Lab 01: Maps on Android using Google Play Services

Prerequisites

You will need a development environment, either a Mac or a Windows PC with the Android SDK and Xamarin.Android tools installed. We will be using the Genymotioni emulator to test the code we are building, so make sure you have successfully built and deployed a test project first. See the Xamarin.Android setup documentation if you need help getting your environment setup:

http://docs.xamarin.com/guides/Android/getting_started/installation/

Please note: Google Maps requires you to acquire a unique API key and have Google Services enabled on your emulator. Please review the Setting Up Google Maps.pdf provided along with this lab. This process may take 15-30 minutes to walk through.

Assets

This lab includes a set of starter projects you will use to work with maps in Android. You will find all of these projects at:

Content/Exercises/MapsAndroid/Lab Resources

Lab Goals

The goal of this lab will be to demonstrate how to use Google maps on Android. We will cover how to show a map, change the map style, plot points on a map, and how to query for nearby locations and addresses.

The code samples are structured to incrementally add functionality as you progress through the chapter. There are five sample projects included in the solution:

- MappingAndroidLab_demo1 – Show initial map, set the map type to change its style.
- MappingAndroidLab_demo2 – Show current location and animate the camera view to a location.
- MappingAndroidLab_demo3 – Add a point annotation, use custom markers, and use custom overlay images.
- MappingAndroidLab_demo4 – Handle clicks on map markers and info windows, draw circles and polygons.
- MappingAndroidLab_demo5 – Lookup a street address, lookup locations near a place.

The code introduced in this chapter is already present in the demo projects, but it's commented out. Remember you can use *cmd + /* hot key to comment and uncomment whole sections of code at a time by selecting the lines of code and then using the hot key.

The tasks for this lab are also marked with a `// TODO:` task identifier and can be found quickly using the Tasks pane in Xamarin Studio or Visual Studio without having to navigate to individual files.

Steps

Open the Starting Solution

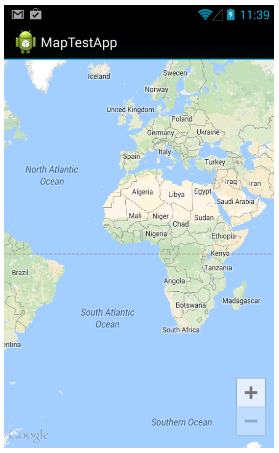
1. Launch Xamarin Studio and open the starting solution – MappingAndroidLab_Begin\MappingAndroidLab.sln.

MappingAndroidLab_demo1 – Add a basic map

1. Set the MappingAndroidLab_demo1 project as the startup project (right click, choose 'Set as Startup Project').
2. Run the application in the Android or Genymotion emulator or on a physical device – you should just see an empty basic screen. Stop the app.
3. The first thing we need to do is to add the Google Play Services component to our project. Your sample projects should already have this listed under the Components folder of your project. If you didn't already have this component, you would simply browse the component store and add the Google Play Services component.
4. The next thing we need to do is to specify our Google Maps API key. If you haven't gotten your key yet, please look at the prerequisites section at the top of this document on how to get one.
5. Open up the GoogleMapsKey.cs file – in here you will find a custom attribute which adds metadata into the <application /> tag in the Android manifest. In this case we are setting the com.google.android.maps.v2.API_KEY, which should be set to our API key. This file is then linked to all the projects so we can set it once here and it will be changed for each exercise we do today.
6. Find the comment `TODO: Step 1a - Enter your API key` … and replace the dummy string value with your API key.
7. Open the Main.axml layout file in the Resources/layout folder. Locate the comment `TODO: Step 1b - Add map fragment` … and uncomment the following lines to add a Map fragment onto our layout:

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

8. You can now run the app. If everything is setup properly you should see a basic map:



9. Next, we will change the map style to show the satellite view. First we want to be able to access the map fragment and Google map from our code behind. Open MainActivity.cs, and take note of the following class member variables:

```
private GoogleMap _map;
private MapFragment _mapFragment;
```

10. In the `OnCreate()` method get a handle on the `MapFragment` and `GoogleMap` objects within our view by locating the comment `TODO: Step 1c - Get a handle on the map element` and uncommenting the following lines:

```
_mapFragment = FragmentManager.FindFragmentById(Resource.Id.map) as Ma
pFragment;
_map = _mapFragment.Map;
```

