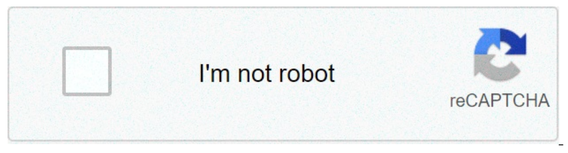

Continue

Flask redirect with parameters

[{ "type": "thumb-down", "id": "missingTheInformationINeed", "label":"Missing the information I need" },{ "type": "thumb-down", "id": "tooComplicatedTooManySteps", "label":"Too complicated / too many steps" },{ "type": "thumb-down", "id": "outOfDate", "label":"Out of date" },{ "type": "thumb-down", "id": "samplesCodeIssue", "label":"Samples/Code issue" },{ "type": "thumb-down", "id": "otherDown", "label":"Other" }] [{ "type": "thumb-up", "id": "easyToUnderstand", "label":"Easy to understand" },{ "type": "thumb-up", "id": "solvedMyProblem", "label":"Solved my problem" },{ "type": "thumb-up", "id": "otherUp", "label":"Other" }] This document explains how web server applications use Google API Client Libraries or Google OAuth 2.0 endpoints to implement OAuth 2.0 authorization to access Google APIs. OAuth 2.0 allows users to share specific data with an application while keeping their usernames, passwords, and other information private. For example, an application can use OAuth 2.0 to obtain permission from users to store files in their Google Drives. This OAuth 2.0 flow is specifically for user authorization. It is designed for applications that can store confidential information and maintain state. A properly authorized web server application can access an API while the user interacts with the application or after the user has left the application. Web server applications frequently also use service accounts to authorize API requests, particularly when calling Cloud APIs to access project-based data rather than user-specific data. Web server applications can use service accounts in conjunction with user authorization. Note: Given the security implications of getting the implementation correct, we strongly encourage you to use OAuth 2.0 libraries when interacting with Google's OAuth 2.0 endpoints. It is a best practice to use well-debugged code provided by others, and it will help you protect yourself and your users. For more information, see Client libraries. Client libraries The language-specific examples on this page use Google API Client Libraries to implement OAuth 2.0 authorization. To run the code samples, you must first install the client library for your language. When you use a Google API Client Library to handle your application's OAuth 2.0 flow, the client library performs many actions that the application would otherwise need to handle on its own. For example, it determines when the application can use or refresh stored access tokens as well as when the application must reacquire consent. The client library also generates correct redirect URLs and helps to implement redirect handlers that exchange authorization codes for access tokens. Client libraries are available for the following languages: Go Java .NET Node.js PHP Python Ruby Prerequisites Enable APIs for your project Any application that calls Google APIs needs to enable those APIs in the API Console. To enable an API for your project: Open the API Library in the Google API Console. If prompted, select a project, or create a new one. The API Library lists all available APIs, grouped by product family and popularity. If the API you want to enable isn't visible in the list, use search to find it, or click View All in the product family it belongs to. Select the API you want to enable, then click the Enable button. If prompted, enable billing. If prompted, read and accept the API's Terms of Service. Create authorization credentials Any application that uses OAuth 2.0 to access Google APIs must have authorization credentials that identify the application to Google's OAuth 2.0 server. The following steps explain how to create credentials for your project. Your applications can then use the credentials to access APIs that you have enabled for that project. Go to the Credentials page. Click Create credentials > OAuth client ID. Select the Web application application type. Fill in the form and click Create. Applications that use languages and frameworks like PHP, Java, Python, Ruby, and .NET must specify authorized redirect URIs. The redirect URIs are the endpoints to which the OAuth 2.0 server can send responses. These endpoints must adhere to Google's validation rules. For testing, you can specify URIs that refer to the local machine, such as . With that in mind, please note that all of the examples in this document use as the redirect URI. We recommend that you design your app's auth endpoints so that your application does not expose authorization codes to other resources on the page. After creating your credentials, download the client_secret.json file from the API Console. Securely store the file in a location that only your application can access. Important: Do not store the client_secret.json file in a publicly-accessible location. In addition, if you share the source code to your application — for example, on GitHub — store the client_secret.json file outside of your source tree to avoid inadvertently sharing your client credentials. Identify access scopes Scopes enable your application to only request access to the resources that it needs while also enabling users to control the amount of access that they grant to your application. Thus, there may be an inverse relationship between the number of scopes requested and the