

1. Introduction

One of the most common features of a website is displaying a [Google Map](#) that highlights one or more locations for a business, institution, or any kind of entity with a physical presence. How these are implemented can vary greatly depending on requirements, such as the number of locations, and the frequency with which they change.

In this codelab, we'll look at the simplest use-case — a small number of locations that rarely change, like a store locator for a business with a chain of stores. In this case, we can use a relatively low-tech approach, without any server-side programming. But that's not to say we can't be creative, and we'll do so by leveraging the [GeoJSON](#) data format to store and render arbitrary information about each store on our map, as well as customizing the markers and overall style of the map itself.

Lastly, as an added bonus we'll use tools like [Google Cloud Shell](#) and [Google Cloud Storage](#) to develop and host our store locator. While using these tools isn't strictly required, doing so allows us to develop the store locator from any device running a web browser, and make it available online to the public.

What you'll build

In this codelab, you'll build a web page which will:

- Display a map with a set of store locations and information from a GeoJSON document
- Use custom markers based on the store's information
- Display extra information about the store when its marker is clicked

What you'll learn

How to populate a Google Map with a set of locations and data, using GeoJSON

How to customize the look of markers, info windows and the map itself

How to use Google Cloud services like Google Cloud Shell

What you'll need

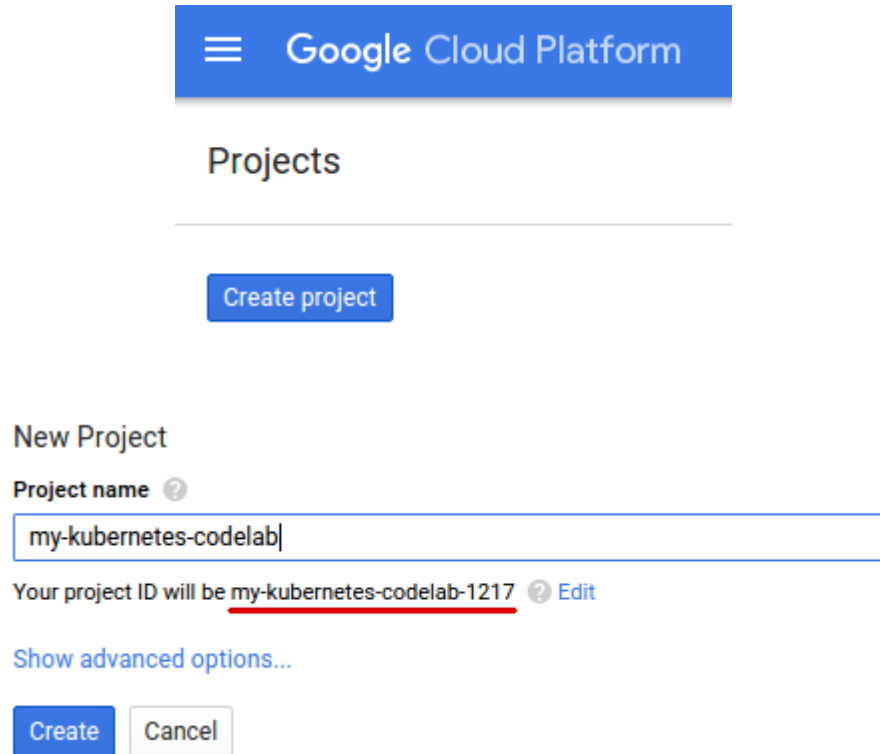
- Basic knowledge of HTML and JavaScript

2. Getting set up

Self-paced environment setup

If you don't already have a Google Account (Gmail or Google Apps), you must [create one](#).

Sign in to [Google Cloud Platform console](#) and create a new project:



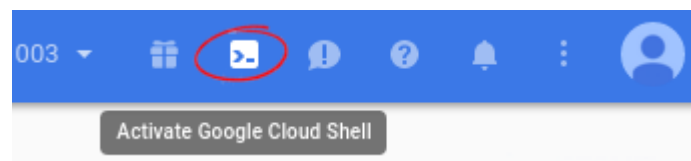
The screenshot shows the Google Cloud Platform console interface. At the top, there's a blue header with the Google Cloud Platform logo. Below it, the word 'Projects' is displayed. A blue button labeled 'Create project' is visible. The 'New Project' section contains a 'Project name' field with a question mark icon, where 'my-kubernetes-codelab' is entered. Below this, it states 'Your project ID will be my-kubernetes-codelab-1217' with a question mark icon and an 'Edit' link. A link 'Show advanced options...' is also present. At the bottom of the form are 'Create' and 'Cancel' buttons.