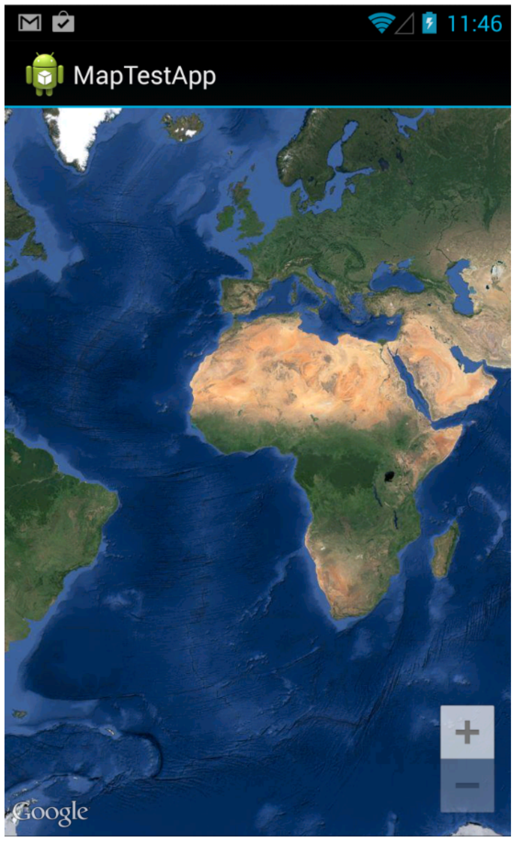
11. Next we can set the map type we want to see by setting MapType on our _map object. Locate the comment `TODO: Step 1d - Set the map type` and uncomment the following line:

```
_map.MapType = GoogleMap.MapTypeSatellite;
```

12. We can also disable different interactions that the map has turned on by default. Locate the comment `TODO: Step 1ef - disable interactions` and uncomment the following lines:

```
_map.UiSettings.RotateGesturesEnabled = false;
_map.UiSettings.TiltGesturesEnabled = false;
_map.UiSettings.ScrollGesturesEnabled = false;
_map.UiSettings.ZoomControlsEnabled = false;
_map.UiSettings.ZoomGesturesEnabled = false;
```

13. Run the application again – you should now see a satellite map view that can't be interacted with:



MappingAndroidLab_demo2 – Show current location

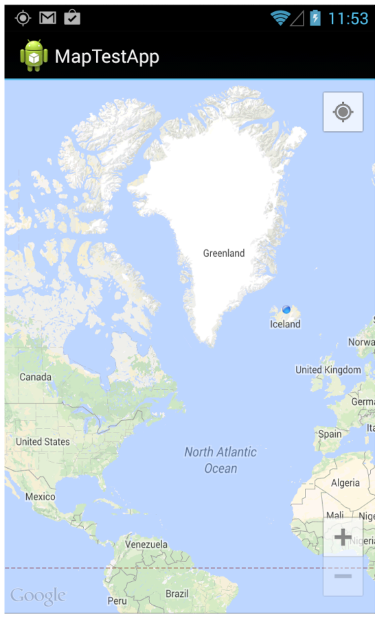
1. Set the MappingAndroidLab_demo2 project as the startup project. You can run the project if you'd like – you should just see a basic map.

2. The first thing we want to see is our own location on the map. Locate the comment `TODO: Step 2a - show user location` in MainActivity.cs and uncomment the following lines to see your location on the map:

```
_map.UiSettings.MyLocationButtonEnabled = true;
_map.MyLocationEnabled = true;
```

   The first line above adds a button in the upper right of the map view that will focus on your location, and the second line tells the map to turn on showing our location.

3. Run the app, you should see your location now on the map. If you are using a Genymotion emulator, make sure you click on the GPS icon and turn the GPS of the device on. You can click the location button in the upper right to focus on your location (you can see the screenshot was defaulted to Iceland – but you can set the location in the Genymotion GPS settings):



The next thing we want to do is to listen for location change events. In our example we will listen for them so that we can adjust the view of the map to focus on the user's location.

