AndroCom – Infrastructure less Communication

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

Spring - 2024

Supervised By
Mr. Bilal Ahmed

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

Submission Form for Final-Year

PROJECT REPORT

Version	V 3	No. of Members	3

Title	AndroCom – Infrastructure less Communication

Supervisor Name	Mr. Bilal Ahmed

Member Name	Reg. No.	Email Address
Umer Ahmed	BCS203182	Umerahmed1000@gmail.com
Muhammad Harris	BCS203193	Harris20014@gmail.com
Wasia	BCS203233	Wasiaibrar9892@gmail.com

Member's Signatures

_____

_____

_____

Supervisor's Signature

APPROVAL CERTIFICATE

This project, entitled as “AndroCom – Infrastructure less Communication” 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

I/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.

Member's Signatures

ACKNOWLEDGEMENTS

We are deeply grateful to Allah for helping us to successfully complete this project. Furthermore, we extend our heartfelt appreciation to our parents for their unwavering support and valuable guidance, which were crucial in various phases of the project. We would also like to express our sincere gratitude to our supervisor, Mr. Bilal Ahmed, for his invaluable guidance and support throughout this project. His insightful suggestions and instructions were pivotal to our success.

Executive Summary

In today's world, communication is heavily dependent on a functioning internet. If the internet were to go down, our ability to connect with each other would be severely hampered.

AndroCom offers a solution to this problem. It's an Android application designed to allow users to send encrypted text messages, make voice calls, and even video calls, all without relying on any existing network infrastructure. AndroCom achieves this by creating a localized ad hoc network using a Raspberry Pi 3B+. The portability and low power consumption of the Raspberry Pi make it ideal for deploying AndroCom on-the-go whenever communication needs arise.

Table of Contents

List of Figures

LIST OF TABLES

Chapter 1

Introduction

1.1. Project Introduction

Our reliance on the internet for communication creates a major vulnerability: outages can cripple our ability to connect. AndroCom, an Android app, offers a solution. AndroCom allows users to send encrypted text messages, make voice calls, and even video calls – all without relying on an internet connection. This unique feature makes it ideal for situations like remote locations with no internet access or as a reliable backup during emergencies.

The secret behind AndroCom's functionality lies in its ability to create an ad hoc network. An ad hoc network is a temporary, decentralized network set up on the fly without needing any existing infrastructure like routers or access points. It utilizes a Raspberry Pi, a small and portable computer, to establish an "ad hoc" network. This network allows devices running the AndroCom app to communicate directly with each other.

AndroCom goes a step further by ensuring the security of your communications. Using AES (Advanced Encryption Standard), it protects your messages, calls, and videos from unauthorized access. This makes it a valuable tool for situations where privacy is paramount. The combination of offline functionality, portability, and secure communication makes AndroCom a versatile and important tool for various situations.

1.2. Existing Examples / Solutions

Even though there isn't a direct equivalent to AndroCom, several technologies do exist that provide the ability to communicate without internet access or existing network infrastructure. First, we have satellite communication systems. With a long history, they offer reliable connections in remote areas. However, their cost, limited bandwidth, and dependence on specialized equipment make them less than ideal for everyone. Additionally, bad weather can disrupt signal quality.

Mesh networks, a more recent development, provide an alternative to traditional Wi-Fi. They distribute the internet connection through interconnected devices, extending coverage. While offering redundancy and wider reach, setting up a mesh network can be more complex. Furthermore, their overall range may still be limited compared to satellite communication.

Finally, walkie-talkies and short-range radios represent some of the oldest forms of off-grid communication. They're simple, reliable for voice communication within a short range, and readily available. However, they lack the ability to transmit data like text messages or video calls, and physical barriers or distance can disrupt communication.

Existing solutions clearly have limitations. AndroCom steps in to bridge these gaps. By leveraging affordable hardware like Raspberry Pi, it offers a more cost-effective approach compared to satellite communication. Setting up an ad hoc network with AndroCom is simpler than configuring mesh networks or requiring specialized satellite equipment. Additionally, AndroCom goes beyond voice communication, enabling text messaging, voice calls, and even video calls, providing a richer communication experience. Finally, the compact size of the Raspberry Pi allows for easy deployment on-the-go, making AndroCom more versatile than fixed solutions like satellite communication. Through these improvements, AndroCom aims to provide a more accessible, user-friendly, and feature-rich solution for communication during internet outages.

Table 1.1: Existing Solutions Features Comparison

Sr. No.	Feature	Satellite Communication System	Mesh Network	Walkie-Talkie	AndroCom Ad Hoc Network (Proposed System)
1	Cost Effective			✓	✓
2	Long Range	✓	✓		
3	Portable			✓	✓
4	Scalable	✓	✓		✓
5	Security	✓			✓
6	Quality				✓
7	Text Messages	✓	✓		✓
8	Voice Call	✓		✓	✓
9	Video Call				✓

1.3. Business Scope

AndroCom presents a significant business opportunity by offering a unique and cost-effective solution for communication during internet outages. Beyond its initial target audience, AndroCom has the potential to serve various commercial markets. With minimal additional resources, such as user interface refinement and marketing materials, AndroCom can be deployed as a successful commercial product.

1.4. Useful Tools & Technologies

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

1.4.1 Programming Languages:

- **Bash:** Bash is a type of shell scripting language for Unix-based operating systems. It's used to execute commands from the command line and automate tasks.
- **Kotlin:** Kotlin is a general-purpose programming language initially developed by JetBrains for the Android platform. It's known for being concise, interoperable with Java, and supporting functional programming features.
- **Java:** Java is a general-purpose, class-based, object-oriented programming language that is widely used for developing applications of various kinds. It's known for its platform independence and code reusability.
- **Python:** Python is a general-purpose, interpreted, high-level programming language that's known for its readability and ease of use. It's widely used for web development, data science, scripting, and more.

1.4.2 Markup Language:

- **XML:** XML is a markup language designed for encoding documents in a machine-readable format. It's used for data interchange between applications and for configuring settings in software programs.

1.4.3 Database:

- **SQLite:** SQLite is a relational database management system (RDBMS) that is embedded directly into the application. It's a lightweight, self-contained, open source database that doesn't require a separate server process.

1.4.4 Development Tools:

- **Android Studio:** Android Studio is an integrated development environment (IDE) specifically built for Android app development. It provides a comprehensive set of tools for designing, developing, testing, and debugging Android apps.
- **Thonny IDE:** Thonny is a free and open-source Python IDE designed for beginners. It provides a user-friendly interface with features like syntax highlighting, code completion, and debugging tools.
- **IntelliJ IDEA:** IntelliJ IDEA is an IDE from JetBrains that supports various programming languages, including Java and Kotlin. It offers a wide range of features for development, testing, and debugging applications.

1.4.5 Design Tool:

- **Figma:** Figma is a web-based design tool used for creating user interfaces and user experiences for web and mobile applications.

1.4.6 Hardware:

- **Raspberry Pi 3B+:** Raspberry Pi is a series of small single-board computers, it's a popular platform for hobbyists and makers due to its affordability and wide range of capabilities.

Table 1.2: Raspberry Pi 3B+ Specifications

Sr. No.	Specification	
1	Processor	1.4 GHz 64-bit quad-core ARM Cortex-A53 CPU (BCM2837B0)
2	Memory	1 GB LPDDR2 SDRAM
3	Graphics	VideoCore IV 3D graphics core @ 400MHz/300MHz
4	Storage	MicroSD card slot
5	Connectivity	Gigabit Ethernet port Dual-band Wi-Fi (2.4 GHz and 5 GHz) IEEE 802.11.b/g/n/ac Bluetooth 4.2 Low Energy (BLE) 4 x USB 2.0 ports 40-pin GPIO header for connecting various electronic components
6	Power	Micro USB power supply (5V, 2.5A)
7	OS	Pi Desktop version 4.3 (based on Debian 11)

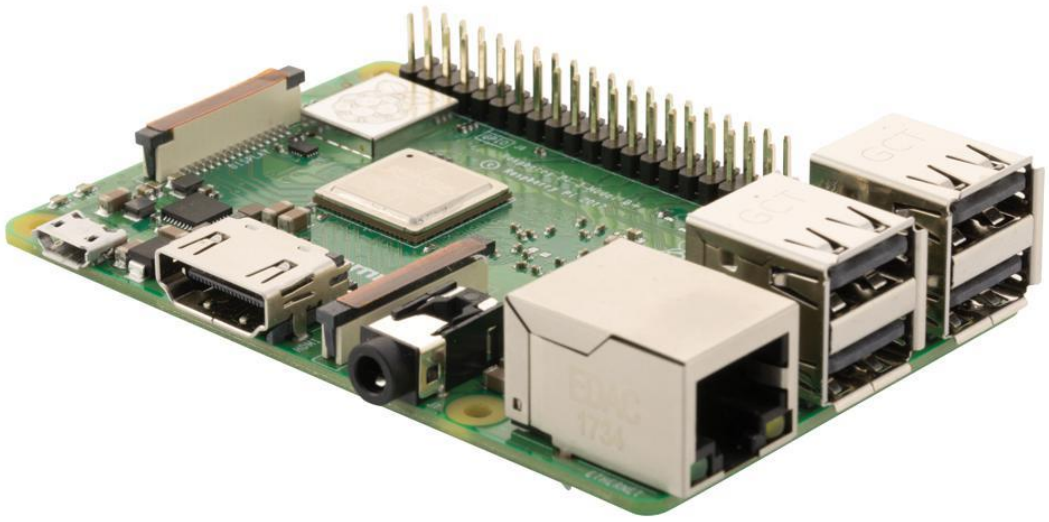


Figure 1.1: Raspberry Pi 3B+

1.5. Project Work Break Down

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 1.2.

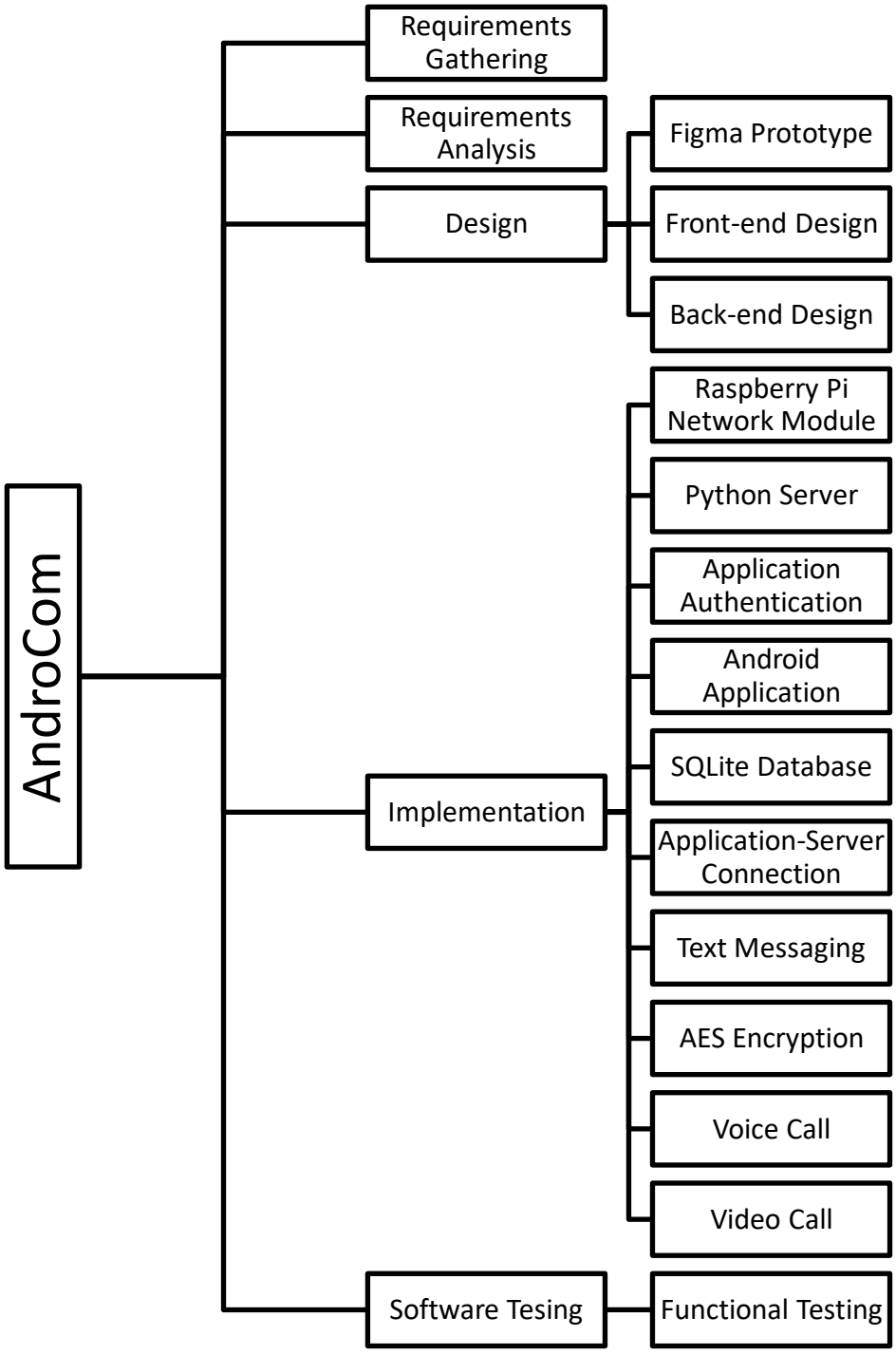


Figure 1.2: Project Work Breakdown

1.6. Project Time Line

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 1.3.

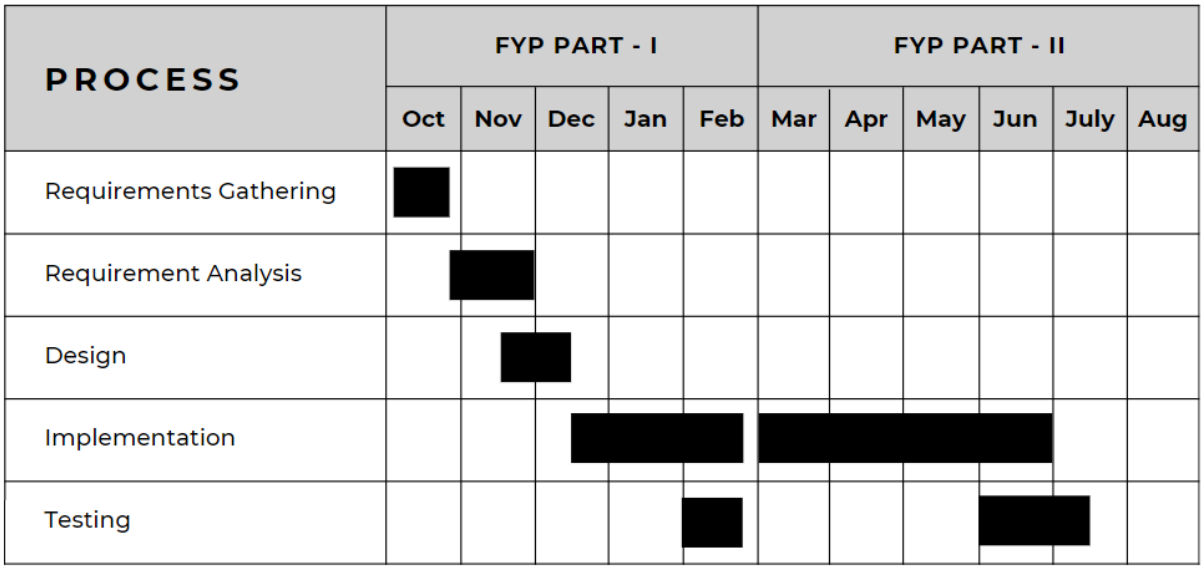


Figure 1.3: Project Time Line

Chapter 2

Requirement Specification and Analysis

Requirement analysis in software development is the process of understanding what the software needs to do. This involves gathering information from stakeholders, analyzing that information, and documenting the requirements in a clear and concise way. This Chapter documents the specification and analysis of requirements for AndroCom. Furthermore, it 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	Configure Raspberry Pi for ad hoc network	Core	Complete
2	User profile setup in android application	Core	Complete
3	Broadcast active user list to android application	Core	Complete
4	Block or unblock users	Intermediate	Complete
5	Mute notifications	Intermediate	Complete
6	Text messaging	Core	Complete
7	Voice Calling	Core	Complete
8	Video Calling	Core	Complete
9	Mute mic or disable camera within a call	Intermediate	Complete

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	End-to-end text encryption	Security
3	User friendly UI	Usability
4	Optimize resource usage on Raspberry Pi	Performance
5	Low power consumption and lightweight android application	Performance
6	Optimize ad hoc network usage	Performance

2.3. Selected Functional Requirements

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

Table 2.3: Selected Functional Requirement

S. No.	Functional Requirement	Type
1	Block or unblock users	Intermediate
2	Mute notifications	Intermediate
3	Voice Calling	Core
4	Video Calling	Core
5	Mute mic or disable camera within a call	Intermediate

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.

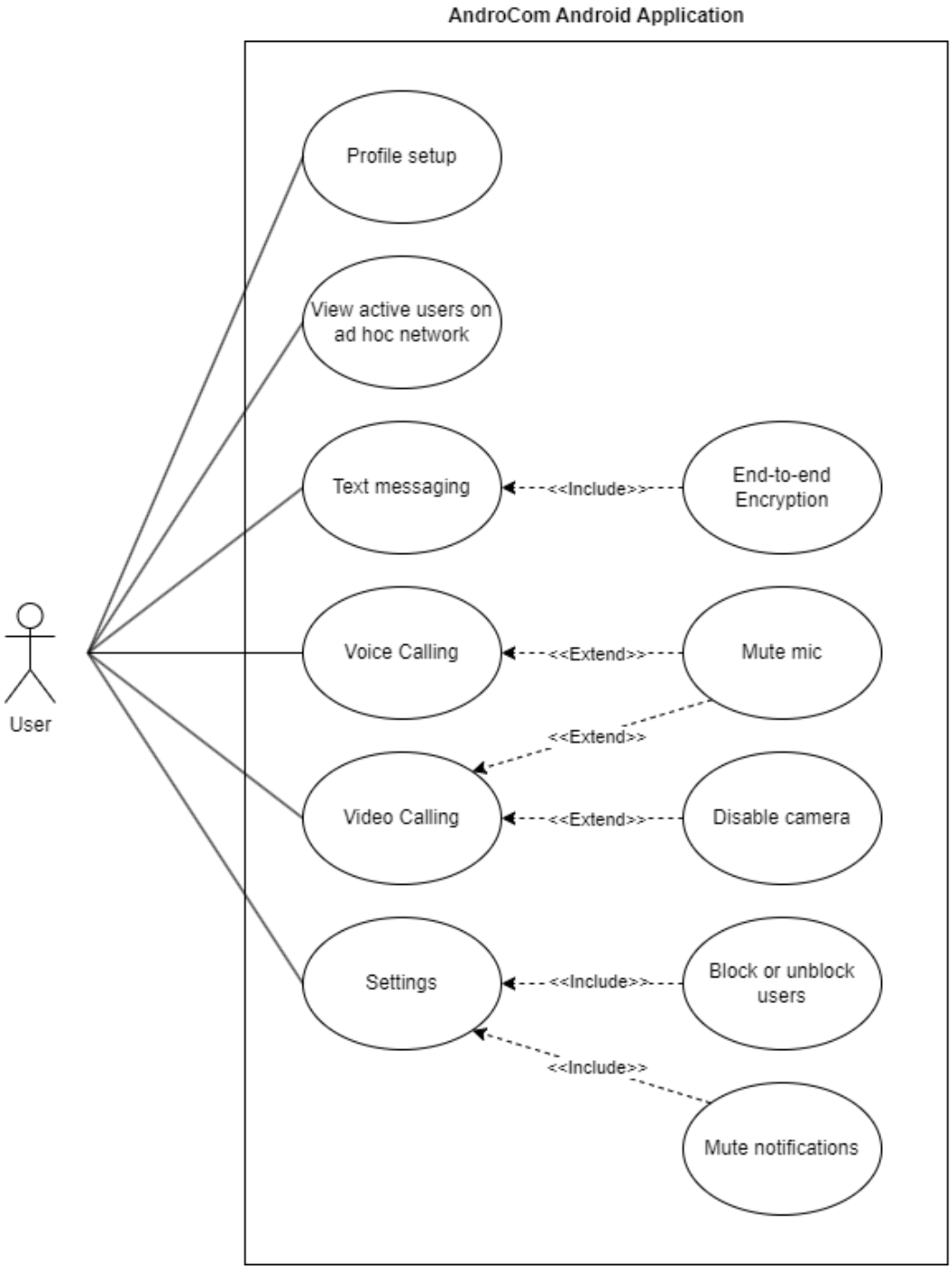


Figure 2.1: System Use Case Diagram

Use Case 1 (Configure Raspberry Pi for Ad Hoc Network):

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 messaging):

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):

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 notifications):

Table 2.10: Use Case 7

Use Case ID:	UC7		
Use Case Name:	Mute notifications		
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.		

Use Case 8 (Video call):

Table 2.11: Use Case 8

Use Case ID:	UC8		
Use Case Name:	Video call with active users		
Created By:	Wasia	**Last Updated By:**	Muhammad Harris
Date Created:	21-4-2024	**Last Revision Date:**	25-4-2024
Actors:	User		
Description:	User can initiate and receive a video call from an active user within application.		
Trigger:	The user will press the video 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 video call.		
Post conditions:	The user is able to make a video call with another active user on the network.		
Normal Flow:	User	System	
	1. The user selects the recipient of the video call and initiates the video call.	The system sends a video call request to the recipient over the network.	
	2. The users are able to talk to each other over the video connection.	The system establishes a video connection between two users.	
Alternative Flows:	The recipient user rejects the video call request and call is not established.		
Exceptions:	1. User is not logged into the application. 2. Select active user.		

