

# Google Assistant: Build a Restaurant Locator with the Places API

GSP486



Google Cloud Self-Paced Labs

# Overview

[Google Assistant](#) is a personal voice assistant that offers a host of actions and integrations. From sending texts and setting reminders, to ordering coffee and playing music, the 1 million+ actions available suit a wide range of voice command needs.

[Cloud Functions](#) is a lightweight compute solution for developers to create single-purpose, stand-alone functions that respond to Cloud events without the need to manage a server or runtime environment.

The [Places API](#) is a service that returns information about points of interest by using HTTP requests. More specifically, you will take advantage of the [Place Details](#) and [Place Photos](#) services to receive detailed information and photos of establishments.

By utilizing Cloud Functions and the Places API, you will build an Assistant application that takes in a user's current location and restaurant preferences to generate the ideal restaurant for them to visit, complete with names, addresses, and photos.

## What you will learn

In this lab, you will learn how to:

- Build an Assistant application pipeline that consists of an Actions project, a Dialogflow agent with custom intents and entities, a webhook, and a Cloud Function to handle fulfillment.
- Generate the proper authentication credentials and install necessary dependencies to use the Places API.
- Add fulfillment logic to the Cloud Function to handle Places API calls.
- Deploy your application and test it with the Actions Simulator.

## Prerequisites

This is an **advanced level** lab. This assumes familiarity with Dialogflow and Cloud Functions. Basic knowledge of APIs is recommended. Experience with JavaScript and the Node.js runtime is recommended, but not required. If you need to brush up on these skills, please take one of the following labs before attempting this one:

- [Google Assistant: Build an Application with Dialogflow and Cloud Functions](#)
- [Introduction to APIs in Google](#)

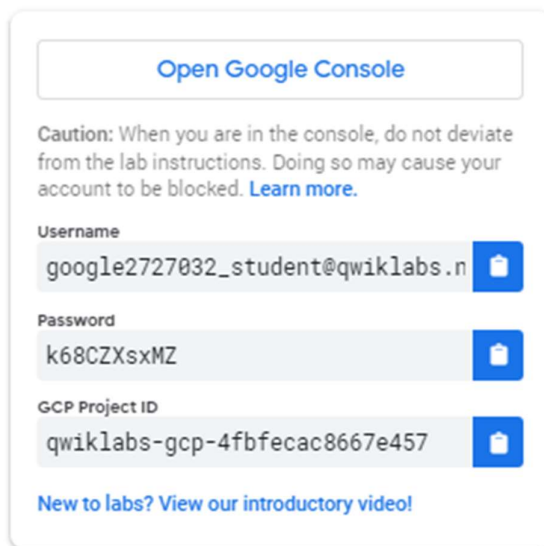
Once you're ready, scroll down and follow the steps below to get your lab environment set up.

# Setup

## Cloud Console

### How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



Open Google Console

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. [Learn more.](#)

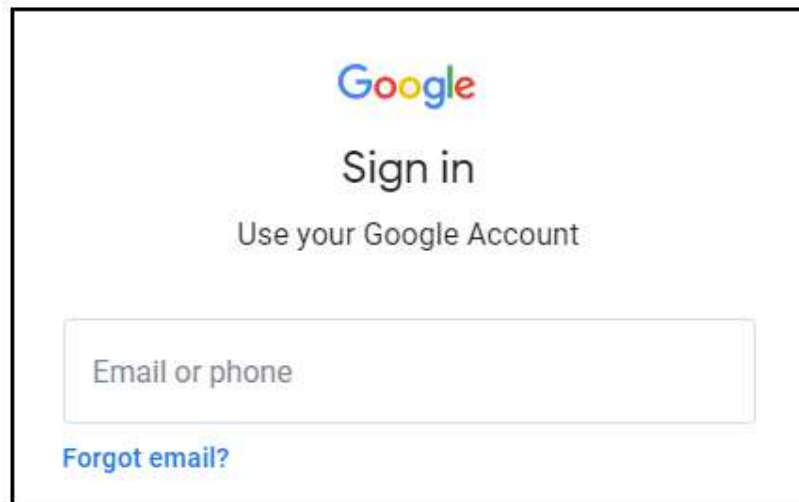
Username  
google2727032\_student@qwiklabs.n

Password  
k68CZXsxMZ

GCP Project ID  
qwiklabs-gcp-4fbfecac8667e457

[New to labs? View our introductory video!](#)

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Google

Sign in

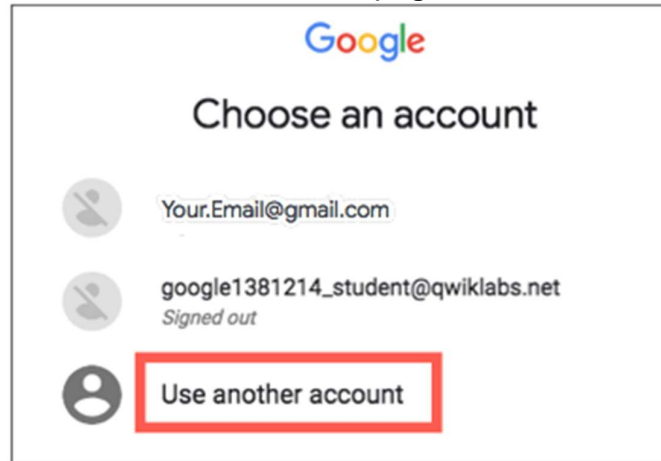
Use your Google Account

Email or phone

[Forgot email?](#)

**Tip:** Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another**



**Account.**

3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

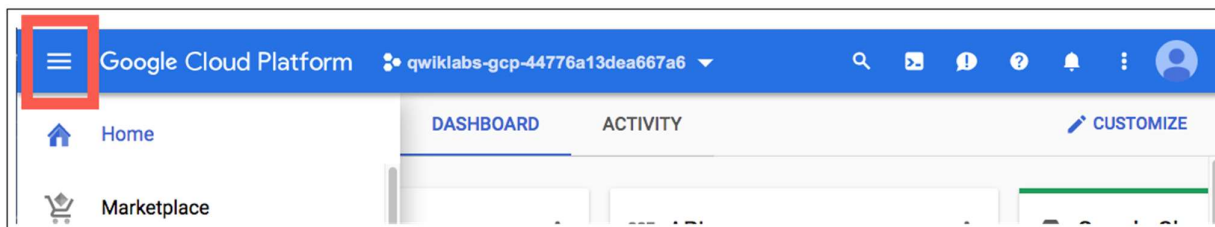
**Important:** You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

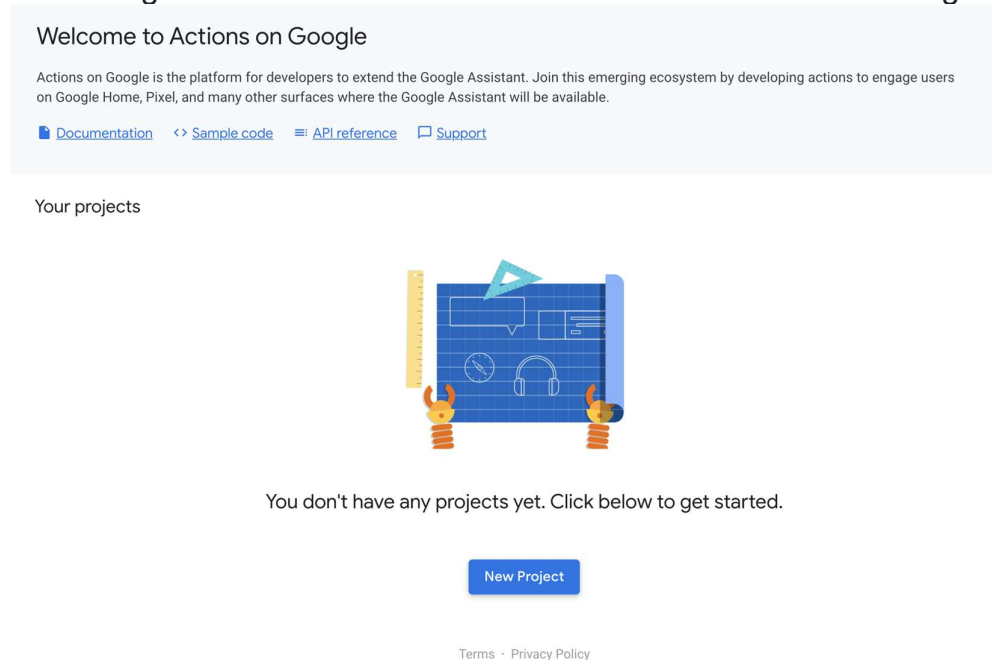
**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



