

SHOPPING RECOMMENDING SYSTEM

(ANDROID APPLICATION)

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

In

Computer Science and Engineering

By

Kunal Jandial (121245)

Under the supervision of

Ms. Nishtha Ahuja

To



Jaypee University of Information Technology

CERTIFICATE

Candidate's Declaration

I hereby declare that the work presented in this report entitled “**SHOPPING RECOMMENDING SYSTEM**” 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.

An authentic record of my own work carried out over a period from August 2015 to December 2015 under the supervision of Ms.Nishtha Ahuja(CSE Department)

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

(Student Signature)

Kunal Jandial

121245

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

(Supervisor Signature)

Nishtha Ahuja

Designation

Department name

Dated:

ACKNOWLEDGEMENT

It was a great chance for learning and professional development we had with JUIT for the help and Support.

We would like to express our deepest gratitude and special thanks to Ms.Nishtha Ahuja who in spite of being extraordinarily busy with her duties, took time out to hear, guide and keep us on the correct path and allowing us to carry out our project. We choose this moment to acknowledge her contribution gratefully.

We would like to express our special gratitude and thanks to JUIT for giving us such facilities and to the People who have willingly supported us with the Project, invoking their own efforts as well.

Contents

LIST OF FIGURES	vii
ABSTRACT.....	viii
CHAPTER- 1 INTRODUCTION.....	1
1.1 Introduction.....	1
Android Components	1
1.2 Background Statement	2
1.3 Problem Statement	3
1.4 Objective	3
1.5 Methodology	4
1.6 Organisation	5
Chapter -2 LITERATURE SURVEY	6
2.1 Android Architecture	6
Applications Layer.....	7
Application Framework	7
Libraries	7
Android Runtime Libraries	8
Linux Kernel	8
2.2 Main Components of Android Application	9
Activities	9
Broadcast Receivers.....	10
Services	10
Content Providers.....	11
Intents.....	11
fig 2.2 Activity Stack	13
2.1.6 Processes and Threads	13
Processes and Threads	14
2.1.7 Multi-Tasking	16
2.3 Review of Relevant Technologies.....	18
2.4 Related Research Fields	21

2.5 Operators Perspective	22
CHAPTER- 3 SYSTEM DEVELOPMENT.....	23
3.1. Introduction	23
3.2. The Existing System.....	23
3.2.1. Review of Existing System.....	23
3.2.2. Advantages of the Existing System	23
3.2.3. Limitations of the Existing System.....	23
3.3. The Proposed System	24
3.3.1. Review of the Proposed System	24
3.3.2. Advantages of the Proposed System.....	24
3.3.3. Limitations of the Proposed System	24
3.4. System Design.....	25
3.5. Modelling the System.....	25
3.5.1. UML (UNIFIED MODELLING LANGUAGE) MODELLING	25
3.7. System Analysis	31
3.7.1 Location acquisition.....	32
3.7.2 Location Discovery	33
3.7.3 Recommendation System.....	34
3.7.4 Recommendation using Location	35
3.7.5 System Overview	36
CHAPTER -4 SYSTEM IMPLEMENTATION	45
4.1. Introduction	45
4.2. Choice of Programming Language	45
4.2.1 Platform.....	45
4.2.2 Advantages of Java Technology	46
4.2.3 Android Software Development Kit(sdk).....	47
4.2.4 Eclipse.....	48
4.2.5 Google Places Api.....	49
4.2.6 Four Square Api.....	49
4.3. System Requirements	50
4.3.1) Hardware Requirements	50
4.3.2) Software Introduction	50
4.3.3) Software Requirements:	51
4.4 Testing Fundamentals	52
CHAPTER -5 CONCLUSION	58

5.1. Summary	58
5.2. Conclusion.....	58
5.3. Recommendations	58
5.4. Problems Encountered.....	58
5.5. Scope for Further Works	59
REFERENCES	60
SCREEN SHOT OF APPLICATION	62

LIST OF FIGURES

Figure 2.1 Android Architecture

Figure 2.2 Activity Stack

Figure 2.3 Activity on Background

Figure 2.4 Illustration Of Recommendation System

Figure 3.1 Use Case Diagram

Figure 3.2 Class Diagram

Figure 3.3 Component Diagram

Figure 3.4 Deployment Diagram

Figure 3.5 Flow Diagram

Figure 3.6 Comparision Of Placebo with Conventional Systems

Figure 3.7 System Architecture

Figure 3.8 Estimation Algorithm

Figure 3.9 Bayesian Estimation

Figure 3.10 List Of Frequently Visited Shops

Figure 3.11 Divide City Map into Areas

Figure 4.1 Java Platform

ABSTRACT

A recommender system which will tell you the nearby places to shop. And tells you about the offers available at certain stores.

Basically, user would be able to get List to nearby places to shop after which user can get details about various places in list and able to add details as per his/her preferences. This recommender system can facilitate people's travel not only near their living areas but also to a city that is new to them. This project is an attempt to provide the advantages of online shopping to customers of a real shop. It helps buying the products in the shop anywhere through internet by using an android device. This system can be implemented to any shop in the locality or to multinational branded shops having retail outlet chains.

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 manipulation, 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 the 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 Maker](#) allows 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 Background Statement

The advances in location-acquisition and wireless communication technologies enable people to add a location dimension to traditional social networks, fostering a bunch of location-based social networking services (or LBSNs) [1], e.g., Foursquare, Loopt, and GeoLife [2], where users can easily share life experiences in the physical world via mobile devices. Location as one of the most important components of user context implies extensive knowledge about an individual's interests and behavior, thereby providing us with opportunities to better understand users in a social structure according to not only online user behavior but also the user mobility and activities in the physical world. Under such a

circumstance, a location recommender system is a valuable but unique application in location-based social networking services, in terms of what a recommendation is and where a recommendation is to be made [3, 1]. Specifically, location recommendations provide a user with some venues (e.g., an electronic shop) that match her personal interests within a geospatial [1]. This application becomes more worthy when people travel to an unfamiliar area, where they have little knowledge about the neighborhoods. So we provide user with 6 options as follows: -Laptops, Phones, LED, Consoles, Clothing and Watch. Out of these user will provide information about shops available in each category.

1.3 Problem Statement

Inferring the rating for a location is very challenging using a user's location history in a LBSN. First, a user can only visit a limited number of physical locations. This results in a sparse user location matrix for most existing location recommendation systems, e.g., [7, 5], which directly play a collaborative filtering based model [4, 6] over physical locations. Nevertheless, a high quality location recommendation has to simultaneously consider the following three factors. 1) *User preferences*: the shoppingaholics would pay more attentions to nearby shopping malls [8]. 2) *The current location of a user*: As the users prefer the nearby locations, this location indicates the spatial range of the recommended venues and may affect the ratings of these recommendations [7]. 3) *The opinions of a location given by the other users*: Social opinions from the nearby users is a valuable resource for making a recommendation [5].

1.4. Objective

The literature on and implementation of shopping recommendation system is quite scattered. Different research papers that have been brought out on recommendation system may refer to the same type of institution but they mostly deal with different kinds of assignments, i.e., decisions like the number of items, assigning shops to items, or assigning events to locations. Moreover, each institution has its own characteristics which are reflected in the problem definition (Robertus, 2002). Yet, there have been no leveling grounds for developing a system that can work for most of these institutions.

The aim of this work is the generation of shopping recommendation system which provide user with list of items and based on location shops will be recommended according to price and

quality. Provide user with sufficient information about shops value so user can shop with his own choice.

The objective of the project is to make an application in android platform to purchase items in an existing shop. In order to build such an application complete web support need to be provided. A complete and efficient web application which can provide the online shopping experience is the basic objective of the project. The web application can be implemented in the form of an android application with web view.

1.5. Methodology

This research is concerned with the problem of constructing recommendation system. The central concept of the application is to allow the customer to shop virtually using the Internet and allow customers to buy the items and articles of their desire from the store. The information pertaining to the products are stores on an RDBMS at the server side (store).

The Server process the items and the shops are submitted by them. The application was designed into two modules first is for the customers who wish to see the articles. Second is for the storekeepers who maintains and updates the information pertaining to the articles and those of the customers. The end user of this product is a departmental store where the application is hosted on the web and the administrator maintains the database. The application which is deployed at the customer database, the details of the items are brought forward from the database for the customer view based on the selection through the menu and the database of all the products are updated at the end of each transaction. Data entry into the application can be done through various screens designed for various levels of users. Once the authorized personnel feed the relevant data into the system, several reports could be generated as per the security.

1.6. Organisation

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 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's OpenCORE; 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.

2.2 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 (a `ContentProvider` 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 to `Context.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 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.

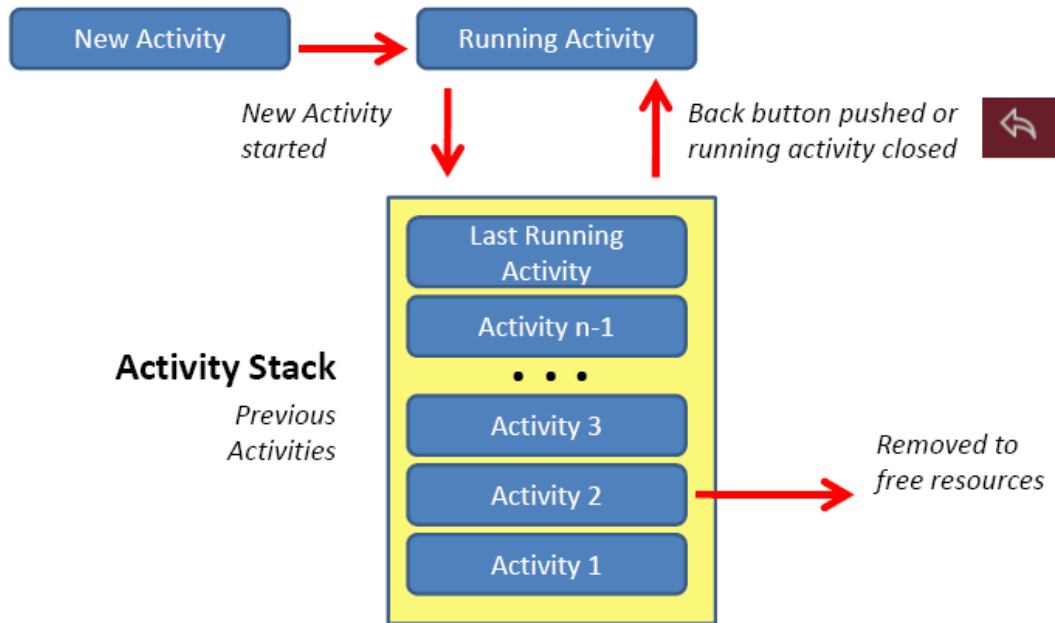


fig 2.2 Activity Stack

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.

