Introduction

Learn to use Google Maps Platform's Maps and Places APIs to build a local business search, which geolocates the user and shows interesting places around them. The app integrates location, place details, place photos, and more.

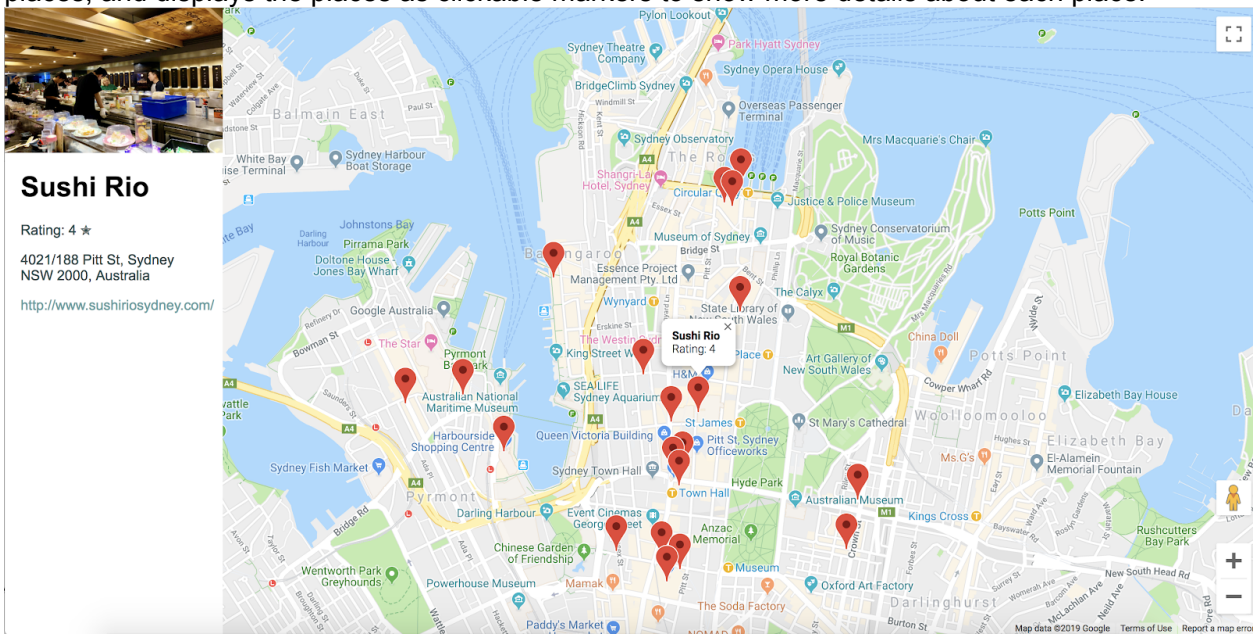
What is Google Maps Platform?

Google Maps Platform brings the wealth of Google's location-based information to your app. Millions of developers use Google Maps Platform to help their users navigate the world around them, on the web and on mobile devices. The Google Maps Platform offers three core products:

- **Maps** enables you to build customized, agile experiences that bring the real world to your users with static and dynamic maps, and 360° Street View imagery.
- **Places** helps your users discover the world with rich location data for over 150 million places by using phone numbers, addresses, and real-time signals.
- **Routes** gives your users the best way to get from A to Z with high-quality directions that factor in real-time traffic updates. Determine the route a vehicle travels to create more precise itineraries.

What you'll build

In this lab, you're going to build a webpage that displays a Google map centered on the user's location, finds nearby places, and displays the places as clickable markers to show more details about each place.



What you'll learn

How to create a customizable map

How to geolocate the user

How to search for nearby places and display the results

How to fetch and display details about a place

What you'll need

- A web browser. Choose a well-known one like Google Chrome (recommended), Firefox, Safari or Internet Explorer, based on your platform
- Your favorite text or code editor

- Basic knowledge of HTML, CSS, and JavaScript

Get the Sample Code

1. Open your command line interface (Terminal on Mac, Command Prompt on Windows) and download the sample code with this command:

```
git clone https://github.com/googlecodelabs/google-maps-nearby-search-js/
```

If that doesn't work, click the following button to download all the code for this codelab, then unzip the file:

Download the code

2. Change into the directory you just cloned or downloaded:

```
cd google-maps-nearby-search-js
```

The `stepN` folders contain the desired end state of each step of this codelab. They are there for reference. Do all your coding work in the directory called `work`.

