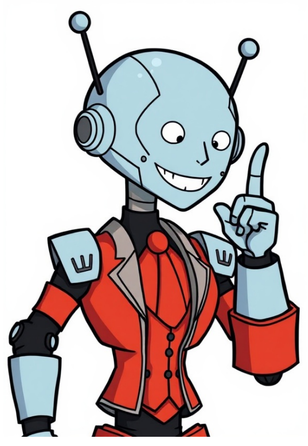
Continue



Dear all, I just found this two different sentences: 1:- The secret to keeping the weight down and energy up is to serve meals that are calorie intensive. 2.- The secret to do this trick you need to choose a month with five Wednesdays in it. The question is, when I should use secret to + (-ing verb), o secret to + (base verb)? I looked them up in grammar books and dictionaries, but I couldn't find any good answer. I'd really appreciate your response. Usually, use the "ing" form. For example "The secret to do this trick" is not correct and would be better with "The secret to doing this trick" The reason is "The secret to doing..." is a noun phrase and is the subject of the sentence. In other uses, you can do it the other way, eg "I don't know how to do it" In that sentence, "I" is the subject and "how to do it" is the object. Sorry I can't give a better explanation, some of the grammar experts can probably put it more concisely. Welcome to the forum! Thank you very much! It is perfectly clear now. So, I must understand that when a noun phrase is the object of a sentence, always goes with an -ing form? Thank you very much! It is perfectly clear now. So, I must understand that when a noun phrase is the object of a sentence, always goes with an -ing form? Actually, when the noun phrase is the subject of a sentence you will use 'ing'. Another example is "Going to the store, bores me" "Going to the store..." is the subject, saying ""Go/to go to the store is boring" would be wrong. But there will be other cases in which 'ing' will be used so don't take this as the final word on the subject. Now it's as clear as a bell. Thanks. ...saying ""Go/to go to the store is boring" would be wrong. Grammatically it is correct with the infinitive: To go to the store is boring. Although we wouldn't usually say it like that, it breaks no grammar rule, and in certain situations such a construction is used. That said, I agree that the gerund is far more common in this case. gengo, I can't think of the rule against it either, but it sounds wrong. "To go to the store would be boring" sounds better but "going" is best. Hello, everyone! According to English for Everyone with Kevin and Liza, this is the difference: 1. the secret to + a gerund = "The secret to understanding" (here you convey the idea of achieving a goal). If what comes after "to" can be perfectly replaced by a noun (and I would add "or by a pronoun"), then you can use a gerund as well: "The secret to success", "The secret to it", "The secret to understanding". 2. a secret to + base form = "A secret to help you understand" (here you convey the idea of "a secret that will help you understand"). The first structure, according to this channel, is way more common than the first. Have a good day. Generally speaking, because this is about generalizations rather than "rules," the gerund denotes processes/activities (physical or mental), while the infinitive denotes propositions. That's why we tend to say Smoking is prohibited, Going to the store is boring, Singing in the rain is fun, Thinking makes me tired because all of these suggest "activities," meaning that some effort is involved. That's not to say that To sing in the rain is fun is ungrammatical; it's just that the gerund is preferred. But this is not a "rule," so I'm sure there are cases where the gerund and infinitive are, for all practical purposes, interchangeable. The other thing to say is that, all things being equal, the infinitive as "subject" tends to undergo extraposition, which is a syntactic transformation that shifts the infinitive to the back of the sentence and introduces dummy it as new subject. So, instead of To redo the bathroom is a mistake, some intuitively say It's a mistake to redo the bathroom. Again, this is a general observation, not a "rule." I concur with all that 7Days said in #9. An example of a case where the infinitive is preferred (but the gerund is also grammatically possible) is the famous Shakespeare quotation: To be or not to be, that is the question. Después de las preposiciones en inglés usamos la forma con "-ing" del verbo (in sellling, for selling, about selling...) Ese to en "the secret to" es la mayoría de las veces una preposición, por lo que irá seguido de "ing". Piénsalo así: To puede funcionar como a) marca de infinitivo (to sell) b) preposición (to selling) Para saber cuándo has de usar cada uno, te basta con sustituir el verbo con un sustantivo. Si la preposición se mantiene con el sustantivo, es obvio que no era parte del verbo, no era una marca del infinitivo del verbo, era una preposición: The secret to it --> The secret to keeping your weight down (to = preposición) I want to go --> I want it (to = marca de infinitivo) This page is no longer up-to-date. For current instructions on how to set up and manage your Google OAuth Client, please see the Manage OAuth Clients article. To use OAuth 2.0 in your application, you need an OAuth 2.0 client ID, which your application uses when requesting an OAuth 2.0 access token. To create an OAuth 2.0 client ID in the console: Go to the API Console. From the projects list, select a project or create a new one. If the APIs & services page isn't already open, open the console left side menu and select APIs & services. On the left, click Credentials. Click New Credentials, then select OAuth client ID. Note: If you're unsure whether OAuth 