Recommending Systems are not new there is lot of ubiquitous research but Recommending peripheral Shops based on user current location is a part of interest in advancement of technology . So to bring this system in android to assuage people all around the world by just carry out phone and get location of shops providing Quality items at Affordable price is main motive of the team. A location recommender system is a valuable but unique application in location-based social networking services, in terms of what a recommendation is and where a recommendation is to be made and that what need to be seen

2.3. Review of Relevant Technologies

We found that more than half of the recommendation approaches applied content-based filtering (55%). Collaborative filtering was applied by only 18% of the reviewed approaches, and graph-based recommendations by 16%. Other recommendation concepts included stereotyping, item-centric recommendations, and hybrid recommendations. The content-based filtering approaches mainly utilized papers that the users had authored, tagged, browsed, or downloaded. TF-IDF was the most frequently applied weighting scheme. In addition to simple terms, n-grams, topics, and citations were utilized to model users' information needs. Our review revealed some shortcomings of the current research. First, it remains unclear which recommendation concepts and approaches are the most promising. For instance, researchers reported different results on the performance of content-based and collaborative filtering. Sometimes content-based filtering performed better than collaborative filtering and sometimes it performed worse. We identified three potential reasons for the ambiguity of the results. A)

Several evaluations had limitations. They were based on strongly pruned datasets, few participants in user studies, or did not use appropriate baselines. B) Some authors provided little information about their algorithms, which makes it difficult to re-implement the approaches. Consequently, researchers use different implementations of the same recommendations approaches, which might lead to variations in the results. C) We speculated that minor variations in datasets, algorithms, or user populations inevitably lead to strong variations in the performance of the approaches. Hence, finding the most open-source frameworks.

We use the term "idea" to refer to a hypothesis about how recommendations could be effectively generated. To differentiate how specific the idea is, we distinguish between recommendation classes, approaches, algorithms, and implementations (Figure 4). We define a "recommendation class" as the least specific idea, namely a broad concept that broadly describes how recommendations might be given. For instance, the recommendation concepts collaborative filtering (CF) and content-based filtering (CBF) fundamentally differ in their underlying ideas: the underlying idea of CBF is that users are interested in items that are similar to items the users previously liked. In contrast, the idea of CF is that users like items that the users' peers liked. However, these ideas are rather vague and leave room for different approaches.

A "recommendation approach" is a model of how to bring a recommendation class into practice. For instance, the idea behind CF can be realized with user-based CF [221], content-boosted CF [222], and various other approaches [223]. These approaches are quite different but are each consistent with the central idea of CF. Nevertheless, these approaches to represent a concept are still vague and leave room for speculation on how recommendations are calculated.

A "recommendation algorithm" precisely specifies a recommendation approach. For instance, an algorithm of a CBF approach would specify whether terms were extracted from the title of a document or from the body of the text, and how terms are processed (e.g. stop-word removal or stemming) and weighted (e.g. TF-IDF). Algorithms are not necessarily complete. For instance, pseudo-code might contain only the most important information and ignore basics, such as weighting schemes. This means that for a particular recommendation approach there might be several algorithms.

Finally, the "implementation" is the actual source code of an algorithm that can be compiled and applied in a recommender system. It fully details how recommendations are generated and leaves no room for speculation. It is therefore the most specific idea about how recommendations might be generated

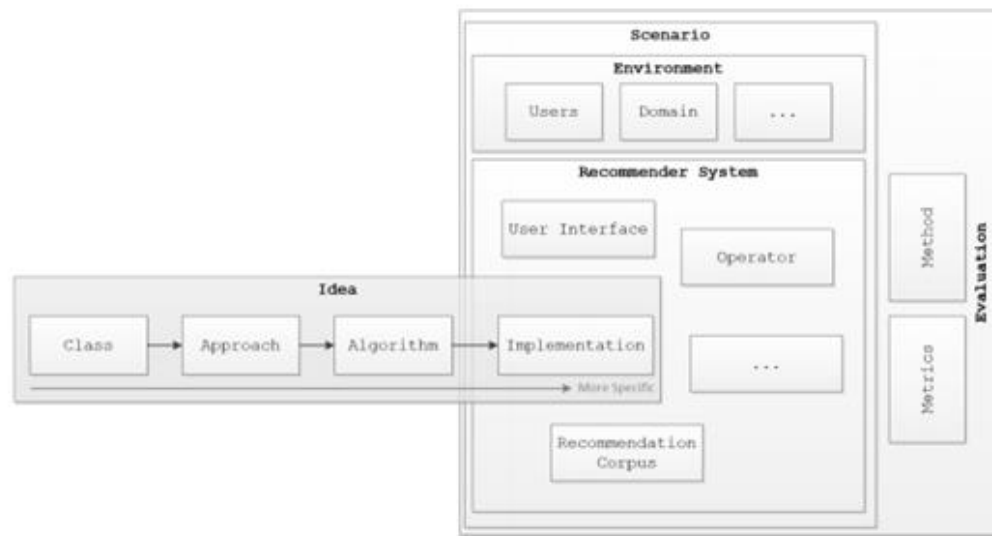


FIG 2.4 Illustration Of Recommendation System

A "recommender system" is a fully functional software system that applies at least one implementation to make recommendations. In addition, recommender systems feature several other components, such as a user interface, a corpus of recommendation candidates, and an operator that owns/runs the system. Some recommender systems also use two or more recommendation approaches: CiteULike, a service for discovering and managing scholarly references, lets their users choose between two approaches.

The "recommendation scenario" describes the entire setting of a recommender system, including the recommender system and the recommendation environment, i.e. the domain and user characteristics.

2.4 Related Research Fields

Several research fields are related to user modeling and (research-paper) recommender systems. Although we do not survey these fields, we introduce them so interested readers can broaden their research. Research on academic search engines deals with calculating relevance between research papers and search queries. The techniques are often similar to those used by research-paper recommender systems. In some cases, recommender systems and academic search engines are even identical. As described later, some recommender systems require their users to provide keywords that represent their interests. In these cases, research-paper recommender systems do not differ from academic search engines where users provide keywords to retrieve relevant papers. Consequently, these fields are highly related and most approaches for academic search engines are relevant for researchpaper recommender systems. The reviewer assignment problem targets using information-retrieval and informationfiltering techniques to automate the assignment of conference papers to reviewers. The differences from research-paper recommendations are minimal: in the reviewer assignment problem a relatively small number of paper submissions must be assigned to a small number of users, i.e. reviewers; research-paper recommender systems recommend a few papers out of a large corpus to a relatively large number of users. However, the techniques are usually identical. The reviewer assignment problem was first addressed by Dumais and Nielson in 1992 six years before Giles et al. introduced the first research-paper recommender system. A good survey on the reviewer assignment problem was published by Wang et al. [Scientometrics deals with analyzing the impact of researchers, research articles and the

links between them. Scientometrics researchers use several techniques to calculate document relatedness or to rank a collection of articles. Some of the measures – h-index, co-citation strength and bibliographic coupling strength– have also been applied by research-paper recommender systems. However, there are many more metrics in scientometrics that might be relevant for research-paper recommender systems.

User modeling evolved from the field of Human Computer Interaction. One thing user modeling focuses on is reducing users' information overload making use of users' current tasks and backgrounds. User modeling shares this goal with recommender systems, and papers published at the major conferences in both fields (UMAP15 and RecSys16) often overlap. User modeling is a central component of recommender systems because modeling the users' information needs is crucial for providing useful recommendations. For some comprehensive

surveys about user modeling in the context of web personalization, refer to . Other related research fields include book recommender systems educational recommender systems academic alerting services expert search automatic summarization of academic articles academic news feed recommenders academic event recommenders venue recommendations citation recommenders for patents recommenders for academic datasets and plagiarism detection. Plagiarism detection, like many research-paper recommenders, uses text and citation analysis to identify similar

documents Additionally, research relating to crawling the web and analyzing academic articles can be useful for building research-paper recommender systems, for instance, author name extraction and disambiguation title extraction or citation extraction and matching Finally, most of the research on content-based or collaborative filtering from other domains, such as movies or news can also be relevant for research-paper recommender systems.

2.5 Operators Perspective

It is commonly assumed that the objective of a recommender system is to make users "happy" by satisfying their needs . However, there is another important stakeholder who is often ignored: the operator of a recommender system . It is often assumed that operators of recommender systems are satisfied when their users are satisfied, but this is not always the case. Operators may also want to keep down costs of labor, disk storage, memory, computing power, and data transfer . Therefore, for operators, an effective recommender system may be one that can be developed, operated, and maintained at a low cost. Operators may also want to generate a profit from the recommender system . Such operators might prefer to recommend items with higher profit margins, even if user satisfaction is not optimal. For instance, publishers might be more interested in recommending papers the user must pay for than papers the user can freely download.

