AndroCom: Communication Without Internet

Wasia	**BCS203233**
Muhammad Harris	**BCS203193**
Umer Ahmed	**BCS203182**

Fall - 2023

Supervised By
Mr. Bilal Ahmed

Department of Computer Science
Capital University of Science & Technology, Islamabad

Submission Form for Final-Year

PROJECT REPORT

Version	V 1

NUMBER OF MEMBERS	3

TITLE	AndroCom: Communication Without Internet

SUPERVISOR NAME	**Mr. Bilal Ahmed**

MEMBER NAME	REG. NO.	EMAIL ADDRESS
Muhammad Harris	BCS203193	Harris20014@gmail.com
Wasia	BCS203233	Wasiaibrar9892@gmail.com
Umer Ahmed	BCS203182	Umerahmed1000@gmail.com

MEMBERS' SIGNATURES

_____

_____

_____

Supervisor's Signature

APPROVAL CERTIFICATE

This project, entitled as “AndroCom – Communication Without Internet” has been approved for the award of

Bachelors of Science in Computer Science

Committee Signatures:

Supervisor: _____

(Mr. Bilal Ahmed)

Project Coordinator: _____

(Mr. Bilal Ahmed)

Head of Department: _____

(Dr. Abdul Basit)

DECLARATION

We, hereby, declare that "No portion of the work referred to, in this project has been submitted in support of an application for another degree or qualification of this or any other university/institute or other institution of learning". It is further declared that this undergraduate project, neither as a whole nor as a part thereof has been copied out from any sources, wherever references have been provided.

MEMBERS' SIGNATURES

ACKNOWLEDGEMENTS

We express our deepest gratitude to God for granting us the ability to successfully complete this project. Our sincere thanks go to our supervisor, Mr. Bilal Ahmad, whose invaluable guidance has been instrumental in shaping and ensuring the success of this endeavor. His insightful suggestions and instructions have been pivotal throughout the project. Furthermore, we extend our heartfelt appreciation to our parents for their unwavering support and valuable guidance, which has been crucial in various phases of the project's completion.

Executive Summary

AndroCom is an innovative communication solution designed to address the challenges posed by limited or no internet access scenarios. In today's interconnected world, where traditional communication methods heavily rely on internet connectivity, AndroCom stands out by providing users with the ability to send text messages, make voice calls, and engage in video calls using a secure, end-to-end encrypted Android app. The uniqueness of AndroCom lies in its offline functionality, enabled through an AD HOC network created by a Raspberry Pi microcontroller. This approach ensures secure and efficient communication even in situations with no internet connection. The app caters to a wide range of scenarios, including emergency services, disaster recovery, and temporary gatherings, making it a versatile and valuable tool. With a focus on data privacy, security, and global accessibility, AndroCom represents a significant contribution to communication technology, offering users a reliable alternative in challenging connectivity situations.

Table of Contents

List of Figures

LIST OF TABLES

Chapter 1
Introduction

1.1 Project Introduction

In today's world, the internet has become such an integral part of our lives that if it were to go down tomorrow, most forms of communication, such as messaging, calls, and video communication, would cease to function. AndroCom is an Android app that enables its users to send text messages, make voice calls, and engage in video calls with complete end-to-end encryption when communicating with other users without the need of an internet connection.

AndroCom has significant market potential due to its unique features. It can work in places with no internet, help universities with daily tasks, and serve as a reliable backup during internet outages or emergencies. Its versatility and special capabilities make it valuable in various situations, making it an important tool in the market.

This functionality is implemented through an AD HOC network that is created using a microcontroller, specifically a Raspberry Pi, which serves as a critical component of the system. A server is created on the Raspberry Pi using Python, enabling packet transfer between the Raspberry Pi and the devices using AndroCom. This innovative setup ensures secure and efficient communication while bypassing the need for a traditional internet connection, addressing the challenges posed by internet interruptions or limited access scenarios.

1.2 Existing Examples / Solutions

At present, a noticeable gap exists in the market for apps that offer communication functionality independent of an internet connection. AndroCom, by enabling text messaging, voice calls and video calls with end-to-end encryption using Raspberry Pi, distinguishes itself as an innovative solution that fills this void. Unlike conventional applications that rely on internet connectivity, AndroCom offers users a novel approach to communication in scenarios where such connectivity may be unavailable or limited, addressing a critical need in today's interconnected world.

1.3 Business Scope

The business scope of AndroCom is promising, offering a unique solution for communication in scenarios with limited or no internet access. It caters to a niche market and educational institutions, presents a valuable tool for disaster recovery and emergency services, and has the potential to serve as a backup communication service during internet outages. With its AD HOC networking capabilities, it can find use in various temporary gathering scenarios. The app's focus on data privacy and security also appeals to users prioritizing secure communication, while its potential global reach ensures a broad user base.

1.4 Useful Tools and Technologies

Following is a list of technologies that are used for designing, development and testing phases of the project:

- Kotlin
- Java
- Android Studio
- Figma
- Raspberry Pi
- Python

In our application development, we will employ a hybrid approach, primarily utilizing Kotlin for its modern features and conciseness, while also integrating Java where necessary for specific algorithms, socket programming and several modules. Android Studio will serve as our development environment of choice, offering a comprehensive set of tools for efficient coding and testing. Notably, our app will not rely on an internet connection due to its offline functionality. An AD HOC network is created using Raspberry Pi with a server implemented in Python.

1.5 Project Work Breakdown

A project work breakdown diagram is a way to break down a complex project into smaller, more manageable tasks. The project work breakdown for the AndroCom is given in *Figure 1.1*.

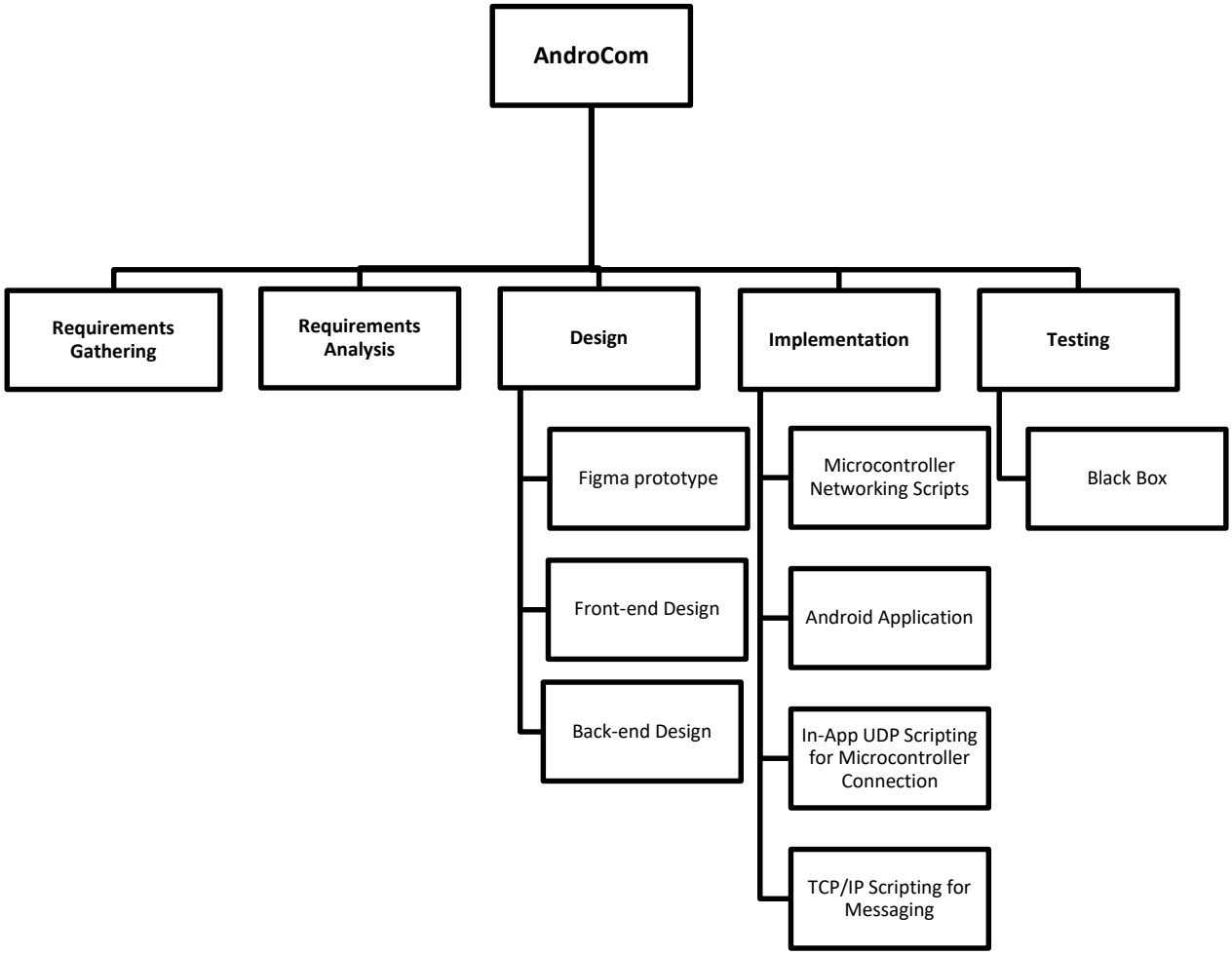

Figure 1.1: Project Work Breakdown

1.6 Project Timeline

A project timeline diagram is a visual representation of the tasks and milestones in a project, showing their start and end dates. In simple words, it's a bar chart that shows when things need to happen in order for your project to finish on time. The project timeline for AndroCom is given in *Figure 1.2*.

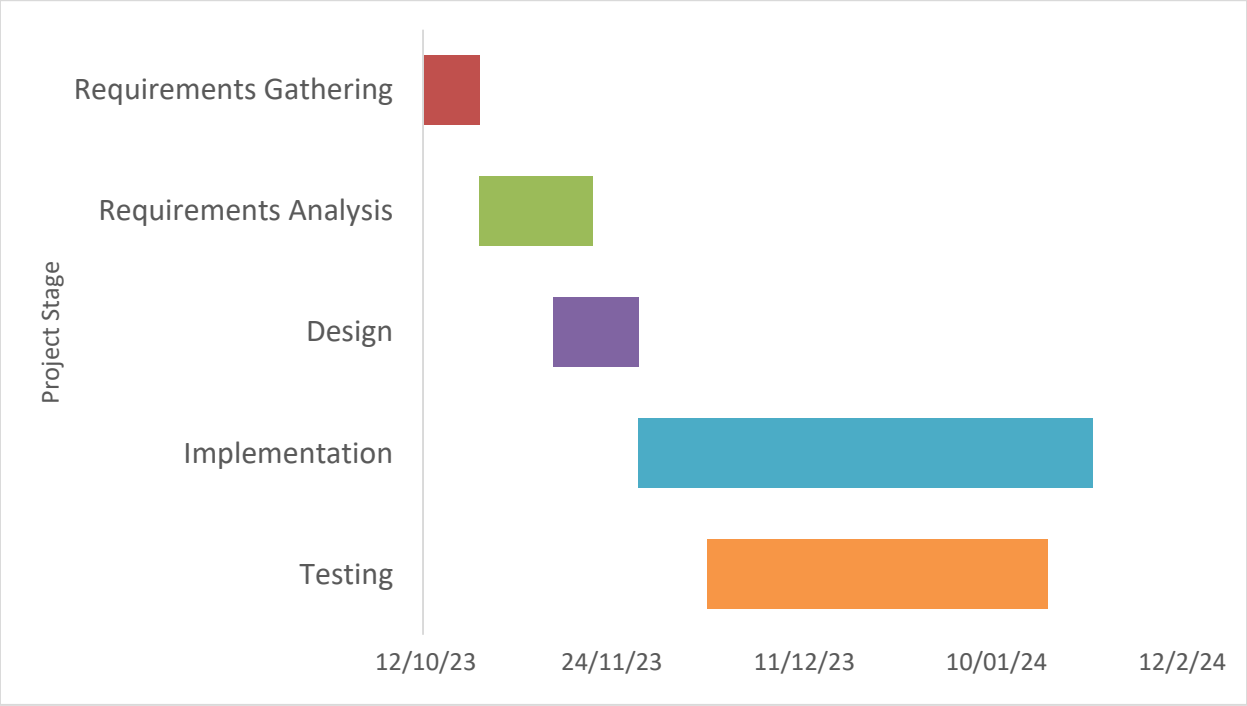

Figure 1.2: Project Time Line

Chapter 2
Requirement Specification and Analysis

This chapter documents the specification and analysis of requirements for *AndroCom*. We have systematically sorted out the development requirements for the software, emphasizing the importance of identifying every small detail at this stage to prevent future software alterations.

This chapter covers the following specifications for the required software:

- Functional & Non-Functional Requirements
- Use Case Diagram
- Brief Description of Each Use Case
- Detailed Sequence Diagram for Each Use Case
- Domain Model
- System Architecture

2.1 Functional Requirements

Functional requirements are what a system must do to meet user needs. The functional requirements for AndroCom are given in *Table 2.1*.