2.0 is appropriate for your project, select Help me choose and follow the instructions to pick the right credentials. Select the appropriate application type for your project and enter any additional information required. Application types are described in more detail in the following sections. If this is your first time creating a client ID, you can also configure your consent screen by clicking Consent Screen. (The following procedure explains how to set up the Consent screen.) You won't be prompted to configure the consent screen after you do it the first time. Click Create client ID To delete a client ID, go to the Credentials page, check the box next to the ID, and then click Delete. User consent When you use OAuth 2.0 for authentication, your users are authenticated after they agree to terms that are presented to them on a user consent screen. Google verifies public applications that use OAuth 2.0 and meet one or more of the verification criteria. Learn more about public versus internal applications below. For more information about the verification process, see the OAuth Application Verification FAQ. If your application uses sensitive scopes without verification, the unverified app screen displays before the consent screen for users who are outside of your G Suite organization. To remove the unverified app screen, you can request OAuth developer verification by our team when you complete the Google API Console OAuth consent screen page. To set up your project's consent screen and request verification: Go to the Google API Console OAuth consent screen page. Add required information like a product name and support email address. Click Add Scope. On the dialog that appears, select the scopes your project uses. Sensitive scopes display a lock icon next to the API name. To select scopes for registration, you need to enable the API, like Drive or Gmail, from APIs & Services > API Library. You must select all scopes used by the project. When you're finished adding details to the OAuth consent screen, click Submit for verification. A Verification required window displays. Add scopes justification, a contact email address, and any other information that can help the team with verification, then click Submit. Note: The consent screen settings within the console are set at the project level, so the information that you specify on the Consent screen page applies across the entire project. When a project goes through verification, the current status displays under Verification status: Not Published: your OAuth consent screen is not published and verification is not required for use. Needs Verification: an OAuth developer verification is needed. To start the verification process, click Submit for Verification. For information about when verification is required, see the FAQ "When does my app have to be verified by Google?". Being verified: an OAuth developer verification is in progress. Published: your OAuth consent screen passed the verification and your project is verified. Failed verification: your OAuth consent screen didn't pass the verification. Your last approved consent screen is still in use. You'll receive more information at the contact email you provided. For more information about user authentication, see the OAuth 2.0 documentation. Public and internal applications A public application allows access to users outside of your organization (@your-organization.com). Access can be from consumer accounts, like @gmail.com, or other organizations, like @partner-organization.com. Public applications need to go through verification as detailed above. An internal application will only allow access to users from your organization (@your-organization.com). For more information about setting up organizations and organization access, see the GCP Organizations documentation. Authorized domains To protect you and your users, Google restricts your OAuth 2.0 application to using Authorized Domains. If you have verified the domain with Google, you can use any Top Private Domain as an Authorized Domain. After you add an Authorized Domain, you can use any of its subdomains or pages, and any other associated country codes. Add your Authorized Domains before you add your redirect or origin URIs, your homepage URL, your terms of service URL, or your privacy policy URL. This page is no longer up-to-date. For current instructions on how to set up and manage your Google OAuth Client, please see the Manage OAuth Clients article. Service accounts, web applications, and native applications For information about setting up service accounts, web applications, or device-native applications, see the following topics. Service accounts A service account is used in an application that calls APIs on behalf of an application that does not access user information. This type of application needs to prove its own identity, but it does not need a user to authorize requests. For example, if your project employs server-to-server interactions such as those between a web application and Google Cloud Storage, then you need a private key and other service account credentials. To generate these credentials, or to view the email address and public keys that you've already generated, do the following: Open the API Console Credentials page. If it's not already selected, select the project that you're creating credentials for. To set up a new service account, click New credentials and then select Service