A Selective and Restrictive Communication Model Based Mobile Application

A Project Report Submitted in partial fulfillment of the degree of

Master of Computer Applications

By
SHAWETA KUMARI (21MCMC08)
MANVENDER TOPPO (21MCMC19)

School of Computer and Information Sciences
University of Hyderabad, Gachibowli Hyderabad – 500046, India

MAY, 2023

CERTIFICATE

This is to certify that the project work entitled **"A Selective and restrictive communication model based mobile application"** being submitted by Shaweta Kumari, bearing Reg. No. 21MCMC08 and Manvender Toppo, bearing Reg. No. 21MCMC19, in partial fulfilment of the requirements for the award of Master of Computer Applications, to the University of Hyderabad, is a bonafide work carried out by them under my Supervision.
The matter in this has not been submitted to any other University or Institution for the award of any degree or diploma**.**

Prof. Siba K Udgata

School of Computer and Information Sciences,
University of Hyderabad

Prof. Atul Negi
Dean,

School of Computer and Information Sciences,

University of Hyderabad

DECLARATION

We, **Shaweta Kumari** and **Manvender Toppo** hereby declare that this dissertation entitled "**A Selective and Restrictive Communication Model Based Mobile Application**" submitted by us under the guidance and supervision of **Dr. Siba Kumar Udgata** is a bonafide work. We also declare that it has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Date:

Shaweta Kumari
Reg. No.: 21MCMC08

Signature of the Student

Manvender Toppo
Reg. No.: 21MCMC19

Signature of the Student

Acknowledgments

We have great pleasure in acknowledging my sincere gratitude to our guide **Prof. Siba Kumar Udgata**, School of Computer Information Sciences for his encouragement, co-operation, noble ideas, guidance, and endless support during this project.
We thank to Dean of School **Prof Atul Ne**gi for his co-opration in completion of the project work. We thank **Prof Alok Singh**, **Prof Rajeev Wankar**, **Dr Sai Prasad P.S.V.S. and Dr Saifulla Md.Abdul** for showing the interest in our project and providing valuable comments during the mid-term presentation.
We take this opportunity to express our gratitude to our family members and dear friends for their constant encouragement and moral support.

Shaweta Kumari & Manvender Toppo

Abstract

Developing the android application “A Selective and restrictive communication model based mobile application to enable IoT device communications”. The arrival of smartphones and intelligent mapping platforms has put location awareness on human fingertips. It is an objective of the present work to enable restrictive communications between selective and authorized users. We aim to provide a location retrieval users on demand without intervention of the user using a specially developed IoT device.

In this work, we develop a smart phone based mobile App as a supplement to an IoT device. This App contact another device (embedded with a Email Id) through its Email Id. This App will contact only one Email Id initially and we are also the feature of increasing it to multiple phones/devices .

Requirement Specifications for Mobile App: Registration (While setting up the App)

- Name
- Mobile Number

Sending a message to the Email Id

- Capability to receive a message from the predefined Email Id and decode the message which normally consists of an URL of the locations Show the location of the device (contact Email Id) on Google Maps Storing the log of interactions with the predefined Email Id

Contents

Chapter 1

Introduction

The selective and restrictive communication model-based mobile application is designed to give users the ability to manage their communication channels effectively. It offers features that allow users to control who can contact them, the type of information they can share, and the level of access they have to their personal data.

Key Features:

1. **Real-time Tracking:** The location tracing application offers real-time tracking capabilities, allowing users to monitor the exact location of the target on a map. This feature provides up-to-date information and enables users to track the movement and progress of individuals or assets in real-time**.**
2. **Privacy and Security:** Location tracing applications prioritize user privacy and security. They often implement secure authentication methods, encryption of location data, and user-controlled access permissions. This ensures that only authorized individuals have access to the tracked location information, promoting privacy and data protection.
3. **History and Playback:** The application stores historical location data, enabling users to review past movements and routes. Users can access this information through the application's interface and playback the tracked locations on the map, giving them a comprehensive view of the target's activities over a specific period.

Benefits:

Enhanced Safety: Location tracing applications can contribute to personal safety, particularly for individuals in potentially risky situations or vulnerable groups. It enables quick response in emergencies, helps locate lost or missing individuals, and enhances overall security.

What is Android?

Android is a Linux-based operating system for mobile devices such as smart phones and tablet computers. It is developed by the open Handset Alliance led by Google. The Android platform includes an operating system based upon the Linux kernel, a GUI, a web browser and end-user applications that can be downloaded. Although the initial demonstrations of Android featured a generic QWERTY smartphone and large VGA screen, the operating system was written to run on relatively inexpensive handsets with conventional numeric keypads.
Android was released under the Apache v2 open-source license; this allows for many variations of the OS to be developed for other devices, such as gaming consoles and digital cameras. Android is based on open-source software, but most Android devices come preinstalled with a suite of proprietary software, such as Google Maps, YouTube, Google Chrome, and Gmail.

Android Development Requirements

1. **JAVA:** Java is the programming language that underpins all Android development. For those who have gained most of their programming experience in languages like JavaScript and Ruby, there can be a learning curve when picking up the Java programming language for the first time.

2. **UNDERSTANDING OF XML:** XML was created as a standard way to encode data for internet-based mobile applications. It is a structured markup language, sharing many features in common with HTML – you may recognize the angled brackets, the <opening> and </closing> tag types, and the deep nesting of

3. **ANDROID SDK:** SDK stands for Software Development Kit, which, though it may conjure up images of a briefcase full of spy tools, is just a fancy name for a set of pre-packaged code. The Android SDKs are modules of Java code that give developers access to mobile device functions like the camera and accelerometer.

4. **ANDROID STUDIO:** The integrated development environment (IDE) of choice for Android developers is called Android Studio. Android Studio is built on top of the well-respected IntelliJ IDE, and it comes with great out-of-the-box support for many of the most common Android SDKs.

5. **FIREBASE:** Firebase is a Backend-as-a-Service (BaaS) which started as a YC11 startup. It grew up into a next-generation app-development platform on Google Cloud Platform. Firebase (a NoSQLjSON database) is a real-time database that allows storing a list of objects in the form of a tree. We can synchronize data between different devices.

ROOT CONTENTS

- AndroidManifest.xml - Which is an XML file describing the application being built and what components – activities services etc –are being supplied by that application.
- Build .xml-which is an Ant script for compiling the application and installing it on the device.
- bin/ - which hold the application once it is complied.
- Src/ - which hold the java source code for the application.
- Res/- which holds the resources, such as icons, GUI layouts, and the like, that get packaged with the compiled java in the application.
- Assets/- Which holds other static files we wish packaged with the application for deployment onto the device.

Android Design and Architecture:

Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvik dex-code which is usually translated from java byte code. Android's kernel based on Linux. The main hardware platform for Android is the ARM architecture. There is support for x86 from the Android x86 project. The architecture for android is given in fig in the next page.

