AugGraffitiDev

Author

By Jianan Yang and Wenhao Chen

Goal

AugGraffiti is a mobile application on the Android for users to create their own Graffiti on there mobile device.

Install

Download AugGraffitiDev\MyApplication\app\app-release.apk to your phone and make sure you allow permission to use your location.

Developement

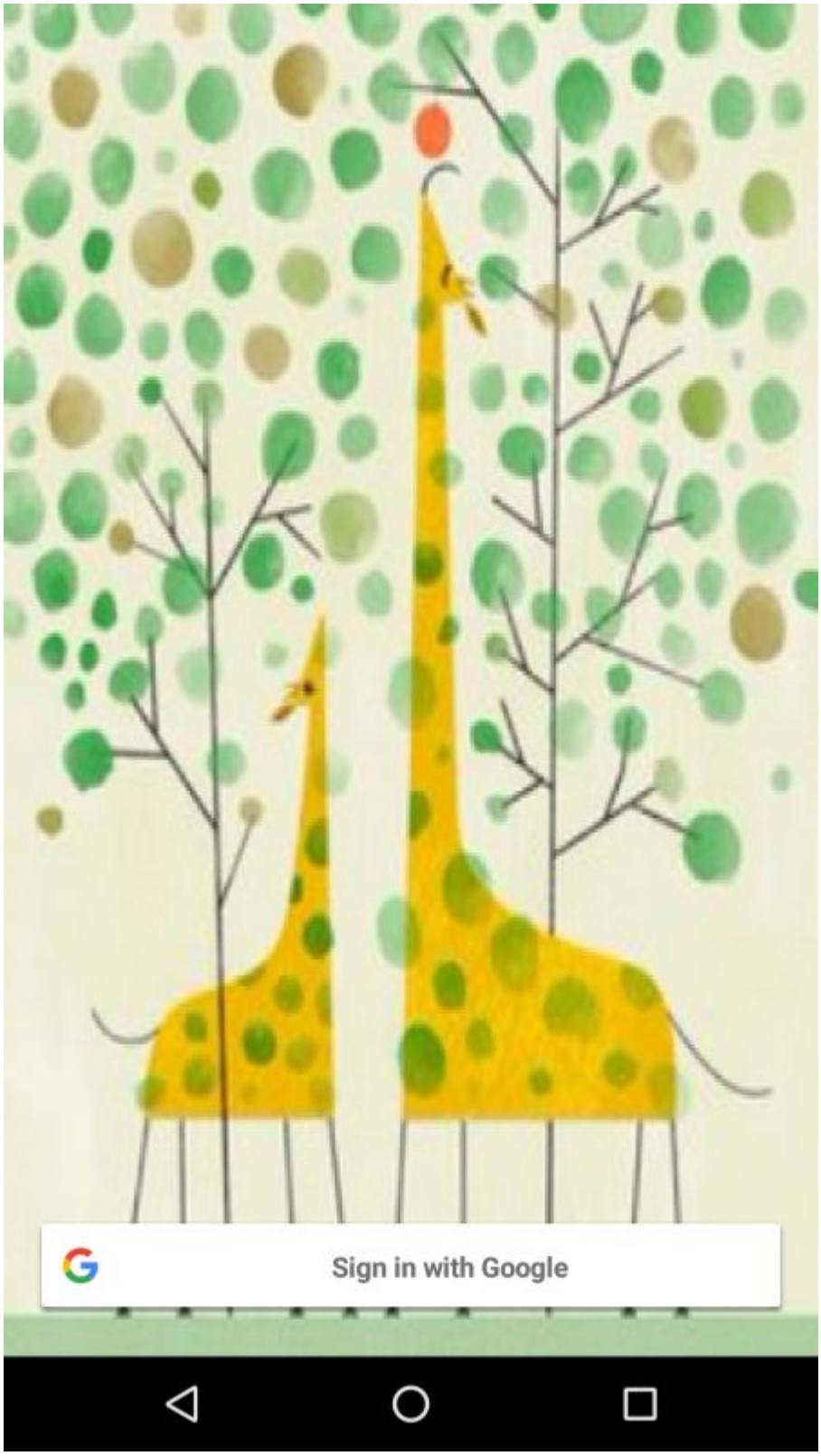
To develope, you need to confingure the build.gradle (app) firstly. You need to change directory `storeFile file('D:/asu/asu/EEE598/AugGraffitiDev/keystore.jks')` to your local directory

```
debug {
            keyAlias 'AugGraffiti'
            keyPassword '940205'
            storeFile file('D:/asu/asu/EEE598/AugGraffitiDev/keystore.jks')
            storePassword '940205'
        }
```

Design

A complete AugGraffitiDev consists 5 different screens, i.e. Login, Map, Place, Collect and Gallery. As of now, users can only interact with Login and Map screen. The other three screens will be developed and added to this app in next step.

Step 1: To open the app, click the icon named "MyApplication". Once it is opened succeffully, the Login screen pops up.



