



**NEW HORIZON
COLLEGE OF ENGINEERING**

New Horizon Knowledge Park, Ring Road, Marathalli

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC

Accredited by NAAC with 'A' Grade, Accredited by NBA

“FILE TRANSFER AND CHAT APPLICATION”

A MINI PROJECT REPORT

Submitted by

Under the guidance of,

Dr R J Anandhi

Head Of The Department,ISE,NHCE

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

INFORMATION SCIENCE AND ENGINEERING

FOR

COURSE NAME: MINI PROJECT

COURSE CODE :19ISE49



NEW HORIZON COLLEGE OF ENGINEERING

New Horizon Knowledge Park, Ring Road, Marathalli
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

CERTIFICATE

Certified that the project work entitled Facial Expression Detection Using deep Learning carried out by Mr. PUNITH KUMAR S, bearing USN 1NH18IS079, a bonafide student of 3rd semester in partial fulfillment for the award of Bachelor of Engineering in Information Science & Engineering of the Visveswaraiah Technological University, Belagavi during the year 2019-20. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated. The project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the said Degree.

Name & Signature of Guide

Dr R J Anandhi

Name & Signature of HOD

Dr R J Anandhi

Name & Signature of Principal

Dr. Manjunatha

Examiners :

Name

Signature

1.

.....

2.

.....

PLAGIARISM CERTIFICATE

FILE TRANSFER AND CHAT APP

ORIGINALITY REPORT

10%

SIMILARITY INDEX

%

INTERNET SOURCES

10%

PUBLICATIONS

%

STUDENT PAPERS

ABSTRACT

The file transfer and chat app is developed in java used by android studio GUI application based as it user friendly to design the application. Which is based on object oriented methodology. There are several packages in java, but mainly socket packages and networking has been utilized in developing this project. This application project in java can be successfully operated in real environment and can be tested practically. The files and chat messages can be transferred to the selected clients without any errors are perfectly executed upon download.

This application uses Wi-Fi pair connection or peer-to-peer connections and shares the data or files among the devices, this feature is very useful for applications have to transfer files among users.

Methods:-

1. Initialize();
2. Connect();
3. Discoverpeers();
4. Cancelconnect();
5. Remove();

Features:-

1. Easy connecting
2. Fast data transfer as it use Wi-Fi
3. Friendly chatting without data

System Requirements:-

Hardware Requirements:-

System :- i5 processor
Hard Disk :- 500GB
Ram:- 4GB

Software Requirements:-

Operating System :- Windows 10
Coding language :- java
IDE :- Android Studio
Development as:- Android Application

ACKNOWLEDGEMENT

Any project is a task of great enormity and it cannot be accomplished by an individual without support and guidance. I am grateful to a number of individuals whose professional guidance and encouragement has made this project completion a reality.

I have a great pleasure in expressing my deep sense of gratitude to the beloved Chairman **Dr. Mohan Manghnani** for having provided me with a great infrastructure and well-furnished labs.

I take this opportunity to express my profound gratitude to the Principal **Dr. Manjunatha** for his constant support and management.

I am grateful to **Dr. R J Anandhi**, Professor and Head of Department of ISE, New Horizon College of Engineering, Bengaluru for his strong enforcement on perfection and quality during the course of my project work.

I would like to express my thanks to the guide **Dr. R J Anandhi**, Head Of The Department, Department of ISE, New Horizon College of Engineering, Bengaluru who has always guided me in detailed technical aspects throughout my project.

I would like to mention special thanks to all the Teaching and Non-Teaching staff members of Information Science and Engineering Department, New Horizon College of Engineering, Bengaluru for their invaluable support and guidance.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Introduction	3
1.2 Problem statement.....	4
1.4 Objective	4

CHAPTER 2 LITERATURE REVIEW

2.1 Existing System.....	6
2.1 Proposed System.....	6

CHAPTER 3 SYSTEM REQUIREMENTS

3.1 Android Studio Design.....	7
3.2 System requirements.....	11
3.2 Frameworks.....	12

CHAPTER 4 PREREQUISITES

4.1 Accessing Phone Permission.....	15
-------------------------------------	----

CHAPTER 5 FLOWCHART

5.1 Flowchart of File transfer.....	16
5.2 Flowchart of Chatting	17

CHAPTER 6 CODE AND IMPLEMENTATION

6.1 Java modules and Libraries.....	18
6.2 Code Implement.....	21

CHAPTER 7.2 OUTPUTS AND CONCLUSION

7.1 Outputs of APP	30
7.2 Conclusion	34

BIBLIOGRAPHY	35
--------------------	----

REFERENCES

LIST OF TABLES

1. Table 3.2: System Requirements.....	11
2. Table 3.3.1: Wi-Fi P2P Methods.....	13
3. Table 3.3.2: Wi-Fi P2P Listener.....	13
4. Table 3.3.3: Wi-Fi P2P Intents.....	14

LIST OF FIGURES

Figure 1.1 Android Studio Window.....	4
Figure 1.3 Android Studio visual Code.....	5
Figure 3.1.1 Code before formatting.....	8
Figure 3.1.2 Code after formatting.....	8
Figure 3.1.3 Inline Variable value.....	10
Figure 5.1 File transfer flowchart.....	16
Figure 5.2.Chat flowchart.....	17
Figure 6.1 Libraries.....	18
Figure 6.2.1 WiFiDirectBoardcastReciver.....	22
Figure 6.2.2 WiFiOn/OFF.....	23
Figure 6.2.3 Discovering Peers.....	23
Figure 6.2.4 Connecting peers.....	24
Figure 6.2.5 Host/server Thread.....	24
Figure 6.2.6 Client Thread.....	25
Figure 6.2.7 Send/Receive Thread.....	26
Figure 6.2.8 Receive File Request	27
Figure 6.2.9 Respond File selection.....	28
Figure 6.2.10 Grant File Permission.....	29
Figure 6.2.11 Share File.....	29

LIST OF ABBREVIATIONS

Wi-Fi Wireless Fidelity

GUI Graphical user interface

IDE Integrated development environment

P2P Peer-to-peer

INTRODUCTION

1.1 INTRODUCTION:-

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance our productivity when building Android apps. As :-

- ✓ A flexible Gradle-based build system.
- ✓ A fast and feature-rich emulator.
- ✓ Extensive testing tools and frameworks.
- ✓ A unified environment where we can develop for all android devices.
- ✓ Lint tools to catch performance, usability, version compatibility, and other problems.

All the build files are visible at top level under Gradle Scripts and each app module contains the following folders:-

- **Manifests:** Contains the AndroidManifest.xml file.
- **Java:** Contains the Java source code files, including JUnit test code.
- **Res:** Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android Studio main window is made up of several logical areas identified in figure 1.

1. The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
2. The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
5. The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

