ABSTRACT

Communication is the key on any given day. Companies often try to communicate with their prospective customers through different methods, be it through newspapers, television, billboards or social networking sites. This often leads to encumbering of information about different products which the customer might actually ignore due to an excess of it which in no way does any benefit to both parties involved. So, what if the customers choose to know about a certain product and the organization tries to be in touch with interested customers. Present system companies are spending lakhs of rupees daily to promote their products, like it costs almost 1 lakh per/day to have an advertisement on a hoarding. In the similar way advertisements is only medium between the producer and consumer and getting the product into customer reach is something every company is focusing on but they are in need of all possible ways which leads to cost less promotion of their products.

What makes Fickle stand out is its zero cost business model. The customer who ever uses our app which even contains games, chats etc. other applications in order to reveal those features user must visit a retail store and get into the hotspot reach which in turn automatically delivers offers to user. As soon as offers were detected on the user's phone, features can be revealed. In an urgent position he may either forward that offer message to his friends to avail those offers. The hotspot provider need not spend any money on internet connection. The provider can let all the people connected with local area network (LAN) have the information about the offers, issues, warnings, feedback and customers can also ask questions.

The project is developed in Java Programming language by using Android studio (IDE). We use Android Software Development Kit (SDK) which includes in the variety of custom tools that help us to develop mobile applications on the Android platform. The most important of these are the Android Emulator, Firebase, etc.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

CHAPTER II

EXISTING SYSTEM AND PROPOSED SYSTEM

CHAPTER II

LITERATURE SURVEY

CHAPTER VII

IMPLEMENTATION

CHAPTER VIII

SOFTWARE TESTING

CHAPTER IX

CONCLUSION

CHAPTER X

FUTURE ENHANCEMENT

CHAPTER XI

REFERENCE

1. INTRODUCTION

1.1 Introduction

Communication is the key on any given day because it is how one can convey their feelings or ideas to others. Even companies try to do the same in order to sell their products. They often try to communicate with their prospective customers. In order to make the customer aware of their product they showcase their product features and usability. They may choose different methods, be it through newspapers, television, billboards or social networking sites to communicate product information to the customer. They choose these platforms because that is where customer spends his/her time, meanwhile they wanted to make customer aware at the same time. Thus one thing is common in all these ways of advertising, they actually wanted to distract the customer from his/her work and make him/her know about the product, which actually frustrates user while it might be during watching his favorite movie in television interrupting every 15 mins of break or colorful dashboard while we are driving or while playing games in mobiles. This often leads to encumbering of information about different products which the customer might actually ignore due to an excess of unwanted advertisements which in turn does any benefit to both parties involved. Why doesn't such Environment be created where both parties are willing to do the same. Why don't we make advertising as part of their lifestyle. In order to answer all these questions, we are introducing Fickle, an app which lets the customers know about the offers and many more for a product/business if he connects to the hotspot of the store.

1.2 Purpose

In present system, companies are spending lakhs of rupees daily to promote their products, in such a way that it costs almost 1 lakh per/day to have an advertisement on a hoarding. In the similar way advertisements are only medium between the producer and consumer through which the product information can be reached to customer. So every company is focusing on all the communication methods which leads to less expensive promotion of their products. So our idea is to serve that purpose.

2. EXISTING SYSTEM AND PROPOSED SYSTEM

2.1 Existing system

As advertising has evolved into a vastly complex form of communication, we can have thousands of different ways for a business to get a message to the consumer. Today's advertiser has a vast array of choices at his or her disposal. The Internet alone provides many of these, with the advent of branded viral videos, banners, advertorials, sponsored websites, branded chat rooms and so much more. These types are advertising is quite expensive.

Local stores which wanted to promote their products cannot afford to such high priced advertising.

2.2 Proposed System

Mobile phones have become a basic need for a man now-a-days. Everybody is having a smart phone. Hence, using mobile applications is one of the easier and promising medium of advertising.

Fickle, an android application, is also an advertising platform for the store owners to be in touch with their customers and attract them towards their promotional offers.

What makes Fickle stand out is, its zero cost business model. The customer who uses the application comes to know about the offers of the various products in the nearby stores. The store owner (hotspot provider) need not spend any money on internet connection, pamphlets or billboards. The store owner lets all the people nearby, to connect to his local area network (LAN), i.e. by enabling hotspot in his mobile phone, and broadcasts the information about the offers, issues, warnings, feedback and accept questionnaire from the customers.

So in this manner one will be having the information about various products, only when one needs them.

3. LITERATURE SURVEY

3.1 Why Develop for Mobile?

In market terms, the emergence of modern mobile smartphones - multifunction devices including a phone but featuring a full-featured web browser, cameras, media players, Wi-Fi, and location-based services — has fundamentally changed the way people interact with their mobile devices and access the Internet.

Mobile-phone ownership easily surpasses computer ownership in many countries, with more than 3 billion mobile phone users worldwide. 2009 marked the year that more people accessed the Internet

for the first time from a mobile phone rather than a PC. Many people believe that within the next 5 years more people will access the Internet by mobile phone rather than using personal computers.

The increasing popularity of modern smartphones, combined with the increasing availability of high speed mobile data and Wi-Fi hotspots, has created a huge opportunity for advanced mobile applications.

The ubiquity of mobile phones, and our attachment to them, makes them a fundamentally different platform for development from PCs. With a microphone, camera, touchscreen, location detection, and environmental sensors, a phone can effectively become an extra-sensory perception device.

Smartphone applications have changed the way people use their phones. This gives you, the application developer, a unique opportunity to create dynamic, compelling new applications that become a vital part of people's lives.

3.2 Android: An Open Platform for Mobile Development

More recently, Android has expanded beyond a pure mobile phone platform to provide a development platform for an increasingly wide range of hardware, including tablets and televisions.

Put simply, Android is an ecosystem made up of a combination of three components:

- A free, open-source operating system for embedded devices
- An open-source development platform for creating applications
- Devices, particularly mobile phones, that run the Android operating system and the applications created for it

More specifically, Android is made up of several necessary and dependent parts, including the following:

- A Compatibility Definition Document (CDD) and Compatibility Test Suite (CTS) that

describe the capabilities required for a device to support the software stack.

- A Linux operating system kernel that provides a low-level interface with the hardware, memory management, and process control, all optimized for mobile and embedded devices.
- Open-source libraries for application development, including SQLite, WebKit, OpenGL, and a media manager.
- A run time used to execute and host Android applications, including the Dalvik Virtual Machine (VM) and the core libraries that provide Android-specific functionality. The run time is designed to be small and efficient for use on mobile devices.
- An application framework that agnostically exposes system services to the application layer, including the window manager and location manager, databases, telephony, and sensors.
- A user interface framework used to host and launch applications.
- A set of core pre-installed applications.
- A software development kit (SDK) used to create applications, including the related tools, plug-ins, and documentation.

What really makes Android compelling is its open philosophy, which ensures that you can fix any deficiencies in user interface or native application design by writing an extension or replacement. Android provides you, as a developer, with the opportunity to create mobile phone interfaces and applications designed to look, feel, and function exactly as you imagine them.

3.2.1 Why Develop for Android?

Android represents a clean break, a mobile framework based on the reality of modern mobile devices designed by developers, for developers. With a simple, powerful, and open SDK, no licensing fees, excellent documentation, and a thriving developer community, Android represents an opportunity to create software that changes how and why people use their mobile phones.

The barrier to entry for new Android developers is minimal:

- No certification is required to become an Android developer.
- Google Play provides free, up-front purchase, and in-app billing options for distribution and monetization of your applications.
- There is no approval process for application distribution.
- Developers have total control over their brands.

From a commercial perspective, more than 850,000 new Android devices are activated daily,

with many studies showing the largest proportion of new smartphone sales belonging to Android devices.

As of March 2012, Google Play (formerly Android Market) has expanded its support for application sales to 131 countries, supporting more than 10 billion installs at a growth rate of 1 billion downloads per month.

Android sits alongside a new wave of modern mobile operating systems designed to support application development on increasingly powerful mobile hardware. Platforms like Microsoft's Windows Phone and the Apple iPhone also provide a richer, simplified development environment for mobile applications; however, unlike Android, they're built on proprietary operating systems. In some cases they prioritize native applications over those created by third parties, restrict communication among applications and native phone data, and restrict or control the distribution of third-party applications to their platforms.

3.2.2 Factors Driving Android's Adoption

Developers have always been a critical element within the Android ecosystem, with Google and the OHA betting that the way to deliver better mobile software to consumers is to make it easier for developers to write it.

As a development platform, Android is powerful and intuitive, enabling developers who have never programmed for mobile devices to create innovative applications quickly and easily. It's easy to see how compelling Android applications have created demand for the devices necessary to run them, particularly when developers write applications for Android because they can't write them for other platforms.

As Android expands into more form-factors, with increasingly powerful hardware, advanced sensors, and new developer APIs, the opportunities for innovation will continue to grow.

Open access to the nuts and bolts of the underlying system is what's always driven software development and platform adoption. The Internet's inherent openness has seen it become the platform for a multibillion-dollar industry within 10 years of its inception. Before that, it was open systems such

as Linux and the powerful APIs provided as part of the Windows operating system that enabled the explosion in personal computers and the movement of computer programming from the arcane to the mainstream.

This openness and power ensure that anyone with the inclination can bring a vision to life at minimal cost.

3.2.3 Android Applications

Android applications are usually developed in the Java language using the Android Software Development Kit. Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play** or the **Amazon Appstore**. Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fast. Every day more than 1 million new Android devices are activated worldwide. This tutorial has been written with an aim to teach you how to develop and package Android application. We will start from environment setup for Android application programming and then drill down to look into various aspects of Android applications.

3.3 Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

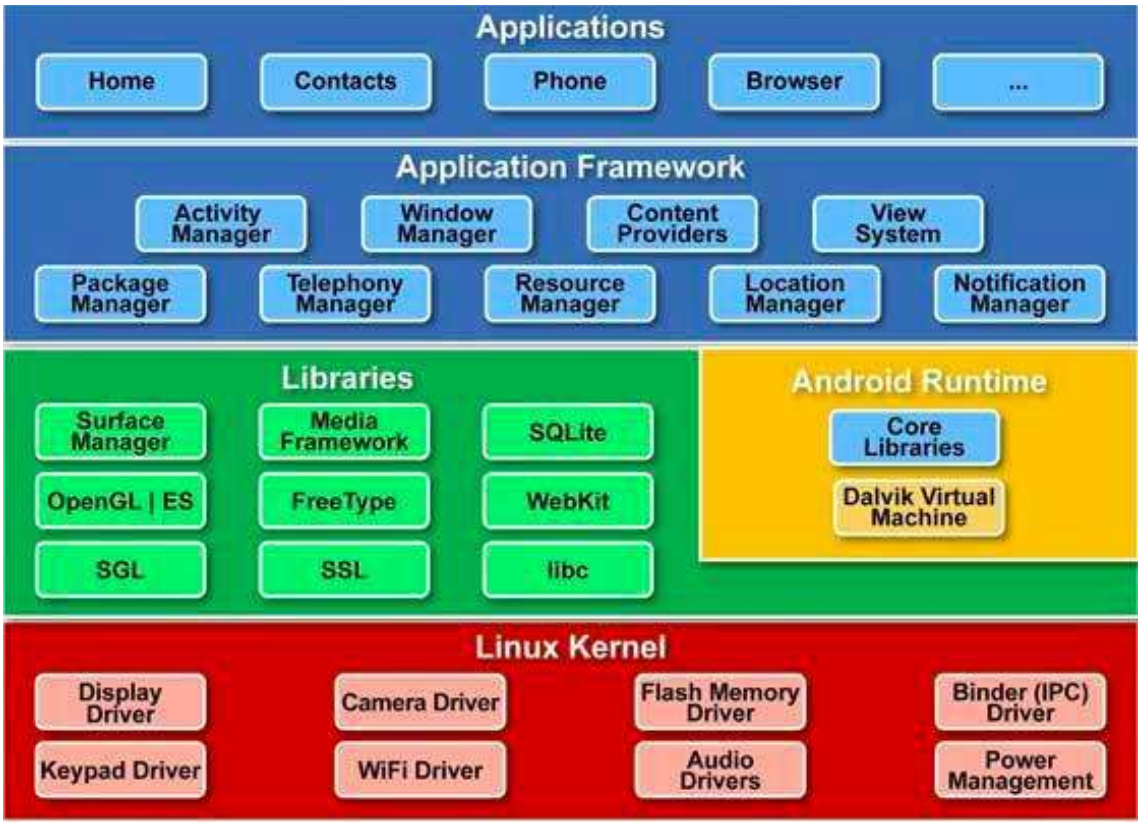

Figure 3.1 : Architecture of Android Operating System

3.3.1 Linux kernel

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device

management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

3.3.2 Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine Web Kit, well known library libs, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

3.3.3 Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Android Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android. The AVM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The AVM enables every Android application to run in its own process, with its own instance of the android virtual machine. The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

3.3.4 Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

3.3.5 Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

3.4 Android SDK Features

The true appeal of Android as a development environment lies in its APIs.
As an application-neutral platform, Android gives you the opportunity to create applications that are as much a part of the phone as anything provided out-of-the-box. The following list highlights some of the most noteworthy Android features:

- GSM, EDGE, 3G, 4G, and LTE networks for telephony or data transfer, enabling you to make or receive calls or SMS messages, or to send and retrieve data across mobile networks
- Comprehensive APIs for location-based services such as GPS and network-based location detection
- Full support for applications that integrate map controls as part of their user interfaces
- Wi-Fi hardware access and peer-to-peer connections
- Full multimedia hardware control, including playback and recording with the camera and microphone
- Media libraries for playing and recording a variety of audio/video or still-image formats
- APIs for using sensor hardware, including accelerometers, compasses, and barometers
- Libraries for using Bluetooth and NFC hardware for peer-to-peer data transfer
- IPC message passing
- Shared data stores and APIs for contacts, social networking, calendar, and multi-media
- Background Services, applications, and processes
- Home-screen Widgets and Live Wallpaper
- The ability to integrate application search results into the system searches
- An integrated open-source HTML5 Web Kit-based browser
- Mobile-optimized, hardware-accelerated graphics, including a path-based 2D graphics library and support for 3D graphics using OpenGL ES 2.0
- Localization through a dynamic resource framework
- An application framework that encourages the reuse of application components and the replacement of native applications

3.5Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project:

S. No.	Folder, File & Description
1	**src :** This contains the **.java** source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
2	**gen :** This contains the **.R** file, a compiler-generated file that references all the resources found in your project. You should not modify this file.
3	**bin :** This folder contains the Android package files **.apk** built by the ADT during the build process and everything else needed to run an Android application.

