INTRODUCTION

1.1 ABOUT THE PROJECT

The project aims to design and develop an application which provides user with optimized job search based on his location. The purpose is to design an application which automates the processes involved and allows users to perform various operations.

1.2 OBJECTIVE

The objective is to develop an application which provides the details of various job opportunities user has based on his location. This allows user to search or post a job as per his requirement.

When the user provides us with his details and logs in into the application, We will display a list of all the available jobs present in our database and help him to find an optimal job for his need according to his skillset.

2. SYSTEM ANALYSIS

2.1 PROPOSED SYSTEM

As the application provides opportunities in a wide range of areas one can have a huge scope of earning.

The requirements for doing a job are not specific as the tasks can be unprofessional. This feature of the application helps users in earning money just with their interest among the numerous opportunities that are available to them.

On the other hand the users who post the advertisements to get their job done will also find this application a good platform to make their task easy.

Unlike the other applications the user can post the advertisements from the very basic unprofessional things like babysitting, gardening, tutoring, etc to the most professional jobs. In a way this application also acts like classifieds.

2.3 SCOPE OF THE PROJECT

One of the fastest growing industries nowadays is mobile industry. There are many competitors in this area who are doing research and development on new platforms & user experience. One such technology is Android from Google which is supported for Google phones. These phones are described as next Generation mobiles [As described by Google]. This project is mainly developed for providing an easy communication between the management and the client.

This project can also be upgraded by adding new modules and functionalities to the application and making it be par with various upcoming trends in the mobile computing industry.

2.4 MODULES DESCRIPTION

There are particularly three modules present in this system. They are :

1) Do it User Module.
2) GetitDone User Module.
3) Server Module.

1. GetitDone Module

In this module, we have a user who can login into the application using credentials provided by him at the time of registration. He logs into the application and posts a new advertisement which contains all the necessary details for a new job opportunity.

This advertisements may range from small household jobs to complex program development jobs, when the user clicks on post button after filling all the details of the advertisements the post is updated in the real-time database located in the server. This advertisement or post can be viewed by any user who is registered with the application.

2. Doit User Module

In this module, we have user who can login with the credentials which have been provided by him/her while registering, thus after logging in into the application the user can now select "**Doit"** section where various job opportunities are available for selection.

After going through various Ad postings the user may select the job he/she is interested to do and contact the person who have posted the advertisement by means provided by the poster.

3. Server Module

In this module, we can understand the working and role of server in our project. We use firebase server for our uses and link our application with the project using dependencies. The server has our real time database where all our posts and other authentication data is stored.

It also uses in-built algorithms and google AI services to provide various app analytics, crash assistance, authentication and various other services to the user and the developer.

2.5 SYSTEM CONFIGURATION

The purpose of the specifications of the software requirements specifications and the intentions of the SRS target audience. Software Requirements Specifications: A description of the software product, program or set of programs that performs a number of functions in the target environment. Non-functional requirements are specified even if present. The remaining section of the SRS specifies the functionality of these systems. Moreover, the need also interfaces described in the following part of SRS. The criteria for non-functional requirements, system constraints and assumptions and dependencies (if any) are also described in the remaining sections.

Hardware Requirements

- System : Intel i3 2.4 GHz.
- Hard Disk : 240 GB
- Monitor : 15 VGA Colour.
- Mouse : Optical Mouse
- Ram : 2-4 GB

Software Requirements

- Operating system : Windows 7/8/8.1/10.
- Coding Language : JAVA/J2EE, XML.

- IDE : Google Android Studio.
- Server : Firebase
- Database : SQLite with JSON Framework.

3. LITERATURE OVERVIEW

The Literature Overview contains the information of the technology and its tools on which the project is being run.

3.1 ANDROID

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touch screen mobile devices such as smart phones and tablets. The android user interface is mainly based on direct manipulation, using touch gestures that corresponds to the actions of the real world, such as scrolling, touch and pinch to manipulate objects on the screen, along with the virtual keyboard for entering text. In addition to the touch screen devices Google developed Android TV for televisions, Android Auto, Android wear for watches, each with specialized user interface. Android versions are also used in laptops, video game consoles, digital cameras and other electronic devices.

The Source code of Android is released by Google under an open source license, although the majority of the Android devices finally provided with a combination of free and open-source software and proprietary information, including proprietary software required to access Google services. Android is popular among technology companies that require a ready-made operating system. Inexpensive and customizable for hi-tech devices. Its open nature has encouraged a large community of developers and enthusiasts using open source code as the basis for the projects conducted by the community which provides update to older devices and adds new features to advanced users or brings Android devices originally provided with other operating systems. Historically, the fragmentation of the Android platform has caused problems with security, where most of the Android devices do not receive security patches, but recent developments has improved the situation. The Android's

success has made it a target for patent and copyright litigation rights in the smart phone wars among technology companies.

There are various types of applications which can be developed using android operating system. They are:

1. Web Application.

2. Mobile-Web Application.

ibraries provide the java support required for the android development. 3. Mobile Application.

1. Web Application: Web application can be defined as a simple website which is developed by using Java or PHP. We open and use them in our web browsers.

2. Mobile-Web Application: This are also known as websites which are developed with a concept called Responsive Design, which automatically adjusts screen size depending upon the screen size which we are using.

3. Mobile Application: Mobile application is developed exclusively for mobile devices. There are two types of mobile applications. They are:

1. Native Application.

2. Hybrid Application.

3.1 Native Application: Native Application can be defined as an application which is developed solely for a single platform i.e. Android or IOS.

3.2 Hybrid Application: Hybrid application can be defined as a mobile application which can run on both the platforms and takes less time to develop.

We use various middleware for our project execution. They are:

1. **Core Libraries:** This l
2. **Dalvik Virtual Machine:** This is an optimized version of JVM which takes care of android application execution.
3. **Native Libraries:** It provides support of all other languages required in the project.

3.1.1 Features

Interface – The interface used in android is primarily based on direct manipulation, using tactile inputs that match the actions of the real world as browse, touch, pinch and reverse pinch to manipulate objects on the screen, along with a virtual keyboard. Game controllers and full-size physical keyboards are compatible with Bluetooth or USB Devices, thus response to a user is designed to be immediate and provides a touch interface fluid, often using the functionality of the device to vibrate to provide a tactical feedback to the user. Some applications use internal hardware such as accelerometers, gyroscopes and proximity sensors present in our devices to meet user actions.

Applications – The applications that extend the functionality of the devices are written using the Software Development Kit (SDK) for Android, and often, Java programming language is also used. Android has a growing selection of third-party applications that can be downloaded by the user and install the application through .APK file or downloaded through a program application store that allows users to install, update and remove applications from devices. Because of the open nature of Android, there are several applications marketed third-party play store application, These act as a substitute for those mobile devices which do not support Google Play store or when user need to download an application which is against the user-policy.

Memory Management – Since Android devices typically run on batteries, Android has been designed to handle multiple processes at a time and give optimal battery life. When application is not in use, the system suspends its operation in such a way that it does not use system resources. Android automatically manages the application data stored in the memory. When the memory is low, the system automatically starts closing the background inactive processes starting from those that have been inactive for more amount of time.

Open-Source: Android is not property of a single company, it is a product of OHA-Open Handset Alliance which is led by Google, most of the handset manufacturing companies which

use android operating system in their devices are part of these alliance and work with google in order to develop new and secured “Android Operating System” every year.

Android software development is the process by which new applications are created for the Android operating system. Applications are usually developed in the Java programming language using the Android Software Development Kit.

ADT (Android Development Tools) is the software used to develop android apps. It basically encases Eclipse IDE, which is a multi-language Integrated development environment (IDE) comprising a base workspace and an extensible plug-in system for customizing the environment.. The latest version comes with ADT plugin preinstalled and bundled to the IDE.

Application programming interface (API) specifies how some software components should interact with each other. In practice in most of the cases an API is a library that usually includes specification for routines, data structures, object classes, and variables. An API specification can take many forms, including an International Standard such as POSIX, vendor documentation such as the Microsoft Windows API, the libraries of a programming language, e.g., Standard Template Library in C++ or Java API. Google APIs can be downloaded from Google Code, Google’s site for developer tools, APIs and technical resources.

The Google Data API allow programmers to create applications that read and write data from Google services. Currently, these include APIs for Google Apps, Google Analytics, Blogger, Google Base, Google Book Search, Google Calendar, Google Code Search, Google Earth, Google Spreadsheets, Google Notebook, and Picasa Web Albums. SDK (Software Development KIt or "devkit") is typically a set of software development tools that allows for the creation of applications for a certain software package,software framework, hardware platform, computer system, video game console, operating system, or similar development platform. It may be something as simple as an application programming interface (API) in the form of some files to interface to a particular programming language or include sophisticated hardware to communicate with a certain embedded system. Common tools include debugging aids and other utilities often presented in an integrated development environment (IDE). In the latest version of ADT, the android SDK adds on to the IDE automatically as soon as you unzip and load the IDE. SDK Manager enables us to download Google APIs and use them in our code.