Use Case 9 (Mute mic or turn-off camera):

Table 2.12: Use Case 9

Use Case ID:	UC9		
Use Case Name:	Mute mic and turn-off the camera		
Created By:	Umer Ahmed	**Last Updated By:**	Muhammad Harris
Date Created:	21-4-2024	**Last Revision Date:**	25-4-2024
Actors:	User		
Description:	User mutes their microphone and turn-off the camera to avoid unwanted noise and to maintain focus on call content.		
Trigger:	The user will press the mute mic and turn-off the camera icon on the call screen.		
Preconditions:	The user must be connected to the network and is logged into the application. The user is participating in a call.		
Post conditions:	The user's microphone and camera can be independently muted or turned off, and no audio or video is transmitted by the user's device.		
Normal Flow:	User	System	
	1. The user decides to mute the microphone or turn-off the camera.	The system provides both microphone mute and turn-off camera buttons to the user.	
	2. The user either clicks the mute or turn-off camera button.	The system confirms the actions of user.	
Alternative Flows:	The user clicks the mute or turn-off camera button but the microphone remains unmuted and the camera isn't turned off.		
Exceptions:	The call with another user crashes and the mute mic or turn off camera functionality becomes unavailable.		

2.5. System Sequence Diagrams

Sequence diagrams are created to show the sequence of events among user and the system to complete an action / use case. Following are the system sequence diagrams for AndroCom.

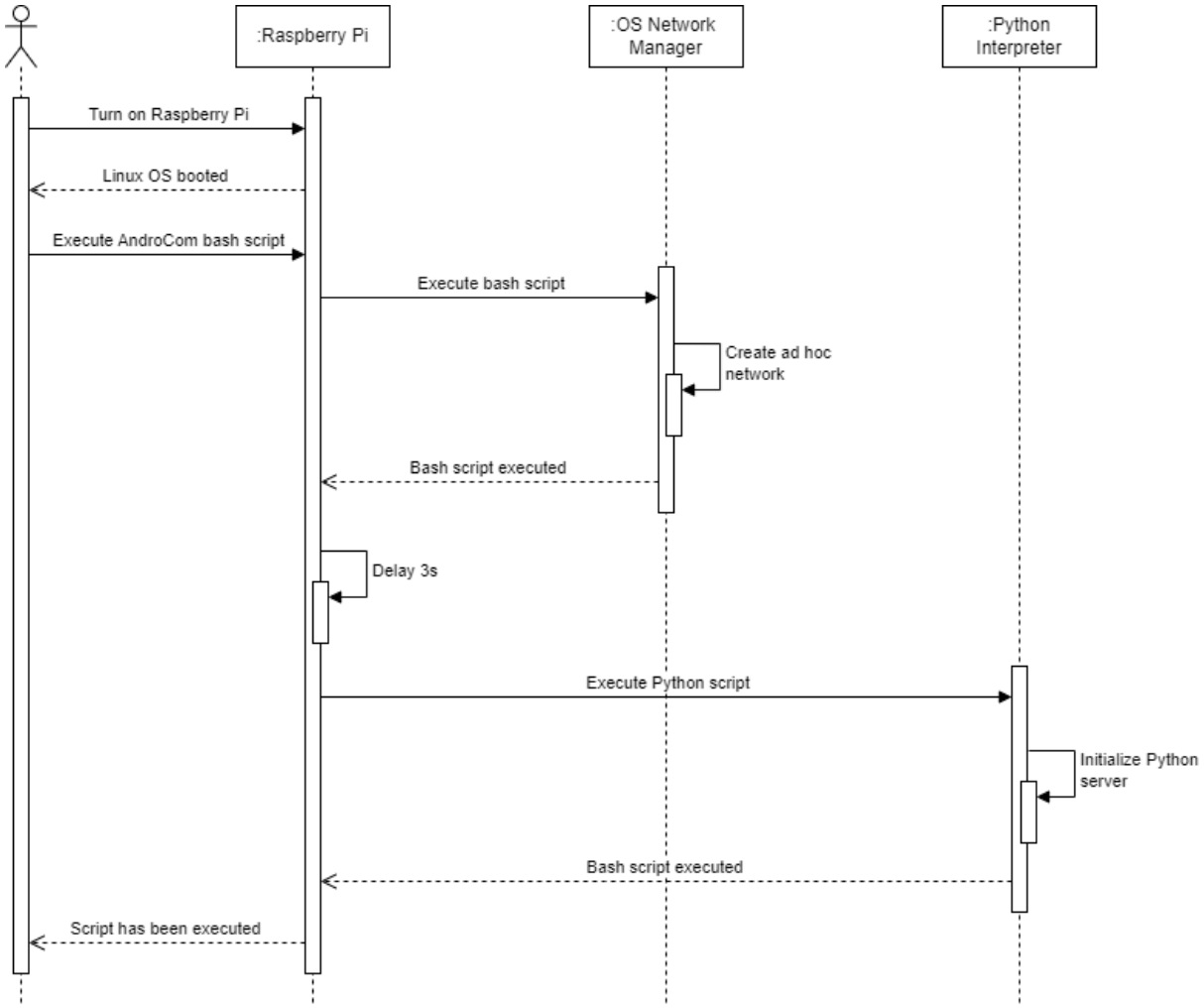


Figure 2.2: System Sequence Diagram 1 (Configure Raspberry Pi)

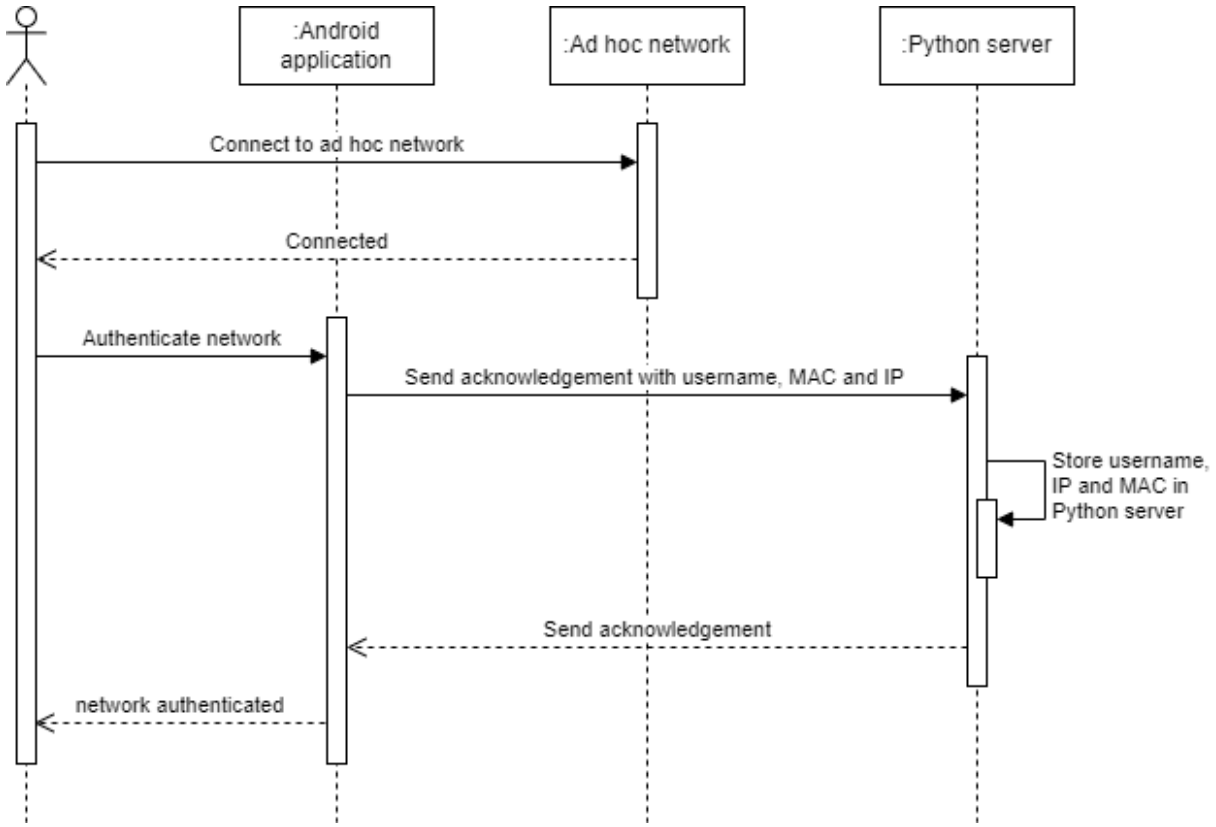


Figure 2.3: System Sequence Diagram 2 (Network authentication)

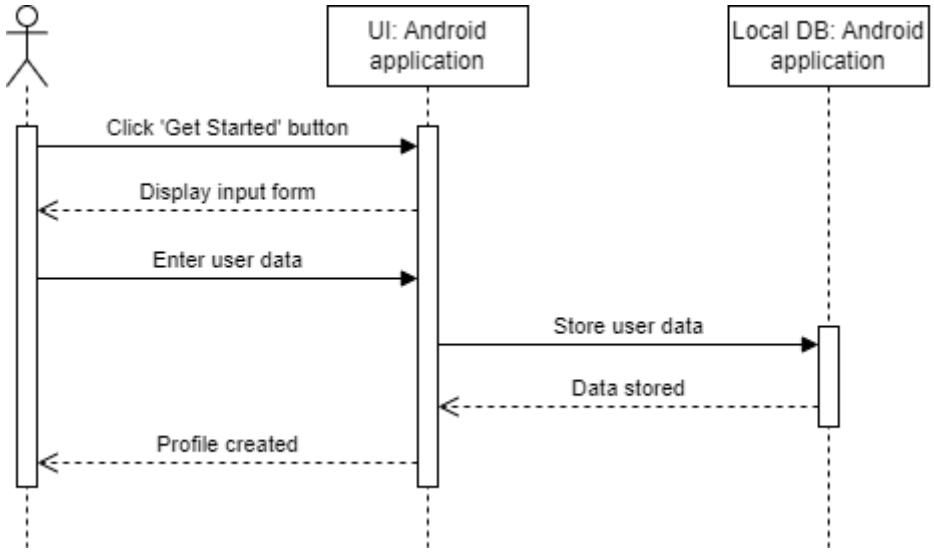


Figure 2.4: System Sequence Diagram 3 (Profile setup)

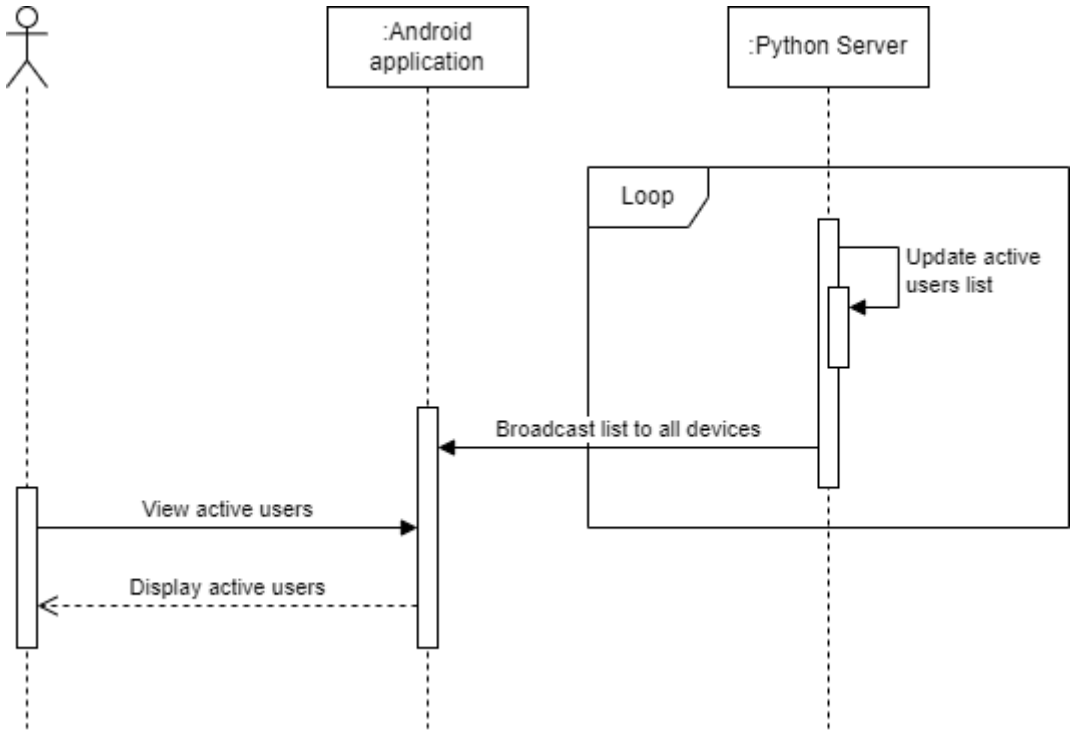


Figure 2.5: System Sequence Diagram 4 (Broadcast active user list)

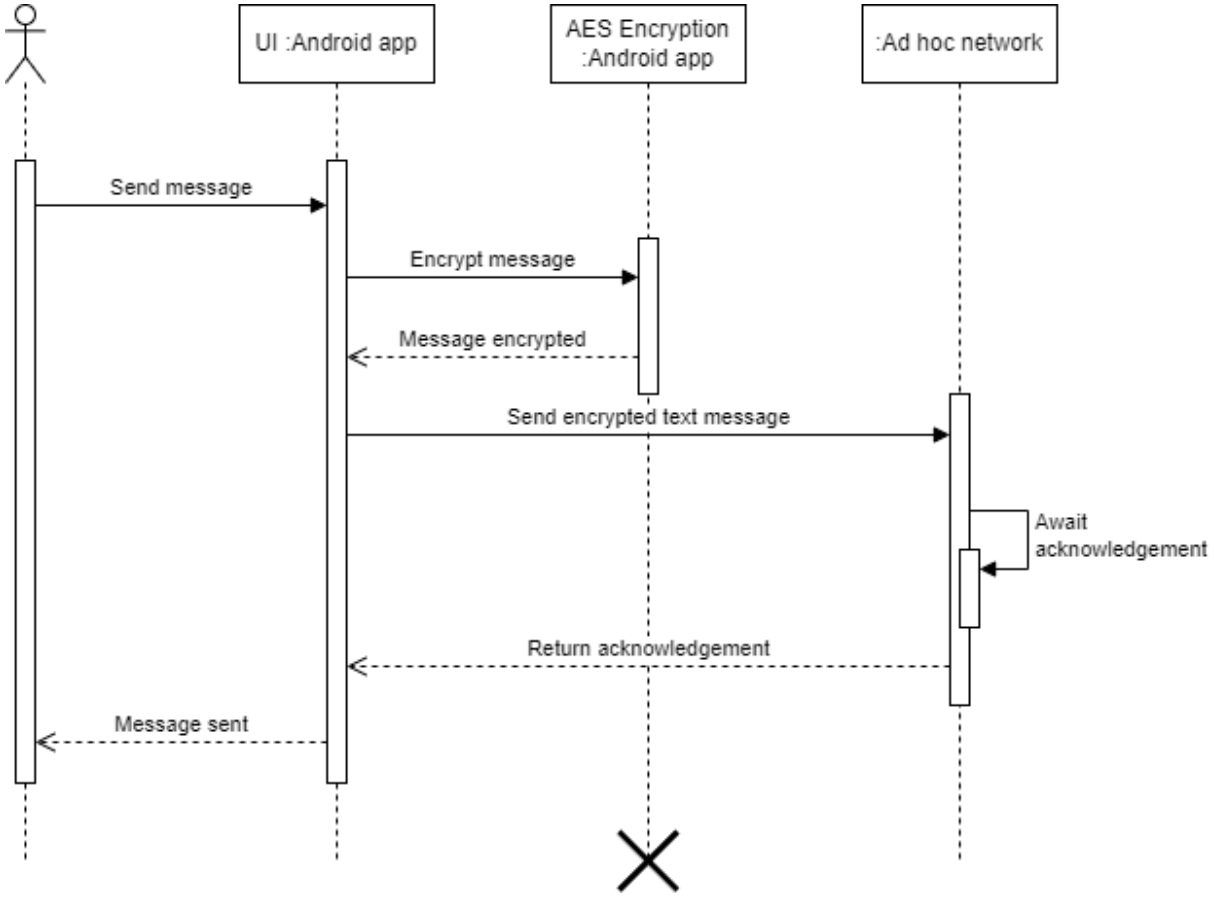


Figure 2.6: System Sequence Diagram 5 (Send text message)

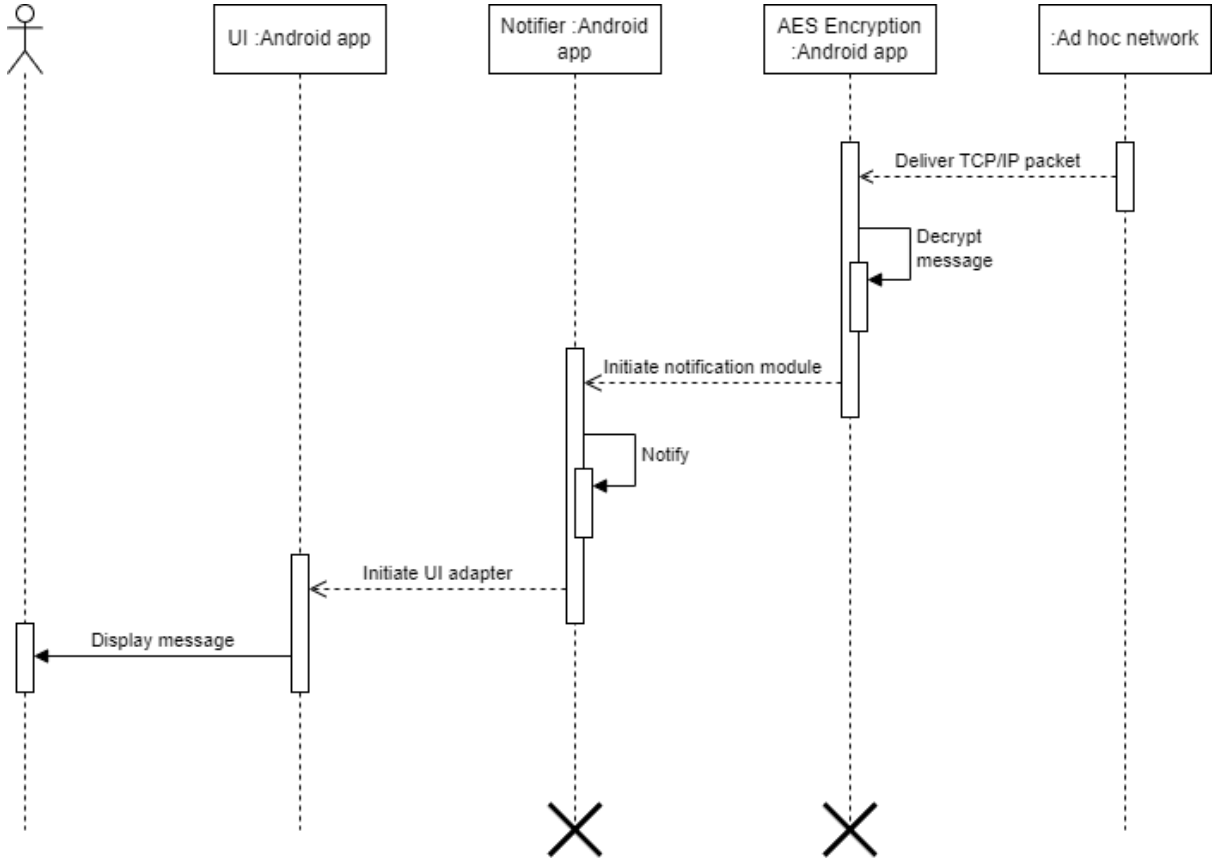


Figure 2.7: System Sequence Diagram 6 (Receive text message)

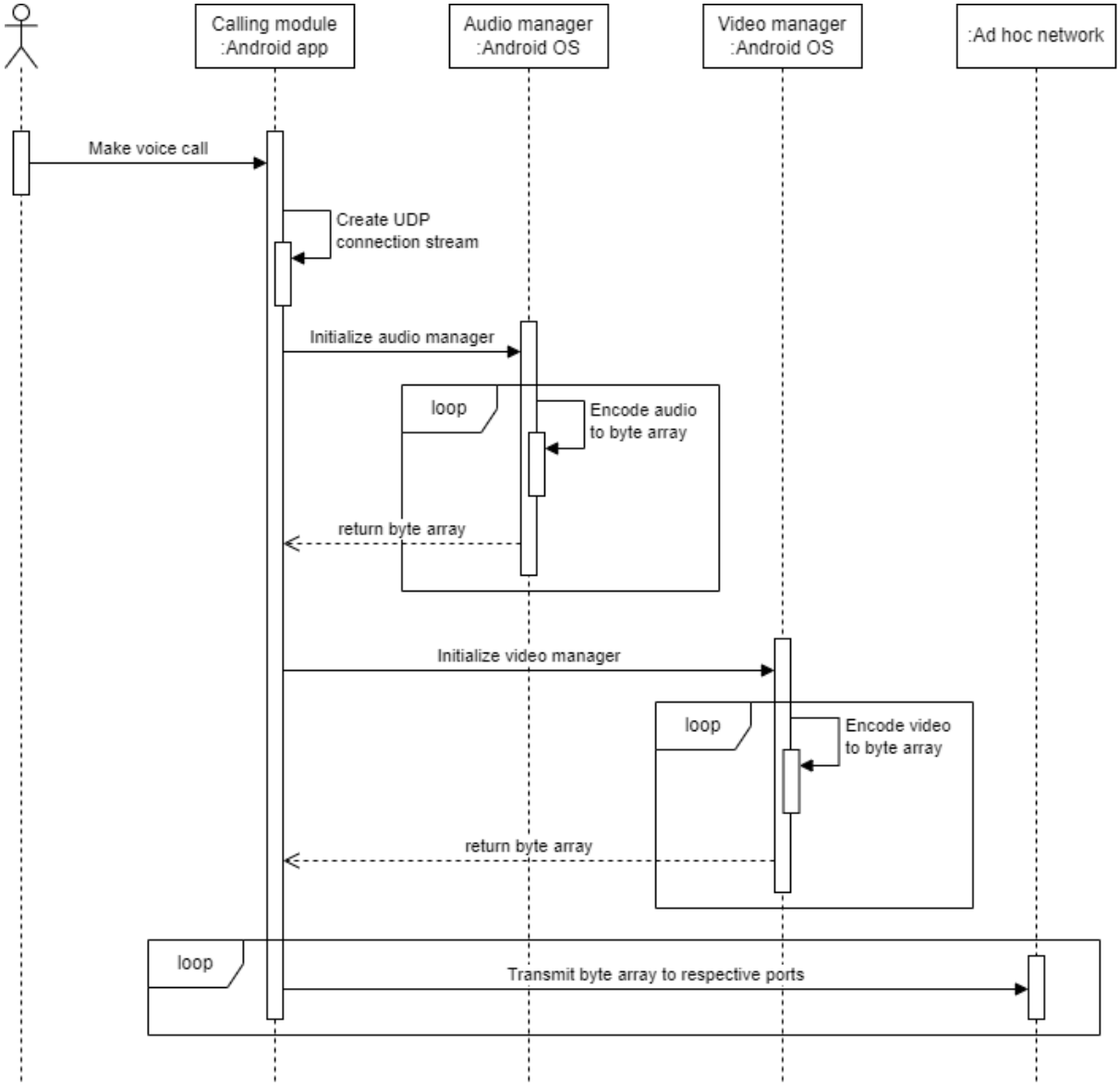


Figure 2.8: System Sequence Diagram 7 (Make voice or video call)