4	**res/drawable-hdpi :** This is a directory for drawable objects that are designed for high-density screens.
5	**res/layout :** This is a directory for files that define your app's user interface.
6	**res/values :** This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.
7	**AndroidManifest.xml :** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
8	**activity_main.xml :** layout file available that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application.

Table 3.1 : Important file and directories of an Android Project

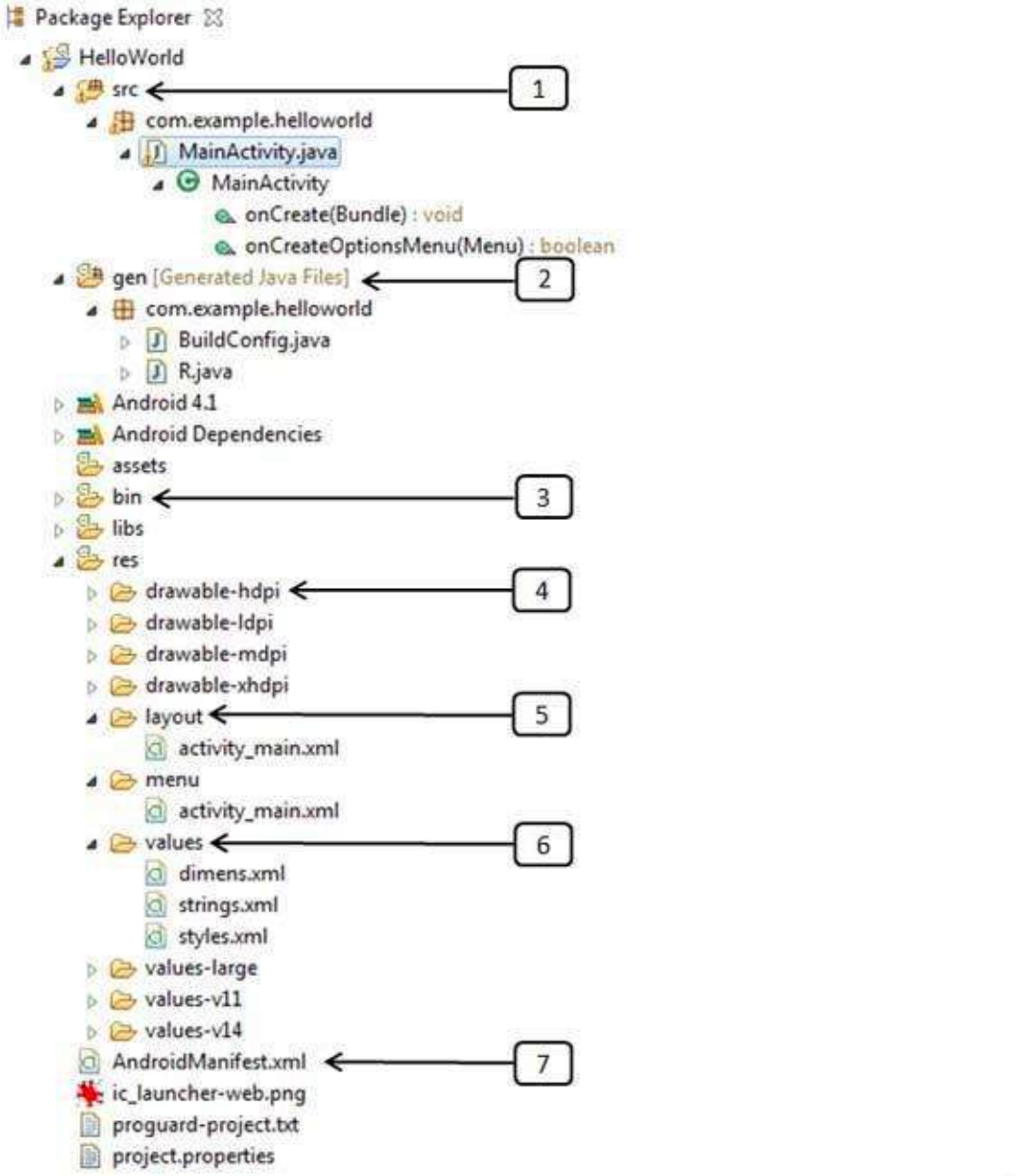

Figure 3.2 Anatomy of Android Application

3.5.1 Application Components

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact. There are following four main components that can be used within an Android application:

Components	Description
Activities	They dictate the UI and handle the user interaction to the smartphone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

Table 3.2 : Basic Android Application Components

3.5.2 Activities and Lifecycle methods

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.

An activity is implemented as a subclass of **Activity** class as follows:

```
public class MainActivity extends Activity { }
```

If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched. If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity lifecycle diagram:

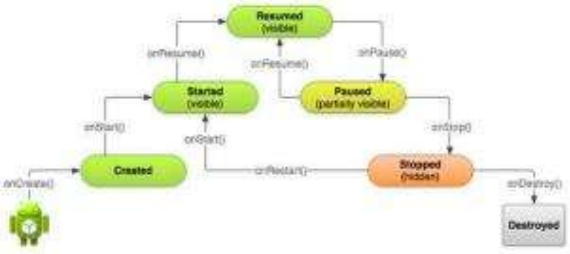

Figure 3.3 Activity Lifecycle

The Activity class defines the following callbacks i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Callback	Description
onCreate()	This is the first callback and called when the activity is first created.
onStart()	This callback is called when the activity becomes visible to the user.
onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.

Table 3.3 : Activity Lifecycle methods

3.5.3 Services

A service is a component that runs in the background to perform long-running operations without needing to interact with the user. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. A service is implemented as a subclass of **Service** class as follows:

```
public class MyService extends Service {}
```

A service can essentially take two states:

State	Description
Started	A service is **started** when an application component, such as an activity, starts it by

	calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
Bound	A service is **bound** when an application component binds to it by calling*bindService()*. 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).

Table 3.4 Android Services States

A service has lifecycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the lifecycle when the service is created with startService() and the diagram on the right shows the lifecycle when the service is created with bindService():

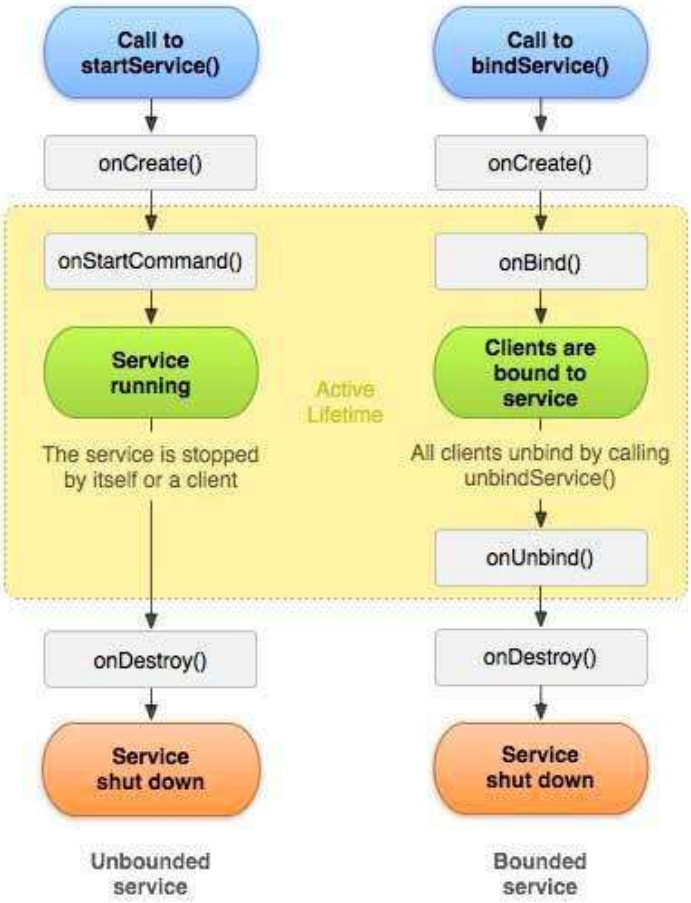

Figure 3.4 : Lifecycle of Services

To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Callback	**Description**
onStartCommand()	The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.
onBind()	The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*.
onUnbind()	The system calls this method when all clients have disconnected from a particular interface published by the service.
onRebind()	The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its*onUnbind(Intent)*.
onCreate()	The system calls this method when the service is first created usingonStartCommand() or onBind(). This call is required to perform one-time setup.
onDestroy()	The system calls this method when the

	service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

Table 3.5 Lifecycle methods of Services

3.5.4 Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action. A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcasted as an **Intent** object.

```
public class MyReceiver extends BroadcastReceiver { }
```

There are following two important steps to make BroadcastReceiver works for the systen broadcasted intents:

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.

Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in*AndroidManifest.xml* file. Consider we are going to register *MyReceiver* for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.

Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver *MyReceiver*and implemented logic inside *onReceive()* will be executed. There are several system generated events defined as final static fields in the **Intent** class. The following table lists a few important system events.

Event Constant	Description
android.intent.action.BATTERY_CHANGED	Sticky broadcast containing the charging state, level, and other information about the battery.
android.intent.action.BATTERY_LOW	Indicates low battery condition on the device.
android.intent.action.BATTERY_OKAY	Indicates the battery is now okay after being low.
android.intent.action.BOOT_COMPLETED	This is broadcast once, after the system has finished booting.
android.intent.action.BUG_REPORT	Show activity for reporting a bug.
android.intent.action.CALL	Perform a call to someone specified by the data.
android.intent.action.CALL_BUTTON	The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
android.intent.action.DATE_CHANGED	The date has changed.
android.intent.action.REBOOT	Have the device reboot.

Table 3.6 : System Events

Broadcasting Custom Intents

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the *sendBroadcast()* method inside your activity class. If you use the*sendStickyBroadcast(Intent)* method, the Intent is **sticky**, meaning the *Intent* you are sending stays around after the broadcast is complete.

