CLASS OF 2021

RF Coverage Insights

JUNE 30, 2021

Table OF Content

Abstract

Traditional drive testing tools have been expensive and time consuming to perform RCA (Root Cause Analysis). Hence, telecom industry needs cost-effective solutions like App based drive testing to meet its mobile network testing objectives and fulfill users' expectations. We introduced Android based smartphone app and Web platform to assist testers monitor signaling traffic at control points and then centrally examine to get valuable insights and refine overall network health. To perform such action, telecom operators need to undergo drive test analysis as per the specific network requirements. The app can easily process thousands of data to conduct mass drive testing.

Problem with drive test our project solve it :

- **Time:**

  Drive test takes a long time to measure the parameters because they go to all places by themselves by car.

- **Money:**

  it also costs a lot of money because they need cars to go to all places, devices and drive testers salaries, etc.

  our project they do not need for all that because the users will help them to get these parameters in their places.

Our Project Consist of:

- **Android Application:**

  The application collects different data, such as: cell id, cell type, MCC, MNC, IMEI, IMSI, device model, location, Country, Operator, and Signal strength from the phone, display it, and send it to Database through Web Service.

- **Web Platform:**

  This platform takes the data stored in the Database through the Android application and display it on a map via Web Service and also display the results of some analyzes on a bar chart, one of them displays the average network strength for different operators and the other shows the average signal strength for different types of phones with different operators.

Introduction

Everyone uses his mobile phone every day without thinking about the used technology or the generation or even without thinking about the service(s) he gets from the telecommunication service provider, in this time while the world witnesses a great development in the field of 5G deployment, and the service providers and the industrial departments are racing to provide the most reliable and satisfying service, should users consider knowing about the development in telecom sector.

In the early appearance of mobile telecommunication services, no one was being interested to know about the technology , because there isn't much to say about it more the it was exactly the same as fixed phone except it was mobile, but now with this great development in the technology that make the mobile phone provides countless services thus we become not able to call it a phone any more, people start to think about the technology, services, and even about their cyber security, mobile generations have based along a long journey of development which made the mind unable to realize all of these services and enhancements.

This industry started with what's called the 0G which provided a very heavy and inefficient phone. This phone can be used only in a car because it was so heavy, but it was a revolutionary one at that time. The development started with the 1st generation which provided light phones compared to the 0G. this generation used the Frequency Division Multiple Access (FDMA), which assign a specified range of frequency for each user. This wasn't of course an efficient way to use the frequency spectrum anyway. So, operators had to develop the 2nd generation which use a different access technology Time Division Multiple Access (TDMA). This was the soar of mobile technologies as we knew today because of using completely new technologies like TDMA instead of FDMA also using digital signals instead of analog signals. In the 2nd generation new services had been appeared like SMS and the mobile internet thanks to new technologies such as EDGE and GPRS, but it was very slow, and operators needed to improve it. The 3rd generation or UMTS appeared to improve the internet speed in 2nd generation, and also uses a new access technique Code Division Multiple Access (CDMA). The 4th generation also significantly improve the data rates, but it also aimed at providing new services.

As Known, the mobile signals are transmitted as Electromagnetic waves which can be blocked for many reasons, what makes mobile operators in a big contest to provide the most reliable and stable signal coverage for users. Some of reasons that can block the Electromagnetic wave signals are interference or jamming. Or even normal reasons of the Electromagnetic waves propagation like scattering, reflection or refraction, but let's look at some of those challenges in some of details.

Interference

All electromagnetic waves can be superimposed upon each other without limit. The electric and magnetic fields simply add at each point. If two waves with the same frequency are combined there will a be a constant interference pattern caused by their superposition. Interference can either be constructive, meaning the strength increases as result, or destructive where the strength is reduced.
The amount of interference depends on the phase difference at a particular point. It can be shown that constructive interference occurs for phase differences of 0-1200, and 240-3600. Thus, destructive interference occurs from 120-2400. For two identical waves, no phase shift results in total constructive interference, where the strength is maximum and 1800 phase shift will create total destructive interference (no signal at all).

Multipath Fading

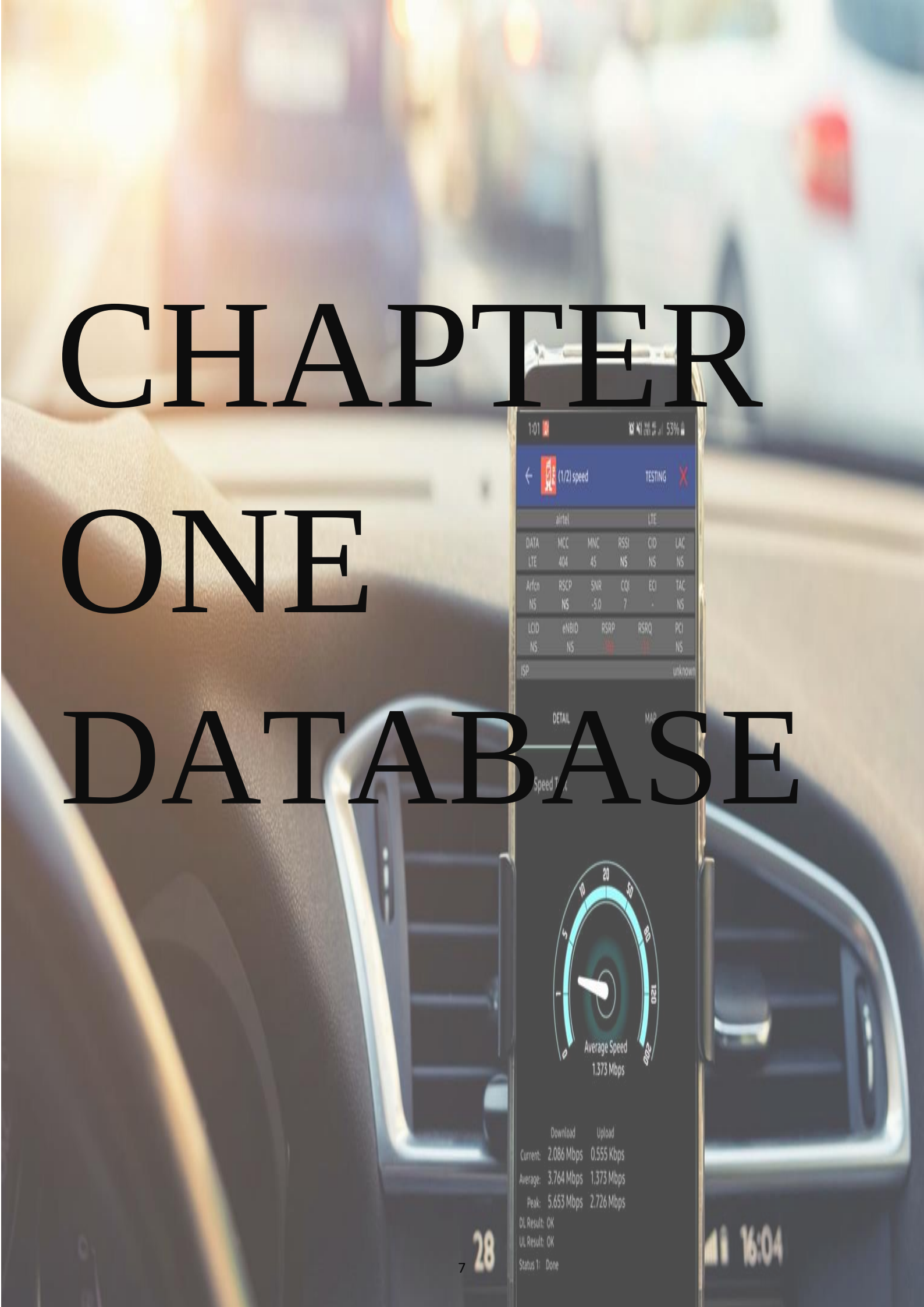
Multipath fading is a feature that needs to be taken into account when designing or developing a radio communications system. In any terrestrial radio communications system, the signal will reach the receiver not only via the direct path, but also as a result of reflections from objects such as buildings, hills, ground, water, etc that are adjacent to the main path.

The overall signal at the radio receiver is a summation of the variety of signals being received. As they all have different path lengths, the signals will add and subtract from the total dependent upon their relative phases.

At times there will be changes in the relative path lengths. This could result from either the radio transmitter or receiver moving, or any of the objects that provides a reflective surface moving. This will result in the phases of the signals arriving at the receiver changing, and in turn this will result in the signal strength varying as a result of the different way in which the signals will sum together. It is this that causes the fading that is present on many signals.