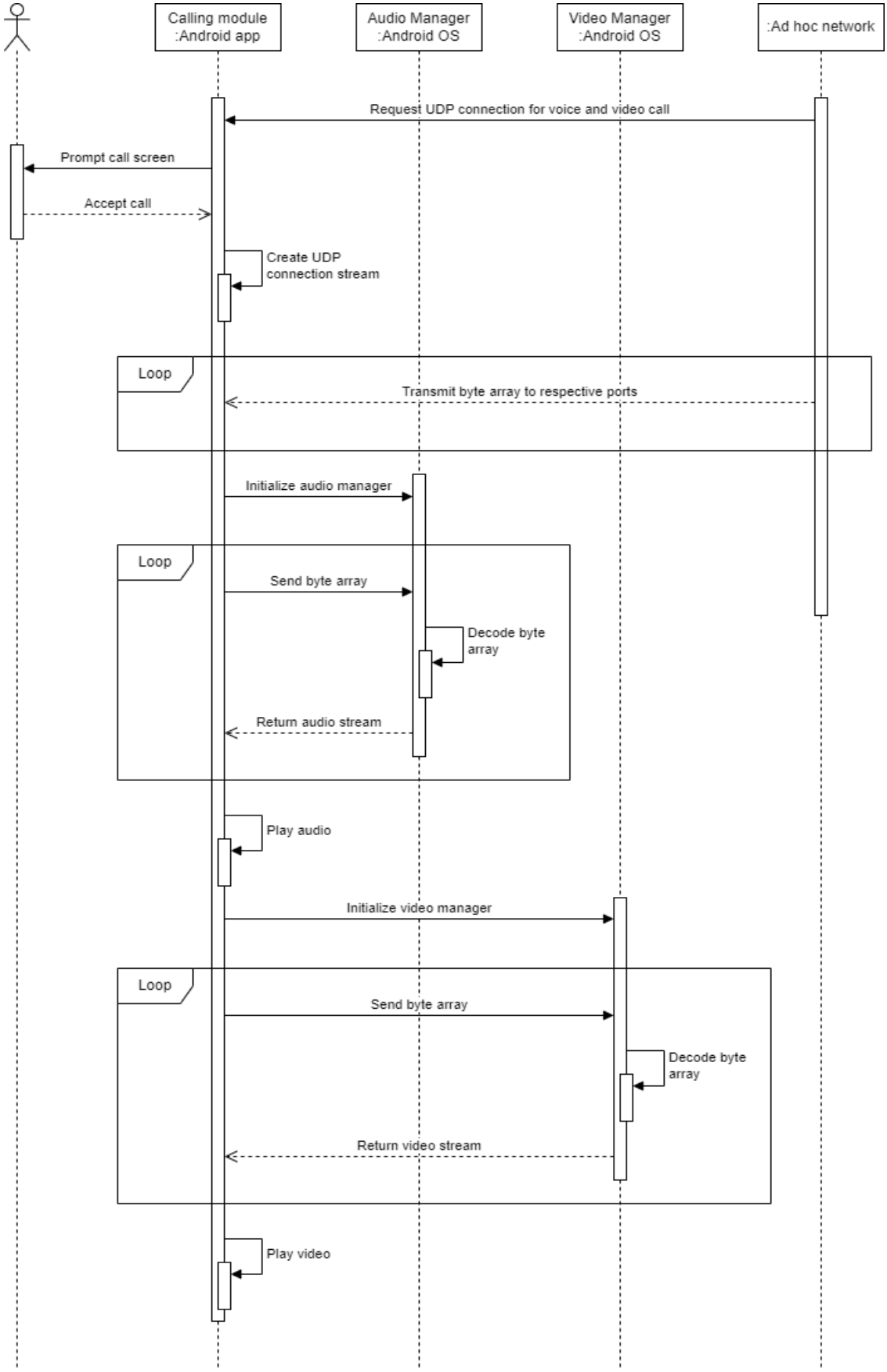


Figure 2.9: System Sequence Diagram 8 (Receive voice or video call)

2.6. Domain Model

A domain model is a simplified view of the important parts and how they relate in a particular software application's area of expertise. It helps developers and others on the project understand what the software is supposed to do. Domain Model for AndroCom is provided in *figure 2.10*.

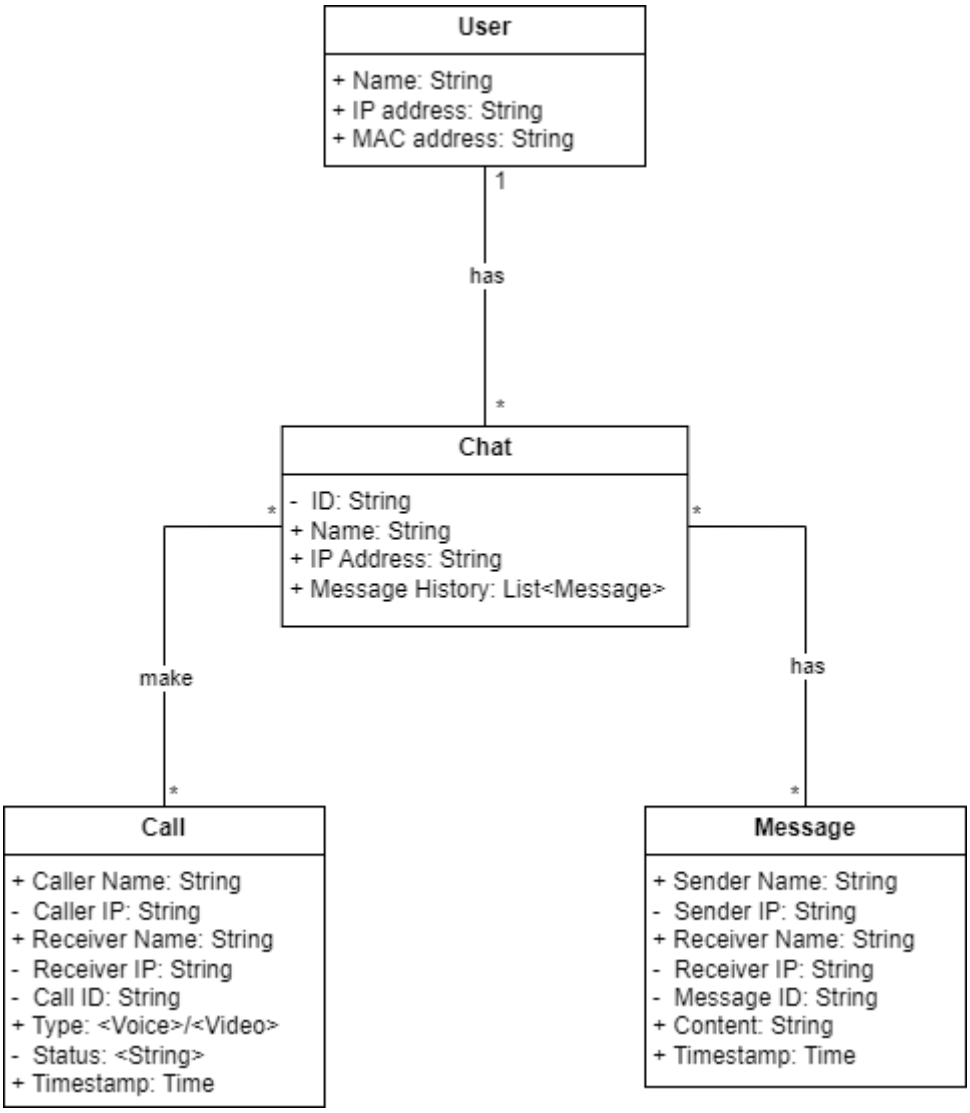


Figure 2.10: Domain Model

2.7. System Architecture

The system architecture of AndroCom is designed for reliable and secure communication without an internet connection. It consists of a client app and a Raspberry Pi 3B+. The client app establishes a connection to the ad hoc network managed by a Python server running on the Raspberry Pi. This server facilitates message exchange (sending and receiving) and voice/video calls between clients.

The Raspberry Pi, due to its portability, can be attached to vehicles and drones. Client apps and the server communicate using TCP/IP sockets. Additionally, the server utilizes UDP sockets for broadcasting messages to all clients on the ad hoc network.

Diagrams illustrating the system architecture and design can be found in figures 2.11 and 2.12.

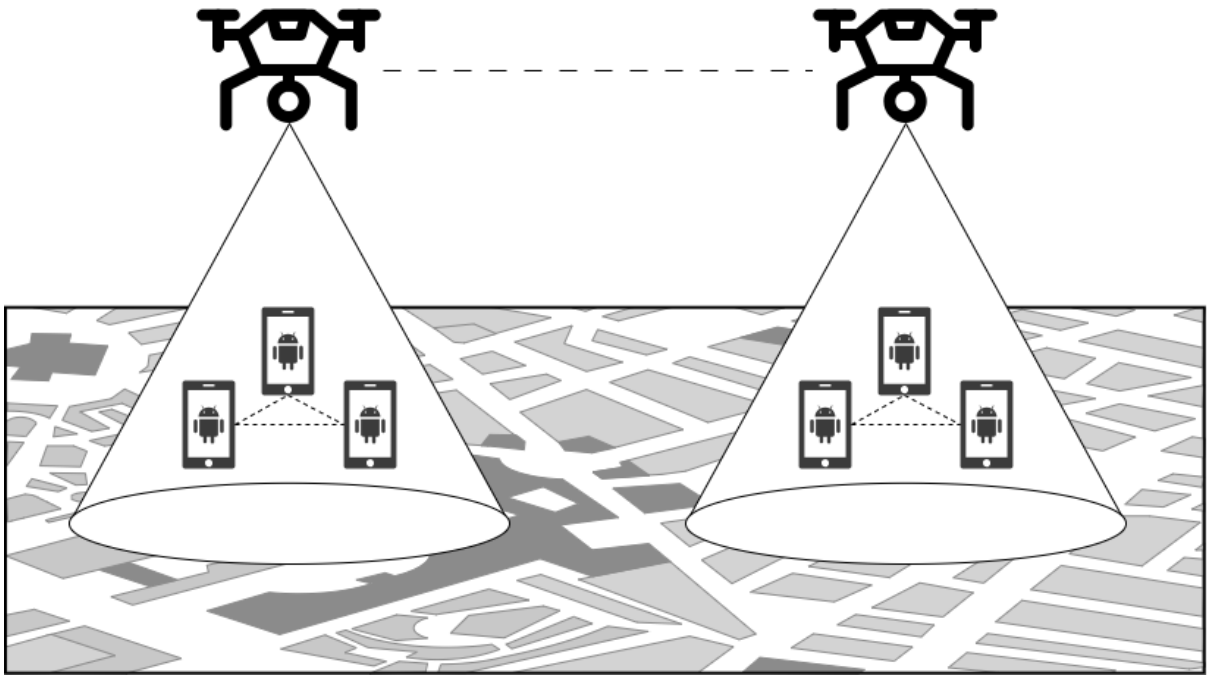


Figure 2.11: System Design

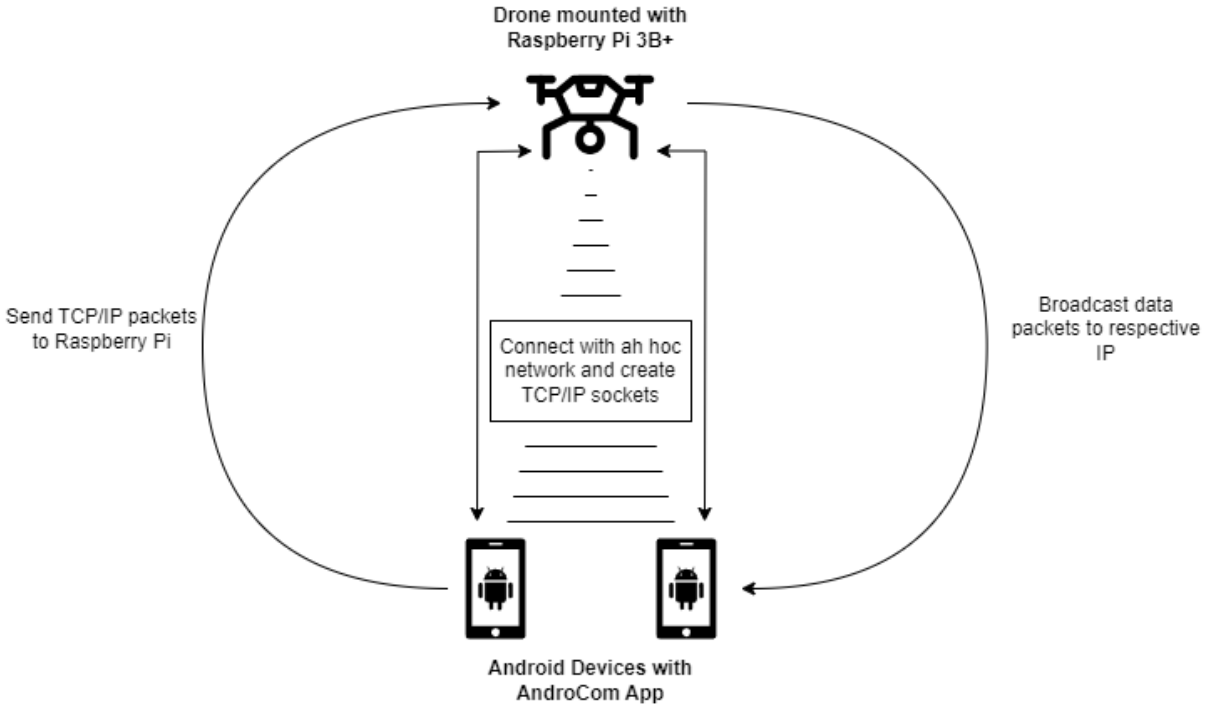


Figure 2.12: System Architecture

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

Layer	Description
Application Layer (AndroCom App)	This layer is responsible for providing user interfaces and other key functionalities.
Network Layer (Raspberry Pi)	This layer is responsible for communicating with the Raspberry Pi over the ad hoc network.

3.1.1. 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 Raspberry Pi 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 Raspberry Pi.

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*.

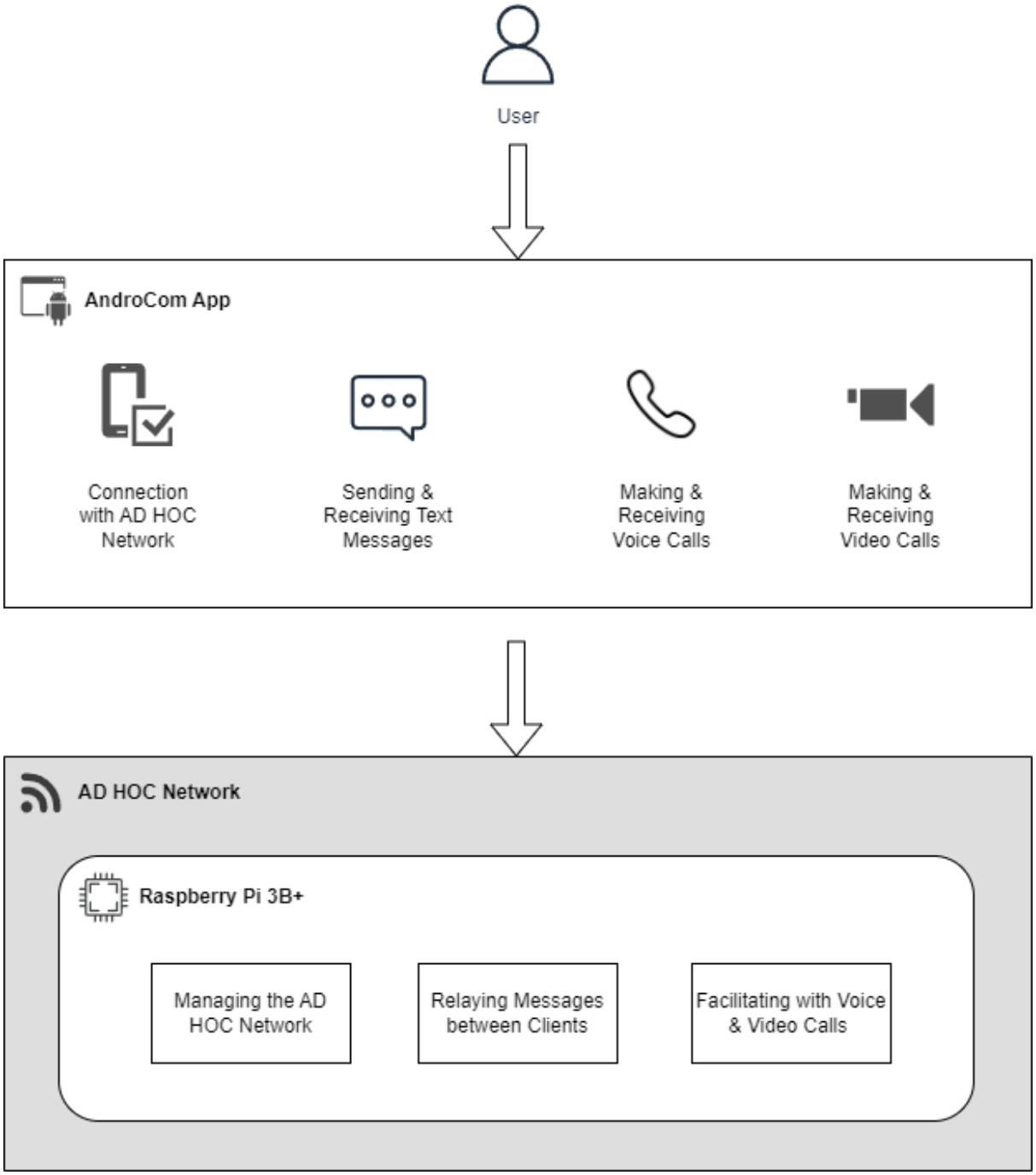


Figure 3.1: Software Architecture Diagram

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.

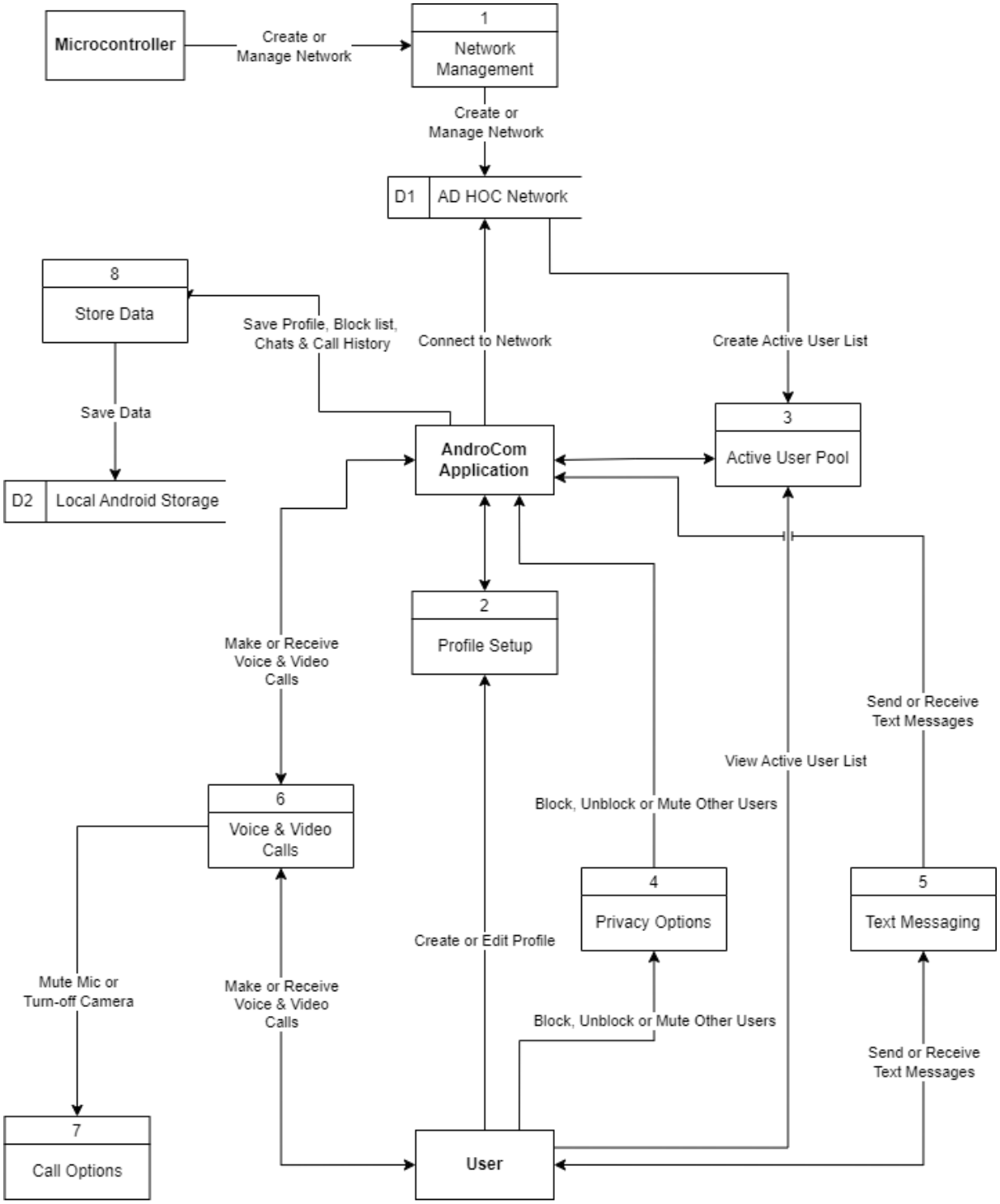


Figure 3.2: Data Flow Diagram

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

This screen appears whenever the user opens the Androcom Android app.