3.5.5 Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely. A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends ContentProvider { }
```

Each Android applications runs in its own process with its own permissions which keeps an application data hidden from another application. But sometimes it is required to

share data across applications. This is where content providers become very useful. Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content usingg insert(), update(), delete(), and query() methods. In most cases this data is stored in an **SQlite** database. A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

Part	**Description**
prefix	This is always set to content://
authority	This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as*com.tutorialspoint.statusprovider*
data_type	This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this *content://contacts/people*
Id	This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this*content://contacts/people/5*.

Table 3.7 : Content providers

Create Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the *ContentProviderbase*class.

Second, you need to define your content provider URI address which will be used to access the content.

- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's databse. When your application is launched, the*onCreate()* handler of each of its Content Providers is called on the main application thread.

- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your acitivity file using <provider> tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working:

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

Components	**Description**
Fragments	Represents a behavior or a portion of user interface in an Activity.
Views	UI elements that are drawn onscreen including buttons, lists forms etc.
Layouts	View hierarchies that control screen format and appearance of the views.
Intents	Messages wiring components together.
Resources	External elements, such as strings, constants and drawables pictures.
Manifest	Configuration file for the application.

Table 3.8 : Additional Components

3.6 Network Programming

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols −

- **TCP** − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

- **UDP** − UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

This chapter gives a good understanding on the following two subjects −

- **Socket Programming** − This is the most widely used concept in Networking and it has been explained in very detail.

- **URL Processing** − This would be covered separately. Click here to learn about URL Processing in Java language.

3.6.1 Socket Programming

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets −

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.

- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a two-way communication protocol; hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.

3.6.2 ServerSocket Class Methods

The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests.

The ServerSocket class has four constructors –

S. No.	Method & Description
1	**public ServerSocket(int port) throws IOException** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
2	**public ServerSocket(int port, int backlog) throws IOException**

	Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
3	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on.
4	**public ServerSocket() throws IOException** Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.

Table 3.9 : ServerSocket Constructors

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

Following are some of the common methods of the ServerSocket class −

Sr.No.	Method & Description
1	**public int getLocalPort()** Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2	**public Socket accept() throws IOException** Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out

	value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.
3	**public void setSoTimeout(int timeout)** Sets the time-out value for how long the server socket waits for a client during the accept().
4	**public void bind(SocketAddress host, int backlog)** Binds the socket to the specified server and port in the SocketAddress object. Use this method if you have instantiated the ServerSocket using the no-argument constructor.

Table 3.10 : Methods of ServerSocket class

When the ServerSocket invokes accept(), the method does not return until a client connects. After a client does connect, the ServerSocket creates a new Socket on an unspecified port and returns a reference to this new Socket. A TCP connection now exists between the client and the server, and communication can begin.

3.6.3 Socket Class Methods

The **java.net.Socket** class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

The Socket class has five constructors that a client uses to connect to a server -

S. No.	Method & Description
1	**public Socket (String host, int port) throws UnknownHostException, IOException.**

	This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
2	**public Socket(InetAddress host, int port) throws IOException** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.
3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** Connects to the specified host and port, creating a socket on the local host at the specified address and port.
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String.
5	**public Socket()** Creates an unconnected socket. Use the connect() method to connect this socket to a server.

Table 3.11 Socket class Constructors

When the Socket constructor returns, it does not simply instantiate a Socket object but it actually attempts to connect to the specified server and port.

Some methods of interest in the Socket class are listed here. Notice that both the client and the server have a Socket object, so these methods can be invoked by both the client and the server.

S. No.	Method & Description
1	**public void connect(SocketAddress host, int timeout) throws IOException** This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor.
2	**public InetAddress getInetAddress()** This method returns the address of the other computer that this socket is connected to.
3	**public int getPort()** Returns the port the socket is bound to on the remote machine.
4	**public int getLocalPort()** Returns the port the socket is bound to on the local machine.
5	**public SocketAddress getRemoteSocketAddress()** Returns the address of the remote socket.
6	**public InputStream getInputStream() throws IOException** Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7	**public OutputStream getOutputStream() throws IOException** Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

8	**public void close() throws IOException** Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

Table 3.12 : Methods of Socket Class

3.6.4 InetAddress Class Methods

This class represents an Internet Protocol (IP) address. Here are following usefull methods which you would need while doing socket programming −

S. No.	Method & Description
1	**static InetAddress getByAddress(byte[] addr)** Returns an InetAddress object given the raw IP address.
2	**static InetAddress getByAddress(String host, byte[] addr)** Creates an InetAddress based on the provided host name and IP address.
3	**static InetAddress getByName(String host)** Determines the IP address of a host, given the host's name.
4	**String getHostAddress()** Returns the IP address string in textual presentation.
5	**String getHostName()** Gets the host name for this IP address.
6	**static InetAddress InetAddress getLocalHost()**

	Returns the local host.
7	**String toString()** Converts this IP address to a String.

Table 3.13 Methods of InetAdress class

4. FEASIBILITY STUDY

4.1 Introduction :

A *feasibility study* is used to determine the viability of an idea. The objective of such a study is to ensure a project is operationally and technically and economically feasible. It tells us whether a project is worth the investment or not.

It study evaluates the project's potential for success; therefore, perceived objectivity is an important factor in the credibility of the study for potential investors and lending institutions.

1. **Operational Feasibility -** this involves undertaking a study to analyze and determine whether your business needs can be fulfilled by using the proposed solution. It also measures how well the proposed system solves problems and takes advantage of the opportunities identified during scope definition. Operational feasibility studies also analyze how the project plan satisfies the requirements identified in the requirements analysis phase of system development. To ensure success, desired operational outcomes must inform and guide design and development. These include such design-dependent parameters such as reliability, maintainability, supportability, usability, disposability, sustainability, affordability, and others.

2. **Technical Feasibility -** assessment is centered on the technical resources available to the organization. It helps organizations asess if the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves evaluation of the hardware and the software requirements of the proposed system.

3. **Economic Feasibility -** helps organizations assess the viability, cost, and benefits associated with projects before financial resources are allocated. It also serves as an independent project assessment, and enhances project credibility, as a result. It helps decision-makers determine the positive economic benefits to the organization that the proposed system will provide, and helps quantify them. This assessment typically involves a cost/ benefits analysis of the project.

The Feasibility Study for our project is as follows:

4.2 Operational Feasibility

The application is quite easy to learn and fun to use due to its simple and attractive interface. User requires no special training for operating the application.

The application is pleasing to the customers to use as it would help them to know the various products' information and promotional offers only at the time they wanted. By the way, the information is also present in the device which is always nearer and dearer to the customers now-a-days.

4.3 Technical Feasibility

The technical requirement for the system is economic and it does not use any other additional hardware and software apart from

- a mobile phone
- with Android OS
- with hotspot feature (for the store owners)
- with Wi-Fi connectivity (for the customers)

4.4 Economic Feasibility

Fickle makes it possible for even small scale store owners to promote their business and offers with a small cost; in fact, no cost these days.

The application helps the store owners to reduce their cost incurred on huge hoardings, bill boards, pamphlets and internet agencies for advertising. Thus, it is cost effective in the sense that it reduces the cost of manpower, paper and broadcasting which is incurred during conventional advertising.

5. SYSTEM ANALYSIS

System analysis involves a detailed study of the current system , Leading to specification of a new system.System Analysis is a detailed study of various operations performed by a system and their relationships within and outside the system.During analysis, data are collected on the available files.decision points and transactions handled by the present system .Interviews, on-site observation and questionnaire are the tools used for system analysis.Using the following steps it becomes easy to draw the exact boundary of the new system under considerations:

- Keeping in view the problems and new requirements.
- Workout the pros and cons including new areas of the system.

All procedures, requirements must be analysed and documented in the form of detailed data flow diagrams(DFD's) , data dictionary ,Logical data structures and miniatures specifications. system analysis also includes sub- dividing of complex process involving the entire system, identification of data store and manual processes.

The main points to be discussed in system analysis area

- Specification of what the new system is to accomplish based on the user requirements.
- Functional hierarchy showing the functions to be performed by the new system and relationship with each-other.
- Function network which are similar to function hierarchy but they highlight the function which are common to more than one procedure.
- List of attributes of the entities -these are the data items which need to be held about each entity(Record).

5.1 REQUIREMENTS SPECIFICATION

A Requirement specification for a software system is a complete description of the behavior of the system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. Non - functional requirements are requirements which impose constraints on the design or implementation.

5.1.1 Functional requirements

Functional requirements define a function of a software system or its component . A function is described as a set of inputs , the behaviour , and outputs. functional requirements may be calculations, technical details, data manipulations and processing and other specific

functionality that define what a system is supposed to accomplish. Behavior requirements describing all the cases various system uses the functional requirements are captured in use cases.

5.1.1.1 Store Owner Module

As the name suggests, this module is used by the store owner. The store owner firstly has to register and provide the location details of the store. Then he/she can add or remove the offers. He/she have to activate the offers he/she wants to notify his/her customers.

5.1.1.2 Customer Module

This module is used by any person who wants the information about a product or offers regarding them. The customer should also register with his/her personal details. The customer to get the notifications from the store owners, have to enable the Wi-Fi feature in his/her mobile phone.

5.1.2 Non-Functional requirements

Non - functional requirements are requirements that specify criteria that can be used to judge the operation of the system , rather than specific behaviors. this should be contrasted with functional requirements that define specific behavior or functions.

Non-functional requirements are often called qualities of the system. Other terms for non - functional requirements are "constraints ", "quality attributes", "quality goals", "quality of service requirements " and " non-behavioral requirements" can be divided into two main categories:

Execution qualities, such as security, usability, reusability and performance which are observable at run time.

Evolution qualities, such as maintainability, extensibility and portability, which are embedded into static structure of a software system.

5.1.2.1 Execution qualities

- ✓ **Security**

Password based security is offered to users to keep their data safe and secrecy secure. Security of the application can be ensured as a code is not visible to end users so it cannot be tampered, by others.

- ✓ **Usability**

The application is easy to use. Even a laymen can use the application via the graphical user interface provided.

- ✓ **Reusability**

The application can be reused any number of times by any number of users without any technical difficulty. It can also be reused and modified or extended to develop similar but different applications.

✓ **Performance**

The response time, utilization and throughput behavior of the system are optimized and care is taken so as to ensure the system with comparatively high performance.

5.1.2.2 Evolution qualities

✓ **Maintainability**

All modules are clearly designed and implemented. Through thoughtful and effective software engineering, all steps of the software development process are well documented to ensure maintainability of the application through its lifetime.

✓ **Extensibility**

The ease with which software can accommodate changes is modifiability or extensibility. The application is easily adaptable to changes that are useful to it withstand the needs of the users. The extra functionality can be provided through either internally or externally coded extensions.

✓ **Portability**

The application can be deployed under different android OS versions starting 4 onwards. The only restriction to portability is that this application can be deployed only on a device that supports android OS.

5.2. System Requirements

The resources needed to run the application effectively are :

5.2.1 Hardware requirements

Android mobile/tablet with

- Processor: 1.5 GHz Dual core or more
- Hard disk: 32GB
- RAM: 3GB

5.2.2 Software Requirements

- Device OS: Android Marshmallow
- SDK: Android SDK 6.0
- IDE: Android Studio

5.3 SYSTEM ARCHITECTURE

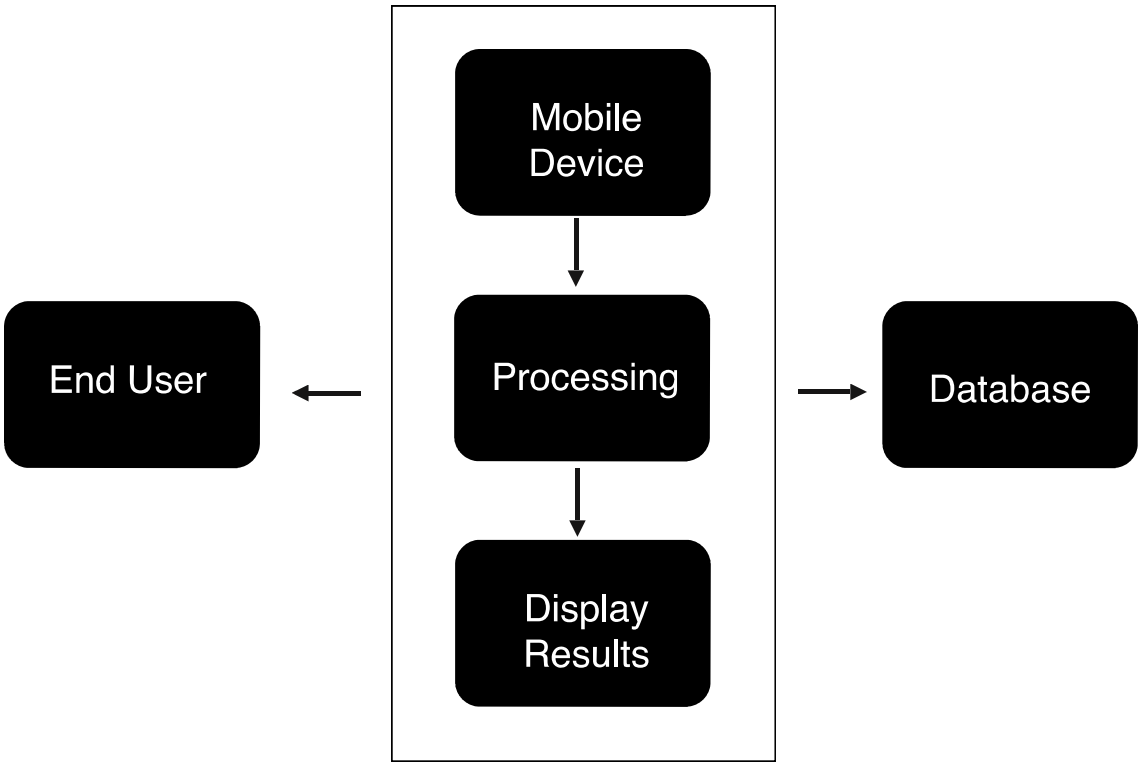

Figure 5.1 shows system architecture of the entire system for Fickle

The end users, that is the customer as well as the store owner, install the application in their mobile devices. They enter their necessary details which are stored in the database. When the store owner updates his/her offers and activates it using the hotspot, the advertisements are displayed to the customer and can be availed.

6. DESIGN

6.1 INTRODUCTION

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages. Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

6.2 UML DIAGRAMS

6.2.1 Class Diagram

The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application.

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages.

The purpose of the class diagram is to model the static view of an application.

The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a *structural diagram*.

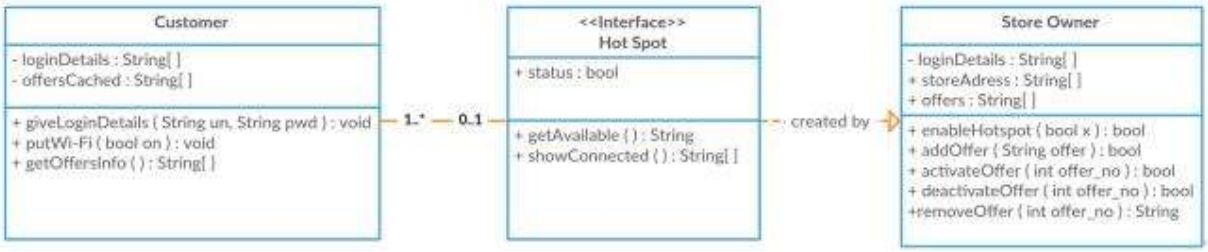

Figure 6.1 : Class Diagram

6.2.2 Use Case Diagram

To model a system the most important aspect is to capture the dynamic behaviour. To clarify a bit in details, *dynamic behaviour* means the behaviour of the system when it is running or operating.

The use case diagrams consist of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.

The purpose of use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

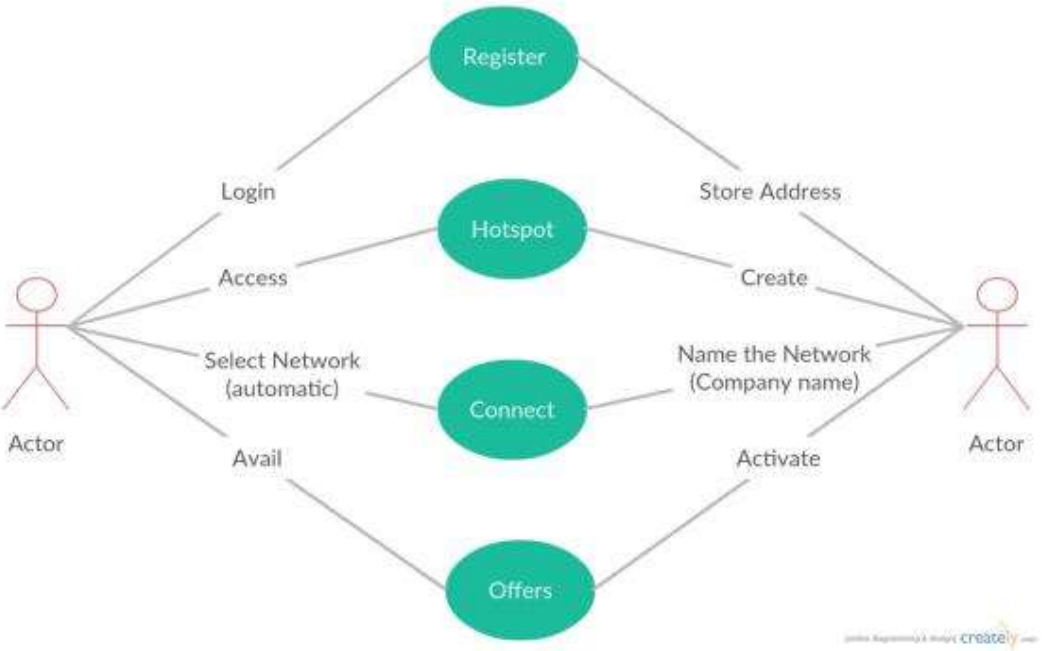

Figure 6.2: Use case Diagram

6.2.3 Sequence Diagram

The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case. With the advanced visual modeling capability, you can create complex sequence diagram in few clicks. Besides, Visual Paradigm can generate sequence diagram from the flow of events which you have defined in the use case description.

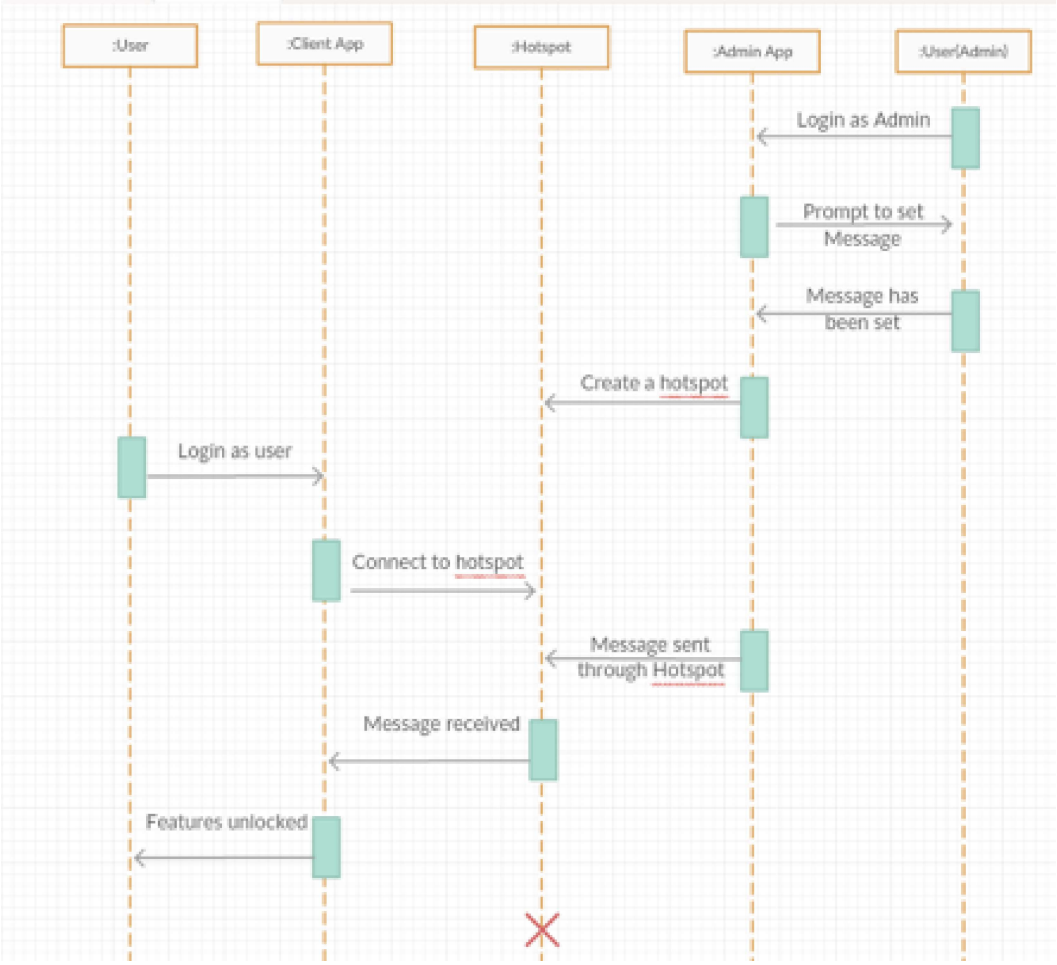

Figure 6.3: Sequence Diagram

6.2.4 State chart Diagram

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A Statechart diagram describes a state machine. Now to clarify it state machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of Statechart diagram is to model life time of an object from creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model reactive system.

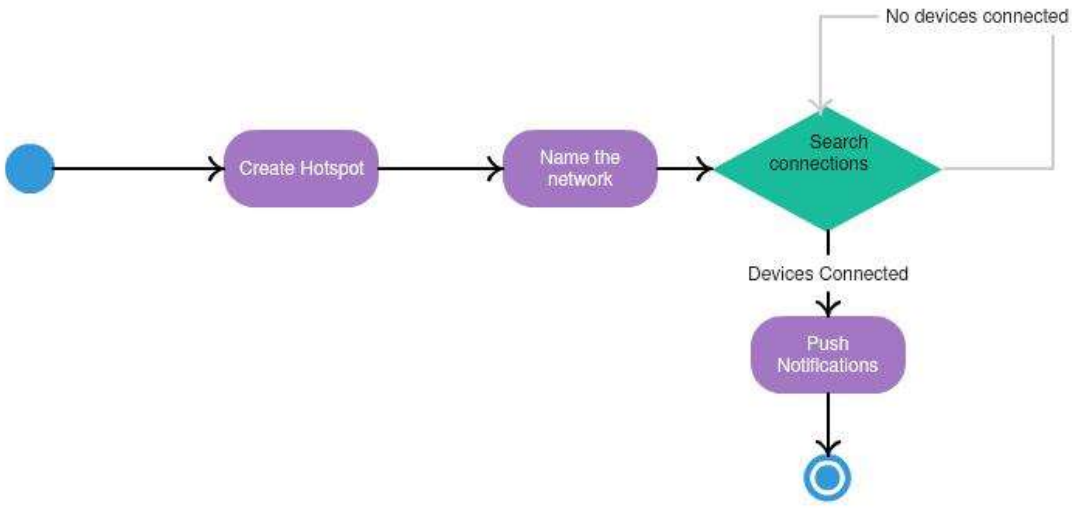

Figure 6.4: State chart Diagram

6.2.5 Deployment Diagram

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.

So deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

The name *Deployment* itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components where software components are deployed. Component diagrams and deployment diagrams are closely related.
Deployment diagrams are made to focus on hardware topology of a system. Deployment diagrams are used by the system engineers.

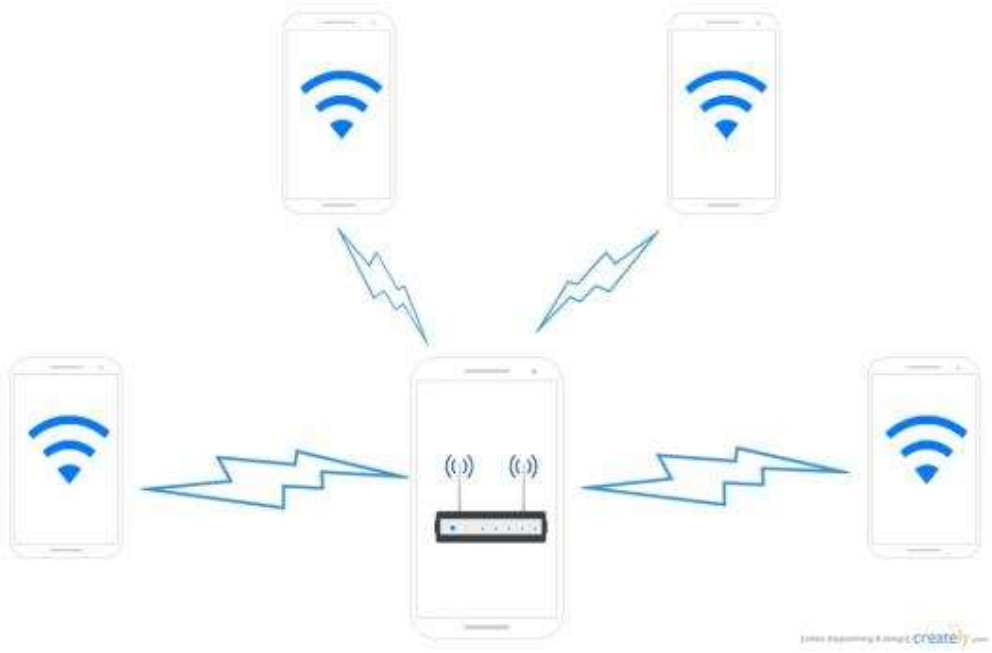

Figure 6.5: Deployment Diagram

7. IMPLEMENTATION

7.1 CODE FOR IMPLEMENTATION START ACTIVITY

In AddressActivity.java we store the store address and save it in app data for further usage . In order to Store data we are using Shared Preference concept and Corresponding XML file which displays the look of it.

AddressActivity.java

