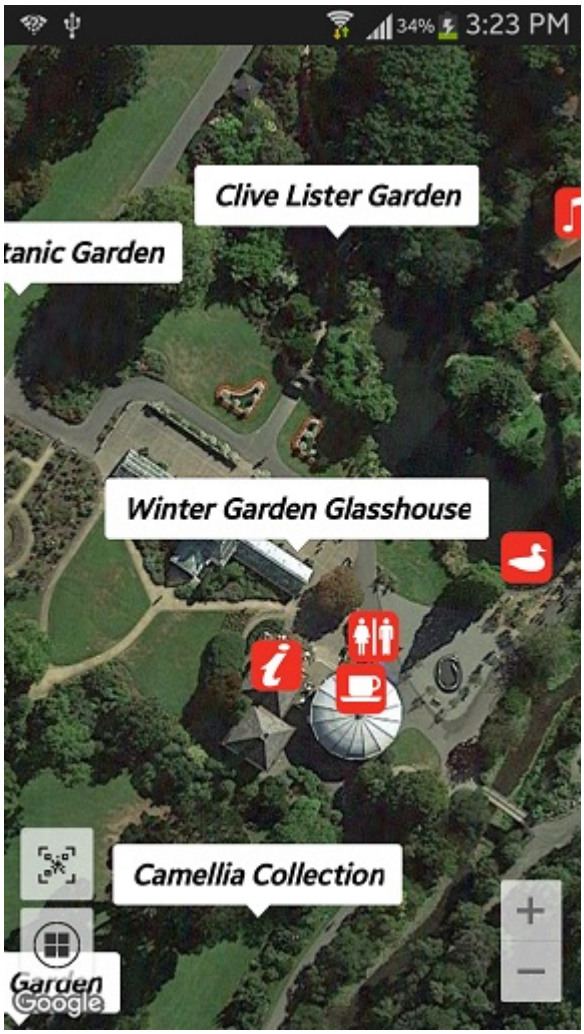
Pathmapper Android Documentation

Custom Marker Labels

By default, Google Map Markers only allow for one Marker title to be visible on the Map at any one time. To be able to effectively display titles for all the Collections in the Botanic Gardens we needed to create custom Markers. The Icon Generator class creates a Bitmap image which is then used to set the icon property of the Marker. It performs exactly the same as a Marker and adjusts to zoom and click events.



```
public void CreateLabels()
{
    for (int i = 0; i < TOTAL_LABELS; i++) {

        String title = labelList[i].getTitle();

        SpannableStringBuilder sb = new SpannableStringBuilder(title);
        sb.setSpan(new StyleSpan(Typeface.BOLD_ITALIC), 0, title.length(), SPAN_EXCLUSIVE_EXCLUSIVE);
        sb.setSpan(new ForegroundColorSpan(Color.BLACK), 0, title.length(), SPAN_EXCLUSIVE_EXCLUSIVE);
        BitmapDescriptor image = BitmapDescriptorFactory.fromBitmap(generator.makeIcon(sb));
        featureIcons.add(image);    // Add to feature marker list
    }
}
```

The challenge with using Labels in this way meant the screen became very cluttered and hard to distinguish each label. The solution we used involved changing the labels into a star icon when the zoom level reached a specific value. This improved readability and gave the user more visible screen area to view the gardens when zoomed out. The delay in transitioning from Labels into Star Icons is very minimal.

```
// Listener for Zoom level on Camera Movement
public class CameraZoomHandler implements GoogleMap.OnCameraMoveListener {
    @Override
    public void onCameraMove() {

        // Read current zoom level
        float zoomLevel = map.getCameraPosition().zoom;

            // Swap Labels for Star Icons when best zoom limit reached
            if (zoomLevel < 18.0) {

                if(!starIconsVisible)
                {
                    for (Marker marker : markerList) {
                        if (marker != null && marker.getTag() != null)
                            marker.setIcon(BitmapDescriptorFactory.fromResource(R.drawable.star));
                    }
                    starIconsVisible = true;
                }
            }
            else {

                if(starIconsVisible)
                {
                    int markerCounter = 0;
                    //Loop through each marker and turn it back into a label from a star
                    for (Marker marker : markerList) {
                        if (marker != null && marker.getTag() != null) {
                            BitmapDescriptor b;
                            if (hasConnection)
                                b = collectionMapMarker.getCollectionIcons().get(markerCounter);
                            else
                                b = featureMapMarker.getFeatureIcons().get(markerCounter);
                            marker.setIcon(b);
                            markerCounter++;
                        }
                    }
                    starIconsVisible = false;
```