Login screen

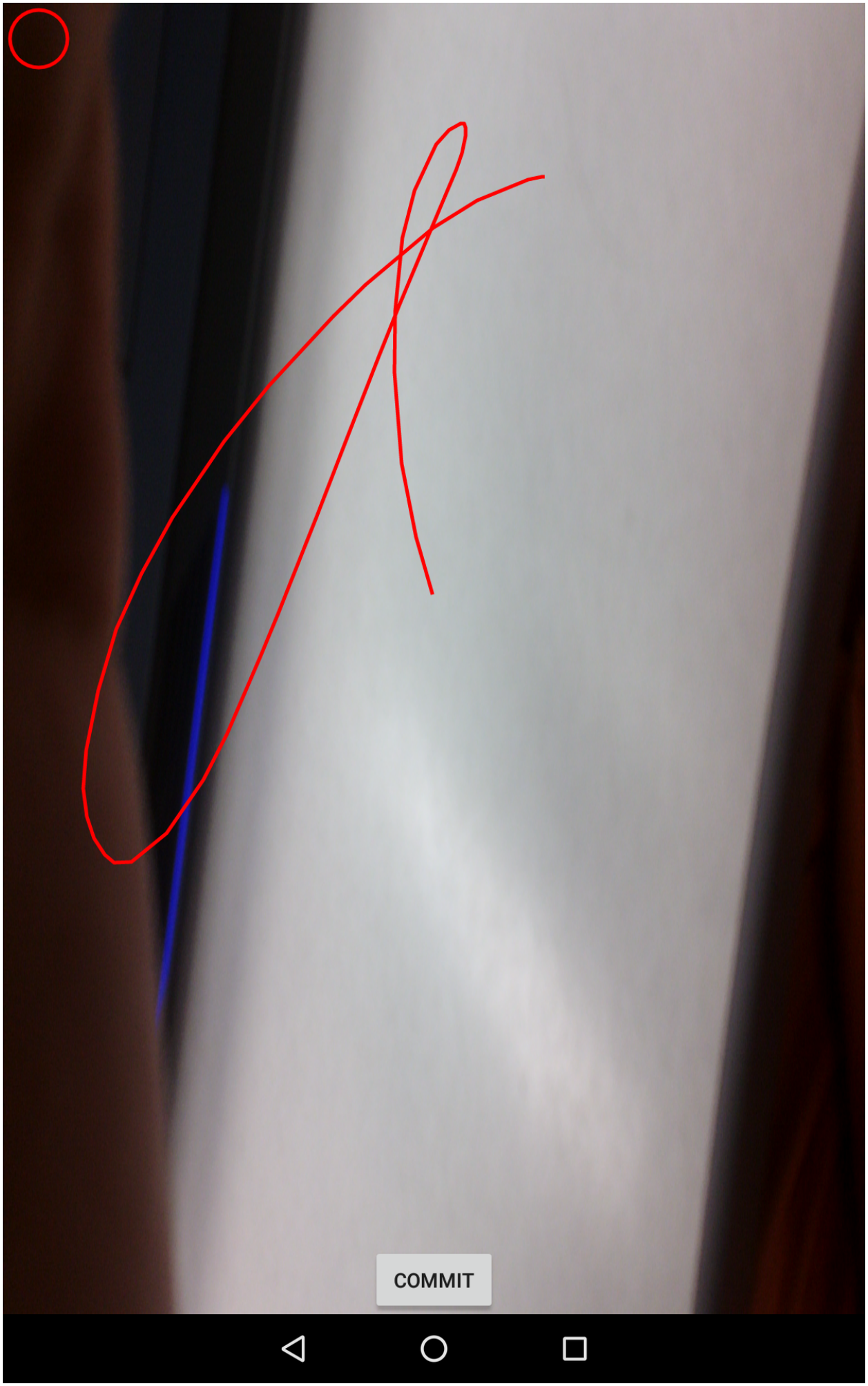
Step 2: Use your "Google Account" to login by clikcing "Sigin in with Google" button. If the login succeeds, you will be directed to Map screen. In this screen, a "P" tag located in the center represnets your current location. This "P" button also allows you to place a tag on current location (this function is currently unavailable and will be implemented in next step). By tapping the "SIGN OUT" button on top of the screen, you will be directed back to Login screen.

Map screen

Step 3: Walking around ASU campus and find tags! Tags within 50m of your current location will appear on your Map screen. This is shown in picture below in which the "C" are tags placed by other Users. Further steps regarding how to place tags and collect tags will be added here in next step.

Map screen with tags

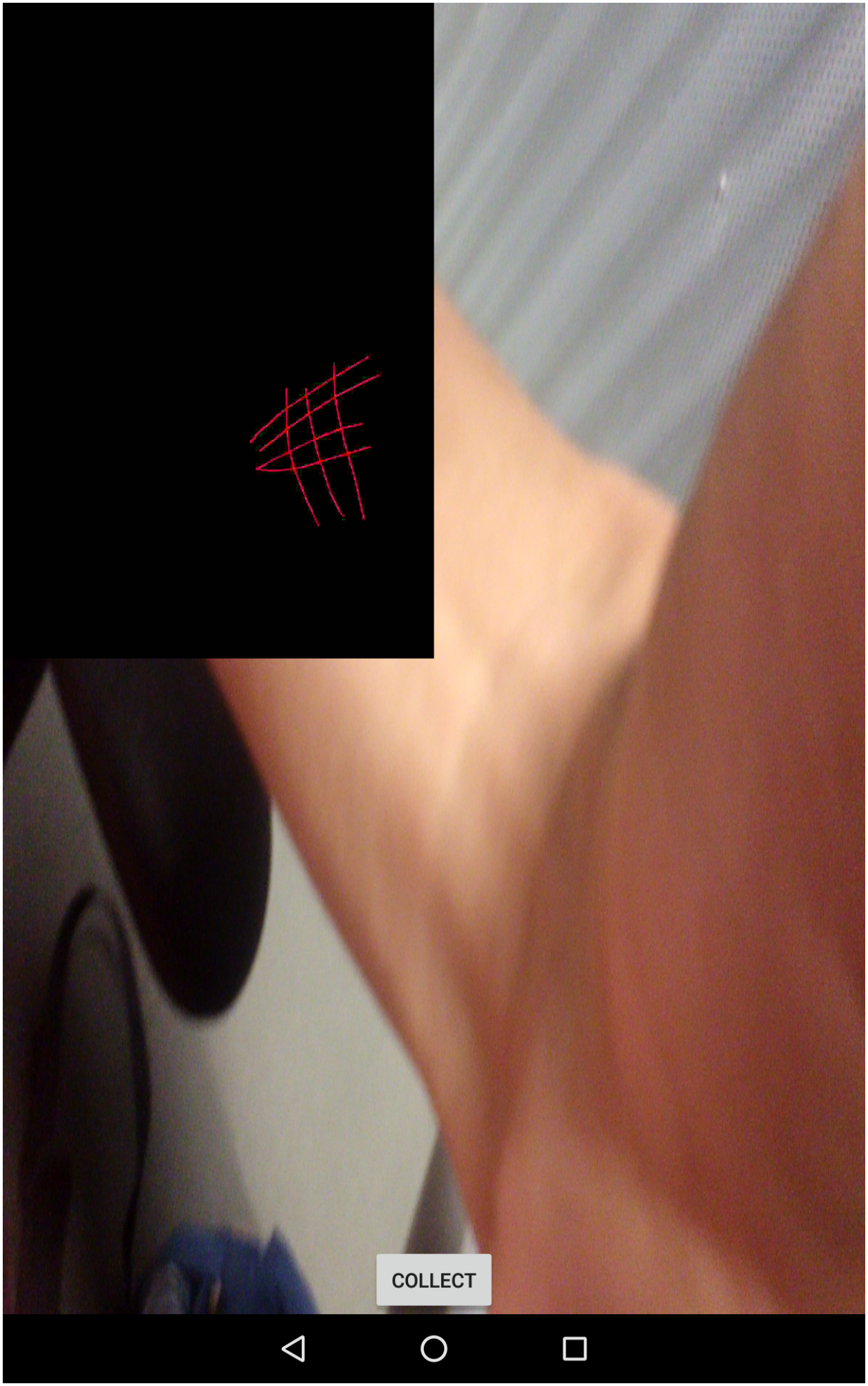
Step 4: This screen is invoked when clicking the "P" marker in the google map screen. Once the screen is opened, the camera is opened, and you can draw anything you like by simply tapping and dragging on screen. After drawing, you can send your draw (with the camera background) to sever by clicking the "COMMIT" button located at the bottom.



