

ACTIVITY 3.2.2

Using Google Play Services

INTRODUCTION

Location services are one of the many benefits of running an application on a mobile device, but while convenient and powerful, location services open the door to security issues. Does a user *want* to announce his or her location? Should the app be able to decide this issue?

Security issues such as these are a growing concern among mobile app developers and their users. Many developers believe your location is considered to be *your* private data that you may wish to share or not share. You will modify TripTracker to take advantage of location services, but only if the user permits it. To do this, you will need to perform a number of steps on the device, in your app, and on the back-end services you will be using. This activity guides you through the various steps.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student
- A Google account and a project on the Google API Console

RESOURCES



Lesson 3.2 Reference Card for Google Play Services
Resources available online

Procedure

Part I: Modify the Trip Table

During this activity, you will introduce temporary syntax errors and bugs as you add functionality to your app. You will not be able to compile your app until you are instructed to do so.

- 1 Review the following slideshow.

Implementing Location Detection

- This activity introduces one of the unique features of mobile apps: *location awareness*. Mobile devices go everywhere, and adding location awareness to your app makes for a more contextual experience.
- The location API is available in *Google Play services*.
- Use it to add location awareness to your app with automated location detection and tracking.

Computer Science A

© 2016 Project Lead The Way, Inc.

Backendless: The new CheckIns Column

Column Type	Column Name	Description
String	name	Name of the trip.
String	description	Description of the trip.
Date	startDate	Date the trip starts.
Date	endDate	Date the trip ends.
Boolean	public	Set to “true” if the trip is to be made public and available to all users. Default value is “false”.
GeoPoints relationship	locations	This relationship column can save a list of GeoPoints. Each time a user saves a location, you will add to this list of GeoPoints. In your Trip class, this column is represented as an ArrayList of GeoPoints.

Computer Science A

© 2016 Project Lead The Way, Inc.

In Backendless, you will create a new GeoPoints relationship column which allows you to save a list of GeoPoint objects.

In Backendless, prepare your Trip table to store map locations, called GeoPoints. A **GeoPoint** is an object that has a latitude value and a longitude value.

GeoPoint

A Backendless object that contains variables and methods for latitude and longitude.

- 2 Open your Trip table in Backendless and navigate to the **SCHEMA** page. Create a **New** column called **locations** with a data type of **GEO POINT RELATIONSHIP**.

The screenshot shows the 'New Column' dialog box. The 'Name' field contains 'locations'. The 'Type' dropdown menu is open, displaying a list of data types: 'GEO POINT RELATIONSHIP' (highlighted), 'HVT', 'DOUBLE', 'BOOLEAN', 'FILE REFERENCE', 'DATA OBJECT RELATIONSHIP', and 'GEO POINT RELATIONSHIP'. The 'Constraints' field is empty. The 'Cardinality' field is empty.

- 3 Keep the Constraints the default value. For Cardinality select **One to Many**. Click **CREATE**.
- 4 Return to the **DATA BROWSER** page and select the check box labeled **autoload** in the locations column.

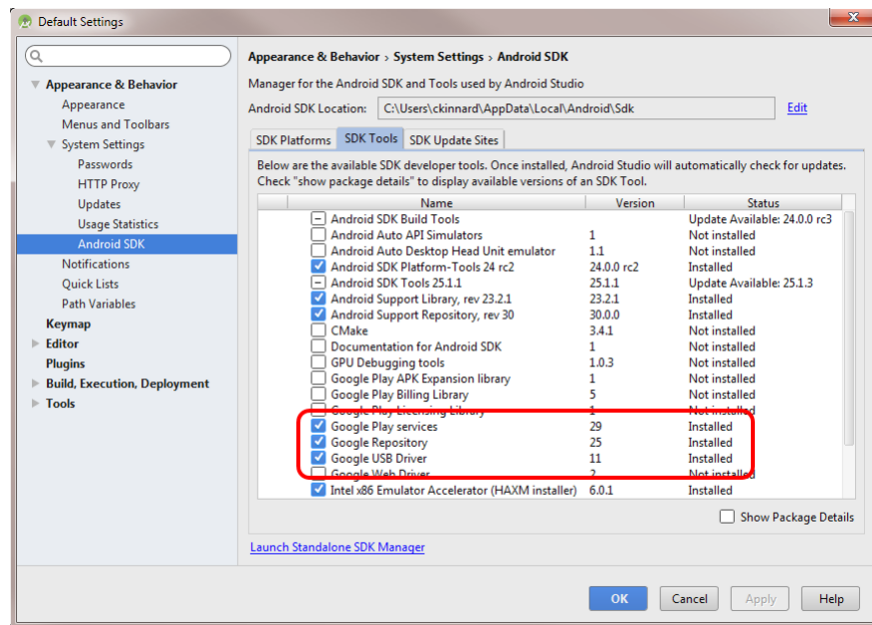
The screenshot shows the 'DATA BROWSER' page. The 'locations' column is selected, and the 'autoload' checkbox is checked. The 'locations' column is highlighted in blue. The 'autoload' checkbox is checked, and the 'name' field is visible.

