

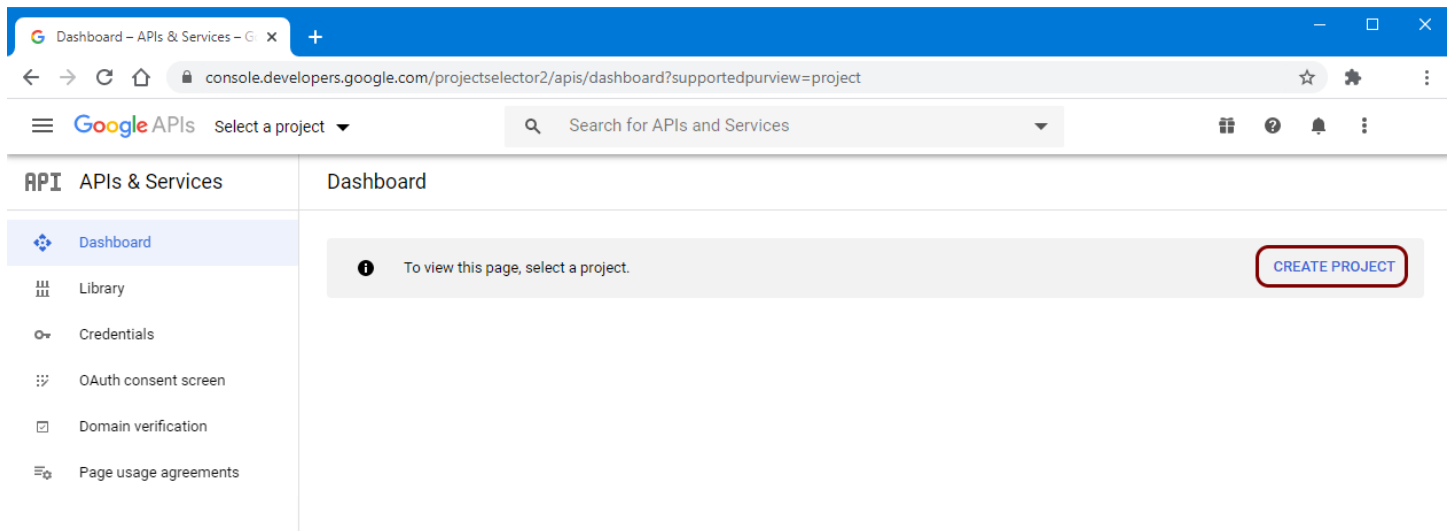
Google OAuth 2.0 Quick Start Guide

This guide will show you how to obtain OAuth 2.0 bearer tokens (also called access tokens) for use with Gmail accounts. To get started, you will need the following:

1. A Google account which you will need to register your application. You can create this for free at <https://accounts.google.com/SignUp>
2. A Gmail account which you can use for testing purposes. You can create a free Gmail account at the sign up process linked above, or you can sign up for G Suite at <https://gsuite.google.com/products/gmail/>
3. A copy of the Postman utility which can be downloaded for free from <https://www.postman.com>. We will use this to compose the queries to Google's servers to obtain the authorization code, bearer, and refresh tokens.
4. SocketTools 10 Build 1245 which supports authentication using OAuth 2.0 bearer tokens. If you have a current development license, this is a free update. If you have an older version of SocketTools, you will need to upgrade to the current version.

If you already have a Google account which uses a different (non-Gmail) email address, you can add a Gmail address to your existing account as described at <https://support.google.com/accounts/answer/72198>

Once you have created your Google account, the first step is to open the Google Developers Console at <https://console.developers.google.com/>




From there, select **Create Project**. If you already have projects associated with your account, you can also click on **Select a project** and then create a new project. Your project is what you'll use to register your application with Google.

The project name should be simple and is typically the name of the application you're going to be registering. Project names can contain spaces.

New Project

Project name *
Test Application ?


Project ID: test-application-287516. It cannot be changed later. [EDIT](#)

Location *
 No organization [BROWSE](#)

Parent organization or folder

[CREATE](#) [CANCEL](#)

For this guide, we'll name our project **Test Application** and Google will assign it a unique Project ID. You can change the project ID if you wish, but generally there's no need.

 You don't have any APIs available to use yet. To get started, click "Enable APIs and services" or go to the [API library](#)

After you've created your project, you'll be sent back to the dashboard and you'll see this message. Click on the link or select **Library** on the left sidebar. This will take you to the API Library search form.

Search for **Gmail API** and select it. You will be shown a confirmation screen to enable the API. Even though you won't be using the web API to access the Gmail account, it still needs to be enabled.



Gmail API

Google

Flexible, RESTful access to the user's inbox

[ENABLE](#)

[TRY THIS API](#) 









[OVERVIEW](#)

[DOCUMENTATION](#)

[SUPPORT](#)

After you have enabled the Gmail API, you will be taken to the API & Services section, and you may see a message about creating credentials. Before we do that, there are a few more things we need to configure.