Facility Icons

There are additional markers displayed to the user at all times, to show important locations around the Botanic Gardens. The Red Icons include all the Entrances to the Gardens as well as useful features such as the Café, Information Centre and Toilet facilities.

These Markers were created by extracting the icons from the digital version of the Botanic Gardens Brochure and used as a Bitmap icon similar to the Custom Markers.

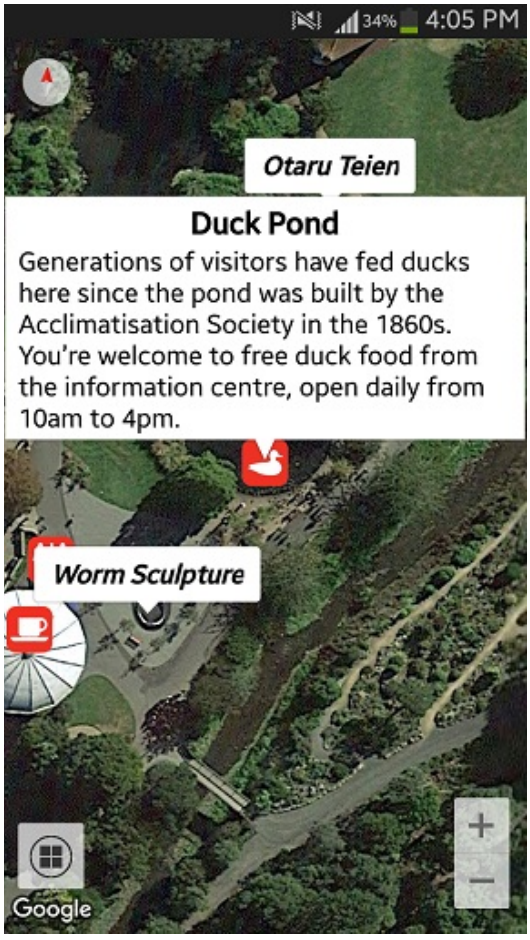
```
public void ShowIcons() {
    ShowEntranceIcons();
    ShowRedIcons();
}

public void ShowEntranceIcons() {

    for (int i = 0; i < ITEMS_SIZE; i++)
    {
        Marker marker = map.addMarker(new MarkerOptions()
                .position(locations[i])
                .title("Entrance")
                .icon(BitmapDescriptorFactory.fromResource(R.drawable.entrance))
                .infoWindowAnchor(0.5f, 0.5f));
        marker.setTag("information");
        entrances.add(marker);
    }
}

public void ShowRedIcons()
{
    Marker MARK_1 = map.addMarker(new MarkerOptions()
            .position(TOILET_1)
            .title("Toilets")
            .icon(BitmapDescriptorFactory.fromResource(R.drawable.toilets))
            .infoWindowAnchor(0.5f, 0.5f));
    MARK_1.setTag("information");
    redIcons.add(MARK_1);
    Marker MARK_2 = map.addMarker(new MarkerOptions()
            .position(TOILET_2)
            .title("Toilets")
```

