**INDEX**

**1. INTRODUCTION--------------------------------------------------------------------------4-5**

**2. BACKGROUND**

**2.1 Theoretical Background------------------------------------------------------------6-13**

**2.2 Technical Background-------------------------------------------------------------14-23**

**2.3 Software profile ---------------------------------------------------------------23-26**

**3. ANALYSIS**

**3.1 Problem definition---------------------------------------------------------------------27**

**3.2 Proposed solution----------------------------------------------------------------------27**

**3.3 Functional Requirements----------------------------------------27**

**3.4 Non-Functional Requirements----------------------------------28**

**3.5 Feasibility Report**

**3.5.1 Technical Feasibility----------------------------------------28**

**3.5.2 Operational Feasibility-------------------------------------29**

**3.5.3 Economical Feasibility--------------------------------------29**

**3.6 Software Requirement Specification------------------------------------------------29**

**4. DESIGN**

**4.1 Unified Modeling Language-------------------------------------------------------30-31**

**4.1.1 Use case Diagram-----------------------------------------------------------------31**

**4.1.2 Sequence Diagram----------------------------------------------------------------32**

**4.1.3 Activity Diagram-----------------------------------------------------------------33**

**5. CODING ---------------------------------------------------------------------------------34-41**

**6. TESTING ------------------------------------------------------------------------- -------42-51**

**7. OUTPUT SCREENS ---------------------------------------------------------- --------52-53**

**8. CONCLUSION -----------------------------------------------------------------------------54**

**9. BIBLIOGRAPHY -------------------------------------------------------------------------54**

**Abstract**

By redirecting unspoiled food from landfill to our neighbors in need, individuals can support their local communities and reduce environmental impact. Non-perishable and unspoiled perishable food can be donated. Donated food can also include leftovers from events and surplus food inventory. Over 90% of the food distributed by food banks in network is donated by the public – that’s why your food donations are absolutely vital to our ability to give everyone referred to us a balanced and nutritious three day supply of food. Without your goodwill, our food banks would really struggle to operate.

On taking into awareness to donate food and reducing the volunteer search we came with a android application called “DONATE FOOD”. Mainly it helps when having the queries like where to donate? How to donate Food? It comes with an advantageous feature of showing all the requests to the volunteer.

**About the project:**

According to this project, those who like to donate food should post the request using “Donate Food” application and volunteer can register initially for viewing those requests and responds to the respecting donator.

**Existing system with limitations:**

In existing system when there is a need of donating food or remains food, searching and finding for the volunteers to collect the food is hard.

**Disadvantage of Existing System:**

* Due to this hardness of finding volunteer to collect food is difficult no proper approach may happened
* No Hassel free donation takes place

**Proposed system with features:**

In this proposed system, donator can easily donate food using one single registration process. Volunteer can check details of donator and can call back for the particulars.

**Advantages of Proposed System:**

* Single request process can reach the maximum volunteers and get back call from volunteer to take food.
* Maximum Volunteers can receive the requests

**Disadvantage of Proposed System:**

* Only android devices supports and not suited for IOS devices

**Hardware & Software:**

**Software Requirements:**

You have to download java SDK and android studio IDE (with SDK bundle) which is official IDE for android application development.

**Hardware Requirements:**

Although you can run Android studio in 2 GB Ram but I highly recommend to use at least 4 GB ram, and if you 8 GB ram, then it will be great experience.

Now for processing power, Intel core i3 clocked at nearly 2 GHz is enough to handle most normal android application, but if you are writing big apps, you'll need better processor like i5 or i7.

And last, if you have a good one android device, use it for testing, because physical device work fast and more powerful than pc emulator.

**Programming Language:**

* Java
* PHP

**Modules:**

* Volunteer
* Donator

1. **INTRODUCTION**

In present days in the entire world mobiles are used most usually. So the people are expecting some new technologies in mobiles as we know android is an open source.

As introductory to android application, there is a lot of scope to reach the people with one app approach. As per the idea of “Donate Food” , we implemented with two modules in this application as Volunteer and Donator.

**Scope of the Project:**

We believe in there is highly scope for the future extension of this project by adding as many as features for the donating and receiving aspects.

**2. BACKGROUND**

**2.1 Theoretical Background:**

**Android**

[Page semi-protected](http://en.wikipedia.org/wiki/Wikipedia:Protection_policy#semi)

|  |  |
| --- | --- |
|  | |
| **[Android robot.svg](http://en.wikipedia.org/wiki/File:Android_robot.svg)[Android logo.png](http://en.wikipedia.org/wiki/File:Android_logo.png)** | |
| C:\Users\fit\Desktop\samsungs7.png Home screen displayed by [Samsung](http://en.wikipedia.org/wiki/Samsung) [Galaxy S7](http://en.wikipedia.org/wiki/Galaxy_Nexus), running [Android 7.0 “Naught](http://en.wikipedia.org/wiki/Ice_Cream_Sandwich_%28operating_system%29)” | |
|  |  |

**Android** is a Linux-based operating system for mobile devices such as smart phones and tablet computers. It is developed by the Open Handset Alliance, led by Google, and other companies.

Google purchased the initial developer of the software, Android Inc., in 2005. The unveiling of the Android distribution in 2007 was announced with the founding of the Open Handset Alliance, a consortium of 86 hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. Google releases the Android code as open-source, under the Apache License. The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android.

Android has a large community of developers writing applications ("apps") that extend the functionality of the devices. Developers write primarily in a customized version of Java. Apps can be downloaded from third-party sites or through online stores such as Google Play (formerly *Android Market*), the app store run by Google. In October 2011, there were more than 500,000 apps available for Android, and the estimated number of applications downloaded from the Android Market as of December 2011 exceeded 10 billion.

Android became the world’s leading Smartphone platform at the end of 2010. For the first quarter of 2012, Android had a 59% Smartphone market share worldwide, with a 331 million devices installed base and 85 million activations or 934,000 per day. Analysts point to the advantage to Android of being a multi-channel, multi-carrier OS.

**Foundation:**

Android, Inc. was founded in [Palo Alto, California](http://en.wikipedia.org/wiki/Palo_Alto,_California), United States in October, 2003 by [Andy Rubin](http://en.wikipedia.org/wiki/Andy_Rubin) Danger [Rich Miner](http://en.wikipedia.org/wiki/Rich_Miner) (co-founder of Wildfire Communications, Inc.) Nick Sears (once VP at [T-Mobile](http://en.wikipedia.org/wiki/T-Mobile_USA)), and Chris White (headed design and interface development at [WebTV](http://en.wikipedia.org/wiki/WebTV)) to develop, in Rubin's words "...smarter mobile devices that are more aware of its owner's location and preferences”. Despite the obvious past accomplishments of the founders and early employees, Android Inc. operated secretly, revealing only that it was working on software for mobile phones. That same year, Rubin ran out of money. [Steve Perlman](http://en.wikipedia.org/wiki/Steve_Perlman), a close friend of Rubin, brought him $10,000 in cash in an envelope and refused a stake in the company.

**Security:**

Android applications run in a Sand Box, an isolated area of the operating system that does not have access to the rest of the system's resources, unless access permissions are granted by the user when the application is installed. Before installing an application, the Play Store displays all required permissions. A game may need to enable vibration, for example, but should not need to read messages or access the phonebook. After reviewing these permissions, the user can decide whether to install the application. The sandboxing and permissions system weakens the impact of vulnerabilities and bugs in applications, but developer confusion and limited documentation has resulted in applications routinely requesting unnecessary permissions, reducing its effectiveness. The complexity of inter-application communication implies Android has a large attack surface.

Several security firms have released antivirus software for Android devices, in particular, AVG Technologies, Avast!, F-Secure,[Kaspersky](http://en.wikipedia.org/wiki/Kaspersky), [McAfee](http://en.wikipedia.org/wiki/McAfee) and Symantec. This software is ineffective as sandboxing also applies to such applications, limiting their ability to scan the deeper system for threats.

### Privacy:

Android Smartphone’s have the ability to report the location of Wi-Fi access points, encountered as phone users move around, to build databases containing the physical locations of hundreds of millions of such access points. These databases form electronic maps to locate Smartphone’s, allowing them to run apps like Foursquare, Latitude, Places, and to deliver location-based ads.