Shadow Fading

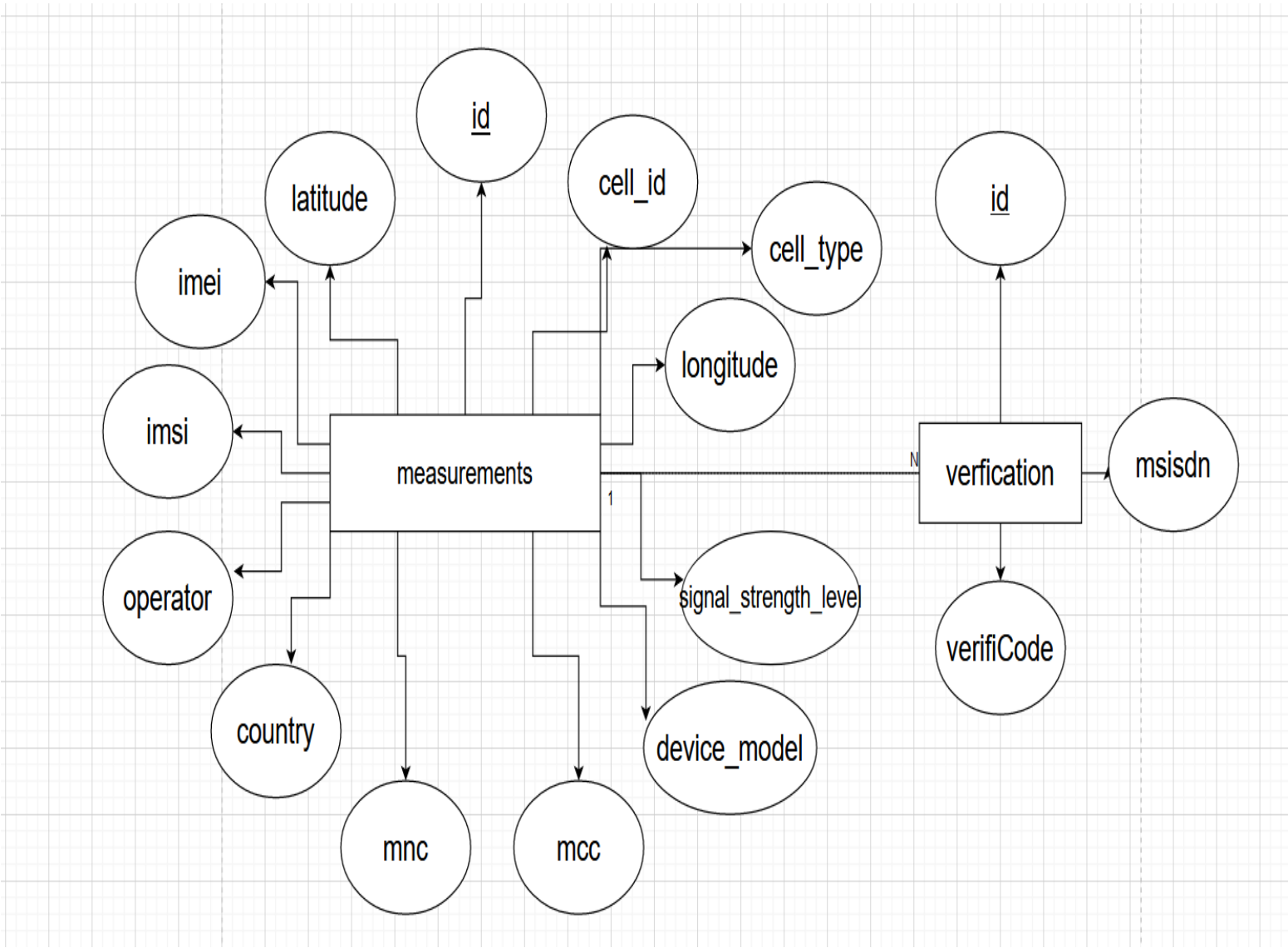
Shadow fading is the large-scale fluctuation of the signal envelope due to large (with respect to a wavelength) objects obstructing the propagation paths between the transmitter and the receiver. The shadowing obstacle absorbs and/or blocks the transmitted signal and determines the relatively slow signal variations with respect to the nominal value given by path loss models. Hence the shadowing characterization results are usually used to adjust link budget path loss calculations for statistical coverage estimates in wireless network planning. An effect analogous to obstruction can also be observed in some channels with a line-of-sight (LOS) component, in which multiple reflections can induce relatively slow variation (in addition to the more rapid small-scale fading.



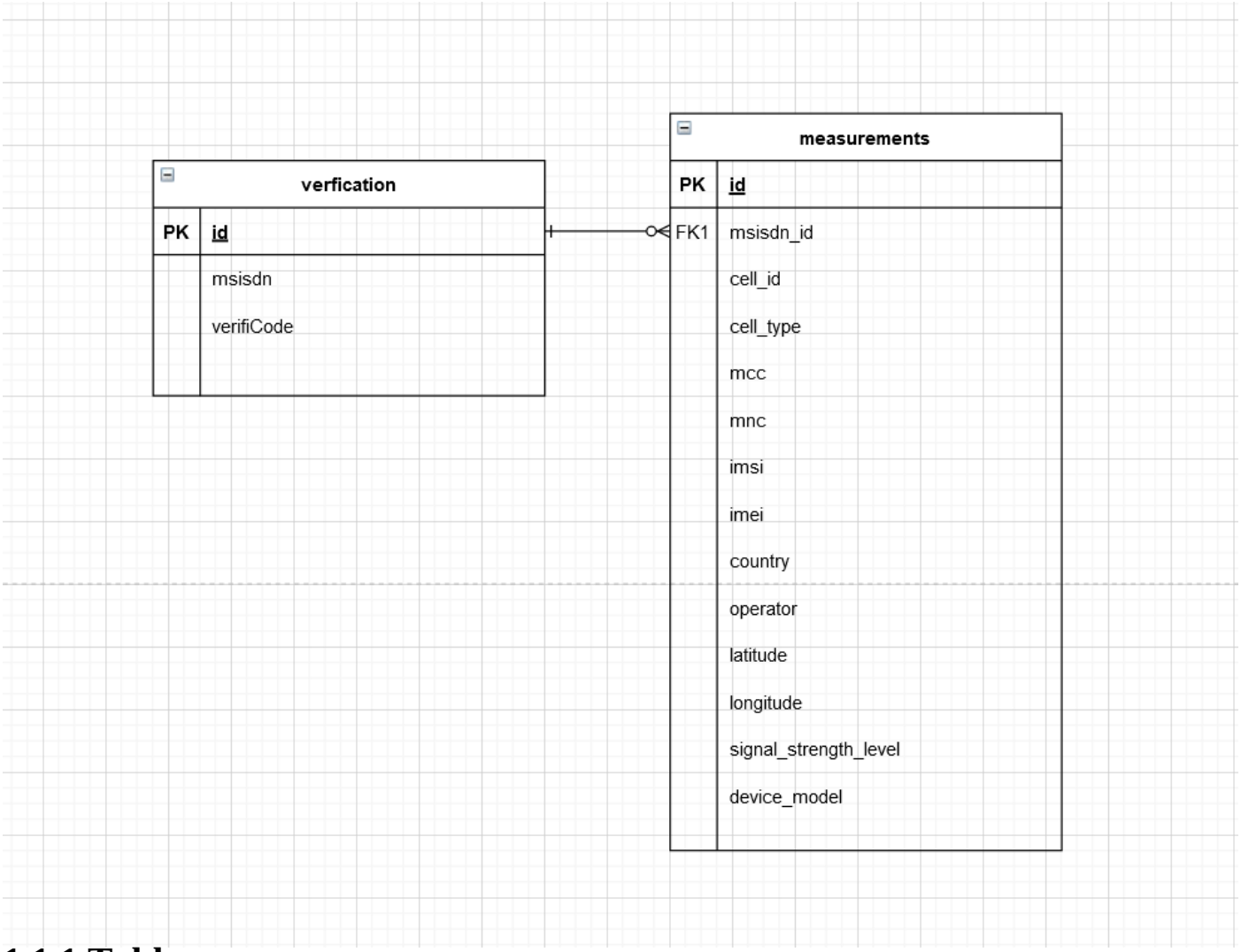
CHAPTER ONE DATABASE

CH01: DATABASE

1.0 ERD:



1.1 Schema:



1.1.1 Tables:

1.1.2 Table Measurements:

Data Output | Explain | Messages | Notifications

cell_id text	cell_type text	country text	id [PK] integer	imei text	imsi text	latitude numeric	longitude numeric	mcc text	mnc text	operator text	signal_strength_level integer	device_model text	msisdn_id integer

1.1.3 Table Verification:

id [PK] integer	msisdn text	verifiCode text

CHAPTER Two Android App

CH02: Android Application

2.0 What is Android:

Android is a complete set of software for mobile devices such as tablet computers, notebooks, smartphones, electronic book readers, set-top boxes etc.