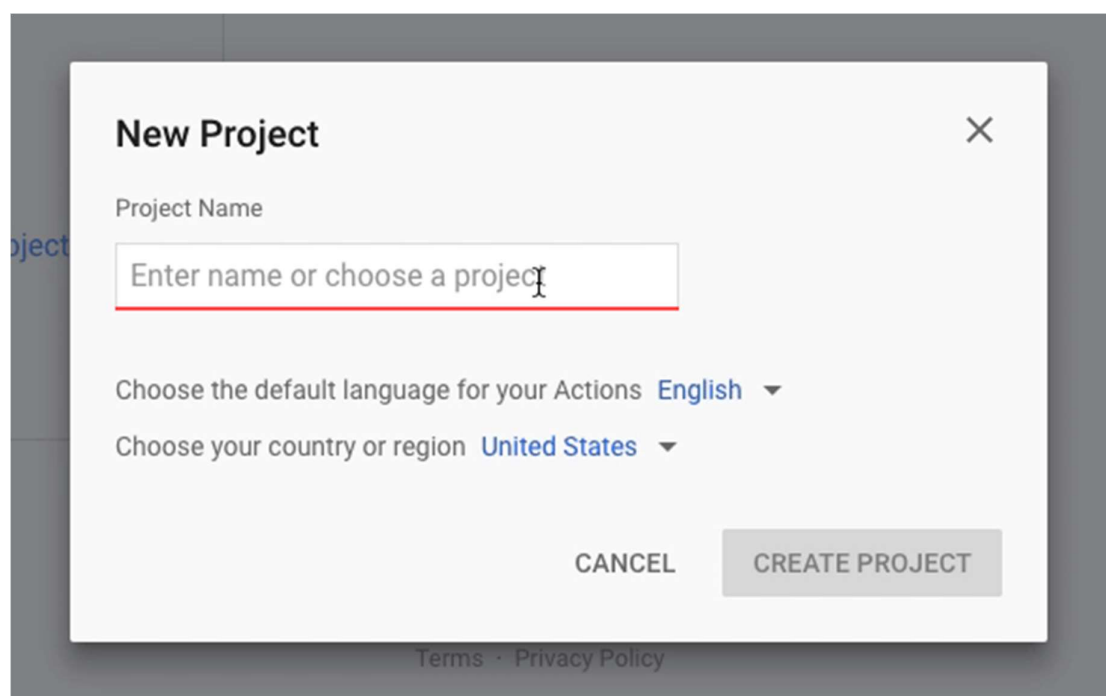
# Create an Actions project

In order to build any Assistant application, you will first have to make an Actions project.

Open a new tab in your browser and go to the [Actions on Google Developer Console](#). Then sign in with your Qwiklabs credentials if prompted. Once you're signed in, you should be looking at a clean Actions console that resembles the following:



Click **New Project** and agree to Actions on Google's terms of service when prompted. Then click into the project field and select your Qwiklabs Google Cloud project ID. Then click **Import project**:



Refresh your page. Your console should now resemble the following:

New Project

Get started


Next

What kind of Action do you want to build?

Select the category that best fits the type of experience you want to build for the Google Assistant.


Smart Home

Let users control your smart home devices with the Google Assistant and the Google Home app




Food ordering

Integrate your food ordering flow with Google Search, Maps, and the Assistant




Game

Build anything from a trivia game to a fully immersive, multiplayer gaming experience



Custom

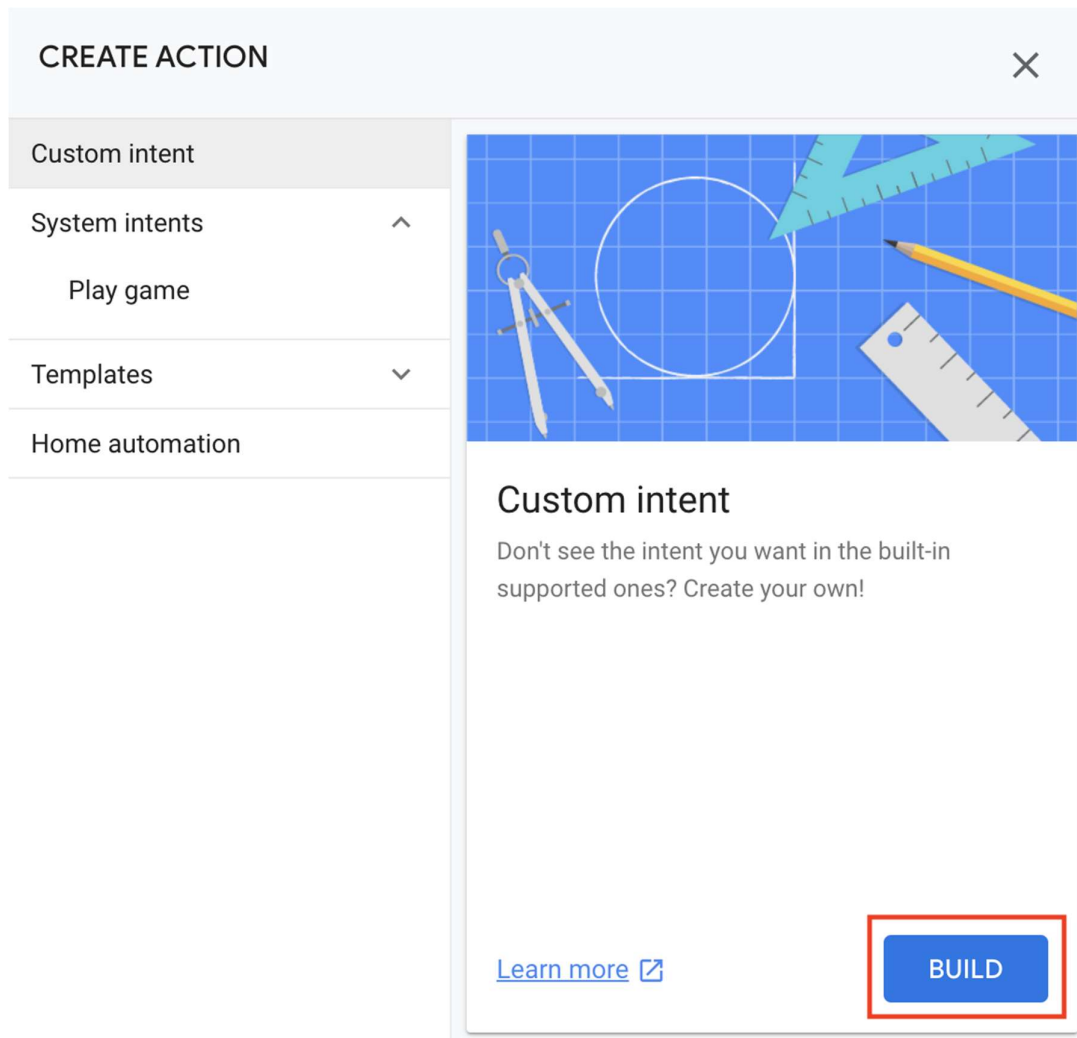
Don't see your category? Build a unique conversational experience for your users



Then click on "Actions Console" in the top left corner and select your Project (title has your Qwiklabs Project ID as the name.)