Table 2.1: Functional Requirements

S. No.	Functional Requirement	Type	Status
1	Configured microcontroller for network connection	Core	Complete
2	User profile setup in application	Core	Complete
3	List of active users connected with the network	Core	Complete
4	Block or unblock users	Intermediate	Complete
5	Mute messages and chat notifications of users	Intermediate	Complete
6	Text Messages with active users	Core	Complete
7	Voice Calls with active users	Core	Incomplete
8	Video Calls with active users	Core	Incomplete
9	Mute mic or turn-off camera when in call	Intermediate	Incomplete

10	Multicast broadcast by creating groups	Intermediate	Incomplete
11	Traffic Prioritization	Intermediate	Incomplete

2.2 Non-Functional Requirements

Non-functional requirements are the constraints on how a system should work. The non-functional requirements for AndroCom are given in *Table 2.2.*

Table 2.2: Functional and Non-Functional Requirement

S. No.	Non-Functional Requirements	Category
1	Prompt when connected to the wrong network	Security
2	De-authorize unauthorized users	Security
3	End-to-End text encryption	Security
4	User friendly UI	Usability
5	View last seen time in active users list	Usability
6	View Signal Strength with network	Performance
7	Optimize resource usage on Microcontroller	Performance
8	Low consumption and Lightweight application	Performance

2.3 Selected Functional Requirements

The Selected function requirements of AndroCom for FYP Part-I are given in *Table 2.3.*

Table 2.3: Selected Functional Requirement

S. No.	Functional Requirement	Type
1	Configured microcontroller for network connection	Core
2	User profile setup in application	Core
3	List of active users connected with the network	Core
4	Block or unblock users	Intermediate
5	Mute message or call notifications of users	Intermediate
6	Text Messages with active users	Core
7	Voice Calls with active users	Core

2.4 System Use Case Modeling

A system use case diagram is a visual representation of the different ways that users can interact with a system. The system use case diagram of AndroCom is shown in *Figure 2.1.*

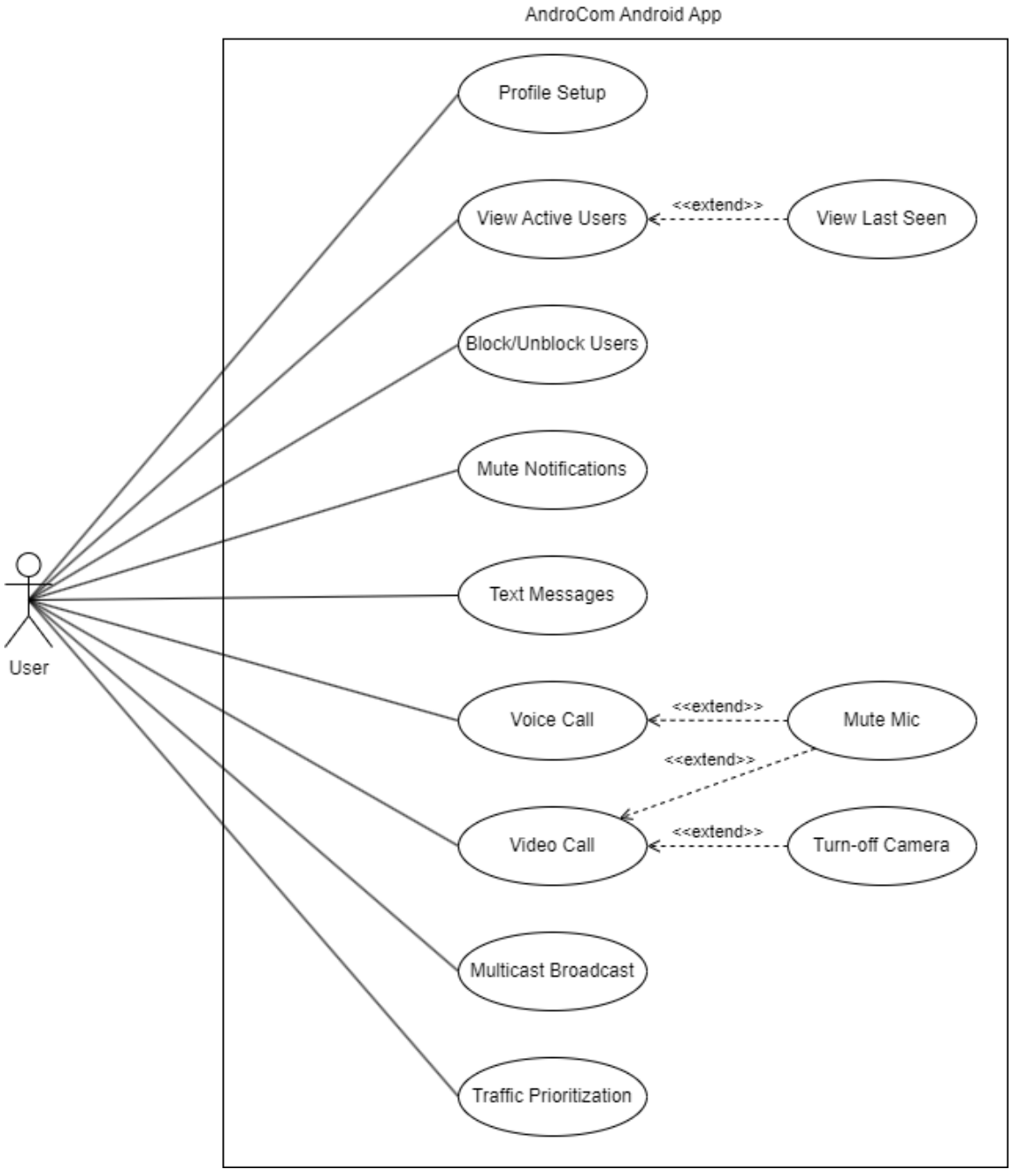

Figure 2.1: System Use Case Diagram

Use Case 1 (Configured Microcontroller for Network Connection):

Table 2.4: Use Case 1

Use Case ID:	UC1		
Use Case Name:	Configured Microcontroller for network connection		
Created By:	Wasia	**Last Updated By:**	Muhammad Harris
Date Created:	22-10-2023	**Last Revision Date:**	29-10-2023
Actors:	User		
Description:	The user will turn on the microcontroller and connect with its network on his android device		
Trigger:	The user wants to create an ad hoc network		
Preconditions:	The user has no internet and wants to use AndroCom for his communication		
Post conditions:	An ad hoc network will be created via the microcontroller		
Normal Flow:	User	System	
	1.User turns on the microcontroller	Microcontroller creates an ad hoc network	
	2.User connects with ad hoc network via his android device	User is connected with the network and is allowed to communicate on the network	
Alternative Flows:	None		
Exceptions:	1. Microcontroller doesn't turn on. 2. No network is created by the microcontroller. 3. Network doesn't show on available networks list.		

Use Case 2 (User Profile Setup):

Table 2.5: Use Case 2

Use Case ID:	UC2		
Use Case Name:	User profile setup		
Created By:	Umer Ahmed	**Last Updated By:**	Wasia
Date Created:	22-10-2023	**Last Revision Date:**	30-10-2023
Actors:	User		
Description:	The user will be presented with a sign-up screen where he enters their first name, last name and optionally uploads their profile picture in app.		
Trigger:	The user installs the application and wishes to complete their profile setup.		
Preconditions:	The user has successfully installed the application and is on the initial sign-up screen.		
Post conditions:	The user's profile information is saved and he can access, use their profile within application.		
Normal Flow:	User	System	
	1.User clicks get started button to request for sign-up.	The system provides a User initial page for profile setup.	
	2.User provide first name, last name and clicks continue.	The system re-direct the User to a newly created profile page.	
Alternative Flows:	The user cancels the profile setup.		
Exceptions:	1. The User has not filled the form correctly. 2. The system is not responding.		

Use Case 3 (List of active users connected with network):

Table 2.6: Use Case 3

Use Case ID:	UC3		
Use Case Name:	List of active users connected with network		
Created By:	Wasia	**Last Updated By:**	Umer Ahmed
Date Created:	22-10-2023	**Last Revision Date:**	30-10-2023
Actors:	User		
Description:	System will provide a list of active users who are currently connected to the network for monitoring and management purposes.		
Trigger:	The user selects the “Active Users” option from the application menu.		
Preconditions:	User is logged into the application and a connection is established.		
Post conditions:	The application displays a list of all active users connected to the network.		
Normal Flow:	User	System	
	1. The user selects the “Active Users” option from the application menu.	The system displays the list of active users to the user.	
Alternative Flows:	Network connection is not established and error message will be displayed.		
Exceptions:	Network server is unavailable.		

Use Case 4 (Text messages with active users):

Table 2.7: Use Case 4

Use Case ID:	UC4		
Use Case Name:	Text messages with active users		
Created By:	Muhammad Harris	**Last Updated By:**	Wasia
Date Created:	22-10-2023	**Last Revision Date:**	30-10-2023
Actors:	User		
Description:	User can send and receive text messages to other active users on the network.		
Trigger:	The user selects the chat icon from the application menu.		
Preconditions:	The user must be connected to network and is logged into the application.		
Post conditions:	The user is able to send and receive text messages to other active users on a network.		
Normal Flow:	User	System	
	1. The user selects the chat icon from application menu.	The system displays the chat section the application.	
	2.The user selects the recipient of a text message.	The system highlights selected recipient.	
	3. The user enters the text message and sends it.	The system send text message to the recipient over network.	
Alternative Flows:	Recipient is not active and text message is not delivered.		
Exceptions:	1. User not logged into the application 2. Network server is unavailable.		

Use Case 5 (Block or unblock users):

Table 2.8: Use Case 5

Use Case ID:	UC5		
Use Case Name:	Block and unblock users		
Created By:	Umer Ahmed	**Last Updated By:**	Wasia
Date Created:	22-10-2023	**Last Revision Date:**	30-10-2023
Actors:	User		
Description:	User can block and unblock other user within application. Blocking user will prevent further communication from or to the blocked user.		
Trigger:	The user selects the “Block or unblock users” option from the application menu.		
Preconditions:	User must be connected to network and is logged into the application interacting with the user whose status they want to change.		
Post conditions:	The selected user is either blocked or unblocked, as per the user’s action.		
Normal Flow:	User	System	
	1. The user selects the “Block or unblock users” from application menu.	The system displays a list of all active users on the network.	
	2.The user selects the user they want to block or unblock.	The system blocks or unblocks the selected user.	
Alternative Flows:	User is already blocked or selected user is not active.		
Exceptions:	1. User not logged into the application. 2.Network connection is not established.		

Use Case 6 (Voice call with active users):

Table 2.9: Use Case 6

Use Case ID:	UC6		
Use Case Name:	Voice call with active users		
Created By:	Umer Ahmed	**Last Updated By:**	Muhammad Harris
Date Created:	10-10-2023	**Last Revision Date:**	11-10-2023
Actors:	User		
Description:	User can initiate and receive a voice call from an active user within application.		
Trigger:	The user will press the call icon from the text chat section.		
Preconditions:	User must be connected to network and is logged into the application. The user has selected an active user from the list and indicated to make a voice call.		
Post conditions:	The user is able to a voice call with another active user on the network.		
Normal Flow:	User	System	
	1. The user selects the recipient of the voice call and initiates the voice call.	The system sends a voice call request to the recipient over the network.	
	2. The users are able to talk to each other over the voice connection.	The system establishes a voice connection between two users.	
Alternative Flows:	The recipient user rejects the voice call request and call is not established.		
Exceptions:	1. User not logged into the application. 2.Selected active user is no longer available or active.		

Use Case 7 (mute messages and call notifications of users):

Table 2.10: Use Case 7

Use Case ID:	UC7		
Use Case Name:	Mute messages and call notifications of users		
Created By:	Wasia	**Last Updated By:**	Muhammad Harris
Date Created:	10-10-2023	**Last Revision Date:**	19-10-2023
Actors:	User		
Description:	User can mute messages and call notifications of specific users within application.		
Trigger:	The user selects the “Mute messages and call notification” option from the settings or user preferences section.		
Preconditions:	User must be connected to network and is logged into the application. Also, user has identified specific users from whom they want to mute notifications.		
Post conditions:	The selected user’s message and call notifications are either muted or unmuted, as per user’s action.		
Normal Flow:	User	System	
	1. The user selects the “Mute messages and call notifications” option.	The system displays a list of all active users on the network.	
	2. The user selects the users whose messages and call notifications they want to mute.	The system mutes messages and call notifications from the selected user.	
Alternative Flows:	The selected user is already muted or a user is not active.		
Exceptions:	1. User not logged into the application and is not connected to the network.		

2.5 System Sequence Diagrams

A system use case sequence diagram is a visual representation of the steps involved in a particular use case of a system, showing the interactions between the different actors and components of the system. Following are the sequence diagrams for AndroCom application.

