Risan Bagja Follow
Mar 16, 2018 · 8 min read

Track User’s Location and Display it on Google Maps

Globe by Nicole Harrington on Unsplash.

This is actually my answer to someone’s question on PHP Indonesia Facebook group. How can we track user’s location continuously using the Geolocation API and display it on Google Maps?

Before getting started, we’re going to need an API Key to use the Google Maps JavaScript API. As a safety measure, don’t forget to set the HTTP referrers on your API key to your own domains.

Displaying Google Map on your Page

First, let’s start with a simple example: display a map on our HTML page using Google Maps Javascript API.

Our HTML Structure

Create a new file and put the following HTML structure:

Safe it as `index.html`. Don't worry about the `styles.css` and `script.js`, we'll create these files later.

The `div` with the id of `map` is where we'll load the map. Also, don't forget to replace the `YOUR_API_KEY` part with your own API key from Google.

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

The optional `callback` property is where we can define which function to call once the Google Maps JavaScript API is loaded. In our case it's the `init` function, we'll define it later.

Add Some Styles

Next, let's add some styles to our HTML page. Create a new CSS file and put the following CSS rules:

Save this file as `styles.css`. This CSS file simply to make our map's `div` occupies the whole screen. We achieve this through flexbox:

```
body {
    height: 100%;
}

.container {
    display: flex;
    flex-direction: column;
    height: 100%;
}

.map {
    flex: 1;
}
```

Initialize The Map

Now, all we have to do is create a new JavaScript file for displaying the map. Type the following codes and save it as `script.js`.

```
function init() {
  new google.maps.Map(document.getElementById('map'), {
    center: { lat: 59.325, lng: 18.069 },
    zoom: 15
  });
}
```

This `init()` function will be automatically invoked once the Google Maps JavaScript API is loaded. Inside this function we create a new Map instance:

```
new google.maps.Map(element, options);
```

The first parameter is the DOM element where the map will be displayed. In our case, it's the `div` with the id of `map`. The second parameter is our map configuration object. The `center` property defines the initial coordinate of our map's center. The `zoom` property defines the initial zoom level. You can read more about other options in Map Options Documentation.

Now if we open up our page in the browser, we should see our map is successfully loaded like this:

Our map is loaded!

Adding Marker to Your Map

Within the `init` function, we add a new statement that will create an instance of Marker class.

```
new google.maps.Marker(options);
```

It accepts a single argument: an object of marker's options. The `map` property is where the marker will be displayed, that's why we pass the created `Map` instance to this property. The `position` defines the marker's position. Check out all of the available options in Marker Options Documentation.

We should now see the marker placed on the map.

Marker on our map.

Get User's Location

Our next step would be retrieving user's location using the JavaScript

Open up our `scripts.js` file again. Then add the new code section for retrieving user's location:

The following code checks whether the GeoLocation API is supported by the browser. We simply check if the `geolocation` property exists within the `navigator` object.

```
if ('geolocation' in navigator) {
    // Geolocation API is supported
} else {
    // Geolocation API is not supported
    alert('Geolocation is not supported by your browser.');
}
```

If the GeoLocation API is supported, we can then safely use the `getCurrentPosition()` method to retrieve user's location.

```
navigator.geolocation.getCurrentPosition(success, error, [options])
```

This method accepts three parameters:

- `success`: A callback function that will be invoked once the user's location is retrieved successfully. The callback receives a `Position` object that holds the user's location.
- `error`: A callback function that will be invoked if user's location is failed to retrieve. The callback accepts a `PositionError` object.
- `options`: Is an optional `PositionOptions` object that can be used to configure the retrieval process

Open our page in the browser. It will ask your permission to get your current location. Click **Allow** to give it permission and proceed.

Our page ask for the permission to get the user's location.

> ⚠️ Geolocation API is only available in HTTPS
>
> Note that this Geolocation API is only available in secure contexts. So you'll need to access your page through HTTPS protocol.

If it's successful, you'll get your location printed on the console similar to this:

Simulating User’s Location on Chrome

On Chrome, we can simulate the user’s location. Open up your developer tools. Click on the three-vertical-dots button on the top-right of your developer tools screen. Click on **More tools** » **Sensors** menu. It will bring a new tab named **Sensors** where you can easily override the position.

The **Sensor** tab on developer tools.