account key. Choose the service account to use for the key. Choose whether to download the service account's public/private key as a standard P12 file, or as a JSON file that can be loaded by a Google API client library. Your new public/private key pair is generated and downloaded to your machine; it serves as the only copy of this key. You are responsible for storing it securely. Your project needs the private key when requesting an OAuth 2.0 access token in server-to-server interactions. Google does not keep a copy of this private key, and this screen is the only place to obtain this particular private key. When you click Download private key, the PKCS #12-formatted private key is downloaded to your local machine. As the screen indicates, you must securely store this key yourself. The name of the downloaded private key is the key's thumbprint. When inspecting the key on your computer, or using the key in your application, you need to provide the password notasecret. Note that while the password for all Google-issued private keys is the same (notasecret), each key is cryptographically unique. You can generate multiple public-private key pairs for a single service account. This makes it easier to update credentials or roll them over without application downtime. However, you cannot delete a key pair if it is the only one created for that service account. Use the email address when granting the service account access to supported Google APIs. For more details, see the OAuth 2.0 Service Accounts documentation. Note: When you use a service account, you are subject to the Terms of Service for each product, both as an end user and as a developer. Web applications A web application is accessed by web browsers over a network. Applications that use client-side JavaScript to access Google APIs must specify authorized JavaScript origins. The origins identify the domains from which your application can send API requests. Applications that access Google APIs from a server (often using languages and frameworks like Node.js, Java, .NET, and Python) must specify authorized redirect URIs. The redirect URIs are the endpoints of your application server to which the OAuth 2.0 server can send responses. Native applications If your application is going to be installed on a device or computer (such as a system running Android, iOS, Universal Windows Platform, Chrome, or any desktop OS), you can use Google's OAuth 2.0 Mobile and desktop apps flow. If your application runs on devices with limited input capabilities, such as smart TVs, you can use Google's OAuth 2.0 TV and limited-input device flow. Android Note: Currently, obtaining OAuth 2.0 access tokens via AccountManager works for Android Ice Cream Sandwich (4.0) and newer versions. You need to specify your Android app's package name and SHA1 fingerprint. In the Package name field, enter your Android app's package name. In a terminal, run the keytool utility to get the SHA1 fingerprint for your digitally signed .apk file's public certificate. keytool -exportcert -alias androiddebugkey -keystore path-to-debug-or-production-keystore -list -v Note: For the debug.keystore, the password is android. For Android Studio, the debug keystore is typically located at ~/.android/debug.keystore. The Keytool prints the fingerprint to the shell. For example: $ keytool -list -v -keystore ~/.android/debug.keystore Enter keystore password: Type "android" if using debug.keystore Keystore type: JKS Keystore provider: SUN Your keystore contains 1 entry Alias name: androiddebugkey Creation date: Mar 13, 2020 Entry type: PrivateKeyEntry Certificate chain length: 1 Certificate[1]: Owner: C=US, O=Android, CN=Android Debug Issuer: C=US, O=Android, CN=Android Debug Serial number: 1 Valid from: Fri Mar 13 09:59:25 PDT 2020 until: Sun Mar 06 08:59:25 PST 2050 Certificate fingerprints: SHA1: D9:E9:59:FA:7A:46:72:4E:69:1F:96:18:8C:F9:AE:82:3A:5D:2F:03 SHA256: 92:59:1E:F4:C9:BC:72:43:1C:59:57:24:AD:78:CA:A2:DB:C7:C5:AC:B1:A3:E8:52:04:B2:00:37:53:04:0B:8E Signature algorithm name: SHA1withRSA Subject Public Key Algorithm: 2048-bit RSA key Version: 1 Copy the SHA1 fingerprint from the results that appear in your terminal. Important: When you prepare to release your app to your users, follow these steps again in a production project and create a new OAuth 2.0 client ID for your production app. For production apps, use your own private key to sign the production app's .apk file. For more information, see Signing your applications. Paste the SHA1 fingerprint into the form where requested. (Optional) Verify ownership of your Android application. You can verify ownership of your Android application to reduce the risk of app impersonation. Learn more about verifying ownership of your Android application. Click Create. iOS If your application accesses APIs directly from iOS, you will need the application's Bundle ID and, optionally, its Apple App Store ID and Team ID: The application's Bundle ID is the bundle identifier as listed in the app's .plist file. For example: com.example.myapp. The application's App Store ID is in the app's App Store URL, if the app was published in the Apple App Store. For example, in the app URL the App Store ID is 284815942. The application's Team ID is a 10-character string that Apple assigns to your team. For information about your Team ID, see Locating your Team ID in the Apple App Distribution Guide. After creating your iOS credentials