3.1.2 API level:

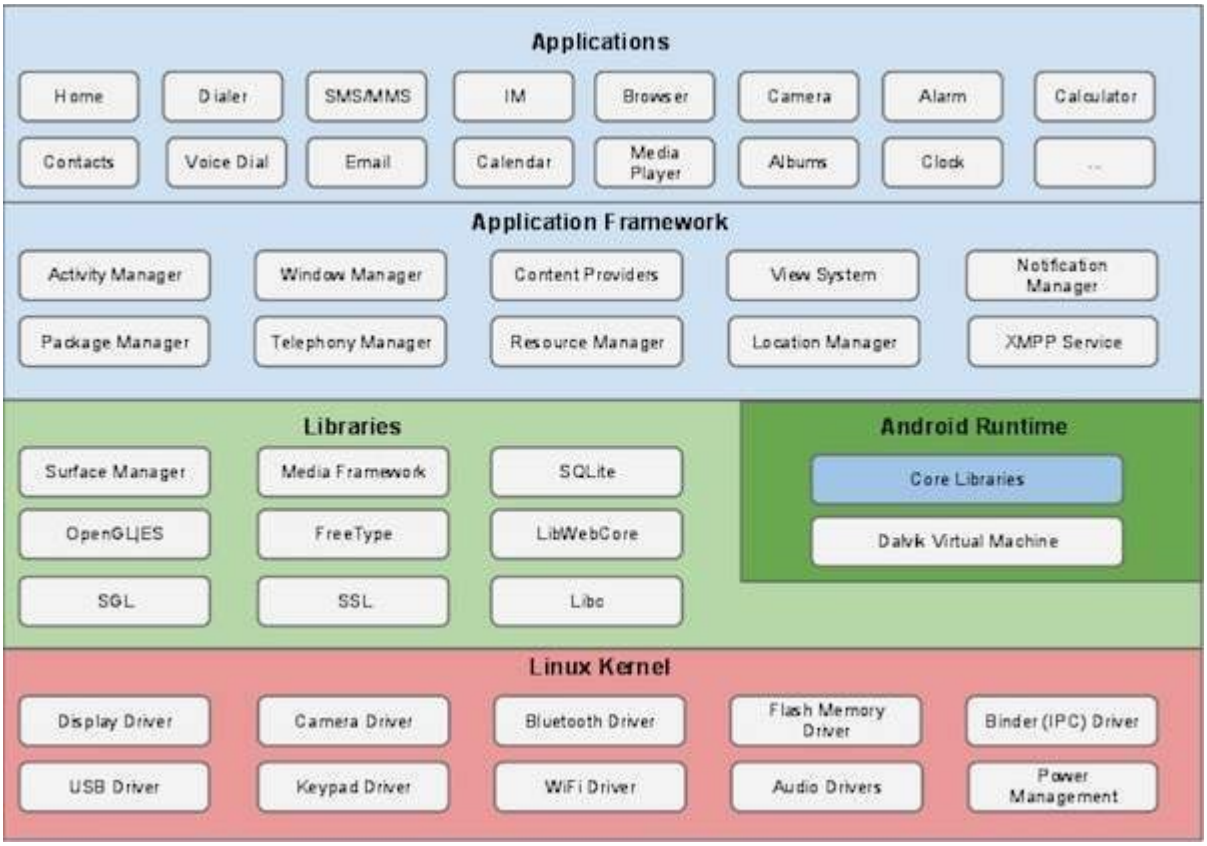
API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

Platform Version	API Level	VERSION_CODE	
Android 8.0	26	Oreo	
Android 7.0	24	Nougat	
Android 6.0	23	MARSHMALLOW	
Android 5.1	22	LOLLIPOP_MR1	
Android 5.0	21	LOLLIPOP	
Android 4.4W	20	KITKAT_WATCH	KitKat for Wearables Only
Android 4.4	19	KITKAT	
Android 4.3	18	JELLY_BEAN_MR2	
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	

Android 4.1, 4.1.1	16	JELLY_BEAN	
Android 4.0.3	15	ICE_CREAM_SANDWICH_MR1	
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH	
Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	
Android 3.0.x	11	HONEYCOMB	
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1	
Android 2.3.2 Android 2.3	9	GINGERBREAD	
Android 2.2.x	8	FROYO	
Android 2.1.x	7	ECLAIR_MR1	
Android 2.0.1	6	ECLAIR_0_1	
Android 2.0	5	ECLAIR	
Android 1.6	4	DONUT	

Android 1.5	3	CUPCAKE	
Android 1.1	2	BASE_1_1	
Android 1.0	1	BASE	

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.



Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers 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.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, 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.

3.1.3 Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

- **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content access, publishing and messaging between applications and application components.
- **android.database** − Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** − Used to render and manipulate text on a device display.
- **android.view** − The fundamental building blocks of application user interfaces.

- **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.

- **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

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 **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik 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.

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.

The Android framework includes the following key services −

- **Activity Manager** − Controls all aspects of the application lifecycle and activity stack.

- **Content Providers** − Allows applications to publish and share data with other applications.

- **Resource Manager** − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.

- **Notifications Manager** − Allows applications to display alerts and notifications to the user.

- **View System** − An extensible set of views used to create application user interfaces.

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.

3.1.4 UI Layouts:

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The ViewGroup is a subclass of View and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using View/ViewGroup objects or you can declare your layout using simple XML file main_layout.xml which is located in the res/layout folder of your project.

Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	**Linear Layout** LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	**Relative Layout** RelativeLayout is a view group that displays child views in relative positions.
3	**Table Layout** TableLayout is a view that groups views into rows and columns.
4	**Absolute Layout** AbsoluteLayout enables you to specify the exact location of its children.
5	**Frame Layout** The FrameLayout is a placeholder on screen that you can use to display a single view.
6	**List View** ListView is a view group that displays a list of scrollable items.
7	**Grid View**

	GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the view.
2	**android:layout_width** This is the width of the layout.
3	**android:layout_height** This is the height of the layout
4	**android:layout_marginTop** This is the extra space on the top side of the layout.

5	**android:layout_marginBottom** This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft** This is the extra space on the left side of the layout.
7	**android:layout_marginRight** This is the extra space on the right side of the layout.
8	**android:layout_gravity** This specifies how child Views are positioned.
9	**android:layout_weight** This specifies how much of the extra space in the layout should be allocated to the View.
10	**android:layout_x** This specifies the x-coordinate of the layout.
11	**android:layout_y** This specifies the y-coordinate of the layout.

12	**android:layout_width** This is the width of the layout.
13	**android:layout_width** This is the width of the layout.
14	**android:paddingLeft** This is the left padding filled for the layout.
15	**android:paddingRight** This is the right padding filled for the layout.
16	**android:paddingTop** This is the top padding filled for the layout.
17	**android:paddingBottom** This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

A **View** is an object that draws something on the screen that the user can interact with and a ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

- You define your layout in an XML file which offers a human-readable structure for the layout, similar to HTML.

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

Sr.No.	UI Control & Description
1	**TextView** This control is used to display text to the user.
2	**EditText** EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	**AutoCompleteTextView** The AutoCompleteTextView is a view that is similar to EditText, except that it

	shows a list of completion suggestions automatically while the user is typing.
4	**Button** A push-button that can be pressed, or clicked, by the user to perform an action.
5	**Image_Button** An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	**CheckBox** An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	**ToggleButton** An on/off button with a light indicator.
8	**RadioButton** The RadioButton has two states: either checked or unchecked.
9	**RadioGroup**

	A RadioGroup is used to group together one or more RadioButtons.
10	**ProgressBar** The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	**Spinner** A drop-down list that allows users to select one value from a set.
12	**TimePicker** The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	**DatePicker** The DatePicker view enables users to select a date of the day.

3.1.5 Event Handling:

Events are a useful way to collect data about a user's interaction with interactive components of Applications**.** Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are following three concepts related to Android Event Management −

- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.

onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.

onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu( )	**onCreateContextMenuItemListener()** This is called when the context menu is being built(as the result of a sustained "long click)

SQLite Database:

SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below

SQLSiteDatabase mydatabase=openOrCreateDatabase(“yourdatabase name”,MODE_PRIVATE,null);

Database – Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet=mydatabase.rawQuery("select * from Tablename",null);

resultSet.moveToFirst();

String username=resultSet.gwtString(0);

String password=resultSet.getString(1);
```

SDK Manager:

To download and install latest android APIs and development tools from the internet, android provide us with android SDK manager. Android SDK Manager separates the APIs, tools and different platforms into different packages which you can download.

Android SDK manager comes with the Android SDK bundle. You can't download it separately. You can download the android sdk from here.

Running Android SDK Manager

Once downloaded, you can launch Android SDK Manager in one of the following ways −

- Click tools->Android-> SDK Manager option in Eclipse.
- Double Click on the SDK Manager.exe file in the Android SDK folder.

Android Manifest :

AndroidManifest.xml file is necessary for all android applications and must have this name in its root directory. In the manifest you can find essential informations about the application for the Android system, informations that the system must have before it can run any of the application's code. Here is what you can find in the Android manifest: -The name of the Java package for the application. The package name serves as a unique identifier for the application. -The description of the components of the application : the activities, services, broadcast receivers, and content providers that the application is composed of and under what conditions they can be launched . -The processes that will host application components. -The permissions the application must have in order to access protected parts of the API and interact with other applications. -The permissions that others are required to have in order to interact with the application's 16 components. -The list of the Instrumentation classes that provide profiling and other information as the application is running. These declarations are present in the manifest only while the application is being developed and tested; they're removed before the application is published. -The minimum level of the Android API that the application requires. -The list of the libraries that the application must be linked against.

3.1.6 FIREBASE:

Firebase is a mobile and web application development platform developed by Firebase, Inc.

Firebase evolved from Envolve, a prior startup founded by James Tamplin and Andrew Lee in 2011. Envolve provided developers an API that enables the integration of online chat

functionality into their websites. After releasing the chat service, Tamplin and Lee found that it was being used to pass application data that weren't chat messages. Developers were using Envolve to sync application data such as game state in real time across their users. Tamplin and Lee decided to separate the chat system and the real-time architecture that powered it. They founded Firebase as a separate company in April 2012.

Firebase Inc. raised seed funding in May 2012. The company further raised Series A funding in June 2013. In October 2014, Firebase was acquired by Google. In October 2015, Google acquired Divshot to merge it with the Firebase team. Since the acquisition, Firebase has grown inside Google and expanded their services to become a unified platform for mobile developers. Firebase now integrates with various other Google services to offer broader products and scale for developers. In January 2017, Google acquired Fabric and Crashlytics from Twitter to join those services to the Firebase team.

As of now, Firebase offers variety of services ranging from Cloud messaging, Crash Analytics, Real-Time Database, Performance Monitoring, Remote Configuration, Dynamic Links, App Indexing, Adwords.

Firebase Real-Time Database:

Firebase Real-time database is a cloud hosted database that supports multiple platforms Android, iOS and Web. All the data is stored in JSON format and any changes in data, reflects immediately by performing a sync across all the platforms & devices. This allows us to build more flexible realtime apps easily with minimal effort.

How is data stored?

Database is stored in the database in JSON format. Firebase realtime database is a schemaless database in which the data is stored in JSON format. Basically the entire database is a big JSON tree with multiple nodes. So when you plan your database, you

need to prepare the json structure in way that the data is accessible in easier way by avoiding nesting of child nodes.

Here is an example of storing list of user profiles and posts in json tree

