Location Based Services Using Android

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Gaurav Rana(121313 C.S.E)

Under the supervision of

Ms. Nishtha Ahuja

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CERTIFICATE

Candidate's Declaration

I hereby declare that the work presented in this report entitled **" Location Based Services Using Android"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,**

Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2015 to June 2016 under the supervision of **Ms. Nishtha Ahuja Grade 1 Professor Department of CSE**

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Gaurav Rana (121313)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Signature)

Supervisor Name: Nishtha Ahuja

Designation:Assistant Professor(Grade 1)

Department name: Computer Science & Engineering

Dated:

Acknowledgement

Words betray to express our heartfelt sentiments towards the two towering peaks of our lives, our parents who have helped us to surpass all the academic pursuits and our life.

I sincerely express my thanks and heartfelt gratitude to Prof. Nishtha Ahuja to permit me to undergo this project.

I extend my cordial thanks to all those who directly or indirectly helped in the successful fulfillment of this project work.

Date:28th Mays 2016
Place: JUIT, Waknaghat

Gaurav Rana

Table of Contents

ABBRIVIATIONS

1.	AAC	Advanced Audio Coding
2.	ADB	Android Debug Bridge
3.	AMR	Adaptive-Multi Rate
4.	.apk	Android Application Package File
5	API	Application Program Interface
6	Apps	Applications
7	BSD	Berkley Software Distribution
8	CDMA	Code division multiple access
9	EDGE	Enhanced Data Rates for GSM Evolution
10	FOAH	Findings and Order After Hearing
11	gcj	GreenHeck Caps Jobs
12	GIF	Graphics Interchange Format
13	GPS	Global Positioning System
14	GPRS	General Packet Radio Service
15	GSM	Global System for Mobile
16	HTML	Hyper Text Markup Language
17	HTTP	Hypertext Transfer Protocol
18	H.264	High quality video format
19	IT	Information Technology
20	IPC	Inter Process Communication
21	I/O	Input Output
22	IDE	Integrated development environment
23	JDT	Java Development Tool
24	JPG	Joint Photographic Group

25	MPEG4	Motion Picture Experts Group Layer-4
26	MP3	MPEG Audio Layer 3
27	OS	Operating System
28	PNG	Portable Network Graphics
29	QoS	Quality of Service
30	SQLite	standardized query lite
31	SMS	Short Message Service
32	SDK	Software Development Kit
33	UI	User Iterface
34	VM	Virtual Manager
35	URL	Uniform Resource Locator
36	WiFi	Wireless Fidelity
37	3G	3rd Generation
38	.dex	Dalvik Executable

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The motivation for every location based information system is: “To assist with the exact information, at right place in real time with personalized setup and location sensitiveness”. In this era we are dealing with palmtops and iPhones, which are going to replace the bulky desktops even for computational purposes. We have vast number of applications and usage where a person sitting in a roadside café needs to get relevant data and information. Such needs can only be catered with the help of LBS. These applications include security related jobs, general survey regarding traffic patterns, decision based on vehicular information for validity of registration and license numbers etc. A very appealing application includes surveillance where instant information is needed to decide if the people being monitored are any real threat or an erroneous target. We have been able to create a number of different applications where we provide the user with information regarding a place he or she wants to visit. But these applications are limited to desktops only. We need to import them on mobile devices. We must ensure that a person when visiting places need not carry the travel guides with him. All the information must be available in his mobile device and also in user customized format

This app uses LBS very well for tourists or any person new to any place. It list down almost all places categories available on google developer site and user can select any place of his/her choice and view images of that place before visiting it. Moreover the reviews of that place by earlier visiters are also available which helps in easy decision before visiting that palce and easy search option with autocomplete property.

Chapter -1 INTRODUCTION

1.1 Introduction

Android is a mobile operating system (OS) currently developed by Google, based on the Linux Kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct maniulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game console, digital cameras, and other electronics.

Android Components

- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source **Web Kit** engine
- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and Wi-Fi** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)

Google Maps: Whether searching for the perfect restaurant, checking out the best hotels or finding the nearest bank, millions of people around the world get Google Maps to do the hard work for them. So why not do the same for your own website? The Google Maps API is one of those clever bits of Google technology that helps you take the power of Google Maps and put it directly on your own site. It lets you add relevant content that is useful to your visitors and customise sthe look and feel of the map to fit with the style of your site. With over 150,000 sites already using the Google Maps API, we couldn’t fit them all into this booklet so we picked out a few of the most useful and innovative examples to help inspire you. And if after that you’re still hungry for more, check out the back of this booklet for links to more examples and technical information.

Google maps is a great way of viewing the area around the property that you are interested in, saving you hours of time and frustration being shown properties that do not match your search criteria. Google maps, together with viewing the full video of the property, serves as a powerful tool when short listing properties that you wish to physically view. With Google maps you will be able to move up and down the street as if you were walking along it. You can see satellite views of the property and surrounding area, view a map, get directions and GPS coordinates to the property and even search nearby facilities such as schools, churches and shopping centre. As you become more familiar with Google maps you will discover that there are different ways to perform certain tasks. You will also learn about many other features that Google maps has to offer, that are not all covered in this document.

Google Maps is a desktop web mapping service developed by Google. It offers satellite imagery, street maps, 360° panoramic views of streets (street View) real-time traffic conditions (Google traffic) , and route planning for traveling by foot, car, bicycle , or transportation.

Google Maps began as a C++ desktop program designed by Lars and Jens Rasmussen at Where 2 Technologies. In October 2004, the company was acquired by Google, which converted it into a web application. After additional acquisitions of a geospatial data visualization company and a realtime traffic analyzer, Google Maps was launched in February 2005. The service's front end utilizes Javascript,XML, and Ajax. Google Maps offers an API that allows maps to be embedded on third-party websites,[1]and offers a locator for urban businesses and other organizations in numerous countries around the world. Google Maps Makerallows users to collaboratively expand and update the service's mapping worldwide.

Google Maps' satellite view is a "top-down" view; most of the high-resolution imagery of cities is aerial photography taken from aircraft flying at 800 to 1,500 feet (240 to 460 m), while most other imagery is from satellites.[2] Much of the available satellite imagery is no more than three years old and is updated on a regular basis. Google Maps uses a close variant of the Mercator projection, and therefore cannot accurately show areas around the poles.

The current redesigned version of the desktop application was made available in 2013, alongside the "classic" (pre-2013) version. Google Maps for mobile was released in September 2008 and features GPS turn by turn navigation. In August 2013, it was determined to be the world's most popular app for smartphones , with over 54% of global smartphone owners using it at least once.

1.2 Problem Statement

Currently there are android apps in the market which will help you to book hotel rooms and apps that track your locations but no single app with both these properties. So we are developing this app with both these features(finding tourist spots and hotels near your location and tracking you moreover its reviews and gallery to view images).

1.3 Objectives

1. To help tourists to search and find tourist spots near their location .
2. To reduce their search time and problem of visiting one websites/app to find tourist spots and other website/app for booking.
3. Providing best prices in hotels(linkages,discounts,coupons).
4. Light app which doesn't consumes much net.
5. Track the user location and help him to reach the desired location.

Map is the ideal way to implement our app. It offers our users to discover new places all over the world.

1.4 Methodology

Today's software products are complex and constantly evolving throughout the entire product cycle. That makes it harder than ever to hold everyone on the same page.

Agile methods reduce overhead, but increase risk by removing process steps which are key to managing larger projects. The answer is to not abandon time-tested processes and documents. The answer is to make the creation and maintenance of software engineering documents dramatically faster and more effective.

The Agile method is often used by startups for new Android mobile solutions where the mobile developers need rapid feedback. It's also utilized by IT organizations whose internal customers can't agree on what they want.

At Android Developers Ltd we use agile development extensively across Android mobile development projects of different scale and needs.

Agile software development caters to fast turns and evolving requirements.

Software engineering is the practice of using selected process techniques to enhance the quality of a software development effort. This is based on the assumption, subject to endless debate and supported by patient experience, that a methodical approach to software development leads to fewer defects and, therefore, ultimately provides shorter delivery times and better value.

Which agile development methods should you consider? They have to not only respond flexibly to changing requirements, they must also provide a disciplined framework with predictable schedules.

Two methods which have proven to work when requirements are not well-understood are the Rational Unified Process and Agile Software Development.

Benefits with Agile Software Development:

1. Minimized project risk

2. Maximized project visibility

3.Enhanced predictability, and adaptability

4.Generation of relatively high quality software

5.Availability of options to track and review the project regularly

6.Cost Control

Most agile programming methodologies lay emphasis on building releasable software in brief time periods just like other iterative development models. Agile development differs from other development models in two key aspects. These primary differences are the short and strict time periods for every iteration, each of which do not exceed week, and a highly collaborative manner of working with the client.

Some of the key steps involved in an agile development process include:

1. Initiation of the Project: Initiation involves setting up the plan, specifying the business needs, freezing the specifications, building the team, and preparing initial architecture modeling.

2. Development Iterations: This repetitive stage involves active client participation, rigorous and collaborative development by professionals, adding new features upon requirement, confirmatory and investigative testing, and internal deployment of software.

3. Release of the product: This stage involves final system testing, final user acceptance testing, and deployment of system into production and regular and updated delivery of software which meets the changing needs of the client. Release phase allows the end user to review the software and send the feedback as well as request for more features if required.

4. Production: This phase involves the regular maintenance of the system and identification of defects to enhance the system performance as well as stability.

1.5 Organization

Chapter 1: Highlights and Underlines of the Location based services. In this chapter, the introduction Location based services is covered. The key focus defining the problem statement and specifying the objectives of the project

Chapter 2: The detailed literature review from the research paper, books, journals and conferences are done. In this chapter, the extracts from assorted research papers on HCI, Location Tracking, Tourism.

Chapter 3: Covers the system development which is the key aspect of this work. In this chapter, the proposed model, algorithm, UML diagrams and related parameters are emphasized.

Chapter 4: : The simulation of implementation results with the relative performance analysis is shown in this chapter. The simulation results and screenshots are revealed to depict and defend the proposed work.

Chapter 5: Detailed conclusion and scope of the future work which guides the upcoming students and research scholars to enhance the current work with higher efficiency and effectiveness on Location racking and toursim.

Chapter -2 LITERATURE SURVEY

2.1 Literature Survey

Earlier, handheld GPS receivers such as a Garmin Etrex, were in use for a number of years and have been used to enhance student learning through geocaching and mobile mapping. With improvement, last decade has observed a new ability to geotag photographs by using two separate devices: a GPS receiver and a camera. These devices are linked by the time and date settings on each of the devices and then an external application was used to synchronize them.

Now smartphones have both digital cameras and assisted GPS in-built to the device. The relative ease and accessibility of geotagging has "generated a wave of geo-awareness" .US News & World Report lists geotagging photos as one of "50 ways to improve your life in 2009". Friedland & Sommer state that "all the major smartphone makers are now offering models allowing instantaneous upload of geotagged photos, videos, and even text messages to sites such Flickr, YouTube, and Twitter."

In a recent international survey of geography and bioscience higher education practitioners conducted by the authors, geotagging was repeatedly cited as one area of technology that practitioners expect to see expanding over the next five years, a sentiment shared by Luo et al. who suggest that, "with the availability of internet, GPS devices and smartphones, the proliferation and availability of geotagged media will continue to expand". Similarly, Johnson et al. (2010) identify the use of mobile phones in education as one of the key areas in which they expect significant growth in next 12 months.

In context with the accuracy, there have been suggestions in improving the Quality of Service (QoS) using crowdsourcing . Also comparative study with respect to accuracy of location data has been done between Android and iOS .

2.2 Android Architecture

Android architecture has mainly 4 following layers:

1. Applications Layer
2. Application Framework
3. Libraries along with android runtime libraries
4. Linux Kernel.

The following diagram shows the architecture in proper stack. Each of the layer is explained below.

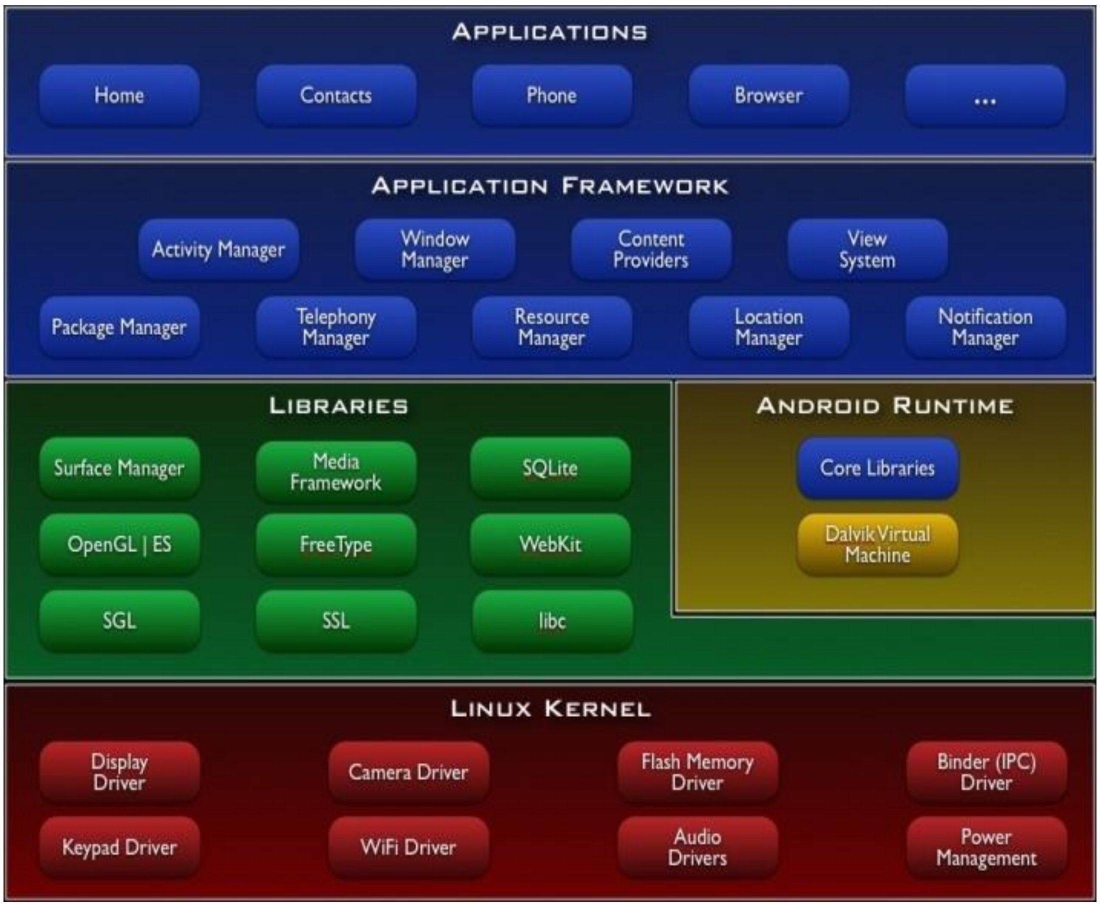


fig 2.1 Android Architecture

Applications Layer

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. These applications comprise the application layer of android.