and obtaining a Client ID, you use the Installed Application OAuth 2.0 flow to communicate with Google APIs. Universal Windows Platform (UWP) If your application runs on Universal Windows Platform, you will need your app's 12-character Store ID. You can find this value in the Partner Center, on the App identity page of the App management section. This value can also be found as the last part of your app's Microsoft Store URL. For example: Chrome apps Google Chrome apps and extensions are a special case of installed applications. Chrome exposes JavaScript APIs to allow your Chrome apps and extensions to perform various operations. Some of these APIs rely on knowing the identity of the user who is signed in to Chrome. If you're writing a Chrome app or extension that calls APIs that need to know the user's identity, and you want your app or extension to get user authorization for these requests using OAuth 2.0, then choose Chrome as the platform when you create your credentials. You will need to enter your Chrome app or extension's Application ID. For more information about these APIs, see the User Authentication documentation. Verify app ownership You can verify ownership of your Chrome application to reduce the risk of app impersonation. Learn more about verifying ownership of your Chrome application. TVs & limited-input devices The console does not require any additional information to create OAuth 2.0 credentials for applications running on limited-input devices, such as TVs. Desktop apps The console does not require any additional information to create OAuth 2.0 credentials for desktop applications. This page is no longer up-to-date. For current instructions on how to set up and manage your Google OAuth Client, please see the Manage OAuth Clients article. Rotating your client secrets Client secrets or credentials should be treated with extreme care as described in the OAuth 2.0 policies, because they allow anyone who has them to use your app's identity to gain access to user information. With the client secret rotation feature, you can add a new secret to your OAuth client configuration, migrate to the new secret while the old secret is still usable, and disable the old secret afterwards. This is useful when the client secret has been inadvertently disclosed or leaked. This also ensures good security practices by occasionally rotating your secrets without causing downtime of your app. In addition, Google started to issue more secure client secrets recommended by RFC 6749 in 2021. While apps that were created earlier are able to continue using the old secrets, we recommend that you migrate to the new secret with this rotation feature. To rotate your client secret, please follow the following steps: Step 1: Create a new client secret Go to the API Console Credentials page. If it's not already selected, select the project that you want to update. From the list of OAuth 2.0 Client IDs, click the client you want to generate a new client secret for. On the client details page, click Add Secret on the right side to add a new secret. A new secret will appear below the old secret. You can also differentiate them by the secret creation time. The new secret will be in "Enabled" state and ready to be used. Note 1: Both secrets can be used until you manually disable them. You must update your app to use the new secret and disable the old one as soon as possible after creating it to minimize security risks. Note 2: You can only have two client secrets at maximum. If the client already has two secrets, to create a new secret, you must first disable and delete an existing secret. Step 2: Configure your app to use the new secret Next, update your app to use the new secret. Remember to handle your client secrets securely as described in the OAuth 2.0 policies. You need to monitor your app and make sure the new secret has fully taken effect. In other words, make sure the old secret is not used anywhere in your app. Check the metrics and configurations used by your app to confirm that only the new client secret is used, for example: References in code or configurations. Your app or server logs. The rollout status of your updated app version or configuration. Any other metrics you may have. Step 3: Disable the old secret Having more than one enabled secrets for a client increases security risks. Once you confirm that your app has fully migrated to the new secret per the instructions in Step 2, you must disable the old secret. Go to the API Console Credentials page. From the list of OAuth 2.0 Client IDs, click the client you want to update. Find the old secret you want to disable. Generally it should be the one with the earlier creation time. Click Disable on the right side. The old secret will be invalid shortly. Note: A disabled client secret will be rejected in OAuth flows. You are expected to continuously monitor your app and see if it's working properly. In case you notice the app is failing because it is still using the old secret, you may click Enable to reinstate the secret on your client details page in API Console Credentials page. In this case, you should redo this step after completing the migration. Step 4: Delete the old secret Once you've confirmed that your app is working seamlessly with the new client secret, you are safe to delete the disabled old secret. To delete the secret, click the delete button next to it. Note that this cannot be undone. You can sync