Return back to the Developers Console dashboard at <https://console.developers.google.com/> select OAuth consent screen from the left sidebar.


API	APIs & Services	OAuth consent screen
	Dashboard	<p>Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.</p> <h3>User Type</h3> <p><input type="radio"/> Internal </p> <p>Only available to users within your organization. You will not need to submit your app for verification.</p> <p><input checked="" type="radio"/> External </p> <p>Available to any user with a Google Account.</p> <p>Let us know what you think about our OAuth experience</p>
	Library	
	Credentials	
	OAuth consent screen	
	Domain verification	
	Page usage agreements	


You want to select **External** as the user type and then click on **Create** and you will be shown the consent form where you enter information about your application.

OAuth consent screen

Before your users authenticate, this consent screen will allow them to choose whether they want to grant access to their private data, as well as give them a link to your terms of service and privacy policy. This page configures the consent screen for all applications in this project.

Verification status
Not published

Application name 
The name of the app asking for consent

Application logo 
An image on the consent screen that will help users recognize your app

This information is presented to the user whenever your application wants access to their account. At a minimum you'll need to provide an application name, which is displayed to the user. For this guide, we'll name our application **My Test Application**.

You can also select a support email address, enter an authorized domain name and links to privacy and policy pages on your website. For our purposes, this won't be required.

Scopes for Google APIs

Scopes allow your application to access your user's private data. [Learn more](#)

If you add a sensitive scope, such as scopes that give you full access to Calendar or Drive, Google will verify your consent screen before it's published.

email

profile

openid

Add scope

One additional piece of information you'll need to configure is the scope. Click the **Add scope** button and a list of available scopes will be displayed.

Add scope

Scopes are used to grant an application different levels of access on behalf of the end user. [Learn more about OAuth 2.0](#)
Only scopes for [enabled APIs](#) are listed.


<input type="checkbox"/>		Gmail API	../auth/gmail.readonly	View your email messages and settings
<input type="checkbox"/>		Gmail API	../auth/gmail.modify	View and modify but not delete your email
<input type="checkbox"/>		Gmail API	../auth/gmail.insert	Insert mail into your mailbox
<input checked="" type="checkbox"/>		Gmail API	https://mail.google.com/	Read, compose, send, and permanently delete all your email from Gmail
<input type="checkbox"/>		Gmail API	../auth/gmail.send	Send email on your behalf
<input type="checkbox"/>		Gmail API	../auth/gmail.addons.current.message.readonly	View your email messages when the add-on is running
<input type="checkbox"/>		Gmail API	../auth/gmail.compose	Manage drafts and send emails

Scroll down to the scope <https://mail.google.com/> and select it. Then click **Add** and you'll see a warning on the consent screen that you've added a scope which requires verification. Then click on **Save** to complete the process.

API APIs & Services

Credentials [+ CREATE CREDENTIALS](#) [DELETE](#)

Create credentials to access your enabled APIs. [Learn more](#)

 To protect you and your users, your consent screen and application need to be verified by Google. [Learn more](#) [CONFIGURE CONSENT SCREEN](#)

API Keys

<input type="checkbox"/>	Name	Creation date ↓	Restrictions	Key	Usage with all services (last 30 days) ?
No API keys to display					

OAuth 2.0 Client IDs

<input type="checkbox"/>	Name	Creation date ↓	Type	Client ID
No OAuth clients to display				

Service Accounts [Manage service accounts](#)

<input type="checkbox"/>	Email	Name ↑	Usage with all services (last 30 days) ?
No service accounts to display			

The next step is to create your credentials. Select **Credentials** on the Developers Console and then click on **Create Credentials**. Select **OAuth client ID** from the list.

[←](#) Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information.

