T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

LOCAL NETWORK CHAT APP

GÖKBEY GAZI KESKİN

SUPERVISOR
DR. GÖKHAN KAYA

GEBZE
2023

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

LOCAL NETWORK CHAT APP

GÖKBEY GAZI KESKİN

SUPERVISOR
DR. GÖKHAN KAYA

2023
GEBZE

GRADUATION PROJECT
JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 22/06/2023 by the following jury.

JURY

Member
(Supervisor) : Dr. Gökhan KAYA

Member : Prof. Dr. İbrahim SOĞUKPINAR

ABSTRACT

The aim of this project is to develop a Cross-Platform Local Network Chat App capable of functioning in areas with limited or no internet connection, providing a secure means of communication. The inspiration behind this project stems from the devastating earthquakes that occurred in our country in February 2023, where several regions experienced a complete loss of reception and internet access. In such critical situations, having an app that enables communication within the local network can prove invaluable. Individuals could reach out to their families and neighbors, even when trapped under rubble, by simply establishing a LAN access point (hotspot) using their mobile phones or laptops. This paper explores the development of the app and its potential benefits.

Keywords: LAN, Chat, Security, Communication.

ÖZET

Bu projenin amacı; sınırlı internet bağlantısı olan veya hiç olmayan bölgelerde kullanılmak üzere, güvenli iletişim sağlayan bir Çoklu Platform (Cross-Platform) Yerel Ağ mesajlaşma uygulaması geliştirmektir. Projenin ilham kaynağı, ülkemizde 2023 yılının şubat ayında yaşanan büyük depremlerdir. Bu depremlerde bazı bölgelerde telefon sinyali tamamen kesilmiş, pek çok bölgede ise buna ek olarak internet bağlantısı da sağlanamamıştır. Bu tür kritik durumlarda, yerel ağ içinde iletişimi mümkün kılan bir uygulamaya sahip olmak son derece değerli olabilir. Bireyler, mobil telefon veya dizüstü bilgisayarlarını kullanarak sadece bir LAN erişim noktası (hotspot) oluşturarak, enkaz altında dahi olsalar, ailelerine ve komşularına ulaşabilirler. Bu makale, uygulamanın geliştirilmesini ve potansiyel faydalarını incelemektedir.

Anahtar Kelimeler: LAN, Mesajlaşma, Güvenlik, İletişim.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project supervisor, Dr. Gökhan Kaya, for his exceptional understanding, availability, and effective communication throughout the duration of this project. Despite his busy schedule, he consistently provided valuable guidance and support, which greatly contributed to the success of this project.

I would also like to extend my thanks to Prof. Dr. İbrahim Soğukpınar for his kind comments and constructive criticism, which helped to refine and improve the project.

Furthermore, I am deeply grateful to my friends who have been by my side throughout my four years of college: Kahraman Arda Kılıç, Baran Solmaz, Burak Yıldırım, Fatih Doğaç, Rian Ryzhov, Serdil Anıl Ünlü, Ahmet Tuğkan Ayhan, and Yunus Emre Öztürk.

Gökbey Gazi KESKİN

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
LAN	:	Local Area Network
AES	:	Advanced Encryption Standart
ms	:	Milliseconds

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In today's interconnected world, reliable communication is essential, regardless of the circumstances. However, there are situations where traditional communication channels fall short, such as remote areas with no internet access, disaster scenarios[4] or situations where secure communication is essential and you don't want to send your in-building messages through internet. This project aims to address these challenges by developing a versatile Cross-Platform Local Network Chat App capable of functioning in various contexts.

The Local Network Chat App provides an encrypted and reliable means of communication within a local network environment. By using LAN (Local Area Network) technology, users can establish communication links even in areas where internet access is unavailable. This makes the app ideal for remote locations, where individuals can establish a local network using their mobile phones or laptops, enabling wireless communication with others within the range.

The App eliminates the need for a separate server application. When a user opens the app, it initiates a search for an available access point within the local network. If an access point is found, the user connects to it, enabling immediate communication with other connected devices. However, in cases where no access point is initially available, the app takes the initiative and creates a new access point, allowing other users within range to connect to it. This dynamic access point feature ensures that communication remains uninterrupted, even if the original access point device logs out or loses connection unexpectedly. In such cases, the access point is automatically transferred to another online device, guaranteeing continuous communication availability within the local network. This decentralized approach enhances the app's reliability and resilience, making it well-suited for various scenarios.

The App also incorporates a unique "Trusted Devices" feature to enhance control over chat participants. When a new user sends a connection request for the first time, the current host device receives a prompt to either accept or decline the request. If accepted, the new device is added to the list of trusted devices, allowing it to join the session. On the other hand, declined devices are added to the "Banned Devices". This information regarding trusted and banned devices is automatically synchronized among all logged-in clients. Therefore, if a host device has previously rejected a user, that user cannot join any session hosted by other devices that are aware of the previous ban, unless the current host[1] "unbans" them via the settings screen. The Trusted Devices feature

[1]Keywords: "Access Point", "Host Device" and, "Server" are used interchangeably through the paper.

offers users the ability to manage and control who can participate in their sessions, ensuring a secure and controlled communication environment.

Overall, the Cross-Platform Local Network Chat App provides a secure and reliable means of communication within a local network with features like private chat, general chat, and image sharing.

2. UI/UX

Pages and interactions of the app are explained in this chapter.

2.1. Login Page

In the login page, users click on connect button (Figure 2.1) and type their names (Figure 2.2).

Figure 2.1: Login Page

Figure 2.2: Login Page after Clicking Connect.