your Google Calendar events with other computer applications, like Outlook or Apple Calendar. Sync or view your calendar There are two ways to view Google Calendar in another calendar application. You can add your calendar to view in another application, and some applications will also let you edit events. Sync your Google Calendar (view & edit) With some calendar applications, you can sync your calendar using your Google Account. This means you can add and edit events from either Google Calendar or your other application. Open your other calendar application. Look for an option to add another account. This might be in "Settings" or "Preferences." Use "Sign in with Google" to start the process of giving access to your Google Account. Follow the steps to add your Google Account. Get your calendar (view only) If your calendar application doesn't have a full sync option, or if you want a read-only view of one calendar, you can sync your calendar to the application using a link to iCal. On your computer, open Google Calendar. In the top right, click Settings Settings. On the left panel, under "Settings for my calendars," click the name of the calendar you want to use. Click Integrate calendar. In the "Secret address in iCal format" section, copy the link. Paste the link as directed by your other calendar application. Important: Only you should know the Secret Address for your calendar. Do not share this address with other people. If you accidentally shared your calendar's Secret Address, click Reset to create a new Secret Address. I can't find the Secret Address If you're using a Google Account for work, school, or other organization, your admin might've changed the sharing settings for your calendar. If you can't find the Secret Address, ask your admin for help. Learn about Calendar addresses Secret Address The Secret Address lets you view your calendar in other applications, like Outlook or Apple Calendar. If you want to let someone else view your calendar, you can share your calendar. Important: You should not give your Secret Address to other people. If you accidentally shared your Secret Address, click Reset to invalidate the old address and create a new one. Public address Related resources Share your calendar with someone Add a Google calendar to your website Post to the help community Get answers from community members Your OAuth client is the credential which your application uses when making calls to Google OAuth 2.0 endpoint to receive an access token or ID token. After creating your OAuth client, you will receive a client ID and sometimes, a client secret. Think of your client ID like your app's unique username when it needs to request an access token or ID token from Google's OAuth 2.0 endpoint. This ID helps Google identify your app and ensure that only authorized applications can access user data. Client ID and Client Secret Similar to how you would use a username and password to log to online services, many applications use a client ID paired with a client secret. The client secret adds an extra layer of security, acting like your app's password. Applications are categorized as either public or private clients: Private Clients: These apps, like web server applications, can securely store the client secret because they run on servers you control. Public Clients: Native apps or JavaScript-based apps fall under this category. They cannot securely store secrets, as they reside on user devices and as such do not use client secrets. To create an OAuth 2.0 client ID in the console: Navigate to the Google Auth Platform Clients page. You will be prompted to create a project if you do not have one selected. You will be prompted to register your application to use Google Auth if you are yet to do so. This is required before creating a client. Click CREATE CLIENT Select the appropriate application type for your application and enter any additional information required. Application types are described in more detail in the following sections. Fill out the required information for the select client type and click the CREATE button to create the client. Note: Your application's client secret will only be shown after you create the client. Store this information in a secure place such as Google Cloud Secret Manager because it will not be visible or accessible again. Learn more. Application types Web Applications A web application is accessed by web browsers over a network. Authorized JavaScript origins Applications that use client-side JavaScript to access Google APIs must specify authorized JavaScript origins. The origins identify the domains from which your application can send API requests. Specified origins must adhere to the following rules : JavaScript origins must use the HTTPS scheme, not plain HTTP. Localhost URIs (including localhost IP address URIs) are exempt from this rule. Hosts cannot be raw IP addresses. Localhost IP addresses are exempted from this rule. If you use a port other than 80, you must specify it. For example: Host TLDs (Top Level Domains) must belong to the public suffix list. Host domains cannot be "googleusercontent.com". JavaScript origins cannot contain URL shortener domains (e.g. goo.gl) unless the app owns the domain. JavaScript origins cannot contain the userinfo subcomponent. JavaScript origins cannot contain the path component. JavaScript origins cannot contain the query component. JavaScript origins cannot contain the fragment