Application type *
Desktop app

[Learn more](#) about OAuth client types

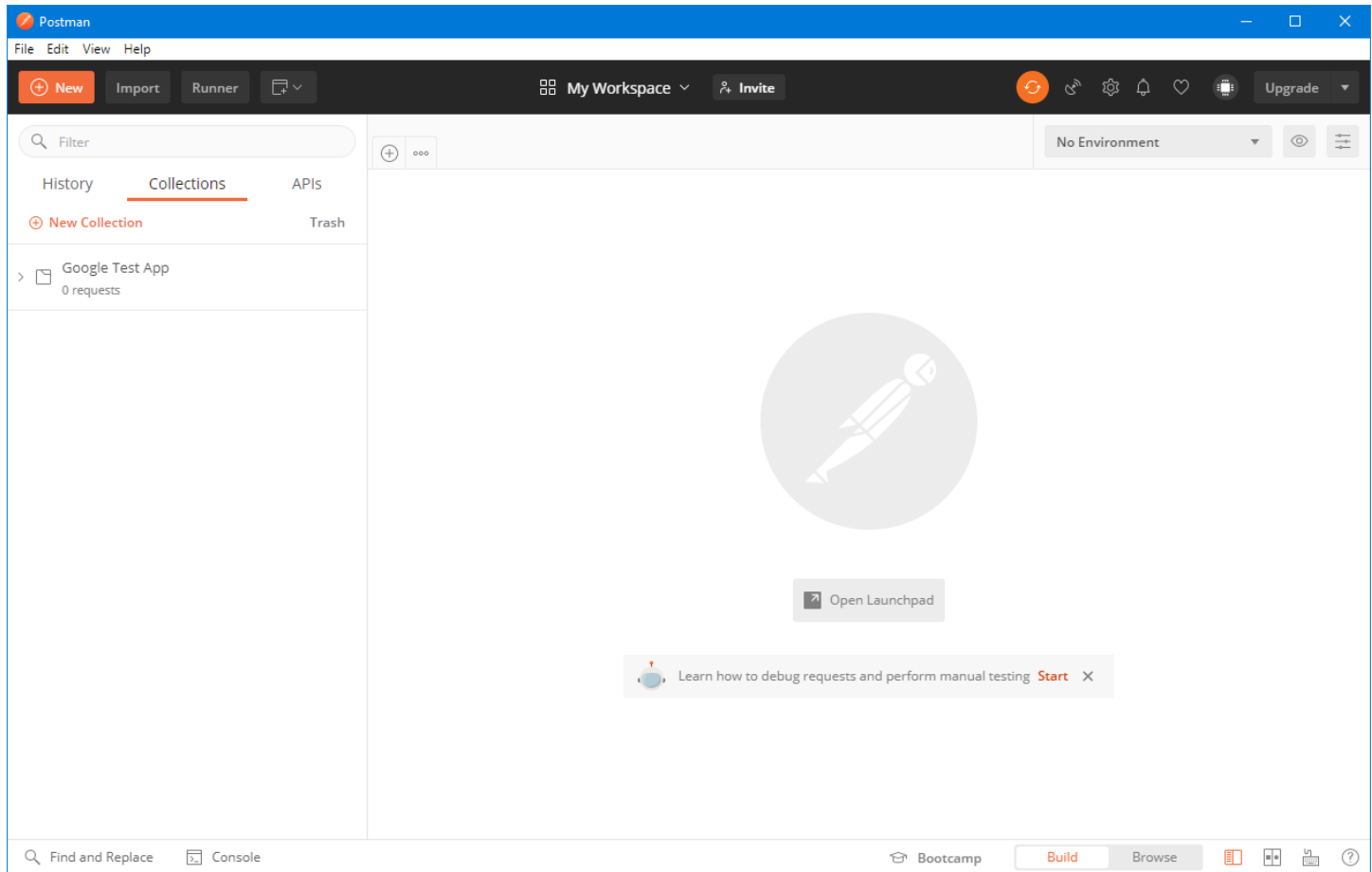
Name *
Test Mail Client

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

[CREATE](#) [CANCEL](#)

Select **Desktop app** as the application type, and you can assign it whatever name you wish. As noted on the form, this is not what is shown to end users. This is for your own internal use in the console. Click on **Create**.

Open Postman and create a new collection by selecting **New Collection** and give it a name. For this example, we will name the collection "**Google Test App**". You can also add a description of it if you wish, then press **Create**.



These collections are where you will store the requests we will send to Google’s servers. We will create the requests first, and then we’ll begin the process of obtaining the tokens we need.

The three requests we’ll create in Postman are:

Authorization

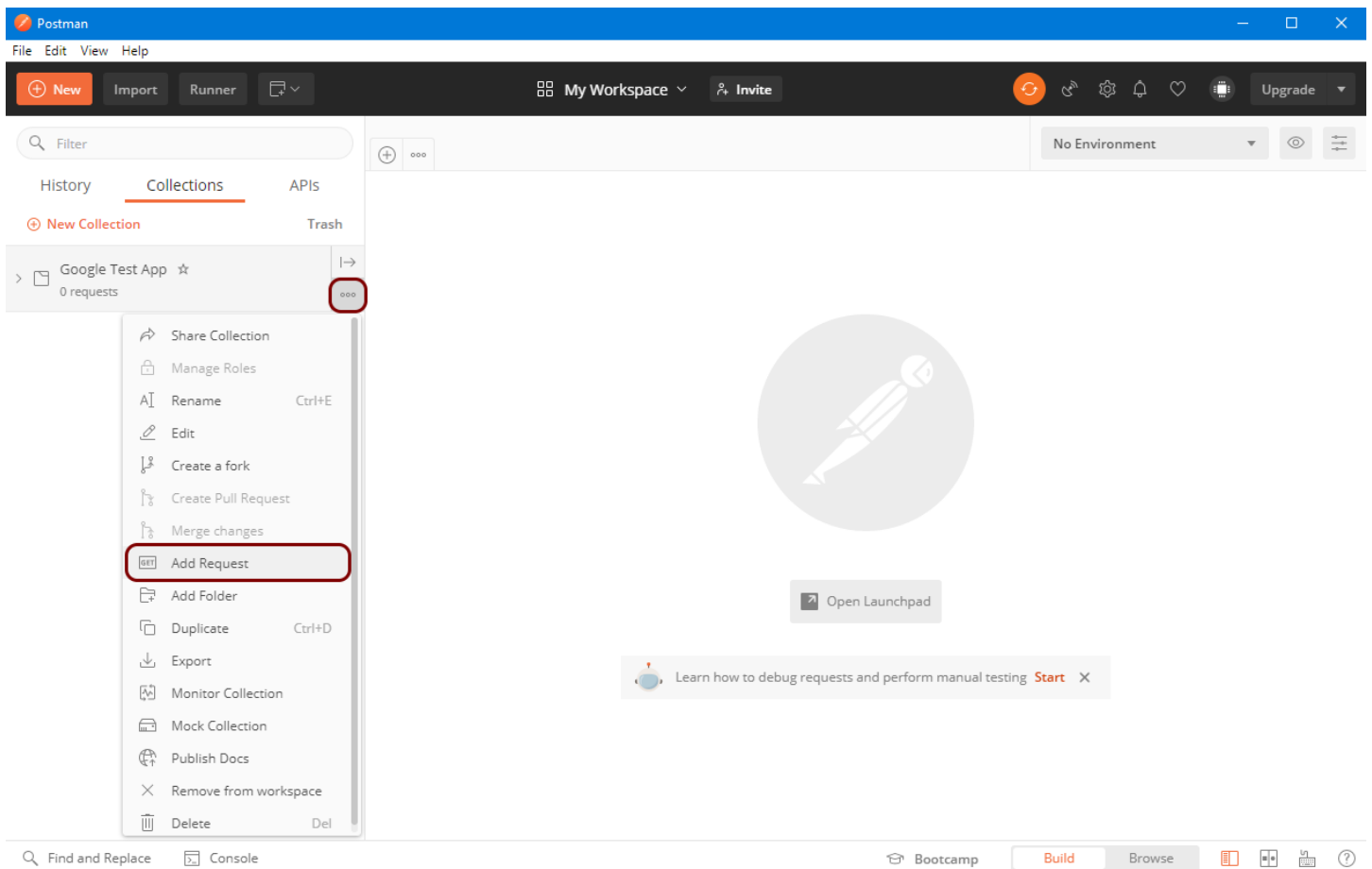
This request is used to obtain the authorization code and is the first step towards getting the bearer token. This request is sent with GET to <https://accounts.google.com/o/oauth2/v2/auth> and the query parameter will provide your client ID and other required information.

Request Token

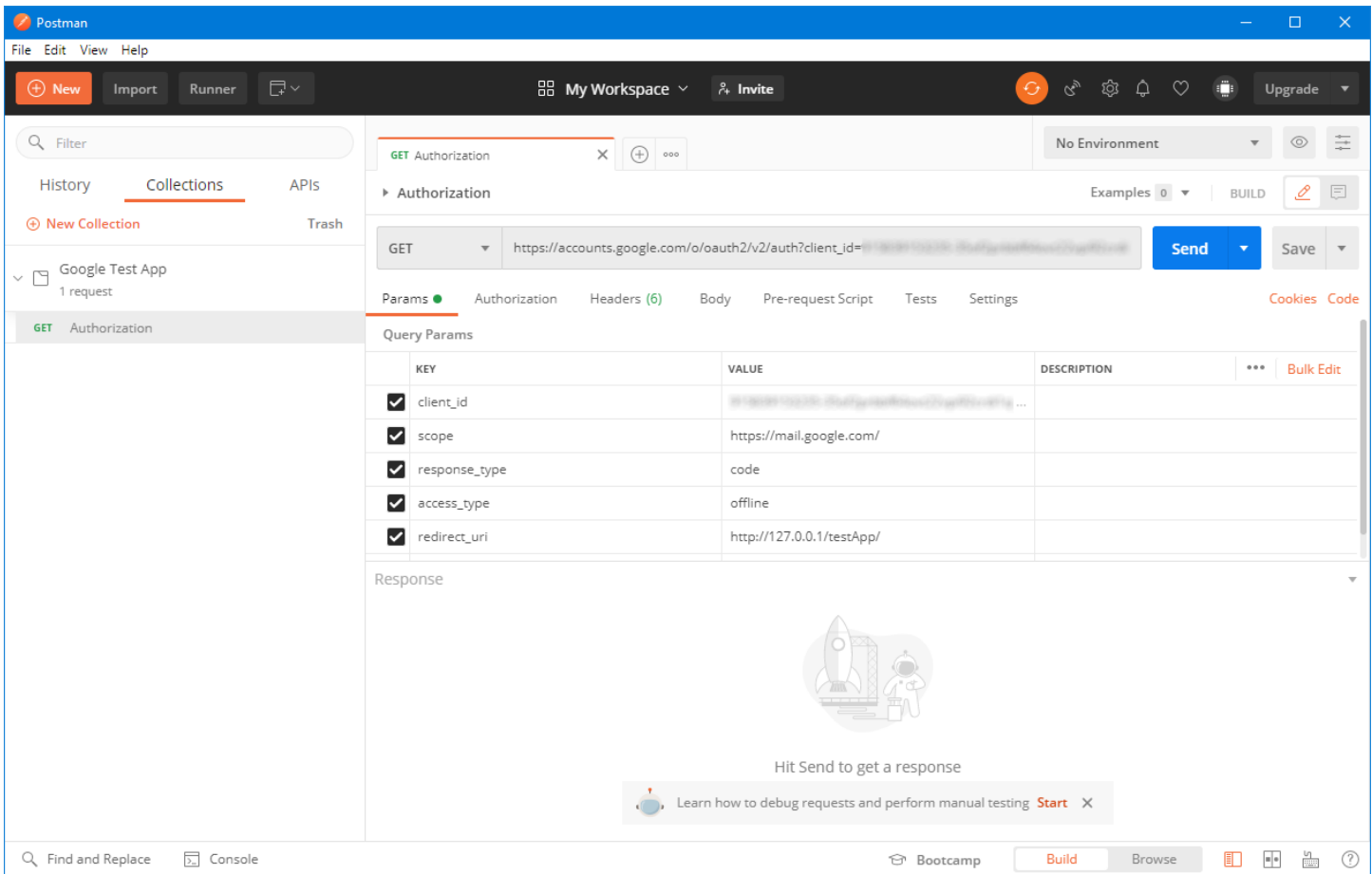
This request is used to obtain the bearer and refresh tokens needed to access the user’s account. This is done using a POST to <https://www.googleapis.com/oauth2/v4/token> and the post data will contain the authorization code provided in the previous step.