Since the locations column can store multiple GeoPoints, this will configure Backendless to conveniently get all locations at once whenever a query is performed (instead of requiring a second query). The downside to this setting: if a single trip contains hundreds of locations, the query can take a long time to retrieve all of the data.

Part II: Configure Android Studio

Add the Google Play services SDK to Android Studio:

- 5 Open your TripTracker app in Android Studio. If you were unable to complete the last TripTracker activity from Lesson 3.1, *Activity 3.1.8 Public vs. Private Trips*, import *3.1.8TripTrackerApp_Solution* as directed by your teacher.
- 6 Open the SDK manager (**Tools > Android > SDK Manager**).
- 7 Select **Google Play services**, **Google Repository**, and if you plan to use a USB connected device, **Google USB Driver**.



- 8 Click **OK** and the packages will be installed.

Part III: Configure the Manifest

You need to add new configuration data and resources to your TripTracker application.

- 9 Modify *AndroidManifest.xml*
 - a. Give the application permission to use location detection after your other permissions are granted.


```
1: <uses-permission android:name="android.permission.ACCESS_
  NETWORK_STATE" />
2: <uses-permission android:name="android.permission.ACCESS_
  FINE_LOCATION" />
```

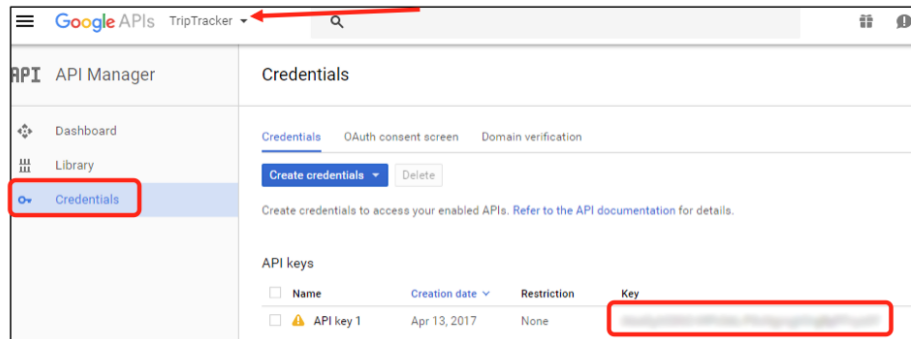
- b. Define a new map activity and two new Google services as shown on lines 1–15.

```
1: <activity
2:     android:name=".MapActivity"
3:     android:label="@string/trip_map">
4:
5:     <meta-data
6:         android:name="android.support.PARENT_ACTIVITY"
7:         android:value=".TripActivity"/>
8:
9: </activity>
10:
11: <meta-data
12:     android:name="com.google.android.gms.version"
13:     android:value="@integer/google_play_services_
14:         version"/>
15:
16: <meta-data
17:     android:name="com.google.android.geo.API_KEY"
18:     android:value="@string/google_app_id"/>
19:
20: </application>
21:
22: </manifest>
```

- 10 Define a new string value for `trip_map` in `strings.xml`:

```
1: <!-- 3.2.2 location awareness -->
2: <string name="trip_map">Trip Map</string>
```

- 11 To define a `strings.xml` value for `google_app_id`, retrieve your Google API Key from the TripTracker project you created in the previous activity:
- Navigate to the  **Google API Console** and log in with your Google account, if necessary.
 - Observe the navigation bar to confirm that you are on the TripTracker project page. Select the **Credentials** link on the left side of the page to open your Credentials page. Select and copy your API Key:



c. Back in Android Studio, in `strings.xml`, create an entry for `google_app_id` that contains your Google Android API key value.