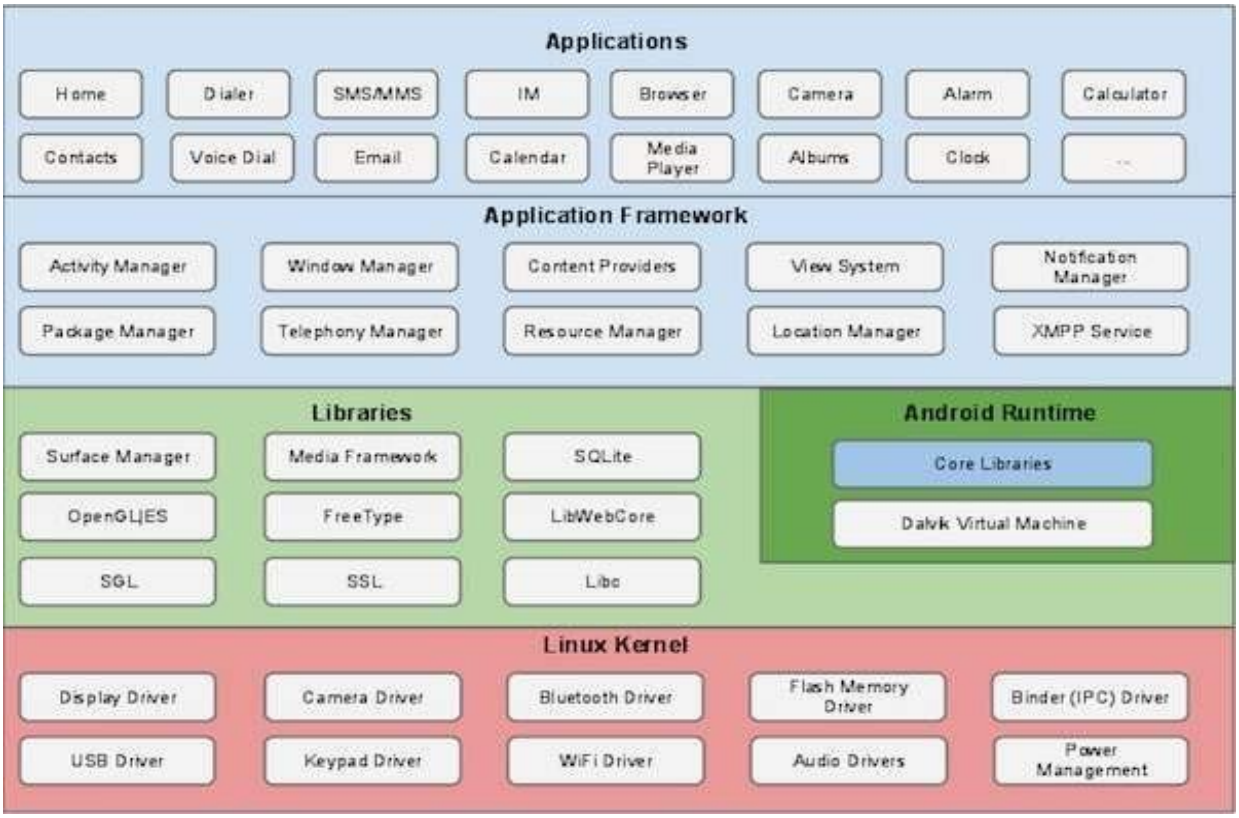


Fig: Android SDK overview

Installing platform for Android project on Android Studio

Before we start, we should already have the SDK as well as ADT plugin installed on our Android Studio. To run our application, we need to install at least one Android platform in our SDK environment. If we have not already performed this step, we need to do it now.
To install a platform in Android Studio

1. In the Android SDK Manager, choose Available Package in the left panel.
2. In the right panel, expand the Android Repository list to display the components available for installation
3. Select at least one platform to install and click Install selected. If you are not sure which platform to install, use the latest.

To run our application in the Android Emulator. Before we can launch the Emulator, we must create an Android Virtual Device (AVD). An AVD defines the system image and device setting used by the emulator, to create an AVD:-

1. In Android Studio, select Window > AVD Manager.
2. Select Virtual Devices in the left panel.
3. Click New. The Create new AVD dialog appears.
4. Type the name of the AVD, such as “My AVD”.
5. Choose a target: The target is the platform (i.e., the version of the Android SDK, such as 2.3.3) you want to run on the Emulator. For this application, we chose the latest platform that we have installed and ignore the rest of the fields.
   Click create AVD. After all this discussion we must imagine how emulator looks

Like:

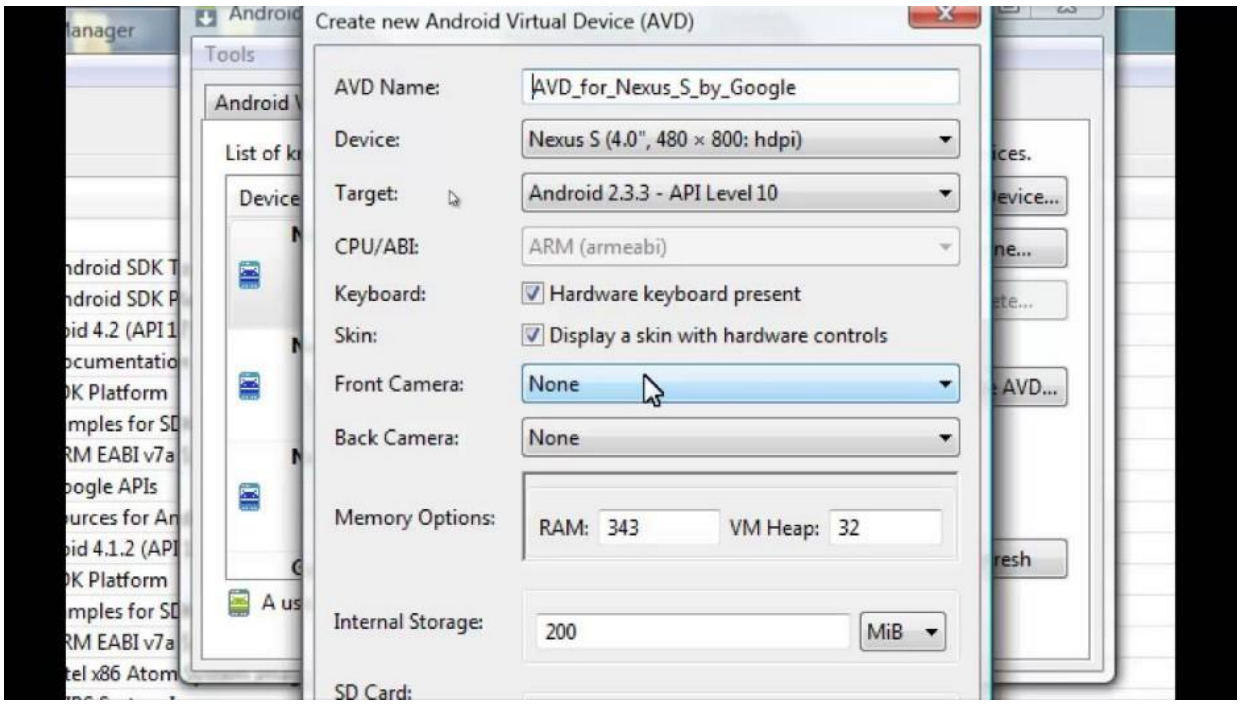
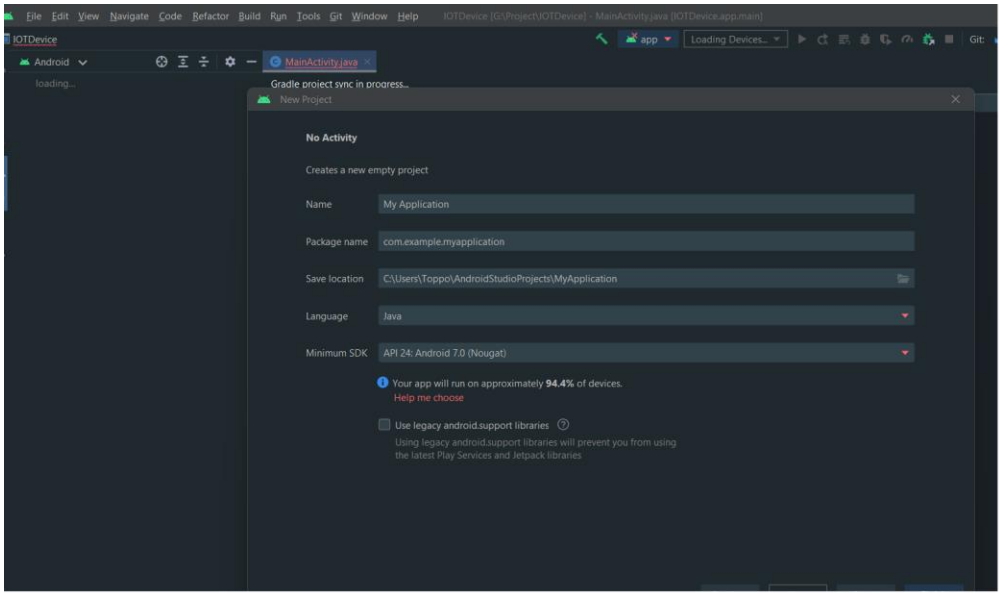