# Build an Action

From the center Console, select **Build your action > Add Action(s) > Get Started**. Then select **Custom Intent > BUILD**:



This will take you to the Dialogflow console.

Select your Qwiklabs account and click **Allow** when Dialogflow prompts you for permission to access your Google Account.

If prompted, click the "Sign-in with Google" button.

When you land on the Dialogflow page, check the box next to **Yes, I have read and accept the agreement** and click **Accept**.

If you are brought to the following Dialogflow agent creation page, click **CREATE**:

qwiklabs-gcp-8aaed5810f687c99

CREATE

DEFAULT LANGUAGE ?

English — en

Primary language for your agent. Other languages can be added later.

DEFAULT TIME ZONE

(GMT-8:00) America/Los\_Angeles

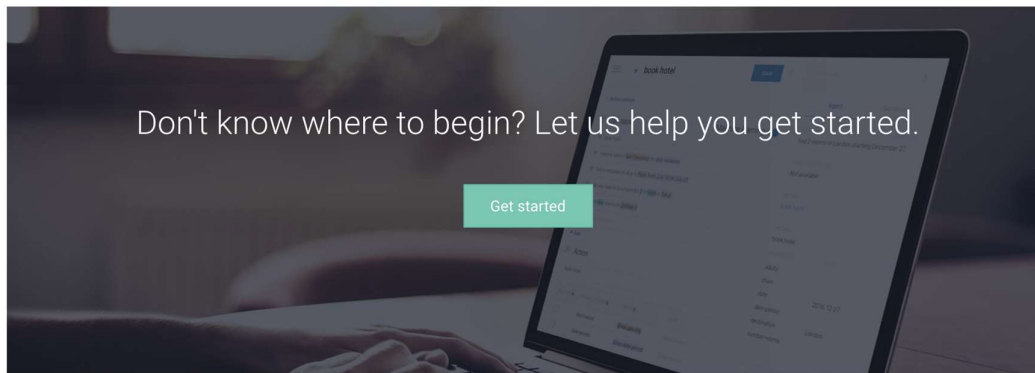
Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Agent will be linked with [qwiklabs-gcp-8aaed5810f687c99](#) Google Project

If you are brought to this page instead:

 Welcome to Dialogflow!



Now it's time to create your first agent.

CREATE AGENT

**Close the Dialogflow agent creation tab.** You will return to the Actions Console.

Click **Get Started** > **Custom Intent** > **BUILD**.

Select your Qwiklabs account and click **Allow** when Dialogflow prompts you for permission to access your Google Account.

Now click **CREATE**:



qwiklabs-gcp-8aaed5810f687c99

CREATE

DEFAULT LANGUAGE ?

English — en

Primary language for your agent. Other languages can be added later.

DEFAULT TIME ZONE

(GMT-8:00) America/Los\_Angeles

Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Agent will be linked with [qwiklabs-gcp-8aaed5810f687c99](#) Google Project

**Note:** If you are not getting the Dialogflow agent creation page, click on **Create Agent**, give the name as your project ID. Select the Google project and click **Create**

## Test Completed Task

Click **Check my progress** to verify your performed task.

You will now enable webhook fulfillment in Dialogflow so that your Cloud Function can be passed the necessary arguments to create this application. Click **Fulfillment** from the left-hand menu.

Move the **Disabled** slider for Webhook to the right to **Enabled**.

Enter the dummy URL <https://google.com> in for the URL field. You will update this URL when you build and deploy your fulfillment:

⚡ Fulfillment

Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL\*


<https://google.com>

Scroll down and click **Save** in the bottom right corner. Then click **Intents** from the left-hand menu and select **Default Welcome Intent**:


## Intents

---

---

 Default Fallback Intent

---

 Default Welcome Intent

---

## Conversational design

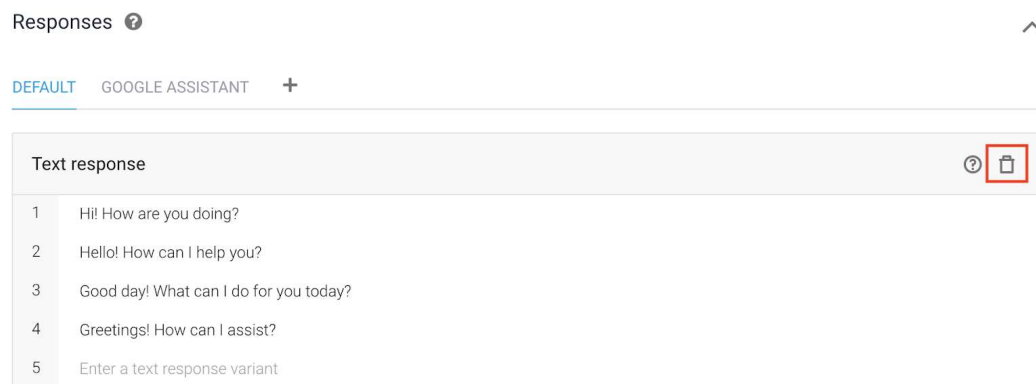
Before you begin creating an Assistant application, it's always a good idea to map out a conversation by writing sample dialog(s). These dialogs give you a good reference point when you eventually start building your app. Here is a sample dialog for the application we're building:

- **App:** *Hello there and welcome to Restaurant Locator! What is your location?*
- **User:** *345 Spear Street, San Francisco*
- **App:** *Okay. How far are you willing to travel?*
- **User:** *About a half mile.*
- **App:** *What type of food or cuisine are you looking for?*
- **User:** *Thai food.*
- **App:** *Fetching your request... Okay, the restaurant name is "X" and the address is "Y". The following photo from a Google Places user might whet your appetite!*

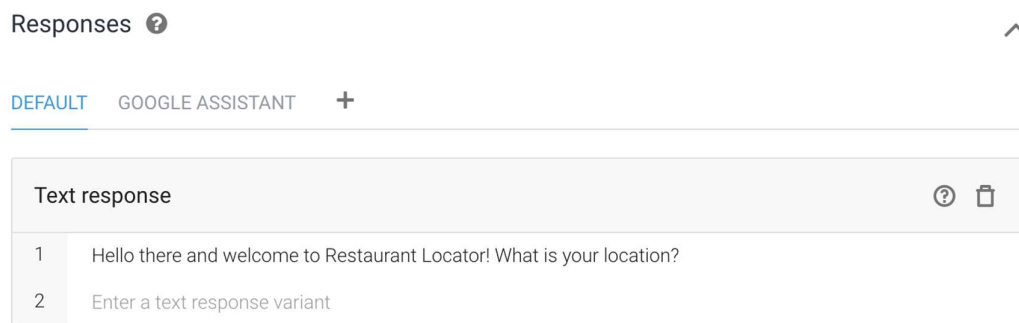
**Conversational design tips:** This example shows only a couple of the dialogs that you could write for your own apps. See this [Design Walkthrough](#) for more examples of dialogs.

# Build the Default Welcome Intent

Make sure that you are in the Default Welcome Intent. Scroll down to the **Responses** section and click the **trash icon** to scrap all of the default text responses:



Now click on **ADD RESPONSES > Text response**, and type in the following: *Hello there and welcome to Restaurant Locator! What is your location?*



Now scroll up and click **Save** in the top-right corner. Now, when users invoke your app, they know that they're entering your app's experience and what to say next.

## Test Completed Task

Click **Check my progress** to verify your performed task.