likelihood of obtaining user consent. Before you start implementing OAuth 2.0 authorization, we recommend that you identify the scopes that your app will need permission to access. We also recommend that your application request access to authorization scopes via an incremental authorization process, in which your application requests access to user data in context. This best practice helps users to more easily understand why your application needs the access it is requesting. The OAuth 2.0 API Scopes document contains a full list of scopes that you might use to access Google APIs. If your public application uses scopes that permit access to certain user data, it must complete a verification process. If you see unverified app on the screen when testing your application, you must submit a verification request to remove it. Find out more about unverified apps and get answers to frequently asked questions about app verification in the Help Center. Language-specific requirements To run any of the code samples in this document, you'll need a Google account, access to the Internet, and a web browser. If you are using one of the API client libraries, also see the language-specific requirements below. To run the PHP code samples in this document, you'll need: PHP 5.4 or greater with the command-line interface (CLI) and JSON extension installed. The Composer dependency management tool. The Google APIs Client Library for PHP: php composer.phar require google/apiclient:^2.0 To run the Python code samples in this document, you'll need: Python 2.6 or greater The pip package management tool. The Google APIs Client Library for Python: pip install --upgrade google-api-python-client The google-auth, google-auth-oauthlib, and google-auth-httplib2 for user authorization. pip install --upgrade google-auth google-auth-oauthlib google-auth-httplib2 The Flask Python web application framework. pip install --upgrade flask The requests HTTP library. pip install --upgrade requests To run the Ruby code samples in this document, you'll need: Ruby 2.2.2 or greater The Google APIs Client Library for Ruby: gem install google-api-client The Sinatra Ruby web application framework. gem install sinatra You do not need to install any libraries to be able to directly call the OAuth 2.0 endpoints. The following steps show how your application interacts with Google's OAuth 2.0 server to obtain a user's consent to perform an API request on the user's behalf. Your application must have that consent before it can execute a Google API request that requires user authorization. The list below quickly summarizes these steps: Your application identifies the permissions it needs. Your application redirects the user to Google along with the list of requested permissions. The user decides whether to grant the permissions to your application. Your application finds out what the user decided. If the user granted the requested permissions, your application retrieves tokens needed to make API requests on the user's behalf. Step 1: Set authorization parameters Your first step is to create the authorization request. That request sets parameters that identify your application and define the permissions that the user will be asked to grant to your application. If you use a Google client library for OAuth 2.0 authentication and authorization, you create and configure an object that defines these parameters. If you call the Google OAuth 2.0 endpoint directly, you'll generate a URL and set the parameters on that URL. The tabs below define the supported authorization parameters for web server applications. The language-specific examples also show how to use a client library or authorization library to configure an object that sets those parameters. The code snippet below creates a Google_Client() object, which defines the parameters in the authorization request. That object uses information from your client_secret.json file to identify your application. (See creating authorization credentials for more about that file.) The object also identifies the scopes that your application is requesting permission to access and the URL to your application's auth endpoint, which will handle the response from Google's OAuth 2.0 server. Finally, the code sets the optional access_type and include_granted_scopes parameters. For example, this code requests read-only, offline access to a user's Google Drive: $client = new Google_Client(); $client->setAuthConfig('client_secret.json'); $client->addScope(Google_Service_Drive::DRIVE_METADATA_READONLY); $client->setRedirectUri('http://' . $_SERVER['HTTP_HOST'] . '/oauth2callback.php'); // offline access will give you both an access and refresh token so that // your app can refresh the access token without user interaction. $client->setAccessType('offline'); // Using "consent" ensures that your application always receives a refresh token. // If you are not using offline access, you can omit this. $client->setApprovalPrompt("consent"); $client->setIncludeGrantedScopes(true); // incremental auth The request specifies the following information: Parameters client_id Required The client ID for your application. You can find this value in the API Console Credentials page. In PHP, call the setAuthConfig function to load authorization credentials from a client_secret.json file. $client = new