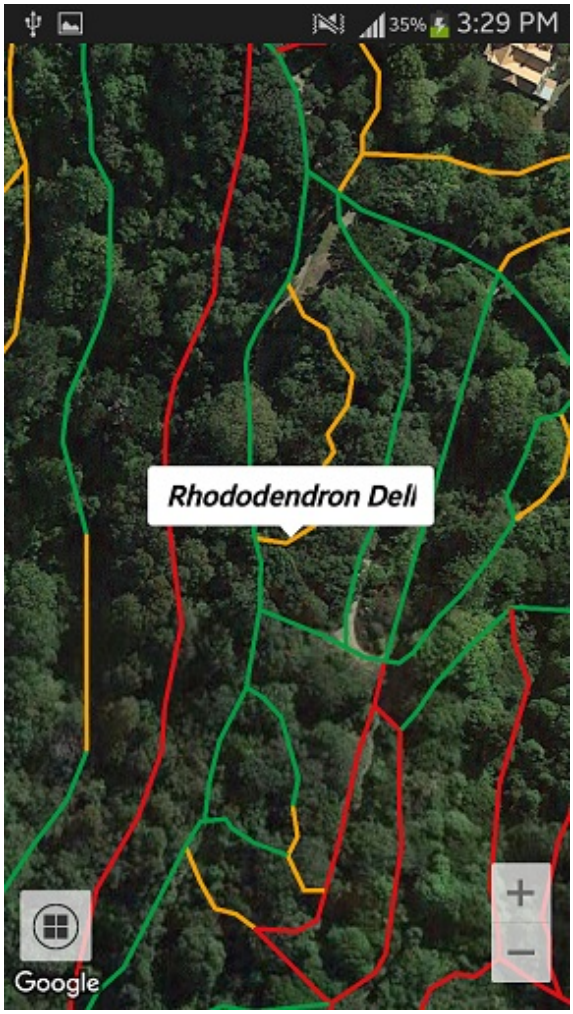

17% 2:08 PM
Car Park
Google

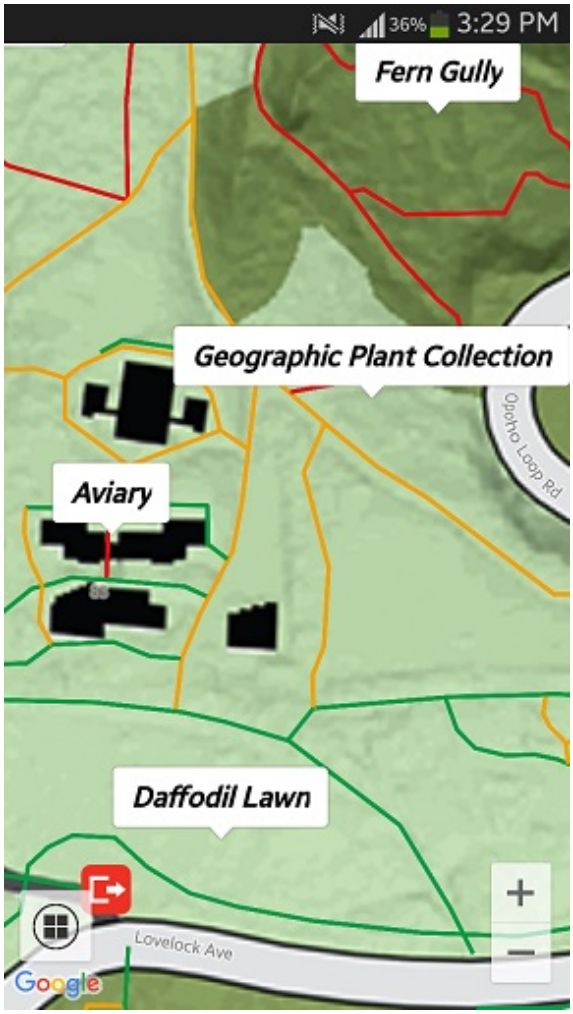

34% 4:05 PM
Otaru Teien
Duck Pond
Generations of visitors have fed ducks here since the pond was built by the Acclimatisation Society in the 1860s. You're welcome to free duck food from the information centre, open daily from 10am to 4pm.
Worm Sculpture
Google

Accessibility

This track displays all accessible walking areas in the Botanic Gardens using Red, Orange or Green to represent the difficulty for these tracks.

