Tampereen ammattikorkeakoulu

Mobile App Development 2

Native Android Project

Chou-Ping Ding

13/10/2023

Contents

1 Native Android Project – GeoLocation Distance Calculation App

1.1 Development Process

The basic functionalities of this application are generally covered within the lecture and examples demonstrated; thus, the base idea and vague structure of the application is already set. Therefore, the challenge lies in combining the two functions and integrate them in to one single app while manipulating the data gained to make the calculation.

First part of the development process I started with the structure of QR code scanning demonstrated (also in accordance with the source reference from Digital Ocean), which have a separate view and a button in the main activity to redirect the user to the scan view. I had planned to first combine the scanner and the result from the scanning, location data of the device and the calculated value of distance. Initially I felt since transferring data between the activities required more processing, it would be the best that just to keep everything in one page and the main activity can just include all the functionalities. But then the problem starts to appear with the camera permission when the app is launched for the first time, the intent needs to be restarted after obtaining user permissions in order to initiate the camera. In the end I took the method that the permission check is activated with the start of the main activity, and the main functionality of the app are located in a separate scan activity.

```
@Override
protected void onStart() {
    super.onStart();
    checkLocationPermissions();
}

1 usage
private void initViews() {
    scanButton = findViewById(R.id.startButton);
    scanButton.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    startActivity(new Intent( packageContext: MainActivity.this, scanQR.class));
}
```

The permission check was set inside a function to keep the layout clean and more manageable.

```
private boolean checkLocationPermissions() {
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
            || ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED
            || ActivityCompat.checkSelfPermission( context: this, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
        // Request permissions
        String[] permissions = {Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.CAMERA};
        ActivityCompat.requestPermissions( activity: this, permissions,  requestCode: 42);
        return false; // Permissions not granted yet
    }
    return true; // Permissions are already granted
}
```

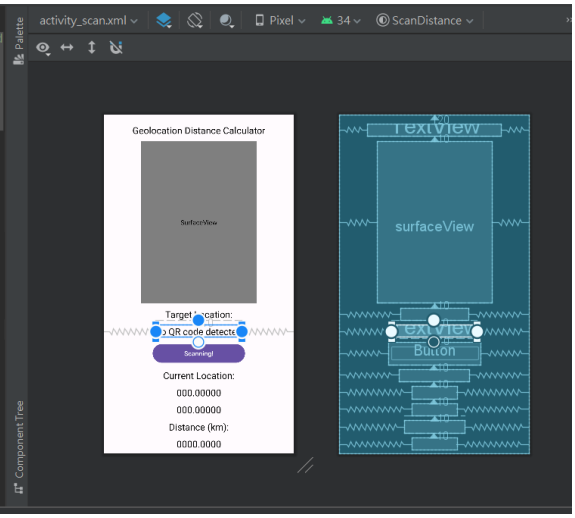
After entering the scan activity, the view includes a surface view for the QR code scanning a text field for displaying the obtained information and a button which activates the calculation process. I've added content verification to the code, so when the data contained in the QR code does not fit the desired format (geo URI), the button will display not valid.

```
no usages
@Override
public void receiveDetections(@NonNull Detector.Detections<Barcode> detections) {
    final SparseArray<Barcode> barcodes = detections.getDetectedItems();
    if (barcodes.size() != 0) {
        final String scanResult = barcodes.valueAt( index: 0).displayValue;
        txtBarcodeValue.post(() -> {
            txtBarcodeValue.setText(scanResult);
            if (isGeolocationFormat(scanResult)) {
                actionButton.setText(R.string.use_this_location);
            } else {
                actionButton.setText(R.string.not_a_valid_code);
            }
        });
    }
```

When creating this activity, I began with the main structure of the tutorial an example code and added the needed text field and button for clear and easy to read layout. By setting the layout first, it can be easier while assign values to the text boxes and also the id naming of the objects can be more uniformed.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/titleSection"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:layout_marginTop="20dp"
        android:text="Geolocation Distance Calculator"
        android:textColor="@android:color/black"
        android:textSize="20sp" />

    <SurfaceView
        android:id="@+id/surfaceView"
        android:layout_width="250dp"
        android:layout_height="350dp"
        android:layout_centerHorizontal="true"
        android:layout_below="@id/titleSection"
        android:layout_marginTop="10dp" />
```



And for obtaining the goe location data, the initial problem I encountered was that the data only update once since the functionality was set with the permission check. I moved this part of the code to a function and run this function whenever the activity is or resumed, similar code is also implemented within the calculation function which will be activated when the calculation button is pressed.