The operator's perspective has been widely ignored in the reviewed articles. Costs of building a recommender system, or implementing an approach were not reported in any article. Costs to run a recommender system were reported by Jack from Mendeley . He stated that the costs on Amazon's S3 were \$66 a month plus \$30 to update the recommender system that served 20 requests per second generated by 2 million users.

CHAPTER- 3 SYSTEM DEVELOPMENT

3.1. Introduction

System Analysis is the study of a business problem domain to recommend improvements and specify the business requirements and priorities for the solution. It involves the analyzing and understanding a problem, then identifying alternative solutions, choosing the best course of action and then designing the chosen solution.

It involves determining how existing systems work and the problems associated with existing systems. It is worthy to note that before a new system can be designed, it is necessary to study the system that is to be improved upon or replaced, if there is any.

3.2. The Existing System

3.2.1. Review of Existing System

The current system for shopping is to visit the shop manually and from the available product choose the item customer want and buying the item by payment of the price of the item. It is traditional and time-consuming method without the aid of digital application.

3.2.2. Advantages of the Existing System

The timetable generation process by the education center staff is:

- Subjective and can be made better through collaboration with the different entities involved.

3.2.3. Limitations of the Existing System

- Repeated time allocations may be made for a particular course thereby leading to data redundancy.
- A lot of administrative error may occur as a result of confusing time requirements.

- User must go to shop and select products.
- It is difficult to identify the required product.
- Description of the product limited.
- It is a time consuming process
- Not in reach of distant users.
- It is not flexible as changes may not be easily made.

3.3. The Proposed System

3.3.1. Review of the Proposed System

The proposed systems was developed to solve the problem people face everyday while shopping electronic items,gadgets,accessories etc.With this system shops will be recommended based on user location .Location will be traced and shops in user periphery will be shown with price of chosen item which aid the purchase of item.

3.3.2. Advantages of the Proposed System

The timetable generation process by the education center staff is:

- Unlike the manual shopping, the system offers flexibility.
- It utilizes minimal processing/computing power.
- It greatly reduces the time needed to generate shop.
- It provides an easy means for recommending items through an intuitive interface.
- It increases productivity.

3.3.3. Limitations of the Proposed System

The following are the challenges in the system due to time constraints:

- The proposed system can only generate recommendation based on a few hard course constraints.
- The proposed system can only generate recommendation based on price.
- Recommendations generated by this system is still subject to revision by end user.
- Not all of the algorithm principles are implemented in the system.

3.4. System Design

System design is the specification or construction of a technical, computer-based solution for the business requirements identified in a system analysis. It gives the overall plan or model of a system consisting of all specifications that give the system its form and structure i.e. the structural implementation of the system analysis.

3.5. Modelling the System

Modeling a system is the process of abstracting and organizing significant features of how the system would look like. Modeling is the designing of the software applications before coding. Unified Modeling Language (UML) tools were used in modeling this system.

3.5.1. UML (UNIFIED MODELLING LANGUAGE) MODELLING

The Unified modeling language is an object-oriented system notation that provides a set of modeling conventions that is used to specify or describe a software system in terms of objects. The Unified Modeling Language (UML) has become an object modeling standard and adds a variety of techniques to the field of systems analysis and development hence its choice for this project.

UML offers ten different diagrams to model a system. These diagrams are listed below:

- Use case diagram
- Class diagram
- Object diagram
- Sequence diagram
- Collaboration diagram
- State diagram
- Activity diagram
- Component diagram
- Deployment diagram
- Package Diagram

In this project, the Use case diagram and Class diagram will be used for system modeling.

3.5.1.1 Use Case Diagram

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis of use case diagrams is on what a system does rather than how. They are used to show the interactions between users of the system and the system. A use case represents the several users called actors and the different ways in which they interact with the system.

ACTORS

- User
- Shopping Recommending System

USE CASES

- Choose The Product
- Get The Location
- Search Nearby Shops
- Fetch The Shops
- Show The Shops Along With Offers
- Recommend Shop

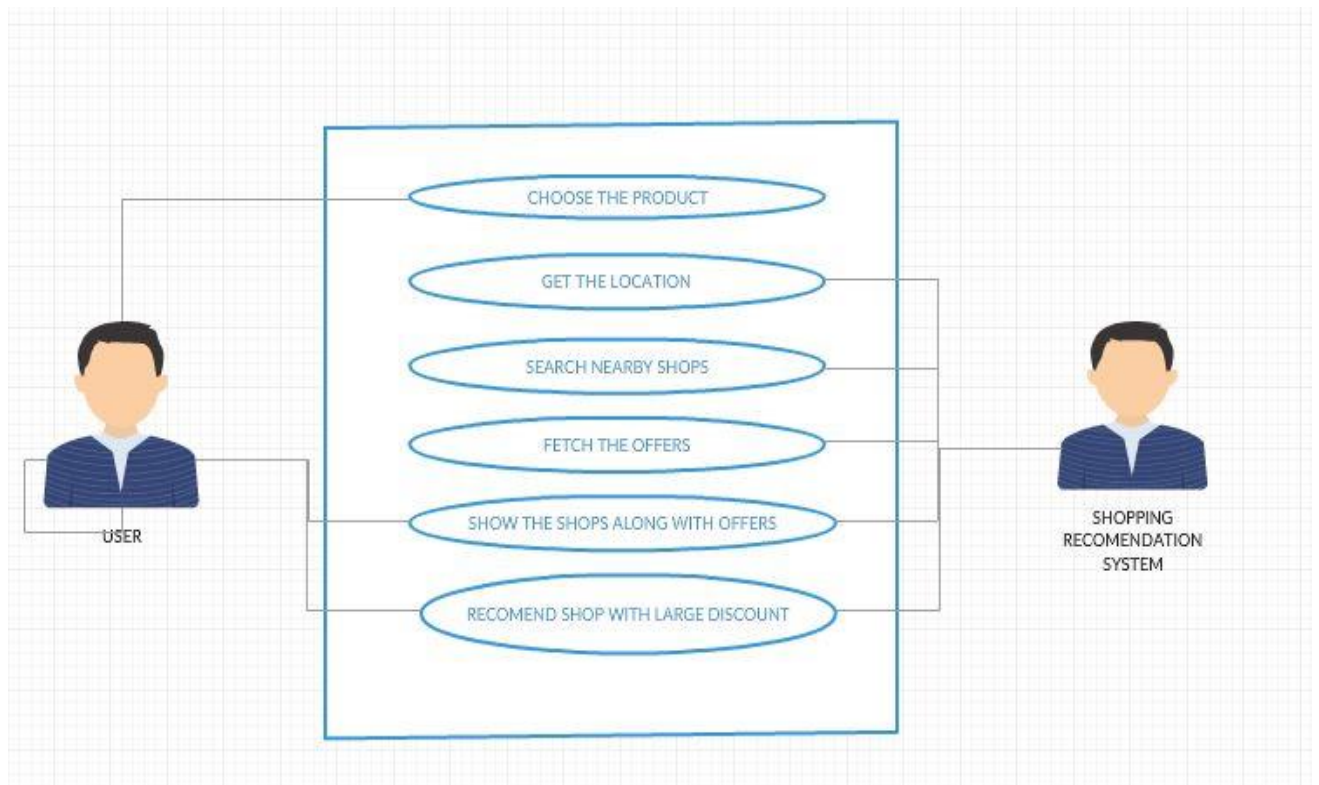


Figure 3.1.: Use Case Diagram to show the interaction between the system and user

3.5.1.2. Class Diagram

A class diagram is an organization of related objects. It gives an overview of a system by showing its classes and the relationships among them. Class diagrams only display what interacts but not what happens during the interaction hence they are static diagrams.

CLASSES

- Item
- Laptop
- Clothes
- Mobile
- Console
- Watches
- TV
- Shops
- User

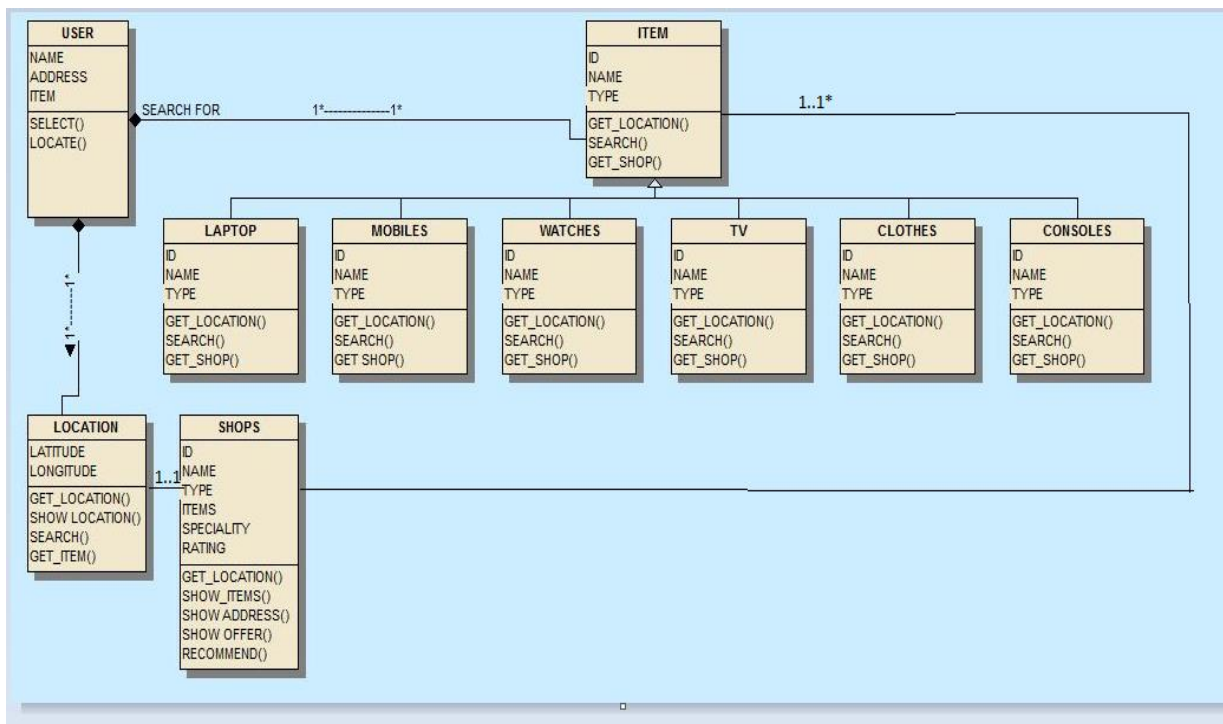


Figure 3.2.: Class Diagram to show the relationships between the different classes associated with the system

