

TOURIST GUIDE

Project report submitted

In partial fulfilment of the requirements for the degree of

Bachelor of Computer Applications

by

Amul Bakshish Singh (1601060005)

Sandipta Subir Khare (1601060002)

Under the Supervision Of

Ms. Anjali Singh

Assistant Professor, CSE



School of Engineering and Technology

K. R. Mangalam University, Gurugram, Haryana, India

May 2019

CERTIFICATE

It is certified that the work contained in the project report titled "Tourist Guide" by the following students:

Name of the Student	Roll Number
AMUL BAKSHISH SINGH	1601060005
SANDIPTA SUBIR KHARE	1601060002

has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Ms. Anjali Singh,
Assistant Professor,
Computer Science and Engineering,
K R Mangalam University, Gurugram, Haryana, India

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of the Student	Roll Number	Signature
Amul Bakshish Singh	1601060005	
Sandipta Subir Khare	1601060002	

Date: _____

APPROVAL SHEET

This project report entitled **Tourist Guide** by **Amul Bakshish Singh and Sandipta Subir Khare** is approved for the degree of **Bachelor of Computer Applications, School of Engineering and Technology**

Dean, SOET

Prof. (Dr.) Ranjit Varma

Supervisor

Ms. Anjali Singh, Assistant Prof., CSE, SOET

Date: _____

Place: _____

ACKNOWLEDGEMENT

It gives me immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide **Ms. Anjali Singh, Assistant Prof., Computer Science and Engineering, SOET**, for her valuable guidance, encouragement and help for completing this work. Her useful suggestions for this whole work and cooperation are sincerely acknowledged.

We would like to express our sincere thanks to **Prof. (Dr.) Ranjit Varma, Dean, SOET** for giving us the opportunity to undertake this project. We are also grateful to all our teachers and the Computer Science and Engineering department, SOET, for the constant support and guidance.

We also wish to express our indebtedness to our parents as well as our family member whose blessings and support always helped us to face the challenges ahead.

At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during this project work.

Place: Gurugram

Amul Bakshish Singh (1601060005)

Date

Sandipta Subir Khare (1601060002)

ABSTRACT

Tourist Guide is the software aimed at providing a wide range of access to the clients to get know the locality of their nearby places in just seconds by just selecting the type of place they want to visit.

The clients can even maintain their favourite place that they frequently visit and can save them for further usage.

The app will use a Google account of a client to maintain their favourite places in a database.

All the Data is collected in the app is from Google so the client can be satisfied about the place they are visiting will be genuine. The app also provide the navigation to the place the client want to visit using Google maps client just need to click the button and the app will take you to the Google map.

The app also provides the location and number of the place so the client can directly reach the place to clear his/her query.

TABLE OF CONTENTS

Certificate	i.
Candidate's Declaration	ii.
Approval Sheet	iii.
Acknowledgement	iv.
Abstract	v.
Chapter 1: INTRODUCTION	1.
1.1 Overview	1.
1.2 Different Phases of the Development Process:	1.
1.3 Android Software Development	3.
Chapter 2: LITERATURE REVIEW	8.
Chapter 3: OBJECTIVES AND METHODOLOGY	9.
3.1 Objective	9.
3.2 Methodology	9.
Chapter 4: IMPLEMENTATION / CODING	10.
4.1 The Classes	10.
4.2 Main Activity	10.
4.3 The Database Connection	24.
4.4 Location Service	25.
4.5 Navigate Service	29.
4.6 The Google Services	30.
4.7 Connection Detector	32.
Chapter 5: RESULT AND DISCUSSION	33.
5.1 Snapshots/Results	33.
5.2 Discussion	39.

Chapter 6: CONLUSIONS AND FUTUE SCOPE	40.
6.1 Conclusions	40.
6.2 Future Scope	40.
REFRENCES	41.

LIST OF FIGURES

Figure Number	Page Number
1.1	3
4.1	10
4.2	10
4.3	11
4.4	11
4.5	12
4.6	12
4.7	13
4.8	13
4.9	14
4.10	14
4.11	15
4.12	15
4.13	16
4.14	16
4.15	17
4.16	17
4.17	18
4.18	18
4.19	19
4.20	19
4.21	20
4.22	20
4.23	21
4.24	22
4.25	22
4.26	22
4.27	23
4.28	23
4.29	24
4.30	24
4.31	25
4.32	25
4.33	26
4.34	26
4.35	27
4.36	27
4.37	28
4.38	28
4.39	29
4.40	29
4.41	30
4.42	30
4.43	31
4.44	31
4.45	32
4.46	32
5.1	33
5.2	34
5.3	35
5.4	35
5.5	36
5.6	36
5.7	37
5.8	37
5.9	38

CHAPTER 1: INTRODUCTION

1.1 Overview

Tourist Guide application is a Place to Visit application useful to find the Popular places to visit in India. Following things can be done with the application:

- User will also be able view a geographical map of the area.
- The data fed into the application will be used to search for the Popular Places in India and same as place to visit in that location.
- The user will only have to provide a radius of area to be searched.

The project is on the Tourist Guide Android Application. In this Application we would have few things and that are:

- **Places to Visit in India:** When you click on Place to Visit in India, with the help of Google map you would get many places to visit such as Amritsar, Himachal Pradesh, etc.
- **Popular area to Visit in the Area which you had chosen in Places to Visit in India.** In other words, it means when you choose the option in Places to visit in India, there are many places to visit in that location. For example, if you choose the option Amritsar in Places to Visit in India, the Popular Area to Visit would show many option such as Golden Temple, Wagah Border, etc.

1.2 Different Phases of The Development Process:

1.1.2 Requirement Analysis:

Requirement analysis is done in order to understand the problem the software system is to solve. The problem could be automating an existing manual process, developing a new automated system, or a combination of the two. The emphasis in requirements analysis is on identifying what is needed from the system, not how the system will achieve its goals. There are atleast two parties involved in the software development-a client and a developer. The developer has to develop the system to satisfy the client's needs. The developer does not understand the client's problem domain, and the client does not understand the issues involved

in the software systems. This causes a communication gap, which has to be adequately bridged during requirements analysis.

Software Requirement Specification

If a document that completely describes what the proposed software should do without describing how the software will do it, The basic goal of requirements specific to produce the requirements, which describes the complete external behaviour of the proposed software.

However, producing is said to be done.

The basic limitation is that the user's need keep on changing as the environment in which system was to function changes with time. This happens more in complex application where all the needs may not be known to any set of people during the requirement phase. Types of error in this specification can clearly be avoided or at least considerably, by properly performing the requirement phase.

As many times a developer or administrator does not know what a user and the supplier on what the software product will do. This is the only document which tells the time agreement and procedure of complete project as well as what a user wants.

There are some steps which are considered in the Software Requirement Specification Such As

- Information Description
- Function Description
- Behavioural Description
- Validation Checks and Criteria
- Reference

1.2.2 Software Design

The purpose of the design phase is to plan a solution of the problem specified by the requirements documents. This phase is the first step in moving from the problem domain to the solution domain. Starting with what is needed, design takes us toward how to satisfy the needs. The design of a system is perhaps the most critical factor affecting the quality of the software. It has a major impact on the later phases, particularly testing and maintenance.

The design activity is divided into two phases: System Design and Detailed Design. In system design the focus is on identifying the modules, whereas during detailed design the focus is on designing the logic for each of the modules.

1.2.3 Coding

The goal of the coding phase is to translate the design of the system into code in a given programming language. Hence during coding, the focus should be on developing programs that are easy to read and understand, and not simply on developing programs that are easy to write.

1.2.4 Testing

Testing is the major quality control measure used during software development. Its basic function is to detect errors in the software. Testing not only uncover errors introduced during coding, but also errors introduced during the previous phases. Thus, the goal of the testing is to uncover requirement, design and coding errors in the programs. Therefore, different levels of testing are used. Testing is an extremely critical and time-consuming activity. It requires proper planning of the overall testing process. The output of the testing phase is the test report and the error report. Test report contains the set of test cases and the result of executing the code with these test cases. The error report describes the errors encountered and the action taken to remove the errors.

1.2.5 Operation & maintenance phase

Software maintenance is a task that every development group has to face, when the software is delivered to the customer's site, installed and is operational. Software maintenance is a very broad activity that includes error correction, enhancement of capabilities, deletion of obsolete capabilities and optimization.

1.3 Android Software Development

Android software development is the process by which new applications are created for the Android operating system. Applications are usually developed in the Java programming language using the Android Software Development Kit, but other development tools are available. As of October 2012, more than 700,000 applications have been developed for Android, with over 25 billion downloads. A June 2011 research indicated that over 67% of mobile developers used the platform, at the time of publication. In Q2 2012; around 105 million

units of Android smart phones were shipped which acquires a total share of 68% in overall smart phones sale till Q2 2012.



Fig. No. 1.1

1.3.1 Software development tools

1.3.1.1 Android SDK

The Android software development kit (SDK) includes a comprehensive set of development tools.^[6] 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, Windows XP or later; for the moment one cannot develop Android software on Android itself. The officially supported integrated development environment (IDE) is 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. 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.

Android applications are packaged in .apk format and stored under /data/app folder on the Android OS (the folder is accessible only to the root user for security reasons). APK package contains .dex files (compiled byte code files called Dalvik executables), resource files, etc.

1.3.1.2 Android Debug Bridge

The Android Debug Bridge (ADB) is a toolkit included in the Android SDK package. It consists of both client and server-side programs that communicate with one another. The ADB is typically accessed through the command-line interface.

The format for issuing commands through the ADB is typically:

```
adb [-d|-e|-s <serialNumber>] <command>
```

In a security issue reported in March 2011, ADB was targeted as a vector to attempt to install a rootkit on connected phones using a "resource exhaustion attack".^[13]

1.3.1.3 Fast boot

Fast boot is a diagnostic protocol included with the SDK package used primarily to modify the flash file system via a USB connection from host computer. It requires that the device be started in a boot loader or Second Program Loader mode in which only the most basic hardware initialization is performed. After enabling the protocol on the device itself, it will accept a specific set of commands sent to it via USB using a command line. Some of most commonly used fast boot commands include:

- Flash - Rewrites a partition with a binary image stored on the host computer.
- Erase - Erases a specific partition.
- Reboot - Reboots the device into either the main operating system, the system recovery partition or back into its boot loader.
- Devices - Displays a list of all devices (with the serial number) connected to the host computer.
- Format - Format a specific partition. The file system of the partition must be recognized by the device.

1.3.1.4 Native development kit

Libraries written in C and other languages can be compiled to ARM, MIPS or x86 native code and installed using the Android Native Development Kit. Native classes can be called from Java code running under the Dalvik VM using the System.loadLibrary call, which is part of the standard Android Java classes.