Application Framework

Application framework provides developers full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

Underlying all applications is a set of services and systems, including:

- A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser.
- Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data.
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files.
- A Notification Manager that enables all applications to display custom alerts in the status bar.
- An Activity Manager that manages the lifecycle of applications and provides a common navigation backstack.

Libraries
Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

- **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
- **Media Libraries** - based on PacketVideo'sOpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- **Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
- **SGL** - the underlying 2D graphics engine
- **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- **FreeType** - bitmap and vector font rendering

- **SQLite** - a powerful and lightweight relational database engine available to all applications

Android Runtime Libraries

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Linux Kernel

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

2.3 Main Components of Android Application

There are 5 components around which an android applications revolves. They are

1. Activities
2. Broadcast Receivers
3. Services
4. Intents
5. Content Providers

Activities

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

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

When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods. There are several callback methods that an activity might receive, due to a change in its state—whether the system is creating it, stopping it, resuming it, or destroying it—and each callback provides you the opportunity to perform specific work that's appropriate to that state change. For instance, when stopped, your activity should release any large objects, such as network or database connections. When the activity resumes, you can reacquire the necessary resources and resume actions that were interrupted. These state transitions are all part of the activity lifecycle.

Broadcast Receivers

Broadcast Receiver is actually a mechanism to send and receive events so that all interested applications can be informed when something happens. There are heaps of System events which get broadcast by Android OS such as SMS related events, Connectivity related events, and camera related events and many more. We are able to broadcast our application specific events as well, so for example if we have a RSS news reader application and we want to do something whenever a new item is available, it would be a good idea to use Broadcast Receiver method, not only because it will separate your event handling code but most importantly because it will enable other applications to register and receive a notification whenever that event takes place.

There are two major classes of broadcasts that can be received:

- **Normal broadcasts** are completely asynchronous. All receivers of the broadcast are run in an undefined order, often at the same time. This is more efficient, but means that receivers cannot use the result or abort APIs included here.
- **Ordered broadcasts** are delivered to one receiver at a time. As each receiver executes in turn, it can propagate a result to the next receiver, or it can completely abort the broadcast so that it won't be passed to other receivers. The order receivers run in can be controlled with the priority attribute of the matching intent-filter; receivers with the same priority will be run in an arbitrary order.

Even in the case of normal broadcasts, the system may in some situations revert to delivering the broadcast one receiver at a time. In particular, for receivers that may require the creation of a process, only one will be run at a time to avoid overloading the system with new processes. In this situation, however, the non-ordered semantics hold: these receivers still cannot return results or abort their broadcast.

Services

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

A service can essentially take two forms:

- Started:- A service is "started" when an application component (such as an activity) starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.

Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

- Bound:- A service is "bound" when an application component binds to it by calling bindService(). 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 inter-process communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Content Providers

Content providers store and retrieve data and make it accessible to all applications. They're the only way to share data across applications; there's no common storage area that all Android packages can access.

Android comes with a number of content providers for common data types (audio, video, images, personal contact information, and so on). You can see some of them listed in the android.provider package. You can query these providers for the data they contain (although, for some, you must acquire the proper permission to read the data).

If you want to make your own data public, you have two options: You can create your own content provider (aContentProvider subclass) or you can add the data to an existing provider — if there's one that controls the same type of data and you have permission to write to it.

Intents

Three of the core components of an application — activities, services, and broadcast receivers — are activated through messages, called intents. Intent messaging is a facility for late run-time binding between components in the same or different applications. The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed — or, often in the case of broadcasts, a description of something that has happened and is being announced. There are separate mechanisms for delivering intents to each type of component:

- An Intent object is passed toContext.startActivity() or Activity.startActivityForResult()to launch an activity or get an existing activity to do something new. (It can also be passed to Activity.setResult() to return information to the activity that called startActivityForResult().)
- An Intent object is passed to Context.startService() to initiate a service or deliver new instructions to an ongoing service. Similarly, an intent can be passed to context.bindService() to establish a connection between the calling component and a target service. It can optionally initiate the service if it's not already running.
- Intent objects passed to any of the broadcast methods are delivered to all interested broadcast receivers. Many kinds of broadcasts originate in system code.

In each case, the Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary. There is no overlap within these messaging systems: Broadcast intents are delivered only to broadcast receivers, never to activities or services. An intent passed to startActivity() is delivered only to an activity, never to a service or broadcast.

Activity Stack

• Activities in the system y are managed as an activity stack.

• When a new activity is started, it is placed on the top of the stack and becomes the running activity -- the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

• If the user presses the Back Button the next activity on the stack moves up and becomes active.

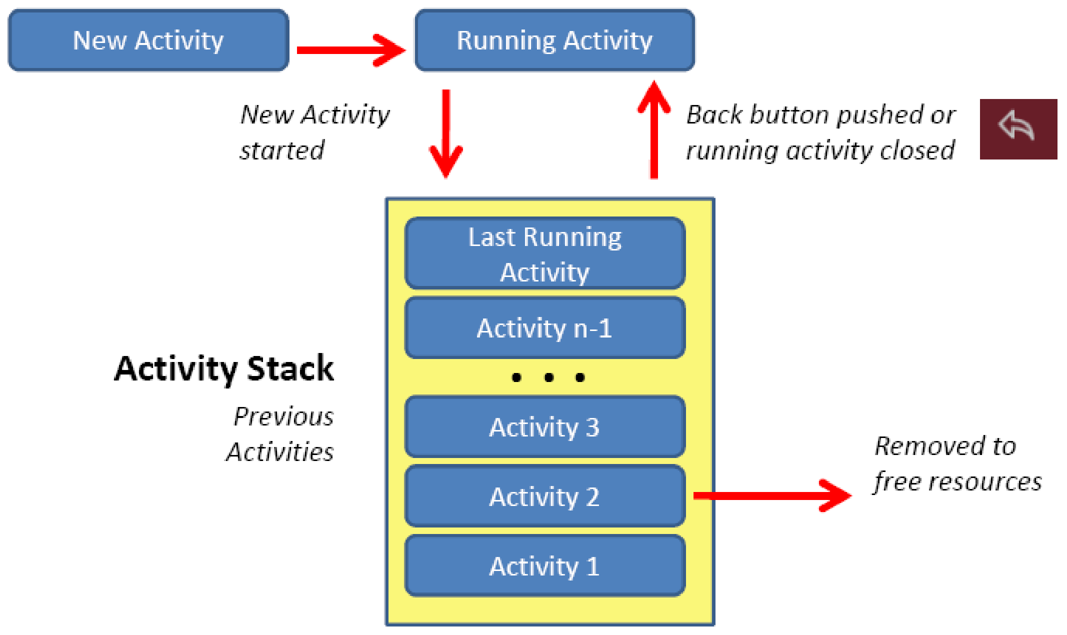


fig 2.2 Activity Stack

2.1.6 Processes and Threads

When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution. By default, all components of the same application run in the same process and thread (called the "main" thread). If an application component starts and there already exists a process for that application (because another component from the application exists), then the component is started within that process and uses the same thread of execution. However, you can arrange for different components in your application to run in separate processes, and you can create additional threads for any process.

When deciding which processes to kill, the Android system weighs their relative importance to the user. For example, it more readily shuts down a process hosting activities that are no longer visible on screen, compared to a process hosting visible activities. The decision whether to terminate a process, therefore, depends on the state of the components running in that process.

When an application is launched, the system creates a thread of execution for the application, called "main." This thread is very important because it is in charge of dispatching events to the appropriate user interface widgets, including drawing events. It is also the thread in which your application interacts with components from the Android UI toolkit. As such, the main thread is also sometimes called the UI thread.

The system does not create a separate thread for each instance of a component. All components that run in the same process are instantiated in the UI thread, and system calls to each component are dispatched from that thread. Consequently, methods that respond to system callbacks (such as onKeyDown() to report user actions or a lifecycle callback method) always run in the UI thread of the process.

For instance, when the user touches a button on the screen, your app's UI thread dispatches the touch event to the widget, which in turn sets its pressed state and posts an invalidate request to the event queue. The UI thread dequeues the request and notifies the widget that it should redraw itself.

When your app performs intensive work in response to user interaction, this single thread model can yield poor performance unless you implement your application properly. Specifically, if everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI. When the thread is blocked, no events can be dispatched, including drawing events. From the user's perspective, the application appears to hang. Even worse, if the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "application not responding" (ANR) dialog. The user might then decide to quit your application and uninstall it if they are unhappy.

Additionally, the Android UI toolkit is not thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread. Thus, there are simply two rules to Android's single thread model:

1. Do not block the UI thread

2. Do not access the Android UI toolkit from outside the UI thread

Processes and Threads

When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution. By default, all components of the same application run in the same process and thread (called the "main" thread). If an application component starts and there already exists a process for that application (because another component from the application exists), then the component is started within that process and uses the same thread of execution. However, you can arrange for different components in your application to run in separate processes, and you can create additional threads for any process.

When deciding which processes to kill, the Android system weighs their relative importance to the user. For example, it more readily shuts down a process hosting activities that are no longer visible on screen, compared to a process hosting visible activities. The decision whether to terminate a process, therefore, depends on the state of the components running in that process.

When an application is launched, the system creates a thread of execution for the application, called "main." This thread is very important because it is in charge of dispatching events to the appropriate user interface widgets, including drawing events. It is also the thread in which your application interacts with components from the Android UI toolkit. As such, the main thread is also sometimes called the UI thread.

The system does not create a separate thread for each instance of a component. All components that run in the same process are instantiated in the UI thread, and system calls to each component are dispatched from that thread. Consequently, methods that respond to system callbacks (such as onKeyDown() to report user actions or a lifecycle callback method) always run in the UI thread of the process.

For instance, when the user touches a button on the screen, your app's UI thread dispatches the touch event to the widget, which in turn sets its pressed state and posts an invalidate request to the event queue. The UI thread dequeues the request and notifies the widget that it should redraw itself.

When your app performs intensive work in response to user interaction, this single thread model can yield poor performance unless you implement your application properly.

Specifically, if everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI. When the thread is blocked, no events can be dispatched, including drawing events. From the user's perspective, the application appears to hang. Even worse, if the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "application not responding" (ANR) dialog. The user might then decide to quit your application and uninstall it if they are unhappy.

Additionally, the Android UI toolkit is not thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread. Thus, there are simply two rules to Android's single thread model:

1. Do not block the UI thread
2. Do not access the Android UI toolkit from outside the UI thread

2.1.7 Multi-Tasking

An application usually contains multiple activities. Each activity should be designed around a specific kind of action the user can perform and can start other activities. For example, an email application might have one activity to show a list of new email. When the user selects an email, a new activity opens to view that email.

An activity can even start activities that exist in other applications on the device. For example, if the application wants to send an email, we can define intent to perform a "send" action and include some data, such as an email address and a message. An activity from another application that declares itself to handle this kind of intent then opens. In this case, the intent is to send an email, so an email application's "compose" activity starts (if multiple activities support the same intent, then the system lets the user select which one to use). When the email is sent, your activity resumes and it seems as if the email activity was part of your application. Even though the activities may be from different applications, Android maintains this seamless user experience by keeping both activities in the same task.

A task is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the "back stack"), in the order in which each activity is opened.

The device Home screen is the starting place for most tasks. When the user touches an icon in the application launcher (or a shortcut on the Home screen), that application's task comes to the foreground. If no task exists for the application (the application has not been used recently), then a new task is created and the "main" activity for that application opens as the root activity in the stack.

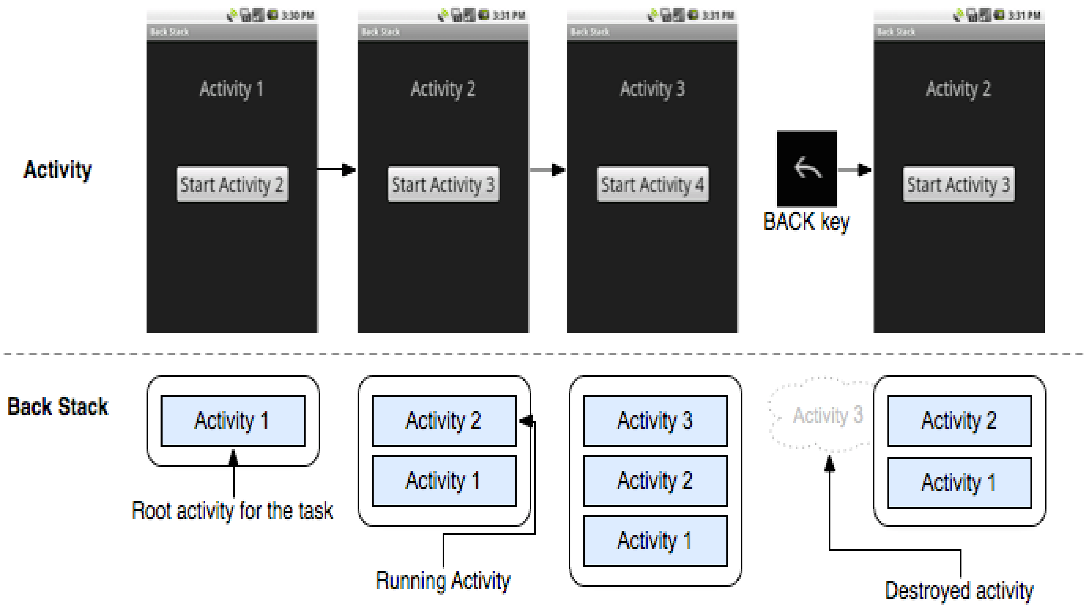


fig 2.3 Activity in background

When the current activity starts another, the new activity is pushed on the top of the stack and takes focus. The previous activity remains in the stack, but is stopped. When an activity stops, the system retains the current state of its user interface. When the user presses the BACK key, the current activity is popped from the top of the stack (the activity is destroyed) and the previous activity resumes (the previous state of its UI is restored). Activities in the stack are never rearranged, only pushed and popped from the stack—pushed onto the stack when started by the current activity and popped off when the user leaves it using the BACK key. As such, the back stack operates as a "last in, first out" object structure. Figure 1 visualizes this behavior with a timeline showing the progress between activities along with the current back stack at each point in time.

