Google Sign-In for Android

Try Sign-In for Android

Use our Android sample app to see how Sign-In works, or add Sign-In to your existing app.

Required: The latest versions of Android Studio and Google Play Services.

Get the project

If this is your first time using a Google services sample, check out the google-services repository.

$ git clone https://github.com/googlesamples/google-services.git

Open Android Studio.

Select **File > Open**, browse to where you cloned the google-services repository, and open google-services/android/signin.

Configure a Google API project

To use the sample, you need to provide some additional information to finish setting up your project. Click the button below, and specify the package name *com.google.samples.quickstart.signin* when prompted. You will also need to provide the SHA-1 hash of your signing certificate. See Authenticating Your Client for information.

CONFIGURE A PROJECT

The sample's *IdTokenActivity* and *ServerAuthCodeActivity* examples require you to specify an OAuth 2.0 web client ID. In a real app, this client ID would represent your app's backend server. A client ID for this purpose was created when you configured the project above.

Find this value by opening the Google API Console:

GOOGLE API CONSOLE

Your web server client ID is displayed next to Web client (Auto-created for Google Sign-in). Copy and paste the client ID into your project's strings.xml file:

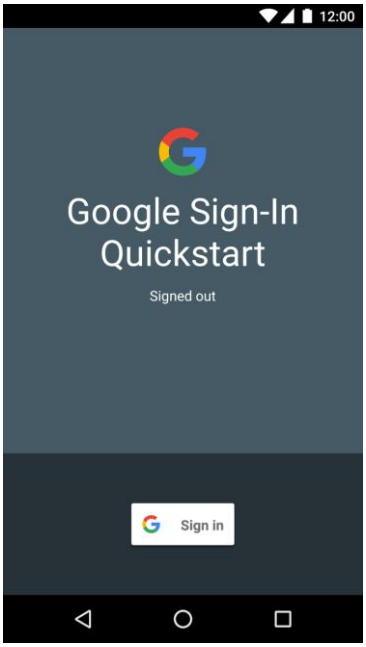
<string name="server_client_id">YOUR_SERVER_CLIENT_ID</string>

Última actualización: febrero 15, 2019.

Run the sample

Now you're ready to build the sample and run it from Android Studio.

Build the sample and click the run button and select a connected device or emulator with the latest version of Google Play services.



How it works

The application builds a GoogleSignInClient, specifying the sign-in options it needs. Then, when the sign-in button is clicked, the application starts the sign-in intent, which prompts the user to sign in with a Google account.

```
// Configure sign-in to request the user's ID, email address, and basic
// profile. ID and basic profile are included in DEFAULT_SIGN_IN.
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
      .requestEmail()
      .build();
// Build a GoogleSignInClient with the options specified by gso.
mGoogleSignInClient = GoogleSignIn.getClient(this, gso);
private void signIn() {
   Intent signInIntent = mGoogleSignInClient.getSignInIntent();
   startActivityForResult(signInIntent, RC_SIGN_IN);
}
```

Última actualización: febrero 15, 2019.

Next steps

If you want to see how you can implement Google Sign-In in your own app, take a look at our implementation guide.

ADD SIGN-IN TO YOUR APP

Start Integrating Google Sign-In into Your Android App

Before you can start integrating Google Sign-In in your own app, you must configure a Google API Console project and set up your Android Studio project. The steps on this page do just that. The next steps then describe how to integrate Google Sign-In into your app.

Prerequisites

Google Sign-In for Android has the following requirements:

- A compatible Android device that runs Android 4.1 or newer and includes the Google Play Store or an emulator with an AVD that runs the Google APIs platform based on Android 4.2.2 or newer and has Google Play services version 15.0.0 or newer.
- The latest version of the Android SDK, including the SDK Tools component. The SDK is available from the Android SDK Manager in Android Studio.
- A project configured to compile against Android 4.1 (Jelly Bean) or newer.
- The Google Play services SDK:
    - In Android Studio, select Tools > Android > SDK Manager.
    - Scroll to the bottom of the package list and select Extras > Google Repository. The package is downloaded to your computer and installed in your SDK environment at android-sdk-folder/extras/google/google_play_services.

This guide is written for users of Android Studio, which is the recommended development environment.