2.1.1. Trusted Device Prompts

After login, the host device might reject the user (Figure 2.3) and add the user to the list of banned devices. In that case, rejected message is prompted to that user (Figure 2.5).

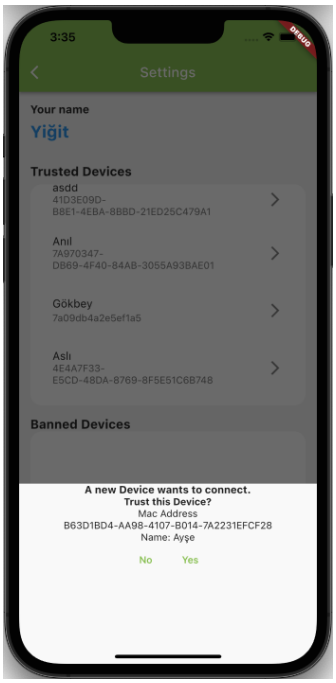

Figure 2.3: Connection Request (Server)

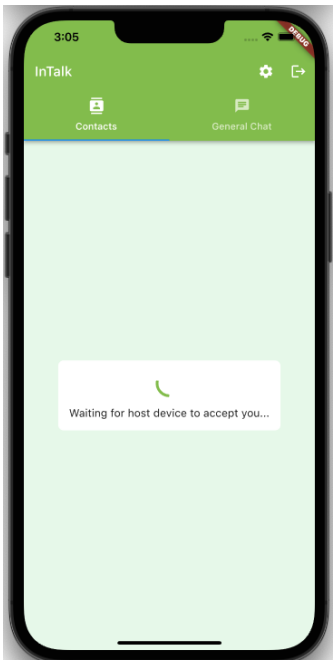

Figure 2.4: Waiting for response (Client)

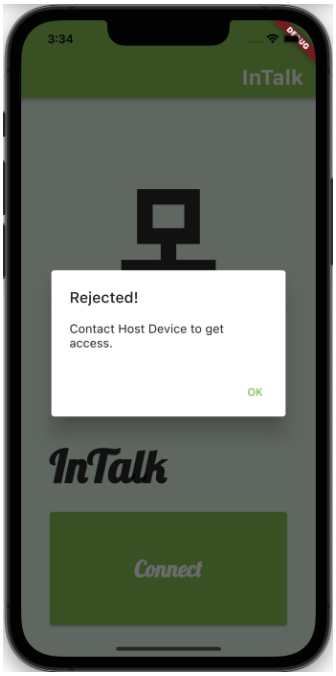

Figure 2.5: Connection Rejected (Client)

2.2. Home Page

The home page offers several functionalities. Upon accessing the home page, users can view a list of online users, enabling them to select and engage in individual chats. Additionally, there is a General Chat tab where users can participate in group

conversations with everyone connected to the network simultaneously. The home page also provides convenient access to the settings screen, allowing users to customize their preferences, and a logout option for a smooth session termination. Closing the app without logging out is also handled as discussed in the subsection 3.2.1 and the logout part of the subsection 3.3.3.

Notably, if the user is the current owner of the access point, the home page displays the label ”(HOST)” on the top left corner, indicating their status as the host device responsible for managing the network connection. This label helps users identify their role within the network.

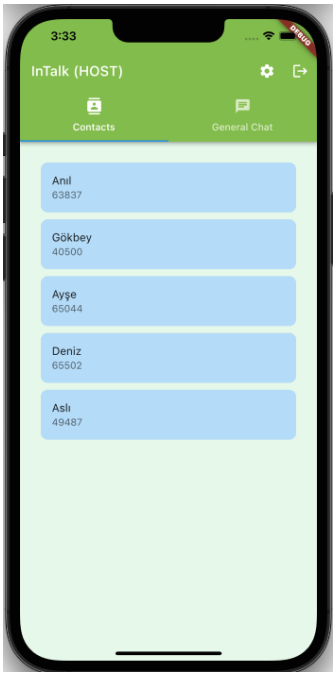

Figure 2.6: Contacts Tab.

Figure 2.7: General Chat Tab.

2.3. Chat Page

The Cross-Platform Local Network Chat App features two types of chat pages: the general chat (Figure 2.7) and private chat (Figure 2.8). While both types of chat pages share the same underlying technical framework, they serve distinct purposes as implied by their names.

Figure 2.8: Private Chat

When a user logs out, everyone currently chatting with that user are redirected to home page and prompted that the user they were chatting with just logged out.

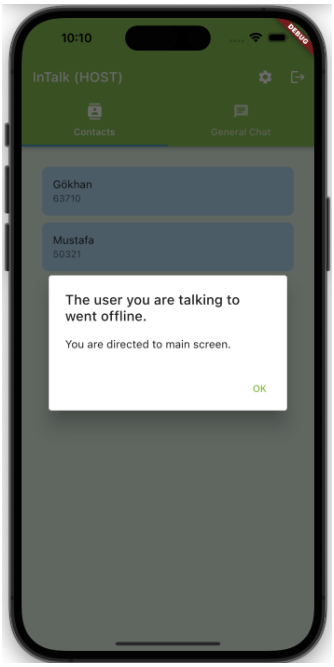

Figure 2.9: User logged out

2.4. Settings Page

Settings page provides the functionality to modify user information, including the ability to change the user's name, view the list of trusted and banned devices, and unban or untrust devices.