Fig: Android virtual device

Creating new Android project

After we have create an AVD we can move to the next step and state a new Android project with Android Studio.

1. In Android Studio, Select File>NEW>PROJECT…
   If the ADT plug in for Android Studio has been successfully installed, the resulting dialog should have a folder labeled “Android” which should contain “Android project”. (After you create one or more Android project and entry for “Android XML File” will also be available.)
2. Select “Android Project” and click next. A window wizard to create android project.



3. Fill in the project details with the following values: -

   Project name: This is Android Studio name- the name of the directory that contains the project files.

   Build Target: This is the version of the Android SDK that we are using to build our application. For example, if we chose Android 2.1, our application will be compiled against the 2.1 platform library. The target we chose here does not have to match the target we chose for our AVD; however, the target

must be equal to or lower than the target you chose for our AVD. Android applications are forward-compatible, which means an application will run on the platform against which it is built as well as all platforms that are released in the future. For example, an application that is built against the 2.1 platform library will run normally on an AVD or device that is running the 2.3.3. The reverse is not true. In short select a platform version that is equal to or lower than the target we chose for our AVD.

Application name: This is the human-readable title for our application – the name that appears on the Android device.

Package name: com.map this is the package namespace that we want all our source code to reside under. This also sets the package name must be unique across all packages installed on the android system; for this reason, it's important to use a standard domain-style package for our applications. The example above uses the "com.example" namespace, which is a namespace reserved for example documentation – when we develop our own applications. We should use a namespace that is appropriate to our organization or entity

Create Activity: This is the name for the class stub that is generated by the plug in. This is a subclass of Android's activity class. An Activity is simply a class that can run and do work. It can create a UI if it chooses, but it does not need to. As the checkbox suggests, this is optional, but an Activity is almost always used as the basis for an application.

Min SDK Version: This value specifies the minimum API Level on which our application will run. The Min SDK version should be the same as the build target we chose. For example, if the build target is Android then the min SDKversion should be 7 or lower for more information, see android API Levels.

Run the Application

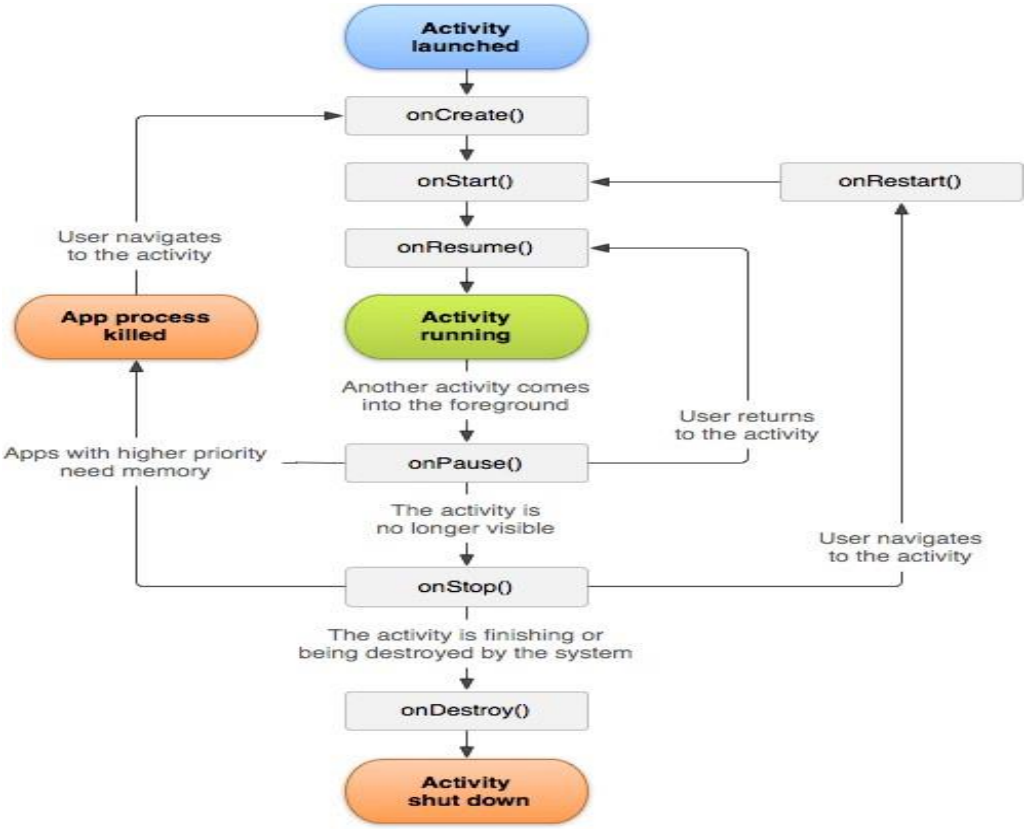
The Android Studio in makes it easy to run our applications:

1. Select Run > Run.
2. Select “Android Application”.

The Android Studio plug in automatically creates a new run configuration for Our project and then launches the Android Emulator. Depending on our environment, the Android emulator might take several minutes to boot fully, so please be patient. When the emulator is booted, the Android Studio in installs our application and launches the default Activity.

Android Life Cycle

A general overview of the lifecycle of an application is shown in the following figure, which is taken from the documentation for Activity.



Notice that when another activity comes to the foreground it does not generally Mean that task originally in the foreground is discarded. The seven methods contained in square boxes in the preceding diagram are called lifecycle method because they govern the lifecycles of Android applications. Their properties are

The Three Lives of Android

It is useful to think of an application in Android having three "lifetimes" associated with the preceding diagram and table (see the documentation of Activityfor a more thorough discussion).