```
private void checkAndRequestLocationUpdates() {
    if (checkLocationPermissions()) {
        startLocationUpdates();
    }
}

1 usage
private void startLocationUpdates() {
    // Start listening for location updates here
    fusedLocationClient.getLastLocation()
            .addOnSuccessListener( activity: this, new OnSuccessListener<Location>() {
                no usages
                @Override
                public void onSuccess(Location location) {
                    if (location != null) {
                        txtCurrentLatGeo.setText(Double.toString(location.getLatitude()));
                        txtCurrentLonGeo.setText(Double.toString(location.getLongitude()));
                    }
                }
            });
}
```

```
@Override
protected void onResume() {
    super.onResume();
    initialiseDetectorsAndSources();
    checkAndRequestLocationUpdates();
}
```

After laying out the generally frame for the application and have all the desired function working as expected, the next is the details of verification of the data retrieved. Since there's possibility of invalid data, verification of the data is also done, with different cases taken into consideration.

First is that the value obtained from the QR code don't began with "geo:", this will be filtered from the beginning so that the test will not enter the parsing pro-

cess. And the calculation button will also have a text display to inform the user about invalid QR code.

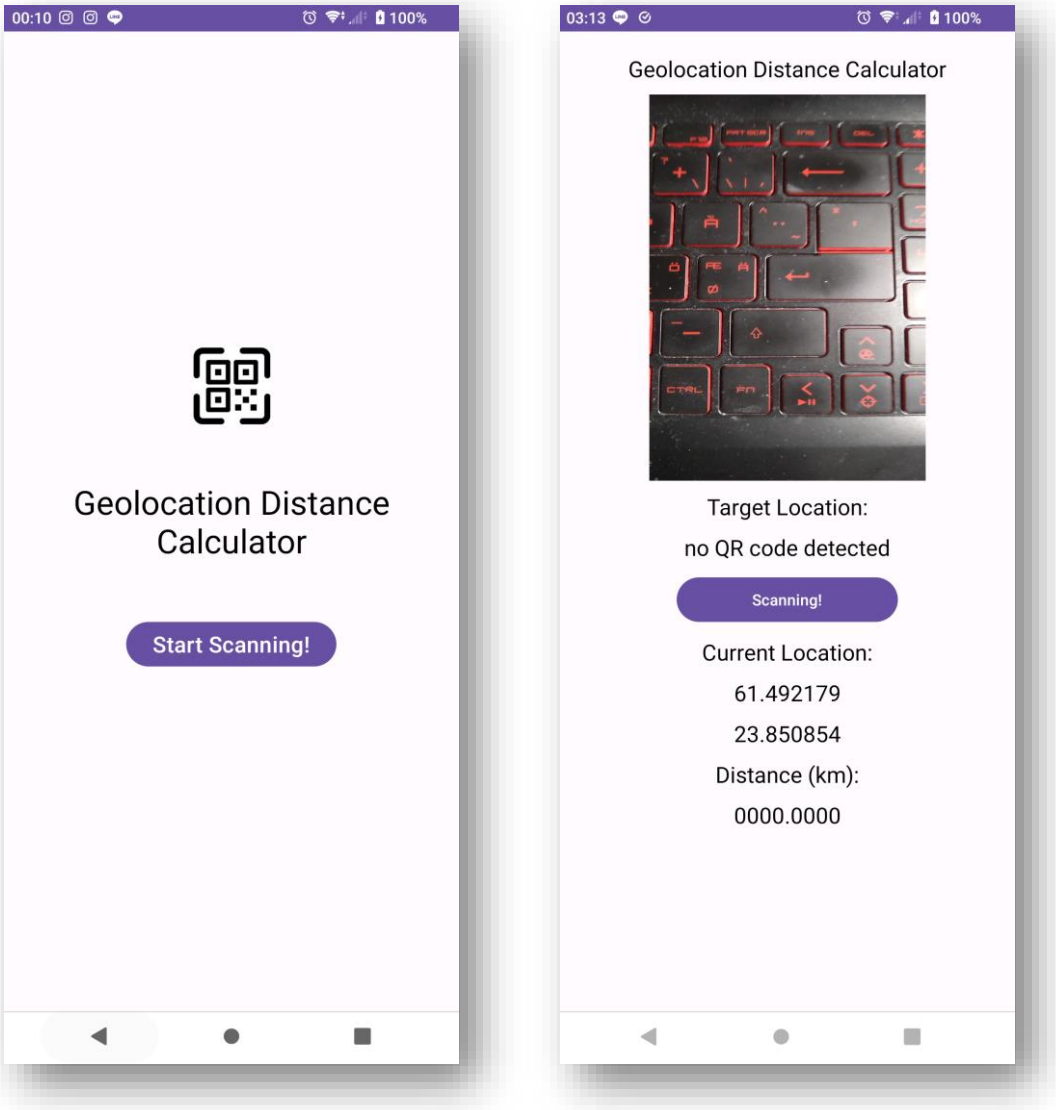
```
private boolean isGeolocationFormat(String content) {
    // Assuming geolocation format is "geo:latitude,longitude"
    return content != null && content.startsWith("geo:") && content.split( regex: ",").length == 2;
}
```

If the data have proper format, then it will enter the parsing phase and the result will be used in the function which calculation the distance between two geolocations. But if the text derived from the QR code failed to parse into numerical value the calculation result text box will also give user a warning about invalid data.

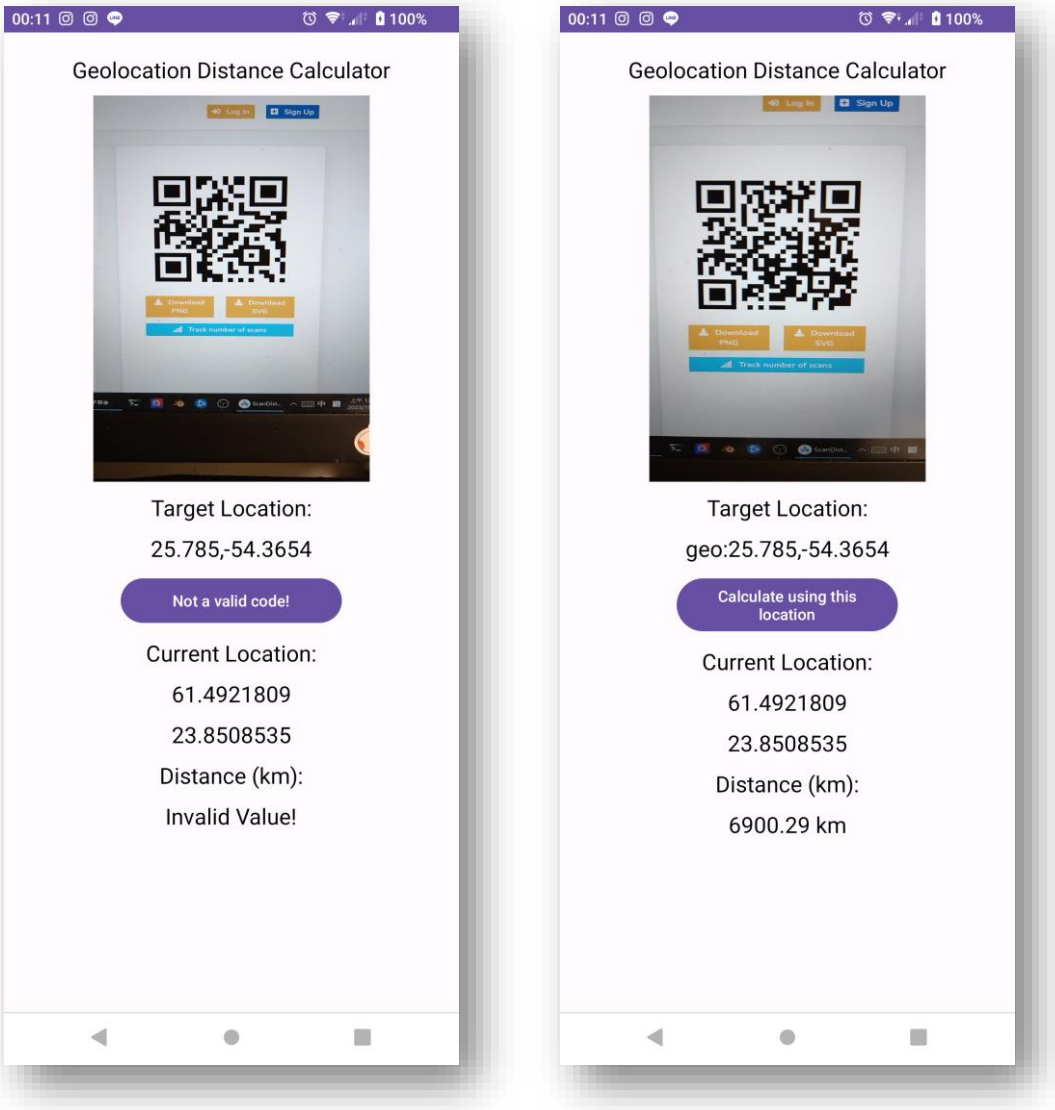
```
public void calculateDistance() {
    String qrCodeContents = txtBarcodeValue.getText().toString();