3.5.1.3. Component Diagram

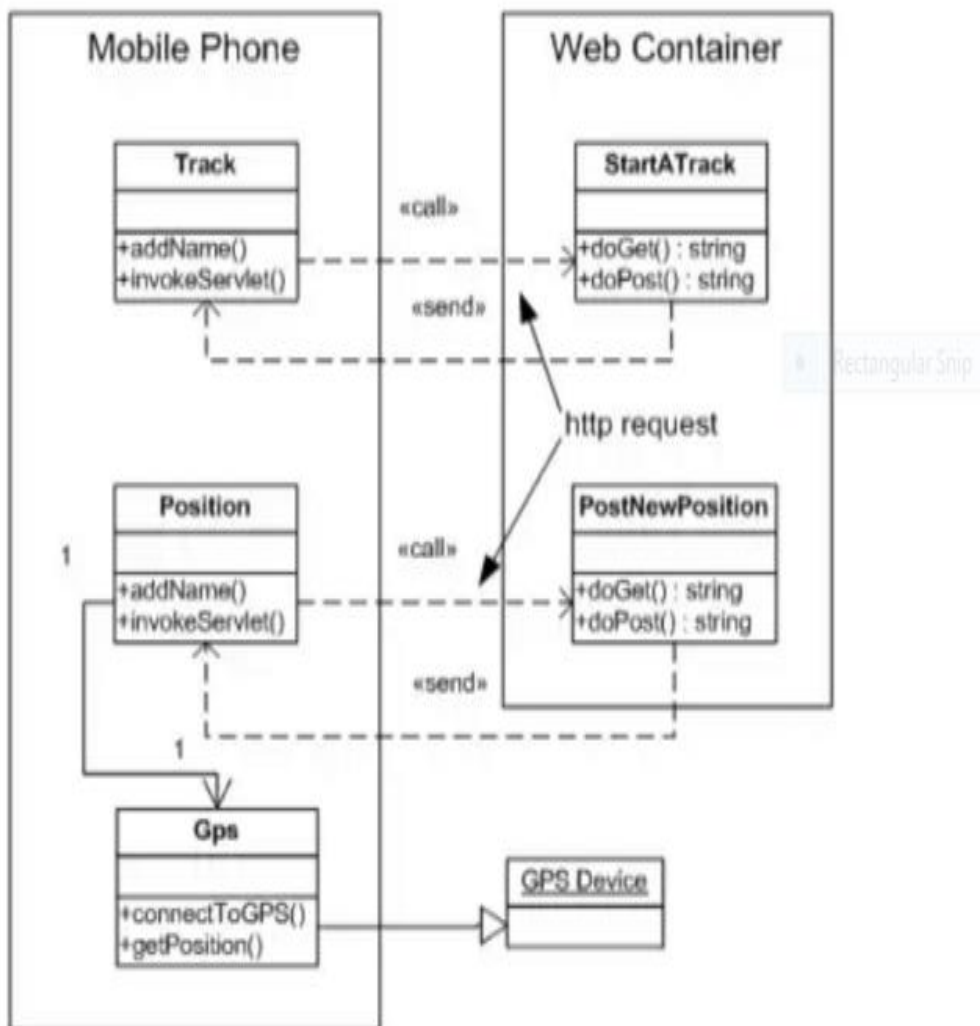


Fig 3.3

3.5.1.4. Deployment Diagram

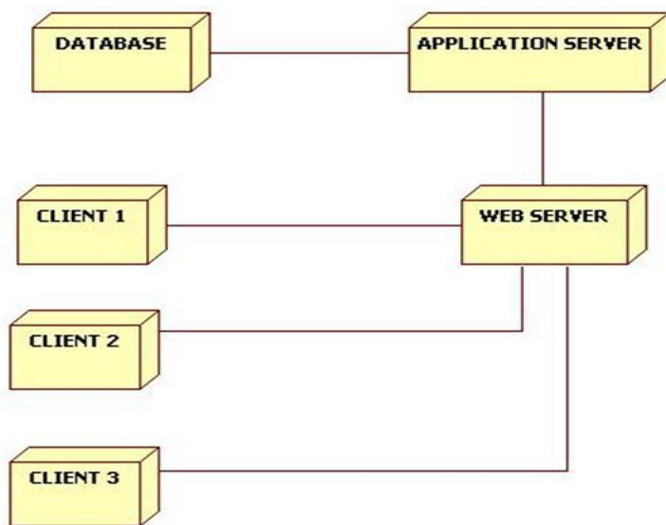


Fig 2.4

3.5.1.5. Flow Diagram

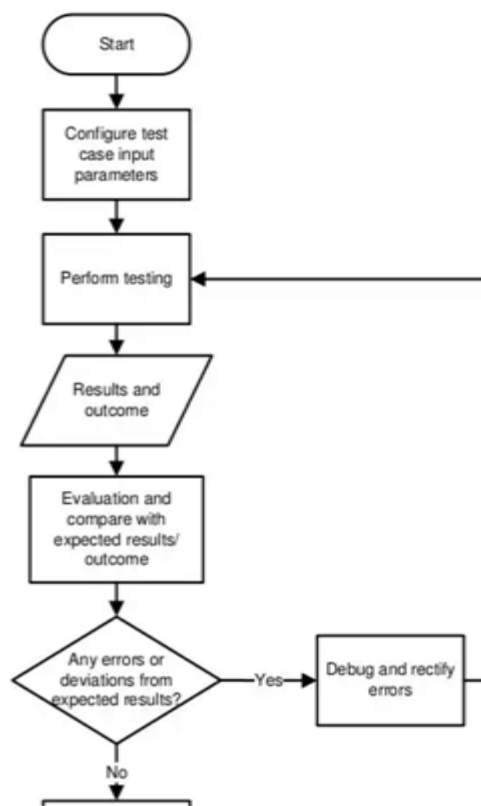


Fig 2.5

3.7. System Analysis

TODAY, recommendation systems are used in many online shopping sites. Their chief function is to recommend items that match each customer's preferences and needs, which are estimated from information gathered through their past activities at the site, for example which items they bought or which items they showed interest in.

Recommendation systems have numerous benefits. Studies in experimental psychology say that, when used effectively, recommendation systems in shopping sites have the ability to increase the perceived credibility of the site[9], thus leading customers into buying more items. It is this ability of generating increased profit that has brought about the current popularity of recommendation systems among shopping sites. When seen from a customer's point of view, recommendation systems are helpful in the sense that they assist them to easily find items that match their tastes. They are perceived as an intelligent tool that effectively helps them out as consumers in this age of information overload.

But despite the benefits that recommendation systems grant to its customers and shopping site owners, we believe that a single fact is severely limiting us from appreciating them to their full potential: the single fact, that recommendation systems can only be used for shopping on the Internet, not for shopping in the city, or in other words, *in the real world*.

Despite the growing popularity of online shopping, the majority of shopping activities is still done at real-world shops. Thus it can be inferred that, for us to be able to fully appreciate the benefits of recommendation systems, we must apply them to real-world shopping. The difficulty of this task lies in that it is extremely more difficult to acquire sufficient customer activities needed for estimating preferences in real-world shopping, compared to online shopping where all user activities can be easily recorded in the server. In this paper, we introduce Placebo, a real-world recommendation system designed to run on mobile devices, which recommends shops to users based on preferences estimated from their past location history. Location data can be easily acquired using means such as GPS, and it contains rich information about each user's personal preferences. Our system effectively applies location data to the widely used item-based collaborative filtering algorithm, by transforming continuously recorded location data into a form of a list that contains each user's {¥it frequently

visited shops}, and rating values which indicate how fond the user is of each shop. This list can be directly used as input to the filtering algorithm to make recommendations in the exact same manner as conventional recommendation systems. We have devised a custom algorithm for this transformation of data, which automatically finds each user's frequently visited shops and calculates rating values without any need of explicit user manipulation. We have also enabled the system to take into account information such as the user's usual shopping routes, and the ease of access from the user's current location to each shop, to provide more timely recommendations.

To assess the effectiveness of our system, we have conducted an evaluation test at Daikanyama, one of Tokyo's most revered shopping districts. The results show great promise in the system's ability to make accurate recommendations, although various aspects of the system still needs polish.

3.7.1 Location acquisition

One important characteristic of our system is that it makes extensive use of location data. Numerous studies on location-based systems have been previously conducted, with various methods of location acquisition.

The Olivetti Active Badge[10] and the ParcTab[11] acquire user location using infrared waves. Infrared, being cheap, convenient and unregulated, has long been a popular choice for location acquisition, although it has some disadvantages such as limited accuracy and poor performance in environments abound with obstacles.

RADAR[12][13] detects location using wireless LAN (WLAN). WLAN is increasingly becoming popular as the method of choice for location-aware systems, since in most cases no special equipments are needed, as many recent PCs and mobile devices are already equipped with built-in WLAN capabilities, and the number of hotspots in urban areas is rapidly increasing. RADAR acquires user location from the strengths of signals observed by several WLAN base stations, by using a predefined map consisting of observed signal strengths for sample locations spreading throughout the environment.