1. The Entire Lifetime: the period between the first call to onCreate() to a single final call to onDestroy() . We may think of this as the time between setting up the initial global state for the app in onCreate() and the release of all resources associated with the app in onDestroy() .
2. The Visible Lifetime The period between a call to onStart() until a corresponding call to onStop() . Although this is termed the "visible lifetime", the app may not be directly visible and interacting with the user at any one time if it is not in the foreground. The feature that distinguishes this lifetime is that, even if not in the foreground, the app maintains resources such that it can instantaneously return to the foreground.

   The Foreground Lifetime The period between a call to onResume() until a corresponding call to onPause() . During foreground lifetime the activity is in front of all other activities and interacting with the user.

NOTE: From these definitions that in general more than one app may be in its visible lifetime at a particular time, though only one of these apps will be in the foreground and literally visible and interacting with the user at any one time.
Because multiple apps may be in their "visible lifetime" at a given time, we may expect that their corresponding and methods could be invoked often if they are moved in and out of the foreground by user actions (for example, a user jumping among a calendar, a weather app, a news app, texting, a contacts list, an incoming phone call, and so on). Thus, we should endeavor to keep the code in these methods as lightweight as possible.

Android Security

Android incorporates industry-leading security features and works with developers and

device implementers to keep the Android platform and ecosystem safe. A robust security model is essential to enable a vigorous ecosystem of apps and devices built on and around the Android platform and supported by cloud services. As a result, through its entire development lifecycle, Android has been subject to a rigorous security program.

Android is designed to be open: Android apps use advanced hardware and software, as well as local and served data, exposed through the platform to bring innovation and value to consumers. To realize that value, the platform offers an app environment that protects the confidentiality, integrity, and availability of users, data, apps, the device, and the network.
Securing an open platform requires a strong security architecture and rigorous security programs. Android was designed with multi layered security that is flexible enough to support an open platform while still protecting all users of the platform. For information about reporting security issues and the update process, see Security Updates and Resources.

Android is designed for developers: Security controls were designed to reduce the burden on developers. Security-savvy developers can easily work with and rely on flexible security controls. Developers less familiar with security are protected by safe defaults. In addition to providing a stable platform to build upon, Android gives additional support to developers in several ways. The Android security team looks for potential vulnerability in apps and suggests ways to fix those issues. For devices with Google Play, Play Services delivers security updates for critical software libraries, such as OpenSSL, which is used to secure app communications. Android security released a tool for testing SSL.

Android Performance

Performance can mean a lot of different things to different people. When it comes to mobile apps, performance can describe how an app works, how efficiently it works, or if it was enjoyable to use. In the context of this book, we are looking at performance in terms of efficiency and speed.
From an Android perspective, performance is complicated by thousands of different devices, all with different levels of computing power. Sometimes just getting your app to run across your top targeted devices feels like an accomplishment on its own.

CHAPTER 2

Building block of A Selective and Restrictive Communication Model Based Mobile Application

INTRODUCTION

We develop a smart phone based mobile App as a supplement to an IoT device. This App will contact another device (embedded with a GSM module) through its Email Id. This App will contact only one Email Id initially.

Buliding Block of a Selective and Restrictive Communication Model Based Mobile Application

After an advantageous description about android and building blocks now we are going to describe building blocks of our project Selective and restrictive communication model based mobile.
There are two main building blocks of Selective and Restrictive communication model based mobile

1. Activity
2. Intents

ACTIVITY: An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView (View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with R. attr. windowIsFloatingset), Multi-Window mode or embedded into other windows. There are two methods almost all subclasses of Activity will implement:
onCreate (Bundle) is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI and using findViewById(int)to retrieve the widgets in that UI that you need to interact with programmatically. onPause () is where you deal with the user pausing active interaction with the activity. Any changes made by the user should at this point be committed (usually to the Content Provider holding the data). In this state the activity is still visible on screen.

```
import android.app.Activity
public class    MainActivity extends Activity
{
.
.
.
}
```

INTENTS:

In Android, it is quite usual for users to witness a jump from one application to another as a part of the whole process, for example, searching for a location on the browser and witnessing a direct jump into Google Maps or receiving payment links in Messages Application (SMS). This process of taking users from one application to another is achieved by passing the Intent to the system. Intents, in general, are used for navigating among various activities within the same application, but note, is not limited to one single application, i.e., they can be utilized from moving from one application to another as well.

Intents could be Implicit, for instance, Mail intended actions and explicit as well, such as opening another activity after some operations like onClick or anything else. Below are some applications of Intents:

1. Sending the User to Another App
2. Getting a Result from an Activity
3. Allowing Other Apps to Start Your Activity

```
StringphoneNo=txtemailId.getText().toString();
Intent=newIntent(Intent..ACTION_MAIL);
i.setData(Uri.parse("mail"+emailId);
startActivity(i);
```

Chapter 3

PROJECT ARCHITECTURE

Here we will present an architectural view of my application OF a Selective and restrictive communication model based mobile.

Presentation Layer:

- **User Interface:** This layer consists of the user interface components, including screens, menus, and forms, through which users interact with the application.
- **User Input Handling:** It manages user inputs, such as selecting contact filters, privacy settings, and communication modes.

Application Layer:

- **Communication Management:** This component handles the logic related to managing communication channels, including filtering contacts, defining privacy settings, and enforcing time-based restrictions.

Group Management: It handles the creation, modification, and management of communication groups, including assigning roles and permissions to group members.

Data Layer:

- **User Profile Database:** It stores user profile information, including contact preferences, privacy settings, and communication restrictions.
- **Contact Database:** This database holds the contact details of the user's contacts, including their communication preferences and access levels.
- **Group Database:** It stores information about communication groups, including group members, roles, and permissions.

Security Layer:

- **Authentication:** This component handles user authentication and ensures that only authorized individuals can access the application and modify communication settings.

- **Encryption:** It encrypts sensitive user data, such as personal information, communication preferences, and group details, to ensure data confidentiality.
- **Access Control:** This component enforces access control policies, ensuring that users can only view and modify communication settings and data according to their permissions.

WORKING FUNCTIONALITIES

This application let's user to say where is he the application says the best way of current location this app contains :-
Registration (While setting up the App)

- Name
- Mobile Number
- Mail-id
- Phone number to be contacted Calling the phone number
- Sending a message to the Email Id
- Capability to receive a message from the predefined Email Id and
  decode the message which normally consists of an URL of the locations
  Show the location of the device (contact Email Id) on Google Maps Storing the log of interactions with the predefined email Id

Chapter 4

Execution Result of a Selective and restrictive communication Model based mobile

This section discusses the execution part of our application after a long discussion and development about our application a selective and restrictive communication model based mobile.
As we described in chapter 1 that we will be using Emulator to run our application, we also described about how to install Emulator on our Android Studio . So, following are the Snapshots