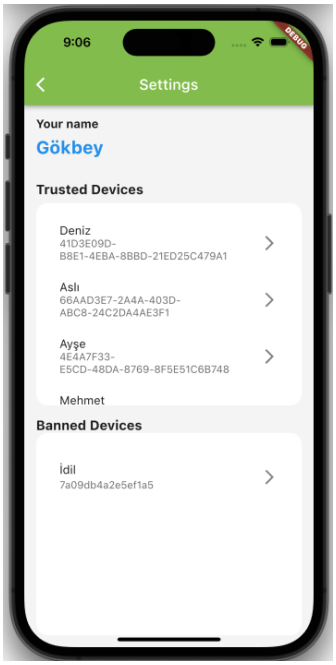

Figure 2.10: Settings Screen.

2.4.1. Changing Name

Users can click on their name to change it. This change is immediately broadcasted to all clients.

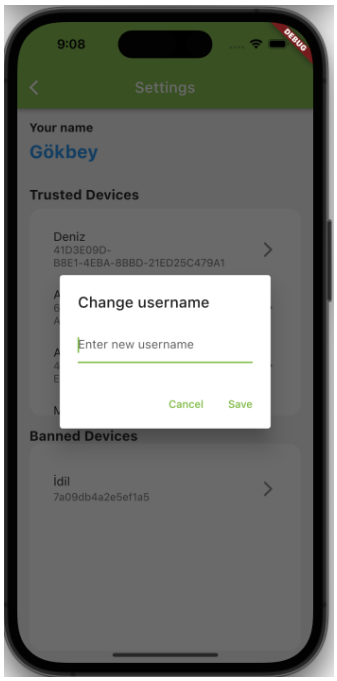

Figure 2.11: Change Name.

2.4.2. Forget & Unban Users

Users have the ability to remove devices from their trusted devices list, effectively forgetting them, or to unban devices from their session. If the user is the host, these changes are automatically broadcasted to all clients connected to the network. Therefore, if a device is banned by a user, it will remain banned for all future sessions initiated by currently online clients. However, it's important to note that each client has the autonomy to manage their own trusted and banned devices lists, allowing them to customize their session participants according to their preferences.

After customizing the device lists, users are prompted with an option to undo the change within a three-second timeframe. The prompt is displayed at the bottom of the screen, providing the opportunity to revert any accidental or unintended deletions and restore the previous device entry.

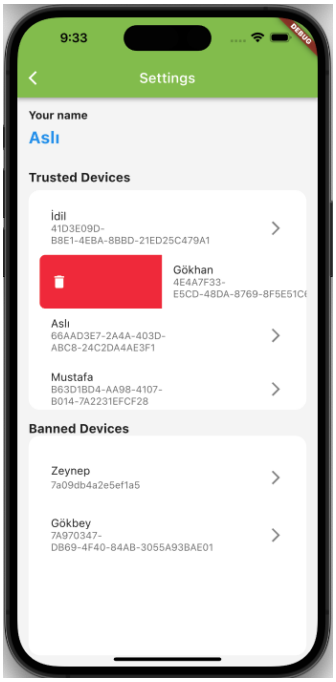

Figure 2.12: Forget Feature.

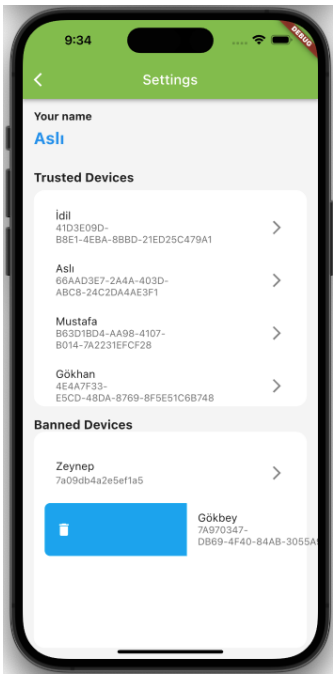

Figure 2.13: Unban Feature.

Lastly, when a user tries to login without being connected to a Local Area Network (LAN) or lose connection with the LAN, they are redirected to login screen and prompted that they lost connection.

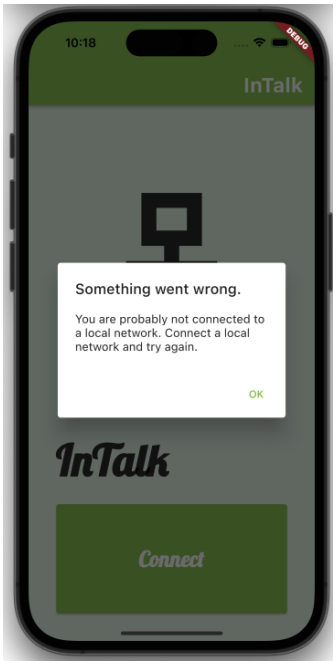

Figure 2.14: Something went Wrong.

2.5. UI Transition Diagram

All the transitions the User Interface has are visualized in (Figure 2.15).