Complete applications can be compiled and installed using traditional development tools.^[16] The ADB debugger gives a root shell under the Android Emulator which allows native ARM, MIPS or x86 code to be uploaded and executed. Native code can be compiled using GCC on a standard PC. Running native code is complicated by Android's use of a non-standard C library (libc, known as Bionic). The graphics library that Android uses to arbitrate and control access to this device is called the Skia Graphics Library (SGL), and it has been released under an open source license. Skia has back ends for both Win32 and Unix, allowing the development of cross-platform applications, and it is the graphics engine underlying the Google Chrome web browser.

Unlike Java application development based on the Eclipse IDE, the NDK is based on command-line tools and requires invoking them manually to build, deploy and debug the apps. Several third-party tools allow integrating the NDK into Eclipse and Visual Studio.

1.3.1.5 Android Open Accessory Development Kit

The Android 3.1 platform (also backported to Android 2.3.4) introduces Android Open Accessory support, which allows external USB hardware (an Android USB accessory) to interact with an Android-powered device in a special "accessory" mode. When an Android powered device is in accessory mode, the connected accessory acts as the USB host (powers the bus and enumerates devices) and the Android-powered device acts as the USB device. Android USB accessories are specifically designed to attach to Android-powered devices and adhere to a simple protocol (Android accessory protocol) that allows them to detect Android powered devices that support accessory mode.

1.3.1.6 App Inventor for Android

On 12 July 2010, Google announced the availability of App Inventor for Android, a Web-based visual development environment for novice programmers, based on MIT's Open Blocks Java library and providing access to Android devices' GPS, accelerometer and orientation data, phone functions, text messaging, speech-to-text conversion, contact data, persistent storage, and Web services, initially including Amazon and Twitter. "We could only have done this because Android's architecture is so open," said the project director, MIT's Hal Abelson. Under development for over a year, the block-editing tool has been taught to non-majors in computer science at Harvard, MIT, Wellesley, Trinity College (Hartford,) and the University of San Francisco, where Professor David Wolber developed an introductory computer science course and tutorial book for non-computer science students based on App Inventor for Android.

1.3.1.7 Hyper net Android Creator

Hyper net Android Creator (HAC) is a software development system aimed at beginner programmers that can help them create their own Android apps without knowing Java and the Android SDK. It is based on HyperCard that treated software as a stack of cards with only one card being visible at any one time and so is well suited to mobile phone applications that have only one window visible at a time. Hyper net Android Creator's main programming language is simply called hyper net and is loosely based on HyperCard's HyperTalk language. Hyper net is an interpreted English-like language and has many features that allow creation of Android applications. It supports a growing subset of the Android SDK including its own versions of the GUI control types and automatically runs its own background service, so apps can continue to run and process information while in the background.

1.3.1.8 SDL

The SDL library offers also a development possibility beside Java, allowing the development with C and the simple porting of existing SDL and native C applications. By injection of a small Java shim and JNI the usage of native SDL code is possible, allowing Android ports like e.g. the Jagged Alliance 2 video game.

CHAPTER 2: LITERATURE REVIEW

For this application we basically used help of YouTube and Google to know the

- Structure [2]
- Code [2]
- GUI [2]
- Internal Programs [2]
- Working of an application [2]

Some Quotes about Tourist Guide:

- ^[1] “Travel makes one modest. You see what a tiny place you occupy in the world.” - Gustave Flaubert
- ^[1] “See the world. It's more fantastic than any dream made or paid for in factories. Ask for no guarantees, ask for no security.” -Ray Bradbury, Fahrenheit 451
- ^[1] “It is always sad to leave a place to which one knows one will never return. Such are the melancholies du voyage: perhaps they are one of the most rewarding things about traveling.” -Gustave Flaubert, Flaubert in Egypt: A Sensibility on Tour

CHAPTER 3: OBJECTIVES AND METHODOLOGY

3.1 Objective •

To facilitate the tour operators by providing them with qualified tourist guides for guiding assignments.

- Applications provide you all helps that your travel needs, such as location details, Restaurant details and explore local experience
- The use of mobile apps for traveling will increase and continue to evolve in the coming days.
- Travel apps are also used as powerful marketing tools. You can consider this idea to promote your business.

3.2 Methodology

- The application is made by a software ANDROID STUDIO
- We used YouTube to understand and how to make the application

CHAPTER 4: IMPLEMENTATION / CODING

4.1 The Classes

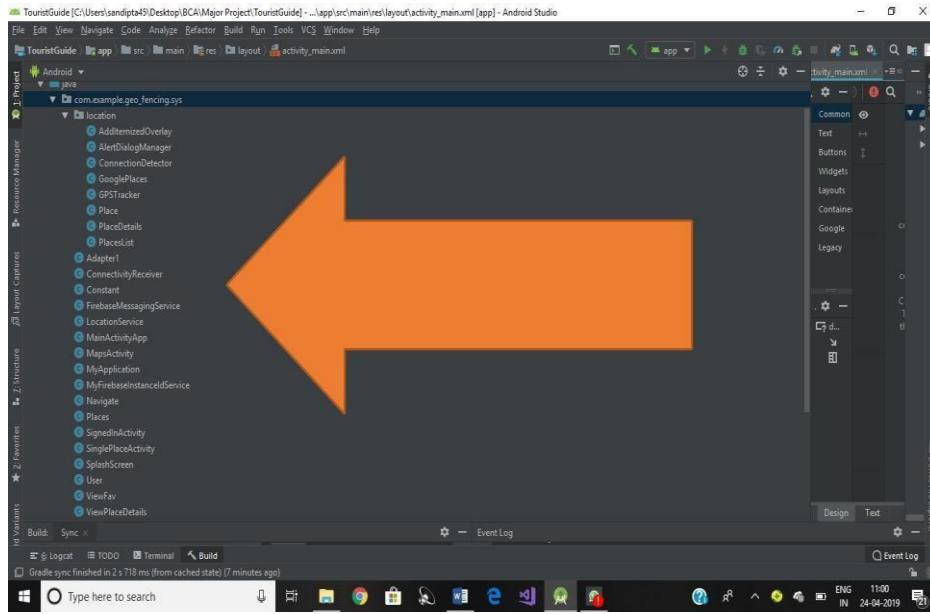


Fig 4.1: The Classes

4.2 Main Activity

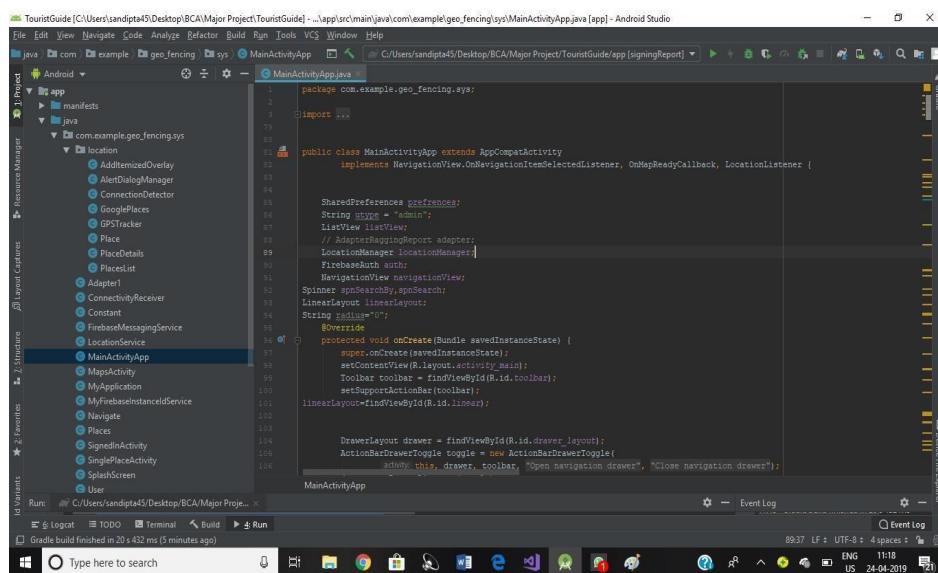


Fig 4.2

The screenshot shows the Android Studio interface with the code editor open to `MainActivityApp.java`. The code is part of a class that handles navigation and search functionality. It includes imports for various Android components like `NavigationView`, `ActionBarDrawerToggle`, and `AdapterView`. The main logic involves setting up a drawer, initializing navigation views, and managing a list view with a custom adapter. It also interacts with Firebase and Google Places services.

```
drawer.setDrawerListener(toggle);
toggle.syncState();

navigationView = findViewById(R.id.nav_view);
navigationView.setOnNavigationItemSelectedListener(this);

context = this;
preferences = PreferenceManager.getDefaultSharedPreferences(context);
listView = findViewById(R.id.list);
// adapter = new AdapterRaggingReport(this, R.layout.custom_list, lstRaggingList);
// listView.setAdapter(adapter);
database = FirebaseDatabase.getInstance();
myRef = database.getReference(Constant.DB);
if (utype.equals("admin")) {

}

spnSearchBy=findViewById(R.id.spnSearchBy);
spnSearch=findViewById(R.id.spnSearch);
txtSearch = findViewById(R.id.txtSearch);

spnSearch.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        if(spnSearch.getSelectedItem().toString().equalsIgnoreCase(anotherString("Amusement park"))){
            search="amusement park";
        }else if(spnSearch.getSelectedItem().toString().equalsIgnoreCase(anotherString("Aquarium"))){
            search="aquarium";
        }else if(spnSearch.getSelectedItem().toString().equalsIgnoreCase(anotherString("Restaurant"))){
            search="restaurant";
        }else if(spnSearch.getSelectedItem().toString().equalsIgnoreCase(anotherString("Museum"))){
            search="museum";
        }
    }
});
```

Fig 4.3

This screenshot shows the same `MainActivityApp.java` file as Fig 4.3, but with a different portion of the code visible. The class definition and some initial setup code are shown. The code includes imports for `AppCompatActivity`, `NavigationView`, and `LocationListener`. It defines shared preferences and initializes various views and adapters. The `onCreate` method is partially visible, showing the setup of the navigation drawer and toolbar.

```
package com.example.geo_fencing.sys;

import ...

public class MainActivityApp extends AppCompatActivity
    implements NavigationView.OnNavigationItemSelectedListener, OnMapReadyCallback, LocationListener {

    SharedPreferences preferences;
    String utype="admin";
    ListView listView;
    // AdapterRaggingReport adapter;
    LocationManager locationManager;
    FirebaseAuth auth;
    NavigationView navigationView;
    Spinner spnSearchBy, spnSearch;
    LinearLayout linearLayout;
    String radius="0";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        linearLayout=findViewById(R.id.linear);

        DrawerLayout drawer = findViewById(R.id.drawer_layout);
        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            this, drawer, toolbar, "Open navigation drawer", "Close navigation drawer");
    }
```

