX_ROUTE_LENGTH_EXCEEDED
- INVALID_REQUEST
- OVER_QUERY_LIMIT
- REQUEST_DENIED
- UNKNOWN_ERROR
>
Example of Directions Services
The following code has input where you indicate the place origin and destination to search and return an efficient path.
1. Initialize variables **directionsService**, **directionsRenderer** and **map** with the construct of Google Maps API. Add a little **form** on the front page so that the user can input the fields **origin** and **destination**: Add the code below to the body of a **basic html**:

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

    <title>Hello Google Maps</title>

</head>

<body>
<div id="map" style="width: 600px; height: 400px;"></div>
    <div>
        <label>Origin</label>
        <input id="origin" type="textbox">
        <label>Destination</label>
        <input id="destination" type="textbox">
        <input type="button" value="Search Route"
onclick="calcRoute()">
    </div>
<script src="https://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&callback=initMap&v=weekly"
      defer></script>
<script>
        let directionsService;
        let directionsRenderer
        let map;
        // Initialize and add the map
        function initMap() {
            directionsService = new
google.maps.DirectionsService();
            directionsRenderer = new
google.maps.DirectionsRenderer();
            const coordinates = { lat: 37.422040, lng:
-122.082810 };

            map = new
google.maps.Map(document.getElementById("map"), {
                zoom: 13,
                center: coordinates,
            });
            directionsRenderer.setMap(map);
        }
        </script>
        </body>

</html>
```

2. Add the function responsible for obtaining an origin and a destination

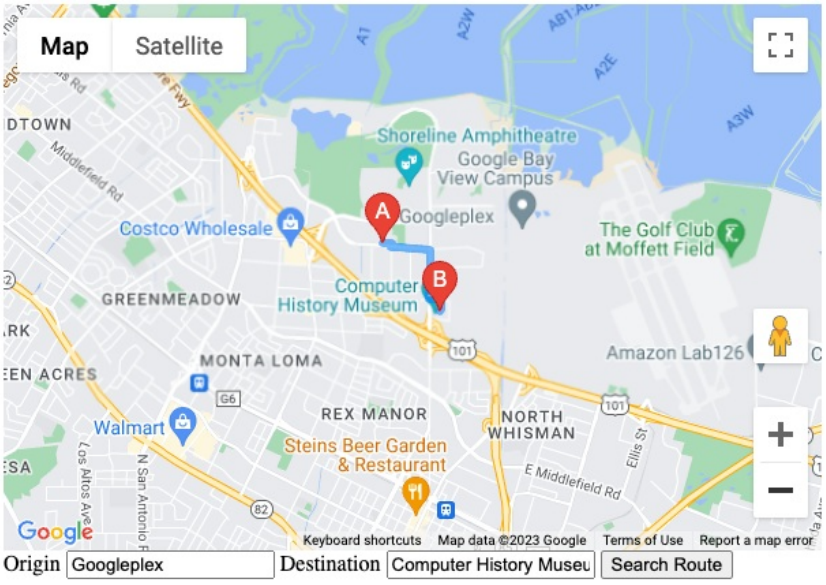
on the front page and render the result on map:

> **Tip**: Make sure to add the function **inside the `script` tag** and **under the `initMap` function**

