How to Use this Template

1. Make a copy [File → Make a copy...]
2. Rename this file: "**Capstone_Stage1**"
3. Replace the text in green

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"

GitHub Username: arjun.chouhan68@gmail.com

Mark Off

Description

Mark Off is a geometrical app that uses google maps to calculate area and perimeter of any polygon you make on the map, either manually or by using GPS. It also calculates distance between several points on the map and using accurate Lat-Lng translation provides computations in different units.

Plotting markers on the map is as easy as tapping on the map and the polygon gets constructed as you add subsequent markers to the map. It also uses GPS for constructing the polygon in case you don't want to manually plot the markers.

Calculation is not limited to any specific area or distance as the polygon can be drawn at any surface and can by dynamically resized.

The subsequent updates to the app will include more features which will make the plotting and computations even easier.

Intended User

This app is for all age groups as well as all class of people who want to know the area of a patch or anything on the map or distance between two sources. Since the calculations are based on highly precise trigonometry, the app can also be used by civil engineers to know the exact area of a location or site.

Features

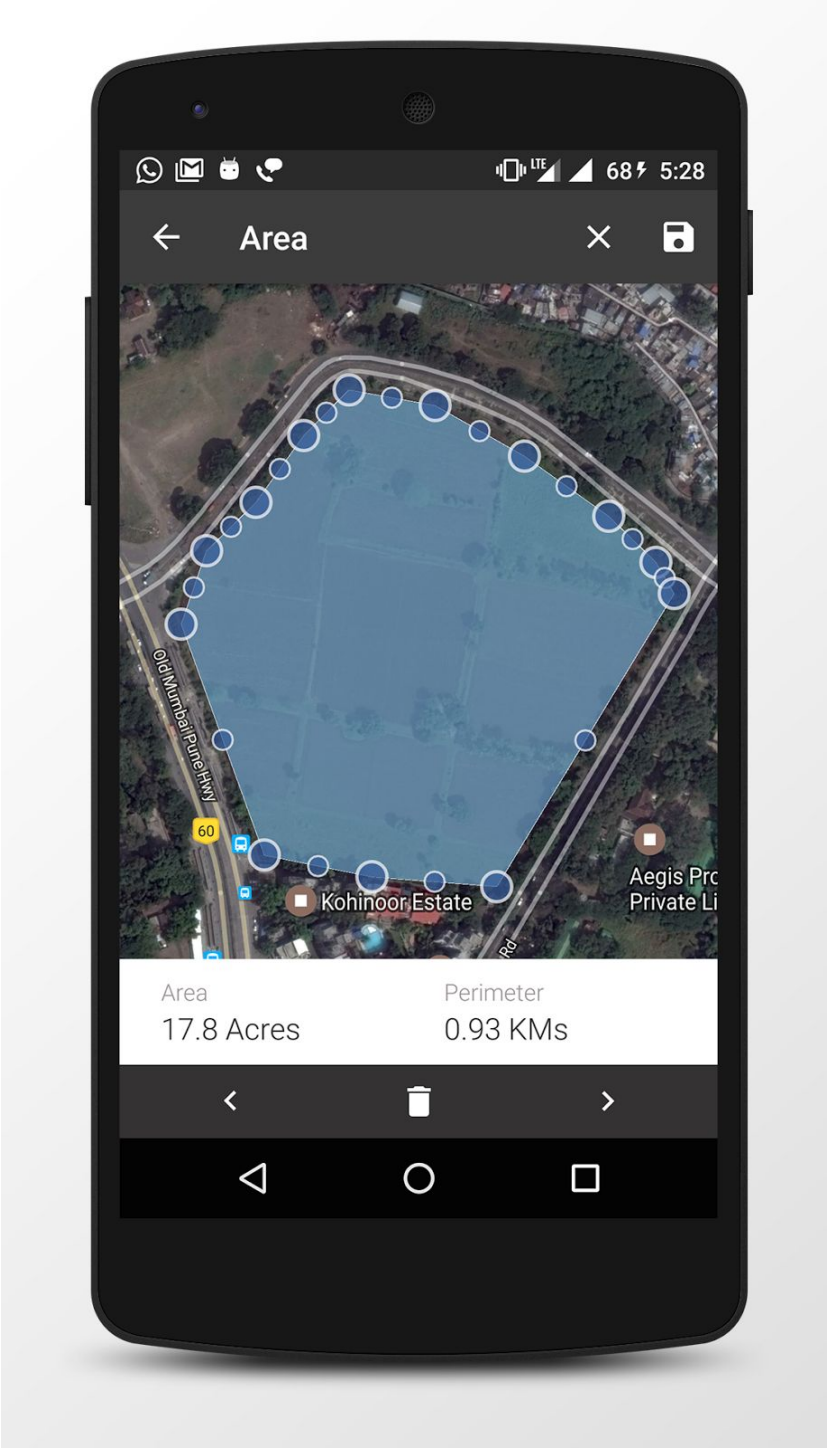
Main Features Of The App Are:-