Fig 4.4

TouristGuide [C:\Users\sandipta45\Desktop\BCA\Major Project\TouristGuide] - ...app\src\main\java\com\example\geo_fencing\sys>MainActivityApp.java [app] - Android Studio

```

138     }else if(spnSearch.getSelectedItem().toString().equalsIgnoreCase("Zoo")){
139         search="zoo";
140     }else if(spnSearch.getSelectedItem().toString().equalsIgnoreCase("Shopping Mall")){
141         search="shopping mall";
142     }else if(spnSearch.getSelectedItem().toString().equalsIgnoreCase("Ac")){
143         search="ac";
144     }else if(spnSearch.getSelectedItem().toString().equalsIgnoreCase("Hospital")){
145         search="hospital";
146     }else{
147         search="Select Option";
148     }
149 }
150
151 @Override
152 public void onNothingSelected(AdapterView<?> parent) {
153 }
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169

```

such = FirebaseAuth.getInstance();
if (auth.getCurrentUser() == null) {
startActivityForResult(
AuthUI.getInstance() .createSignInIntentBuilder()
.setTosUrl("https://superapp.example.com/terms-of-service.html") .SigninIntentBuilder
.setPrivacyPolicyUrl("https://superapp.example.com/privacy-policy.html") .SigninIntentBuilder
.setAvailableProviders(
Arrays.asList(

new AuthUI.IdpConfig.Builder(AuthUI.GOOGLE_PROVIDER).build()

Fig 4.4

TouristGuide [C:\Users\sandipta45\Desktop\BCA\Major Project\TouristGuide] - ...app\src\main\java\com\example\geo_fencing\sys>MainActivityApp.java [app] - Android Studio

```

165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199

```

SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
.findFragmentById(R.id.map);
mapFragment.getMapAsync(onMapReadyCallback: this);
Intent intent=new Intent(getApplicationContext(),LocationService.class);
startService(intent);

ArrayList<Places> lstSchools = new ArrayList<>();
ProgressDialog dialog;
String search;

public void searchSchools(View v) {

 String place = txtSearch.getText().toString();

 if (!search.equalsIgnoreCase("Select Option")) {
 if (place.length() > 0) {
 place = Uri.encode(place);
 dialog = ProgressDialog.show(context,
 title: "Please Wait ...",
 message: "Getting Places...", indeterminate: true, cancelable: false);
 final com.loopj.android.http.RequestParams params = new com.loopj.android.http.RequestParams();
 // params.add("key", Constants.api_key);
 // params.add("query", place);
 // params.put("radius", 500);
 }
 }
}

Fig 4.5

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TouristGuide". The "app" module contains "Manifests" and "java" packages. The "java" package includes a "com.example.geo_fencing.sys" package which contains a "location" package. Inside "location" are classes: AddItemizedOverlay, AlertDialogManager, ConnectionDetector, GooglePlaces, GPSTracker, Place, PlaceDetails, PlacesList, Adapter1, ConnectivityReceiver, Constant, FirebaseMessagingService, LocationService, MainActivityApp, MapsActivity, MyApplication, MyFirebaseInstanceService, Navigate, Places, SignedInActivity, SinglePlaceActivity, SplashScreen, and User.
- MainActivityApp.java:** The code implements the `onCreate` method. It adds a "parking" parameter to a params map, sets a sensor to false, and initializes an AsyncHttpClient. It then constructs a URL for a place search API call, sets up a response handler, and handles the success response by dismissing a dialog and printing the JSON results to the log. It also handles the failure response by printing the error message.
- Run Tab:** The run configuration is set to "C:/Users/sandipta45/Desktop/BCA/Major Proj...".
- Event Log:** The event log shows a build completed successfully in 20 s 432 ms (9 minutes ago).

Fig 4.6

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TouristGuide". The package structure is visible under "com.example.geo_fencing.sys".
- MainActivityApp.java:** This file contains Java code for handling location data and displaying it on a map.
- Code Snippet:** The code uses a JSON array to parse location data and creates a Places object for each entry. It then adds these places to a list and logs the success message.

```
String icon = jsonArray.getJSONObject(count).getString("icon");
String title = jsonArray.getJSONObject(count).getString("name");
String id = jsonArray.getJSONObject(count).getString("reference");
String formatted_address = jsonArray.getJSONObject(count).getString("formatted_address");
Places places = new Places();
places.setPlaceId(id);
places.setImage(icon);
places.setLatitude(lat);
places.setLongitude(lon);
places.setName(title);
places.setDescription(formatted_address);

listSchools.add(places);

Log.i(TAG, msg: "onSuccess:id=" + id);
Log.i(TAG, msg: "onSuccess:icon=" + icon);
if (count == 0) {
    String params[] = {icon, id, title, Double.toString(lat), Double.toString(lon), "true", formatAddress};
    addMarker.addMarker = new AddMarker();
    addMarker.execute(params);
} else {
    String params[] = {icon, id, title, Double.toString(lat), Double.toString(lon), "false", formatAddress};
    addMarker.addMarker = new AddMarker();
    addMarker.execute(params);
}
}

count++;
}
```

- Run Tab:** The run configuration is set to "C:/Users/sandipt45/Desktop/BCA/Major Project..."
- Event Log:** The event log is empty.
- Bottom Bar:** Shows the system tray with icons for battery, signal, and time (11:22).

Fig 4.7

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TouristGuide" located at "C:/Users/sandipta45/Desktop/BCA/Major Project/TouristGuide".
- File Path:** The current file is "MainActivityApp.java" under the package "com.example.geo_fencing.sys".
- Code Editor:** The code for the `MainActivityApp.onCreate()` method is displayed. It includes logic for handling place search results and selecting place types.
- Toolbars and Menus:** Standard Android Studio menus like File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help are visible.
- Side Panels:** The Resource Manager, Layout Captures, and Device File Explorer panels are open on the left.
- Bottom Bar:** The bottom bar shows the run configuration ("Run: C:/Users/sandipta45/Desktop/BCA/Major Proj..."), the Event Log tab, and the status bar indicating the build was successful in 9 minutes ago.

Fig 4.8

The screenshot shows the Android Studio interface with the code editor open to `MainActivityApp.java`. The code implements a `GetAddressTask` class that extends `AsyncTask<String, Void, Bitmap>`. It has two overridden methods: `onPreExecute()` and `doInBackground(String... params)`. The `onPreExecute()` method contains a TODO comment. The `doInBackground` method takes parameters for title, id, latitude, longitude, move, and formatted address, then uses Picasso to load an image from a URL and return it as a bitmap.

```
253     }
254 }
255 class GetAddressTask extends AsyncTask<String, Void, Bitmap> {
256     String title, id;
257     LatLng latlng;
258     String move;
259     String formatted_address;
260
261     @Override
262     protected void onPreExecute() {
263         // TODO Auto-generated method stub
264         super.onPreExecute();
265     }
266
267     @Override
268     protected Bitmap doInBackground(String... params) {
269         id = params[0];
270         title = params[1];
271         move = params[2];
272         formatted_address = params[3];
273         latlng = new LatLng(Double.parseDouble(params[4]), Double.parseDouble(params[5]));
274         Bitmap bmp = null;
275         try {
276             bmp = Picasso.with(context).load(params[6]).get();
277         } catch (IOException e) {
278             e.printStackTrace();
279         }
280         return bmp;
281     }
282 }
283 }
```

The project structure on the left shows the `com.example.geo_fencing.sys` package containing a `location` folder with various location-related classes like `AddMarker`, `AlertDialogManager`, `ConnectionDetector`, `GooglePlaces`, `GPSTracker`, `Place`, `PlaceDetails`, `PlacesList`, `Adapter1`, `ConnectivityReceiver`, `Constant`, `FirebaseMessagingService`, `LocationService`, `MainActivityApp`, `MapsActivity`, `MyApplication`, `MyFirebaseInstanceService`, `Navigate`, `Places`, `SignedInActivity`, `SinglePlaceActivity`, `SplashScreen`, and `User`.

Fig 4.9

The screenshot shows the Android Studio interface with the file `MainActivityApp.java` open. The code is part of the `onPostExecute` method. It handles a move event by adding markers to a map. If the move is "true", it adds a marker with a title and snippet, sets its info window, and moves the camera. If the move is "false", it adds a marker with a title and snippet, sets its info window, and animates the camera. The code uses `MarkerOptions` and `BitmapDescriptorFactory` to define the markers.

```
    @Override
    protected void onPostExecute(Bitmap result) {
        super.onPostExecute(result);

        if (move.equals("true")) {
            Marker marker = mMap.addMarker(new MarkerOptions()
                .position(latLong)
                .title(title)
                .snippet(formatted_address)
                .alpha(1.0f));
        }
        Marker marker = mMap.addMarker(new MarkerOptions()
            .position(latLong)
            .title(title)
            .snippet(formatted_address)
            .alpha(1.0f)
            .icon(BitmapDescriptorFactory.fromBitmap(result)));
        marker.showInfoWindow();
        marker.setTag(id);
        mMap.moveCamera(CameraUpdateFactory.newLatLng(latLong));
        mMap.animateCamera(CameraUpdateFactory.newLatLng(latLong));
    } else {
        Marker marker = mMap.addMarker(new MarkerOptions()
            .position(latLong)
            .title(title)
            .snippet(formatted_address)
            .alpha(1.0f));
    }
}
```

Fig 4.10

The screenshot shows the Android Studio interface with the file `MainActivityApp.java` open. The code is part of the `onMapReady` method. It initializes variables for a search bar, database, and reference. It then checks for location permissions and moves the camera to a specific latitude and longitude. If permission is granted, it adds a marker for the user's location.

```
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        latitude = Double.parseDouble(preferences.getString("latitude", "28.7"));
        longitude = Double.parseDouble(preferences.getString("longitude", "77.33"));
        // Add a marker in Sydney and move the camera
        LatLng sydney = new LatLng(latitude, longitude);
        mMap.addMarker(new MarkerOptions().position(sydney).title("My Location"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
        mMap.setMinZoomPreference(9.0f);
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED && ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
    
```

Fig 4.11

```

    ContextCompat.checkSelfPermission(context, Manifest.permission.CALL_PHONE) == PackageManager.PERMISSION_GRANTED) {
        mMap.setMyLocationEnabled(true);
        mMap.getUiSettings().setMyLocationButtonEnabled(true);

        mMap.setOnMarkerClickListener(marker) {
            return false;
        };
        mMap.setOnInfoWindowClickListener(marker) {
            if (marker.getSnippet() != null) {
                Intent intent = new Intent(getApplicationContext(), SinglePlaceActivity.class);
                intent.putExtra("name", "reference", marker.getTag().toString());
                startActivity(intent);
            }
        };
    } else {
        ActivityCompat.requestPermissions(activity, new String[]{
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.CALL_PHONE,
            Manifest.permission.ACCESS_COARSE_LOCATION
        }, requestCode: 1);
    }
}

```

The screenshot shows the Android Studio interface with the code editor open to the `MainActivityApp.java` file. The code handles location permissions by checking if the `CALL_PHONE` permission is granted. If granted, it enables my location and my location button. It also sets listeners for marker clicks and info windows. If the permission is not granted, it requests the `ACCESS_FINE_LOCATION`, `CALL_PHONE`, and `ACCESS_COARSE_LOCATION` permissions with a request code of 1.