Refresh Token

This request is used to refresh the bearer token after it has expired. Bearer tokens have a short lifespan, typically an hour, and will expire. Refresh the token is also done using a POST to <https://www.googleapis.com/oauth2/v4/token> with the refresh token provided in the post data.



Create the first request, which is Authorization. Select **Add Request** from the menu on the left side of the display, and name it. You can also add a description if you wish.

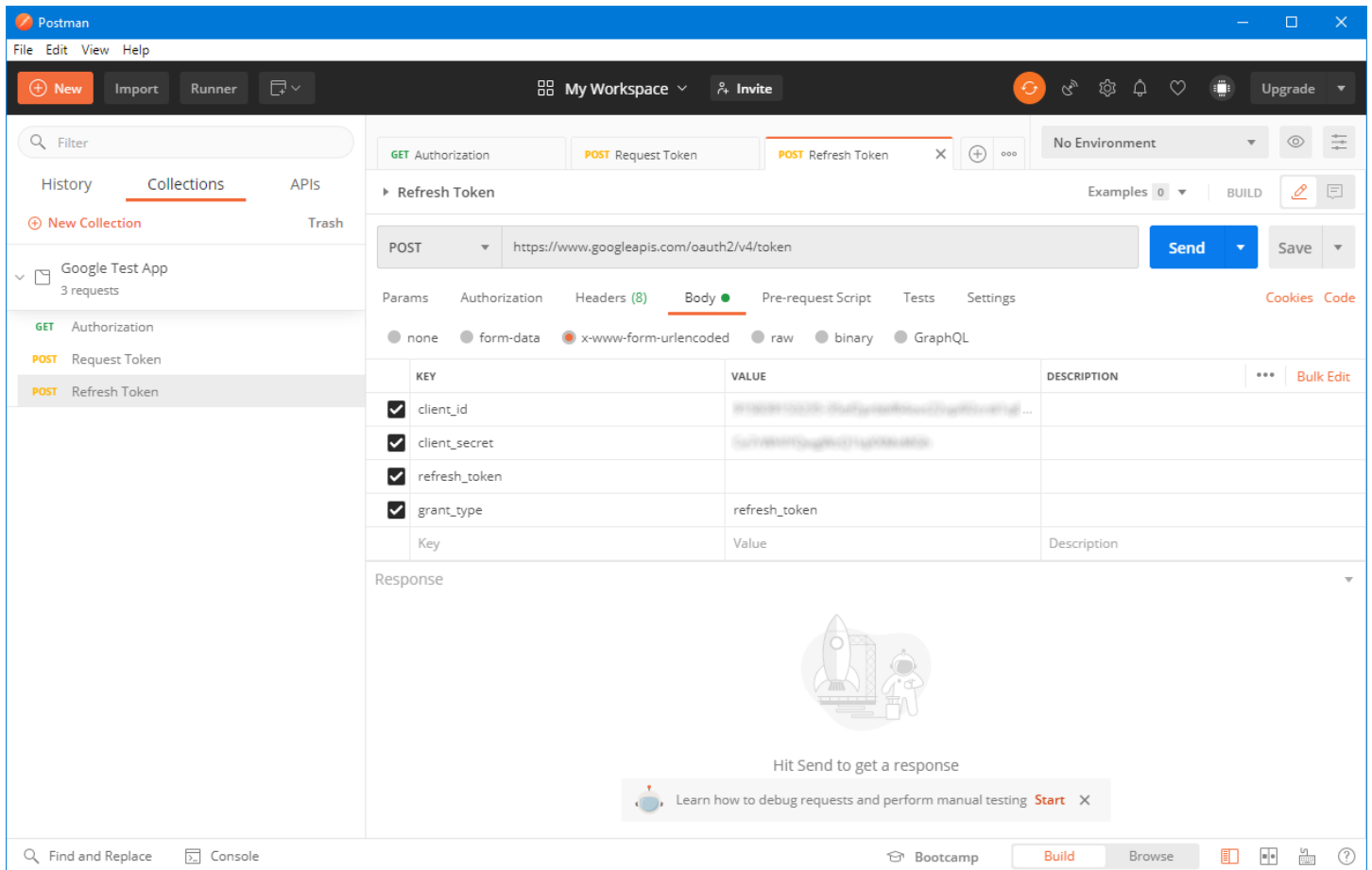


Name the request **Authorization** and it will default to using GET and enter the URL to Google’s authorization server `https://accounts.google.com/o/oauth2/v2/auth`

Populate the query parameters with five values which comprise the request.

- client_id** The client ID you obtained when you registered your application with Google.