It contains a Linux-based Operating System, middleware, and key mobile applications.

It can be thought of as a mobile operating system. But it is not limited to mobile only. It is currently used in various devices such as mobiles, tablets, televisions etc.

It is developed by Google and later the OHA (Open Handset Alliance). Java language is mainly used to write the android code even though other languages can be used like Kotlin.

2.1 Features of Android:

1) It is open source.

2) Anyone can customize the Android Platform.

3) There are a lot of mobile applications that can be chosen by the consumer.

4) It provides many interesting features like weather details, opening screen, live RSS (Really Simple Syndication) feeds etc.

2.2 Categories of Android applications:

- Entertainment
- Tools
- Communication
- Productivity
- Personalization
- Music and Audio
- Social
- Media and Video
- Travel and Local etc.

2.3 Android Architecture:

android architecture or Android software stack is categorized into five parts**:**

- Linux kernel.
- native libraries (middleware).
- Android Runtime.
- Application Framework.
- Applications.

2.3.1 Linux kernel:

It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access.

2.3.2 Native Libraries:

On the top of Linux kernel, there are Native libraries such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc. The WebKit library is responsible for browser support, SQLite is for database, FreeType for font support, Media for playing and recording audio and video formats.

2.3.3 Android Runtime:

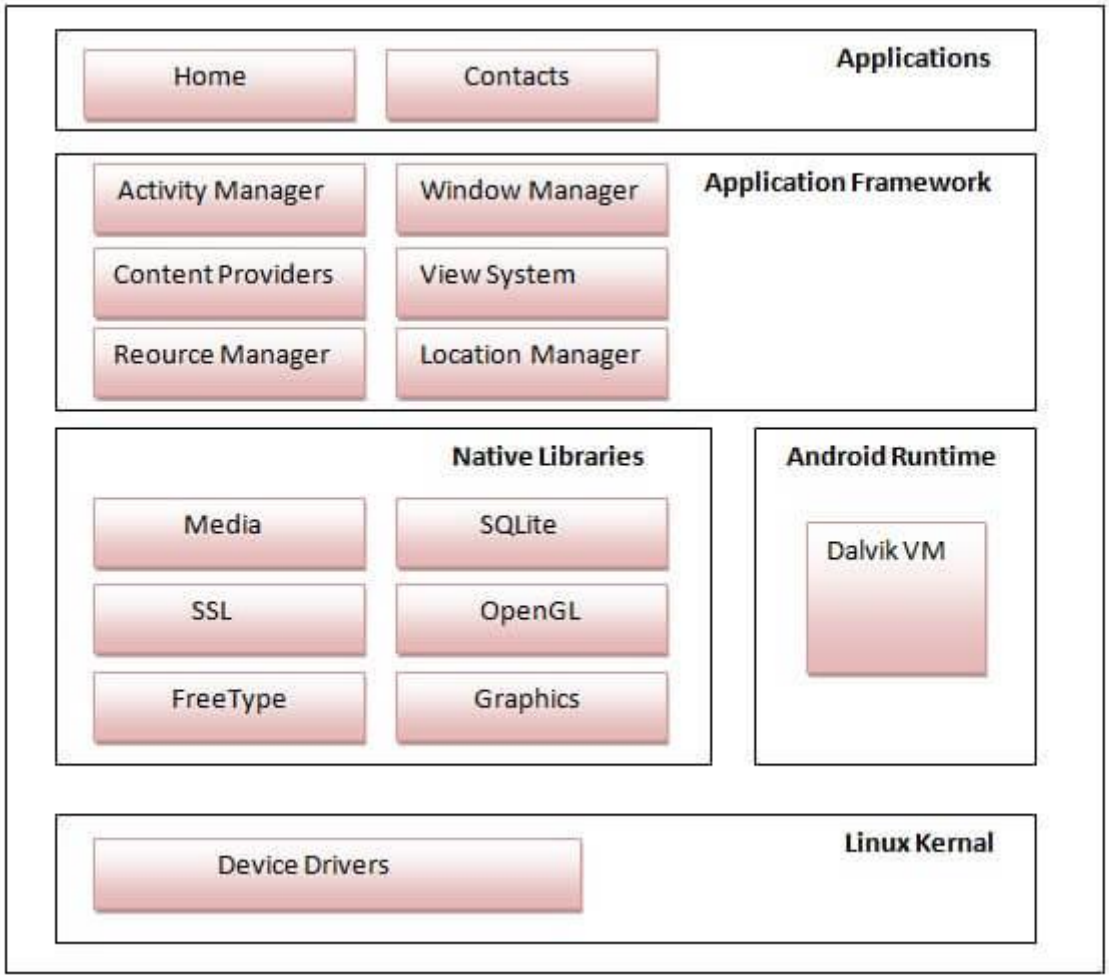
In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM, but it is optimized for mobile devices. It consumes less memory and provides fast performance.

2.3.4 Android Framework:

On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

2.3.5 Applications:

On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using Linux kernel.



2.4Android Core Building Blocks:

An android component is simply a piece of code that has a well defined life cycle e.g. Activity, Receiver, Service etc. The core building blocks, or fundamental components of android are activities, views, intents, services, content providers, fragments and AndroidManifest.xml.

2.4.1 Activity

An activity is a class that represents a single screen. It is like a Frame in AWT.

2.4.2 View:

A view is the UI element such as button, label, text field etc. Anything that you see is a view.

2.4.3 Intent:

Intent is used to invoke components. It is mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

2.4.4 Service:

Service is a background process that can run for a long time. There are two types of services local and remote. Local service is accessed from within the application whereas remote service is accessed remotely from other applications running on the same device.

2.4.5 Content Provider:

Content Providers are used to share data between the applications.

2.4.6 Fragment:

Fragments are like parts of activity. An activity can display one or more fragments on the screen at the same time.

2.4.7 AndroidManifest.xml:

It contains information about activities, content providers, permissions etc. It is like the web.xml file in Java EE.

2.4.8 Android Virtual Device (AVD)

It is used to test the android application without the need for mobile or tablet etc. It can be created in different configurations to emulate different types of real devices.

2.5Activity Lifecycle:

As a user navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle. The Activity class provides a number of callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.

the Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). The system invokes each of these callbacks as an activity enters a new state.

2.5.1 Lifecycle callbacks:

2.5.2 onCreate():

You must implement this callback, which fires when the system first creates the activity. On activity creation, the activity enters the *Created* state. In the onCreate() method, you perform basic application startup logic that should happen only once for the entire life of the activity. For example, your implementation of onCreate() might bind data to lists, associate the activity with a ViewModel, and instantiate some class-scope variables. This method receives the parameter savedInstanceState, which is a Bundle object containing the activity's previously saved state. If the activity has never existed before, the value of the Bundle object is null.

2.5.3 onStart()

When the activity enters the Started state, the system invokes this callback. The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. For example, this method is where the app initializes the code that maintains the UI. When the activity moves to the started state, any lifecycle-aware component tied to the activity's lifecycle will receive the ON_START event. The onStart() method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the activity enters the *Resumed* state, and the system invokes the onResume() method.

2.5.4 onResume()

When an interruptive event occurs, the activity enters the *Paused* state, and the system invokes the onPause() callback. If the activity returns to the Resumed state from the Paused state, the system once again calls onResume() method. For this reason, you should implement onResume() to initialize components that you release during onPause() and perform any other initializations that must occur each time the activity enters the Resumed state.

2.5.5 onPause()

The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode). Use the onPause() method to pause or adjust operations that should not continue (or should continue in moderation) while the Activity is in the Paused state, and that you expect to resume shortly. There are several reasons why an activity may enter this state. For example:

- Some event interrupts app execution, as described in the onResume() section. This is the most common case.
- In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system pauses all of the other apps.
- A new, semi-transparent activity (such as a dialog) opens. As long as the activity is still partially visible but not in focus, it remains paused.