Add Google Play services

In your project's top-level build.gradle file, ensure that Google's Maven repository is included:

```
allprojects {
  repositories {
    google()

    // If you're using a version of Gradle lower than 4.1, you must instead use:
```


Última actualización: febrero 15, 2019.

```
        // maven {
        //     url 'https://maven.google.com'
        // }
    }
}
```

Then, in your app-level build.gradle file, declare Google Play services as a dependency:

```
apply plugin: 'com.android.application'
    ...

    dependencies {
        implementation 'com.google.android.gms:play-services-auth:17.0.0'
    }
```

Configure a Google API Console project

To configure a Google API Console project, click the button below, and specify your app's package name when prompted. You will also need to provide the SHA-1 hash of your signing certificate. See Authenticating Your Client for information.

CONFIGURE A PROJECT

Get your backend server's OAuth 2.0 client ID

If your app authenticates with a backend server or accesses Google APIs from your backend server, you must get the OAuth 2.0 client ID that was created for your server. To find the OAuth 2.0 client ID:

1. Open the Credentials page in the API Console.
2. The Web application type client ID is your backend server's OAuth 2.0 client ID.

Note: If you haven't recently created a new Android client, you might not have a Web application type client ID. You can create one by opening the Credentials page and clicking New credentials > OAuth client ID.

Pass this client ID to the requestIdToken or requestServerAuthCode method when you create the GoogleSignInOptions object.

Next steps

Now that you have configured a Google API Console project and set up your Android Studio project, you can integrate Google Sign-In into your app.

Última actualización: febrero 15, 2019.

Integrating Google Sign-In into Your Android App

To integrate Google Sign-In into your Android app, configure Google Sign-In and add a button to your app's layout that starts the sign-in flow.

Before you begin

Configure a Google API Console project and set up your Android Studio project.

Configure Google Sign-in and the GoogleSignInClient object

1. In your sign-in activity's onCreate method, configure Google Sign-In to request the user data required by your app. For example, to configure Google Sign-In to request users' ID and basic profile information, create a GoogleSignInOptions object with the DEFAULT_SIGN_IN parameter. To request users' email addresses as well, create the GoogleSignInOptions object with the requestEmail option.

```
// Configure sign-in to request the user's ID, email address, and basic
// profile. ID and basic profile are included in DEFAULT_SIGN_IN.
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
    .requestEmail()
    .build();
```

If you need to request additional scopes to access Google APIs, specify them with requestScopes. For the best user experience, on sign-in, only request the scopes that are required for your app to minimally function. Request any additional scopes only when you need them, so that your users see the consent screen in the context of an action they performed. See Requesting Additional Scopes.

2. Then, also in your sign-in activity's onCreate method, create a GoogleSignInClient object with the options you specified.

```
// Build a GoogleSignInClient with the options specified by gso.
mGoogleSignInClient = GoogleSignIn.getClient(this, gso);
```

Check for an existing signed-in user

In your activity's onStart method, check if a user has already signed in to your app with Google.

Última actualización: febrero 15, 2019.

```
// Check for existing Google Sign In account, if the user is already signed in
// the GoogleSignInAccount will be non-null.
GoogleSignInAccount account = GoogleSignIn.getLastSignedInAccount(this);
updateUI(account);
```

If GoogleSignIn.getLastSignedInAccount returns a GoogleSignInAccount object (rather than null), the user has already signed in to your app with Google. Update your UI accordingly—that is, hide the sign-in button, launch your main activity, or whatever is appropriate for your app.

If GoogleSignIn.getLastSignedInAccount returns null, the user has not yet signed in to your app with Google. Update your UI to display the Google Sign-in button.

Note: If you need to detect changes to a user's auth state that happen outside your app, such as access token or ID token revocation, or to perform cross-device sign-in, you might also call GoogleSignInClient.silentSignIn when your app starts.

Add the Google Sign-in button to your app

The standard Google sign-in button Add the SignInButton in your application's layout:

```
<com.google.android.gms.common.SignInButton
 android:id="@+id/sign_in_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

Optional: If you are using the default sign-in button graphic instead of providing your own sign-in button assets, you can customize the button's size with the setSize method.