Remember the project ID, as you'll use it later. The project ID is a unique name across all Google Cloud projects. The name above has already been taken and will not work for you. Insert your own project ID wherever you see `YOUR_PROJECT_ID` in this codelab.

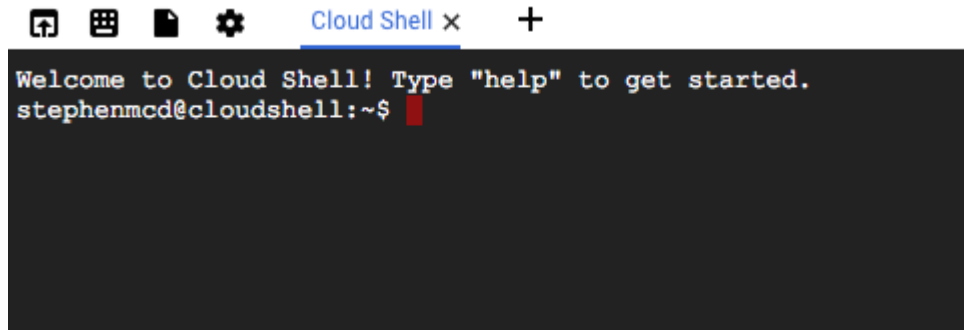
Start Cloud Shell

While Google Cloud can be operated remotely from your laptop, in this codelab you'll use [Google Cloud Shell](#), a command line environment running in the Cloud. Cloud Shell provides an automatically provisioned Linux virtual machine pre-loaded with all the development tools you need (`gcloud` , and more). It offers a persistent 5GB home directory, and runs on the Google Cloud, greatly enhancing network performance and authentication.

To activate Cloud Shell, from the Cloud platform console click the button on the top right-hand side (it should only take a few moments to provision and connect to the environment):



Clicking on the above button will open a new shell in the lower part of your browser, after possibly showing an introductory interstitial:



```
Welcome to Cloud Shell! Type "help" to get started.
stephenmcd@cloudshell:~$
```

Once connected to the Cloud Shell, you should see that you are already authenticated and that the project is already set to your `YOUR_PROJECT_ID` :

```
$ gcloud auth list
Credentialed accounts:
- <myaccount>@<mydomain>.com (active)
```

Note: `gcloud` is the powerful and unified command-line tool for Google Cloud Platform. Full documentation is available at <https://cloud.google.com/sdk/gcloud>. `gcloud` comes pre-installed on Cloud Shell and you will surely enjoy its support for tab-completion.

```
$ gcloud config list project
[core]
project = <YOUR_PROJECT_ID>
```

If for some reason the project is not set, run the following command:

```
$ gcloud config set project <YOUR_PROJECT_ID>
```

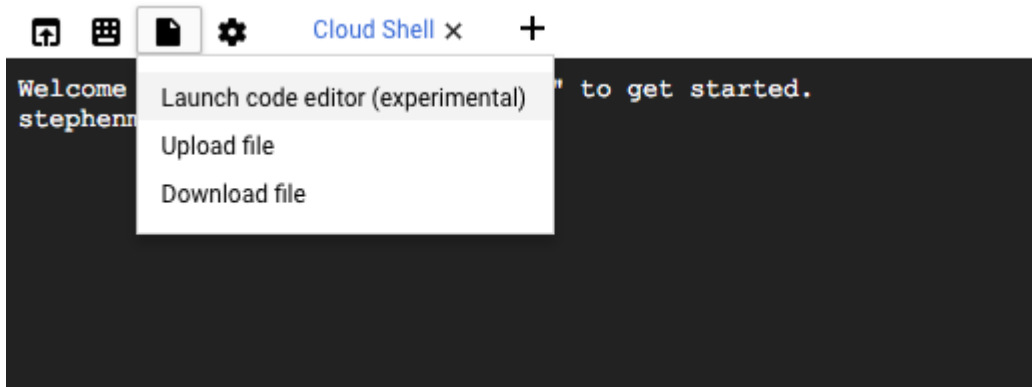
Note: When you run `gcloud` on your own machine, the config settings will persist across sessions. But in Cloud Shell, you need to set this for every new session or reconnection.