The Active Bat[14] system uses ultrasound waves to acquire extremely precise location. The system consists of a small device (a Bat) with ultrasound-emitting capabilities, and a dense array of receivers mounted on ceilings. The relatively slow speed of ultrasound allows the

system to correctly measure the distance between each receiver and the Bat, and the precise location of the user can be calculated by triangulation with an accuracy of around 3 centimeters.

The CyberGuide[15] is an example of a system using GPS. The advantages of using GPS are that no equipment other than a GPS receiver is needed, and that fairly high accuracy can be achieved, if only outdoors.

3.7.2 Location Discovery

Our system, **Placebo**, makes recommendations based on information about each user's frequently visited shops, which are automatically estimated by the system. The task of finding frequented places from GPS input (*location discovery*) has been investigated in a number of past researches.

Commotion[16] is one example of a location discovery system. The system tracks users' locations using GPS, and identifies frequently visited locations (buildings), by keeping track of positions where GPS signals were continuously lost. Places where signals were often lost are defined as frequently visited locations. Then, the user can annotate the defined locations with information such as location names, memos, and to-do lists.

Ashbrook and Starner[17], and Zhou et al[18], proposes location discovery methods based on clustering algorithms, which offer more precise results but are more computationally expensive.

The system we propose in this paper finds users' frequently visited buildings in a somewhat similar manner as the above systems, but instead of using a clustering approach, we introduce a completely new algorithm for location discovery, which is much more computationally efficient. This is possible because in our system we only need to search for frequently visited *shops*, which are located in discrete locations, as opposed to conventional location discovery systems which search for frequented *places*, which exist continuously in a two-dimensional space. The discrete nature of shops allows our system to look for frequented places in a much smaller search space, and therefore the computation cost can be reduced greatly.

3.7.3 Recommendation System

The concept of recommendation systems have long been explored, and there have already been numerous researches conducted on the field. At the core of recommendation systems is the filtering algorithm, which filters out unnecessary data and decides which data should be recommended to the user. The two most commonly used types of filtering algorithms are *content-based filtering*, and *collaborative filtering*.

1) *Content-based Filtering*: The basic idea of content-based filtering[19][20] is to express the content of each data in a form that can be objectively evaluated, and filter out data whose content doesn't match the user's preferences. The most commonly used method for expressing content is the feature vector method.

According to the feature vector method, the content of each piece of data is expressed in the form of a vector, consisting of values for a set of *features*. Features are defined so that they can effectively convey the content of each data, and that they can be expressed in numerical values. For example, in the case of text data, features are defined as the frequency with which several keywords appear in the text. If the keywords are cleverly chosen, the resulting vector should be able to communicate the content of the text with significant accuracy. The preferences of each user is also expressed as a vector using the same set of features, and if a vector for a piece of data is similar to the content of the text with significant accuracy. The preferences of each user is also expressed as a vector using the same set of features, and if a vector for a piece of data is similar to the vector for the user preference, there should be good chance that the user will like the data, and thus the data is recommended to the user. Most content-based systems are intended only for recommending text data, since appropriately expressing the content of each data is difficult for other types of data.

2) *Collaborative Filtering*: Collaborative filtering[21][22] recommends data that was given high ratings by a number of users, with similar preferences as the user who requested the recommendation.

The first step of collaborative filtering is calculating the similarity of preferences between users. Then, several users with the highest similarity values (*nearest neighbors*) are picked out. Data that has received high ratings among the nearest neighbors, and that the user who requested the recommendation has not yet evaluated, is recommended.

The biggest advantage of collaborative filtering over content-based filtering is that collaborative filtering requires no previous knowledge about the content of the data, and thus can be applied to any type of data, regardless of content.

On the other hand, collaborative filtering also has some disadvantages. First, the only data that can be recommended using collaborative filtering are ones that have already been evaluated by some other user, which means that it may take some time before a piece of data newly introduced in the data space can have a chance of being recommended. When the relative size of the data space is extremely large compared to the number of users, a considerable portion of the data space will not be available for recommendation. Next, collaborative filtering only functions properly when there are users with similar preferences. When the number of users is small, the nearest neighbors found by the system might not necessarily have similar preferences, which may result in the system producing inaccurate recommendations. Finally, the computation cost of collaborative filtering can be a problem, especially when the number of users is large.

There is a variation of collaborative filtering called item-based collaborative filtering[23]. In this approach, instead of calculating the similarity between users, the similarity between items are calculated. Items which show high similarity with the items that the user has given high ratings are recommended.

3.7.4 Recommendation using Location

There have been numerous studies of recommendation systems and shopping assistance systems which make use of location data, but so far they seem to be limited to using only the *current* user location. The personal shopping assistant[24] by Asthana et al. presents users with information on special deals according to their current location. Pilgrim[25] is a website recommendation system which takes into account the location from which the user had accessed websites. To the best of our knowledge, our proposed system is the only

recommendation system which uses the *history* of continuously recorded location data for recommendation.

3.7.5 System Overview

The basic idea of our recommendation system, **Placebo**, is to estimate the users' individual preferences from the history of their location data collected using GPS, and recommend shops upon request. The system is intended to be useful for various kinds of shoppers, in various situations. For example, the system can be helpful for shoppers new to the area wanting to find shops that match their tastes, or for shoppers more familiar to the area willing to try something new.

Fig. 2 illustrates how **Placebo** compares with conventional recommendation systems for online shopping. Whereas conventional systems estimate users' preferences from their online activity records, such as items bought or



Fig. 3.6. Comparison of **Placebo**(right) with conventional recommendation systems(left)

checked in the past, our system estimates preferences based on their location history during shopping in the city.

It should be noted that our recommendation system recommends shops, as opposed to conventional systems in shopping sites which recommends items. We dismissed the idea of recommending items, for the two reasons discussed below.

First, in order to recommend items, information about each specific item that the user has bought or has showed interest in must be obtained, to estimate preferences. In online shopping, these information can be easily acquired from the server log. But in the real-world, these information can only be acquired if every item is equipped with a smart tag like an RFID, or every shop employs a strict customer surveillance system. The latter method is obviously unrealistic, due to privacy concerns. The former method seems more plausible, since smart tags are assumed to become pervasive and replace bar codes in the coming years. However, if we are to base our recommendation system on smart tags, the issue of *coverage rate* must be considered. If every item in every shop becomes equipped with a smart tag, the coverage rate will be 100%. But considering the current coverage rate of bar codes, assuming that the actual rate will become anything close to this is unreasonable. Therefore, a recommendation system based on smart tags will inevitably be a crippled one, since it automatically excludes a considerable portion of items sold in the city. In contrast, our system can include every shop except those in places where GPS does not work, for example inside large building, and the coverage rate is relatively high. The rate should become even higher in the near future, with the advance of alternative location acquisition techniques like the Wi-Fi.

Next, there is no effective filtering algorithm applicable for recommending real-world items. Content-based filtering is almost exclusively used for recommending text data, which is rarely seen in real-world shopping. Collaborative filtering is incapable of dealing with the extremely fast cycle in which new items appear and disappear in the real world, since it cannot recommend items that had not yet been evaluated by a respectable number of users. The algorithm can only work for items like books which are continuously sold for a significant time span, or items that are produced in large quantities. In case of items which are produced in scarce numbers and sold for only a short period of time, for example fashion items made by lesser known producers, sufficient evaluation results needed for collaborative filtering to properly function cannot be achieved. In contrast, our system is unaffected by this fact because shops exist for a relatively long period of time, enough for gathering evaluations.

Placebo is solely based on location data acquired using GPS, and does not require any other sensors. Considering the growing popularity of GPS receivers and GPS-embedded mobile phones, a system that can function using only GPS has a chance of being widely used, and developing such a system should be a worthy attempt.

Fig. 3 illustrates the system architecture of Placebo. The main components of our system are client devices carried by users, and a server that performs recommendations. The client device

can be any mobile computer device which is or can be equipped with Internet and GPS capabilities. Potential client platforms include PDAs, notebook PCs and mobile phones. Our system analyzes, and transforms raw location data received from GPS into a list of each user's frequently visited shops. This list is used as input to our filtering algorithm, and so it must be kept inside the server in order to carry out recommendations. If the client device has enough memory space and high computational capabilities, the transformation of location data can be done inside the client. But if the capabilities of the client device is insufficient, the transformation must be done inside the server, and thus the users' raw location data has to be sent to the server. This results in decreased privacy, and should be avoided if possible.

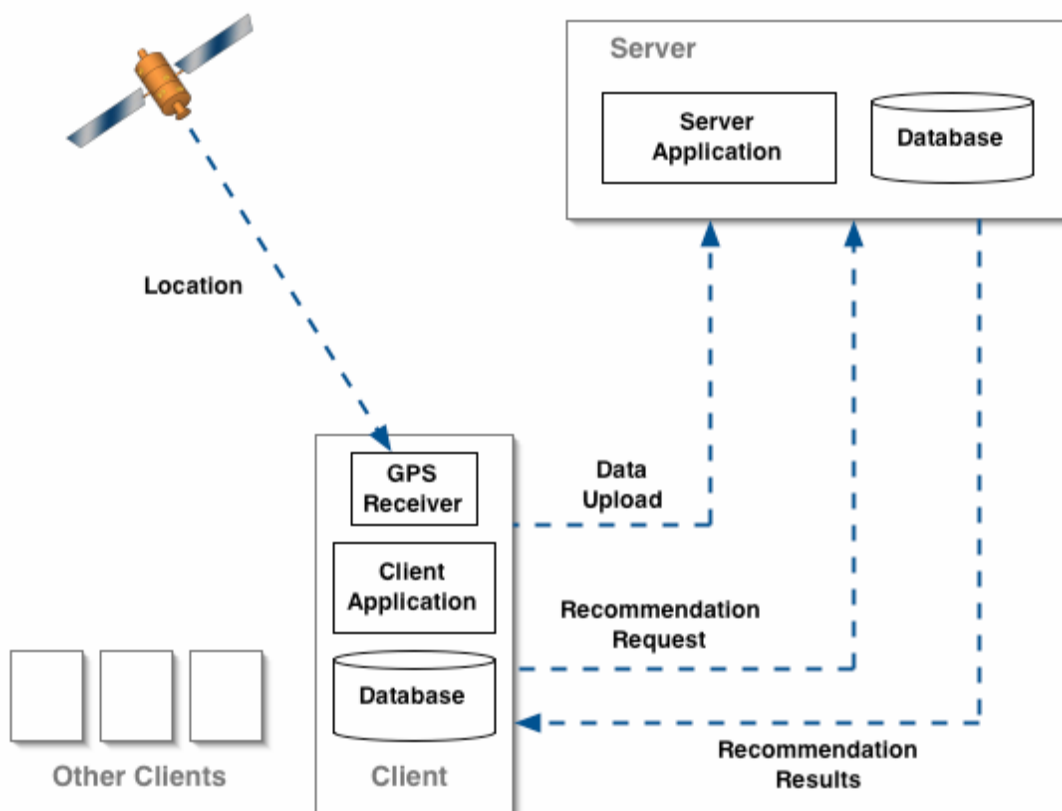


Fig. 3. 7 The Placebo system architecture

Below, we describe the process by which the system makes recommendations. The recommendation process can be divided into two phases, the data acquisition phase and the recommendation phase.