    if (qrCodeContents.startsWith("geo:")) {
        String geoData = qrCodeContents.substring( beginIndex: 4);
        String[] parts = geoData.split( regex: ",");
        if (parts.length >= 2) {
            try {
                double latitude = Double.parseDouble(parts[0]);
                double longitude = Double.parseDouble(parts[1]);
                if (ActivityCompat.checkSelfPermission( context: this, Manifest.pe
                        && ActivityCompat.checkSelfPermission( context: this, Mani
            } catch (NumberFormatException e) {
                txtCalculatedDis.setText("Bad QR code!");
                System.err.println("Invalid latitude or longitude input: " + e.
            }
        }
    } else {
        txtCalculatedDis.setText("Invalid Value!");
    }
}
```

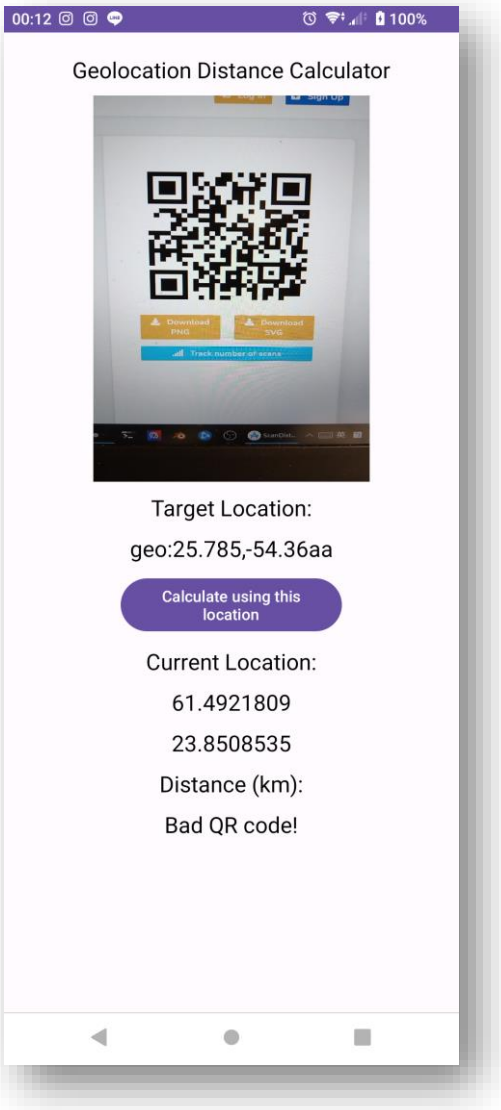
1.2 End results



(Left) Main activity, containing the button to enter the scanning functionality.
(Right) The display while there's no QR code detected by the camera.



(Left) The content of the QR code doesn't fit the format of a geo:URI (do not begin with "geo:"), thus the button will show that the code is not valid.
(Right) The content of the QR code contained proper geo:URI information and the calculation can be done with pressing the button.



In the case of the code contains the right format (begins with geo:), but the numerical section of the data can’t be parsed successfully (containing non-numerical symbol), the calculation result will show bad input.

1.3 Reflects

It has been a while since the last time I work with android studio and on mobile application development. There were some catching up to do at the beginning and I had some struggle dealing with what to run onCreate, onStart, and onResume etc., there are for sure plenty of things to improve on this seemingly simple project, but I felt through on hand practice and try and error process, I understood more about gaining user permission, permission verification timing and their effects on activities.

Sources used

1. QR Code Scanner - Barcode Scanner for Android. https://www.digitalocean.com/community/tutorials/qr-code-barcode-scanner-android
2. User Location. https://developer.android.com/training/location
3. geo URI scheme. https://en.wikipedia.org/wiki/Geo_URI_scheme
4. How to get and use location data in your Android app. https://www.androidauthority.com/get-use-location-data-android-app-625012/