Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Android Google Services Part 1

Google Sign-in

Google Services Overview *

- Overview of Google Play Services and Setup
- Detailed look at
  - Google Sign-in and Authentication (Part 1)
  - Location & Geocoding (Part 2)
  - Google Maps (Part 3)

Google Services Overview

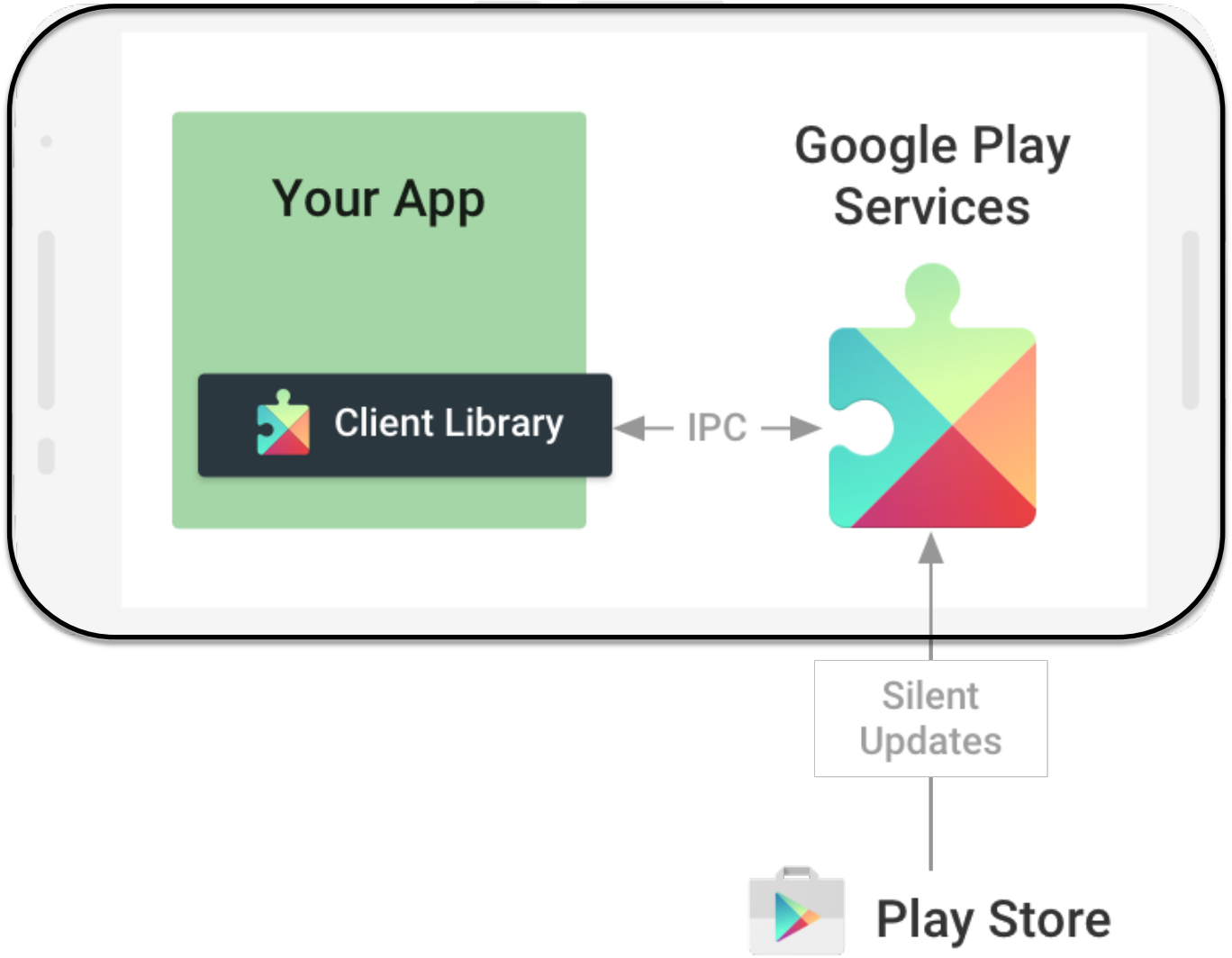
- Overview of Google Play Services and Setup
- Detailed look at
  - Google Sign-in and Authentication (Part 1)

General Overview – What is it?

- Google Play Services is "*a single place that brings in all of Google's APIs on Android 2.2 and above*."
- With Google Play Services, your app can take advantage of the latest, Google-powered features such as Maps, Google+, and more, with automatic platform updates distributed as an APK through the Google Play store.
- This makes it faster for users to receive updates and easier for developers to integrate the newest that Google has to offer.

Overview – How it Works (Google Play Services)

- The Google Play services APK on user devices receives regular updates for new APIs, features, and bug fixes.



Overview – How it Works (Google API Client)

