Networking In Android

- Android lets your application connect to the internet or any other local network and allows you to perform network operations.
- Before you add networking functionality to your app, you need to ensure that data and information within your app stays safe when transmitting it over a network. To do so, follow these networking security best practices:
    - Minimize the amount of sensitive or personal user data that you transmit over the network.
    - Send all network traffic from your app over SSL.
    - Consider creating a network security configuration, which allows your app to trust custom CAs or restrict the set of system CAs that it trusts for secure communication.
- Most network-connected Android apps use HTTP to send and receive data. The Android platform includes the `HttpsURLConnection` client.

Permissions

- In order to connect to the network, your app must have the following permissions in the manifest file:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
```

Checking the network:

- Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c. For this android provides **ConnectivityManager** class.

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

- Then, use **getAllNetworkInfo** to get information about all the networks.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

- The last thing you need to do is to check **Connected State** of the network.

```
for (int i = 0; i<info.length; i++){
   if (info[i].getState() == NetworkInfo.State.CONNECTED){
      Toast.makeText(context, "Internet is connected
      Toast.LENGTH_SHORT).show();
   }
}
```

Accessing the network

- After checking that you are connected to the internet, you can perform any network operation. Here we are fetching the html of a website from a url.
- Android provides **HttpURLConnection** and **URL** class to handle these operations. You need to instantiate an object of URL class by providing the link of website.

  This exactly the same as URL in java, hence the full explanation is not given.

```
String link = "http://www.google.com";
URL url = new URL(link);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.connect();

InputStream is = conn.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(is,
"UTF-8"));
String webPage = "",data="";

while ((data = reader.readLine()) != null){
   webPage += data + "\n";
}
```

Sending Email

- One can send emails on Android using implicit intents.
- Implicit intents broadcast to all installed apps on the phone, the *intention* to perform a certain action.
- You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

- To send an email you need to
    - use setData() method to specify **mailto:** as URI
    - use setType() method to set data type to **text/plain**
- Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client.

```
1  emailIntent.putExtra(Intent.EXTRA_EMAIL   , new String[]{"Recipient"});
2  emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject");
3  emailIntent.putExtra(Intent.EXTRA_TEXT    , "Message Body");
```

Location Services

Displaying Maps

- In order to display maps, we must use the Google Play Services API.
- To do that, first, download and enable the Google Play Services API in your Android SDK manager.
- Then, visit the Google Maps API website, login, and obtain an API key.
- Then, add the Google Maps fragmen to your activity

```
1  <fragment
2     android:id="@+id/map"
3     android:name="com.google.android.gms.maps.MapFragment"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"/>
```

- Thereafter, specify the required permissions in the manifest file:

```
1  <!--Permissions-->
2
3  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
   />
4  <uses-permission android:name="android.permission.INTERNET" />
5  <uses-permission
   android:name="com.google.android.providers.gsf.permission.
6     READ_GSERVICES" />
7  <uses-permission
   android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- Then add the API key to the manifest as well.

```
<!--Google MAP API key-->

<meta-data
   android:name="com.google.android.maps.v2.API_KEY"
   android:value="AIzaSyDKymeBXNeiFWY5jRUejv6zItpmr2MVyQ0" />
```

- Other optional customizations:

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
//SET MAP TYPE

googleMap.getUiSettings().setZoomGesturesEnabled(true);
//ENABLE OR DISABLE ZOOM

final LatLng ek_jagah = new LatLng(21 , 57);
Marker TP = googleMap.addMarker(new MarkerOptions()
   .position(ek_jagah).title("Yeh Ek Jagah Hai"));
//ADD A MARKER TO THE MAP
```

Getting Location

- To get location, we must declare permissions in the manifest.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
```

- Then, in the java code, create a LocationManager

```
LocationManager locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
```

- Then, request the current co-ordinates.

```
LocationListener locationListener = new MyLocationListener();
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
5000, 10, locationListener);
```

- Then, create a Listener class to retrieve the information.

```
private class MyLocationListener implements LocationListener {

    @Override
    public void onLocationChanged(Location loc) {
      //THIS METHOD IS CALLED WHEN LOCATION IS CHANGED


        Toast.makeText(
                getBaseContext(),
                "Location changed: Lat: " + loc.getLatitude() + " Lng:
"
                    + loc.getLongitude(), Toast.LENGTH_SHORT).show();


    }
}
```

- This method can be used to retrieve co-ordinates using GPS.
- For this, location services needs to enabled on the Android device.
- Furthermore, on later android versions, the app needs to be granted the Location permission to use the GPS radio.

Geocoder

- **Geocoding** is the process of **transforming** a **street address** or other description of a location **into a (latitude, longitude) coordinate**.
- **Reverse geocoding** is the process of transforming a **(latitude, longitude) coordinate** into a **(partial) address**.
- The amount of detail in a reverse geocoded location description may vary,
    - For example one might contain the full street address of the closest building, while another might contain only a city name and postal code.
- The Geocoder class requires a backend service that is not included in the core android framework.

```
import android.location.Geocoder;
```

- To use geocoding, the following code can be used:

```
Geocoder gc = new Geocoder(context);
  if(gc.isPresent()){

    List<Address> list =
      gc.getFromLocationName(
      “RD National College, Waterfield Road, Bandra, Mumbai, India”,
1);

    Address address = list.get(0);
    double lat = address.getLatitude();
    double lng = address.getLongitude();
}
```

- To use reverse GeoCoding, use the following code:

```
Geocoder gc = new Geocoder(context);
  if(gc.isPresent()){
    List<address> list = gc.getFromLocation(37.42279, -122.08506,1);
    Address address = list.get(0);
    StringBuffer str = new StringBuffer();
    str.append(“Name: ” + address.getLocality() + “\n”);
    str.append(“Sub-Admin Ares: ” + address.getSubAdminArea() + “\n”);
    str.append(“Admin Area: ” + address.getAdminArea() + “\n”);
    str.append(“Country: ” + address.getCountryName() + “\n”);
    str.append(“Country Code: ” + address.getCountryCode() + “\n”);
    String strAddress = str.toString();
}
```

Using Multimedia

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using `MediaPlayer` APIs.

Playing Audio

- To play audio, you must first create an object of the MediaPlayer class...

```
MediaPlayer mp = new MediaPlayer();
```

- Set the file path, prepare the source 9decoding, buffering, etc), and then start playing!

```
try {
        mp.setDataSource(path + File.separator + fileName);
        mp.prepare();
        mp.start();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

Recording Audio

- To record audio, one must first declare the following permission:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

- Then, create a MediaRecorder object

```
MediaRecorder myAudioRecorder = new MediaRecorder();
```

- The following parameters of the object need to be set:

```
myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
//MIC SE RECORD KARNA HAI

myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
//.3GP FILE BANEGI

myAudioRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
//SYNTAX HAI. KARNA PADTA HAI.

myAudioRecorder.setOutputFile(outputFile);
//KONSI FILE MEIN SAVE HOGA. THE PARAMETER IS A JAVA File OBJECT
```

- Once set, we can prepare the hardware to record audio, and then start recording

```
myAudioRecorder.prepare();
myAudioRecorder.start();
```

- Had enough? Stop the recording!

```
mediaRecorder.stop();
```

> *Unit 3 is currently incomplete. Started with Unit IV as only few topics are left in this unit.*