Third party monitoring software such as Taint Droid,an academic research-funded project, can, in some cases, detect when personal information is being sent from applications to remote servers.

In March 2012 it was revealed that Android Apps can copy photos without explicit user permission, Google responded they "originally designed the Android photos file system similar to those of other computing platforms like Windows and Mac OS. *[...]* we're taking another look at this and considering adding permission for apps to access images. We've always had policies in place to remove any apps *[on Google Play]* that improperly access your data."

**Design:**



[http://bits.wikimedia.org/static-1.20wmf2/skins/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:System-architecture.jpg)

**Fig. Architecture diagram**

The Android OS is roughly divided into five sections in four main layers:

* **Linux kernel:** This is the kernel on which Android is based. This layer contains all the low level device drivers for the various hardware components of an Android device.
* **Libraries:** These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The Web Kit library provides functionalities for web browsing.
* **Android runtime:** At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process; with its own instance of the Dalvik virtual machine (Android applications are compiled into Dalvik executables). Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.
* **Application framework:** Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.
* **Applications:** At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

**Features:-**

* **Storage**: Uses My PHP Admin, a lightweight relational database, for data storage.
* **Connectivity:** Android supports connectivity technologies including [GSM](http://en.wikipedia.org/wiki/GSM)/[EDGE](http://en.wikipedia.org/wiki/Enhanced_Data_Rates_for_GSM_Evolution), [IDEN](http://en.wikipedia.org/wiki/Integrated_Digital_Enhanced_Network), [CDMA](http://en.wikipedia.org/wiki/Code_division_multiple_access), [EV-DO](http://en.wikipedia.org/wiki/Evolution-Data_Optimized), [UMTS](http://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System), [Bluetooth](http://en.wikipedia.org/wiki/Bluetooth), [Wi-Fi](http://en.wikipedia.org/wiki/Wi-Fi), [LTE](http://en.wikipedia.org/wiki/LTE_Advanced), [NFC](http://en.wikipedia.org/wiki/Near_field_communication) and [WiMAX](http://en.wikipedia.org/wiki/WiMAX).
* **Multiple language support:** Android supports multiple languages.
* **Messaging:** Supports both SMS and MMS.
* **Web browser**: The web browser available in Android is based on the open-source [Web Kit](http://en.wikipedia.org/wiki/WebKit) layout engine, coupled with [Chrome](http://en.wikipedia.org/wiki/Google_Chrome)'s [V8 JavaScript engine](http://en.wikipedia.org/wiki/V8_JavaScript_engine). The browser scores 100/100 on the [Acid3](http://en.wikipedia.org/wiki/Acid3#Mobile_browsers) test on Android 4.0.
* **Java support**: While most Android applications are written in [Java](http://en.wikipedia.org/wiki/Java_%28programming_language%29), there is no [Java Virtual Machine](http://en.wikipedia.org/wiki/Java_Virtual_Machine) in the platform and Java byte code is not executed. Java classes are compiled into Dalvik executables and run on [Dalvik](http://en.wikipedia.org/wiki/Dalvik_virtual_machine) , a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU. [J2ME](http://en.wikipedia.org/wiki/J2ME) support can be provided via third-party applications.
* **Media support**: Android supports the following audio/video/still media formats: [WebM](http://en.wikipedia.org/wiki/WebM), [H.263](http://en.wikipedia.org/wiki/H.263), [H.264](http://en.wikipedia.org/wiki/H.264) (in [3GP](http://en.wikipedia.org/wiki/3GP) or [MP4](http://en.wikipedia.org/wiki/MP4) [container](http://en.wikipedia.org/wiki/Container_format_%28digital%29)), [MPEG-4 SP](http://en.wikipedia.org/wiki/MPEG-4_Part_2), [AMR](http://en.wikipedia.org/wiki/Adaptive_multi-rate_compression), [AMR-WB](http://en.wikipedia.org/wiki/AMR-WB) (in 3GP container), [AAC](http://en.wikipedia.org/wiki/Advanced_Audio_Coding), [HE-AAC](http://en.wikipedia.org/wiki/HE-AAC) (in MP4 or 3GP container), [MP3](http://en.wikipedia.org/wiki/MP3), [MIDI](http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface), [Ogg Vorbis](http://en.wikipedia.org/wiki/Vorbis), [FLAC](http://en.wikipedia.org/wiki/Free_Lossless_Audio_Codec), [WAV](http://en.wikipedia.org/wiki/WAV), [JPEG](http://en.wikipedia.org/wiki/JPEG), [PNG](http://en.wikipedia.org/wiki/Portable_Network_Graphics), [GIF](http://en.wikipedia.org/wiki/Graphics_Interchange_Format), [BMP](http://en.wikipedia.org/wiki/BMP_file_format),Web.
* **Additional hardware support**: Android can use video/still cameras, [touch screens](http://en.wikipedia.org/wiki/Touchscreen), [GPS](http://en.wikipedia.org/wiki/Global_Positioning_System), [accelerometers](http://en.wikipedia.org/wiki/Accelerometer), [gyroscopes](http://en.wikipedia.org/wiki/Gyroscope), [barometers](http://en.wikipedia.org/wiki/Barometer), [magnetometers](http://en.wikipedia.org/wiki/Magnetometer), dedicated gaming controls, [proximity](http://en.wikipedia.org/wiki/Proximity_sensor) and [pressure sensors](http://en.wikipedia.org/wiki/Pressure_sensor), [thermometers](http://en.wikipedia.org/wiki/Thermometer), accelerated 2D [bit blits](http://en.wikipedia.org/wiki/Bit_blit) (with hardware orientation, scaling, pixel format conversion) and accelerated 3D graphics.
* **Multi-touch**: Android has native support for [multi-touch](http://en.wikipedia.org/wiki/Multi-touch) which was initially made available in handsets such as the [HTC Hero](http://en.wikipedia.org/wiki/HTC_Hero). The feature was originally disabled at the kernel level (possibly to avoid infringing Apple's patents on touch-screen technology at the time). Google has since released an update for the [Nexus One](http://en.wikipedia.org/wiki/Nexus_One) and the [Motorola Droid](http://en.wikipedia.org/wiki/Motorola_Droid) which enables multi-touch natively.

**Activities:**

An [Activity](http://developer.android.com/reference/android/app/Activity.html) is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the *Back* button, it is popped from the stack (and destroyed) and the previous activity resumes. (The back stack is discussed more in the [Tasks and Back Stack](http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html) document.)

**Creating an Activity:**

To create an activity, you must create a subclass of [Activity](http://developer.android.com/reference/android/app/Activity.html) (or an existing subclass of it). In your subclass, you need to implement callback methods that the system calls when the activity transitions between various states of its lifecycle, such as when the activity is being created, stopped, resumed, or destroyed. The two most important callback methods are:

[onCreate()](http://developer.android.com/reference/android/app/Activity.html#onCreate%28android.os.Bundle%29)

You must implement this method. The system calls this when creating your activity. Within your implementation, you should initialize the essential components of your activity. Most importantly, this is where you must call [setContentView()](http://developer.android.com/reference/android/app/Activity.html#setContentView%28android.view.View%29) to define the layout for the activity's user interface.

[onPause()](http://developer.android.com/reference/android/app/Activity.html#onPause%28%29)

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). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).



**Fig:Activitylifecycle**

**Service:**

It is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform inter process communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

A service can essentially take two forms:

**Started:**

A service is "started" when an application component (such as an activity) starts it by calling [startService()](http://developer.android.com/reference/android/content/Context.html#startService%28android.content.Intent%29). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

**Bound:**

A service is "bound" when an application component binds to it by calling [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService%28android.content.Intent,%20android.content.ServiceConnection,%20int%29). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.Although this documentation generally discusses these two types of services separately, your service can work both ways—it can be started (to run indefinitely) and also allow binding. It's simply a matter of whether you implement a couple callback methods: [onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand%28android.content.Intent,%20int,%20int%29) to allow components to start it and [onBind()](http://developer.android.com/reference/android/app/Service.html#onBind%28android.content.Intent%29) to allow binding.