Fig 4.11

```

private static final int NOTIFY_ME_ID = 133;
@Override
public void onLocationChanged(Location location) {
    Log.i(TAG, msg: "onLocationChanged:" + location.getLatitude() + "," + location.getLongitude());
    latitude = location.getLatitude();
    longitude = location.getLongitude();

    Log.i("Lat", msg: latitude + "" + longitude);

    try {
        jsonArray = new JSONArray(preferences.getString("cart_items", null));
        int count = 0;

        while (count < jsonArray.length()) {
            JSONObject jsonObject = jsonArray.getJSONObject(count);

            double distance = distFrom(Double.parseDouble(jsonObject.getString("latitude")),
                Double.parseDouble(jsonObject.getString("longitude")),
                location.getLatitude(), location.getLongitude());

            Log.i("distance", msg: "distance");
            if (firstReminder && distance <= 10000) {

                Intent i = new Intent(context, SinglePlaceActivity.class);
                i.putExtra("name", "speak", value: "You are near " + jsonObject.getString("name"));
                i.putExtra("name", "description", jsonObject.getString("address"));
                i.putExtra("name", "reference", jsonObject.getString("id"));
                PendingIntent resultPendingIntent =
            }
        }
    }
}

```

The screenshot shows the Android Studio interface with the code editor open to the `MainActivityApp.java` file. The code handles location changes by logging the new location coordinates and then iterating through a JSON array of cart items. For each item, it calculates the distance between the current location and the item's location. If the first reminder is true and the distance is less than or equal to 10,000 meters, it creates an intent to start the `SinglePlaceActivity` and includes extra data such as the item's name, address, and ID.

Fig 4.12

Fig 4.13

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TouristGuide". The main module is "MainActivityApp".
- File Path:** The code is located at `com.example.geo_fencing.MainActivityApp.java`.
- Code Content:** The code implements a `LocationService` and defines a `distFrom` method to calculate distance between two points.
- Code Snippet:**

```
        Log.i("GEOFENCE", "broadcast sent ");
        preferences.edit().putString("latitude", Double.toString(latitude)).putString("longitude", Double.toString(longitude));
    }

    boolean firstReminder = true;
    //in meter
    public static double distFrom(double lat1, double lon1, double lat2, double lon2) {
        double earthRadius = 6371000; //meters
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
            Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
            Math.sin(dLon / 2) * Math.sin(dLon / 2);
        double c = 2 * Math.asin(Math.sqrt(a));
        double dist = (double) (earthRadius * c);

        return dist;
    }

    /**
     * Calculate distance between two points in latitude and longitude taking
     * into account height difference. If you are not interested in height
     * difference pass 0.0. Uses Haversine method as its base.
     * <p>
     * lat1, lon1 Start point lat2, lon2 End point ell Start altitude in meters
     * ell End altitude in meters
     * </p>
     * <returns> Distance in Meters
     */
}
```
- Toolbar:** Standard Android Studio toolbar with icons for File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Bottom Bar:** Includes Run, Logcat, TODO, Terminal, Build, and Gradle build status.
- Bottom Status Bar:** Shows the current time (12:41), language (ENG), and system status (4 spaces).

Fig 4.14

```

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
    ...
}

@Override
public void onProviderEnabled(String provider) {
    ...
}

@Override
public void onProviderDisabled(String provider) {
    ...
}

// Called when a view has been clicked.
// param v The view that was clicked.
String token_id;

@Override
protected void onResume() {
    super.onResume();
}

```

Fig 4.15

```

if (auth.getCurrentUser() != null) {
    checkRegID();
    requestLocation();
}

if (auth.getCurrentUser() != null) {
    navigationView.getMenu().findItem(R.id.nav_sign_in).setTitle(auth.getCurrentUser().getDisplayName());
    View navView = navigationView.getHeaderView(0);
    ImageView navUser = (ImageView) navView.findViewById(R.id.imageView);
    TextView txtEmail = (TextView) navView.findViewById(R.id.txtEmail);
    TextView txtName = (TextView) navView.findViewById(R.id.txtName);
    txtEmail.setText(preferences.getString("email", ""));
    txtName.setText(preferences.getString("name", ""));
    txtName.setText(preferences.getString("name", ""));
    try {
        Picasso.with(this).load(preferences.getString("image", "")).into(navUser);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

if (ContextCompat.checkSelfPermission(context, Manifest.permission.ACCESS_FINE_LOCATION) ==
    PackageManager.PERMISSION_GRANTED &&
    ContextCompat.checkSelfPermission(context, Manifest.permission.ACCESS_COARSE_LOCATION) ==
    PackageManager.PERMISSION_GRANTED &&
    (ContextCompat.checkSelfPermission(context, Manifest.permission.WRITE_EXTERNAL_STORAGE) ==
        PackageManager.PERMISSION_GRANTED) &&
    (ContextCompat.checkSelfPermission(context, Manifest.permission.READ_EXTERNAL_STORAGE) ==
        PackageManager.PERMISSION_GRANTED)) {
    locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    if (locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
        locationManager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER, minTime, 0,

```

Fig 4.16

```

    ...
    } else if (locationManager
            .isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
        locationManager.requestLocationUpdates(
                LocationManager.NETWORK_PROVIDER, minTime, 0,
                minDistance, listener, this);
    } else {
        Toast.makeText(getApplicationContext(), text:"Enable Location", Toast.LENGTH_LONG).show();
    }
} else {
    ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission.WRITE_EXTERNAL_STORAGE, Manifest.permission.READ_EXTERNAL_STORAGE}, requestCode: 1);
}
}

public void checkRegID() {
    if (playServicesAvailable()) {
        Log.i(TAG, msg: "onCreate: " + token_id);
        token_id = FirebaseInstanceId.getInstance().getToken();
        if (auth.getCurrentUser() != null & preferences.getBoolean("reg_status", false)) {
            User user = new User(auth.getCurrentUser().getDisplayName(), auth.getUid(), auth.getCurrentUser().getPhotoUrl().toString());
            myRef.child("users").child(auth.getUid()).setValue(user);
            preferences.edit().putString("token_id", token_id).commit();
        }
    }
}

private void playServicesAvailable() {
    GoogleApiAvailability apiAvailability = GoogleApiAvailability.getInstance();
    int resultCode = apiAvailability.isGooglePlayServicesAvailable( context: this );
    if (resultCode != ConnectionResult.SUCCESS) {
        if (apiAvailability.isUserResolvableError(resultCode)) {
            apiAvailability.getErrorDialog( activity: this, resultCode, PLAY_SERVICES_RESOLUTION_REQUEST )
                    .show();
        } else {
            Log.i(TAG, msg: "This device is not supported.");
            finish();
        }
        return false;
    }
    return true;
}

@Override
public void onBackPressed() {
}

```

Fig 4.17

```

    ...
    Log.i(TAG, msg: "onCreate: " + token_id);
} else {
    Log.i(TAG, msg: "sendNotificationToUser: no play services");
    // ... log error, or handle gracefully
}

private static final int PLAY_SERVICES_RESOLUTION_REQUEST = 9000;

private boolean playServicesAvailable() {
    GoogleApiAvailability apiAvailability = GoogleApiAvailability.getInstance();
    int resultCode = apiAvailability.isGooglePlayServicesAvailable( context: this );
    if (resultCode != ConnectionResult.SUCCESS) {
        if (apiAvailability.isUserResolvableError(resultCode)) {
            apiAvailability.getErrorDialog( activity: this, resultCode, PLAY_SERVICES_RESOLUTION_REQUEST )
                    .show();
        } else {
            Log.i(TAG, msg: "This device is not supported.");
            finish();
        }
        return false;
    }
    return true;
}

@Override
public void onBackPressed() {
}

```

Fig 4.18

Fig 4.19

```

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}

private static final int RC_SIGN_IN = 123;

```

Fig 4.20

```

    if (id == R.id.nav_sign_in) {
        FirebaseAuth auth = FirebaseAuth.getInstance();
        if (auth.getCurrentUser() != null) {
            item.setTitle(auth.getCurrentUser().getDisplayName());
            Intent intent = new Intent(getApplicationContext(), SignedInActivity.class);
            startActivity(intent);
            // already signed in
        } else {
            startActivityForResult(
                AuthUI.getInstance().AuthUI.getInstance()
                    .createSignInIntentBuilder()
                    .setTosUrl("https://superapp.example.com/terms-of-service.html")
                    .setPrivacyPolicyUrl("https://superapp.example.com/privacy-policy.html")
                    .setAvailableProviders(
                        Arrays.asList(
                            new AuthUI.IdpConfig.Builder(AuthUI.GOOGLE_PROVIDER).build()
                        )
                    ).build(),
                new AuthUI.IdpConfig.Builder(AuthUI.GOOGLE_PROVIDER).build()
            );
        }
    }
}

```

The screenshot shows the Android Studio interface with the file `MainActivityApp.java` open. The code handles various intents and Firebase authentication logic. A search bar at the bottom contains the placeholder "Type here to search".

```
    RC_SIGN_IN);  
        // not signed in  
    }  
    // Handle the camera action  
} else if (id == R.id.nav_logout) {  
    FirebaseAuth auth = FirebaseAuth.getInstance();  
    if (auth.getCurrentUser() != null) {  
        auth.signOut();  
        preferences.edit().clear().commit();  
        finish();  
        Toast.makeText(context, text "Logout success", Toast.LENGTH_SHORT).show();  
    }  
    else if (id == R.id.nav_search) {  
        FirebaseAuth auth = FirebaseAuth.getInstance();  
        if (auth.getCurrentUser() != null) {  
            Intent intent = new Intent(getApplicationContext(), ViewPav.class);  
            startActivity(intent);  
        }  
    }  
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);  
    drawer.closeDrawer(GravityCompat.START);  
    return true;  
}  
String path;  
String TAG = "MAINACTIVITY";  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    super.onActivityResult(requestCode, resultCode, data);  
}
```

Fig 4.21

The screenshot shows the same `MainActivityApp.java` file as Fig 4.21, but with a specific comment highlighted: `// RC_SIGN_IN is the request code you passed into startActivityForResult(...)`. The code continues to handle sign-in logic.

```
// RC_SIGN_IN is the request code you passed into startActivityForResult(...) when starting the sign in flow.  
if (requestCode == RC_SIGN_IN) {  
    IdpResponse response = IdpResponse.fromResultIntent(data);  
    // Successfully signed in  
    if (resultCode == RESULT_OK) {  
        Intent intent = new Intent(getApplicationContext(), SignedInActivity.class);  
        startActivity(intent);  
        return;  
    } else {  
        // Sign in failed  
        if (response == null) {  
            // User pressed back button  
            showSnackbar("Sign in cancelled");  
            return;  
        }  
        if (response.getErrorCode() == ErrorCode.NO_NETWORK) {  
            showSnackbar("No internet connection");  
            return;  
        }  
        if (response.getErrorCode() == ErrorCode.UNKNOWN_ERROR) {  
            showSnackbar("An unknown error occurred");  
            return;  
        }  
    }  
}
```