```
{
  "users": [
    {
      "name": "Raj Bharath",
      "email": "raj@gmail.com",
      "address": "XXX, XXXX, 1234"
    }
  ],
  "posts": [
    {
      "id": 110,
      "author": "Raj Bharath",
      "content": "This is awesome firebase database...",
      "timestamp": "13892733894"
    }
  ]
}
```

Offline Data:

Firebase provides great support when comes to offline data. It automatically stores the data offline when there is no internet connection. When the device connects to internet, all the data will be pushed to real time database. However enabling **disk persistence** stores the data offline even though app restarts. Disk persistence can be enabled by calling below one line code.

The code can be given as follows:

FirebaseDatabase.getInstance( ).setPersistenceEnabled(true);

In order to perform any operation on to database whether it can be read or write, you need to get the reference to database first. The below code gives you reference to database JSON top node. From here you need to use the child node names to traverse further.

private DatabaseReference mDatabase;

mDatabase = FirebaseDatabase.getInstance().getReference();

Inserting Data:

To insert data, you can use **setValue()** method on to database reference path. This will create or update the value on path provided. The code to do that is:

DatabaseReference mRef = mDatabase.getReference("Your node name ");

The database accepts multiple datatypes **String**, **Long**, **Double**, **Boolean**, **Map<String, Object>**, **List<Object>** to store the data. You can also use **custom java objects** to store the data which is very helpful when storing model class directly in database.

Reading Data:

To read the data, you need to attach the **ValueEventListener()** to the database reference. This event will be triggered whenever there is a change in data in realtime. In **onDataChange()** you can perform the desired operations onto new data.

Updating Data:

To update data, you can use the same **setValue()** method by passing new value. You can also use **updateChildren()** by passing the path to update data without disturbing other child nodes data.

Deleting Data

To delete data, you can simply call **removeValue()** method on to database reference. You can also pass **null** to **setValue()** method which do the same delete operation.

Security & Rules:

Firebase rules provides a way to identify user role while performing read and write operations. These rules will acts a security layer on the server before perform any CRUD operation. By default the rules allows user to perform read & write operation only after authentication.

The below rules allow authenticated users only to read or write data.

```
{
 "rules": {
   ".read": "auth != null",
   ".write": "auth != null"
 }
}
```

Below rules allows everyone to read & write data without authentication.