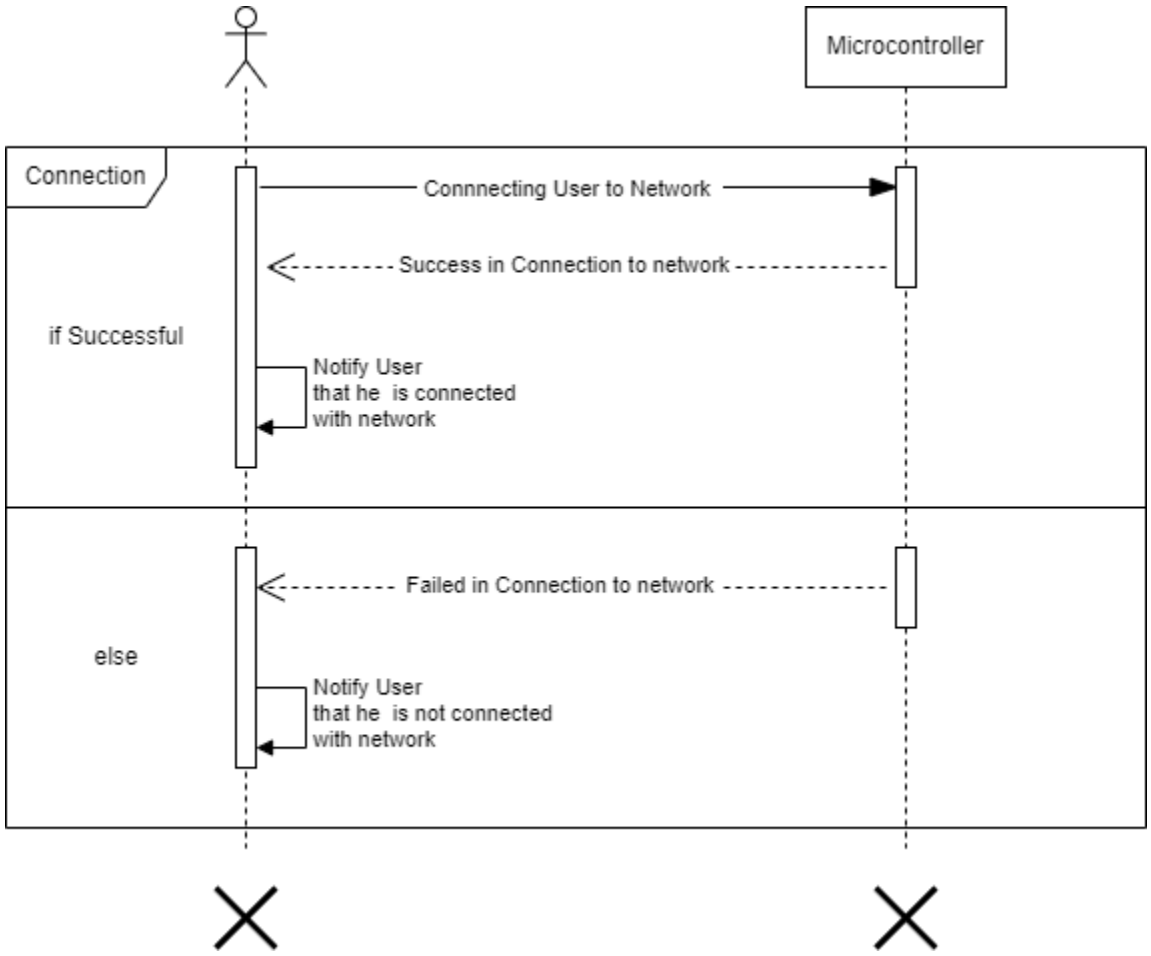

Figure 2.2: Establishing connection with microcontroller

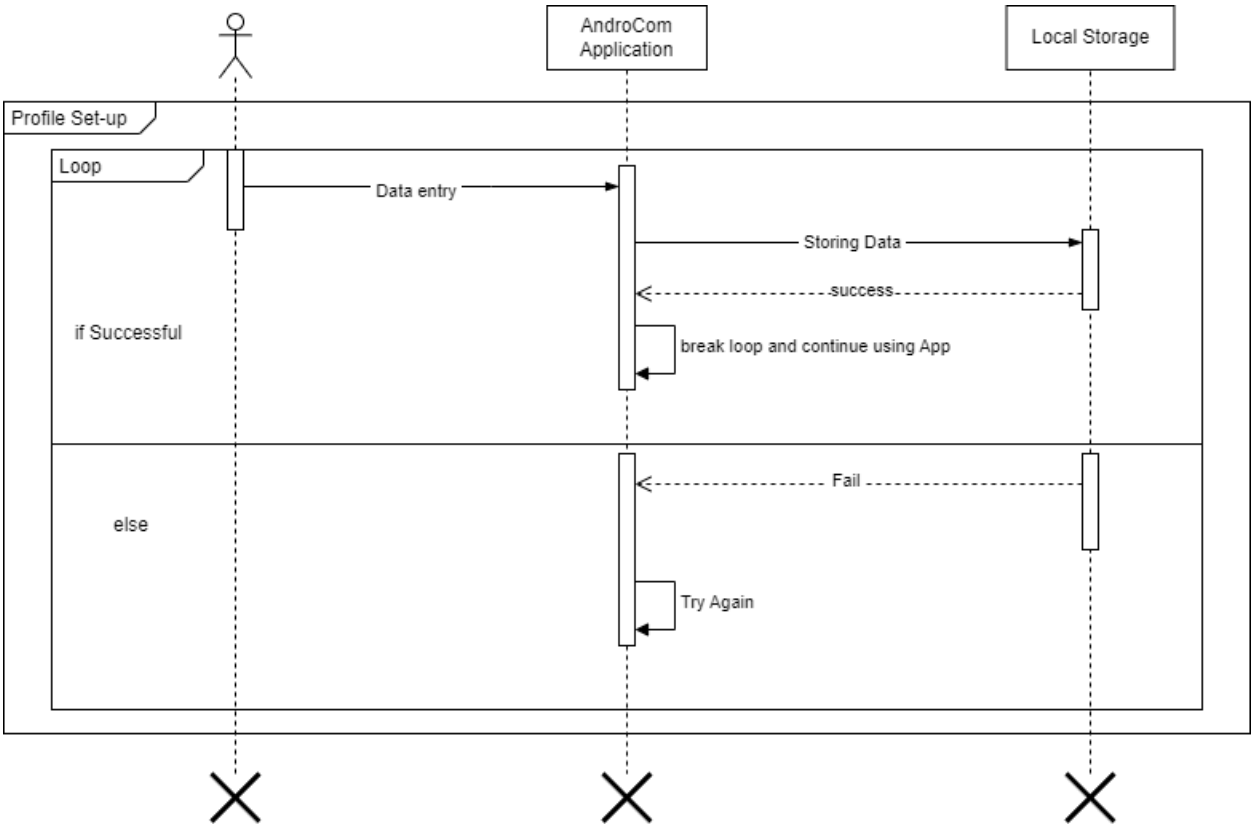

Figure 2.3: User profile setup

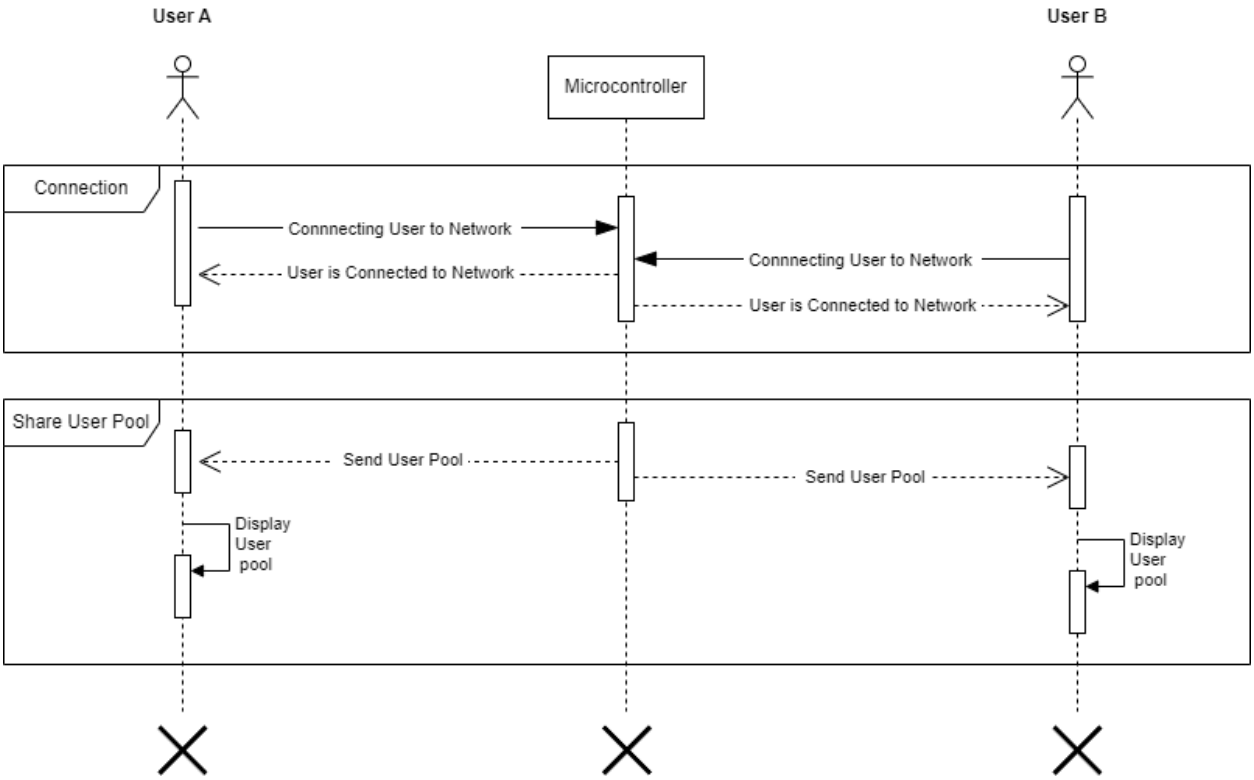

Figure 2.4: Sharing active user pool with user

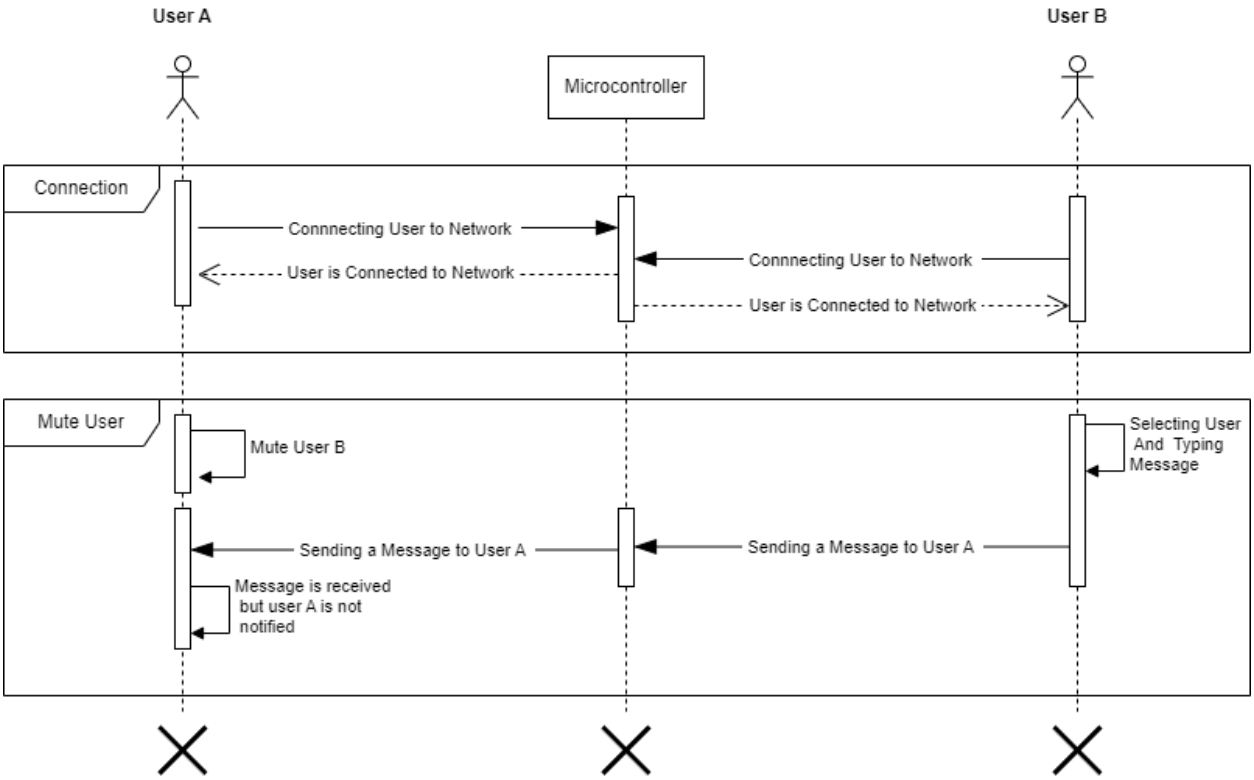

Figure 2.5: Mute user notifications

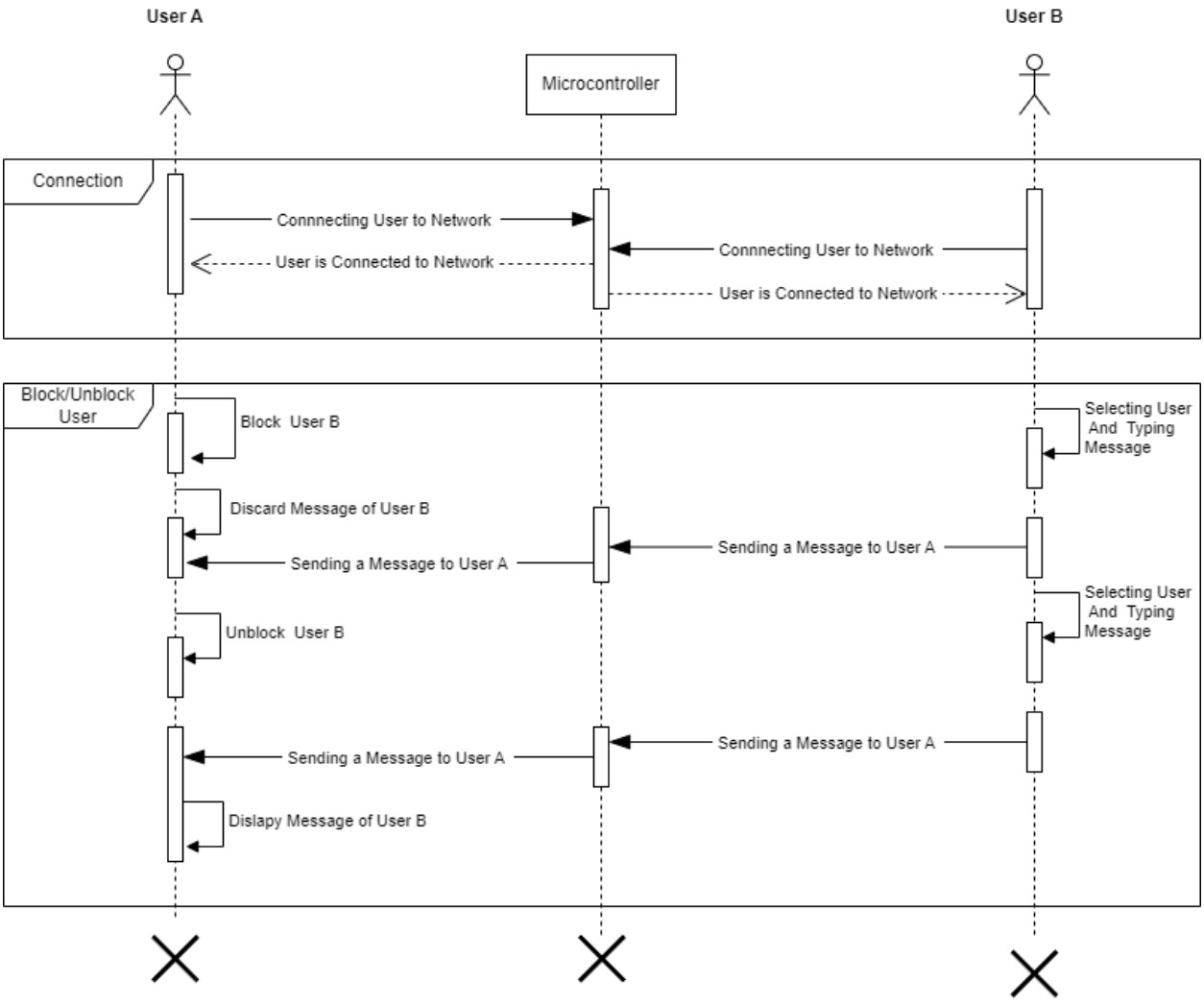

Figure 2.6: Block or Unblock other users

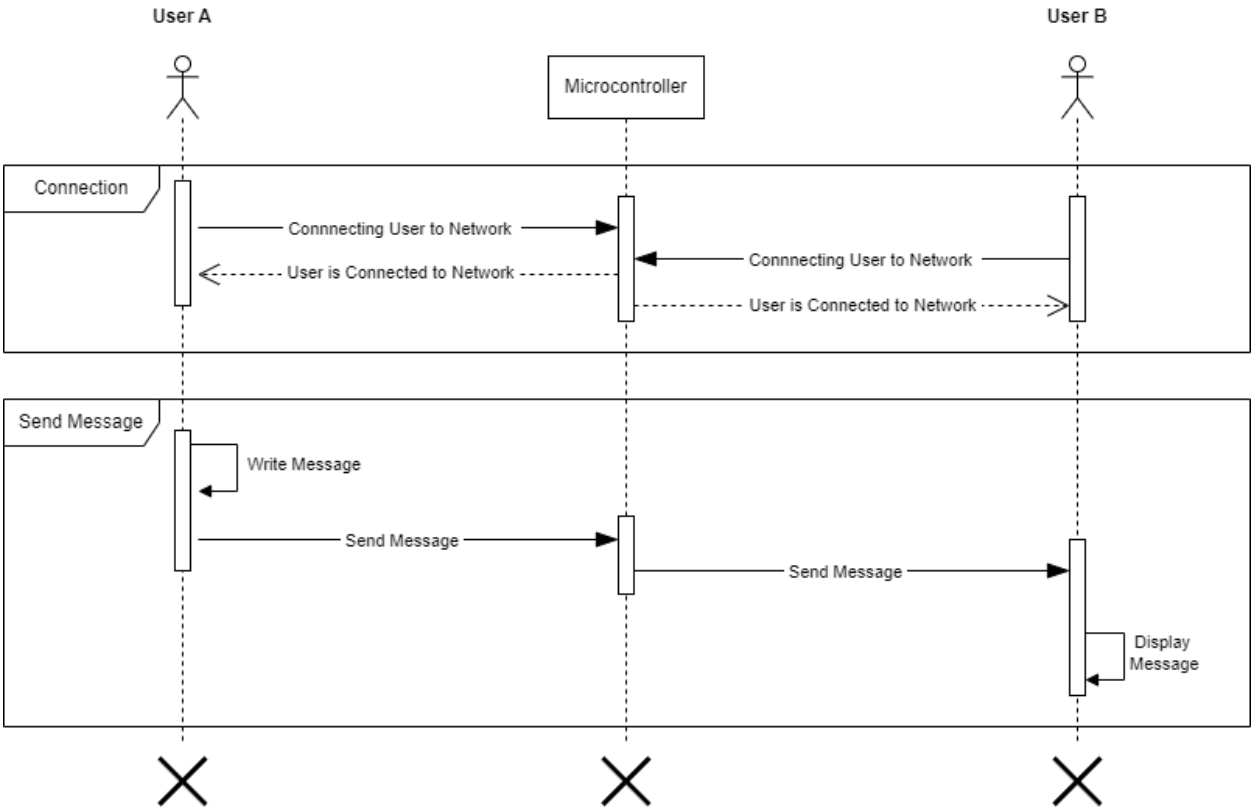

Figure 2.7: Send text message to other users

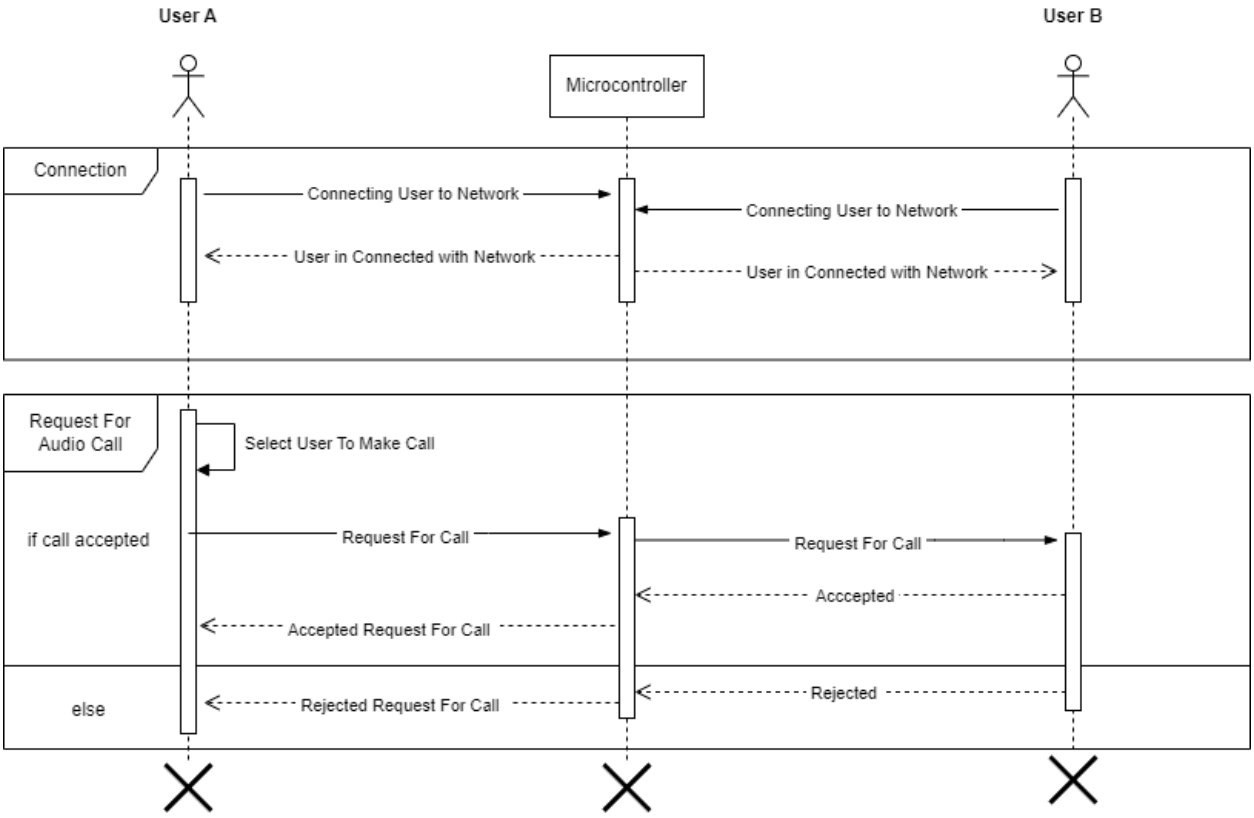

Figure 2.8: Making a voice call

2.6 Domain Model

A domain model is a conceptual map of the key concepts and relationships in a problem domain. Domain models are used to help developers understand and design software. The domain model for AndroCom is given below in *figure 2.9.*

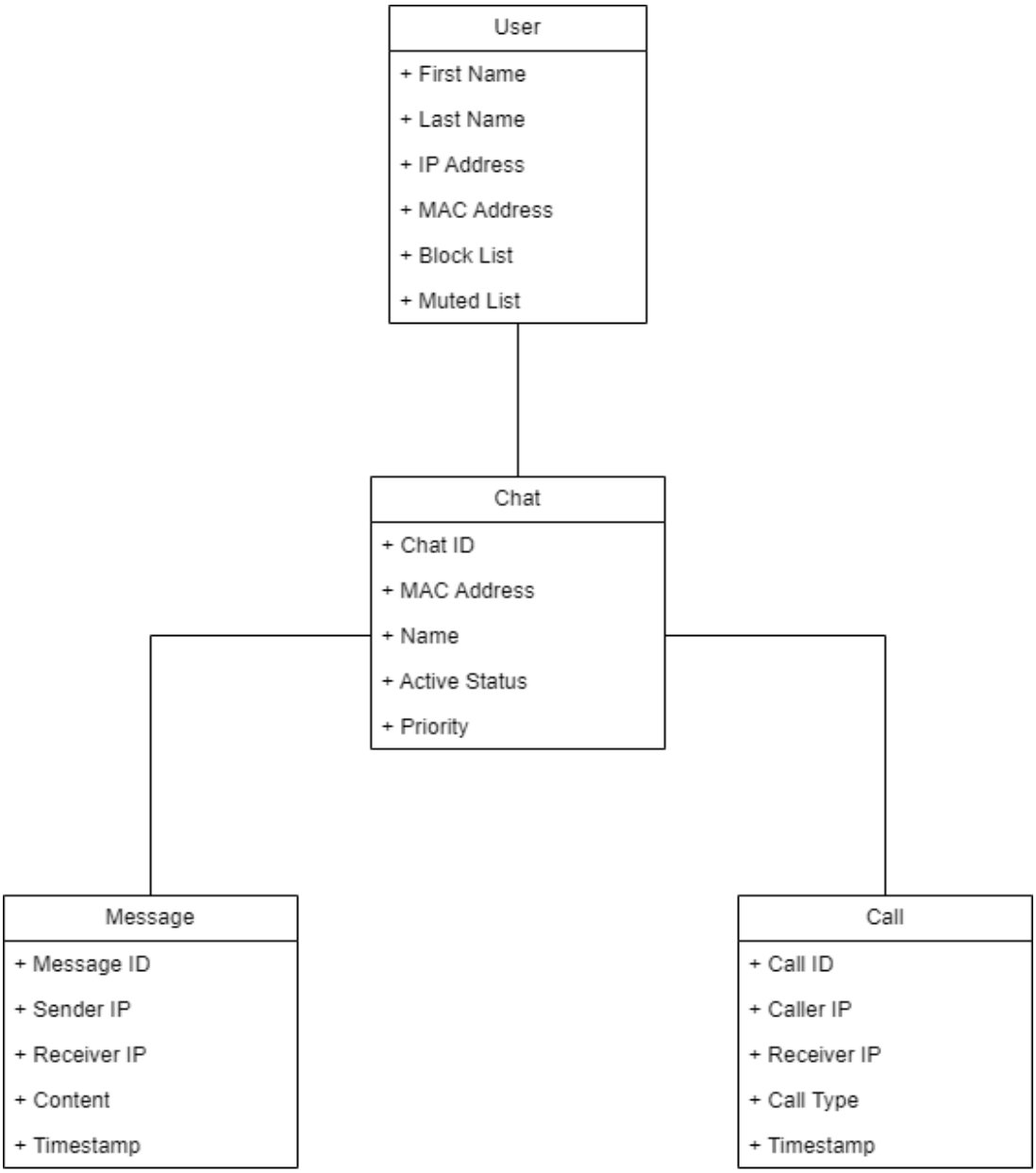

Figure 2.9: Domain Model

2.7 System Architecture

The system architecture of AndroCom is designed to provide a reliable and secure way for users to communicate without an internet connection. The system consists of a client app and a microcontroller. The client app is responsible for establishing a connection to the AD HOC network, sending and receiving messages, and making and receiving voice and video calls. The microcontroller is responsible for managing the AD HOC network and relaying messages between clients. The microcontroller is very portable and can be attached to transport vehicles and drones.

The client app and the server communicate with each other using TCP/IP sockets. The server also uses UDP sockets to broadcast messages to all clients on the AD HOC network.

The system architecture and system design are shown in *figure 10.10* and *figure 10.11.*

