"MESH MESSAGING SERVICE USING BLUETOOTH"

A Socially Relevant Project-I
Report submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR.

In Partial Fulfillment of the Requirements for the Award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
BY

P.HITESH REDDY	**19121A05H1**
P.ASHISH	**19121A05G0**
R.B.MOULYA	**19121A05J6**
P.RUCHITHA	**19121A05J4**
S.AMISHA	**19121A05N2**

Under the Guidance of

Dr.J.Aswini
Professor

Department of Computer Science and Engineering
SREE VIDYANIKETHAN ENGINEERING COLLEGE
(Affiliated to JNTUA, Anantapuramu)
Sree Sainath Nagar, Tirupathi – 517 102

2021-2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION AND MISSION

VISION

To become a Center of Excellence in Computer Science and Engineering by imparting high-quality education through teaching, training, and research.

MISSION

The Department of Computer Science and Engineering is established to provide undergraduate and graduate education in the field of Computer Science and Engineering to students with a diverse background in foundations of software and hardware through a broad curriculum and strongly focused on developing advanced knowledge to become future leaders.

Create knowledge of advanced concepts, innovative technologies and develop research aptitude for contributing to the needs of industry and society.

Develop professional and soft skills for improved knowledge and employability of students.

Encourage students to engage in life-long learning to create awareness of the contemporary developments in computer science and engineering to become outstanding professionals.

Develop attitude for ethical and social responsibilities in professional practice at regional, National and International levels.

Program Educational Objectives (PEO’s)

1. Pursuing higher studies in Computer Science and Engineering and related disciplines

2. Employed in reputed Computer and I.T organizations and Government or have established startup companies.

3. Able to demonstrate effective communication, engage in teamwork, exhibit leadership skills, ethical attitude, and achieve professional advancement through continuing education.

Program Specific Outcomes (PSO’s)

PSO1: Use mathematical methodologies to model real-world problems, employ modern tools and platforms for efficient design and development of computer-based systems.

PSO2: Apply adaptive algorithms and methodologies to develop intelligent systems for solving problems from inter-disciplinary domains.

PSO3: Apply suitable models, tools, and techniques to perform data analytics for effective decision-making.

PSO4: Design and deploy networked systems using standards and principles, evaluate security measures for complex networks, apply procedures and tools to solve networking issues.

Program Outcomes (PO's)

1. Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems (**Engineering knowledge**).

2. Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences (**Problem analysis**).

3. Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and the cultural, societal, and environmental considerations (**Design/development of solutions**).

4. Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions (**Conduct investigations of complex problems**).

5. Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations (**Modern tool usage**)

6. Apply to reason informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice (**The engineer and society**)

7. Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development (**Environment and sustainability**).

8. Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice (**Ethics**).

9. Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings (**Individual and teamwork**).

10. Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions (**Communication**).

11. Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments (**Project management and finance**).

12. Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change (**Life-long learning**).

Course Outcomes

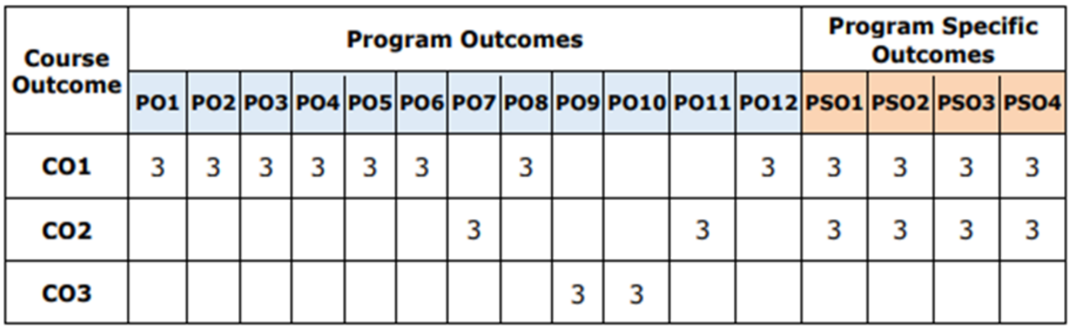
CO1. Create/Design engineering systems or processes to solve complex societal problems using appropriate tools and techniques following relevant standards, codes, policies, regulations, and latest developments.

CO2. Consider the environment, sustainability, economics, and project management in addressing societal problems.

CO3. Perform individually or in a team besides communicating effectively in written, oral and graphical forms on socially relevant project

Socially Relevant Project- I

CO-PO Mapping



Course Outcome	Program Outcomes												Program Specific Outcomes			
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	3	3	3	3	3	3		3				3	3	3	3	3
CO2							3				3		3	3	3	3
CO3									3	3						

(Note: 3-High, 2-Medium, 1-Low)

DECLARATION

We hereby declare that this project report titled **"Mesh Messaging Service using Bluetooth"** is a genuine Socially Relevant Project - I work carried out by us, in **B.Tech *(Computer Science and Engineering)*** degree course of **Jawaharlal Nehru Technological University Anantapur** and has not been submitted to any other course or University for the award of any degree by us.

Signature of the student

1. P. Hitesh Reddy
2. P. Ashish
3. R.B. Moulya
4. P. Ruchitha
5. S. Amisha

SREE VIDYANIKETHAN ENGINEERING COLLEGE

(AFFILIATED TO JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR)

SREE SAINATH NAGAR, A. RANGAMPET, TIRUPATI – 517 102, CHITTOOR DIST., A.P.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the

Socially Relevant Project-I entitled

"**Mesh Messaging Service using Bluetooth** "

is the bonafide work done by

P. HITESH REDDY	**19121A05H1**
P. ASHISH	**19121A05G0**
R.B. MOULYA	**19121A05J6**
P. RUCHITHA	**19121A05J4**
S. AMISHA	**19121A05N2**

In the Department of Computer Science and Engineering, Sree Vidyanikethan Engineering College, A. Rangampet. is affiliated to JNTUA, Anantapuramu in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering.

This is work has been carried out under my guidance and supervision.

The results embodied in this Project report have not been submitted to any University or Organization for the award of any degree or diploma.

Internal Guide	**Head**
Dr.J.Aswini Kumar Rao	**Dr. B. Narendra**
Professor	Prof & Head
Dept of CSE	Dept of CSE
Sree Vidyanikethan Engineering College Engineering College	Sree Vidyanikethan
Tirupathi	Tirupathi

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We are extremely thankful to our beloved Chairman and founder **Dr. M. Mohan Babu** who took a keen interest to provide us with the infrastructural facilities for carrying out the project work.

We are highly indebted to **Dr. B.M. Satish**, Principal of Sree Vidyanikethan Engineering College for his valuable support and guidance in all academic matters.

We are very much obliged to **Dr. B. Narendra Kumar Rao,** Professor & Head, Department of CSE, for providing us the guidance and encouragement in the completion of this project.

We would like to express our indebtedness to the project coordinator, **M.Venkatesh**, Assistant Professor, Department of CSE for his valuable guidance during the course of project work.

We would like to express our deep sense of gratitude to **Dr.J.Aswini**, Professor, Department of CSE, for the constant support and invaluable guidance provided for the successful completion of the project.

We are also thankful to all the faculty members of the CSE Department, who have cooperated in carrying out our project. We would like to thank our parents and friends who have extended their help and encouragement either directly or indirectly in the completion of our project work.

ABSTRACT

The idea is to create a service that enables users to communicate with other users using a Mesh Network established between their devices. The Mesh network is to be built using the Bluetooth service available in the user devices. There are apps built using this technology such as Bridgefy, Mesh Messaging, Firechat, etc. But the proposed application enables users to send messages to other users through the Bluetooth network established between the user devices. The application does not require any Internet connection. The idea is to transfer data from the sender to the receiver where it finds its way using a Dynamic routing algorithm. The middle devices (user devices that are not participating in the communication) act as routers in the network. The app encrypts the messages which are sent from the sender so that the middle devices cannot see them.

TABLE OF CONTENTS

CHAPTER -1
INTRODUCTION

Bluetooth is an open standard for implementing short-range wireless communication. The data is transferred through the network with the help of mesh networking technology. The working principle is that the devices that take part in the network act as the endpoint devices as well as the routers of the network. The more the number of devices, the better the functionality of the network. It is deployed with a Distance Vector Routing algorithm, which will be customized to work with Bluetooth technology. To protect the messages from third persons it will be deployed with a strong and efficient Encryption algorithm (Advanced Encryption). App has been built in VS code using Flutter for Android devices. With the developed application the users can message even without an internet connection or 2G, 3G, or 4G network coverage using Bluetooth.

STATEMENT OF PROBLEM

One of the most available modes of communication is messaging over either the internet or mobile network. Over 6 billion texts are sent every day. Platforms like WhatsApp, Twitter, Instagram have become the primary source of information through communication. They are enabled through either an Internet connection or through a Mobile Network.

The idea we propose aims to provide an alternate way for communication using Bluetooth. It is to build an app that lets the user send messages without any of the regular connectivity. The users can connect to each other directly and pass messages over the network they create. The messages sent over this network hop across it until it reaches its destination. This service is also entirely free of cost. The effectiveness of this idea increases as the number of active users increases.

OBJECTIVES

1. Designing a clean and simple user interface, so the users can use the application with ease.
2. Adding mobile number authentication during the initial run of the application, to verify the legitimacy of the users.
3. Creating a database to store the data locally on the user device, so that it can be available when offline.
4. Implementing an asymmetric encryption service to ensure that the privacy of the users is respected.
5. Adding a functionality which enables the users to start a new conversation by scanning a QR code on the other user's application.
6. Designing a dynamic routing algorithm that can respond to the constant changes in the topology of the network.
7. To prevent clogging up of the network by duplicate packets, adding a time to live for the network packets.

SCOPE

This project model is primarily targeted on the populations that have a difficulty in communicating through the conventional media of communication. This can prove to be very useful where there is little to no carrier signal and internet service.

Although, this application can be used by anyone as it has very little dependency on the external factors and is also free of cost when compared to the other modes of communication.

APPLICATIONS

1. **Natural Calamities:**

   During the aftermath of a natural disaster, the basic means of communication might not be available, since there is a high probability that the signal towers, cables, and routers are damaged. At times like these, a service like the proposed one can come in handy and might help save lives.

2. **Public protests:**

   During the recent Hong Kong protests the government took down internet services to prevent the protesters from communicating. To overcome this problem the protesters used similar technology to create a platform of their own to plan and organize themselves.

3. **Remote locations:**

   People that live in remote areas like hill stations, or in the middle of a forest might not always have a network reception and neither do they have access to proper internet. This not only cuts them off from the rest of the world but also prevents them from communicating among themselves. This idea can help them establish a network within their community and use it to communicate with ease.

4. **Closed communities:**

   In schools or offices, creating a network only for themselves saves a lot of time that might otherwise be spent walking corridors or climbing stairs.

5. **Unavailability of other media:**

   Everyone may have faced this situation when someone is trying to send some important message (like office messages) to another, suddenly their carrier signals may be lost or get a low signal that leads to frustration then This application becomes handy.

LIMITATIONS

- Bluetooth uses short-wavelength UHF radio waves. These waves can easily be disrupted or blocked by solid objects. This leads to the inability of the connection between devices.
- The application must keep checking for changes in the network topology to update its routing table. This requires a decent amount of constant memory utilization as well as battery power.
- The speed of the delivery of the message entirely depends upon the availability of middle devices (more users). This uncertainty of the delivery of the message is a drawback to this model.
- Since the networks are formed entirely between the user devices, the usage of this model can lead to multiple networks that are unrelated to each other when two populations are divided by a distance.
- Since the network doesn’t exactly specify the location of the receiver, the routing algorithm must implement the flooding technique, which has a reputation of clogging up the network if not optimized.

CHAPTER - 2

LITERATURE SURVEY

Bluetooth Low Energy:

Bluetooth Low Energy or BLE is a wireless local area network technology that consumes lesser power than traditional Bluetooth technology. The devices that intend to transfer data through a BLE connection must first create a channel. Android provides APIs that applications can use to discover nearby devices, query for services, and transmit information.

Mesh Network:

A Mesh network is a kind of network where the devices that participate in the communication also act as the routers for the network. This allows various devices to link together branching off each other allowing the signal and network to spread further. There are two types of mesh networks: wired mesh networks and wireless mesh networks.

Wireless mesh networks can easily and effectively connect large areas using inexpensive, existing technology. In this kind of network, the network connection is spread out among various wireless mesh nodes that “talk” to each other to share the network connection across a large area.