**2.2 Technical Background:-**

### Volley

Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster. Volley is available on [GitHub](https://github.com/google/volley).

Volley offers the following benefits:

* Automatic scheduling of network requests.
* Multiple concurrent network connections.
* Transparent disk and memory response caching with standard HTTP [cache coherence](http://en.wikipedia.org/wiki/Cache_coherence).
* Support for request prioritization.
* Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
* Ease of customization, for example, for retry and backoff.
* Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
* Debugging and tracing tools.

Volley excels at RPC-type operations used to populate a UI, such as fetching a page of search results as structured data. It integrates easily with any protocol and comes out of the box with support for raw strings, images, and JSON. By providing built-in support for the features you need, Volley frees you from writing boilerplate code and allows you to concentrate on the logic that is specific to your app.

Volley is not suitable for large download or streaming operations, since Volley holds all responses in memory during parsing. For large download operations, consider using an alternative like [DownloadManager](https://developer.android.com/reference/android/app/DownloadManager.html).

The core Volley library is developed on [GitHub](https://github.com/google/volley) and contains the main request dispatch pipeline as well as a set of commonly applicable utilities, available in the Volley "toolbox." The easiest way to add Volley to your project is to add the following dependency to your app's build.gradle file:

dependencies {

...

compile 'com.android.volley:volley:1.1.1'

}

### GPS system:

### Google Maps Navigation is a mobile application developed by Google for the Android and [iOS](https://en.wikipedia.org/wiki/IOS" \o "IOS) operating systems that was later integrated into the Google Maps mobile app. The application uses an Internet connection to a GPS navigation system to provide turn-by-turn voice-guided instructions on how to arrive at a given destination.[[1]](https://en.wikipedia.org/wiki/Google_Maps_Navigation#cite_note-1) The application requires connection to Internet data (e.g. 3G, 4G, [WiFi](https://en.wikipedia.org/wiki/WiFi" \o "WiFi), etc.) and normally uses a GPS satellite connection to determine its location. A user can enter a destination into the application, which will plot a path to it. The app displays the user's progress along the route and issues instructions for each turn.

### Security:

Receivers used with the [Context](http://developer.android.com/reference/android/content/Context.html) APIs are by their nature a cross-application facility, so you must consider how other applications may be able to abuse your use of them. Some things to consider are:

* The Intent namespace is global. Make sure that Intent action names and other strings are written in a namespace you own, or else you may inadvertantly conflict with other applications.
* When you use [registerReceiver(BroadcastReceiver, IntentFilter)](http://developer.android.com/reference/android/content/Context.html#registerReceiver%28android.content.BroadcastReceiver,%20android.content.IntentFilter%29), any application may send broadcasts to that registered receiver. You can control who can send broadcasts to it through permissions described below.
* When you publish a receiver in your application's manifest and specify intent-filters for it, any other application can send broadcasts to it regardless of the filters you specify. To prevent others from sending to it, make it unavailable to them with android:exported="false".
* When you use [sendBroadcast(Intent)](http://developer.android.com/reference/android/content/Context.html#sendBroadcast%28android.content.Intent%29) or related methods, normally any other application can receive these broadcasts. You can control who can receive such broadcasts through permissions described below. Alternatively, starting with [ICE\_CREAM\_SANDWICH](http://developer.android.com/reference/android/os/Build.VERSION_CODES.html#ICE_CREAM_SANDWICH), you can also safely restrict the broadcast to a single application with [Intent.setPackage](http://developer.android.com/reference/android/content/Intent.html#setPackage%28java.lang.String%29).

None of these issues exist when using [LocalBroadcastManager](http://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html), since intents broadcast it never go outside of the current process.

Access permissions can be enforced by either the sender or receiver of a broadcast.

To enforce a permission when sending, you supply a non-null permission argument to [sendBroadcast(Intent, String)](http://developer.android.com/reference/android/content/Context.html#sendBroadcast%28android.content.Intent,%20java.lang.String%29) or [sendOrderedBroadcast(Intent, String, BroadcastReceiver, android.os.Handler, int, String, Bundle)](http://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast%28android.content.Intent,%20java.lang.String,%20android.content.BroadcastReceiver,%20android.os.Handler,%20int,%20java.lang.String,%20android.os.Bundle%29). Only receivers who have been granted this permission (by requesting it with the [<uses-permission>](http://developer.android.com/reference/android/R.styleable.html#AndroidManifestUsesPermission) tag in their AndroidManifest.xml) will be able to receive the broadcast.

To enforce a permission when receiving, you supply a non-null permission when registering your receiver -- either when calling [registerReceiver(BroadcastReceiver, IntentFilter, String, android.os.Handler)](http://developer.android.com/reference/android/content/Context.html#registerReceiver%28android.content.BroadcastReceiver,%20android.content.IntentFilter,%20java.lang.String,%20android.os.Handler%29) or in the static [<receiver>](http://developer.android.com/reference/android/R.styleable.html#AndroidManifestReceiver) tag in your AndroidManifest.xml. Only broadcasters who have been granted this permission (by requesting it with the [<uses-permission>](http://developer.android.com/reference/android/R.styleable.html#AndroidManifestUsesPermission) tag in their AndroidManifest.xml) will be able to send an Intent to the receiver.

See the [Security and Permissions](http://developer.android.com/guide/topics/security/security.html) document for more information on permissions and security in general.

### Receiver Lifecycle:

* A BroadcastReceiver object is only valid for the duration of the call to [onReceive(Context, Intent)](http://developer.android.com/reference/android/content/BroadcastReceiver.html#onReceive%28android.content.Context,%20android.content.Intent%29). Once your code returns from this function, the system considers the object to be finished and no longer active.
* This has important repercussions to what you can do in an [onReceive(Context, Intent)](http://developer.android.com/reference/android/content/BroadcastReceiver.html#onReceive%28android.content.Context,%20android.content.Intent%29) implementation: anything that requires asynchronous operation is not available, because you will need to return from the function to handle the asynchronous operation, but at that point the BroadcastReceiver is no longer active and thus the system is free to kill its process before the asynchronous operation completes.
* In particular, you may *not* show a dialog or bind to a service from within a BroadcastReceiver. For the former, you should instead use the [NotificationManager](http://developer.android.com/reference/android/app/NotificationManager.html) API. For the latter, you can use [Context.startService()](http://developer.android.com/reference/android/content/Context.html#startService%28android.content.Intent%29) to send a command to the service.

### Process Lifecycle:

* A process that is currently executing a BroadcastReceiver (that is, currently running the code in its [onReceive(Context, Intent)](http://developer.android.com/reference/android/content/BroadcastReceiver.html#onReceive%28android.content.Context,%20android.content.Intent%29) method) is considered to be a foreground process and will be kept running by the system except under cases of extreme memory pressure.
* Once you return from onReceive(), the BroadcastReceiver is no longer active, and its hosting process is only as important as any other application components that are running in it. This is especially important because if that process was only hosting the BroadcastReceiver (a common case for applications that the user has never or not recently interacted with), then upon returning from onReceive() the system will consider its process to be empty and aggressively kill it so that resources are available for other more important processes.
* This means that for longer-running operations you will often use a [Service](http://developer.android.com/reference/android/app/Service.html) in conjunction with a BroadcastReceiver to keep the containing process active for the entire time of your operation.

**Native Android Actions:**

Native Android applications also use Intents to launch Activities and sub-Activities.

The following (noncomprehensive) list shows some of the native actions available as static string constants in the Intent class. When creating implicit Intents, you can use these actions, known as *Activity Intents*, to start Activities and sub-Activities within your own applications.