- 12 In the `build.gradle(Module:app)` file, add the Google Play services feature listed on line 3.

```
1: dependencies {
2:     ...
3:     compile 'com.google.android.gms:play-services:8.4.0'
4: }
```

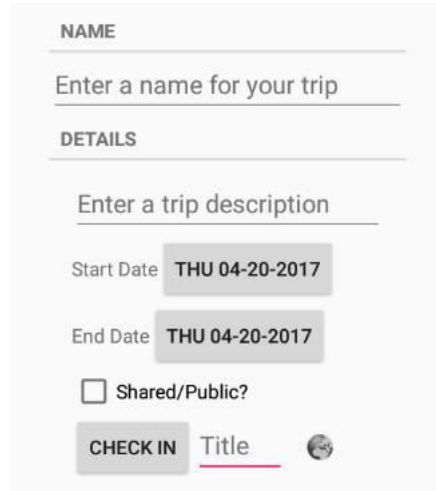
- 13 Compile your app. You will not have any new functionality; just confirm that you have no syntax or configuration errors.

Part IV: Include New Drawables and Views

- 14 Obtain a copy of the 3.2.2 *TripTracker_StarterCode* from your teacher and copy or extract the files to a temporary location.
- 15 Copy the `map.png` file into your project's `res/drawable` folder. You will use it in an image button in the next step.
- 16 Open your TripTracker in Android Studio and in the `strings.xml`, add the resources shown below. You will use these string values in the next few steps.

```
1: <string name="check_in_button_label">Check In!</string>
2: <string name="location_hint">Enter location title</string>
3: <string name="check_in_error_message">Please enter a location
  title.</string>
4: <string name="check_in_error_title">Missing Information!</
  string>
5: <string name="location_error_message">Your location is not
  detected. Make sure location services are turned on.</string>
6: <string name="show_map_button_label">Show Map</string>
7: <string name="location_permission_denied">You will not be
  able to create locations on the Map or use the Check-In
  feature for your trips.</string>
```

- 17 Open your *fragment_trip.xml* and add new elements to manage the location awareness features. Since there will be more elements on your screen, you will also make a few modifications to your existing widgets. Use the image below to guide you.



- Arrange the Start Date widgets in a horizontal layout and the End Date widgets in another horizontal layout.
 - Arrange three new widgets in a horizontal layout below the Shared/Public widget:
 - A Check In Button to save the location and its title.
 - An EditText for the Title of the location that you can use to remember the places you visit.
 - An image button used to start a “map” activity.
 - You may also want to shorten the hints for your other EditText widgets.
- 18 You will be creating a new map activity, so create a new layout resource file called *activity_map.xml*.
- 19 Replace the default XML code with the following:

```
1: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2:     android:layout_width="fill_parent"
3:     android:layout_height="fill_parent">
4:
5:     <fragment
6:         android:layout_width="match_parent"
7:         android:layout_height="match_parent"
8:         android:id="@+id/tripMap"
9:         android:name="com.google.android.gms.maps.
           MapFragment" />
10:
11: </LinearLayout>
```

For an explanation of the `fragment` UI component and the `MapFragment`, refer to this slide:

activity_map.xml

MapFragment

```
<LinearLayout ...
    ...
    <fragment
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/tripMap"
        android:name="com.google.android.gms.maps.MapFragment"
    />

</LinearLayout>
```

Computer Science A

© 2016 Project Lead The Way, Inc.

- So far, you have seen a fragment as a Java class. You can also use a fragment UI component in the layout file, as long as it is associated with a class that extends `FragmentActivity`. In this lesson, your `MapActivity` class will use the `activity_map` layout file to extend `FragmentActivity`.
- `MapFragment` is a map UI component available in the Google Play services SDK. You can retrieve it in the presenter class (`FragmentActivity`) to access the Google Map object.
- Note: If you were building an app to support Android API 11 or below, then you would need to replace `MapFragment` with `SupportMapFragment`.
- Another option for the map UI is to eliminate the fragment UI and embed a `MapView` component in a `LinearLayout` or `RelativeLayout` instead. This will require more code for forwarding life cycle events to the `MapView`. (For more information about `MapView`, access: <https://developers.google.com/maps/documentation/android/reference/com/google/android/gms/maps/MapView>.)