This lab is focused on using the Google Maps JavaScript API and its Places Library. Non-relevant concepts and code blocks are glossed over and are provided for you to simply copy and paste.

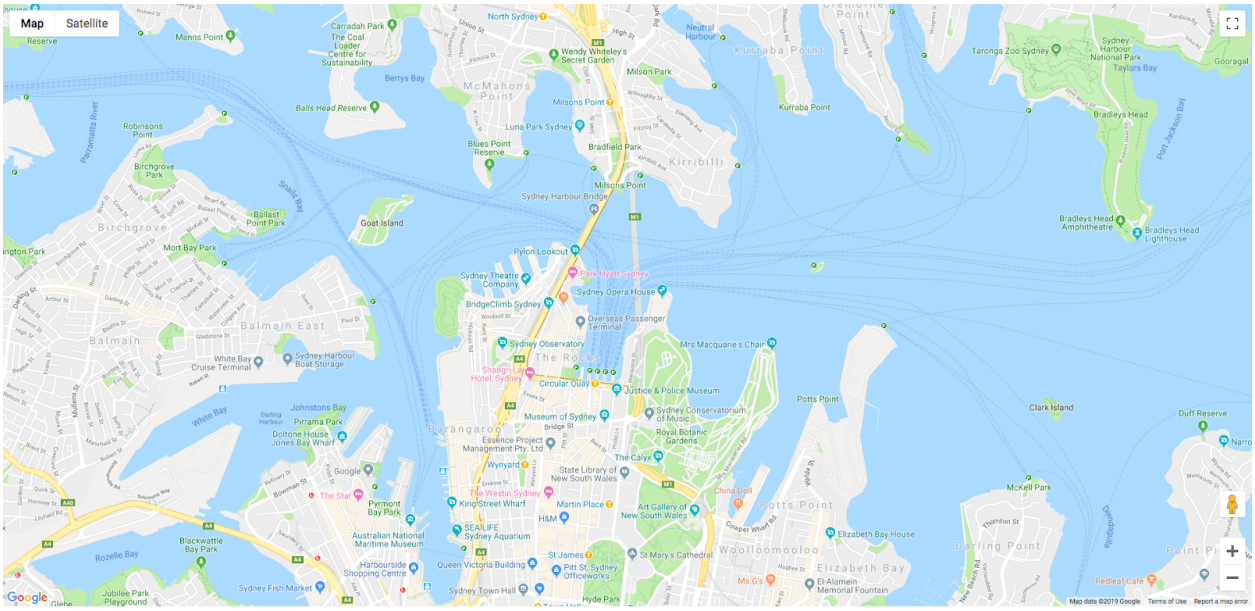
Step 1: Create a Map with a Default Center

There are three steps to creating a Google map on your web page:

1. Create an HTML page
2. Add a map
3. Get an API key

A. **Create an HTML page**

Below is the map you'll create for this step. The map is centered on the Sydney Opera House in Sydney, Australia. If the user denies permission to get their location, the map will default to this location and still be able to provide interesting search results.



1. Change directories into your work/ folder. Throughout the rest of the lab, make your edits in the version in the work/ folder.

```
cd work
```

2. In the work/ directory, use your text editor to create a blank file called index.html .
3. Copy the code below into index.html .

index.html

```
<!DOCTYPE html>
<html>

<head>
  <title>Sushi Finder</title>
  <meta name="viewport" content="initial-scale=1.0, user-scalable=no">
  <meta charset="utf-8">
```

```
  <style>
   /* Always set the map height explicitly to define the size of the div
    * element that contains the map. */
   #map {
     height: 100%;
     background-color: grey;
   }

   /* Optional: Makes the sample page fill the window. */
   html,
   body {
     height: 100%;
     margin: 0;
     padding: 0;
   }

   /* TODO: Step 4A1: Make a generic sidebar */
  </style>
 </head>

 <body>
  <!-- TODO: Step 4A2: Add a generic sidebar -->

  <!-- Map appears here -->
  <div id="map"></div>

  <!-- TODO: Step 1B, Add a map -->
 </body>

 </html>
```

Note that this is a very basic page with a single div element and a style element. You can add any content you like to the web page. You should see a large div with a grey background spanning the full visible window.