Figure 3.4: Getting Started

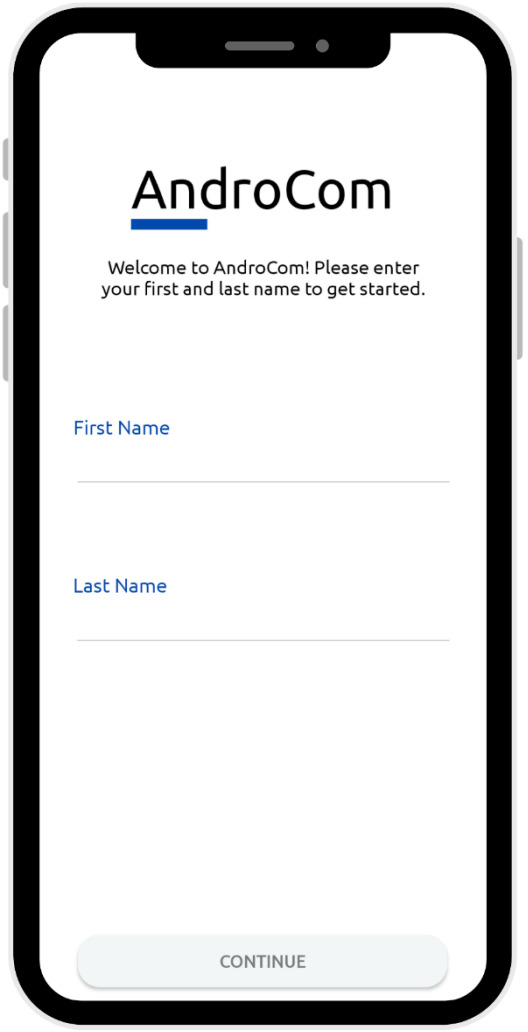


Figure 3.5: Initial Form

The above screens are shown to the user when they open the app for the first time. After touching the 'Get Started' button, they are shown an input form requiring their first and last name. They can continue using the app once both input fields are filled.

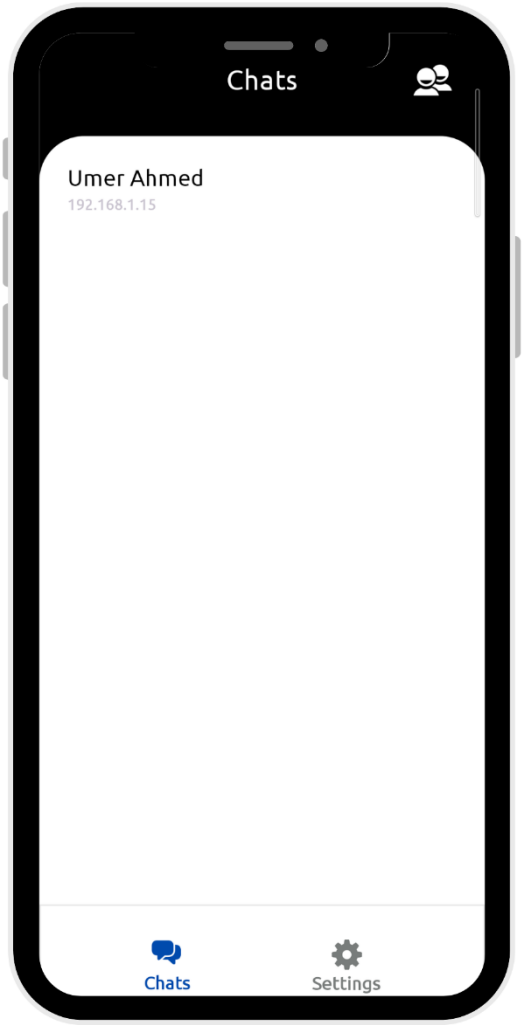


Figure 3.6: Recent Chats

This screen is shown whenever the user opens our app. It displays all the users' recent chats. Additionally, the user can click the 'Active Users' button to view users currently active on the network.

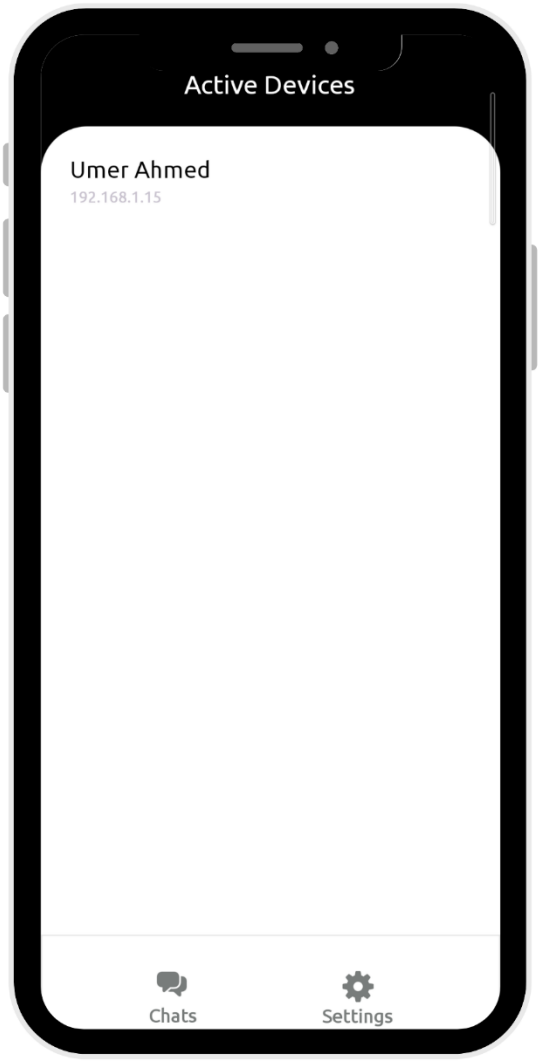


Figure 3.7: Active Users List

On this screen, users can view other users who are currently active on the ad hoc network. They can see information about these active users, such as name and IP address. To initiate a chat, users can click on a user's name. Additionally, users can navigate to the home screen or settings through the navigation bar.

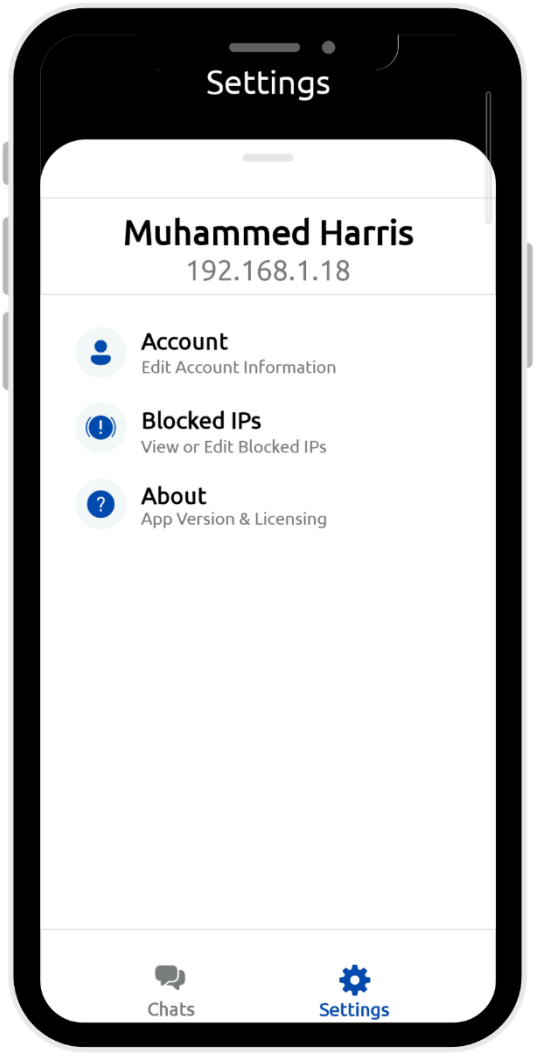


Figure 3.8: Settings

On the settings screen, the user can view their own information, such as name and IP address. They also have several options available, including changing their name, unblocking blocked users, and viewing the app's licensing information.

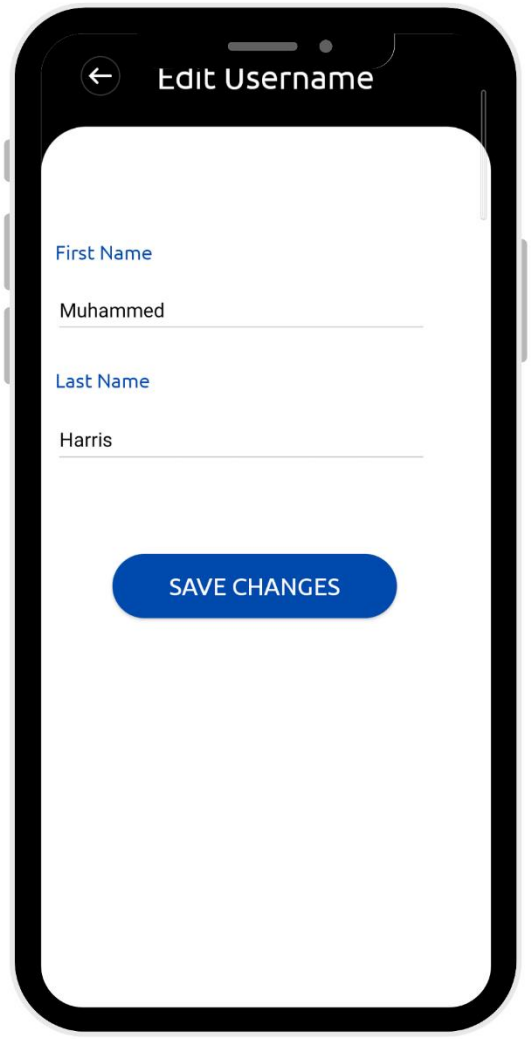


Figure 3.9: Edit Username

On this screen, the user can edit their username.

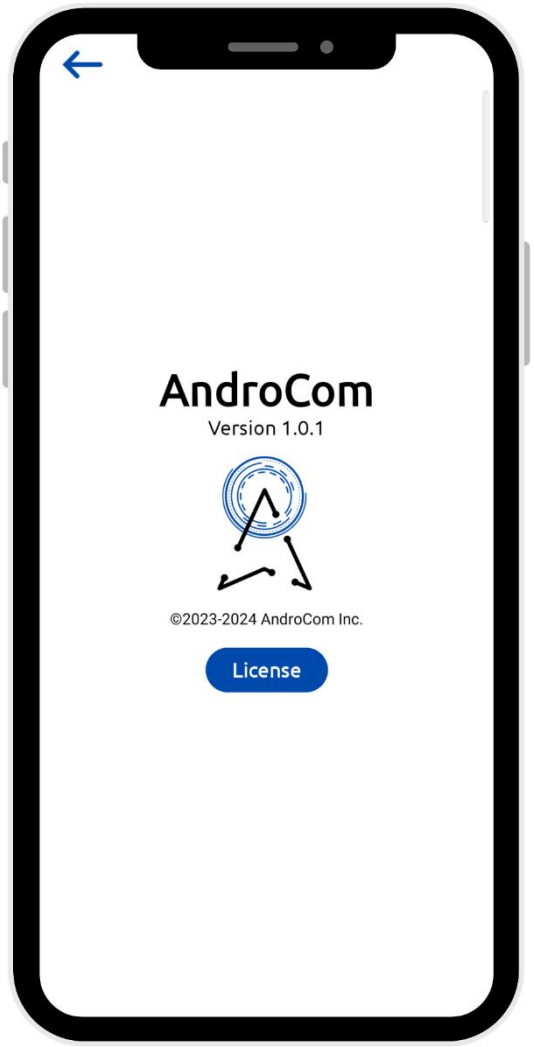


Figure 3.10: License Popup

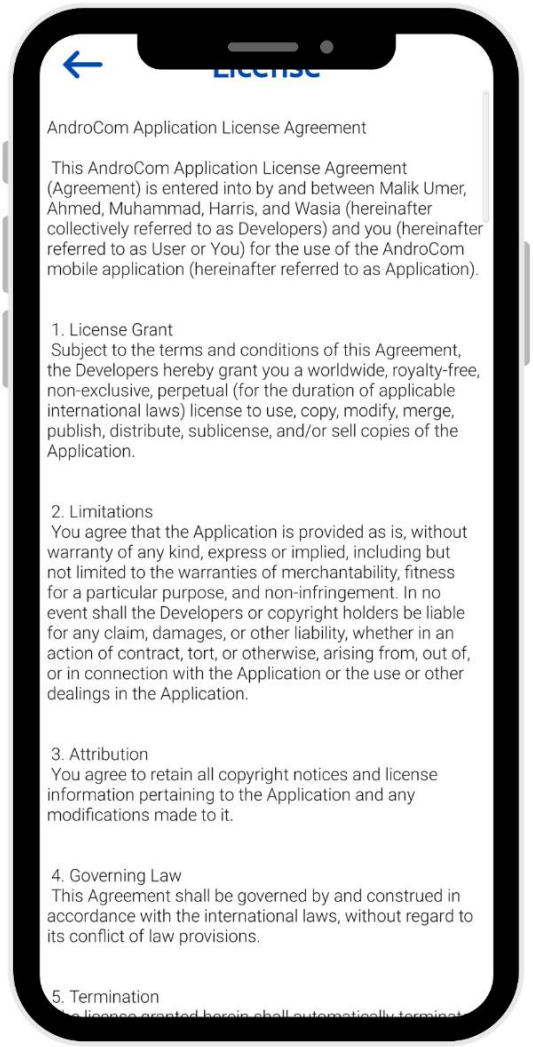


Figure 3.11: License

This screen can be accessed via settings. Here, the user can view the app's license.

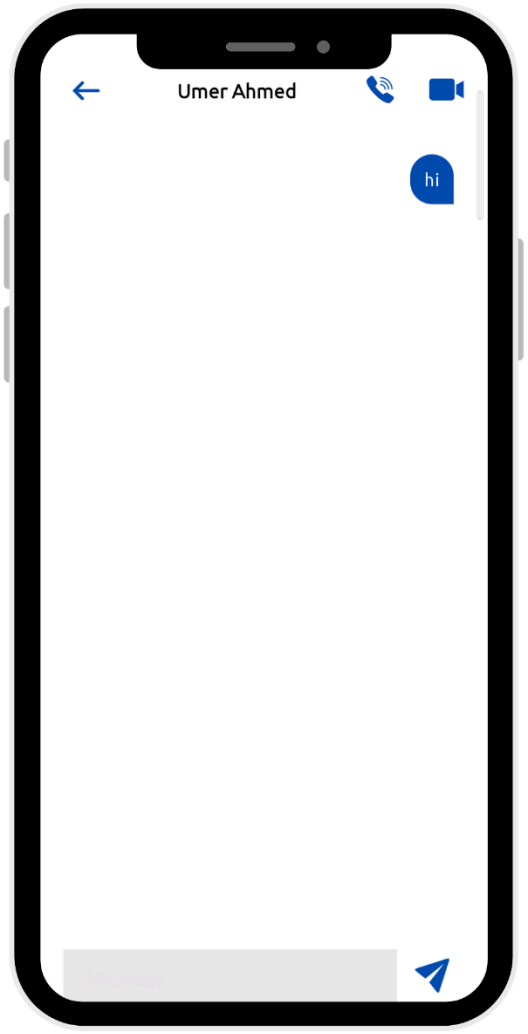
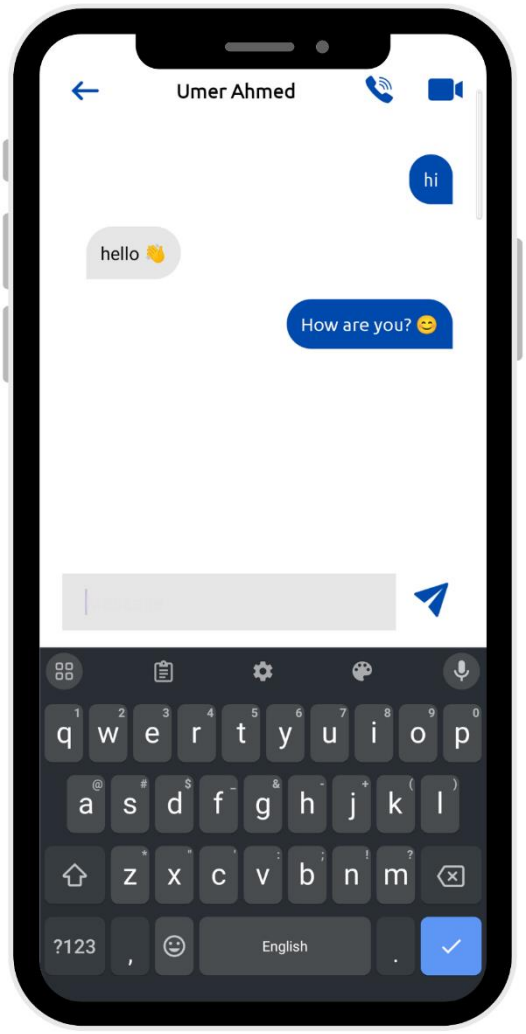


Figure 3.12: Chat



3.13: Chat with Keyboard

On this screen, users can chat with other users who are currently active on the ad hoc network. They can send emoji along with text messages. Additionally, users can make voice and video calls with the same user.

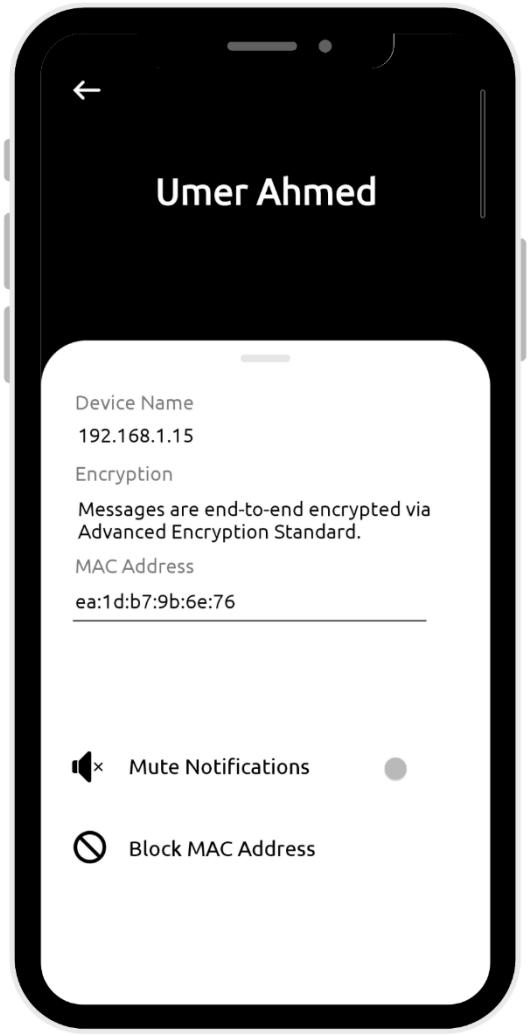


Figure 3.14: Chat Details

On this screen, the user can view information about the contact they are chatting with, such as name, IP address, and MAC address. They can also mute chat notifications or block the user.

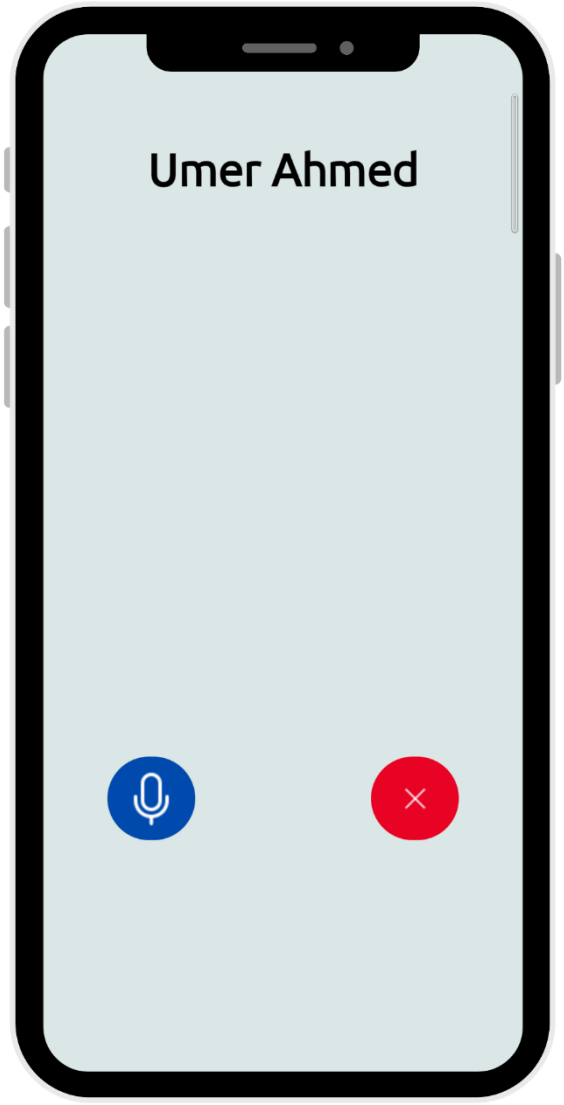


Figure 3.15: Voice Call

This screen appears when a user initiates a voice call.