In this case, the layout would be:

```
<LinearLayout>

    <com.google.android.gms.maps.MapView
        android:id="@+id/tripMap"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

- a. What class should the `MapFragment` be associated with?
- b. If you are planning to support Android API version 11 or older, which map class should you use?

Part V: Create a New MapActivity

- 20 From the provided source code, copy *MapActivity.java* into your project's *java* folder. This file will have some errors, which you will fix in the coming steps.
- 21 Before you fix the errors, get to know your new activity. Review the following slides:

Google Play Services The GoogleApiClient Interface

Main integration point to Google Play services, including:

- Google Drive
- Google Maps
- Location Services
- Games

```
1. mGoogleApiClient = new GoogleApiClient.Builder(getActivity())
2.                     .addConnectionCallbacks(this)
3.                     .addOnConnectionFailedListener(this)
4.                     .addApi(LocationServices.API)
5.                     .build();
```

Computer Science A

© 2016 Project Lead The Way, Inc.

GoogleApiClient is an interface that acts as the main entry point for Google Play services integration.

The code above shows the GoogleApiClient being instantiated: It builds and configures a GoogleApiClient instance and specifies the objects that contain the Callback and Listener methods.

- Line 1: For TripTracker, create a new GoogleApiClient in TripFragment's onCreate(Bundle) method.
- Line 2: The callback object is the TripFragment itself, specified with the parameter this. TripFragment will implement the onConnected and onConnectionSuspended callbacks.
- Line 3: The listener object for a failed connection is also the TripFragment itself (this). TripFragment will implement onConnectionFailed listener.
- Line 4: Specifies the Google APIs to be used. In this case, it is only the LocationServices.API, but you can add as many APIs as needed by adding more addApi() calls before the build() call.

Details about the callbacks and listeners are further described on Slide 9.

To use a GoogleApiClient, fragments or activities need to implement the GoogleApiClient.ConnectionCallbacks and GoogleApiClient.OnConnectionFailedListener interfaces. Implementation details for overriding the following methods are provided for you in the activity, but are also summarized here:

- onStart() and Resume(): calls connect() which must be done before any service operation is to be executed
- onStop() and onPause(): calls disconnect() since your app will no longer be using the service
- onConnected(Bundle): for Location services, get the last location that the app knew about
- onConnectedSuspended(int): no code required, the service will automatically attempt to reconnect
- onConnectionFailed(ConnectionResult): attempt to resolve the problem with connectionResult.startResolutionForResult(...)

MapActivity

```
1. public class MapActivity extends FragmentActivity {
2.     @Override
3.     protected void onCreate(Bundle savedInstanceState) {
4.         super.onCreate(savedInstanceState);
5.         setContentView(R.layout.activity_map);
6.         setUpMapIfNeeded();
7.         ....
8.     }
9.     protected void onResume() {
10.        super.onResume();
11.        setUpMapIfNeeded();
12.    }
13.    private void setUpMapIfNeeded() {
14.        if (mGoogleMap == null) {
15.            mGoogleMap = (MapFragment) getFragmentManager()
16.                .findFragmentById(R.id.tripMap)
17.                .getMap();
18.        }
19.        mGoogleMap.setMyLocationEnabled(true);
20.    }
21. }
```



Computer Science A

© 2016 Project Lead The Way, Inc.

MapActivity displayMarkers()