4. Open the file index.html in your web browser.

```
open index.html
```

> From this point forward, all testing/verification (e.g. the **Test It Out** sections in subsequent steps) will involve you reloading this page in your browser to see the changes you've made.

B. **Add a map**

This section shows you how to load the Maps JavaScript API into your web page, and how to write your own JavaScript that uses the API to add a map to the webpage.

Add this script code where you see <!-- TODO: Step 1B, Add a map --> after the map div and before the close </body> tag.

step1/index.html

```
<!-- TODO: Step 1B, Add a map -->
<script>
  /* Note: This example requires that you consent to location sharing when
```

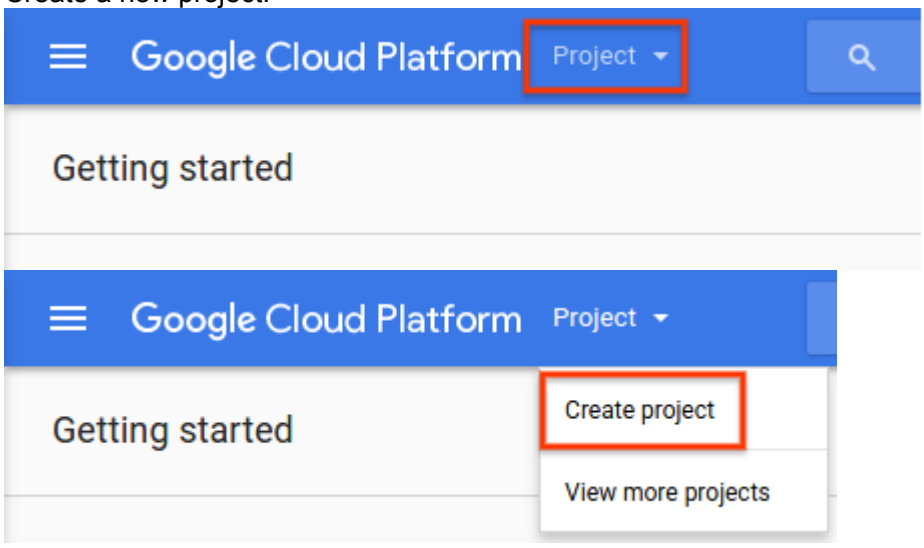
```
      * prompted by your browser. If you see the error "Geolocation permission
      * denied.", it means you probably did not give permission for the browser * to locate you. */

    /* TODO: Step 2, Geolocate your user
     * Replace the code from here to the END TODO comment with new code from
     * codelab instructions. */
    let pos;
    let map;
    function initMap() {
      // Set the default location and initialize all variables
      pos = {lat: -33.857, lng: 151.213};
      map = new google.maps.Map(document.getElementById('map'), {
        center: pos,
        zoom: 15
      });
    }
    /* END TODO: Step 2, Geolocate your user */
  </script>

  <!-- TODO: Step 1C, Get an API key -->
  <!-- TODO: Step 3A, Load the Places Library -->
  <script async defer src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap">
  </script>
```

C. Enable Google Maps Platform APIs and get an API key

This section explains how to authenticate your app to the Maps JavaScript API and Places API using your own API key.

Go to the Google Cloud Platform Console.

Create a new project.



When you create a new project, you're prompted to choose which of your billing accounts you want to link to the project. If you have a single billing account, that account is automatically linked to your project.

> Google offers a 12 month free trial for up to $300 worth of Google Cloud Platform. Find out more details at https://cloud.google.com/free/. In addition, accounts that use Maps Platform APIs receive $200 in credits applied to Maps Platform API usage every month.

If you don't have a billing account, you must create one and enable billing for your project before you can use many Google Cloud Platform features. To create a new billing account:

1. Open the Google Cloud Platform Console navigation menu and select Billing.
2. Click the **Add billing account** button. (If this is not your first billing account, click the Create account button)
3. Enter the name of the billing account and enter your billing information. The options you see depend on the country of your billing address. Note that for United States accounts, you cannot change tax status after the account is created.
4. Click **Submit and enable billing**.
5. By default, the person who creates the billing account is a billing administrator for the account.
6. For information about verifying bank accounts and adding backup methods of payment, see Add, remove, or update a payment method.