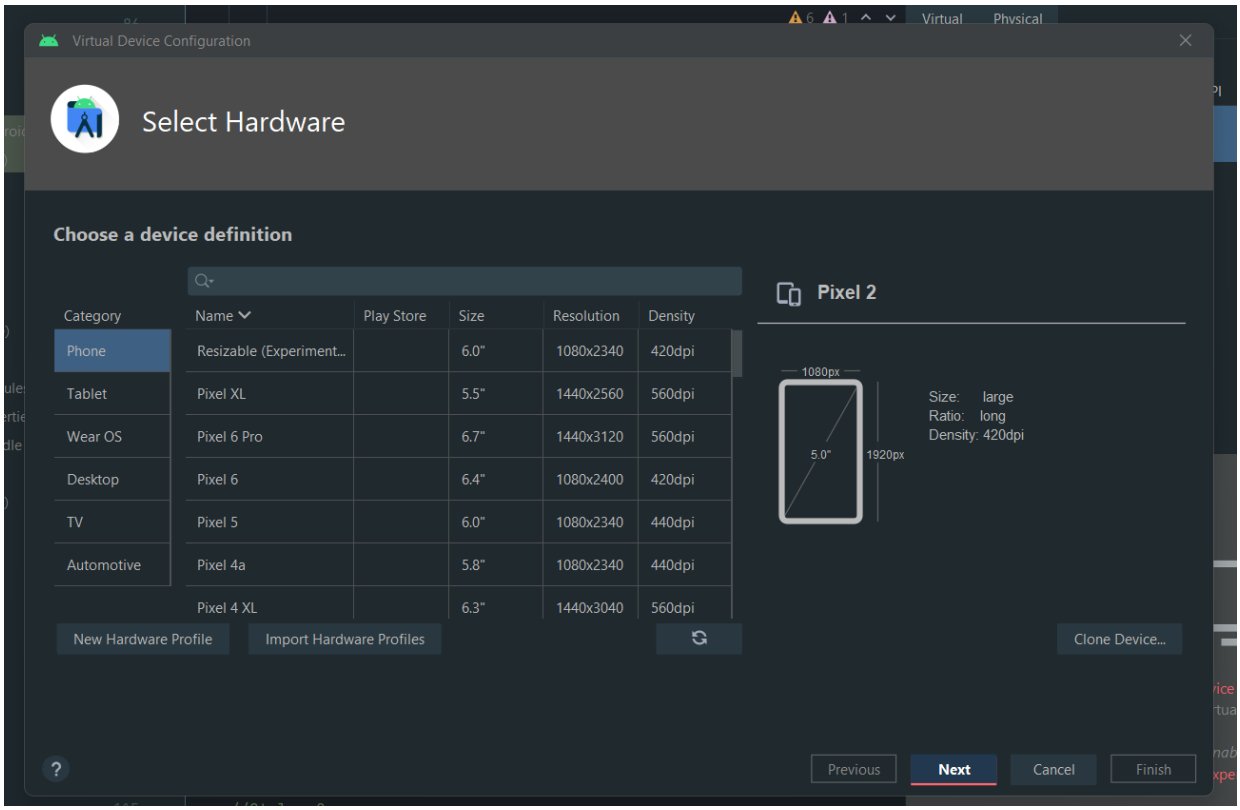


Fig: Install Virtual Device

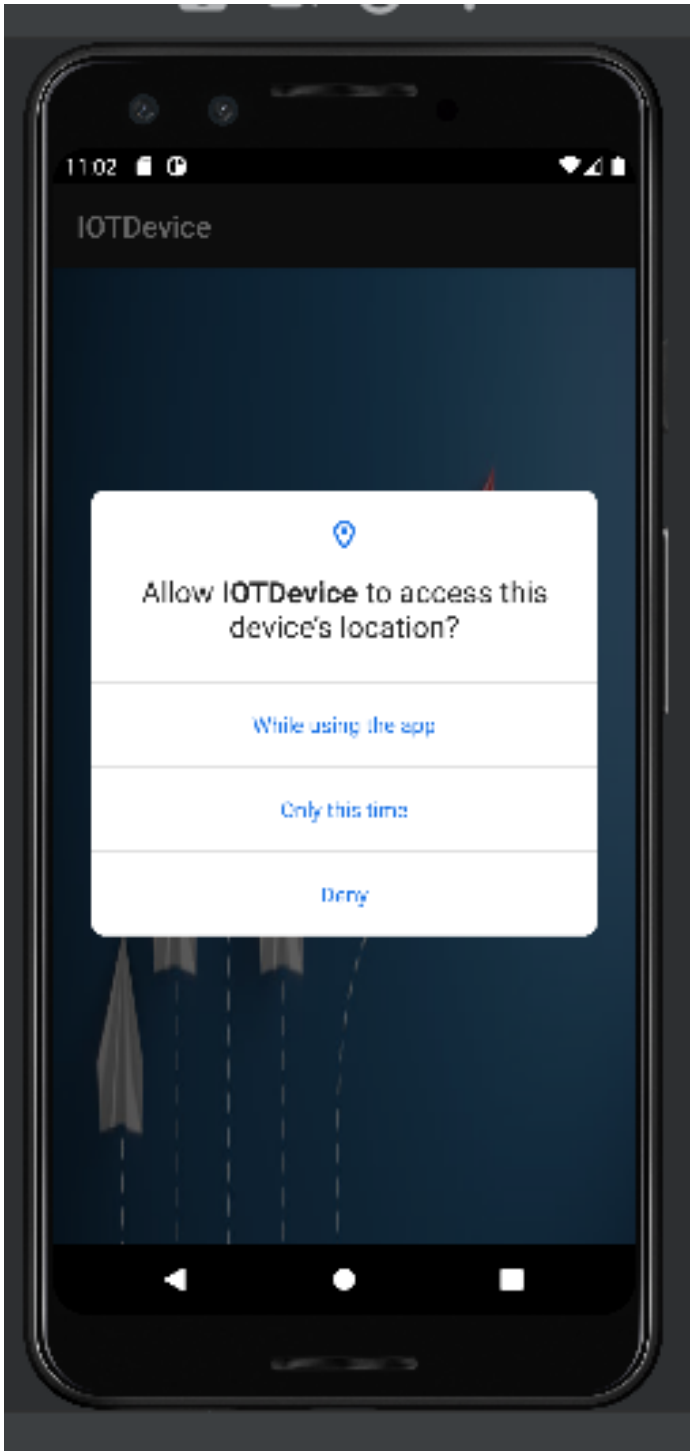


Fig 4.(a): App Permission

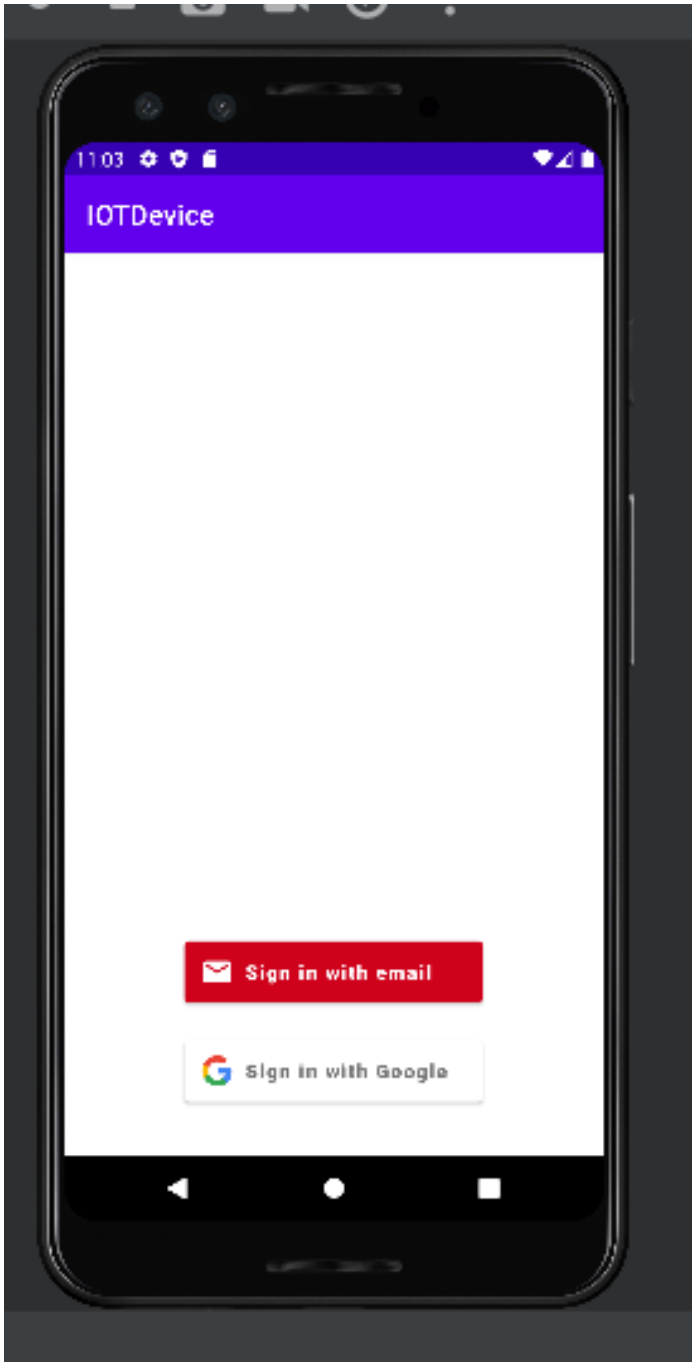


Fig 4.(b): The Login screen of the application

Fig 4.(c): The Home screen of the application

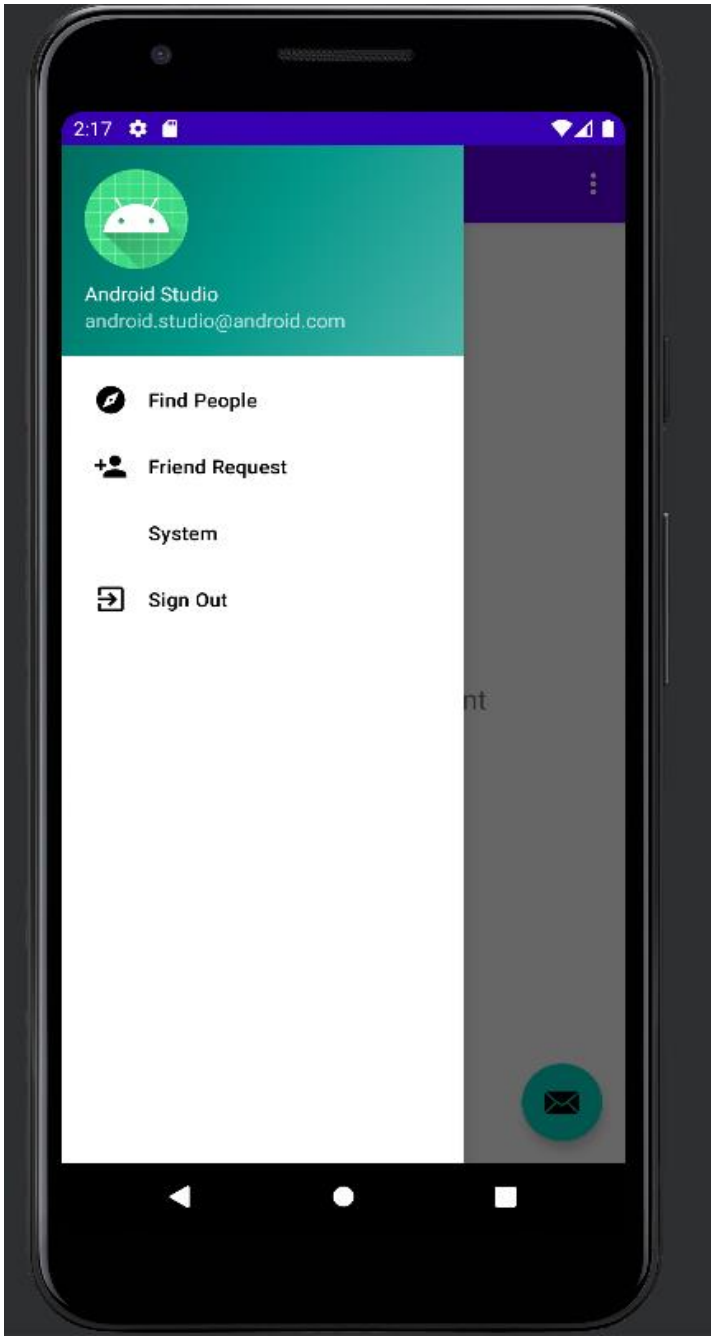


Fig 4.(d): The Hamburger Menu of the application

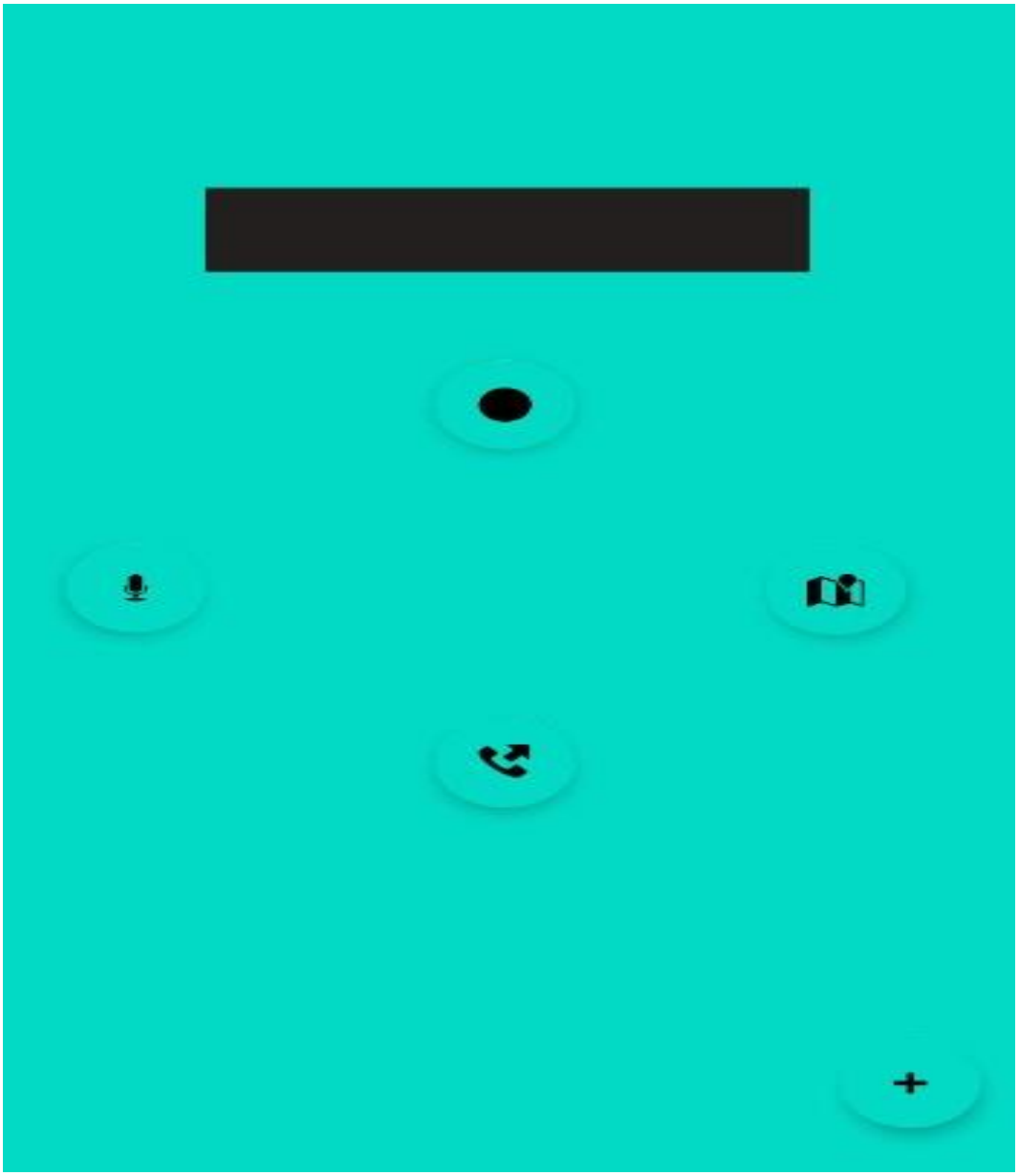


Fig 4.(e):The main communication module screen of the application

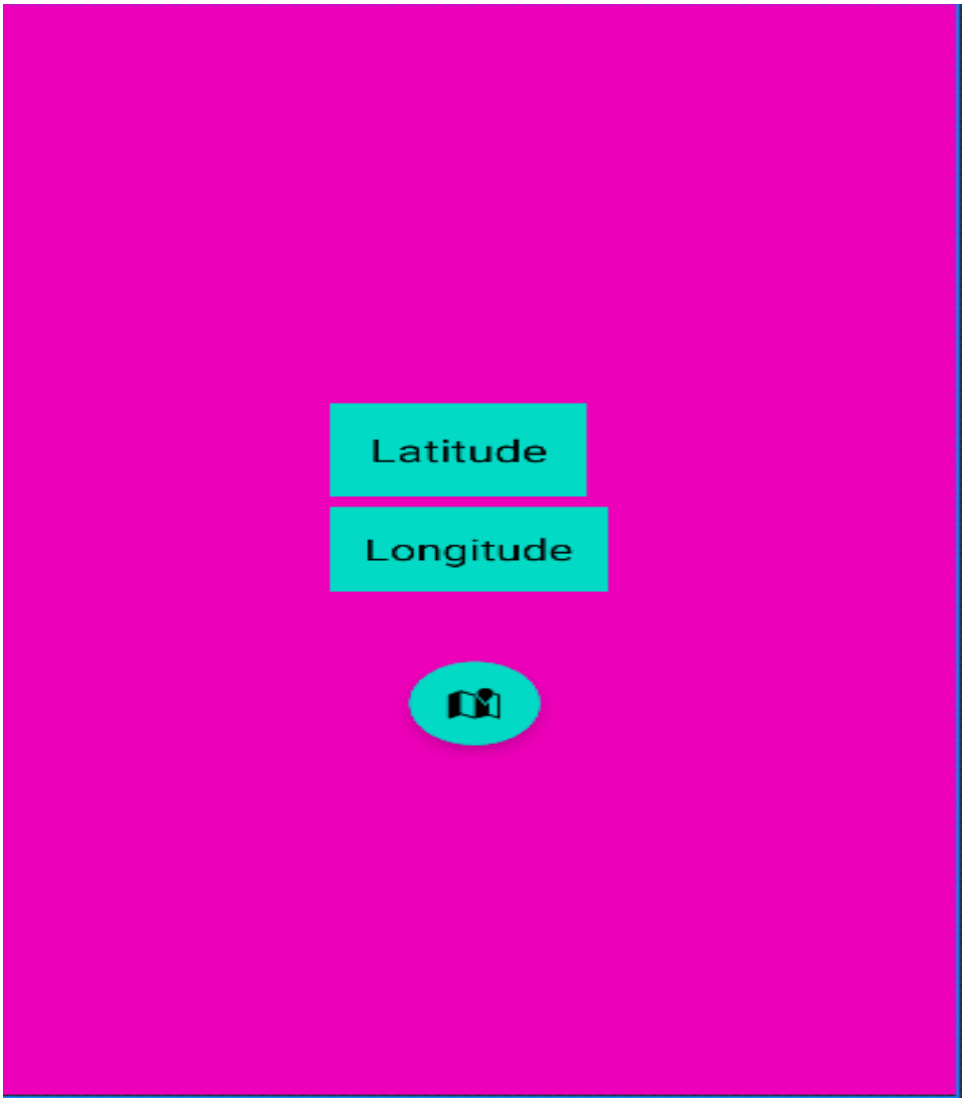


Fig 4.(f): The location fetch module screen of the application

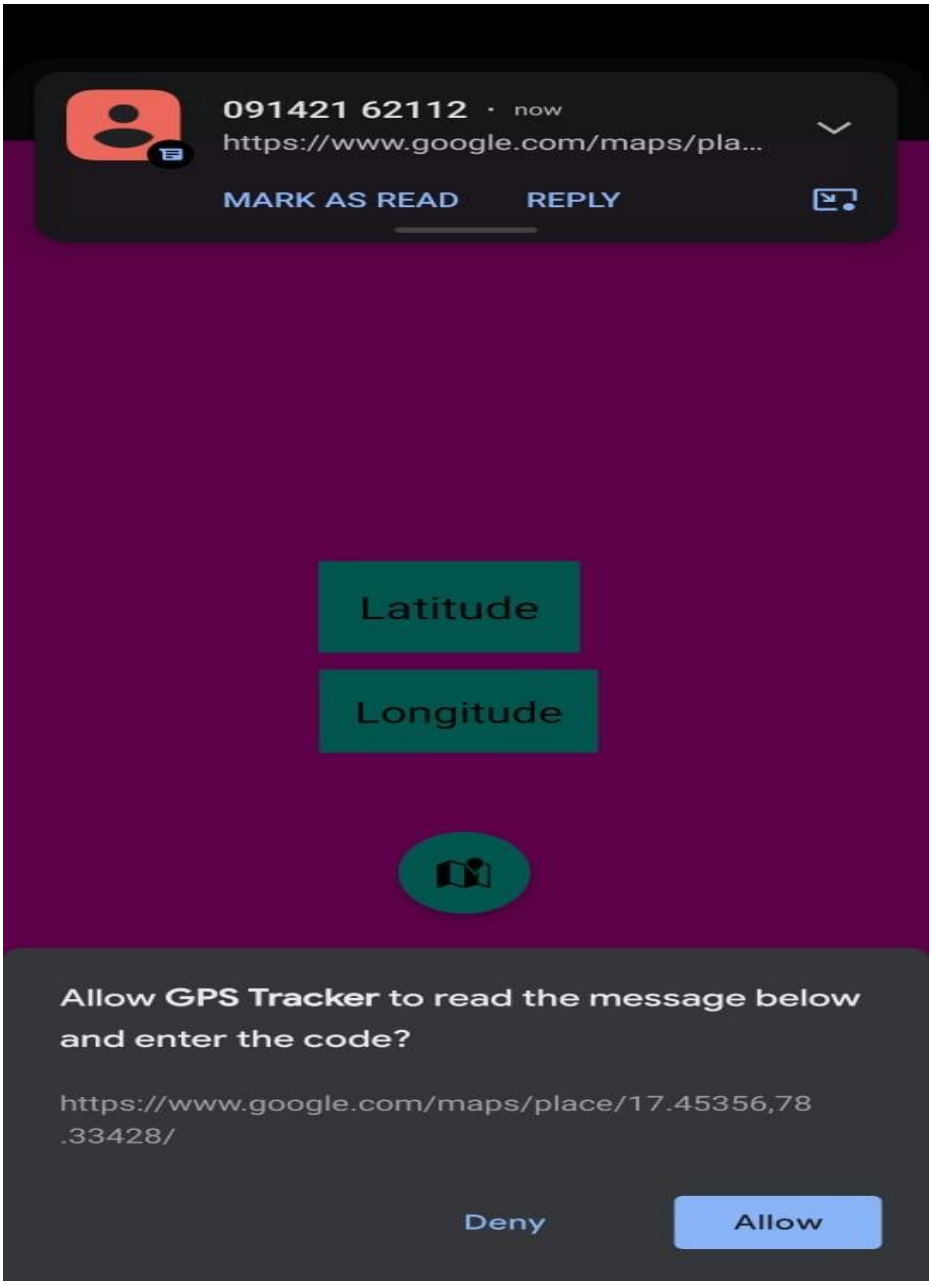


Fig 4.(g): The location receive module screen of the application

Fig 4.(h): Location Screen

Chapter 5

CODE

1. MainActivity.java

```
import static android.content.ContentValues.TAG;
import android.Manifest;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import com.example.iotdevice.Model.User;
import com.example.iotdevice.util.common;
import com.firebase.ui.auth.AuthUI;
import com.firebase.ui.auth.IdpResponse;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.messaging.FirebaseMessaging;
import com.karumi.dexter.Dexter;
import com.karumi.dexter.MultiplePermissionsReport;
import com.karumi.dexter.PermissionToken;
public class MainActivity extends AppCompatActivity {
   DatabaseReference user_information;
   private static final int MY_REQUEST_CODE = 7117;
   List<AuthUI.IdpConfig> providers;
   @Override
```

```
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // setContentView(R.layout.activity_main);
    Paper.init(this);
    user_information =
FirebaseDatabase.getInstance().getReference(common.USER_INFORMATION);
    //Init provider
    providers = Arrays.asList(
        new AuthUI.IdpConfig.EmailBuilder().build(),