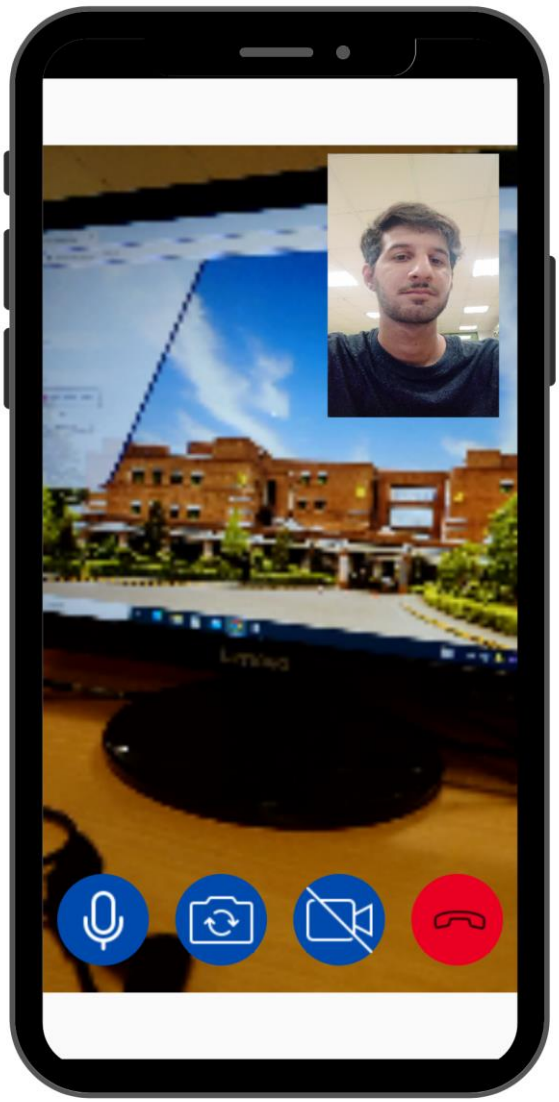


Figure 3.16: video Call

This screen appears when a user initiates a voice call.

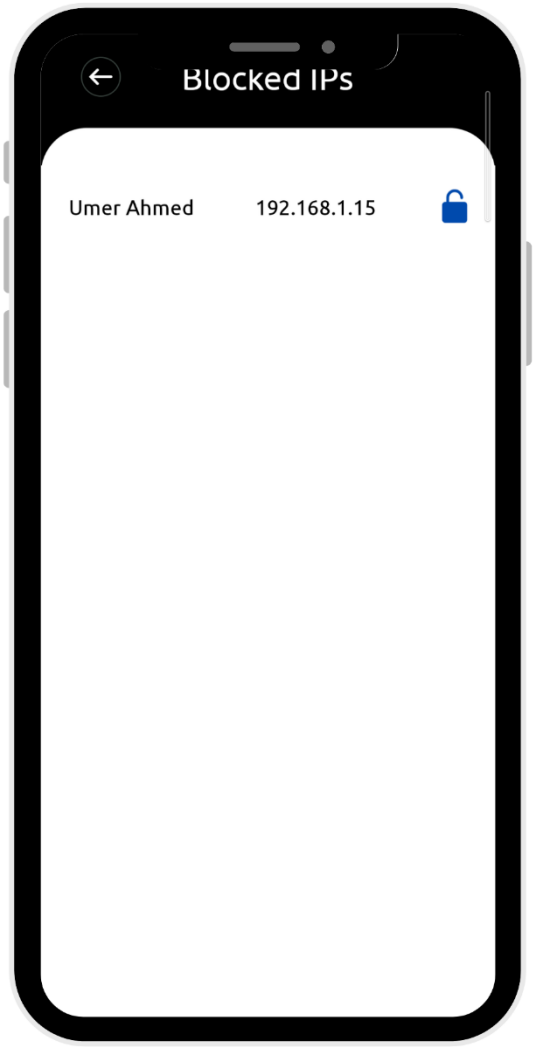


Figure 3.17: Block Settings

This screen can be accessed through the app settings. Here, the user can unblock blocked IP addresses.

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 & Bash**
    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 & Java**
    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 (Raspberry Pi Scripting)**: Bash was for creating AD HOC network in the Raspberry Pi, facilitating communication between android application and Raspberry Pi.
- **Python (Network Side Programming):** Utilized for programming the Raspberry Pi, ensuring efficient execution of tasks on the Raspberry Pi side such as sending IP addresses & MAC addresses of connected devices to android application.
- **Kotlin & Java (Android Application):** Chosen as the primary languages for Android application development, providing a robust framework for building user interfaces and implementing functionalities.
- **Thonny IDE (Python on Raspberry Pi):** Specifically used for Python development on the Raspberry Pi, 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 four major libraries used in the development of our project,

- **Socket (Python):** It was used for creating UDP packets on Raspberry Pi and sending it over the android application through AD HOC network.
- **Java Socket (Kotlin & Java):** 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 Raspberry Pi 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 network information
SSID="Ad-HocNetwork"

# Delete existing connection
sudo nmcli connection delete $SSID

# 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

# 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
```

Description: This script sets up an ad-hoc WiFi network using NetworkManager. Initially, it checks if NetworkManager is installed on the system; if not, it prompts the user to install it and exits. The script then defines network details, including the SSID and password. It deletes any existing WiFi connection with the same SSID to avoid conflicts. Next, it creates a new WiFi connection with a /27 subnet, setting the network address accordingly. The WiFi connection is then activated, and its status is displayed. Additionally, the script includes a function to list connected devices, which is executed when the script is interrupted with Ctrl+C. Optional sections for setting up WiFi security and continuously listing connected devices are commented out but can be enabled if needed.

Snippet 2

```
import socket
import json
import threading
import subprocess
import time
import os

# Global list to store data from apps
Active_User_list = []
Active_User_lock = threading.Lock()

def clear_screen():
    os.system('cls' if os.name == 'nt' else 'clear')

def handle_client_connection(client_socket):
    try:
        request = client_socket.recv(1024).decode('utf-8')
        parsed_data = parse_received_message(request)
        if parsed_data:
            with Active_User_lock:
                existing_entry = next((user for user in Active_User_list if
user["IP Address"] == parsed_data["IP Address"]), None)
                if existing_entry:
                    existing_entry.update(parsed_data)
                else:
                    Active_User_list.append(parsed_data)
        client_socket.send("ACK".encode('utf-8'))
    except Exception as e:
        print(f"Error handling client connection: {e}")
```

```
    finally:
        client_socket.close()

def start_server(host, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.bind((host, port))
        server_socket.listen(5)
        clear_screen()
        print(f"Server listening on {host}:{port}")

        while True:
            client_socket, _ = server_socket.accept()
            clear_screen()
            print("Accepted connection from:", client_socket.getpeername())
            handle_client_connection(client_socket)

def parse_received_message(data):
    try:
        name, ip_address = data.strip('"').split(': ')
        mac_address = get_mac_address(ip_address)
        return {"Name": name, "IP Address": ip_address, "Mac Address":
mac_address}
    except Exception as e:
        clear_screen()
        print(f"Error parsing message: {e}")
        return None

def get_mac_address(ip_address):
    try:
        arp_output = subprocess.check_output(['arp', '-a'],
universal_newlines=True)
        for line in arp_output.split('\n'):
            if ip_address in line:
                parts = line.split()
                return parts[3].strip()
    except subprocess.CalledProcessError as e:
        print(f"Error executing the arp command: {e}")
    return None

def get_connected_devices():
    connected_devices = {}
    try:
        arp_output = subprocess.check_output(['arp', '-a'],
universal_newlines=True)
        for line in arp_output.split('\n'):
            parts = line.split()
            if len(parts) >= 4:
                ip_address = parts[1].strip('()')
```

```
                mac_address = parts[3]
                connected_devices[ip_address] = mac_address
    except subprocess.CalledProcessError as e:
        clear_screen()
        print(f"Error executing the arp command: {e}")
    return connected_devices

def send_udp_broadcast(broadcast_ip, broadcast_port):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as broadcast_socket:
        broadcast_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        while True:
            try:
                connected_devices = get_connected_devices()
                with Active_User_lock:
                    Active_User_list[:] = [user for user in Active_User_list
if user['IP Address'] in connected_devices]
                    json_data = {"active_users": Active_User_list}
                    udp_message = json.dumps(json_data, indent=4)
                    broadcast_socket.sendto(udp_message.encode(),
(broadcast_ip, broadcast_port))
                    clear_screen()
                    print(f"Broadcasted: {udp_message}")
            except Exception as e:
                clear_screen()
                print(f"Error occurred while sending UDP broadcast: {e}")
            time.sleep(2)

if __name__ == "__main__":
    HOST = '192.168.1.1'
    PORT = 49150

    tcp_server_thread = threading.Thread(target=start_server, args=(HOST,
PORT))
    tcp_server_thread.start()

    broadcast_ip = '192.168.1.31'
    broadcast_port = 54321

    udp_broadcast_thread = threading.Thread(target=send_udp_broadcast,
args=(broadcast_ip, broadcast_port))
    udp_broadcast_thread.start()

    tcp_server_thread.join()
    udp_broadcast_thread.join()
```

Description: This script sets up a TCP server to receive client connections and a UDP broadcaster to broadcast the list of active users. It begins by defining global variables and functions for clearing the screen, handling client connections, parsing messages, retrieving MAC addresses, and listing connected devices. The start_server function initializes the TCP server, which listens for incoming connections, parses the received messages, updates the active user list, and sends an acknowledgment to the client. The send_udp_broadcast function periodically broadcasts the list of active users to a specified IP and port. The script is executed by creating and starting threads for both the TCP server and UDP broadcaster, and then waiting for them to finish.

Snippet 3

```
package com.application.androcom

import android.os.Build
import androidx.annotation.RequiresApi
import javax.crypto.Cipher
import javax.crypto.spec.SecretKeySpec
import java.util.Base64

class AESEncryption {

    @RequiresApi(Build.VERSION_CODES.O)
    fun encrypt(text: String, secretKey: String): String {
        val key = generateKey(secretKey)
        val cipher = Cipher.getInstance("AES/ECB/PKCS5Padding")
        cipher.init(Cipher.ENCRYPT_MODE, key)
        val encryptedBytes = cipher.doFinal(text.toByteArray(Charsets.UTF_8))
        return Base64.getEncoder().encodeToString(encryptedBytes)
    }

    @RequiresApi(Build.VERSION_CODES.O)
    fun decrypt(encryptedText: String, secretKey: String): String {
        val key = generateKey(secretKey)
        val cipher = Cipher.getInstance("AES/ECB/PKCS5Padding")
        cipher.init(Cipher.DECRYPT_MODE, key)
        val decodedBytes = Base64.getDecoder().decode(encryptedText)
        val decryptedBytes = cipher.doFinal(decodedBytes)
        return String(decryptedBytes, Charsets.UTF_8)
    }

    private fun generateKey(secretKey: String): SecretKeySpec {
        val keyData = ByteArray(16)
        val secretKeyBytes = secretKey.toByteArray(Charsets.UTF_8)
        for (i in secretKeyBytes.indices) {
            keyData[i % 16] = (keyData[i % 16] + secretKeyBytes[i]).toByte()
        }
```

```
        return SecretKeySpec(keyData, "AES")
    }
}
```

Description: This code defines an AESEncryption class that provides methods for encrypting and decrypting text using the AES (Advanced Encryption Standard) algorithm in ECB mode with PKCS5 padding. The encrypt method converts plaintext into an encrypted string, while the decrypt method reverses the process to obtain the original text. A helper method, generateKey, creates a 16-byte AES key from the provided secret key. The code ensures compatibility with Android versions supporting the required cryptographic libraries.

Snippet 4

```
package com.application.androcom

import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.os.Build
import android.os.Bundle
import android.util.Log
import android.widget.EditText
import android.widget.ProgressBar
import android.widget.TextView
import android.widget.Toast
import androidx.annotation.RequiresApi
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.AppCompatImageView
import androidx.recyclerview.widget.RecyclerView
import com.google.gson.Gson
import kotlinx.coroutines.*
import java.io.IOException
import java.io.OutputStreamWriter
import java.net.DatagramPacket
import java.net.DatagramSocket
import java.net.Socket
import java.net.SocketException


class activity_chat : AppCompatActivity() {

    val LOG_TAG = "UDPchat"
    private val LISTENER_PORT = 50003
    private val BUF_SIZE = 1024
```

```
    private val displayName: String? = null
    private val STARTED = false
    private var IN_CALL = false
    private var LISTEN = false

    private val coroutineScope = CoroutineScope(Dispatchers.Main +
SupervisorJob())

    @RequiresApi(Build.VERSION_CODES.O)
    override fun onCreate(savedInstanceState: Bundle?) {


        val receivedIP = intent.getStringExtra("ip")
        val receivedMac = "${intent.getStringExtra("mac")}"
        val receivedName = "${intent.getStringExtra("username")}"
        Log.i("Test", "Name: ${receivedName}\n IP: ${receivedIP}\n MAC:
$receivedMac");

}


// Start the coroutine to load messages for the given IP every 3 seconds
        coroutineScope.runLoadMessagesCoroutine(receivedIP)

        // event listener for contact profile
        textname.setOnClickListener {
            val intent = Intent(this, activity_UserProfile::class.java)

            intent.putExtra("username", receivedName)
            intent.putExtra("ip", receivedIP)
            intent.putExtra("mac", receivedMac)
            startActivity(intent)

        }

        // event listener for send button
        sendButton.setOnClickListener {
            var message = inputMessage.text.toString()
            val receivedIP = intent.getStringExtra("ip")
            if(message.isNotEmpty()) {
                // send message
                if (receivedIP != null) {
                    Log.i("Bug", "Message is Sent: ${message}")
                    if (!ChatDatabaseHelper(this).isIPBlocked(receivedIP)){
                            Send_Message(Name=receivedName,IP_Address=received
IP,Mac=receivedMac,Message= message)
                    }
```

```
                }
                // reset input to null
                inputMessage.setText("")
            }else{
                runOnUiThread {
                    Toast.makeText(this, "Enter Message Before Sending",
Toast.LENGTH_LONG).show()
                }
            }

        }

        val username = intent.getStringExtra("username")
        if (username != null) {
            textname.text = username
        }
        intent.putExtra("myuser", textname.text.toString())

        // event listener for back
        backicon.setOnClickListener{
            val intent = Intent(this, activity_HomeScreen::class.java)
            startActivity(intent)
            finish()
        }

        chatRecyclerView = findViewById(R.id.chatRecyler)
    }
    // Make sure to cancel the coroutine when it's no longer needed (e.g., in
onDestroy of the activity)
    override fun onDestroy() {
        super.onDestroy()
        Log.i("onDestroy","coroutineScope canceled")
        coroutineScope.cancel()
    }


    // function to send message
    @RequiresApi(Build.VERSION_CODES.O)
    fun Send_Message(Name: String,IP_Address: String,Mac: String, Message:
String) {
        // load user information from SharedPreferences
        val sharedPreferences: SharedPreferences =
getSharedPreferences(PREFS_FILENAME, Context.MODE_PRIVATE)
        val userData = sharedPreferences.getString(USER_DATA_KEY, null)
        val user: activity_setting_up.User?
        var name: String = ""
        if (userData != null) {
```

```
            user = gson.fromJson(userData,
activity_setting_up.User::class.java)
            name = "${user.firstName} ${user.lastName}"
        }

        // adjust mac Address
        Thread {
            try {
                val senderIp = LocalIpAddressProvider().getLocalIpAddress()
                // create an instance of encryption
                val aes = AESEncryption()
                // encrypt message
                val encryptedMessage =
aes.encrypt("{$senderIp:$name:${LocalIpAddressProvider().getLocalIpAddress()}}
: $Message", "AndroCom")

                // socket programming
                val sendSocket = Socket(IP_Address, SocketPorts.MESSAGE_PORT)
                val writer = OutputStreamWriter(sendSocket.getOutputStream())
                writer.write(encryptedMessage)
                writer.flush()
                sendSocket.close()

                // save message to SQLite database
                val sentMessage = Message(Message, true)
                saveMessage(message=sentMessage, receiverIp= IP_Address,
name=Name,mac=Mac,isSent = true)

            } catch (e: IOException) {
                Log.e("Exception", "error sending message: $e")
            }
        }.start()

    }


    // function to save message to SQLite database
    @RequiresApi(Build.VERSION_CODES.O)
    fun saveMessage(message: Message ,receiverIp: String, isSent:
Boolean,name: String,mac:String) {

        val dbHelper = ChatDatabaseHelper(this)
        val isSaved = dbHelper.addChat(receiver = receiverIp, message =
message.text, name = name,mac=mac, isSent = isSent)

        // if message is saved in database
        if (isSaved) {
            Log.d("saveMessage", "Data saved successfully")
```

```
        } else {
            Log.d("saveMessage", "Failed to save data")
        }
    }
}
```

Description: The provided Kotlin code defines the `activity_chat` class for an Android application, handling the user interface and core functionalities for a chat activity. The class includes methods to send and receive messages, update the chat interface, and save chat data to an SQLite database. On creation, the activity retrieves user details such as IP address, MAC address, and username from the intent and starts a coroutine to periodically load messages from the database. Event listeners are set up for the send button and profile view, allowing the user to send messages and view the contact's profile. The `Send_Message` method handles encrypting the message using AES encryption and sending it over a socket connection to the recipient's IP address, then saves the message to the local SQLite database. The `saveMessage` method logs whether saving the message was successful. The coroutine is canceled in the `onDestroy` method to prevent memory leaks. The activity also includes functionality for starting and stopping a call listener to handle incoming call requests, though details of this listener are not fully included in the provided snippet.

Snippet 5

```
package com.application.androcom;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
import android.util.Log;

public class AudioCall {

    private static final String LOG_TAG = "AudioCall";
    private static final int SAMPLE_RATE = 16000; // Hertz
    private static final int BUF_SIZE = SAMPLE_RATE / 2; // Bytes
    private InetAddress address;
    private int port = 50000; // Port for audio packets
    private boolean mic = false; // Mic enabled?
```

```
    private boolean speakers = false; // Speakers enabled?

    public AudioCall(InetAddress address) {
        this.address = address;
    }

    public void startCall() {
        mic = true;
        speakers = true;
        new Thread(this::startMic).start();
        new Thread(this::startSpeakers).start();
    }

    public void endCall() {
        mic = false;
        speakers = false;
    }

    private void startMic() {
        Log.i(LOG_TAG, "Mic thread started.");
        try (AudioRecord audioRecorder = new
AudioRecord(MediaRecorder.AudioSource.VOICE_COMMUNICATION, SAMPLE_RATE,
                AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT,
BUF_SIZE * 10);
             DatagramSocket socket = new DatagramSocket()) {
            audioRecorder.startRecording();
            byte[] buf = new byte[BUF_SIZE];
            while (mic) {
                int bytesRead = audioRecorder.read(buf, 0, BUF_SIZE);
                socket.send(new DatagramPacket(buf, bytesRead, address,
port));
                Log.i(LOG_TAG, "Bytes sent: " + bytesRead);
                Thread.sleep(20);
            }
        } catch (Exception e) {
            Log.e(LOG_TAG, "Mic error: ", e);
        }
    }

    private void startSpeakers() {
        Log.i(LOG_TAG, "Speakers thread started.");
        try (AudioTrack track = new AudioTrack(AudioManager.STREAM_MUSIC,
SAMPLE_RATE, AudioFormat.CHANNEL_OUT_MONO,
                AudioFormat.ENCODING_PCM_16BIT, BUF_SIZE,
AudioTrack.MODE_STREAM);
             DatagramSocket socket = new DatagramSocket(port)) {
            track.play();
            byte[] buf = new byte[BUF_SIZE];
```

```
            while (speakers) {
                DatagramPacket packet = new DatagramPacket(buf, BUF_SIZE);
                socket.receive(packet);
                track.write(packet.getData(), 0, packet.getLength());
                Log.i(LOG_TAG, "Bytes received: " + packet.getLength());
            }
        } catch (Exception e) {
            Log.e(LOG_TAG, "Speakers error: ", e);
        }
    }
}
```

Description: This Java class AudioCall facilitates real-time audio communication over a network using UDP. It sets up an audio call by capturing and transmitting microphone input to a specified IP address and port, and by receiving and playing back incoming audio packets. The class uses AudioRecord for capturing audio and AudioTrack for playback. The startCall method initializes threads for both sending and receiving audio. The startMic method captures audio data, packages it into UDP packets, and sends it to the recipient. The startSpeakers method receives incoming audio packets and plays them through the device's speakers. Both methods handle resource management and logging. The endCall method stops the audio transmission and playback.

Snippet 6

```
package com.application.androcom;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.hardware.Camera;
import android.os.Bundle;
import android.view.Surface;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.WindowManager;
import android.widget.ImageButton;
import android.widget.ImageView;

import androidx.appcompat.app.AppCompatActivity;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
```

```
public class ActivityVideoCall extends AppCompatActivity implements
SurfaceHolder.Callback, Camera.PreviewCallback, handleReceiveData {

    private Camera camera;
    private SurfaceView surfaceView;
    private SurfaceHolder surfaceHolder;
    private UDPServer up;
    private UDPServer upa;
    private ImageView imageView;
    private float mCameraOrientation;
    private AudioCall call;
    private boolean IN_CALL = false;
    private int currentCameraId = Camera.CameraInfo.CAMERA_FACING_FRONT;
    private boolean isCameraOn = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        if (getSupportActionBar() != null) {
            getSupportActionBar().hide();
        }
        setContentView(R.layout.activity_video_call);
        initializeViews();
        initializeCamera();
        startCallThread();
        startServiceThread();
    }

    private void initializeViews() {
        surfaceView = findViewById(R.id.surfaceView);
        surfaceHolder = surfaceView.getHolder();
        surfaceHolder.addCallback(this);
        imageView = findViewById(R.id.imageView);

        findViewById(R.id.buttonReject).setOnClickListener(v -> {
            endCall();
            finish();
        });