If the user continues to press BACK, then each activity in the stack is popped off to reveal the previous one, until the user returns to the Home screen (or to whichever activity was running when the task began). When all activities are removed from the stack, the task no longer exists.

A task is a cohesive unit that can move to the "background" when users begin a new task or go to the Home screen, via the HOME key. While in the background, all the activities in the task are stopped, but the back stack for the task remains intact—the task has simply lost focus while another task takes place. A task can then return to the "foreground" so users can pick up where they left off. Suppose, for example, that the current task (Task A) has three activities in its stack—two under the current activity. The user presses the HOME key, then starts a new application from the application launcher. When the Home screen appears, Task A goes into the background. When the new application starts, the system starts a task for that application (Task B) with its own stack of activities. After interacting with that application, the user returns Home again and selects the application that originally started Task A. Now, Task A comes to the foreground—all three activities in its stack are intact and the activity at the top of the stack resumes.

Chapter -3 SYSTEM DEVELOPMENT

3.1 System Requirements

The system and software requirements for developing Android applications using the Android SDK are as follows:

3.1.1 Supported Operating Systems

- Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Ubuntu Linux)
    - GNU C Library (glibc) 2.7 or later is required.
    - On Ubuntu Linux, version 8.04 or later is required.
    - 64-bit distributions must be capable of running 32-bit applications.

3.1.2 Supported Development Environments

Eclipse IDE

- Eclipse 3.4 (Ganymede) or greater
- Eclipse JDT plugin (included in most Eclipse IDE packages)
- If you need to install or update Eclipse, you can download it from http://www.eclipse.org/downloads/.

  Several types of Eclipse packages are available for each platform. For developing Android applications, we recommend that you install one of these packages:

    - Eclipse IDE for Java Developers
    - Eclipse Classic (versions 3.5.1 and higher)
    - Eclipse IDE for Java EE Developers
- JDK 5 or JDK 6 (JRE alone is not sufficient)
- Android Development Tools plugin (required)
- **Not** compatible with Gnu Compiler for Java (gcj)

Other development environments or IDEs

- JDK 5 or JDK 6 (JRE alone is not sufficient)
- Apache Ant 1.8 or later
- **Not** compatible with Gnu Compiler for Java (gcj)

3.1.3 Hardware requirements

The Android SDK requires disk storage for all of the components that you choose to install. The table below provides a rough idea of the disk-space requirements to expect, based on the components that you plan to use.

Component type	**Approximate size**	**Comments**
SDK Tools	35 MB	Required.
SDK Platform-tools	6 MB	Required.
Android platform (each)	150 MB	At least one platform is required.
SDK Add-on (each)	100 MB	Optional.
USB Driver for Windows	10 MB	Optional. For Windows only.
Samples (per platform)	10M	Optional.
Offline documentation	250 MB	Optional.

table 3.1 Components required by SDK

3.2 Material Design

You may have noticed a new look to your familiar Google Apps. Drive, Gmail, Calendar and the Docs editors on Android devices are all revamped to follow material design principles, with additional updates across platforms planned for the coming months. This guide explains how these changes will enhance your experience using these apps, and eventually all Google products.

3.2.1 What is material design? Google developed material design as a new visual language for its products that is consistent across devices, so that as you switch from one screen to another, the experience feels the same. The focus is on creating a consistent and predictable experience that refers to our material world. For example, we are all familiar with the way paper and ink behave in the real world. Aspects of that material behavior are reflected in this new design standard. nterface elements behave predictably in this new visual language. For example, tap a menu item (e.g. All) to display the menu as a temporary sheet of paper that always overlaps the app bar and then disappears, instead of behaving like an extension of the app bar.

When you use an application, your actions have a starting point. Movement from there is smooth and predictable. When you touch the menu icon to open the menu, it follows your touch to open in a smooth, natural manner. When you touch screen elements, they respond with a gray touch ripple effect, giving a clear response to your touch. Rather than impose the same layout for each view, the new design takes advantage of each device layout and adapts accordingly. One of the key features of this new design standard is consistency across platforms. The user experience is fundamentally the same on all devices and operating systems.

This is the tablet view for Gmail. Notice how all of the major elements are still there: The app bar, icons, avatars, and action button are all where you expect them to be. The additional space is used to present the selected conversation, which only adds to the usability of the app on this platform. Room to grow and improve Google recognizes that good design is never finished, and the best designs improve and contribute to better designs over time. This new design system is intended to grow and evolve with our customers and their needs. We are aiming for a unified and consistent experience across all Google products in the coming months.

A material metaphor is the unifying theory of a rationalized space and a system of motion. The material is grounded in tactile reality, inspired by the study of paper and ink, yet technologically advanced and open to imagination and magic.

Surfaces and edges of the material provide visual cues that are grounded in reality. The use of familiar tactile attributes helps users quickly understand affordances. Yet the flexibility of the material creates new affordances that supercede those in the physical world, without breaking the rules of physics.

The fundamentals of light, surface, and movement are key to conveying how objects move, interact, and exist in space and in relation to each other. Realistic lighting shows seams, divides space, and indicates moving parts.

The foundational elements of print-based design – typography, grids, space, scale, color, and use of imagery – guide visual treatments. These elements do far more than please the eye. They create hierarchy, meaning, and focus. Deliberate color choices, edge-to-edge imagery, large-scale typography, and intentional white space create a bold and graphic interface that immerse the user in the experience.

An emphasis on user actions makes core functionality immediately apparent and provides waypoints for the user.
Motion respects and reinforces the user as the prime mover. Primary user actions are inflection points that initiate motion, transforming the whole design.

All action takes place in a single environment. Objects are presented to the user without breaking the continuity of experience even as they transform and reorganize.

Motion is meaningful and appropriate, serving to focus attention and maintain continuity. Feedback is subtle yet clear. Transitions are efficient yet coherent. The material environment is a 3D space, which means all objects have x, y, and z dimensions. The z-axis is perpendicularly aligned to the plane of the display, with the positive z-axis extending towards the viewer. Every sheet of material occupies a single position along the z-axis and has a standard 1dp thickness, equivalent to one pixel of thickness on screens with a pixel density of 160.

On the web, the z-axis is used for layering and not for perspective. The 3D world is emulated by manipulating the y-axis. Within the material environment, virtual lights illuminate the scene. Key lights create directional shadows, while ambient light creates soft shadows from all angles.

Shadows in the material environment are cast by these two light sources. In Android development, shadows occur when light sources are blocked by sheets of material at various positions along the z-axis. On the web, shadows are depicted by manipulating the y-axis only. The following example shows the card with a height of 6dp.

3.2.2 Usability

A well-designed product is accessible to users of all abilities, including those with low vision, blindness, hearing impairments, cognitive impairments, or motor impairments. Improving your product’s accessibility enhances the usability for everyone who uses it. It’s also the right thing to do.

Material design's built-in accessibility considerations will help you accommodate all of your users. This section primarily applies to mobile UI design. For more information on designing and developing fully accessible products, visit the Google accessibility site.

3.3 Working of the app

1. USE CASE DIAGRAM

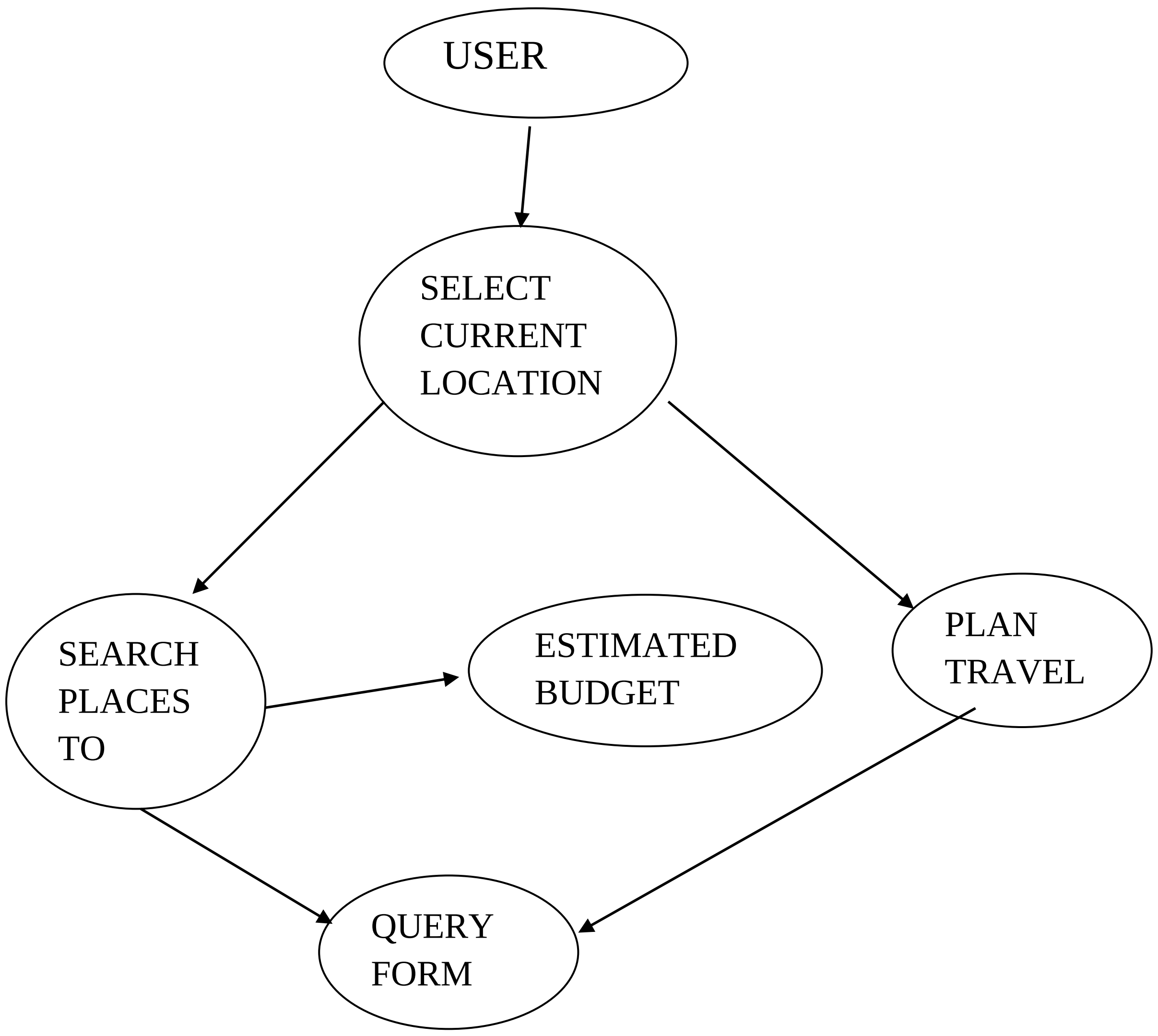


fig 3.1a

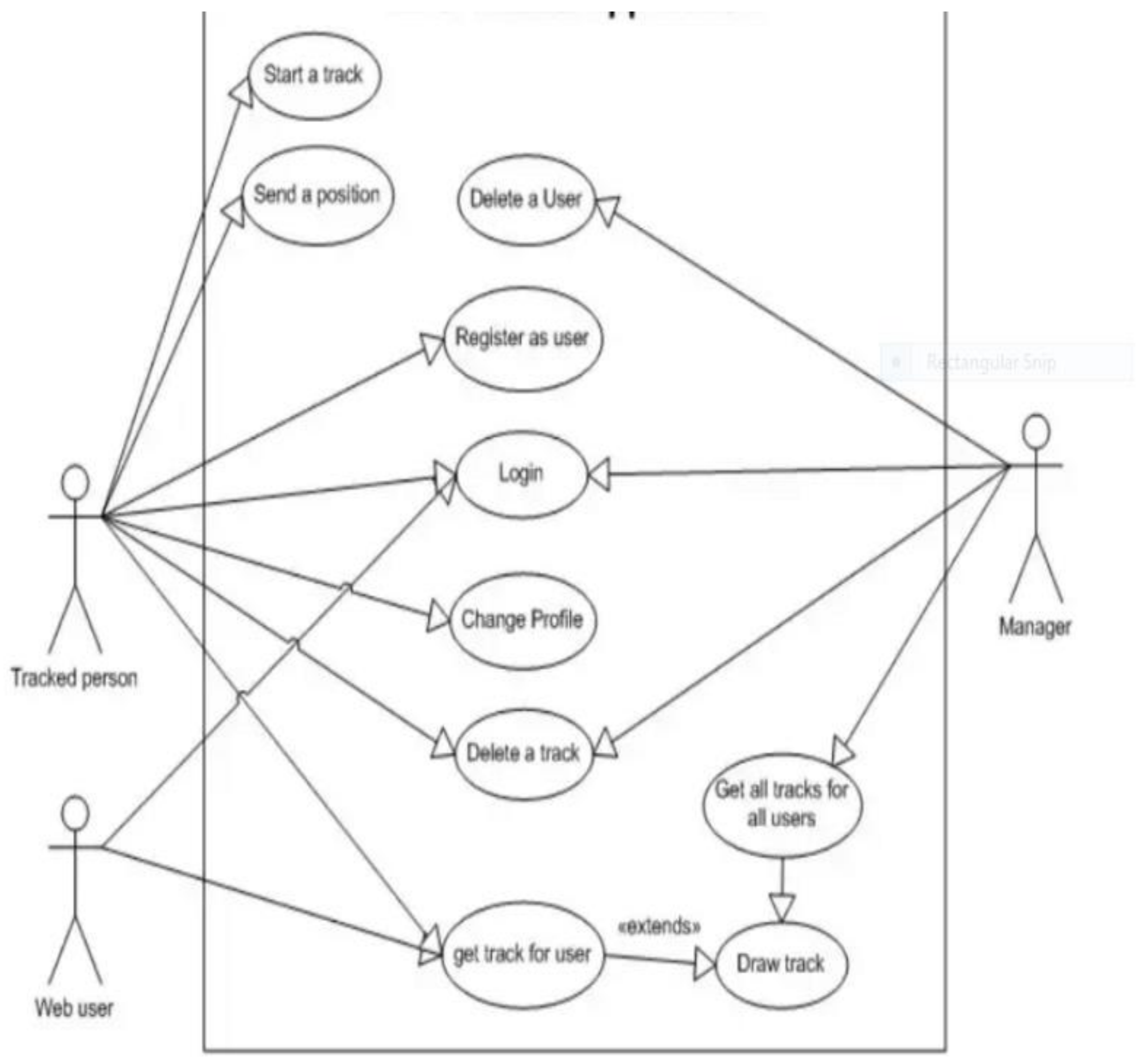


fig 3.1b
This use case shows the various ways in which the app interacts with the user