There’s also some presets for various city locations that we can choose from. Select some cities and reload the page, you should get the city’s location printed on the console.

Handling Errors

From the `Sensors` tab, you can also simulate the position unavailable error. From the drop-down select `Location unavailable` option and reload the page. You’ll get an alert like this:

Simulate location unavailable error.

We only got the error `code`, but the `message` property is empty. Apparently, the specification already specifies that this `message` property is for debugging only and not to be shown directly to the user. That’s why we should rely on the `code` and provide our own error message instead.

The `PositionError.code` may have the following values:

- `1`: The permission is denied.
- `2`: The position is unavailable, the Geolocation failed to retrieve the user’s location.
- `3`: Timeout—when the device does not return the user’s location within the given `timeout` option.

Let’s modify our `script.js` to print out the appropriate error message:

Now if we select the `Location unavailable` option from the **Sensors** tab again and reload the page. We should get an alert: `Error (2): Position unavailable.`.

Display User’s Location on Google Maps

We successfully retrieved user’s location. But our map and marker still displaying the given initial location. Let’s modify our `script.js` file to properly center the map and marker to user’s location.

We use the marker’s `setPostion()` method to change the location of the

```
marker.setPosition({
  lat: position.coords.latitude,
  lng: position.coords.longitude
});
```

And we also use the map's `panTo()` method to center the map to user's location:

```
map.panTo({
  lat: position.coords.latitude,
  lng: position.coords.longitude
});
```

Now if we load our page, both the map and marker should show the retrieved user's location.

Let's Refactor our Code

Our code now is getting a bit messy. Let's refactor it by extracting each step into its own function.

We use ES2015 object destructuring to easily get access to `position.coords.latitude` and `position.coords.longitude` and rename them `lat` and `lng` accordingly:

```
onSuccess: ({ coords: { latitude: lat, longitude: lng } }) => {
  //
},
```

Track User's Location with watchPosition

Up until now, we only retrieve user's location once. How can we track user's location continuously? By using `setInterval()`? Well, we can, but it's not the best solution. To track user's location continuously, we can use the provided `watchPostion()` method.

Let's modify our `script.js` file to use the `watchPosition` instead.

```
id = navigator.geolocation.watchPosition(success[, error[, options]])
```

The `watchPosition()` accepts three parameters, it exactly the identical parameters like the one provided for `getCurrentPosition()`. The only difference is the `error` callback is optional for the `watchPosition()`.

It also returns an `id` like `setTimeout()` or `setInteval()` functions. This `id` can be used to stop tracking user's location with `clearWatch()` method.

If we reload our page, both the marker and the map position should now be updated every time the location is changed.

The PositionOptions

Both `getCurrentPosition()` and `watchPosition()` accept the optional third parameter. It's an object of `PositionOptions`. There are three properties that we can configure:

- `enableHighAccuracy`: It's a `boolean` to indicate whether we'd like to get the best possible accuracy or not. Enabling this option may result in slower response time and increase power consumption.
- `timeout`: It's a `long` value in milliseconds that will limit the time for a device to return user's location. If the device hasn't returned the location within the specified time, it will throw a timeout error. By default, the value is `Infinity` which means it will never timeout.
- `maximumAge`: It's a `long` value in milliseconds representing the maximum age of a cached position. By default, it set to `0` and it means that we don't want the device to use a cached position.

Let's configure our `trackLocation()` function to enable the high accuracy and set a maximum `timeout` to 5 seconds.

The Final Touch

For the final touch, let’s integrate both the retrieved user’s coordinate and any error message into the UI. Open the `index.html` file and add a new `div` for displaying this information:

```
<main class="container">
    <div id="map" class="map"></div>
    <!-- For displaying user's coordinate or error message. -->
    <div id="info" class="info"></div>
</main>
```

Now open the `styles.css` file and add the following rules:

```
.info {
    padding: 1rem;
    margin: 0;
}

.info.error {
    color: #fff;
    background: #dc3545;
}
```

Finally, let’s update our `script.js` to print out the coordinate and the error into the `#info` div.

Now we should get a nice user’s coordinate and error at the bottom of the map.

You can check out the live demo and play around with it. You can also get the source for this tutorial on Github: google-maps-geolocation-example.

Credits:

- Globe by Nicole Harrington on Unsplash.

. . .

Originally published at risanb.com on March 16, 2018.

163 7