component. JavaScript origins cannot contain certain characters including: Wildcard characters ('*') Non-printable ASCII characters Invalid percent encodings (any percent encoding that does not follow URL-encoding form of a percent sign followed by two hexadecimal digits) Null characters (an encoded NULL character, e.g., %00, %C0%80) If you send a request to a Google OAuth 2.0 endpoint from an unregistered JavaScript origin, you will receive an origin_mismatch error. Authorized redirect URIs Applications that access Google APIs from a server (often using languages and frameworks like Node.js, Java, .NET, and Python) must specify authorized redirect URIs. The redirect URIs are the endpoints of your application server to which the OAuth 2.0 server can send responses. Users are redirected to this path after they have authenticated with Google. Redirect URIs must adhere to the following rules : Redirect URIs must use the HTTPS scheme, not plain HTTP. Localhost URIs (including localhost IP address URIs) are exempt from this rule. Hosts cannot be raw IP addresses. Localhost IP addresses are exempted from this rule. Host TLDs (Top Level Domains) must belong to the public suffix list. Redirect URIs cannot contain URL shortener domains (e.g. goo.gl) unless the app owns the domain. Furthermore, if an app that owns a shortener domain chooses to redirect to that domain, that redirect URI must either contain "/google-callback/" in its path or end with "/google-callback". Redirect URIs cannot contain the userinfo subcomponent. Redirect URIs cannot contain a path traversal (also called directory backtracking), which is represented by an "/.." or "\.." or their URL encoding. Redirect URIs cannot contain open redirects. Redirect URIs cannot contain the fragment component. Redirect URIs cannot contain certain characters including: Wildcard characters ('*') Non-printable ASCII characters Invalid percent encodings (any percent encoding that does not follow URL-encoding form of a percent sign followed by two hexadecimal digits) Null characters (an encoded NULL character, e.g., %00, %C0%80) If the redirect_uri passed in the authorization request does not match an authorized redirect URI for the OAuth client ID, you will receive a redirect_uri_mismatch error. Note: It may take 5 minutes to a few hours for changes made to these settings to take effect Native Applications (Android, iOS, Desktop, UWP, Chrome Extensions, TV and Limited Input) If your application is going to be installed on a device or computer (such as a system running Android, iOS, Universal Windows Platform, Chrome, or any desktop OS), you can use Google's OAuth 2.0 Mobile and desktop apps flow. If your application runs on devices with limited input capabilities, such as smart TVs, you can use Google's OAuth 2.0 TV and limited-input device flow. Android Note: Currently, obtaining OAuth 2.0 access tokens via AccountManager works for Android Ice Cream Sandwich (4.0) and newer versions. You need to specify your Android app's package name and SHA1 fingerprint. In the Package name field, enter your Android app's package name. In a terminal, run the keytool utility to get the SHA1 fingerprint for your digitally signed .apk file's public certificate. keytool -list -v -keystore path-to-debug-or-production-keystore -alias androiddebugkey Note: For the debug.keystore, the password is android. For Android Studio, the debug keystore is typically located at ~/.android/debug.keystore. The Keytool prints the fingerprint to the shell. For example: $ keytool -list -v -keystore ~/.android/debug.keystore Enter keystore password: Type "android" if using debug.keystore Keystore type: JKS Keystore provider: SUN Your keystore contains 1 entry Alias name: androiddebugkey Creation date: Mar 13, 2020 Entry type: PrivateKeyEntry Certificate chain length: 1 Certificate[1]: Owner: C=US, O=Android, CN=Android Debug Issuer: C=US, O=Android, CN=Android Debug Serial number: 1 Valid from: Fri Mar 13 09:59:25 PDT 2020 until: Sun Mar 06 08:59:25 PST 2050 Certificate fingerprints: SHA1: D9:E9:59:FA:7A:46:72:4E:69:1F:96:18:8C:F9:AE:82:3A:5D:2F:03 SHA256: 92:59:1E:F4:C9:BC:72:43:1C:59:57:24:AD:78:CA:A2:DB:C7:C5:AC:B1:A3:E8:52:04:B2:00:37:53:04:0B:8E Signature algorithm name: SHA1withRSA Subject Public Key Algorithm: 2048-bit RSA key Version: 1 Copy the SHA1 fingerprint from the results that appear in your terminal. Important: When you prepare to release your app to your users, follow these steps again in a production project and create a new OAuth 2.0 client ID for your production app. For production apps, use your own private key to sign the production app's .apk file. For more information, see Signing your applications. Paste the SHA1 fingerprint into the form where requested. (Optional) Verify ownership of your Android application. You can verify ownership of your Android application to reduce the risk of app impersonation. Learn more about verifying ownership of your Android application. Click Create. iOS If your application accesses APIs directly from iOS, you will need the application's Bundle ID and, optionally, its Apple App Store ID and Team ID: The application's Bundle ID is the bundle identifier as listed in the app's .plist file. For example: com.example.myapp. The application's App Store ID is in the app's App Store URL, if the app was published in the Apple App Store. For example, in the app URL the App Store ID is 284815942. The