2. Activity Diagram:(Login or New Registration)

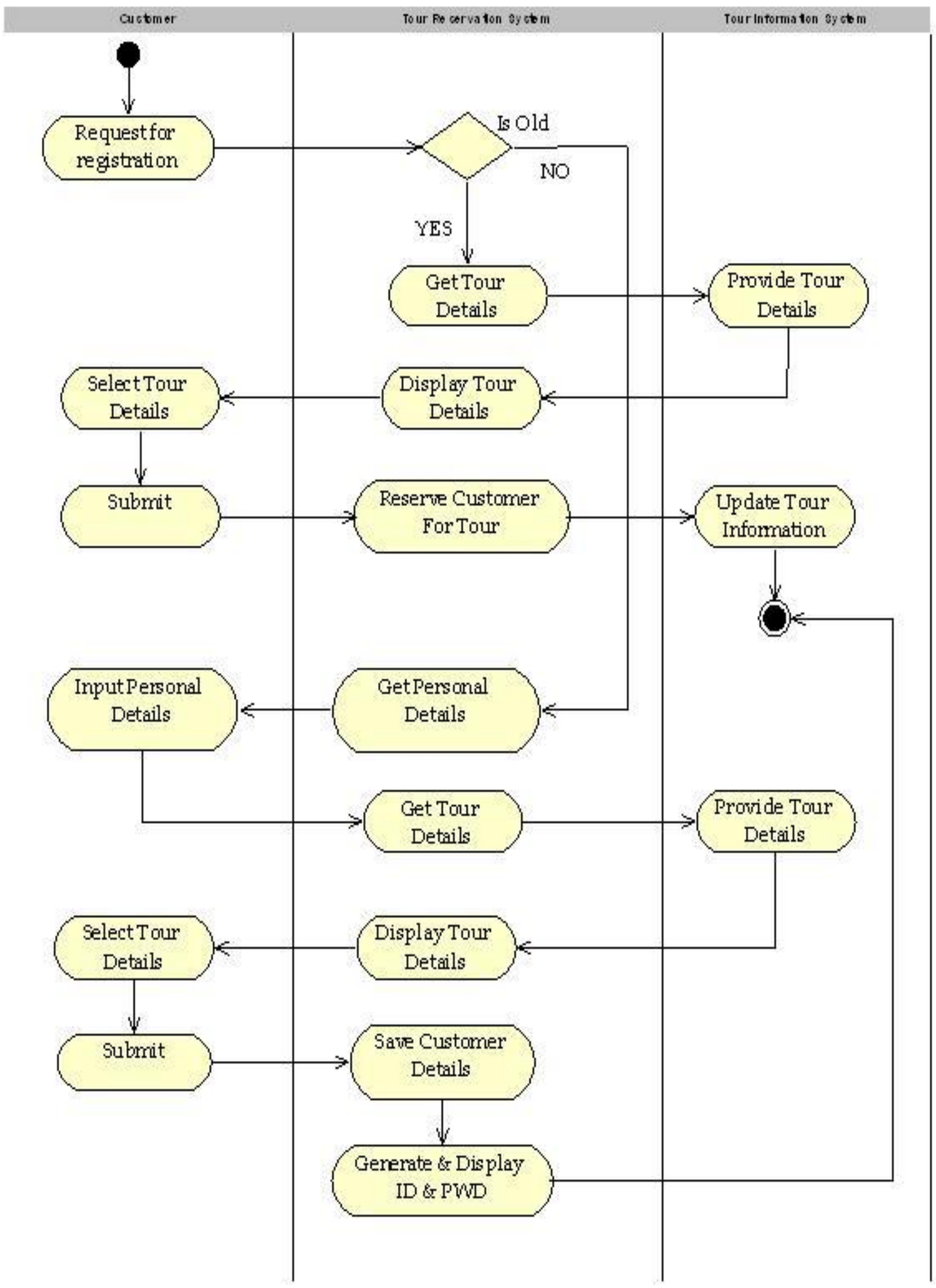


fig 3.2

This Activity Diagram shows graphical representation of workflows of
stepwise activities and actions of our project. How new user can register or existing user can login and choose packages(Future Aspects)

3. Collaboration Diagram:

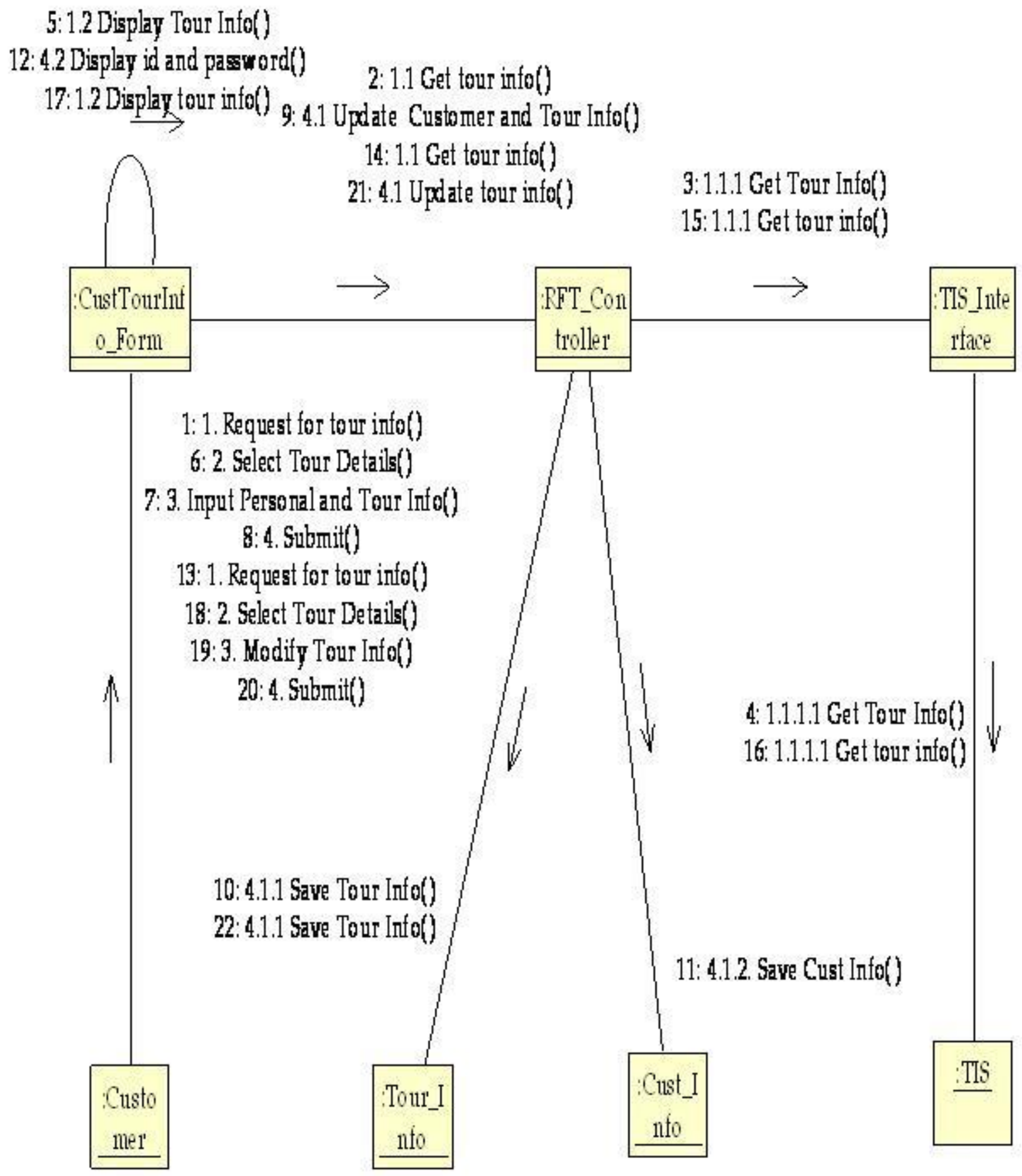

5: 1.2 Display Tour Info( )
12: 4.2 Display id and password()
17: 1.2 Display tour info( )
2: 1.1 Get tour info()
9: 4.1 Update Customer and Tour Info()
14: 1.1 Get tour info( )
21: 4.1 Update tour info( )
3: 1.1.1 Get Tour Info()
15: 1.1.1 Get tour info()
:CustTourInfo_Form
:RFT_Controller
:TIS_Interface
1: 1. Request for tour info()
6: 2. Select Tour Details()
7: 3. Input Personal and Tour Info()
8: 4. Submit()
13: 1. Request for tour info()
18: 2. Select Tour Details()
19: 3. Modify Tour Info()
20: 4. Submit()
4: 1.1.1.1 Get Tour Info()
16: 1.1.1.1 Get tour info()
10: 4.1.1 Save Tour Info()
22: 4.1.1 Save Tour Info()
11: 4.1.2. Save Cust Info( )
:Customer
:Tour_Info
:Cust_Info
:TIS

fig 3.3

This diagram tells how components of our app are interacting with each other as the user uses it.

4. Component Diagram:

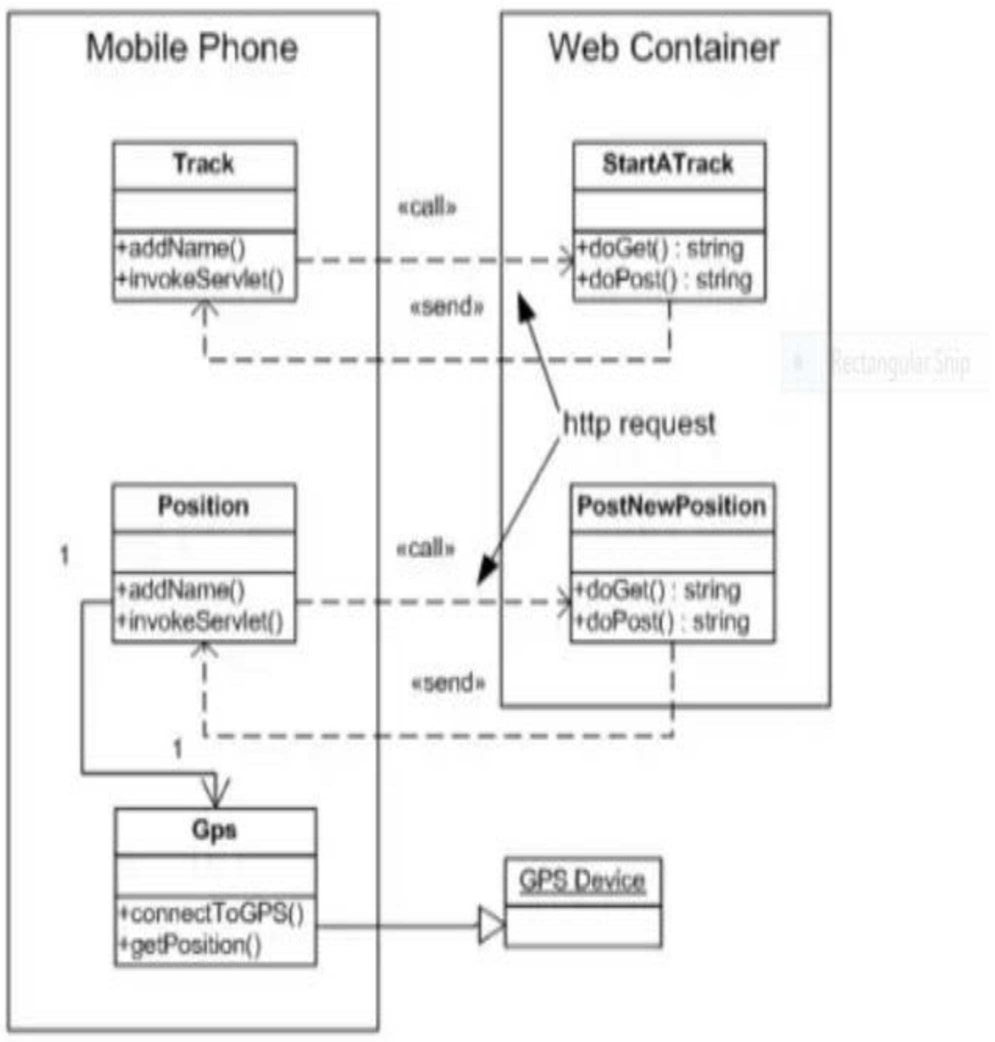


fig 3.4

This diagram shows the physical aspects of a system.