```
public class AddressActivity extends AppCompatActivity {
    ImageView imgv;
    EditText add;
    private TextInputLayout input_tid;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.store_address);

overridePendingTransition(R.anim.trans_left_in,R.anim.trans_left_out);
        imgv = (ImageView) findViewById(R.id.addr_image);
        add = (EditText) findViewById(R.id.address_text);
        input_tid = (TextInputLayout) findViewById(R.id.input_add);
    }
    public void onClickAddress(View view){
        String addr = add.getText().toString();
                  SharedPreferences sharedPref =
         this.getPreferences(Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPref.edit();
        editor.putString("address", addr);
        editor.commit();
        validate();


    }
    public void validate(){
        String address= add.getText().toString();
        if(address.isEmpty()){
            input_tid.setError("Invalid Email");
            return;
        }else {
            input_tid.setErrorEnabled(false);
```

```
        }
        Intent i= new Intent(this,OfferActivity.class);
        startActivity(i);
    }
    public void onBackPressed() {
        super.onBackPressed();
        overridePendingTransition(R.anim.trans_right_in,
R.anim.trans_right_out);
    }
}
```

store_address.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <include layout="@layout/toolbar"/>
        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@color/colorBack">
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                android:id="@+id/linearLayout"
                android:layout_marginTop="62dp"
                android:layout_alignParentTop="true"
                android:layout_alignParentLeft="true"
                android:layout_alignParentStart="true">
                <android.support.v7.widget.CardView
                    android:id="@+id/card_view"
                    android:layout_gravity="center"
                    android:layout_width="match_parent"
                    android:layout_height="250dp"
                    android:layout_marginLeft="20dp"
                    android:layout_marginRight="20dp"
```

```
    card_view:cardCornerRadius="8dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:paddingRight="20dp"
        android:paddingLeft="20dp"
        android:paddingBottom="10dp"
        android:paddingTop="40dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:weightSum="4"
        android:orientation="vertical">
<android.support.design.widget.TextInputLayout
            android:layout_width="match_parent"
            android:id="@+id/input_add"
            android:layout_weight="1"
            android:layout_height="fill_parent">
    <EditText
        android:hint="Enter your store address here!"
        android:lines="8"
        android:lineSpacingExtra="5dp"
        android:textSize="14dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginBottom="10dp"
        android:gravity="top|left"
        android:id="@+id/address_text" />
</android.support.design.widget.TextInputLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="fill_parent"
            android:layout_gravity="center"
            android:gravity="center"
            android:layout_weight="3">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="15dp"
            android:text="Enter"
            android:clickable="true"
```

```
                            android:onClick="onClickAddress"
                            android:textColor="#BE3929"
                            />
                        </LinearLayout>
                    </LinearLayout>
                    </LinearLayout>
                </android.support.v7.widget.CardView>
            </LinearLayout>

            <de.hdodenhof.circleimageview.CircleImageView
                android:id="@+id/addr_image"
                android:layout_width="66dp"
                android:layout_height="66dp"
                android:clipToPadding="false"
                android:src="@drawable/airtel"
                card_view:civ_border_width="0dp"
                card_view:civ_fill_color="@color/cardview_dark_background"
                android:layout_marginLeft="38dp"
                android:layout_marginStart="38dp"
                app:civ_border_width="2dp"
                android:elevation="8dp"
                android:layout_marginTop="30dp"
                android:layout_alignParentTop="true"
                android:layout_alignParentLeft="true"
                android:layout_alignParentStart="true" />
        </RelativeLayout>
</LinearLayout>
```