Placing

tag screen

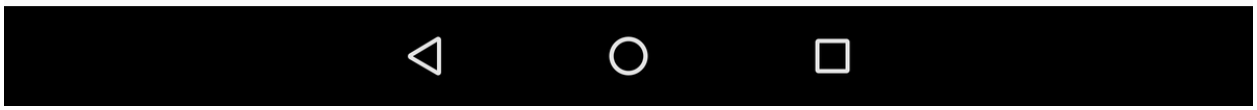
Step 5: This Collecting Tag screen is invoked when you tap the "C" tag on google map screen. In this screen, the up-right shows the tag image drawn by somebody else, by changing the distance, orientation of the device's screen, this image will change correspondingly. The image will be saved in gallery when you tap the "COLLECT" button on the bottom. Unfortunately, the image's background is black, and we couldn't figure out how to use real picture as the image's background.


COLLECT

Collecting tag screen

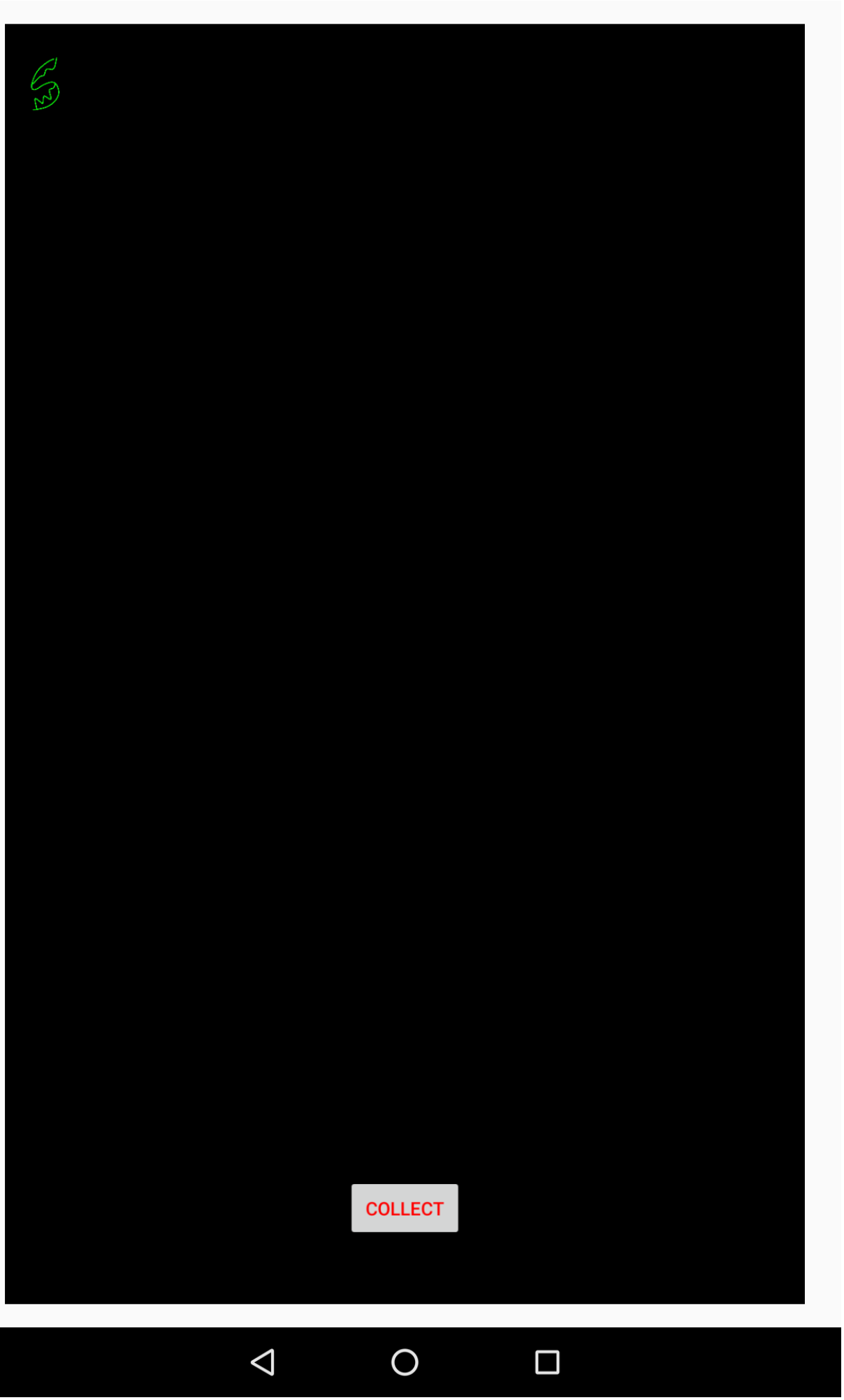
Step 6: The two picture below show the gallary. When you click the "GALLERY" button on google map screen. You will be directed to the "Gallery screen-1", it lists all the tags you have collected by listing the urls. By clicking any url, you will be directed to "Gallery screen-2" which shows the actual tag image.

http://roblkw.com/msa/collect//57fdb9ec35dd0.png
http://roblkw.com/msa/collect//57fdba38b14a6.png
http://roblkw.com/msa/collect//57fdba9c1b5cb.png
http://roblkw.com/msa/collect//57fdc1d71affc.png
http://roblkw.com/msa/collect//57fec160a91f2.png
http://roblkw.com/msa/collect//57ffed39ed6ba.png



Gallery

screen-1



Gallery

screen-2

Design (coding part)

This app is developed in Android Studio. The entire codes is composed of two major parts - .xml and .java file. The .xml files define the layouts of user interface while the .java files form the backbone of this application and enable the functionalies.