- scope** This informs Google what kind of access you want to the user’s mail account. In this guide, we will use `https://mail.google.com/` because this is the required scope for IMAP and SMTP access.
- response_type** The response type should always be **code** for our purposes. This tells Google we want an authorization code.
- access_type** The access type should always be **offline** so Google will provide us with refresh tokens which can be used to obtain new bearer tokens after they expire.
- redirect_uri** This is the redirect URI we will use with the application.

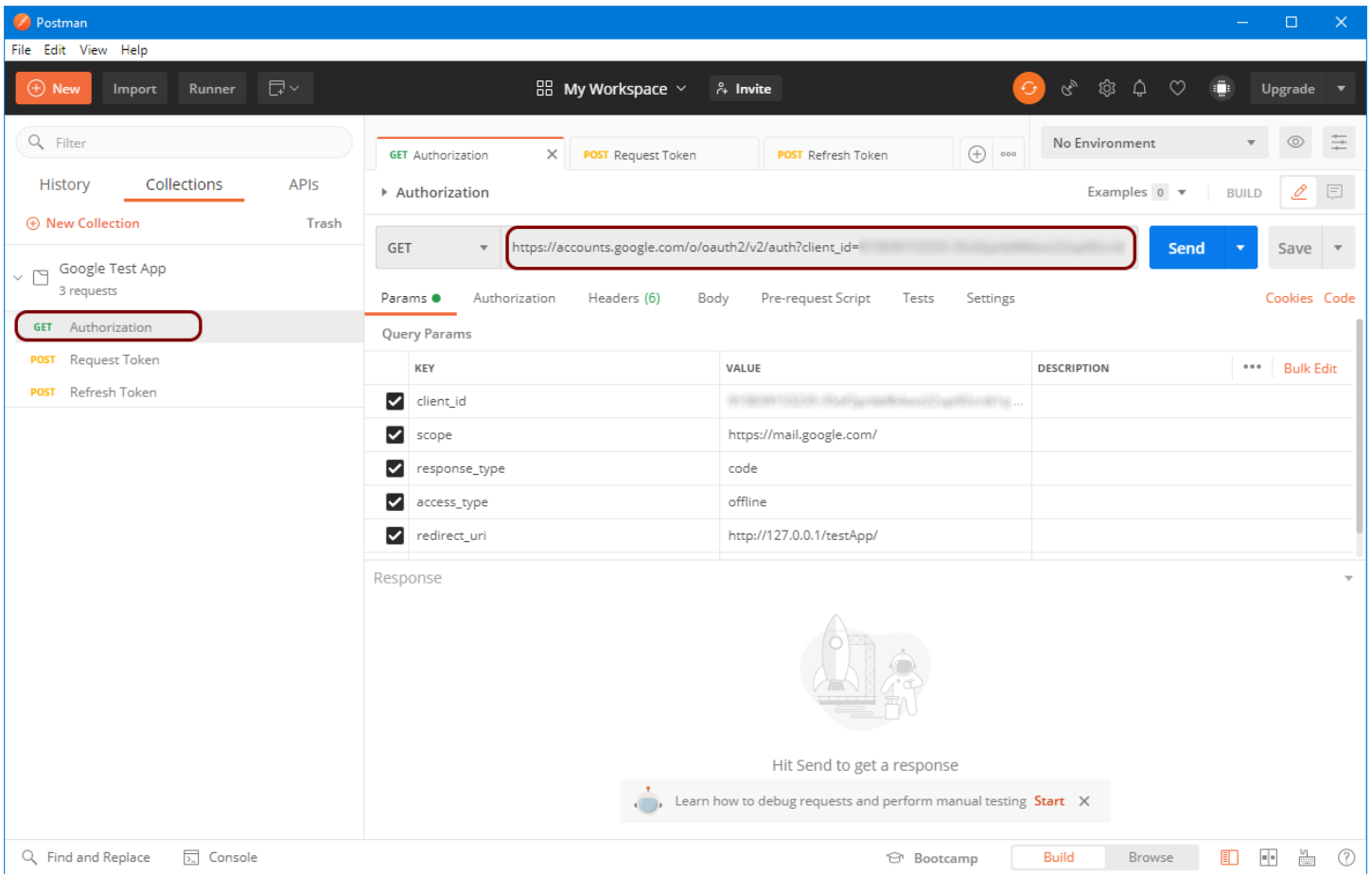
You should be sure to use a unique address for the redirect URI. We recommend following it with an abbreviated name for your application. It needs to be a valid URI and you should keep it short and concise. For this guide, we will use the redirect URI `http://127.0.0.1/testApp/`