1. Manual Area Calculation - Calculate area by manually plotting markers on the map.
2. GPS Area Calculation - Calculate area using GPS co-ordinates. For eg. Walking or driving a car around an area.
3. Dynamically Resize The Polygon - Select the existing marker to be re-positioned on the map and simply drag it with your finger.
4. Middle Markers - There are mid markers between any two added markers which can be dragged to add a main marker and two mid makers between the newly added marker and its adjacent two markers. This ensures fine tuning of the polygon.
5. Distance Calculation - In the same way the area is calculated (both manually and with GPS) calculate distance manually or with GPS.
6. Delete The Added Markers - With the delete button, delete the selected marker.. The polygon adjusts itself after the deletion of the marker.
7. Move Between Markers - Using the given left and right controls on the bottom sheet you can select a marker and move between (select) other markers so no need to select individual marker by tapping.
8. Change Fill And Stroke Colors - Customise the fill and stroke colors of manual as well as GPS polygons with your desired color combinations.
9. Change Area and Perimeter Units - Choose between different units to show the calculated area and perimeter or distance.
10. Save Your Polygons - Save your manual or GPS polygons for later analysis.

User Interface Mocks

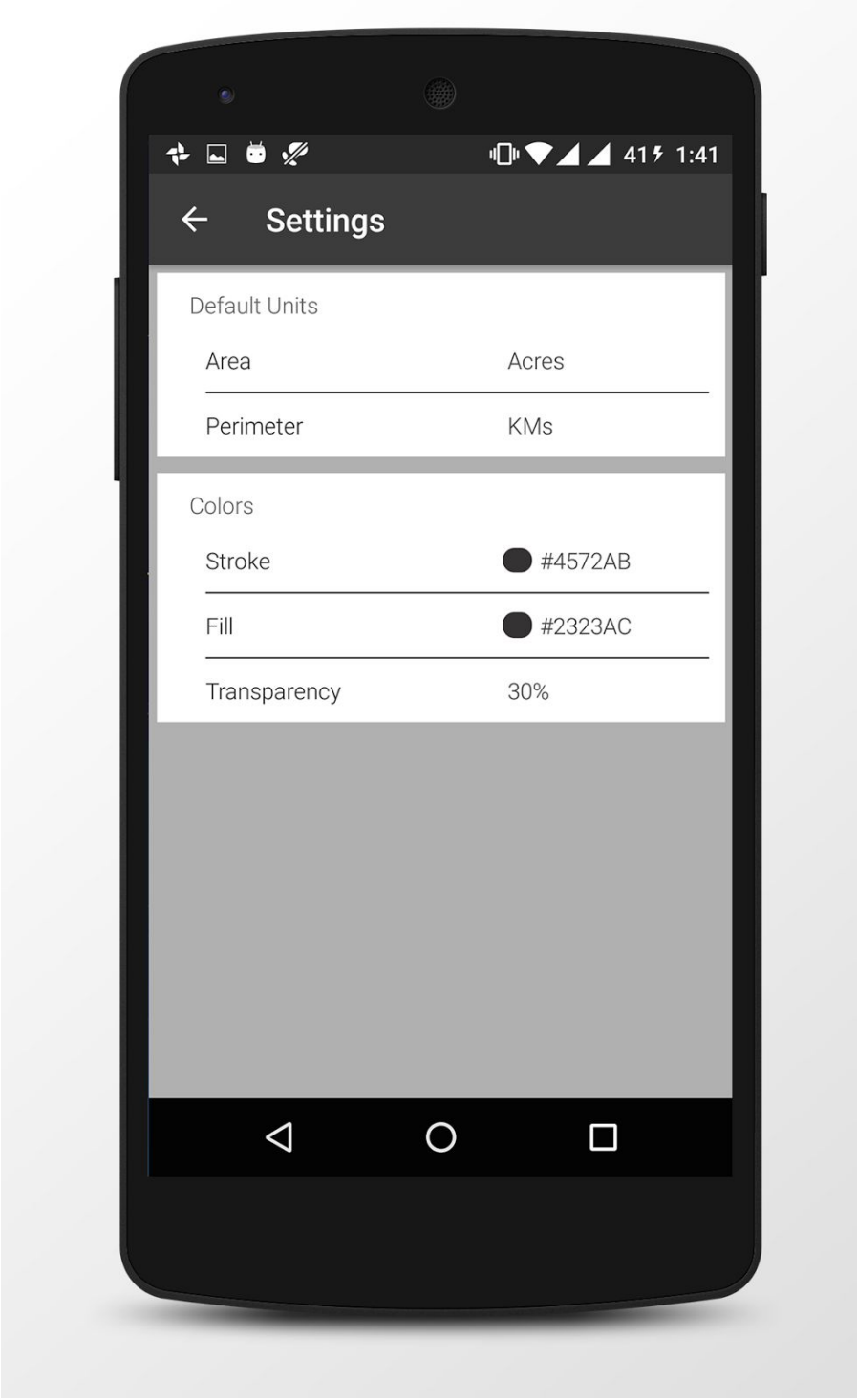
These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Photoshop or Balsamiq.