In general, the app's responses should guide users on what to say next and to stay within your conversations' grammar, which will be further defined in the following section.

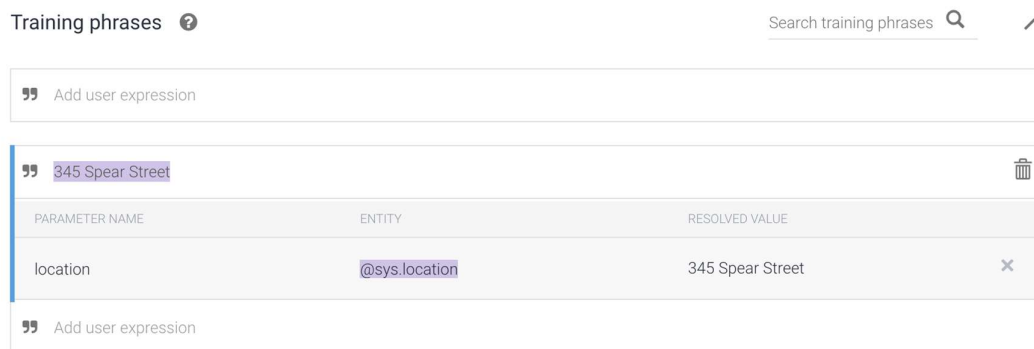
# Build the Custom Intent

Dialogflow intents define your conversation's grammar and what tasks to carry out when users speak specific phrases. You will now create an intent that parses user input for fields that are required by the Places API.

From the left-hand menu, click on the **+** icon by the **Intents** menu item. For the **Intent name** field, enter `get_restaurant`.

In the Training phrases field click **Add Training Phrases**. Then in the Add user expression field enter in `345 Spear Street` and hit **Enter**.

Highlight the address and then assign it the `@sys.location` entity. Your console should now resemble the following:



The screenshot shows the 'Training phrases' section in the Dialogflow console. At the top, there is a search bar labeled 'Search training phrases' and a plus icon. Below this, there is a text input field with the placeholder 'Add user expression'. The first training phrase is '345 Spear Street', which is highlighted in purple. To the right of this phrase is a trash icon. Below the phrase, there is a table with three columns: 'PARAMETER NAME', 'ENTITY', and 'RESOLVED VALUE'. The table contains one row with the following data:

PARAMETER NAME	ENTITY	RESOLVED VALUE
location	@sys.location	345 Spear Street

At the bottom of the table, there is a plus icon and the text 'Add user expression'.

By assigning this portion of the User says to be an entity, Dialogflow can extract parameters from the user input, validate the parameter and provide it to your fulfillment as a variable.

You will now add some more user expressions in the **Training phrases** section. Add the following training phrases by entering them in the user expression field:

- *1600 Amphitheatre Parkway, Mountain View*
- *20 W 34th St, New York, NY 10001*

Highlight the addresses and then assign them the `@sys.location` entity. Your Training phrases section should now resemble the following—note the highlighted addresses, which align with the `@sys.location` entity:

” Add user expression

” 20 W 34th St, New York, NY 10001

PARAMETER NAME	ENTITY	RESOLVED VALUE	
location	@sys.location	20 W 34th St, New York, NY 10001	×

” 1600 Amphitheatre Parkway, Mountain View, CA

” 345 Spear Street

If one of these values weren't assigned the `@sys.location` entity, click on the address and manually assign it so that it resembles the above screenshot.

Now that your training phrases have been added, scroll down and under the **Actions and parameters** section, click **Manage Action and Parameter**.

Now check the **REQUIRED** checkbox for the `@sys.location` entity. This tells Dialogflow not to trigger the intent until the parameter is properly provided by the user.

Now click on **Define prompts** for location (right-hand side) and provide a re-prompt phrase. Enter in `What is your address?` in the prompt field and then click **Close**:

### Prompts for "location"

NAME	ENTITY	VALUE
location	@sys.location	\$location

PROMPTS

1 What is your address?

2 Enter a prompt variant

CLOSE

This phrase will be spoken to the user repeatedly until Dialogflow detects an address in the user input.

Still in the Action and parameters section, click **+ New Parameter** and add the following information:

- **Required** - Select the checkbox
  - **Parameter name** - proximity
  - **Entity** - @sys.unit-length
  - **Value** - \$proximity
  - **Prompts** - How far are you willing to travel?
- Once again, click **+ New Parameter** and add the following information:

- **Required** - Select the checkbox
  - **Parameter name** - cuisine
  - **Entity** - @sys.any
  - **Value** - \$cuisine
  - **Prompts** - What type of food or cuisine are you looking for?
- Your actions and parameters section should now resemble the following:

Action and parameters ^

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?	PROMPTS ?
<input checked="" type="checkbox"/>	location	@sys.location	\$location	<input type="checkbox"/>	What is your ad... <span>⌵ ⋮</span>
<input checked="" type="checkbox"/>	proximity	@sys.unit-length	\$proximity	<input type="checkbox"/>	How far are you...
<input checked="" type="checkbox"/>	cuisine	@sys.any	\$cuisine	<input type="checkbox"/>	What type of fo...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	—

[+ New parameter](#)

Now scroll to the top of the page and click **Save**.

Now scroll down to the **Fulfillment** section. Click the dropdown arrow and click **Enable fulfillment**. Then click the **Enable webhook call for this intent** slider:

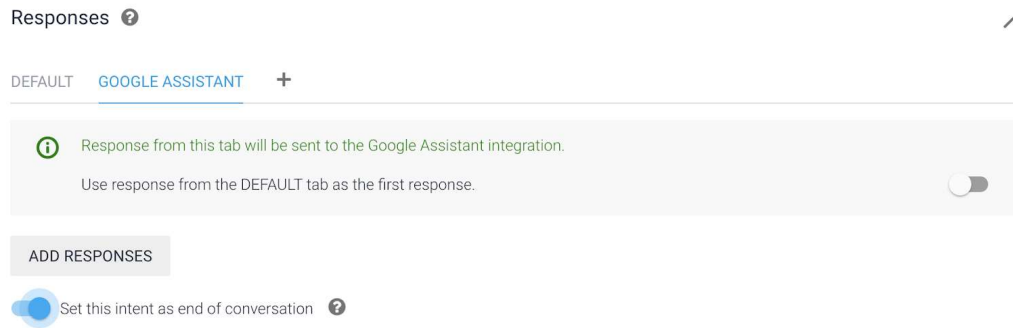
## Fulfillment ?

☒ Enable webhook call for this intent

☐ Enable webhook call for slot filling

In the **Responses** section above fulfillment, click on **+** icon and select the **Google Assistant** tab.

Move the toggle for **Set this intent as end of conversation**. This tells Dialogflow to relinquish control back to the Google Assistant after your fulfillment returns a response to the user.



Then scroll to the top of the intent and click **SAVE** once more to save the entire intent.