The .xml files are stored in `app\src\main\res\layout` folder:

- `activity_main.xml` - decribes the Login screen, in which the "Sign in with Google" is defined as below

```
<com.google.android.gms.common.SignInButton
    android:id="@+id/login"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom">
</com.google.android.gms.common.SignInButton>
```

- `activity_google_map.xml` - decribes the Map screen, in which the "Googe Map" is contained in a `fragment` view

```
<fragment
   android:layout_width="match_parent"
   android:layout_height="422dp"
   android:name="com.google.android.gms.maps.MapFragment"
   android:id="@+id/mapFragment"
   android:layout_alignParentTop="true"
   android:layout_weight="1.21" />
```

- `activity_place.xml` - decribes the tag placing screen, in which the camera and drawing is implemented in two seperated fullscreen views

```
<com.example.jianan.auggraffiti.CameraPreview
        android:id="@+id/camera_preview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true" />
```

```
 <com.example.jianan.auggraffiti.Graphique
        android:id="@+id/graph"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
```

```
        android:layout_alignBottom="@+id/button_save"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />
```

- `activity_collect.xml` - decribes the tag collecting screen, in which the camera and tagImage is implemented in fullscreen view and ImageView, respectively.

```
<com.example.jianan.auggraffiti.CameraPreview
        android:id="@+id/camera_preview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true" />

    <ImageView
        android:id="@+id/tag_image"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="300dp" />
```

- `activity_gallery.xml` - decribes the gallery screen, in which collected images are stored and shown in a `ListView`.

```
<ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/list"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:choiceMode="singleChoice" />
```

The .java files are stored in `app\src\main\java\com\example\jianan\auggraffiti` folder:

- `MainActivity.java` - enables Google Sign-In, database interactions and GoogleMapActivity initiation.

Once the app icon is tapped, the app is created by calling `onCreate` callback function. During the creation period, it create a `GoogleSignInOptions` object `signInOptions` which configures sign-in to request users basic sign-in information, here we request user's email for sign-in by calling `.requestEmail()` methods. In the same time, a `GoogleApiClient` object is instantiated as `googleApiClient` to access Google's Sign-In API, the options are specified in `signInOptions` argument. Once the Sign-In button is clicked, the ```onClick`` callback function is invoked and it starts the Google Sign-In activity.

```
protected void onCreate(Bundle savedInstanceState) {
        ...
        signInOptions = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN).requestEmail().bui
ld();
        googleApiClient = new GoogleApiClient.Builder(this).enableAutoManage(this,
this).addApi(Auth.GOOGLE_SIGN_IN_API, signInOptions).build();
        ...

        login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent signInIntent =
Auth.GoogleSignInApi.getSignInIntent(googleApiClient);
                startActivityForResult(signInIntent, REQUEST_CODE);
            }
        });
    }
```

The Google Sign-In results are rerived by `onActivityResult` callback function. The Sign-In results are stored in `result` which is a `GoogleSignInResult` object. If Sign-In succeeds (`result.isSuccess()`), the sign-in person's personal email is returned through `getEmail()` method. In the next step, the personal email is post to database. In this process, a request queue is first initiated by calling a `Volley` method `newRequestQueue()` with an argument `FragmentActivity` type. Then a `StringRequest` object `stringRequest` is added to the queue to talk to database. The `StringRequest` is instantiated by calling a `postStringRequest` function.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == REQUEST_CODE) {
            GoogleSignInResult result =
Auth.GoogleSignInApi.getSignInResultFromIntent(data);

            //to make sure continue to the next screen if the sign failed for
unknown reason
            if(result.isSuccess()) {
                GoogleSignInAccount account = result.getSignInAccount();
                personEmail = account.getEmail();

                RequestQueue queue = Volley.newRequestQueue(this);
                String url = "http://roblkw.com/msa/login.php";
                final Map<String, String> params = new HashMap<String, String>();
                params.put("email", personEmail);

                // Request a string response from the provided URL.
                StringRequest stringRequest = postStringRequest(params, url);
                // Add the request to the RequestQueue.
                queue.add(stringRequest);
            }
            else{
                Status status= result.getStatus();
                Toast.makeText(this,status.toString(), Toast.LENGTH_LONG).show();
```

```
            }

        }
    }
```

In `postStringRequest` function, it post a `<"email",personalEmail>` hashmap onto the specified `url` and the response is retrived by a `onResponse` callback function. If the database login succeeds, it replies a `String 0`. Once `"0"` is received, a `sendMessage()` method is invoked. This method starts an `Intent` to open the `GoogleMapActivity.class` activity and it also passess a `String` argument named `personalEmail` to that activity for furtuer use.