Screen 1



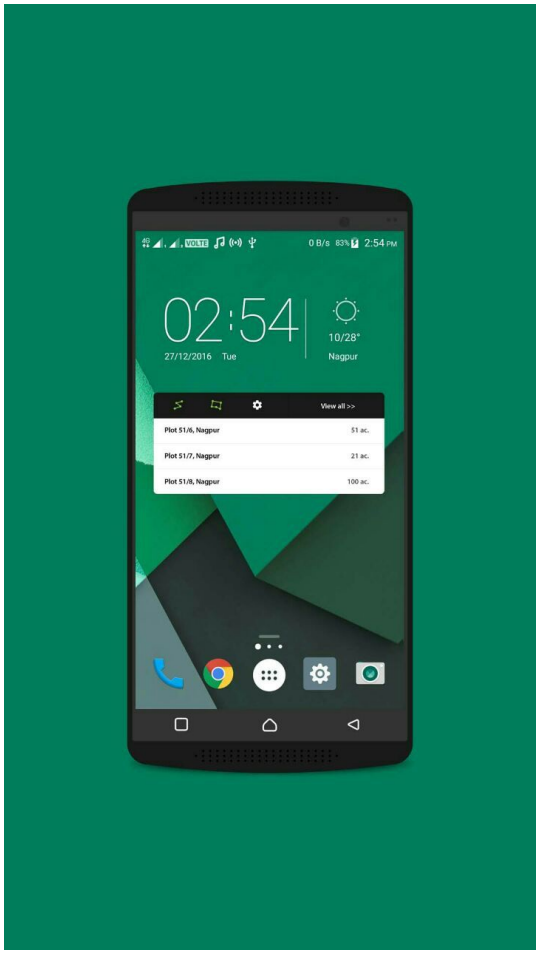
The above mock up shows a manually drawn polygon with its area and perimeter. The process of plotting simply starts by tapping on the map's desired location and adding more markers by similar process.

Screen 2



Customize the computation units and fill, stroke colors according to your requirement and combinations.

Screen 3



View all your computed measurements through the widget and access them from the same as well.

Key Considerations

How will your app handle data persistence?

The setting preferences will be stored in a SharedPreference and will be loaded when the app starts and will be updated if changed from the settings Activity. Polygon Lat-Lngs will also be saved in a SharedPreference instance which can be loaded later for further analysis.

Describe any corner cases in the UX.

1. Since the app uses support map fragment and an activity to hold it, any back press event that occurs will be received by activity and not the fragment. The app takes care of this

corner case by sending appropriate broadcasts from fragment to the activity and vice-versa to know the occurrence of a back press event and later by showing a material dialog. This ensures proper flow of UX.

Describe any libraries you'll be using and share your reasoning for including them.

UI Libraries like Material Dialogs and Color Picker have been used for showing interactive dialogs to user and for selection of their own custom colors for fill and stroke.