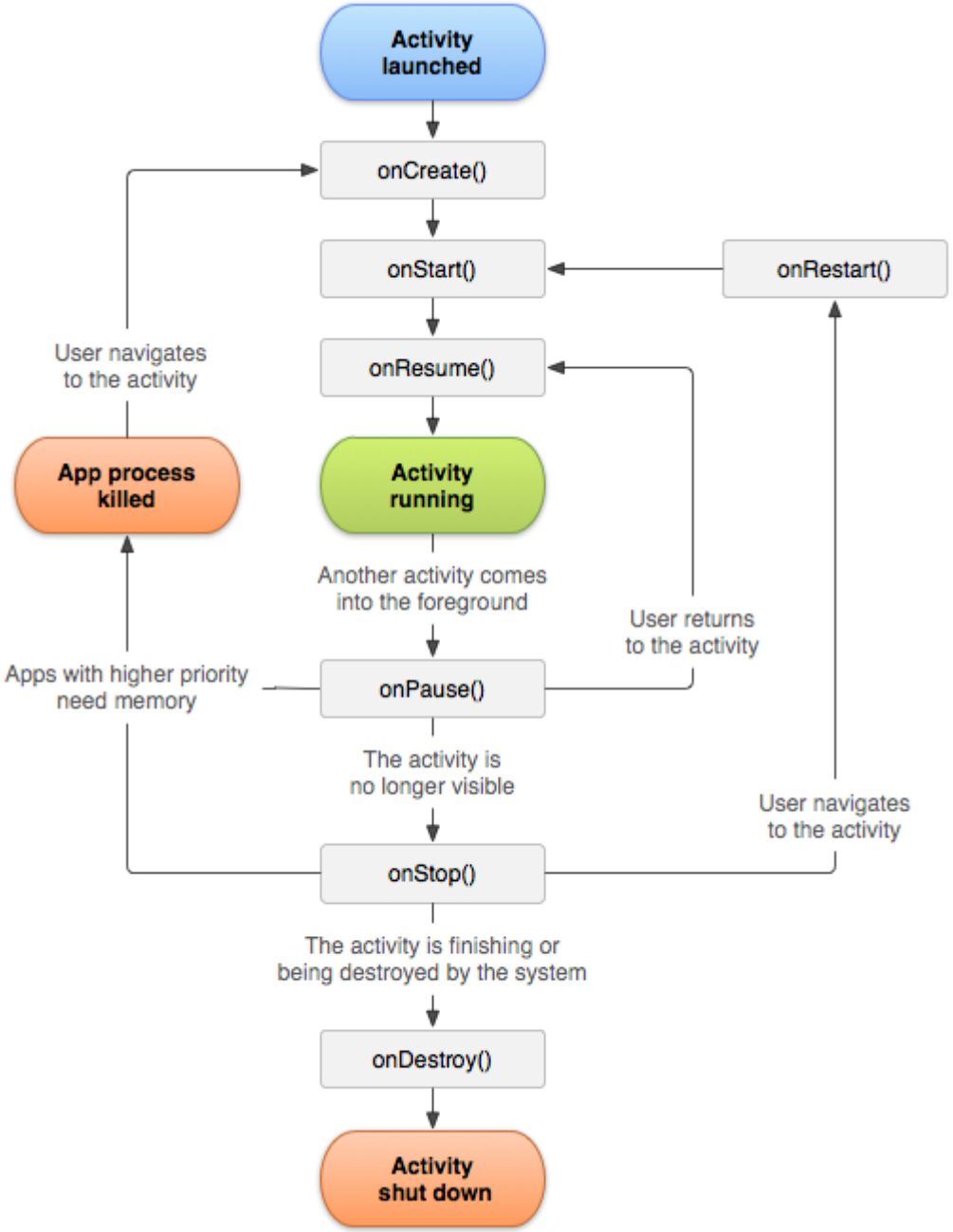
2.5.6 onStop():

When your activity is no longer visible to the user, it has entered the *Stopped* state, and the system invokes the onStop() callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call onStop() when the activity has finished running and is about to be terminated.

2.5.7 onDestroy():

onDestroy() is called before the activity is destroyed. The system invokes this callback either because:

1. the activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity), or
2. the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode).



2.6 Device Information:

2.6.1 International Mobile Station Equipment Identity (IMEI):

The International Mobile Station Equipment Identity (IMEI) looks more like a serial number which distinctively identifies a mobile station internationally. This is allocated by the equipment manufacturer and registered by the network operator, who stores it in the Equipment Identity Register (EIR). By means of IMEI, one recognizes obsolete, stolen, or non-functional equipment.

Following are the parts of IMEI:

- **Type Approval Code (TAC)** − 6 decimal places, centrally assigned.
- **Final Assembly Code (FAC)** − 6 decimal places, assigned by the manufacturer.
- **Serial Number (SNR)** − 6 decimal places, assigned by the manufacturer.
- **Spare (SP)** − 1 decimal place.

Thus, IMEI = TAC + FAC + SNR + SP. It uniquely characterizes a mobile station and gives clues about the manufacturer and the date of manufacturing.

2.6.2 Phone Type:

Returns a constant indicating the device phone type. This indicates the type of radio used to transmit voice calls.

Possible phone types:

- PHONE_TYPE_NONE
- PHONE_TYPE_GSM
- PHONE_TYPE_CDMA
- PHONE_TYPE_SIP

2.7 Network Info:

2.7.1 Signal Strength:

Decibels offer a more accurate and useful measure of cell phone signal strength. Signal strength for cellphones is calculated using dBm (or decibel milliwatts) as its standard unit of measure. On a signal strength meter, dBm is typically expressed as a negative number, such as -88. The closer to zero the dBm reading is, the stronger the cell phone signal.

- **Close to no signal strength**= -110 dBm
- **Poor signal strength** = -85 dBm to -100 dBm
- **Good signal strength** = -65 dBm to -84 dBm

- **Excellent signal strength** = -64 dBm to -50 dBm

2.5.2 **Network Type:**

A constant indicating the radio technology (network type) currently in use on the device for data transmission. If this object has been created with createForSubscriptionId(int), applies to the given sub id. Otherwise, applies
to SubscriptionManager#getDefaultDataSubscriptionId() Requires
Permission: READ_PHONE_STATE or that the calling app has carrier privileges
(see hasCarrierPrivileges()). Requires Manifest.permission.READ_PHONE_STATE

Possible network types:

- NETWORK_TYPE_UNKNOWN
- NETWORK_TYPE_GPRS

- NETWORK_TYPE_EDGE
- NETWORK_TYPE_UMTS
- NETWORK_TYPE_HSDPA
- NETWORK_TYPE_HSUPA
- NETWORK_TYPE_HSPA

- NETWORK_TYPE_CDMA
- NETWORK_TYPE_EVDO_0
- NETWORK_TYPE_EVDO_A
- NETWORK_TYPE_EVDO_B
- NETWORK_TYPE_1xRTT

- NETWORK_TYPE_IDEN
- NETWORK_TYPE_LTE
- NETWORK_TYPE_EHRPD
- NETWORK_TYPE_HSPAP
- NETWORK_TYPE_NR

2.8 SIM Card Information:

2.8.1 International Mobile Subscriber Identity (IMSI):

Every registered user has an original International Mobile Subscriber Identity (IMSI) with a valid IMEI stored in their Subscriber Identity Module (SIM). IMSI comprises of the following parts :

- **Mobile Country Code (MCC)** − 3 decimal places, internationally standardized.
- **Mobile Network Code (MNC)** − 2 decimal places, for unique identification of mobile network within the country.
- **Mobile Subscriber Identification Number (MSIN)** − Maximum 10 decimal places, identification number of the subscriber in the home mobile network.
- **Mobile Country Code (MCC)** − 3 decimal places, internationally standardized.
- **Mobile Network Code (MNC)** − 2 decimal places, for unique identification of mobile network within the country.

2.9 Android TelephonyManager:

Provides access to information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information. Applications can also register a listener to receive notification of telephony state changes.

Note that access to some telephony information is permission-protected. Your application cannot access the protected information unless it has the appropriate permissions declared in its manifest file. Where permissions apply, they are noted in the methods through which you access the protected information.