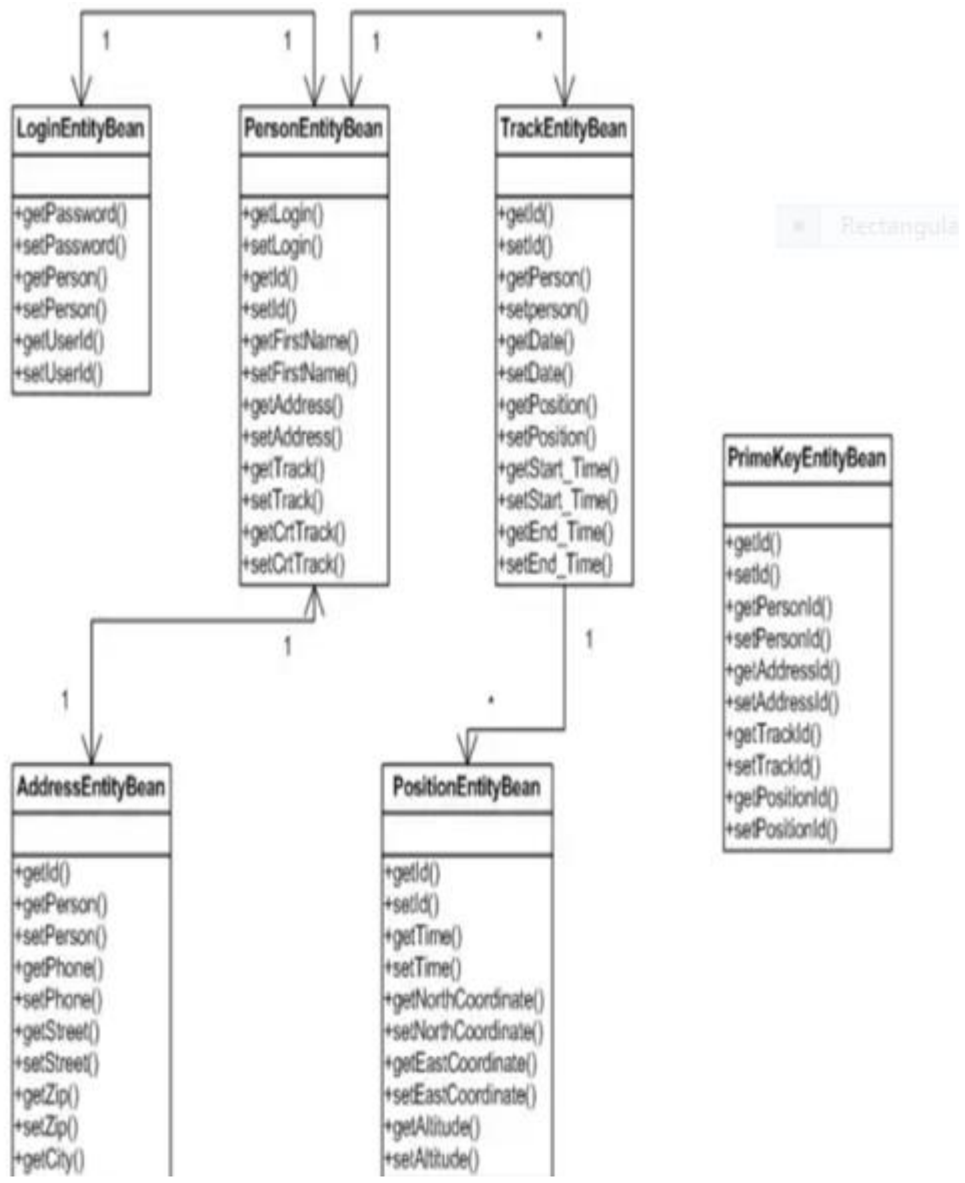


fig 3.5

These are the components used to build our app.

5. Deployment Diagram:

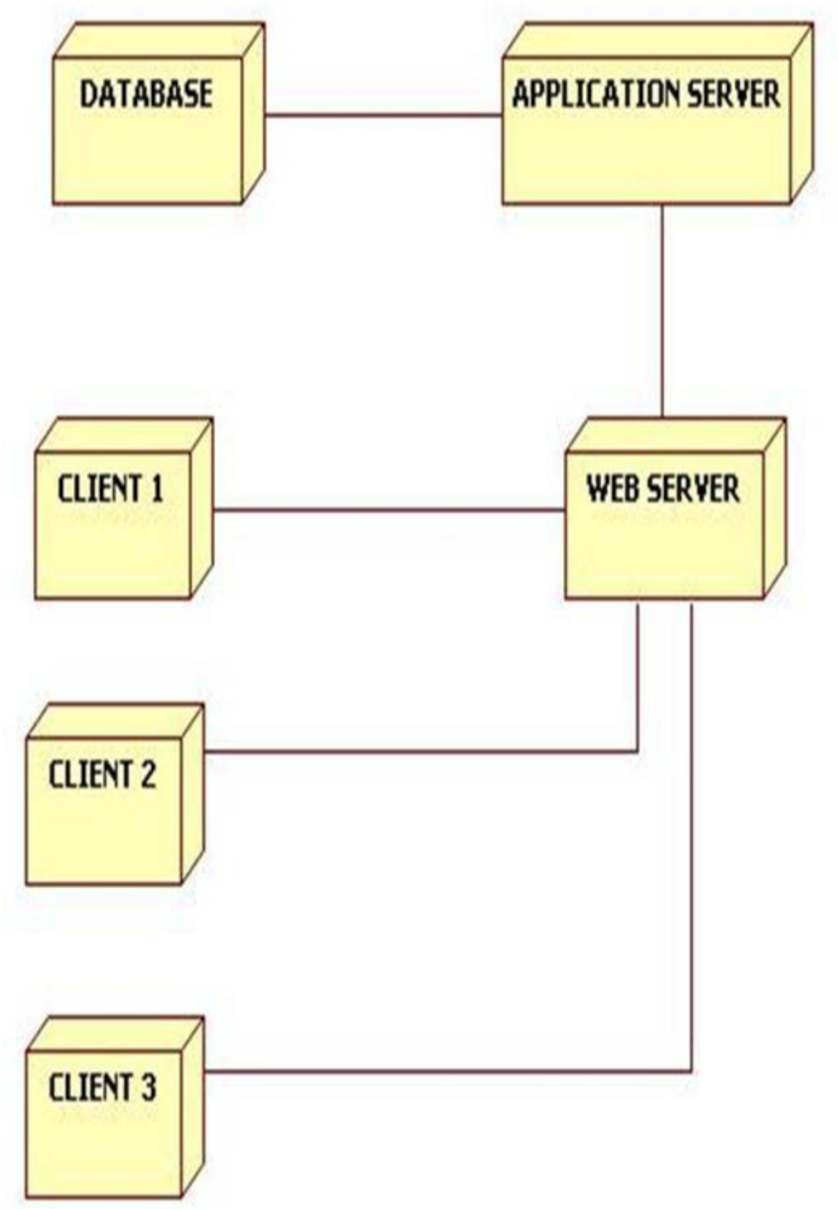


fig 3.6

These are the hardware components and software components existing .

6. Sequence Diagram:

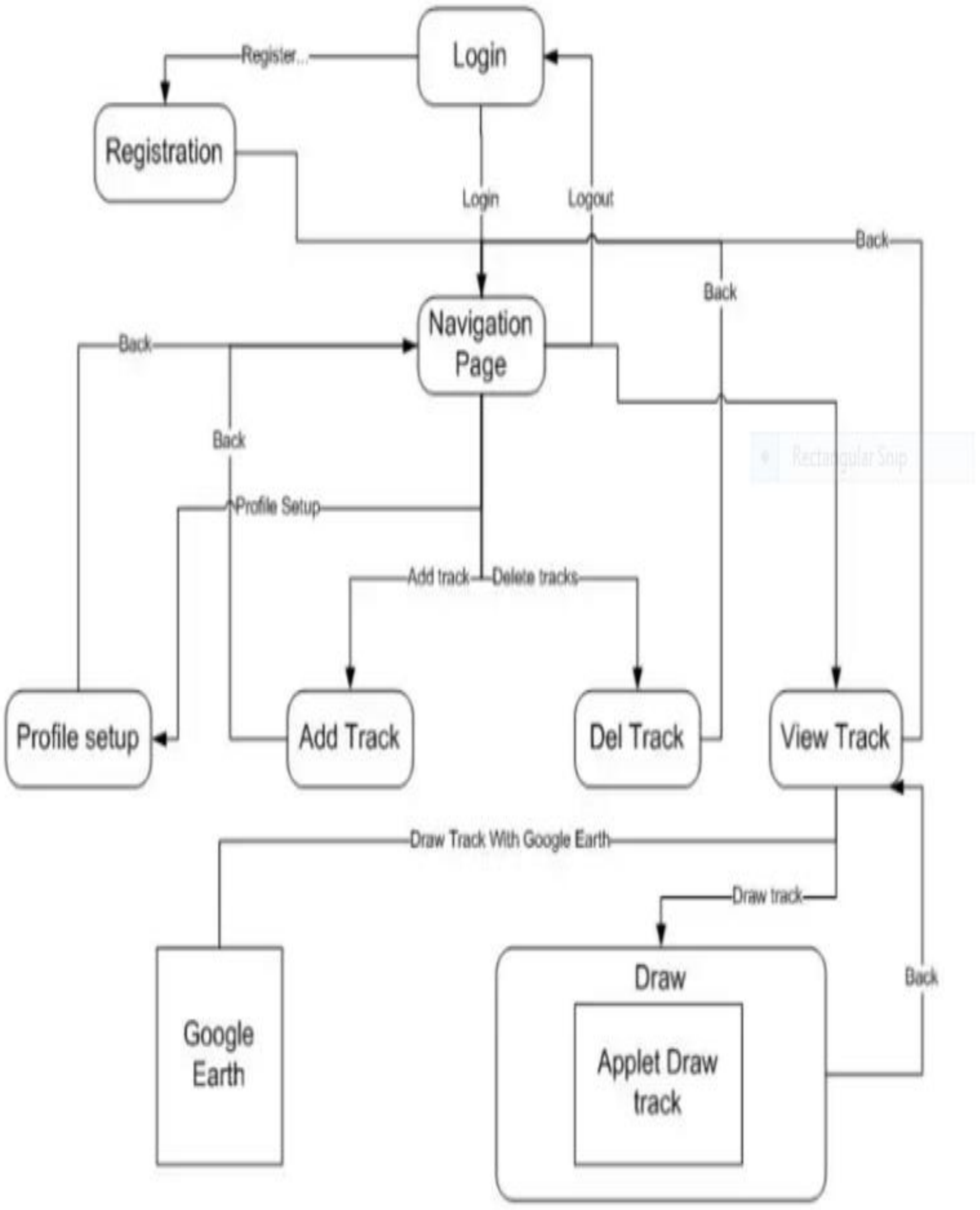


fig 3.7

this diagram shoes how objects are intercating with each other in the system

7. Class Diagram:

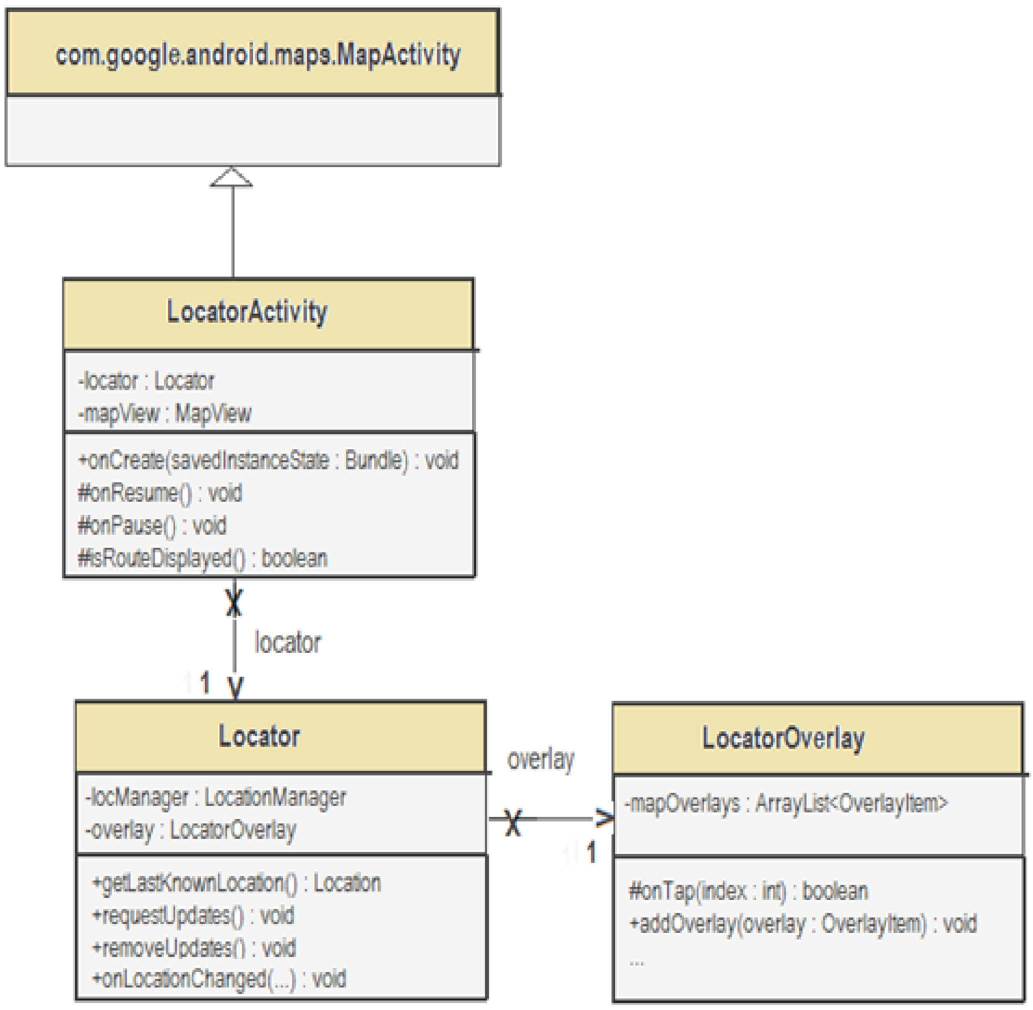

com.google.android.maps.MapActivity
LocatorActivity
-locator : Locator
-mapView : MapView
+onCreate(savedInstanceState : Bundle) : void
#onResume() : void
#onPause() : void
#isRouteDisplayed() : boolean
locator
1
Locator
-locManager : LocationManager
-overlay : LocatorOverlay
+getLastKnownLocation() : Location
+requestUpdates() : void
+removeUpdates() : void
+onLocationChanged(...) : void
overlay
1
LocatorOverlay
-mapOverlays : ArrayList<OverlayItem>
#onTap(index : int) : boolean
+addOverlay(overlay : OverlayItem) : void
...

fig 3.8

this diagram visualizes, describes and document different aspects of a system and also tells how to construct executable code of the software application.