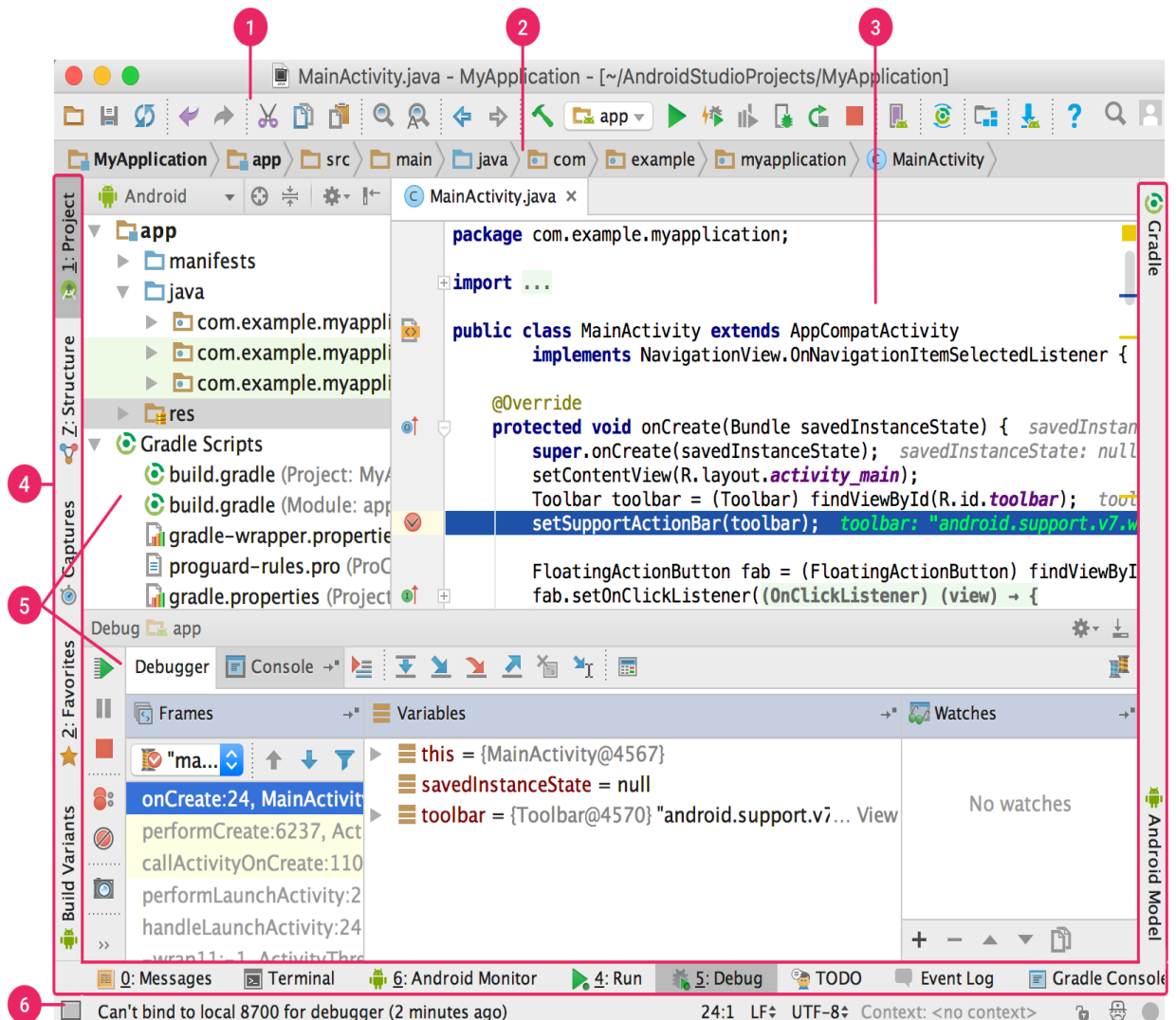


Figure 1.1 The Android Studio window.

File transfer and Chat App is developed in java used by android studio GUI application based as it user friendly to design the application. Which is based on object oriented methodology. This application project in java can be successfully operated in real environment and can be tested practically. The files and chat messages can be transferred to the selected clients without any errors are perfectly executed upon download. This App is basically have two parts as one for chatting and another for file transfer user can ensure which task to do. The app is based on Wi-Fi and hotspot so data is begin transferred easily with some minor seconds. App includes all the Wi-Fi files inside it, app will help to transfer files from server to client easily.

1.2 PROBLEM STATEMENT:-

File transfer and chat app is used in data transfer from one to other server to client and to start a connection with client server and start transferring the files and chat with them. The files and chat messages can be transferred to the selected clients without any errors. The app has Wi-Fi connectivity so that the transfer rate of will be fast,

1.3 OBJECTIVE:-

- ✓ To make easy connection between the users.
- ✓ To transfer all types of files within a short duration.
- ✓ Friendly chatting without data.
- ✓ This application is totally secure for the users. Only if the receiver has the key he can decrypt the ciphered message to the correct message.
- ✓ To develop code with Android Studio using JAVA frameworks, Android Studio helps in developing the Graphical User Interface and it can develop and Android Package Kit.
- ✓ The APK can be installed in all the Android devices and can run the application to do the necessary tasks.

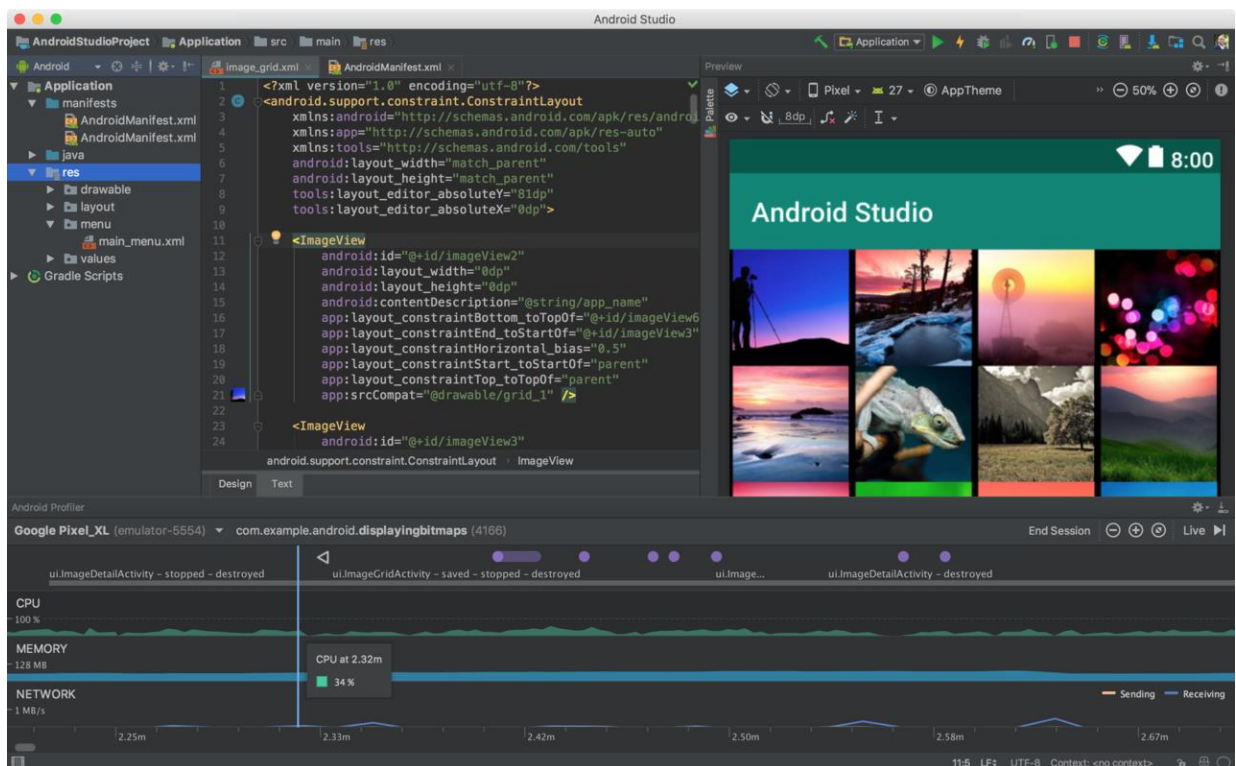


Figure 1.3 Android Studio visual Code

CHAPTER 2

LITERATURE REVIEW

2.2 Existing System:-

The existing systems are not application based and chatting and file transferred are not same, as it use Wi-Fi hotspot and the sharing of files will be less priority. The chat and file transfer has not been defined by any application with inside. So the existing system cannot achieve the goal chat and file transfer app.

2.3 Proposed System:-

The proposed chat and file transfer has both in it and it is developed in Android studio that it can be transfer to any of the android device and APK format helps to download easily and install and use it, and the Studio provides various framework and modules so the implementation of the APK is being easy but the Keywords of that modules should be read properly and used without any Errors.

The name of the App is “BlaB Me” as it means talk something it says like we can chat easily so on transferring files make the application so helpful for friends to share the contents like (Images, documents, Audio, Video). And the app is designed colorful so that it attracts the users in chatting.

SYSTEM REQUIREMENTS

3.1 Android Studio Design:-

Tools:-

Instead of using preset perspectives, Android Studio follows your context and automatically brings up relevant tool windows as you work. By default, the most commonly used tool windows are pinned to the tool window bar at the edges of the application window.

Code completion:-

Android Studio has three types of code completion:-

Basic Completion:-

Displays basic suggestions for variables, types, methods, expressions, and so on. If you call basic completion twice in a row, you see more results, including private members and non-imported static members.

Smart Completion:-

Displays relevant options based on the context. Smart completion is aware of the expected type and data flows. If you call Smart Completion twice in a row, you see more results, including chains.

Statement Completion:-

Completes the current statement for you, adding missing parentheses, brackets, braces, formatting, etc.

Here are some tips to help you move around Android Studio.

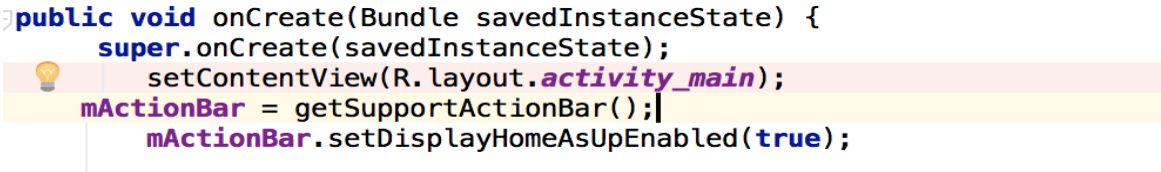
- Switch between your recently accessed files using the *Recent Files* action. Press **Control+E** to bring up the Recent Files action. By default, the last accessed file is selected. You can also access any tool window through the left column in this action.
- View the structure of the current file using the *File Structure* action. Bring up the File Structure action by pressing **Control+F12**. Using this action, you can quickly navigate to any part of your current file.
- Search for and navigate to a specific class in your project using the *Navigate to Class* action. Bring up the action by pressing **Control+N**. Navigate to Class supports sophisticated expressions, including camel humps, paths, line navigate to, middle name matching, and many more. If you call it twice in a row, it shows you the results out of the project classes.
- Navigate to a file or folder using the *Navigate to File* action. Bring up the Navigate to File action by pressing **Control+Shift+N** (**Command+Shift+O** on a Mac). To search for folders rather than files, add a / at the end of your expression.