4. Find the following line in `MainActivity.OnCreate` and uncomment it – this will give us a handle on the Location Manager system service. Locate the comment `TODO: Step 2b - setup a location manager` and uncomment the following line:

```
_locationManager = GetSystemService(Context.LocationService) as Locati
onManager;
```

5. We will now update our Activity to be able to handle location change events. We do this by adding `ILocationListener` as an interface we implement and adding the following methods. Locate the comment `TODO: Step 2c - implement ILocationListener` and uncomment the following methods:

```
public void OnLocationChanged(Location location)
{
    _map.AnimateCamera(CameraUpdateFactory.NewLatLngZoom(new Android.G
ms.Maps.Model.LatLng(location.Latitude, location.Longitude), 5.0f));
}

public void OnProviderDisabled(string provider)
{
    // throw new NotImplementedException();
}

public void OnProviderEnabled(string provider)
{
    // throw new NotImplementedException();
}

public void OnStatusChanged(string provider, Availability status, Bund
le extras)
{
    // throw new NotImplementedException();
}
```

Just overriding these methods won't do anything yet, but you can see that we will be handling the `OnLocationChanged()` method to animate the map's camera to the location that the user is at once we start to get location change events.

In order for our location updates to start firing, we need to start requesting location updates and tell our implementation of `ILocationListener` to get notified.

6. In the `OnResume()` method, locate the comment `TODO: Step 2d - use a GPS provider` and uncomment the following lines to get notified when the user's location changes:

```
string Provider = LocationManager.GpsProvider;
if(_locationManager.IsProviderEnabled(Provider))
{
    _locationManager.RequestLocationUpdates (Provider, 2000, 1, this);
}
else
{
    Log.Info("error", Provider + " is not available. Does the device h
ave location services enabled?");
}
```

7. Run the app. Notice that the camera now animates to your location. If you are using Genymotion you can change your GPS position and watch the map refocus on your location again. If you are using an Android emulator you can find instructions on how to send positions to the emulator using DDMS here: http://stackoverflow.com/questions/2279647/how-to-emulate-gps-location-in-the-android-emulator

8. Now let's remove the code we just added to `OnResume` above and instead add in the following code. Locate the comment `TODO: Step 2e - use a generic location provider instead` and uncomment the following code:

```
Criteria locationCriteria = new Criteria();
locationCriteria.Accuracy = Accuracy.Coarse;
locationCriteria.PowerRequirement = Power.Medium;

var locationProvider = _locationManager.GetBestProvider(locationCriter
ia, true);
if (locationProvider != null)
{
    _locationManager.RequestLocationUpdates(locationProvider, 2000, 1,
 this);
} else
{
    Log.Info("error", "Best provider is not available. Does the device
 have location services enabled?");
}
```

What we've done now is to use `GetBestProvider()` instead of explicitly using `GpsProvider`. This means that we can still get location information even if it isn't from GPS specifically. For instance, if the user has GPS on their phone off (or their phone doesn't support GPS), the phone may still know their location through their cellular connection or data known about the WIFI network they are attached to.

9. Run the app one last time and see that your location is now reported even if GPS isn't available.

MappingAndroidLab_demo3 – Working with point markers and overlay images

1. Set the MappingAndroidLab_demo3 project as the startup project. You can run the project if you'd like – it should be just as left off after demo2.
2. First lets add a few locations we can use later when placing our points. Locate the comment `TODO: Step 3a - add lat/long position constants to use` in MainActivity.cs and uncomment the following lines:

```
private static readonly LatLng Location_NewYork = new LatLng(40.714353
, -74.005973);
private static readonly LatLng Location_Chicago = new LatLng(41.878114
, -87.629798);
private static readonly LatLng Location_Xamarin = new LatLng(37.797679
, -122.401816);
private static readonly LatLng Location_Atlanta = new LatLng(33.755, -
84.39);
```

3. In our `OnCreate()` method we can now add our first two markers. Locate the comment `TODO: Step 3b - Add points on the map` and uncomment the following lines:

```
MarkerOptions marker1 = new MarkerOptions();
marker1.SetPosition(Location_Xamarin);
marker1.SetTitle("Xamarin");
marker1.InvokeIcon(BitmapDescriptorFactory.DefaultMarker(BitmapDescrip
torFactory.HueBlue));
_map.AddMarker(marker1);

MarkerOptions marker2 = new MarkerOptions();
marker2.SetPosition(Location_Atlanta);
marker2.SetTitle("Atlanta, GA");
marker2.InvokeIcon(BitmapDescriptorFactory.DefaultMarker(BitmapDescrip
torFactory.HueRed));
_map.AddMarker(marker2);
```

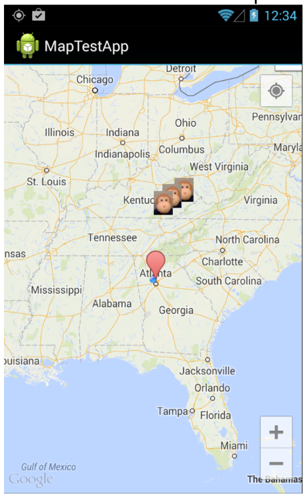
4. Run the app and you will see our two markers, one in San Francisco, CA and the other in Atlanta, GA. If you tap one you will see the title that we specified when creating them.