8. Flow Diagram:

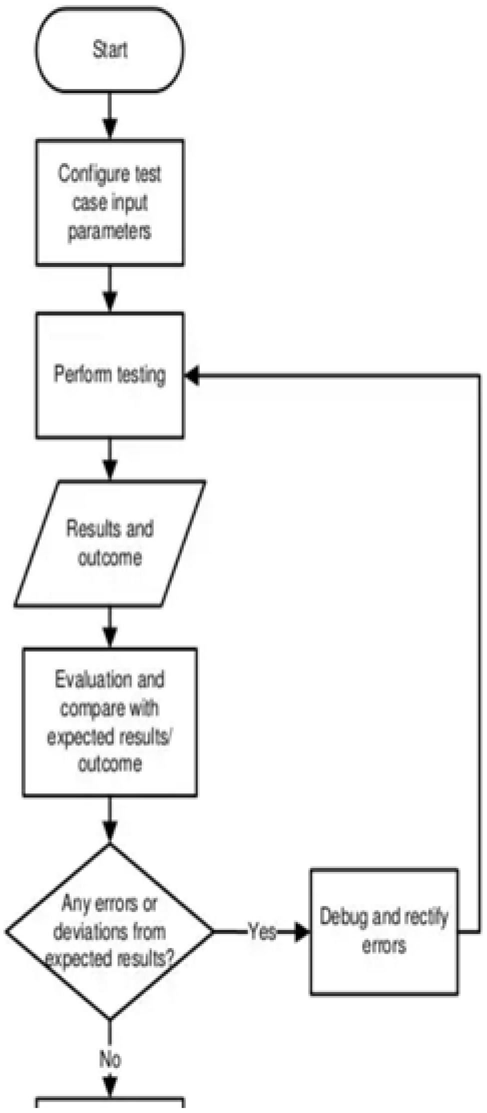

Start
Configure test case input parameters
Perform testing
Results and outcome
Evaluation and compare with expected results/ outcome
Any errors or deviations from expected results?
Yes
Debug and rectify errors
No

fig 3.9a

this diagram reveals the underlined sructure

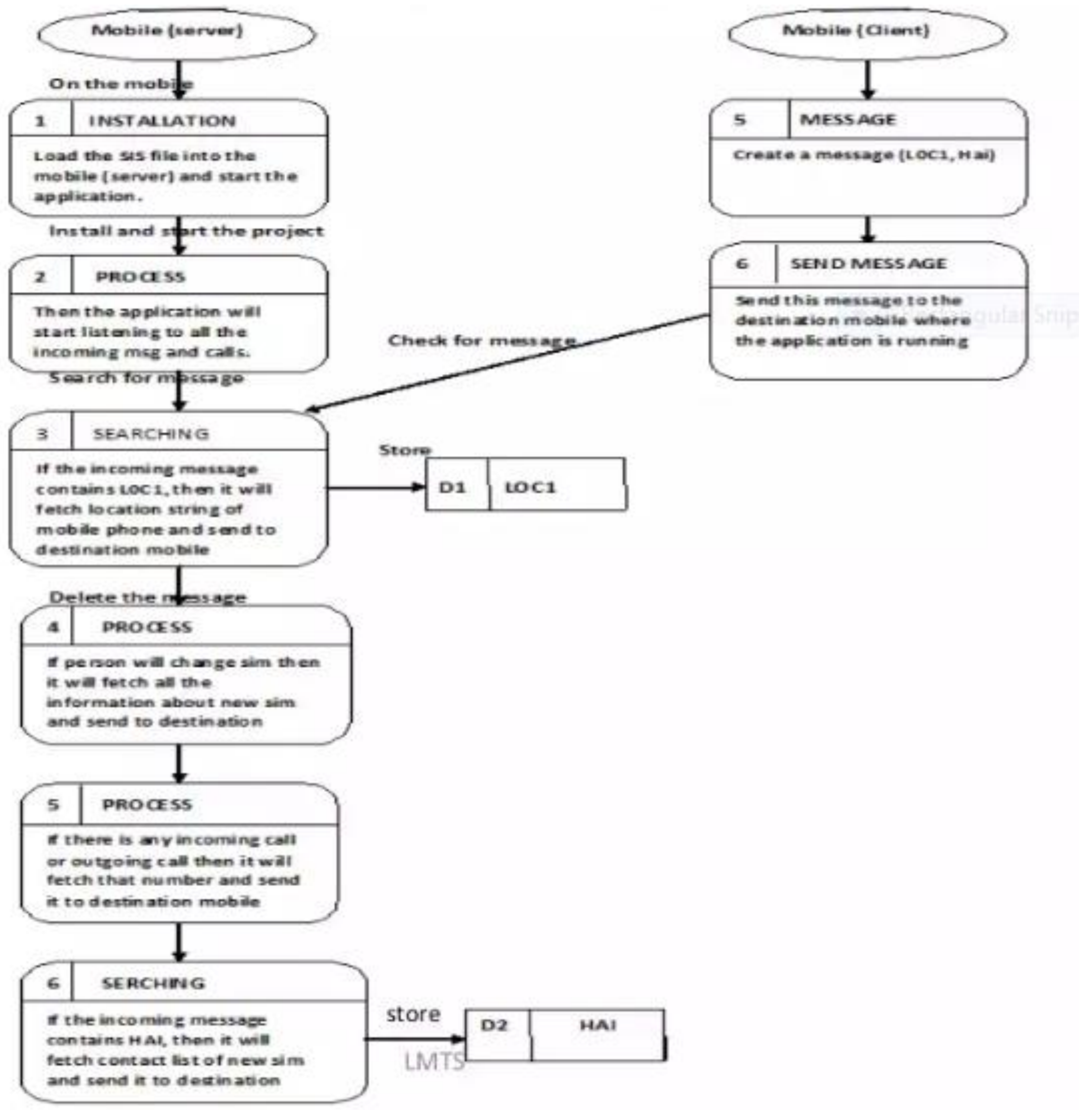

Mobile (server)
On the mobile
1 INSTALLATION
Load the SIS file into the mobile (server) and start the application.
Install and start the project
2 PROCESS
Then the application will start listening to all the incoming msg and calls.
Search for message
3 SEARCHING
If the incoming message contains LOC1, then it will fetch location string of mobile phone and send to destination mobile
Store
D1 LOC1
Delete the message
4 PROCESS
If person will change sim then it will fetch all the information about new sim and send to destination
5 PROCESS
If there is any incoming call or outgoing call then it will fetch that number and send it to destination mobile
6 SERCHING
If the incoming message contains HAI, then it will fetch contact list of new sim and send it to destination
store
D2 HAI
LMTS
Mobile (Client)
5 MESSAGE
Create a message (LOC1, Hai)
6 SEND MESSAGE
Send this message to the destination mobile where the application is running
Check for message

fig 3.9b

3.4 Screenshots:

1 Splash Screen: splash screen displayed on starting app. Uses animation to move the icons on the splash screen. The icons rotates at their own position.

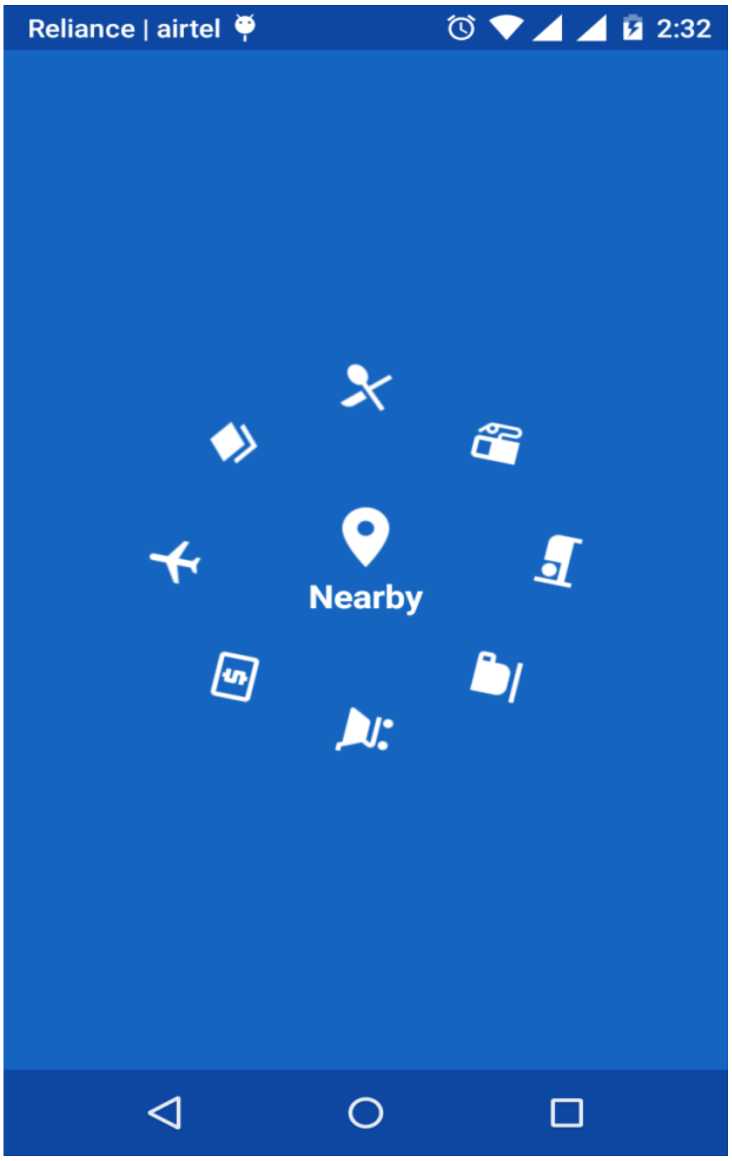


fig 3.10

2. Categories: category list as tiles view, there are over 50 categories picked up from google developers website from food to health which makes it easy for the user to locate any type of place of interest.

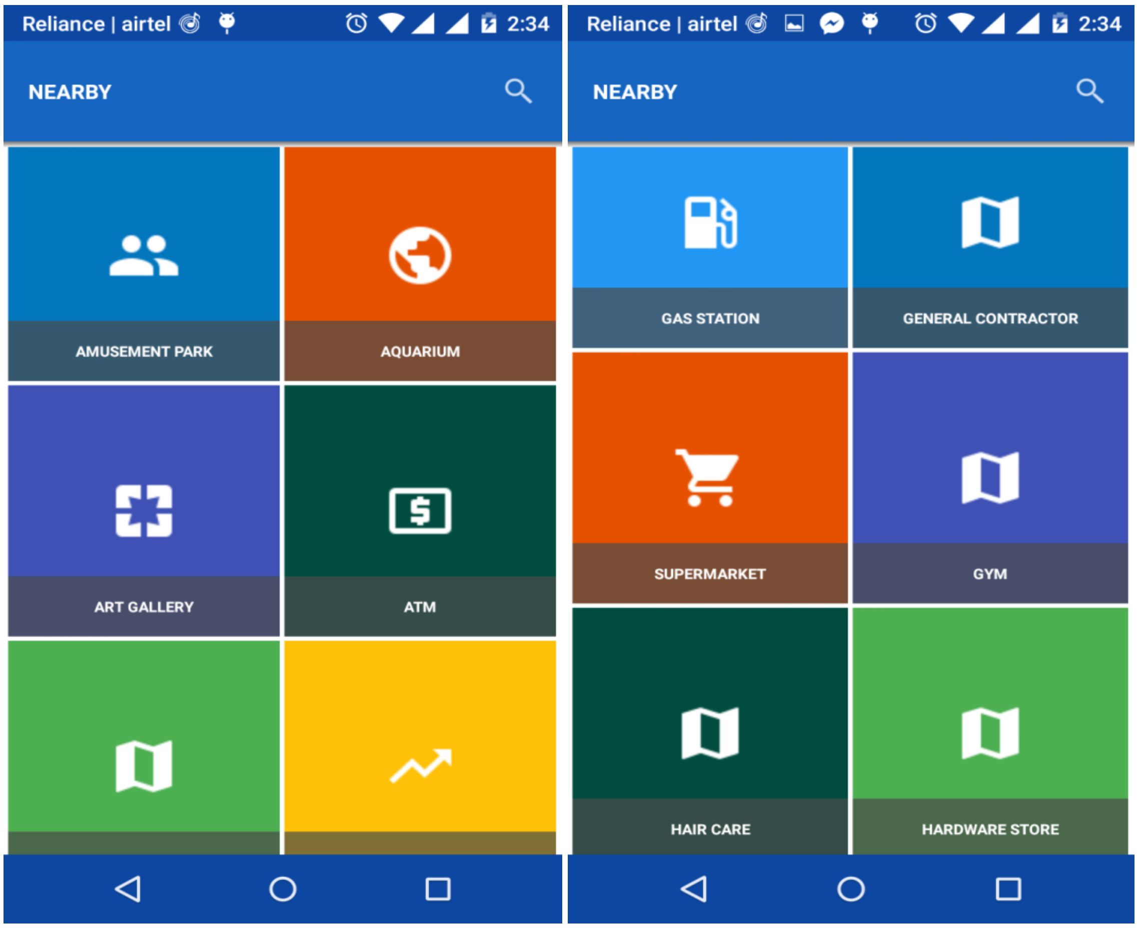
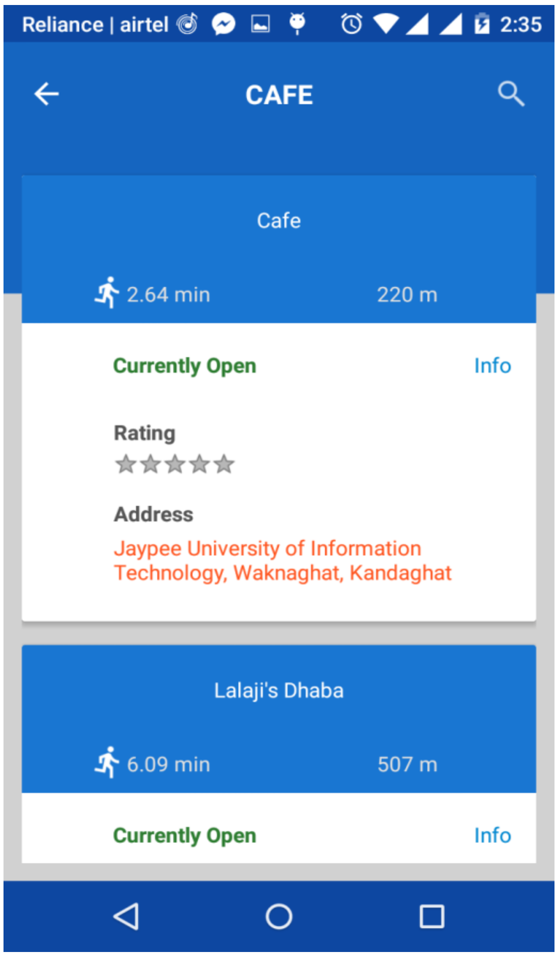


fig 3.11

3. Details: on clicking cafe it list down the all the cafes within 10 km of range with its current status, rating and address. Uses material design concept to beautifully diplay the details. On selecting cafe it takes you to new activity which displays its position on map its reviews and gallery. It also displays two icons on map one for directions and other for google maps page