```
// Set the dimensions of the sign-in button.
SignInButton signInButton = findViewById(R.id.sign_in_button);
signInButton.setSize(SignInButton.SIZE_STANDARD);
```

In the Android activity (for example, in the onCreate method), register your button's OnClickListener to sign in the user when clicked:

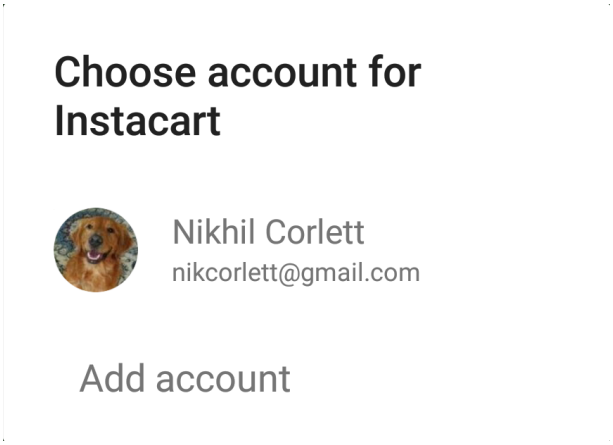
```
findViewById(R.id.sign_in_button).setOnClickListener(this);
```

Start the sign-in flow

1. In the activity's onClick method, handle sign-in button taps by creating a sign-in intent with the getSignInIntent method, and starting the intent with startActivityForResult.

Última actualización: febrero 15, 2019.

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.sign_in_button:
            signIn();
            break;
        // ...
    }
}
```




```
private void signIn() {
    Intent signInIntent = mGoogleSignInClient.getSignInIntent();
    startActivityForResult(signInIntent, RC_SIGN_IN);
}
```

Starting the intent prompts the user to select a Google account to sign in with. If you requested scopes beyond profile, email, and openid, the user is also prompted to grant access to the requested resources.

2. After the user signs in, you can get a GoogleSignInAccount object for the user in the activity's onActivityResult method.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    // Result returned from launching the Intent from GoogleSignInClient.getSignInIntent(...);
    if (requestCode == RC_SIGN_IN) {
        // The Task returned from this call is always completed, no need to attach
        // a listener.
```

Última actualización: febrero 15, 2019.

```
        Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
        handleSignInResult(task);
    }
}
```

The GoogleSignInAccount object contains information about the signed-in user, such as the user's name.

```
private void handleSignInResult(Task<GoogleSignInAccount> completedTask) {
    try {
        GoogleSignInAccount account = completedTask.getResult(ApiException.class);

        // Signed in successfully, show authenticated UI.
        updateUI(account);
    } catch (ApiException e) {
        // The ApiException status code indicates the detailed failure reason.
        // Please refer to the GoogleSignInStatusCodes class reference for more information.
        Log.w(TAG, "signInResult:failed code=" + e.getStatusCode());
        updateUI(null);
    }
}
```

You can also get the user's email address with getEmail, the user's Google ID (for client-side use) with getId, and an ID token for the user with getIdToken. If you need to pass the currently signed-in user to a backend server, send the ID token to your backend server and validate the token on the server.

Getting Profile Information

After you have signed in a user with Google, if you configured Google Sign-In, with the DEFAULT_SIGN_IN parameter or the requestProfile method, you can access the user's basic profile information. If you configured Google Sign-In with the requestEmail method, you can also get their email address.

Important: Do not use a user's email address or user ID to communicate the currently signed-in user to your app's backend server. Instead, send the user's ID token to your backend server and validate the token on the server, or use the server auth code flow.

Before you begin

Configure your Android Studio project
Integrate Google Sign-In into your app

Última actualización: febrero 15, 2019.

Retrieve profile information for a signed-in user

Use the GoogleSignIn.getLastSignedInAccount method to request profile information for the currently signed in user.

```
GoogleSignInAccount acct = GoogleSignIn.getLastSignedInAccount(getActivity());
if (acct != null) {
  String personName = acct.getDisplayName();
  String personGivenName = acct.getGivenName();
  String personFamilyName = acct.getFamilyName();
  String personEmail = acct.getEmail();
  String personId = acct.getId();
  Uri personPhoto = acct.getPhotoUrl();
}
```

Note: A Google account's email address can change, so don't use it to identify a user. Instead, use the account's ID, which you can get on the client with GoogleSignInAccount.getId, and on the backend from the sub claim of the ID token.

For additional profile data that might be available, see GoogleSignInAccount. Note that any of the profile fields can be null, depending on which scopes you requested and what information the user's profile includes.

Signing Out Users and Disconnecting Accounts

You can enable your users to sign out of your app, and to disconnect their accounts from your app entirely.

Sign out users

To add a sign out button to your app, first create a button in your app to act as your sign out button. Then, attach an onClickListener to the button and configure the onClick method to call signOut.