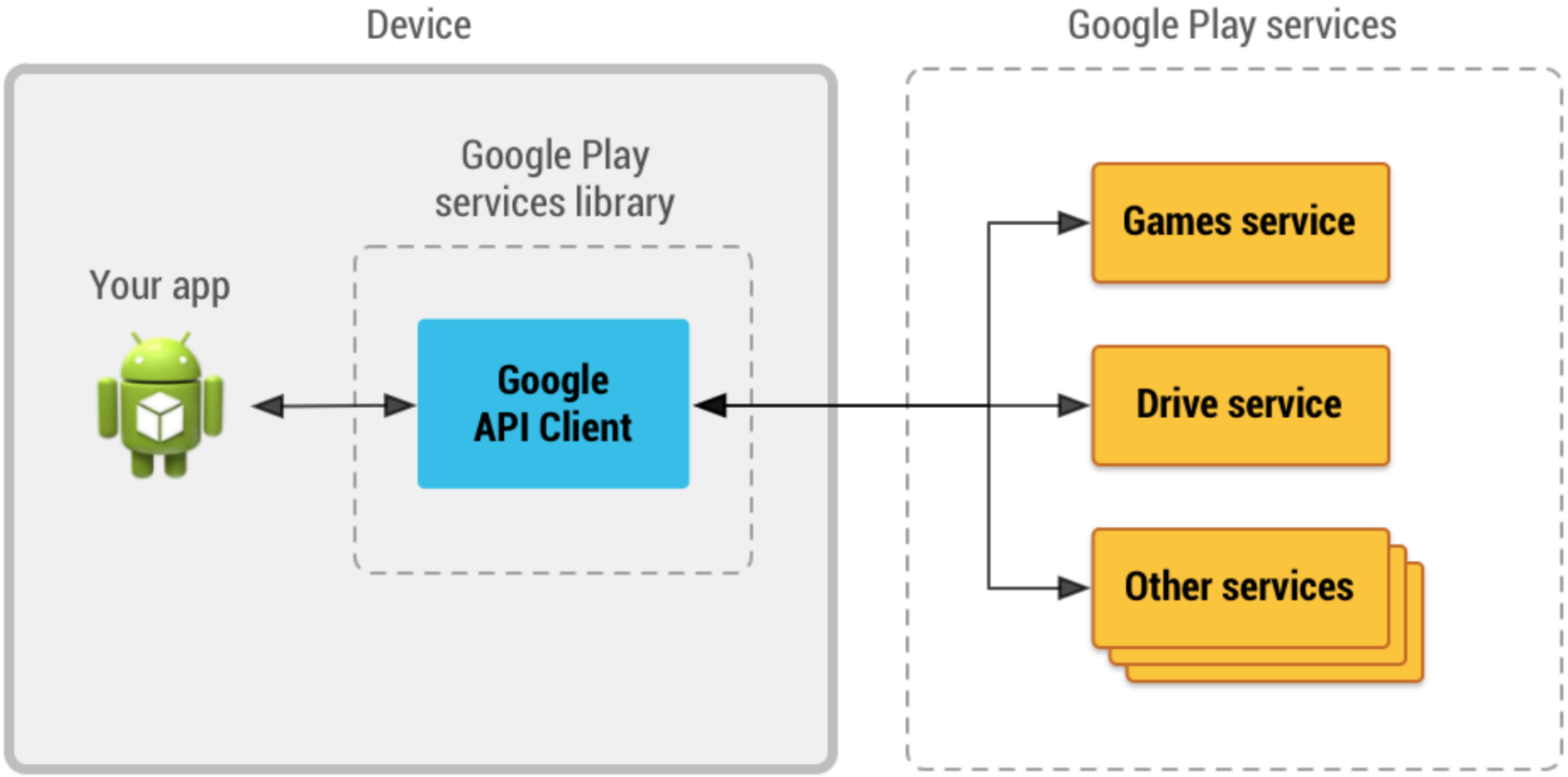


Figure 1: An illustration showing how the Google API Client provides an interface for connecting and making calls to any of the available Google Play services such as Google Play Games and Google Drive.

Overview – What You Get

Build better apps with Google

Take advantage of the latest Google technologies through a single set of APIs, delivered across Android devices worldwide as part of Google Play services.

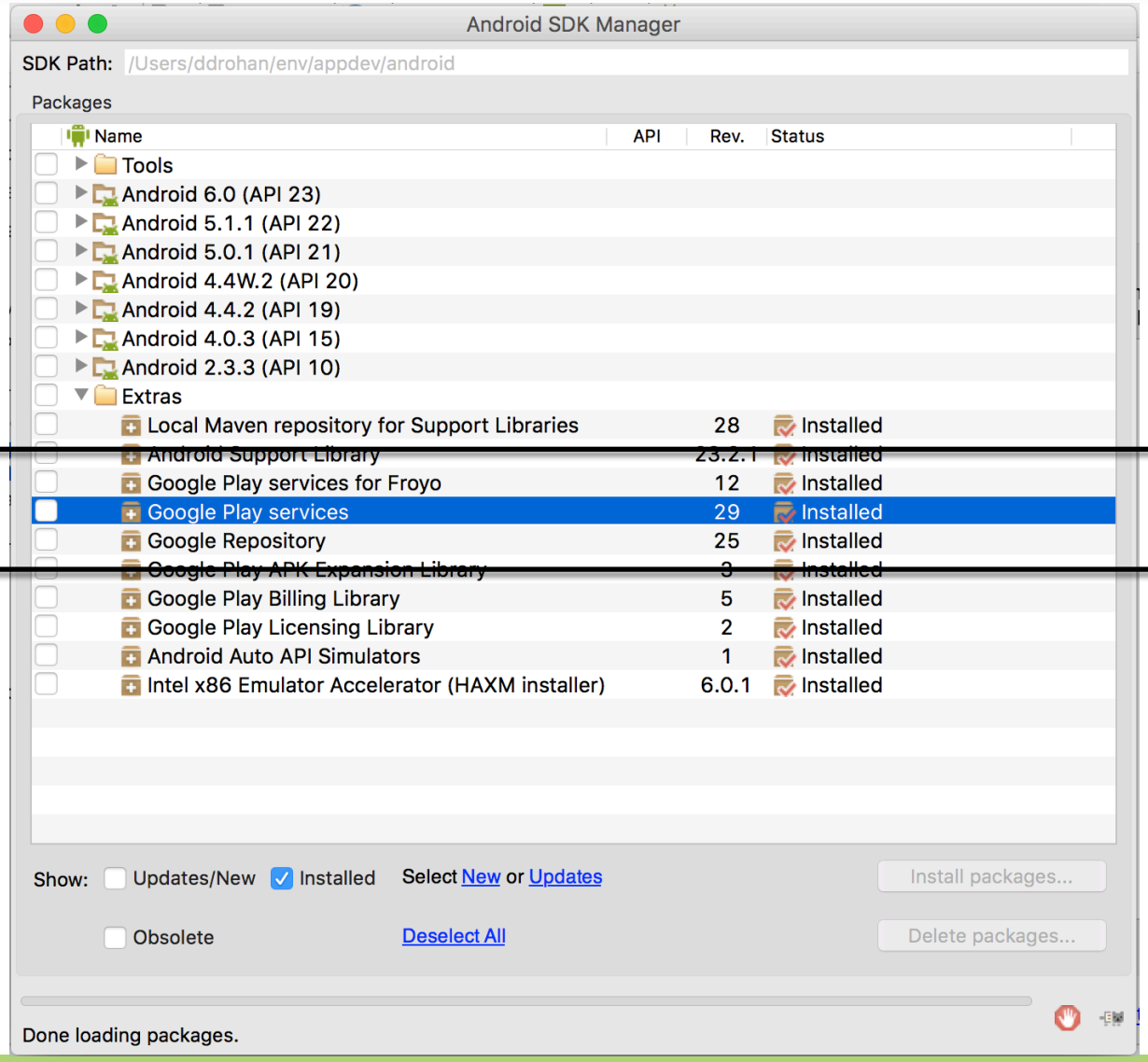
Start by setting up the Google Play services library, then build with the APIs you need.

- Set up Google Play services
- API Reference

https://developers.google.com/android/guides/overview

Setting Up Google Play Services

Download Google Play Services (Android SDK Manager)



Setting Up Google Play Services

(https://developer.android.com/google/play-services/setup.html)

- Make sure that the Google Play services SDK is installed, as shown on the previous slide.
- Create an application using Android Studio.
- In Android Studio under "Gradle Scripts", edit the build.gradle file for "Module: app"

  (not the build.gradle file for the project)

  Under dependencies (near the bottom), add the following line at the end:

```
compile 'com.google.android.gms:play-services-location:9.8.0'
```

- Save the changes and click "Sync Project with Gradle Files" in the toolbar, or click on menu item

  Tools → Android → Sync Project with Gradle Files.