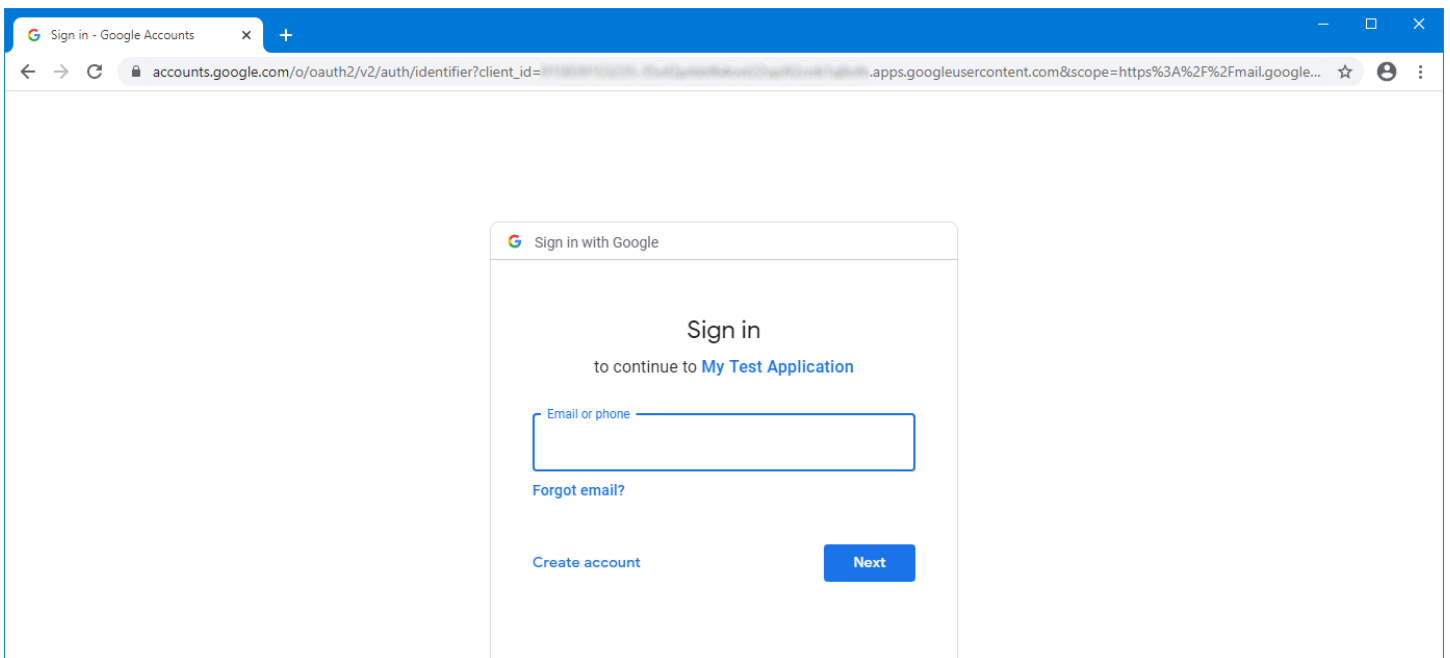
The final request type will be **Refresh Token** which is used to refresh an expired bearer token. This will also use a POST to `https://www.googleapis.com/oauth2/v4/token`

There are four values which comprise the request.