```
@Override
public void onClick(View v) {
   switch (v.getId()) {
      // ...
      case R.id.button_sign_out:
         signOut();
         break;
```


Última actualización: febrero 15, 2019.

```
      // ...
    }
}
private void signOut() {
    mGoogleSignInClient.signOut()
            .addOnCompleteListener(this, new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    // ...
                }
            });
}
```

This code clears which account is connected to the app. To sign in again, the user must choose their account again.

Disconnect accounts

It is highly recommended that you provide users that signed in with Google the ability to disconnect their Google account from your app. If the user deletes their account, you must delete the information that your app obtained from the Google APIs.

The following code shows a simple example of calling the revokeAccess method:

```
private void revokeAccess() {
    mGoogleSignInClient.revokeAccess()
            .addOnCompleteListener(this, new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    // ...
                }
            });
}
```

In the completion listener, you can respond to the event and trigger any appropriate logic in your app or your back-end code.

Authenticate with a backend server

If you use Google Sign-In with an app or site that communicates with a backend server, you might need to identify the currently signed-in user on the server. To do so securely, after a user successfully signs in, send the user's ID token to your server using HTTPS. Then, on the server,

Última actualización: febrero 15, 2019.

verify the integrity of the ID token and use the user information contained in the token to establish a session or create a new account.

Warning: Do not accept plain user IDs, such as those you can get with the GoogleSignInAccount.getId() method, on your backend server. A modified client application can send arbitrary user IDs to your server to impersonate users, so you must instead use verifiable ID tokens to securely get the user IDs of signed-in users on the server side.

Send the ID token to your server

First, when the user signs in, get their ID token:

1. When you configure Google Sign-in, call the requestIdToken method and pass it your server's web client ID.

```
// Request only the user's ID token, which can be used to identify the
// user securely to your backend. This will contain the user's basic
// profile (name, profile picture URL, etc) so you should not need to
// make an additional call to personalize your application.
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
      .requestIdToken(getString(R.string.server_client_id))
      .requestEmail()
      .build();
```

2. When your app starts, check if the user has already signed in to your app using Google, on this device or another device, by calling silentSignIn:

```
GoogleSignIn.silentSignIn()
   .addOnCompleteListener(
      this,
      new OnCompleteListener<GoogleSignInAccount>() {
        @Override
        public void onComplete(@NonNull Task<GoogleSignInAccount> task) {
          handleSignInResult(task);
        }
      });
```

3. If the user can't sign in silently, present your normal signed-out experience, giving the user the option to sign in. When the user does sign in, get the user's GoogleSignInAccount in the activity result of the sign-in intent:

```
// This task is always completed immediately, there is no need to attach an
```


Última actualización: febrero 15, 2019.

```
// asynchronous listener.
Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
handleSignInResult(task);
```

4. After the user signs in silently or explicitly, get the ID token from the GoogleSignInAccount object:

```
private void handleSignInResult(@NonNull Task<GoogleSignInAccount> completedTask) {
   try {
      GoogleSignInAccount account = completedTask.getResult(ApiException.class);
      String idToken = account.getIdToken();

      // TODO(developer): send ID Token to server and validate

      updateUI(account);
   } catch (ApiException e) {
      Log.w(TAG, "handleSignInResult:error", e);
      updateUI(null);
   }
}
```

If you didn't configure GoogleSignInOptions with requestIdToken, getIdToken will return null.

Then, send the ID token to your server with an HTTPS POST request:

```
HttpClient httpClient = new DefaultHttpClient();
HttpPost httpPost = new HttpPost("https://yourbackend.example.com/tokensignin");

try {
  List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(1);
  nameValuePairs.add(new BasicNameValuePair("idToken", idToken));
  httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