5. Next we will uncomment the `AddMonkeyMarkersToMap()` call in `OnCreate()` and the method itself. Locate the two comments `TODO: Step 3c - Add custom marker images on the map` and uncomment them both:

```
AddMonkeyMarkersToMap();
```

```
private void AddMonkeyMarkersToMap()
{
    LatLng[] locationForCustomIconMarkers = new[]
    {
        new LatLng(40.741773, -74.004986),
        new LatLng(41.051696, -73.545667),
        new LatLng(41.311197, -72.902646)
    };

    for (int i = 0; i < locationForCustomIconMarkers.Length; i++)
    {
        BitmapDescriptor icon = BitmapDescriptorFactory.FromResource(R
esource.Drawable.monkey);
        MarkerOptions markerOptions = new MarkerOptions()
            .SetPosition(locationForCustomIconMarkers[i])
            .InvokeIcon(icon)
            .SetSnippet(String.Format("This is marker #{0}.", i))
            .SetTitle(String.Format("Marker {0}", i));
        _map.AddMarker(markerOptions);
    }
}
```

6. Run the app and you will see that we have drawn Monkey marker images in 3 locations on the map.



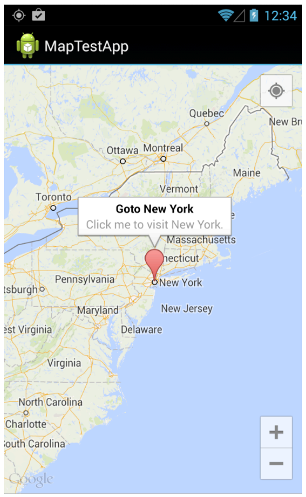
As you can see in the code above, we still use `_map.AddMarker()` and are just providing a different set of `MarkerOptions` that specify an icon image to use. You'll also notice a sub-title text when you tap these new markers, this is set above in the `SetSnippet()` call.

7. Next we will see how to add an informational callout. Uncomment the call to `AddInitialNewYorkBarToMap()` and it's corresponding implementation. Locate the two comments `TODO: Step 3d - Add custom arrow callout on map` and uncomment the following code:

```
AddInitialNewYorkBarToMap();
```

```
private void AddInitialNewYorkBarToMap()
{
    MarkerOptions markerOptions = new MarkerOptions()
        .SetSnippet("Click me to visit New York.")
        .SetPosition(Location_NewYork)
        .SetTitle("Goto New York");
    _newYorkMarker = _map.AddMarker(markerOptions);
    _newYorkMarker.ShowInfoWindow();
}
```

8. Run the app and you'll see the new marker is added, and then told to show with `ShowInfoWindow()` so it is immediately visible:



9. Finally, let's look at how to add an image overlay onto the map. Find the two comments `TODO: Step 3e - Add custom overlay image on the map` and uncomment the following code:

```
PositionChicagoGroundOverlay(Location_Chicago);
```

```
private void PositionChicagoGroundOverlay(LatLng position)
{
    if (_chicagoOverlay == null)
    {
        BitmapDescriptor image = BitmapDescriptorFactory.FromResource(
Resource.Drawable.monkey);
        GroundOverlayOptions groundOverlayOptions = new GroundOverlayO
ptions()
            .Position(position, 150000, 200000)
            .InvokeImage(image);
        _chicagoOverlay = _map.AddGroundOverlay(groundOverlayOptions);
    }
    else
    {
        _chicagoOverlay.Position = Location_Chicago;
    }
}
```

10. Run the app again and you will see how we loaded the monkey image and display it at a specific size on the map as an image overlay.