Fig 4.22

```

    showSnackbar("Unknown response from AuthUI sign-in");
} else if (requestCode == CAMERA_REQUEST && resultCode == Activity.RESULT_OK) {
    Bitmap photo = (Bitmap) data.getExtras().get("data");
    //imgCalmImage.setImageBitmap(photo);
    byteArrayOutputStream = new ByteArrayOutputStream();
    photo.compress(Bitmap.CompressFormat.JPEG, quality, 100);
    path = MediaStore.Images.Media.insertImage(getApplicationContext(), photo, title, "DA");
    photo.recycle();
}
else {
    super.onActivityResult(requestCode, resultCode, data);
}

/**
 * Calculate distance between two points in latitude and longitude taking
 * into account height difference. If you are not interested in height
 * difference pass 0.0. Uses Haversine method as its base.
 */
public static double distance(double lat1, double lon1, double lat2, double lon2,
                               double el1, double el2) {
    final int R = 6371; // Radius of the earth
    ...
}

```

Fig 4.23

```

    double latDistance = Math.toRadians(lat2 - lat1);
    double lonDistance = Math.toRadians(lon2 - lon1);
    double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
            + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
            * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c * 1000; // convert to meters
    double height = el1 - el2;
    distance = Math.pow(distance, 2) + Math.pow(height, 2);
    return Math.sqrt(distance);
}

public String getRealPathFromURI(Uri uri) {
    Cursor cursor = getContentResolver().query(uri, null, null, null, null);
    cursor.moveToFirst();
    int idx = cursor.getColumnIndex(MediaStore.Images.ImageColumns.DATA);
    return cursor.getString(idx);
}

public void showSnackbar(final int id) {
    Toast.makeText(getApplicationContext(), id, Toast.LENGTH_SHORT).show();
}

Location location = null;

public void requestLocation() {
}

```

Fig 4.24

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TouristGuide". The package structure is visible under "app": com.example.geo_fencing.sys.MainActivityApp.
- Code Editor:** The file "MainActivityApp.java" is open. The code is as follows:

```
Log.i(TAG, msg: "requestLocation: requesting location");

try {
    LocationManager locationManager;
    String context = Context.LOCATION_SERVICE;
    locationManager = (LocationManager) getSystemService(context);
    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    criteria.setAltitudeRequired(false);
    criteria.setBearingRequired(false);
    criteria.setCostAllowed(true);
    criteria.setPowerRequirement(Criteria.POWER_LOW);
    String provider = locationManager.getBestProvider(criteria, enabledOnly: false);
    int rc = ActivityCompat.checkSelfPermission(context, Manifest.permission.ACCESS_FINE_LOCATION);

    if (rc == PackageManager.PERMISSION_GRANTED) {
        location = locationManager.getLastKnownLocation(provider);
        latitude = location.getLatitude();
        longitude = location.getLongitude();

        //seconds and meter
        locationManager.requestLocationUpdates(provider, minTime: 0, minDistance: 0,
                                              locationListener);
    } else {
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

- Run Bar:** The run configuration is set to "C:/Users/sandipta45/Desktop/BCA/Major Project..."
- Bottom Bar:** Includes icons for Logcat, TODO, Terminal, Build, Run, and Gradle build status.
- System Tray:** Shows battery level (11%), network (LF), and system status (4 spaces).

Fig 4.25

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TouristGuide". The "app" module contains a "java" directory with a package named "com.example.geo_fencing.sys". Inside this package, there is a "location" directory containing various classes like LocationListener, AdditizedOverlay, AlertDialogManager, ConnectionDetector, GooglePlaces, GPSTracker, Place, PlaceDetails, PlaceList, Adapter1, ConnectivityReceiver, Constant, FirebaseMessagingService, LocationService, MainActivityApp (which is selected), MapsActivity, MyApplication, MyFirebaseinstanceService, Navigate, Places, SignedinActivity, SinglePlaceActivity, SplashScreen, and User.
- Code Editor:** The code editor displays the "MainActivityApp.java" file. The code defines a class that implements the LocationListener interface. It includes methods for onLocationChanged, onProviderDisabled, onProviderEnabled, and onStatusChanged. It also contains a private method updateWithNewLongitude and a main method MainactivityApp().
- Toolbars and Menus:** The top menu bar includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Bottom Bar:** The bottom bar shows the current file path (C:/Users/sandipta45/Desktop/BCA/Major Project/TouristGuide/app/src/main/java/com/example/geo_fencing/sys/MainActivityApp.java), the build variant (Debug), and the run configuration (Run).
- Logcat:** The bottom right corner shows the logcat output with the message "7873 LF : UTF-8 4 spaces".

Fig 4.26

```

@Override
public void onProviderEnabled(String provider) {
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
}

double latitude = 0.0, longitude = 0.0;

private void updateWithNewLongitude(Location location) {
    // myLocationText = (TextView) findViewById(R.id.myLocationText);
    if (location != null) {
        latitude = location.getLatitude();
        longitude = location.getLongitude();

        // Toast.makeText(context, "Current Loc"+latitude+","+longitude, Toast.LENGTH_SHORT).show();
    }
}

```

Fig 4.27

4.3 The Database Connection

```

package com.example.geo_fencing.sys;
import ...;

public class FirebaseMessagingService extends com.google.firebase.messaging.FirebaseMessagingService {
    String TAG="FCM";
    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        ...
        // TODO(developer): Handle FCM messages here.
        // Not getting messages here? See why this may be: https://goo.gl/39bRNJ
        Log.d(TAG, "msg: " + remoteMessage.getFrom());
        ...
        // Check if message contains a data payload.
        if (remoteMessage.getData().size() > 0) {
            Log.d(TAG, "msg: Message data payload: " + remoteMessage.getData());
        }
        ...
        // Check if message contains a notification payload.
        if (remoteMessage.getNotification() != null) {
            Log.d(TAG, "msg: Message Notification Body: " + remoteMessage.getNotification().getBody());
        }
        ...
        sendNotification(remoteMessage.getData().get("body"));
        ...
        // Also if you intend on generating your own notifications as a result of a received FCM
        // message, here is where that should be initiated. See sendNotification method below.
    }

    private void sendNotification(String messageBody) {
        ...
    }
}

```

Fig 4.28

The screenshot shows the Android Studio interface with the code editor open to the `FirebaseMessagingService.java` file. The code implements a service that handles FCM messages. It includes methods for handling token refreshes and sending notifications. The code uses JSON parsing to extract message body details like title, longitude, and latitude.

```
    sendNotification(remoteMessage.getData().get("body"));

    // Also if you intend on generating your own notifications as a result of a received FCM
    // message, here is where that should be initiated. See sendNotification method below.

    private void sendNotification(String messageBody) {
        JSONObject jsonObject;
        String msg=null,from=null,requester_token=null,status=null,title=null,latitude=null,longitude=null;
        try {
            jsonObject=new JSONObject(messageBody);
            from=jsonObject.getString("from");
            msg=jsonObject.getString("msg");
            title=jsonObject.getString("title");
            requester_token=jsonObject.getString("token_id");
            latitude=jsonObject.getString("latitude");
            longitude=jsonObject.getString("longitude");
            status=jsonObject.getString("status");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onTokenRefresh() {
        String refreshedToken = FirebaseInstanceId.getInstance().getToken();
        Log.d("FCM", "Refreshed token: " + refreshedToken);
        sendRegistrationToServer(refreshedToken);
    }

    private void sendRegistrationToServer(String token) {
        Intent intent = new Intent(this, LocationService.class);
        intent.putExtra("token", token);
        startService(intent);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
    }

    @Override
    protected void onStart() {
        super.onStart();
        FirebaseInstanceId.getInstance().registerForNewToken();
    }

    @Override
    protected void onStop() {
        super.onStop();
        FirebaseInstanceId.getInstance().unregister();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        stopSelf();
    }

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        Log.d("FCM", "From: " + remoteMessage.getFrom());
        Log.d("FCM", "Notification: " + remoteMessage.getNotification());
        Log.d("FCM", "Data: " + remoteMessage.getData());
    }
}
```

Fig 4.29

4.4 Location Service

The screenshot shows the Android Studio interface with the code editor open to the `LocationService.java` file. This service extends the `Service` class and implements the `ConnectivityReceiver` interface. It handles location updates and broadcast intents. The code includes methods for getting the current location and handling service starts.

```
    package com.example.geo_fencing.sys;

    import android.app.Service;
    import android.content.Intent;
    import android.location.Location;
    import android.os.Binder;
    import android.os.IBinder;
    import android.util.Log;
    import android.widget.Toast;

    import androidx.annotation.Nullable;
    import androidx.core.app.ConnectivityReceiver;
    import androidx.core.app.ConnectivityReceiverListener;
    import androidx.localbroadcastmanager.PendingIntent;

    public class LocationService extends Service implements ConnectivityReceiver.ConnectivityReceiverListener {
        String latitude = "0.0", longitude = "0.0";

        private Intent broadcastIntent;
        boolean firstReminder = true;

        @Override
        public void onStart(Intent intent, int startId) {
            super.onStart(intent, startId);

            Log.i("BB", "service started");

            getLatLng();
            context = this;
        }

        @Override
        public IBinder onBind(Intent intent) {
            return null;
        }

        private void getLatLng() {
            LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
            Location location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
            if (location != null) {
                latitude = String.valueOf(location.getLatitude());
                longitude = String.valueOf(location.getLongitude());
            }
        }

        private void context() {
            if (context != null) {
                Intent intent = new Intent("com.example.geo_fencing.sys.LOCATION_UPDATES");
                intent.putExtra("lat", latitude);
                intent.putExtra("long", longitude);
                LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
            }
        }

        @Override
        public void onNetworkConnectionChanged(boolean isConnected) {
            if (isConnected) {
                if (firstReminder) {
                    firstReminder = false;
                    broadcastIntent = new Intent("com.example.geo_fencing.sys.LOCATION_UPDATES");
                    LocalBroadcastManager.getInstance(context).sendBroadcast(broadcastIntent);
                }
            } else {
                broadcastIntent = new Intent("com.example.geo_fencing.sys.LOCATION_UPDATES");
                LocalBroadcastManager.getInstance(context).sendBroadcast(broadcastIntent);
            }
        }
    }
```

Fig 4.30

The screenshot shows the Android Studio interface with the project 'TouristGuide' open. The code editor displays the `LocationService.java` file. The code implements a static method to calculate the distance between two locations using the Haversine formula. It also includes a method to get the current location using the LocationManager.