Reliance | airtel 2:35

CAFE

Cafe

2.64 min 220 m

Currently Open Info

Rating

★★★★★

Address

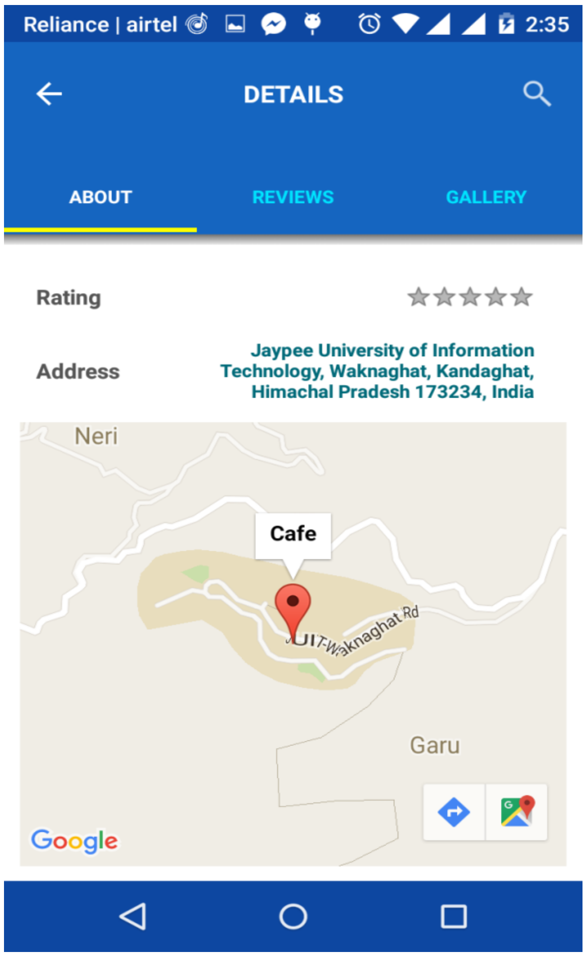
Jaypee University of Information Technology, Waknaghat, Kandaghat

Lalaji's Dhaba

6.09 min 507 m

Currently Open Info

fig 3.12



Reliance | airtel 2:35

DETAILS

ABOUT REVIEWS GALLERY

Rating ★★★★★

Address Jaypee University of Information Technology, Waknaghat, Kandaghat, Himachal Pradesh 173234, India

fig 3.13

4.Maps: shows directions for the place selected. It shows paths in yellow color and also diatance and time by foot,car or tran if available.

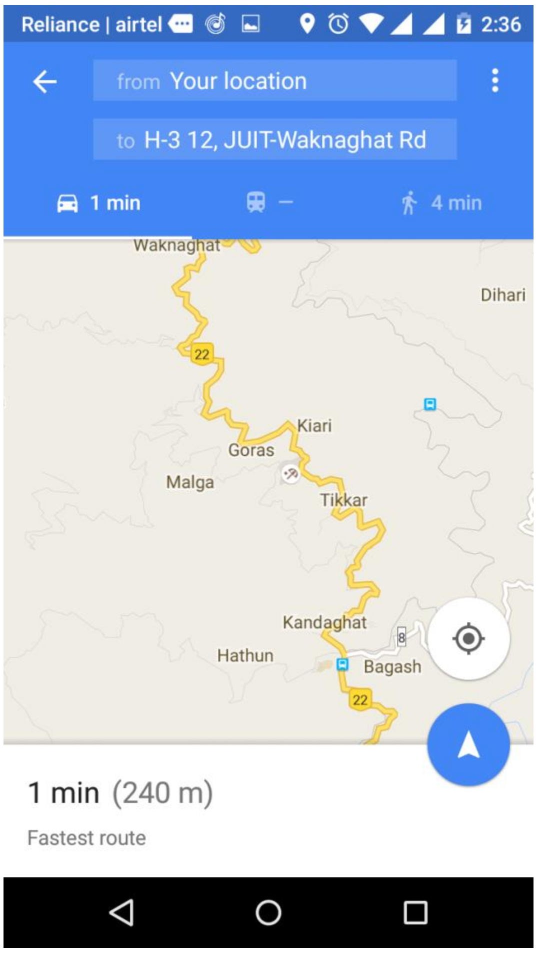
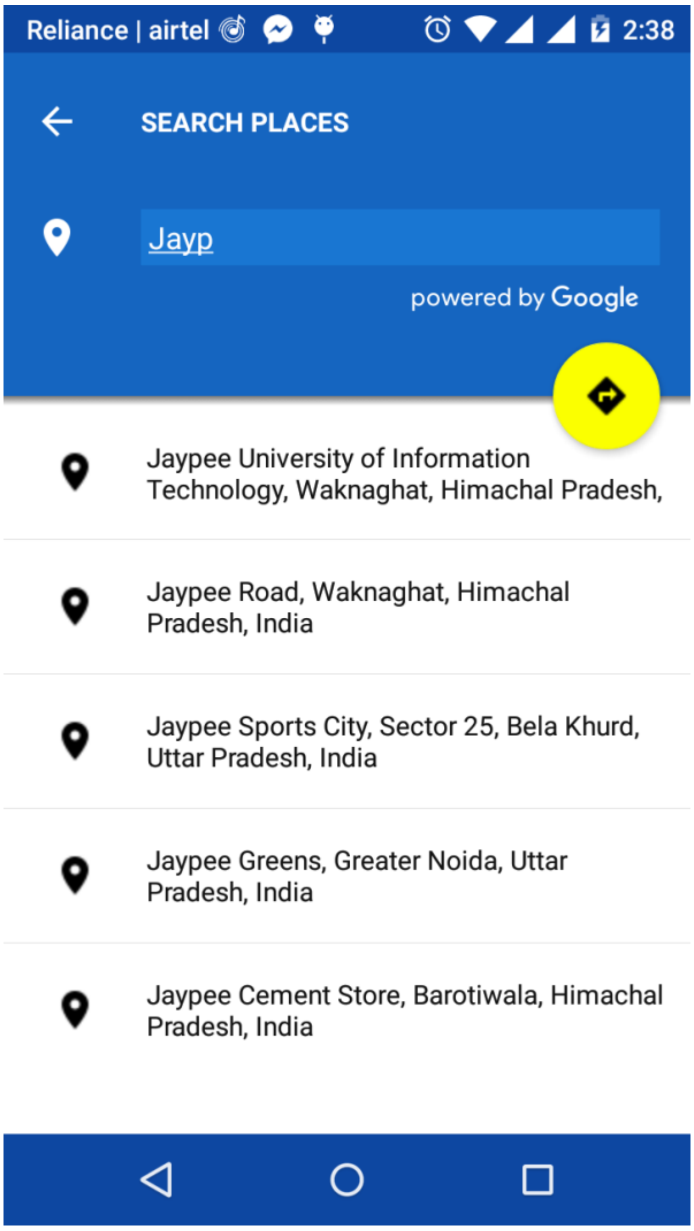


fig 3.14

5. Searching: search option with autocomplete facility and on clicking yellow sign it shows direction on google maps. User can search any place whether in locality or not, it can be any place in any part of the earth.



Jaypee University of Information Technology, Waknaghat, Himachal Pradesh,

Jaypee Road, Waknaghat, Himachal Pradesh, India

Jaypee Sports City, Sector 25, Bela Khurd, Uttar Pradesh, India

Jaypee Greens, Greater Noida, Uttar Pradesh, India

Jaypee Cement Store, Barotiwala, Himachal Pradesh, India

fig 3.15

6. Gallery: gallery activity shows images of selected place. It picks up pictures from google database if anyone has uploaded by tagging that place.

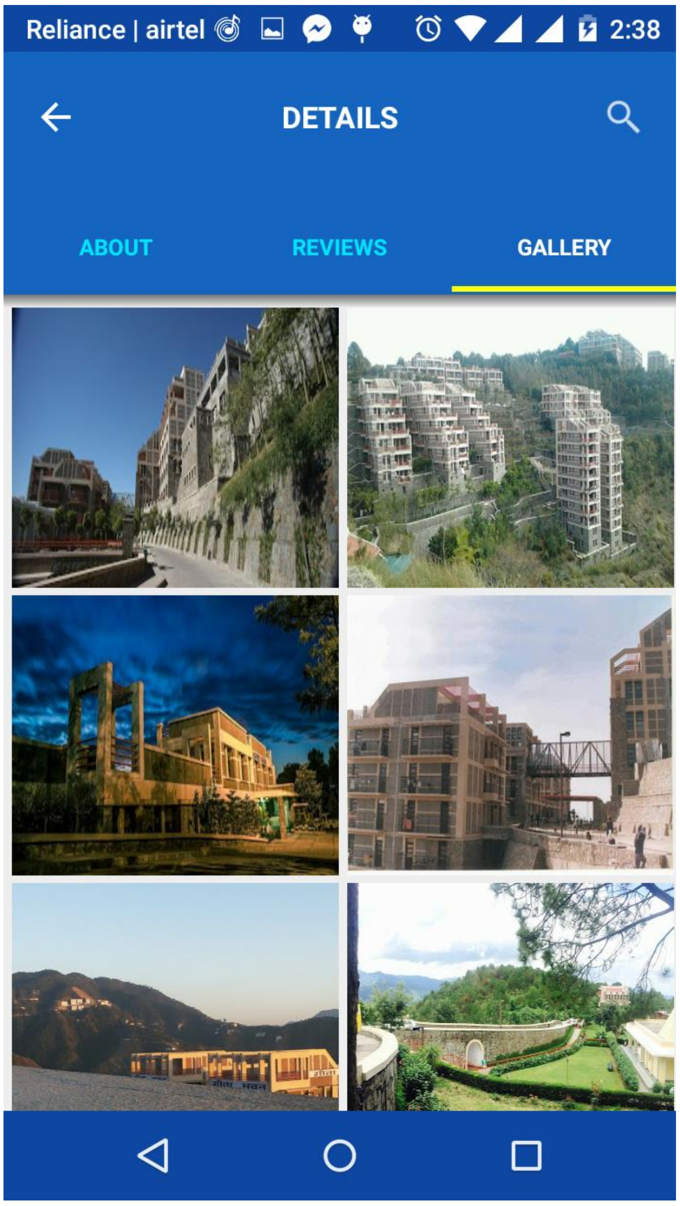


fig 3.16

Chapter-4 PERFORMANCE ANALYSIS

4.1 Testing Fundamentals

Android provides an integrated framework that helps you test all aspects of your app. The Android platform and Testing Support Library include tools and APIs for setting up and running test apps within an emulator or physical device.

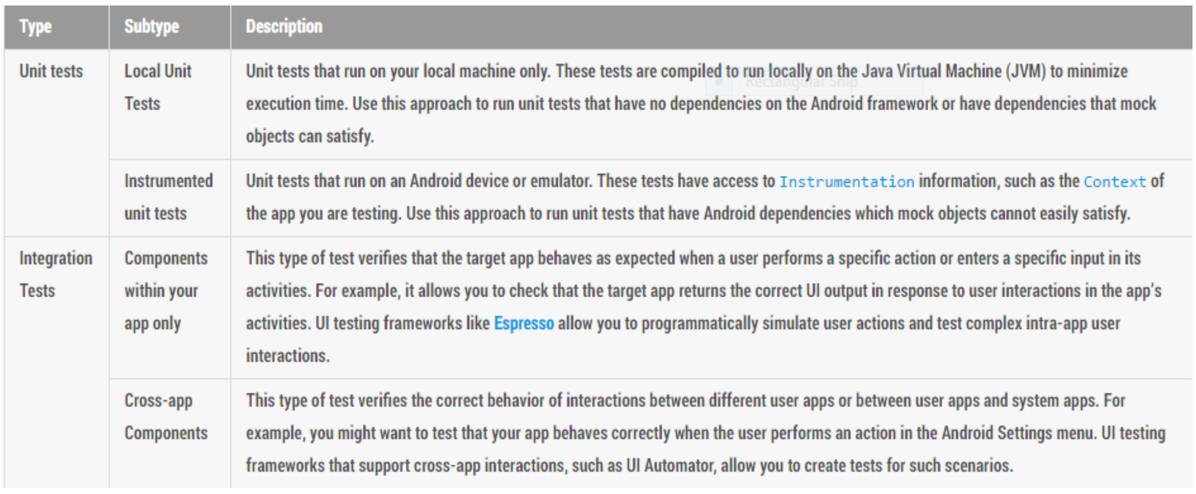
This document guides you through key concepts related to Android app testing, and provides an overview of the testing tools and APIs developed by Google. If you want to skip the conceptual overview, and start learning how to build and run your tests using these APIs and tools, go to Getting started with testing in Android Studio. If you are not using Android Studio, go toTesting from the command line.

Testing Concepts

Android testing is based on JUnit. In general, a JUnit test is a method whose statements test a part of the app. You organize test methods into classes called test cases, and group test cases into test suites.

In JUnit, you build one or more test classes and use a test runner to execute them on your local machine. With Android Studio, you can build one or more test source files into an Android test app and use it to test your app on the Emulator or physical Android device.