3. Hello World, with a Map

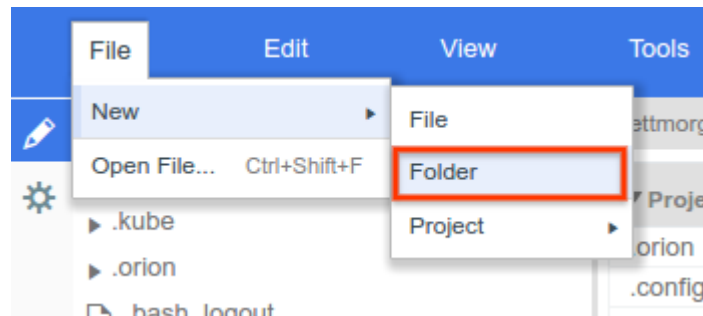
Starting developing with a **Map**

In your Cloud Shell instance, you'll start by creating an HTML page that will serve as the basis for the rest of the codelab.

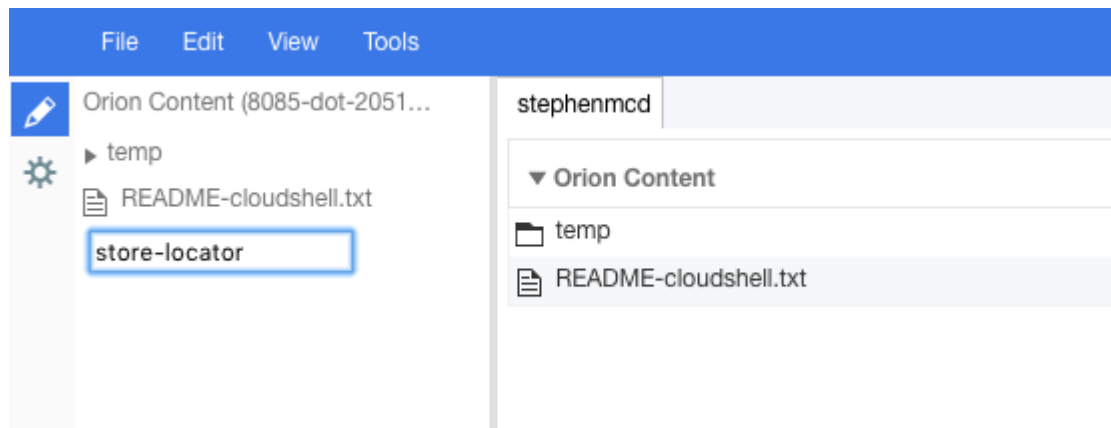
In the toolbar of the Cloud Shell, click on the file icon to open a menu, and then select **Launch code editor** to open a code editor in a new tab. This web based code editor allows you to easily edit files in the Cloud Shell instance.



Create a new `store-locator` directory for your application in the code editor, by opening the **File** menu, and selecting **New > Folder**.



Name the new folder `store-locator` :



Next you'll create a web page with a map.

Create a file in the `store-locator` directory named `index.html` . Put the following content in the `index.html` file:

index.html

```

<html>
<head>
  <title>Store Locator</title>
  <style>
    .map {height: 100%;}
    html, body {height: 100%; margin: 0; padding: 0;}
  </style>
</head>
<body>
  <div class="map"></div>

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

```

This is the HTML page that displays the map. It contains some CSS to ensure the map visually takes up the entire page, a `<div>` tag to hold the map, and a pair of `<script>` tags. The first script tag loads a JavaScript file called `app.js`, which contains all of the JavaScript code. The second script tag loads the API key, and specifies the name of the JavaScript function that runs once the JavaScript API is loaded, namely `initMap`.

Before we continue there is one quick diversion you need to make here, which is to replace the text `YOUR_API_KEY` in the above source with an actual Google Maps API key. Follow these steps to get an API key:

1. Go to the [Google API Console](#). (Use this link, because it has some smarts to enable the correct APIs for you.)
2. Select the project you created earlier in this code lab.
3. Click **Continue** to enable the API and any related services.
4. On the Credentials page, click **What credentials do I need?** then get the **API key**. Don't worry about adding restrictions to this API Key for the purposes of this codelab, but for a production store locator, you must restrict it to your website.
5. Copy the API Key and replace `YOUR_API_KEY` with it in the `index.html` file.