application's Team ID is a 10-character string that Apple assigns to your team. For information about your Team ID, see Locating your Team ID in the Apple App Distribution Guide. After creating your iOS credentials and obtaining a Client ID, you use the Installed Application OAuth 2.0 flow to communicate with Google APIs. Enable App Check When you enable App Check, Apple's App Attest service is used to verify that OAuth 2.0 requests originating from your OAuth client are genuine and come from your app. This helps to reduce the risk of app impersonation. Learn more about enabling App Check for your iOS app. To enable App Check, turn on the Protect your OAuth client from abuse with Firebase App Check toggle button in the create/edit view of your iOS client. The following requirements must be met to successfully enable App Check for your iOS client: You must specify a team ID for your iOS client. You must not use a wildcard in your bundle ID since it can resolve to more than one app. This means that the bundle ID must not include the asterisk (*) symbol. Universal Windows Platform (UWP) If your application runs on Universal Windows Platform, you will need your app's 12-character Store ID. You can find this value in the Partner Center, on the App identity page of the App management section. This value can also be found as the last part of your app's Microsoft Store URL. For example: Chrome Extension Google Chrome apps and extensions are a special case of installed applications. Chrome exposes JavaScript APIs to allow your Chrome apps and extensions to perform various operations. Some of these APIs rely on knowing the identity of the user who is signed in to Chrome. If you're writing a Chrome app or extension that calls APIs that need to know the user's identity, and you want your app or extension to get user authorization for these requests using OAuth 2.0, then choose Chrome as the platform when you create your credentials. You will need to enter your Chrome app or extension's Application ID. The Item ID is the last part of your Chrome Extension's Chrome Web Store URL. For more information about these APIs, see the User Authentication documentation. Verify app ownership You can verify ownership of your Chrome application to reduce the risk of app impersonation. Learn more about verifying ownership of your Chrome application. TVs and Limited-input devices The console does not require any additional information to create OAuth 2.0 credentials for applications running on limited-input devices, such as TVs. Desktop apps The console does not require any additional information to create OAuth 2.0 credentials for desktop applications. Delete OAuth Clients To delete a client ID, go to the Clients page, check the box next to the ID you want to delete, and then click the DELETE button. Before deleting a Client ID, ensure to check the ID is not in use by monitoring your traffic in the overview page. You can restore deleted clients within 30 days of the deletion. To restore a recently deleted client, navigate to the Deleted credentials page to find a list of clients you recently deleted and click the RESTORE button for the client you want to restore. Any client deleted over 30 days ago cannot be restored and is permanently deleted. Note : Clients can also be automatically deleted if they become inactive. Learn more. Rotating your clients secrets Client secrets or credentials should be treated with extreme care as described in the OAuth 2.0 policies, because they allow anyone who has them to use your app's identity to gain access to user information. With the client secret rotation feature, you can add a new secret to your OAuth client configuration, migrate to the new secret while the old secret is still usable, and disable the old secret afterwards. This is useful when the client secret has been inadvertently disclosed or leaked. This also ensures good security practices by occasionally rotating your secrets without causing downtime of your app. In addition, Google started to issue more secure client secrets recommended by RFC 6749 in 2021. While apps that were created earlier are able to continue using the old secrets, we recommend that you migrate to the new secret with this rotation feature. To rotate your client secret, please follow the following steps: Step 1: Create a new client secret Go to the Google Auth Platform Clients page. If it's not already selected, select the project that you want to update. From the list of OAuth 2.0 Client IDs, click the client you want to generate a new client secret for. On the client details page, click Add Secret on the right side to add a new secret. A new secret will appear below the old secret. You can also differentiate them by the secret creation time. The new secret will be in "Enabled" state and ready to be used. Note 1: Both secrets can be used until you manually disable them. You must update your app to use the new secret and disable the old one as soon as possible after creating it to minimize security risks. Note 2: You can only have two client secrets at maximum. If the client already has two secrets, to create a new secret, you must first disable and delete an existing secret. Step 2: Configure your app to use the new secret Next, update your app to use the new secret. Remember to handle your client secrets securely as described in the OAuth 2.0 policies. You need to monitor your app and make sure the new secret has fully