- Navigate to a method or field by name using the *Navigate to Symbol* action. Bring up the *Navigate to Symbol* action by pressing **Control+Shift+Alt+N** (**Command+Option+O** on a Mac).
- Find all the pieces of code referencing the class, method, field, parameter, or statement at the current cursor position by pressing **Alt+F7** (**Option+F7** on a Mac).

Style and formatting:-

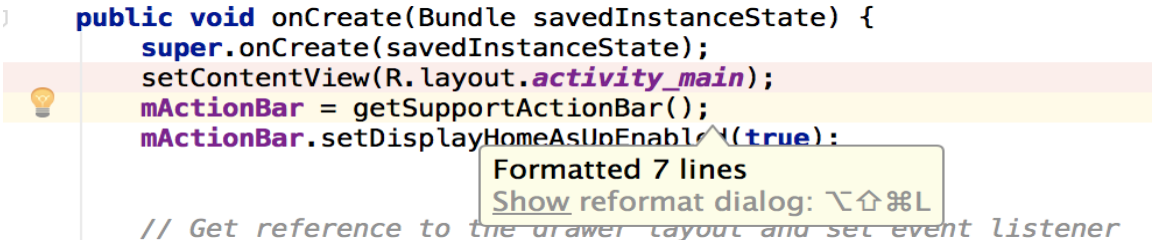
As you edit, Android Studio automatically applies formatting and styles as specified in your code style settings. You can customize the code style settings by programming language, including specifying conventions for tabs and indents, spaces, wrapping and braces, and blank lines.

Although the IDE automatically applies formatting as you work, you can also explicitly call the *Reformat Code* action by pressing **Control+Alt+L**, or auto-indent all lines by pressing **Control+Alt+I**.



```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mActionBar = getSupportActionBar();
    mActionBar.setDisplayHomeAsUpEnabled(true);
}
```

Figure 3.1.1 Code before formatting.



```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mActionBar = getSupportActionBar();
    mActionBar.setDisplayHomeAsUpEnabled(true);
    // Get reference to the drawer layout and set event listener
}
```

Figure 3.1.2 Code after formatting.

Version control basics:-

Android Studio supports a variety of version control systems (VCS's), including Git, GitHub, CVS, Mercurial, Subversion, and Google Cloud Source Repositories.

After importing your app into Android Studio, use the Android Studio VCS menu options to enable VCS support for the desired version control system, create a repository, import the new files into version control, and perform other version control operations.

Gradle build system:-

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android plug-in for Gradle. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app, with different features using the same project and modules.
- Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are named `build.gradle`. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plug-in for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

Resource shrinking:-

Resource shrinking in Android Studio automatically removes unused resources from your packaged app and library dependencies. For example, if your application is using Google Play services to access Google Drive functionality, and you are not currently using Google Sign-In, then resource shrinking can remove the various drawable assets for the `SignInButton` buttons.

Managing dependencies:-

Dependencies for your project are specified by name in the `build.gradle` file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your `build.gradle` file. Android Studio configures projects to use the Maven Central Repository by default. (This configuration is included in the top-level build file for the project.) For more information about configuring dependencies, read [Add build dependencies](#)

Debug and profile tools:-

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

Inline debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object
- Method return values
- Lambda and operator expressions

- Tooltip values

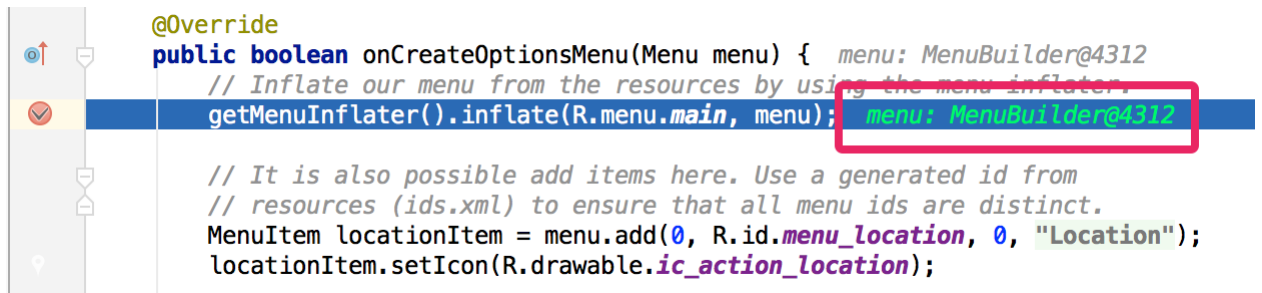


Figure 3.1.3 Inline variable value.

Performance profilers:-

Android Studio provides performance profilers so you can more easily track your app's memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyze network requests. With your app running on a device or emulator, open the **Android Profiler** tab.

Memory Profiler:-

You can use Memory Profiler to track memory allocation and watch where objects are being allocated when you perform certain actions. Knowing these allocations enables you to optimize your app's performance and memory use by adjusting the method calls related to those actions.

Data file access:-

The Android SDK tools, such as Systrace, and logcat, generate performance and debugging data for detailed app analysis.

Code inspections:-

Whenever you compile your program, Android Studio automatically runs configured Lint and other IDE inspections to help you easily identify and correct problems with the structural quality of your code. The Lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization. In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline your coding workflow.

Annotations in Android Studio:-

Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the Support-Annotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection.

Log messages:-

When you build and run your app with Android Studio, you can view adb output and device log messages in the **Logcat** window.

3.2. System Requirements

Table 3.1: System Requirements

COMPONENTS
➤ LAPTOP
➤ ANDROID STUDIO 3.6.2(version: 1.8.0_212-release-1586-b04 amd64) -VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o -Windows 10 10.0 -GC: ParNew, ConcurrentMarkSweep -Memory: 1263M -Cores: 2
➤ EAMULATOR:-NOX PLAYER(VERSION: V6.6.0.5205) -Windows 10 -Intel CPU (with Virtualization Technology (VT-x or AMD-V) support preferred) - At least 1.5 GB RAM - GPU with OpenGL 2.0+ support - At least 1.5 GB of free disk space under installation path (for saving the data of the apps/games you may install) and 300MB under system disk
➤ RAM:- 4GB
➤ ANDROID PHONE(VERSION:- Grater Than Android 4.1)

3.3 Frameworks:-

Wi-Fi P2P Framework(Wi-Fi Direct):-

Wi-Fi Direct(P2P) allows Android 4.0 (API level 14) or later devices with the appropriate hardware to connect directly to each other via Wi-Fi without an intermediate access point. Using these APIs, you can discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection. This is useful for applications that share data among users, such as a multiplayer game or a photo sharing application.

The Wi-Fi P2P APIs consist of the following main parts:

- Methods that allow you to discover, request, and connect to peers are defined in the `WifiP2pManager` class.
- Listeners that allow you to be notified of the success or failure of `WifiP2pManager` method calls. When calling `WifiP2pManager` methods, each method can receive a specific listener passed in as a parameter.
- Intents that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.

You often use these three main components of the APIs together. For example, you can provide a `WifiP2pManager . ActionListener` to a call to `discoverPeers()`, so that you can be notified with the `ActionListener.onSuccess()` and `ActionListener.onFailure()` methods.

A `WIFI_P2P_PEERS_CHANGED_ACTION` intent is also broadcast if the `discoverPeers()` method discovers that the peers list has changed.

API overview:-

The `WifiP2pManager` class provides methods to allow you to interact with the Wi-Fi hardware on your device to do things like discover and connect to peers. The following actions are available:

Table 3.3.1 Wi-Fi P2P Methods

Method	Description
<code>initialize()</code>	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.
<code>connect()</code>	Starts a peer-to-peer connection with a device with the specified configuration.
<code>cancelConnect()</code>	Cancels any ongoing peer-to-peer group negotiation.
<code>requestConnectInfo()</code>	Requests a device's connection information.
<code>createGroup()</code>	Creates a peer-to-peer group with the current device as the group owner.
<code>removeGroup()</code>	Removes the current peer-to-peer group.
<code>requestGroupInfo()</code>	Requests peer-to-peer group information.
<code>discoverPeers()</code>	Initiates peer discovery
<code>requestPeers()</code>	Requests the current list of discovered peers.

WifiP2pManager methods let you pass in a listener, so that the Wi-Fi P2P framework can notify your activity of the status of a call. The available listener interfaces and the corresponding WifiP2pManager method calls that use the listeners are described in the following table:

Table 3.3.2 Wi-Fi P2P Listeners

Listener interface	Associated actions
<code>WifiP2pManager.ActionListener</code>	<code>connect()</code> , <code>cancelConnect()</code> , <code>createGroup()</code> , <code>removeGroup()</code> , and <code>discoverPeers()</code>
<code>WifiP2pManager.ChannelListener</code>	<code>initialize()</code>
<code>WifiP2pManager.ConnectionInfoListener</code>	<code>requestConnectInfo()</code>
<code>WifiP2pManager.GroupInfoListener</code>	<code>requestGroupInfo()</code>

WifiP2pManager.PeerListListener requestPeers()

The Wi-Fi P2P APIs define intents that are broadcast when certain Wi-Fi P2P events happen, such as when a new peer is discovered or when a device's Wi-Fi state changes. You can register to receive these intents in your application by creating a broadcast receiver that handles these intents:

Table 3.3.3 Wi-Fi P2P Intents

Intent	Description
WIFI_P2P_CONNECTION_CHANGED_ACTION	Broadcast when the state of the device's Wi-Fi connection changes.
WIFI_P2P_PEERS_CHANGED_ACTION	Broadcast when you call discoverPeers(). You usually want to call requestPeers() to get an updated list of peers if you handle this intent in your application.
WIFI_P2P_STATE_CHANGED_ACTION	Broadcast when Wi-Fi P2P is enabled or disabled on the device.
WIFI_P2P_THIS_DEVICE_CHANGED_ACTION	Broadcast when a device's details have changed, such as the device's name.