  HttpResponse response = httpClient.execute(httpPost);
  int statusCode = response.getStatusLine().getStatusCode();
  final String responseBody = EntityUtils.toString(response.getEntity());
  Log.i(TAG, "Signed in as: " + responseBody);
} catch (ClientProtocolException e) {
  Log.e(TAG, "Error sending ID token to backend.", e);
} catch (IOException e) {
  Log.e(TAG, "Error sending ID token to backend.", e);
}
```


Última actualización: febrero 15, 2019.

Verify the integrity of the ID token

After you receive the ID token by HTTPS POST, you must verify the integrity of the token. To verify that the token is valid, ensure that the following criteria are satisfied:

- The ID token is properly signed by Google. Use Google's public keys (available in JWK or PEM format) to verify the token's signature. These keys are regularly rotated; examine the Cache-Control header in the response to determine when you should retrieve them again.
- The value of aud in the ID token is equal to one of your app's client IDs. This check is necessary to prevent ID tokens issued to a malicious app being used to access data about the same user on your app's backend server.
- The value of iss in the ID token is equal to accounts.google.com or https://accounts.google.com.
- The expiry time (exp) of the ID token has not passed.
- If you want to restrict access to only members of your G Suite domain, verify that the ID token has an hd claim that matches your G Suite domain name.

Rather than writing your own code to perform these verification steps, we strongly recommend using a Google API client library for your platform, or a general-purpose JWT library. For development and debugging, you can call our tokeninfo validation endpoint.

Using a Google API Client Library

Using one of the Google API Client Libraries (e.g. Java, Node.js, PHP, Python) is the recommended way to validate Google ID tokens in a production environment.

To validate an ID token in Java, use the GoogleIdTokenVerifier object. For example:

```
import com.google.api.client.googleapis.auth.oauth2.GoogleIdToken;
import com.google.api.client.googleapis.auth.oauth2.GoogleIdToken.Payload;
import com.google.api.client.googleapis.auth.oauth2.GoogleIdTokenVerifier;

...

GoogleIdTokenVerifier verifier = new GoogleIdTokenVerifier.Builder(transport, jsonFactory)
    // Specify the CLIENT_ID of the app that accesses the backend:
    .setAudience(Collections.singletonList(CLIENT_ID))
    // Or, if multiple clients access the backend:
    //.setAudience(Arrays.asList(CLIENT_ID_1, CLIENT_ID_2, CLIENT_ID_3))
    .build();

// (Receive idTokenString by HTTPS POST)
```


Última actualización: febrero 15, 2019.

```
GoogleIdToken idToken = verifier.verify(idTokenString);
if (idToken != null) {
  Payload payload = idToken.getPayload();

  // Print user identifier
  String userId = payload.getSubject();
  System.out.println("User ID: " + userId);

  // Get profile information from payload
  String email = payload.getEmail();
  boolean emailVerified = Boolean.valueOf(payload.getEmailVerified());
  String name = (String) payload.get("name");
  String pictureUrl = (String) payload.get("picture");
  String locale = (String) payload.get("locale");
  String familyName = (String) payload.get("family_name");
  String givenName = (String) payload.get("given_name");

  // Use or store profile information
  // ...

} else {
  System.out.println("Invalid ID token.");
}
```

The GoogleIdTokenVerifier.verify() method verifies the JWT signature, the aud claim, the iss claim, and the exp claim.

If you want to restrict access to only members of your G Suite domain, also verify the hd claim by checking the domain name returned by the Payload.getHostedDomain() method.

Calling the tokeninfo endpoint

An easy way to validate an ID token for debugging is to use the tokeninfo endpoint. Calling this endpoint involves an additional network request that does most of the validation for you, but introduces some latency and the potential for network errors.

To validate an ID token using the tokeninfo endpoint, make an HTTPS POST or GET request to the endpoint, and pass your ID token in the id_token parameter. For example, to validate the token "XYZ123", make the following GET request:

https://oauth2.googleapis.com/tokeninfo?id_token=XYZ123

Última actualización: febrero 15, 2019.

If the token is properly signed and the iss and exp claims have the expected values, you will get a HTTP 200 response, where the body contains the JSON-formatted ID token claims. Here's an example response:

```
{
 // These six fields are included in all Google ID Tokens.
 "iss": "https://accounts.google.com",
 "sub": "110169484474386276334",
 "azp": "1008719970978-hb24n2dstb40o45d4feuo2ukqmcc6381.apps.googleusercontent.com",
 "aud": "1008719970978-hb24n2dstb40o45d4feuo2ukqmcc6381.apps.googleusercontent.com",
 "iat": "1433978353",
 "exp": "1433981953",