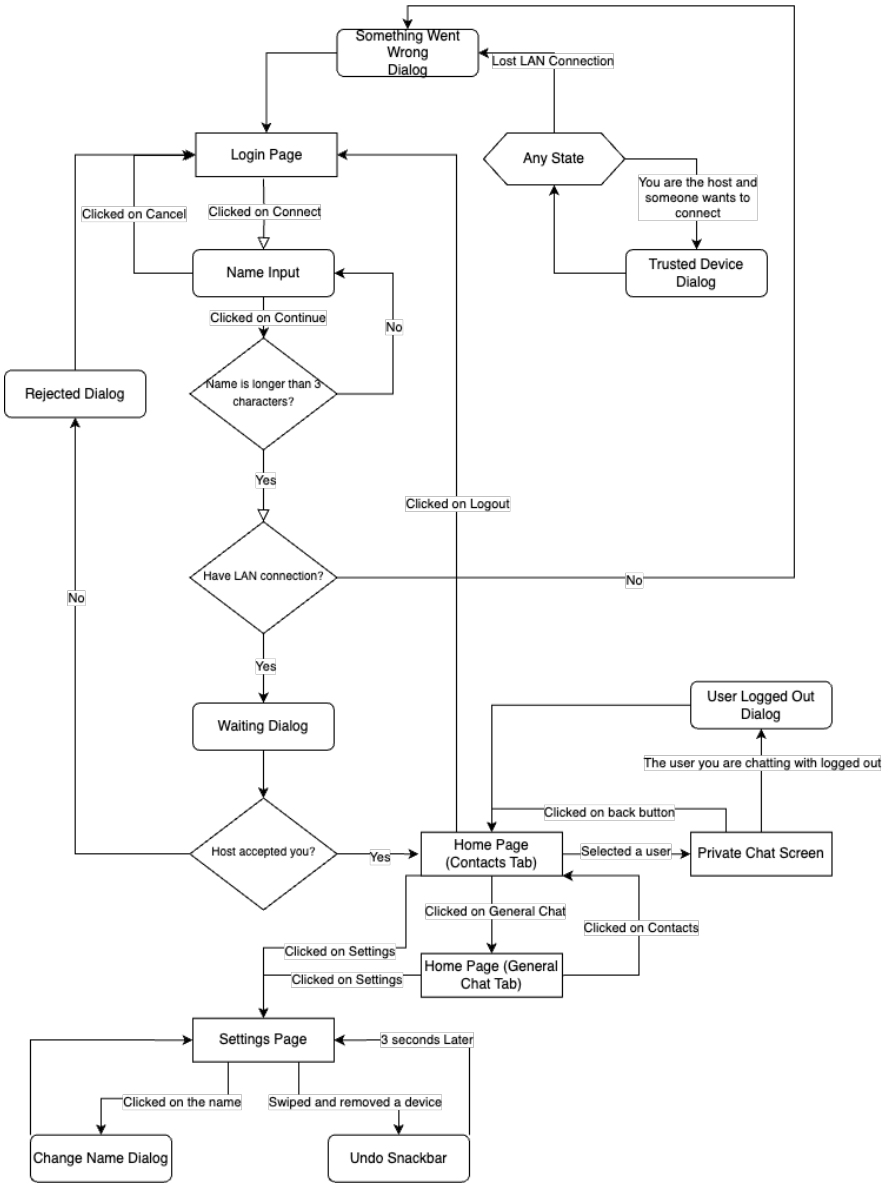

Figure 2.15: UI Transitions Diagram.

3. SYSTEM DESIGN

This chapter presents a comprehensive overview of the system design, including the protocol, architecture, encryption mechanisms, and algorithms. By detailing the underlying system design, this chapter provides a clear understanding of how the app operates and how it meets the requirements outlined in the previous sections.

3.1. Development Framework

Flutter was chosen as the development framework for several reasons. Firstly, it offers fast development capabilities, allowing for efficient and rapid creation of applications. Additionally, Flutter provides native performance, ensuring that the resulting applications perform optimally on both iOS and Android platforms. Another advantage of Flutter is its ability to deliver a consistent user interface across different devices, ensuring a consistent user experience. Furthermore, Flutter's cross-platform nature enables the development of applications that can run on multiple platforms with a single codebase. Lastly, the large and active Flutter community provides extensive support and resources making it a favorable choice.

3.2. Architecture

3.2.1. Server-Client model

Since there is no separate server app, the first user who logs in instantiates both a server and a client instance. This design decision was made to simplify the server implementation while maintaining a generic client structure. By treating its own client as any other client, the server component operates seamlessly within the app. This approach allows for a streamlined and efficient implementation of the server functionality, without adding unnecessary complexity to the overall system architecture.

Only the main classes that are relevant to server-client model are included in the class diagram (Figure 3.1) while omitting the majority of the simple helper classes to better visualize the server-client model and improve readability.