* ACTION\_ALL\_APPS — Opens an Activity that lists all the installed applications. Typically, this is handled by the launcher.
* ACTION\_ANSWER — Opens an Activity that handles incoming calls. This is normally handled by the native in-call screen.
* ACTION\_BUG\_REPORT — Displays an Activity that can report a bug. This is normally handled by the native bug-reporting mechanism.
* ACTION\_CALL — Brings up a phone dialer and immediately initiates a call using the number supplied in the Intent’s data URI. This action should be used only for Activities that replace the native dialer application. In most situations it is considered better form to use ACTION\_DIAL.
* ACTION\_CALL\_BUTTON — Triggered when the user presses a hardware “call button.” This typically initiates the dialer Activity.
* ACTION\_DELETE — Starts an Activity that lets you delete the data specified at the Intent’s data URI.
* ACTION\_DIAL — Brings up a dialer application with the number to dial prepopulated fromthe Intent’s data URI. By default, this is handled by the native Android phone dialer. The dialer can normalize most number schemas — for example, tel:555-1234 and <tel:(212)> 555 1212 are both valid numbers.
* ACTION\_EDIT — Requests an Activity that can edit the data at the Intent’s data URI.
* ACTION\_INSERT — Opens an Activity capable of inserting new items into the Cursor specified in the Intent’s data URI. When called as a sub-Activity, it should return a URI to the newly inserted item.
* ACTION\_PICK — Launches a sub-Activity that lets you pick an item from the Content Provider specified by the Intent’s data URI. When closed, it should return a URI to the item that was picked. The Activity launched depends on the data being picked — for example, passing content://contacts/people will invoke the native contacts list.
* ACTION\_SEARCH — Typically used to launch a specific search Activity. When it’s fired without a specific Activity, the user will be prompted to select from all applications that support search. Supply the search term as a string in the Intent’s extras using SearchManager.QUERY as the key.
* ACTION\_SEARCH\_LONG\_PRESS — Enables you to intercept long presses on the hardware search key. This is typically handled by the system to provide a shortcut to a voice search.
* ACTION\_SENDTO — Launches an Activity to send data to the contact specified by the Intent’s data URI.
* ACTION\_SEND — Launches an Activity that sends the data specified in the Intent. The recipient contact needs to be selected by the resolved Activity. Use setType to set the MIME type of the transmitted data. The data itself should be stored as an extra by means of the key EXTRA\_TEXT or EXTRA\_STREAM, depending on the type. In the case of email, the native Android applications will also accept extras via the EXTRA\_EMAIL, EXTRA\_CC, EXTRA\_BCC, and EXTRA\_SUBJECT keys. Use the ACTION\_SEND action only to send data to a remote recipient (not to another application on the device).
* ACTION\_VIEW — This is the most common generic action. View asks that the data supplied in the Intent’s data URI be viewed in the most reasonable manner. Different applications will handle view requests depending on the URI schema of the data supplied. Natively http: addresses will open in the browser; tel: addresses will open the dialer to call the number;geo: addresses will be displayed in the Google Maps application; and contact content will be displayed in the Contact Manager.
* ACTION\_WEB\_SEARCH — Opens the Browser to perform a web search based on the query supplied using the SearchManager.QUERY key.

**2.3 Software profile:**

The following terms are used in this document.

**.apk extension:**

The extension for an Android package file, which typically contains all of the tiles related to a single Android application. The file itself is a compressed collection of an AndroidManifest.xml file, application code (.dex files), resource files, and other files. A project is compiled into a single .apk file.

**.dex extension:**

Android programs are compiled into .dex (Dalvik Executable) files, which are in turn zipped into a single .apk file on the device. .dex files can be created by automatically translating compiled applications written in the java programming language.

**Action:**

A description of something that an Intent sender wants done. An action is a string value assigned to intent. Action strings can be defined by Android or by a third-party developer.

**Activity:**

A single screen in an application, with supporting Java code, derived from the activity class.

**Adb:**

Android Debug Bridge, a command-line debugging application shipped with the SDK. It provides tools to browse the device, copy tools on the device, and forward ports for debugging.

**Application:**

A collection of one or more activities, services, listeners, and intent receivers. An application has a single manifest, and is compiled into a single .apk tile on the device.

**Content Provider:**

A class built on Content Provider that handles content query strings of a specific format to return data in a specific format.

**Content URI:**

A type of URI. See the URI entry

**Dalvik:**

The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory map able execution. The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included “ex” tool. The VM runs on top of posix-complaint operating systems, which it relies on for underlying functionality. The Dalvik core class library is intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device.

**DDMS:**

Dalvik Debug Monitor Service, a GUI debugging application application shipped with the SDK. It provides screen capture, log dump, and process examination capabilities.

**Drawable:**

A compiled visual resource that can be used as a background, title, or other part of the screen. It is compiled into an android.graphics.drawable subclass.

**Intent:**

A class that contains several fields describing what a caller would like to do. The caller sends this Intent to Android’s intent resolver, which looks through the intent filters of all applications to find the activity most suited to handle this intent. Intent fields include the desired action category, a data string, MIME type of the data, a handling class, and the restrictions.

**Intent Filter:**

Activities and intent receivers include one or more filters in their manifest too describe what kind of intents or messages they can handle or want to receive. An intent filter lists a set of requirements, such as data type, action requested, and URI format, that the intent or message must fulfill. For activities, Android searches for the activity with the most closely matching valid match between the Intent and the activity filter. For messages, Android will forward a message to all receivers with matching intent filters.

**Intent Receiver:**

An application class that listens for messages broadcast by calling Context.sendBroadcast ().

**Layout resource:**

An XML file that describes the layout of an Activity screen.

**Manifest:**

An XML file associated with each Application that describes the various activities, intent filters, services, and other items that it exposes.

**URIs:**

Android uses URI string both for requesting data (e.g. a list of contacts) and for requesting actions (e.g., opening a Web page in a browser). Both are valid URI strings, but have different values. All requests for data must start with the string “content://”. Action strings are valid URIs that can be handled approximately by applications on the device; for example, a URI starting with “http://” will be handled by the browser.

**3. ANALYSIS**

**3.1 Problem Definition:**

In present days in the entire world mobiles are using very rashly so the people are expecting some new technologies in mobiles as we know android is an open source.

Defined to the analysis, there is a lot of problem at present scenario on wastage of food, even the government had investing in the form of camps and awareness through media and cinemas.

**3.2 Proposed Solution:**

The proposed system is a simple and user friendly application for donating food and collecting donators list to the volunteer.

**Advantages Of Proposed System:**

* Most of the persons can use with this low size application.
* Simple and user friendly design
* Single request is enough

**3.3 Functional Requirements Specification:**

The system is required to perform the following functions:

* Display all the information about the application that is being developed and some set of instructions the user might want to remember before he sets up the system for configuring global time.
* Sign in your application with jar signer before running your application.
* Install your apk file with android bridge (i.e.; adb).
* After executing your application Tower of Hanoi
* Ability to drag the disks and drop them.
* Able to display score.
* Ability to pause and resume & stop the application.

**3.4 Non-Functional Requirements:**

* Application framework enabling reuse and replacement of components.
* Dalvik virtual machine optimized for mobile devices.
* Integrated browser based on the open source Web Kit engine.
* Optimized graphics powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional).
* SQLite for structured data storage.
* GPS
* Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plug-in for the Eclipse IDE.
* The system is expected to run on low memory devices also.
* The system should not consume lot of bandwidth so that the other applications will block for the internet.

**3.5 Feasibility Report:-**

**3.5.1Technical Feasibility:** Technical feasibility centers on the existing mobile system and to what extent it support the proposed addition if the budget is serious constraint, then the project is judged not feasible. The technical feasibility is important role in my project because I am using Android operating system.

**3.5.2 Operational Feasibility:**

People are inherently resistant to change and mobiles have been known to facilitate change. In my project a technical person requires to configure the software and technical background is necessary to work on the sensors.

**3.5.3 Economic Feasibility:**

This procedure is to determine the benefits and savings that are expected from the candidate system and compare with cost. If benefits outweigh cost then the decision is to make design and implement the system. Otherwise further justification or alterations in the proposed system that have to make if it is having a change of the system life cycle. For my project I am not expecting any feasibility costs spent on this project because here I am using open source environments.

**Software Requirements:**

* Operating System : Android, Linux, Windows XP
* Software : J2SE, ADT plug-in
* Development Tools : Android SDK, Android Emulator, Eclipse Helios

**Hardware Requirements:**

* Pentium IV with 2 GHZ
* 1GB RAM
* 40 GB Hard Drive
* Android Phone (optional)

**4. DESIGN**

**4.1 Unified Modeling Language:**

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process.  The UML uses mostly graphical notations to express the design of software projects.  Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

**Goals of UML:-**

The primary goals in the design of the UML were:

* Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
* Provide extensibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development processes.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of the OO tools market.
* Support higher-level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**Why Use UML?**

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs.