Google_Client(); $client->setAuthConfig('client_secret.json'); redirect_uri Required Determines where the API server redirects the user after the user completes the authorization flow. The value must exactly match one of the authorized redirect URIs for the OAuth 2.0 client, which you configured in your client's API Console Credentials page. If this value doesn't match an authorized redirect URI for the provided client_id you will get a redirect_uri_mismatch error. Note that the http or https scheme, case, and trailing slash ('/') must all match. To set this value in PHP, call the setRedirectUri function. Note that you must specify a valid redirect URI for the provided client_id. $client->setRedirectUri(' '); scope Required A space-delimited list of scopes that identify the resources that your application could access on the user's behalf. These values inform the consent screen that Google displays to the user. Scopes enable your application to only request access to the resources that it needs while also enabling users to control the amount of access that they grant to your application. Thus, there is an inverse relationship between the number of scopes requested and the likelihood of obtaining user consent. To set this value in PHP, call the addScope function: $client->addScope(Google_Service_Drive::DRIVE_METADATA_READONLY); We recommend that your application request access to authorization scopes in context whenever possible. By requesting access to user data in context, via incremental authorization, you help users to more easily understand why your application needs the access it is requesting. access_type Recommended Indicates whether your application can refresh access tokens when the user is not present at the browser. Valid parameter values are online, which is the default value, and offline. Set the value to offline if your application needs to refresh access tokens when the user is not present at the browser. This is the method of refreshing access tokens described later in this document. This value instructs the Google authorization server to return a refresh token and an access token the first time that your application exchanges an authorization code for tokens. To set this value in PHP, call the setAccessType function: $client->setAccessType('offline'); state Recommended Specifies any string value that your application uses to maintain state between your authorization request and the authorization server's response. The server returns the exact value that you send as a name=value pair in the URL query component (?) of the redirect_uri after the user consents to or denies your application's access request. You can use this parameter for several purposes, such as directing the user to the correct resource in your application, sending nonces, and mitigating cross-site request forgery. Since your redirect_uri can be guessed, using a state value can increase your assurance that an incoming connection is the result of an authentication request. If you generate a random string or encode the hash of a cookie or another value that captures the client's state, you can validate the response to additionally ensure that the request and response originated in the same browser, providing protection against attacks such as cross-site request forgery. See the OpenID Connect documentation for an example of how to create and confirm a state token. To set this value in PHP, call the setState function: $client->setState($sample_passthrough_value); include_granted_scopes Optional Enables applications to use incremental authorization to request access to additional scopes in context. If you set this parameter's value to true and the authorization request is granted, then the new access token will also cover any scopes to which the user previously granted the application access. See the incremental authorization section for examples. To set this value in PHP, call the setIncludeGrantedScopes function: $client->setIncludeGrantedScopes(true); prompt Optional A space-delimited, case-sensitive list of prompts to present the user. If you don't specify this parameter, the user will be prompted only the first time your project requests access. See Prompting re-consent for more information. To set this value in PHP, call the setApprovalPrompt function: $client->setApprovalPrompt('consent'); Possible values are: none Do not display any authentication or consent screens. Must not be specified with other values. consent Prompt the user for consent. select_account Prompt the user to select an account. The following code snippet uses the google-auth-oauthlib.flow module to construct the authorization request. The code constructs a Flow object, which identifies your application using information from the client_secret.json file that you downloaded after creating authorization credentials. That object also identifies the scopes that your application is requesting permission to access and the URL to your application's auth endpoint, which will handle the response from Google's OAuth 2.0 server. Finally, the code sets the optional access_type and include_granted_scopes parameters. For example, this code requests read-only, offline access to a user's Google Drive: import google.oauth2.credentials import google_auth_oauthlib.flow # Use the client_secret.json file to identify the application requesting # authorization. The client