Setting Up Google Play Services (continued)

- When we're finished CoffeeMate, our 'dependencies' will look something like this (version numbers may differ)…

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'

    compile 'com.android.support:appcompat-v7:23.2.1'
    compile 'com.android.support:design:23.2.1'
    compile 'com.google.code.gson:gson:2.2.3'
    compile 'com.google.android.gms:play-services-maps:8.1.0'
    compile 'com.google.android.gms:play-services-location:8.1.0'
    compile 'com.google.android.gms:play-services:8.1.0'
    compile 'com.android.support:support-v4:23.2.1'
}
```

Setting Up Google Play Services (continued)

- Edit file **`AndroidManifest.xml`** and add the following tag as a child of the <application> element:

```
<meta-data android:name="com.google.android.gms.version"
           android:value="@integer/google_play_services_version"/>
```

Note: You can ignore instructions about creating a ProGuard exception if you are building in debug mode (i.e., not release mode).

Testing Google Play Services

To test an application using the Google Play services SDK, you must use either

- A compatible Android device that runs Android 2.3 or higher and includes Google Play Store
- An Android emulator (virtual device) that runs the Google APIs platform based on Android 4.2.2 or higher (Genymotion is a good one to use – Part 2)

Popular APIs (100+ in total) *

Google Cloud APIs
- Compute Engine API
- BigQuery API
- Cloud Storage Service
- Cloud Datastore API
- Cloud Deployment Manager API
- Cloud DNS API
- More

Google Maps APIs
- Google Maps Android API
- Google Maps SDK for iOS
- Google Maps JavaScript API
- Google Places API for Android
- Google Places API for iOS
- Google Maps Roads API
- More

Google Apps APIs
- Drive API
- Calendar API
- Gmail API
- Sheets API
- Google Apps Marketplace SDK
- Admin SDK
- More

YouTube APIs
- YouTube Data API
- YouTube Analytics API
- YouTube Reporting API

Advertising APIs
- AdSense Management API
- DCM/DFA Reporting And Trafficking API
- Ad Exchange Seller API
- Ad Exchange Buyer API
- DoubleClick Search API
- DoubleClick Bid Manager API

Other popular APIs
- Analytics API
- Translate API
- Custom Search API
- URL Shortener API
- PageSpeed Insights API
- Fusion Tables API
- Web Fonts Developer API

Popular APIs (100+ in total) *

Mobile APIs

Google Cloud Messaging

Google Play Game Services

Google Play Developer API

Google Places API for Android

Social APIs

Google+ API

Blogger API

Google+ Pages API

Google+ Domains API

Part 1
Google Sign-in

Introduction

- With the Google+ Platform for Android, you can allow application users to sign in with their existing Google accounts.
- It helps you in knowing your end users and providing them with a better enriched experience in your application.
- As soon as a user allows your app to use Google Sign In, you can easily get info about the user and people in the users circles.
- You can also get access to post on Google+ on the users behalf.

Overall, it is quick and easy way to engage end users in your application.

Google Sign-in Requirements

- For integrating Google Sign-in into your Android Application, we need to complete the following :

1. Enable Google+ API on The Developers Console and create credentials for your application authentication
2. Configuring Google Play Services in Android Studio
3. Create your Android Application with Google Sign-in

1. Enable Google+ API on The Developers Console

① Go to Google Developers Console

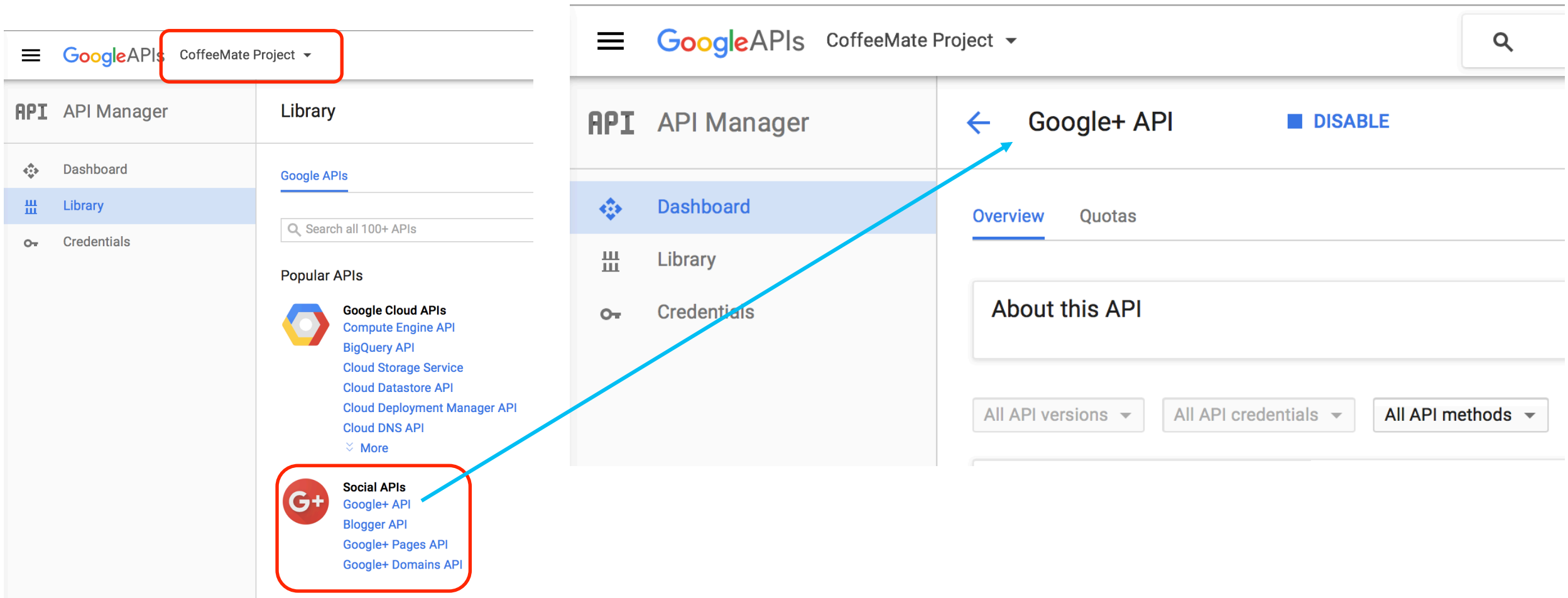
② If you don't have any existing projects, Create Project.

③ Select your project and choose ENABLE API on the menu.

④ Browse for Google+ API (under Social APIs) and turn ON its status by accepting terms and conditions.

Do not close developers console yet, you'll still use it to generate your authentication key in the next few steps.

1. Enable Google+ API on The Developers Console



1. Enable Google+ API on The Developers Console

⑤ Generate your SHA1 fingerprint

1. You can either use the java **keytool** utility, like so

```
keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

1. Enable Google+ API on The Developers Console

⑤ Generate your SHA1 fingerprint

1. You can
    1. Click on your package and choose New -> Google -> Google Maps Activity
    2. Android Studio redirects you to google_maps_api.xml with your SHA1