Follow these steps to enable the APIs used in this lab and get an API key:

1. From the Navigation menu, select **APIs & Services > Library**.
2. Click the Maps JavaScript API tile and click the Enable button.
3. Repeat steps 1 and 2 to enable the Places API.
4. From the Navigation menu, select **APIs & Services > Credentials**.
5. Click **Create Credentials** and choose **API key**.
6. In the line after <!-- TODO: Step 1C, Get an API key --> , copy and replace the value of the key parameter in the script source URL with the API key you just created.

step1/index.html

```
<!-- TODO: Step 1C, Get an API key -->
<!-- TODO: Step 3A, Load the Places Library -->
<script async defer src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap">
</script>
```

7. Save the html file you've been working on.

Tip: To prevent quota theft, secure your API key. See Protecting API keys for best practices and Restrict an API key for specific settings to edit in your Google Cloud Platform Console.

Note: Understand billing, including the number of API calls that can be made within the monthly Google Maps Platform credit on the Maps JavaScript API Usage and Billing page.

Test It Out

Reload your browser view of the file you've been editing. You should see a map appear now where the grey rectangle was before. If you see an error message instead, make sure you have replaced " YOUR_API_KEY " in the final <script> tag with your own API key. See above for how to get an API key if you don't already have one.

Full Sample Code

The full code for this project through Step 1 is available on Github.

Step 2: Geolocate your user

Next, you'll want to display the geographic location of the user or device on a Google map, using your browser's HTML5 Geolocation feature along with the Maps JavaScript API.

Here is an example of a map that displays your geographic location if you were browsing from Mountain View, CA:

What is Geolocation?

Geolocation refers to the identification of the geographic location of a user or computing device via a variety of data collection mechanisms. Typically, most geolocation services use network routing addresses or internal GPS devices to determine this location. This application uses the web browser's W3C Geolocation standard navigator.geolocation property to determine the user's location.

Try it yourself

Replace the code between the comment TODO: Step 2, Geolocate your user and END TODO: Step 2, Geolocate your user with the code below:

step2/index.html

```
/* TODO: Step 2, Geolocate your user
   * Replace the code from here to the END TODO comment with this code
   * from codelab instructions. */
let pos;
let map;
let bounds;
let infoWindow;
let currentInfoWindow;
let service;
let infoPane;
function initMap() {
  // Initialize variables
  bounds = new google.maps.LatLngBounds();
  infoWindow = new google.maps.InfoWindow;
  currentInfoWindow = infoWindow;
  /* TODO: Step 4A3: Add a generic sidebar */
```

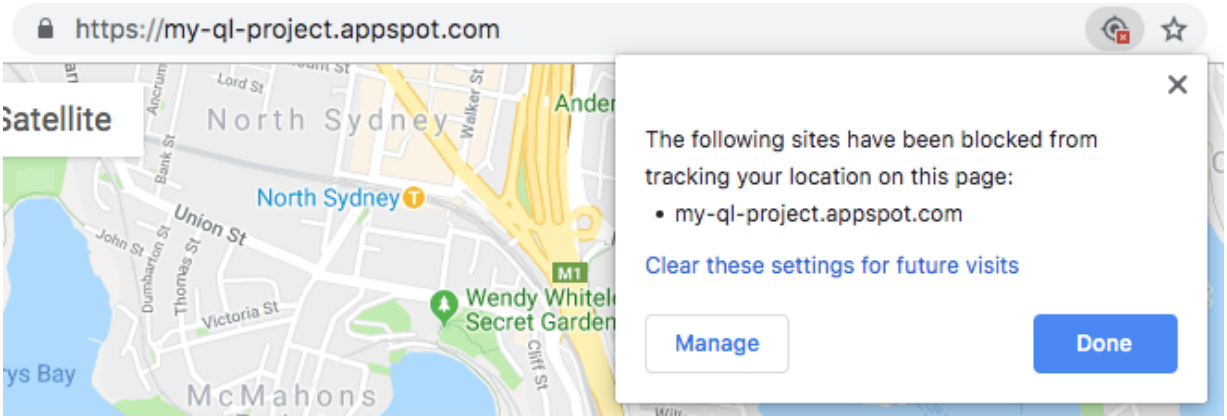
```
    // Try HTML5 geolocation
    if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(position => {
        pos = {
        lat: position.coords.latitude,
        lng: position.coords.longitude
        };
        map = new google.maps.Map(document.getElementById('map'), {
        center: pos,
        zoom: 15
        });
        bounds.extend(pos);

        infoWindow.setPosition(pos);
        infoWindow.setContent('Location found.');
        infoWindow.open(map);
        map.setCenter(pos);

        /* TODO: Step 3B2, Call the Places Nearby Search */
    }, () => {
        // Browser supports geolocation, but user has denied permission
        handleLocationError(true, infoWindow);
    });
    } else {
    // Browser doesn't support geolocation
    handleLocationError(false, infoWindow);
    }
}

// Handle a geolocation error
function handleLocationError(browserHasGeolocation, infoWindow) {
    // Set default location to Sydney, Australia
    pos = {lat: -33.856, lng: 151.215};
    map = new google.maps.Map(document.getElementById('map'), {
    center: pos,
    zoom: 15
    });

    // Display an InfoWindow at the map center
    infoWindow.setPosition(pos);
    infoWindow.setContent(browserHasGeolocation ?
    'Geolocation permissions denied. Using default location.' :
    'Error: Your browser doesn\'t support geolocation.');
    infoWindow.open(map);
    currentInfoWindow = infoWindow;

    /* TODO: Step 3B3, Call the Places Nearby Search */
}
/* END TODO: Step 2, Geolocate your user */
/* TODO: Step 3B1, Call the Places Nearby Search */
```