**Relations:**

**1. Dependency:**

Dependency is a semantic relationship between two things in which change to one thing may affect the semantics of other things. Graphically a dependency rendered as a dashed-line, possibly directed occasionally including a label.



Fig: Dependency

**2. Association:**

Association is a structural relationship that describes set of links.

A link being a connection among objects. Aggregation is a special kind of association a structural relationship between a whole and its parts.

It is graphically rendered as a solid-line. Such as multiplicity and role name

Employer employee

Fig: Association

**3. Generalization:**

Generalization is a specialization/generalization relationship in which objects of specialized element (child) are substitutable for object of the generalized element (parent).in this way the child shares the structure and behavior of the parent.

Graphically rendered as a solid-line with a hollow over-head pointing to the parent.

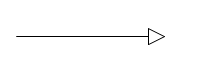


Fig: Generalization

**4. Realization:**

A realization is a semantic relationship between classifiers. Where in one classifier specifies a contract that another classifier guarantees to carry-out you will encounter relationships in two places between interfaces and classes or components between use case collaborations.

Graphically rendered as cross between generalization and dependency relationships.

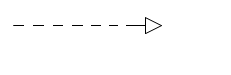


Fig: Realization

**UML Diagrams**:

UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. UML diagrams commonly created in visual modeling tools include.

**4.1Usecase Diagram: -**

A usecase diagram shows as set of use cases and actors and their relationships. Use case diagrams are especially important in organizing and modeling behavior of a system.

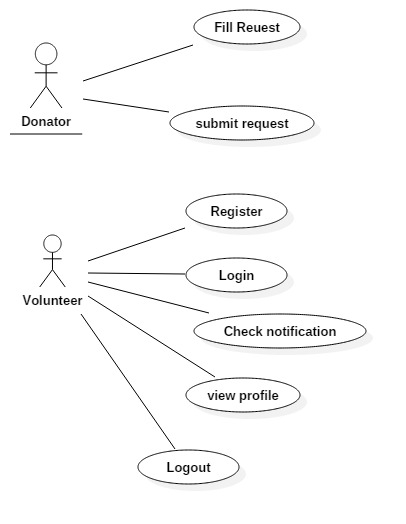


Fig: Use case diagram

**4.2 Sequence Diagram: -**

A sequence diagram is an interaction diagram that emphasizes the time ordering of messages. A sequence diagram shows a set of objects and messages sent and receive by those objects. The objects are typically named or anonymous instances of other things, such as collaborations, components and nodes. We can use sequence diagrams to illustrate the dynamic view of a system.

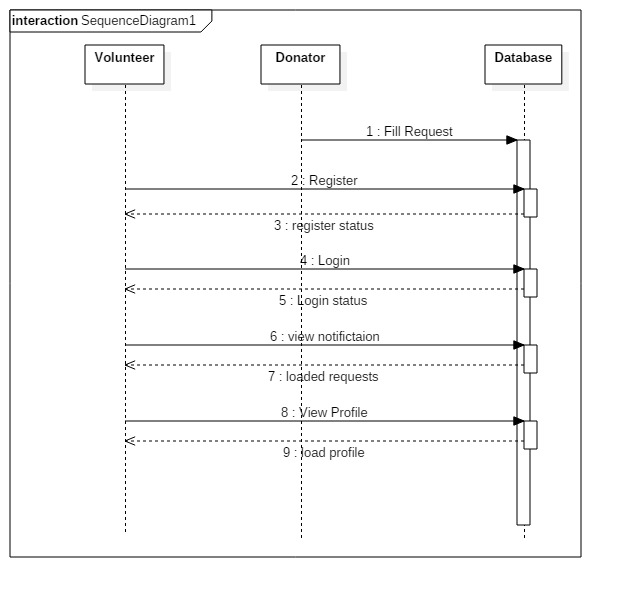
****

Fig: Sequence diagram

**4.4 Activity Diagram: -**

Activity diagrams describe the workflow behavior of a system.  Activity diagrams are similar to state diagram because activities are the state of doing something.  The diagrams describe the state of activities by showing the sequence of activities performed.  Activity diagrams can show activities that are conditional or parallel.

**When to Use: Activity Diagrams**

Activity diagrams should be used in conjunction with other modeling techniques such as interaction diagrams and state diagrams.  The main reason to use activity diagrams is to model the workflow behind the system being designed.  Activity Diagrams are also useful for: analyzing a use case by describing what actions needs to take place and when they should occur; describing a complicated sequential algorithm; and modeling applications with parallel processes.

However, activity diagrams should not take the place of interaction diagrams and state diagrams.  Activity diagrams do not give detail about how objects behave or how objects collaborate.

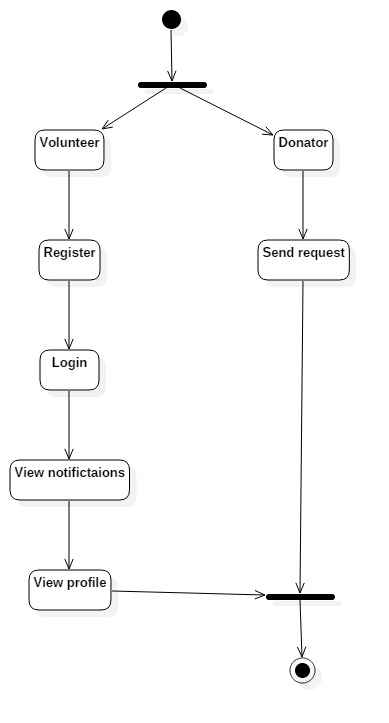


Fig: Activity diagram

**5. CODING**

**Java File:-**

**package** com.ravi.fit.donatefood;  
  
**import** android.content.Intent;  
**import** android.content.SharedPreferences;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.text.TextUtils;  
**import** android.util.Log;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.android.volley.AuthFailureError;  
**import** com.android.volley.Request;  
**import** com.android.volley.Response;  
**import** com.android.volley.VolleyError;  
**import** com.android.volley.VolleyLog;  
**import** com.android.volley.toolbox.StringRequest;  
**import** com.ravi.fit.donatefood.Network.API;  
**import** com.ravi.fit.donatefood.Network.VolleySingleton;  
  
**import** org.json.JSONArray;  
**import** org.json.JSONObject;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public class** Volunteer\_Login **extends** AppCompatActivity {  
TextView **register\_volun**;  
EditText **userids**,**passwords**;  
Button **login\_btn**;  
SharedPreferences **shre**;  
 String **user\_name**,**password**;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_volunteer\_\_login***);  
 **if** (SharedPrefManager.*getInstance*(**this**).isLoggedIn()) {  
 finish();  
 startActivity(**new** Intent(**this**, VolunteerHome.**class**));  
 }  
 **shre** = getSharedPreferences(**"pref"**,***MODE\_PRIVATE***);  
 **register\_volun** = findViewById(R.id.***register\_volunteer***);  
 **login\_btn** = findViewById(R.id.***vol\_login***);  
 **userids** = findViewById(R.id.***userids\_vol***);  
 **passwords** = findViewById(R.id.***passwords\_vol***);  
  
  
 **login\_btn**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 volunteerLogin();  
 }  
 });  
 **register\_volun**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 finish();  
 Intent i = **new** Intent(getApplicationContext(),Register\_Volunteer.**class**);  
 startActivity(i);  
 }  
  
 });  
 }  
  
 **private void** volunteerLogin() {  
  
 **user\_name** = **userids**.getText().toString();  
 **password** = **passwords**.getText().toString();  
  
 *//validating inputs* **if** (TextUtils.*isEmpty*(**user\_name**)) {  
 **userids**.setError(**"Please enter your username"**);  
 **userids**.requestFocus();  
 }  
  
 **if** (TextUtils.*isEmpty*(**password**)) {  
 **passwords**.setError(**"Please enter your password"**);  
 **passwords**.requestFocus();  
 }  
  