Lastly, let's create the `app.js` file with the following code:

app.js

```

function initMap() {

  // Create the map
  const map = new google.maps.Map(document.getElementsByClassName('map')[0], {
    zoom: 17,
    center: {
      lat: 51.512457,
      lng: -0.1291657
    }
  });
}

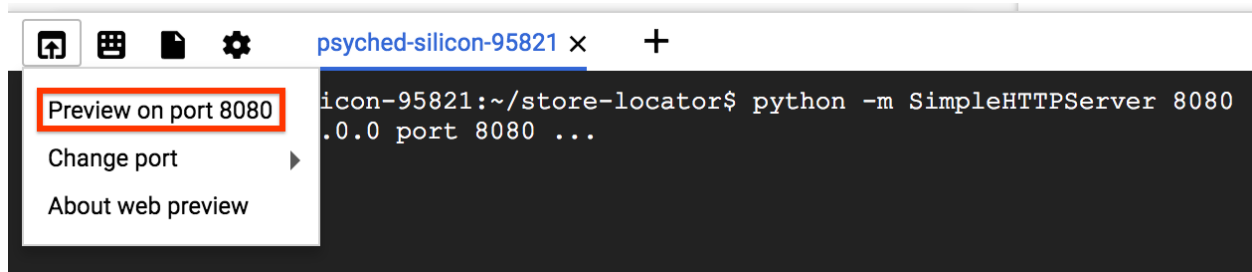
```

That's the minimum required code for creating a map. We pass in a reference to our `<div>` tag to hold the map, and specify the center and zoom level.

To test this application, you can run the simple Python HTTP server inside the Cloud Shell instance. Go back to the Cloud Shell command line, and type the following:

```
$ cd store-locator  
$ python -m SimpleHTTPServer 8080
```

You will see some lines of log output showing you that you are indeed running the simple HTTP server on the Cloud Shell instance, with the web app listening on localhost port 8080. You can open a web browser tab on this app by selecting the Preview on port 8080 menu item from the Cloud Shell toolbar.



Clicking on this menu item will open a new tab in your web browser with the content of the HTML served from the simple Python HTTP server. If everything went well, you should see a map centered on London, England. To stop the simple HTTP server, type CTRL+C.

4. Hosting with Google Cloud Storage

Now let's look at using [Google Cloud Storage](#) to host our web page. Google Cloud Storage is an online file storage web service for storing and accessing data on Google's infrastructure. The service combines the performance and scalability of Google's cloud with advanced security and sharing capabilities. It also offers a [free tier](#), which makes it great for hosting our simple store locator.

With Google Cloud Storage, files are stored in buckets, which are similar to directories on your computer. To host our web page, we first need to create a bucket. You'll need to choose a unique name for your bucket, perhaps by using your name as part of the bucket name. Once you decide on a name, run the following command back in your Cloud Shell instance:

```
$ gsutil mb gs://yourname-store-locator
```

`gsutil` is the tool for interacting with Google Cloud Storage. The `mb` command creatively stands for "make bucket". For more information on all of the commands available including the ones we'll use, see the [Cloud Storage Documentation](#).

By default, your buckets and files hosted on Google Cloud Storage are private. For our store locator however, we'll want all the files to be public, so that they're accessible to everyone over the internet. We could make each file public after we upload it, but that would be tedious. Instead we can simply set the default access level for the bucket we created, and all of the files we upload to it will inherit that access level. Run the following command, replacing `yourname-store-locator` with the name you chose for your bucket:

```
$ gsutil defacl ch -u AllUsers:R gs://yourname-store-locator
```

Now we can upload all our files in the current directory (currently just our `index.html` and `app.js` files) with the following command:

```
$ gsutil -h "Cache-Control:no-cache" cp * gs://yourname-store-locator
```

(Note that the `-h "Cache-Control:no-cache"` part of the above command prevents the files from being cached, so that we can view the latest version each time we upload them during development. For your production store locator, you would omit this.)

Voila! You should now have a web page with a Google Map online. The URL to view it will be `http://storage.googleapis.com/yourname-store-locator/index.html` - again with the `yourname-store-locator` part replaced with the bucket name you previously chose.

5. Populating the Map with GeoJSON

Let's now take a look at the data for the stores. [GeoJSON](#) is a data format that represents simple geographical features, like points, lines or polygons on a map. The features can also contain arbitrary data. This makes GeoJSON an excellent candidate for representing the stores, which are essentially points on a map with a bit of additional data, like the store's name, opening hours and phone number. Most importantly, GeoJSON has first-class support in Google Maps, which means we can send a GeoJSON document to a Google Map, and it will render it on the map appropriately.