2.9.1 Method we use it :

- **getImei():**

Returns the IMEI (International Mobile Equipment Identity). Return null if IMEI is not available. See getImei(int) for details on the required permissions and behavior when the caller does not hold sufficient permissions.
Requires android.Manifest.permission.READ_PRIVILEGED_PHONE_STATE.

- **createForSubscriptionId(int subId):**

  Create a new TelephonyManager object pinned to the given subscription ID.

- **isNetworkRoaming ():**

  Returns true if the device is considered roaming on the current network, for GSM purposes. Availability: Only when user registered to a network.

2.10 SubscriptionManager:

SubscriptionManager is the application interface to SubscriptionController and provides information about the current Telephony Subscriptions.

2.10.1 Method we use it :

- **getSubscriptionIds (int slotIndex):**
  Get an array of Subscription Ids for specified slot Index. Return subscription Ids or null if the given slot Index is not valid or there are no active subscriptions in the slot.
- **getActiveSubscriptionInfoForSimSlotIndex (int slotIndex):**

  Get the active SubscriptionInfo associated with the slotIndex. Requires Permission: READ_PHONE_STATE or that the calling app has carrier privileges (see TelephonyManager#hasCarrierPrivileges). Requires Manifest.permission.READ_PHONE_STATE.

2.11 Build:

Information about the current build, extracted from system properties.

2.11.1 Method we use it :

- **MANUFACTURER:**

  The manufacturer of the product/hardware.

- **MODEL:**

  The end-user-visible name for the end product.

- **CPU_ABI:**

  The name of the instruction set (CPU type + ABI convention) of native code.

- **Build.VERSION.RELEASE:**

  The user-visible version string. E.g., "1.0" or "3.4b5" or "bananas". This field is an opaque string. Do not assume that its value has any particular structure or that values of RELEASE from different releases can be somehow ordered.

- **Build.VERSION.SDK:**

  The user-visible SDK version of the framework in its raw String representation; use SDK_INT instead.

2.12 Android Web Service:

2.12.1 HttpURLConnection:

A URLConnection with support for HTTP-specific features. See the spec for details.

Uses of this class follow a pattern:

1. Obtain a new HttpURLConnection by calling URL.openConnection() and casting the result to HttpURLConnection.
2. Prepare the request. The primary property of a request is its URI. Request headers may also include metadata such as credentials, preferred content types, and session cookies.
3. Optionally upload a request body. Instances must be configured with setDoOutput(true) if they include a request body. Transmit data by writing to the stream returned by URLConnection.getOutputStream().
4. Read the response. Response headers typically include metadata such as the response body's content type and length, modified dates and session cookies. The response body may be read from the stream returned by URLConnection.getInputStream(). If the response has no body, that method returns an empty stream.
5. Disconnect. Once the response body has been read, the HttpURLConnection should be closed by calling disconnect(). Disconnecting releases, the resources held by a connection so they may be closed or reused.

- **Posting Content**

  To upload data to a web server, configure the connection for output using setDoOutput(true). For best performance, you should call either setFixedLengthStreamingMode(int) when the body length is known in advance, or setChunkedStreamingMode(int) when it is not. Otherwise HttpURLConnection will be forced to buffer the complete request body in memory before it is transmitted, wasting (and possibly exhausting) heap, and increasing latency.

2.12.2 Volley:

Volley is an HTTP library that makes networking very easy and fast, for Android apps. It was developed by Google and introduced during Google I/O 2013. It was developed because there is an absence in Android SDK, of a networking class capable of working without interfering with the user experience. Although Volley is a part of the Android Open-Source Project(AOSP), Google announced in January 2017 that Volley will move to a standalone library. It manages the processing and caching of network requests, and it saves developers valuable time from writing the same network call/cache code again and again.

- **Features of Volley:**

1. Request queuing and prioritization
2. Effective request cache and memory management
3. Extensibility and customization of the library to our needs
4. Cancelling the requests

- **Classes in Volley Library:**

  Volley has two main classes:

1. **Request Queue:** It is the interest one uses for dispatching requests to the network. One can make a request queue on demand if required, but typically it is created early on, at startup time, and keep it around and use it as a Singleton.
2. **Request:** All the necessary information for making web API call is stored in it. It is the base for creating network requests(GET, POST).

- **Usage of Volley in the project:**

  At the first time the user installs the application, we use Volley to send Post requests from the android application to the web service and get the response from the web service the first one when the user enters the number we send the phone number to the web service to connect with Twilio and generate the code that sends to the user in SMS and save it in the database, the second one sends the verification code to the web service to compare it with the saved in the database and send a response if the same or not this process happens once.

2.13 Android Services:

2.13.1 What are services?

A *service* is a component which runs in the background without direct interaction with the user. As the service has no user interface, it is not bound to the lifecycle of an activity. Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing, updating content providers and the like. Services run with a higher priority than inactive or invisible activities and therefore it is less likely that the Android system terminates them. Services can also be configured to be restarted if they get terminated by the Android system once sufficient system resources are available again. It is possible to assign services the same priority as foreground activities. In this case it is required to have a visible notification active for the related service. It is frequently used for services which play videos or music.

2.13.2 Communication with services:

- **Using Intent data:**

  In a simple scenario no direct communication is required. The service receives the intent data from the starting Android component and performs its work. No notification is necessary. For example, in case the service updates a content provider, the activity is notified by the content provider and no extra step in the service is necessary. This approach works for local and services running in their own process.

- **Using receiver:**

  You can use broadcasts and registered receivers for the communication. For example, your activity can register a broadcast receiver for an event and the service sends outs corresponding events. This is a very typical scenario, in which the service need to signal to the activity that his processing has finished.

- **Activity binding to local service:**

  If the service is started in the same process as the activity, the activity can directly bind to the service. This is a relatively simple and efficient way to communicate and recommended for activities which need to have a fast communication layer with the service. This approach works for local services.

- **Handler and ResultReceiver or Messenger:**

  If the service should be communicating back to the activity, it can receive an object of type Messenger via the Intent data it receives from the activity. If the Messenger is bound to a Handler in the activity, the service can send objects of type Message to the activity. A Messenger is parcelable, which means it can be passed to another process and you can use this object to send Messages to the Handler in the activity. Messenger also provides the method getBinder() which allows passing a Messenger to the activity. The activity can therefore send Messages to the service. This approach works for local services running in their own process.

- **AIDL for services in a different process:**

  To bind to a service which runs in a different process, you need to use Inter Process Communication (IPC) to community the data. To do so, you need to create a AIDL file which looks similar to a Java interface but ends with the .AIDL file extension and is only allowed to extend other AIDL files. This approach is required if you need to bind to a service running in another process, i.e., if your service is consumed by other Android applications.

2.13.3 Scheduling service:

- Options for scheduling:

  If you have a repetitive task in your Android app, you need to consider that activities and services can be terminated by the Android system to free up resources. Therefore, you cannot rely on standard Java schedule like the TimerTasks class.

  The Android system currently has two main means to schedule tasks:

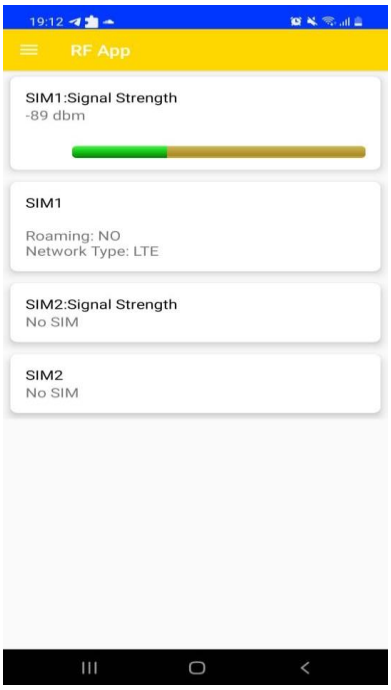
- the (outdated) AlarmManager
- the JobScheduler API.
- **The job scheduler API:**

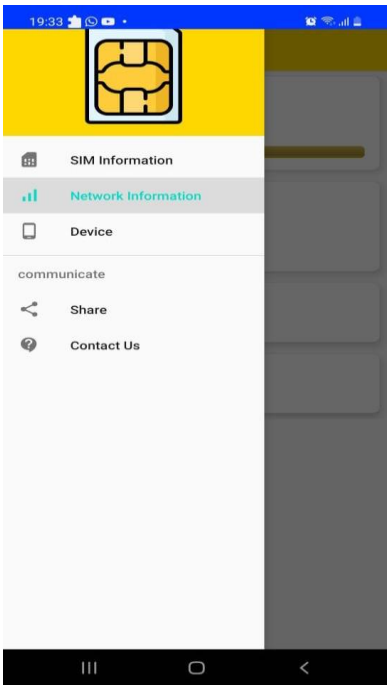
  The Android 5.0 Lollipop (API 21) release introduces a job scheduler API via the JobScheduler class. This API allows to batch jobs when the device has more resources available. In general, this API can be used to schedule everything that is not time critical for the user.