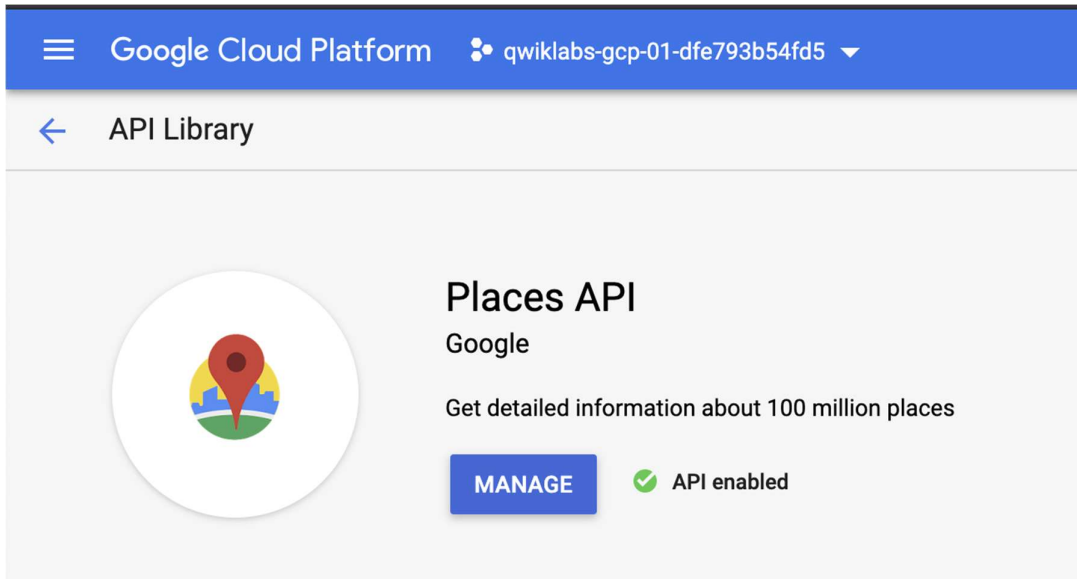
## Test Completed Task

Click **Check my progress** to verify your performed task.

# Enable APIs and retrieve an API key

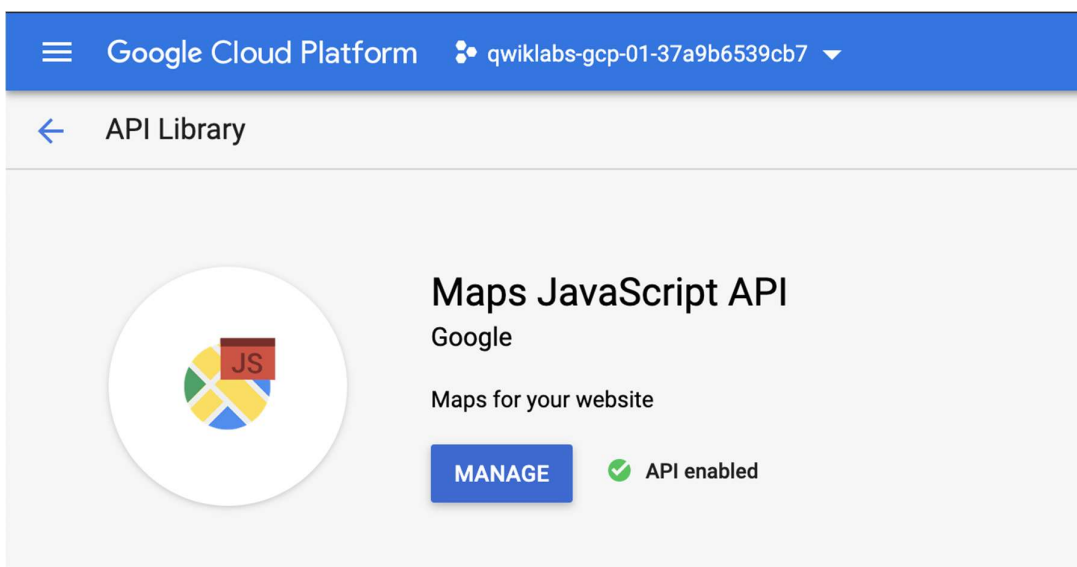
Return to the Cloud Console for this step. Open the Navigation menu and select **APIs & Services > Library**.

Search for the "Places API" and click **Enable**. Now if you check that page again, you should see a green check mark in the place of where "Disabled" was:



Open the Navigation menu and select **APIs & Services > Library**.

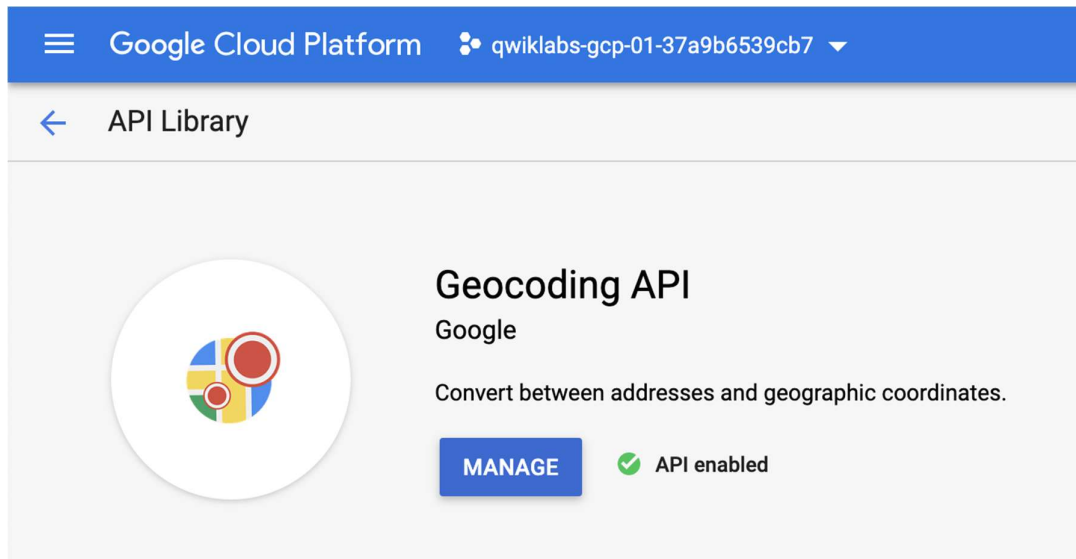
Search for the "Maps JavaScript API" and click **Enable**. Now if you check that page again, you should see a green check mark in the place of where "Disabled" was:



Finally, open the Navigation menu and select **APIs & Services > Library**.



Search for the "Geocoding API" and click **Enable**. Now if you check that page again, you should see a green check mark in the place of where "Disabled" was:



Now open the Navigation menu and select **APIs & Services > Credentials**.

Then from the top-hand menu click **+ CREATE CREDENTIALS > API Key**. You will be presented with a similar panel:


## API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key

AIzaSyDWGPTIeXSJnpgkS\_vqqz1R3G8NDy3MA1o



 Restrict your key to prevent unauthorized use in production.

[CLOSE](#)

[RESTRICT KEY](#)

Copy the API key and save it (in a notepad, for example)—you will use it in the following step.

# Initialize a Cloud Function

You will now build a Cloud Function to handle your fulfillment logic through the various APIs discussed earlier.

Open the Navigation menu and select **Cloud Functions**, which is located under the compute header. Then click **Create function**.

This will open a template to create a new Cloud Function. Your page will resemble the following:

Cloud Functions

Create function

1 Configuration

2 Code

### Basics

Function name \*

restaurant\_locator

?

Region

us-central1

▼

?

### Trigger

⌕ HTTP

Trigger type

HTTP

▼

URL

📄

https://us-central1-qwiklabs-gcp-02-5ea1e8027a0b.cloudfunctions.net/restaurant...

#### Authentication

☒ Allow unauthenticated invocations

Check this if you are creating a public API or website.

☐ Require authentication

Manage authorized users with Cloud IAM.

SAVE

CANCEL

For the Cloud **Function name** field, enter **restaurant\_locator**.

Then scroll down to the authentication section and check the box next to "Allow unauthenticated invocations" and click **Save**.

### Trigger

HTTP

### URL

https://us-central1-qwiklabs-gcp-04-04eae503ec68.cloudfunctions.net/dialogflowFirebaseFulfillment3

### Authentication



#### Allow unauthenticated invocations

Check this if you are creating a public API or website.

This is a shortcut to assign the IAM Invoker role to the special identifier allUsers. You can use IAM to edit this setting after the function is created.

**Forgetting to do the above will cause your simulation test to fail at the end!**

Click **Next**. Find the inline editor for index.js and package.json. Make sure that the index.js tab is open. This file defines your fulfillment logic and is used to create and deploy a Cloud Function.