Test it out

Save your file. Now, if you reload your page, your browser should now ask you for permissions to share your location with the app.

- Try clicking "Block" one time to see if it handles the error gracefully and remains centered on Sydney.
- Reload again and try clicking "Allow" to see if the geolocation works and moves the map to your current location.

Note: Your browser might save the last setting of permissions you granted to this page. In order to choose a different permission (allow when you previously blocked, or block when you previously allowed), you may need to clear your browser settings. In Chrome, there is a location icon at the right side of the address bar. Clicking the icon brings up a window that describes your current setting and allows you to clear that setting for future visits. Click "Clear these settings for future visits" any time you want to test handling of different block/allow permissions of geolocation.



Full Sample Code

The full code for this project through Step 2 is available on Github.

Step 3: Search for Nearby Places

A Nearby Search lets you search for places within a specified area by keyword or type. A Nearby Search must always include a location, which can be specified in one of two ways:

- a `LatLngBounds` object defining a rectangular search area.
- a circular area defined as the combination of the `location` property — specifying the center of the circle as a `LatLng` object — and a radius, measured in meters.

Initiate a Nearby Search with a call to the `PlacesService` 's `nearbySearch()` method, which will return an array of `PlaceResult` objects.

A. **Load the Places Library**

First, to access the Places Library services, update your script source URL to introduce the `libraries` parameter and add `places` as a value.

step3/index.html

```
<!-- TODO: Step 3A, Load the Places Library -->
<script async defer
  src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries=places&callback=initMap">
```

B. Call the Places Nearby Search request and handle the response

Next, form a PlaceSearch Request. The minimum required fields are:

Either of:

- `bounds` , which must be a `google.maps.LatLngBounds` object defining the rectangular search area; or
- a `location` and a `radius` ; the former takes a `google.maps.LatLng` object, and the latter takes a simple integer, representing the circle's radius in meters. The maximum allowed radius is 50 000 meters. Note that when `rankBy` is set to `DISTANCE` , you must specify a location but you cannot specify a radius or bounds.

end either of:

- A `keyword` - A term to be matched against all available fields, including but not limited to name, type, and address, as well as customer reviews and other third-party content.
- A `type` - Restricts the results to places matching the specified type. Only one type may be specified (if more than one type is provided, all types following the first entry are ignored). See the list of supported types.

For this lab, you will use the user's current position as the location for the search and rank results by distance.