```
1. private void displayMarkers() {
2.     if ((mGoogleMap != null) && (mTripCheckIns != null)) {
3.
4.         for (GeoPoint checkIn : mTripCheckIns) {
5.             String checkInTitle =
6.                 (String) checkIn.getMetadata("title");
7.             MarkerOptions options = new MarkerOptions();
8.             options.position(new LatLng(checkIn.getLatitude(),
9.                                         checkIn.getLongitude())); //here
10.            options.title(checkInTitle);
11.
12.            mGoogleMap.addMarker(options).showInfoWindow();
13.        }
14.        moveCameraToFirstLocation();
15.    }
16. }
```

Computer Science A

© 2016 Project Lead The Way, Inc.

- Lines 4–11: Loop through mTripCheckIns which is a List<GeoPoint> to load each point into a Marker on the map.
- Line 5: Each “checkIn” GeoPoint has metadata that can store its title. GeoPoint metadata is a feature of the Backendless GeoPoint object.
- Line 7: Creates a new instance of the class MarkerOptions. This class is used to specify the attributes that will be set when adding a marker to a map. This includes the position, title, and other options (unused by TripTracker) like an icon, a description, etc.
- Lines 8–10: Set the position and the title options for the marker. Note that the position(LatLng) method requires one LatLng parameter. Therefore, you create a new LatLng object using the getLatitude() and getLongitude() methods provided in com.backendless.geo.GeoPoint.
- Line 11: Adds a new marker to the mGoogleMap object using the MarkerOptions object, which specifies the position and title of the marker. It also shows the info window to display the title.
- Line 14: Calls the custom method moveCameraToFirstLocation() which is explained on the next slide.

MapActivity moveCameraToFirstLocation()

```
1. private void moveCameraToFirstLocation() {  
2.  
3.     GeoPoint geoLocation = mTripCheckIns.get(0);  
4.  
5.     LatLng firstLatLng = new LatLng(geoLocation.getLatitude(),  
6.                                     geoLocation.getLongitude());  
7.  
8.     CameraPosition firstPosition = new CameraPosition.Builder()  
9.                                     .target(firstLatLng)  
10.                                    .zoom(15)  
11.                                    .build();  
12.  
13.     mGoogleMap.animateCamera(  
14.         CameraUpdateFactory.newCameraPosition(firstPosition));  
15. }
```

Computer Science A

© 2016 Project Lead The Way, Inc.

The purpose of this method is to zoom in on the first check-in location for that trip.

- Line 3 gets the first checkIn from the mTripCheckIns variable, which is an ArrayList<GeoPoint> using the get(0) method.
- Lines 5–6 create an object of type LatLng with the geoLocation object from line 3.
- Line 8 calls the CameraPosition.Builder() method in preparation to build a CameraPosition that will be used when updating the camera in line 12.
- Line 9 sets the target of the CameraPosition to the LatLng object created in line 5.
- Line 10 sets the zoom amount, which can be a decimal number between 1 and 20 – approximately. More information can be found at [https://developers.google.com/android/reference/com/google/android/gms/maps/model/CameraPosition.Builder#zoom\(float\)](https://developers.google.com/android/reference/com/google/android/gms/maps/model/CameraPosition.Builder#zoom(float))
- Line 11 calls the build() method on the Builder to create the instance of the CameraPosition class and set it in the variable firstPosition on line 8.
- Line 13 animates the camera on the Google Map object to move to the position and zoom level identified in the CameraPosition (firstPosition).

Your MapActivity will start when the user selects the Map image button (*map.png*) on the Trip Details page.

- a. List the methods in MapActivity.
- b. As best you can, describe the algorithm in loadTripCheckIns, including a description of displayMarkers.
- c. Why is the setUpMapIfNeeded() called in onCreate() and onResume()?

Part VI: Modify Trip

- 22 In Android Studio, open *Trip.java*.
 - a. Create a new instance variable called `locations` as an `ArrayList` of `GeoPoints`. Note that there are two types of `GeoPoints`; select the `com.backendless.geo.GeoPoint`.
 - b. Initialize your `locations` array list in the constructor.
 - c. Create the getter and setter for `locations`.

Part VII: Modify TripActivity

- 23 Two activities in your app require permissions to use location services, the new map activity and your trip activity. `MapActivity` came with location detection permission; you will use it to add location detection permission to your `TripActivity`. In *MapActivity.java* `onCreate`, find the `if` statement along with the Log message that asks the user if the activity can access locations. Copy this code block.
- 24 In *TripActivity.java*, paste the code block in a location similar to its location in `MapActivity`.
- 25 Also in *MapActivity.java*, find the callback method called `onRequestPermissionsResult` that handles the permission request. Copy this method and paste it into `TripActivity`.
- 26 The previous step introduced a syntax error related to the `TAG` constant; fix it.

The two activities that require permission to use location services are now set to ask the user's permission to do so.

Part VIII: Modify TripFragment

The Check In and Map features require callbacks, listeners, and connection methods. These methods are in the source code file called *Partial_TripFragment.java*.

- 27 Copy and paste these methods into your *TripFragment.java*. It does not matter where, as long as you don't paste them *into* another method.
- 28 Review the following slides which show you how to make `TripFragment` become a **Google API Client**.

GoogleApiClient

The interface that represents the main entry point for Google Play services integration. Its abstract methods include the `connect()` and `disconnect()` methods to start and end the connection with Google Play services.

TripFragment Callbacks

- Implements interfaces