RSA Encryption:

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e., public key and private key. As the name describes that the public key is given to everyone and private key is kept private.

The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So, if somebody can factorize the large number, the private key is compromised. Therefore, encryption strength totally lies on the key size and if we double or triple the key size, the strength of the encryption increase exponentially. RSA keys can be typically 1024 or 2048 bits long but experts believe that 1024-bit keys could be broken in the near future. But till now it seems to be an infeasible task. The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers.

Distance Vector Routing:

A distance vector routing protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm or the Bellman-Ford algorithm.

Bellman Ford Basics - each router maintains a distance vector table containing the distance between itself and all possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbors' distance vectors. Associated with each link connected to a router, there is a link cost.

- A router transmits its distance vector to each of its neighbors in a routing packet.
- Each router receives and saves the most recently received distance vector from each of its neighbors.
- a router recalculates its distance vector when it receives a distance vector from a neighbor containing different information than before or when it discovers that a link to a neighbor has gone down.

CHAPTER - 3
ANALYSIS

The Android application provides a complete user interface for the users to send messages, receive messages, read the past messages and also manage all the conversations they have. The application was built with the Flutter toolkit and Dart libraries.

At the initial run of the application, the users are prompted to provide their mobile number, which will be used to create an account for them with the help of Firebase Phone Authentication. Upon a successful account creation, the application generates an RSA key pair. This is done with the help of the PointyCastle dart library. Further details on how the keys are generated are provided in the Implementation chapter of this report.

Also, a shared preference is created to store the login state of the user, so that the next time the user opens the app, it automatically logs them in.

As the application is intended to work offline, it stores all the data locally on the user’s device. It does so by creating a local database on the user device. The application uses the SQLite dart library to create the database. The database maintains two tables, one for the conversations and one for the messages. A detailed relational model of the database is explained in the Design chapter of this report.

To start a conversation, the user must scan a QR code, which is uniquely generated for each user at the time of their account creation, of the other user. The QR code is encoded with a string that is a concatenation of both the RSA public key of the user, their Bluetooth address, their username and their mobile number. Upon scanning a QR code a conversation is created in the user’s conversation list and also the details of the other person are stored into the local database.

To send a message to a contact, the user taps on their conversation and types the message into the message field located at the bottom of the screen. When the user taps on the send button, the process of message transmission begins. This process includes a series of steps as: encryption, routing in the network, decryption. Also, the message is stored into the local database along with its details.

The encryption involves using the public key of the receiver to convert the plain text of the message to cipher text. The address of the receiver is appended to the processed message. The routing part involves looking for the Bluetooth address of the receiver in the nearby devices table. This table contains a list of all the nearby devices. If the address of the receiver is not in the list, then the processed message is flooded to all the nearby devices. The nearby devices then look for the address of the receiver in their tables. This flooding process is repeated at all the intermediate devices until the message reaches the receiver. To prevent the clogging of the network from all the flooded messages, we specify the time to live for all the messages. Once this time elapses, the message packet is destroyed.

Once the message has arrived at the receiver, the receiver application decrypts it by using the receiver's private key. This yields the text of the actual message. This message is then added to the local database of the receiver and also displayed in the corresponding conversation.

CHAPTER - 4
DESIGN

UML design:

Use Case diagram:

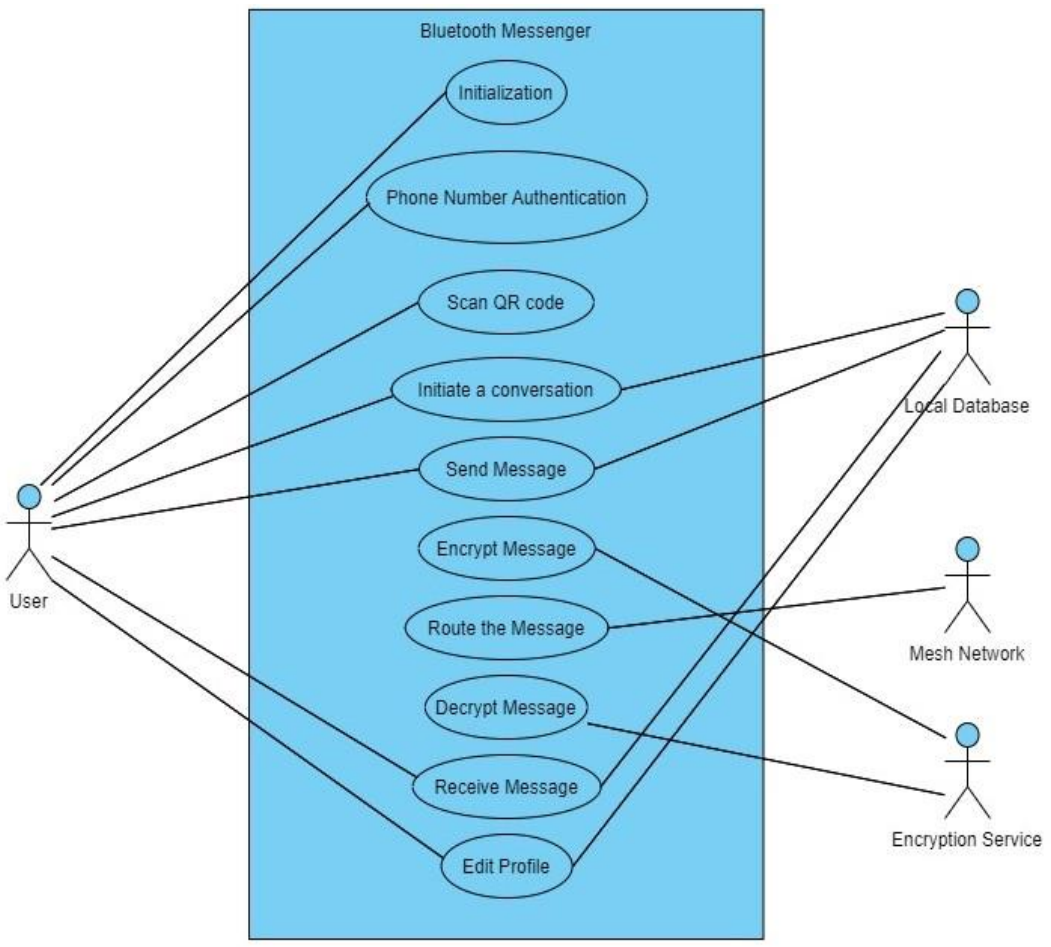


Figure 4.1 Use case Diagram

Class Diagram:

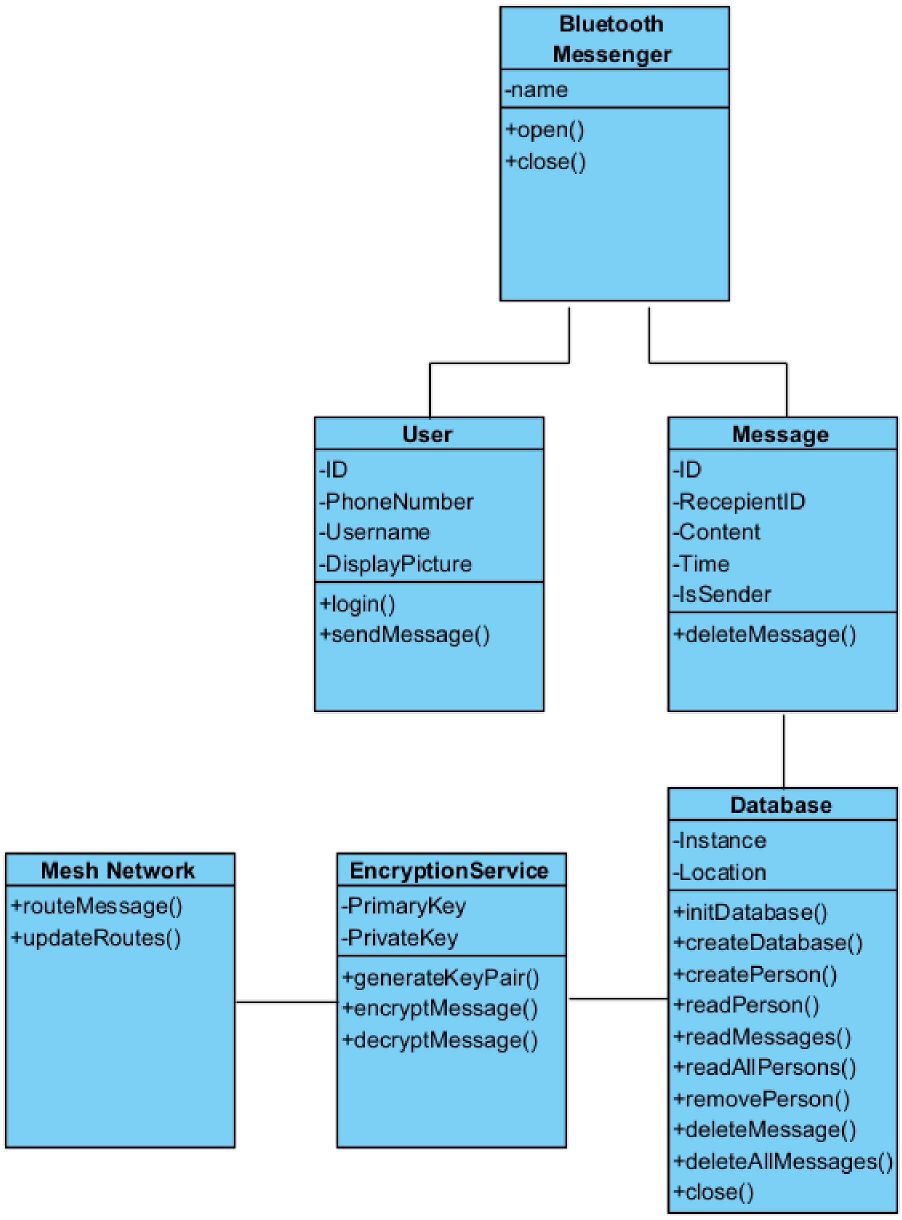


Figure 4.2 Class Diagram

Sequence Diagram:

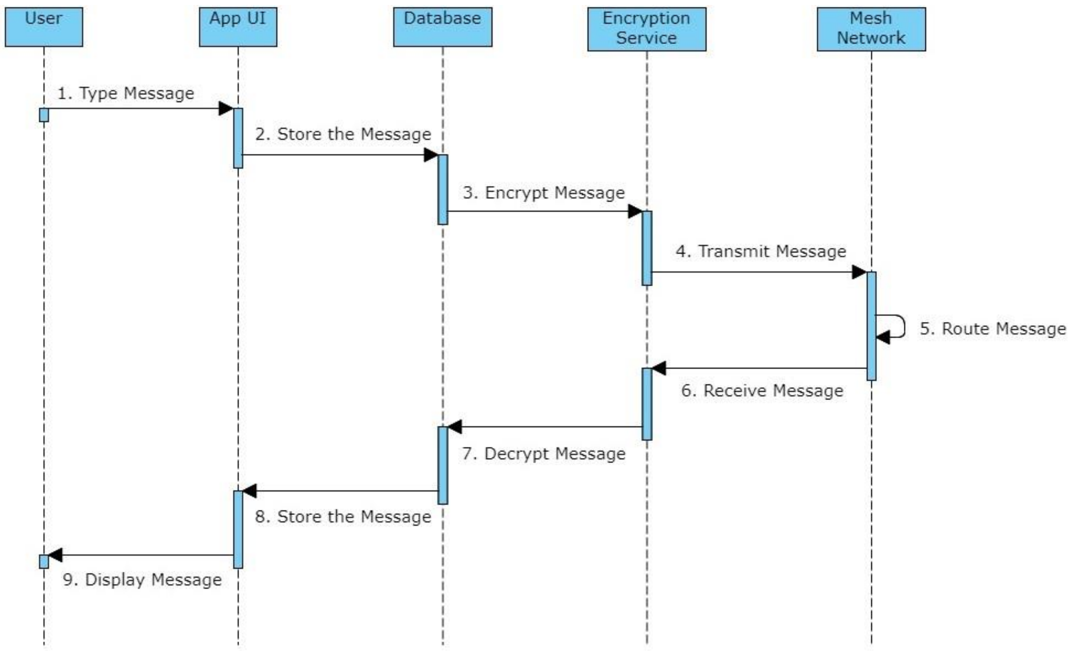


Figure 4.3 Sequence Diagram

Activity Diagram:

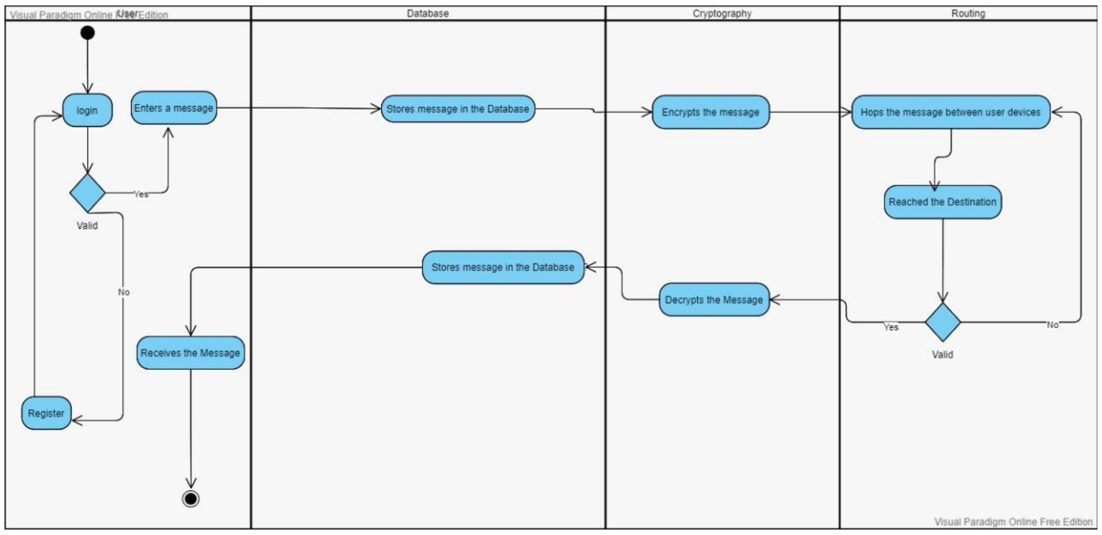

Visual Paradigm Online Free Edition
User
Database
Cryptography
Routing
login
Enters a message
Yes
Valid
No
Register
Stores message in the Database
Encrypts the message
Hops the message between user devices
Reached the Destination
Valid
Yes
No
Decrypts the Message
Stores message in the Database
Receives the Message
Visual Paradigm Online Free Edition

Figure 4.4 Activity Diagram

Database Design:

The Relational Model of the database is represented as an entity relationship in Figure 4.5.

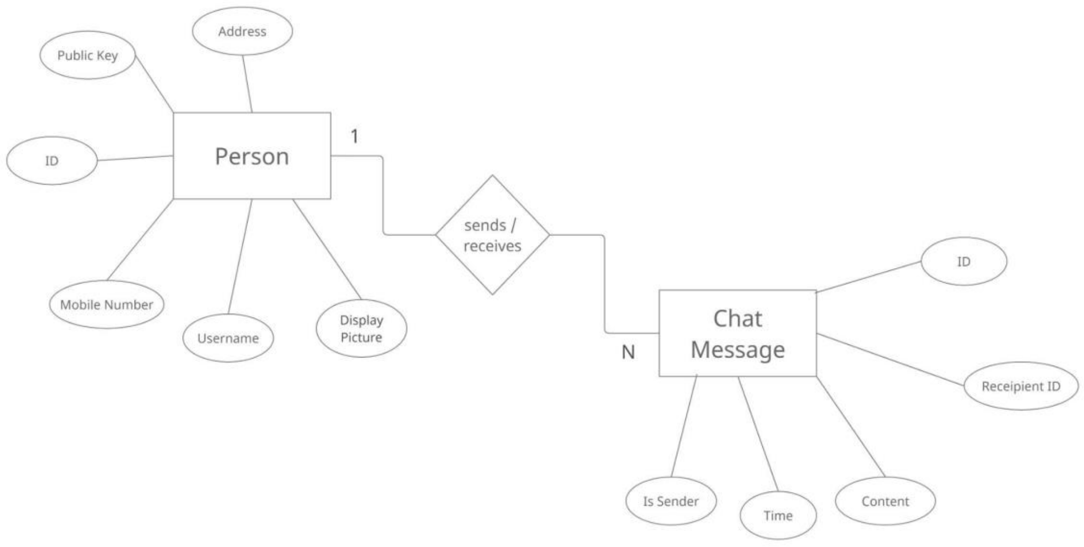


Figure 4.5 Entity Relationship Diagram

The database is designed to have two tables. The first table is to store the conversations and its details. The second table stores the messages, both sent and received, along with their details.

The attribute *'id'* acts as the primary key for the *'persons'* table. This is an arbitrary integer value, and it has the auto increment property, which means its value automatically increments for each new row added to the table. All other attributes of this table are of string type.

In the *'messages'* table, the *'id'* attribute acts as the primary key. This attribute is similar to the one in the *'persons'* table. The *'recipient id'* attribute is a foreign key that references to the *'id'* attribute of *'persons'* table. All other attributes of this table are of string type.

CHAPTER - 5
IMPLEMENTATION

Dependencies:

The application uses the following Dart libraries

- firebase_core 1.10.0
- firebase_auth 3.2.0
- sqflite 2.0.0+3
- pointycastle 1.0.0-rc4
- asn1lib 1.0.3
- shared_preferences 2.0.11

These dependencies must be specified in the *'pubspec.yaml'* file in the flutter application root directory.

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^1.10.0
  firebase_auth: ^3.2.0
  sqflite: ^2.0.0+3
  pointycastle: ^1.0.0-rc4
  asn1lib: ^1.0.3
  shared_preferences: ^2.0.11
```

firebase_core and firebase_auth libraries add the authentication functionality to the application.

sqflite library is used to implement the local database for the application.

pointycastle and asn1lib libraries provide the encryption services for the application.

shared_preferences library lets the application to create shared preferences, which stores small amount of primitive data as a key/value pair.

Theme:

The application uses the following Theme data

Primary color: #AB20FD

Primary accent color: #7D12FF

Secondary color: #212121

Secondary accent color: #323232

Font: Ubuntu (Google Fonts)

Navigation:

There are seven screens in the application and each of them are implemented in an individual dart file. They are mapped as in Table 5.1.

Screen name	File name
Welcome Screen	welcome_scren.dart
Registration Screen	signin.dart
Conversations Screen	chats.dart
Chat Screen	chat.dart
Edit Profile Screen	edit_profile.dart
Settings Screen	setting.dart
Profile Screen	profile.dart

Table 5.1 Application Screens and their files

Additionally, there are other dart files that provide functional support to the files that are mentioned in Table 5.1. These files and their functions are specified in Table 5.2.

File name	**Function**
main.dart	It contains the dart script that runs the application.
constants.dart	It contains the pre-defined values such as theme data and also definitions of custom data types.
chat_database.dart	It contains the definition of the SQLite database that is used by the application and also the methods to perform operations on the database.
encrypt_service.dart	It contains the methods that are used to generate RSA key pairs,
router.dart	It contains the routing table and the routing algorithm.

Table 5.2 Back-end code files and their functionalities

The application starts from the *'main.dart'* file, which is basically is runner code for the application. It returns a MaterialApp object, which actually is the main application.

At startup, the application starts in the welcome screen While in this page, the application checks if the *'LoggedIn'* shared preference has been set or not. If it is set, then the application navigates the conversations page, else the application navigates to the registration screen.

In the registration screen, the user interface consists of a text field that accepts mobile numbers. Once the user enters their mobile number and taps on the submit button, an OTP will be sent to the provided mobile number and a new text field to enter the OTP appears on the screen. If the user enters a valid OTP and taps on the verify button, their login state will be persisted by setting a shared preference name *'LoggedIn'*. Upon successful authentication, the application navigates to an empty conversations screen.

To start a new conversation, the user must tap on a floating action button at the bottom right of the screen, which opens the camera of the device. The user must scan the QR code of the other user with whom they intend to start a new conversation, which is available in the settings page of the application. To navigate to the settings screen, the user can tap on the settings icon available at the top right of the application screen. The QR code encodes a string which contains the Bluetooth address of the user, RSA public key, mobile number, username. Scanning the QR code decodes the string and adds the conversation into the *'persons'* table in the database. The display picture for this conversation must be manually set by the user, else a default display picture will be displayed.

The edit profile screen can be opened by tapping on the profile icon in the appbar of the conversations screen. In the edit profile screen, the user is provided with the ability to change their display picture, username, and add some lines about them.

To send a message, the user must tap on the conversation of interest, which navigates the application to the chat screen. The chat screen consists of an appbar which displays the username and display picture of that particular conversation, along with a back button. The main body of the chat screen is a list that displays the messages of the conversation. The sent messages are displayed to the right of the screen, while the received messages are displayed to the left of the screen. They are further differentiated by having a different bubble color. At the bottom of the chat screen is a bar with the text field to type the message in, and the send message. The user can type the message in the text field and tap on the send button to start the process of transmitting the message to the recipient. Tapping on the appbar navigates the application to the profile screen.

The profile screen displays the display picture of the conversation, the username, mobile number and its Bluetooth address.

Database:

The SQLite database is created locally on the user's device in their '*Android/data*' directory. The database is created on the initial run of the application and for every other run, the existing database is fetched. The exact location of the database is not visible to the user among the files of the application, it is protected by the Android system.

The database consists of two tables, one to store the conversations and one to store the messages. The table that stores the conversations is named as '*persons*' and the table that stores the messages is named as '*messages*' in the database. Table 5.3 and Table 5.4 provide an imaginary representation of the '*persons*' table and '*messages*' table.

persons

id	**number**	**username**	**Display Picture**	**publicKey**	**address**

Table 5.3 A model of the '*persons*' table

messages

id	**recipient**	**content**	**time**	**isSender**

Table 5.4 A model of the '*messages*' table

The methods that are used to perform operations on the database are listed in Table 5.5.

Method	**Parameters**	**Returns**	**Functionality**
createPerson	A Person object	id of the row	Inserts a row into the '*persons*' table
removePerson	id of the	None	Removes a row

	person		from the *'persons'* table
readPerson	mobile number of the person	Person object	Selects a row from the *'persons'* table
readAllPersons	None	List of Person objects	Selects all rows from the *'persons'* table
postMessage	A ChatMessage object	ChatMessag e object	Inserts a row into the *'messages'* table
deleteMessage	id of the message	None	Removes a row from the *'messages'* table
readMessage	id of the message	ChatMessag e object	Selects a row from the *'message'* table
readMessages	id of the recipient	List of ChatMessag e objects	Selects all rows of a particular conversation from the *'messages'* table
getLastMessage	id of the recipient	content of a ChatMessag	Retrieves the content the last

		e object	message of a particular conversation from the *'messages'* table
deleteAllMessages	id of the recipient	None	Removes all rows of a particular conversation from the *'messages'* table

Table 5.5 Database methods

Encryption:

At the time of account creation, the application runs a piece of code that initializes the encryption service for the application. It generates a RSA key pair and stores them with shared preferences. The public key is also used to generate the QR code for the user.

Firstly, the parameters for the key generator are initialized.

```
  var keyParams =
RSAKeyGeneratorParameters(BigInt.from(65537), 2048, 5);
```

The encryption service uses FortunaRandom class to synchronously generate random numbers. This FortunaRandom class must be intialiazed with a list of random integers known as seeds.

```
  var secureRandom = FortunaRandom();
  var random = Random.secure();

  List<int> seeds = [];
  for (int i = 0; i < 32; i++) {
    seeds.add(random.nextInt(255));
  }


secureRandom.seed(KeyParameter(Uint8List.fromList(seeds))
);
```

Finally, the service creates a RSAKeyGenerator object and initializes it with the previously initialized parameters.

```
  var rngParams = ParametersWithRandom(keyParams,
secureRandom);
  var k = RSAKeyGenerator();
  k.init(rngParams);
```

After the key generator is completely initialized, it can be used to generate the RSA key pair.