```
private Context context;
private static double distFrom(double lat1, double lng1, double lat2, double lng2) {
    double earthRadius = 6371000; //meters
    double dLat = Math.toRadians(lat2 - lat1);
    double dLng = Math.toRadians(lng2 - lng1);
    double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
        Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
        Math.sin(dLng / 2) * Math.sin(dLng / 2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double dist = (earthRadius * c);
    return dist;
}
Location location = null;
LocationManager locationManager;
public void getLatLong() {
    try {
        LocationManager locationManager;
        String context = Context.LOCATION_SERVICE;
        locationManager = (LocationManager) getSystemService(context);
        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(true);
        criteria.setPowerRequirement(Criteria.POWER_HIGH);
        String provider = locationManager.getBestProvider(criteria, enabledOnly: true);
        locationManager.requestLocationUpdates(provider, minTime: 0, minDistance: 0,
            locationListener);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
JSONArray jsonArray;
private final LocationListener locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) updateLocation(location);
    @Override
    public void onProviderDisabled(String provider) {
        // updateWithNewLatitude(null);
        // updateWithNewLongitude(null);
    }
    @Override
    public void onProviderEnabled(String provider) {
    }
}
```

Fig 4.31

The screenshot shows the Android Studio interface with the project 'TouristGuide' open. The code editor displays the `LocationService.java` file. The code includes logic to check for the `ACCESS_FINE_LOCATION` permission and to handle location updates using a LocationListener.

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setCostAllowed(true);
criteria.setPowerRequirement(Criteria.POWER_HIGH);
String provider = locationManager.getBestProvider(criteria, enabledOnly: true);
checkCallingPermission(Manifest.permission.ACCESS_FINE_LOCATION);
// seconds and meter
locationManager.requestLocationUpdates(provider, minTime: 0, minDistance: 0,
    locationListener);
} catch (Exception e) {
    e.printStackTrace();
}
}
JSONArray jsonArray;
private final LocationListener locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) updateLocation(location);
    @Override
    public void onProviderDisabled(String provider) {
        // updateWithNewLatitude(null);
        // updateWithNewLongitude(null);
    }
    @Override
    public void onProviderEnabled(String provider) {
    }
}
```

Fig 4.32

Fig 4.33

The screenshot shows the Android Studio interface with the following details:

- Project Structure:** The project is named "TouristGuide". The package structure is com.example.geo_fencing.sys.LocationService.
- Code Editor:** The file "LocationService.java" is open. The code implements a service that sends broadcast intents based on location changes and user proximity to specific places.
- Code Snippet:**

```
editor.commit();

Log.i( tag: "Lat", msg: latitude + "" + longitude);

context.sendBroadcast(broadcastIntent);

try {
    jsonArray = new JSONArray(preferences.getString( $0 "cart_items", $1 null));
    int count = 0;

    while (count < jsonArray.length()) {
        JSONObject jsonObject=jsonArray.getJSONObject(count);

        double distance = distFrom(jsonObject.getDouble("latitude"),
            Double.parseDouble(jsonObject.getString("longitude")),
            location.getLatitude(), location.getLongitude());

        Log.i( tag: "distance", msg: "distance: "+distance);
        if (firstReminder && distance <= 1000) {

            Intent i = new Intent(context, SinglePlaceActivity.class);
            i.putExtra( name: "speak", value: You are near "+jsonObject.getString( name: "name"));
            i.putExtra( name: "description", jsonObject.getString( name: "desc"));
            i.putExtra( name: "reference", jsonObject.getString( name: "id"));
            PendingIntent resultPendingIntent =
                PendingIntent.getActivity(
                    context,
                    requestCode: 0,
                    i,
                    PendingIntent.FLAG_ONE_SHOT);
            long pattern[] = {1000, 100, 1000};

```
- Toolbars and Status Bar:** Standard Android Studio toolbars and status bar are visible at the bottom, showing the build time and system information.

Fig 4.34

```

181     NotificationCompat.Builder mBuilder =
182             new NotificationCompat.Builder(context)
183                 .setSmallIcon(R.mipmap.ic_launcher)
184                 .setContentTitle("You are near your fav place")
185                 .setContentIntent(resultPendingIntent)
186                 .setVibrate(pattern)
187                 .setContentText("You are near " + jsonObject.getString("name"));
188         // Because clicking the notification opens a new ("special") activity, there's
189         // no need to create an artificial back stack.
190
191         NotificationManager mNotifyMgr =
192             (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
193
194         mNotifyMgr.notify(NOTIFY_ME_ID, mBuilder.build());
195         // to close the first reminder
196         firstReminder = false;
197     }
198
199     count++;
200
201     } catch (Exception e) {
202         e.printStackTrace();
203     }
204     Log.i( tag, "ONCTS", msg: "broadcast sent ");
205
206 }
207
208 }
209
210 }
211

```

LocationService

Fig 4.35

```

212
213     @Override
214     public void onNetworkConnectionChanged(boolean isConnected) {
215         // TODO Auto-generated method stub
216     }
217
218     /**
219      * Calculate distance between two points in latitude and longitude taking
220      * into account height difference. If you are not interested in height
221      * difference pass 0.0. Uses Haversine method as its base.
222      *
223      * @lat1, @lon1 Start point lat1, lon2 End point ell Start altitude in meters
224      * @ell1 End altitude in meters
225      * @returns Distance in Meters
226      */
227     public double distance(double lat1, double lat2, double lon1,
228                           double lon2, double ell1, double ell2) {
229
230         final int R = 6371; // Radius of the earth
231
232         Double latDistance = Math.toRadians(lat2 - lat1);
233         Double lonDistance = Math.toRadians(lon2 - lon1);
234         Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
235             + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
236             * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
237         Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
238         double distance = R * c * 1000; // convert to meters
239
240         double height = ell1 - ell2;
241
242     }
243
244 }
245
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311

```

LocationService

Fig 4.36

```

public double distance(double lat1, double lon1, double lat2, double lon2, double ell, double e12) {
    final int R = 6371; // Radius of the earth
    Double latDistance = Math.toRadians(lat2 - lat1);
    Double lonDistance = Math.toRadians(lon2 - lon1);
    Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
            + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
            * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
    Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double distance = R * c * 1000; // convert to meters
    double height = ell - e12;
    distance = Math.pow(distance, 2) + Math.pow(height, 2);
    return Math.sqrt(distance);
}

```

Fig 4.37

4.5 Navigate Service

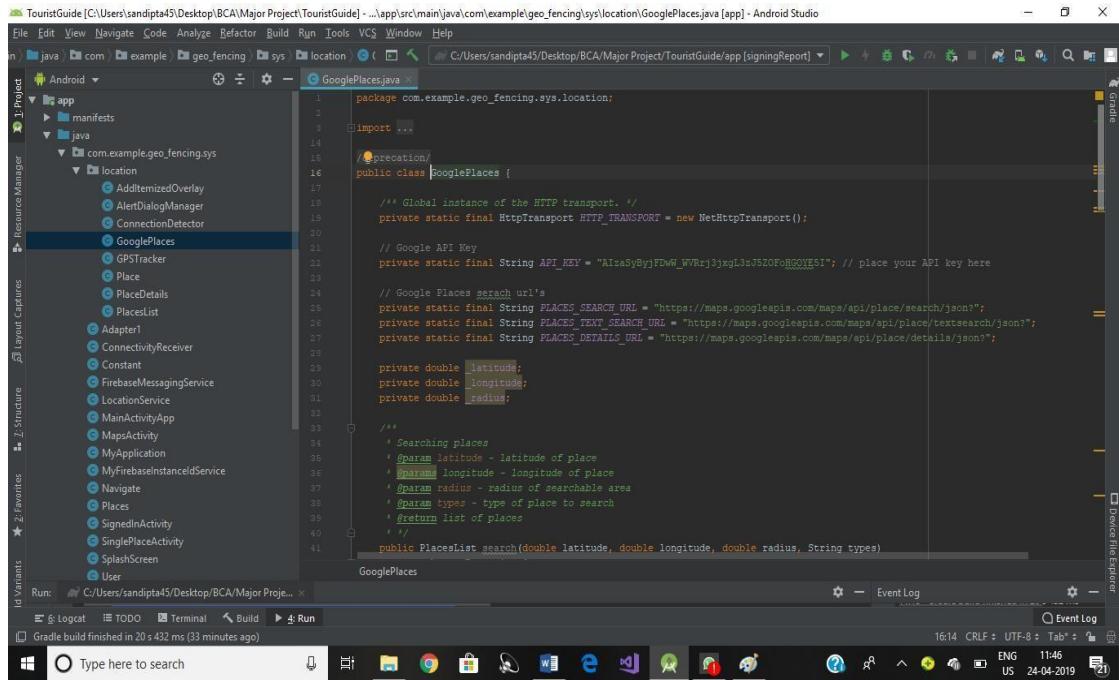
```

package com.example.geo_fencing.sys;
import ...
public class Navigate extends Activity {
    SharedPreferences preferences;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent intent = getIntent();
        String strlat=preferences.getString("latitude","");
        String strlon=preferences.getString("longitude","");
        String strtolat1.getStringExtra("toLat");
        String strtolon1.getStringExtra("toLon");
        String url = "http://maps.google.com/maps?&addr=" + strlat + "," + strlon + "&addr=" + strtolat1 + "," + strtolon1;
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        intent.setComponent(new ComponentName("com.google.android.apps.maps",
                "com.google.android.maps.MapActivity"));
        startActivity(intent);
    }
}

```

Fig 4.38

4.6 The Google Services

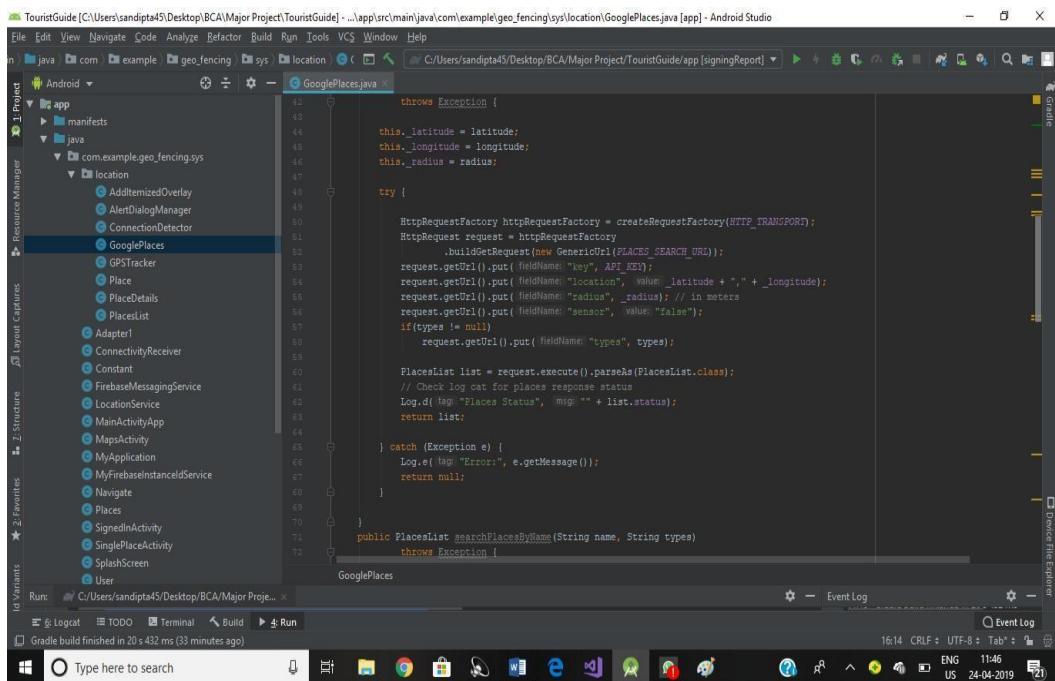


```

package com.example.geo_fencing.sys.location;
import ...;
public class GooglePlaces {
    private static final String API_KEY = "AIzaSyByFDwW_WVRrj3jxgL3s5ZOFoHGOVE5I"; // place your API key here
    private static final String PLACES_SEARCH_URL = "https://maps.googleapis.com/maps/api/place/search/json?";
    private static final String PLACES_TEXT_SEARCH_URL = "https://maps.googleapis.com/maps/api/place/textsearch/json?";
    private static final String PLACES_DETAILS_URL = "https://maps.googleapis.com/maps/api/place/details/json?";
    private double _latitude;
    private double _longitude;
    private double _radius;
    public PlacesList search(double latitude, double longitude, double radius, String types) {
        ...
    }
}