         new AuthUI.IdpConfig.GoogleBuilder().build()
    );
//Request permission location
    Dexter.withActivity(this)
        .withPermissions(Manifest.permission.ACCESS_FINE_LOCATION)
        .withListener(new MultiplePermissionsListener() {
          @Override
          public void onPermissionsChecked(MultiplePermissionsReport
multiplePermissionsReport) {
            if(multiplePermissionsReport.areAllPermissionsGranted()){
              showSignInOptions();
            }//else
if(multiplePermissionsReport.isAnyPermissionPermanentlyDenied()){
              //
this.onPermissionDenied(multiplePermissionsReport.getDeniedPermissionResponses
().get(0));
            else {
              Toast.makeText(MainActivity.this,"You must accept permission to use
app",Toast.LENGTH_SHORT).show();
            }
          }
          @Override
          public void
onPermissionRationaleShouldBeShown(List<PermissionRequest> list,
PermissionToken permissionToken) {
            permissionToken.continuePermissionRequest();
```

```
            }
        }).check();
@Override
  protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == MY_REQUEST_CODE)
    {
      IdpResponse response = IdpResponse.fromResultIntent(data);
      if(resultCode == RESULT_OK)
      {
        FirebaseUser firebaseUser = FirebaseAuth.getInstance().getCurrentUser();
        //Check if User exists on Database
        user_information.orderByKey()
            .equalTo(firebaseUser.getUid())
            .addListenerForSingleValueEvent(new ValueEventListener() {
              @Override
              public void onDataChange(@NonNull DataSnapshot snapshot) {
                if(snapshot.getValue() == null)//If user is not exists
                {
                  if(!snapshot.child(firebaseUser.getUid()).exists())//if key uid not