Login activity stores information about store in app data

———> loginActivity.java

```
public class LoginActivity extends AppCompatActivity {
    private int PICK_IMAGE_REQUEST = 1;
    private EditText fullname,email,phno;
    Boolean flag_img=Boolean.FALSE;
    Bitmap bitmap;
    TextView warn;
    private TextInputLayout inp_email,inp_full,inp_phno;
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.store_login);

getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_AL
WAYS_HIDDEN);
        fullname = (EditText) findViewById(R.id.fullname);
        email = (EditText) findViewById(R.id.emailid);
        phno = (EditText) findViewById(R.id.phno);
        inp_email =(TextInputLayout) findViewById(R.id.input_email);
        inp_phno =(TextInputLayout) findViewById(R.id.input_phno);
        inp_full =(TextInputLayout) findViewById(R.id.input_fullname);
        warn = (TextView) findViewById(R.id.warn_upload);
    }
    @Override
          protected void onActivityResult(int requestCode, int resultCode,
      Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

              if (requestCode == PICK_IMAGE_REQUEST && resultCode ==
       RESULT_OK && data != null && data.getData() != null) {

            Uri uri = data.getData();

            try {
     bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), uri);
                // Log.d(TAG, String.valueOf(bitmap));

       ImageView imageView = (ImageView) findViewById(R.id.profile_image);
                imageView.setImageBitmap(bitmap);
                flag_img=true;
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public void onClickUploadImage(View view){
        Intent intent = new Intent();
// Show only images, no videos or anything else
        intent.setType("image/*");
        intent.setAction(Intent.ACTION_GET_CONTENT);
```

```
// Always show the chooser (if there are multiple options available)
startActivityForResult(Intent.createChooser(intent, "Select Picture"),
PICK_IMAGE_REQUEST);
    }
    public void onClickLogin(View v){

        validate();
    }
    public void validate(){
        String emaili = email.getText().toString();
        String fullnamei = fullname.getText().toString();
        String phnoi = phno.getText().toString();
        if(!flag_img){
            warn.setVisibility(View.VISIBLE);
            return;
        }else {
            warn.setVisibility(View.GONE);

        }
        if(fullnamei.isEmpty()){
            inp_full.setError("Enter your Name!");
            return;
        }else{
            inp_full.setErrorEnabled(false);
        }
        if(phnoi.isEmpty()){
            inp_phno.setError("Enter your Phone No!");
            return;
        }else{
            inp_phno.setErrorEnabled(false);
        }
        if(emaili.isEmpty() || !isValidEmail(emaili)){
            inp_email.setError("Invalid Email");
            return;
        }else{
            inp_email.setErrorEnabled(false);

        }
                SharedPreferences sharedPref =
         this.getPreferences(Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPref.edit();
        editor.putString("fullname", fullnamei);
```

```
        editor.putString("phno", phnoi);
        editor.putString("email", emaili);
        editor.commit();
        Intent i= new Intent(this,AddressActivity.class);
        startActivity(i);

    }
    private static boolean isValidEmail(String email) {
        return !TextUtils.isEmpty(email) &&
android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches();
    }

}
```

store_login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">
    >
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#EEEEEE"

    >
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:background="#EEEEEE"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingTop="80dp"
    android:orientation="vertical">
    <de.hdodenhof.circleimageview.CircleImageView
        android:id="@+id/profile_image"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:src="@drawable/upload_image"
        android:layout_marginTop="9dp"
        android:layout_gravity="center"
        android:clickable="true"
        android:onClick="onClickUploadImage"
        app:civ_border_width="0dp"
        android:elevation="8dp"
         />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:gravity="center"
        android:textColor="#e42c2c"
        android:id="@+id/warn_upload"
        android:visibility="gone"
        android:text="Upload image!"
        />

<android.support.design.widget.TextInputLayout
    android:layout_width="match_parent"
    android:paddingTop="20dp"
    android:id="@+id/input_fullname"
    android:layout_height="wrap_content">
    <EditText
        android:layout_width="match_parent"
        android:layout_marginTop="10dp"
        android:id="@+id/fullname"
        android:inputType="text"
        android:lines="1"
        android:layout_height="wrap_content"
        android:hint="Full Name"
```

```
        />
</android.support.design.widget.TextInputLayout>
    <android.support.design.widget.TextInputLayout
        android:layout_width="match_parent"
        android:id="@+id/input_phno"
        android:layout_height="wrap_content">

    <EditText
        android:layout_width="match_parent"
        android:layout_marginTop="20dp"
        android:id="@+id/phno"
        android:inputType="phone"
        android:lines="1"
        android:layout_height="wrap_content"
        android:hint="Phone No"
     />
</android.support.design.widget.TextInputLayout>

<android.support.design.widget.TextInputLayout
        android:layout_width="match_parent"
    android:id="@+id/input_email"
        android:layout_height="wrap_content">
    <EditText
        android:layout_width="match_parent"
        android:layout_marginTop="10dp"
        android:id="@+id/emailid"
        android:lines="1"
        android:inputType="textEmailAddress"
        android:layout_height="wrap_content"
        android:hint="email ID"
        />
</android.support.design.widget.TextInputLayout>



<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Next"
        android:onClick="onClickLogin"
        android:layout_marginTop="10dp"
        android:textColor="#BE3929"
```

```
            android:layout_gravity="center"/>
    </LinearLayout>
</LinearLayout>
</ScrollView>
    <include layout="@layout/toolbar"/>

</android.support.design.widget.CoordinatorLayout>
```

In offerActivity.java previous offers are loaded so that he can make them live for cutomers

offerActivity.java

```
public class OfferActivity extends AppCompatActivity {
    EditText off;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.store_offer);
                overridePendingTransition(R.anim.trans_left_in,
         R.anim.trans_left_out);
        off = (EditText) findViewById(R.id.offer_text);

    }

    public void onBackPressed() {
        super.onBackPressed();
            overridePendingTransition(R.anim.trans_right_in,
     R.anim.trans_right_out);
    }
    public void onClickOffer(View view){
        String offer = off.getText().toString();
                SharedPreferences sharedPref =
         this.getPreferences(Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPref.edit();
        editor.putString("offer", offer);
        editor.commit();
        Intent i= new Intent(this,MainActivity.class);
                i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
         IntentCompat.FLAG_ACTIVITY_CLEAR_TASK);
        startActivity(i);
        overridePendingTransition( R.anim.trans_slide_in_up,
R.anim.trans_slide_out_up );
```

```
    }
}
```

store_offer.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <include layout="@layout/toolbar"/>


    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@color/colorBack">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical"
            android:id="@+id/linearLayout"
            android:layout_marginTop="62dp"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true">

            <android.support.v7.widget.CardView
                android:id="@+id/card_view"
                android:layout_gravity="center"
                android:layout_width="match_parent"
                android:layout_height="300dp"
                android:layout_marginLeft="20dp"
                android:layout_marginRight="20dp"
                card_view:cardCornerRadius="8dp"
                >
                <LinearLayout
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:paddingRight="20dp"
android:paddingLeft="20dp"
android:paddingBottom="10dp"
android:paddingTop="55dp"
>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="4"
    android:orientation="vertical">
    <android.support.design.widget.TextInputLayout
        android:layout_width="match_parent"
        android:id="@+id/input_add"
        android:layout_weight="1"
        android:layout_height="fill_parent">
        <EditText
            android:hint="Enter Offer Details here”
            android:lines="8"
            android:lineSpacingExtra="5dp"
            android:textSize="14dp"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginBottom="10dp"
            android:gravity="top|left"
            android:id="@+id/offer_text" />
    </android.support.design.widget.TextInputLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:layout_gravity="center"
        android:gravity="center"
        android:layout_weight="3"
        >
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="15dp"
            android:onClick="onClickOffer"
```

```
                                android:text="Save"
                                android:textColor="#BE3929"
                                />
                        </LinearLayout>?
                    </LinearLayout>
                </LinearLayout>
            </android.support.v7.widget.CardView>
        </LinearLayout>

        <de.hdodenhof.circleimageview.CircleImageView
            android:id="@+id/profile_image"
            android:layout_width="66dp"
            android:layout_height="66dp"
            android:clipToPadding="false"
            android:src="@drawable/airtel"
            card_view:civ_border_width="0dp"
            card_view:civ_fill_color="@color/cardview_dark_background"
            android:layout_marginLeft="38dp"
            android:layout_marginStart="38dp"
            app:civ_border_width="0dp"
            android:elevation="8dp"
            android:layout_marginTop="30dp"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true" />
    </RelativeLayout>
</LinearLayout>
```

In MainActivity.java we are using recycler view to display list of previous offers.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private List<MainModel> offers;
    RecyclerView rv;
    EditText off;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
        rv=(RecyclerView)findViewById(R.id.recycler_view);
        off = (EditText) findViewById(R.id.offer_text);
        LinearLayoutManager llm = new LinearLayoutManager(this);
        rv.setLayoutManager(llm);
        MainAdapter adapter = new MainAdapter(offers);
        rv.setAdapter(adapter);
        initializeData();
        initializeAdapter();
    }
    private void initializeData(){
        offers = new ArrayList<>();
        offers.add(new MainModel("\n" +
                "Amazing Recharge Offers on MyAirtel App\n" +
                "+ Upto 3.5% Cashback From CouponDunia ",true));
        offers.add(new MainModel("Get amazing offers on International
Roaming Smartpicks. Offer is valid only for postpaid sims.",false));
        offers.add(new MainModel("Get amazing offers and promo codes on
various transactions made via Airtel Money. Offer valid for prepaid
recharges only.",false));
    }
    private void initializeAdapter(){
        MainAdapter adapter = new MainAdapter(offers);
        rv.setAdapter(adapter);
    }
    public void onClickNew(View view){

    }
}
```

activity_main.xml

```
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">
```

```
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:background="@color/colorBack"
    android:layout_height="match_parent">
    <include layout="@layout/toolbar"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:layout_marginTop="10dp">

        <android.support.v7.widget.RecyclerView
            android:id="@+id/recycler_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:clipToPadding="false"
            android:scrollbars="vertical" />
    </LinearLayout>
</LinearLayout>
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:background="?attr/selectableItemBackground"
        android:clickable="true"
        android:onClick="onClickNew"
        android:layout_margin="16dp"
        android:src="@drawable/ic_add_white_24dp"
       />

</android.support.design.widget.CoordinatorLayout>
```

————> main_list_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:paddingRight="3dp"
    android:paddingLeft="3dp"
    android:paddingBottom="8dp">

    <android.support.v7.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:cardCornerRadius="8dp"
        android:id="@+id/cv"
        >

       <LinearLayout
           android:layout_width="match_parent"
           android:layout_height="match_parent"
           android:orientation="horizontal"
           android:weightSum="5"
           android:padding="16dp">
           <TextView
               android:layout_width="match_parent"
               android:layout_height="fill_parent"
               android:layout_weight="1"
               android:id="@+id/dis_offer"
               android:lineSpacingExtra="4dp"
               android:text="STAR Sports India brings to you all…/>
           <LinearLayout
               android:layout_width="match_parent"
               android:layout_height="fill_parent"
               android:gravity="center"
               android:layout_weight="4">
               <LinearLayout
                   android:layout_width="match_parent"
                   android:layout_height="match_parent"
                   android:gravity="center"
                   android:layout_gravity="center"
                   android:id="@+id/llive"
                   android:orientation="vertical"
                   />
```

```
                <ImageView
                    android:layout_width="20dp"
                    android:layout_height="20dp"
                    android:layout_gravity="center"
                    android:id="@+id/stat"
                    android:paddingBottom="4dp"
                    android:visibility="visible"
                    android:src="@drawable/live_off"/>
                   <TextView
                       android:layout_width="wrap_content"
                       android:visibility="invisible"
                       android:id="@+id/ststtt"
                       android:layout_height="wrap_content"
                       android:text="LIVE"/>
               </LinearLayout>
           </LinearLayout>
       </LinearLayout>

    </android.support.v7.widget.CardView>

</LinearLayout>
```