This gives you A LOT of extra 'bolierplate' code that you might not even need (if you're not using maps)

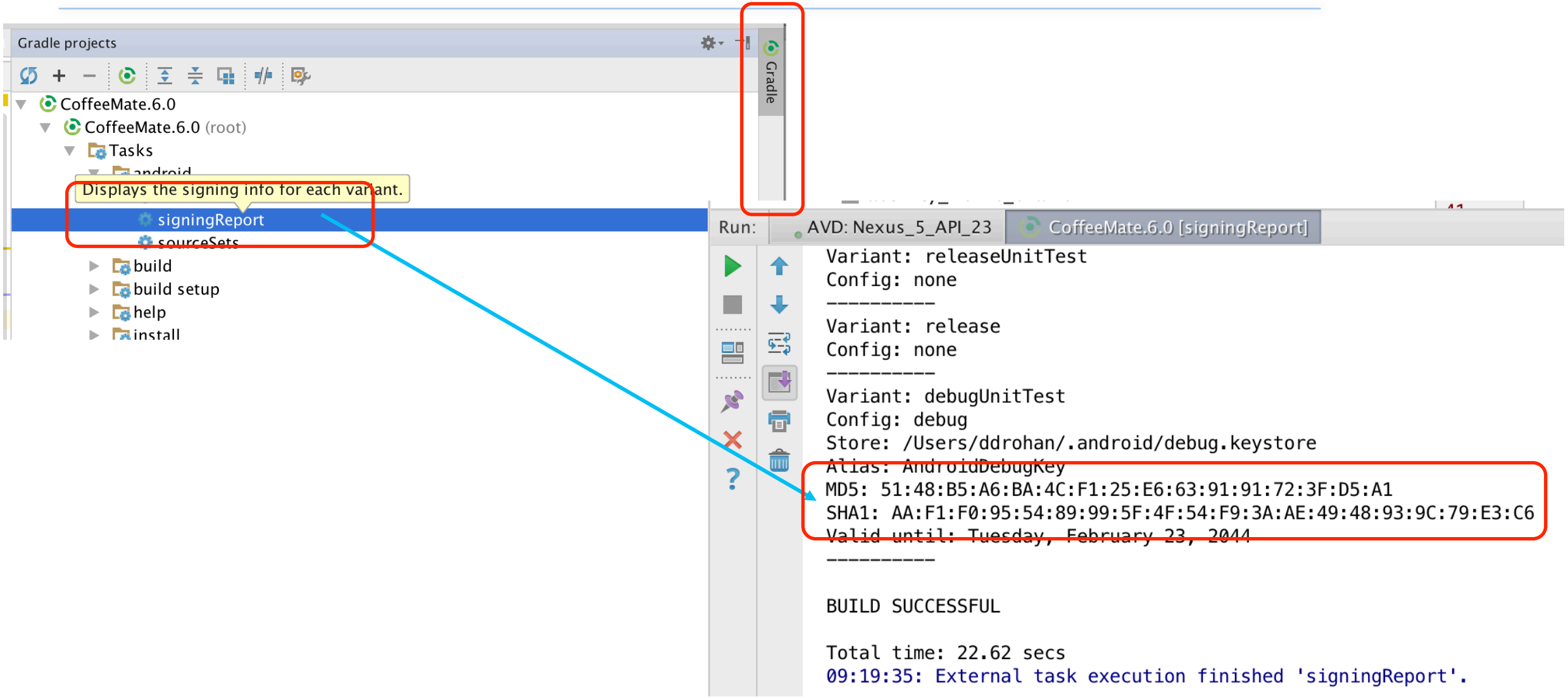
1. Enable Google+ API on The Developers Console

⑤ Generate your SHA1 fingerprint

1. Or you can
    1. Open/View Your Project
    2. Click on Gradle (From Right Side Panel, you will see Gradle Bar)
    3. Click on Refresh (Click on Refresh from Gradle Bar, you will see List Gradle scripts of your Project)
    4. Click on Your Project (Your Project Name from List (root))
    5. Click on Tasks
    6. Click on android
    7. Double Click on signingReport (You will get SHA1 and MD5 in Run Bar)

This is probably the simplest approach with minimal fuss! IMHO

1. Enable Google+ API on The Developers Console



1. Enable Google+ API on The Developers Console

⑥ Navigate to Credentials -> Create Credentials -> API Key -> Android Key,

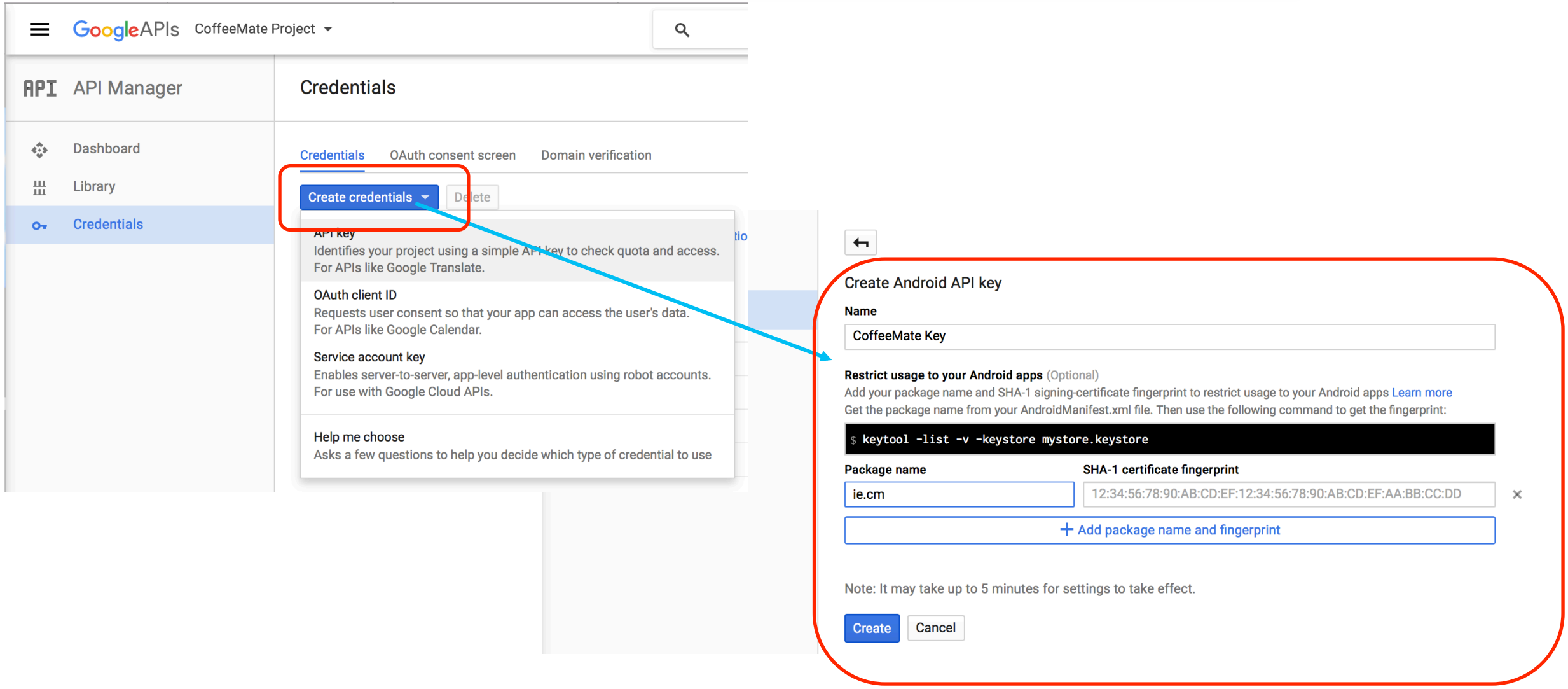
⑦ It will ask to Configure consent screen if not configured before. Fill in the necessary information and save. It will redirect you back to the creation page.