PREREQUISITES

The project has certain prerequisites for the current implementation of the project which has been discussed in this chapter.

4.1 Accessing Phone Permission and Request To send Data:-

- ✓ There should be permission to access Wi-Fi from the user.
- ✓ The APK also asks the permission to access of Files and data.
- ✓ Android App will have both chat and file transfer so it will be user Friendly.
- ✓ And there should be Google support so that user can access the application.
- ✓ There is no group chat as it is Wi-Fi direct can connect to only with server and client.
- ✓ There should be support of APK files that Android version should be greater than 4.1

So that this Application can be used.

FLOWCHART

5.1 Flowchart of file transfer:-

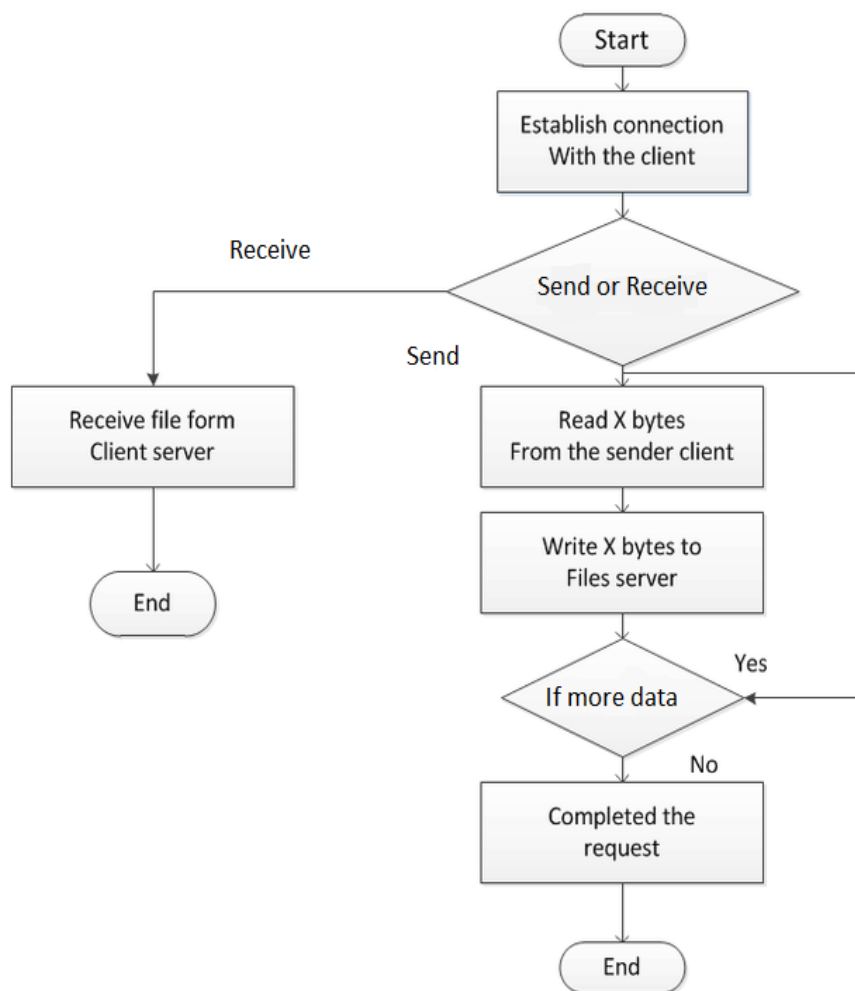


Figure 5.1 File Transfer Flowchart

The above flow chart of file transfer includes starting the establish of peers than connect to the client if the user want to receive go to receive or to send data move to send and start selecting the file to be transferred and if done click on exit to kill that connection.

5.2 Flowchart of chatting:-

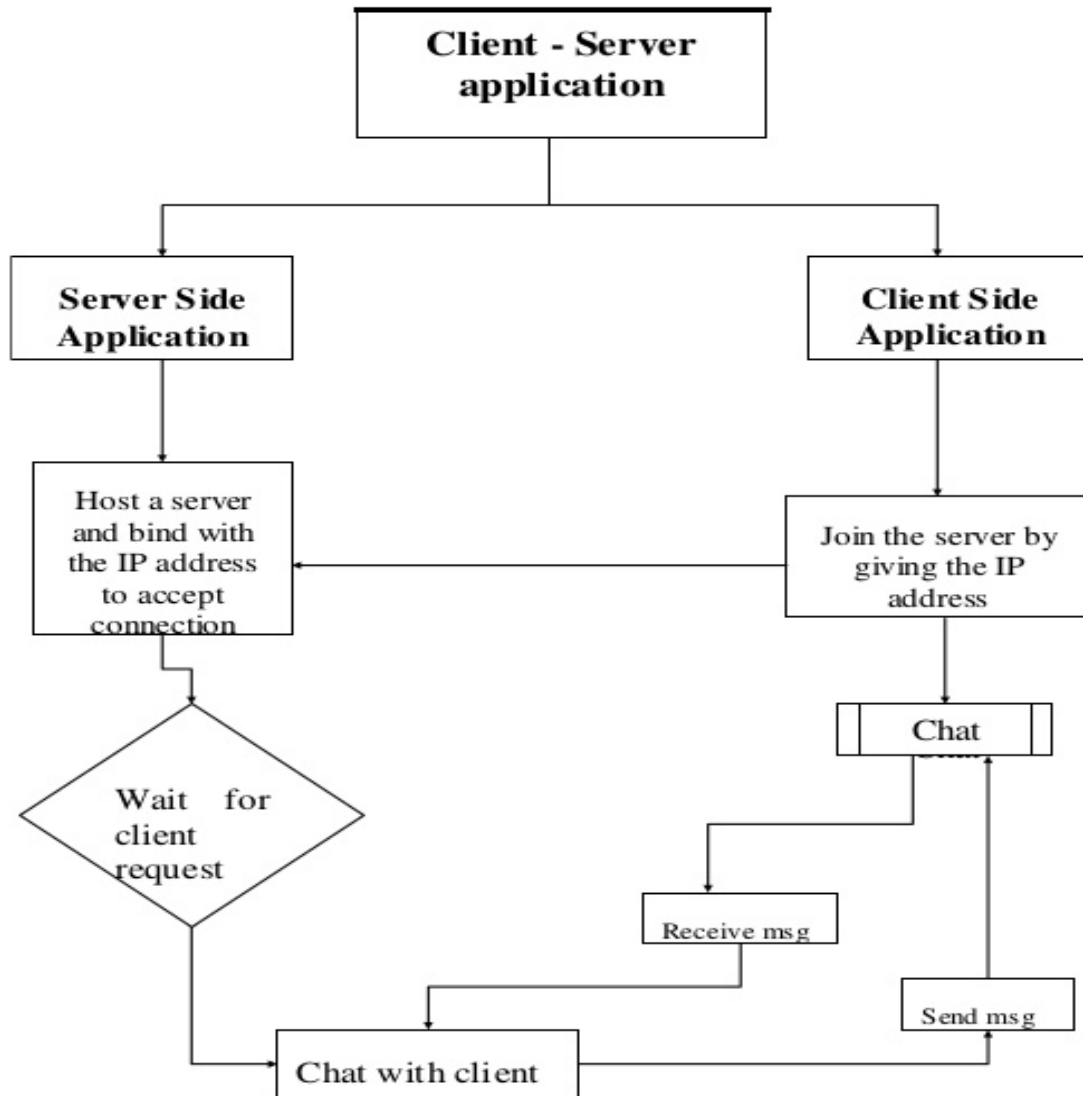


Figure 5.2 Chat Flowchart

The chat flowchart express that if the host and client start on wifi and discover the peers than they will be connected after establish of connection they can start chatting and they are connected by Wi-Fi P2p so that it is only two pipeline. The server and client can exchange data using buffers.