 *//calling url* String serverurl = API.*login*;  
 *//sending request to url for response Or Request Constructer with 4 parameters* StringRequest sr = **new** StringRequest(Request.Method.***POST***, serverurl, **new** Response.Listener<String>() {  
 @Override  
 **public void** onResponse(String response) {  
  
 **try**{  
 JSONObject jsonObject=**new** JSONObject(response);  
 String res=jsonObject.getString(**"result"**);*//result should be matched with url link response ie,{"result":"success"}* **if**(res.equals(**"success"**)) *//array key* {  
 Toast.*makeText*(getApplicationContext(),**"Login : "**+res,Toast.***LENGTH\_SHORT***).show();  
 SharedPreferences.Editor editor = **shre**.edit();  
 editor.putString(**"user"**,**user\_name**);  
 editor.apply();  
 Intent i = **new** Intent(getApplicationContext(),VolunteerHome.**class**);  
 startActivity(i);  
  
 }**else** {  
 Toast.*makeText*(getApplicationContext(),**"error"**,Toast.***LENGTH\_SHORT***).show();  
 }  
  
  
 } **catch** (Exception e) {  
 Log.*e*(**"ERROR"**,**"Exception"**);  
 }  
  
 }  
 }, **new** Response.ErrorListener() {  
 @Override  
 **public void** onErrorResponse(VolleyError error) {  
 VolleyLog.*d*(**"Main"**, **"Error: "** + error.getMessage());  
 Log.*d*(**"Main"**, **""** + error.getMessage() + **","** + error.toString());  
  
 }  
 }){  
 @Override  
 **public** Map<String,String> getParams() **throws** AuthFailureError {  
  
 Map<String,String> data= **new** HashMap<String, String>();*//to bind group of data  
 //to insert data from edit feilds into table feilds* data.put(**"email"**,**user\_name**);  
 data.put(**"password"**,**password**);  
 **return** data;  
 }  
 };  
 *//TO add request to Volley* VolleySingleton.*getInstance*(**this**).addToRequestQueue(sr);  
 }  
 @Override  
 **public void** onBackPressed(){  
 Intent i = **new** Intent(getBaseContext(),SplashActivity.**class**);  
 startActivity(i);  
 finish();  
 }  
}

**Manifest File:**

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.ravi.fit.donatefood"**>  
  
 <**uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"** />  
 <**uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"** />  
 <**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** />  
 <**uses-permission android:name="android.permission.INTERNET"**/>  
 <**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"**/>  
 <**uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"** />  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@drawable/fooddonate"  
 android:label="@string/app\_name"  
 android:roundIcon="@drawable/fooddonate"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".SplashActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
 <**activity android:name=".Volunteer\_Login"** />  
 <**activity android:name=".Register\_Volunteer"** />  
 <**activity android:name=".VolunteerHome"** />  
 <**activity android:name=".Donator"**>  
 </**activity**>  
 </**application**>  
  
</**manifest**>

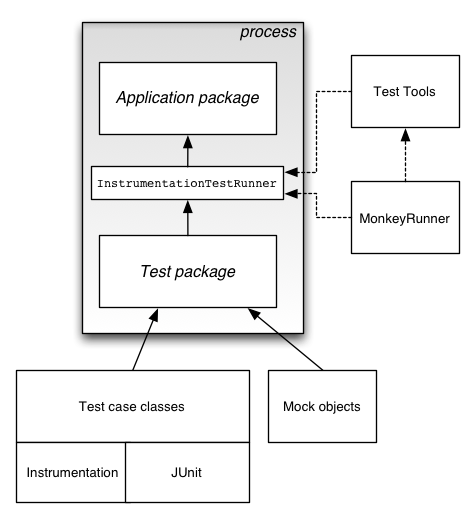
**6. TESTING**

The Android testing framework, an integral part of the development environment, provides architecture and powerful tools that help you test every aspect of your application at every level from unit to framework.

The testing framework has these key features:

* Android test suites are based on
* . You can use plain JUnit to test a class that doesn't call the Android API, or Android's JUnit extensions to test Android components. If you're new to Android testing, you can start with general-purpose test case classes such as Android test case and then go on to use more sophisticated classes.
* The Android JUnit extensions provide component-specific test case classes. These classes provide helper methods for creating mock objects and methods that help you control the lifecycle of a component.
* Test suites are contained in test packages that are similar to main application packages, so you don't need to learn a new set of tools or techniques for designing and building tests.
* The SDK tools for building and tests are available in Eclipse with ADT, and also in command-line form for use with other IDES. These tools get information from the project of the application under test and use this information to automatically create the build files, manifest file, and directory structure for the test package.
* The SDK also provides monkey runner , an API testing devices with Python programs, and UI/Application exercise runner, a command-line tool for stress-testing UIs by sending pseudo-random events to a device.
* This document describes the fundamentals of the Android testing framework, including the structure of tests, the APIs that you use to develop tests, and the tools that you use to run tests and view results. The document assumes you have a basic knowledge of Android application programming and JUnit testing methodology.

The following diagram summarizes the testing framework:

[](http://developer.android.com/images/testing/test_framework.png)

**Test Structure:**

Android's build and test tools assume that test projects are organized into a standard structure of tests, test case classes, test packages, and test projects.

Android testing is based on JUnit. In general, a JUnit test is a method whose statements test a part of the application under test. You organize test methods into classes called test cases (or test suites). Each test is an isolated test of an individual module in the application under test. Each class is a container for related test methods, although it often provides helper methods as well.

In JUnit, you build one or more test source files into a class file. Similarly, in Android you use the SDK's build tools to build one or more test source files into class files in an Android test package. In JUnit, you use a test runner to execute test classes. In Android, you use test tools to load the test package and the application under test, and the tools then execute an Android-specific test runner.

**Android Test Case:**

A useful general test case class, especially if you are just starting out with Android testing, is [Android Test Case](http://developer.android.com/reference/android/test/AndroidTestCase.html). It extends both [Test Case](http://developer.android.com/reference/junit/framework/TestCase.html) and [Assert](http://developer.android.com/reference/junit/framework/Assert.html). It provides the JUnit-standard setUp() and tearDown() methods, as well as all of JUnit's Assert methods. In addition, it provides methods for testing permissions, and a method that guard against memory leaks by clearing out certain class references.

**Component-specific test cases:**

A key feature of the Android testing framework is its component-specific test case classes. These address specific component testing needs with methods for fixture setup and teardown and component lifecycle control. They also provide methods for setting up mock objects. These classes are described in the component-specific testing topics:

* [Activity Testing](http://developer.android.com/guide/topics/testing/activity_testing.html)
* Content Provider Testing
* [Service Testing](http://developer.android.com/guide/topics/testing/service_testing.html)

Android does not provide a separate test case class for Broadcast Receiver. Instead, test a Broadcast Receiver by testing the component that sends it Intent objects, to verify that the Broadcast Receiver responds correctly.

**Application Test Case:**

You use the [Application Test Case](http://developer.android.com/reference/android/test/ApplicationTestCase.html) class to test the setup and teardown of [Application](http://developer.android.com/reference/android/app/Application.html) objects. These objects maintain the global state of information that applies to all the components in an application package. The test case can be useful in verifying that the <application> element in the manifest file is correctly set up. Note, however, that this test case does not allow you to control testing of the components within your application package.

**Instrumentation Test Case:**

If you want to use instrumentation methods in a test case class, you must use [Instrumentation Test Case](http://developer.android.com/reference/android/test/InstrumentationTestCase.html) or one of its subclasses. The [Activity](http://developer.android.com/reference/android/app/Activity.html) test cases extend this base class with other functionality that assists in Activity testing.