⑧ Give your Key a Name and Add package name and fingerprint

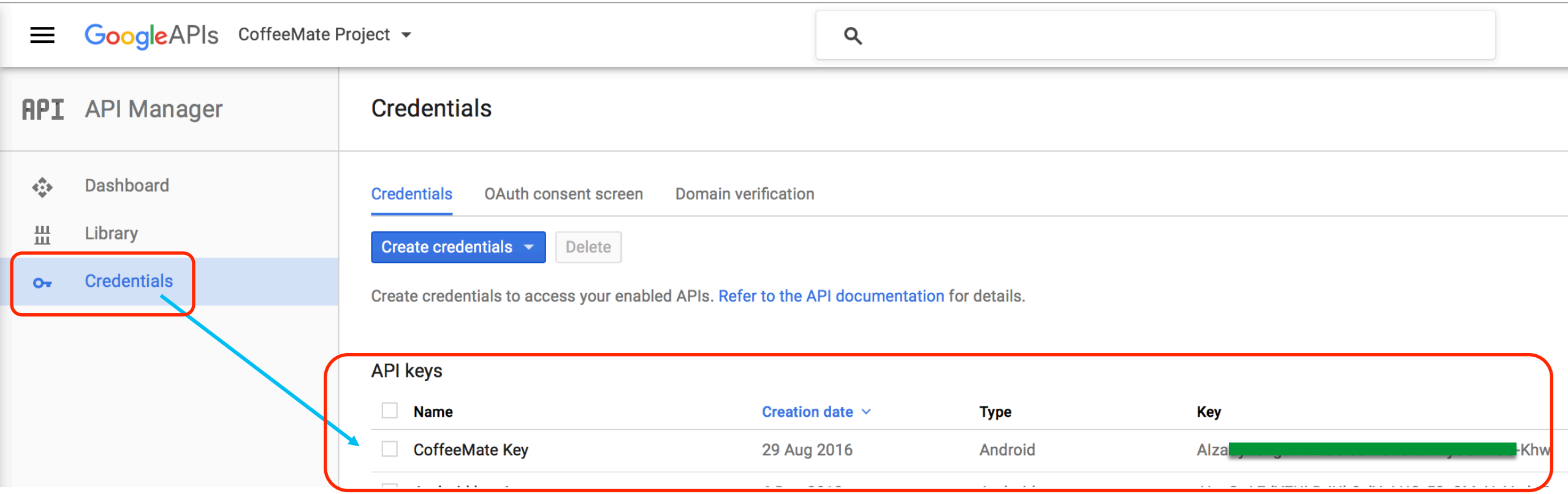
⑨ Enter your package name and your SHA1 fingerprint (generated previously) and click Create

You now have your API Key which you can use in your Android Apps to use the Google APIs

1. Enable Google+ API on The Developers Console



1. Enable Google+ API on The Developers Console



2. Configure Google Play Services

- Already Done! (from previous slides…)

3. Create your Android App (CoffeeMate)

- You'll cover this in the Labs, but we'll have a look at some of the code next

Steps in Integrating Google Sign-In into your App

- Import classes/interfaces.
- Declare that the activity implements callback interfaces.
- Declare/build `GoogleApiSignInOptions` object
- Declare/build `GoogleApiClient` object.
- Implement callback interfaces.
- Implement methods `onStart()` and `onStop()` (and possibly other lifecycle methods such as `onPause()` and `onResume()`) to gracefully handle connections to Google Play Services

Integrating Google Sign-In into Your Android App

https://developers.google.com/identity/sign-in/android/sign-in

Configure Google Sign-In & GoogleApiClient 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();
```

SignInActivity.java

If you need to request additional scopes to access Google APIs, specify them with `requestScopes`.

Configure Google Sign-In & GoogleApiClient object *

2. Then, also in your sign-in activity's `onCreate` method, create a `GoogleApiClient` object with access to the Google Sign-In API and the options you specified.

```
// Build a GoogleApiClient with access to the Google Sign-In API and the
// options specified by gso.
mGoogleApiClient = new GoogleApiClient.Builder(this)
        .enableAutoManage(this /* FragmentActivity */, this /* OnConnectionFailedListener */)
        .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
        .build();
```

SignInActivity.java

★ **Note:** To use **`enableAutoManage`**, your activity must extend **`FragmentActivity`** or **`AppCompatActivity`** (a subclass of **`FragmentActivity`**), both of which are part of the Android Support Library. You can use **`GoogleApiClient`** in a **`Fragment`**; however, the fragment's parent activity must be a **`FragmentActivity`**. If you can't extend **`FragmentActivity`**, you must manually manage the **`GoogleApiClient`** connection lifecycle.

Aside – Connecting to Google Drive Example *

```
GoogleApiClient mGoogleApiClient = new GoogleApiClient.Builder(this)
    .enableAutoManage(this /* FragmentActivity */,
                      this /* OnConnectionFailedListener */)
    .addApi(Drive.API)
    .addScope(Drive.SCOPE_FILE)
    .build();
```

You can add multiple APIs and multiple scopes to the same `GoogleApiClient` by appending additional calls to `addApi()` and `addScope()`.

Add the Google Sign-In button to your app *

1. 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" />
```

2. **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 and color scheme with the `setSize` and `setScopes` methods. Also, if you specify a Google+ social scope to `setScopes`, the sign-in button will be rendered with the red Google+ branding.

```
SignInButton signInButton = (SignInButton) findViewById(R.id.sign_in_button);
signInButton.setSize(SignInButton.SIZE_STANDARD);
signInButton.setScopes(gso.getScopeArray());
```