CODE AND IMPLEMENTATION

6.1 Java Modules and Libraries:-

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.IntentFilter;
import android.net.InetAddresses;
import android.net.wifi.WifiManager;
import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pDeviceList;
import android.net.wifi.p2p.WifiP2pInfo;
import android.net.wifi.p2p.WifiP2pManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import org.w3c.dom.Text;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
```

Figure 6.1 Libraries

This the libraries used in this app creations:-

BroadcastReceiver:-

A *broadcast receiver (receiver)* is an Android component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens. the BroadcastReceiver is receiving an Intent broadcast. During this time you can use the other methods on BroadcastReceiver to view/modify the current result values.

Wi-Fi Manager:-

Wi-Fi manager allows applications to access to view the access the state of the wireless connections at very low level. Android provides Wi-Fi API through which applications can communicate with the lower-level wireless stack that provides Wi-Fi network access. Almost all information from the device supplicant is available, including the connected network's link speed, IP address, negotiation state, and more, plus information about other networks that are available. Some other API features include the ability to scan, add, save, terminate and initiate Wi-Fi connections.

Handler:-

A Handler allows you to send and process `Message` and `Runnable` objects associated with a thread's `MessageQueue`. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler it is bound to a `Looper`. It will deliver messages and runnables to that Looper's message queue and execute them on that Looper's thread.

Socket:-

A socket is an endpoint for communication between two machines. The actual work of the socket is performed by an instance of the `Socket` class. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall. The main difference between them is that a server socket is listening for

incoming connection requests. In this snippet I will try to show you a simple connection between an Android client device and a Java server app over a local network.

Wi-fiP2p Manager:-

WifiP2p manager provides the API for managing Wi-Fi peer-to-peer connectivity. This lets an application discover available peers, setup connection to peers and query for the list of peers. When a p2p connection is formed over Wi-Fi, the device continues to maintain the uplink connection over mobile or any other available network for internet connectivity on the device.

Toast:-

An Android Toast is a small message displayed on the screen, similar to a tool tip or other similar popup notification. A Toast is displayed on top of the main content of an activity, and only remains visible for a short time period.

AppCompatActivity:-

AppCompatActivity is a specific type of activity that allows you to use the support library action bar features. Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities

Adaptor View:-

Android Adapters (and AdapterViews) ... An AdapterView is a group of widgets (aka view) components in Android that include the ListView, Spinner, and GridView. In general, these are the widgets that provide the selecting capability in the user interface (read about AdapterView widgets here).

6.2 Code implement:-

Create a broadcast receiver for Wi-Fi P2P intents:-

A broadcast receiver allows you to receive intents broadcast by the Android system, so that your application can respond to events that you are interested in. The basic steps for creating a broadcast receiver to handle Wi-Fi P2P intents are as follows:

Create a class that extends the `BroadcastReceiver` class. For the class' constructor, you most likely want to have parameters for the `WifiP2pManager`, `WifiP2pManager.Channel`, and the activity that this broadcast receiver will be registered in. This allows the broadcast receiver to send updates to the activity as well as have access to the Wi-Fi hardware and a communication channel if needed.

In the broadcast receiver, check for the intents that you are interested in `onReceive()`. Carry out any necessary actions depending on the intent that is received. For example, if the broadcast receiver receives a `WIFI_P2P_PEERS_CHANGED_ACTION` intent, you can call the `requestPeers()` method to get a list of the currently discovered peers.

The following code shows you how to create a typical broadcast receiver. The broadcast receiver takes a `WifiP2pManager` object and an activity as arguments and uses these two classes to appropriately carry out the needed actions when the broadcast receiver receives an intent:

A `BroadcastReceiver` that notifies of important Wi-Fi p2p events.

`WIFI_P2P_CONNECTION_CHANGED_ACTION`

Applications can use `requestConnectionInfo()`, `requestNetworkInfo()`, or `requestGroupInfo()` to retrieve the current connection information.

`WIFI_P2P_THIS_DEVICE_CHANGED_ACTION`

Applications can use `requestDeviceInfo()` to retrieve the current connection information.

```

package com.example.blabme;

import ...

public class WifiDirectBroadcastReceiver extends BroadcastReceiver {
    private WifiP2pManager mmanager;
    private WifiP2pManager.Channel mchannel;
    private Chat mactivity;

    public WifiDirectBroadcastReceiver(WifiP2pManager mmanager, WifiP2pManager.Channel mchannel, Chat mactivity) {
        this.mmanager=mmanager;
        this.mchannel=mchannel;
        this.mactivity=mactivity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action=intent.getAction();
        if(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)){
            int state=intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);

            if(state==WifiP2pManager.WIFI_P2P_STATE_ENABLED){
                Toast.makeText(context, "WiFi is ON", Toast.LENGTH_SHORT).show();
            }else{
                Toast.makeText(context, "WiFi is OFF", Toast.LENGTH_SHORT).show();
            }
        }
        else if(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)){
            if(mmanager!=null){
                mmanager.requestPeers(mchannel,mactivity.peerListListener);
            }
        }
        else if(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)){
            if(mmanager==null){
                return;
            }
            NetworkInfo networkInfo=intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
            if(networkInfo.isConnected()){
                mmanager.requestConnectionInfo(mchannel,mactivity.connectionInfoListener);
            }else{
                mactivity.connectionstatus.setText("Device Disconnected");
            }
        }
        else if(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)){
            //do something
        }
    }
}

```

Figure 6.2.1 WifiDirectBoardcastReciver

Wi-Fi On and Off:-

Exqlistener is the function which checks if wifi is enabled or not than set the wifi to on or off And tells user that its on or off.

```
private void exqlistener() {  
    btnonoff.setOnClickListener((v) -> {  
        if (wifiManager.isWifiEnabled()) {  
            wifiManager.setWifiEnabled(false);  
            btnonoff.setText("Wifi ON");  
        } else {  
            wifiManager.setWifiEnabled(true);  
            btnonoff.setText("Wifi OFF");  
        }  
    });  
}
```

Figure 6.2.2 WiFiOn_OFF

Discovering Peers:-

This method finds the nearest peers and gives a list of peers in the device list and the selection on the peer can be done based on the name of the peer.

```
WifiP2pManager.PeerListListener peerListListener=new WifiP2pManager.PeerListListener() {  
    @Override  
    public void onPeersAvailable(WifiP2pDeviceList peerlist) {  
        if (!peerlist.getDeviceList().equals(peers)) {  
            peers.clear();  
            peers.addAll(peerlist.getDeviceList());  
  
            devicenamearray=new String[peerlist.getDeviceList().size()];  
            devicearray=new WifiP2pDevice[peerlist.getDeviceList().size()];  
            int index=0;  
            for(WifiP2pDevice device:peerlist.getDeviceList()){  
                devicenamearray[index]=device.deviceName;  
                devicearray[index]=device;  
                index++;  
            }  
            ArrayAdapter<String> adapter =new ArrayAdapter<>(getApplicationContext(),android.R.layout.simple_list_item_1,devicenamearray);  
            listview.setAdapter(adapter);  
        }  
        if(peers.size()==0){  
            Toast.makeText(getApplicationContext(), text: "NO Device Found",Toast.LENGTH_SHORT).show();  
        }  
    }  
};
```

Figure 6.2.3 Discovering Peers

Connect to peer:-

When we click on peer it start connecting to peer's with wifi connected and on success of connection it shows "Connected to device (peer name) Successfully" on failure it shows " Not connected".

```
listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?> adapterView, View view, int position, long id) {  
  
        final WifiP2pDevice device=devicearray[position];  
        WifiP2pConfig config=new WifiP2pConfig();  
        config.deviceAddress=device.deviceAddress;  
  
        mmanager.connect(mchannel, config, new WifiP2pManager.ActionListener() {  
            @Override  
            public void onSuccess() {  
                Toast.makeText(getApplicationContext(), text: "Connected to"+device.deviceName,Toast.LENGTH_SHORT).show();  
            }  
  
            @Override  
            public void onFailure(int i) {  
                Toast.makeText(getApplicationContext(), text: "Not Connected",Toast.LENGTH_SHORT).show();  
            }  
        });  
    }  
});
```

Figure 6.2.4 Connecting Peers

Server Thread:-

It create the host or sever thread and assign a port number for the server and it start accepts message and send message form sendreceive class.

```
public class serverclass extends Thread{  
    Socket socket;  
    ServerSocket serverSocket;  
  
    @Override  
    public void run() {  
        try{  
            serverSocket =new ServerSocket( port: 8888);  
            socket=serverSocket.accept();  
            sendrecv=new sendrecv(socket);  
            sendrecv.start();  
        }  
        catch (IOException e){  
            e.printStackTrace();  
        }  
    }  
}
```

Figure 6.2.5 Host/server Thread

Client Thread:-

Client thread gets the host address and start connecting to the server thread by the host port (Wi-Fi direct), and start receiving data are can also send data by sendreceive thread.