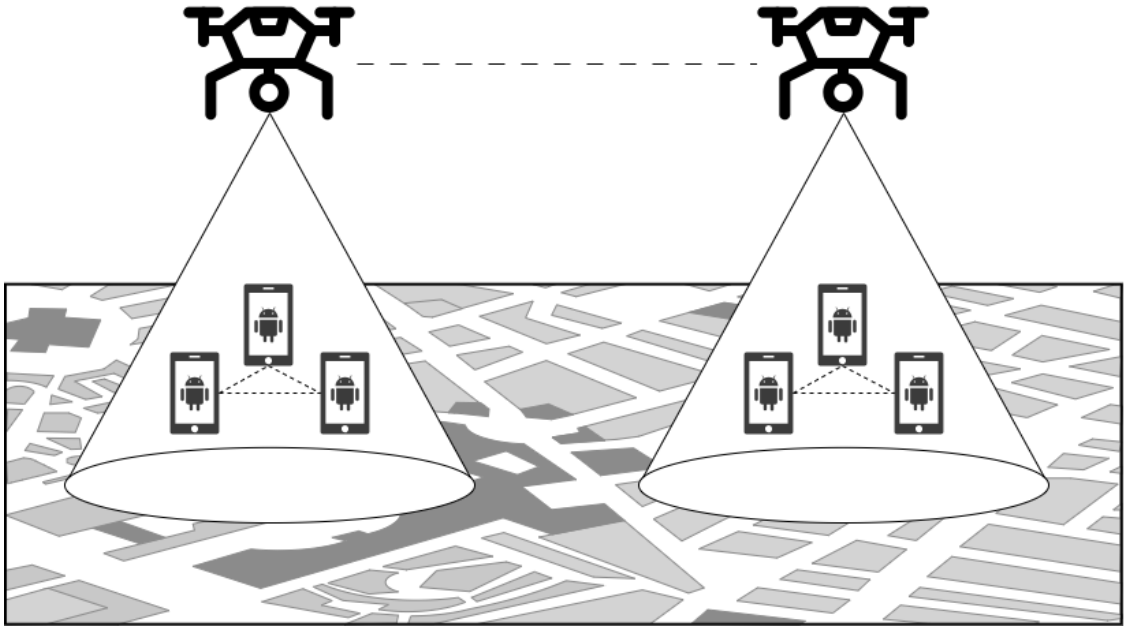

Figure 2.10: System Design

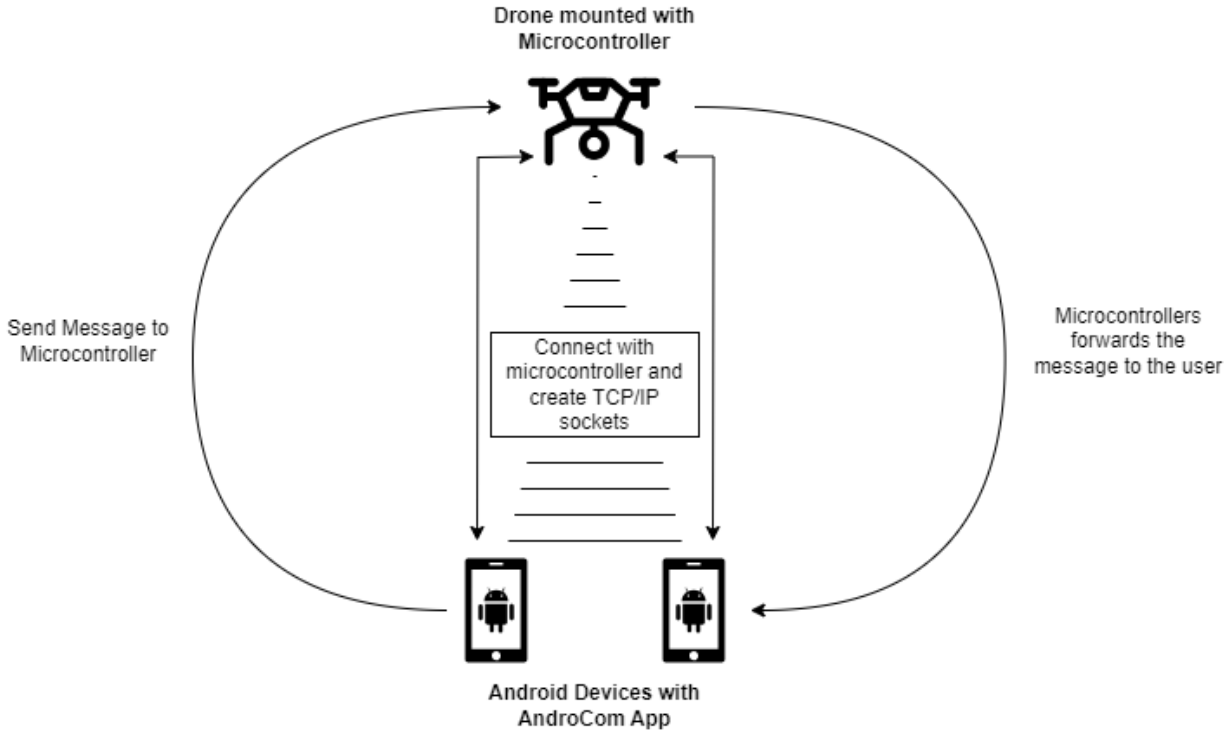

Figure 2.11: System Architecture

Chapter 3
System Design

The purpose of this chapter is to provide information that is complementary to the code. Without an adequate design that delivers required function as well as quality attributes, the project will fail. But communicating architecture to its stakeholders is as important a job as creating it in the first place.

This chapter covers the following specifications for the required software:

- Layer Definition
- Software Architecture
- Data Flow Diagram
- User Interface Design

3.1 Layer Definition

A layer definition is a description of the purpose, functionality, and interfaces of a layer in a layered system. In simple words, it is a definition of what a layer does and how it interacts with other layers. Layer definition for AndroCom is given in *Table 3.1.*

Table 3.1: Layer Definition

Layers	Description
Application Layer (AndroCom App)	This layer is responsible for providing user interfaces and other key functionalities.
Network Layer (Microcontroller)	This layer is responsible for communicating with the microcontroller over a network.

3.1.2 Application Layer

This layer is responsible for implementing the core functionality of the application, such as connecting sockets, sending and receiving messages, making & receiving voice and video calls, and interacting with the user interface.

3.1.2 Network Layer

This layer is responsible for communicating with the microcontroller over a network. The AndroCom App layer sends and receives messages to and from the network layer. The network layer then sends and receives messages to and from the microcontroller.