        ImageButton toggleMic = findViewById(R.id.toggleMic);
        toggleMic.setOnClickListener(v -> {
            call.toggleMic();
            toggleMic.setImageResource(call.micCheck() ? R.drawable.mike :
R.drawable.unmute);
        });
```

```
        findViewById(R.id.buttonSwitchCam).setOnClickListener(v ->
switchCamera());
        findViewById(R.id.buttonOffCam).setOnClickListener(v -> {
            if (isCameraOn) {
                releaseCamera();
            } else {
                initializeCamera();
            }
            isCameraOn = !isCameraOn;
        });
    }

    private void startCallThread() {
        new Thread(() -> {
            try {
                InetAddress address =
InetAddress.getByName(getIntent().getStringExtra("ip"));
                call = new AudioCall(address);
                IN_CALL = true;
                call.startCall();
            } catch (UnknownHostException e) {
                throw new RuntimeException(e);
            }
        }).start();
    }

    private void startServiceThread() {
        new Thread(() -> {
            try {
                up = new UDPServer(8804);
                up.setReceiveCallback(ActivityVideoCall.this);
                up.start();

                upa = new UDPServer(8805);
                upa.setReceiveCallback(data -> {
                    // Handle receiving data here
                });
                upa.start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();
    }

    private void initializeCamera() {
        releaseCamera();
        camera = Camera.open(currentCameraId);
        try {
```

```
            camera.setPreviewDisplay(surfaceHolder);
            camera.setPreviewCallback(this);
            configureCameraOrientation();
            camera.startPreview();
        } catch (IOException e) {
            camera.release();
            camera = null;
        }
    }

    private void switchCamera() {
        currentCameraId = (currentCameraId ==
Camera.CameraInfo.CAMERA_FACING_FRONT)
                ? Camera.CameraInfo.CAMERA_FACING_BACK
                : Camera.CameraInfo.CAMERA_FACING_FRONT;
        initializeCamera();
    }

    private void releaseCamera() {
        if (camera != null) {
            camera.setPreviewCallback(null);
            camera.stopPreview();
            camera.release();
            camera = null;
        }
    }

    @Override
    public void onPreviewFrame(byte[] data, Camera camera) {
        Camera.Size previewSize = camera.getParameters().getPreviewSize();
        int[] rgb = decodeYUV420SP(data, previewSize.width,
previewSize.height);
        Bitmap bmp = Bitmap.createBitmap(rgb, previewSize.width,
previewSize.height, Bitmap.Config.ARGB_8888);
        Bitmap bmpSmall = getSmallBitmap(previewSize, bmp);
        Bitmap bmpSmallRotated = rotateBitmap(bmpSmall, mCameraOrientation);

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        bmpSmallRotated.compress(Bitmap.CompressFormat.WEBP, 100, baos);
        up.sendMsg(baos.toByteArray(), getIntent().getStringExtra("ip"));
    }

    private Bitmap getSmallBitmap(Camera.Size previewSize, Bitmap bmp) {
        int dimension = 200;
        int smallWidth, smallHeight;
        if (previewSize.width > previewSize.height) {
            smallWidth = dimension;
            smallHeight = dimension * previewSize.height / previewSize.width;
```

```
        } else {
            smallHeight = dimension;
            smallWidth = dimension * previewSize.width / previewSize.height;
        }
        return Bitmap.createScaledBitmap(bmp, smallWidth, smallHeight, false);
    }

    private Bitmap rotateBitmap(Bitmap bmp, float orientation) {
        Matrix matrix = new Matrix();
        matrix.postRotate(orientation);
        return Bitmap.createBitmap(bmp, 0, 0, bmp.getWidth(), bmp.getHeight(),
matrix, false);
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        initializeCamera();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
int height) {
        configureCameraOrientation();
        camera.startPreview();
    }

    private void configureCameraOrientation() {
        Camera.Parameters parameters = camera.getParameters();
        Camera.CameraInfo info = new Camera.CameraInfo();
        Camera.getCameraInfo(currentCameraId, info);

        int rotation = getWindowManager().getDefaultDisplay().getRotation();
        int degrees = getDegreesFromRotation(rotation);

        int resultA = (currentCameraId ==
Camera.CameraInfo.CAMERA_FACING_FRONT)
                ? (360 + 360 - info.orientation - degrees) % 360
                : (info.orientation - degrees + 360) % 360;
        int resultB = (info.orientation + degrees) % 360;

        camera.setDisplayOrientation(resultA);
        parameters.setRotation(resultB);
        mCameraOrientation = resultB;
        camera.setParameters(parameters);
    }

    private int getDegreesFromRotation(int rotation) {
        switch (rotation) {
```

```
            case Surface.ROTATION_90: return 90;
            case Surface.ROTATION_180: return 180;
            case Surface.ROTATION_270: return 270;
            default: return 0;
        }
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        releaseCamera();
    }

    @Override
    public void handleReceive(final byte[] data) {
        runOnUiThread(() -> {
            Bitmap bitmap = BitmapFactory.decodeByteArray(data, 0,
data.length);
            imageView.setImageBitmap(bitmap);
        });
    }

    private void endCall() {
        if (IN_CALL) {
            call.endCall();
        }
        if (up != null) {
            up.stop();
        }
        if (upa != null) {
            upa.stop();
        }
    }

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

    public int[] decodeYUV420SP(byte[] yuv420sp, int width, int height) {
        final int frameSize = width * height;
        int[] rgb = new int[width * height];
        for (int j = 0, yp = 0; j < height; j++) {
            int uvp = frameSize + (j >> 1) * width, u = 0, v = 0;
            for (int i = 0; i < width; i++, yp++) {
                int y = (0xff & ((int) yuv420sp[yp])) - 16;
                if (y < 0) y = 0;
                if ((i & 1) == 0) {
```