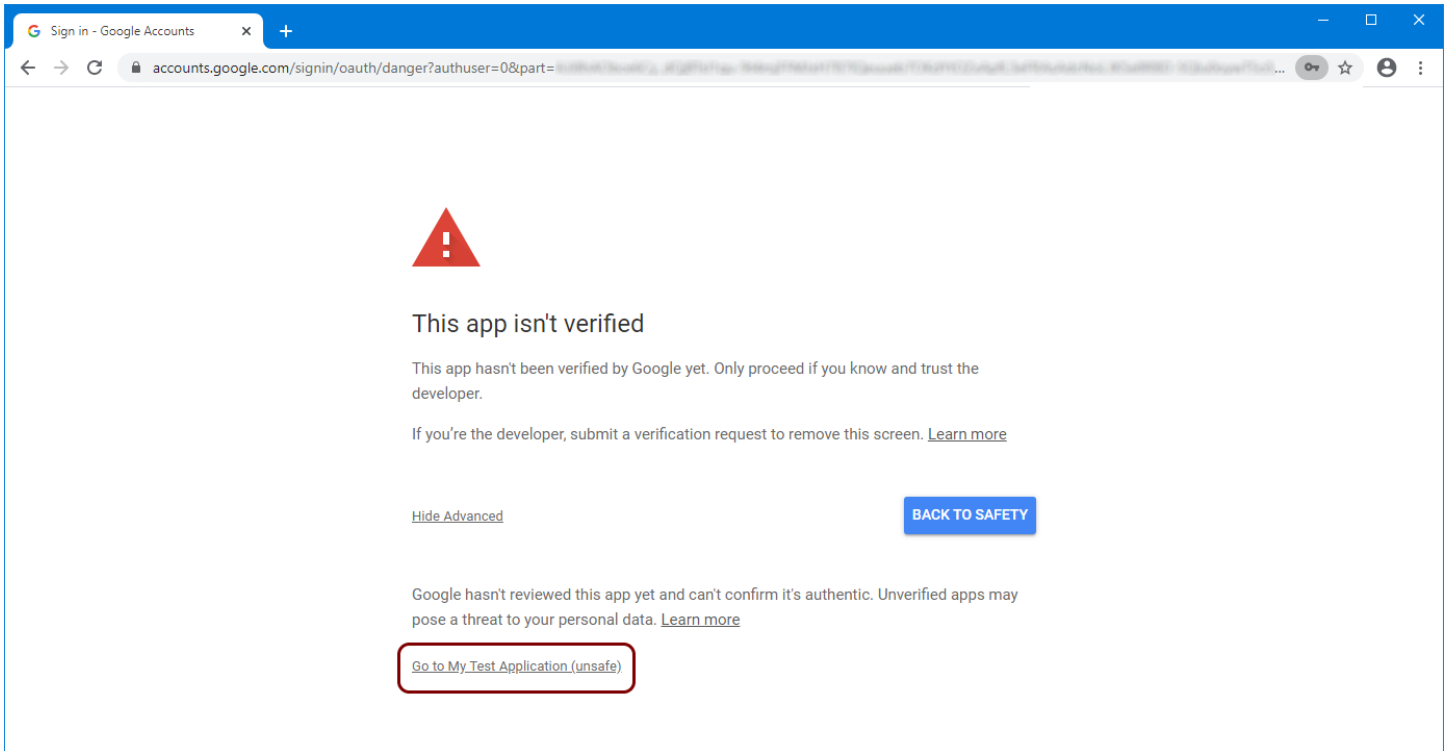
- client_id** The client ID you obtained when you registered your application with Google.
- client_secret** The client secret you obtained when you registered your application with Google.
- refresh_token** The refresh token you obtained when the initial authorization was granted. This token has a long validity period and is required to get a new bearer token after it has expired. We don't have the refresh token yet, so leave this blank.
- grant_type** This value should be **refresh_token** which tells Google we want a new bearer token.



Now that we have created all three request types and saved them in Postman, copy the URL from the **Authorization** request and paste it into a browser address bar. This step requires you to login with your Gmail account and must be done interactively in a browser.



If you have two-factor authentication enabled with your Gmail account, you will need to confirm the login attempt. After you've logged in, you will be asked to confirm you want **My Test Application** to have access to your account.



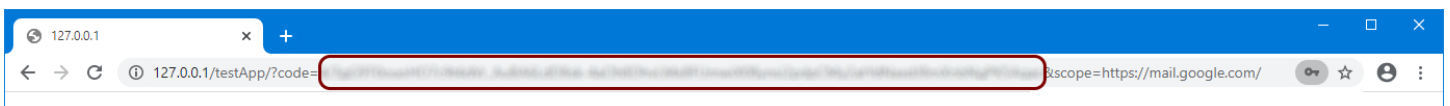
Because your application hasn't been verified, you will get a warning from Google about your test application accessing your account. This is normal during the development and testing period for your application. However, when it's time for you to release your application to the public, you will need to go through the verification process.

You can learn more about [unverified apps](#) on Google's website.

After you have completed the login process and confirmed you want your test application to have access to your Gmail account, the browser will redirect back to the URI you've specified. Because there won't be any web server able to respond, an error will be displayed. However, the information we need is returned in the address bar.

You will notice the address bar now has a value which looks like your return URI:

`http://127.0.0.1/testApp/?code=[your_authorization_code]&scope=https://mail.google.com/`



What follows the request URI between the `?code=` and `&scope=` portion is the authorization code we're interested in. This is going to be a string of letters, numbers, and symbols which you should copy and paste into **Request Token** request we've created in Postman.

With the authorization code pasted into the **code** field, press **Send** to send the request.