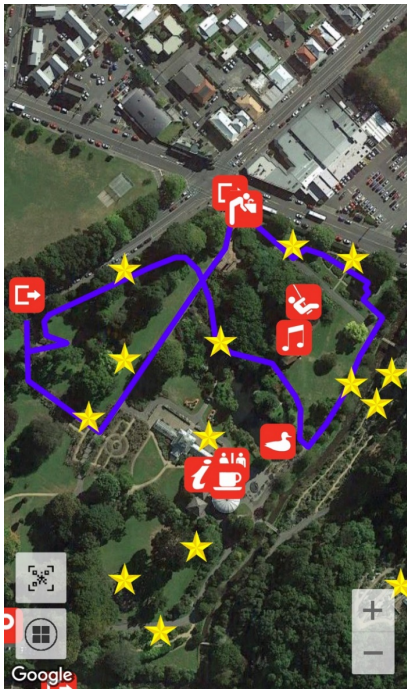
Red, Orange, Green



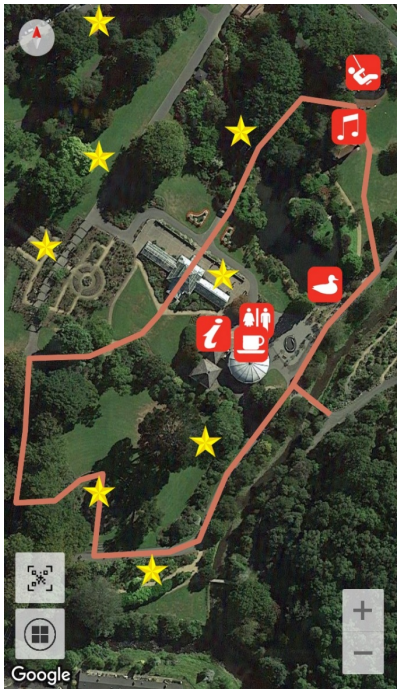


Walking Tracks

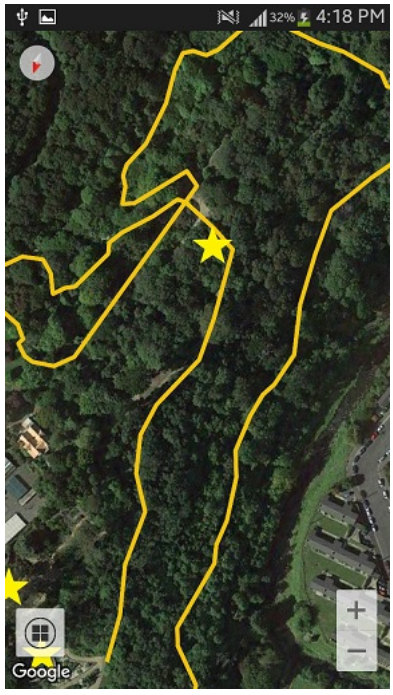
There are 5 Walking Tracks in the Botanic Gardens which can selected from the Options Menu. One track is displayed at any one time and they can be turned off via the options menu also. The following tracks are displayed.