A. Data Acquisition Phase

In the data acquisition phase, raw location data from GPS is reconstructed into a list of each user's frequently visited shops. The process can be further divided into three sub-phases: monitoring user location, detecting visits to shops, and finding frequently visited shops.

1) *Monitoring user location:* The location of the user is consistently monitored using GPS. Data acquired by GPS contains errors deriving from a variety of causes. Even in ideal conditions where no tall buildings are present, an error of around 10 meters will inevitably exist, mainly due to the effects of the ionosphere. Also, since GPS signals cannot penetrate through building walls, the system cannot acquire the user's location when he/she is indoors. The location of the user is periodically recorded into a database, inside the client device or the server if the client has insufficient memory. This information is used later to identify the user's usual shopping routes through the city.

2) *Detecting visits to shops:* We exploit the fact that GPS signals cannot penetrate through walls, to detect if the user is indoors or outdoors. But naively using the loss of GPS signals as a proof that the user is inside leads to frequent errors. There are two possible user situations when GPS signals cannot be received: either the user is inside a building, or is surrounded by tall buildings and signals are blocked. On the other hand, simply believing that the user is outdoors because of the availability of GPS signals also leads to errors, since the user may either be outdoors, or inside a building that allows GPS signals to penetrate through its walls due to its structure and building materials (buildings that have high ceilings and walls consisting mainly of glass often fit into this category).

To reduce these errors, we incorporate two timers for indoor and outdoor detection (Fig. 4) The indoor judgment timer (IJT) is initially set at zero, and starts counting up at the moment when GPS signals are lost. IJT keeps counting up as long as GPS signals are continuously lost, and returns to zero every time when signals become available again, if even for a moment. When IJT reaches a predefined time limit, the system judges the user as indoors. The time limit is decided according to the GPS-friendliness of the area: in rural areas it should be set to a low number, and in urban areas it should be a high number, because of the commonness of blocked signals. The outdoor judgment timer (OJT) works in almost the same manner as the IJT. When GPS signals become available even for an instant, the OJT starts counting up from zero. OJT keeps counting up as long as GPS signals are available, and returns to zero when signals are blocked. When OJT reaches the time limit, the system judges the user as outdoors. The time limit, like in the case of IJT, is predefined according to the area. In rural areas they are set high,

and in urban areas they are set low, because there can be frequent signal blocks even when the user is outside.

The system determines that the user has visited a building when the user is first judged as indoors, then as outdoors. Then, the system records the current user location (latitude, longitude) and the approximate duration of the visit.

3) *Finding frequently visited shops*: From the record of users' visits to shops, we can reveal the presence of frequently visited shops, by searching for clusters of recorded visits. But since the locations of the recorded visits contain errors, the exact shops that the user is frequently visiting cannot be directly determined. Here, we propose a technique which automatically finds out the exact shops which the user is frequently visiting using an estimation algorithm. The estimation algorithm is based on

***t*-test.**

Whenever a new visit to a shop is detected, the system picks up shops that are located within a predefined radius from the location at which the visit took place. Then, for each of those shops, the system searches for past visits within a certain distance from the shop (*sample visits*). From the sample visits, the system evaluates if it is plausible that the shop is a frequently visited shop, by applying two tailed *t*-test to the sample visits. A more detailed explanation of this method is as follows. First, we assume that the observed latitude and longitude values of visits to a certain shop follow a normal distribution, with the actual location of the shop as the mean. Given this assumption, the latitude and longitude values of the sample visits around a shop will follow *t*-distributions. Then, we use *t*-test to evaluate if the means of the *t*-distributions can be regarded as *statistically identical* to the actual location of the shop. If the shop is a frequently visited shop, the two locations should be statistically identical. Fig. 5 illustrates the procedure of this method.

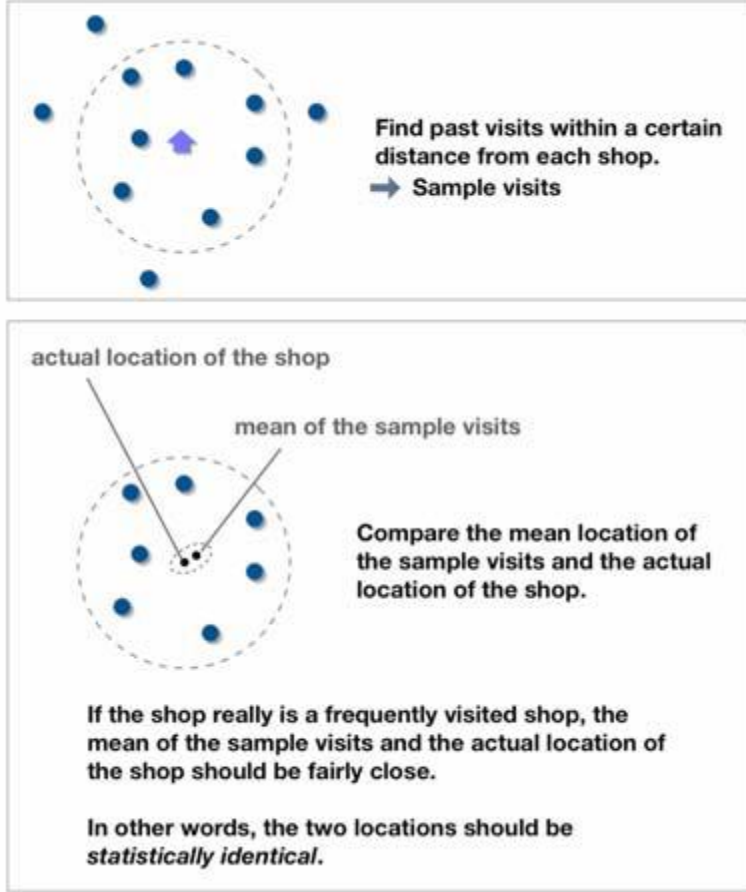


Fig. 3.8 . Estimation Algorithm (t -test)

If, as the result of the t -test, the two locations can be considered as statistically identical, the shop is judged as a user's frequently visited shop. The shops is included in the user's frequently visited shops list, with a rating value calculated from the number of visits and the average duration of those visits.

If a large number of sample visits can be expected to be acquired, we can use Bayesian estimation instead of t -test. In this case, the *plausibility* that the shop is a frequently visited shop, is calculated using the following equation.

$$P = A \int p(\mu_x, \mu_y) (1)$$

$p(\mu_x, \mu_y)$ is a probability density function calculated using Bayesian estimation. It indicates the probability density with which the shop that caused the sample visits is located at (μ_x, μ_y) . is a small area centered around the actual shop location. Simply put, AP is the estimated probability that the location of the shop at which the sample visits took place is located inside A . If the plausibility P is above a threshold value, the shop is judged as the user's frequently visited shop. The shops are added to the user's frequently visited shops list, with rating values

calculated from the plausibility P and the average duration of the visits. Fig. 6 illustrates this method. We have described two methods for finding frequently visited shops, one based on t -test, and the other based on Bayesian estimation. Whichever method we use, we end up with a list of frequently visited shops and rating values (Fig. 7). This list is stored in the server database and is used for recommendation.

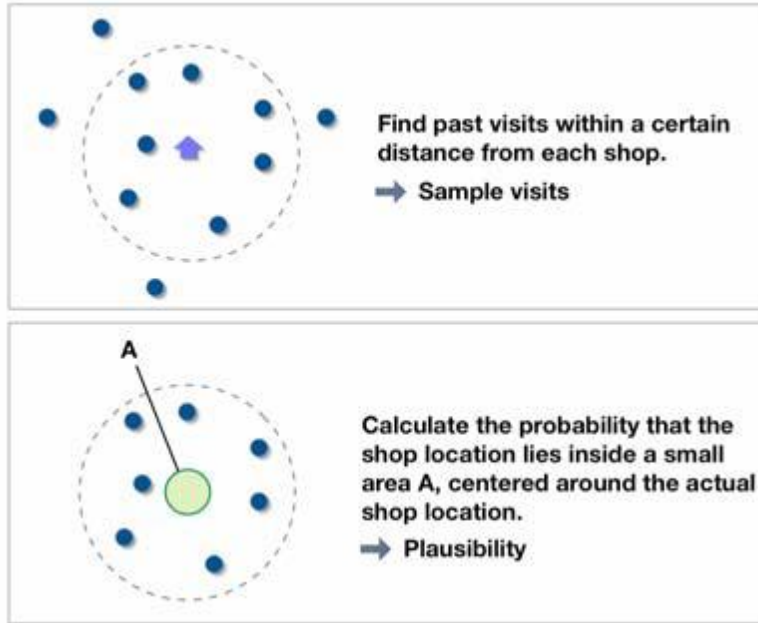


Fig. 3.9. Estimation Algorithm (Bayesian estimation).

with rating values calculated from the plausibility P and the average duration of the visits. Fig. 6 illustrates this method.