```
{
 "rules": {
   ".read": true,
   ".write": true
 }
}
```

You can also use these rules to validate data before inserting into database.

Creating Android Project:

1. First thing you need to do is go to https://firebase.google.com/ and make an account to gain access to their console. After you gain access to the console you can start by creating your first project.
2. Give the package name of your project in which you are going to integrate the Firebase. Here the **google-services.json** file will be downloaded when you press add app button.
3. Create a new project in Android Studio from **File ⇒ New Project**. While filling the project details, use the same package name which you gave in firebase console.
4. Paste the **google-services.json** file to your project's **app** folder. This step is very important as your project won't build without this file.
5. Now open the **build.gradle** located in project's home directory and add google playstore dependency.

   Buildscript {

Dependencies {

classpath ‘com.google.gms:google-services:3.2.0’//google services plugin

classpath ‘com.android.tools.build.gradle:2.2.0’

}

6. Open **app/build.gradle** and add **firebase database** dependency. At the very bottom of the file, add **apply plugin: ‘com.google.gms.google-services’**
7. Now write the code as per your requirement in the class you have created for the purpose.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

Architecture diagram is a diagram of a system, in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. The block diagram is typically used for a higher level, less detailed description aimed at understanding the overall concepts and less at the understanding the details of implementation.

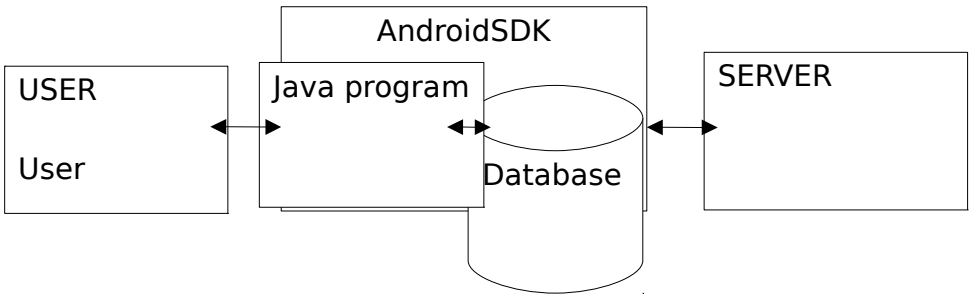


Fig : 4.1

A user is authenticated using the firebase server by using the details provided by him during the time of registration. After logging in into the application we use java programs as backend and declare the intents and methods we require for working of our application and then

user can either post a new advertisement or view from existing one. Our Firebase server contains our database which we use to store and retrieve posts in our application.

4.2 UML DIAGRAMS