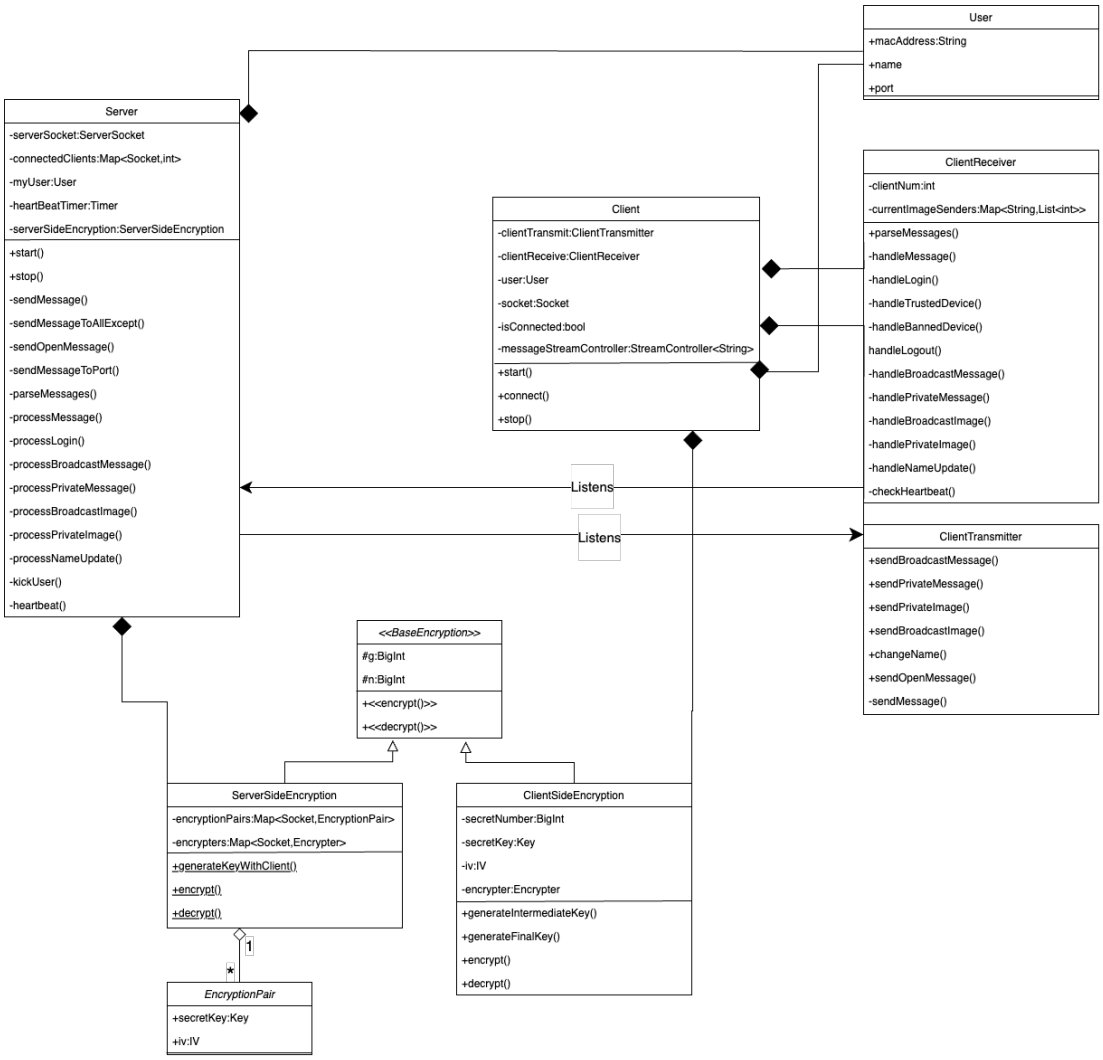

Figure 3.1: Simplified Class Diagram

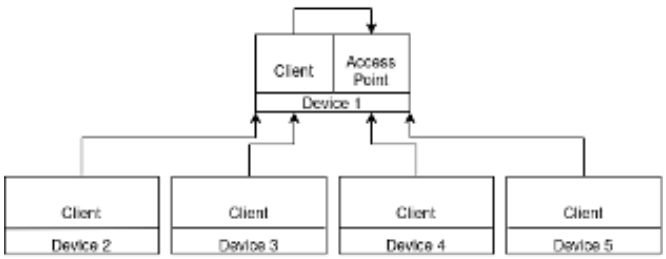

Figure 3.2: Initial Status of the network with 5 online users.

If any of the devices 2 to 5 (Figure 3.2) logs out, the server instance running on device 1 effectively handles the situation by sending relevant information to all connected clients regarding the logged-out client. However, when device 1, which acts as the host device, logs out, a predefined agreement among the clients becomes

essential to determine which client will take the role of the new server. This agreement operates as follows: Each client is assigned a unique number upon login. With client 1 (on the same device with the server) assigned the number 0, client 2 assigned number 1, client 3 assigned number 2, and so on. When a client logs out, the server decreases and updates the assigned numbers of all the clients which were logged in after the client which just logged out. Consequently, in the event of server disconnection, the client with the number 0 undertakes the responsibility of establishing a new server (Figure 3.3). This hierarchical approach ensures continuity of the network connection, enabling clients to sustain their communication even in the event of an unexpected server crash.

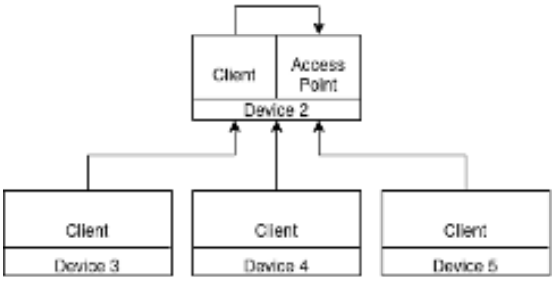

Figure 3.3: Final Status of the network after device 1 logs out.

3.2.2. Encryption

The encryption mechanism employed in the system is based on the 128-bit Advanced Encryption Standard (AES). To establish a secure key exchange, the Diffie-Hellman Key Exchange algorithm is used. This involves the use of predefined public numbers, where the client generates an intermediate key and transmits it to the server. Upon receiving the client's intermediate key, the server generates its own intermediate key and sends it back to the client. By using these "intermediate keys", both the client and server are able to generate the same shared secret key. Each client generates a single secret key while server generates one for each client and keeps them in a map structure.

When a client sends a message to another client, they encrypt it with their secret key and send it to server. Than, Server decrypts it with senders key, retrieves the receiver's information from the decrypted message, encrypts the message back with receiver's key and sends it to receiver. Finally, receiver decrypts it and displays it.

The specific formats and details of the key exchange packages will be further elaborated in the section 3.3.1.