**Assertion classes:**

Because Android test case classes extend JUnit, you can use assertion methods to display the results of tests. An assertion method compares an actual value returned by a test to an expected value, and throws an Assertion Exception if the comparison test fails. Using assertions is more convenient than doing logging, and provides better test performance. Besides the JUnit [Assert](http://developer.android.com/reference/junit/framework/Assert.html) class methods, the testing API also provides the [MoreAsserts](http://developer.android.com/reference/android/test/MoreAsserts.html) and [ViewAsserts](http://developer.android.com/reference/android/test/ViewAsserts.html) classes:

* [MoreAsserts](http://developer.android.com/reference/android/test/MoreAsserts.html) contains more powerful assertions such as [assertContainsRegex(String, String)](http://developer.android.com/reference/android/test/MoreAsserts.html#assertContainsRegex(java.lang.String, java.lang.String)), which does regular expression matching.
* [ViewAsserts](http://developer.android.com/reference/android/test/ViewAsserts.html) contains useful assertions about Views. For example it contains [assertHasScreenCoordinates(View, View, int, int)](http://developer.android.com/reference/android/test/ViewAsserts.html#assertHasScreenCoordinates(android.view.View, android.view.View, int, int)) that tests if a View has a particular X and Y position on the visible screen. These assert simplify testing of geometry and alignment in the UI.

**Mock object classes:**

* To facilitate dependency injection in testing, Android provides classes that create mock system objects such as [Context](http://developer.android.com/reference/android/content/Context.html) objects, [ContentProvider](http://developer.android.com/reference/android/content/ContentProvider.html) objects, [ContentResolver](http://developer.android.com/reference/android/content/ContentResolver.html) objects, and [Service](http://developer.android.com/reference/android/app/Service.html) objects. Some test cases also provide mock [Intent](http://developer.android.com/reference/android/content/Intent.html) objects. You use these mocks both to isolate tests from the rest of the system and to facilitate dependency injection for testing. These classes are found in the Java packages [android.test](http://developer.android.com/reference/android/test/package-summary.html) and [android.test.mock](http://developer.android.com/reference/android/test/mock/package-summary.html).
* Mock objects isolate tests from a running system by stubbing out or overriding normal operations. For example, a [MockContentResolver](http://developer.android.com/reference/android/test/mock/MockContentResolver.html) replaces the normal resolver framework with its own local framework, which is isolated from the rest of the system. MockContentResolver also stubs out the [notifyChange(Uri, ContentObserver, boolean)](http://developer.android.com/reference/android/content/ContentResolver.html#notifyChange(android.net.Uri, android.database.ContentObserver, boolean)) method so that observer objects outside the test environment are not accidentally triggered.
* Mock object classes also facilitate dependency injection by providing a subclass of the normal object that is non-functional except for overrides you define. For example, the Mock Resources object provides a subclass of Resources in which all the methods throw Exceptions when called. To use it, you override only those methods that must provide information.
* These are the mock object classes available in Android

**Contexts for testing**:

Android provides two Context classes that are useful for testing:

* [IsolatedContext](http://developer.android.com/reference/android/test/IsolatedContext.html) provides an isolated [Context](http://developer.android.com/reference/android/content/Context.html), File, directory, and database operations that use this Context take place in a test area. Though its functionality is limited, this Context has enough stub code to respond to system calls.

This class allows you to test an application's data operations without affecting real data that may be present on the device.

* [RenamingDelegatingContext](http://developer.android.com/reference/android/test/RenamingDelegatingContext.html) provides a Context in which most functions are handled by an existing [Context](http://developer.android.com/reference/android/content/Context.html), but file and database operations are handled by a [IsolatedContext](http://developer.android.com/reference/android/test/IsolatedContext.html). The isolated part uses a test directory and creates special file and directory names. You can control the naming yourself, or let the constructor determine it automatically.

This object provides a quick way to set up an isolated area for data operations, while keeping normal functionality for all other Context operations.

**Running Tests:**

Test cases are run by a test runner class that loads the test case class, set ups, runs, and tears down each test. An Android test runner must also be instrumented, so that the system utility for starting applications can control how the test package loads test cases and the application under test. You tell the Android platform which instrumented test runner to use by setting a value in the test package's manifest file.

**Seeing Test Results:**

The Android testing framework returns test results back to the tool that started the test. If you run a test in Eclipse with ADT, the results are displayed in a new JUnit view pane. If you run a test from the command line, the results are displayed in STDOUT. In both cases, you see a test summary that displays the name of each test case and method that was run. You also see all the assertion failures that occurred. These include pointers to the line in the test code where the failure occurred. Assertion failures also list the expected value and actual value.

**Working With Package names:**

In the test environment, you work with both Android application package names and Java package identifiers. Both use the same naming format, but they represent substantially different entities. You need to know the difference to set up your tests correctly.

An Android package name is a unique system name for an .apk file, set by the "android: package" attribute of the <manifest> element in the package's manifest. The Android package name of your test package must be different from the Android package name of the application under test. By default, Android tools create the test package name by appending ".test" to the package name of the application under test.

The test package also uses an Android package name to target the application package it tests. This is set in the "android:targetPackage" attribute of the <instrumentation> element in the test package's manifest.

A Java package identifier applies to a source file. This package name reflects the directory path of the source file. It also affects the visibility of classes and members to each other.