ID (from that file) and access scopes are required. flow = google_auth_oauthlib.flow.Flow.from_client_secrets_file('client_secret.json', scopes=[']) # Indicate where the API server will redirect the user after the user completes # the authorization flow. The redirect URI is required. The value must exactly # match one of the authorized redirect URIs for the OAuth 2.0 client, which you # configured in the API Console. If this value doesn't match an authorized URI, # you will get a 'redirect_uri_mismatch' error. flow.redirect_uri = ' # Generate URL for request to Google's OAuth 2.0 server. # Use kwargs to set optional request parameters. authorization_url, state = flow.authorization_url(# Enable offline access so that you can refresh an access token without # re-prompting the user for permission. Recommended for web server apps. access_type='offline', # Enable incremental authorization. Recommended as a best practice. include_granted_scopes='true') The request specifies the following information: Parameters client_id Required The client ID for your application. You can find this value in the API Console Credentials page. In Python, call the from_client_secrets_file method to retrieve the client ID from a client_secret.json file. (You can also use the from_client_config method, which passes the client configuration as it originally appeared in a client secrets file but doesn't access the file itself.) flow = google_auth_oauthlib.flow.Flow.from_client_secrets_file('client_secret.json', scopes=[']) redirect_uri Required Determines where the API server redirects the user after the user completes the authorization flow. The value must exactly match one of the authorized redirect URIs for the OAuth 2.0 client, which you configured in your client's API Console Credentials page. If this value doesn't match an authorized redirect URI for the provided client_id you will get a redirect_uri_mismatch error. Note that the http or https scheme, case, and trailing slash ('/') must all match. To set this value in Python, set the flow object's redirect_uri property: flow.redirect_uri = ' scope Required A list of scopes that identify the resources that your application could access on the user's behalf. These values inform the consent screen that Google displays to the user. Scopes enable your application to only request access to the resources that it needs while also enabling users to control the amount of access that they grant to your application. Thus, there is an inverse relationship between the number of scopes requested and the likelihood of obtaining user consent. In Python, use the same method you use to set the client_id to specify the list of scopes. flow = google_auth_oauthlib.flow.Flow.from_client_secrets_file('client_secret.json', scopes=[']) We recommend that your application request access to authorization scopes in context whenever possible. By requesting access to user data in context, via incremental authorization, you help users to more easily understand why your application needs the access it is requesting. access_type Recommended Indicates whether your application can refresh access tokens when the user is not present at the browser. Valid parameter values are online, which is the default value, and offline. Set the value to offline if your application needs to refresh access tokens when the user is not present at the browser. This is the method of refreshing access tokens described later in this document. This value instructs the Google authorization server to return a refresh token and an access token the first time that your application exchanges an authorization code for tokens. In Python, set the access_type parameter by specifying access_type as a keyword argument when calling the flow.authorization_url method: authorization_url, state = flow.authorization_url(access_type='offline', include_granted_scopes='true') state Recommended Specifies any string value that your application uses to maintain state between your authorization request and the authorization server's response. The server returns the exact value that you send as a name=value pair in the URL query component (?) of the redirect_uri after the user consents to or denies your application's access request. You can use this parameter for several purposes, such as directing the user to the correct resource in your application, sending nonces, and mitigating cross-site request forgery. Since your redirect_uri can be guessed, using a state value can increase your assurance that an incoming connection is the result of an authentication request. If you generate a random string or encode the hash of a cookie or another value that captures the client's state, you can validate the response to additionally ensure that the request and response originated in the same browser, providing protection against attacks such as cross-site request forgery. See the OpenID Connect documentation for an example of how to create and confirm a state token. In Python, set the state parameter by specifying state as a keyword argument when calling the flow.authorization_url method: authorization_url, state = flow.authorization_url(access_type='offline', state=sample_passthrough_value, include_granted_scopes='true') include_granted_scopes Optional Enables applications to use incremental authorization to request access to additional scopes