3.2.3. Image Sharing

To ensure reliable transmission of large image files over the socket, a compression and splitting mechanism is implemented. Firstly, the images are compressed to reduce their size. Next, they are divided into smaller chunks, each consisting of 512 bytes. These chunks are then sent one by one over the socket, with a 50ms delay introduced between each package. This delay allows the socket to handle the transmission load effectively and prevents data loss or corruption. Additionally, to verify the integrity of the received image, a hash of the original image is also sent. This hash allows the recipient to compare and validate the received image against the original. It is important to note that due to the 50ms delays between packages, the overall time for image transmission can range from one second to 15 seconds, depending on the size of the image and the network conditions. This problem and possible solutions are discussed in the section (5.1)

3.2.4. Trusted and Banned Devices

Upon a user login, the server is presented with the option to accept or decline the client, as described in subsection 2.4.2. Once a decision is made, the information of the connecting device is saved locally on the device's local storage. Furthermore, this information is broadcasted to all currently online clients using the trusted device and banned device tokens, which will be discussed in detail in the subsequent section 3.3.1. This broadcasting mechanism ensures that all connected clients are aware of the newly logged-in device and can appropriately manage their own lists of trusted and banned devices.

3.2.5. Heartbeat

In order to make sure the connections between server and clients are intact, server sends each client a heartbeat message as specified in Table 3.2 every second. If a client doesn't receive a heartbeat in 5 seconds The message "Something went wrong" (Figure 2.14) is displayed.

3.3. The Messaging Protocol

Messages from client-to-server and server-to-client has the following format: **identifier⁋token1⁋token2⁋token3⁋token4**

Tokens are seperated with ⁋ and each message is seperated with a special unicode

character to avoid consecutive reading.

3.3.1. Messages Client-to-Server

Operation	Identifier	Token 1	Token 2	Token 3	Token 4
Login	li	<MAC-address>	<user-name>	<port>	<intermediate-key>
Broadcast Message	bc	<sender-MAC-address>	<message>	-	-
Private Message	pv	<sender-MAC-address>	<receiver-port>	<message>	-
Broadcast Image	bi	<sender-MAC-address>	<chunk-amount >	-	-
Broadcast Image Contd.	cb	<serialized-image>	<sender-MAC-address>	<chunk-number >	-
Broadcast Image End	eb	<hash-of-image>	<sender-MAC-address>	-	-
Private Image	si	<sender-MAC-address>	<receiver-port>	<chunk-amount >	-
Private Image Contd.	ci	<serialized-image-chunk>	<sender-MAC-address>	<receiver-port>	<chunk-number >
Private Image End	ei	<hash-of-image>	<sender-MAC-address>	<receiver-port>	-
Name Update	nu	<MAC-address>	<new-name>	-	-

Table 3.1: Client to Server Messages

3.3.2. Messages Server-to-Client

Operation	Identifier	Token 1	Token 2	Token 3	Token 4
Login	li	<Client-MAC-address>	<user-name>	<port>	-
Logout	lo	<port>	-	-	-
Heartbeat	hb	-	-	-	-
Broadcast Message	bc	<sender-MAC-address>	<message>	-	-
Private Message	pv	<sender-MAC-address>	<receiver-port>	<message>	-
Broadcast Image	bi	<sender-MAC-address>	<chunk-amount >	-	-
Broadcast Image Contd.	cb	<serialized-image>	<sender-MAC-address>	<chunk-number >	-
Broadcast Image End	eb	<image-hash-code>	<sender-MAC-address>	-	-
Private Image	si	<sender-MAC-address>	<receiver-port>	<chunk-amount >	-
Private Image Contd.	ci	<serialized-image-chunk>	<sender-MAC-address>	<receiver-port >	<chunk-number >
Private Image End	ei	<hash-of-image>	<sender-MAC-address>	<receiver-port >	-
Name Update	nu	<MAC-address>	<new-name>	-	-
Trusted Device	td	<MAC-address>	<user-name>	-	-
Banned Device	bd	<MAC-address>	<user-name>	-	-
Untrust a Device	rt	<MAC-address>	-	-	-
Unban a Device	rb	<MAC-address>	-	-	-
Server Intermediate Key (D-H Key Exchange)	sk	<intermediate-key>	<initial-vector>	-	-
Client Number	cn	<client-number>	-	-	-
Rejected	rj	-	-	-	-

Table 3.2: Server to Client Messages

3.3.3. Explanation of the Protocol

All messages except Login (li) from **client to server** and intermediate key (sk) from **server to client** are encrypted with AES. These 2 messages are used for Diffie-Hellman key exchange.

Login: When a client establishes a connection with the server socket, it sends a login message (Table 3.1) containing its MAC address, user name, listening port, and Diffie-Hellman intermediate key. If the new client is already listed in the host device's banned devices list, the server sends the "rejected" message to the new client, without any further prompts to the access-point device. If the server accepts the client or already has the client in it's trusted devices list, it performs the following steps:

- Sends its own intermediate key to the client.
- Saves the client's information and shares it with all other connected clients as a login message.
- Assigns a client number to the new client and sends this assigned number back to the client.
- Sends a "Trusted Device" message to all other clients, including the MAC address of the newly logged-in user.
- Shares the lists of existing trusted and banned devices with the new client.
- Sends the list of existing logged-in clients to the new client one by one as login messages.

However, if the server rejects the client, it follows these steps:

- Sends a "rejected" message to the client.
- Closes the socket connection with the client.
- Informs all clients by sending a "banned device" message.

Please refer to Table 3.2 for the details of the messages sent from the server to the client.