SignInActivity.java

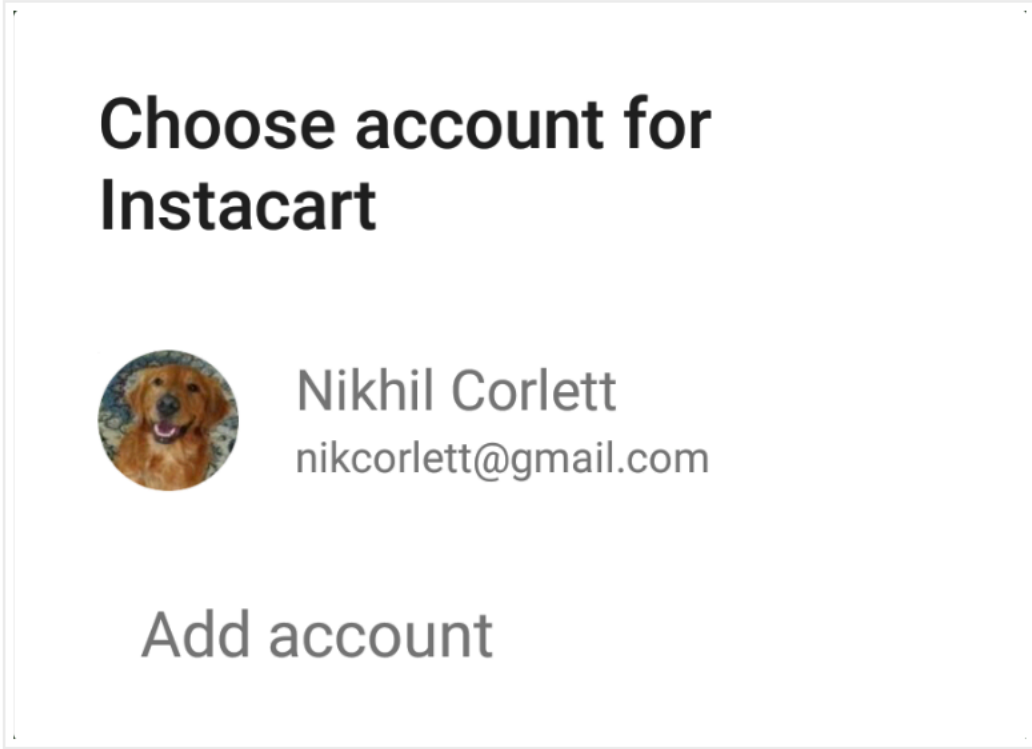
3. In the Android activity (for example, in the `onCreate` method), register your button's `OnClickListener` to sign in the user when clicked:

```
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`.

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

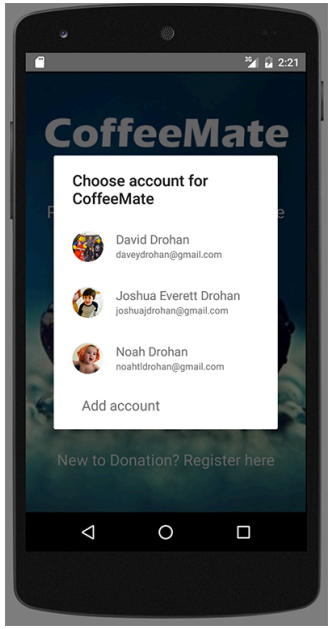


Start the sign-in flow

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

SignInActivity.java

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.



Start the sign-in flow

2. In the activity's `onActivityResult` method, retrieve the sign-in result with `getSignInResultFromIntent`.

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

    // Result returned from launching the Intent from GoogleSignInApi.getSignInIntent(...);
    if (requestCode == RC_SIGN_IN) {
        GoogleSignInResult result = Auth.GoogleSignInApi.getSignInResultFromIntent(data);
        handleSignInResult(result);
    }
}
```

SignInActivity.java

After you retrieve the sign-in result, you can check if sign-in succeeded with the `isSuccess` method. If sign-in succeeded, you can call the `getSignInAccount` method to get a `GoogleSignInAccount` object that contains information about the signed-in user, such as the user's name.

Start the sign-in flow *

```
private void handleSignInResult(GoogleSignInResult result) {
    Log.d(TAG, "handleSignInResult:" + result.isSuccess());
    if (result.isSuccess()) {
        // Signed in successfully, show authenticated UI.
        GoogleSignInAccount acct = result.getSignInAccount();
        mStatusTextView.setText(getString(R.string.signed_in_fmt, acct.getDisplayName()));
        updateUI(true);
    } else {
        // Signed out, show unauthenticated UI.
        updateUI(false);
    }
}
```

SignInActivity.java

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 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.

Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.auth.api.signin`)

❑ GoogleSignInAccount

String	getDisplayName() Returns the display name of the signed in user if you built your configuration starting from **new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)**} or with **requestProfile()** configured; **null** otherwise.
String	getEmail() Returns the email address of the signed in user if **requestEmail()** was configured; **null** otherwise.

❑ GoogleSignInOptions.Builder

GoogleSignInOptions	build() Builds the **GoogleSignInOptions** object.
GoogleSignInOptions.Builder	requestEmail() Specifies that email info is requested by your application.
GoogleSignInOptions.Builder	requestId() Specifies that user ID is requested by your application.
GoogleSignInOptions.Builder	requestIdToken(String serverClientId) Specifies that an ID token for authenticated users is requested.

Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.common.api`)

❑ GoogleApiClient

void	connect(int signInMode) Connects the client to Google Play services using the given sign in mode.
abstract void	disconnect() Closes the connection to Google Play services.
abstract boolean	isConnected() Checks if the client is currently connected to the service, so that requests to other methods will succeed.

❑ GoogleApiClient.Builder

GoogleApiClient.Builder(Context context)
Builder to help construct the **`GoogleApiClient`** object.

GoogleApiClient.Builder(Context context, GoogleApiClient.ConnectionCallbacks connectedListener, GoogleApiClient.OnConnectionFailedListener connectionFailedListener)
Builder to help construct the **`GoogleApiClient`** object.

Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.common.api`)

❑ GoogleApiClient.ConnectionCallbacks

abstract void	onConnected(Bundle connectionHint) After calling **`connect()`**, this method will be invoked asynchronously when the connect request has successfully completed.
abstract void	onConnectionSuspended(int cause) Called when the client is temporarily in a disconnected state.

❑ GoogleApiClient.OnConnectionFailedListener

abstract void	onConnectionFailed(ConnectionResult result) Called when there was an error connecting the client to the service.