**Remove** the boilerplate code from the file. Then **copy and paste** the following code into `index.js`:

```

'use strict';

const {
  dialogflow,
  Image,
  Suggestions
} = require('actions-on-google');

const functions = require('firebase-functions');
const app = dialogflow({debug: true});

function getMeters(i) {
  return i*1609.344;
}

app.intent('get restaurant', (conv, {location, proximity, cuisine}) => {
  const axios = require('axios');
  var api_key = "<YOUR_API_KEY_HERE>";
  var user_location = JSON.stringify(location["street-address"]);
  var user_proximity;
  if (proximity.unit == "mi") {
    user_proximity = JSON.stringify(getMeters(proximity.amount));
  } else {
    user_proximity = JSON.stringify(proximity.amount * 1000);
  }
  var geo_code = "https://maps.googleapis.com/maps/api/geocode/json?address=" +
  encodeURIComponent(user_location) + "&region=<YOUR_REGION>&key=" + api_key;
  return axios.get(geo_code)
    .then(response => {
      var places_information = response.data.results[0].geometry.location;
      var place_latitude = JSON.stringify(places_information.lat);
      var place_longitude = JSON.stringify(places_information.lng);
      var coordinates = [place_latitude, place_longitude];
      return coordinates;
    }).then(coordinates => {
      var lat = coordinates[0];
      var long = coordinates[1];
      var place_search =
      "https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input=" +
      encodeURIComponent(cuisine)
      + "&inputtype=textquery&fields=photos,formatted_address,name,opening_hours,rating&locationbias=circle:" + user_proximity + "@" + lat + "," + long + "&key=" + api_key;
      return axios.get(place_search)
        .then(response => {
          var photo_reference =
          response.data.candidates[0].photos[0].photo_reference;
          var address =
          JSON.stringify(response.data.candidates[0].formatted_address);
          var name = JSON.stringify(response.data.candidates[0].name);
          var photo_request =
          'https://maps.googleapis.com/maps/api/place/photo?maxwidth=400&photoreference=' +
          photo_reference + '&key=' + api_key;
          conv.ask(`Fetching your request...`);
          conv.ask(new Image({
            url: photo_request,
            alt: 'Restaurant photo',
          }))
          conv.close(`Okay, the restaurant name is ` + name + ` and the address is `
+ address + ` . The following photo uploaded from a Google Places user might whet your
appetite!`);
        })
      })
    });

exports.get_restaurant = functions.https.onRequest(app);

```

**Replace** <YOUR\_API\_KEY\_HERE> (line 18) with the API key you generated in the previous step.

Replace `<YOUR_REGION>` (line 26) with your ccTLD (country code top-level domain). [This wikipedia page](#) lists all ccTLDs. This table will help you with conversions:

Country	ccTLD	<YOUR_REGION>
United States	.us	us
India	.in	in
Spain	.es	es

Once your API key and region fields are properly filled out, click on `package.json`. This file declares package dependencies for your fulfillment, including the Actions client library. Replace the contents of the file with the following:

```
{
  "name": "get_reviews",
  "description": "Get restaurant reviews.",
  "version": "0.0.1",
  "author": "Google Inc.",
  "engines": {
    "node": "8"
  },
  "dependencies": {
    "actions-on-google": "^2.0.0",
    "firebase-admin": "^4.2.1",
    "firebase-functions": "1.0.0",
    "axios": "0.16.2"
  }
}
```

Once you have those files configured, find the **Entry point** field above. Enter `in get_restaurant` for the value.

Now click the **Deploy** button below. It will take about a minute for your function to be built.

When the creation completes, your overview page will resemble the following:

Filter functions						Columns
<input type="checkbox"/> Name ^	Region	Trigger	Runtime	Memory allocated	Executed function	Last deployed
<input checked="" type="checkbox"/> restaurant_locator	us-central1	HTTP	Node.js 8	256 MB	get_restaurant	9/30/19, 11:22 AM

Now click on the `restaurant_locator` function to get more details about its configuration. Then click on the **trigger** tab. You will see a function URL that resembles the following:

✓ restaurant\_locator

Version 1, deployed at Mar 25, 2021, 9:36:16 P... ▼

METRICS


DETAILS


SOURCE

VARIABLES

TRIGGER

## ⦿ HTTP

Trigger URL 

[https://us-central1-qwiklabs-gcp-03-5dbd1389f496.cloudfunctions.net/restaurant\\_locator](https://us-central1-qwiklabs-gcp-03-5dbd1389f496.cloudfunctions.net/restaurant_locator) 

**Copy** the function URL. You will use it as the URL for the Dialogflow webhook, which is configured in the next section.

## Test Completed Task

Click **Check my progress** to verify your performed task.

## Configure the webhook

Return to the Dialogflow console and click on the **Fulfillment** menu item from the left hand navigation menu. In the URL field, replace `https://google.com` with the function URL you generated in the previous step. Your webhook should now resemble the following:

## Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<input type="text" value="https://us-central1-qwiklabs-gcp-eea934ee46ceab94.cloudfunctions.net/restaurant_locator"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/>

Scroll down to the bottom of the page and click **Save** in the lower right corner.

Now that your Cloud Function has been deployed and your webhook has been properly set up, you can preview the app in the Actions simulator.

## Change your Google permission settings

In order to test the application that you built over the course of this lab, you need to enable the necessary permissions.

Open a new tab in your browser and visit the [Activity Controls page](#). Sign in with your Qwiklabs credentials if prompted.

Ensure that the following permissions are enabled by sliding the toggles and selecting **TURN ON** for the following cards:

- Web & App Activity  
Now **close** the Activity Controls page.

# Test the application with the Actions simulator

Return to the Dialogflow console. Then from the left-hand menu, select **Integrations**. Then click **PREVIEW MIGRATION**.


## Integrations

### Google Assistant

#### Try Actions Builder <sup>NEW</sup>


Actions Builder visualizes the conversational flow and gives you more control over your users' experience. Try Actions Builder by migrating your Dialogflow project. [Learn more](#)

**PREVIEW MIGRATION**





Not ready yet? Continue with the [integration](#)


Once you land on the following page, click **TEST**:


 **Actions Console**


Overview **Develop** **Test** Deploy Analytics

 Invocation

 **Actions**

 Theme customization


 Account linking

 Backend services

Migrate your project to Actions Builder <sup>NEW</sup>

Actions Builder visualizes the conversational flow and gives you more migrating your Dialogflow project. [Learn more](#)