```
 private StringRequest postStringRequest(final Map<String,String> params, final
String url) {
        return new StringRequest(Request.Method.POST, url,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        if(response.equals("0")) // when get 0, log in successfully
                            sendMessage(personEmail);
                    }
                    ...
```

```
        public void sendMessage(String message) {
        Intent intent = new Intent(getApplicationContext(),
GoogleMapActivity.class);
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
        }
```

- `GoogleMapActivity.java` - enables Google Map, and database interaction for "C" tag unveiling.

During the activity creation, the Google Map is initiated by `initMap()` method. In this method, a `MapFragment` object is instantiated through `findFragmentById()` method with an argument `R.id.mapFragment`. Then, the Google Map starts by calling `getMapAsync()` method.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
            initMap();
        }
        ...
```

```
    private void initMap() {
        MapFragment mapFragment = (MapFragment)
getFragmentManager().findFragmentById(R.id.mapFragment);
        mapFragment.getMapAsync(this);
    }
```

Once the Google Map is started and ready for next operation, the `onMapReady` callback function is invoked, a `GoogleMap` argument is passed to this method. In `onMapReay`, and new `GoogleApiClient` is `build()` and this Api is used to connect to the Google Map server by calling `connect()` method. There is alos a `setOnMarkerClickListener()` method implemented here, it will tracking the clicking-marker event.

```
 public void onMapReady(GoogleMap googleMap) {
        mGoogleMap = googleMap;
        mGoogleApiClinet = new GoogleApiClient.Builder(this)
                .addApi(LocationServices.API)
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .build();
        mGoogleApiClinet.connect();
        mGoogleMap.setOnMarkerClickListener(this);
    }
```

Once the server is connected, a `onConnected()` callback function is invoked and the map location details can be specified in this funciton, such as in the `setPriority()` function, you can specifed the accuracy of Google Map, here we're using high accuracy `PRIORITY_HIGH_ACCURACY` and `ACCESS_FINE_LOCATION`.

```
public void onConnected(Bundle bundle) {
        mLocationRequest = LocationRequest.create();
        mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        mLocationRequest.setInterval(10000);
        if(Build.VERSION.SDK_INT>=Build.VERSION_CODES.M){
            ...
            if (ActivityCompat.checkSelfPermission(this,
android.Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
                Log.v(tag,"not get permission");
                Toast.makeText(this,"Please allow permission to access your
location in the setting", Toast.LENGTH_LONG).show();
                return;
            }
            Log.v(tag,"get permission");
        }

LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClinet,mLocation
Request, this);
    }
```

When any marker is clicked, the `onMarkerClick` callback function is invoked. Then it will decide which marker is clicked, either `Collect` marker or `Place` marker. Then the `sendMessage()` function will invoke different `Intent` based on clicked marker.

```
@Override
```

```
    public boolean onMarkerClick(final Marker marker) {
        String[] strings = marker.getTitle().trim().split(" ");
        switch( strings[0]) {
            case "Collect":
                Toast.makeText(this, "Collect has been clicked ",
                    Toast.LENGTH_SHORT).show();
                sendMessage("Collect", strings[1]);
                break;
            case "Place":
                Toast.makeText(this, "Place has been clicked ",
                        Toast.LENGTH_SHORT).show();
                sendMessage("Place",null);
                break;
        }
        return false;
    }
```

In this `sendMessage()` method, it will either invoke `CollectActivity` activity of `PlaceActivity` activity based on the marker that has been clicked. If `CollectActivity` is invoked, it will also bundle the `tagID` information together with the `Intent`.

```
public void sendMessage(String activity, String tagId) {
        Intent intent = null;
        switch (activity){
            case "Collect":
                 intent = new Intent(getApplicationContext(),
CollectActivity.class);
                intent.putExtra(TAGID_MESSAGE, tagId);
                Log.v("TAGID_MESSAGE",tagId);
                break;
            case "Place":
                intent = new Intent(getApplicationContext(), PlaceActivity.class);
                break;
        }
        if(intent != null) {
            //EditText editText = (EditText) findViewById(R.id.edit_message);

            startActivity(intent);
        }

    }
```

Once you moved and the location changed, the `onLocationChanged(Location location)` callback function is invoked. And the current location is passed in as `Location` type argument. The curent latitude and longitude are stored in `lat` and `lng` respectively with the method `getLatitude()` or `getLongitude()`. The Map screen automatically follows your movement because of calling of methode `animateCamera()` which has an argument of `CameraUpdate` type. In the same time, the current location data is send to database in a `HashMap` format, where the keys and values are all `String` type. The `HashMap` object `params` contains 3 key, vaule pairs. The first key is`"email"` which has a value retrived from the Intent sent from `MainActivity`, the retriving function is `intent.getStringExtra(MainActivity.EXTRA_MESSAGE)`. The second and third key is `loc_long` and

`loc_lat` repectively, their values are retrived from `location`. The `params HashMap` is post to database server in a similar way discussed in previous paragrah. First, a `RequestQueue` is instantiated through method `newRequestQueue` in `Volley`. The content that wshould be posted is in `StringRequest` object called `stringRequest` which is initiated through calling a function `postTagNearByRequest` which has two argumments, a `HashMap` and a `url`. Then, this request is send to database server with `add()` method.

```
public void onLocationChanged(Location location) {//when the location is changed
        ...
             Double lat = location.getLatitude();
             Double lng = location.getLongitude();

             ...

                 CameraUpdate update = CameraUpdateFactory.newLatLngZoom(ll,
19);//use cameraupate to focus the screen to the current position
                 mGoogleMap.animateCamera(update);

             ...

                 RequestQueue queue = Volley.newRequestQueue(this);
                 //post request to get the tags nearby
                 String url = "http://roblkw.com/msa/neartags.php";
                 final Map<String, String> params = new HashMap<String, String>();
                 if(personEmail==null) {
                     Intent intent = getIntent();
                     personEmail =
intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
                 }
                 params.put("email", personEmail);
                 params.put("loc_long", lng.toString());
                 params.put("loc_lat", lat.toString());
                 // Request a string response from the provided URL.
                 StringRequest stringRequest = postTagNearByRequest(params, url);
                 queue.add(stringRequest);
             }
         }
     }
```

Once the request is posted, the response from database is collected through callback function `onResponse`, frome where, the "C" tag data is recieved, analyized and shown on Map screen through `setCollectMarker`method.

```
private StringRequest postTagNearByRequest(final Map<String,String> params, final
String url) {
        return new StringRequest(Request.Method.POST, url,

                 new Response.Listener<String>() {
                     @Override
                     public void onResponse(String response) {
                         //The format of the response is "tagId,Latitude,Longitude"
                         String[] tagLoc = response.trim().split("[,]+");
```

```
                        //First we assume the format of the response never changed
and the response we get is always correct
                        //then we can get the numb int numTag = 0;
                        int numTag = tagLoc.length%3==0?tagLoc.length/3:-1;
                        if(tagList == null){
                            tagList = new LinkedList<Tag>();
                        }
                        int i = numTag;
                        while(--i >=0) {//we have tags near by
                            //  double lat = Double.valueOf(tagLoc[numTag * 3 +
1]);
                            // LatLng ll = new LatLng(Double.valueOf(tagLoc[numTag
* 3 + 1]), Double.valueOf(tagLoc[numTag * 3 + 2]));
                            tagList.add(new Tag(Integer.valueOf(tagLoc[i * 3]), new
LatLng(Double.valueOf(tagLoc[i * 3 + 2]), Double.valueOf(tagLoc[i * 3 + 1]))));
                            setCollectMarker(tagList.get(numTag - i- 1).ll);
                        }

                        ...

    }
```