Create a new file called `stores.json` and paste in the following code:

`stores.json`

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "geometry": {
        "type": "Point",
        "coordinates": [
          -0.145365,
          51.506182
        ]
      },
      "type": "Feature",
      "properties": {
        "category": "patisserie",
        "hours": "10am - 6pm",
        "description": "Modern twists on classic pastries. We're part of a larger chain of patisseries and cafes.",
        "name": "Josie's Patisserie Mayfair",
        "phone": "+44 20 1234 5678"
      }
    },
    {
      "geometry": {
        "type": "Point",
        "coordinates": [
          -2.579623,
          51.452251
        ]
      },
      "type": "Feature",
      "properties": {
        "category": "patisserie",
        "hours": "10am - 6pm",
        "description": "Come and try our award-winning cakes and pastries. We're part of a larger chain of patisseries and cafes.",
        "name": "Josie's Patisserie Bristol",
        "phone": "+44 117 121 2121"
      }
    },
    {
      "geometry": {
        "type": "Point",
        "coordinates": [
          1.273459,
          52.638072
        ]
      },
      "type": "Feature",
      "properties": {
        "category": "patisserie",
        "hours": "10am - 6pm",
        "description": "Come and try our award-winning cakes and pastries. We're part of a larger chain of patisseries and cafes.",
        "name": "Josie's Patisserie London",
        "phone": "+44 20 1234 5678"
      }
    }
  ]
}
```



```

    "type": "Feature",
    "properties": {
      "category": "patisserie",
      "hours": "10am - 6pm",
      "description": "Whatever the occasion, whether it's a birthday or a wedding, Josie's Patisserie has the perfect treat for you. We're part of a larger chain of patisseries and cafes.",
      "name": "Josie's Patisserie Norwich",
      "phone": "+44 1603 123456"
    }
  },
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -1.882509,
        50.848831
      ]
    },
    "type": "Feature",
    "properties": {
      "category": "patisserie",
      "hours": "10am - 6pm",
      "description": "A gourmet patisserie that will delight your senses. We're part of a larger chain of patisseries and cafes.",
      "name": "Josie's Patisserie Wimborne",
      "phone": "+44 1202 343434"
    }
  },
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -2.985933,
        53.408899
      ]
    },
    "type": "Feature",
    "properties": {
      "category": "patisserie",
      "hours": "10am - 6pm",
      "description": "Spoil yourself or someone special with our classic pastries. We're part of a larger chain of patisseries and cafes.",
      "name": "Josie's Patisserie Liverpool",
      "phone": "+44 151 444 4444"
    }
  },
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -1.689423,
        52.632469
      ]
    },
    "type": "Feature",
    "properties": {
      "category": "patisserie",
      "hours": "10am - 6pm",
      "description": "Come and feast your eyes and tastebuds on our delicious pastries and cakes. We're part of a larger chain of patisseries and cafes.",
      "name": "Josie's Patisserie Tamworth",
      "phone": "+44 5555 55555"
    }
  }
]

```

```

    }
  },
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -3.155305,
        51.479756
      ]
    },
    "type": "Feature",
    "properties": {
      "category": "patisserie",
      "hours": "10am - 6pm",
      "description": "Josie's Patisserie is family-owned, and our delectable pastries, cakes, and great coffee are renowned. We're part of a larger chain of patisseries and cafes.",
      "name": "Josie's Patisserie Cardiff",
      "phone": "+44 29 6666 6666"
    }
  },
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.725019,
        52.668891
      ]
    },
    "type": "Feature",
    "properties": {
      "category": "cafe",
      "hours": "8am - 9:30pm",
      "description": "Oakham's favorite spot for fresh coffee and delicious cakes. We're part of a larger chain of patisseries and cafes.",
      "name": "Josie's Cafe Oakham",
      "phone": "+44 7777 777777"
    }
  },
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -2.477653,
        53.735405
      ]
    },
    "type": "Feature",
    "properties": {
      "category": "cafe",
      "hours": "8am - 9:30pm",
      "description": "Enjoy freshly brewed coffee, and home baked cakes in our homely cafe. We're part of a larger chain of patisseries and cafes.",
      "name": "Josie's Cafe Blackburn",
      "phone": "+44 8888 88888"
    }
  },
  {
    "geometry": {
      "type": "Point",
      "coordinates": [
        -0.211363,

```