We have described two methods for finding frequently visited shops, one based on t -test, and the other based on Bayesian estimation. Whichever method we use, we end up with a list of frequently visited shops and rating values (Fig. 7). This list is stored in the server database and is used for recommendation.

B. Recommendation Phase

Upon user request, the server recommends shops using the list obtained in the data acquisition phase. Recommendation is done in two steps: filtering, and adding weights according to areas. 1) Filtering: As the filtering algorithm, we use the item-based collaborative filtering algorithm. The similarity between two shops A and B is calculated using the following equation.

$$Sim(A,B)=\frac{R_{u,A}R_{u,B}}{\sum R_{u,A}^2\sum R_{u,B}^2} \quad (2)$$

Here, $R_{u,A}$ indicates the rating value for shop by user, and $AuR_{u,B}$ indicates the rating value for shop B by user. The similarity increases when there is an observed tendency that users who frequently visit shop also frequently visit shop uAB . Since item-based collaborative filtering does not take into account the content of the data, correlations between shops of different categories, such as cafes and clothing stores, can be defined. The system picks out several shops which have high similarity with the user's frequently visited shops. These shops are regarded as having high chances of matching the user's preferences well.

2) *Adding weights according to areas*: Our system is intended to be used in the city, which means that there will be physical distances between users and recommended shops. Compared to online shopping where users can check recommended items with one click of a mouse, our system requires users to overcome these distances before they can appreciate the results of the recommendations. In cities like Tokyo where many people shop on foot, the distances can easily become too demanding for users. Therefore, we must make sure that the recommended shops are relatively easily accessible from the users.

To meet this requirement, we first divide the city into *areas*, and model users' movements using Markov models, with areas as nodes. The shops picked out by the filtering algorithm are added with weight values according to the areas in which they are located. Shops with higher weight values are given increased chances of being recommended to the user. Shops located in the current area of the user, or in areas where the user is likely to advance next are added large weight values, and thus will more likely to be recommended. Below we explain this procedure in detail.

Frequently visited shops

Name	Rating
Shop A	10
Shop B	4
Shop C	6
Shop D	2
Shop E	9
Shop F	7
.	.
.	.
.	.

Fig. 3.10. List of frequently visited shops

1. **Dividing the city map into areas**-Areas must be defined so that any two points located in the same area are easily accessible from one to the other. Our algorithm for this task, based on cluster analysis, is as follows:

1.	Divide the city map using a square grid. The grid should be fairly dense.
2.	Pick the grid elements which are located on places that can be walked through, such as on streets, and define them as initial clusters.
3.	For all combinations of two clusters, do{
4.	For all combinations of two grid elements in the cluster, do{
5.	Calculate the distance between the two grid elements using Dijkstra's algorithm. Keep a record of the calculated distances.
6.	}
7.	Look for the two grid elements that give the largest distance, and define their distance as the accessibility of the combination of clusters.
8.	}
9.	Merge the two clusters that combine to make the smallest accessibility.
10.	Repeat 3 ~ 9 until the number of clusters become sufficiently small.

Fig 3.11 Dividing city map into area

The resulting clusters are chosen as the areas. The algorithm defines areas so that the maximum distance the user has to cover to travel between two points in the same area is minimized.

3.7.6. LIMITATIONS Of ALGORITHM

- The proposed system can only generate recommendation based on a few hard course constraints.
- The proposed system can only generate recommendation based on price.
- Recommendations generated by this system is still subject to revision by end user.
- Not all of the algorithm principles are implemented in the system.

CHAPTER -4 SYSTEM IMPLEMENTATION

4.1. Introduction

The system implementation defines the construction, installation, testing and delivery of the proposed system. After thorough analysis and design of the system, the system implementation incorporates all other development phases to produce a functional system.

4.2. Choice of Programming Language

Java is the language of choice for this project because of its high speed and low memory usage. The recommendation system is combinatorial in nature hence the need for a programming language that has enhanced CPU optimizing capabilities for the development of an algorithm like the Collaborative filtering algorithm which optimizes search space and avoids local optima.

4.2.1 Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do?, highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.

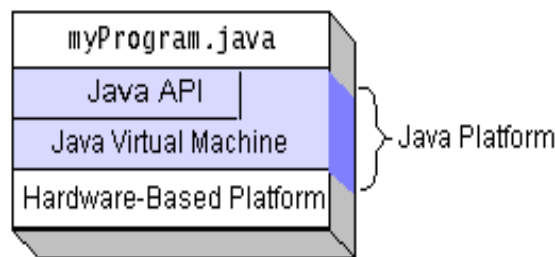


Figure 8: Java Platform

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

4.2.2 Advantages of Java Technology

Java Technology makes your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.

- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code:** The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- **Develop programs more quickly:** Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java:** You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java™ Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.
- **Write once, run anywhere:** Because 100% Pure Java programs are compiled into machine-independent bytecodes, they run consistently on any Java platform.

4.2.3 Android Software Development Kit(sdk)

The Android [software development kit](#) (SDK) includes a comprehensive set of development tools. These include a [debugger,libraries](#), a handset [emulator](#) based on [QEMU](#), documentation, sample code, and tutorials. Currently supported development platforms include computers running [Linux](#) (any modern desktop [Linux distribution](#)), [Mac OS X](#) 10.5.8 or later, and [Windows XP](#) or later. As of March 2015, the SDK is not available on Android itself, but the software development is possible by using specialized Android applications.

Until around the end of 2014, the officially supported [integrated development environment](#) (IDE) was [Eclipse](#) using the Android Development Tools (ADT) Plugin, though [IntelliJ IDEA](#) IDE (all editions) fully supports Android development out of the box, and

[NetBeans](#) IDE also supports Android development via a plugin. As of 2015, [Android Studio](#), made by Google and powered by IntelliJ, is the official IDE; however, developers are free to use others. Additionally, developers may use any text editor to edit Java and XML files, then use [command line](#) tools ([Java Development Kit](#) and [Apache Ant](#) are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).

Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

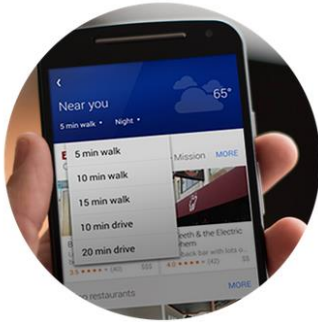
4.2.4 Eclipse

In [computer programming](#), **Eclipse** is an [integrated development environment](#) (IDE). It contains a base [workspace](#) and an extensible [plug-in](#) system for customizing the environment. Eclipse is written mostly in [Java](#) and its primary use is for developing Java applications, but it may also be used to develop applications in other [programming languages](#) through the use of plugins, including: [Ada](#), [ABAP](#), [C](#), [C++](#), [COBOL](#), [Fortran](#). It can also be used to develop packages for the software [Mathematica](#). Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

The initial [codebase](#) originated from [IBM VisualAge](#). The Eclipse [software development kit](#) (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

Released under the terms of the [Eclipse Public License](#), Eclipse [SDK](#) is [free and open-source software](#) (although it is incompatible with the [GNU General Public License](#)). It was one of the first IDEs to run under [GNU Classpath](#) and it runs without problems under [IcedTea](#).

4.2.5 Google Places Api



Location Awareness

Use the power of [mobile](#) to give your users contextual information about where they are, when they're there.



Search Everywhere

[Search for](#) and [retrieve](#) rich information about local businesses and points of interest, available on every screen.



Autocomplete

Add [autocomplete](#) to any application, providing type-ahead location-based predictions like the search on Google Maps.

4.2.6 Four Square Api

You'll need your client ID and client secret to make a userless venue search or explore request ("venues" are what we call places on Foursquare). This is essentially the simplest way to interact with the Foursquare API—there's no need to use OAuth. If you choose not to use a client library, you can directly make HTTP requests to the `venues/search` or `venues/explore` endpoints and get back a JSON response.

In the HTTP request, you need to pass in your client ID, client secret, a version parameter, and any other parameters that the endpoint requires:

`https://api.foursquare.com/v2/venues/search`

`?client_id=CLIENT_ID`

`&client_secret=CLIENT_SECRET`

`&v=20130815`


`&ll=40.7,-74`


`&query=sushi`

What Can You Do With Foursquare?

 Search for Places in an Area

 Connect With Foursquare Users

 Know When Somebody Checks In

 Get Global Streaming Check-In Data

4.3. System Requirements

Below are the conditions a computer system on which the timetable software will be run:

Hardware is best described as a device that is physically connected to your computer or something that can be physically touched. Most hardware will contain a circuit board, ICs, and other electronics. A perfect example of hardware is a computer monitor, which is an output device that lets you see what you're doing on the computer. Without any hardware, your computer would not exist, and software would not be able to run. In the image to the right, are a webcam and an example of an external hardware peripheral that allows users to make videos or pictures and transmit them over the Internet.

4.3.1) Hardware Requirements

The system must have the following hardware requirements:

- Pentium IV Processors
- Minimum 2GB of RAM
- 5GB of Hard Disk

4.3.2) Software Introduction

Software is a general term for the various kinds of programs used to operate computers and related devices. (The term hardware describes the physical aspects of computers and related devices.)