MainAdapter.java it is used in keeping the data in each view of recycler view

7.2 OUTPUT SCREENS

CLIENT SCREENS

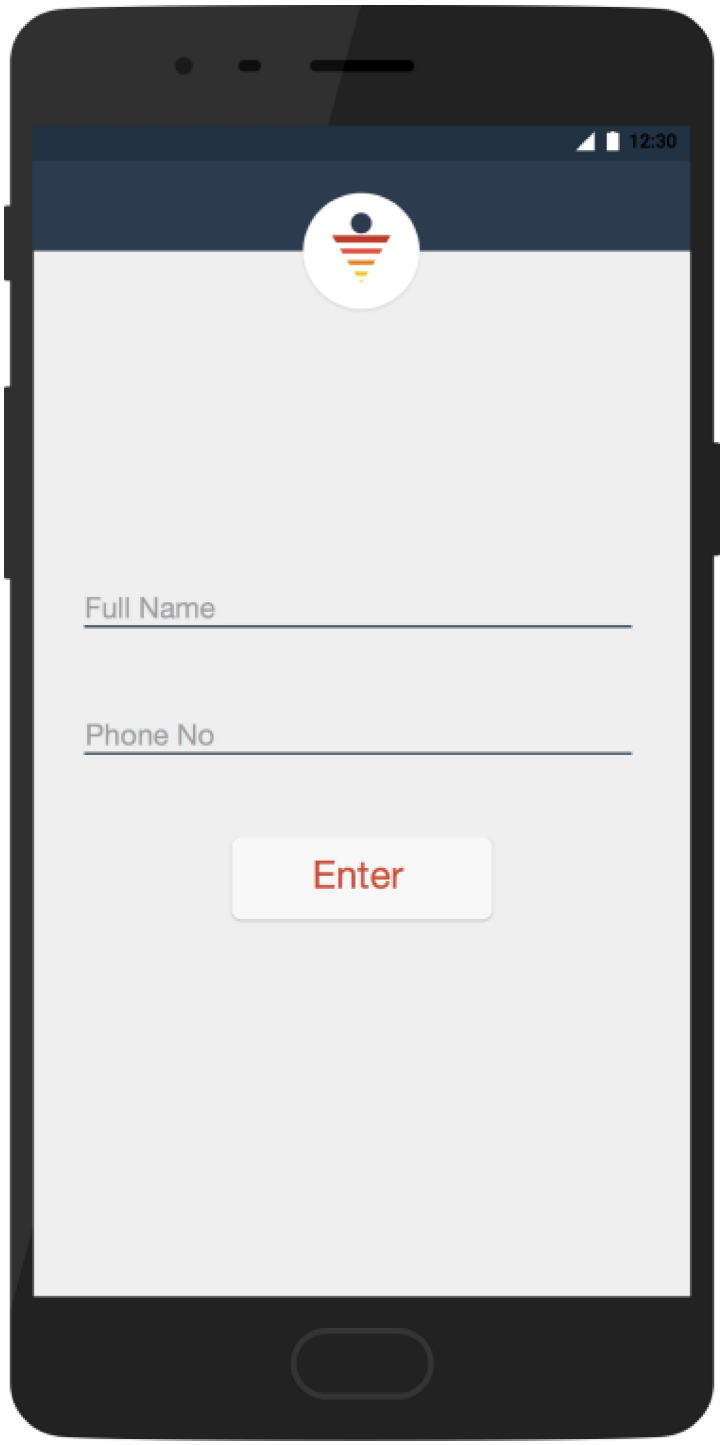

Fig.7.1 Client Login Screen

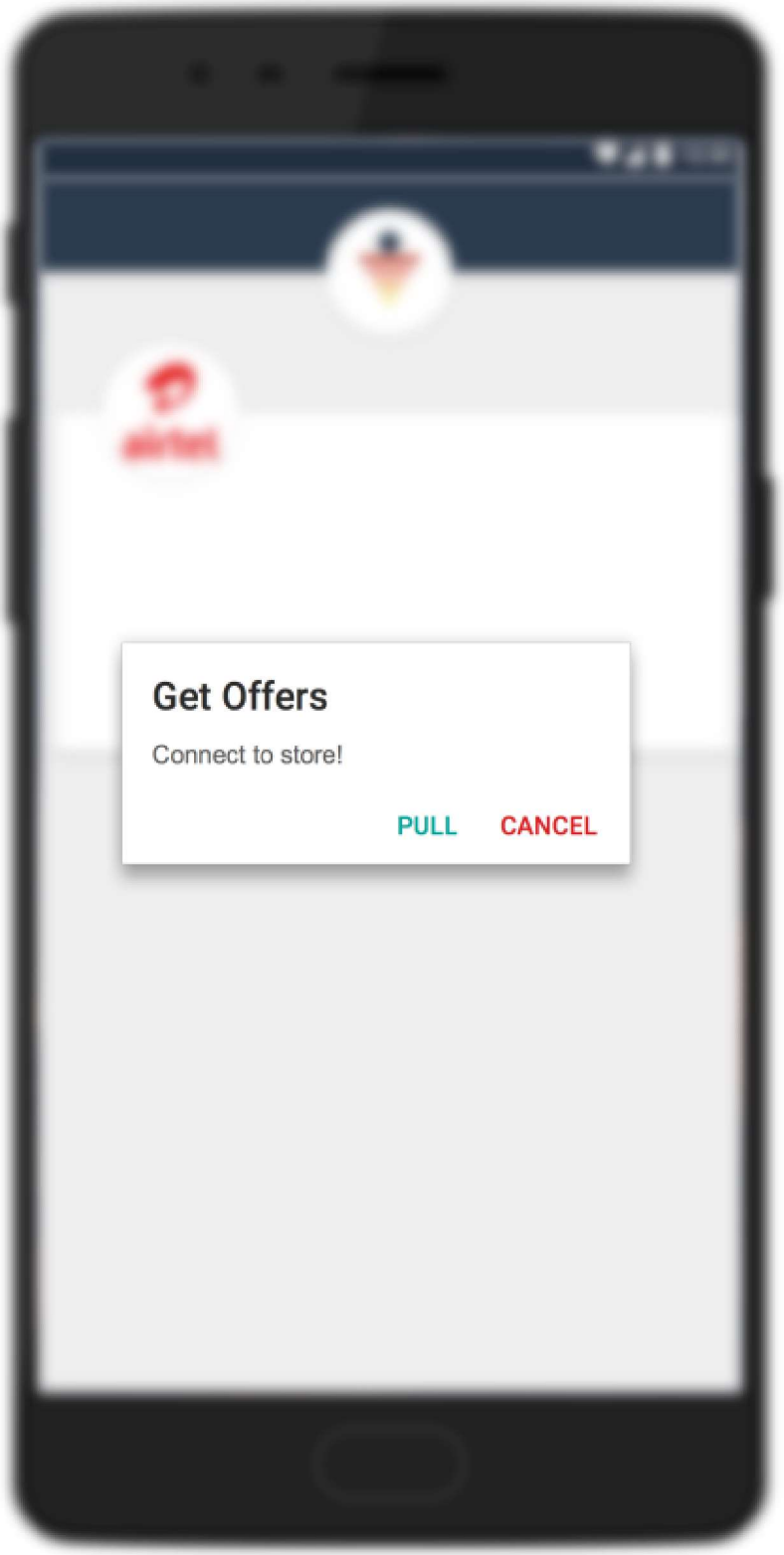

Fig.7.2 Client Prompt to turn Wi-Fi On

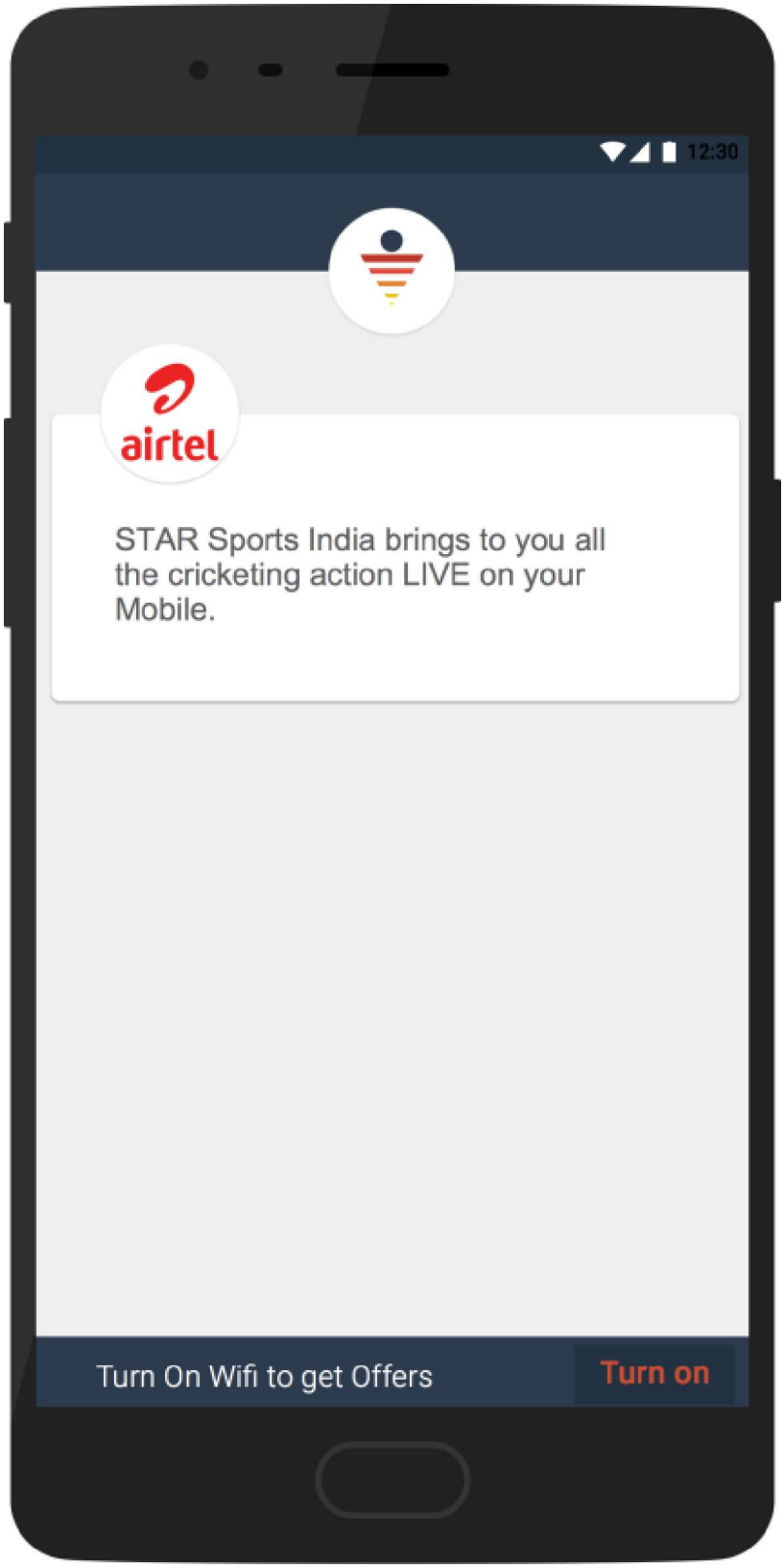

Fig.7.3 Main Home Screen

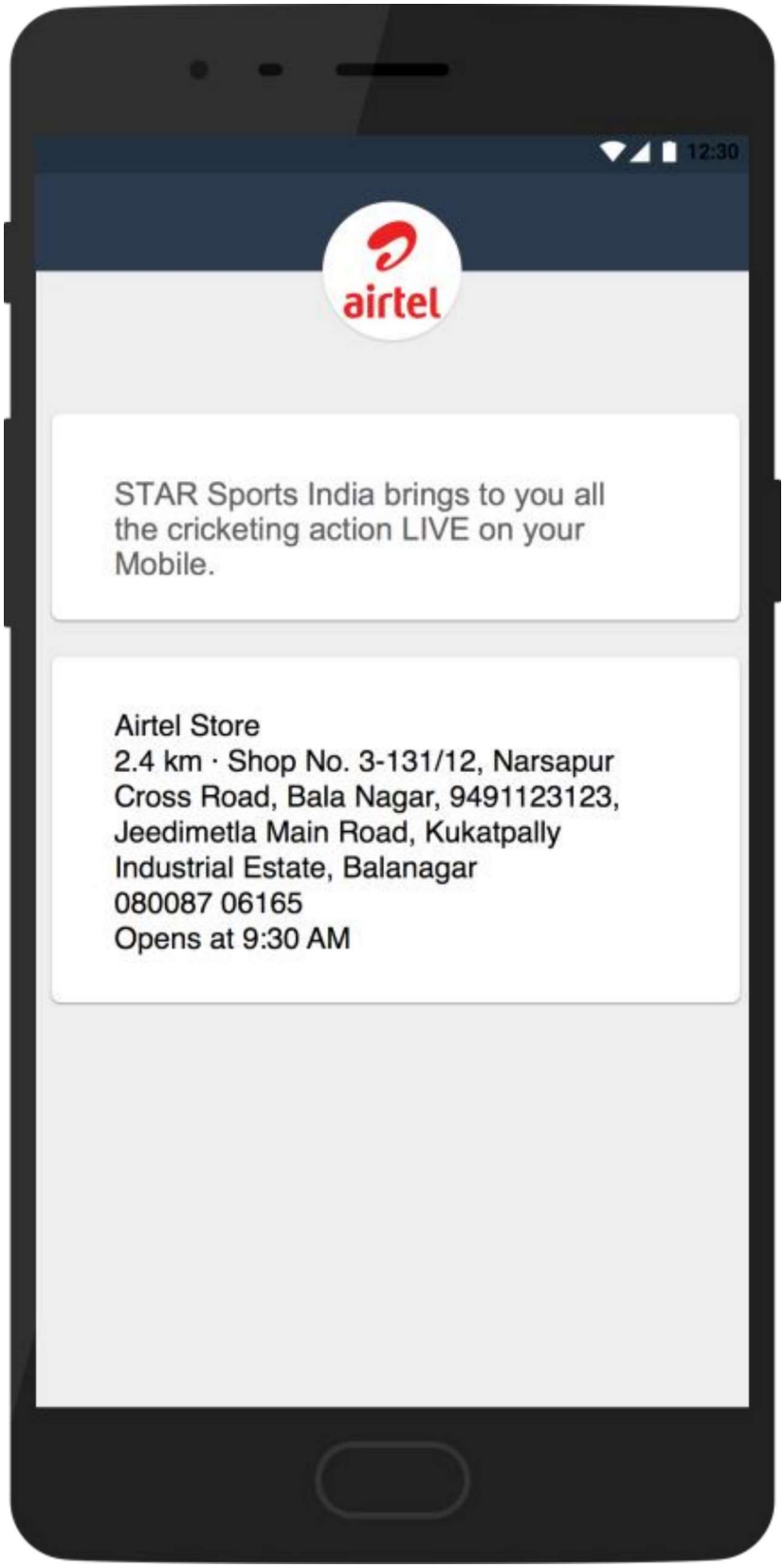

Fig.7.4 Client store Details

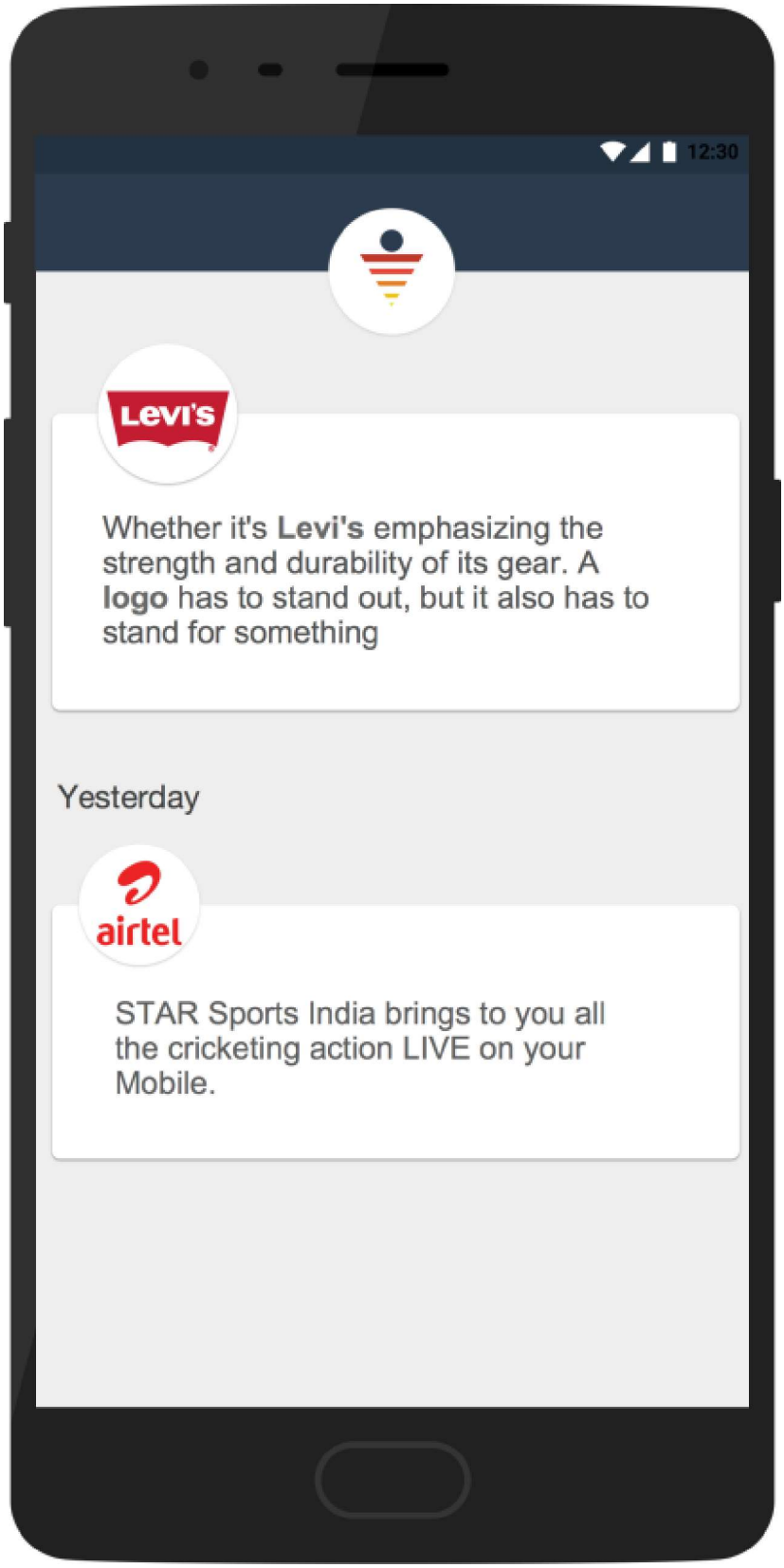

Fig.7.5 Client Home

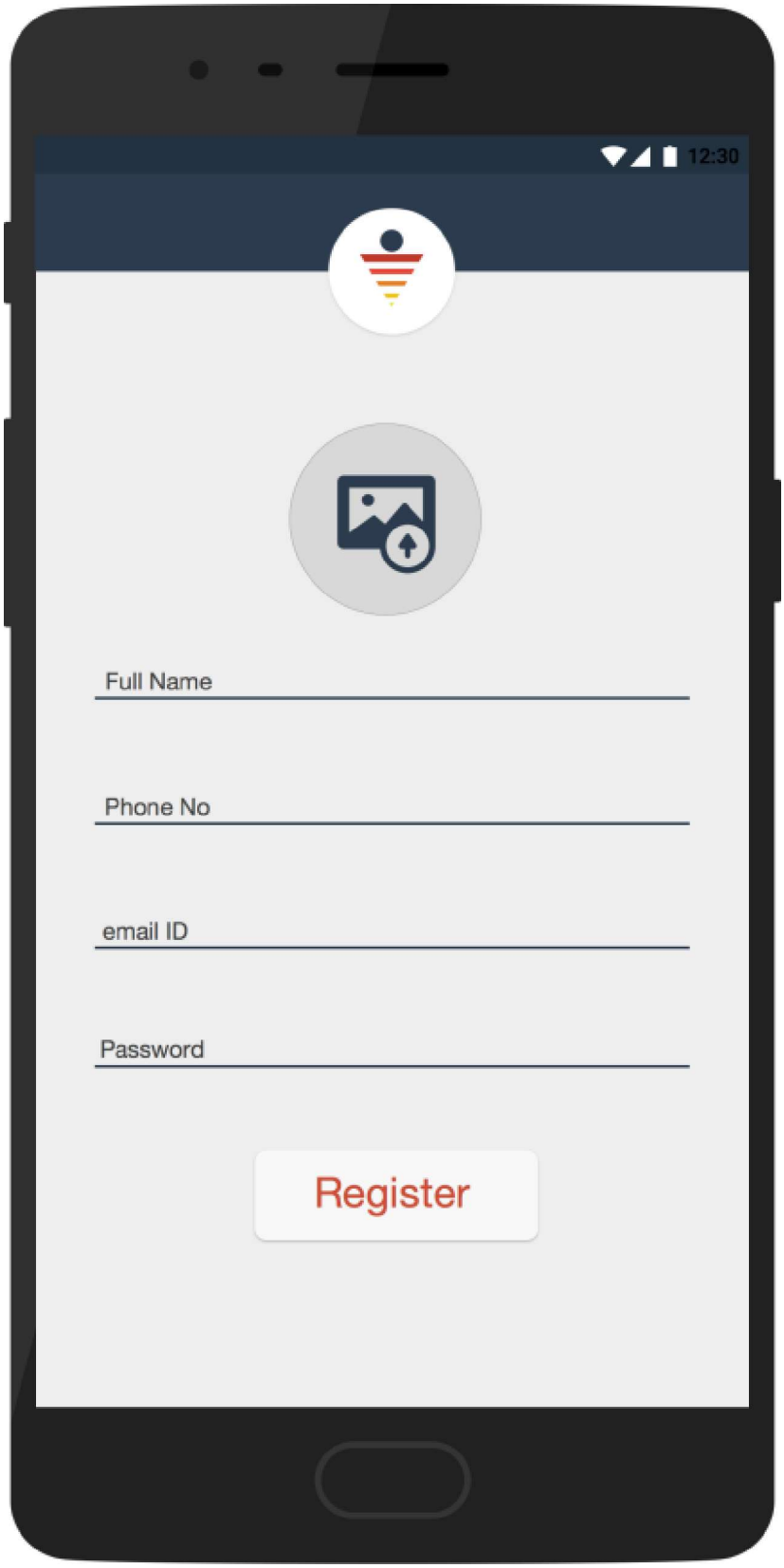

Fig.7.6 Server Login Page

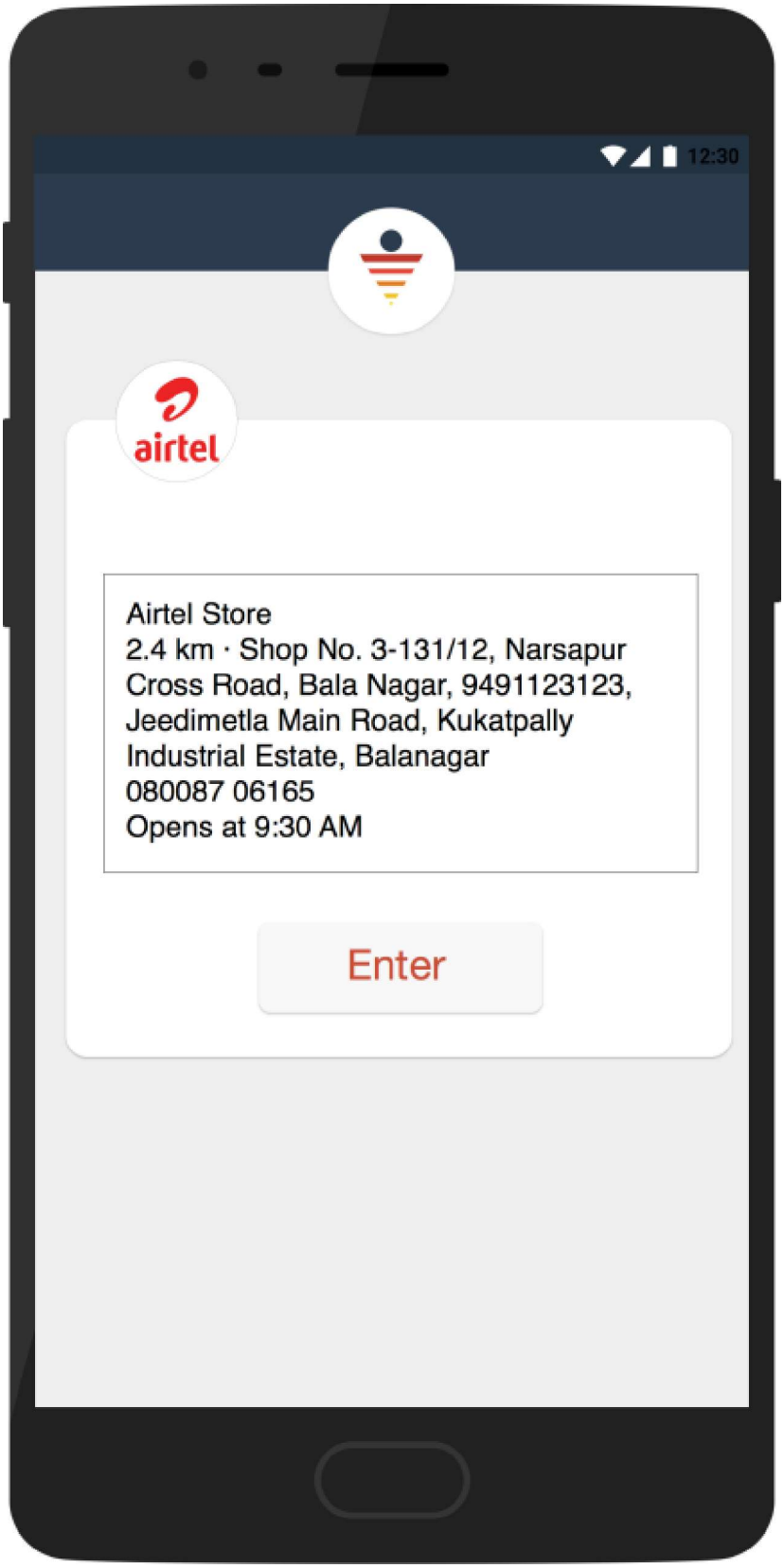

Fig.7.7 Store Address page

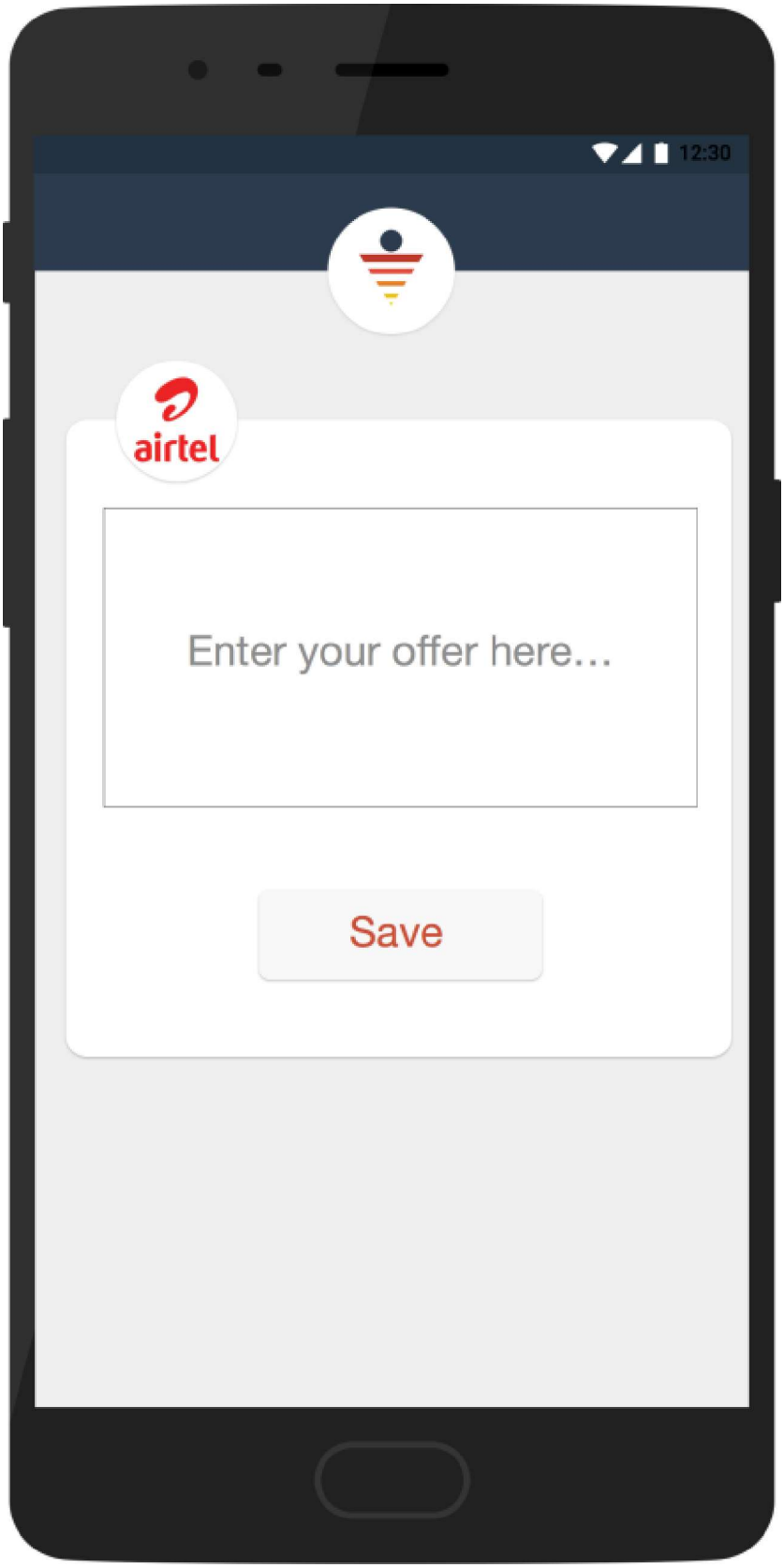

Fig.7.8 Store Offer Page

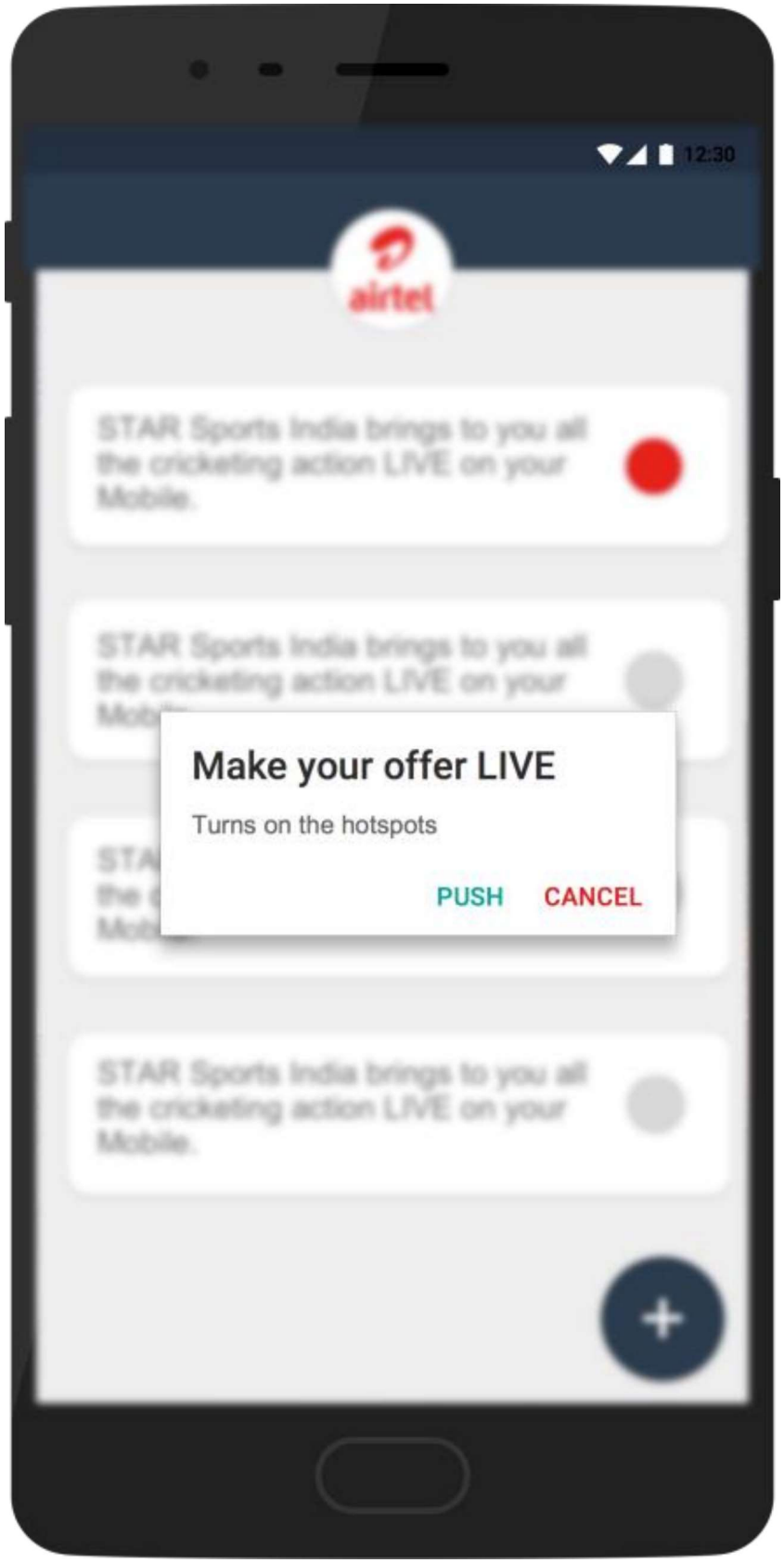

Fig.7.9 Store Prompt to Turn Hotspot On

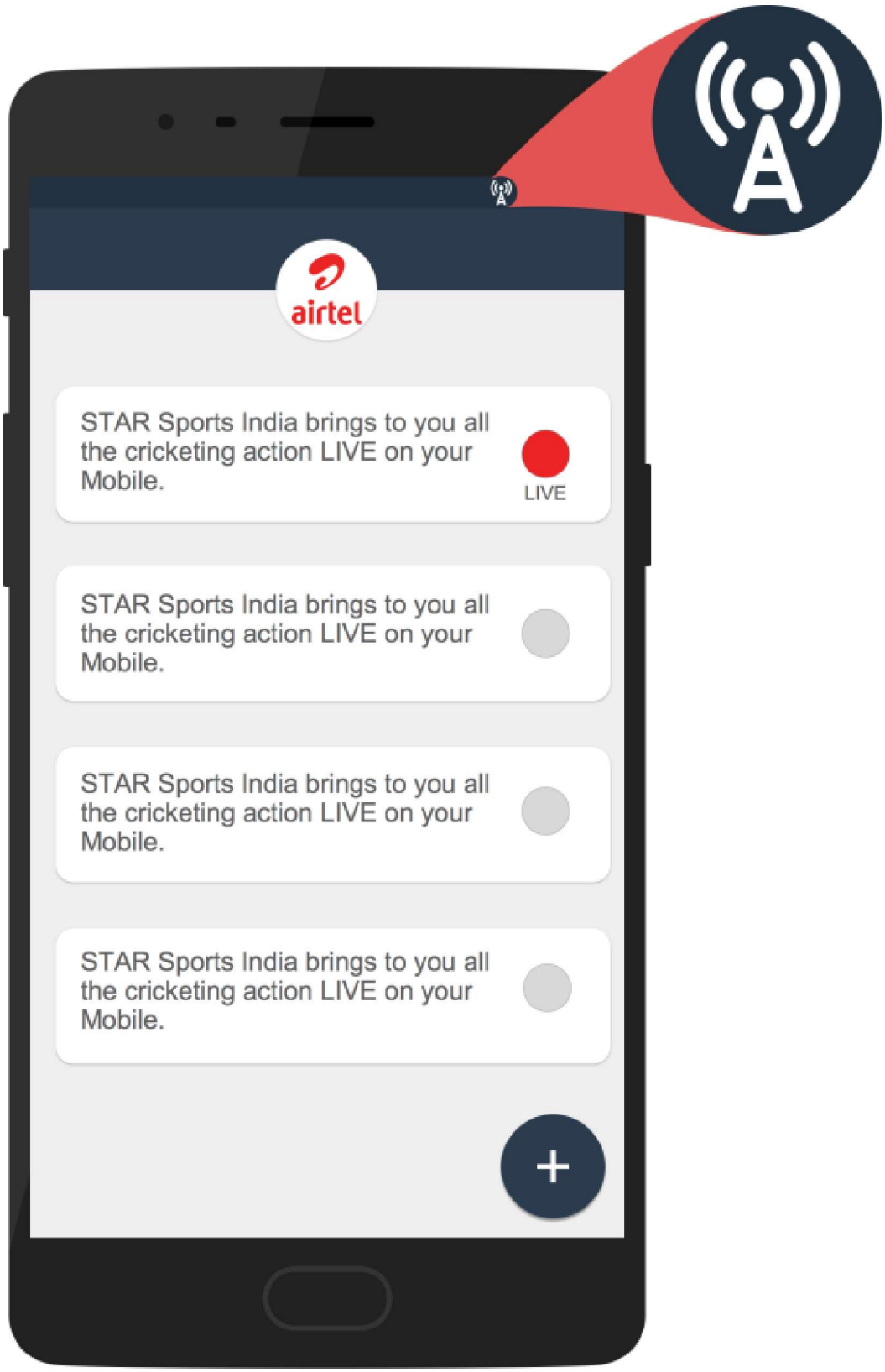

Fig.8.0 Store Hotspot Turned on

8. SOFTWARE TESTING

8.1 Introduction

Testing Methodologies

Testing is the process of finding differences between the expected behavior specified by system models and the observed behavior implemented system. From modelling point of view, testing is the attempt of falsification of the system with respect to the system models. The goal of testing is to design tests that exercise defects in the system and to reveal problems.

The process of executing a program with intent of finding errors is called testing. During testing. The program to be tested is executed with a set of test cases , and the output of the program for the test cases is evaluated to determine if the program is performing as expected . Testing forms the first step in determining the errors in the program. The success of testing in revealing errors in program depends critically on test cases.

Strategic Approach to Software Testing:

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirements analysis where the information domain. Functions, behavior, performance, constraints and validation criteria for software are established. moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decreases the level of abstraction on each item. A Strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing will progress by moving outward along the spiral to integration testing, where the focus on the design and the concentration of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.

Each individual module has been tested against the requirement with some test data.

8.2 Test Report:

The applications is working properly, provided the user has to enter valid information. All data entry forms have tested with specified test cases and all data entry forms are working properly.