Logout: When a client socket connection is closed for any reason, the server handles the logout process as follows. Firstly, the server sends a logout message to all other connected clients, notifying them that a client has logged out. This ensures that the remaining clients are aware of the logout event. Secondly, the server adjusts the client numbers for the logged-in clients. If there are clients who were logged in after the client that has logged out, their client numbers are decreased by one and they are notified with their new client number. This ensures that the consistency of client numbering within the server is maintained. This step is needed because in specific situations where the server itself logs out or loses LAN connection, the client with the number 0 assumes the role of the new server. Other clients then establish connections with this new server. This transition allows for the continuity of server-client communication.

Name Change: When a user decides to change their name, they send a name update (nu) message to the server. The server then forwards this message to all connected clients, ensuring that everyone is notified of the name change.

Public and Private Messages: When the server receives a public message, it broadcasts it to all clients except the sender. On the other hand, if the server receives a private message, it forwards the message to the intended recipient identified by their <receiver-port>value.

Public and Private Images: For both public and private images, there are three distinct message types involved. The forwarding protocol remains the same for both types of images. However, clients need to compress the images first and then parse them into 512-byte chunks before sending them to the server.

To initiate the transmission of an image, the sender sends a Broadcast/Private Image message to notify the receiver that it is about to send an image. Subsequently, the client sends multiple Broadcast/Private Image Contd. messages, each containing a chunk of the image data. Finally, the client sends a Broadcast/Private Image End message, which includes the hash of the image. This hash serves as a means of ensuring the integrity of the image during transmission.

The server does not modify these packages and directly forwards them to the intended recipient client(s). To avoid any potential corruption, there is a 50ms delay between each image chunk. This causes larger image files to take longer to be sent (up to $\sim$15 seconds) but avoid corruption. If there is a corruption, receiving users are notified on the chat screen by a note under the image which says "... Sent an image, but it was corrupted" (Figure 4.1). The image corruption problem and possible solutions are addresed in the section 5.1.

Unban&Untrust: When the host device unbans or untrusts another device from the settings screen, this is broadcasted to all online devices. Ensuring that the preferences of the host devices is shared amongst all clients.

Heartbeat: Server sends a heartbeat message to all clients every second in order to make sure the connection is intact.

4. EVALUATION OF SUCCESS CRITERIA & TEST RESULTS

In this chapter, the success criteria of the project are evaluated. Initially, during the preliminary presentation, four success criteria were identified. These criteria included supporting a minimum of 10 clients, successfully sharing trusted devices, achieving less than 50ms latency on LAN connections, and maintaining latency below 100ms on Bluetooth connections.

However, during discussions with the project supervisor Dr. Gökhan Kaya following the secondary presentation, it was determined that incorporating Bluetooth connection capabilities would not contribute significant new features to the project but would impose a considerable additional workload. As a result, the decision was made to shift the project's focus towards successfully transferring the server ownership in scenarios involving unexpected host-device disconnections. Consequently, the fourth success criterion was revised to emphasize the successful transfer of access-point in all cases.

The following evaluation assesses the achievement of these revised success criteria by analyzing the project's performance in relation to each criterion.

4.1. Supporting at least 10 clients

To evaluate the success criterion of supporting at least 10 clients, a test was conducted involving a total of 12 clients. Among these clients, 10 were simulated iOS emulators, while 2 were physical Android devices. The purpose of the test was to assess the system's ability to handle multiple clients and facilitate communication between them.

During the test, all 12 clients were able to establish connections, view each other, and engage in messaging and image sharing simultaneously. This demonstrated that the system successfully supported the desired number of clients, meeting the established success criterion.

However, a minor issue was detected when 5-6 or more clients attempted to send images concurrently, particularly when the images were large in size. In such cases, some of the images experienced minor corruptions, as depicted in Figure 4.1. It was observed that this issue was specific to simultaneous large image transfers. This problem and the possible solutions are further evaluated in the section 5.1

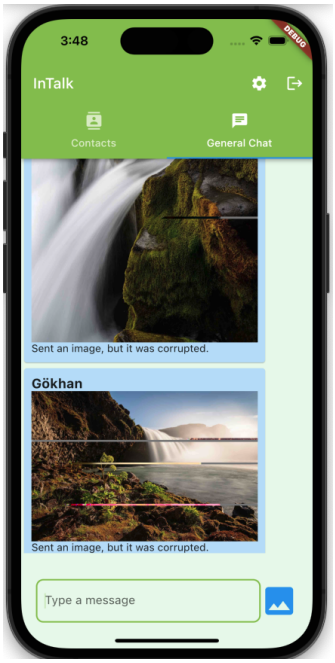

Figure 4.1: Corrupted Images.

4.2. Successfully sharing trusted devices

In order to evaluate the success criterion of successfully sharing trusted devices, a test was performed with a total of 12 clients, following the same test setup as mentioned previously. The objective of this test was to verify the system's capability to share trusted and banned device information accurately and consistently among all connected clients.

During the test, it was confirmed that the server effectively shared the current states of the trusted and banned devices lists with all 12 clients. Each client received the updated information regarding trusted and banned devices, ensuring that all clients were aware of the devices that were deemed trustworthy or banned within the system.

This successful sharing of trusted and banned devices demonstrates that the system met the defined success criterion. It highlights the system's ability to maintain consistent and synchronized device lists across all clients, enabling effective management of device access and security within the network.

It should be noted that users have ability to modify their own device lists. In this regard, clients have the capability to remove devices from their banned device list if they choose to do so, even if the device was initially flagged as banned by the host device. Furthermore, it is important to consider that users who are not currently logged in will not receive updates regarding changes to device lists of the host device. Therefore, when the access point is transferred, there is a possibility that some logged-in users

may be automatically kicked out if they happen to be included in the banned device list of the new host device. This behavior is intentional and aligns with the intended functionality of the system.

