API Details

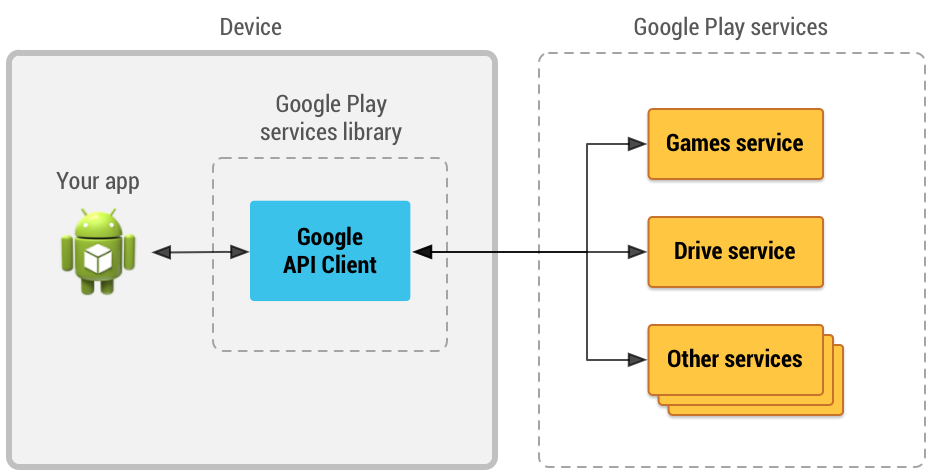
**Google sign in:**

We use sign in to authenticate the user, this is our primary level of security and identification.

This is needed because we need a way to create a secure login and we need to identify the users who are using the app. The app will be running in background, and we need to authenticate the user and identify him to the server using this .

Google sign in service is a part of google play services library. Accessing this service inside the app requires “Google Api Client”. To access any of googles sign in functions an object of the google api client should be created. This google api client provides a common entry point to all google play services and manages the network connection between the users device and each google service.

The following figure shows how the google api client provides an interface for connecting and making calls to any of the available google play services such as google play games and google drive.



Before any operation is executed, the GoogleApiClient must be connected. The simplest way to manage the connection is to use enableAutoManage(FragmentActivity, GoogleApiClient.OnConnectionFailedListener).

**GoogleApiClient Nested Class Summary:**

|  |  |  |
| --- | --- | --- |
| class | [GoogleApiClient.Builder](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.Builder.html) | Builder to configure a [GoogleApiClient](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.html). |
| interface | [GoogleApiClient.ConnectionCallbacks](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.ConnectionCallbacks.html) | Provides callbacks that are called when the client is connected or disconnected from the service. |
| interface | [GoogleApiClient.OnConnectionFailedListener](https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient.OnConnectionFailedListener.html) | Provides callbacks for scenarios that result in a failed attempt to connect the client to the service. |

**Firebase login: (alternative to google sign in)**

Alternatively we can use googles firebase authentication. We can let users authenticate with Firebase using their Google Accounts by integrating Google Sign-In into your app.

**Firebase can be used to login with facebook/google credentials.**

To add Firebase to your app you'll need a Firebase project and a Firebase configuration file for your app.

1. Create a Firebase project in the Firebase console, if you don't already have one. If you already have an existing Google project associated with your mobile app, click Import Google Project. Otherwise, click Create New Project.
2. Click Add Firebase to your Android app and follow the setup steps. If you're importing an existing Google project, this may happen automatically and you can just download the config file.
3. When prompted, enter your app's package name. It's important to enter the package name your app is using; this can only be set when you add an app to your Firebase project.
4. At the end, you'll download a google-services.json file. You can download this file again at any time.
5. If you haven't done so already, copy this into your project's module folder, typically app/.

Configuring firebase:

1. [Add Firebase to your Android project](https://firebase.google.com/docs/android/setup).
2. Add the dependencies for Firebase Authentication and Google Sign-In to your app-level build.gradle file:
3. compile 'com.google.firebase:firebase-auth:9.4.0'
4. compile 'com.google.android.gms:play-services-auth:9.4.0'
5. If you haven't yet connected your app to your Firebase project, do so from the [Firebase console](https://firebase.google.com/console/).
6. Enable Google Sign-In in the Firebase console:
   1. In the [Firebase console](https://firebase.google.com/console/), open the **Auth** section.
   2. On the **Sign in method** tab, enable the **Google** sign-in method and click **Save**.

## Authenticate with Firebase:

1.Integrate Google Sign-In into your app by following the steps on the [Integrating Google Sign-In into Your Android App](https://developers.google.com/identity/sign-in/android/sign-in) page. When you configure the GoogleSignInOptions object, call requestIdToken.

You must pass your [server's client ID](https://developers.google.com/identity/sign-in/android/start-integrating#get_your_backend_servers_oauth_20_client_id) to the requestIdToken method. To find the OAuth 2.0 client ID:

1. Open the [Credentials page](https://console.developers.google.com/apis/credentials) in the API Console.
2. The **Web application** type client ID is your backend server's OAuth 2.0 client ID.

2.In your sign-in activity's onCreate method, get the shared instance of the FirebaseAuth object.

3.Set up an AuthStateListener that responds to changes in the user's sign-in state.

4. After a user successfully signs in, get an ID token from the GoogleSignInAccount object, exchange it for a Firebase credential, and authenticate with Firebase using the Firebase credential.

To sign out a user, call [signOut](https://firebase.google.com/docs/reference/android/com/google/firebase/auth/FirebaseAuth.html#signOut%28%29):

FirebaseAuth.getInstance().signOut();

**Google Location Based Services:**

Location based services api are required to fetch the position of the user. We need the position to find the closest user to the person requesting blood. The app will be running as a background activity and it can be called upon to provide the location of the user.

## Interfaces:

|  |  |
| --- | --- |
| [GpsStatus.Listener](https://developer.android.com/reference/android/location/GpsStatus.Listener.html) | This interface was deprecated in API level 24. use [*GnssStatus.Callback*](https://developer.android.com/reference/android/location/GnssStatus.Callback.html) instead. |
| [GpsStatus.NmeaListener](https://developer.android.com/reference/android/location/GpsStatus.NmeaListener.html) | This interface was deprecated in API level 24. use [*OnNmeaMessageListener*](https://developer.android.com/reference/android/location/OnNmeaMessageListener.html) instead. |
| [LocationListener](https://developer.android.com/reference/android/location/LocationListener.html) | Used for receiving notifications from the LocationManager when the location has changed. |
| [OnNmeaMessageListener](https://developer.android.com/reference/android/location/OnNmeaMessageListener.html) | Used for receiving NMEA sentences from the GNSS. |

|  |  |
| --- | --- |
| [Location](https://developer.android.com/reference/android/location/Location.html) | A data class representing a geographic location. |
| [LocationManager](https://developer.android.com/reference/android/location/LocationManager.html) | This class provides access to the system location services. |
| [LocationProvider](https://developer.android.com/reference/android/location/LocationProvider.html) | An abstract superclass for location providers. |

The *RequestLocationUpdates* methods are used to request regular updates of location changes using *LocationListner*. LocationListner also contains hooks for changes in a provider’s status and availability.

The requestLocationUpdates method accepts either a specific location provider name or a set of criteria to determine the provider to use. To optimize efficiency and minimize cost and power use, you can also specify the minimum time and the minimum distance between location change updates.