in context. If you set this parameter's value to true and the authorization request is granted, then the new access token will also cover any scopes to which the user previously granted the application access. See the incremental authorization section for examples. In Python, set the include_granted_scopes parameter by specifying include_granted_scopes as a keyword argument when calling the flow.authorization_url method: authorization_url, state = flow.authorization_url(access_type='offline', include_granted_scopes='true') prompt Optional A space-delimited, case-sensitive list of prompts to present the user. If you don't specify this parameter, the user will be prompted only the first time your project requests access. See Prompting re-consent for more information. In Python, set the prompt parameter by specifying prompt as a keyword argument when calling the flow.authorization_url method: authorization_url, state = flow.authorization_url(access_type='offline', prompt='consent', include_granted_scopes='true') Possible values are: none Do not display any authentication or consent screens. Must not be specified with other values. consent Prompt the user for consent. select_account Prompt the user to select an account. Use the client_secrets.json file that you created to configure a client object in your application. When you configure a client object, you specify the scopes your application needs to access, along with the URL to your application's auth endpoint, which will handle the response from the OAuth 2.0 server. For example, this code requests read-only, offline access to a user's Google Drive: require 'google/apis/drive_v2' require 'google/api_client/client_secrets' client_secrets = Google::APIClient::ClientSecrets.load auth_client = client_secrets.to_authorization auth_client.update!(:scope => ' , :redirect_uri => ' , :additional_parameters => { "access_type" => "offline", # offline access "include_granted_scopes" => "true" # incremental auth }) Your application uses the client object to perform OAuth 2.0 operations, such as generating authorization request URLs and applying access tokens to HTTP requests. Google's OAuth 2.0 endpoint is at . This endpoint is accessible only over HTTPS. Plain HTTP connections are refused. The Google authorization server supports the following query string parameters for web server applications: Parameters client_id Required The client ID for your application. You can find this value in the API Console Credentials page. redirect_uri Required Determines where the API server redirects the user after the user completes the authorization flow. The value must exactly match one of the authorized redirect URIs for the OAuth 2.0 client, which you configured in your client's API Console Credentials page. If this value doesn't match an authorized redirect URI for the provided client_id you will get a redirect_uri_mismatch error. Note that the http or https scheme, case, and trailing slash ('/') must all match. response_type Required Determines whether the Google OAuth 2.0 endpoint returns an authorization code. Set the parameter value to code for web server applications. scope Required A space-delimited list of scopes that identify the resources that your application could access on the user's behalf. These values inform the consent screen that Google displays to the user. Scopes enable your application to only request access to the resources that it needs while also enabling users to control the amount of access that they grant to your application. Thus, there is an inverse relationship between the number of scopes requested and the likelihood of obtaining user consent. We recommend that your application request access to authorization scopes in context whenever possible. By requesting access to user data in context, via incremental authorization, you help users to more easily understand why your application needs the access it is requesting. access_type Recommended Indicates whether your application can refresh access tokens when the user is not present at the browser. Valid parameter values are online, which is the default value, and offline. Set the value to offline if your application needs to refresh access tokens when the user is not present at the browser. This is the method of refreshing access tokens described later in this document. This value instructs the Google authorization server to return a refresh token and an access token the first time that your application exchanges an authorization code for tokens. state Recommended Specifies any string value that your application uses to maintain state between your authorization request and the authorization server's response. The server returns the exact value that you send as a name=value pair in the URL query component (?) of the redirect_uri after the user consents to or denies your application's access request. You can use this parameter for several purposes, such as directing the user to the correct resource in your application, sending nonces, and mitigating cross-site request forgery. Since your redirect_uri can be guessed, using a state value can increase your assurance that an incoming connection is the result of an authentication request. If you generate a random string or encode the hash of a cookie or another value that captures the client's state, you can validate the