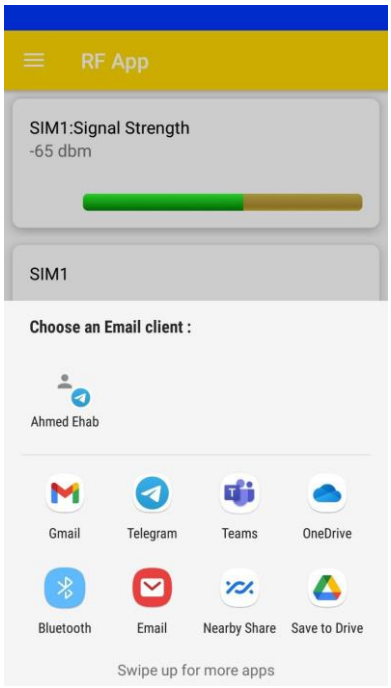
- **Advantages of the job scheduler API**

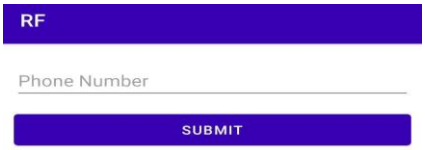
  Compared to a custom SyncAdapter or the alarm manager, the JobScheduler supports batch scheduling of jobs. The Android system can combine jobs so that battery consumption is reduced. JobManager makes handling uploads easier as it automatically handles the unreliability of the network.

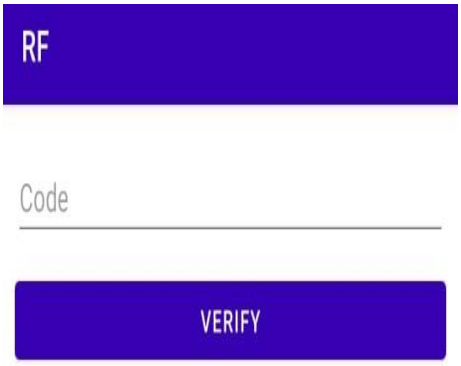
2.14 Screenshot :











CHAPTER Three Web Platform

3.1 Twilio

Twilio is an American cloud communications platform as a service (CPaaS) company based in San Francisco, California. Twilio allows software developers to programmatically make and receive phone calls, send and receive text messages, and perform other communication functions using its web service APIs.

3.1.1 Twilio Messaging API

The Twilio Messaging API makes it easy to send and receive SMS and MMS messages as well as query meta-data about text messages such as delivery status, associated media, and leverage tools like Copilot to manage your messages globally at scale.

It helps you add robust messaging capabilities to your applications. Using this REST API, you can send and receive SMS messages, track the delivery of sent messages, and retrieve and modify message history.

The API is served over HTTPS. To ensure data privacy, unencrypted HTTP is not supported.

We built API is Called ‘sendCode’ that uses Twilio Messaging API to send message contains verification code to the user, when he Entered his msisdn in Android App ,then he enter the received code to log into his account.

3.1.2 Twilio Voice API

Twilio's Voice API makes it easy to make, retrieve, control and monitor calls. Using this REST API, you can make outgoing calls, modify calls in progress, and query metadata about calls.

3.1.3 Twilio Autopilot

Build, train, and deploy AI bots, Conversational IVRs, and Alexa skills using natural language understanding and machine learning.

3.2 PositionStack API

PositionStack is a REST API that offers forward and reverse geocoding, covering more than 2 billion places and addresses worldwide. You can use this API and embed maps in your application to show their exact location. This will be useful for many businesses e.g., e-commerce, shipping etc. Even it will allow you to know about your users to meet some specific needs based on their location.

Geocoding is the process of taking input text, such as an address or the name of a place and returning a latitude/longitude location on the Earth's surface for that place.

- PositionStack API Features:
    1. **Real-time geocoding**: Geocode any global address or set of coordinates in real-time and lookup location components, country, and timezone data.
    2. **Batch Geocoding Requests**: Allows you to send batch requests to get multiple geocoding requests to the API at once.
    3. **Multi-language**: Translate the result to your preferred language by specifying the 2-letter (e.g. en) or the 3-letter code (e.g. eng) of the language.
    4. **Embeddable map URLs**: Provides an embeddable map associated with the location result.
    5. **Result formats**: API responses can be in JSON, XML, JSONP or GeoJSON formats.
    6. **Fast**: Fast API response. Average API response times range between 10ms and 100ms depending on the size of your request.
    7. **Scalable**: Powered by scalable apilayer cloud infrastructure, handling billions of geocode requests with 99.9% uptime.
    8. **Secure**: Secured API with 256-bit HTTPS Encryption. Each registered positionstack account is associated with a unique API access key, which is used to authenticate with the API when geocoding.
    9. **Customizable Fields**: you can specify preferred response fields to limit the API result and bandwidth.
    10. **Supports Multiple Languages**: API can be used in a client-side scripting language such as JavaScript, jQuery as well as server-side languages such as PHP, Python, Nodejs, Ruby, etc.

The positionstack API is easy to use. Just sign-up free and get your authentication key immediately.

Once you get your authentication key, start sending API request from your preferred language and get the response, as shown below.

```
https://api.positionstack.com/v1/forward?access_key=YOUR_ACCESS_KEY
&query=1600 Pennsylvania Ave NW, Washington DC
```

Example API Response:


```
{
    "data": {
    "results": [
    {
        "latitude": 38.897675,
        "longitude": -77.036547,
        "label": "1600 Pennsylvania Avenue NW, Washington, DC, USA",
        "name": "1600 Pennsylvania Avenue NW",
        "type": "address",
        "number": "1600",
        "street": "Pennsylvania Avenue NW",
        "postal_code": "20500",
        "confidence": 1,
        "region": "District of Columbia",
        "region_code": "DC",
        "administrative_area": null,
        "neighbourhood": "White House Grounds",
        "country": "United States",
        "country_code": "US",
        "map_url": "http://map.positionstack.com/38.897675,-77.036547"
         }
      ]
   }
}
```


We built API that takes from the user a specific address to search for this location in a map so, we use PostionStack API to get the coordinates of this address and then we can move to this location on map and know about the RF coverage in this Location.

3.3 Leaflet

Leaflet is an open source JavaScript library used to build web mapping applications. First released in 2011,it supports most mobile and desktop platforms, supporting HTML5 and CSS3. Among its users are FourSquare, Pinterest and Flickr.

Leaflet allows developers without a GIS background to very easily display tiled web maps hosted on a public server, with optional tiled overlays. It can load feature data from GeoJSON files, style it and create interactive layers, such as markers with popups when clicked.

- Usage
  A typical use of Leaflet involves binding a Leaflet "map" element to an HTML element such as a div. Layers and markers are then added to the map element.

```
// create a map in the "map" div, set the view to a given place and zoom
var map = L.map('map').setView([51.505, -0.09], 13);

// add an OpenStreetMap tile layer
// Tile Usage Policy applies: https://operations.osmfoundation.org/policies/tiles/
L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&amp;copy; <a href="http://openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);
```

- **Features**
  Leaflet supports **Web Map Service (WMS)** layers, **GeoJSON** layers, **Vector** layers and **Tile** layers natively. Many other types of layers are supported via plugins.
  Like other web map libraries, the basic display model implemented by Leaflet is one basemap, plus zero or more translucent overlays, with zero or more vector objects displayed on top.

- **Comparison with other libraries**
  Leaflet is directly comparable with OpenLayers, as both are open source, client-side only JavaScript libraries. The library as a whole is much smaller, around 7,000 lines of code compared to OpenLayers' 230,000 (as of 2015).

It has a smaller code footprint than OpenLayers (around 123 KB vs 423 KB) due partly to its modular structure. The code base is newer, and takes advantage of recent features of JavaScript, plus HTML5 and CSS3.
However, Leaflet lacks features OpenLayers supports, such as Web Feature Service (WFS) and native support for projections other than Google Web Mercator (EPSG 3857).