```
public class clientClass extends Thread{
    Socket socket;
    String getHostAddress;

    public clientClass(InetAddress hostAddress){
        getHostAddress=hostAddress.getHostAddress();
        socket =new Socket();
    }

    @Override
    public void run() {
        try {
            socket.connect(new InetSocketAddress(getHostAddress, port: 8888), timeout: 500);
            sendreceive=new sendreceive(socket);
            sendreceive.start();
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

Figure 6.2.6 Client Thread

Send/Receive Thread:-

```
Handler handler= new Handler(new Handler.Callback() {
    @Override
    public boolean handleMessage(@NonNull Message msg) {
        switch(msg.what){
            case messageread:
                byte[] readbuff= (byte[]) msg.obj;
                String tempmsg=new String(readbuff, offset: 0,msg.arg1);
                readmsgbox.setText(tempmsg);
                break;
        }
        return true;
    }
});
```

```

public sendreceive(Socket skt){
    socket=skt;
    try {
        inputStream=socket.getInputStream();
        outputStream=socket.getOutputStream();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void run() {
    byte[] buffer=new byte[1024];
    int bytes;
    while (socket!=null){
        try {
            bytes = inputStream.read(buffer);
            if(bytes>0){
                handler.obtainMessage(messageread,bytes, arg2: -1,buffer).sendToTarget();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public void write (byte[] bytes){
    try {
        outputStream.write(bytes);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

Figure 6.2.7 Send/Receive Thread

The send/receive method is called for data to be sent and receive from client to server, Handler is the method which handles the data to receive and send handler will store the data temporarily. It checks that if a message been written the data is transfer it is done by write function.

File sharing:-

Once you have set up your app to share files using content URIs, you can respond to other apps' requests for those files. One way to respond to these requests is to provide a file selection interface from the server app that other applications can invoke. This approach allows a client application to let users select a file from the server app and then receive the selected file's content URI.

Receive file requests:-

To receive requests for files from client apps and respond with a content URI, your app should provide a file selection activity. Client apps start this Activity by calling `startActivityForResult()` with an Intent containing the action `ACTION_PICK`. When the client app calls `startActivityForResult()`, your app can return a result to the client app, in the form of a content URI for the file the user selected. Next, define an Activity subclass that displays the files available from your app's `files/images/` directory in internal storage and allows the user to pick the desired file. The following snippet demonstrates how to define this Activity and respond to the user's selection:

```
public class MainActivity extends Activity {
    private File privateRootDir;
    private File imagesDir;
    File[] imageFiles;
    String[] imageFileNames;
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        resultIntent =
            new Intent("com.example.myapp.ACTION_RETURN_FILE");
        privateRootDir = getFilesDir();
        imagesDir = new File(privateRootDir, "images");
        imageFiles = imagesDir.listFiles();
        setResult(Activity.RESULT_CANCELED, null);        ...
    }
    ...
}
```

Figure 6.2.8 Receive File Request

Respond to a file selection:-

Once a user selects a shared file, your application must determine what file was selected and then generate a content URI for the file. Since the Activity displays the list of available files in a ListView, when the user clicks a file name the system calls the method `onItemClick()`, in which you can get the selected file. Remember that you can only generate content URIs for files that reside in a directory you've specified in the meta-data file that contains the `<paths>` element, as described in the section `Specify sharable directories`. If you call `getUriForFile()` for a File in a path that you haven't specified, you receive an `IllegalArgumentException`.

```
protected void onCreate(Bundle savedInstanceState) {  
    listView.setOnItemClickListener(  
        new AdapterView.OnItemClickListener() {  
            @Override  
            public void onItemClick(AdapterView<?> adapterView,  
                                    View view,  
                                    int position,  
                                    long rowId) {  
                File requestFile = new File(imageFilename[position]);  
  
                try {  
                    fileUri = FileProvider.getUriForFile(  
                        MainActivity.this,  
                        "com.example.myapp.fileprovider",  
                        requestFile);  
                } catch (IllegalArgumentException e) {  
                    Log.e("File Selector",  
                        "The selected file can't be shared: " + requestFile.toString());  
                }  
            }  
        });  
}
```

Figure 6.2.9 Respond File selection

Grant permissions for the file:-

Now that you have a content URI for the file you want to share with another app, you need to allow the client app to access the file. To allow access, grant permissions to the client app by adding the content URI to an Intent and then setting permission flags on the Intent. The permissions you grant are temporary and expire automatically when the receiving app's task stack is finished.

```

protected void onCreate(Bundle savedInstanceState) {
    fileListView.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int position,
                long rowId) {
                if (fileUri != null) {
                    resultIntent.addFlags(
                        Intent.FLAG_GRANT_READ_URI_PERMISSION);
                }
            }
        });
}

```

Figure 6.2.10 Grant File Permission

Share the file with the requesting app:-

To share the file with the app that requested it, pass the Intent containing the content URI and permissions to setResult(). When the Activity you have just defined is finished, the system sends the Intent containing the content URI to the client app.

```

protected void onCreate(Bundle savedInstanceState) {
    fileListView.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int position,
                long rowId) {