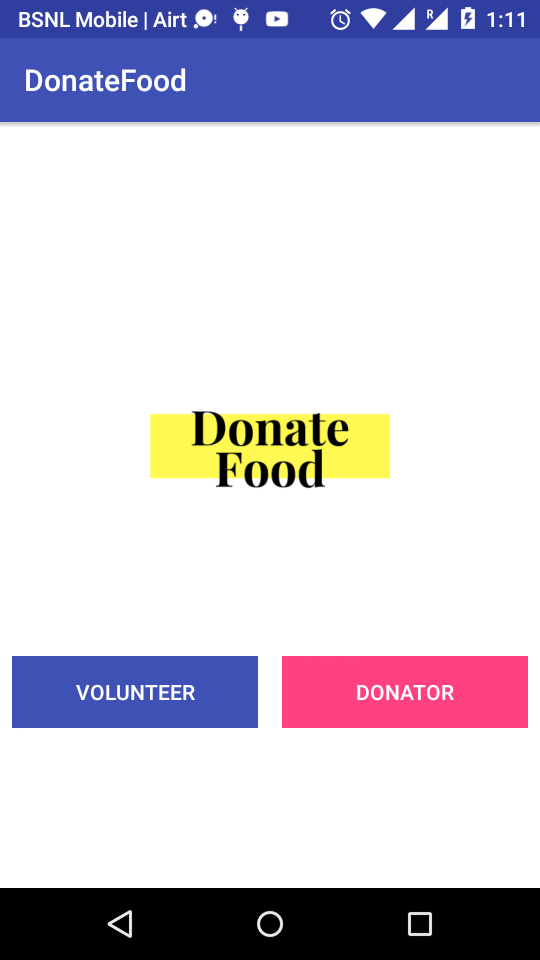
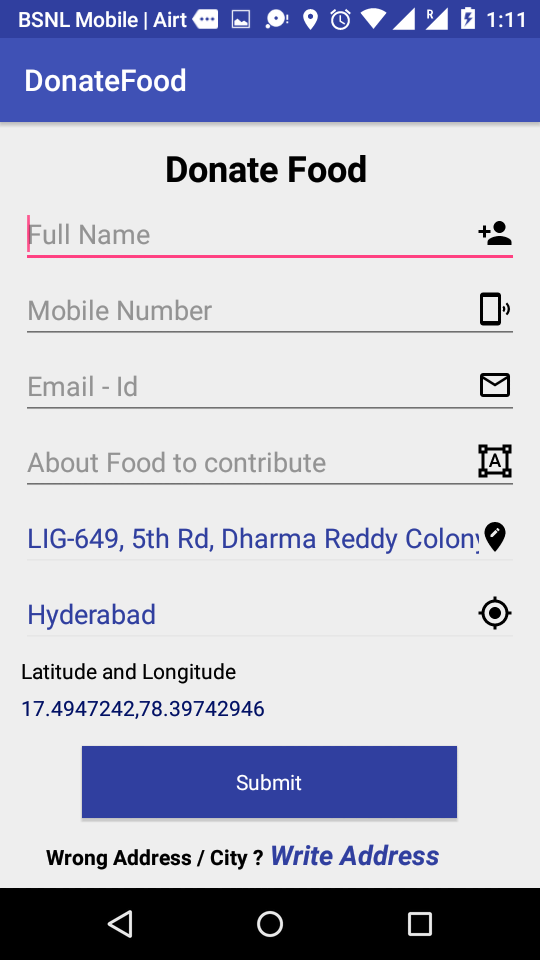
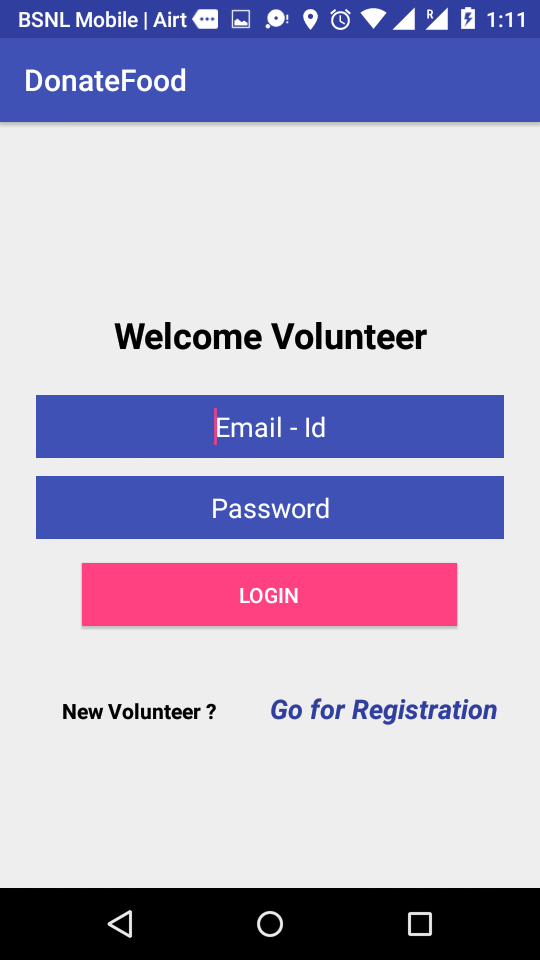
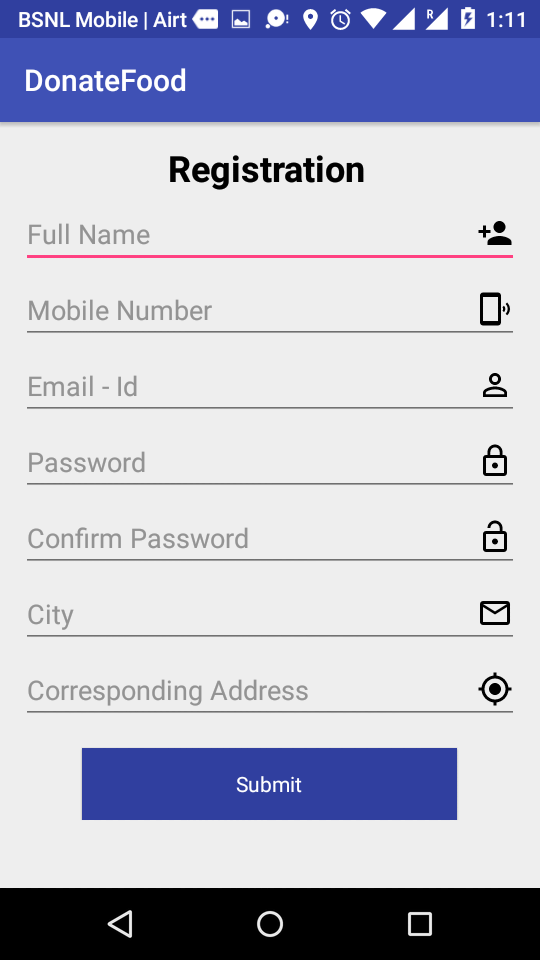
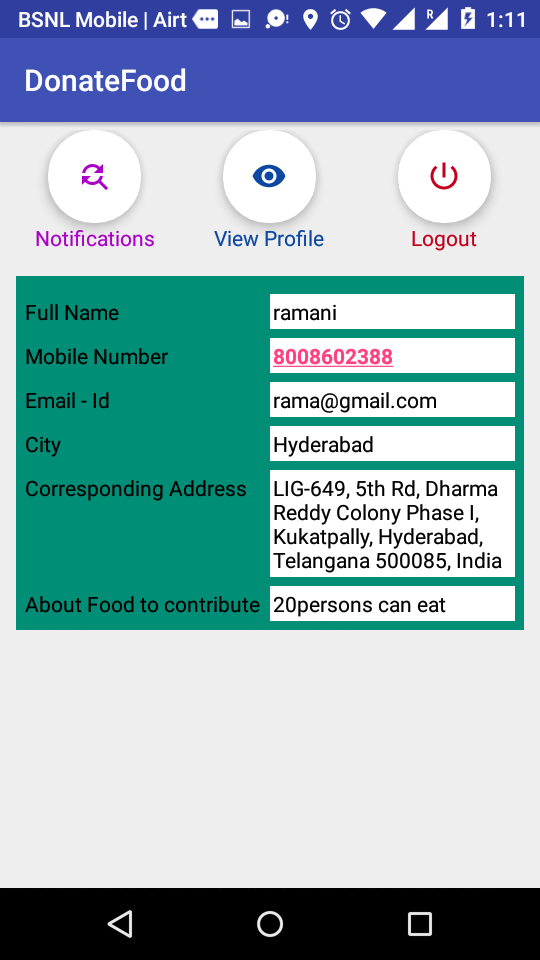
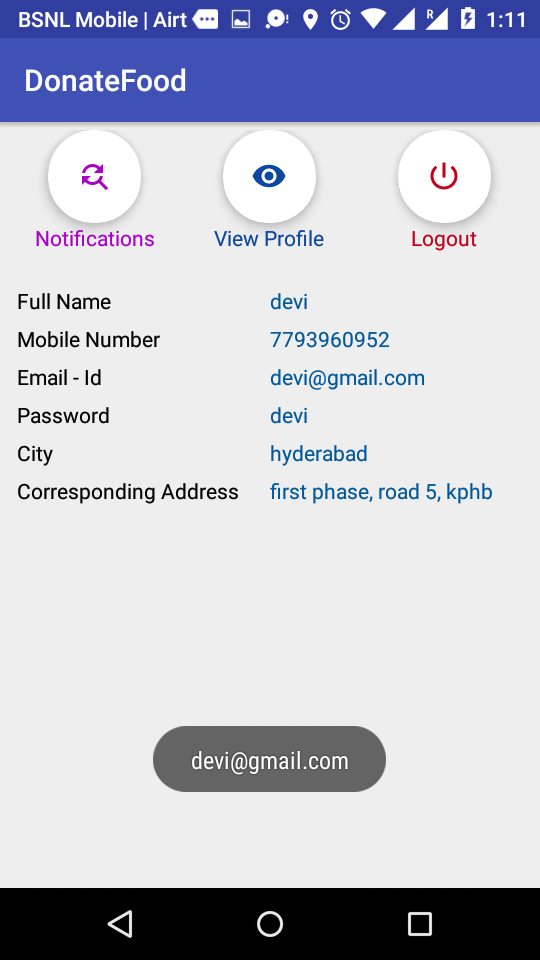
**What to Test:**

The topic [What To Test](http://developer.android.com/guide/topics/testing/what_to_test.html) describes the key functionality you should test in an Android application, and the key situations that might affect that functionality.

Most units testing are specific to the Android component you are testing. The topics [Activity Testing](http://developer.android.com/guide/topics/testing/activity_testing.html), [Content Provider Testing](http://developer.android.com/guide/topics/testing/contentprovider_testing.html), and [Service Testing](http://developer.android.com/guide/topics/testing/service_testing.html) each have a section entitled "What to Test" that lists possible testing areas.

When possible, you should run these tests on an actual device. If this is not possible, you can use the [Android Emulator](http://developer.android.com/guide/developing/devices/emulator.html) with Android Virtual Devices configured for the hardware, screens, and versions you want to test

**7.SCREENSHOT**

**S**

**8. CONCLUSION**

Here concluding that “Donate Food” is an helpful android application which can avails free and useful thing if it is go for publishing and reaches maximum Donators.

**9. BIBLOGRAPHY**

[www.developers.android.com](http://www.developers.android.com)

[www.androidtools.com](http://www.androidtools.com)

[www.stackoverflow.com](http://www.stackoverflow.com)