Software can be thought of as the variable part of a computer and hardware the invariable part. Software is often divided into application software (programs that do work users are directly

interested in) and system software (which includes operating systems and any program that supports application software). The term middleware is sometimes used to describe programming that mediates between application and system software or between two different kinds of application software (for example, sending a remote work request from an application in a computer that has one kind of operating system to an application in a computer with a different operating system). An additional and difficult-to-classify category of software is the *utility*, which is a small useful program with limited capability. Some utilities come with operating systems. Like applications, utilities tend to be separately installable and capable of being used independently from the rest of the operating system.

This project involves SQLite for Saving the Product Detail, Location Manager for Phone Locations and SimpleCursorAdapter for Data Binding and Displaying Records in Row And Column format the Records are shown in Scrollable View

Software can be purchased or acquired as shareware (usually intended for sale after a trial period), liteware (shareware with some capabilities disabled), freeware (free software but with copyright restrictions), public domain software (free with no restrictions), and open source (software where the source code is furnished and users agree not to limit the distribution of improvements).

4.3.3) Software Requirements:

At Client Side:-

The system must have the following software requirements:

- JDK (Java Development Kit)
- Eclipse.
- Android Development Tools for Developing Android Application

At Developer Side :-

- Android SDK Tools
- Java 2SE JDK v6.
- Eclipse version 4.6
- Emulator

4.4 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 to [Testing 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:

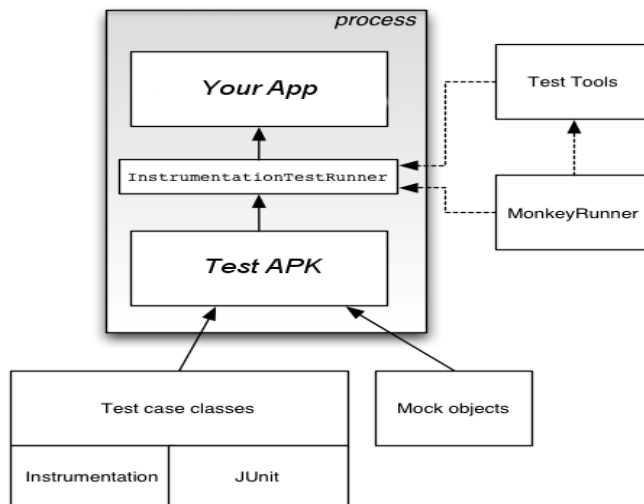


fig 4.1

Normally, an Android component runs in a lifecycle that the system determines. For example, an Activity object's lifecycle starts when an Intent activates the Activity. The system calls the object's onCreate() method, then the onResume() method. When the user starts another app, the system calls the onPause() method. If the Activity code calls the finish() 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](#).

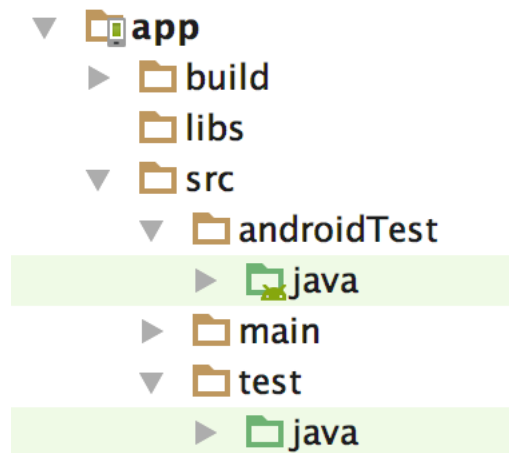


fig 4.2 Default app module test directories.

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 the junit.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 **ActivityTestRule** 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

CHAPTER -5 CONCLUSION

5.1. Summary

This study was carried out as is to reduce the intense manual effort being put into creating and developing Shopping Recommendations. The Recommendation automation system currently is a conceptual work in progress but has the capability to generate near optimal Recommendations based on two unit courses with minimized course constraints.

5.2. Conclusion

In this paper, we have proposed Placebo, a shop recommendation system for real-world shopping based on user location history. The results of the evaluation test show that Placebo estimates users' frequently visited shops with acceptable accuracy when there is a sufficient amount of data, and also demonstrate its potential ability to provide users with beneficial recommendations.

5.3. Recommendations

In furtherance of this work, the following are recommended:

- The Shopping recommendation system developed as the outcome of this project should be made open to avid students of computing who can collaborate and improve on the techniques and ideas inherent in this project.
- Further works on developing a Shopping recommendation system should be based on this research work so as to utilize the incremental model of software development.
- A collaborative model of Shopping Recommendation system which utilizes a computer network can also be built which entails different departments and entities allocating courses and constraints concurrently while the system threads and reports clashes.

5.4. Problems Encountered

Timing constraints was a major issue in the development of this system due to the robustness of the system. Given the time allotment for this project, the system developed could not meet up to the intended robustness.

5.5. Scope for Further Works

Since the amount of data collected in our evaluation test was too small for statistical analysis, we must conduct another evaluation test in a larger scale. We are planning of porting Placebo to mobile phones and making it publicly available, which should help us gather users for our evaluation test.

One modification which should definitely be done in future versions of Placebo is introducing some constraints to the recommendation results regarding some basic information about the user, such as age or sex. In the evaluation test, a shop which only sells fashion items for women was recommended by our system to a male user, and consequently received a rating of 1. Obvious mismatches like this should be excluded from the recommendation results, by defining several basic attributes for each shop and filtering out data with unwanted attributes.

REFERENCES

1. Y Zheng. Location-based social networks: Users. In *Computing with*
2. *SpatialTrajectories*, Zheng, Y and Zhou, X, Ed. Springer, 2011.
3. Y. Zheng, X. Xie, and W.Y.Ma. Geolife: A collaborative social networkingservice among user, location and trajectory. *IEEE Data Engineering Bulletin*,33(2):32–40, 2010.
4. Mohamed Mokbel, JieBao, Ahmed Eldawy, Justin Levandoski, and MohamedSarwat. Personalization, Socialization, and Recommendations inLocation-based Services 2.0. In *PersDB. VLDB*, 2011.
5. Marmasse, N. Schmandt, C. Location-aware Information Delivery with Commotion. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing table of contents Bristol, UK. Pages: 157 - 171*, 2000.
6. Ashbrook, D. and Starner, T. Learning signifcantlocations ?nd predicting user movement with GPS. In *Proc. of 6th IEEE Intl. Symp. on Wearable Computers*, Seattle, WA, 2002.
7. J.A. Konstan, B.N. Miller, D. Maltz, J.L. Herlocker, L.R. Gordon, and J. Riedl.Grouplens: applying collaborative filtering to usenet news. *Communications ofthe ACM*, 40(3):77–87, 1997.
8. J. Levandoski,MohamedSarwat, Ahmed Eldawy, and Mohamed Mokbel. Lars:A location-aware recommender system. In *ICDE*, 2012.
9. TzvetanHorozov, NityaNarasimhan, and VenuVasudevan. Using location forpersonalized poi recommendations in mobile environments. In *SAINT*, pages124–129, 2006.
10. Fogg, B. J., *Persuasive Technology : Using Computers to Change What We Think and Do*. Morgan Kaufmann Publishers, 2003.
11. Want, R., Hopper, A., Falcao, V., and Gibbons, J. The Active Badge Location System. *ACM Transactions on Information Systems (TOIS)*, v.10 n.1, p.91-102, Jan. 1992.
12. Want, R., Schilit, B., Adams, A., Gold, R., Petersen, K., Goldberg, D., Ellis, J., and Weiser, M. The ParcTab Ubiquitous Computing Experiment. Technical Report CSL-95-1, Xerox Palo Alto Research Center, March 1995.
13. Bahl, P., and Padmanabhan, V. N. RADAR: An In-Building RF Based User Location and Tracking System. In *INFOCOM 2000*, pages 775– 784, 2000.

14. Bahl, P., and Padmanabhan, V. N. *A Software System for Locating Mobile Users: Design, Evaluation, and Lessons*. MSR Technical Report, February 2000.
15. Ward, A., Jones, A., Hopper, A. *A New Location Technique for the Active Office*. In *IEEE Personal Communications*, Vol. 4, No. 5, October 1997, pp 42-47.
16. Aboud, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M., *Cyberguide: A mobile context-aware tourguide*, *ACM Wireless Networks*, 1997, pp.421-433.
17. Marmasse, N. Schmandt, C. *Location-aware Information Delivery with Commotion*. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing table of contents Bristol, UK. Pages: 157 - 171, 2000*.
18. Ashbrook, D. and Starner, T. *Learning significant locations and predicting user movement with GPS*. In *Proc. of 6th IEEE Intl. Symp. on Wearable Computers*, Seattle, WA, 2002.
19. Zhou, C., Frankowski, D., Ludford, P., Shekhar, S., Terveen, L. *Discovering Personal Gazetteers: An Interactive Clustering Approach*. In *Proceedings of ACM GIS 2004*, Washington DC, 2004, pp. 266-273.
20. Chen, L., and Sycara, K. *WebMate: Personal Agent for Browsing and Searching*. *Proceedings of the 2nd International Conference on Autonomous Agents and Multi Agent Systems, AGENTS '98*, ACM, May, 1998, pp. 132 - 139.
21. Lang, K. *NewsWeeder: Learning to Filter Netnews*. In *Proceedings of 12th International Conference on Machine Learning*, Lake Tahoe, CA, Morgan Kaufmann, pp. 331-339, 1995.
22. Goldberg, D., Nichols, D., Oki, Brian M., Terry, D., *Using Collaborative Filtering to Weave an Information Tapestry*. *Communications of the ACM*, v.35 n.12, p.61-70, Dec. 1992.
23. Shardanand, U., Maes, P., *Social Information Filtering: Algorithms for Automating "word of mouth"*. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*. New York, (pp. 210-217).
24. Sarwar, B., Karypis, G., Konstan, J., Riedl, J. *Item-based Collaborative Filtering Recommendation Algorithms*. In *Proceedings of 10th International World Wide Web Conference*, ACM Press, 2001, pp.

SCREEN SHOT OF APPLICATION