```
  AsymmetricKeyPair<PublicKey, PrivateKey> keyPair =
k.generateKeyPair();
  RSAPrivateKey privateKey = keyPair.privateKey as
RSAPrivateKey;
  RSAPublicKey publicKey = keyPair.publicKey as
RSAPublicKey;
```

The encryption service also provides two methods: encrypt and decrypt. These methods are used to convert plain text to cipher text and cipher text to plain text respectively. The encrypt method takes plain text and RSA public key as parameters and return the cipher text. The decrypt method takes cipher text and RSA private key as parameters and returns the plain text.

Routing:

After the encryption service generates the cipher text, it passes it to the routing mechanism of the application. The routing mechanism of the application has three jobs to do:

1. Crafting the network packet.
2. Searching for the recipient in its routing table.
3. Forwarding the packet to its successor node(s).

Along with these jobs, the mechanism constantly updates its routing table by scanning for new Bluetooth devices in its range.

The structure of the network packet that is used in this mechanism is described as follows:

Destination address + Cipher text + Sent time + Time to live

The destination address part of the packet is the Bluetooth address of the recipient. If the address is not available in the current device's routing table, it means the recipient is not the range of the device. The device then floods the packet to all the devices in its routing table.

The cipher text part of the packet contains the actual content of the message in its encrypted format.

In a routing algorithm where flooding is involved, it is customary to include the packet's time to live. This ensures that the network is not clogged by duplicate copies of a packet that could not reach its destination. Every device that the packet arrives at checks the sent time and time to live of the packet. If the sum of the sent time and time to live exceeds the current time, it means that the packet's lifetime has expired. The packet is then discarded. If the packet still has time left, it is again forwarded to the successor nodes of the network.

When a packet reaches its destination, the routing mechanism extracts the cipher text from the packet and forwards it to encryption service of the application.

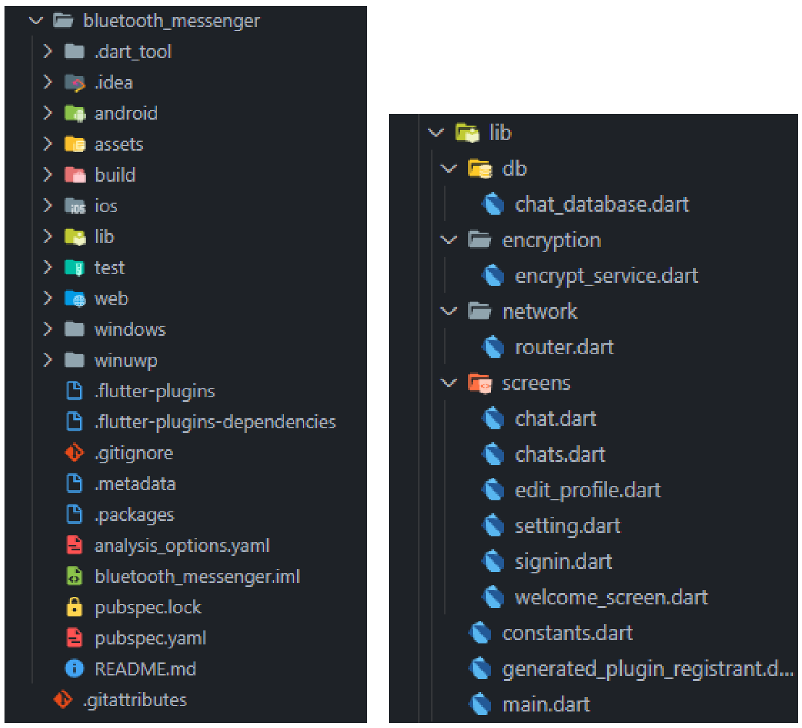
CHAPTER - 6
EXECUTION PROCEDURE AND TESTING

The application is developed with the Flutter toolkit and is programmed in Dart programming language. Both the frontend and backend of the application is written in Dart language. The details the environment in which we developed the application is as follows:

- Visual Studio code with Flutter and Dart plugins as the text editor.
- Android emulator provided by the Android Studio SDK (Pixel 4XL API 30 running Android 11.0).

To run the application on an emulator, we have to create a virtual android device. This can be done with the help of AVD manager in Android Studio. We chose to create a virtual android device of the real-world smartphone Pixel 4XL.

To run the application for testing, we need to open the root directory of the flutter project in VS code. The editor must also be able to detect the presence of any created virtual devices on the computer. It provides a list of available virtual devices for us to choose one from. Once we choose a virtual device, it is booted and connected to the editor. The directory structure of the project is as mentioned in Figure 6.1.



```
bluetooth_messenger
  .dart_tool
  .idea
  android
  assets
  build
  ios
  lib
  test
  web
  windows
  winuwp
  .flutter-plugins
  .flutter-plugins-dependencies
  .gitignore
  .metadata
  .packages
  analysis_options.yaml
  bluetooth_messenger.iml
  pubspec.lock
  pubspec.yaml
  README.md
.gitattributes
```

```
lib
  db
    chat_database.dart
  encryption
    encrypt_service.dart
  network
    router.dart
  screens
    chat.dart
    chats.dart
    edit_profile.dart
    setting.dart
    signin.dart
    welcome_screen.dart
  constants.dart
  generated_plugin_registrant.d...
  main.dart
```

Figure 6.1 Directory structure

The main code of the application is in the /lib/ directory in the root directory of the flutter project. The file named *'main.dart'* is where the application starts from. Debugging this file runs the application on the virtual device in debug mode. We can use the application in the same way one would do on a real smartphone.

Testing:

The entire application was thematically divides into 4 modules:

- Front-end
- Database
- Encryption Service
- Routing

Each of these modules were tested individually to verify if they work as intended. The testing was done by running the app on the Android Virtual Device.

The Front-end module consists of the entire user interface of the applications. We tested the application on different screen sizes to ensure that the application UI is adaptive and looks as we have imagined it to be.

The Database module was tested by performing all the operations on the SQLite database and verifying if those operation have the effect that we desired, on the database.

Encryption Service was tested by building some test cases where plain text is passed to it and the output was the encrypted cipher text and the decrypted plain text.

The routing module was tested by sending dummy packets over the network and observing how they were distributed. We also tested if the packets were actually reaching the intended destination.

All these modules were integrated into the final application and were tested again, to check if they work as intended when integrated with the other modules.

CHAPTER 7
RESULTS & PERFORMANCE EVALUATION

The test we conducted on the application yielded the following results.

Front-end test result:

The UI of the application is responsive and simple to read. We ran the application on different screen sizes to check if the elements of the application are sized correctly and we found that the layout of the application fits as intended on every screen size.

Database test result:

The local database that was implemented for the application successfully created the relational model that was designed for the application. The operations on the database also returned expected results. The runtime of all of the operations is fairly quick. The initialization of the database during the application startup took a bit longer than estimated, but this issue is not experienced in the consequent application launches.

Encryption test result:

The RSA encryption algorithm is known to be a slow encryption algorithm. The test we conducted observed the time it takes to generate the RSA key pair and also the time it takes to encrypt and decrypt text. The generation of the key pair takes a significant amount of time, but since it happens only one time for every user, we decided to not optimize it. The time to encrypt and decrypt texts is insignificant.

Routing test result:

The test we conducted on the routing mechanism verifies if the packets reach their destination, the time it takes for the packet to reach its destination, and how these factors change with the distance the communicators.

The results that were observed in the test are tabulated in Table 7.1. We observed that when under 10 m, the message is transmitted directly from the sender device to receiver device, but for a distance greater than 10m, a third device had to act as the middle device. The inclusion of a third device increases the processing time two-fold.

Distance	**Time**	**Reachability**
1 m	500 ms	Yes
2 m	600 ms	Yes
5 m	800 ms	Yes
10 m	1000 ms	Yes
11 m	1500 ms	Yes
12 m	1600 ms	Yes
15 m	1800 ms	Yes
20 m	2000 ms	Yes

Table 7.1 Routing test result

CHAPTER 8
CONCLUSION AND FUTURE WORK

Conclusion:

This application was built based on the experiences and situations faced by us and also our friends in the real time while sending messages and thought as providing solution to many of the people facing this problem like us and that extends providing communication at the time of natural calamities, providing communication in the remote places, providing communication in the closed places. So, we had built a complete ideology to solve this problem and built this application on it. This application has been built using flutter framework and Dart Programming Language in the view of extending it to the iOS in future. This application uses QR code to get the details of the receiver and save their chatid and uses Distance Vector routing algorithm for the routing of messages between the devices. This application uses RSA encryption algorithm to make messages secured using encryption and decryption mechanisms. This application would be widely useful for all the people and make them overcome the problem faced while transmitting messages.

Future Work:

- Basically, for now this application was built only for Android and in future we have a plan to extend this to iOS.
- We have a plan of Deploying This application into messaging platforms which run through the internet like WhatsApp, Facebook messenger, Instagram messenger.
- Many of us may have faced this situation that when we are using the internet based applications we may have lost internet connections and less bandwidth and all and felt

irritated when that is an important message to be sent.

- So, this becomes Handy when our API is deployed in the internet-based messenger applications such as the person can send the message through Bluetooth messenger API and when it reaches the other person and if he has a strong internet connection then the message has to follow its way through the internet to reach the receiver.
- Yes, it is the plan of interchanging the networks based on the situation and availability.
- This is under the process of feasibility study and we are researching whether it is possible to interchange the networks based on the situation.
- This needs a lot of background work and if it is successfully built, it would help a lot of people and also overcomes many of the drawbacks of the proposed application such as speed of message transfer.

APPENDIX

Source Code:

main.dart

```
import 'package:bluetooth_messenger/constants.dart';
import
'package:bluetooth_messenger/screens/welcome_screen.dart'
;
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
void main() {
  WidgetsFlutterBinding.ensureInitialized();
  Firebase.initializeApp();
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: CustomTheme.dark,
      title: 'Bluetooth Messenger',
      debugShowCheckedModeBanner: false,
      home: WelcomeScreen(context: context),
    );
  }
}
```

constants.dart

```
import 'package:flutter/material.dart';
const primaryColor = Color(0xffAB20FD);
const primaryColorAccent = Color(0xff7D12FF);
const secondaryColor = Color(0xff212121);
const secondaryColorAccent = Color(0xff323232);
const errorColor = Color(0xffff9494);
class CustomTheme {
  static ThemeData get dark {
    return ThemeData(
      primaryColor: primaryColor,
      scaffoldBackgroundColor: secondaryColor,
      fontFamily: 'Ubuntu',
      textTheme: const TextTheme(
        bodyText1: TextStyle(),
        bodyText2: TextStyle(),
        headline1: TextStyle(),
        headline2: TextStyle(),
        headline3: TextStyle(),
        headline4: TextStyle(),
        headline5: TextStyle(),
        headline6: TextStyle(),
      ).apply(
        bodyColor: primaryColor,
        displayColor: primaryColor,
      ),
      dividerTheme: const DividerThemeData(
        color: secondaryColor,
      ),
    );
  }
}
const String persons = 'persons'
const String messages = 'messages';
class PersonFields {
  static final List<String> values = [
    id,
    number,
    username,
    displayPicture,
  ];
  static const String id = 'id';
  static const String number = 'number';
  static const String username = 'username';
  static const String displayPicture = 'displayPicture';
}
class MessageFields {
  static final List<String> values = [
    id,
    receipient,
    content,
    time,
    isSender,
  ];
  static const String id = 'id';
  static const String receipient = 'receipient';
  static const String content = 'content';
  static const String time = 'time';
  static const String isSender = 'isSender';
}
class Person {
  final int? id;
  final int number;
  final String username;
  final String displayPicture;
  const Person({
    this.id,
    required this.number,
    required this.username,
    required this.displayPicture,
  });
  Map<String, Object?> toJson() => {
        PersonFields.id: id,
        PersonFields.number: number,
        PersonFields.username: username,
        PersonFields.displayPicture: displayPicture,
      };
  static Person fromJson(Map<String, Object?> json) =>
Person(
        id: json[PersonFields.id] as int,
        number: json[PersonFields.number] as int,
        username: json[PersonFields.username] as String,
        displayPicture: json[PersonFields.displayPicture]
as String,
      );
  Person copy({
    int? id,
    int? number,
    String? username,
    String? displayPicture,
    String? lastMessage,
  }) =>
      Person(
        id: id ?? this.id,
        number: number ?? this.number,
        username: username ?? this.username,
        displayPicture: displayPicture ??
this.displayPicture,
      );
}
class ChatMessage {
  final int? id;
  final int? receipient;
  final String content;
  final String time;
  final bool isSender;
  const ChatMessage({
    this.id,
    required this.receipient,
    required this.content,
    required this.time,
    required this.isSender,
  });
  Map<String, Object?> toJson() => {
        MessageFields.id: id,
        MessageFields.receipient: receipient,
        MessageFields.content: content,
        MessageFields.time: time,
        MessageFields.isSender: isSender ? 1 : 0,
      };
  static ChatMessage fromJson(Map<String, Object?> json)
=> ChatMessage(
        id: json[MessageFields.id] as int,
        receipient: json[MessageFields.receipient] as
```