response to additionally ensure that the request and response originated in the same browser, providing protection against attacks such as cross-site request forgery. See the OpenID Connect documentation for an example of how to create and confirm a state token. include_granted_scopes Optional Enables applications to use incremental authorization to request access to additional scopes in context. If you set this parameter's value to true and the authorization request is granted, then the new access token will also cover any scopes to which the user previously granted the application access. See the incremental authorization section for examples. prompt Optional A space-delimited, case-sensitive list of prompts to present the user. If you don't specify this parameter, the user will be prompted only the first time your project requests access. See Prompting re-consent for more information. Possible values are: none Do not display any authentication or consent screens. Must not be specified with other values. consent Prompt the user for consent. select_account Prompt the user to select an account. Redirect the user to Google's OAuth 2.0 server to initiate the authentication and authorization process. Typically, this occurs when your application first needs to access the user's data. In the case of incremental authorization, this step also occurs when your application first needs to access additional resources that it does not yet have permission to access. Generate a URL to request access from Google's OAuth 2.0 server: $auth_url = $client->createAuthUrl(); Redirect the user to $auth_url: header('Location: ' . filter_var($auth_url, FILTER_SANITIZE_URL)); This example shows how to redirect the user to the authorization URL using the Flask web application framework: return flask.redirect(authorization_url) Generate a URL to request access from Google's OAuth 2.0 server: auth_uri = auth_client.authorization_uri.to_s Redirect the user to auth_uri. An example URL is shown below, with line breaks and spaces for readability. scope=https%3A//www.googleapis.com/auth/drive.metadata.readonly& access_type=offline& include_granted_scopes=true& response_type=code& state=state_parameter_passthrough_value& redirect_uri=https%3A//oauth2.example.com/code& client_id=client_id After you create the request URL, redirect the user to it. Google's OAuth 2.0 server authenticates the user and obtains consent from the user for your application to access the requested scopes. The response is sent back to your application using the redirect URL you specified. Step 3: Google prompts user for consent In this step, the user decides whether to grant your application the requested access. At this stage, Google displays a consent window that shows the name of your application and the Google API services that it is requesting permission to access with the user's authorization credentials and a summary of the scopes of access to be granted. The user can then consent to grant access to one or more scopes requested by your application or refuse the request. Your application doesn't need to do anything at this stage as it waits for the response from Google's OAuth 2.0 server indicating whether any access was granted. That response is explained in the following step. Requests to Google's OAuth 2.0 authorization endpoint may display user-facing error messages instead of the expected authentication and authorization flows. Common error codes and suggested resolutions are listed below. The Google Account is unable to authorize one or more scopes requested due to the policies of their Google Workspace administrator. See the Google Workspace Admin help article Control which third-party & internal apps access Google Workspace data for more information about how an administrator may restrict access to all scopes or sensitive and restricted scopes until access is explicitly granted to your OAuth client ID. The authorization endpoint is displayed inside an embedded user-agent disallowed by Google's OAuth 2.0 Policies. Android developers may encounter this error message when opening authorization requests in android.webkit.WebView. Developers should instead use Android libraries such as Google Sign-In for Android or OpenID Foundation's AppAuth for Android. Web developers may encounter this error when an Android app opens a general web link in an embedded user-agent and a user navigates to Google's OAuth 2.0 authorization endpoint from your site. Developers should allow general links to open in the default link handler of the operating system, which includes both Android App Links handlers or the default browser app. The Android Custom Tabs library is also a supported option. iOS and macOS developers may encounter this error when opening authorization requests in WKWebView. Developers should instead use iOS libraries such as Google Sign-In for iOS or OpenID Foundation's AppAuth for iOS. Web developers may encounter this error when an iOS or macOS app opens a general web link in an embedded user-agent and a user navigates to Google's OAuth 2.0 authorization endpoint from your site. Developers should allow general links to open in the default link handler of the operating system, which includes both Universal Links handlers or the default browser app. The SFSafariViewController library is also a supported option. The OAuth client ID