3.2 Software Architecture

A software architecture diagram is a visual representation of the structure of a software system, showing the components of the system and how they interact. In simpler words, it is a diagram that shows how a software system is built. The software architecture diagram is given below in *figure 3.1.*

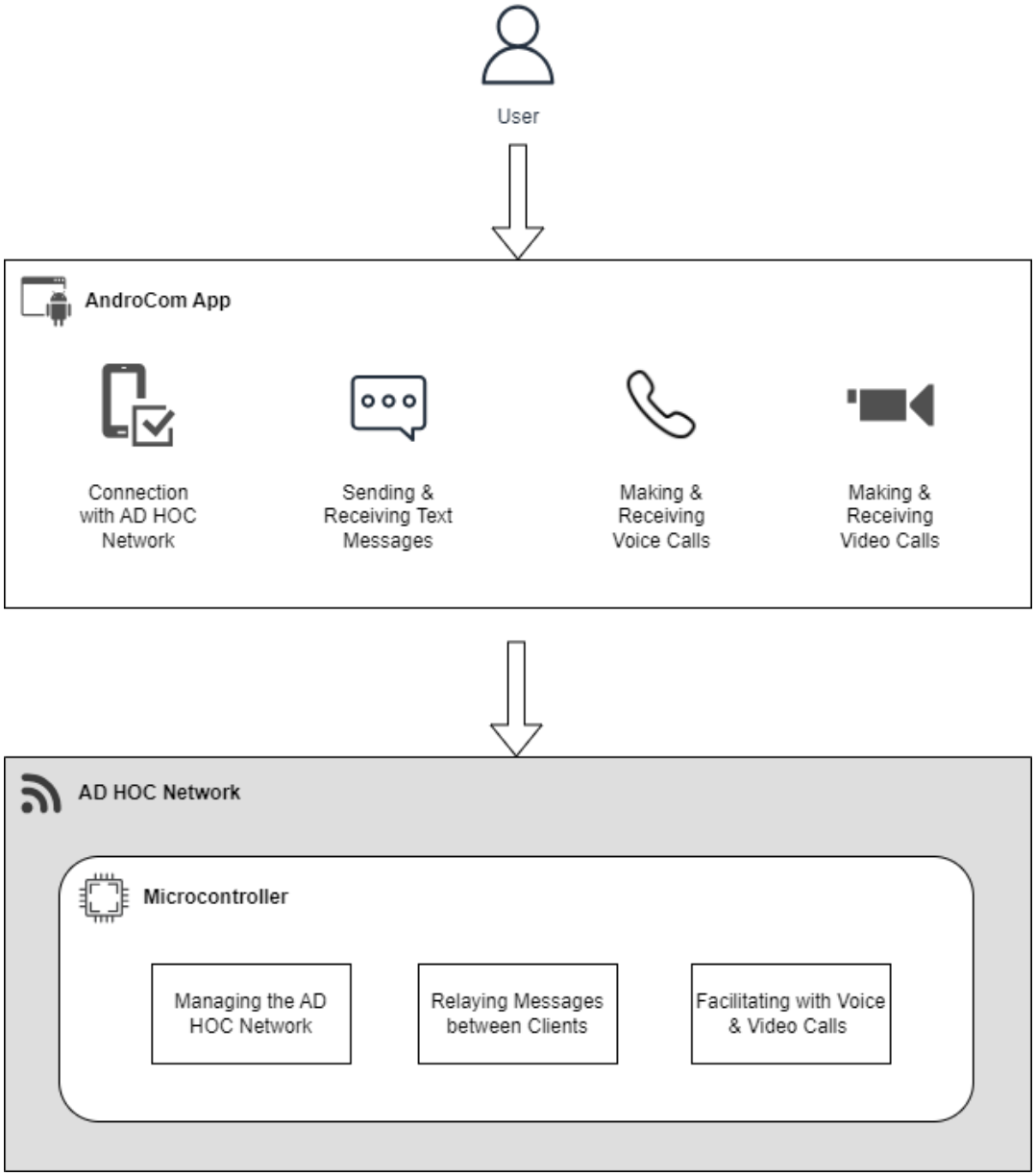

Figure 3.1: Software Architecture Diagram

3.3 Data Flow Diagram

A Data Flow Diagram is a graphical representation of the flow of data through a system, showing its inputs, outputs, and processing steps. The data flow diagram for AndroCom is given in *figure 3.2.*

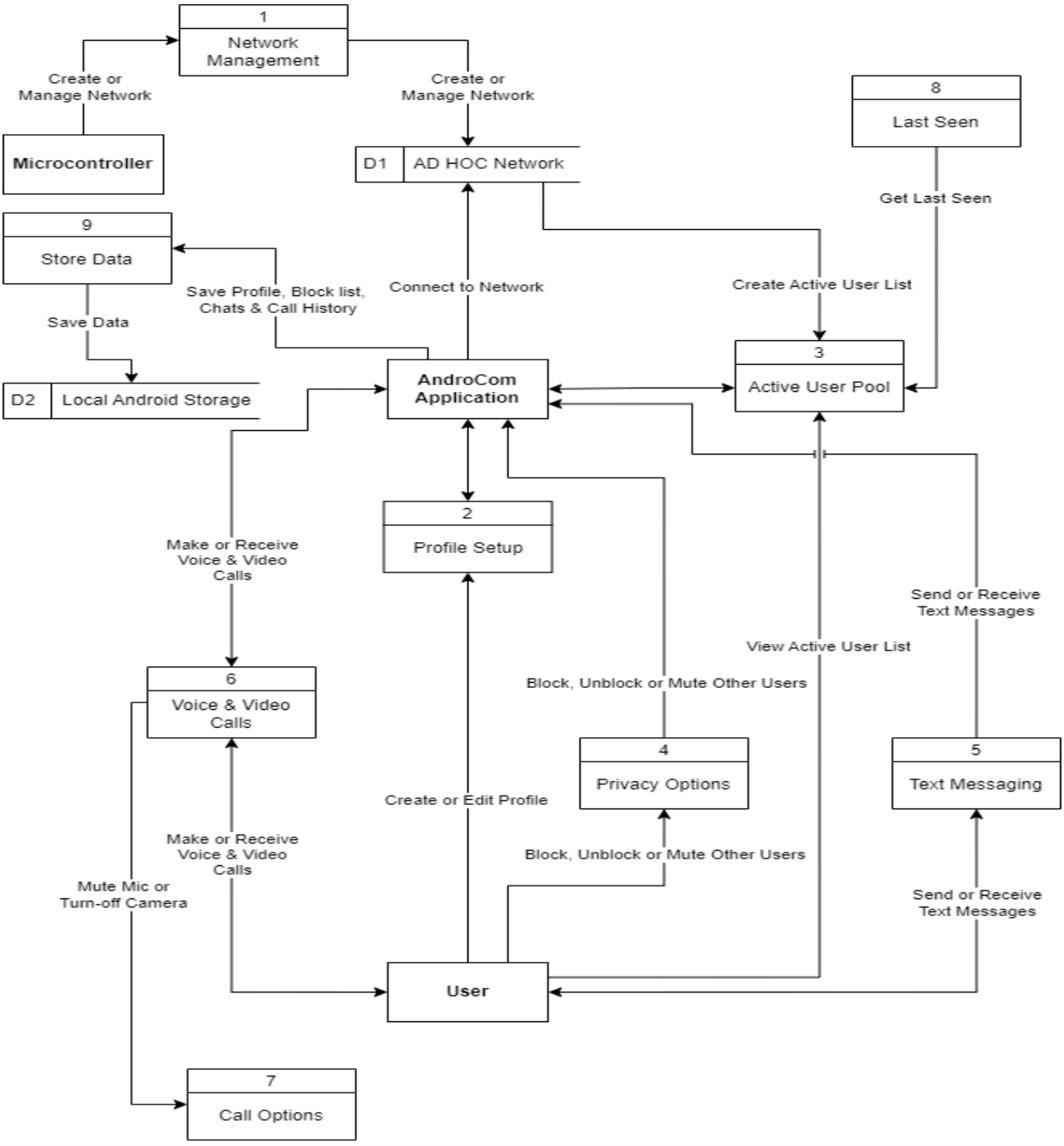

Figure 3.2: Data Flow Diagram

3.4 User Interface Design

User interface design is the process of creating interactive screens that are easy to use and understand. The UI design for AndroCom is given below.

Figure 3.3: Splash Screen

Figure 3.4: Screen upon initial launch

Description:

The AndroCom app's splash screen is the first thing users see when they open the app. It displays the app's logo in the center against a white background. This screen creates a strong initial impression and sets the stage for the user's interaction with the app.

Figure 3.5: Profile Setup

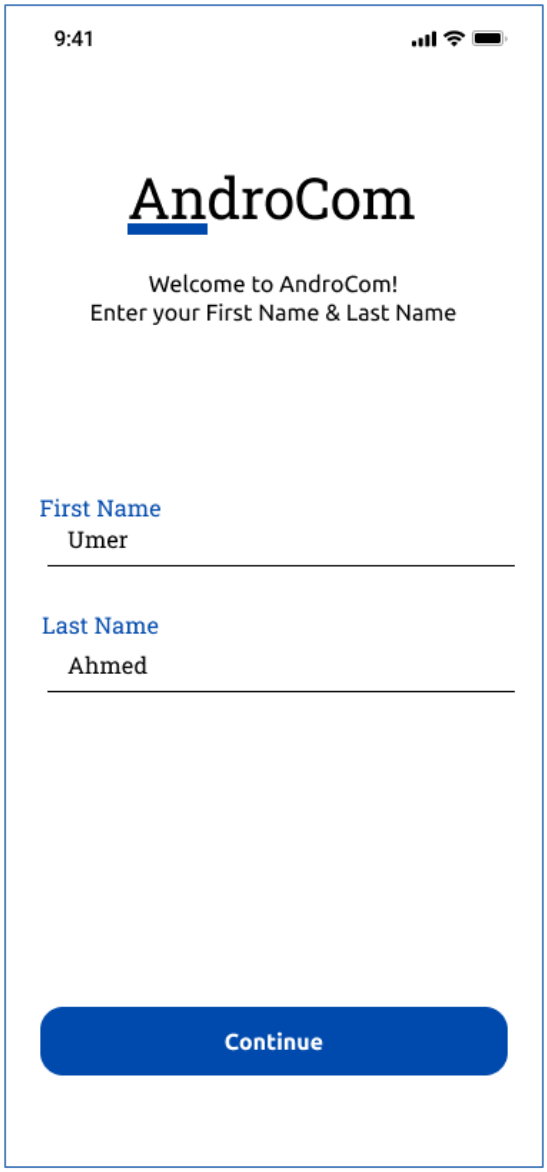

Figure 3.6: Profile Setup (filled)

Description:

The profile setup screen in AndroCom is where users enter their first name and last name. It's a clean and minimalist design, with input fields for both names. What's distinctive is the 'Next' button, which remains inactive until both fields are filled. This encourages users to provide their information, ensuring a complete profile setup.

Figure 3.7: Chats Screen

Description:

The chat screen in AndroCom is where users can seamlessly engage in multiple conversations. The user-friendly interface allows users to view and manage various chat threads.

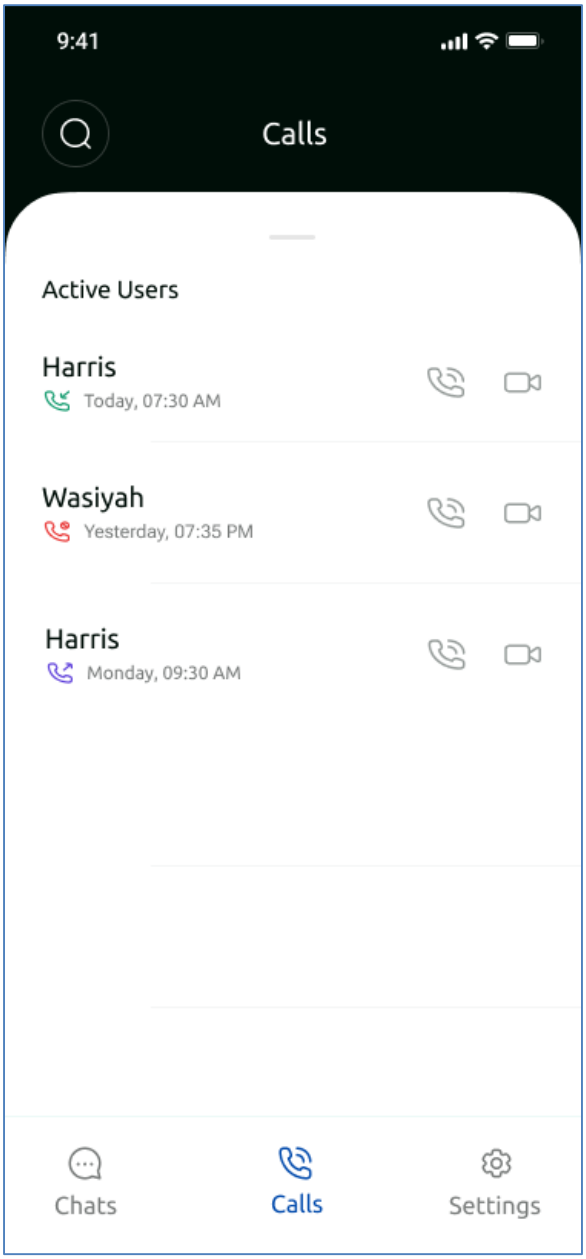

Figure 3.8: Call Log

Description:

The call log screen in AndroCom provides users with a comprehensive record of their recent calls and also displays active users with whom call can be made.

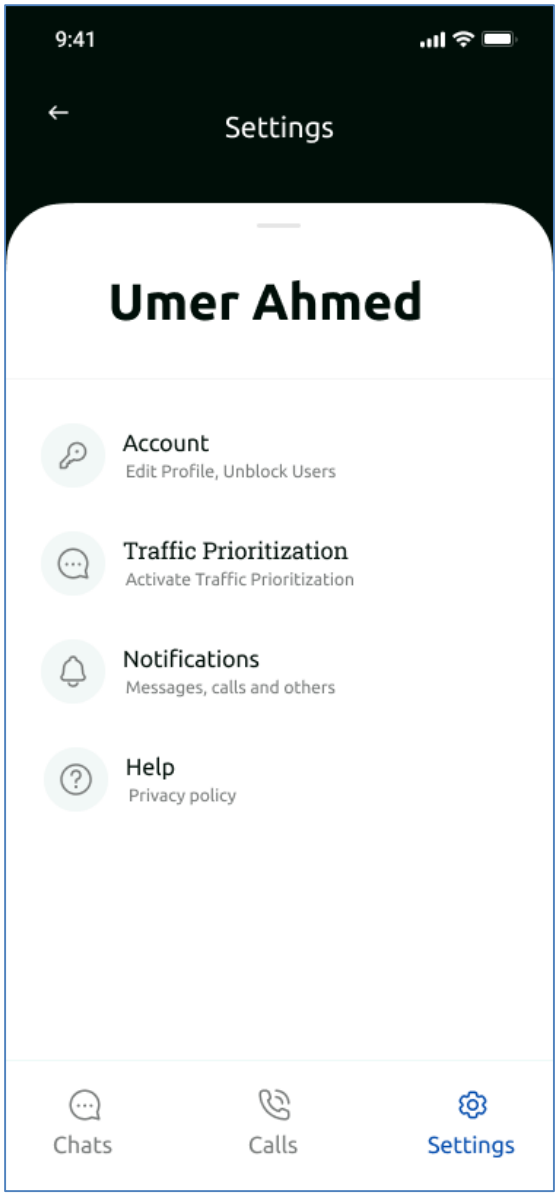

Figure 3.9: Settings Menu

Description:

The settings screen in AndroCom is where you can easily view and edit your display name, unblock users, activate network prioritization, adjust notification settings, and access the privacy policy and manual.