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

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

A UML system is represented using five different views that describe the system from a different point of view. Each view is defined by a set of diagrams, They are:

User Model View

- This view represents the system from the user point of view.
- The representation of the analysis describes a scenario from end user's point of view.

Structural Model View

- In this model the data and functions are obtained from within a system.
- This view model of static structures models.

Behavioral Model View

- It represents the dynamic behavior as parts of the system, representing interactions between various collection structural elements described in the user model and the structural model view.

Implementation Model View

- In this models are constructed in the structural and behavioral parts of the system.

4.2.1 Class Diagram

Class Diagram in the UML is a type of static structure diagram that describes the structure of a system that shows the system classes, attributes and relationships between classes. Private visibility hidden information of any element that is outside the class partition, Public visibility allows all other classes see the marked information. A class has three sections. The upper part contains the name of the class. This central part contains the attributes of the class. The fund provides methods or operations that can take or initiate class.

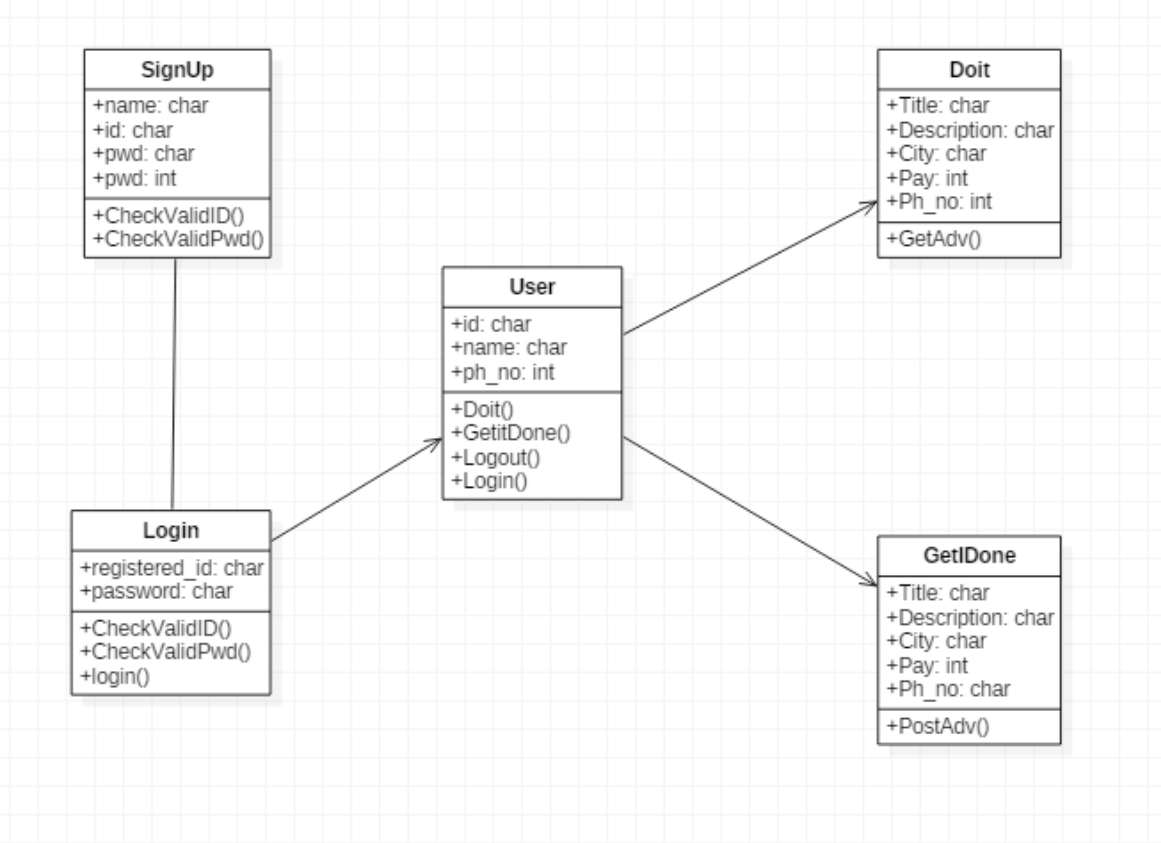

SignUp
+name: char
+id: char
+pwd: char
+pwd: int
+CheckValidID()
+CheckValidPwd()
Doit
+Title: char
+Description: char
+City: char
+Pay: int
+Ph_no: int
+GetAdv()
User
+id: char
+name: char
+ph_no: int
+Doit()
+GetitDone()
+Logout()
+Login()
Login
+registered_id: char
+password: char
+CheckValidID()
+CheckValidPwd()
+login()
GetIDone
+Title: char
+Description: char
+City: char
+Pay: int
+Ph_no: char
+PostAdv()

Fig: 4.2.1

4.2.2 Use case Diagram

A diagram of the use cases is a type of behavior pattern created by an analysis of use cases. The purpose of the use case is to present an overview of the functionality provided by the system in terms of actors, targets and dependencies between use cases.

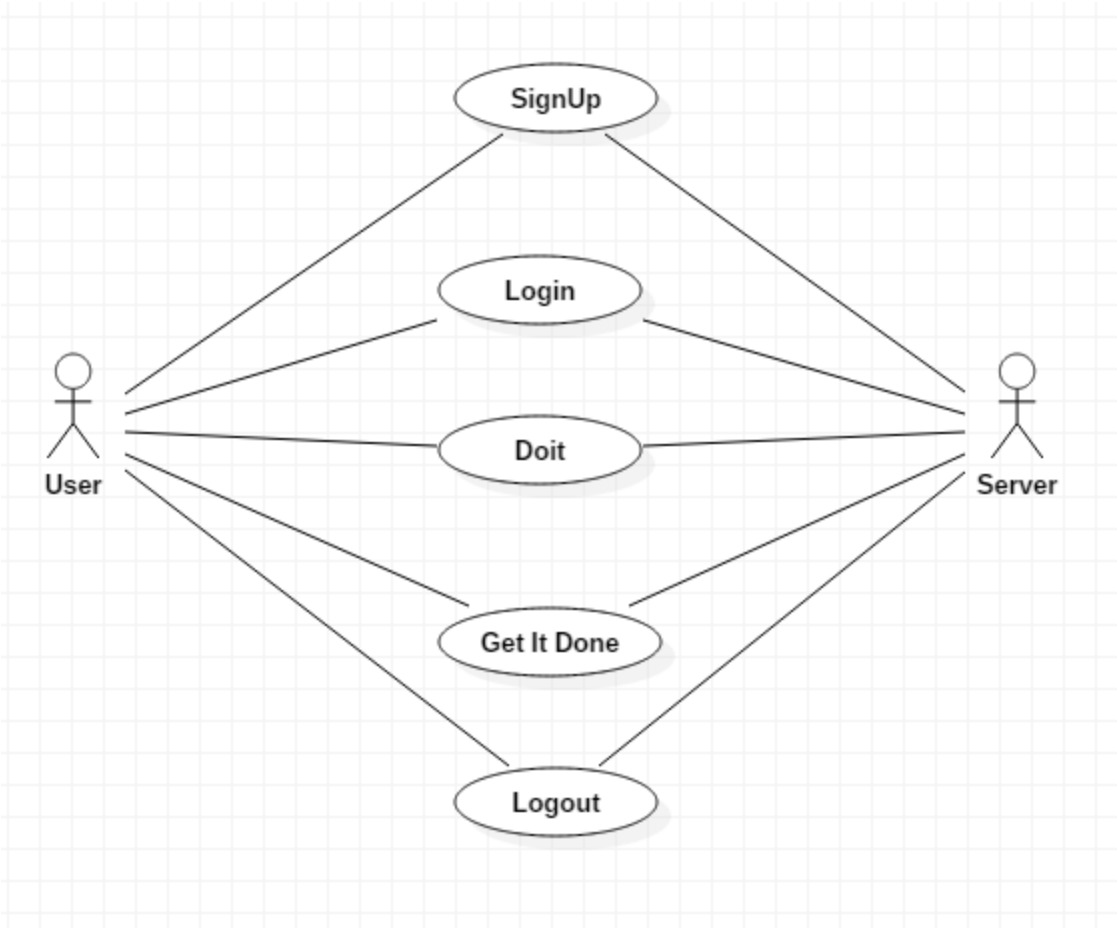