```
int,
        content: json[MessageFields.content] as String,
        time: json[MessageFields.time] as String,
        isSender: json[MessageFields.isSender] == 1,
      );
  ChatMessage copy({
    int? id,
    int? receipient,
    String? content,
    String? time,
    bool? isSender,
  }) =>
      ChatMessage(
        id: id ?? this.id,
        receipient: receipient ?? this.receipient,
        content: content ?? this.content,
        time: time ?? this.time,
        isSender: isSender ?? this.isSender,
      );
}
```

welcome_screen.dart

```
import 'dart:async';
import 'package:bluetooth_messenger/screens/chats.dart';
import 'package:bluetooth_messenger/constants.dart';
import 'package:bluetooth_messenger/screens/signin.dart';
import 'package:flutter/material.dart';
import
'package:permission_handler/permission_handler.dart';
import
'package:shared_preferences/shared_preferences.dart';
class WelcomeScreen extends StatefulWidget {
  final BuildContext context;
  const WelcomeScreen({Key? key, required this.context})
: super(key: key);
  @override
  _WelcomeScreenState createState() =>
```

```
_WelcomeScreenState();
}
class _WelcomeScreenState extends State<WelcomeScreen> {
  bool flag = false;
  bool contactPermissionGranted = false;
  @override
  void initState() {
    getLoginState();
    checkContactStatus();
    Timer(
      const Duration(seconds: 3),
      () => flag
          ? Navigator.pushReplacement(
              context,
              MaterialPageRoute(
                builder: (context) => const
ChatsScreen(),
              ),
            )
          : Navigator.pushReplacement(
              context,
              MaterialPageRoute(
                builder: (context) => const
SignInScreen(),
              ),
            ),
    );
  }
  void checkContactStatus() async {
    final prefs = await SharedPreferences.getInstance();
    if (prefs.getBool('contactPermission') ?? false) {
      contactPermissionGranted = true;
    } else {
      final PermissionStatus contactPermissionStatus =
await _getPermission();
      if (contactPermissionStatus ==
PermissionStatus.granted) {
        prefs.setBool('contactPermission', true);
        contactPermissionGranted = true;
      }
    }
  }
  Future<PermissionStatus> _getPermission() async {
    final PermissionStatus contactPermission =
        await Permission.contacts.request();
    if (contactPermission == PermissionStatus.granted) {
      return contactPermission;
    } else if (contactPermission !=
PermissionStatus.denied) {
      showDialog(
        context: super.context,
        builder: (BuildContext context) {
          return const SimpleDialog(
            backgroundColor: secondaryColor,
            title: Text(
              "Permission for contacts denied",
              style: TextStyle(
                color: primaryColorAccent,
              ),
            ),
          );
        },
      );
      return contactPermission;
    } else {
      showDialog(
        context: super.context,
        builder: (BuildContext context) {
          return const SimpleDialog(
            backgroundColor: secondaryColor,
            title: Text(
              "Allow access to contacts in settings
page",
              style: TextStyle(
                color: primaryColorAccent,
              ),
            ),
          );
        },
      );
      return contactPermission;
    }
  }
  void getLoginState() async {
    final prefs = await SharedPreferences.getInstance();
    bool loggedIn = prefs.getBool('LoggedIn') ?? false;
    flag = loggedIn;
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Center(
          child: Column(
            children: [
              const Spacer(
                flex: 1,
              ),
              Expanded(
                  flex: 4,
                  child: Column(
                    mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
                    children: [
                      const Spacer(),
                      const Text(
                        "Welcome",
                        style: TextStyle(
                          color: primaryColor,
                          fontSize: 80,
                          fontWeight: FontWeight.w300,
                        ),
                      ),
                      const Spacer(),
                      Row(
                        mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
                        children: const [
                          Icon(

Icons.bluetooth_connected_rounded,
                            color: primaryColor,
                            size: 35,
                          ),
                          Icon(
                            Icons.circle,
                            color: primaryColorAccent,
                            size: 8,
                          ),
                          Icon(
                            Icons.circle,
                            color: primaryColorAccent,
                            size: 8,
                          ),
                          Icon(
                            Icons.circle,
                            color: primaryColorAccent,
                            size: 8,
                          ),
                          Icon(
                            Icons.send_rounded,
                            color: primaryColor,
                            size: 35,
                          ),
                          Icon(
                            Icons.circle,
                            color: primaryColorAccent,
                            size: 8,
                          ),
                          Icon(
                            Icons.circle,
                            color: primaryColorAccent,
                            size: 8,
                          ),
                          Icon(
                            Icons.circle,
                            color: primaryColorAccent,
                            size: 8,
                          ),
                          Icon(

Icons.bluetooth_connected_rounded,
                            color: primaryColor,
                            size: 35,
                          ),
                        ],
                      )
                    ],
                  )),
              const Spacer(
                flex: 1,
              ),
              const Text(
                'Chat with anyone in your Bluetooth
network',
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontSize: 16,
                  color: primaryColorAccent,
                ),
              ),
              const Spacer(
                flex: 1,
              ),
              const Spacer(
                flex: 1,
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

signin.dart

```
import 'package:bluetooth_messenger/constants.dart';
import 'package:bluetooth_messenger/screens/chats.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import
'package:shared_preferences/shared_preferences.dart';



class SignInScreen extends StatefulWidget {
  const SignInScreen({Key? key}) : super(key: key);

  @override
  SignInScreenState createState() => SignInScreenState();
}

class SignInScreenState extends State<SignInScreen> {
  final phoneNumberRetriever = TextEditingController();
  final OTPRetriever = TextEditingController();
  String errorText = "";
  Color textFieldColor = primaryColor;
  String? verificationIDReceiver;
  bool OTPsent = false;
  String buttonText = "Submit";
  String? phoneNumber;

  @override
  void initState() {
    super.initState();
  }
  @override
  Widget build(BuildContext context) {
    final double screenWidth =
MediaQuery.of(context).size.width;
    final double screenHeight =
MediaQuery.of(context).size.height;
    return Scaffold(
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal:
20, vertical: 0),
          child: ListView(
            physics: const BouncingScrollPhysics(),
            children: [
              Divider(
                height: screenHeight / 10,
              ),
              const Text(
                'Get started',
                style: TextStyle(
                  fontSize: 50,
                  color: primaryColor,
                  fontWeight: FontWeight.w300,
                ),
                textAlign: TextAlign.center,
              ),
              Divider(
                height: screenHeight / 10,
              ),
              TextField(
                controller: phoneNumberRetriever,
                decoration: InputDecoration(
                  enabledBorder: OutlineInputBorder(
                    borderSide: BorderSide(color:
textFieldColor, width: 1.0),
                  ),
                  focusedBorder: OutlineInputBorder(
                    borderSide: BorderSide(color:
```

```
textFieldColor, width: 1.0),
                  ),
                  hintText: 'Phone number',
                  hintStyle: const TextStyle(
                    color: primaryColorAccent,
                    fontSize: 12,
                  ),
                ),
                style: const TextStyle(
                  color: primaryColor,
                ),
                keyboardType: TextInputType.number,
              ),
              Text(
                errorText,
                style: const TextStyle(
                  color: errorColor,
                  fontSize: 14,
                ),
              ),
              //try {
              Divider(
                height: screenHeight / 50,
              ),
              Visibility(
                visible: OTPsent,
                child: TextField(
                  maxLength: 6,
                  controller: OTPRetriever,
                  decoration: InputDecoration(
                    enabledBorder: OutlineInputBorder(
                      borderSide: BorderSide(color:
textFieldColor, width: 1.0),
                    ),
                    focusedBorder: OutlineInputBorder(
                        borderSide:
                            BorderSide(color:
textFieldColor, width: 1.0)),
                    hintText: 'Enter OTP',
                    hintStyle: const TextStyle(
                      color: primaryColorAccent,
                      fontSize: 12,
                    ),
                  ),
                  style: const TextStyle(
                    color: primaryColor,
                  ),
                  keyboardType: TextInputType.number,
                ),
              ),

              Divider(
                height: screenHeight * 3 / 10,
              ),
              Container(
                padding: const EdgeInsets.all(5),
                width: screenWidth / 4,
                height: screenHeight / 15,
                alignment: Alignment.center,
                decoration: BoxDecoration(
                  border: Border.all(
                    color: secondaryColor,
                  ),
                  borderRadius:

BorderRadius.all(Radius.circular(screenHeight / 30)),
                  color: primaryColor,
                ),
                child: TextButton(
                  onPressed: () async {
                    if (buttonText == "Submit") {
                      setErrorText();
                    } else if (buttonText == "Verify") {
                      await FirebaseAuth.instance

.signInWithCredential(PhoneAuthProvider.credential(
                              verificationId:
verificationIDReceiver!,
                              smsCode:
OTPRetriever.text))
                          .then((value) async {
                        if (value.user != null) {
                          setLoginState();
                          Navigator.pushAndRemoveUntil(
                              context,
                              MaterialPageRoute(
                                  builder: (context) =>
const ChatsScreen()),
                              (route) => false);
                        }
                      });
                    }
                  },
                  child: Text(
                    buttonText,
                    style: const TextStyle(
                      fontSize: 18,
                      color: secondaryColor,
                    ),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }

  void setErrorText() {
    if (phoneNumberRetriever.text == "" ||
        phoneNumberRetriever.text.length != 10) {
      setState(() {
        errorText = "Please enter a valid Phone number";
        textFieldColor = errorColor;
      });
    } else {
      setState(() {
        phoneNumber = "+91" + phoneNumberRetriever.text;
        errorText = "";
        textFieldColor = primaryColor;
        OTPsent = true;
        buttonText = "Verify";
        verifyPhone();
      });
    }
  }

  verifyPhone() async {
    await FirebaseAuth.instance.verifyPhoneNumber(
      phoneNumber: phoneNumber!,
      codeSent: (String verificationID, int? resendToken)
{
        setState(() {
          verificationIDReceiver = verificationID;
        });
      },
      codeAutoRetrievalTimeout: (String verificationId) {
        setState(() {
          verificationIDReceiver = verificationId;
        });
      },
      verificationCompleted: (PhoneAuthCredential
phoneAuthCredential) async {
        await FirebaseAuth.instance
            .signInWithCredential(phoneAuthCredential)
            .then((value) async {
          if (value.user != null) {
            setLoginState();
            Navigator.pushAndRemoveUntil(
                context,
                MaterialPageRoute(builder: (context) =>
const ChatsScreen()),
                (route) => false);
          }
        });
      },
      verificationFailed: (FirebaseAuthException error) {
        print(error.message);
      },
    );
  }

  setLoginState() async {
    final prefs = await SharedPreferences.getInstance();
    prefs.setBool('LoggedIn', true);
  }
}
```