```

Fig 4.39



```

throws Exception {
    this._latitude = latitude;
    this._longitude = longitude;
    this._radius = radius;
    try {
        HttpRequestFactory httpRequestFactory = createRequestFactory(HTTP_TRANSPORT);
        HttpRequest request = httpRequestFactory
            .buildGetRequest(new GenericUrl(PLACES_SEARCH_URL));
        request.getUri().put("key", API_KEY);
        request.getUri().put("location", value(_latitude + "," + _longitude));
        request.getUri().put("radius", _radius); // in meters
        request.getUri().put("sensor", value("false"));
        if(types != null)
            request.getUri().put("types", types);
        PlacesList list = request.execute().parseAs(PlacesList.class);
        // Check log cat for places response status
        Log.d(TAG, "Places Status", msg "" + list.getStatus());
        return list;
    } catch (Exception e) {
        Log.e(TAG, "Error", e.getMessage());
        return null;
    }
}
public PlacesList searchPlacesByName(String name, String types)
    throws Exception {
}

```

Fig 4.40

The screenshot shows the Android Studio interface with the code editor open to the `GooglePlaces.java` file. The code implements a `PlacesList` interface, handling place search requests. It uses a `HttpRequestFactory` to build a request for the `PLACES_TEXT_SEARCH_URL`. The `getPlaceDetails` method retrieves details for a specific place using the `PLACES_DETAILS_URL`.

```
try {
    Log.i( tag: "Google place", msg: "searchPlacesByName: "+PLACES_TEXT_SEARCH_URL+"query='"+name+"&types="+types+"&key="+API_KEY);
    HttpRequestFactory httpRequestFactory = createRequestFactory(HTTP_TRANSPORT);
    HttpRequest request = httpRequestFactory
        .buildGetRequest(new GenericUrl(PLACES_TEXT_SEARCH_URL));
    request.getUrl().put( fieldName: "key", API_KEY );
    request.getUrl().put( fieldName: "query", name );

    request.getUrl().put( fieldName: "sensor", value: "false");
    if(types != null)
        request.getUrl().put( fieldName: "types", types);

    PlacesList list = request.execute().parseAs(PlacesList.class);
    // Check log cat for places response status
    Log.d( tag: "Places Status", msg: "" + list.getStatus());
    return list;
} catch (Exception e) {
    Log.e( tag: "Error:", e.getMessage());
    return null;
}

public PlaceDetails getPlaceDetails(String reference) throws Exception {
    try {
        Log.i( tag: "GOOGLE", msg: "getPlaceDetails: "+request.getUrl().toString());
        PlaceDetails place = request.execute().parseAs(PlaceDetails.class);

        return place;
    } catch (Exception e) {
        Log.e( tag: "Error inDetails", e.getMessage());
        throw e;
    }
}

// Creating http request Factory
private static HttpRequestFactory createRequestFactory() {
    final HttpTransport transport = return transport.createRequestFactory((request) -> {
        GoogleHeaders headers = new GoogleHeaders();
        headers.setApplicationName("Androidhive-Places-Test");
        request.setHeaders(headers);
        JsonHttpResponseParser parser = new JsonHttpResponseParser(new JacksonFactory());
        request.addParser(parser);
    });
}
```

Fig 4.41

The screenshot shows the same Android Studio interface as Fig 4.41, but with additional code added to the `getPlaceDetails` method. This code creates an `HttpRequestFactory` and returns it from a static method. The `createRequestFactory` method initializes a `HttpTransport` and creates a `RequestFactory` with a `JsonHttpResponseParser`.

```
HttpRequestFactory httpRequestFactory = createRequestFactory(HTTP_TRANSPORT);
HttpRequest request = httpRequestFactory
    .buildGetRequest(new GenericUrl(PLACES_DETAILS_URL));
request.getUrl().put( fieldName: "key", API_KEY );
request.getUrl().put( fieldName: "reference", reference);
request.getUrl().put( fieldName: "sensor", value: "false");
Log.i( tag: "GOOGLE", msg: "getPlaceDetails: "+request.getUrl().toString());
PlaceDetails place = request.execute().parseAs(PlaceDetails.class);

return place;

} catch (Exception e) {
    Log.e( tag: "Error inDetails", e.getMessage());
    throw e;
}

// Creating http request Factory
private static HttpRequestFactory createRequestFactory() {
    final HttpTransport transport = return transport.createRequestFactory((request) -> {
        GoogleHeaders headers = new GoogleHeaders();
        headers.setApplicationName("Androidhive-Places-Test");
        request.setHeaders(headers);
        JsonHttpResponseParser parser = new JsonHttpResponseParser(new JacksonFactory());
        request.addParser(parser);
    });
}
```

Fig 4.42

The screenshot shows the Android Studio interface with the file `GooglePlaces.java` open in the editor. The code implements a `PlacesClient` interface, handling place details requests. It includes exception handling for network errors and a static method to create an `HttpRequestFactory`. The code uses `OkHttp` for networking and `Jackson` for JSON parsing.

```
1 package com.example.geo_fencing.sys.location;
2
3 import ...
4
5 public class GooglePlaces {
6
7     public void getPlaceDetails(PlaceRequest request, PlacesClient_placesCallback callback) {
8         PlaceDetails place = request.execute();
9         if (place != null) {
10             callback.onSuccess(place);
11         } else {
12             callback.onError(new IOException("No place found"));
13         }
14     }
15
16     public void getPlaceDetails(PlaceRequest request, PlacesClient_placesCallback callback) {
17         try {
18             PlaceDetails place = request.execute();
19             callback.onSuccess(place);
20         } catch (IOException e) {
21             Log.e("Error inDetails", e.getMessage());
22             callback.onError(e);
23         }
24     }
25
26     /**
27      * Creating http request Factory
28      */
29     public static HttpRequestFactory createRequestFactory() {
30         final OkHttpClient transport = OkHttpTransport.create();
31         return transport.createRequestFactory();
32         final Headers headers = new Headers();
33         headers.setApplicationName("Androidhive-Places-Test");
34         headers.setHeaders(headers);
35         final JsonHttpParser parser = new JsonHttpParser(new JacksonFactory());
36         request.addParser(parser);
37     }
38 }
```

Fig 4.43

4.7 Connection Detector

The screenshot shows the Android Studio interface with the file `ConnectionDetector.java` open in the editor. The code defines a `ConnectionDetector` class that checks for internet connectivity using the `ConnectivityManager` API. It iterates through all network info to find a connected provider.

```
1 package com.example.geo_fencing.sys.location;
2
3 import ...
4
5 public class ConnectionDetector {
6
7     private Context _context;
8
9     public ConnectionDetector(Context context) { this._context = context; }
10
11     /**
12      * Checking for all possible internet providers
13      */
14     public boolean isConnectingToInternet() {
15         ConnectivityManager connectivity = (ConnectivityManager) _context.getSystemService(Context.CONNECTIVITY_SERVICE);
16         if (connectivity != null) {
17             NetworkInfo[] info = connectivity.getAllNetworkInfo();
18             if (info != null) {
19                 for (int i = 0; i < info.length; i++) {
20                     if (info[i].getState() == NetworkInfo.State.CONNECTED)
21                         return true;
22                 }
23             }
24         }
25         return false;
26     }
27 }
```