Similar to `postTagNearByRequest()`, the `postScoreStringRequest()` method is used to post request to server to get the score updated. It has two input argument, which is `Map` and `String`, the id, location information is stored in `Map<String,String>` input argument, the url information is store in `String` argument. This mehod return a `StringRequest` object which has four input argument, i.e. `Request.Method.POST`, `url`, `Response.Listener` and `onErrorResponse`. There is a callback function `onResponse` in the `Response.Listener` anonymous inner class. When the server has a response, the reponse string is stored in the input argument of `onResponse` callback. And the score value is stored in the `score` textView.

```
private StringRequest postScoreStringRequest(final Map<String,String> params, final
String url) {
        return new StringRequest(Request.Method.POST, url,

                new Response.Listener<String>() {
                    @Override
                    // no idea why enter in this function 3 times when just sending
GPS information just once.
                    public void onResponse(String response) {
                        Log.v(tag,response);
                        if(score == null){
                            score = (TextView) findViewById(R.id.score);
                        }
                        score.setText(response);
                    }
                }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                //mTextView.setText("That didn't work!");
            }
        }){
            @Override
```

```
            protected Map<String, String> getParams() {
                return params;
            }
        };
    }
```

This is `setCollectMarker()` method, which shows the marker on the googleMap, it has two input arguments, location and marker id. And the marker image is load from `drawable` foler, it uses `mGoogleMapActiviity.addMarker()` to load the marker onto map.

```
private Marker setCollectMarker(LatLng ll, int tagID){
        MarkerOptions optionsCollect = new MarkerOptions()
                .title("Collect")
                .icon(BitmapDescriptorFactory.fromResource(R.drawable.c))
                .position(ll)
                .snippet("Clicking me to collect a tag");
        return mGoogleMapActivity.addMarker(optionsCollect);
    }
```

Similar to `setCollectMarker`, below is `setPlaceMarker` method, it loads and shows the placeMarker onto googleMap with a similar way.

```
private void setPlaceMarker(double lat, double lng) {
        if(placeMarker != null){
            placeMarker.remove();
        }
        MarkerOptions optionsPlace = new MarkerOptions()
                .title("Place")

//.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN))
                .icon(BitmapDescriptorFactory.fromResource(R.drawable.pm))
                .position(new LatLng(lat,lng))
                .snippet("Clicking me to place a tag");
        placeMarker = mGoogleMap.addMarker(optionsPlace);
    }
```

- `CameraPreview.java` - defines how camera is opened and shown on activity screen.

This `CameraPreview` class extends `SurfaceView` class and implements `SurfaceHolder` interface. This class has two attributes, i.e. `private Camera camer` and `private SurfaceHolder holder`, and four implemented functions: the `public void init` function inits the Camera by calling the `initSurfaceHolder()` function.

```
public void init(Camera camera) {
        this.camera = camera;
        initSurfaceHolder();
    }
```

```
private void initSurfaceHolder() {
        holder = getHolder();
        holder.addCallback(this);
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
```

Once the `initSurfaceHolder()` function is excuted, the `surfaceCreated` callback function will be invoked. And in this function, it will run `initCamera` function.

```
public void surfaceCreated(SurfaceHolder holder) {
        initCamera(holder);
    }
```

In `initCamera` function, the camera view will be opened by `camera.startPreview()` function, only after the `camera.setPreviewDisplay(holder)` is ready. Any errors will be shown in Log according to the `catch (Exception e)` method.

```
private void initCamera(SurfaceHolder holder) {
        try {
            camera.setPreviewDisplay(holder);
            camera.startPreview();
        } catch (Exception e) {
            //Log.d("Error setting camera preview", e);
        }
    }
```

- `Graphique.java` - this class defines how to draw graffiti on the activity screen and how to save it as base64 .JPEG format.

This `Graphique` class extends `View` class. In this class, there are three attributes, i.e. `Path`, `Paint` and `Canvas`, the first one defines the coordinates of every drawing pixel, the second one defines the propertie of the draiwing pixel, and the third one defines the drawing medium (background).

The method `init` initiates the property of drawing pixels, it set the color to be `RED` with `setColor` function, and it smoothes the drawing lines with `setAntiAlias` function, it set the line style to be `STROKE` in `setStyle` funtion, sets the line width to be `5f` in `setStrokeWidth` function, etc.

```
 private void init(){
        paint.setColor(Color.RED);
        paint.setAntiAlias(true);
        paint.setStrokeJoin(Paint.Join.ROUND);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(5f);
        this.setDrawingCacheEnabled(true);
    }
```

The `onTouchEvent` callback funtion is invoked once touching screen event happens, it acquires the

touching coordinates (X,Y) through the methode `getX()` or `getY()` in `motionEvent` instance. While `ACTION_DOWN` or `ACTION_MOVE` it retrives the coordinates and send it to `path` instance.

```
@Override
    public boolean onTouchEvent(MotionEvent motionEvent){
        float motionX = motionEvent.getX();
        float motionY = motionEvent.getY();

        switch(motionEvent.getAction()){
            case MotionEvent.ACTION_DOWN:
                path.moveTo(motionX,motionY);
                break;
            case MotionEvent.ACTION_MOVE:
                path.lineTo(motionX,motionY);
                break;
            case MotionEvent.ACTION_UP:
                break;
        }
        invalidate();

        return true;
    }
```