chats.dart

```
import 'dart:async';

import 'package:bluetooth_messenger/constants.dart';
import
'package:bluetooth_messenger/db/chat_database.dart';
import 'package:bluetooth_messenger/screens/chat.dart';
import
'package:bluetooth_messenger/screens/edit_profile.dart';
import 'package:contacts_service/contacts_service.dart';
import 'package:flutter/material.dart';
import
'package:shared_preferences/shared_preferences.dart';


const chatRoute = './chat';

class ChatsScreen extends StatelessWidget {
  const ChatsScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final double screenWidth =
MediaQuery.of(context).size.width;
    final double screenHeight =
MediaQuery.of(context).size.height;
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        backgroundColor: primaryColorAccent,
        actions: [
          IconButton(
            onPressed: () {},
            icon: const Icon(Icons.search_rounded),
            color: secondaryColor,
          ),
          PopupMenuButton(
            icon: const Icon(
              Icons.more_vert_rounded,
              color: secondaryColor,
            ),
            shape: const OutlineInputBorder(
              borderRadius:
BorderRadius.all(Radius.circular(6)),
            ),
            color: secondaryColor,
            itemBuilder: (context) => [
              const PopupMenuItem(
                child: Text(
                  "Settings",
                  style: TextStyle(
                    color: Colors.white,
                  ),
                ),
                value: 0,
              ),
            ],
            onSelected: (result) {
              if (result == 0) {
                Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => const
EditProfilePage()));
              }
            },
          ),
        ],
        title: const Text(
          "Chats",
          style: TextStyle(
            color: secondaryColor,
          ),
        ),
      ),
      body: Chats(screenWidth, screenHeight),
    );
  }
}

class Chats extends StatefulWidget {
  final double screenWidth;
  final double screenHeight;

  const Chats(this.screenWidth, this.screenHeight, {Key?
key})
      : super(key: key);

  @override
  _ChatsState createState() => _ChatsState(screenWidth,
screenHeight);
}

class _ChatsState extends State<Chats> {
  final double screenWidth;
  final double screenHeight;
  late List<Person> chats;
  Iterable<Contact> _contactsList = [];
  List<String> lastMessages = [];
  bool isLoading = false;
  bool flag = false;

  _ChatsState(this.screenWidth, this.screenHeight);

  @override
  void initState() {
    super.initState();
    checkContacts();
    refreshChats();
  }

  getContacts() async {
    final Iterable<Contact> contactsList = await
ContactsService.getContacts();
    _contactsList = contactsList;
    if (_contactsList.isEmpty) {
      print("empty");
    }
  }

  Future refreshChats() async {
    setState(() => isLoading = true);

    chats = await ChatDatabase.instance.readAllPersons();
    lastMessages = [];
    if (chats.isNotEmpty) {
      for (int i = 0; i < chats.length; i++) {
        lastMessages
            .add(await
ChatDatabase.instance.getLastMessage(chats[i].id) ?? "");
      }
    }
    setState(() => isLoading = false);
  }
  void checkContacts() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() {
      flag = prefs.getBool('contactPermission') ?? false;
    });
  }

  @override
  Widget build(BuildContext context) {
    return isLoading
        ? const Center(
            child: CircularProgressIndicator(
              color: primaryColorAccent,
              strokeWidth: 3,
            ),
          )
        : (chats.isEmpty)
            ? Scaffold(
                body: const Center(
                  child: Text(
                    "No chats to display",
                    style: TextStyle(
                      color: secondaryColorAccent,
                    ),
                  ),
                ),
                floatingActionButton:
FloatingActionButton(
                  onPressed: () => showDialog(
                    context: context,
                    builder: (BuildContext context) {
                      return SimpleDialog(
                        backgroundColor: secondaryColor,
                        contentPadding: const
EdgeInsets.all(20),
                        shape: const
RoundedRectangleBorder(
                          borderRadius: BorderRadius.all(
                            Radius.circular(20),
                          ),
                        ),
                        children: [
                          flag
                              ? SizedBox(
                                  width:
double.maxFinite,
                                  height:
double.maxFinite,
                                  child:
ListView.builder(
                                    itemCount:
_contactsList.length,
                                    itemBuilder:
                                        (BuildContext
context, int index) {
                                      Contact contact =

_contactsList.elementAt(index);
                                      return ListTile(
                                        contentPadding:
                                            const
EdgeInsets.symmetric(
                                                vertical:
2, horizontal: 10),
                                        leading:
(contact.avatar != null &&

contact.avatar!.isNotEmpty)
                                            ?
CircleAvatar(

backgroundImage: MemoryImage(

contact.avatar!),
                                              )
                                            :
CircleAvatar(
                                                child:
Text(contact.initials()),

backgroundColor:

primaryColorAccent,
                                              ),
                                      );
                                    },
                                  ),
                                )
                              : const Center(
                                  child:
CircularProgressIndicator(
                                    color:
primaryColorAccent,
                                    strokeWidth: 3,
                                  ),
                                ),
                        ],
                      );
                    },
                  ),
                  backgroundColor: primaryColor,
                  child: const Icon(
                    Icons.add,
                    color: secondaryColor,
                  ),
                ),
              )
            : Scaffold(
                body: ListView.builder(
                  itemCount: chats.length,
                  itemExtent: 80,
                  itemBuilder: (BuildContext context, int
index) {
```

```
                    String username =
chats[index].username;
                    String displayPicture =
chats[index].displayPicture;
                    return Container(
                      padding: const EdgeInsets.only(top:
3),
                      child: ListTile(
                        leading: CircleAvatar(
                          maxRadius: 30,
                          backgroundImage: AssetImage(
                            displayPicture,
                          ),
                        ),
                        title: Text(
                          username,
                          style: const TextStyle(color:
Colors.white),
                        ),
                        subtitle: Text(
                          lastMessages[index],
                          style: const TextStyle(color:
Colors.white60),
                        ),
                        onTap: () {
                          Navigator.push(
                            context,
                            MaterialPageRoute(
                                builder: (context) =>
ChatScreen(
                                    chats[index],
screenWidth, screenHeight)),
                          ).then(update);
                        },
                      ),
                      decoration: const BoxDecoration(
                        border: Border(
                          bottom: BorderSide(color:
```

```
secondaryColorAccent),
                       ),
                     ),
                   );
                 },
               ),
               floatingActionButton:
FloatingActionButton(
                 onPressed: () => showDialog(
                   context: context,
                   builder: (BuildContext context) {
                     return SimpleDialog(
                       backgroundColor: secondaryColor,
                       contentPadding: const
EdgeInsets.all(20),
                       shape: const
RoundedRectangleBorder(
                         borderRadius: BorderRadius.all(
                           Radius.circular(20),
                         ),
                       ),
                       children: [
                         flag
                             ? SizedBox(
                                 width:
double.maxFinite,
                                 height:
double.maxFinite,
                                 child:
_contactsList.isEmpty
                                     ? const Center(
                                         child: Text(
                                           "No contacts
to display",
                                           style:
TextStyle(
                                             color:
secondaryColorAccent,
                                                   ),
                                                 ),
                                               )
                                             : ListView.builder(
                                                 itemCount:
_contactsList.length,
                                                 itemBuilder:
(BuildContext context,
                                                     int index)
{
                                                   Contact
contact =

_contactsList.elementAt(index);
                                                   return
ListTile(

contentPadding:
                                                         const
EdgeInsets.symmetric(

vertical: 2,

horizontal: 10),
                                                     leading:
(contact.avatar !=

null &&

contact

.avatar!.isNotEmpty)
                                                         ?
CircleAvatar(

backgroundImage:

MemoryImage(
contact.avatar!),
                                                                    )
                                                                  :
CircleAvatar(

child: Text(

contact.initials()),

backgroundColor:

primaryColorAccent,
                                                                    ),
                                                            );
                                                          },
                                                        ),
                                                )
                                              : const SizedBox(
                                                  width:
double.maxFinite,
                                                  height:
double.maxFinite,
                                                  child: Center(
                                                    child:
CircularProgressIndicator(
                                                      color:
primaryColorAccent,
                                                      strokeWidth: 3,
                                                    ),
                                                  ),
                                                ),
                                        ],
                                      );
                                    },
                                  ),
                                  backgroundColor: primaryColor,
                                  child: const Icon(
                    Icons.add,
                    color: secondaryColor,
                  ),
                ),
              );
  }

  void addPerson() {
    ChatDatabase.instance.createPerson(const Person(
      number: 123,
      username: 'Mitsuha',
      displayPicture: 'assets/images/mitsuha.png',
    ));
    refreshChats();
  }

  FutureOr update(dynamic value) {
    refreshChats();
  }
}
```

chat.dart

```
import 'dart:async';

import
'package:bluetooth_messenger/db/chat_database.dart';
import
'package:bluetooth_messenger/screens/edit_profile.dart';
import 'package:flutter/material.dart';
import 'package:bluetooth_messenger/constants.dart';

class ChatScreen extends StatelessWidget {
  final Person user;
  final double screenWidth;
  final double screenHeight;

  const ChatScreen(this.user, this.screenWidth,
this.screenHeight, {Key? key})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        backgroundColor: primaryColorAccent,
        leading: IconButton(
          icon: const Icon(
            Icons.arrow_back_ios_rounded,
            color: secondaryColor,
          ),
          onPressed: () => Navigator.of(context).pop(),
        ),
        title: Row(
          children: [
            Expanded(
              child: InkWell(
                onTap: () {},
                child: Row(
                  children: [
                    CircleAvatar(
                      backgroundImage: AssetImage(
                        user.displayPicture,
                      ),
                    ),
                    SizedBox(
                      width: screenWidth / 35,
                    ),
                    Expanded(
                      child: Column(
                        crossAxisAlignment:
CrossAxisAlignment.start,
                        children: [
                          Text(
                            user.username,
                            style: TextStyle(
                              color: secondaryColor,
                              fontSize: screenHeight /
42,
                              fontWeight:
FontWeight.w600,
                            ),
                          ),
                        ],
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
        actions: [
          PopupMenuButton(
            icon: const Icon(
              Icons.more_vert_rounded,
              color: secondaryColor,
            ),
            shape: const OutlineInputBorder(
              borderRadius:
BorderRadius.all(Radius.circular(6)),
            ),
            color: secondaryColorAccent,
            itemBuilder: (context) => [
              const PopupMenuItem(
                value: 0,
                child: Text(
                  "Settings",
                  style: TextStyle(
                    color: Colors.white,
                  ),
                ),
              ),
            ],
            onSelected: (result) {
              if (result == 0) {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context)
=> EditProfilePage()));
              }
            },
          ),
        ],
      ),
      body: SafeArea(
        child: _Messages(screenWidth, screenHeight,
user),
      ),
    );
  }

  void openProfile(BuildContext context) {
    Navigator.push(
        context, MaterialPageRoute(builder: (context) =>
EditProfilePage()));
  }
}

class _Messages extends StatefulWidget {
  final double screenWidth;
  final double screenHeight;
  final Person user;

  const _Messages(this.screenWidth, this.screenHeight,
this.user, {Key? key})
      : super(key: key);

  @override
  _MessagesState createState() => _MessagesState();
}
```

```
class _MessagesState extends State<_Messages> {
  final messenger = TextEditingController();
  late List<ChatMessage> messages;
  bool isLoading = false;

  @override
  void initState() {
    super.initState();
    refreshMessages();
  }

  Future refreshMessages() async {
    setState(() => isLoading = true);
    messages = await
ChatDatabase.instance.readMessages(widget.user.id);

    setState(() => isLoading = false);
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Expanded(
          child: isLoading
              ? const Center(
                  child: CircularProgressIndicator(
                    color: primaryColorAccent,
                    strokeWidth: 3,
                  ),
                )
              : ListView.builder(
                  padding: const EdgeInsets.all(15),
                  itemCount: messages.length,
                  itemBuilder: (BuildContext context, int
index) =>
                      messageBuilder(context, index),
                ),
        ),
        inputField(),
      ],
    );
  }

  Widget inputField() {
    return Container(
      color: secondaryColorAccent,
      child: SafeArea(
        child: Row(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Expanded(
              flex: 7,
              child: Padding(
                padding: const EdgeInsets.all(10),
                child: TextField(
                  controller: messenger,
                  style: const TextStyle(
                    color: Colors.white,
                  ),
                  decoration: InputDecoration(
                    hintText: 'Enter a Message',
                    hintStyle: const TextStyle(
                      color: Colors.white54,
                    ),
                    border: OutlineInputBorder(
                      borderSide: const BorderSide(
                        color: secondaryColor,
                      ),
                      borderRadius:
BorderRadius.circular(20),
                    ),
                    focusedBorder: OutlineInputBorder(
                      borderSide: const BorderSide(
                        color: primaryColor,
                      ),
                      borderRadius:
BorderRadius.circular(20),
                    ),
                  ),
                ),
              ),
            ),
            Expanded(
              flex: 1,
              child: IconButton(
                onPressed: post,
                padding: EdgeInsets.zero,
                icon: const Icon(
                  Icons.send_rounded,
                  size: 35,
                  color: primaryColorAccent,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }

  Widget messageBuilder(BuildContext context, int index)
{
    final message = messages[index];
    return Padding(
      padding: const EdgeInsets.all(5),
      child: InkWell(
        onLongPress: () {

ChatDatabase.instance.deleteMessage(message.id);
          refreshMessages();
        },
        child: Row(
          mainAxisAlignment: message.isSender
```

```
              ? MainAxisAlignment.end
              : MainAxisAlignment.start,
          children: [
            Container(
              padding: const EdgeInsets.all(15),
              constraints: BoxConstraints(maxWidth:
widget.screenWidth * 0.75),
              decoration: BoxDecoration(
                color: message.isSender
                    ? secondaryColorAccent
                    : primaryColorAccent,
                borderRadius: BorderRadius.circular(30),
              ),
              child: Text(
                message.content,
                style: const TextStyle(
                  color: Colors.white,
                  fontSize: 16,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }

  void post() {
    final String message = messenger.text;
    if (message == '') return;
    DateTime now = DateTime.now();
    ChatDatabase.instance.postMessage(ChatMessage(
      receipient: widget.user.id,
      content: message,
      time: "${now.hour}:${now.minute}",
      isSender: true,
    ));
    refreshMessages();
    setState(() {
      messenger.text = '';
    });
  }
}
```