Fig: 4.2.2

4.2.3 Sequence Diagram

A UML sequence diagram is a type of interaction diagram that shows how processes work together in what order. It is a construction diagram of a message sequence. Sequence diagrams sometimes called tracking schemes events, scenarios of events and timing diagrams.

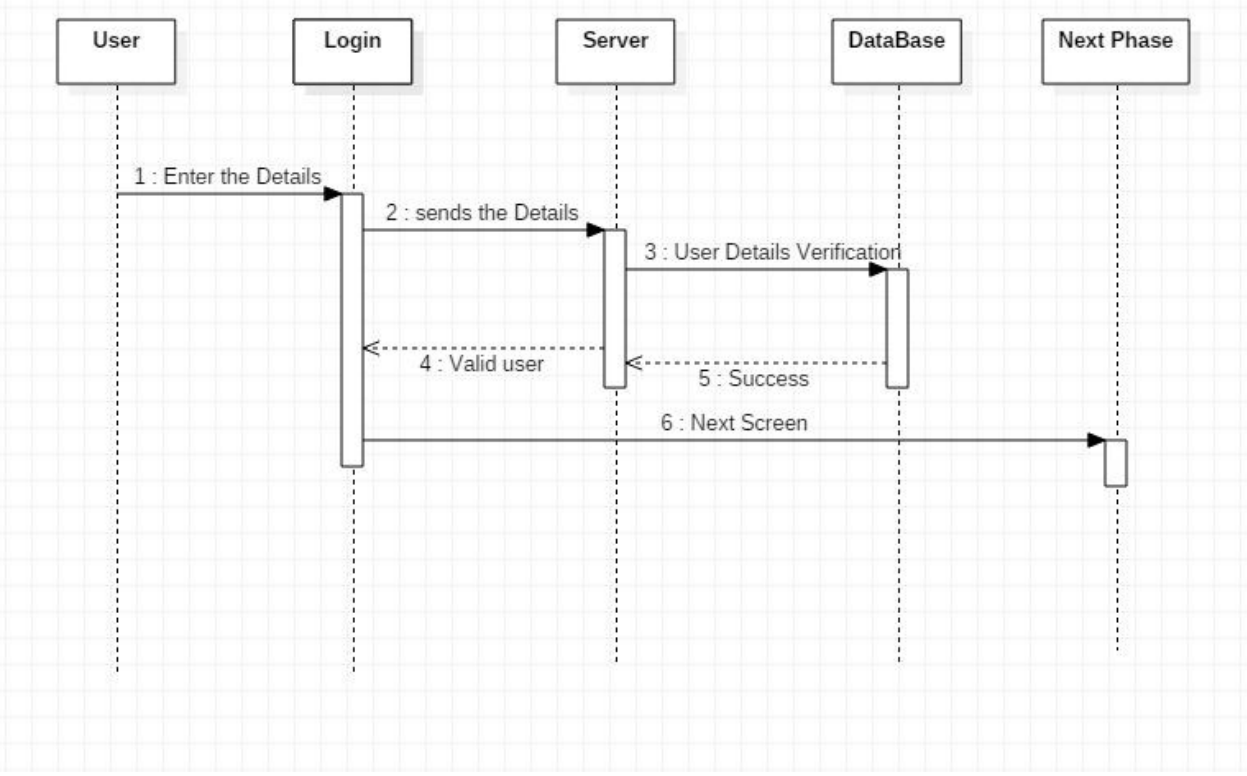


Fig: 4.2.3

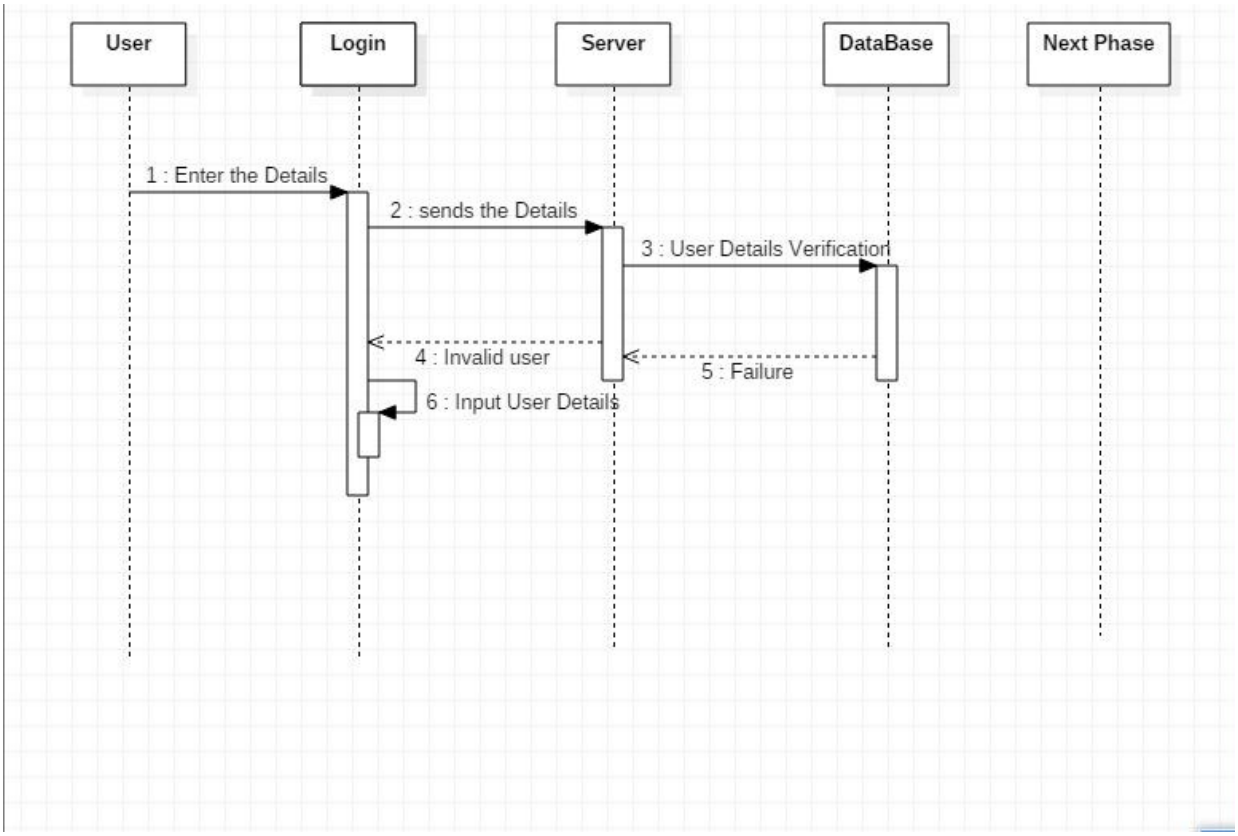

User
Login
Server
DataBase
Next Phase
1 : Enter the Details
2 : sends the Details
3 : User Details Verification
4 : Invalid user
5 : Failure
6 : Input User Details

Fig:4.2.3.1

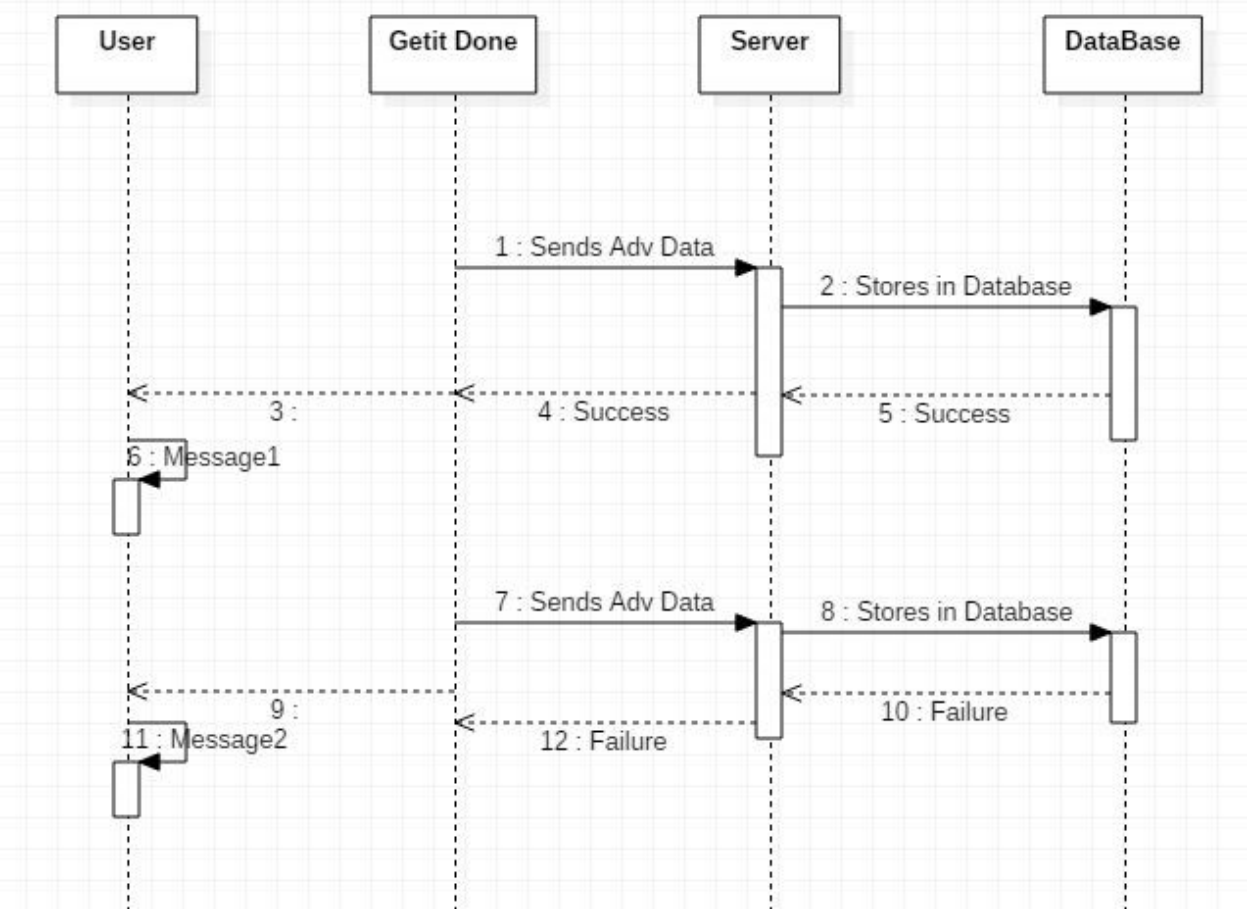

User
Getit Done
Server
DataBase
1 : Sends Adv Data
2 : Stores in Database
3 :
4 : Success
5 : Success
6 : Message1
7 : Sends Adv Data
8 : Stores in Database
9 :
10 : Failure
11 : Message2
12 : Failure

Fig:4.2.3.2

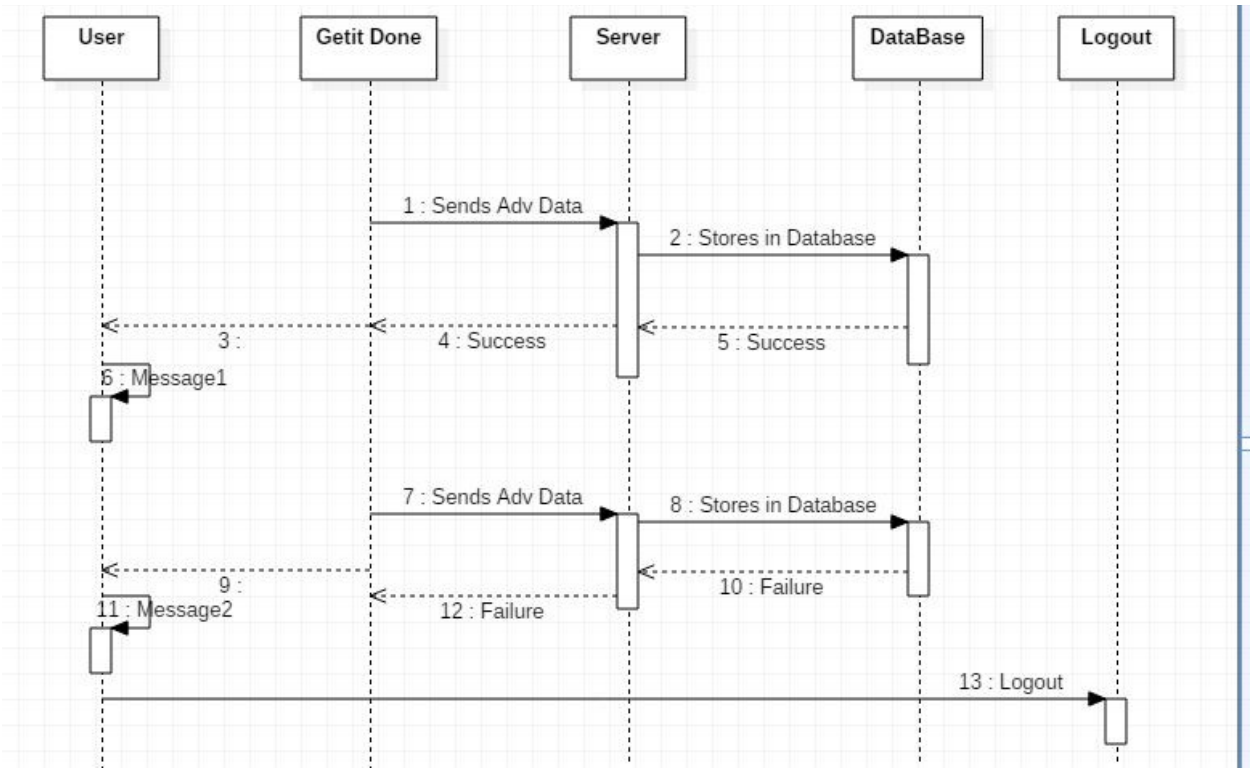

User
Getit Done
Server
DataBase
Logout
1 : Sends Adv Data
2 : Stores in Database
3 :
4 : Success
5 : Success
6 : Message1
7 : Sends Adv Data
8 : Stores in Database
9 :
10 : Failure
11 : Message2
12 : Failure
13 : Logout

Fig:4.2.3.3

4.2.4 Deployment Diagram

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

Deployment diagrams are used to describe the static deployment view of a system. A deployment diagram consists of nodes and their relationships.

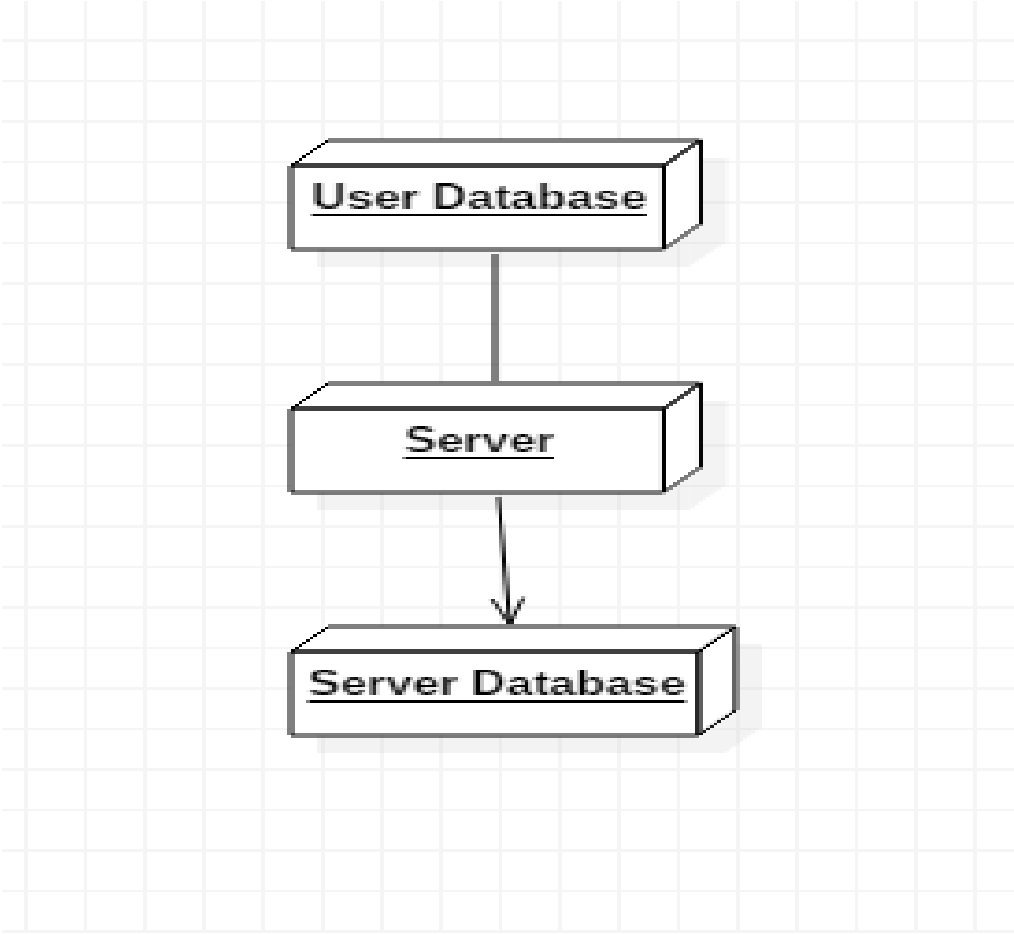
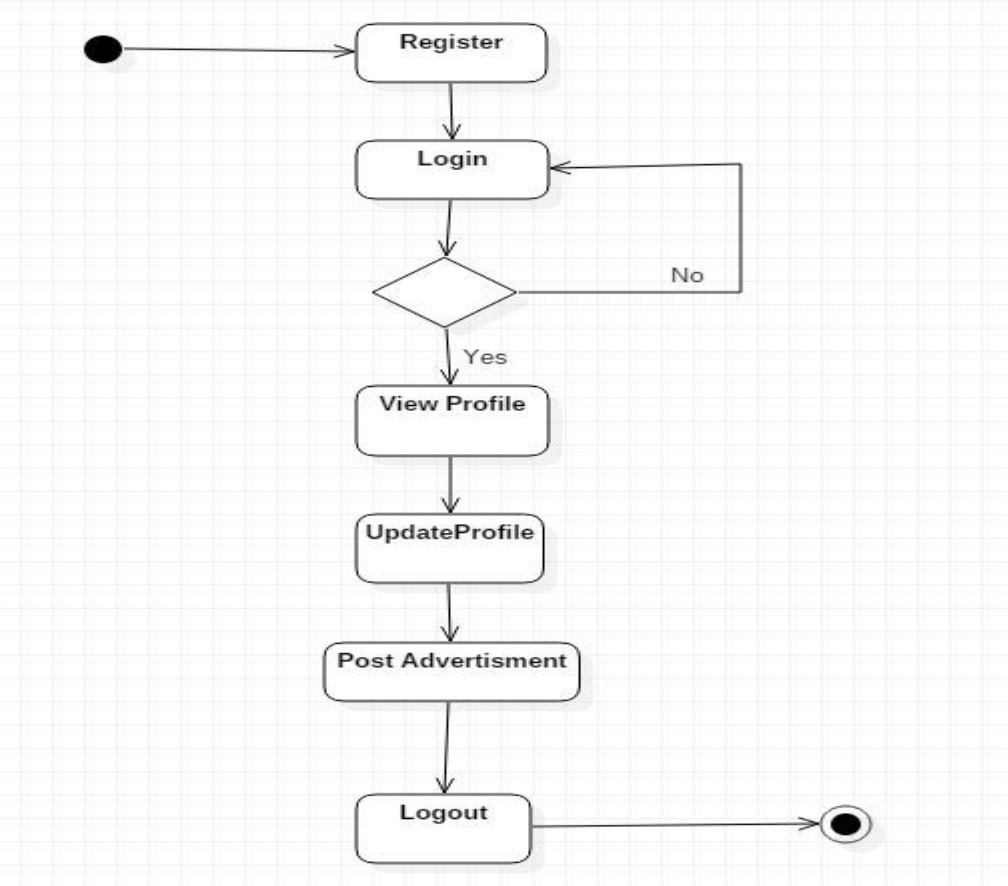


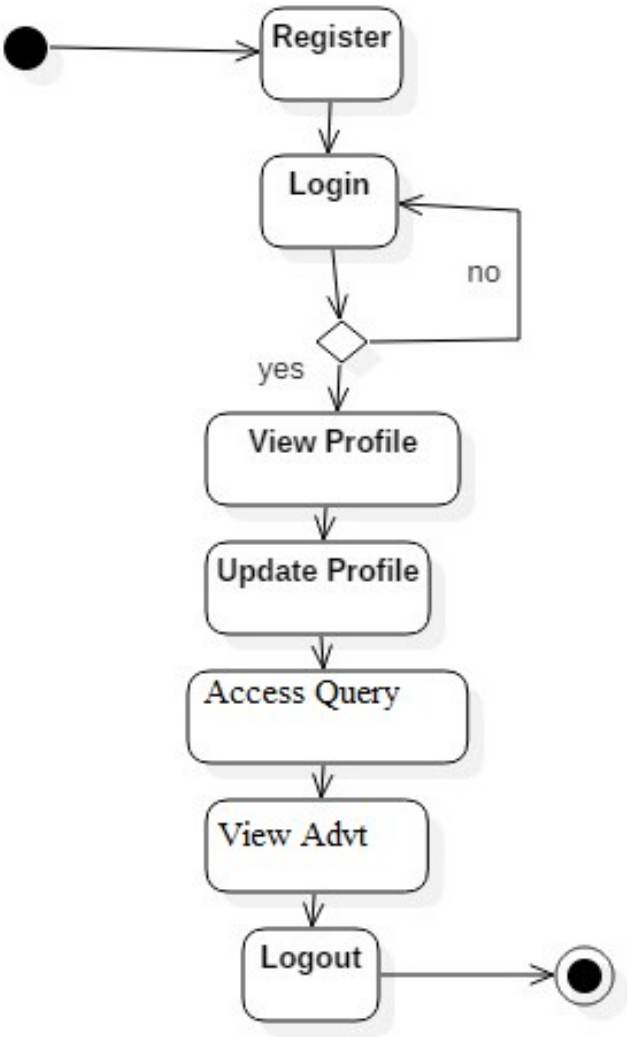
Fig: 4.2.4

4.2.5 Activity Diagram

The activity diagram is a clear defined diagram showing flows of assets and offset actions, with the support for choice, iteration and form competition. In UML, the activity diagrams can be used to describe workflow of the business and operational components of a step system. The UML activity diagrams can model the internal logic of a complex task. In many ways UML activity diagrams are flow charts, object-oriented and diagram data from the equivalent structural development flow. The following diagram shows the workflow activity system.


Register
Login
No
Yes
View Profile
UpdateProfile
Post Advertisment
Logout

Get It Done


Register
Login
no
yes
View Profile
Update Profile
Access Query
View Advt
Logout

DOIT

Fig: 4.2.5

5. SAMPLE CODE

5.1 CODING