The `onDraw` callback function invokes when the view is intatiated and when the `invalidate()` function is called in `onTouchEvent()` callback function. It takes a `Canvas` as input parameter, which allows the view to draw itself. The filled color is set to be `TRANSPARENT` via `drawColor()` method. The `drawPath` defines the shape of draw, it takes `Path` and `paint` as its two input argument which defines the shape and color of the drawing.

```
 @Override
    protected void onDraw(final Canvas canvas) {

        this.canvas = canvas;
        canvas.drawColor(Color.TRANSPARENT);
        canvas.drawPath(path, paint);
    }
 ```

The ```saveCanvasToBitmap()``` function is used to convert what you have drawn into image which will be saved in database. If the ```getDrawingCache()``` method is used to extract the drawing from cache, and save the result in ```Bitmap``` instance ```bitmap```. if the ```bitmap``` is not empty, it compress the image to .JEPG format. This .JEPG image is then transformed into byteArray using ```toByteArray()``` method. The ```byteArray``` is then base64 encoded using ```Base64.encodeToString()``` method with one argument to be ```byteArray``` and another argument to be ```Base64.NO_WRAP```.
```

```
public String saveCanvasToBitmap(){
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    this.buildDrawingCache(true);
```
```

```
    Bitmap bitmap = getDrawingCache();
    if(bitmap == null){
        Log.v("bitmap is null");
        return null;
    }
    else{
        bitmap.compress(Bitmap.CompressFormat.JPEG, 0, byteArrayOutputStream);
        byte[] byteArray = byteArrayOutputStream .toByteArray();
        String base64 = Base64.encodeToString(byteArray, Base64.NO_WRAP);
        Log.v("the length of the compress string is" +
String.valueOf(base64.length()));
        return base64;
    }
}
```

```
- ```PlaceActivity.java``` - this activity implements graph drawing, location and
orientation retriving from sensor and how to commit the tag into server.

This class extends ```Activity``` class and implements ```PictureCallback```,
```LocationListener```, ```SensorEventListener``` interfaces.

In the ```onCreate()``` callback functions. It instatiates ```LocationManager```
which manages the GPS location of device, and ```SensorManager``` which manages
different types of sensors such as ```TYPE_ACCELEROMETER``` and
```TYPE_MAGNETIC_FIELD```. The ```CameraHelper.getCameraInstance()``` obtains a
```Camera``` instance, if the camera is avalable, it will be initiated by
```initCameraPreview()``` method, or it will be finished. The ```Graphique``` is
instantiated as ```graph```. Then this ```graph``` is send to sever by
```sendImageToServer()``` method when clicking ```R.id.button_save``` button.
```

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);

```
    ...

    locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    mMagnetometer =
this.mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

    camera = CameraHelper.getCameraInstance();
    if (CameraHelper.cameraAvailable(camera)) {
        initCameraPreview();
    } else {
        finish();
    }

    graph = (Graphique) findViewById(R.id.graph);
    graph.setVisibility(View.VISIBLE);
    btnSave = (Button) findViewById(R.id.button_save);
    btnSave.setOnClickListener(new View.OnClickListener() {
        @Override
```
```

```
        public void onClick(View v) {
            btnSave.setTextColor(Color.RED);
            Log.v("Click the button");
            sendImageToServer(graph.saveCanvasToBitmap());
        }
    });
    client = new GoogleApiClient.Builder(this).addApi(AppIndex.API).build();
}
```

```
In the ```initCameraPreview()``` method, the camera is opened. First, the
```CameraPreview``` is instatiated and then the ```camera``` is opened by calling
```init(camera)``` method of ```cameraPreview``` class.
```

private void initCameraPreview() { cameraPreview = (CameraPreview) findViewById(R.id.camera_preview); cameraPreview.init(camera); }

```
This ```sendImageToServer()``` method takes String argument (which is the Base64
image) and post it on sever. The posting method is same as talked above. The
```Map<String, String>``` data send to the sever are, listed are keys, email
(```email```), coverted images (```tag_img```), location
(```loc_long```,```loc_lat```), orientation (```orient_azimuth``` and
```orient_altitude```). The ```postScoreStringRequest(final Map<String, String>
params, final String url)``` takes two arguments, the formulated ```Map<String,
String>``` and ```url```, it's same as the ```postScoreStringRequest()``` method in
```GoogleMapActivity.java``` activity.
```

public boolean sendImageToServer(String imgString) { //send request to get score RequestQueue queue = Volley.newRequestQueue(this);

```
    String url = "http://roblkw.com/msa/placetag.php";

    final Map<String, String> params = new HashMap<String, String>();

    params.put("email", "jianan205@gmail.com");
    params.put("tag_img", imgString);
    params.put("loc_long", String.valueOf(lng));
    params.put("loc_lat", String.valueOf(lat));
    params.put("orient_azimuth", String.valueOf(mAzimuth));
    params.put("orient_altitude", String.valueOf(altitude));
    StringRequest stringRequest = postScoreStringRequest(params, url);
    queue.add(stringRequest);
    return true;
}
```

```
In the ```onPause()``` callback, it releases the camera by calling
```releaseCamera()``` function and unregisters the SensorManager by calling
```unregisterListener()``` methode. In the ```releaseCamer()``` method, the Camera
is released by calling ```release()``` method.
```
```

```
@Override
protected void onPause() {
    super.onPause();
    releaseCamera();
    mSensorManager.unregisterListener(this);
}

private void releaseCamera() {
    if (camera != null) {
        camera.release();
        camera = null;
    }
}
```

```
In the ```onResume()``` callback, it register the ```SensorManager``` by calling ```registerListener()``` function and update the location by calling ```requestLocationUpdates()``` function.
```

@Override protected void onResume() { super.onResume(); mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_GAME); mSensorManager.registerListener(this, mMagnetometer, SensorManager.SENSOR_DELAY_GAME); ... locationManager.requestLocationUpdates("gps", (long) 5000, (float) 0.0, this); }

```
The senor data is retrived in the ```onSensorChanged()``` callback. the ```ACCELEROMETER``` data is saved in ```gData```, the ```TYPE_MAGNETIC_FILED``` data is saved in ```mData```, the rotation and inclination data is calculated through ```getRotationMatrix()``` method, and the ```mAzimuth``` value is calcualted in the last line of this code section.
```