setting.dart

```
import 'package:bluetooth_messenger/screens/signin.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import './signin.dart';

class SettingsPage extends StatefulWidget {
  const SettingsPage({Key? key}) : super(key: key);

  @override
  _SettingsPageState createState() =>
_SettingsPageState();
}

class _SettingsPageState extends State<SettingsPage> {
  @override
  Widget build(BuildContext context) {
    final double screenWidth =
MediaQuery.of(context).size.width;
    final double screenHeight =
MediaQuery.of(context).size.height;
    return Scaffold(
      appBar: AppBar(
        backgroundColor:
Theme.of(context).scaffoldBackgroundColor,
        elevation: 1,
        leading: IconButton(
          onPressed: () {
            Navigator.of(context).pop();
          },
          icon: const Icon(
            Icons.arrow_back,
            color: Colors.green,
          ),
        ),
      ),
      body: Container(
        padding: const EdgeInsets.only(left: 16, top: 25,
right: 16),
        child: ListView(
          children: [
            const Text(
              "Settings",
              style: TextStyle(fontSize: 25, fontWeight:
FontWeight.w500),
            ),
            const SizedBox(
              height: 40,
            ),
            Row(
              children: const [
                Icon(
                  Icons.person,
                  color: Colors.green,
                ),
                SizedBox(
                  width: 8,
                ),
                Text(
                  "Account",
                  style: TextStyle(fontSize: 18,
fontWeight: FontWeight.bold),
                ),
              ],
            ),
            const Divider(
              height: 15,
              thickness: 2,
            ),
            const SizedBox(
              height: 10,
            ),
            buildAccountOptionRow(context, "Change 
password"),
            buildAccountOptionRow(context, "Content 
settings"),
            buildAccountOptionRow(context, "Social"),
            buildAccountOptionRow(context, "Language"),
            buildAccountOptionRow(context, "Privacy and 
security"),
            const SizedBox(
              height: 40,
            ),
            Center(
              child: OutlineButton(
                padding: EdgeInsets.symmetric(horizontal: 
40),
                shape: RoundedRectangleBorder(
                    borderRadius: 
BorderRadius.circular(20)),
                onPressed: () {},
                // onPressed: () {
                //   
Navigator.of(context).push(MaterialPageRoute(
                //       builder: (BuildContext context) 
=>
                //           
SignIn(MediaQuery.of(context).size.width,
                //               
MediaQuery.of(context).size.height)));
                // }, getting error for this
                child: const Text("SIGN OUT",
                    style: TextStyle(
                        fontSize: 16, letterSpacing: 2.2, 
color: Colors.grey)),
              ),
            )
```

```
          ],
        ),
      ),
    );
  }

  GestureDetector buildAccountOptionRow(BuildContext
context, String title) {
    return GestureDetector(
      onTap: () {
        showDialog(
            context: context,
            builder: (BuildContext context) {
              return AlertDialog(
                title: Text(title),
                content: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: const [
                    Text("Option 1"),
                    Text("Option 2"),
                    Text("Option 3"),
                  ],
                ),
                actions: [
                  FlatButton(
                      onPressed: () {
                        Navigator.of(context).pop();
                      },
                      child: const Text("Close")),
                ],
              );
            });
      },
      child: Padding(
        padding: const EdgeInsets.symmetric(vertical:
8.0),
        child: Row(
          mainAxisAlignment:
```

```
MainAxisAlignment.spaceBetween,
          children: [
            Text(
              title,
              style: TextStyle(
                fontSize: 18,
                fontWeight: FontWeight.w500,
                color: Colors.grey[600],
              ),
            ),
            const Icon(
              Icons.arrow_forward_ios,
              color: Colors.grey,
            ),
          ],
        ),
      ),
    );
  }
}
```

edit_profile.dart

```
import 'package:bluetooth_messenger/constants.dart';
import 'package:flutter/material.dart';
import
'package:bluetooth_messenger/screens/setting.dart';

class EditProfilePage extends StatefulWidget {
  const EditProfilePage({Key? key}) : super(key: key);

  @override
  _EditProfilePageState createState() =>
_EditProfilePageState();
}

class _EditProfilePageState extends
State<EditProfilePage> {
  bool showPassword = false;
  @override
  Widget build(BuildContext context) {
    final double screenWidth =
MediaQuery.of(context).size.width;
    final double screenHeight =
MediaQuery.of(context).size.height;
    return Scaffold(
      appBar: AppBar(
        automaticallyImplyLeading: false,
        backgroundColor: primaryColorAccent,
        leading: IconButton(
          icon: const Icon(
            Icons.arrow_back_ios_rounded,
            color: secondaryColor,
          ),
          onPressed: () => Navigator.of(context).pop(),
        ),
        actions: [
          IconButton(
            icon: const Icon(
              Icons.settings,
              color: secondaryColor,
            ),
            onPressed: () {

Navigator.of(context).push(MaterialPageRoute(
                builder: (BuildContext context) =>
SettingsPage()));
            },
          ),
        ],
      ),
      body: Padding(
        padding: const EdgeInsets.only(left: 16, top: 25,
right: 16),
        child: GestureDetector(
          onTap: () {
            FocusScope.of(context).unfocus();
          },
          child: ListView(
            physics: const BouncingScrollPhysics(),
            children: [
              const Text(
                "Edit Profile",
                style: TextStyle(
                fontSize: 25,
                fontWeight: FontWeight.w500,
                color: primaryColorAccent,
                ),
              ),
              Divider(
                height: screenHeight / 40,
              ),
              Center(
                child: Stack(
                children: [
                  Container(
                    width: 130,
                    height: 130,
                    decoration: BoxDecoration(
                        border: Border.all(
                          width: 1,
                          color: primaryColorAccent,
                        ),
                        shape: BoxShape.circle,
                        image: const DecorationImage(
                            fit: BoxFit.cover,
                            image:
AssetImage('assets/images/taki.png'))),
                  ),
                  Positioned(
                    bottom: 0,
                    right: 0,
                    child: Container(
                      height: 40,
                    width: 40,
                    decoration: BoxDecoration(
                      shape: BoxShape.circle,
                      border: Border.all(
                        width: 2,
                        color:
Theme.of(context).scaffoldBackgroundColor,
                      ),
                      color: primaryColorAccent,
                    ),
                    child: Center(
                      child: IconButton(
                        enableFeedback: false,
                        splashRadius: 1,
                        icon: const Icon(Icons.edit),
                        color: secondaryColor,
                        onPressed: () {},
                      ),
                    ),
                  ),
                ),
                ],
              ),
            ),
            Divider(
              height: screenHeight / 40,
            ),
            buildTextField("Full Name", "Enter your
name", false),
            buildTextField("E-mail", "Enter your
Email", false),
            buildTextField(
                "About you", "Describe something about
you ", false),
            buildTextField("Pnone", "7672377623",
false),
            Divider(
              height: screenHeight / 40,
            ),
            Row(
              mainAxisAlignment:
MainAxisAlignment.spaceBetween,
              children: [
                MaterialButton(
                padding: const
EdgeInsets.symmetric(horizontal: 50),
                color: secondaryColorAccent,
                shape: RoundedRectangleBorder(
                  borderRadius:
BorderRadius.circular(20),
                ),
                onPressed: () =>
Navigator.of(context).pop(),
                child: const Text(
                  "Cancel",
                  style: TextStyle(fontSize: 14,
color: Colors.white),
                ),
                ),
                MaterialButton(
                onPressed: () {},
                color: primaryColorAccent,
                padding: const
EdgeInsets.symmetric(horizontal: 50),
                elevation: 2,
                shape: RoundedRectangleBorder(
                  borderRadius:
BorderRadius.circular(20),
                ),
                child: const Text(
                  "Save",
                  style: TextStyle(
                    fontSize: 14,
                    color: Colors.white,
                  ),
                ),
                )
              ],
            )
          ],
        ),
      ),
    ),
  );
}

Widget buildTextField(
    String labelText, String placeholder, bool
isPasswordTextField) {
  return Padding(
    padding: const EdgeInsets.fromLTRB(0, 15, 0, 15),
    child: TextField(
      obscureText: isPasswordTextField ? showPassword :
false,
      style: const TextStyle(
        color: primaryColor,
      ),
      decoration: InputDecoration(
        suffixIcon: isPasswordTextField
            ? IconButton(
                onPressed: () {
                setState(() {
                  showPassword = !showPassword;
                });
                },
                icon: const Icon(
                Icons.remove_red_eye,
                color: Colors.grey,
                ),
              )
            : null,
        enabledBorder: const OutlineInputBorder(
          borderSide: BorderSide(
            color: primaryColor,
            width: 1.0,
          ),
        ),
        focusedBorder: const OutlineInputBorder(
          borderSide: BorderSide(
            color: primaryColor,
            width: 1.0,
          ),
        ),
        labelText: labelText,
        labelStyle: const TextStyle(
          color: primaryColorAccent,
          fontSize: 16,
        ),
        floatingLabelBehavior:
FloatingLabelBehavior.never,
        hintText: placeholder,
        hintStyle: const TextStyle(
          color: primaryColorAccent,
          fontSize: 14,
        ),
      ),
    ),
  );
}
}
```

chat_database.dart

```
import 'package:path/path.dart';
import 'package:bluetooth_messenger/constants.dart';
import 'package:sqflite/sqflite.dart';

class ChatDatabase {
  static final ChatDatabase instance =
ChatDatabase._init();

  static Database? _database;

  ChatDatabase._init();
  Future<Database> get database async {
    if (_database != null) return _database!;

    _database = await _initDB('chats.db');
    return _database!;
  }

  Future<Database> _initDB(String filePath) async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, filePath);

    return await openDatabase(path, version: 1, onCreate:
_createDB);
  }

  Future _createDB(Database db, int version) async {
    const idType = "INTEGER PRIMARY KEY AUTOINCREMENT";
    const numberType = 'INTEGER NOT NULL';
    const textType = 'TEXT NOT NULL';
    const boolType = 'BOOLEAN NOT NULL';

    const reciepientType = 'INETGER NOT NULL';

    await db.execute('''
    CREATE TABLE $persons (
      ${PersonFields.id} $idType,
      ${PersonFields.number} $numberType,
      ${PersonFields.username} $textType,
      ${PersonFields.displayPicture} $textType

    )
    ''');

    await db.execute('''
    CREATE TABLE $messages (
      ${MessageFields.id} $idType,
      ${MessageFields.receipient} $reciepientType,
      ${MessageFields.content} $textType,
      ${MessageFields.time} $textType,
      ${MessageFields.isSender} $boolType,
      FOREIGN KEY(${MessageFields.receipient}) REFERENCES
$persons(${PersonFields.id})
    )
    ''');
  }

  //persons

  Future<Person> createPerson(Person person) async {
    final db = await instance.database;

    final id = await db.insert(persons, person.toJson());

    return person.copy(id: id);
  }

  Future<void> removePerson(int id) async {
    final db = await instance.database;

    deleteAllMessages(id);

    await db.delete(
      persons,
      where: '${PersonFields.id} = ?',
      whereArgs: [id],
    );
  }

  Future<Person?> readPerson(int number) async {
    final db = await instance.database;

    final maps = await db.query(
      persons,
      columns: PersonFields.values,
      where: '${PersonFields.number} = ?',
      whereArgs: [number],
    );

    if (maps.isNotEmpty) {
      return Person.fromJson(maps.first);
    } else {
      return null;
    }
  }

  Future<List<Person>> readAllPersons() async {
    final db = await instance.database;

    final result = await db.query(persons);

    return result.map((json) =>
Person.fromJson(json)).toList();
  }

  //messages

  Future<ChatMessage> postMessage(ChatMessage message)
async {
    final db = await instance.database;

    final id = await db.insert(messages,
message.toJson());

    return message.copy(id: id);
  }

  Future<void> deleteMessage(int? id) async {
    final db = await instance.database;

    await db.delete(
      messages,
      where: '${MessageFields.id} = ?',
      whereArgs: [id],
    );
  }

  Future<ChatMessage> readMessage(int? id) async {
    final db = await instance.database;

    final result = await db.query(
      messages,
      where: '${MessageFields.id} = ?',
      whereArgs: [id],
    );
    return ChatMessage.fromJson(result.first);
  }

  Future<List<ChatMessage>> readMessages(int? receipient)
async {
    final db = await instance.database;

    final result = await db.query(
      messages,
      where: '${MessageFields.receipient} = ?',
      whereArgs: [receipient],
    );

    return result.map((json) =>
ChatMessage.fromJson(json)).toList();
  }

  Future<String?> getLastMessage(int? receipient) async {
    final db = await instance.database;

    final lastMessage = await db.query(
      messages,
      orderBy: MessageFields.id,
      where: '${MessageFields.receipient} = ?',
      whereArgs: [receipient],
    );
    if (lastMessage.isNotEmpty) {
      return lastMessage.last[MessageFields.content] as
String;
    } else {
      return null;
    }
  }

  void deleteAllMessages(int receipient) async {
    final db = await instance.database;
    db.delete(
      messages,
      where: '${MessageFields.receipient} = ?',
      whereArgs: [receipient],
    );
  }

  Future close() async {
    final db = await instance.database;

    db.close();
  }
}
```