in the request is part of a project limiting access to Google Accounts in a specific Google Cloud Organization. For more information about this configuration option see the User type section in the Setting up your OAuth consent screen help article. The redirect_uri passed in the authorization request does not match an authorized redirect URI for the OAuth client ID. Review authorized redirect URIs in the Google API Console Credentials page. Step 4: Handle the OAuth 2.0 server response The OAuth 2.0 server responds to your application's access request by using the URL specified in the request. If the user approves the access request, then the response contains an authorization code. If the user does not approve the request, the response contains an error message. The authorization code or error message that is returned to the web server appears on the query string, as shown below: An error response: An authorization code response: Important: If your response endpoint renders an HTML page, any resources on that page will be able to see the authorization code in the URL. Scripts can read the URL directly, and the URL in the Referer HTTP header may be sent to any or all resources on the page. Carefully consider whether you want to send authorization credentials to all resources on that page (especially third-party scripts such as social plugins and analytics). To avoid this issue, we recommend that the server first handle the request, then redirect to another URL that doesn't include the response parameters. Sample OAuth 2.0 server response You can test this flow by clicking on the following sample URL, which requests read-only access to view metadata for files in your Google Drive: scope=https%3A//www.googleapis.com/auth/drive.metadata.readonly& access_type=offline& include_granted_scopes=true& response_type=code& state=state_parameter_passthrough_value& redirect_uri=https%3A//oauth2.example.com/code& client_id=client_id After completing the OAuth 2.0 flow, you should be redirected to which will likely yield a 404 NOT FOUND error unless your local machine serves a file at that address. The next step provides more detail about the information returned in the URI when the user is redirected back to your application. After the web server receives the authorization code, it can exchange the authorization code for an access token. To exchange an authorization code for an access token, use the authenticate method: $client->authenticate($_GET['code']); You can retrieve the access token with the getAccessToken method: $access_token = $client->getAccessToken(); On your callback page, use the google-auth library to verify the authorization server response. Then, use the flow.fetch_token method to exchange the authorization code in that response for an access token: state = flask.session['state'] flow = google_auth_oauthlib.flow.Flow.from_client_secrets_file('client_secret.json', scopes=['], state=state) flow.redirect_uri = flask.url_for('oauth2callback', _external=True) authorization_response = flask.request.url flow.fetch_token(authorization_response=authorization_response) # Store the credentials in the session. # ACTION ITEM for developers: # Store user's access and refresh tokens in your data store if # incorporating this code into your real app. credentials = flow.credentials flask.session['credentials'] = { 'token': credentials.token, 'refresh_token': credentials.refresh_token, 'token_uri': credentials.token_uri, 'client_id': credentials.client_id, 'client_secret': credentials.client_secret, 'scopes': credentials.scopes} To exchange an authorization code for an access token, use the fetch_access_token! method: auth_client.code = auth_code auth_client.fetch_access_token! To exchange an authorization code for an access token, call the endpoint and set the following parameters: Fields client_id The client ID obtained from the API Console Credentials page. client_secret The client secret obtained from the API Console Credentials page. code The authorization code returned from the initial request. grant_type As defined in the OAuth 2.0 specification, this field's value must be set to authorization_code. redirect_uri One of the redirect URIs listed for your project in the API Console Credentials page for the given client_id. The following snippet shows a sample request: POST /token HTTP/1.1 Host: oauth2.googleapis.com Content-Type: application/x-www-form-urlencoded code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7& client_id=your_client_id& client_secret=your_client_secret& redirect_uri=https%3A//oauth2.example.com/code& grant_type=authorization_code Google responds to this request by returning a JSON object that contains a short-lived access token and a refresh token. Note that the refresh token is only returned if your application set the access_type parameter to offline in the initial request to Google's authorization server. The response contains the following fields: Fields access_token The token that your application sends to authorize a Google API request. expires_in The remaining lifetime of the access token in seconds. refresh_token A token that you can use to obtain a new access token. Refresh tokens are valid until the user revokes access. Again, this field is only present in this response if you set the access_type parameter to offline in the initial request to