```
function calcRoute() {
            let start = document.getElementById('origin').value;
            let end =
document.getElementById('destination').value;
            let request = {
                origin: start,
                destination: end,
                travelMode: 'DRIVING'
            };
            directionsService.route(request, function (result,
status) {
                if (status == 'OK') {
                    directionsRenderer.setDirections(result);
                } else { alert("An unexpected error occurred")}
            });
        }
```

In this case, we use **DRIVING** in the field **travelMode**
>

3. This is the end result from **Googleplex** to **Computer History Museum** if we were to **drive**:



Conclusion & Takeaways

- Generally, the Google documentation is clear and simple, so we can easily understand the API and apply it to our development.
- You can mark objects with google.maps.Marker
- You can circle objects with google.maps.Circle
- You can create a polygon object with google.maps.Polygon
- You can create a rectangle object with google.maps.Rectangle
- Other options are available: see documentation.
- Note that a monthly recurring credit of 200 USD will be applied to your Billing Account. During the 90-day trial period, the 200 USD recurring credit will be used first, then anything further is deducted from the 300 USD trial credit. We HIGHLY doubt you will use 200 USD worth of Google Maps API credit for this program, so we encourage you to find other ways to use the 300 USD Cloud Billing credits, within 90-days, to further your learning! You can check any additional credit by selecting your billing account on the Billing page in the Cloud Console.

Attribution

Google Maps Platform |. (n.d.). [Video]. Google Developers. https://developers.google.com/maps

Google Maps Tools

Goals

By the end of this project you will:
- Practice all of the concepts and topics learned in this week
- Explore Google documentation and learn new things
- Improve your skills in Javascript and in working with APIs
>
Introduction
In this project you will apply all of this week's learning. Our focus is to develop a solution with the different Google Maps APIs, which centralize many functions in only one place and from which the user can choose what they need to do.
>
Business Context
The ability to generate a map with customized functionality is an important tool in your web development toolbox. Clients will expect you to have this skill in hand. What's more, giving the map additional functionality can improve the user experience and ultimately deliver better results for the client.

Instructions

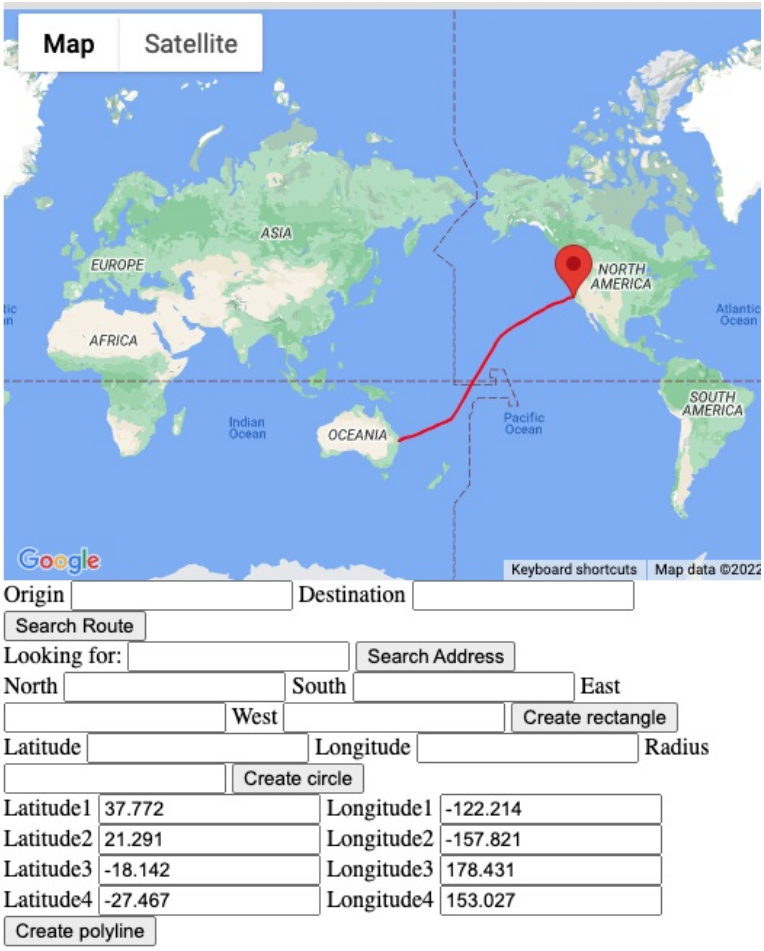
1. Create a form and function to place a marker on the map.
2. Create a form and function to get the path of origin to destination by filling in the form.
3. Create a form and function to place a circle on the map.
4. Create a form and function to place a rectangle on the map.
5. Create a form and function to place a polygon on the map.
6. Create a form and function to place polylines on the map with four coordinates.
7. Create an alert when the user clicks on the map to show the latitude and longitude of the point using the InfoWindow API.

Note: Use HTML and advanced CSS for a better user interface.

Tip: To get the coordinates of a location use the page https://www.latlong.net/ for searching the latitude and longitude.

Solution:

Answers will vary. The result should look something like this:



map example

Attribution

- Google Maps Platform |. (n.d.). [Video]. Google Developers. https://developers.google.com/maps

Project

Communicating on the Web

> For this project, you will be adding something we've studied this week to your personal web page. This makes the code yours, and can be an example of your knowledge. There doesn't necessarily have to be a purpose for what you choose to add; in some ways you simply want to show off your skills to a potential employer. Choose from:
> - Leaflet
> - Asynchronous functions
> - Collect form information in json format
> - Google Maps
>
> Whatever you choose, be sure to have clear comments in your code so a reviewer can understand what you did, and also so you can go back a month or more later and be reminded of what your thought process was.
>
> When finished, submit your code!

Submission steps:

Before you click on "Mark as Completed":

You need to do one of the following: either upload all of your files to Codio or deploy a GitHub Page for this project.

Also, if you mark this project as complete but any of the boxes are blank, your TA will be unable to grade your project.

Codio upload:

- Make sure all of your project code has been uploaded to Codio.
  - If you did not write your code in Codio, you will need to import all of the required files into your workspace file tree.
  - You can do this by going to File => Upload Files, and either manually importing each file, or dragging and dropping your project folder.
  - Please refer to this video if you are unsure of what to do.

GitHub Pages:

- If you would rather upload your project to GitHub, please make sure to have the project deployed as a GitHub page so we can thoroughly test it.
  - If you are unsure of how to do this, please follow these instructions.

- It is important to understand you will need to make a separate repository for every project. You **cannot** deploy multiple pages from the same repository, even with different branches.

Also, no matter whether you uploaded your files from your computer or not, **make sure to thoroughly test your code**! This only takes a few minutes, but will prevent the amount of resubmissions because you missed something.