exists
                  {
                    common.loggedUser                =                new
User(firebaseUser.getUid(),firebaseUser.getEmail());
                    //Add to database
                    user_information.child(common.loggedUser.getUid())
                        .setValue(common.loggedUser);
                  }
                }
```

Chapter: 6

Deploying the Application on Android Mobile

Because the Android operating System is a relative newcomer to the mobile phone market, a strong and cohesive Android Internet Support community appears to be lacking. Without an adequate Android support system in place, many Android users are left in the dark when it comes to many tricky tasks, like installing third party applications to their Android mobile phones. In order to install APK or Android Package files. In this article, we will cover the two ways you can install APK files to your Android phone and show you how to take advantage of the wide variety of third-party Android applications currently available.

INSTALLING APPLICATIONS THROUGH THE ANDROID MARKET

Step 1
Turn on your device, go to the main home screen and press the 'Applications' tab that opens the applications menu on the device.

Step 2
Locate the Android Market application from the menu and press that icon.

Step 3
The icon will open the Android Market, where you can access different apps under varied categories form the apps tab. But make sure you are connected to the internet through GPRS or WiFi.

Step 4
Install the desired application. Users can also make use of the application to search features by entering relevant keywords in the field on the top right-hand side of the home screen.

Installing Applications with Android SDK