1. Add the following at the comment `TODO: Step 3B1` to write two functions to call the search and handle the response. The keyword `sushi` is used as the search term, but you can change it. The code to define the `createMarkers` function is provided in the next section, titled "Generate markers for search results".

step3/index.html

```
/* TODO: Step 3B1, Call the Places Nearby Search */
// Perform a Places Nearby Search Request
function getNearbyPlaces(position) {
  let request = {
  location: position,
  rankBy: google.maps.places.RankBy.DISTANCE,
```

```
    keyword: 'sushi'
    };

    service = new google.maps.places.PlacesService(map);
    service.nearbySearch(request, nearbyCallback);
}

// Handle the results (up to 20) of the Nearby Search
function nearbyCallback(results, status) {
    if (status == google.maps.places.PlacesServiceStatus.OK) {
    createMarkers(results);
    }
}

/* TODO: Step 3C, Generate markers for search results */
```

2. Add this line in to the end of the initMap function at the comment TODO: Step 3B2 .

```
/* TODO: Step 3B2, Call the Places Nearby Search */
// Call Places Nearby Search on user's location
getNearbyPlaces(pos);
```

3. Add this line at the end of the handleLocationError function at the comment TODO: Step 3B3 .

```
/* TODO: Step 3B3, Call the Places Nearby Search */
// Call Places Nearby Search on the default location
getNearbyPlaces(pos);
```

C. Generate markers for search results

A marker identifies a location on a map. By default, a marker uses a standard image. See the Markers guidefor information about customizing marker images.

The google.maps.Marker constructor takes a single Marker options object literal, specifying the initial properties of the marker.

The following fields are particularly important and commonly set when constructing a marker:

- position (required) specifies a LatLng identifying the initial location of the marker.
- map (optional) specifies the Map on which to place the marker. If you do not specify the map on construction of the marker, the marker is created but is not attached to (or displayed on) the map. You may add the marker later by calling the marker's setMap() method.

Add the code below at the comment TODO: Step 3C to set the position, map, and title for one marker per place returned in the response. You also use the extend method of the bounds variable to ensure that the center and all the markers are visible on the map.

step3/index.html

```
/* TODO: Step 3C, Generate markers for search results */
// Set markers at the location of each place result
function createMarkers(places) {
    places.forEach(place => {
    let marker = new google.maps.Marker({
        position: place.geometry.location,
```

```
            map: map,
            title: place.name
        });

        /* TODO: Step 4B: Add click listeners to the markers */

        // Adjust the map bounds to include the location of this marker
        bounds.extend(place.geometry.location);
        });
        /* Once all the markers have been placed, adjust the bounds of the map to
        * show all the markers within the visible area. */
        map.fitBounds(bounds);
    }

    /* TODO: Step 4C: Show place details in an info window */
```

Test it out

- Save and Reload the page and click "Allow" to give geolocation permissions.
- You should see up to twenty red markers around the center location of the map.
- Reload again and block geolocation permissions this time. Do you still get results at the default center of your map (in our sample, the default is in Sydney, Australia)?

Full sample code

The full code for this project through Step 3 is available on Github.

Step 4: Show Place Details on Demand

Once you have a place's Place ID (delivered as one of the fields in the results of your nearby search), you can request additional details about the place, such as its complete address, phone number, user rating and reviews, etc. In this lab, we'll make a sidebar to display rich place detail information and make the markers interactive so the user can select places to view details.

A. Make a generic sidebar

We need a place to display the place details, so here's some simple code for a sidebar that we can use to slide out and display the place details when the user clicks on a marker.

1. Add these to the `style` tag at the comment `TODO: Step 4A1` :

step4/index.html

```
/* TODO: Step 4A1: Make a generic sidebar */
/* Styling for an info pane that slides out from the left.
   * Hidden by default. */
#panel {
    height: 100%;
    width: null;
    background-color: white;
    position: fixed;
    z-index: 1;
    overflow-x: hidden;
    transition: all .2s ease-out;
}

.open {
    width: 250px;
}

/* Styling for place details */
.hero {
    width: 100%;
    height: auto;
    max-height: 166px;
    display: block;
}

.place,
p {
    font-family: 'open sans', arial, sans-serif;
    padding-left: 18px;
    padding-right: 18px;
}

.details {
    color: darkslategrey;
}

a {
    text-decoration: none;
    color: cadetblue;
}
```

2. In the body section just before the map div, add a div for the details panel:

```
<!-- TODO: Step 4A2: Add a generic sidebar -->
<!-- The slide-out panel for showing place details -->
<div id="panel"></div>
```

3. In the initMap() function at the TODO: Step 4A3 comment, initialize the infoPane variable like this:

```
/* TODO: Step 4A3: Add a generic sidebar */
infoPane = document.getElementById('panel');
```

B. Add click listeners to the markers

In the createMarkers function, add a click listener to each marker as you create them. The click listener fetches details about the place associated with that marker and calls the function to display the details.