The application ran successfully and passed all test cases in the Android Virtual Machine.The application also is working successfully in Android Mobile devices

Micromax Canvas, Xiaomi Note 3, Yureka Plus and on versions “Ice cream Sandwich” and above.

8.3 Error Report:

If the user does not enter data in specified order then the user will be prompted with error messages. Error handling was done to handle the expected and unexpected errors.

- The application will not work on non-android mobiles
- The application is not working for devices not having Wi-Fi feature.
- The application is not working properly for devices having versions prior to Icecream Sandwich.

8.4 Test cases

A Test case is a set of input data and expected results that exercises a component with the purpose of causing failure and detecting faults. Test case is an explicit set of instructions designed to detect a particular class of defect in a software system, by bringing about a failure.

A Test case can give rise to many tests.

Test cases can be divided in to two types. First one is Positive test cases and second one is negative test cases. In positive test cases are conducted by the developer intention is to get the output. In negative test cases are conducted by the developer intention is to don’t get the output.

S.NO	Test Case	Expected	Actual	Result
1	Store Owner registering in App	Enters full name, Phone no, email id and uploads image of store	Entered Results are Successfully getting stored in App data	Test case Passed
2	Store person Enters Offer details	Offer details are taken in a text box and stored.	Entered Offer is stored and shown in offers list	Test case Passed
3	Pushing the offers to customers by making the offer online	Store person clicks on available offers and presses push to make offers available for customers	Hotspot of store gets on and its is ready to send the offers as soon as customers start connecting to it.	Test case Passed
4	Customer Registers in App	Enters full name, Phone no.	Entered Results are Successfully getting stored in App data	Test case Passed
5	Getting Offers from near by stores.	Customers Once opens app, it prompts to pull the new offers.On clicking pull Offers get received.	Customer phone connects to Hotspot of store through wifi and gets the offer on mobile.	Test case Passed
6	Customer checking Previous Offers	Customer Once opens the app, it prompts either to pull or cancel. On clicking cancel he can see previous offers	previous Offers have been Already stored as customer doesn't wants to pull any new, old offers will be shown	Test case Passed