```

51.108966
]
},
"type": "Feature",
"properties": {
  "category": "cafe",
  "hours": "8am - 9:30pm",
  "description": "A delicious array of pastries with many flavours, and fresh coffee in an snug cafe. We're part of a larger chain of
patisseries and cafes.",
  "name": "Josie's Cafe Crawley",
  "phone": "+44 1010 101010"
}
},
{
  "geometry": {
    "type": "Point",
    "coordinates": [
      -0.123559,
      50.832679
    ]
  },
  "type": "Feature",
  "properties": {
    "category": "cafe",
    "hours": "8am - 9:30pm",
    "description": "Grab a freshly brewed coffee, a decadent cake and relax in our idyllic cafe. We're part of a larger chain of
patisseries and cafes.",
    "name": "Josie's Cafe Brighton",
    "phone": "+44 1313 131313"
  }
},
{
  "geometry": {
    "type": "Point",
    "coordinates": [
      -3.319575,
      52.517827
    ]
  },
  "type": "Feature",
  "properties": {
    "category": "cafe",
    "hours": "8am - 9:30pm",
    "description": "Come in and unwind at this idyllic cafe with fresh coffee and home made cakes. We're part of a larger chain of
patisseries and cafes.",
    "name": "Josie's Cafe Newtown",
    "phone": "+44 1414 141414"
  }
},
{
  "geometry": {
    "type": "Point",
    "coordinates": [
      1.158167,
      52.071634
    ]
  },
  "type": "Feature",
  "properties": {
    "category": "cafe",
    "hours": "8am - 9:30pm",

```

```

    "description": "Fresh coffee and delicious cakes in an snug cafe. We're part of a larger chain of patisseries and cafes.",
    "name": "Josie's Cafe Ipswich",
    "phone": "+44 1717 17171"
  }
}
]
}

```

That's a lot of data, but once you peruse it, you'll see it's simply the same structure repeated for each store. Each store is represented as a GeoJSON Point along with its coordinates, and the extra data contained under the `properties` key. Interestingly, GeoJSON allows the inclusion of arbitrarily named keys under the `properties` key. In this codelab, those keys are `category`, `hours`, `description`, `name`, and `phone`.

Now we'll edit `app.js` so that it loads the GeoJSON in `stores.js` onto our map:

app.js

```

function initMap() {

  // Create the map.
  const map = new google.maps.Map(document.getElementsByClassName('map')[0], {
    zoom: 7,
    center: {lat: 52.632469, lng: -1.689423} });

  // Load the stores GeoJSON onto the map.
  map.data.loadGeoJson('stores.json');

  const infoWindow = new google.maps.InfoWindow();

  // Show the information for a store when its marker is clicked.
  map.data.addListener('click', event => {

    let name = event.feature.getProperty('name');
    let description = event.feature.getProperty('description');
    let hours = event.feature.getProperty('hours');
    let phone = event.feature.getProperty('phone');
    let position = event.feature.getGeometry().get();
    let content = `
      <h2>${name}</h2><p>${description}</p>
      <p><b>Open:</b> ${hours}<br/><b>Phone:</b> ${phone}</p>
    `;

    infoWindow.setContent(content);
    infoWindow.setPosition(position);
    infoWindow.setOptions({pixelOffset: new google.maps.Size(0, -30)});
    infoWindow.open(map);
  });
}

```

In the above code example, we load our GeoJSON onto the map by calling `loadGeoJson` and passing the name of the JSON file. We also define a function to run each time a marker is clicked. The function can then access the extra data for the store whose marker was clicked, and use the information in an info window that is displayed. To test this application, you can run the simple Python HTTP server using the same command we used before. Go back to the Cloud Shell command line, and type the following:

```
python -m SimpleHTTPServer 8080
```

You should see a map full of markers that you can click to view details about each store. Progress!

6. Customizing the Map

We're almost there — we have a map with all our store markers, and extra information being displayed when clicked. But it looks like every other Google Map out there, how dull! Let's spice it up with a custom map style, markers, logos and Google Street View images.

Here's the final version of `app.js`:

`app.js`