Paste the following code inside the createMarkers function at the code comment TODO: Step 4B .
The showDetails method is implemented in the next section.

step4/index.html

```
/* TODO: Step 4B: Add click listeners to the markers */
// Add click listener to each marker
google.maps.event.addListener(marker, 'click', () => {
  let request = {
  placeId: place.place_id,
  fields: ['name', 'formatted_address', 'geometry', 'rating',
    'website', 'photos']
  };

  /* Only fetch the details of a place when the user clicks on a marker.
  * If we fetch the details for all place results as soon as we get
  * the search response, we will hit API rate limits. */
  service.getDetails(request, (placeResult, status) => {
  showDetails(placeResult, marker, status)
  });
});
```

In the addListener request above, the placeId property specifies a single place for the details request, and the fields property is an array of field names for information you want returned about the place. A full list of fields you can request is in the PlaceResult interface documentation.

C. Show place details in an info window

An info window displays content (usually text or images) in a popup window above the map, at a given location. The info window has a content area and a tapered stem. The tip of the stem is attached to a specified location on the map. Typically info windows are attached to markers, but you can also attach an info window to a specific latitude/longitude.

Add the following code at the comment TODO: Step 4C to create an InfoWindow that displays the name and rating of the business, and attaches that window to the marker. We will define showPanel in the next section for displaying details in a sidebar.

step4/index.html

```
/* TODO: Step 4C: Show place details in an info window */
// Builds an InfoWindow to display details above the marker
function showDetails(placeResult, marker, status) {
  if (status == google.maps.places.PlacesServiceStatus.OK) {
  let placeInfowindow = new google.maps.InfoWindow();
  placeInfowindow.setContent('<div><strong>' + placeResult.name +
    '</strong><br>' + 'Rating: ' + placeResult.rating + '</div>');
  placeInfowindow.open(marker.map, marker);
  currentInfoWindow.close();
  currentInfoWindow = placeInfowindow;
  showPanel(placeResult);
  } else {
  console.log('showDetails failed: ' + status);
  }
}

/* TODO: Step 4D: Load place details in a sidebar */
```

D. **Load place details in a sidebar**

Use the same details returned in the PlaceResult object to populate another div. In this sample, use infoPane which is an arbitrary variable name for the div with the ID " panel ". Each time the user clicks on a new marker, this code closes the sidebar if it was already open, erases the old details, adds the new details, and opens the sidebar.

Add the following code at the comment TODO: Step 4D .

step4/index.html

```
/* TODO: Step 4D: Load place details in a sidebar */
// Displays place details in a sidebar
function showPanel(placeResult) {
  // If infoPane is already open, close it
  if (infoPane.classList.contains("open")) {
  infoPane.classList.remove("open");
  }

  // Clear the previous details
  while (infoPane.lastChild) {
  infoPane.removeChild(infoPane.lastChild);
  }

  /* TODO: Step 4E: Display a Place Photo with the Place Details */

  // Add place details with text formatting
  let name = document.createElement('h1');
  name.classList.add('place');
  name.textContent = placeResult.name;
  infoPane.appendChild(name);
  if (placeResult.rating != null) {
  let rating = document.createElement('p');
  rating.classList.add('details');
  rating.textContent = `Rating: ${placeResult.rating} \u272e`;
  infoPane.appendChild(rating);
  }
  let address = document.createElement('p');
  address.classList.add('details');
  address.textContent = placeResult.formatted_address;
```

```
    infoPane.appendChild(address);
    if (placeResult.website) {
    let websitePara = document.createElement('p');
    let websiteLink = document.createElement('a');
    let websiteUrl = document.createTextNode(placeResult.website);
    websiteLink.appendChild(websiteUrl);
    websiteLink.title = placeResult.website;
    websiteLink.href = placeResult.website;
    websitePara.appendChild(websiteLink);
    infoPane.appendChild(websitePara);
    }

    // Open the infoPane
    infoPane.classList.add("open");
}
```