**Preview migration** [Why migrate?](#)

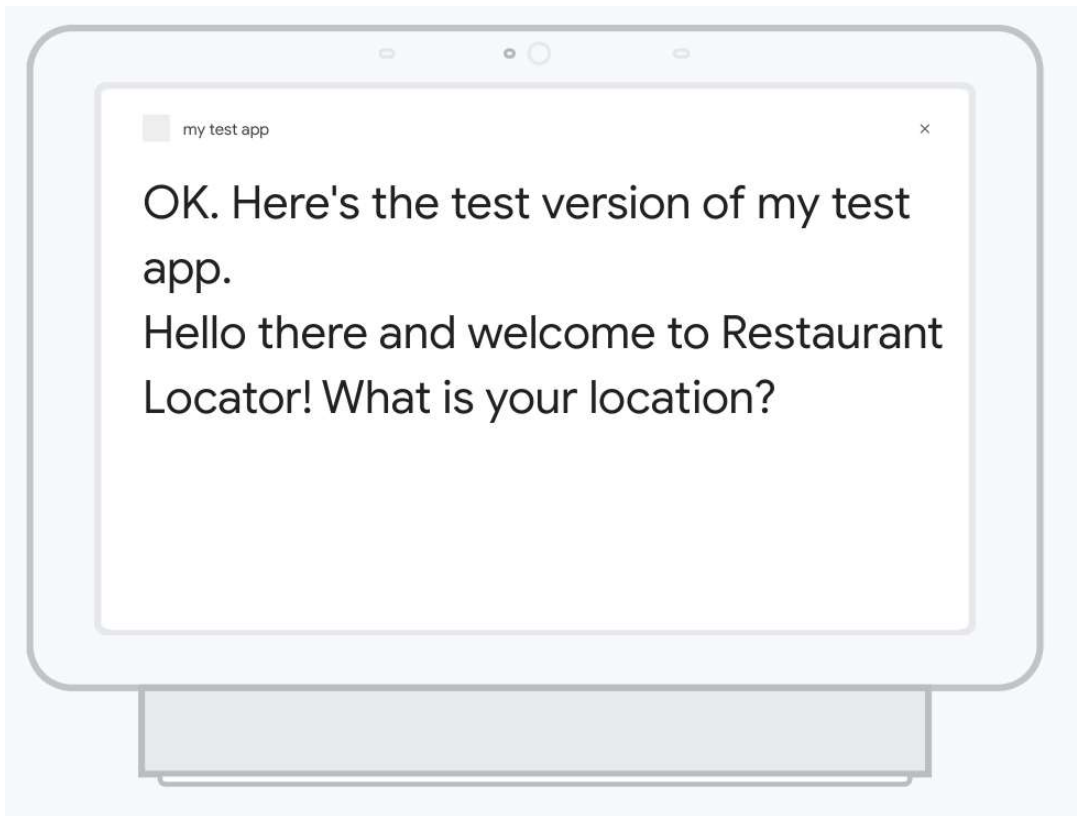
Actions  English ▼

Below is a list of Actions in English

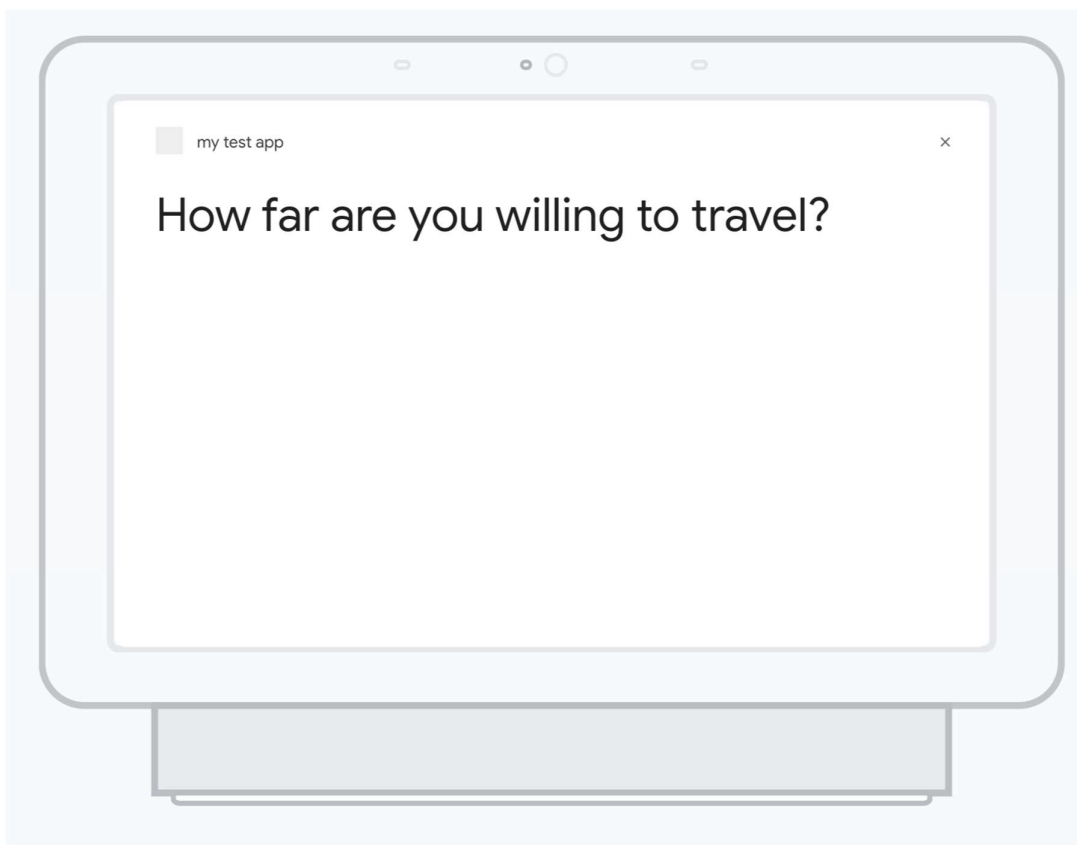
The Actions simulator allows you to test your applications without any hardware like a Google Home.

Enter `Talk to my test app` in the **Input** area. You should receive a similar response:

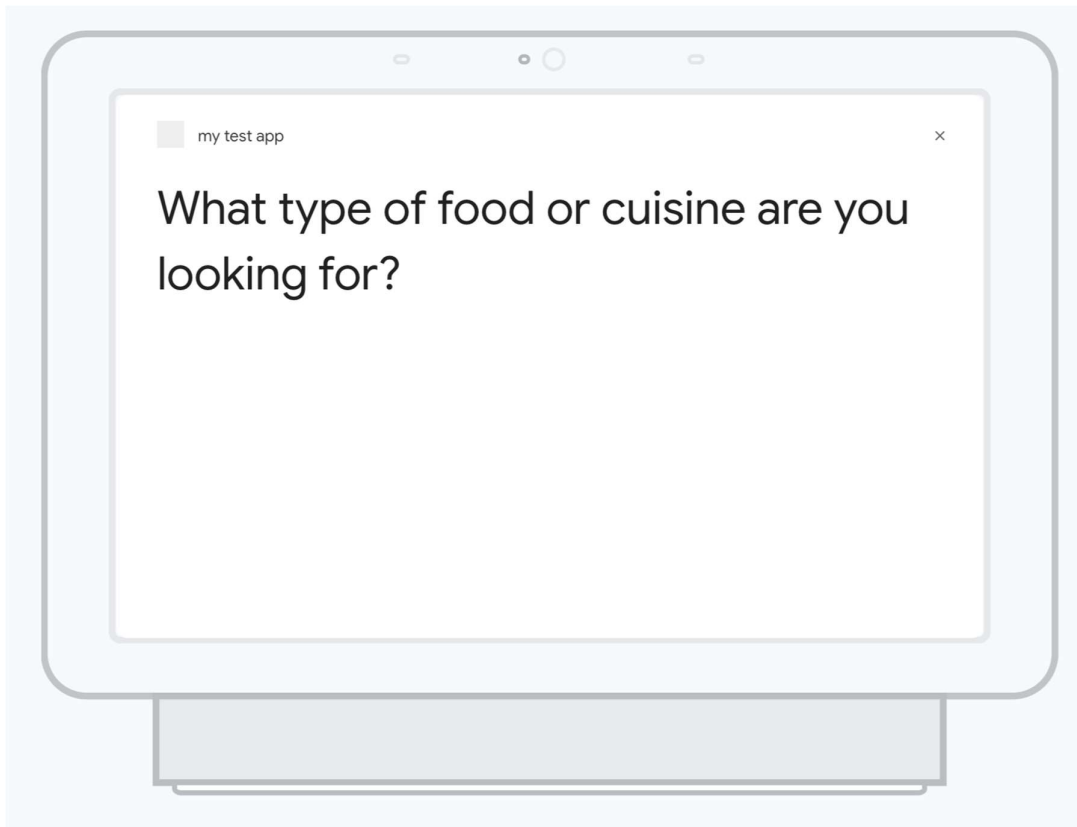




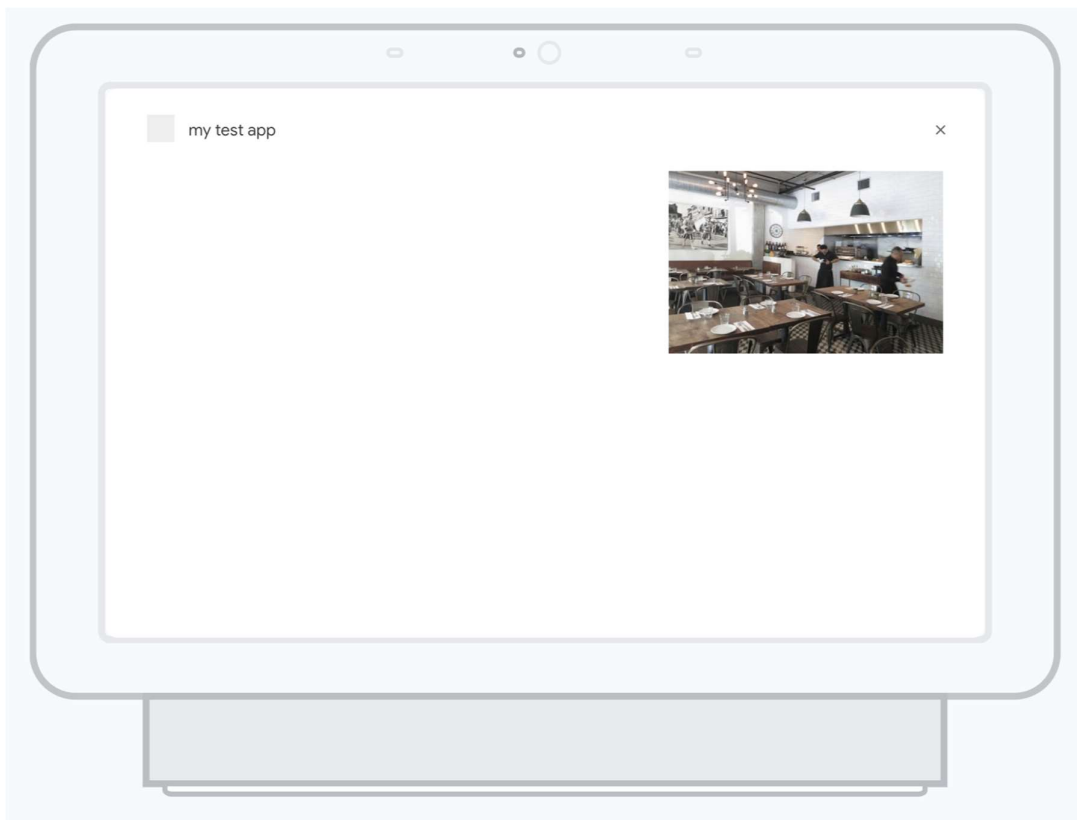
Now enter a valid address (for example, "345 Spear Street San Francisco".) You should then receive the following output:



Now enter in a distance (for example, 0.5 miles away.) You should receive a similar output:



Enter in a cuisine like "Italian" or "Thai food". Soon after you will receive a similar response fitted with a picture in the right hand panel:



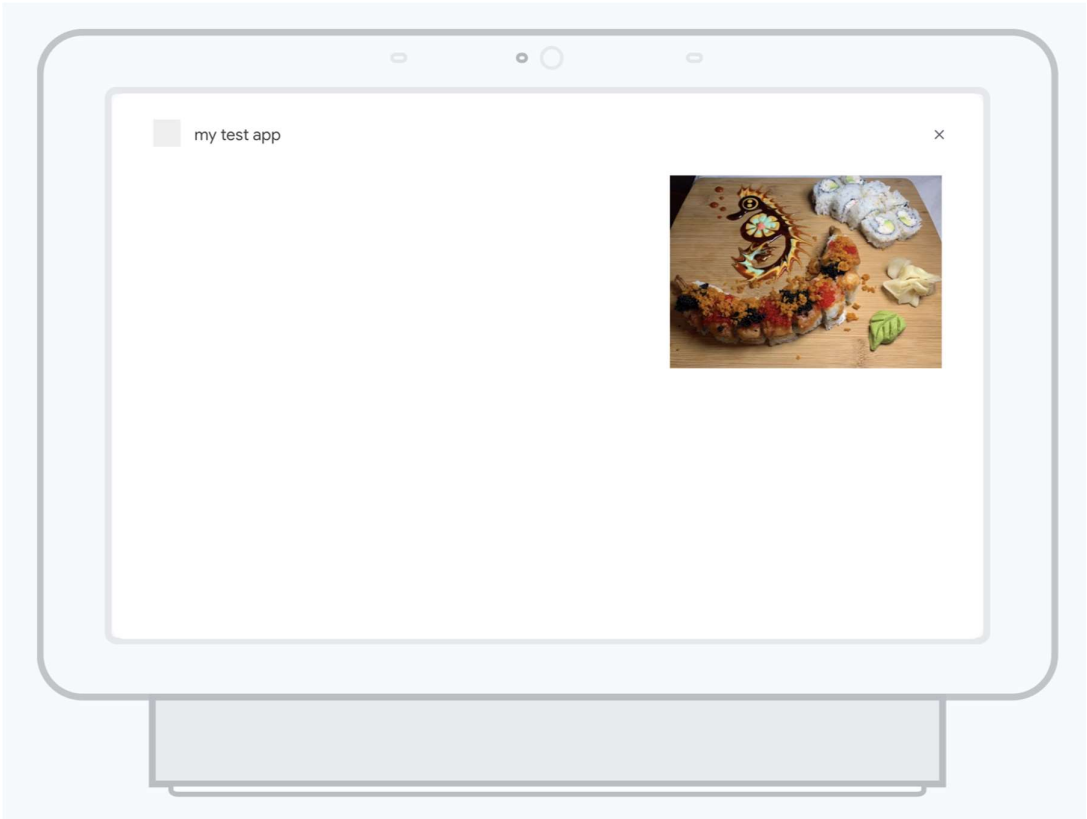
Italian

Fetching your request...

Okay, the restaurant name is "Pazzia Ristorante Italiano" and the address is "337 3rd St, San Francisco, CA 94107, United States". The following photo uploaded from a Google Places user might whet your appetite!

'my test app' left the conversation

Try talking with your test app with new addresses, proximities, and cuisines/types of food!



Sushi

Fetching your request...

Okay, the restaurant name is "Kaisen Sushi" and the address is "71 5th St, San Francisco, CA 94103, United States". The following photo uploaded from a Google Places user might whet your appetite!

'my test app' left the conversation

# Congratulations!

In this lab you built a robust Google Assistant application with Dialogflow and Cloud Function. After creating an Actions project and Action, you configured and built two intents. From there you learned how to initialize Cloud Function and you added your own fulfillment logic and packages to the Cloud Function to handle Places API requests. After deploying the Cloud Function, you updated your webhook URL and tested your Assistant application with the Actions simulator. You are now ready to take more advanced Google Assistant development labs.

## Finish Your Quest



Continue your Quest with [OK Google: Build Interactive Apps with Google Assistant](#). A Quest is a series of related labs that form a learning path. Completing this Quest earns you the badge above, to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in this Quest and get immediate completion credit if you've taken this lab. [See other available Qwiklabs Quests](#).

## Next Steps / Learn More

Be sure to check out the following lab to receive more hands-on practice with Dialogflow:

- [Implementing an AI Chatbot with Dialogflow](#)

## Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. [Our classes](#) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](#) help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated March 25, 2021

Lab Last Tested March 25, 2021

Copyright 2021 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.