```
                    v = (0xff & yuv420sp[uvp++]) - 128;
                    u = (0xff & yuv420sp[uvp++]) - 128;
                }
                int y1192 = 1192 * y;
                int r = (y1192 + 1634 * v);
                int g = (y1192 - 833 * v - 400 * u);
                int b = (y1192 + 2066 * u);
                rgb[yp] = 0xff000000 | ((r < 0 ? 0 : r > 262143 ? 262143 : r)
<< 6 & 0xff0000)
                        | ((g < 0 ? 0 : g > 262143 ? 262143 : g) >> 2 &
0xff00)
                        | ((b < 0 ? 0 : b > 262143 ? 262143 : b) >> 10 &
0xff);
            }
        }
        return rgb;
    }
}
```

Description: This Java class ActivityVideoCall facilitates video calls in an Android application using the device camera and UDP communication. The onCreate method sets up the activity and initializes the camera and audio call. The initializeViews method sets up the UI elements and their corresponding click listeners

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 chose functional testing as our testing methodology because of its efficiency and many advantages. Functional testing checks the functionality of an application without looking into its internal structures or workings. Specifically, functional unit testing was essential in our project. Unit testing, a type of functional testing, evaluates individual units of code to ensure they work as expected based on defined criteria.

5.2. Testing Environment

For our testing environment, we had two primary components: Raspberry Pi and the Androcom application running on Android devices. The Raspberry Pi served as the network side, making it ideal for creating and testing an ad hoc network setup. The Androcom application was tested on Android devices to simulate user interactions, such as setting up profiles, messaging, and making calls.

This environment was selected for several reasons. First, it allowed for a realistic simulation of real-world usage scenarios, helping to identify issues that might not be apparent in a purely virtual setup. Second, both the Raspberry Pi and Android devices are cost-effective, enabling extensive testing without significant financial strain. Overall, this environment facilitated thorough and efficient testing, ensuring the delivery of a reliable and well-functioning application.

5.3. Test Cases

Test Case 1: Verify if the Raspberry Pi is configured to set up an ad hoc network.

Table 5.1: Test Case 1

Date: 02 June 2024	
System: Raspberry Pi	
Objective: To ensure the Raspberry Pi can create a functional ad hoc network.	***Test ID:*** 1
Version: 1	***Test Type:*** Functional Testing
Expected Result: Raspberry Pi successfully establishes an ad hoc network.	
Actual Result: Raspberry Pi established the ad hoc network.	

The test case was generated based on the requirement to establish a wireless ad hoc network using a Raspberry Pi.

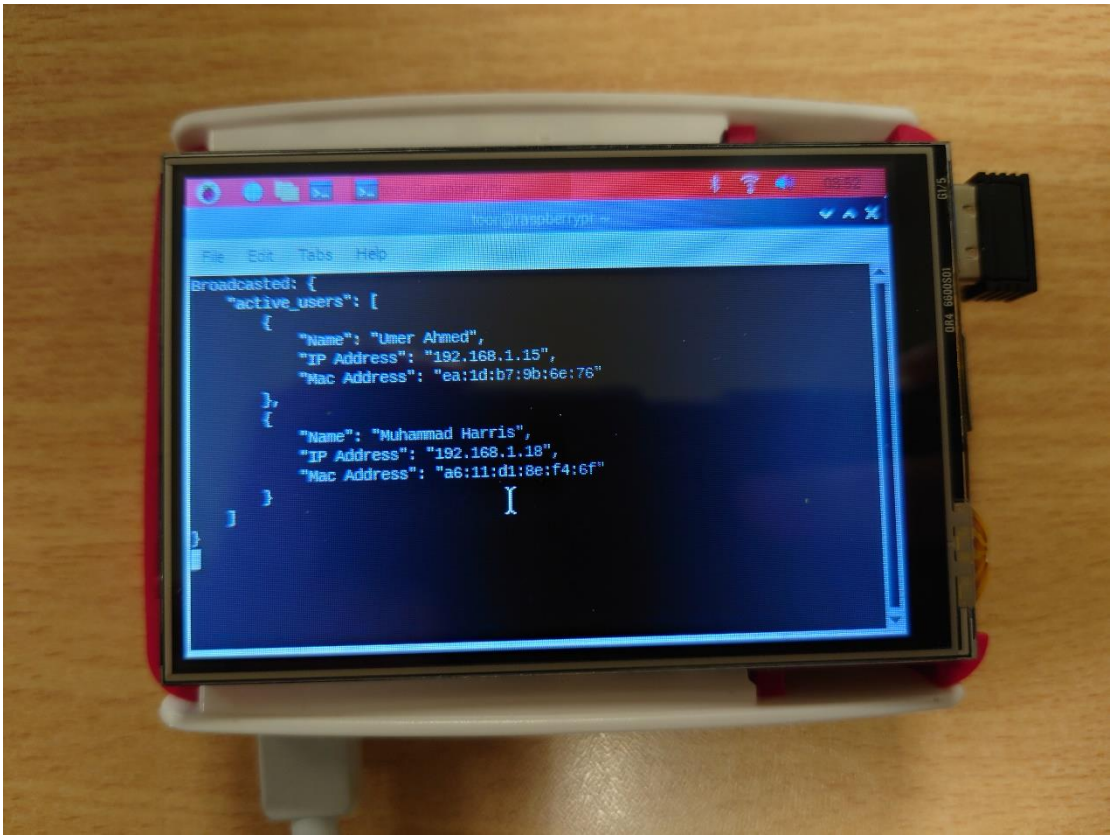


Figure 5.1: Raspberry P 3B+ working

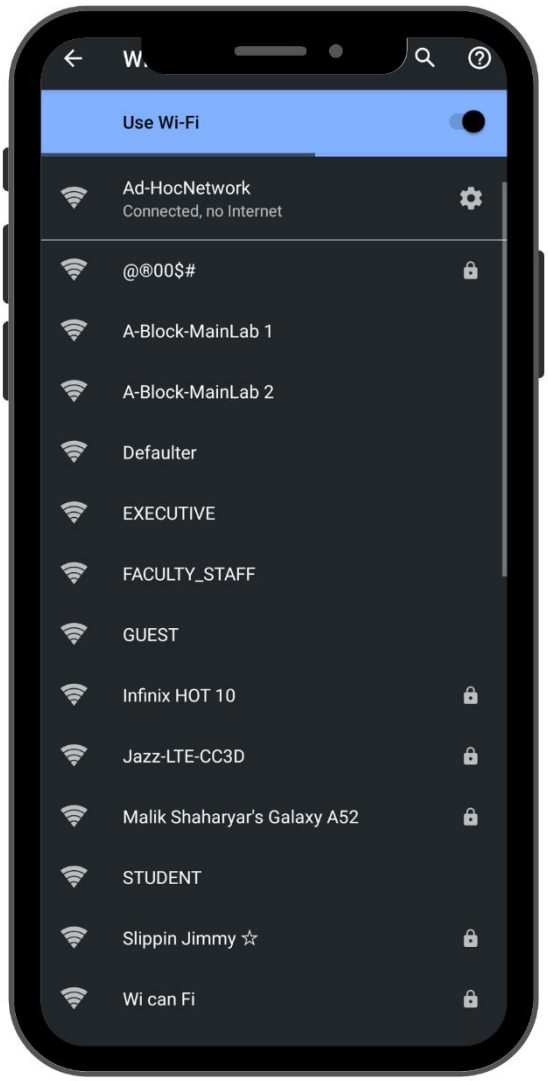


Figure 5.2: Ad hoc network in android device

After running the start.sh script on the Raspberry Pi, it successfully created an ad hoc Wi-Fi network named "Ad hoc network." This network functioned as expected and was readily visible in the Wi-Fi settings on an Android device. Upon connecting to the network, the Android device confirmed that it was configured correctly.

Test Case 2: Verify if a new user can set up their profile in the Android application.

Table 5.2: Test Case 2

Date: 05 June 2024	
System: Androcom Android Application	
Objective: To validate the user profile setup functionality.	***Test ID:*** 2
Version: 3	***Test Type:*** Functional Testing
Expected Result: User profile is created and saved successfully.	
Actual Result: User profile created and saved.	

The test case was generated to ensure user profiles can be created and updated in the application.

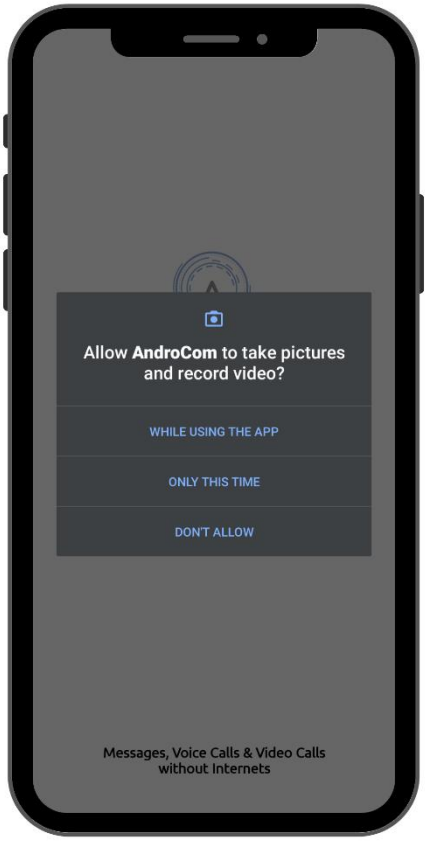


Figure 5.3: Granting picture and video access

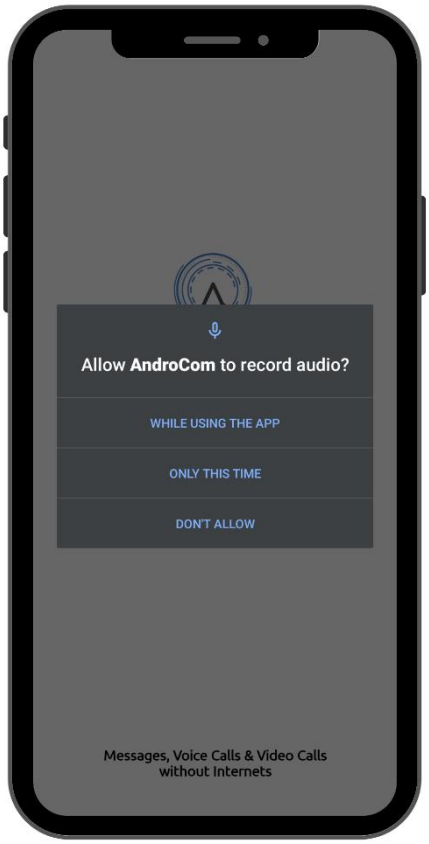


Figure 5.4: Granting audio access

Figure 5.5: Startup screen

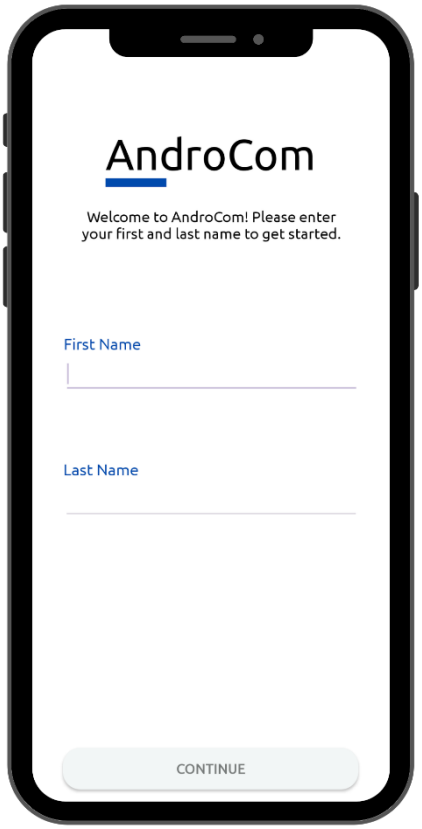


Figure 5.6: Empty setup form

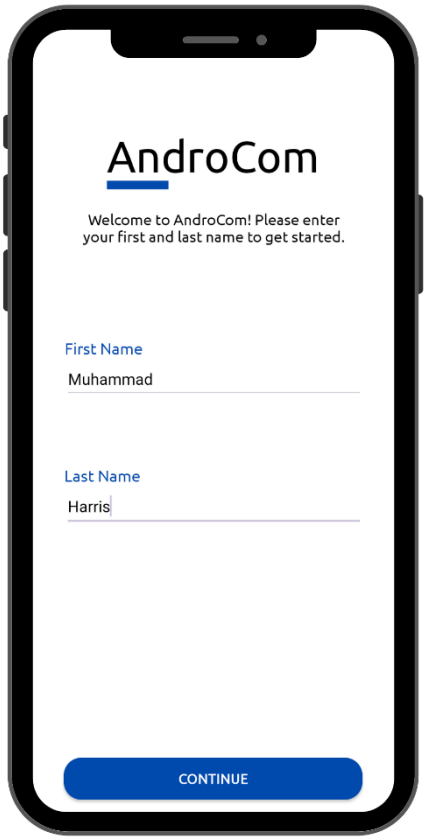


Figure 5.7: Filled startup form

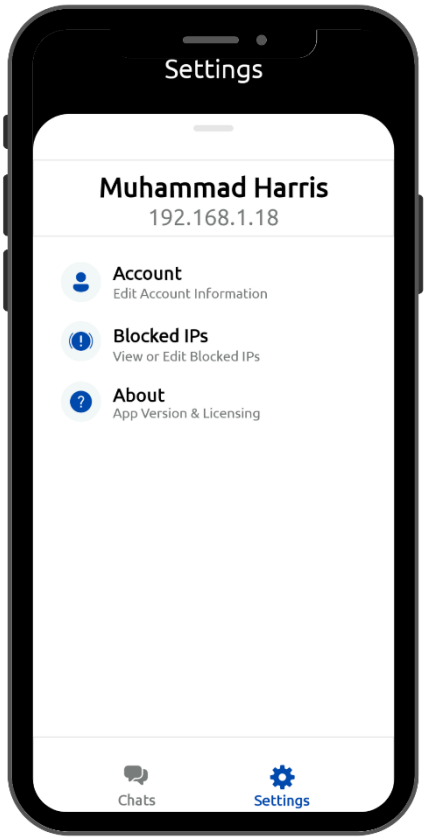


Figure 5.8: Username in settings

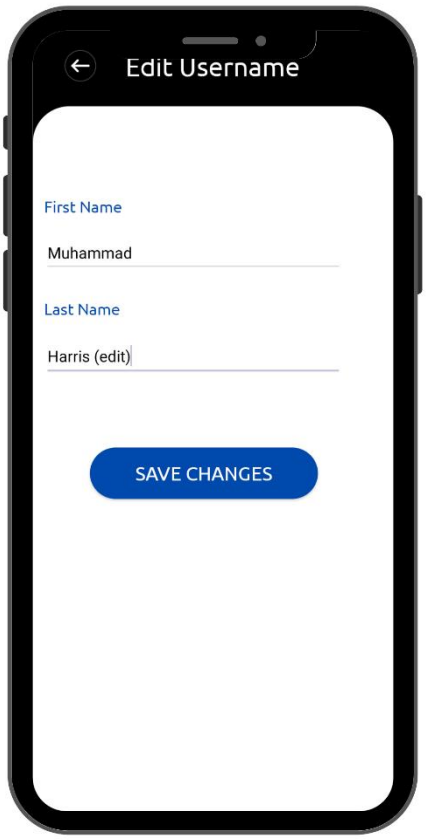


Figure 5.9: Editing username in settings

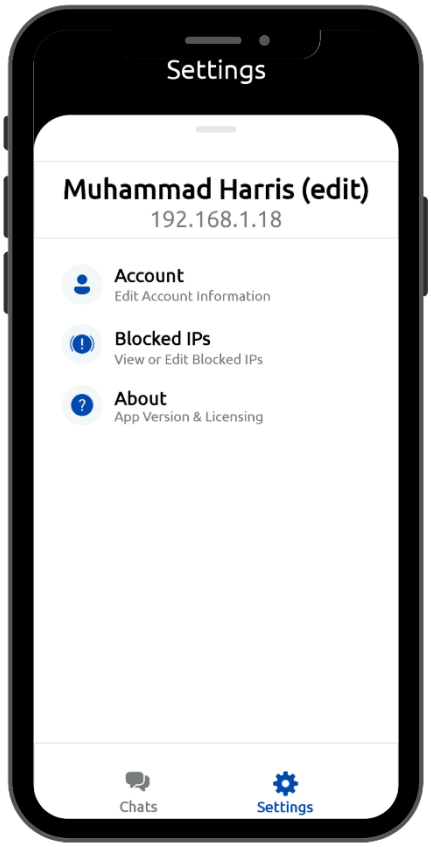


Figure 5.10: Edited username in settings

When a user opens the app for the first time, they are asked to grant two permissions: permission to record pictures and videos using the camera and permission to record audio from the microphone.

Once the user accepts both permissions, they are directed to the startup screen. Upon clicking "Get Started," they are presented with a form with two fields: First Name & Last Name. Upon entering their names, they can continue with the application.

The username can be viewed in settings. In the settings, there is also an option that allows users to edit their username if they wish to do so. We were successfully able to setup a username and later change it in settings.

Test Case 3: Broadcast active user list to Android application.

Table 5.3: Test Case 3

Date: 07 June 2024	
System: Androcom Android Application	
Objective: To ensure active users are broadcasted and displayed correctly.	***Test ID:*** 3
Version: 2	***Test Type:*** Functional Testing
Expected Result: Active user list is displayed in the application.	
Actual Result: Active user list displayed.	

The test case was generated based on the need to display active users in the application.

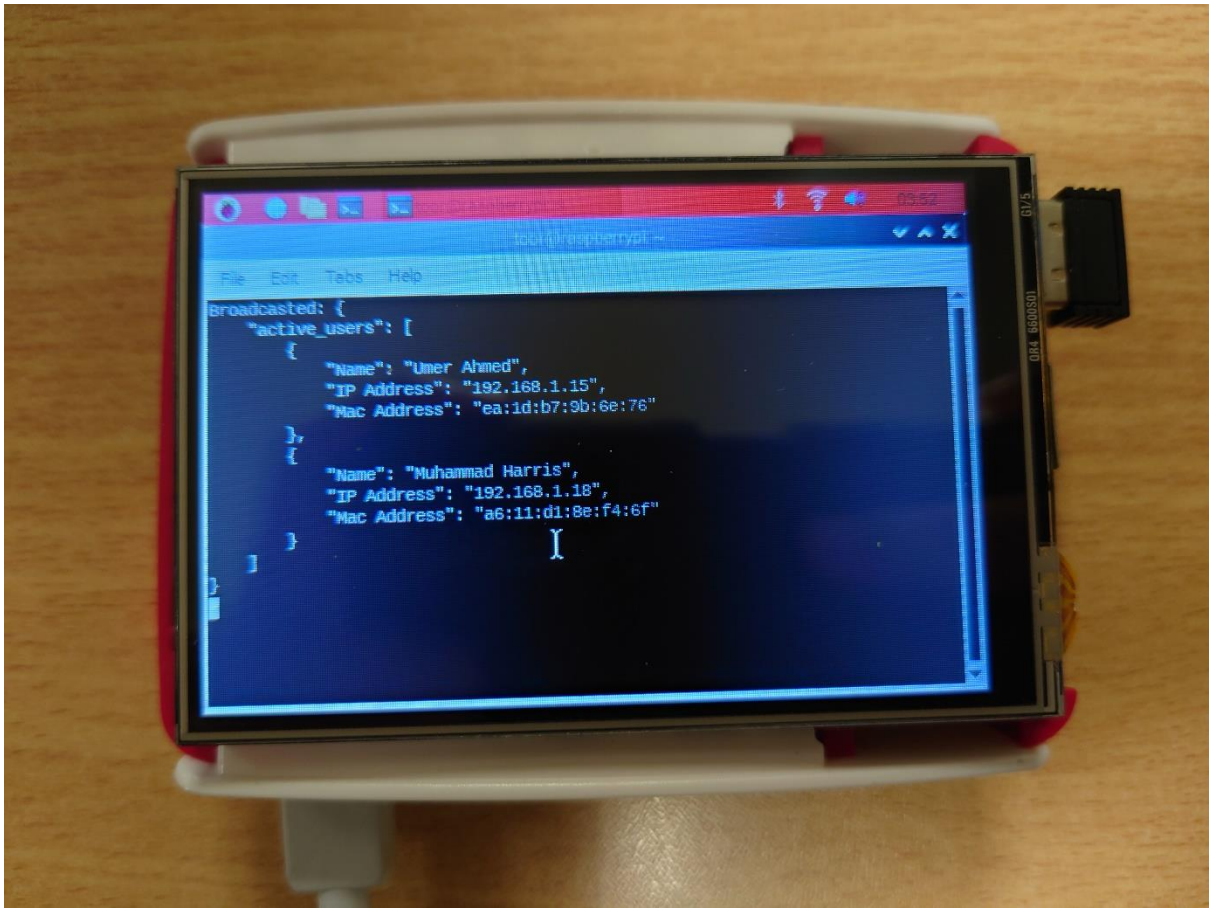


Figure 5.11: Raspberry Pi maintaining a list of active users

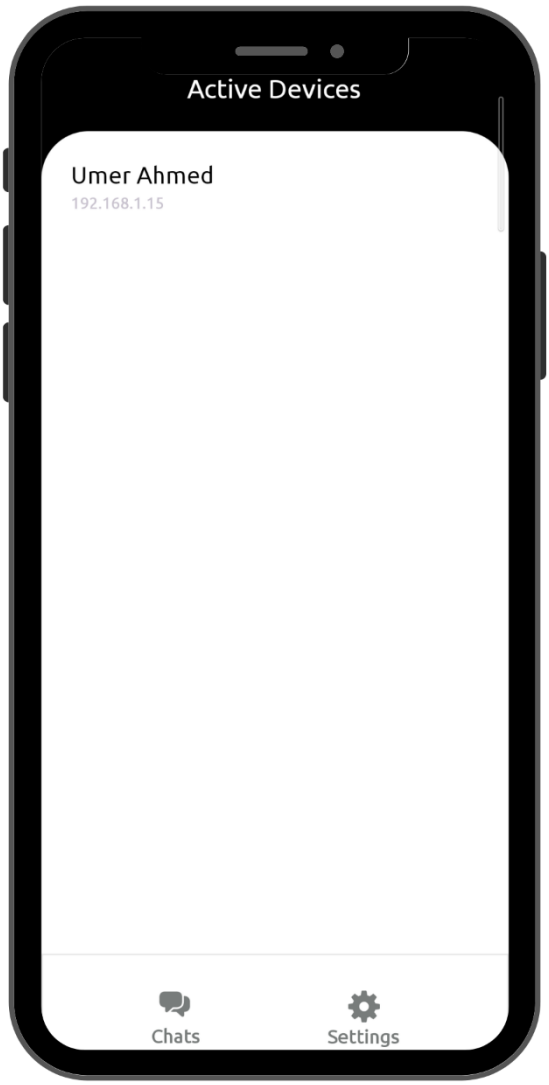


Figure 5.12: Active users list in android app

The Raspberry Pi successfully maintains a list of devices connected to its ad hoc network. This list includes information for each device, such as username, IP address, and MAC address. This information is broadcasted to all Android devices using a UDP socket, where it can be viewed and utilized successfully.

Test Case 4: Verify if a user can block or unblock other users.

Table 5.4: Test Case 4

Date: 10 June 2024	
System: Androcom Android Application	
Objective: To validate the block and unblock functionality.	***Test ID:*** 4
Version: 2	***Test Type:*** Functional Testing
Expected Result: The user can block and unblock other users.	
Actual Result: The user successfully blocked and unblocked other users.	

The test case was generated to ensure users have the functionality to block or unblock other users.

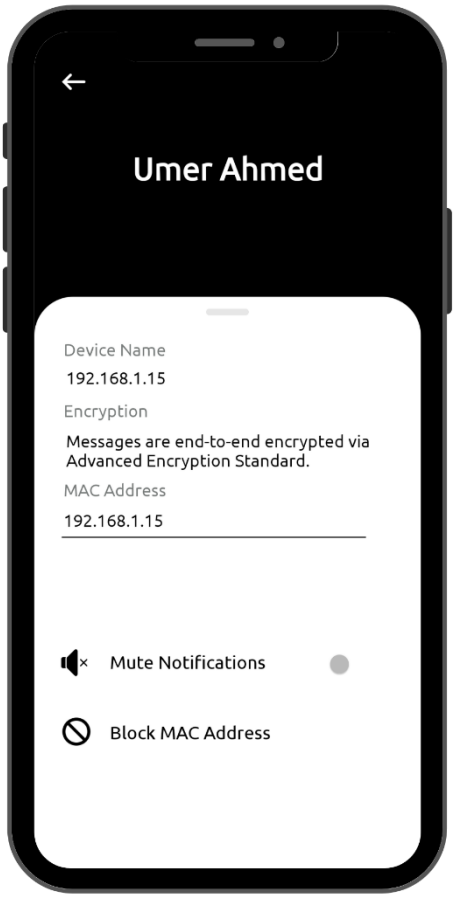


Figure 5.13: Blocking user in profile

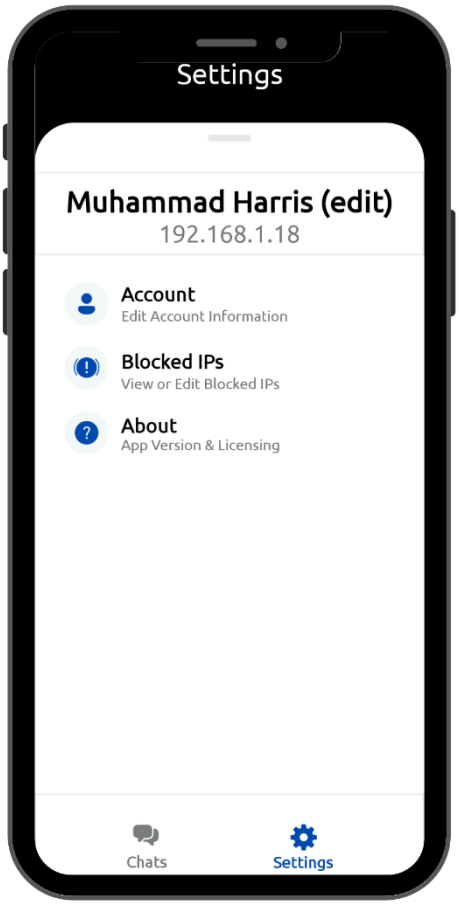


Figure 5.14: Block option in settings

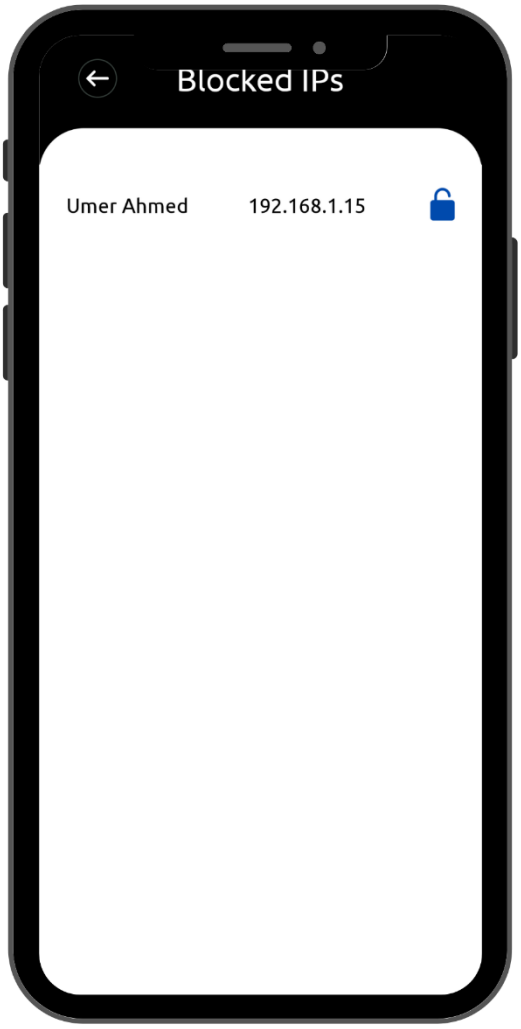


Figure 5.15: Block list

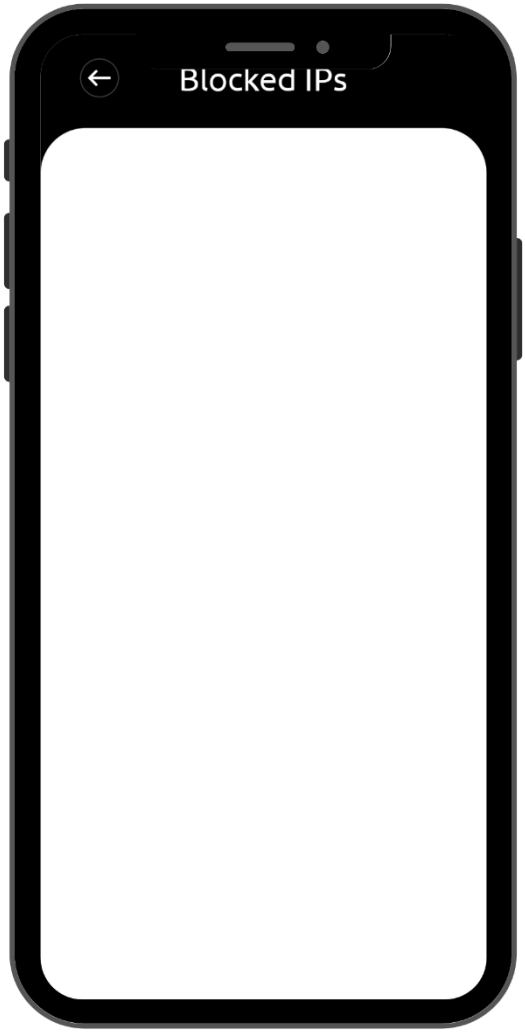


Figure 5.16: Block list (empty)

During testing, we were able to successfully block users from their user profiles. Once blocked, these users cannot send text messages or make voice calls to you. Blocking is implemented based on IP addresses. Users can unblock a contact by navigating to settings.

Test Case 5: Verify if a user can mute notifications.

Table 5.5: Test Case 5

Date: 12 June 2024	
System: Androcom Android Application	
Objective: To ensure the notification mute functionality works correctly.	***Test ID:*** 5
Version: 2	***Test Type:*** Functional Testing
Expected Result: Notifications are muted when the user chooses to do so.	
Actual Result: Notifications were muted successfully.	

The test case was generated to ensure users can mute notifications within the application.

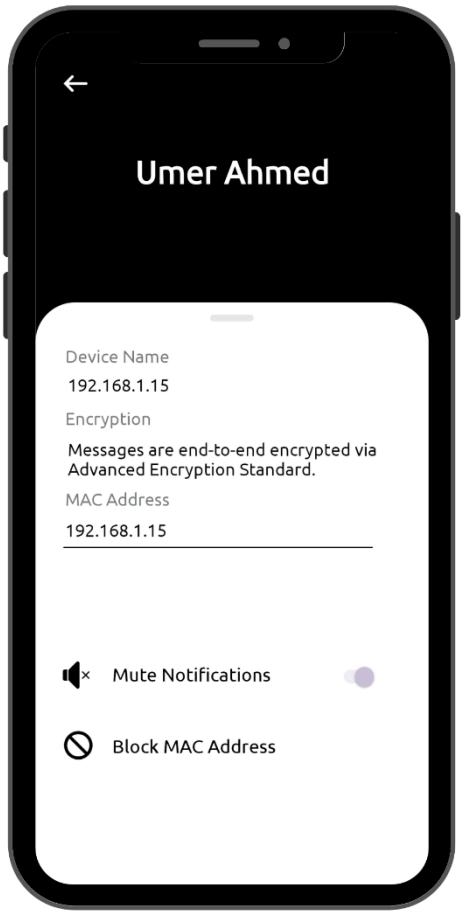


Figure 5.17: Profile (not muted)

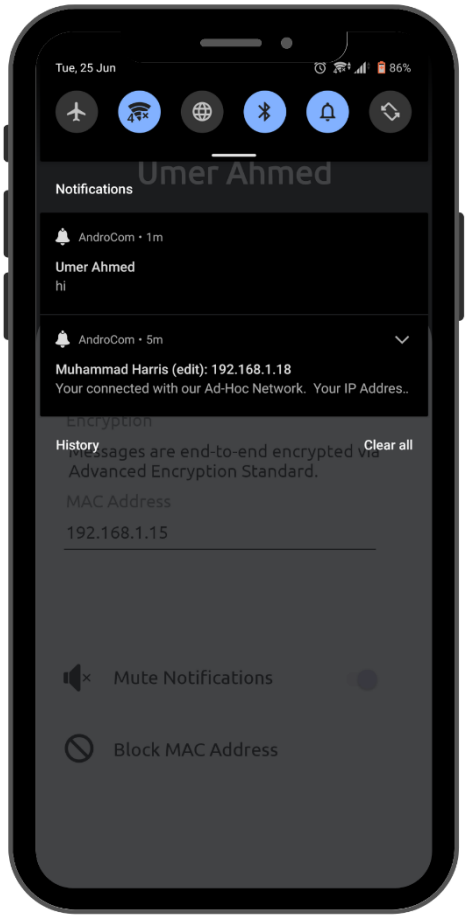


Figure 5.18: Notification Bar

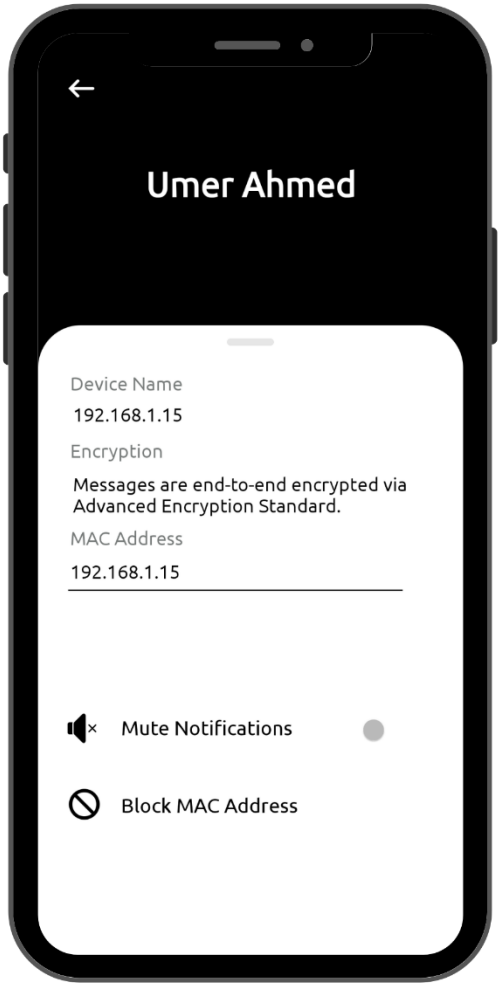


Figure 5.19: Profile (muted)

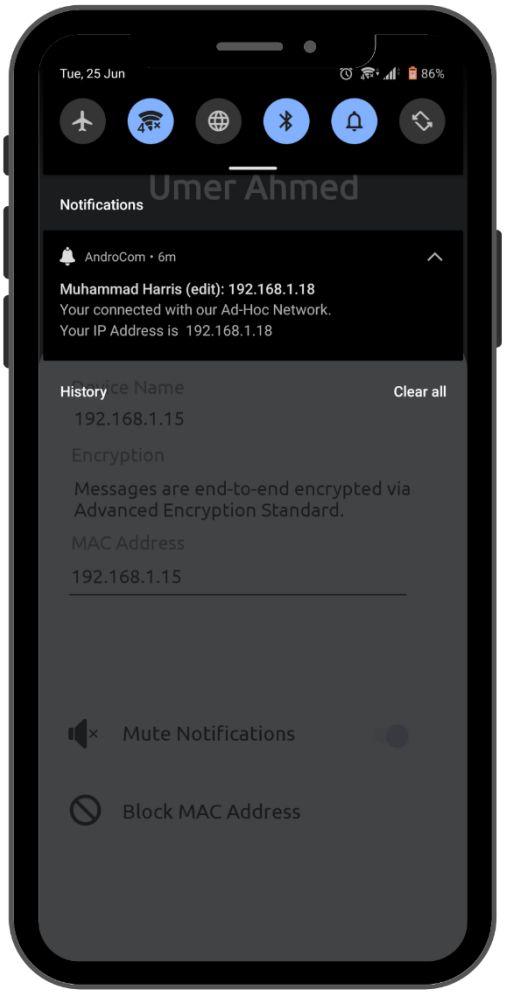


Figure 5.20: Notification Bar

First, we tested notifications without muting the contact. We received all notifications without any problems. Once we muted the contact from their contact profile, we received no notifications

Test Case 6: Verify if users can send and receive text messages.

Table 5.6: Test Case 6

Date: 12 June 2024	
System: Androcom Android Application	
Objective: To ensure the text messaging functionality works correctly.	***Test ID:*** 6
Version: 1	***Test Type:*** Functional Testing
Expected Result: Users can send and receive text messages.	
Actual Result: Users successfully sent and received text messages.	

The test case was generated to validate the text messaging functionality.

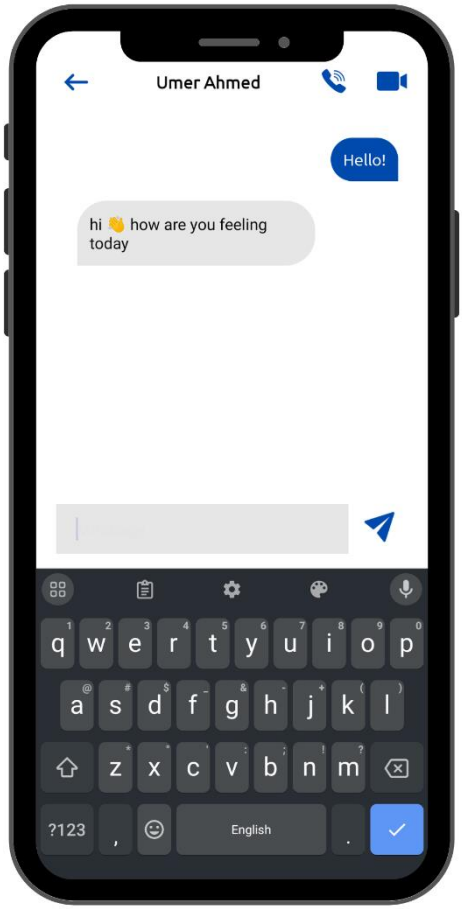


Figure 5.21: Sending and receiving text messages

While testing, text messages were successfully sent and received.

Test Case 7: Verify if users can make voice calls.