E. Display a Place Photo with the Place Details

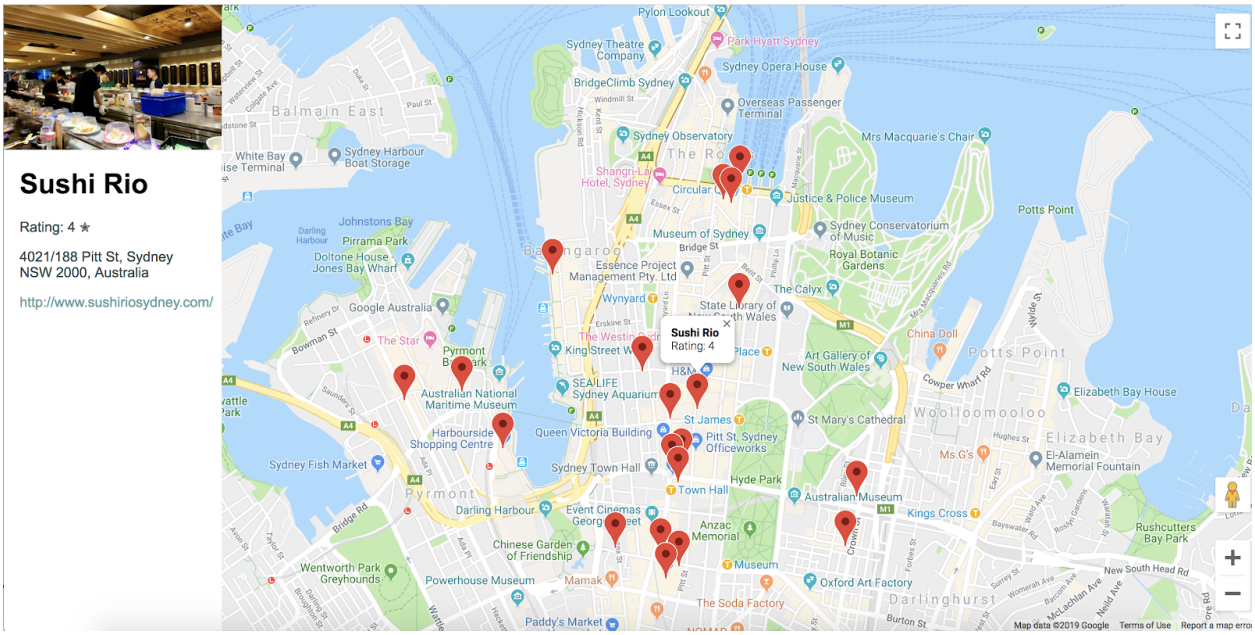
The getDetails result returns an array of up to 10 photos associated with the placeId . Here, we display the first photo above the place name in the sidebar. Place this code above creation of the "name" element if you want the photo to appear at the top of the sidebar.

step4/index.html

```
/* TODO: Step 4E: Display a Place Photo with the Place Details */
// Add the primary photo, if there is one
if (placeResult.photos != null) {
    let firstPhoto = placeResult.photos[0];
    let photo = document.createElement('img');
    photo.classList.add('hero');
    photo.src = firstPhoto.getUrl();
    infoPane.appendChild(photo);
}
```

Test it out

- Save and Reload the page in your browser and allow geolocation permissions.
- Click on a marker to see the InfoWindow pop up from the marker displaying a few details, and the sidebar slide out from the left to display more details.
- Test whether the search also works if you reload and deny geolocation permissions.
- Edit your search keyword for a different query and explore the result returned for that search.



Full sample code

The full code for this project through Step 4 is available on Github.

More with Maps

Congratulations! You've used many features of the Google Maps JavaScript API, including the Places Library.

What we've covered

Creating a map with the google.maps.Map class

Using the user's browser for geolocation and displaying the results on a map

Adding markers to your map and responding to user click events on them

Adding info windows to display more information when a user clicks a marker

Loading the Places Library and performing a Nearby Search

Fetching and displaying place details and place photos

Next steps

To do even more with maps, explore the Maps JavaScript API documentation and Places Library documentation which contain guides, tutorials, the API reference, more code samples, and support channels. Some popular features are Importing Data into Maps, Styling Your Map, and adding the Street View Service.

Which type of codelab would you most like us to build next?

- More examples of using rich Places information
- More codelabs using the Maps Platform JavaScript API
- More codelabs for Android
- More codelabs for iOS
- Visualizing location-based data on maps
- Custom styling of maps
- Using StreetView

Is the codelab you want not listed above? Request it with a new issue here.