```
GoogleApiClient.ConnectionCallbacks,  
GoogleApiClient.OnConnectionFailedListener
```

- Override methods

```
onConnected(Bundle connectionHint)  
onConnectionSuspended(int i)  
  
onConnectionFailed(ConnectionResult connectionResult)
```

Computer Science A

© 2016 Project Lead The Way, Inc.

Overriding TripFragment onConnected()

```
@Override  
public void onConnected(Bundle connectionHint) {  
    Log.i(TAG, "Location services connected.");  
    try {  
        mLastLocation = LocationServices  
            .FusedLocationApi  
            .getLastLocation(mGoogleApiClient);  
    } catch (SecurityException se) { ...}  
  
    if (mLastLocation != null) {  
        Log.i(TAG, "Your location is: " +  
            mLastLocation.getLatitude() + ":" +  
            mLastLocation.getLongitude());  
    }  
    else {  
        Log.i(TAG, "Your location is NULL!!!");  
    }  
}
```

Computer Science A

© 2016 Project Lead The Way, Inc.

This method is called when the GoogleApiClient connects successfully with its service. You need to implement the onConnected method to take action when the connection is successful.

There are multiple APIs that can be used to request location values and updates. In this activity, FusedLocationApi is used because it has a simpler interface. The API manages the location providers behind the scenes (network or GPS), provides high accuracy locations, and is power-efficient (minimizes the app's battery use).

The getLastLocation() method returns the last location detected, which is in most cases, the current location of the device. You can then use mLastLocation.getLatitude() and .getLongitude() to access the location coordinates.

Note that the code checks whether the mLastLocation is not null to avoid any null pointer exceptions in cases such as:

- Testing on an emulator that has not yet detected the location. (If using Genymotion Emulator, refer to supplemental documentation 3.2 Using Google Play Services in Genymotion™ to test on Genymotion virtual devices.)
- Running the app on a device that has location services turned off.

Overriding TripFragment onConnectionSuspended()

```
@Override
public void onConnectionSuspended(int i) {
    Log.i(TAG, "Location services suspended. Attempting
    to reconnect.");
}
```

Computer Science A

© 2016 Project Lead The Way, Inc.

This method is called when the GoogleApiClient connection to the service is temporarily lost (due to weak connections for example).

You need to implement the onConnectionSuspended method to take any action when the connection is suspended, like logging messages. GoogleApiClient automatically attempts to restore the connection and sends a call to onConnected once connection is successful.

Overriding TripFragment onConnectionFailed ()

```
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
    if (connectionResult.hasResolution()) {
        try {
            // Start an Activity that tries to resolve the error
            connectionResult.startResolutionForResult(getActivity(),
                CONNECTION_FAILURE_RESOLUTION_REQUEST);
        }
        catch (Exception e) {
            Log.d(TAG, e.getMessage());
        }
    } else {
        Log.d(TAG, "Location services connection failed: code " +
            connectionResult.getErrorCode());
        ...
    }
}
```

Computer Science A

© 2016 Project Lead The Way, Inc.

The onConnectionFailed method is called when the GoogleApiClient fails to connect to the service. You need to implement onConnectionFailed to take action when the connection fails.

The connectionResult parameter is an object that can be used to handle the connection failure. Some of its methods include:

- hasResolution() returns true if the specific error can be resolved by starting an activity that interacts with the user. For example, if the connection failed because the device is not connected to the Internet, then the error can be resolved by starting an activity with a dialog message asking the user to connect to the Internet first.
- startResolutionForResult() launches the built-in Android™ activity that can help resolve the issue (as explained in previous bullet).
- getErrorCode() returns the error code for the failed connection (like API_UNAVAILABLE, CANCELED, NETWORK_ERROR, TIMEOUT).

Note that the constant CONNECTION_FAILURE_RESOLUTION_REQUEST will be set to 9000 (in the next step of the activity), which is the code for requesting a resolution for a connection failure (set by Android).

- a. When you declare your `TripFragment` class, implement the two required interfaces to enable `TripFragment` as a Google API Client.
- b. Create some new instance variables:

```
1: private GoogleApiClient mGoogleApiClient;  
2: private ImageButton mMapButton;  
3: private Location mLastLocation;  
4: private final static int CONNECTION_FAILURE_RESOLUTION_  
    REQUEST = 9000;  
5: private EditText mLocTitle;  
6: private Button mCheckInButton;
```

- c. In `onCreate`, start the Google services with lines 3–7 below just after you set the activity's title.

```
getActivity().setTitle(R.string.trip_details_text);  
mGoogleApiClient = new GoogleApiClient.Builder(getActivity())  
    .addConnectionCallbacks(this)  
    .addOnConnectionFailedListener(this)  
    .addApi(LocationServices.API)  
    .build();
```

- d. When should `MyCheckInOnClickListener` be invoked?
- e. When should `MyShowMapClickListener` be invoked?
- f. A new object is part of this algorithm, a `Toast`. What does it do? This is an Android object, so refer to the Android Developer's website and summarize in your own words.

- 29 Build your app and run it on the emulator or a connected Android device.
- 30 Navigate to a Trip Details screen and notice the new elements: the Check In button and the Map. They are visible and enabled, but do not yet do anything.
- 31 Review the TODO items in `TripFragment`. Your first TODO will be in `manageLocationViews` to manage the new UI elements.


You will write new functionality so that the location views (location title text, the Check In button, and the Map image button) are only available on an *existing* trip *owned* by the current user.

- 32 Read through the sub-steps below and plan your algorithm with pseudocode. With pseudocode in hand, write the `manageLocationViews` method:
 - a. Determine whether the user is modifying an existing trip or creating a new trip (if necessary, reference your `onCreate` method to determine this).
 - b. For *existing* trips, use the `mEnabled` boolean variable to set the state of the `mLocTitle`, `mCheckInbutton`, and `mMapButton` views.

- c. For a *new* trip, hide the location views. The user must create and save a trip before these views are visible.
- d. Don't forget to invoke this method from `onCreateView`.
- e. Test your app to confirm that the location features are enabled only on an existing trip owned by the current user. Be sure to test in public and private modes.

Your UI should be complete and you should be ready to implement the back-end functionality.

33 Review the incomplete `MyCheckInOnClickListener` `onClick` method and complete the algorithm so the current location is added your trip. Read through all of these sub-steps and plan your algorithm with pseudocode.

- a. Review the documentation and code samples in Backendless at  **Establishing Relations with Geo Points via API.**
- b. Review the `deleteTrip` method. Like the delete, you will need to create a new thread and wait until a `GeoPoint` is saved in the database before you can add it to your trip.
- c. Complete the algorithm:
 - i. Create an array list named `tripLocations` to contain a list of `GeoPoint` objects and initialize it with the trip's current locations.
 - ii. Create a `GeoPoint`. Assign the lat and long values and use `addMetadata` to give it a "title". You do not need to add a category; it is optional.
 - iii. Add the point to `tripLocations`.
 - iv. Save `tripLocations` to `mTrip` with a call to `setLocations`.
 - v. Make the Backendless `setRelation` call to connect the trip to its `tripLocations`:

```
1: Backendless.Data.of(Trip.class).setRelation(mTrip,
    "locations:GeoPoint:n", tripLocations);
```

The `locations:GeoPoints:n` syntax tells Backendless that a column called `locations` will contain many (`n`) `GeoPoints`.

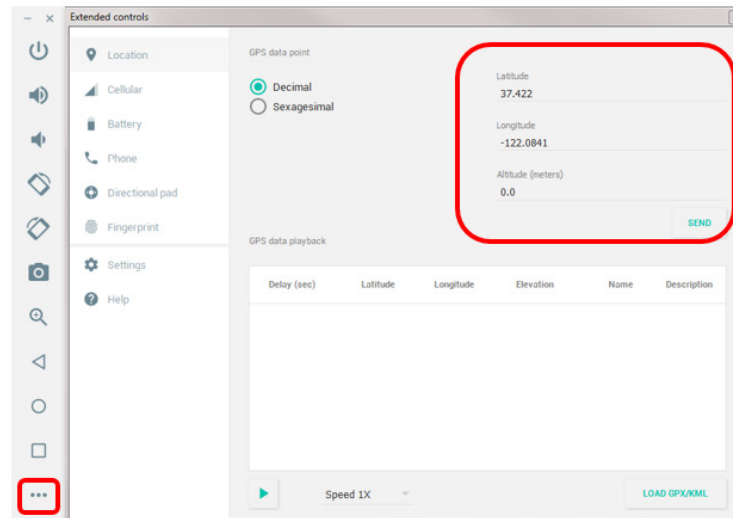
- vi. Modify the message used by `Log` and `Toast` so the app notifies the user that the location was saved.

To test your new algorithm you will to set up location services, which is the next step.

Part IX: Test Location Services

Your app is ready to detect and save locations on a Google map. If you are using an emulator in place of a physical device, you need to simulate locations because the emulator does not have a GPS. This step is detailed below.

- 34 Navigate to the Trip Details screen with an existing trip owned by the current user.
- 35 Click the **Map** icon. If you are using a device, skip to the next step. If you are using an emulator, complete the following sub-steps.
 - a. On the emulator, select the Extended Controls icon. A default Latitude and Longitude are shown.



- b. Keep the default latitude and longitude values (the headquarters of Google) or enter your own values. Click **Send** to send the location to the Map.

Your map window won't change, but you can use the My Location dot on the map to zoom to it.

- c. Close the Extended controls window.

- 36 Return to the Trip Details screen and enter a title for this location.
- 37 Select **Check In**. You should see a message confirming your location.
- 38 Save your trip. In Backendless, confirm that the location is stored in your Trip table.
 - a. In your Trip table, select the text **Multiple Geo Points** for one of your trips.



The Geolocation storage is shown. This is where Backendless stores all Geo Points for your trip, with latitudes, longitudes, and titles. You can also click the **Geolocation** icon in the left panel and the **Default** category to see all Geo Points for all trips.

- b. In this view, you can delete any locations you may want to discard.

- 39 Back in your app, select the trip you just saved and select the **Map** button. The Map should zoom to the location and show the title.
- 40 Right now, the trip does not remember the previous locations you have saved for a trip. Use intent data to save trip locations between your TripList and Trip activities:
- Create a new string in IntentData such as `EXTRA_TRIP_LOCATIONS`. Put the trip locations on the intent like you do with other intent data.
 - To get the locations array list from the intent, you must use `getSerializableExtra` in place of `getStringExtra`, for example,

```
1: locations = (ArrayList<GeoPoint>)intent.  
getSerializableExtra(Trip.EXTRA_TRIP_LOCATIONS);
```

You may ignore the warning in Android Studio that states your code has an unchecked cast. Since you put the array list of locations on the intent, you know you can get an array list of locations, without worrying about a cast exception occurring.

- Test that you can add to a trip's locations after saving it.

- 41 Test your app again: From My Trips/Trip List, use the context menu to delete a trip that has locations. Your app should crash!

The crash occurs because you are trying to delete a trip without first deleting the Geo Points it references.

- 42 To fix this bug, remove all locations from the current trip before you remove the trip from Backendless.

NOTE

You have two `deleteTrip` methods, one in `TripFragment` and one in `TripListFragment`.

- Create a new method in `Trip` that uses the `ArrayList clear()` method to remove all entries in `locations`.
- Use your new method in the `deleteTrip` methods to delete the Geo Points before removing the trip.

Not only does Backendless remove the list of points from the Trip table, it removes the Geo Points from the Geolocation storage.

- 43 Thoroughly test your app.

CONCLUSION

1. Whenever you create a new Java class or modify one to introduce new functionality, you have to consider whether it is going to extend a class and/or implement one or more interfaces.

Discuss the classes that `TripFragment` extends and implements. Why does it extend some and implement others?

2. Security issues around apps and mobile devices continue to evolve. Some questions you may ask:

Is the current user authentication sufficient?

Do you agree with the Private/Public settings as defined by the app?

Should a user know where location data is stored?

Should a user be able to delete all location data, everywhere?

Discuss with your partner security issues that may be of a concern to either of you and write a summary of them here.