Table 8.1 : Test cases

9. Conclusion

Fickle is an android mobile application that allows the big and small scale stores to promote their products and promotional offers to their customers. The project is developed using Java programming language by using the Android Studio and Android Software Development Kit (SDK). At the front end we used xml layouts, events and event handlers. At the networking end, we used the Wifi-Direct technology, i.e., a smart phone can be used to act as a router, accept other devices to connect to it and also exchange data between them. This is done by enabling the hotspot feature in the mobile phone.
During testing android emulators are used to verify the test cases.

The goals that are achieved by the application are :

- ✓ Zero cost business model
- ✓ Instant Access
- ✓ User Friendly
- ✓ Scalable
- ✓ Flexible for further enhancements

10. Future Expansion

We are planning to have an emotional chat application, LAN games, quizzes etc., which attracts people to bridge the interaction gap between the store owners and their customers and hence using our application. This application can be transformed so that it can to be used in hospitals where the doctor is alerted whenever there is an emergency to the patient. Similarly, it can be used in technical fests where different events are organised at different buildings and floors, so that the event organisers can send the information about their event to the participant who is nearby.

" The opportunities this app provides are beyond measure and it can only improve with the increase in user base from different walks of life. "

11. REFERENCES

Books

1. Name: Professional Android 4 Application Development
   Author: Reto Meier
   Publisher : Wiley India

2. The Complete Reference to Java 9th edition
   Author: Herbert Schildt
   Publisher: Oracle Press (McGraw Hill)
3. Learn Java for Android Development: Java 8 and Android 5 Edition
   Author: Jeff Friesen
   Publisher: Apress

Journals

1. International Journal of Multimedia and Ubiquitous Engineering :
   Research and Development of Mobile Application Development for Android Platform by Li Ma, Lei Gu and Jin Wang.
2. International Journal of Computer Trends and Technology :
   Android based Mobile Application Development and its Security by Suhas Holla, Mahima M Katti

Websites

- https://developer.android.com
- https://developer.google.com
- https://firebase.google.com
- https://www.tutorialspoint.com/android