```
package com.project.doit;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.ArrayList;


public class Timeline extends AppCompatActivity
{
   ListView listView;
   EditText search;
   FirebaseDatabase database;
   DatabaseReference posts;
   ArrayList<String> list;
   ArrayAdapter<String> adapter;
   Adv adv ;
   @Override
   protected void onCreate( Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.timeline);
      adv = new Adv( );
      listView = (ListView) findViewById(R.id.ads);
      search=(EditText)findViewById(R.id.search);
      database = FirebaseDatabase.getInstance();
      posts = database.getReference("Aposts");
     list = new ArrayList<>();
      adapter = new ArrayAdapter<String>(this,R.layout.adv_info,R.id.advinfo, list);
      posts.addValueEventListener(new ValueEventListener()
      {
         @Override
         public void onDataChange(DataSnapshot dataSnapshot)
```

```
        {
            for (DataSnapshot ds : dataSnapshot.getChildren())
            {
                adv = ds.getValue(Adv.class);
                list.add(adv.getTitle().toString() + " \n " + adv.getDescription().toString()
+"\n"+adv.getAmt().toString()+"\n"+adv.getCityname().toString()
+"\n"+adv.getPhno().toString());
            }
            listView.setAdapter(adapter);
            search.addTextChangedListener(new TextWatcher() {

                @Override
                public void onTextChanged(CharSequence cs, int arg1, int arg2, int arg3) {
                    // When user changed the Text
                    Timeline.this.adapter.getFilter().filter(cs);
                }

                @Override
                public void beforeTextChanged(CharSequence arg0, int arg1, int arg2,
                                              int arg3) {
                    // TODO Auto-generated method stub

                }

                @Override
                public void afterTextChanged(Editable arg0) {
                    // TODO Auto-generated method stub
                }
            });