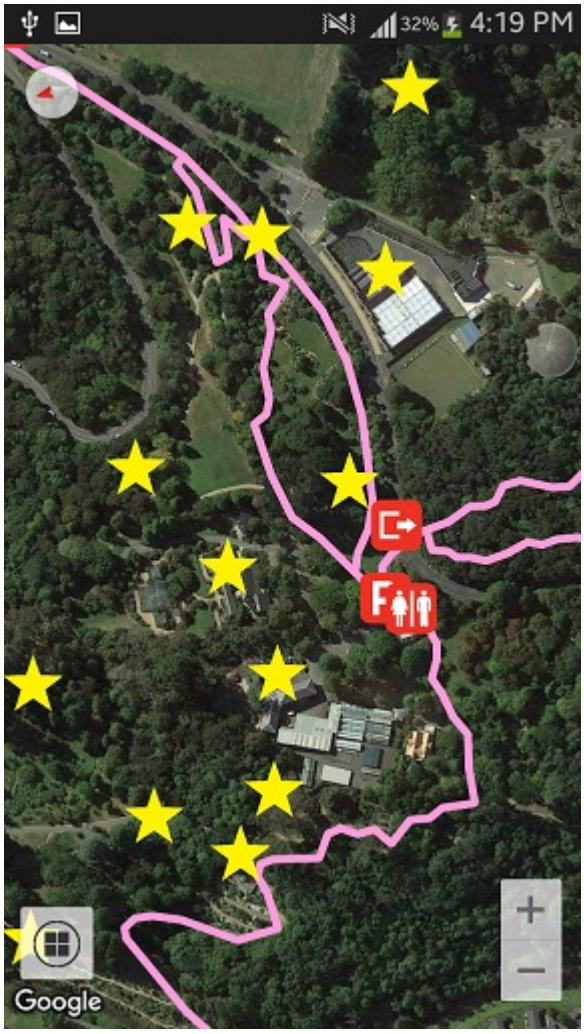
Sample plant collections in 20 Minutes



See Iconic Features in 15 minutes



Cultivated beds and wild woodland in one hilly hour



New Zealand return in one hilly hour

Travel around the world in one hilly hour

Map Type

There are 3 different terrain options to choose from. The 1st option is the Satellite view which is set as the default Map Type when Pathmapper is started. The 2nd option is the Normal Google Map. This provides the user with a choice as to what suits them and is easier to understand.

Satellite View

Normal View

Ground Overlay

The 3rd option is an overlay of the botanic gardens placed upon the Google Map. This image is similar to the image found in the Brochure that is available to the public. Working with Bitmaps in Android has limitations due the Memory available. The overlay image is provided in different resolutions to prevent out of memory errors on older devices. Android Documentation recommends using a Library to manage Bitmap loading and displaying for this reason and we have used the universal image loader to perform async loading of the Overlay.

```
public void setOverlay() {
    imageLoader = ImageLoader.getInstance();
    imageLoader.loadImage(imageUri, new SimpleImageLoadingListener() {
        @Override
        public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
            overlay = loadedImage;

            //Set the bounds of where the overlay will be
            LatLngBounds bounds = new LatLngBounds(
                    new LatLng(-45.86330119466015, 170.5134915580934),       // South west corner
                    new LatLng(-45.85400797018859, 170.529046579201));      // North east corner

            ground = map.addGroundOverlay(new GroundOverlayOptions()
                    .image(BitmapDescriptorFactory.fromBitmap(overlay))
                    .bearing((float) 1.65)
                    .positionFromBounds(bounds));
        }
    });
}
```

HttpManager

Manager for sending async calls to the gardens API. To access the API through HttpManager, the client class must implement the IAPIUser interface.

The HttpManager performs multiple functions that retrieve data from the API, the general formula is:

1. Retrieve JSON string from a web service
2. Convert JSON string into an Object List
3. Pass Object List back to the client class

```
IAPIUser client;
List<Collection> collectionList;

public void GetAllCollections(IAPIUser client)
{
    this.client = client;
    collectionList = new ArrayList<>();

    new HttpWorkerAllCollections().execute("/api/collection/");
}


public class HttpWorkerAllCollections extends AsyncTask<String, Void, String>
{

    // Retrieve JSON string from web service
    @Override
    protected String doInBackground(String... params) { return HTTPGetJSON(params[0]); }

    @Override
    protected void onPostExecute(String data)
    {
        // Convert JSON String to Object List
        extractCollectionArray(data);

        // Pass Object List back to client
        client.ReceiveAllCollections(collectionList);
    }
}
```

Example: Get all Collections

IAPIUser

Interface to return values from API calls. Each interface method to receive data has a corresponding HTTPManager method that accesses the API.