taken effect. In other words, make sure the old secret is not used anywhere in your app. Check the metrics and configurations used by your app to confirm that only the new client secret is used, for example: References in code or configurations. Your app or server logs. The rollout status of your updated app version or configuration. Any other metrics you may have. Step 3: Disable the old secret Having more than one enabled secrets for a client increases security risks. Once you confirm that your app has fully migrated to the new secret per the instructions in Step 2, you must disable the old secret. Go to the Google Auth Platform Clients page. From the list of OAuth 2.0 Client IDs, click the client you want to update. Find the old secret you want to disable. Generally it should be the one with the earlier creation time. Click Disable on the right side. The old secret will be invalid shortly. Note: A disabled client secret will be rejected in OAuth flows. You are expected to continuously monitor your app and see if it's working properly. In case you notice the app is failing because it is still using the old secret, you may click Enable to reinstate the secret on your client details page in the Google Auth Platform Clients page. In this case, you should redo this step after completing the migration. Step 4: Delete the old secret Once you've confirmed that your app is working seamlessly with the new client secret, you are safe to delete the disabled old secret. To delete the secret, click the delete button next to it. Note that this cannot be undone. Unused Client Deletion OAuth 2.0 clients that have been inactive for six months are automatically deleted. This mitigates risks associated with unused client credentials, such as potential app impersonation or unauthorized data access if credentials are compromised. An OAuth 2.0 client is considered unused if neither of the following actions have occurred within the past six months: The client has not been used for any credential or token request via the Google OAuth2.0 endpoint. The client's settings have not been modified programmatically or manually within the Google Cloud Console. Examples of modifications include changing the client name, rotating the client secret, or updating redirect URIs. You will receive an email notification 30 days before an inactive client is scheduled for deletion. To prevent the automatic deletion of a client you still require, ensure it is used for an authorization or authorization request before the 30 days elapses. A notification will also be sent after the client has been successfully deleted. Note : You should only take action to prevent deletion if you actively require the client. Keeping unused clients active unnecessarily increases security risk for your application. If you determine a client is no longer needed, delete it yourself via the Google Auth Platform Clients page. Do not wait for the automatic deletion process. Once an OAuth 2.0 client is deleted: It can no longer be used for Sign in with Google or for authorization for data access. Calls to Google APIs using existing access tokens or refresh tokens associated with the deleted client will fail. Attempts to use the deleted client ID in authorization requests will result in a deleted_client error. Deleted clients are typically recoverable at least 30 days following deletion. To restore a deleted client, navigate to the Deleted Credentials page. Only restore a client if you have a confirmed, ongoing need for it. To ensure that you receive these notifications and others related to your app, review your contact information settings. Client Secret Handling and Visibility Note: This feature is currently available for new clients created after June 2025 and will be extended to existing clients starting November 2025. In April 2025, we announced that client secrets for OAuth 2.0 clients are only visible and downloadable from the Google Cloud Console at the time of their creation. Client secrets add a critical layer of security to your OAuth 2.0 client ID, functioning similarly to a password for your application. Protecting these secrets is important for maintaining application security and privacy. To prevent accidental exposure and increase protection, client secrets are hashed. This means you will only be able to view and download the full client secret once, at the time of its creation. It is important that you download your OAuth 2.0 client secrets immediately upon creation and store them in a secure manner, for example in a secret manager such as Google Cloud Secret Manager. After the initial creation, the Google Cloud Console will only display the last four characters of the client secret. This truncated version is provided solely for identification purposes, allowing you to distinguish between your client secrets. If you lose your client secret, you can use the client secret rotation feature to get a new one. Best Practices for Client Secret Management Never add client secrets directly in your code or check them into version control systems such as Git or Subversion. Do not share client secrets in public forums, email, or other insecure communication channels. Store client secrets securely using a dedicated secret management service like Google Cloud Secret Manager or a similar secure storage solution. Rotate client secrets periodically and change immediately in the case of a leak. Manage client's brand configuration