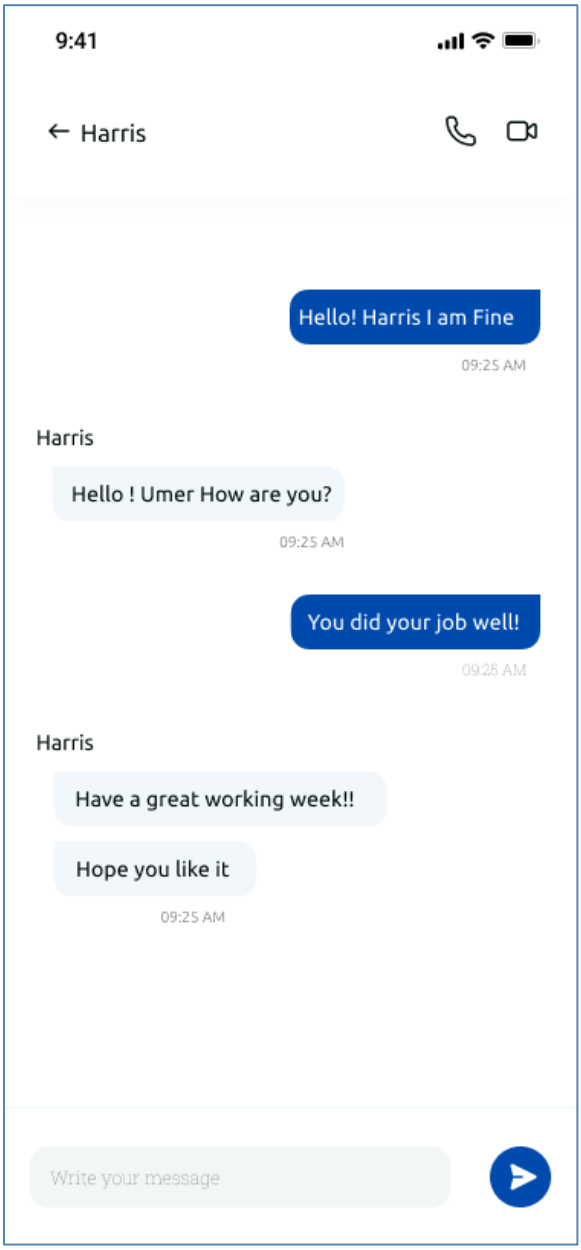

Figure 3.10: Message Screen

Description:

The messaging screen in AndroCom is where you can scroll through older messages and send new ones to active users. Also, you can make voice and video calls directly from this screen, ensuring that your communication needs are all in one place.

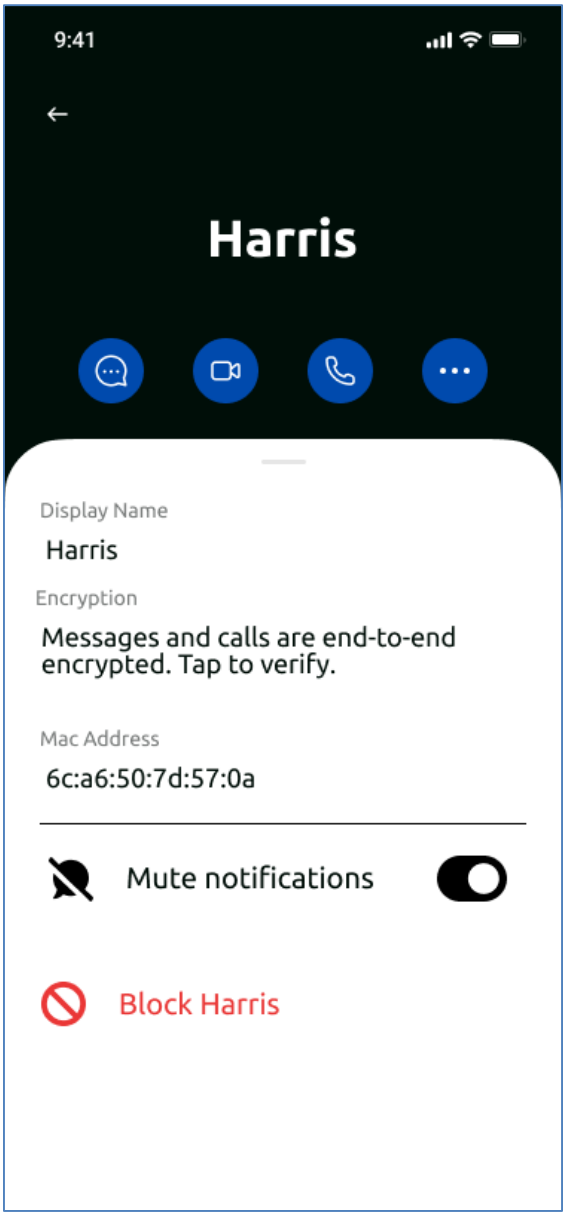

Figure 3.11: Other User's Profile View

Description:

When viewing another person's profile in AndroCom, you'll find essential information at a glance. This includes their name and MAC address, making it easy to identify and connect with them. Additionally, you have the option to mute or block the user.

Figure 3.12: Incoming Voice Call Screen

Figure 3.13: Ongoing Voice Call Screen

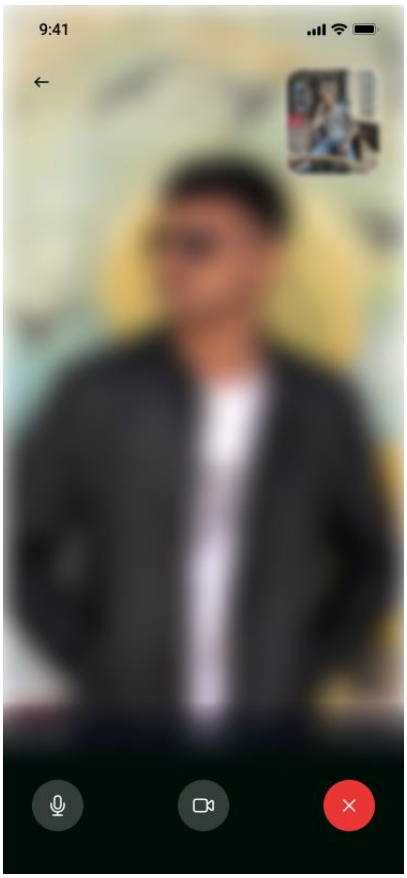

Figure 3.14: Ongoing Video Call Screen

Description:

In AndroCom, when you receive an incoming voice call, the caller's name is prominently displayed. By sliding, you can accept the call. During an ongoing voice call, you have the option to mute your microphone for privacy and convenience. And for video calls, you can mute the microphone or disable the camera.

Chapter 4
Software Development

This section provides an in-depth exploration of the software development phase for AndroCom. Here, we systematically outline the key elements that contribute to the creation of a robust and efficient communication application. Emphasizing the importance of precision in this phase to avoid future alterations, we delve into the specifics of our software development approach.

The following components are covered in this section:

- Coding Standards
- Development Environment
- Software Description

4.1 Coding Standards

The following coding standards were followed in development of this project,

- Python
    1. Indentation: Consistent four spaces.
    2. Declaration: Snake_case for variables (e.g., broadcast_ip).
    3. Naming Convention: Snake_case for variables and functions.
    4. Statement Standard: Each statement on a new line, comments for purpose explanations, and a clear function definition.
- Kotlin
    1. Indentation: Four spaces consistently.
    2. Declaration: CamelCase for class and package names (e.g., SendMessage).
    3. Naming Convention: CamelCase for class names, lowercase for package names.
    4. Statement Standard: Each statement on a new line, comments for code structure explanations.

4.2 Development Environment

The tools and technologies used for the development of AndroCom along with critical libraries and packages are given below. Also, the deployment process for development environment is provided as well.

4.2.1 Tools & Technologies

The following are the tools and technologies used in the development of AndroCom,

- **Bash (Microcontroller Scripting):** Bash was for creating AD HOC network in the microcontroller, facilitating communication between android application and microcontroller.
- **Python (Microcontroller Side Programming):** Utilized for programming the microcontroller, ensuring efficient execution of tasks on the microcontroller side such as sending IP addresses & MAC addresses of connected devices to android application.
- **Kotlin (Android Application):** Chosen as the primary language for Android application development, providing a robust framework for building user interfaces and implementing functionalities.
- **Thonny IDE (Python on Microcontroller):** Specifically used for Python development on the microcontroller, providing a user-friendly interface and tools for coding, testing, and debugging Python scripts on a Linux operating system.
- **Visual Studio Code (Testing and Logic Development):** Used for testing and logic development, providing a versatile and lightweight integrated development environment (IDE) for multiple coding tasks.
- **Android Studio (Android Application Development):** The major IDE for Android app development, offering comprehensive tools and features for designing, coding, testing, and debugging Android applications.

4.2.2 Deployment of Development Environment

The deployment process involved the installation and configuration of the respective development environments on the relevant platforms such as Windows and RaspberrianOS (Linux). Each environment was tailored to suit the specific requirements of the development phase.

4.2.3 Packages and Libraries

The following are the two major libraries used in the development of our project,

- **Socket (Python):** It was used for creating UDP packets on microcontroller and sending it over the android application through AD HOC network.
- **Java Socket (Kotlin):** This is the most critical library to our android application. It provided all the necessary functions needed to create the functionality of receiving UDP packets from microcontroller and sending/receiving messages to other devices using TCP/IP Sockets through AD HOC network.

4.3 Software Description

Snippet 1

```
!/bin/bash

# Check if NetworkManager is installed
if ! command -v nmcli &> /dev/null; then
    echo "NetworkManager is not installed. Please install it
first."
    exit 1
fi

# Set your desired network information
NETWORK_NAME="Ad-HocNetwork"
SSID="Ad-HocNetwork"
PASSWORD="12345678"

#Delete existing connection (if any)
sudo nmcli connection delete "$hotspot_ssid" 2>/dev/null

# Create a new connection with a /27 subnet
nmcli con add type wifi ifname '*' con-name $SSID autoconnect
yes ssid $SSID 802-11-wireless.mode ap 802-11-wireless.band bg
ipv4.method shared
nmcli con modify $SSID 802-11-wireless.security.key-mgmt wpa-psk
nmcli con modify $SSID 802-11-wireless-security.psk $PASSWORD

# Set the network address and subnet mask for a /27 subnet
nmcli con modify $SSID ipv4.addresses "192.168.1.1/27"
```

```
# Bring up the WiFi connection
nmcli con up $SSID

# Display the status of the WiFi connection
nmcli con show $SSID | grep GENERAL.STATE

echo "WiFi network '$SSID' with a /27 subnet is now running."

# Function to list connected devices
list_connected_devices() {
    echo "Connected Devices:"
    sudo arp -a | grep "$wifi_interface" | awk '{print $1, $2}'
}

# Trap to handle SIGINT (Ctrl+C) and list connected devices
before exiting
trap 'list_connected_devices; exit' SIGINT

# Continuously list connected devices
while true; do
    list_connected_devices
    sleep 10  # Adjust the sleep interval as needed
done
```

Description: This Bash script automates the setup of an AD HOC WiFi network using NetworkManager on a Linux system. It begins by checking for the presence of NetworkManager, prompting the user to install it if necessary. The script then defines network parameters like the network name, SSID, and password, attempting to delete any existing connection with the specified SSID. A new WiFi connection is created with specific settings, including AD HOC mode and a /27 subnet. Security configurations such as WPA-PSK are applied, and the network address and subnet mask are set. The script activates the WiFi connection, displays its status, and continuously lists connected devices using the ARP table.

This script streamlines the process of establishing and managing an AD HOC network, providing real-time information on connected devices. Adjustments to network parameters can be made to suit specific requirements.

Snippet 2

```
import subprocess
import json
import time
import socket


# Specify the broadcast IP address and port
broadcast_ip = '192.168.1.31'  # Replace with the broadcast
address of your local network
broadcast_port = 54321

# Create a socket object for broadcasting
broadcast_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
broadcast_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_BROADCAST, 1)

def get_connected_devices():
    try:
        # Run the arp command and capture its output
        result = subprocess.check_output(['arp', '-a'],
universal_newlines=True)

        # Split the output into lines
        lines = result.split('\n')

        # Extract IP addresses and corresponding MAC addresses
        devices = {}
        for line in lines:
            if 'ether' in line:
                parts = line.split()
                ip_address = parts[1]
                mac_address = parts[3]
                devices[mac_address] = ip_address.strip('()')

        return {"devices":devices}

    except subprocess.CalledProcessError:
        print("Error executing the arp command.")
        return None
```

```
# Broadcast the JSON string every 2 seconds (for example)
while True:
    connected_devices = get_connected_devices()

    if connected_devices:
        # Create a JSON object to send
        x = json.dumps(connected_devices, indent=4)
        broadcast_socket.sendto(x.encode(), (broadcast_ip,
broadcast_port))
        print(f"Broadcasted: {x}")
        time.sleep(2)
    else:
        print("Failed to retrieve connected devices.")


# Close the broadcasting socket (this will never be reached in
the example)
broadcast_socket.close()
```

Description: This Python script is programed to discover and broadcast information about connected devices within the AD HOC network. It utilizes the ARP (Address Resolution Protocol) command to retrieve a list of connected devices along with their IP and MAC addresses. The script then packages this information into a JSON object and broadcasts it at regular intervals using a UDP socket. The specified broadcast IP address and port facilitate the dissemination of device information across the network. The loop ensures continuous execution, periodically updating and broadcasting the list of connected devices.

Overall, this script provides a mechanism to dynamically share information about connected devices within the network, serving applications where real-time device presence tracking is essential. Adjustments to the broadcast IP, port, or update frequency can be made based on specific network requirements.