Data	HttpManager Function	IAPIUser Function
Single Item	GetItem	ReceiveItem
Single Collection	GetCollection	ReceiveCollection
All Collections	GetAllCollections	ReceiveAllCollection
All Items in Collection	GetItemsInCollection	ReceiveItemsInCollection

```
public class test extends AppCompatActivity implements IAPIUser
{
    TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_test);

        textView = (TextView) findViewById(R.id.textView);

        int itemID = 7;
        new HttpManager().GetItem(this, itemID);
    }

    @Override
    public void ReceiveItem(Item item)
    {
        textView.setText(item.getName());
    }
}
```

Example: Get single Item

BarCodeReaderActivity

Scan the QR code of an Item and launch the ItemInfoActivity.

```
public class QrDetector implements Detector.Processor<Barcode>
{

    @Override
    public void receiveDetections(Detector.Detections<Barcode> detections)
    {

        final SparseArray<Barcode> barcodes = detections.getDetectedItems();

        new Runnable()
        {
            public void run()
            {
                String barcodeTxt = barcodes.valueAt(0).displayValue;

                int itemID = Integer.parseInt(barcodeTxt);
                new HttpManager().GetItem(BarCodeReaderActivity.this, itemID);

            }
        };
    }
}
```

When a Barcode is detected, query the API to check that the QR value has an associated Item in the database.

```
@Override
public void ReceiveItem(Item item)
{
    if(item != null)
    {
        Intent itemInfo = new Intent(BarCodeReaderActivity.this, ItemInfoActivity.class);
        itemInfo.putExtra("item", item);
        itemInfo.putExtra("type", "item");
        startActivity(itemInfo);
    }
    else
    {
        new InvalidQRDialog().show(getFragmentManager(), "confirm");
    }
}
```

If the API returns an item, continue to the ItemInfoActivity, else show the InvalidQR dialog.

ItemInfoActivity

Display details on items retrieved from the database. Stores a Collection object and a list of the Items in the Collections. This lets the user switch between viewing a Collection and Items.

```
String type = i.getStringExtra("type");

if(type.equals("collection"))
{
    viewShown = eView.COLLECTION;
    collection = i.getParcelableExtra("collection");
    httpManager.GetItemsInCollection(this, collection.getId());

}
else if(type.equals("item"))
{
    viewShown = eView.ITEM;
    Item itemToFind = i.getParcelableExtra("item");
    scannedItemID = itemToFind.getId();
    ReceiveItem(itemToFind);
}
```

Because the ItemInfoActivity can either show details for a Collection or an Item, the Initial type of item must be specified.

If the initial object is a Collection, the Activity makes an API call to retrieve the Items in the Collection then displays the data.

If the initial object is an Item, the Activity retrieves the Item of the given ID and marks it as the scanned Item. Then with the Item, the Activity can retrieve the Item's Collection. Then the Activity makes an API call to retrieve the Items in the Collection then displays the data.

Menu System

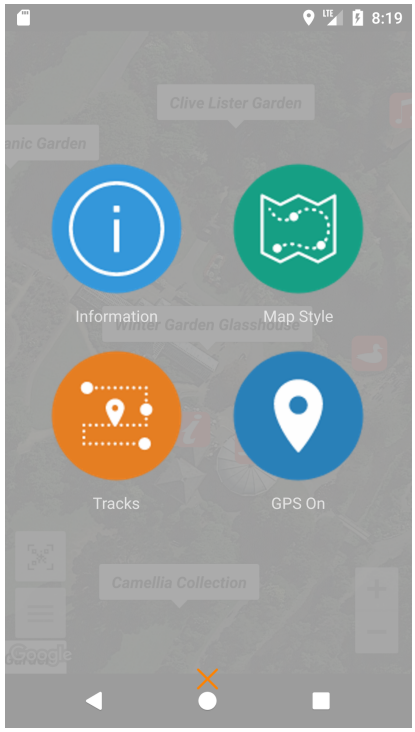
Now PathMapper have a new looking menu system, along with smoother animation and better looking icons.

The idea was inspired by joelan and this menu dependencies is from:

https://github.com/joelan/WeiboPopupMenu

Icons and everything here is copyright free and our team is very happy to see this nice menu system working on our App.

All documentation is in this GitHub page so you are able to easily play around with it.