Fig 4.44

CHAPTER 5: RESULT AND DISCUSSION

5.1 Snapshots/Results



Fig 5.1 : The Main Screen

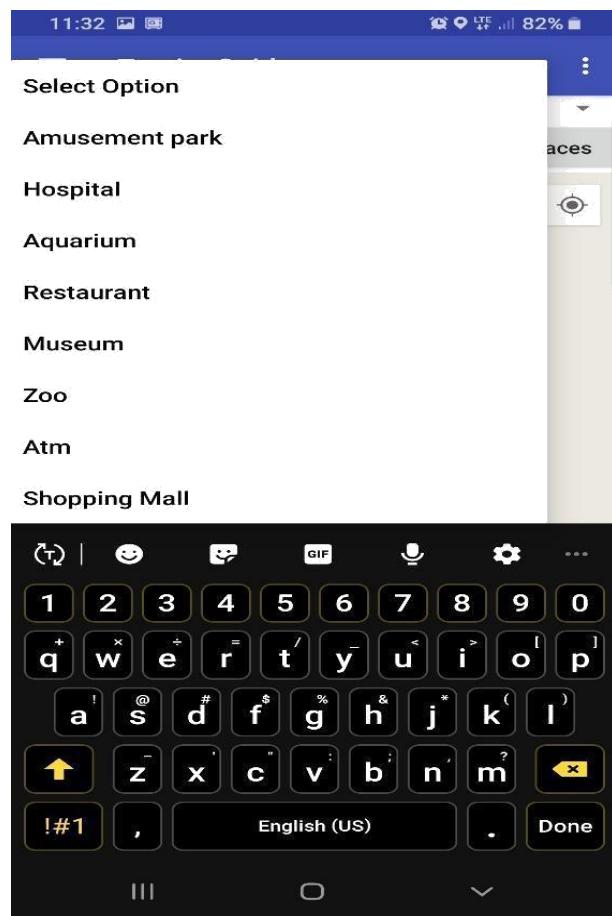


Fig 5.2: Selection of Place

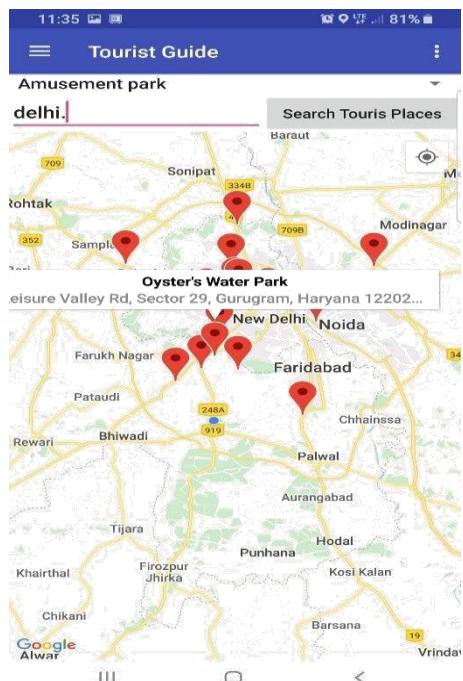


Fig. 5.3: Search Tourist Place

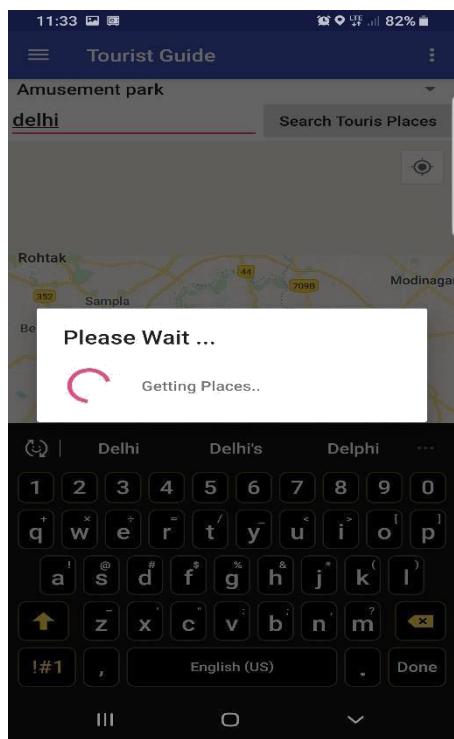


Fig 5.4: Processing

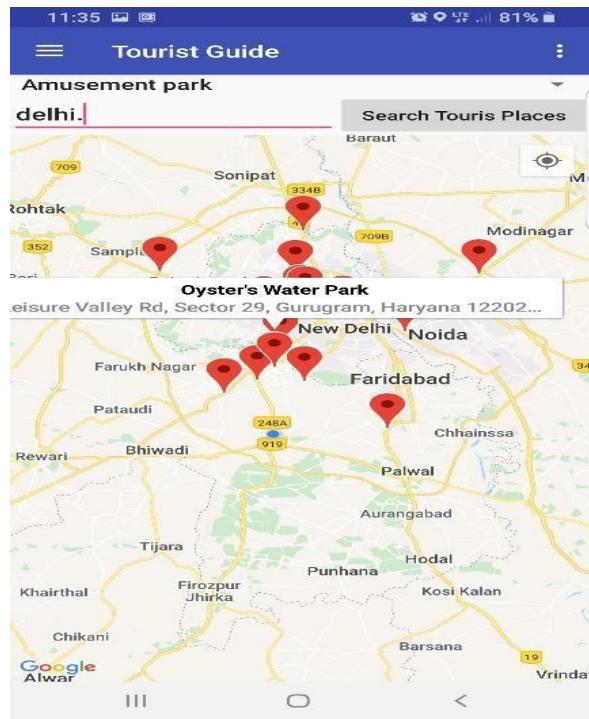


Fig 5.5 Select A Place

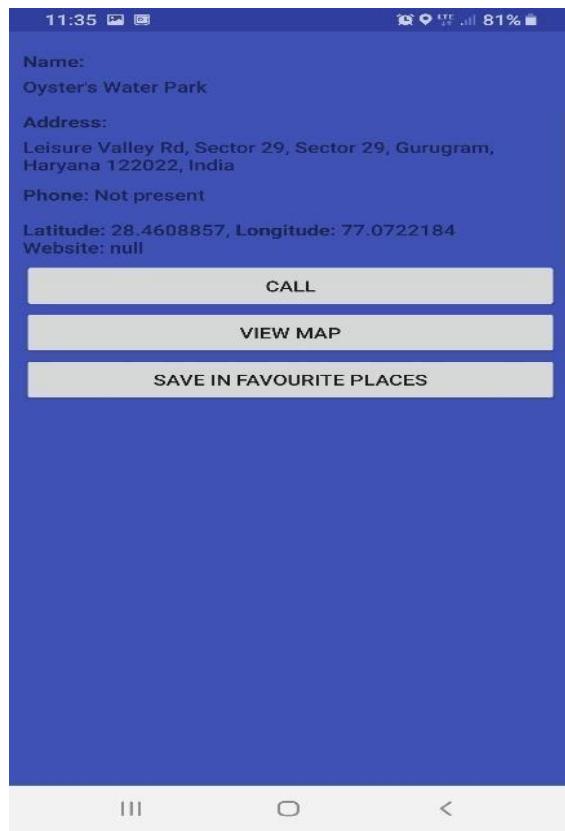


Fig 5.6: Detail of Place

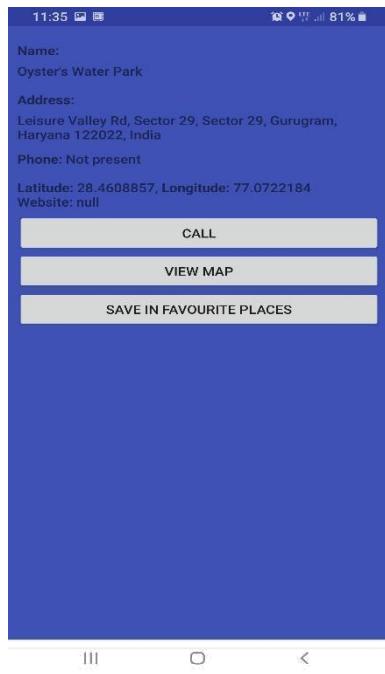


Fig 5.7: Navigation of The Place

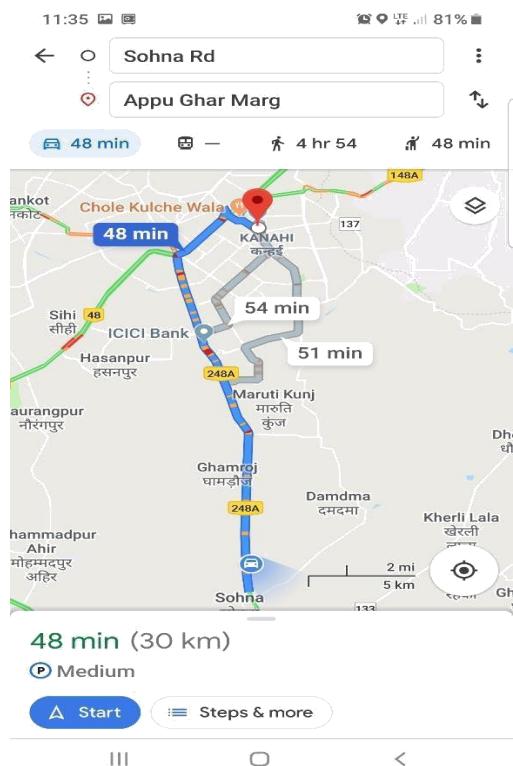


Fig 5.8: Map Google Location

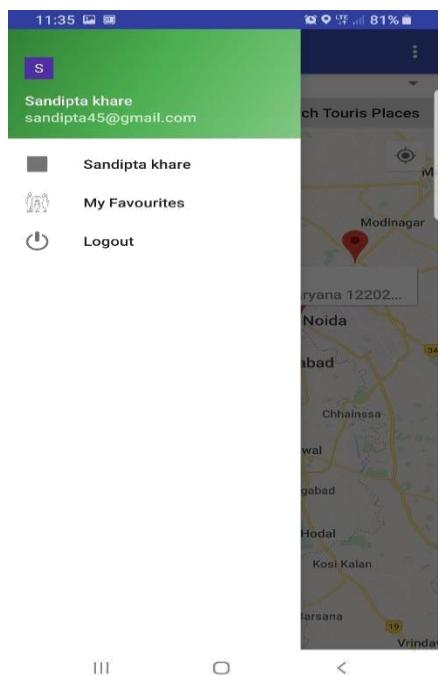


Fig 5.8: Account



Fig 5.9: Database

5.2 Discussion

Our project is on “Tourist Guide” Application which is built on the Android Studio. The above Screen Shots are the output of the Tourist Guide Application. The first Screen Shot, **Fig 5.1**, shows the home page of the Tourist Guide Application. The Home Page contains few things and that are Select Option, Location, Search Tourist Places and Maps. Select Option contains options such as Aquarium, Zoo, Amusement Park, Restaurant, etc. Location is to select place where you must visit. Search Tourist Place means to search place.

In **Fig 5.2**, you would see that we are selecting the Select Option and that is Amusement park.

In **Fig 5.3**, you would see that we are selection the location and that is Delhi. After selecting the location, we would click on Search Tourist Place.

In **Fig 5.4**, you would see that Application is searching for the Location.

In **Fig 5.5**, you would see that there would be various Amusement Park at Delhi. We had selected the Oyster’s Water Park.

In **Fig 5.6**, you would see that the Screen Shot would be showing the details of Oyster’s Water Park.

In **Fig 5.7**, you would see google map which would be showing the route for Oyster’s Water Park from your location. The google map would be open only when a user wants to find the location of the Oyster’s Water Park.

In **Fig 5.8**, you would see that there are three things and that are first is login id, second is Favorite and third is logout.

In **Fig 5.9**, you would see some places name which are stored as favourite. It only happens when the users like place and they click on favourite Button.

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

It is a great way for people to spend their money and for cities to attract other people. In doing so, cities across the world attract millions of dollars just so foreigners get the chance to experience life in a new place. I hope to do much travelling myself because I believe it to be a wonderful way of living life to the fullest. Experiencing culture, religion, foods, sports, entertainment shows, and general ways of life in a new place is a great way to live life to the fullest. I one day hope I could open a hotel chain myself and attract many visitors to new cities. I want to travel the world and once I am free of the responsibility of raising children I want to spend two consecutive years just travelling the world. The biggest plus point for Tourist Guide is it would help anyone anywhere and at any time.

6.2 Future Scope

Application also causes income growth that has its own secondary impact on jobs. These indirect and induced effects could result in an increase in total employment by up to 8 times during the period 2014 to 2016. At its best, the Application economy could generate over “6, 00,000 jobs,” ICRIER Director & Chief Executive Rajat Kathuria said.

Function we can add in future to our application

- SOS function by just triple click on lock button that will send current location to the choices of three number a client want
- Enhanced GUI experience
- More user friendly

REFERENCES

- [1] <https://www.goodreads.com/quotes/tag/tourism>
- [2] <https://www.youtube.com/watch?v=ZLNO2c7nqjw>
- [3] <https://www.tutorialspoint.com/android/>