The structure of your test code and the way you build and run the tests in Android Studio depend on the type of testing you are performing. The following table summarizes the common testing types for Android:



Type	Subtype	Description
Unit tests	Local Unit Tests	Unit tests that run on your local machine only. These tests are compiled to run locally on the Java Virtual Machine (JVM) to minimize execution time. Use this approach to run unit tests that have no dependencies on the Android framework or have dependencies that mock objects can satisfy.
	Instrumented unit tests	Unit tests that run on an Android device or emulator. These tests have access to `Instrumentation` information, such as the `Context` of the app you are testing. Use this approach to run unit tests that have Android dependencies which mock objects cannot easily satisfy.
Integration Tests	Components within your app only	This type of test verifies that the target app behaves as expected when a user performs a specific action or enters a specific input in its activities. For example, it allows you to check that the target app returns the correct UI output in response to user interactions in the app's activities. UI testing frameworks like Espresso allow you to programmatically simulate user actions and test complex intra-app user interactions.
	Cross-app Components	This type of test verifies the correct behavior of interactions between different user apps or between user apps and system apps. For example, you might want to test that your app behaves correctly when the user performs an action in the Android Settings menu. UI testing frameworks that support cross-app interactions, such as UI Automator, allow you to create tests for such scenarios.

table 4.1 (testing types)

Instrumentation

Android instrumentation is a set of control methods, or hooks, in the Android system. These hooks control an Android component independently of its normal lifecycle. They also control how Android loads apps.

The following diagram summarizes the testing framework:

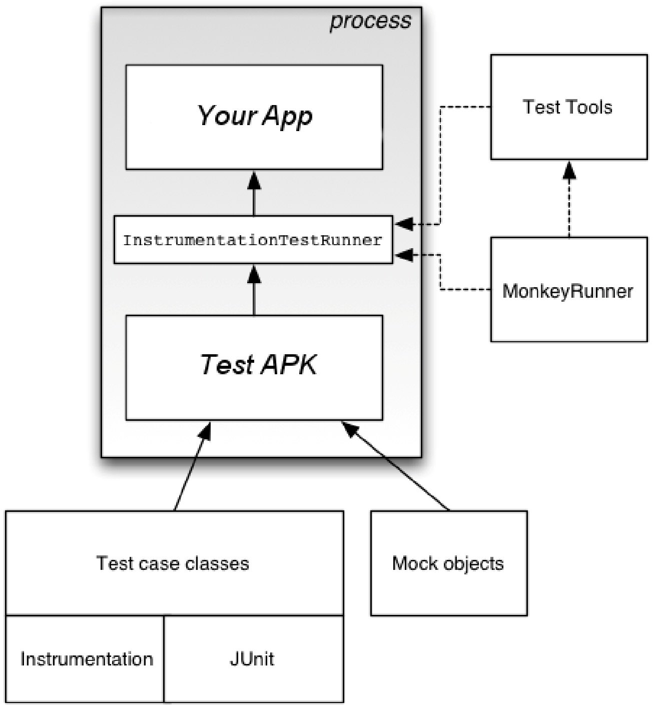


fig 4.1

Normally, an Android component runs in a lifecycle that the system determines. For example, an Acticity object's lifecycle starts when an Intent activates the Acticity. The system calls the object's oncreate() method, on then the onResume() method. When the user starts another app, the system calls the onPause() method. If the Activity code calls the finsih() method, the system calls the onDestroy() method. The Android framework

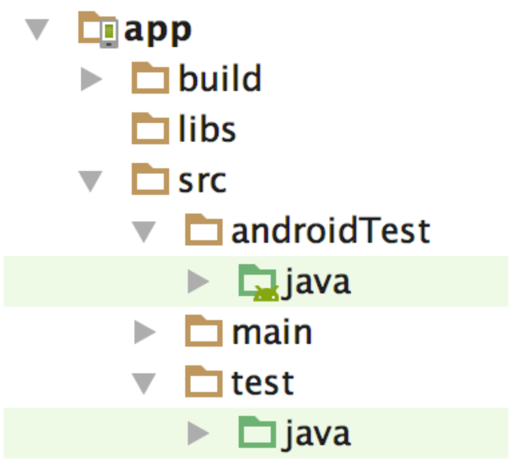
API does not provide a way for your code to invoke these callback methods directly, but you can do so using instrumentation.

The system runs all the components of an app in the same process. You can allow some components, such as content providers, to run in a separate process, but you typically can't force an app onto the same process as another running app.

Instrumentation tests, however, can load both a test APK of your test classes and your app's APK into the same process. Since the components of your app and their tests are in the same process, your tests can invoke methods, and modify and examine fields in your app.

Testing Source Sets:

When you create a new app module, Android Studio creates the src/test/ and src/androidTest/ source set directories for you. Place the test classes you want to run locally on your machine in the test/ source set and the test classes you want to run on an actual Android device in the androidTest/ source set. Gradle uses the androidTest/source set when generating the test APK you use to test your app. To learn more about build variants and source sets, read the Configure your build overview.



```
app
  build
  libs
  src
    androidTest
      java
    main
    test
      java
```

fig 4.2 Default app module test directories.

Tests in the androidTest/ source set are common to all your build variants. However, you can create additional source set directories for tests that are specific to certain build variants:

```
src/
   main/
   androidTest/
```

```
    flavor1/

    androidTestFlavor1/


    flavor2/

    androidTestFlavor2/
```

For example, when building a test APK for the "flavor1" version of your app, Gradle uses both the androidTestFlavor1/ and androidTest/ source sets. By default, all tests run against the debug build type. You can change this to another build type by using the testBuildType property in your module-level build.gradle file, as shown in the following code snippet.

```
android {
    ...
    testBuildType "staging"
}
```

Gradle automatically generates manifest files for your **androidTest/** source sets. Optionally, you can create your own manifest, for example, to specify a different value for minSdkVersion or register run listeners just for your tests. When building your app, Gradle merges multiple manifest files into one manifest.

4.2 Testing APIs:

The following list summarizes the common APIs related to app testing for Android.

JUnit

You should write your unit or integration test class as a JUnit 4 test class. JUnit is the most popular and widely-used unit testing framework for Java. The framework offers a convenient way to perform common setup, teardown, and assertion operations in your test.

JUnit 4 allows you to write tests in a cleaner and more flexible way than its predecessor versions. Unlike the previous approach to Android unit testing based on JUnit 3, with JUnit 4, you do not need to extend the junit.framework.TestCase class. You also do not need to prepend the test keyword to your test method name, or use any classes in thejunit.framework or junit.extensions package.

A basic JUnit 4 test class is a Java class that contains one or more test methods. A test method begins with the @Test annotation and contains the code to exercise and verify a single functionality (that is, a logical unit) in the component that you want to test.

The following snippet shows an example JUnit 4 integration test that uses the Espresso APIs to perform a click action on a UI element, then checks to see if an expected string is displayed.

```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class MainActivityInstrumentationTest {

    @Rule
    public ActivityTestRule mActivityRule = new ActivityTestRule<>(
            MainActivity.class);

    @Test
    public void sayHello(){
        onView(withText("Say hello!")).perform(click());

        onView(withId(R.id.textView)).check(matches(withText("Hello, World!")));
    }
}
```

In your JUnit 4 test class, you can call out sections in your test code for special processing by using the following annotations:

- @Before: Use this annotation to specify a block of code that contains test setup operations. The test class invokes this code block before each test. You can have multiple@Before methods but the order in which the test class calls these methods is not guaranteed.
- @After: This annotation specifies a block of code that contains test tear-down operations. The test class calls this code block after every test method. You can define multiple @After operations in your test code. Use this annotation to release any resources from memory.
- @Test: Use this annotation to mark a test method. A single test class can contain multiple test methods, each prefixed with this annotation.

- @Rule: Rules allow you to flexibly add or redefine the behavior of each test method in a reusable way. In Android testing, use this annotation together with one of the test rule classes that the Android Testing Support Library provides, such as Activity Test Rule or ServiceTestRule.
- @BeforeClass: Use this annotation to specify static methods for each test class to invoke only once. This testing step is useful for expensive operations such as connecting to a database.
- @AfterClass: Use this annotation to specify static methods for the test class to invoke only after all tests in the class have run. This testing step is useful for releasing any resources allocated in the @BeforeClass block.
- @Test(timeout=<milliseconds>): Some annotations support the ability to pass in elements for which you can set values. For example, you can specify a timeout period for the test. If the test starts but does not complete within the given timeout period, it automatically fails. You must specify the timeout period in milliseconds, for example:@Test(timeout=5000).

For more annotations, see the documentation for JUnit Annotations and the Android-Specific Annotations.

You use the JUnit Assert class to verify the correctness of an object's state. The assert methods compare values you expect from a test to the actual results and throw an exception if the comparison fails. Assertion Classes describes these methods in more detail.

Android Testing Support Library APIs

The Android Testing Support Library provides a set of APIs that allow you to quickly build and run test code for your apps, including JUnit 4 and functional UI tests. The library includes the following instrumentation-based APIs that are useful when you want to automate your tests:

AndroidJUnitRunner

A JUnit 4-compatible test runner for Android.

Espresso

A UI testing framework; suitable for functional UI testing within an app.

UI Automator

A UI testing framework suitable for cross-app functional UI testing between both system and installed apps.

Assertion classes

Because Android Testing Support Library APIs extend JUnit, you can use assertion methods to display the results of tests. An assertion method compares an actual value returned by a test to an expected value, and throws an AssertionException if the comparison test fails. Using assertions is more convenient than logging, and provides better test performance.

To simplify test development, you should use the Hamcrest library, which lets you create more flexible tests using the Hamcrest matcher APIs.

Monkey And MonkeuRunner

The SDK provides two tools for functional-level app testing:

Monkey

This is a command-line tool that sends pseudo-random streams of keystrokes, touches, and gestures to a device. You run it with the Android Debug Bridge (adb) tool, and use it to stress-test your app, report back errors any that are encountered, or repeat a stream of events by running the tool multiple times with the same random number seed.

monkeyrunner

This tool is an API and execution environment for test programs written in Python. The API includes functions for connecting to a device, installing and uninstalling packages, taking screenshots, comparing two images, and running a test package against an app. Using the API, you can write a wide range of large, powerful, and complex tests. You run programs that use the API with the monkeyrunner command-line tool.

Chapter-5 CONCLUSION

5.1)CONCLUSION

Location based Services offer many advantages to the mobile users to retrieve the information about their current location and process that data to get more useful information near to their location. With the help of A-GPS in phones and through Web Services using GPRS, Location based Services can be implemented on Android based smart phones to provide these value-added services: advising clients of current traffic conditions, providing routing information, helping them find nearby hotels.

In this paper, we propose the implementation of Location based services through Google Web Services and Walk Score Transit APIs on Android Phones to give multiple services to the user based on their Location.

Location Based Services is an Android App , enabling travelers to plan and book the perfect trip. Location Based Services offers users an easy and simple interface helping them to find places. Upon using the app the user can simply select the city he wants to visit and nearby places.

On selecting the city the user will come across a list of places. These places are the most famous places in the that area. They can be historical monuments like forts, palaces etc.. The user can select any place of his personal liking and then can gather information about that place with the help of a short article or with the pictorial information provided at the interface.

The main purpose of this app is to make the travel experience of the user easy and simple by providing reliable and easy information to the user about the places near him.

In this project we have tried to implement all the above functions with the help of the android app.

5.2)FUTURE SCOPE

For future we can have the following features in the app which will be user friendly and will help the user to easily go to place he wants to visit.

- Add more cities
- Add more place in each city
- Ask for specific travel details

- Do booking of tickets
  for 1)Bus/Train/Plane
  2)Hotel
- Include Review for Hotels
- Include reviews for mode of transport
- Ask for user preferences on which more cities to add
- Ask for user preferences places to be added in the existing cities
- Log in feature for users
- Disount for regular customers

We want to build an app that simplifies the entire user expierence of a person who is visting a new cities by providing him complete details of the entire list of places he wants to visit and provide the user with a happy and memorable trip.
In the end we would like to publish oir app on the play store.

5.2.1) TECHNOLOGIES TO BE IMPLEMENTED IN FUTURE

1.WEB CRWALING

A Web crawler is an Internet bot which systematically browses the World Wide Web, typically for the purpose of Web indexing. A Web crawler may also be called a Web spider, an ant, an automatic indexer, or (in the FOAF software context) a Web scutter.

Web search engines and some other sites use Web crawling or spidering software to update their web content or indexes of others sites' web content. Web crawlers can copy all the pages they visit for later processing by a search engine which indexes the downloaded pages so the users can search much more efficiently.

Crawlers can validate hyperlinks and HTML code. They can also be used for web scraping.

Overview

A Web crawler starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies. If the crawler is performing archiving of websites it copies and saves the information as it goes. The archives are usually stored in such a way they can be viewed, read and navigated as they were on the live web, but are preserved as ‘snapshots'.

The large volume implies the crawler can only download a limited number of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change can imply the pages might have already been updated or even deleted.

The number of possible URLs crawled being generated by server-side software has also made it difficult for web crawlers to avoid retrieving duplicate content. Endless combinations of HTTP GET (URL-based) parameters exist, of which only a small selection will actually return unique content. For example, a simple online photo gallery may offer three options to users, as specified through HTTP GET parameters in the URL. If there exist four ways to sort images, three choices of thumbnail size, two file formats, and an option to disable user-provided content, then the same set of content can be accessed with 48 different URLs, all of which may be linked on the site. This mathematical combination creates a problem for crawlers, as they must sort through endless combinations of relatively minor scripted changes in order to retrieve unique content.

Crawling policy

The behavior of a Web crawler is the outcome of a combination of policies:

1)a selection policy which states the pages to download,

2)a re-visit policy which states when to check for changes to the pages, 3)a politeness policy that states how to avoid overloading Web sites, and

4)a parallelization policy that states how to coordinate distributed web crawlers.

References

[1] Manav Singhal and Anupam Shukla, "Implementation of Location based Services in Android using GPS and Web Services", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2, January 2012.

[2] Isha Sahu and Ishita Chakraborty, "Understanding Location Manager in Android and Implementing an Optimal Image Geotagging Application", (IJCTT) – volume 4 Issue 6–Month 2013.

[3] Alexander Troshkov, Kirill Kulakov, " Tourist Application for Mobile Platforms" , Petrozavodsk State University Petrozavodsk, Russia.

[4] Qusay H. Mahmoud ,"J2ME and Location based Services" - March 2004 http://developers.sun.com/mobility/apis/articles/location.

[5]ValerieBennett,"LocationBasedServices",http://www.ibm.com/developerworks/ibm/library/i-lbs.

[6]Shane Condor and Lauren Darcy ,"Android Wireless Application Development".

[7]Google Places API**,** http://code.google.com/apis/maps/documentation/places/

[9]Google Places API ,https://developers.google.com/places/android-api/start

[10]Material Desigin, https://design.google.com/

[11]Google Placespicker,https://developers.google.com/places/android-api/placepicker

[12]Google Places Autocomplete, https://developers.google.com/places/android-api/autocomplete