It is also comparable to the proprietary, closed source Google Maps API (debuting in 2005) and Bing Maps API, both of which incorporate a significant server-side component to provide services such as **geocoding**, **routing**, **search** and **integration** with features such as Google Earth.

Google Maps API provides speed and simplicity, but is not flexible, and can only be used to access Google Maps services. The new DataLayer part of Google's API does allow external data sources to be displayed, however.

A basic demo using Leaflet.

We use Leaflet library to display tiled maps with markers to obtain signal strength and RF Coverage for each Operator in various Locations.

3.4 amCharts

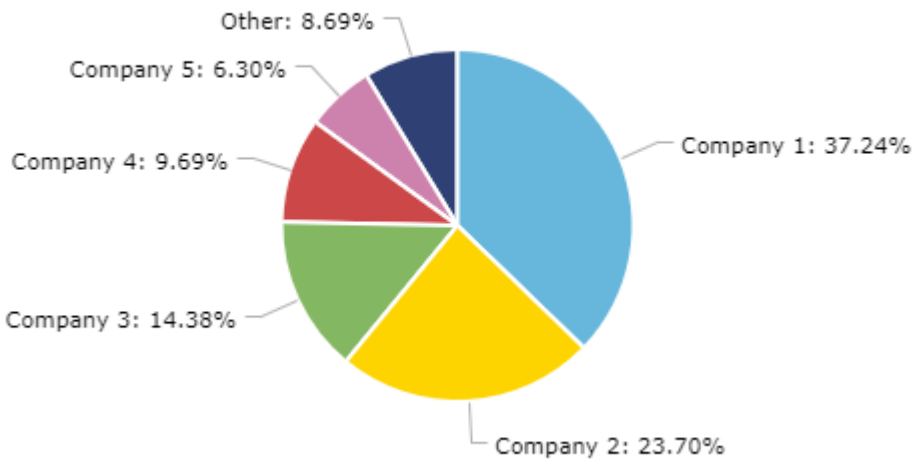
amCharts is a set of JavaScript-based data visualization libraries that includes regular chart types like Serial, Pie, etc. as well as advanced ones like Stock Charts and Maps.

- Features

  1. Free or Commercial
  2. All chart and map types you might need
  3. Works on all modern browsers and also old IE

- **Requirements**

Before we can use amCharts, we will need to include required JavaScript Libraries. There is one main library that is required for all amCharts operations - amcharts.js. It needs to be included first and is mandatory.Each chart type requires chart-type specific include. For example Serial chart, will also require serial.js, a Pie chart will need pie.js, etc. If on the same web page you will be displaying several different chart types, you will need to include all chart-type-specific includes that are being displayed on that page. Besides main required functional includes, you may need other includes, like themes and plugins.



We used amCharts Library to visualize the average Signal strength for different operator in different locations in Egypt .

Also we used it to the average signal strength for different types of mobile phones Like (Samsung, xiaomi,….etc) with different operators

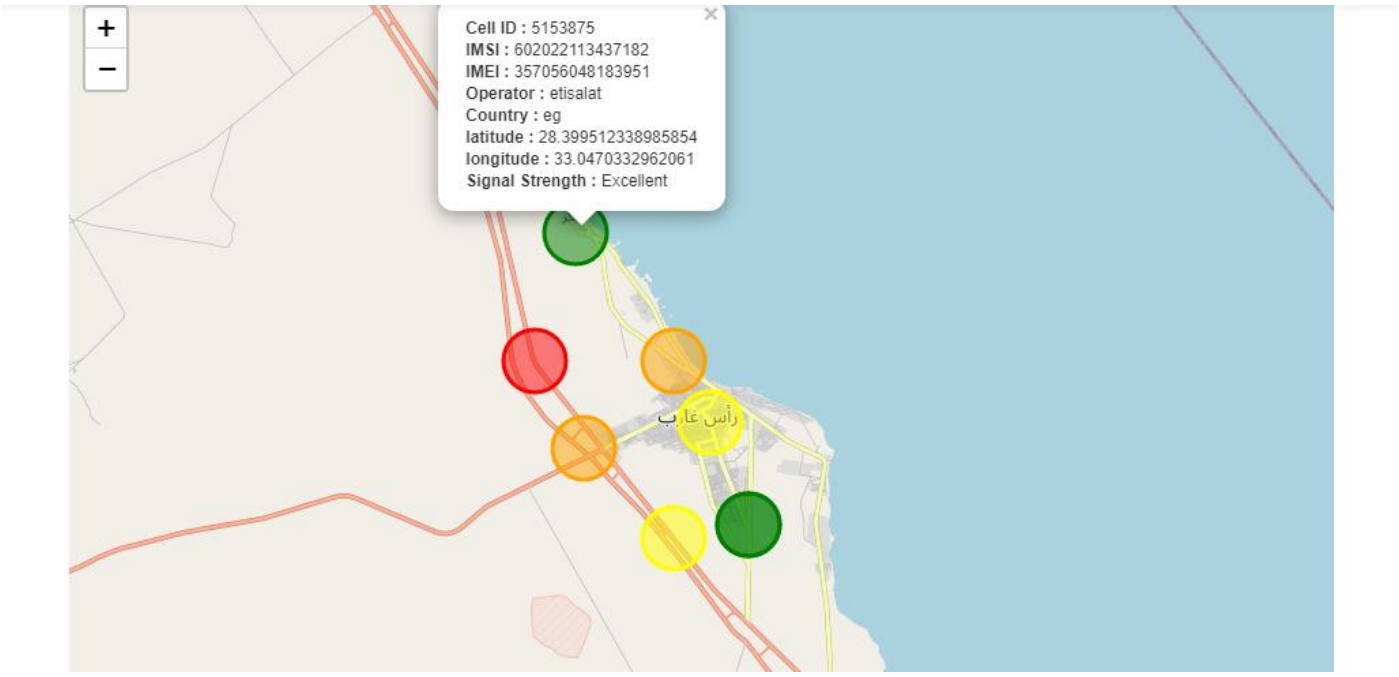
3.5 Web Service

We built many APIs by using Java Restful web Service to perform different functions .

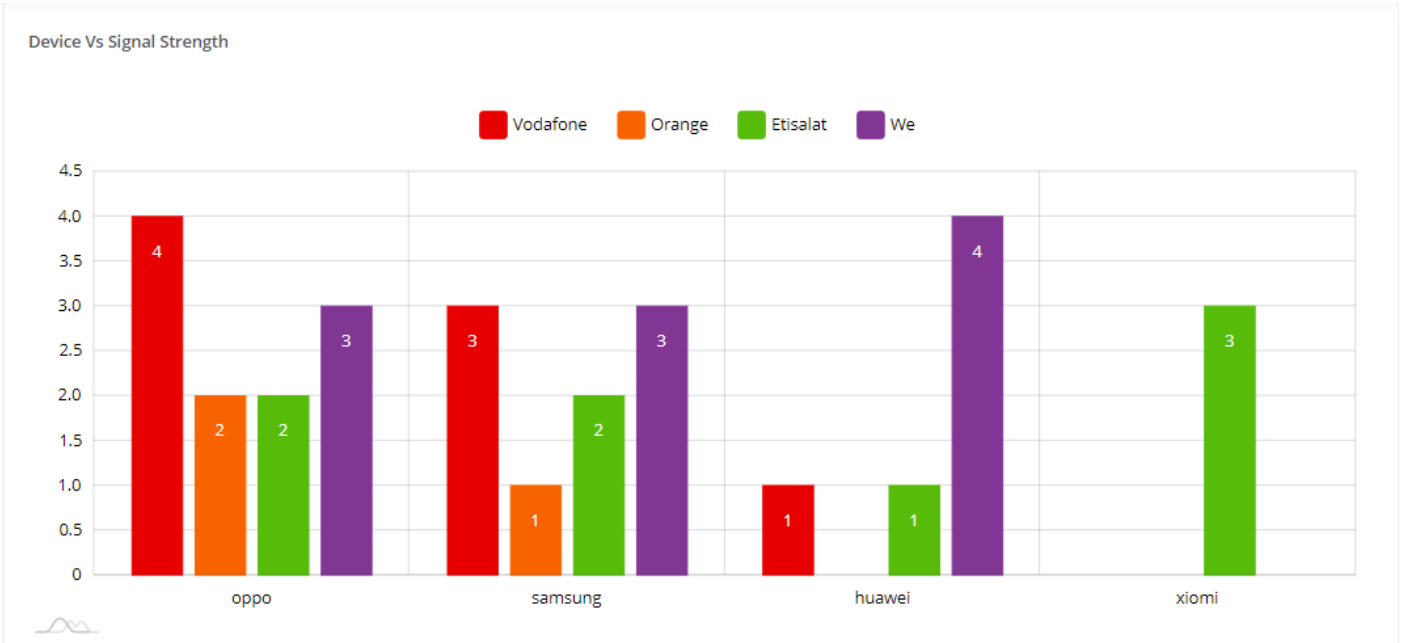
The following table explains the function each API:

API	Description
sendCode	Generate random code , send it to the user by Twilio Messaging API and store this code with user's msisdn into DataBase
isCodeValid	Perform Authentication operation, take verification code and compare it with the stored one in database
createMeasurement	Insert new signal measurement into database
getAllMeasurement	Retrieve all signal measurements from database
getMeasurementByOperator	Retrieve signal measurements from database for specific operator
countAll	Retrieve the count of countries and operators
getDevicesInsightsByOperator	Retrieve average signal strength for different mobile phones with different operators
forwardGeoCoding	Take the address of specific location then return it's coordinates(longitude and latitude)
getSignalStregnthByOperator	Retrieve average signal strength for each operator

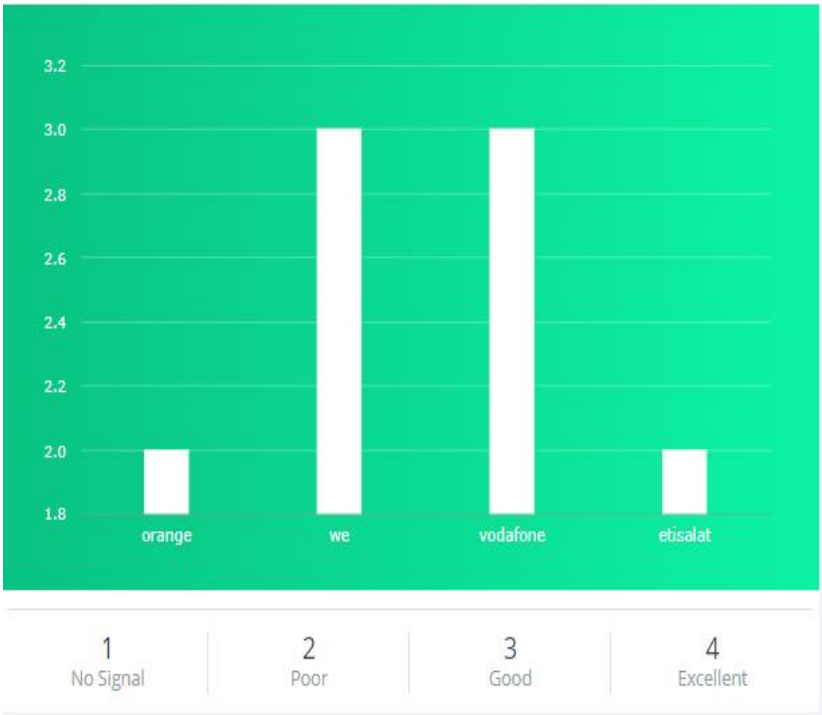
3.6 Web APP Screenshots



RF Coverage

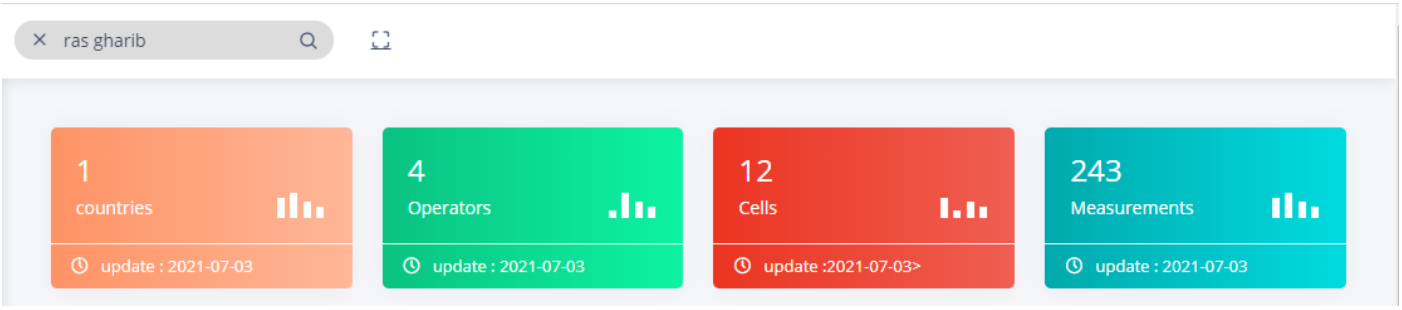


Signal strength of different mobile phones with different operators

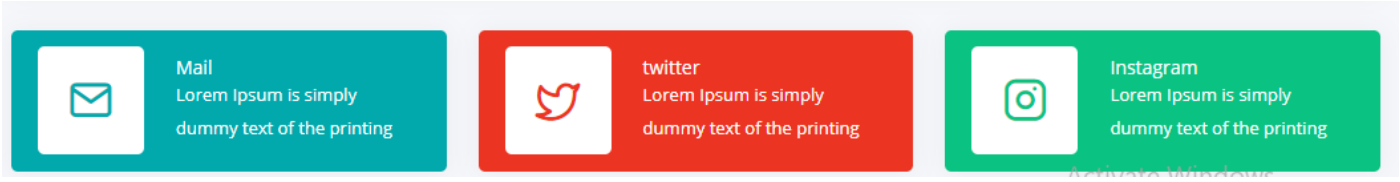


Country	Operator	MCC	MNC
eg	we	602	4
eg	etisalat	602	3
eg	vodafone	602	2
eg	orange	602	1

Average Signal strength for different operators

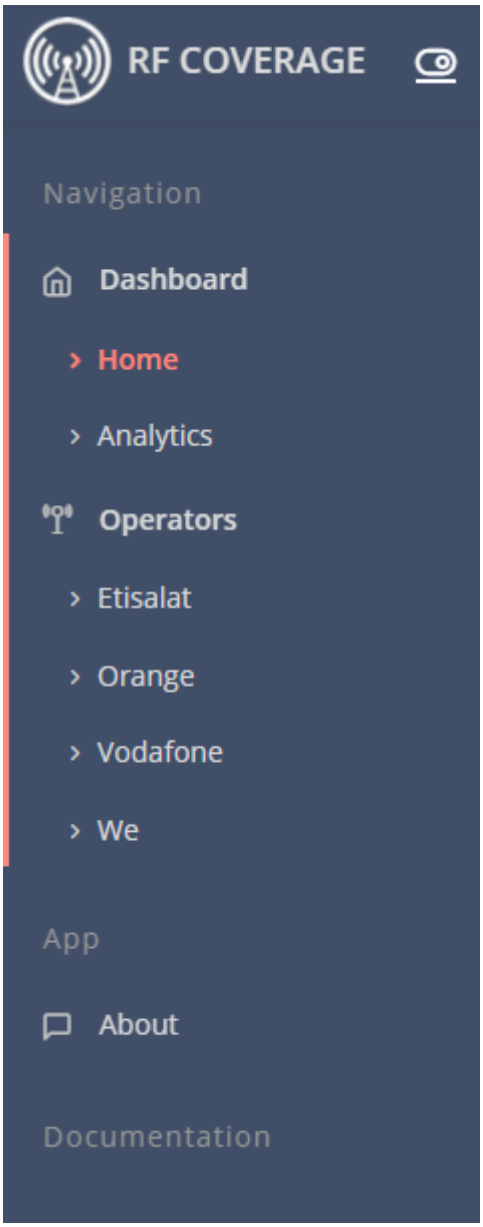


Header



Footer

Side Bar


RF COVERAGE
Navigation
Dashboard
Home
Analytics
Operators
Etisalat
Orange
Vodafone
We
App
About
Documentation

4. References

- **Android Developer Documentation (https://developer.android.com/docs)**
- **Twilio API Reference (https://www.twilio.com/docs/api)**
- **PositionStack API Documentation (https://positionstack.com/documentation)**
- **Leaflet API reference (https://leafletjs.com/reference-1.7.1.html)**
- **amcharts documentation (https://www.amcharts.com/docs/v4/)**
- **Mozilla Developer Network documentation (https://developer.mozilla.org/en-US/)**