Google's authorization server. scope The scopes of access granted by the access_token expressed as a list of space-delimited, case-sensitive strings. token_type The type of token returned. At this time, this field's value is always set to Bearer. Important: Your application should store both tokens in a secure, long-lived location that is accessible between different invocations of your application. The refresh token enables your application to obtain a new access token if the one that you have expires. As such, if your application loses the refresh token, the user will need to repeat the OAuth 2.0 consent flow so that your application can obtain a new refresh token. The following snippet shows a sample response: { "access_token": "1/fFAGRNJru1FTz70BzhT3Zg", "expires_in": 3920, "token_type": "Bearer", "scope": " , "refresh_token": "1//xEoDL4iW3cxlI7yDbSRFYNG01kVKM2C-259HOF2aQbI" } Note: Your application should ignore any unrecognized fields included in the response. Use the access token to call Google APIs by completing the following steps: If you need to apply an access token to a new Google_Client object—for example, if you stored the access token in a user session—use the setAccessToken method: $client->setAccessToken($access_token); Build a service object for the API that you want to call. You build a service object by providing an authorized Google_Client object to the constructor for the API you want to call. For example, to call the Drive API: $drive = new Google_Service_Drive($client); Make requests to the API service using the interface provided by the service object. For example, to list the files in the authenticated user's Google Drive: $files = $drive->files->listFiles(array())->getItems(); After obtaining an access token, your application can use that token to authorize API requests on behalf of a given user account or service account. Use the user-specific authorization credentials to build a service object for the API that you want to call, and then use that object to make authorized API requests. Build a service object for the API that you want to call. You build a service object by calling the googleapiclient.discovery library's build method with the name and version of the API and the user credentials: For example, to call version 2 of the Drive API: from googleapiclient.discovery import build drive = build('drive', 'v2', credentials=credentials) Make requests to the API service using the interface provided by the service object. For example, to list the files in the authenticated user's Google Drive: files = drive.files().list().execute() Use the auth_client object to call Google APIs by completing the following steps: Build a service object for the API that you want to call. For example, to call version 2 of the Drive API: drive = Google::Apis::DriveV2::DriveService.new Set the credentials on the service: drive.authorization = auth_client Make requests to the API service using the interface provided by the service object. For example, to list the files in the authenticated user's Google Drive: files = drive.list_files Alternately, authorization can be provided on a per-method basis by supplying the options parameter to a method: files = drive.list_files(options: { authorization: auth_client }) After your application obtains an access token, you can use the token to make calls to a Google API on behalf of a given user account if the scope(s) of access required by the API have been granted. To do this, include the access token in a request to the API by including either an access_token query parameter or an Authorization HTTP header Bearer value. When possible, the HTTP header is preferable, because query strings tend to be visible in server logs. In most cases you can use a client library to set up your calls to Google APIs (for example, when calling the Drive Files API). You can try out all the Google APIs and view their scopes at the OAuth 2.0 Playground. HTTP GET examples A call to the drive.files endpoint (the Drive Files API) using the Authorization: Bearer HTTP header might look like the following. Note that you need to specify your own access token: GET /drive/v2/files HTTP/1.1 Host: www.googleapis.com Authorization: Bearer access_token Here is a call to the same API for the authenticated user using the access_token query string parameter: GET curl examples You can test these commands with the curl command-line application. Here's an example that uses the HTTP header option (preferred): curl -H "Authorization: Bearer access_token" Or, alternatively, the query string parameter option: curl The following example prints a JSON-formatted list of files in a user's Google Drive after the user authenticates and gives consent for the application to access the user's Drive metadata. To run this example: In the API Console, add the URL of the local machine to the list of redirect URLs. For example, add . Create a new directory and change to it. For example: mkdir ~/php-oauth2-example cd ~/php-oauth2-example Install the Google API Client Library for PHP using Composer: composer require google/apiclient:^2.0 Create the files index.php and oauth2callback.php with the content below. Run the example with a web server configured to serve PHP. If you use PHP 5.4 or newer, you can use PHP's built-in test web server: php -S localhost:8080 ~/php-oauth2-example index.php