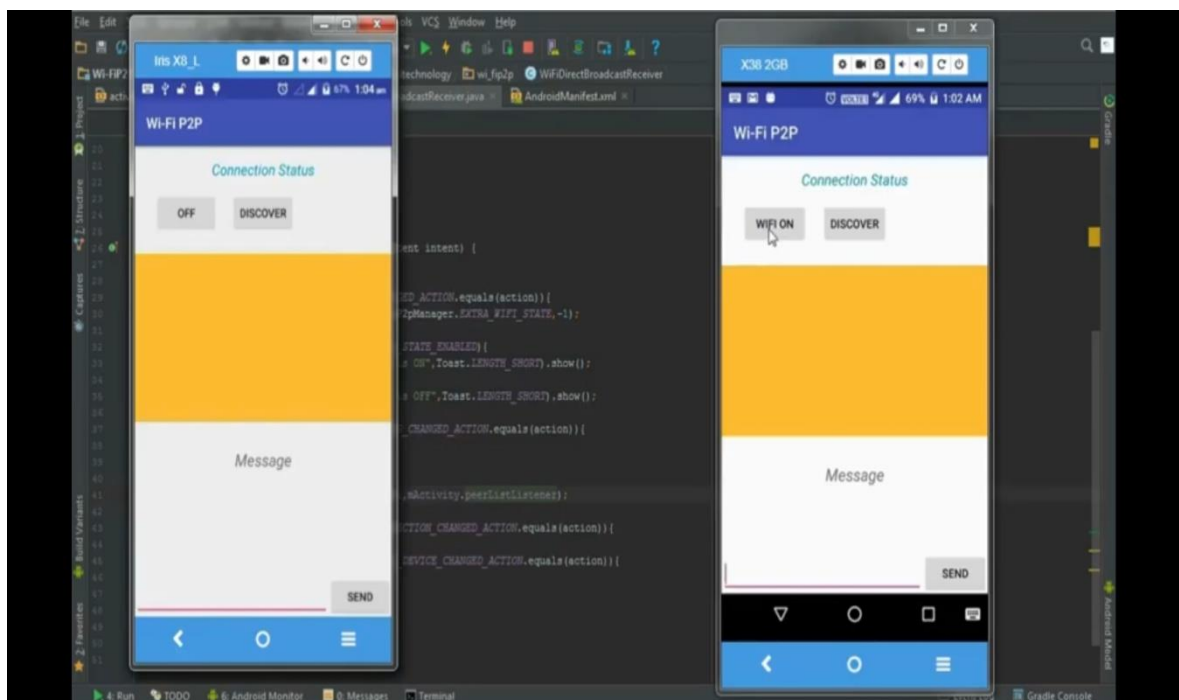
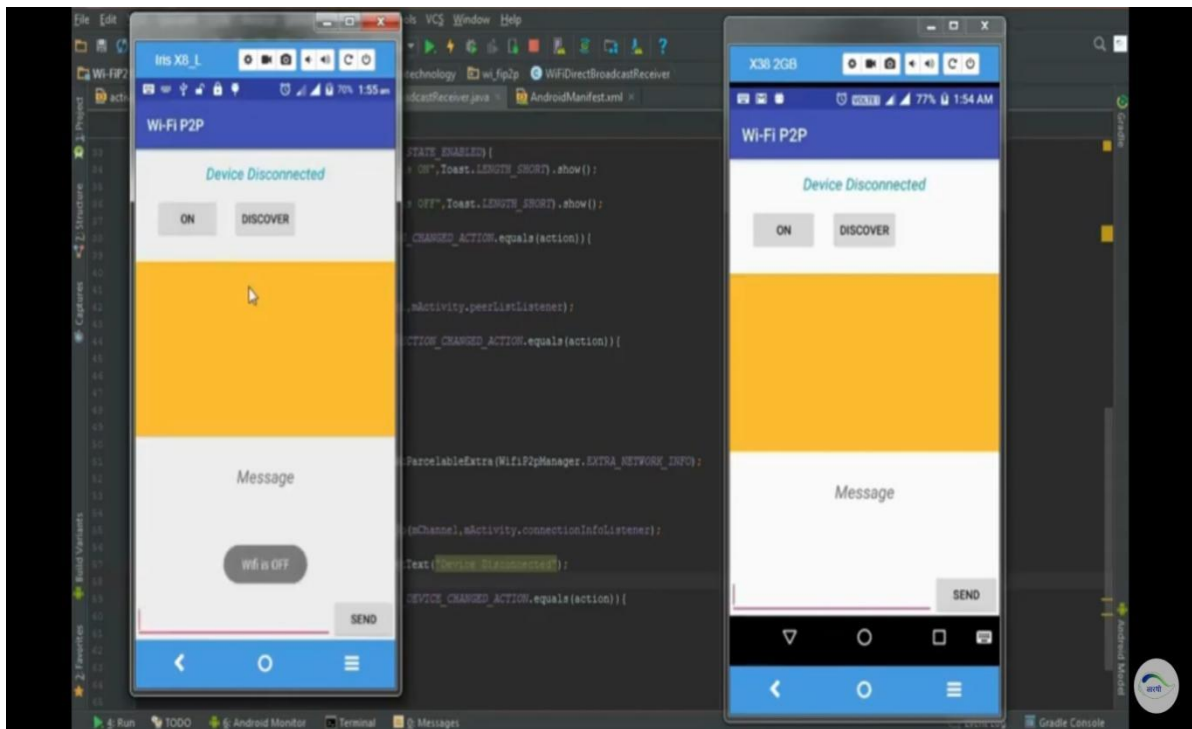
                if (fileUri != null) {
                    resultIntent.setDataAndType(
                        fileUri,
                        getContentResolver().getType(fileUri));
                    MainActivity.this.setResult(Activity.RESULT_OK,
                        resultIntent);
                } else {
                    resultIntent.setDataAndType(null, "");
                    MainActivity.this.setResult(RESULT_CANCELED,
                        resultIntent);
                }
            }
        });
}

```

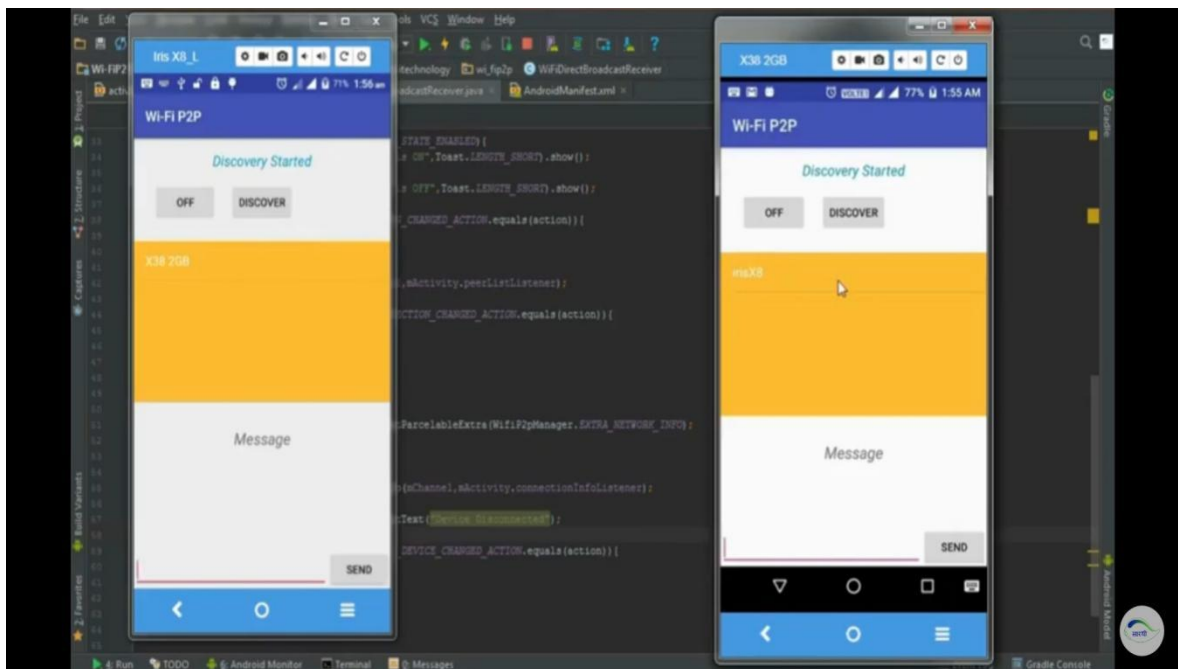
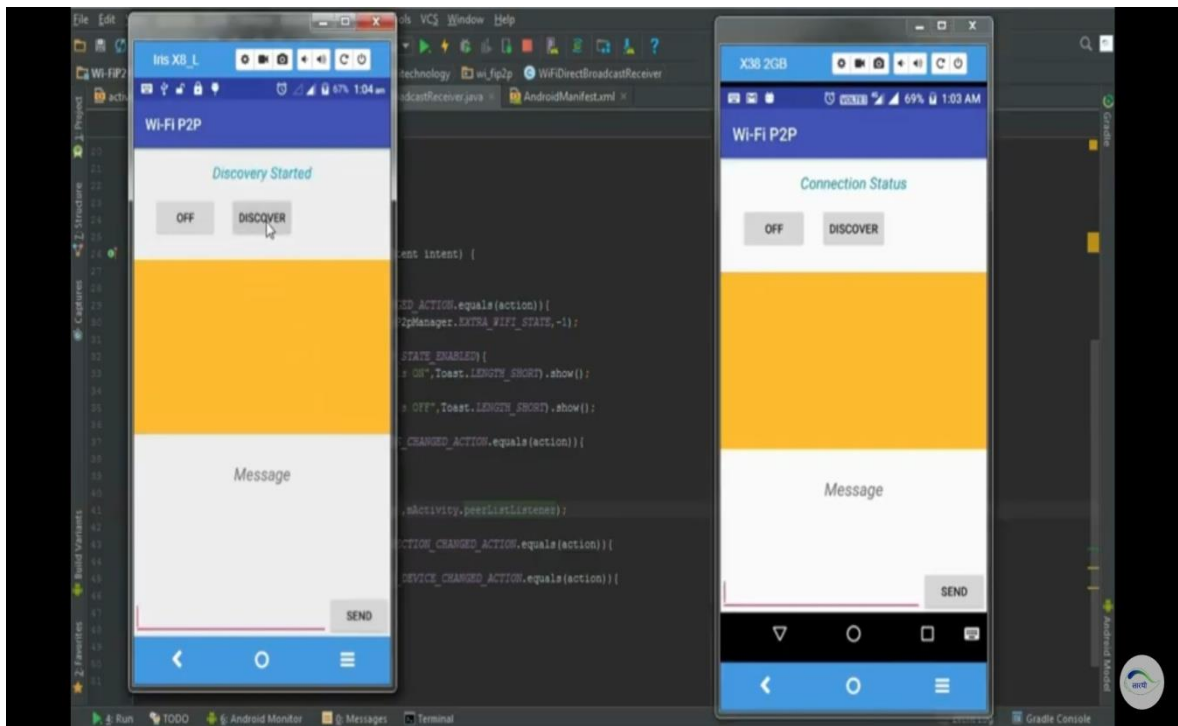
Figure 6.2.11 Share file

OUTPUT AND CONCLUSION

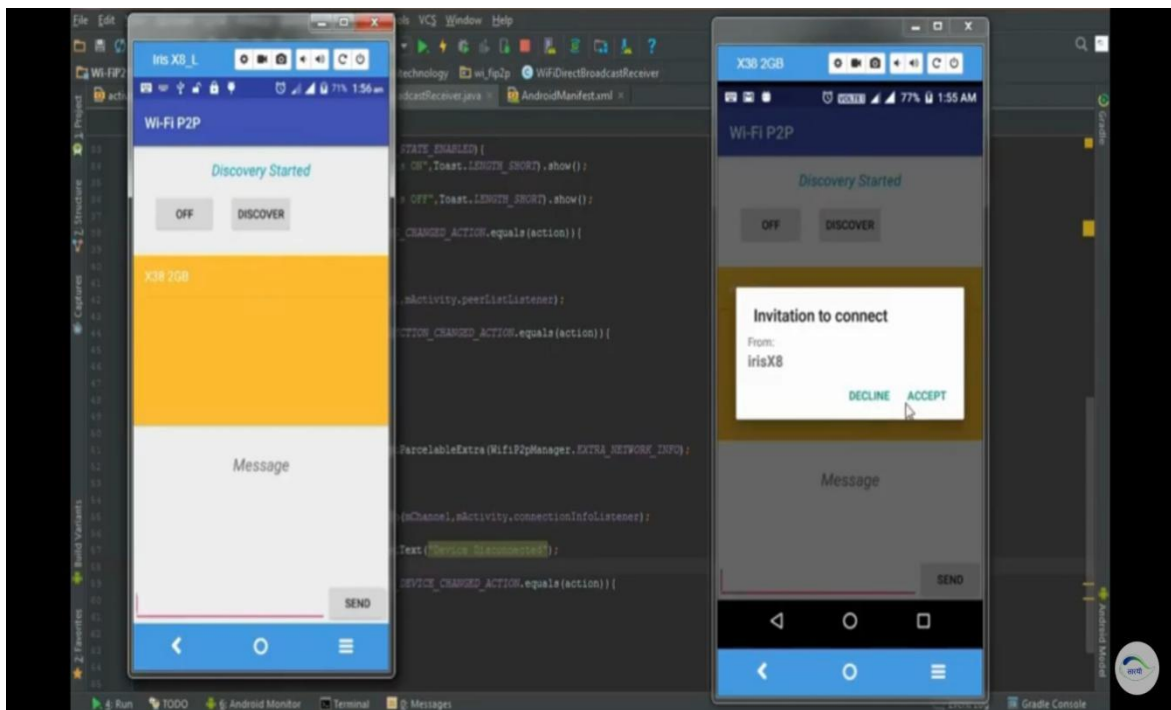
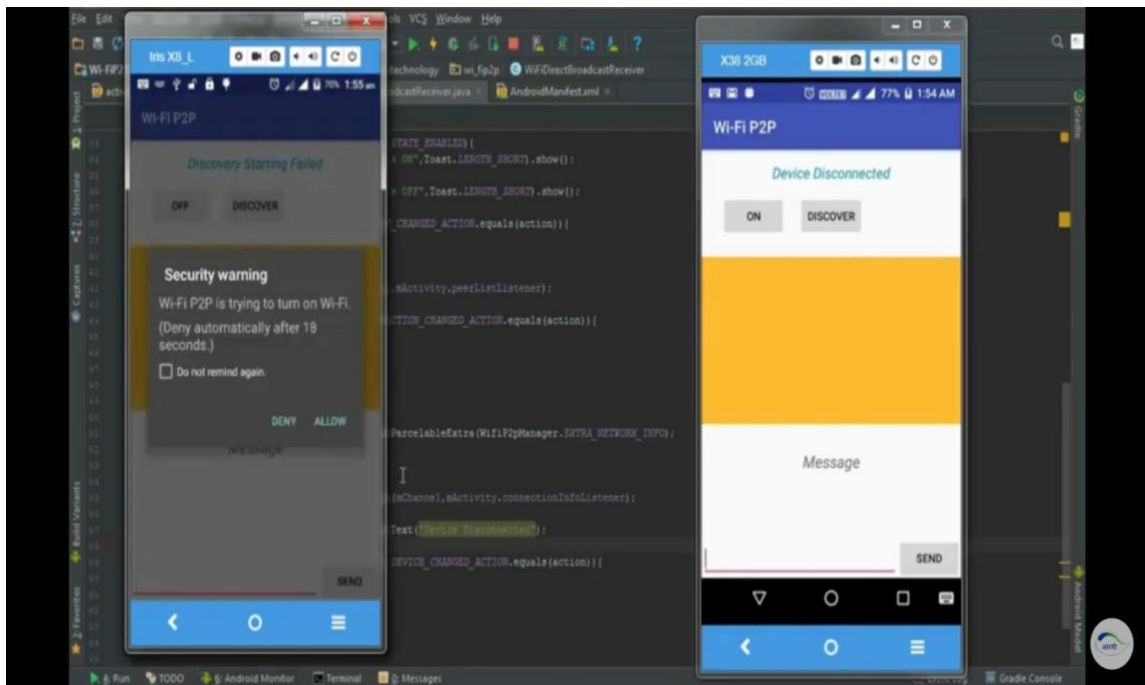
7.1 Outputs Of APP:-



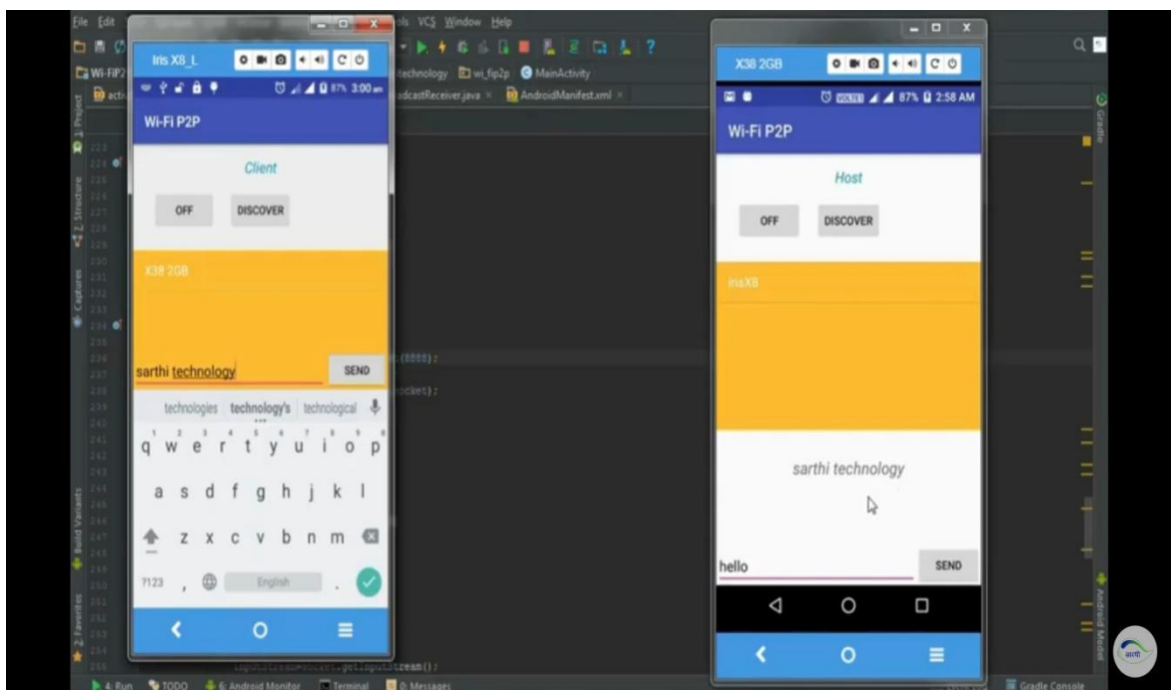
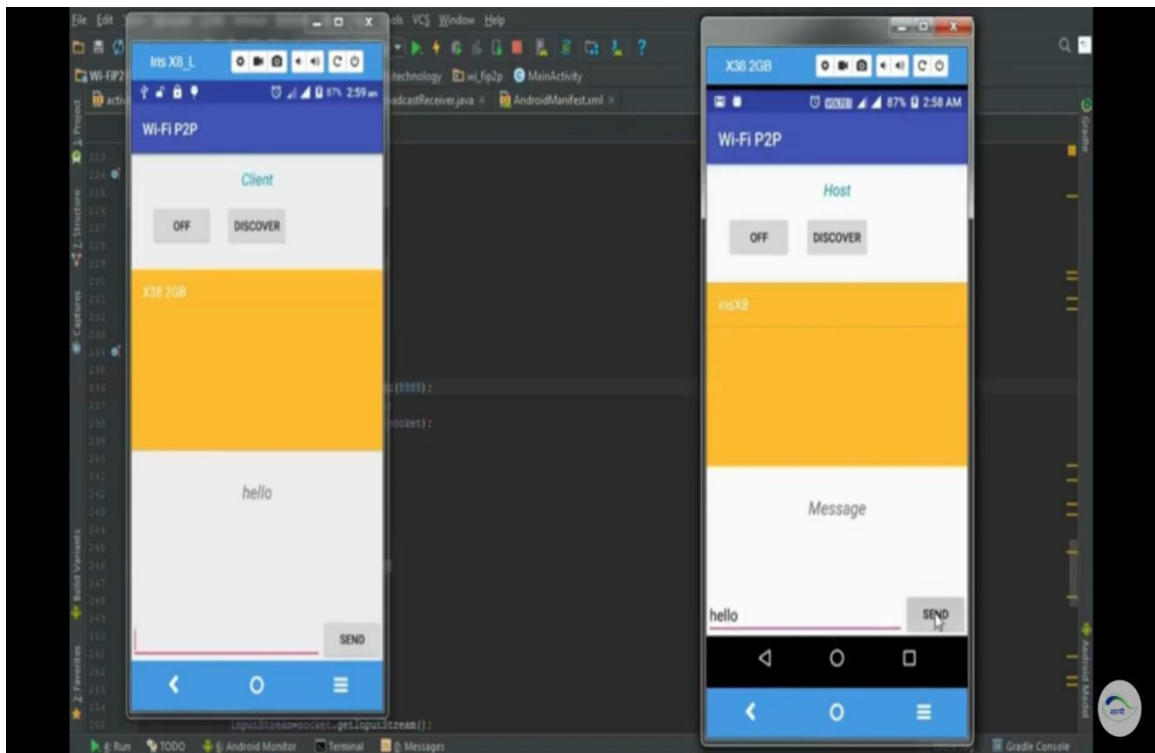
On the Wi-Fi to discover peers on both the server and the client



Discovering peers on both devices by clicking on discover button



Connecting the server and client using Wi-Fi direct



Chatting between server and client and can transfer files

7.2 Conclusion:-

- This file transfer and chat project in android studio can be successfully operated in real environment and can be tested practically.
- The files and chat messages can be sent to the selected clients.
- All the activities provide a feeling like an easy walk over to the user who is interfacing with the system.
- I would like to conclude that the System has been launched successfully in the real environment, and is tested practically.
- The Files and Messages are transferred to the intended clients basing on the selection. Even .EXE files are also been transferred accurately without any errors and perfectly executed after downloading.

BIBLIOGRAPHY

1. <https://developer.android.com/studio/intro/>
2. <https://www.wikipedia.org/>
3. <https://www.instructables.com/id/How-To-Create-An-Android-App-With-Android-Studio/>