MappingAndroidLab_demo4 – Handling events and drawing shapes

1. Set the MappingAndroidLab_demo4 project as the startup.
2. Get a handle on the marker ID of the New York marker we added earlier. Locate the comment `TODO: Step 4a - keep track of marker id` and uncomment the following line:

```
_gotoNewYorkMarkerID = _newYorkMarker.Id;
```

3. Next we want to define the method to get called whenever a marker is clicked on the map. Locate the comment `TODO: Step 4b - handle map marker taps` and uncomment the following code:

```
private void MapOnMarkerClick(object sender, GoogleMap.MarkerClickEven
tArgs markerClickEventArgs)
{
    markerClickEventArgs.Handled = true;

    Marker marker = markerClickEventArgs.P0;
    if (marker.Id.Equals(_gotoNewYorkMarkerID))
    {
        _map.AnimateCamera(CameraUpdateFactory.NewLatLngZoom(Location_
NewYork, 13));
        _gotoNewYorkMarkerID = null;
        _newYorkMarker.Remove();
        _newYorkMarker = null;
    }
    else
    {
       Toast.MakeText(this, String.Format("You clicked on Marker ID {0
}", marker.Id), ToastLength.Short).Show();
    }
}
```

4. You can see that we check the marker's ID to see if it is our New York marker, and if so then we animate in to zoom on it, otherwise we just show a toast message about the event firing.

5. Next we will define the method that gets called when the informational popup window is tapped. Locate the comment `TODO: Step 4c - handle info popup tap` and uncomment the following lines:

```
private void HandleInfoWindowClick(object sender, GoogleMap.InfoWindow
ClickEventArgs e)
{
}
```

6. Now we need to wire up our events. We want to wire up our map listeners in `OnResume()`, and detach them in `OnPause()` to ensure our map listeners are only firing when the app has focus.

7. Locate the comment `TODO: Step 4d - attach map handlers on resume` and add the following in the `OnResume()` method:

```
_map.MarkerClick += MapOnMarkerClick;
_map.InfoWindowClick += HandleInfoWindowClick;
```

8. Next lets see how we can draw a circle and a polygon whenever the info window is tapped. Go back to our `HandleInfoWindowClick()` event handler, locate the comment `TODO: Step 4e - draw a circle on the map`, and add the following code (Step 4e):

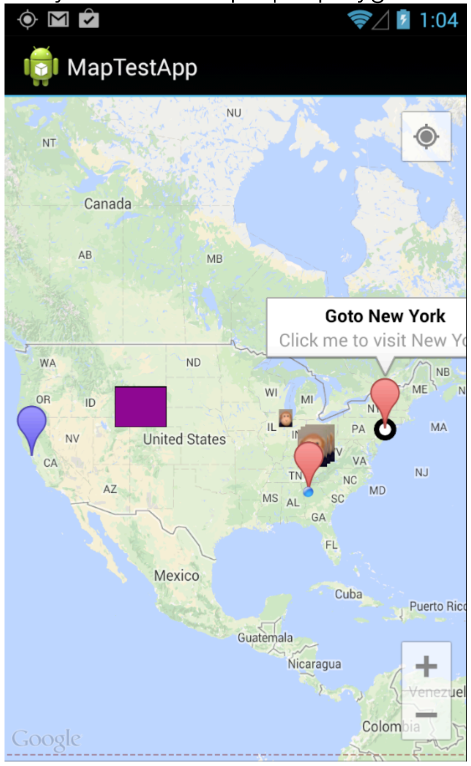
```
    // TODO: Step 4e: draw a circle on the map
    CircleOptions circleOptions = new CircleOptions();
    circleOptions.InvokeCenter(Location_NewYork);
    circleOptions.InvokeRadius(100000.0);
    circleOptions.InvokeFillColor(Android.Graphics.Color.White);
    _map.AddCircle(circleOptions);

    // TODO: Step 4f: draw a polygon (Wyoming) on the map
    PolygonOptions polygonOptions = new PolygonOptions();
    polygonOptions.Add(new LatLng[]
    {
        new LatLng(45.00, -111.00),
        new LatLng(45, -104),
        new LatLng(41, -104),
        new LatLng(41, -111)
    });
    polygonOptions.InvokeFillColor(Android.Graphics.Color.Purple);
    polygonOptions.InvokeStrokeWidth(2);
    _map.AddPolygon(polygonOptions);
```

9. Locate the comment `TODO: Step 4g - detach map handlers on pause` and add the following lines of code in the `OnPause()` method:

```
_map.MarkerClick -= MapOnMarkerClick;
_map.InfoWindowClick -= HandleInfoWindowClick;
```

10. Run the app and note all of the new interactions. If you tap on the red marker for New York, the map will zoom in to focus on it. If you tap on the “Goto New York” info window you will see a large white circle get drawn, and if you zoom

out you will see a purple polygon drawn over Wyoming.