It is possible to install APK files without utilizing the Android Market although the process is more difficult and complex to avoid the android market you need to use Android SDK.

Chapter 7

Conclusion & Future Work

Conclusion

The project helped us to know how we connect with real world entities. Also familiar with different technologies. Along these we learned how to write Android-compatible Application and interact with IOT devices.

Future Plan:

- To record and transmit ambient sounds and voices to the mobile App/ users on demand without intervention of the user of the IoT device
- Implement machine learning algorithms to provide personalized recommendations for contact filtering and communication preferences. The application can learn from user interactions and patterns to suggest relevant filters and settings, making the selective communication experience more intuitive and efficient.
- Enable integration with smart home devices, such as voice assistants or IoT (Internet of Things) devices, to extend selective communication controls beyond the mobile application. Users can manage communication preferences and restrictions through voice commands or automated triggers based on their smart home settings.

Bibliography

1. Nitin Kumar, “Use of Wireless Access Point ID for position determination”, US Patent, No. US2014/ 0243014 A1, 28th August 2014
2. Robert Eisenman, “Identification Card holder with personal locator”, US Patent, No.US2016/0240075 A1, 18th August 2016
3. https://www.javatpoint.com/android-tutorial
4. https://source.android.com/devices/architecture
5. https://www.youtube.com/watch?v= OMnFR-wZU
6. https://www.geeksforgeeks.org/android-tutorial/
7. https://www.sqlite.org/docs.html
8. https://stackoverflow.com/questions/64727665/w-system-ignoring-header-x-firebase-locale-because-its-value-was-null