 // These seven fields are only included when the user has granted the "profile" and
 // "email" OAuth scopes to the application.
 "email": "testuser@gmail.com",
 "email_verified": "true",
 "name" : "Test User",
 "picture": "https://lh4.googleusercontent.com/-
kYgzyAWpZzJ/ABCDEFGHI/AAAJKLMNOP/tIXL9Ir44LE/s99-c/photo.jpg",
 "given_name": "Test",
 "family_name": "User",
 "locale": "en"
}
```

***Warning**: Once you get these claims, you still need to check that the aud claim contains one of your app's client IDs. If it does, then the token is both valid and intended for your client, and you can safely retrieve and use the user's unique Google ID from the sub claim.*

If you are a G Suite customer, you might also be interested in the hd claim, which indicates the hosted domain of the user. This can be used to restrict access to a resource to only members of certain domains. The absence of this claim indicates that the user does not belong to a G Suite hosted domain.

Create an account or session

After you have verified the token, check if the user is already in your user database. If so, establish an authenticated session for the user. If the user isn't yet in your user database, create a new user record from the information in the ID token payload, and establish a session for the user. You can prompt the user for any additional profile information you require when you detect a newly created user in your app.

Última actualización: febrero 15, 2019.

Securing your users' accounts with Cross Account Protection

When you rely on Google to sign in a user, you'll automatically benefit from all of the security features and infrastructure Google has built to safeguard the user's data. However, in the unlikely event that the user's Google Account gets compromised or there is some other significant security event, your app can also be vulnerable to attack. To better protect your accounts from any major security events, use Cross Account Protection to receive security alerts from Google. When you receive these events, you gain visibility into important changes to the security of the user's Google account and you can then take action on your service to secure your accounts.

Enabling Server-Side Access

With the earlier Add Sign-In procedure, your app authenticates the user on the client side only; in that case, you can access the Google APIs only while the user is actively using your app. If you want your servers to be able to make Google API calls on behalf of users—possibly while they are offline—your server requires an access token.

Before you begin

- Configure a project
- Add a Google Sign-In button to your app
- Create an OAuth 2.0 web application client ID for your backend server. This client ID is different from your app's client ID. You can find or create a client ID for your server in the Google API Console.

Enable server-side API access for your app

1. When you configure Google Sign-In, build the GoogleSignInOptions object with the requestServerAuthCode method and specify the scopes that your app's backend needs to access with the requestScopes method.

Pass your server's client ID to the requestServerAuthCode method.

```
// Configure sign-in to request offline access to the user's ID, basic
// profile, and Google Drive. The first time you request a code you will
// be able to exchange it for an access token and refresh token, which
// you should store. In subsequent calls, the code will only result in
// an access token. By asking for profile access (through
// DEFAULT_SIGN_IN) you will also get an ID Token as a result of the
// code exchange.
String serverClientId = getString(R.string.server_client_id);
```

Última actualización: febrero 15, 2019.

```
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
      .requestScopes(new Scope(Scopes.DRIVE_APPFOLDER))
      .requestServerAuthCode(serverClientId)
      .requestEmail()
      .build();
```

2. After the user successfully signs in, get an auth code for the user with getServerAuthCode:

```
Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
try {
   GoogleSignInAccount account = task.getResult(ApiException.class);
   String authCode = account.getServerAuthCode();

   // Show signed-un UI
   updateUI(account);

   // TODO(developer): send code to server and exchange for access/refresh/ID tokens
} catch (ApiException e) {
   Log.w(TAG, "Sign-in failed", e);
   updateUI(null);
}
```

3. Send the auth code to your app's backend using HTTPS POST:

```
HttpPost httpPost = new HttpPost("https://yourbackend.example.com/authcode");

try {
   List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(1);
   nameValuePairs.add(new BasicNameValuePair("authCode", authCode));
   httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

   HttpResponse response = httpClient.execute(httpPost);
   int statusCode = response.getStatusLine().getStatusCode();
   final String responseBody = EntityUtils.toString(response.getEntity());
} catch (ClientProtocolException e) {
   Log.e(TAG, "Error sending auth code to backend.", e);
} catch (IOException e) {
   Log.e(TAG, "Error sending auth code to backend.", e);
}
```


Última actualización: febrero 15, 2019.

4. On your app's backend server, exchange the auth code for access and refresh tokens. Use the access token to call Google APIs on behalf of the user and, optionally, store the refresh token to acquire a new access token when the access token expires.

If you requested profile access, you also get an ID token that contains basic profile information for the user.

For example:

```
// (Receive authCode via HTTPS POST)


if (request.getHeader('X-Requested-With') == null) {
  // Without the `X-Requested-With` header, this request could be forged. Aborts.
}

// Set path to the Web application client_secret_*.json file you downloaded from the
// Google API Console: https://console.developers.google.com/apis/credentials
// You can also find your Web application client ID and client secret from the
// console and specify them directly when you create the GoogleAuthorizationCodeTokenRequest
// object.
String CLIENT_SECRET_FILE = "/path/to/client_secret.json";

// Exchange auth code for access token
GoogleClientSecrets clientSecrets =
   GoogleClientSecrets.load(
      JacksonFactory.getDefaultInstance(), new FileReader(CLIENT_SECRET_FILE));
GoogleTokenResponse tokenResponse =
        new GoogleAuthorizationCodeTokenRequest(
           new NetHttpTransport(),
           JacksonFactory.getDefaultInstance(),
           "https://oauth2.googleapis.com/token",
           clientSecrets.getDetails().getClientId(),
           clientSecrets.getDetails().getClientSecret(),
           authCode,
           REDIRECT_URI)  // Specify the same redirect URI that you use with your web
                    // app. If you don't have a web version of your app, you can
                    // specify an empty string.
           .execute();

String accessToken = tokenResponse.getAccessToken();

// Use access token to call API
```

Última actualización: febrero 15, 2019.

```
GoogleCredential credential = new GoogleCredential().setAccessToken(accessToken);
Drive drive =
    new Drive.Builder(new NetHttpTransport(), JacksonFactory.getDefaultInstance(), credential)
        .setApplicationName("Auth Code Exchange Demo")
        .build();
File file = drive.files().get("appfolder").execute();

// Get profile info from ID token
GoogleIdToken idToken = tokenResponse.parseIdToken();
GoogleIdToken.Payload payload = idToken.getPayload();
String userId = payload.getSubject();  // Use this value as a key to identify a user.
String email = payload.getEmail();
boolean emailVerified = Boolean.valueOf(payload.getEmailVerified());
String name = (String) payload.get("name");
String pictureUrl = (String) payload.get("picture");
String locale = (String) payload.get("locale");
String familyName = (String) payload.get("family_name");
String givenName = (String) payload.get("given_name");
```

Requesting Additional Scopes

For the best user experience, you should request as few scopes as possible when initially signing in users. If your app's core functionality isn't tied to a Google service, the GoogleSignInOptions.DEFAULT_SIGN_IN configuration is often all you need at sign-in.

If your app has features that can make use of Google API data, but are not required as part of your app's core functionality, you should design your app to be able to gracefully handle cases when API data isn't accessible. For example, you might hide a list of recently saved files when the user hasn't granted Drive access.

You should request additional scopes that you need to access Google APIs only when the user performs an action that requires access to a particular API. For example, you might request permission to access the user's Drive only when the user taps a "Save to Drive" button for the first time.

By using this technique, you can avoid overwhelming new users, or confusing users as to why they are being asked for certain permissions.

Requesting permissions required by user actions

Whenever a user performs an action that requires a scope that isn't requested at sign-in, call GoogleSignIn.hasPermissions to check if the user has already granted the required permissions. If

not, call GoogleSignIn.requestPermissions to launch an activity that requests the additional required scopes from the user.

For example, if a user performs an action that requires access to their Drive app storage, do the following:

```
if (!GoogleSignIn.hasPermissions(
      GoogleSignIn.getLastSignedInAccount(getActivity()),
      Drive.SCOPE_APPFOLDER)) {
   GoogleSignIn.requestPermissions(
         MyExampleActivity.this,
         RC_REQUEST_PERMISSION_SUCCESS_CONTINUE_FILE_CREATION,
         GoogleSignIn.getLastSignedInAccount(getActivity()),
         Drive.SCOPE_APPFOLDER);
} else {
   saveToDriveAppFolder();
}
```

In your activity's onActivityResult callback, you can check if the required permissions were successfully acquired, and if so, carry out the user action.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   if (resultCode == Activity.RESULT_OK) {
      if (RC_REQUEST_PERMISSION_SUCCESS_CONTINUE_FILE_CREATION == requestCode) {
         saveToDriveAppFolder();
      }
   }
}
```

You can also pass a GoogleSignInOptionsExtension to hasPermissions and requestPermissions to check for and acquire a set of permissions more conveniently.

Última actualización: febrero 15, 2019.