Describe how you will implement Google Play Services.

Google Maps have been implemented for showing the maps and plotting the markers and get the appropriate Lat-Lng for computation of different geometrical factors. It has been implemented using SupportMapFragment and Appropriate API key has been used. Google Ads have been used for showing banner and interstitial ads for monetisation. SphericalUtils has been used for computation of different geometrical factors on map which uses Lat-Lng for computation.

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and decompose them into tangible technical tasks that you can complete incrementally until you have a finished app.

Task 1: Project Setup

1. Setting up the usual android project with minimum API of 15 and a blank activity.
2. Setting up Google Maps and Google Ads in gradle with appropriate versions.
3. Setting up other UI, UX libraries as required for rich and responsive UI and UX.
4. Include appropriate permissions in the manifest.

Task 2: Implement UI for Each Activity and Fragment

List the subtasks. For example:

1. Build General UI components for the Main Map-Holder Activity.
2. Include support map fragment and add API Key in manifest.
3. Build UI for Settings Activity.
4. Build UI controls for showing computations.
5. Build UI components for alterations to the manual polygon.

Task 3: Implement Addition Of Poly Layer On Google Map

Implement onMapClickListener for addition of markers on map by tapping on the map and later adding the same marker's Lat-Lng to a PolygonOptions and a List to maintain the polygon.

1. Implement onMapClickListener and add a marker on map click.
2. Store this marker's Lat-Lng in a global List for maintaining the polygon's vertices.
3. Update the Polygon List.
4. Polygon can also be a Polyline for distance calculation.

Task 4: Implement Computation Of Geometrical Factors Using AsyncTask

Update the computational factors such as area, perimeter or distance when a marker is added or moved on a map.

1. Take the Lat-Lngs from the List and using the SphericalUtils compute factors in appropriate preciseness.
2. Since there can be many LatLngs to search and interpolate from we schedule this work to an AsyncTask.

Task 5: Implement LocationManager For GPS Computations

Implement LocationManager to get GPS coordinates at a fixed interval which will be added to a central poly list for updating the path of the poly and measurements.

1. Take the coordinates from location manager and update the poly list and in turn measurements.

Task 6: Implement a FrameLayout and the draggable marker

Since the google map has no APIs that directly expose a movable marker without long press or so and the support for addition of custom layers to the map has been removed in Google Maps V3 the workaround for this would be to add a transparent layout to intercept touch event on specific conditions, i.e. the FrameLayout.

1. Add an ImageView to the frame layout with the src as an inverted marker image and add it as the FrameLayout's child.
2. Implement onTouchListener on the FrameLayout and check for bounding box on ACTION_DOWN.

3. If the bounding box of the ImageView contains the touch coordinated then dispatch the touch event to ImageView's onTouchListener so that the touch event will be handled by the ImageView now..
4. Otherwise return false so that the touch is passed to map.
5. When the touch is handled by the imageview the ACTION_MOVE of the same modifies the list maintained of that specific marker and the actual marker is set to invisible.
6. On ACTION_UP of the ImageView the current ImageView is hidden and the invisible marker's location is updated to the final Lat-Lng of the ImageView translated from the screen coordinates using projection.

Task 7: Implement Middle Markers

There has to be a mid marker in between two normal markers which can be dragged on choice to add more normal and mid markers for fine tuning with the area of the polygon.

1. Whenever a Normal marker is added to the screen compute its adjacent markers and interpolate the middle position and insert the mid marker with a slightly smaller image than that of normal marker to differentiate.
2. When a middle marker is moved then make the current middle marker as a normal marker and interpolate middle positions of its adjacent markers and insert two middle markers, one at each interpolated position.

Task 8: Implement alteration controls for modifications

These include the following controls and have been shown in the above mockups:-

1. Delete Marker
2. Select Left Marker
3. Select Right Marker
4. Save the poly
5. Discard

Task 9: Implement ContentProvider

Implement ContentProvider to store the user chosen fill and stroke colors with the prefered geometrical factor units chosen by user. Also to store the Lat-Lngs of the saved poly for later analysis.

Loaders will be used to load the data from ContentProvider into the appropriate views.

Task 10: Implemetation Of Widget

Using RemoteView we show a ListView which has all the saved computations and fetches the same from the ContentProvider.

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"