Table 5.7: Test Case 7

Date: 18 June 2024	
System: Androcom Android Application	
Objective: To validate the voice calling functionality.	***Test ID:*** 7
Version: 1	***Test Type:*** Functional Testing
Expected Result: Users can make and receive voice calls.	
Actual Result: Users successfully made and received voice calls.	

The test case was generated to ensure users can initiate and receive voice calls.

Figure 5.22: Incoming call

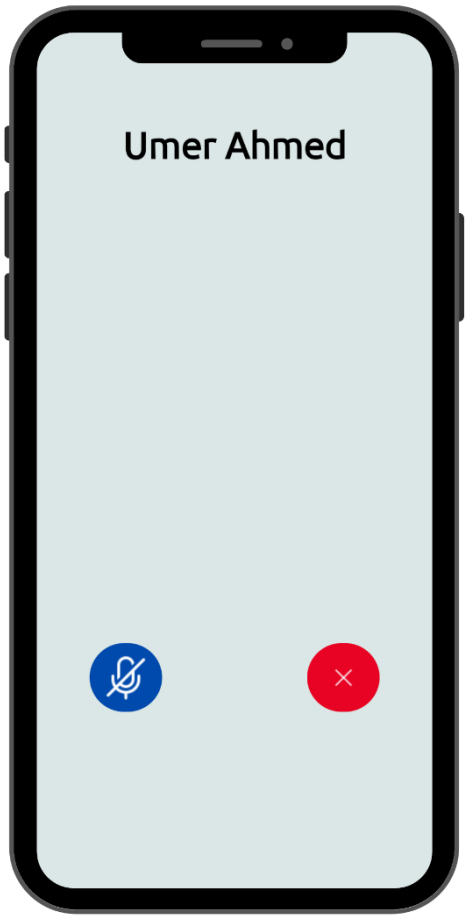


Figure 5.23: Received Call

Voice calls were successfully received and made.

Test Case 8: Verify if users can make video calls.

Table 5.8: Test Case 8

Date: 21 June 2024	
System: Androcom Android Application	
Objective: To validate the video calling functionality.	***Test ID:*** 8
Version: 5	***Test Type:*** Functional Testing
Expected Result: Users can make and receive video calls.	
Actual Result: Users successfully made and received video calls.	

The test case was generated to ensure users can initiate and receive video calls.

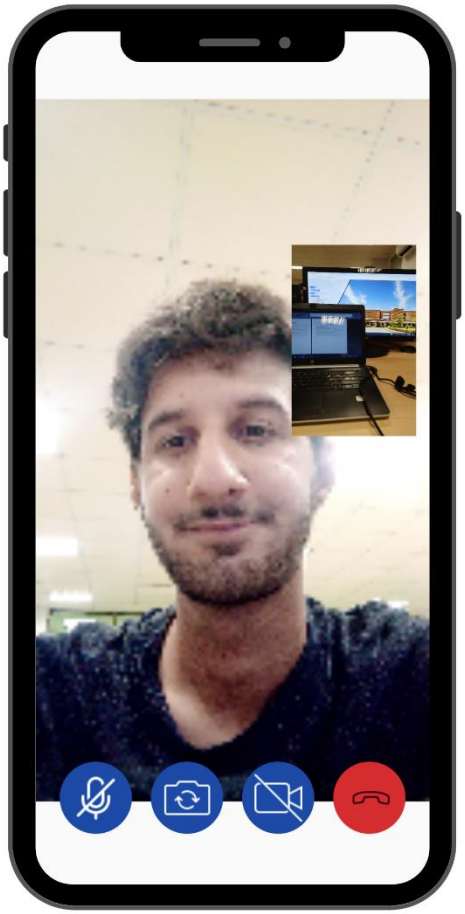


Figure 5.24: Video call

Video calls were successful. However there were some distortions due to network.

Test Case 9: Mute mic or disable camera within a call.

Table 5.9: Test Case 9

Date: 21 June 2024	
System: Androcom Android Application	
Objective: To ensure the functionality of muting the microphone or disabling the camera during calls.	***Test ID:*** 9
Version: 5	***Test Type:*** Functional Testing
Expected Result: Users can mute the microphone or disable the camera during a call.	
Actual Result: Users successfully muted the microphone and disabled the camera during a call.	

The test case was generated to ensure users have control over their microphone and camera during calls.

When in a video or voice call, all the buttons functioned correctly. In voice calls, the mute microphone button successfully mutes and unmutes the microphone. In video calls, the mute button works the same way. There is another button to switch the camera between the front and back. This switch camera button also functioned correctly. During video calls, there is another button for disabling the microphone. Clicking this button stops the video, but it freezes on the last frame instead of displaying a blank picture.

Chapter 6

Software Deployment

In this chapter, we will explore how a user can run an Androcom application on their Android device. We will also learn how a user can set up their Raspberry Pi to create an ad hoc network.

6.1. Installation / Deployment Process Description

To get Androcom up and running smoothly, a user will need a Raspberry Pi 3B+ to create an ad hoc network. The network can also be created with other Raspberry Pi models or any other microcomputer running a Linux operating system and equipped with a built-in Wi-Fi card.

For the Android device, the Androcom Android application is compatible with any device running Android version 8 or higher.

6.1.1 Network Configuration

When configuring the Raspberry Pi, it must be connected to a monitor and keyboard. Alternatively, a touchscreen can be used, as is the case here.

Figure 6.1: Booting Raspberry Pi

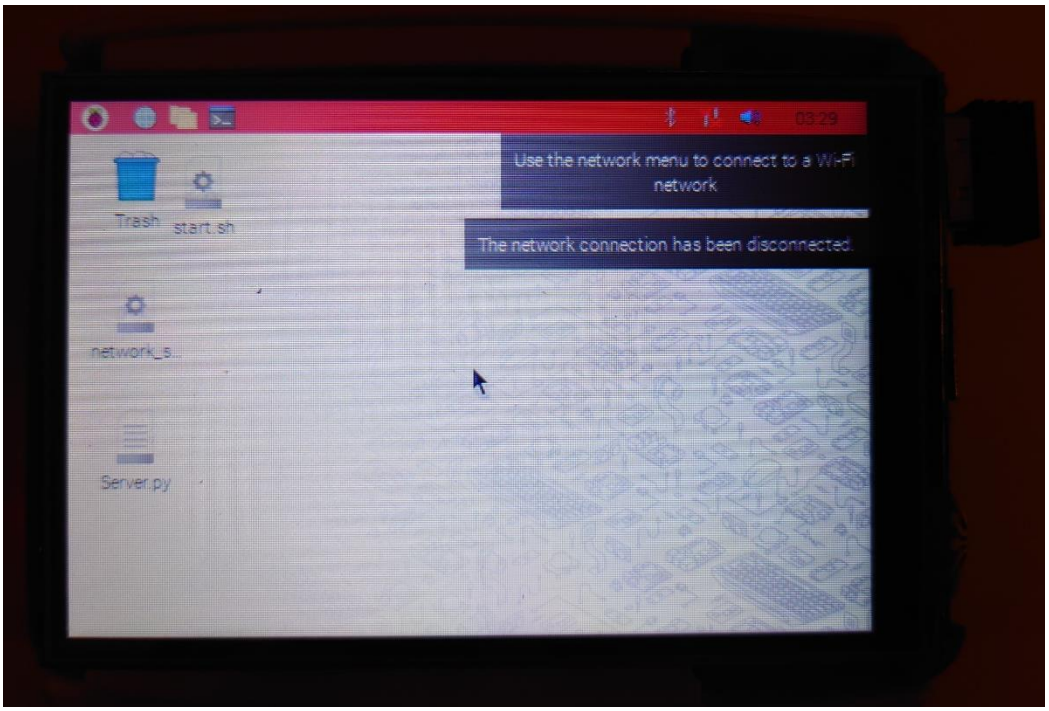


Figure 6.2: Raspberry Pi Desktop

Once the Raspberry Pi is booted up and ready, the user will need three scripts on their system. These files, server.py, network.sh, and start.sh, are available at https://github.com/imharris24/androcom.

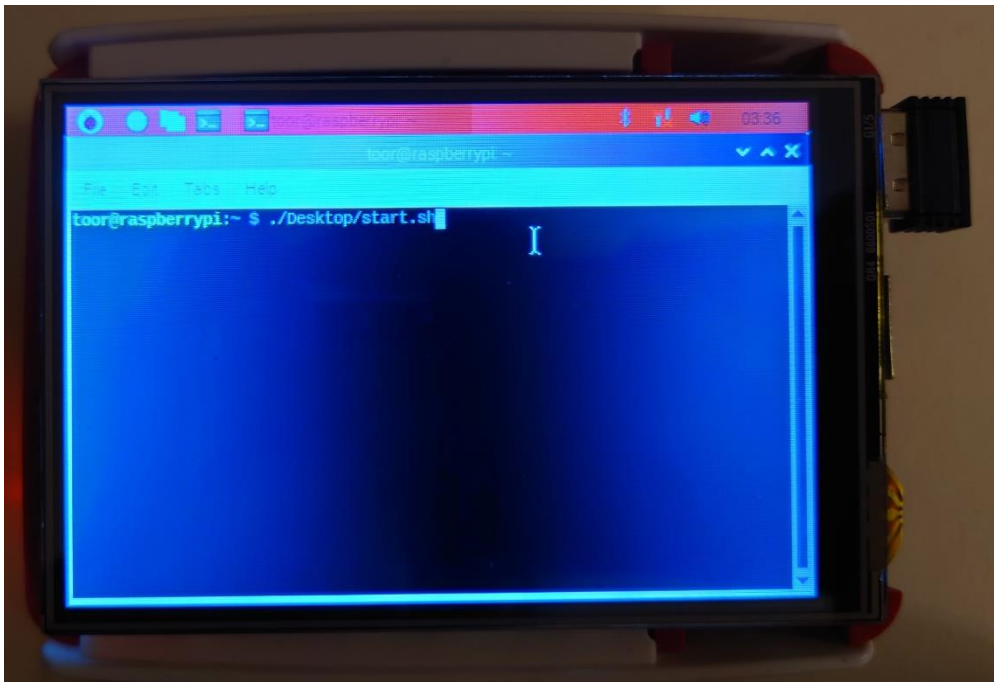


Figure 6.3: Terminal in Raspberry

By running *./start.sh* in the terminal, the server will start on the Raspberry Pi and an ad hoc network will be created.

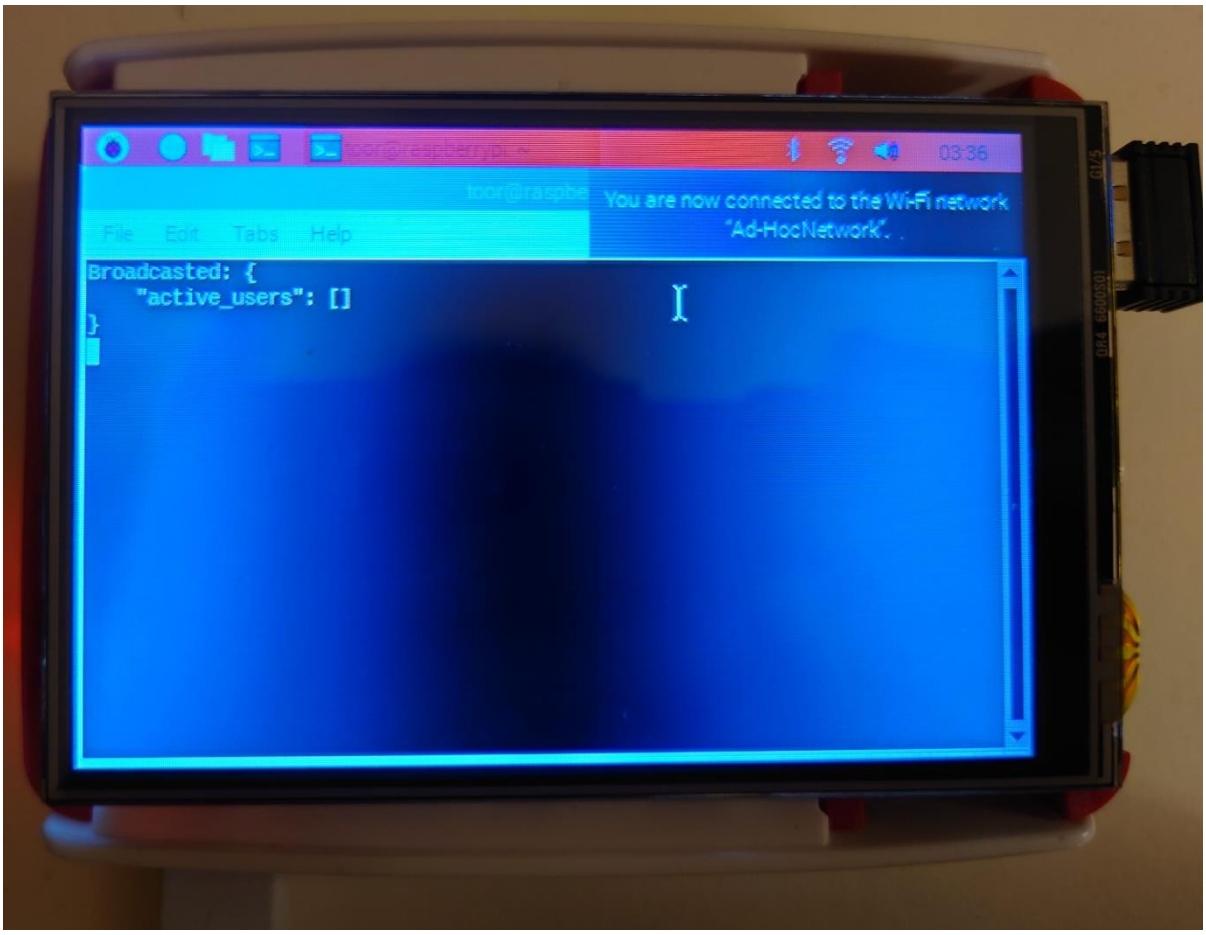


Figure 6.4: Running server in Raspberry

6.1.2 App Installation

Users can download the Androcom application from the Google Play Store. Once installed, users should connect to the ad hoc network through Wi-Fi settings on their devices. After connecting, they can start using Androcom.

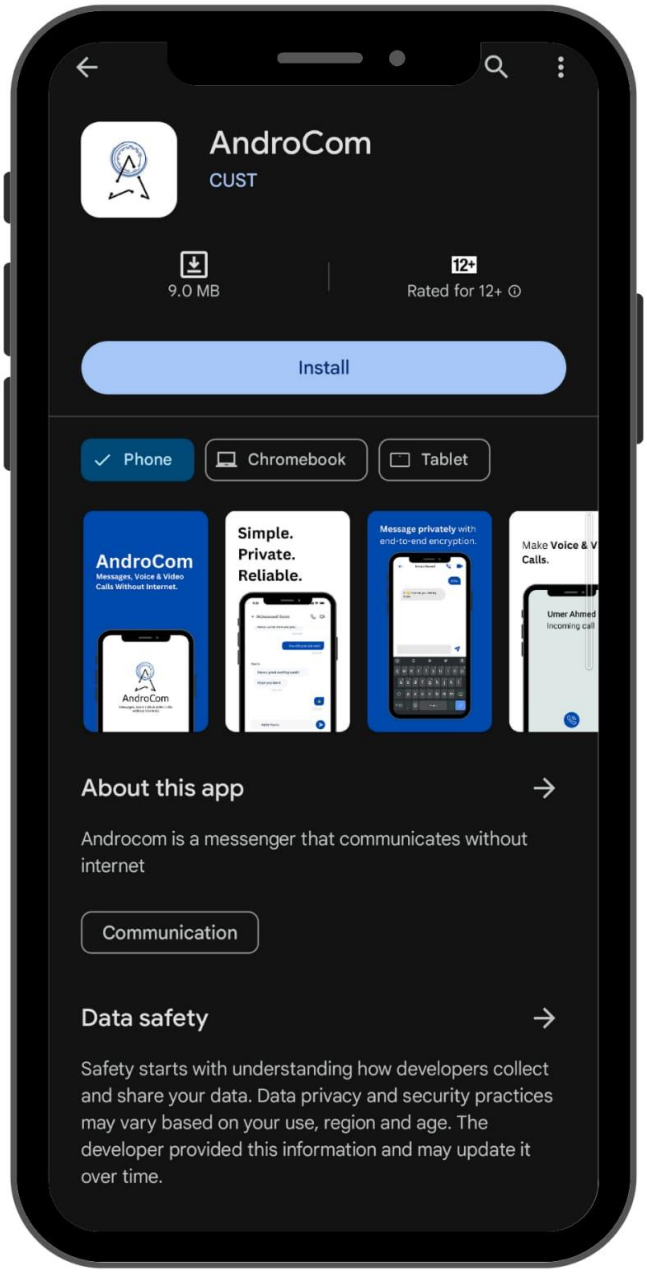


Figure 6.5: Androcom on Google Playstore

Chapter 7

Conclusion

In conclusion, our project, "AndroCom - Infrastructure-less Communication," achieved its goals with resounding success. We established an ad hoc network and a corresponding Android application that can be effectively deployed during emergencies.

The Android app allowed its users to exchange end-to-end encrypted text messages and make voice and video calls entirely independent of the internet or any existing network infrastructure. AndroCom stands out as a pioneering achievement in this regard, eliminating reliance on pre-established infrastructure for communication.

7.1. Future Work

Our successful development of AndroCom lays a strong foundation for further advancements in infrastructure-less communication. Here, we explore potential areas for future development:

i. Currently, AndroCom works on an individual ad hoc network. Future iterations could explore the creation of a mesh network, where multiple ad hoc networks interconnect. This would significantly expand the overall network range and user reach, enabling communication across a wider geographical area.

ii. While the current Raspberry Pi and Linux OS setup is functional, a dedicated hardware and firmware solution could optimize performance, efficiency and reduce cost. Developing custom hardware specifically designed for our ad hoc network could lead to increased network stability, lower power consumption, and potentially a more compact device form factor.

iii. The Android application holds immense potential for feature expansion. Integrating functionalities such as:
    - Utilizing the existing ad hoc network, users could transfer files using the File Transfer Protocol (FTP).
    - The ability to create groups would facilitate communication among multiple users within the network through group chat.
    - Sharing a device's screen within the ad hoc network would be a valuable addition for collaborative purposes.
    - Implementing disappearing messages would cater to users seeking an extra layer of privacy within their communications.

- Adding fingerprint or face ID recognition would improve in-app security, safeguarding user data and privacy.

iv. As user adoption grows, the ad hoc network becomes more prone to Denial-of-Service (DoS) attacks. Future development should prioritize implementing robust security measures to mitigate such threats and ensure network stability.

v. Expanding AndroCom beyond the Android platform would significantly increase its accessibility. Developing the application for iOS, Windows, and Linux operating systems would broaden its user base and cater to a wider range of devices.

Chapter 8

Project Evaluation

This chapter includes the examiners evaluation report, including the points to be revised/included along with the selected requirements in the next iteration.

8.1. Project Evaluation Report

Examiner Name:	
Sr. No.	**Suggestion**
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	

Other Comments (If any):

_____

_____

_____

_____

Signature

References

Books

[1] J. F. Kurose and K. W. Ross, Computer Networks: A Top-Down Approach, 7th ed. Pearson, 2016.

[2] J. Daemen and V. Rijmen, The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, 2002.

Documentation

[3] Android Developers, "Overview," [Online]. Available: https://developer.android.com/docs. [Accessed: 10-Nov-2023].

[4] Oracle, "Java Documentation," [Online]. Available: https://docs.oracle.com/en/java/. [Accessed: 1-Feb-2024].

[5] SQLite, "SQLite Documentation," [Online]. Available: https://www.sqlite.org/docs.html. [Accessed: 5-Feb-2024].

[6] Python Software Foundation, "Python Documentation," [Online]. Available: https://docs.python.org/3/. [Accessed: 15-Sep-2024].

[7] Raspberry Pi Foundation, "Raspberry Pi Documentation," [Online]. Available: https://www.raspberrypi.org/documentation/. [Accessed: 15-May-2024].

Research Papers

[8] W. Drira and A. Masmoudi, "Secure communication in mobile ad hoc networks based on a new key management scheme," Int. J. Commun. Syst., vol. 28, no. 12, pp. 1836-1852, 2015, doi: 10.1002/dac.2773.

[9] S. H. H. Jazi and A. Sadeghi-Niaraki, "Designing an encrypted communication system for mobile ad-hoc networks," Telecommun. Syst., vol. 70, no. 3, pp. 393-405, 2019, doi: 10.1007/s11235-018-0491-6.

[10] S. Khalid, M. Tahir, and M. U. Farooq, "Ad hoc network formation and data dissemination using Raspberry Pi: A case study," IEEE Access, vol. 6, pp. 60790-60799, 2018, doi: 10.1109/ACCESS.2018.2876063.

[11] Y. Lu and Q. Qu, "End-to-end encryption for VoIP and messaging," J. Netw. Comput. Appl., vol. 62, pp. 168-180, 2015, doi: 10.1016/j.jnca.2015.03.016.

[12] A. Vancea and J. L. Dugelay, "Peer-to-peer systems for secure multimedia communications: Challenges and opportunities," IEEE Trans. Multimedia, vol. 18, no. 12, pp. 2355-2369, 2016, doi: 10.1109/TMM.2016.2619922.