Snippet 3

```
package com.example.androcomtest

import java.net.DatagramPacket
import java.net.DatagramSocket

class activity_Contacts : AppCompatActivity() {
    val callsIcon = findViewById<ImageView>(R.id.callsIcon)
    val settingsIcon =
findViewById<ImageView>(R.id.settingsIcon)
    val callText=findViewById<TextView>(R.id.callText)
    val settingText=findViewById<TextView>(R.id.settingText)
    val chatText=findViewById<TextView>(R.id.chatText)
    val chatIcon = findViewById<ImageView>(R.id.chatIcon)
    private lateinit var binding : ActivityMainBinding
    private lateinit var userIPArray: ArrayList<UserIP>
    private lateinit var listView: ListView
    private lateinit var adapter : IPAdapter


    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        callsIcon.setOnClickListener {

            val intent = Intent(this,
calls_activity::class.java)
            startActivity(intent)
        }
        callText.setOnClickListener {
            val intent = Intent(this,
calls_activity::class.java)
            startActivity(intent)
        }
        settingText.setOnClickListener {
            val intent = Intent(this,
activity_app_settings::class.java)
            startActivity(intent)
        }
        settingsIcon.setOnClickListener {
            val intent = Intent(this,
activity_app_settings::class.java)
            startActivity(intent)
        }
```

```
        chatIcon.setOnClickListener {
            val intent = Intent(this,
activity_HomeScreen::class.java)
            startActivity(intent)
        }
        chatText.setOnClickListener {
            val intent = Intent(this,
activity_HomeScreen::class.java)
            startActivity(intent)
        }
        binding =ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        userIPArray= ArrayList()
        listView=findViewById(R.id.contactlist)
        adapter = IPAdapter(this,userIPArray)
        listView.adapter=adapter


        startIPReceiver()


    }
    private fun startIPReceiver() {
        CoroutineScope(Dispatchers.IO).launch {
            try {
                val socket = DatagramSocket(54321)

                while (true) {
                    try {
                        val packet =
DatagramPacket(ByteArray(1024), 1024)
                        socket.receive(packet)

                        val receivedData = String(packet.data,
0, packet.length)

                        try {
                            // Assuming JSON format:
                            val devices =
JSONObject(receivedData)
                            val newUserIPs = ArrayList<UserIP>()
                            val devicesJSONObject =
devices.getJSONObject("devices")
```

```
                            val deviceKeys =
devicesJSONObject.keys()

                            for (ipKey in deviceKeys) {
                                val ip =
devicesJSONObject.getString(ipKey)
                                newUserIPs.add(UserIP("username
","userIP $ip"))
                            }


                            runOnUiThread {
                                userIPArray.clear()
                                userIPArray.addAll(newUserIPs)
                                adapter.notifyDataSetChanged()
                            }
                        } catch (e: JSONException) {
                            handleInvalidDataError(e)
                        }
                    } catch (e: IOException) {
                        handleNetworkError(e)
                    }
                }
            } catch (e: SocketException) {
                handleSocketError(e)
            }
        }
    }

    private fun handleInvalidDataError(e: JSONException) {
        runOnUiThread {

            Toast.makeText(this, "Error parsing received data:
${e.message}", Toast.LENGTH_LONG).show()
        }
    }

    private fun handleNetworkError(e: IOException) {
        runOnUiThread {

            Toast.makeText(this, "Network error: ${e.message}",
Toast.LENGTH_LONG).show()
        }
    }
```

```
    private fun handleSocketError(e: SocketException) {
        runOnUiThread {

            Toast.makeText(this, "Socket error: ${e.message}",
Toast.LENGTH_LONG).show()
        }
    }

}
```

Description: This Kotlin code represents an Android activity named activity_Contacts within the "androcom" application. The main functionality revolves around receiving and processing data from a DatagramSocket which are coming from the microcontroller, presumably containing information about connected devices in the network. The received data, assumed to be in JSON format, is parsed to extract IP addresses and populate a user interface list with UserIP objects. The startIPReceiver function is implemented using Kotlin coroutines to continuously listen for incoming data on a DatagramSocket.

Overall, this code contributes to the creation of a dynamic contact list based on real-time data received from the microcontroller via the AD HOC network, enhancing the communication features of the Android application.

Snippet 5

```
package com.example.androcom

import java.io.BufferedReader
import java.io.InputStreamReader
import java.io.OutputStreamWriter
import java.net.Socket
import android.os.AsyncTask

class activity_chat : AppCompatActivity() {

    lateinit var recyclerView: RecyclerView
    lateinit var progressBar: ProgressBar
    lateinit var inputMessage: EditText
    lateinit var adapter: ChatAdapter
    private val handler = Handler(Looper.getMainLooper())

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_chat)



       val  textname =findViewById<TextView>(R.id.textname)
        val
backicon=findViewById<AppCompatImageView>(R.id.backicon)
        recyclerView = findViewById(R.id.chatRecyler)
        progressBar = findViewById(R.id.progress)
        inputMessage = findViewById(R.id.inputmessage)
        val sendButton =
findViewById<FrameLayout>(R.id.layoutsend)
        adapter = ChatAdapter()
        recyclerView.adapter=adapter
        sendButton.setOnClickListener {
            val message = inputMessage.text.toString()
            SendTask().execute(message)
            inputMessage.setText("")
        }
        val username = intent.getStringExtra("username")
        if (username != null) {
            textname.text = username

        }
        backicon.setOnClickListener{
            val intent = Intent(this,
activity_HomeScreen::class.java)
            startActivity(intent)
        }

    }

    override fun onResume() {
        super.onResume()
        ReceiveTask().execute()
    }

    private inner class ReceiveTask : AsyncTask<Void, String,
Unit>() {
        override fun doInBackground(vararg params: Void?) {
            Recieve_Message()
        }
```

```
        override fun onProgressUpdate(vararg values: String?) {
            // UI updates on receiving messages
            val receivedMessage = Message(values[0] ?: "",
false)
            addMessageToRecyclerView(receivedMessage)
        }
    }
    private fun addMessageToRecyclerView(message: Message) {
        // Access your adapter for the RecyclerView
        val adapter = recyclerView.adapter as ChatAdapter  //
Assuming you have a ChatAdapter
        adapter.addMessage(message)
        recyclerView.smoothScrollToPosition(adapter.itemCount -
1)  // Scroll to bottom
    }

    private inner class SendTask : AsyncTask<String, Void,
String>() {
        private lateinit var sentMessage: String

        override fun doInBackground(vararg params: String):
String? {
            val message = params[0]
            sentMessage = message // Store the message in the
class-level variable
            Send_Message(IP_Address, message)
            return null
        }

        override fun onPostExecute(result: String?) {
            // UI updates after sending message
            val message = sentMessage // Access the stored
message
            val sentMessage = Message(message, true)
            addMessageToRecyclerView(sentMessage)
        }
    }
    var socket: Socket? = null
    fun Send_Message(IP_Address: String, Message: String) {
        try {
            if (socket == null) {
                socket = Socket(IP_Address, 49153)
            }
```

```
            // Use a temporary variable for smart cast
            val currentSocket = socket
            if (currentSocket != null) {
                val writer =
OutputStreamWriter(currentSocket.getOutputStream())
                writer.write(Message)
                writer.flush()
                println("Message Sent")
            } else {
                println("Socket is null")
            }
        } catch (e: IOException) {
            // Handle error
            println("Error sending message: $e")
            socket?.close() // Close if an error occurs
            socket = null
        }
    }



    fun Recieve_Message() {
        try {
            // Create server socket
            val serverSocket = ServerSocket(49153)

            while (true) { // Continuous listening loop
                // Accept client connection
                val clientSocket = serverSocket.accept()

                // Read message
                val reader =
BufferedReader(InputStreamReader(clientSocket.getInputStream()))
                val message = reader.readLine()
                println("Received Message: $message")

                // Handle message (e.g., display in UI)
                handler.post {
                    addMessageToRecyclerView(Message(message,
false))
                }

                // Close client socket (optional)
                clientSocket.close()
```

```
            }
        } catch (e: IOException) {
            // Handle error
            println("Error receiving message: $e")
        }
    }

}
```

Description: This Kotlin code defines an Android activity named activity_chat within the "androcom" application. The activity facilitates real-time chat functionality. The layout includes UI elements such as a RecyclerView for displaying chat messages, a ProgressBar, and an EditText for inputting messages. Additionally, there is a TextView to show the username of the chat, a back icon to navigate to the home screen, and a send button to transmit messages. The RecyclerView is linked to a ChatAdapter for managing and displaying chat messages.

Upon creating the activity, it initializes various UI components and sets up click listeners. It launches an AsyncTask called ReceiveTask in the onResume method, responsible for continuously listening for incoming messages in the background. Received messages are processed in the doInBackground method and updated on the UI using onProgressUpdate.

The SendTask AsyncTask is utilized to handle the process of sending messages. It communicates with the server socket specified by the IP address and port. The Send_Message function establishes a socket connection, sends the message, and handles potential errors.

The Recieve_Message function represents the server-side logic for message reception. It continuously listens for client connections, reads incoming messages, and updates the UI with the received message using the addMessageToRecyclerView function.

The overall structure of the code demonstrates a basic chat application with functionalities for sending and receiving messages, and it incorporates asynchronous tasks to handle networking operations in the background. The use of AsyncTask ensures that networking tasks do not block the main UI thread, providing a smoother user experience. Additionally, error handling mechanisms are implemented to manage potential issues during socket operations.

Chapter 5
Software Testing

This chapter provides a description of the adopted testing procedure, including the selected testing methodology, unit tests, and the test results of the developed software.

5.1 Testing Methodology

We employed black box testing as our chosen testing methodology, leveraging its efficiency and several inherent advantages. Black box testing is a software testing approach that assesses the functionality of an application without delving into its internal structures or workings. This method is versatile and applicable to various levels of software testing, including unit, integration, system, and acceptance testing. Specifically, black box unit testing played a crucial role in our project. Unit testing, a subset of black box testing, evaluates individual units of source code, assessing their fitness for use based on defined criteria.

5.2 Testing Environment

We are conducting manual testing by writing test cases for every module. We provide inputs to these test cases and then verify the results of each test case. During the testing phase of AndroCom, a variety of testing environments have been utilized to ensure the thorough evaluation of the application's functionalities.

Emulated environments have provided insights into the application's compatibility across various virtual devices, enabling testing under different screen sizes, resolutions, and Android versions. Real Android devices have been employed to validate the application's real-world performance, uncovering device-specific issues that may not be evident in emulated environments.

The microcontroller setup, including hardware like Raspberry Pi and Python scripts, has been essential for testing the communication and integration between the Android

application and the microcontroller. Ad hoc network and network interruption testing environments have been created to assess the application's functionality in scenarios with limited or no internet connectivity, as well as its resilience to network disruptions. This comprehensive testing approach aims to guarantee that AndroCom performs reliably across diverse conditions and scenarios, meeting the expectations of end-users.

5.3 Test Cases

Test Case 1: Verify that the microcontroller successfully creates an AD HOC network

Table 5.1: Test Case 1

Date: 06 December 2023	
System: AndroCom	
Objective: Make sure that the created network has the correct SSID, password and network name. Also, the created network should be visible as a Wi-Fi network and devices must be able to connect with it.	*Test ID:* 1
Version: 1	*Test Type:* Unit testing
Input: SSID, Network Name and Password to Bash script in microcontroller.	
Expected Result: Created AD HOC network has correct SSID, Network Name and Password. Also, devices are able to connect with the network.	
Actual Result: Passed	

Description: In this test case, a network was created via the microcontroller using a bash script. The script established a network using an SSID, Network Name, and Password provided within the script. Once the script was executed, the network became visible on all devices with Wi-Fi capability. All devices were able to successfully connect to the network.

Test Case 2: Ensure that the microcontroller accurately broadcasts the IP and MAC addresses of connected devices

Table 5.2: Test Case 2

Date: 08 December 2023	
System: AndroCom	
Objective: Make sure that the microcontroller successfully broadcasts the IP and MAC addresses of connected devices.	*Test ID:* 2
Version: 1	*Test Type:* Unit testing
Input: Connect device with network.	
Expected Result: The UDP packet containing IP and MAC addresses was successfully broadcasted by microcontroller to all connected devices. The packet contained correct IP and MAC addresses.	
Actual Result: Failed	

Description: In this test case, IP and MAC addresses of all connected devices were broadcasted to all android devices. The message was successfully broadcasted by microcontroller and received by the android app. The IP address received was accurate. However, MAC address received did not match the device MAC address.

Test Case 3: Ensure IP and MAC address logs are updated in real-time

Table 5.3: Test Case 3

Date: 08 December 2023	
System: AndroCom	
Objective: Ensure IP and MAC address logs are updated in real-time	*Test ID:* 3
Version: 1	*Test Type:* Unit testing
Input: ARP Table	
Expected Result: The log contains the IP address and MAC address of connected devices. The log is updated in real-time in case a device joins or disconnects with the network.	
Actual Result: Passed	

Description: In this test case, the python script for microcontroller maintained a log of connected devices, with their IP and MAC addresses. The log was real-time and was updated when a device connected or disconnected with the network.

Test Case 4: Testing the app setup process by entering a full user name

Table 5.4: Test Case 4

Date: 01 December 2023	
System: AndroCom	
Objective: Test the app setup process by entering a full user name. Ensure that the application accepts and stores the user name correctly.	*Test ID:* 4
Version: 1	*Test Type:* Unit testing
Input: First Name and Last Name	
Expected Result: Once, the user completes the profile setup, his name is stored in the android app and can be viewed in setting.	
Actual Result: Passed	

Description: In this test case, when the user opens the android application for the first time, he has to setup his profile. This is a onetime process. Once, the enters his first and last name, the continue button appears and user can finish setting up. Null values were not accepted. Entered name was successfully saved and was visible in settings.

Test Case 5: Verify received IP and MAC addresses on android application

Table 5.5: Test Case 5

Date: 10 December 2023	
System: AndroCom	
Objective: Verify that the application correctly displays a list of active devices connected to the microcontroller. This includes checking if the list is updated in real-time as devices connect or disconnect.	*Test ID:* 5
Version: 1	*Test Type:* Unit testing
Input: UDP Packet sent by microcontroller	
Expected Result: The android application successfully receives the UDP packet sent by the microcontroller. The app displays IP and MAC addresses in UI.	
Actual Result: Passed	

Description: In this test case, when the device is connected with the AD HOC network, the user opens the app and checks the connected devices. All the IP and MAC addresses were accurate.

Test Case 6: Testing that the message is successfully sent by the android app

Table 5.6: Test Case 6

Date: 28 December 2023	
System: AndroCom	
Objective: Test the message sending functionality by composing and sending messages from one device to another. Ensure that messages are delivered promptly and accurately.	*Test ID:* 6
Version: 1	*Test Type:* Unit testing
Input: Message	
Expected Result: The message is successfully sent and is visible in user interface.	
Actual Result: Passed	

Description: In this test case, user selects an IP address and sends a text message. The message was successfully sent and was visible on UI.

Test Case 7: Testing that the message is successfully recieved by the android app

Table 5.7: Test Case 7

Date: 28 December 2023	
System: AndroCom	
Objective: Confirm that the application receives and displays incoming messages correctly. Check for real-time updates and accuracy in displaying the sender's information.	*Test ID:* 7
Version: 1	*Test Type:* Unit testing
Input: None	
Expected Result: The message is successfully received and displayed on the UI.	
Actual Result: Passed	

Description: In this test case, when the message was sent, it was successfully received by the android device and was displayed on the UI.

Chapter 6
Software Deployment

In this chapter, the entire installation process for the AndroCom app and microcontroller is provided. A detailed, step-by-step process is outlined on how the user can install the app on their Android device and set up their microcontroller for AndroCom. The user is required to have two things: an Android device and a Raspberry Pi, preferably newer than Raspberry Pi 4B. If a Raspberry Pi is unavailable, the user can emulate scripts on a Raspbian OS (Linux) supported device.

6.1 Installation / Deployment Process Description

6.1.1 RaspberryPi Setup

To run the setup for Raspberry Pi, navigate to the terminal and use the command ***"sudo ./setup_hotspot.sh"*** after locating the ***"setup_adhoc.sh"*** file.