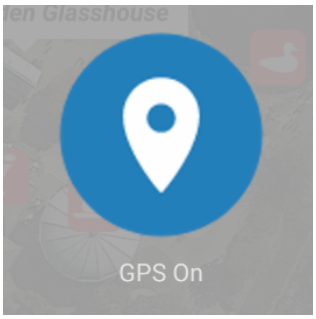


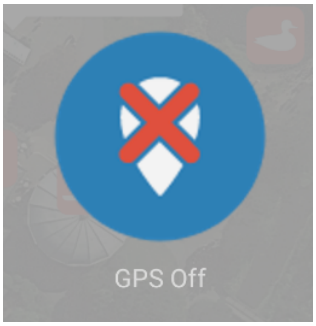
The menu system functions by encapsulating the user's preferences in an Options object, then passing the object back the main map. The buttons of the menu trigger a dialog with the available settings. Selecting an options sets the corresponding parameter in the Options object.

```
PopMenuItem gpsitem;

if (!hasGPS)
{
    gpsitem = new PopMenuItem("GPS Off", getResources().getDrawable(R.drawable.gps_off));
}
else
{
    gpsitem = new PopMenuItem("GPS On", getResources().getDrawable(R.drawable.gps_on));
}
PopMenu mPopMenu = new PopMenu.Builder().attachToActivity(BaseMapActivity.this)
        .columnCount(2)
        .addMenuItem(new PopMenuItem("Information", getResources().getDrawable(R.drawable.info_xxl)))
        .addMenuItem(new PopMenuItem("Map Style", getResources().getDrawable(R.drawable.location_map_flat)))
        .addMenuItem(new PopMenuItem("Tracks", getResources().getDrawable(R.drawable.tracks_icon)))
        .addMenuItem(gpsitem)
        .setOnItemClickListener((popMenu, position) -> {

                if (position == 0)
                {
                    new InfoListDialog().show(getFragmentManager(), "info");
                }
                if (position == 1)
                {
                    new MapTypeDialog().show(getFragmentManager(), "style");
                }
                if (position == 2)
                {
                    new TracksDialog().show(getFragmentManager(), "track");
                }
                if (position == 3)
                {
                    if (!hasGPS)
                    {
                        Toast.makeText(BaseMapActivity.this, "Currently GPS off, you can scan QR code to know where you at.
                    }
                    else
                    {
                        Toast.makeText(BaseMapActivity.this, "Bad GPS accurcy? Simply turn your GPS off on your phone and re
                    }
                }
        })
        .build();
mPopMenu.setmBackGroundColor(Color.parseColor("#f0A6A6A6"));
mPopMenu.setmCloseMenuMarginbottom(30);
mPopMenu.setmMarginTopRemainSpace(2.5f);
mPopMenu.show();
```





Apart having provided method from itself library, the GPS button is also intelligent, which means it can detect GPS state of device and show user by two different intuitive icons.

Options

The Options object encapsulates the user's preferences, as set in the MenuActivity.

Property	Type	Description
collections	boolean	!!!
myLocation	boolean	If user's location is shown
internet	boolean	If user has an internet connection
showStars	boolean	!!!
showLabels	boolean	!!!
showReds	boolean	!!!
mapType	eMapTypes	Styling of the main map
track	eTrack	Track overlay to display on the map

Google Location API

- Using Google Location API rather than implementing original Android GPS code bring more benefit on the accuracy of GPS unit on the device.

```
//=============================================Google Play Service Methods======
protected abstract void start();

public void googleAPIConnection() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addApi(LocationServices.API)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();
    mGoogleApiClient.connect();
}
```

- Simply initialise a data type of GoogleApiClient, then in googleAPIConnection method, build all connection within GoogleApiClient Builder.
- This interface provide you four method(onConnected, onConnectionSuspended, onConnectionFailed, onLocationChanged), in getLastKnownLocation method:
- Firstly check permission
- If there were no lastLocation yet, in startLocationUpdates method, Using FusedLocationApi provided by GoogleApiClient which will then request location updates to get devices last known location

```
// Start location Updates
private void startLocationUpdates() {
    Log.i(TAG, "startLocationUpdates()");
    mLocationRequest = LocationRequest.create()
            .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
            .setSmallestDisplacement(10)
            .setInterval(3500);
    //.setFastestInterval(FASTEST_INTERVAL);

    if (checkPermission()) {
        LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, mLocationRequest, this);
    }
}
```

- After having location data, it will then draw marker showing where is user at on the map.
- Every time device's location is being changed, that onLocationChanges will be fired up to update a newer location of maker.