        }

        @Override
        public void onCancelled(DatabaseError databaseError)
        {


        }
    });
  }
}
```

6. TESTING

6.1 TESTING

Testing is a process of trying to find every conceivable fault or weakness in a work in product. It provides a way to verify the functionality of the components, subassemblies, assemblies, and/or a finished product. It is a process of exercising software intended to ensure that the software system meets the requirements and user expectations and unacceptably unsafe. There are different types of tests, each type of tests response to a need specific test.

The developed software has been successfully tested using the following test strategies and errors are corrected and the back part of the program or procedure or function is tested until all errors are eliminated. A successful test is one that finds an unknown error.

Note that the system test results show that the system works properly. You trust the system designer, system users, avoid frustration during the application process, etc.

There of basically two types of testing approaches. One is Black-Box testing – the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operated. The other is White-Box testing – knowing the internal workings of the product, tests can be conducted to ensure that the internal operation of the product performs according to specifications and all internal components have been adequately exercised. White box and Black box testing methods have been used to test this package. All the loop constructs have been tested for their boundary and intermediate conditions. The test data was designed with a view to check for all the conditions and logical decisions. Error handling has been taken care of by the use of exception handlers. Testing is a set of activities that can be planned in advanced and conducted systematically. A strategy for software testing must accommodation low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. Software testing is one element of verification and validation.

Verification refers to the set of activities that ensure that software correctly implements as specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements. The main objective of software testing to uncover errors. To fulfill this objective, a series of test steps unit, integration, validation and system tests are planned and executed. Each test step is accomplished through a series of systematic test technique that assist in the design of test cases. With each testing step, the level of abstraction with which software is considered is broadened.

Unit Testing

Unit testing focuses verification effort on the smallest unit of software design – the module. The unit test is always white box oriented. The tests that occur as part of unit testing are testing the module interface, examining the local data structures, testing the boundary conditions, executing all the independent paths and testing error-handling paths.

Integration Testing:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. Scope of testing summarizes the specific functional, performance, and internal design characteristics that are to be tested. It employs top-down testing and bottom-up testing methods for this case.

White Box Testing:

The purpose of any security testing method is to ensure the robustness of a system in the face of malicious attacks or regular software failures. White box testing is performed based on the knowledge of how the system is implemented. White box testing includes analyzing data flow, control flow, information flow, coding practices, and exception and error handling within the system, to test the intended and unintended software behavior. White box testing can be performed to validate whether code implementation follows intended design, to validate implemented security functionality, and to uncover exploitable vulnerabilities. White box testing requires access to the source code. Though white box testing can be performed any time in the life cycle after the code is developed, it is a good practice to perform white box testing during the unit testing phase. White box testing requires knowing what makes software secure or insecure,

how to think like an attacker, and how to use different testing tools and techniques. The first step in white box testing is to comprehend and analyze source code, so knowing what makes software secure is a fundamental requirement. Second, to create tests that exploit software, a tester must think like an attacker. Third, to perform testing effectively, testers need to know the different tools and techniques available for white box testing. The three requirements do not work in isolation, but together.

Black Box Testing:

Also known as functional testing. A software testing technique whereby the internal workings of the item being tested are not known by the tester. For example, in a black box test on software design the tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs. The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications. The advantages of this type of testing include: The test is unbiased because the designer and the tester are independent of each other. The tester does not need knowledge of any specific programming languages. The test is done from the point of view of the user, not the designer. Test cases can be designed as soon as the specifications are complete.

System Testing:

System testing validates software once it has been incorporated into a larger system. Software is incorporated with other system elements and a series of system integration and validation tests are conducted. System testing is actually a series of different test whose primary purpose is to fully exercise the computer- based system. Once the system has been developed it has to be tested. In the present system we have to take care of valid property and assessment numbers i.e. there should not exist any duplicate number in each case. Care should be taken that the appropriate data is retrieved in response to the queries. 6.2.6 Alpha Testing Alpha testing is one of the most common software testing strategy used in software development. Its specially used by product development organizations. This test takes place at the developer's site. Developers observe the users and note problems. Alpha testing is testing of an application when development is about to complete. Minor design changes can still be made as a result of alpha testing.

• Alpha testing is typically performed by a group that is independent of the design team, but still within the company, e.g. in-house software test engineers, or software QA engineers. Alpha testing is final testing before the software is released to the general public. It has two phases: In the first phase of alpha testing, the software is tested by in-house developers. They use either debugger software, or hardware-assisted debuggers. The goal is to catch bugs quickly. In the second phase of alpha testing, the software is handed over to the software QA staff, for additional testing in an environment that is similar to the intended use. Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

Beta Testing:

Beta Testing is also known as field testing. It takes place at customer's site. It sends the system/software to users who install it and use it under real-world working conditions. A beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. (Beta is the second letter of the Greek alphabet.) Originally, the term alpha testing meant the first phase of testing in a software development process. The first phase includes unit testing, component testing, and system testing. Beta testing can be considered "pre-release testing. The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user's perspective that you would not want to have in your final, released version of the application. Example: Microsoft and many other organizations release beta versions of their products to be tested by users.

Gamma Testing:

Gamma Testing may be considered as the third level of testing, which is often performed over a software product. It is used to evaluate the working of the completely ready software product with all the specified requirements for its market release. The main intent to carry out gamma testing is to be on the safe side with respect to the software product quality just before its market release. Moreover, it does not provide any scope to bring changes in the software product unless the identified risk or bug is of high severity that may adversely affect the quality.

6.2 TEST CASES

Test Cases	Check Item	Test Case Objective	Expected Result	Obtained Result	Result
TC-001	Login	Check link from login page to home page	Forwarded to home page	Forwarded to home page	PASS
TC-002	Register	Check link from index page to register page	Forwarded to register page	Forwarded to register page	PASS
TC-003	Update Profile	Updating the details of the user without taking blank fields	User profile is updated	User Profile is updated	PASS
TC-004	Send Solution	Sending solution for the absolute Client ID	Solution is sent to correct Client	Solution is sent to correct Client	PASS
TC-005	Login with empty fields	Authorization with empty fields(login and password)	Please enter all fields	Error message “please enter all fields”	PASS

Table : 6.1

7. OUTPUT SCREENS

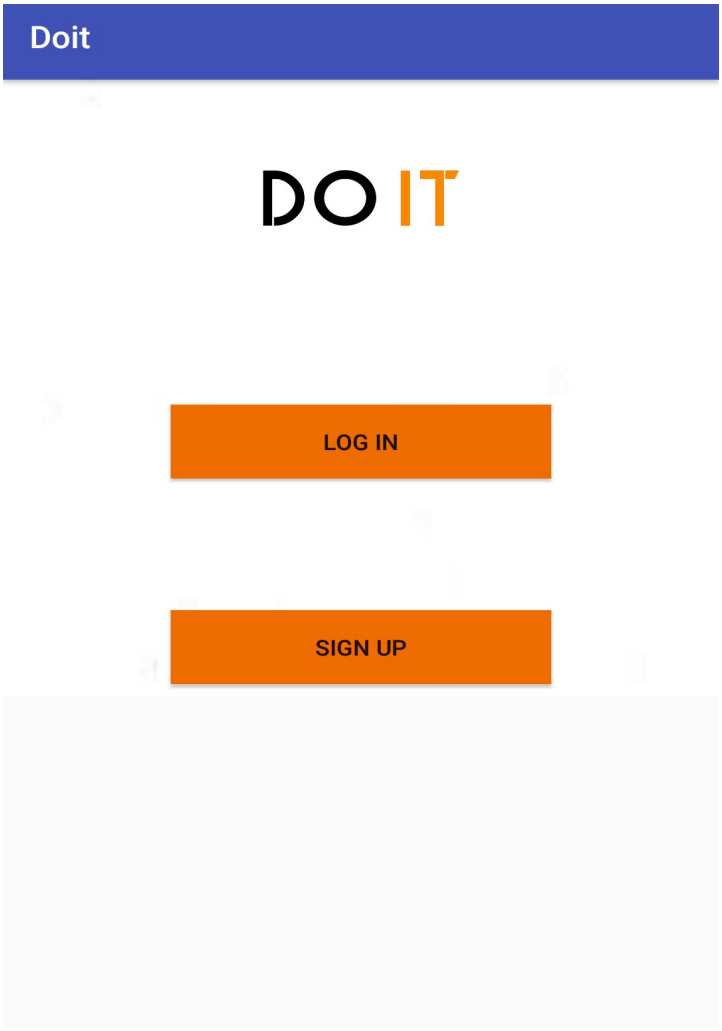


Fig: 7.1 Main Screen

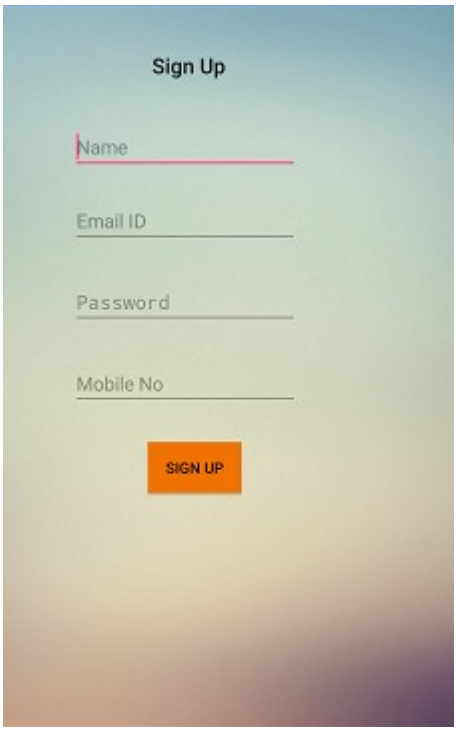


Fig: 7.2 Sign Up Screen

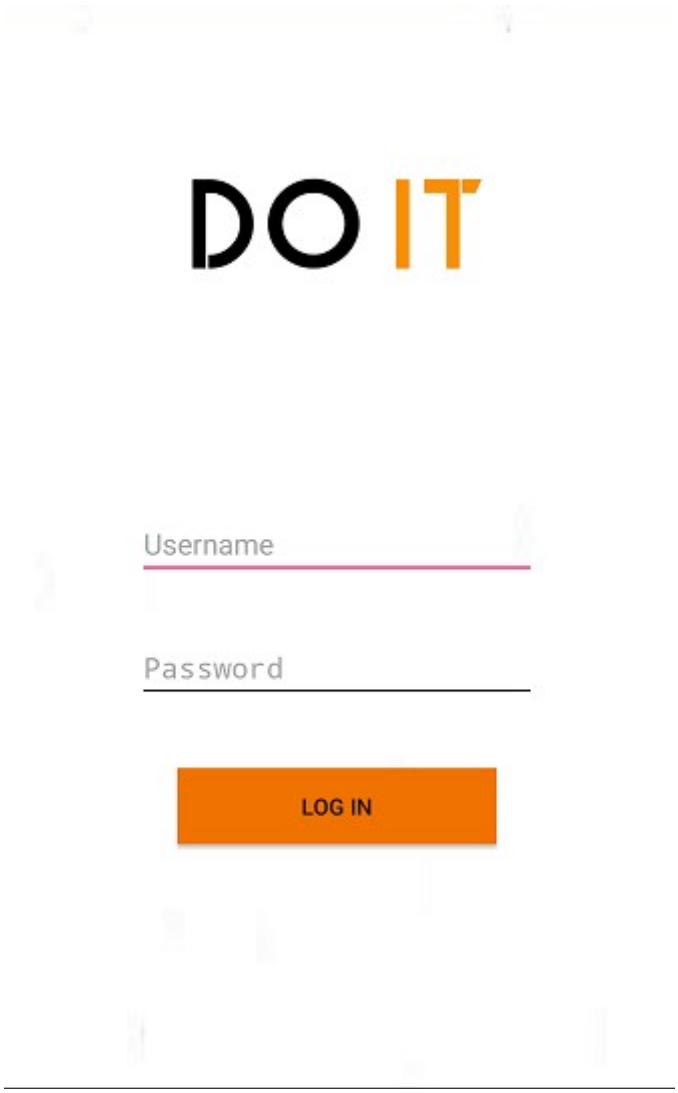


Fig: 7.3 Login Screen

Fig: 7.4 Home Screen

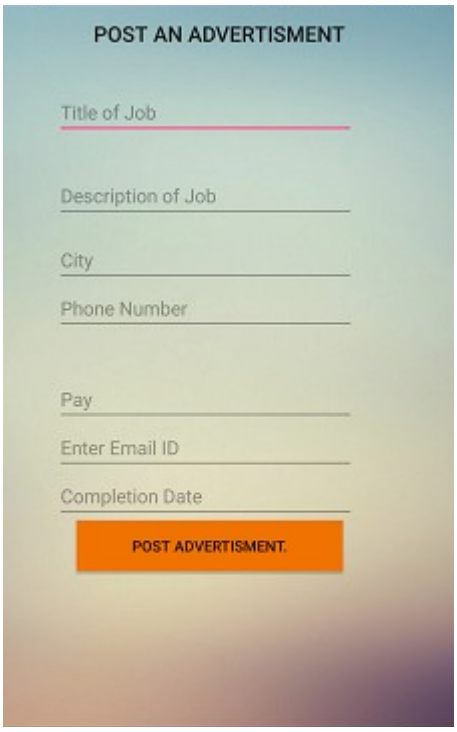


Fig: 7.5 Get It Done Screen

Fig: 7.6 Timeline

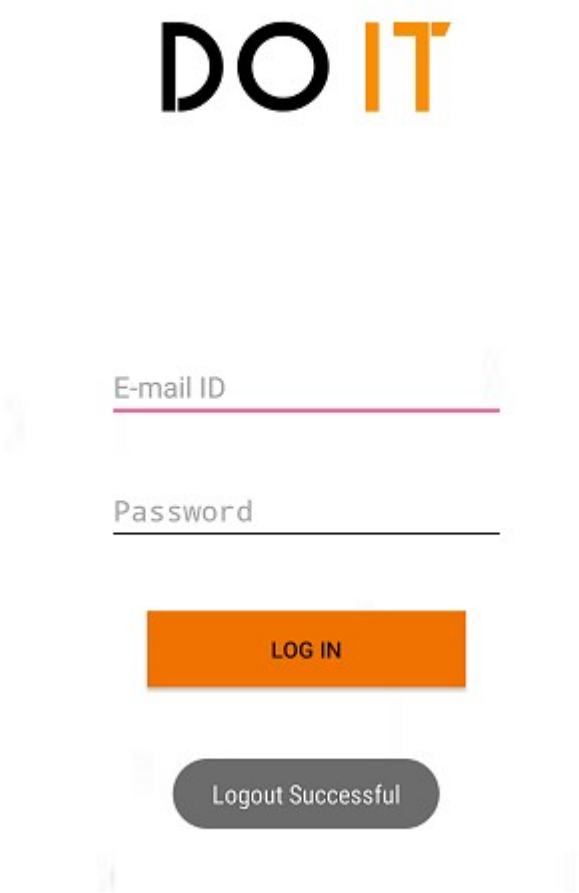


Fig: 7.7 Logout Activity

8. CONCLUSION

8.1 CONCLUSION

At the end of the application the client gets easy and fast response for the problem that has occurred with proper recommendations. The application is user-friendly. It gives better performance to the admin so that he can easily add management and view clients and management. The client can post any query of his problem at any time and can get the solution for that. Time is more consumed with the application rather than searching for the problem in the manual book.

8.2 FURTHER ENHANCEMENTS

- We can add feature like capturing an image or a video and post it. So that it can be easily understood about the problem and solution.
- We can add recording an audio clip and post to the management.
- The client can directly communicate with the management by the call option if further done.
- We can add new features like saving the car details of the client to his ID and intimate him of his car servicing date when it expires.
- We can add GPS tracker so that the client can send his location to the management if the problem is severe.

9. BIBLIOGRAPHY

9.1 BOOKS REFERENCES

- Herbert Schildt.2008 ,"Java Complete Reference", Tata McGraw-Hill ,

  7th Edition, pp. 177-180.

- Grady Booch, James Rambaugh.1998, "Unified Modeling Language User Guide", Addison Wesley Publishing, chapter 8-31.

- Raghu Ramakrishnan, Gehrke. 2003, "Database Management Systems", Tata McGraw-Hill ,3rd Edition.

- Bill Phillips, Pearson. 2013,"Android Programming: The Big Nerd Ranch"

- Bill Stonehem, 2017."Google Android Firebase: Learning The Basics".

9.2 WEBSITES REFERENCES

http://developer.android.com/index.html

http://en.wikipedia.org/wiki/SQLite

https://www.tutorialspoint.com/android/index.html

http://github.com

https://firebase.google.com/docs/