Key Interfaces/Classes for Google Sign-In

(in package `com.google.android.gms.common.api`)

- `GoogleApiClient`
  - main entry point for Google Play services integration
- `GoogleApiClient.ConnectionCallbacks`
  - provides callbacks that are called when the client is connected or disconnected from the service
  - abstract methods:

```
void onConnected(Bundle connectionHint)
void onConnectionSuspended(int cause)
```

- `GoogleApiClient.OnConnectionFailedListener`
  - provides callbacks for scenarios that result in a failed attempt to connect the client to the service
  - abstract method:

```
void onConnectionFailed(ConnectionResult result)
```

CoffeeMate 5.0

Code Highlights

CoffeeMateApp (Application) *

```
/* Client used to interact with Google APIs. */
public GoogleApiClient mGoogleApiClient;
public GoogleSignInOptions mGoogleSignInOptions;

public boolean signedIn = false;
public String googleToken;
public String googleName;
public String googleMail;
public String googlePhotoURL;
public Bitmap googlePhoto;
public int drawerID = 0;
```

- Here we declare our `GoogleSignInOptions` and `GoogleApiClient` references (and other variables) to store users Google+ info.
- We populate these objects in our 'Login' process.
- Volley Request Queue also set up here (see the labs for further details).

Login (Activity) *

```
public class Login extends FragmentActivity implements
        GoogleApiClient.OnConnectionFailedListener,
        OnClickListener {
```

- Our Login Activity implements the relevant interfaces
- Here we ‘Build’ our sign in options

```
// [START configure_signin]
// Configure sign-in to request the user's ID, email address, and basic
// profile. ID and basic profile are included in DEFAULT_SIGN_IN.
app.mGoogleSignInOptions = new GoogleSignInOptions
        .Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestEmail()
        .requestProfile()
        .build();
// [END configure_signin]
```

Login (Activity) *

```
// [START build_client]
// Build a GoogleApiClient with access to the Google Sign-In API and the
// options specified by mGoogleSignInOptions.
app.mGoogleApiClient = new GoogleApiClient.Builder(this)
        .enableAutoManage(this /* FragmentActivity */,
                          this /* OnConnectionFailedListener */)
        .addApi(Auth.GOOGLE_SIGN_IN_API, app.mGoogleSignInOptions)
        .build();
// [END build_client]

setContentView(R.layout.activity_login);
findViewById(R.id.sign_in_button).setOnClickListener(this);
findViewById(R.id.disconnect_button).setOnClickListener(this);
```

- Build our client with the specific sign in options and the APi we want to use - (Google Sign-In)
- Try and sign in to Google

```
// [START signIn]
private void signIn() {
    Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(app.mGoogleApiClient);
    startActivityForResult(signInIntent, RC_SIGN_IN);
}
// [END signIn]
```

Login (Activity) *

```
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // Result returned from launching the Intent from GoogleSignInApi.getSignInIntent(...);
    if (requestCode == RC_SIGN_IN) {
        GoogleSignInResult result = Auth.GoogleSignInApi.getSignInResultFromIntent(data);
        handleSignInResult(result);
    }
}

private void handleSignInResult(GoogleSignInResult result) {
    if (result.isSuccess()) {
        // Signed in successfully, show authenticated UI.
        GoogleSignInAccount acct = result.getSignInAccount();
        app.googleName = acct.getDisplayName();
        app.googleToken = acct.getId();
        app.signedIn = true;
        app.googleMail = acct.getEmail();
        if(acct.getPhotoUrl() == null)
            ; //New Account may not have Google+ photo
        else app.googlePhotoURL = acct.getPhotoUrl().toString();

        startHomeScreen();
    } else
        Toast.makeText(this, "Please Sign in " , Toast.LENGTH_SHORT).show();
}
```

- If sign in result ok, handle it.
- On successful sign in, get the users Google info
- Take the user to the 'Home' screen

CoffeeFragment (extracts) *

- Setting up our ‘Swipe Refresh’
- Retrieving all the coffees from the server

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
  // Inflate the layout for this fragment
  View v = null;
  v = inflater.inflate(R.layout.fragment_home, container, false);
  listView = (ListView) v.findViewById(R.id.coffeeList);

  mSwipeRefreshLayout =   (SwipeRefreshLayout) v.findViewById(R.id.coffee_swipe_refresh_layout);
  setSwipeRefreshLayout();

  CoffeeApi.getAll("/coffees/" + app.googleToken, mSwipeRefreshLayout);

  return v;
}
```

CoffeeFragment (extracts) *

- Attach listener & retrieve coffees
- Detach listener
- Do we need ‘getAll( )’ on the previous slide?

```
@Override
public void onResume() {
  super.onResume();

  CoffeeApi.attachListener(this);
  CoffeeApi.getAll("/coffees/" + app.googleToken, mSwipeRefreshLayout);
}

@Override
public void onPause() {
  super.onPause();
  CoffeeApi.detachListener();
}
```

CoffeeFragment (extracts) *

- Deleting a single coffee

```
public void onCoffeeDelete(final Coffee coffee)
{
  String stringName = coffee.name;
  AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
  builder.setMessage("Are you sure you want to Delete the \'Coffee\' " + stringName + "?");
  builder.setCancelable(false);

  builder.setPositiveButton("Yes", (dialog, id) -> {
      CoffeeApi.delete("/coffees/" + app.googleToken +"/" + coffee._id);
      CoffeeApi.getAll("/coffees/" + app.googleToken, mSwipeRefreshLayout);
  }).setNegativeButton("No", (dialog, id) -> { dialog.cancel(); });
  AlertDialog alert = builder.create();
  alert.show();
}
```

CoffeeFragment (extracts) *

- Deleting multiple coffees

```
public void deleteCoffees(ActionMode actionMode)
{
  for (int i = listAdapter.getCount() - 1; i >= 0; i--) {
    if (listView.isItemChecked(i)) {
      CoffeeApi.delete("/coffees/" + app.googleToken +"/" + listAdapter.getItem(i)._id);
    }
    CoffeeApi.getAll("/coffees/" + app.googleToken, mSwipeRefreshLayout);
  }
  actionMode.finish();

  if (favourites) {
    //Update the filters data
    coffeeFilter = new CoffeeFilter(app.coffeeList,"favourites",listAdapter);
    coffeeFilter.filter(null);
  }
  listAdapter.notifyDataSetChanged();
}
```

Add (Fragment) *

```
if ((coffeeName.length() > 0) && (coffeeShop.length() > 0)
        && (price.length() > 0)) {
    Coffee c = new Coffee(coffeeName, coffeeShop, ratingValue,
            coffeePrice, false,app.googleToken,0,0,app.googlePhotoURL);