4.3. Successfully transferring the access-point

As described in subsections 3.2.1 and 3.3.3, an algorithm has been implemented to ensure the reliable transfer of the access point. To evaluate the effectiveness of this algorithm, a test was performed involving 12 devices. The test consisted of multiple trials to assess the success of the access point transfer under various scenarios.

In the first set of trials, the current host device was systematically logged out and then logged back in, one device at a time. In each trial, the transfer of the access point was accomplished successfully, indicating the algorithm's capability to handle sequential transfers without any issues.

Subsequently, a series of random logouts and logins were performed, with a total of 10 iterations for each action. The objective of this test was to examine whether the algorithm accurately decreased the client numbers as expected when clients reconnected after being logged out. Throughout all the trials, the transfer of the access point was consistently successful, demonstrating that the algorithm effectively managed the client numbers and facilitated seamless transitions.

Lastly, the first 2 tests were repeated but instead of logging out, the app is directly closed. Despite this abrupt closure, the system successfully managed the access point transfer, ensuring uninterrupted communication among the remaining devices.

The results of these tests provide strong evidence that the implemented algorithm reliably ensures the transfer of the access point. It demonstrates the system's ability to maintain continuous connectivity and communication, even in situations involving client disconnections and re-connections.

4.4. Less than 50ms of latency

For this test, five emulators running on the same machine were used since the system time of them is exactly the same. In different physical devices, system time varies between 500ms to a second, so it wouldn't be healthy to measure the time with different systems.

Client1 sent a broadcast message at 16:52:52.844209, and the following are the timestamps when each subsequent client received the message:

- Client2 received the message at 16:52:52.892925.

- Client3 received the message at 16:52:52.894229.
- Client4 received the message at 16:52:52.894106.
- Client5 received the message at 16:52:52.895352.

It should be noted that the calculated times are the times between Client1 taps on the send button and each client receives the message, parses it, and saves it. It includes the overhead of encryption and decryption [2]. Even with this overhead, calculated times are close to approximately 40ms and meet the criterion. However, as discussed in the previous sections, sending images takes more than a second and does not meet the criterion.

5. CONCLUSION

5.1. Known Bugs, Problems & Future Work

During the development of the application, there were a few identified bugs and areas for improvement. The major problem encountered was the time it takes to search for an access point on the initial login, which can take around 1 to 2 second(s). In situations where multiple clients log in simultaneously when there is no existing access point, they may establish separate access points without detecting each other. However, this problem is unlikely to occur in the expected user scenario of maximum 10-20 users within the same household. Additionally, a simple logout and login procedure can resolve the issue.

The solution to address this problem in the future is to implement a structure where multiple access points are listed on the login page. Users would have the option to connect to an existing access point or create their own. This would mitigate the problem by allowing users to easily identify and connect to the correct access point. But due to time constraints during the semester, the implementation of the proposed structure was not possible. Therefore, this remains as a future work for further development of the project.

Another identified issue is the delay in image sending to prevent corruptions. Currently, a delay of 50 ms is added between image chunks. This results in a 1 to 15 second(s) image sending time, depending on the image size. Additionally, even with the delay, minor corruptions might occur in the scenarios where more then 4 clients send images simultaneously. To improve this process, future work includes implementing a checksum algorithm and a mechanism to request and retrieve corrupted image chunks from the sender. This would help minimize the delay in image transmission and ensure reliable delivery. So far, a corruption check mechanism is implemented using SHA-256 hashing algorithm which checks the corruption of the whole image and warns the receiver if the image is corrupted (Figure 4.1). The mentioned algorithm for checking each chunk and retrieving them is a future work.

Another potential improvement is implementing a local database for storing messages. This enhancement is relatively straightforward to implement and is not mandatory for the functionality of the application.

5.2. Summary

Overall, a stable cross-platform application has been developed that enables reliable communication in scenarios where there is no internet connection available and meets the main objectives. The application successfully preserves connections in various conditions, allowing users to securely communicate with others within the reach of the same network.

You can follow the future development of the application from the GitHub Repository of the project.

URL: `https://github.com/gokbeykeskin/InTalk-Local-Network-Chat`

You can watch the trailer video of the project from my YouTube channel.

URL: `https://youtu.be/V0kKCWakVDY`

BIBLIOGRAPHY

[1] O. M. Junio and E. P. Chavez, "Development of offline chat application: Framework for resilient disaster management," in *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*, 2018, pp. 510–514. DOI: 10.1109/IICSPI.2018.8690351.

[2] S. Cavalieri and G. Cutuli, "Implementing encryption and authentication in knx using diffie-hellman and aes algorithms," in *2009 35th Annual Conference of IEEE Industrial Electronics*, 2009, pp. 2459–2464. DOI: 10.1109/IECON.2009.5415232.

[3] M. Belchin and P. Juberias, "Dart on the server side," in *Web Programming with Dart*. Berkeley, CA: Apress, 2015, pp. 367–386, ISBN: 978-1-4842-0556-3. DOI: 10.1007/978-1-4842-0556-3_28. [Online]. Available: https://doi.org/10.1007/978-1-4842-0556-3_28.

[4] H. Yamamura, K. Kaneda, and Y. Mizobata, "Communication problems after the great east japan earthquake of 2011," *Disaster Medicine and Public Health Preparedness*, vol. 8, no. 4, pp. 293–296, 2014. DOI: 10.1017/dmp.2014.49.