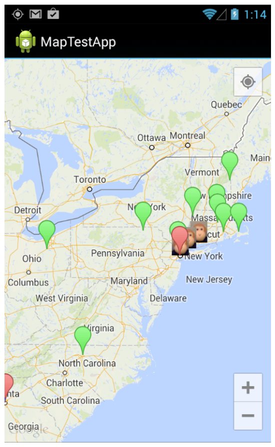
MappingAndroidLab_demo5 – Looking up addresses and nearby locations

1. Set the MappingAndroidLab_demo5 project as the startup project.
2. We have removed the circle and polygon being drawn when the "Goto New York" info window is tapped, and will now instead do an address query when it is tapped. Locate the two comments `TODO: Step 5a – Query for a location and map them` and uncomment the following code:

```
// handle info popup tap
private void HandleInfoWindowClick(object sender, GoogleMap.InfoWindow
ClickEventArgs e)
{
    // TODO: Step 5a - Query for a location and map them
    RunAddressQuery();

   // TODO: Step 5b - Query for point of interest at a given location
   // FindNearestPlace();
}
// TODO: Step 5a - Query for a location and map them
private void RunAddressQuery()
{
    Task.Run(async () =>
    {
        var geoCoder = new Geocoder(this);
        var addresses = await geoCoder.GetFromLocationNameAsync("Burba
nk Street", 50);

        RunOnUiThread(() =>
        {
            if (addresses.Count > 0)
            {
                foreach (var address in addresses)
                {
                    MarkerOptions marker1 = new MarkerOptions();
                    marker1.SetPosition(new LatLng(address.Latitude, a
ddress.Longitude));
                    marker1.SetTitle("Hollywood");
                    marker1.InvokeIcon(BitmapDescriptorFactory.Default
Marker(BitmapDescriptorFactory.HueGreen));
                    _map.AddMarker(marker1);
                }
            }
        });
    });
}
```

As you can see, we are searching for "Burbank Street" and iterating our results and adding markers. Notice that we are careful to make our queries asynchronously so as not to lock up the UI in the event handler

3. Run the application, tap on the New York info window, and see the results get shown as green markers on the map:

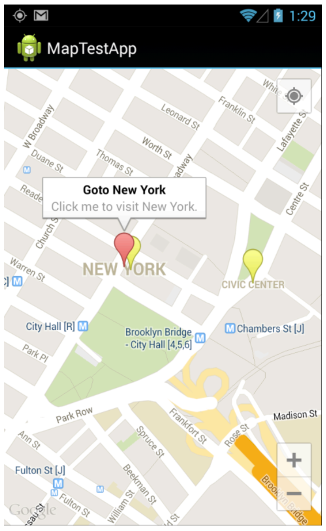


4. Next, lets do a search in a very specific location. Comment out the `RunAddressQuery()` call above, and uncomment the call to `FindNearestPlace()` and its corresponding method. Locate the comments `TODO: Step 5b - Query for point of interest at a given location` and uncomment the code:

```
FindNearestPlace();
```

```
// TODO: Step 5b - Query for point of interest at a given location
private void FindNearestPlace()
{
    Task.Run(async () =>
    {
        var geoCoder = new Geocoder(this);
        var locations = await geoCoder.GetFromLocationAsync(Location_N
ewYork.Latitude, Location_NewYork.Longitude, 2);
        RunOnUiThread(() =>
        {
            if (locations.Count > 0)
            {
                foreach (var location in locations)
                {
                    MarkerOptions marker1 = new MarkerOptions();
                    marker1.SetPosition(new LatLng(location.Latitude,
location.Longitude));
                    marker1.SetTitle(location.FeatureName);
                    marker1.InvokeIcon(BitmapDescriptorFactory.Default
Marker(BitmapDescriptorFactory.HueYellow));
                    _map.AddMarker(marker1);
                }
            }
        });
    });
}
```

5. If you run the app now you will see two yellow markers directly on top of our main New York marker. These markers are the closest points of interest found when querying for a location from a given latitude and longitude:



6. Remove our custom handler for when marker's are tapped so that their default info window pops up instead). Locate the comment `TODO: Step 5c – remove handler so normal popup occurs` and comment out the following line:

```
// _map.MarkerClick += MapOnMarkerClick;
```

7. Run the app one last time to see again how we plot various items based on Google Map location queries.

Summary

In this lab, we showed how to use Google Maps to show a map in your app and do all of the common things you might want to with a map control including graphing locations, and drawing regions.