Collection markers and Item markers

- Those markers are showing which only internet are connected, after you click a collection marker, the map will then show you its corresponding items by markers on the map.

In BaseMapActivity:

```
//==========================================Marker Click Handlers=====================================//

//Listens for any marker click
public class markerClickHandler implements GoogleMap.OnMarkerClickListener, IAPIUser, GoogleMap.OnInfoWindowClickListener {
    @Override
    public boolean onMarkerClick(Marker marker) {

        if(!hasConnection) {
            // Check if a Feature Marker. Allows display of Feature Window contents (Offline)
            if (marker.getTag().equals("information")) {
                chosenMarker = marker;
                return false;
            }
        }
        else {

            // Collection Markers without tags
            if (marker.getTag() == null) {
                //Launch an HTTP request to the API to get all QR points in the selected collection
                //The data is returned via ReceiveItemsInCollection method below
                //The markers are drawn to the map and also saved to the item markers List
                GetQRPointsInCollection();
                chosenMarker = marker;
            }
            if (marker.getTag().equals("QRPoint")) {
                //If marker is a QR Point....
                chosenMarker = marker;
            }
            if (marker.getTag().equals("CurrentLocation"))
            {
                //If marker is a currentLocation Point....
                chosenMarker = marker;
            }
        }

        return false;
    }
```

Each Collection marker have been set a tag with a given label(a text description). In markerClickerHandler, it will check the name of the tag then do works, so in short, this is the method to distinguish which marker that being clicked.

Firstly when user tap a collection marker on the map, markerClickerHandler would then be raised to check what tag it is holding with it, because we have only give a numeric tags(see right code snippets — code to draw all labeled collection marker onto the map) to collection marker, a CustomInfoWindowAdapter would then show up a customised info window.

```
public void ShowLabels(List<BitmapDescriptor> icons)
{
    markerList = new ArrayList<>();
    for (int i = 0; i < collectionList.size(); i++) {

        String collName = collectionList.get(i).getCollectionName();

        double collLat = collectionList.get(i).getLat();
        double collLng = collectionList.get(i).getLng();
        LatLng collLatLng = new LatLng(collLat, collLng);

        BitmapDescriptor b = icons.get(i);

        Marker marker = map.addMarker(new MarkerOptions()
                .position(collLatLng)
                .title(collName)
                .icon(b)
                .infoWindowAnchor(0.5f, 1.1f));
        marker.setTag(i);
        markerList.add(marker);
    }
}
```

After a collection marker being clicked, onInfoWindowClick would then retrieve the numeric tag which representing its position in the collection list to launch a HTTP request for receiving selected collection information.

```
@Override
public void onInfoWindowClick(Marker marker) {
    if (marker.getTag() != "information" || marker.getTag() == null) {

        // Set all Collection Markers invisible
        for (Marker collMarker : markerList) {
            collMarker.setVisible(false);
        }
        lastCollectionClicked = marker.getPosition();
        new HttpManager().GetCollection(this,collectionList.get(collID).getId());
        //new HttpManager().GetItemsInCollection(this, collectionList.get(collID).getId());
    }
    // Remove info window if clicked
    if (marker.getTag() == "information") {
        marker.hideInfoWindow();
    }
    //Check if is QR Point
    if(marker.getTag() == "QRPoint")
    {
        String test = "test";
    }
}
```

After fetching specific collection info back, it is then ready to launch CollectionMap Activity and pass collection info to that activity.

```
@Override
public void ReceiveCollection(Collection collection) {

    Intent intent = new Intent(BaseMapActivity.this, CollectionMap.class);

    intent.putExtra("col",collection);
    startActivityForResult(intent, COLLECTION_VIEW_REQUEST);
}
```

In CollectionMap activity, It will hit another HTTP request to receive all items of queried collection, then draw them onto the map fragment.