```
@Override
public void onSensorChanged(SensorEvent event) {
    float[] data;
    switch (event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            gData = event.values.clone();
            break;
        case Sensor.TYPE_MAGNETIC_FIELD:
            mData = event.values.clone();
            break;
        default:
            return;
    }

    if (SensorManager.getRotationMatrix(rMat, iMat, gData, mData)) {
        mAzimuth = (int) (Math.toDegrees(SensorManager.getOrientation(rMat,
orientation)[0]) + 360) % 360;
    }
}
```

```
- ```CollectActivity.java``` activity defines how the tags shown on googleMap are collected, and score is updated.

This activity implements in a very similar to ```PlaceActivity.java```, except in the ```onCreate()``` callback it retrieve the data intented from ```GoogleMapActivity``` using ```getStringExtra()``` method. The screen size (```screenWidth```, ```screenHeight```) is retrieved from ```getSize()``` function in ```Display``` class, which will be used later to adjust the size of collecting tag.
```

@Override protected void onCreate(Bundle savedInstanceState) { ... Intent intent = getIntent(); tagId = intent.getStringExtra(GoogleMapActivity.TAGID_MESSAGE); Log.v("The collect ID is "+String.valueOf(tagId)); ... Display display = getWindowManager().getDefaultDisplay(); Point size = new Point(); display.getSize(size); screenWidth = size.x; screenHeight = size.y;

```
}
```

```
The collecting tag inforamtion is get by sending request to sever, using ```findImgFromServer()``` method. The data sent by sever is get using ```postScoreStringRequest()``` method, if we get response from sever, the retrived String is parsed including, image string ```tagInfo[0]```, ```orientation_azimuth``` string, and ```orientation_altitude``` string. The details of implementation of this method is the same as what I shown in before, please refer to ```GoogleMapActivity.class``` section.
```

public void onResponse(String response) { if(response!=null){ String[] tagInfo = response.trim().split("[,]+"); loadImg(tagInfo[0]); orientation_azimuth = Integer.valueOf(tagInfo[1]); Log.v("oriazimuth is" + orientation_azimuth); orientation_altitude = Integer.valueOf(tagInfo[2]); }

```
The tag size and orientation is adjusted through ```onSensorChanged()``` callback. The lifetime ```ACCELEROMETER``` and ```MEGNETOMETER``` dada is retrieved using ```getType()``` and ```event.values.clone()``` function. The relative size of the tag image is adjusted with a ```factor``` multiples the screen size.
```

@Override public void onSensorChanged(SensorEvent event) { switch (event.sensor.getType()) { case Sensor.TYPE_ACCELEROMETER: gData = event.values.clone(); break; case Sensor.TYPE_MAGNETIC_FIELD: mData = event.values.clone(); break; default: return; }

```
    ...

     RelativeLayout.LayoutParams imageLayout = new RelativeLayout.LayoutParams(imageMargin);
```

```
    imageLayout.height = (int)(screenHeight*factor);
    imageLayout.width =  (int)(screenWidth*factor);
    imageView.setLayoutParams(imageLayout);

}
```

```
- ```GalleryActivity.java``` activity stores the collected images.

In this ```onCreate()``` callback, it initiated the ```ListView``` which stores all
the collected images. The ```getGallary()``` method post a request to sever to load
the images, the sever sends back the image url which is used to update the
gallery's ```ListView``` using ```setUpList()``` method. The ```showImage()```
method opens the clicked images in another activity which is called through
```Intent``` class.
```

@Override protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.activity_gallery); listView = (ListView) findViewById(R.id.list); listView.setOnItemClickListener(new AdapterView.OnItemClickListener() { @Override public void onItemClick(AdapterView<?> parent, View view, int position, long id) { TextView textView = (TextView) view; showImage(textView.getText().toString()); } }); getGallary(); client = new GoogleApiClient.Builder(this).addApi(AppIndex.API).build(); }

```

```

public boolean getGallary() { RequestQueue queue = Volley.newRequestQueue(this);

```
    String url = "http://roblkw.com/msa/getgallery.php";

    final Map<String, String> params = new HashMap<String, String>();

    params.put("email", "jianan205@gmail.com");
    StringRequest stringRequest = postScoreStringRequest(params, url);
    queue.add(stringRequest);
    return true;
}
```

```

```
    public void showImage(String imgUrl){
        Intent intent = new Intent(getApplicationContext(), ImageActivity.class);
        intent.putExtra(EXTRA_MESSAGE, imgUrl);
        startActivity(intent);
    }
```

```
private void setUpList(String[] imgUrl){
        ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,R.layout.test,imgUrl);
```

```
        listView.setAdapter(adapter);
    }
```

Strategy

1. We split the work into half, one person focuses on coding and the other person focuses on finding resources, cleaning codes and writing documentation.
2. We check our progress week by week, and make sure we can finish the project. We made several internal check points:

```
            - learning basics of android studio;
            - learning GitHub;
            - finishing the front page design of the app;
            - implementing google login;
            - talking with sever;
            - implementing google map;
            - place a marker on google map;
            - implementing camera;
            - implementing screen drawing with camera backgound;
            - saving Base64 image format;
            - control sensors;
            - retrieve data from sever;
```

After completing all the check points, we successfully designed the app.

Challenges

As we are both new to android programing, we meet tremendous challenges:

```
            - first challenge is to learn android studio and get familiar with this
brand programming environment;
            - leanring .xml language consumes us some time, this is the first time
we use .xml language;
            - second challenge is to get familiar with GitHub which we used later
for monitoring progress and version control;
            - this google-API-signIn idea is new to us, we discussed the logic for
several times to figure out how it works;
            - implement google map is not a big problem, there are plenty of
resources online.
            - it takes us some time to figure out how to talk with professor's
sever, sine we also need to under stand the MySQL code
            - we spend a lot of time trying to figure out how to calculate the
orientation of the tag, this is chanllenging;
            - the biggest challenge is how to manage our time to finish this
project but not affecting too much of other important                    things,
such as learnign course materials, studying other courses and doing research.
```

Improvement

There are several improvement (suggestions) for future works:

- using a comercial sever rather than local, private server, by doing this, we can pulish our app and people around the world can play with it.
- we need to improve the UI of this app, since we are a little bit short of time, we are more concentrating on the functionality rather than user friendly.
- the background of collecting tag should be changed from black to realtime view of the camera.