```
const mapStyle = [
  {
    "featureType": "administrative",
    "elementType": "all",
    "stylers": [
      {
        "visibility": "on"
      },
      {
        "lightness": 33
      }
    ]
  },
  {
    "featureType": "landscape",
    "elementType": "all",
    "stylers": [
      {
        "color": "#f2e5d4"
      }
    ]
  },
  {
    "featureType": "poi.park",
    "elementType": "geometry",
    "stylers": [
      {
        "color": "#c5dac6"
      }
    ]
  },
  {
    "featureType": "poi.park",
    "elementType": "labels",
    "stylers": [
      {
        "visibility": "on"
      },
      {
        "lightness": 20
      }
    ]
  },
  {
    "featureType": "road",
    "elementType": "all",
    "stylers": [
      {
```

```

    "lightness": 20
  }
]
},
{
  "featureType": "road.highway",
  "elementType": "geometry",
  "stylers": [
    {
      "color": "#c5c6c6"
    }
  ]
},
{
  "featureType": "road.arterial",
  "elementType": "geometry",
  "stylers": [
    {
      "color": "#e4d7c6"
    }
  ]
},
{
  "featureType": "road.local",
  "elementType": "geometry",
  "stylers": [
    {
      "color": "#fbfaf7"
    }
  ]
},
{
  "featureType": "water",
  "elementType": "all",
  "stylers": [
    {
      "visibility": "on"
    },
    {
      "color": "#acbcc9"
    }
  ]
}
];

function initMap() {

  // Create the map.
  const map = new google.maps.Map(document.getElementsByClassName('map')[0], {
    zoom: 7,
    center: {lat: 52.632469, lng: -1.689423},
    styles: mapStyle
  });

  // Load the stores GeoJSON onto the map.
  map.data.loadGeoJson('stores.json');

  // Define the custom marker icons, using the store's "category".
  map.data.setStyle(feature => {
    return {
      icon: {

```

```

    url: `img/icon_${feature.getProperty('category')}.png`,
    scaledSize: new google.maps.Size(64, 64)
  }
};
});

const apiKey = 'YOUR_API_KEY';
const infoWindow = new google.maps.InfoWindow();

// Show the information for a store when its marker is clicked.
map.data.addListener('click', event => {

  let category = event.feature.getProperty('category');
  let name = event.feature.getProperty('name');
  let description = event.feature.getProperty('description');
  let hours = event.feature.getProperty('hours');
  let phone = event.feature.getProperty('phone');
  let position = event.feature.getGeometry().get();
  let content = `
    
    <div style="margin-left:220px; margin-bottom:20px;">
      <h2>${name}</h2><p>${description}</p>
      <p><b>Open:</b> ${hours}<br/><b>Phone:</b> ${phone}</p>
      <p></p>
    </div>
  `;

  infoWindow.setContent(content);
  infoWindow.setPosition(position);
  infoWindow.setOptions({pixelOffset: new google.maps.Size(0, -30)});
  infoWindow.open(map);
});
}

```

Here's what we've added:

- The `mapStyle` variable contains all the information for styling the map. (As an added bonus, you can even [create your own style](#) if you like.)
- Using the `map.data.setStyle` method, we apply custom markers - a different one for each `category` from the GeoJSON.
- We've modified the `content` variable to include a logo (again using the `category` from the GeoJSON), and a Google Street View image for the store's location.

Before we deploy this, a couple of steps:

- Set the correct value for the `apiKey` variable by replacing the `'YOUR_API_KEY'` string in `app.js` with your own API key from earlier (leaving the quotes intact).
- Run the following command in Cloud Shell to download the marker and logo graphics (make sure you're in the `store-locator` directory; use CTRL+C to stop the simple HTTP server if it's running):

```
$ mkdir -p img; wget https://storage.googleapis.com/gmaps-store-locator/assets.zip -O temp.zip; unzip temp.zip -d img; rm temp.zip
```

Now we can preview the finished store locator:

```
$ python -m SimpleHTTPServer 8080
```


7. Congratulations!

Cleanup

The easiest way to clean up all the resources created in this project is to [shut down the Google Cloud Project](#) that you created at the start of this tutorial:

- Open the [Settings Page](#) in the Google Cloud Platform Console
- Click **Select a project**.
- Select the project you created at the start of this tutorial.
- Click **Delete Project**.
- Enter the Project ID and click **Shut down**.

Congratulations, you have successfully completed this codelab! If you enjoyed this codelab, but would like to dive into the code some more, please have a look at our source code repository at <https://github.com/googlecodelabs/google-maps-simple-store-locator>.