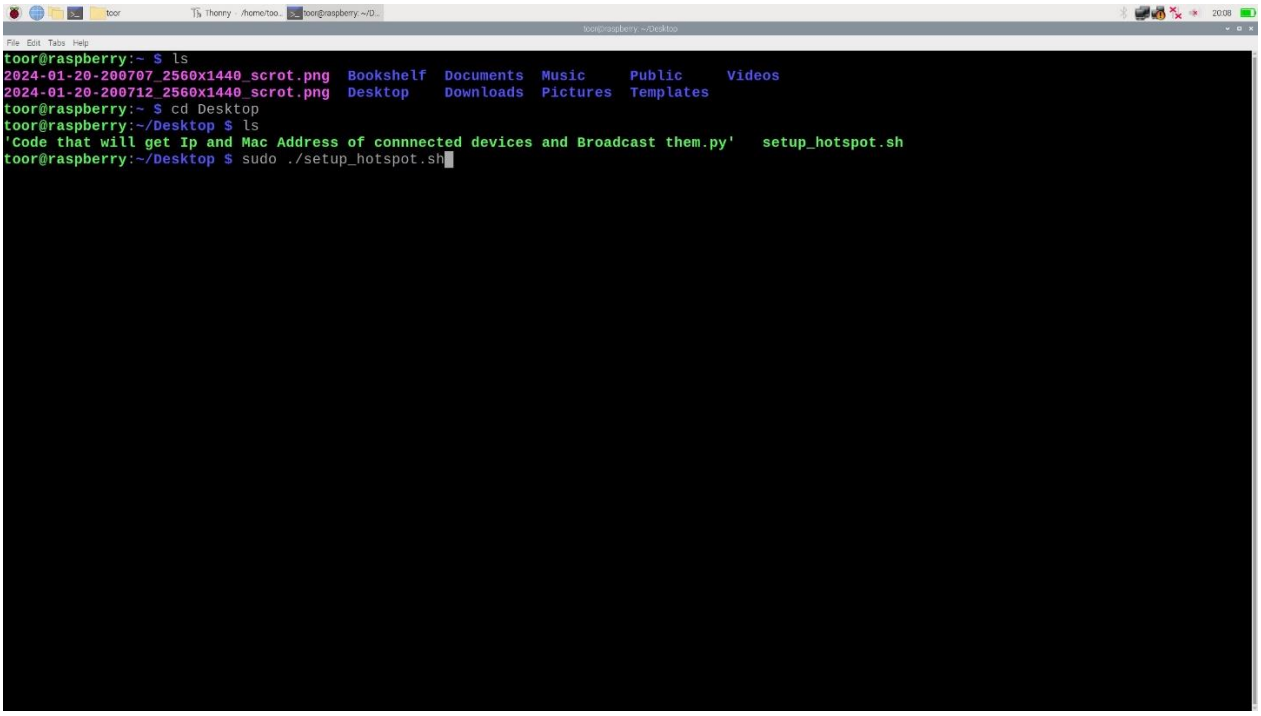

Figure 6.1: Terminal in RaspbarianOS

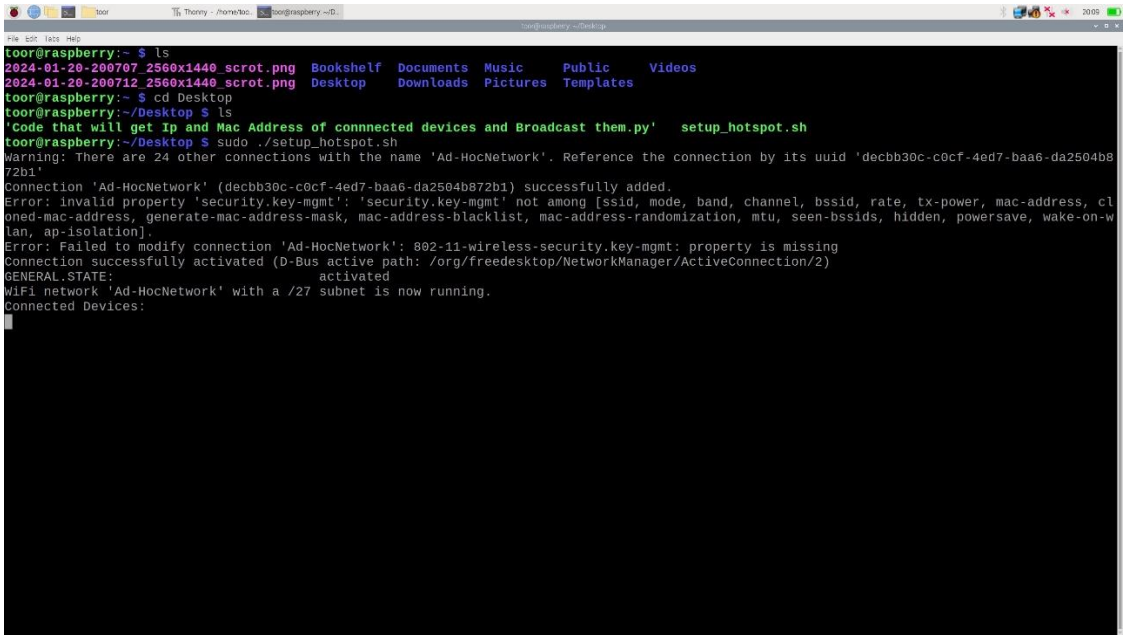

Figure 6.2: Terminal in RaspbarianOS after successfully establishing AD HOC network

Now, run the Python script that logs IP and MAC addresses and broadcasts them. Open the script in Thonny IDE and execute it.

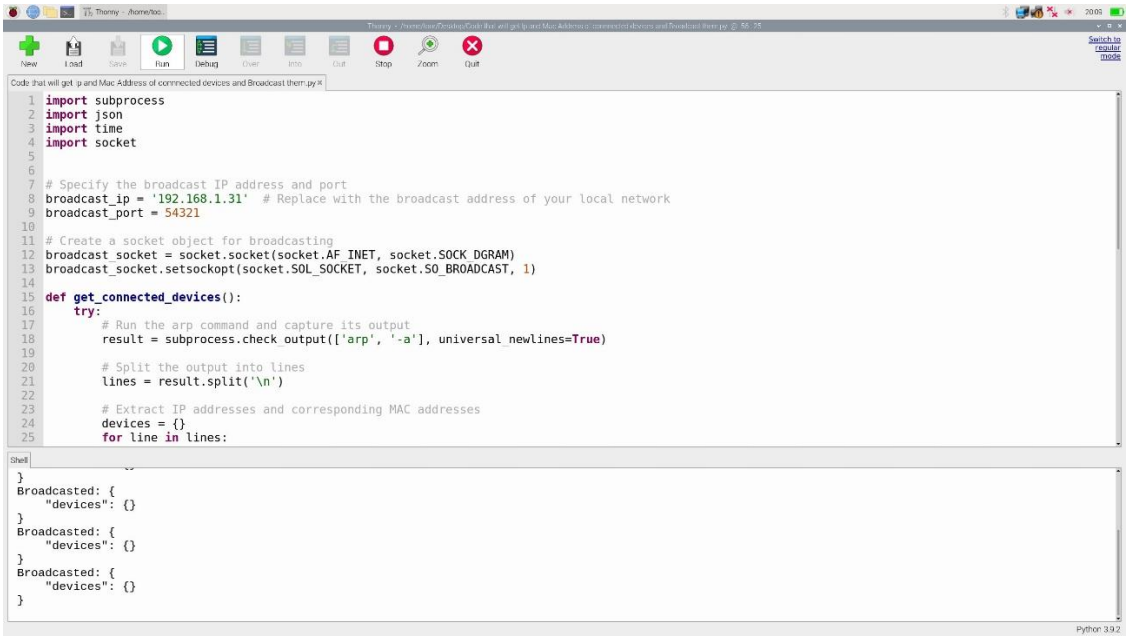

Figure 6.3: Thonny IDE in RaspbarianOS

6.1.2 Android Application

On your Android device, enable developer options. Once enabled, activate wireless debugging and connect it with Android Studio.

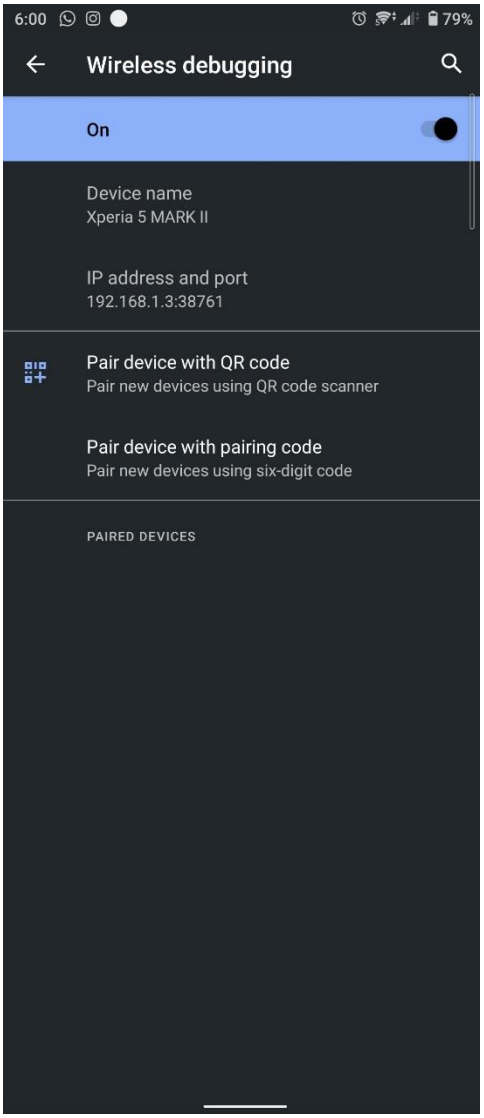

Figure 6.4: Wireless Debugging in Android Settings

Open the **'AndroCom'** project folder in Android Studio. Once opened, connect your Android device via the device manager. Once connected, click on **"Build Project"**.

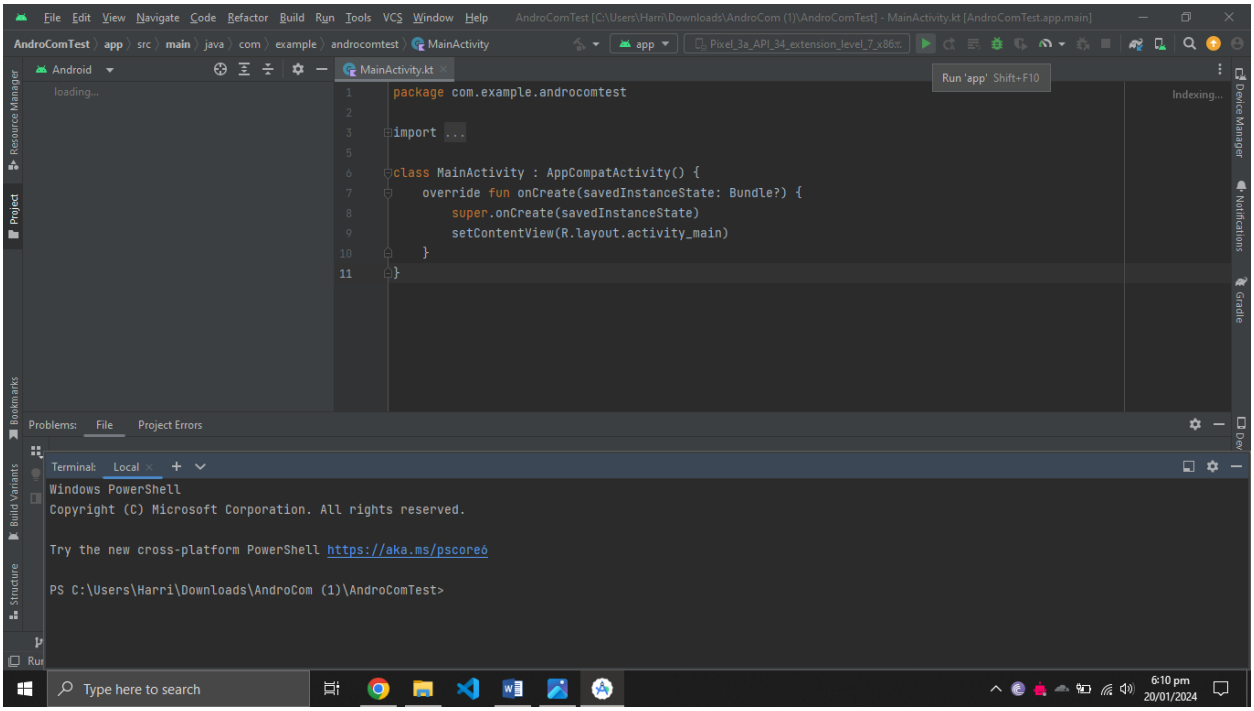

Figure 6.5: Android Studio

Once the project build is complete, the app will appear on the Android device.

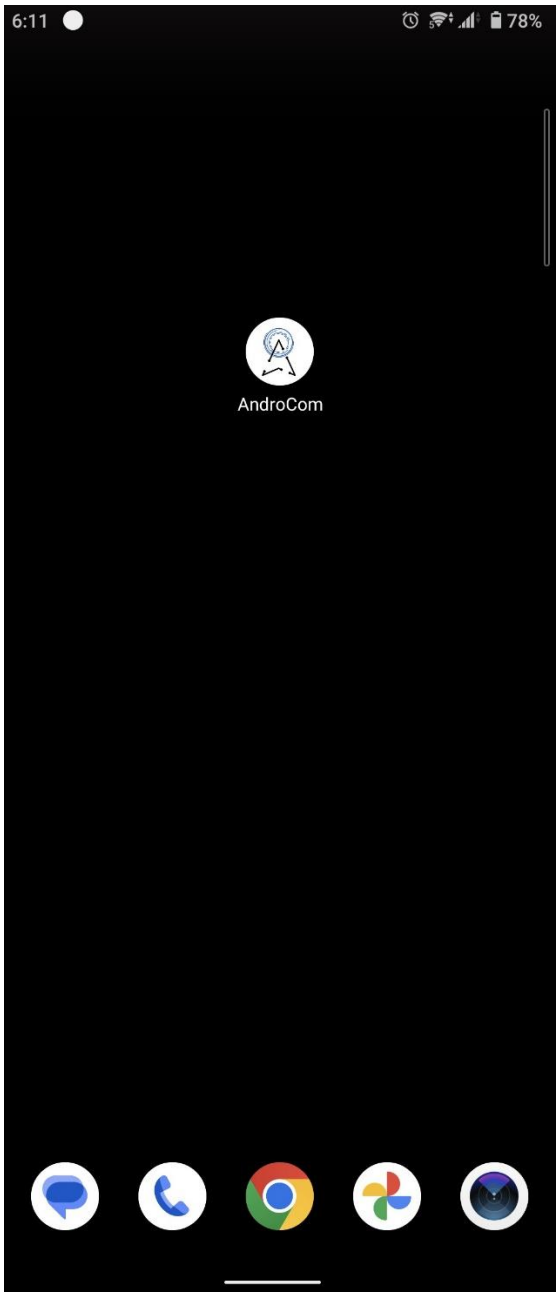

Figure 6.6: AndroCom App in Android Device's Home Screen

References

Book

Kurose, J. F., & Ross, K. W. (2019). *Computer Networking: A Top-Down Approach* (8th edition). Pearson.

Griffiths, D., & Griffiths, D. (2017). *Head First Android Development*. O'Reilly Media.

Webpage

Google. (Updated regularly). Android Developers - Documentation. https://developer.android.com/. Accessed on January 18, 2024.

Python Software Foundation. (Updated regularly). Python Documentation. https://docs.python.org/3/. Accessed on January 1, 2024.

JetBrains. (Updated regularly). Kotlin Documentation. https://kotlinlang.org/docs/home.html. Accessed on January 04, 2024.

Oracle Corporation. (Updated regularly). Java SE Documentation. https://docs.oracle.com/en/java/javase/index.html. Accessed on January 04, 2024.