    CoffeeApi.post("/coffees/" + app.googleToken,c);
    Intent intent = new Intent(getActivity(), Home.class);
    getActivity().startActivity(intent);
```

- Creating a new coffee object
- Note the extra 'Google' parameters
- Updating our coffee data on the server

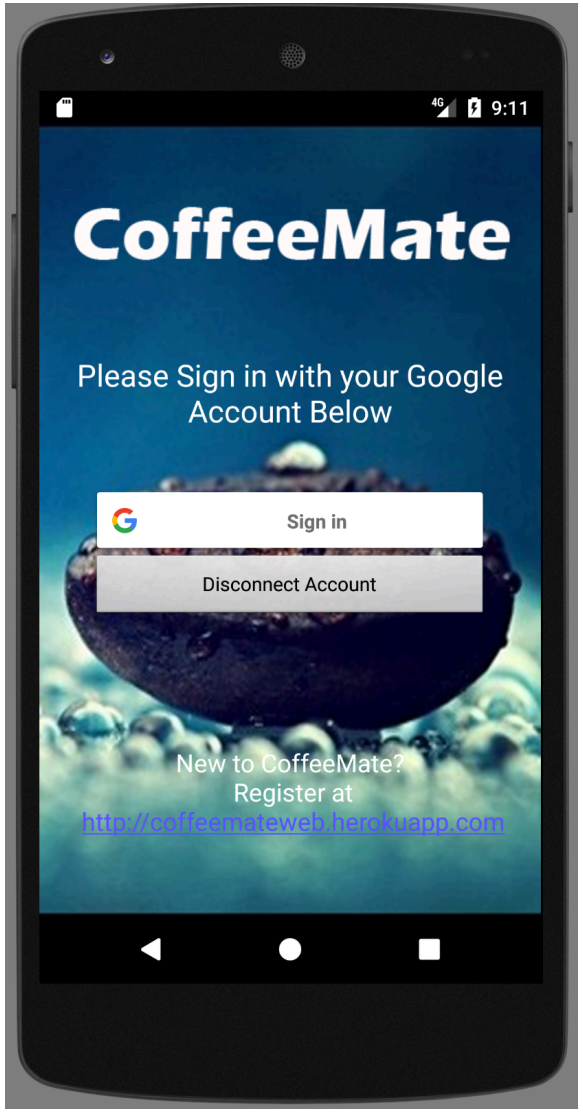
Edit (Fragment) *

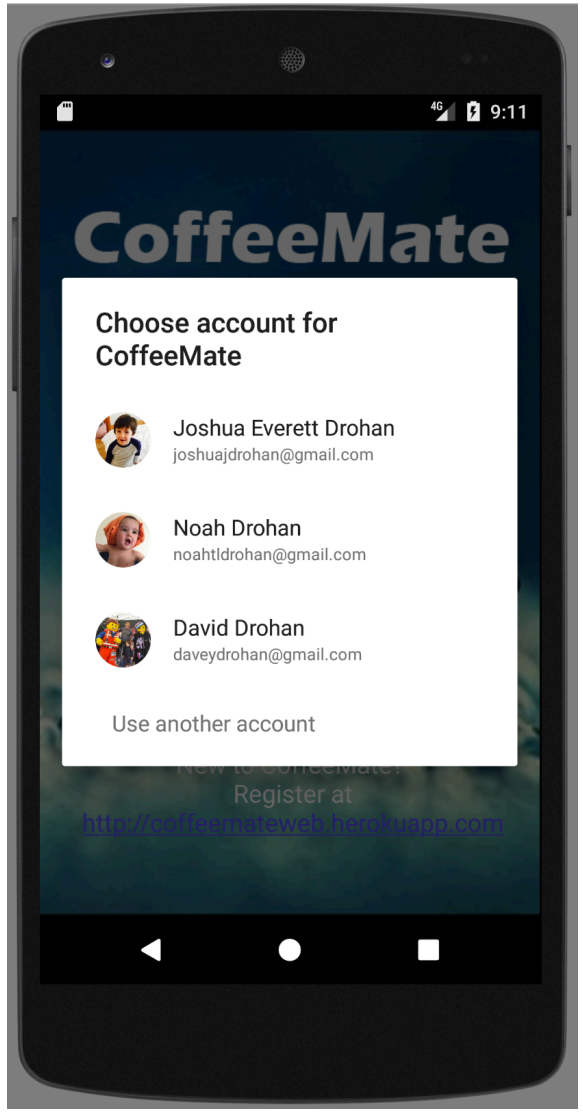
```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if(getArguments() != null) {
        coffeeID = getArguments().getString("coffeeID");
    }
}

@Override
public void onResume() {
    super.onResume();

    CoffeeApi.attachListener(this);
    CoffeeApi.get("/coffees/" + app.googleToken +"/" + coffeeID);
    titleBar = (TextView) getActivity().findViewById(R.id.recentAddedBarTextView);
    titleBar.setText("Update a Coffee");
}
```

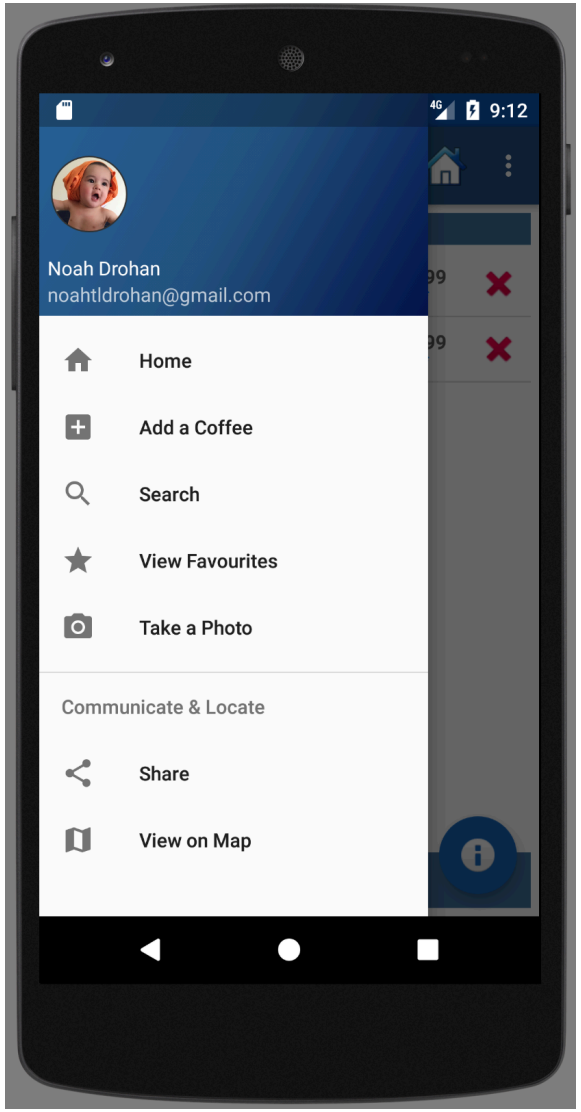
- ID passed from List (now a string value)
- Retrieving coffee data from server
- Saving our coffee data to the server

```
CoffeeApi.put("/coffees/" + app.googleToken +"/" + aCoffee._id, aCoffee);
```

CoffeeMate 5.0+







Relevant Links

- Setting Up Google Play Services
  https://developer.android.com/google/play-services/setup.html
- Integrating Google+ Sign In into your Android Application
  http://androidsrc.net/integrating-google-plus-sign-in-into-your-android-application/
- Official Docs
- https://developers.google.com/identity/sign-in/android/sign-in

Questions?


Thanks!