encrypt_service.dart

```
import 'dart:math';
import 'dart:typed_data';
import 'dart:convert';

import 'package:pointycastle/export.dart';
import "package:asn1lib/asn1lib.dart";

encodePublicKeyToPem(RSAPublicKey publicKey) {
  var algorithmSeq = ASN1Sequence();
  var algorithmAsn1Obj =
ASN1Object.fromBytes(Uint8List.fromList(
      [0x6, 0x9, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0xd, 0x1,
0x1, 0x1]));
  var paramsAsn1Obj =
ASN1Object.fromBytes(Uint8List.fromList([0x5, 0x0]));
  algorithmSeq.add(algorithmAsn1Obj);
  algorithmSeq.add(paramsAsn1Obj);

  var publicKeySeq = ASN1Sequence();
  publicKeySeq.add(ASN1Integer(publicKey.modulus));
  publicKeySeq.add(ASN1Integer(publicKey.exponent));
  var publicKeySeqBitString =

ASN1BitString(Uint8List.fromList(publicKeySeq.encodedByte
s));

  var topLevelSeq = ASN1Sequence();
  topLevelSeq.add(algorithmSeq);
  topLevelSeq.add(publicKeySeqBitString);
  var dataBase64 =
base64.encode(topLevelSeq.encodedBytes);

  return """-----BEGIN PUBLIC KEY-----
\r\n$dataBase64\r\n-----END PUBLIC KEY-----""";
}

String encrypt(String plaintext, RSAPublicKey publicKey)
{
  var cipher = RSAEngine()
    ..init(true,
PublicKeyParameter<RSAPublicKey>(publicKey));
  var cipherText =
cipher.process(Uint8List.fromList(plaintext.codeUnits));

  return String.fromCharCodes(cipherText);
}

String decrypt(String ciphertext, RSAPrivateKey
privateKey) {
  var cipher = RSAEngine()
```

```
    ..init(false,
PrivateKeyParameter<RSAPrivateKey>(privateKey));
  var decrypted =
cipher.process(Uint8List.fromList(ciphertext.codeUnits));

  return String.fromCharCodes(decrypted);
}

void main() {
  var keyParams =
RSAKeyGeneratorParameters(BigInt.from(65537), 2048, 5);

  var secureRandom = FortunaRandom();
  var random = Random.secure();

  List<int> seeds = [];
  for (int i = 0; i < 32; i++) {
    seeds.add(random.nextInt(255));
  }


secureRandom.seed(KeyParameter(Uint8List.fromList(seeds))
);

  //var rnd = new
FixedSecureRandom()..seed(KeyParameter(new
Uint8List.fromList(seeds)));

  var rngParams = ParametersWithRandom(keyParams,
secureRandom);
  var k = RSAKeyGenerator();
  k.init(rngParams);

  AsymmetricKeyPair<PublicKey, PrivateKey> keyPair =
k.generateKeyPair();
  RSAPrivateKey privateKey = keyPair.privateKey as
RSAPrivateKey;
  RSAPublicKey publicKey = keyPair.publicKey as
```

```
RSAPublicKey;

  var message = 'this message will be encrypted';

  var encryptedText = encrypt(message, publicKey);
  print(encryptedText);  var decryptedText =
decrypt(encryptedText, privateKey);
  print(decryptedText);
  print('end');
}
```

List of Abbreviations:

BLE - Bluetooth Low Energy

UHF - Ultra High Frequency

DVR - Distance Vector Routing

ARPANET - Advanced Research Projects Agency Network

RSA - Rivest, Shamir, and Adleman

List of Figures:

List of Tables:

Screenshots:

1. **Welcome Screen**

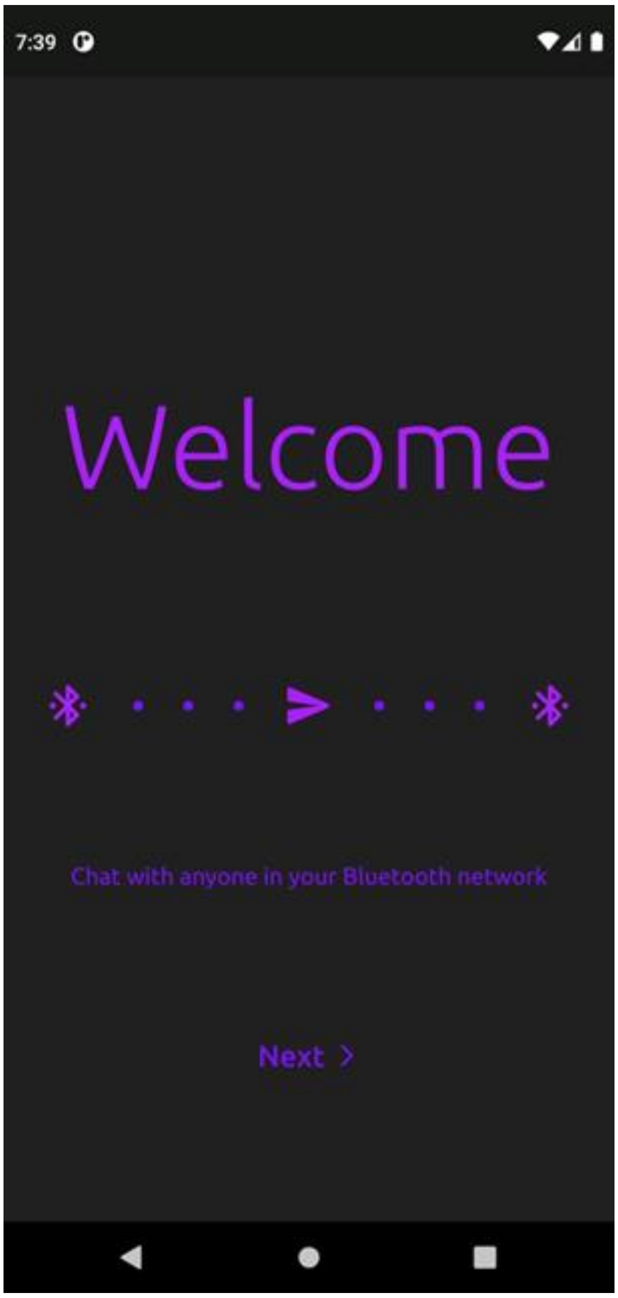


Figure A.1 Welcome Screen

2. Registration Screen

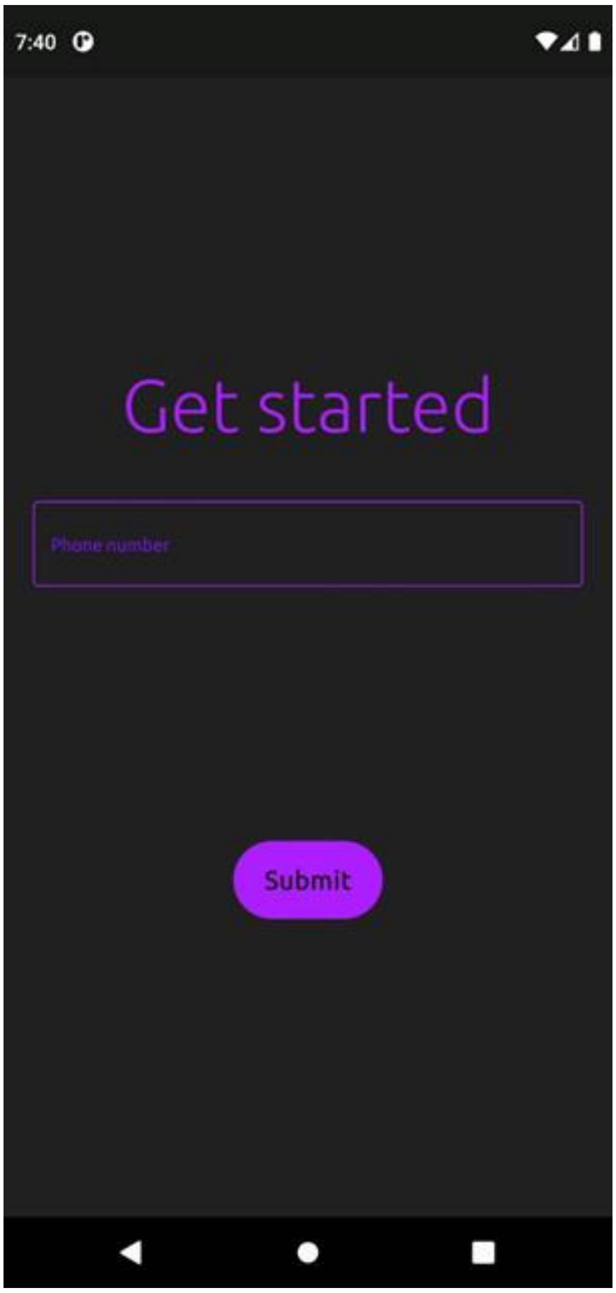


Figure A.2 Registration Screen

3. Conversations Screen

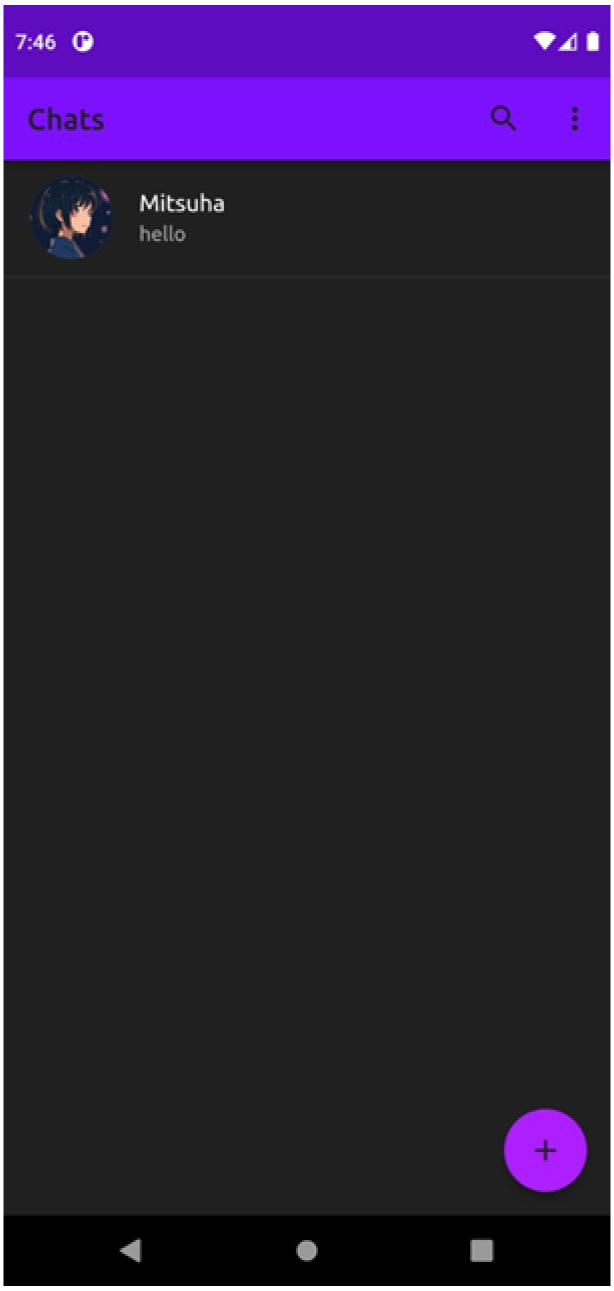


Figure A.3 Conversations Screen

4. Chat Screen

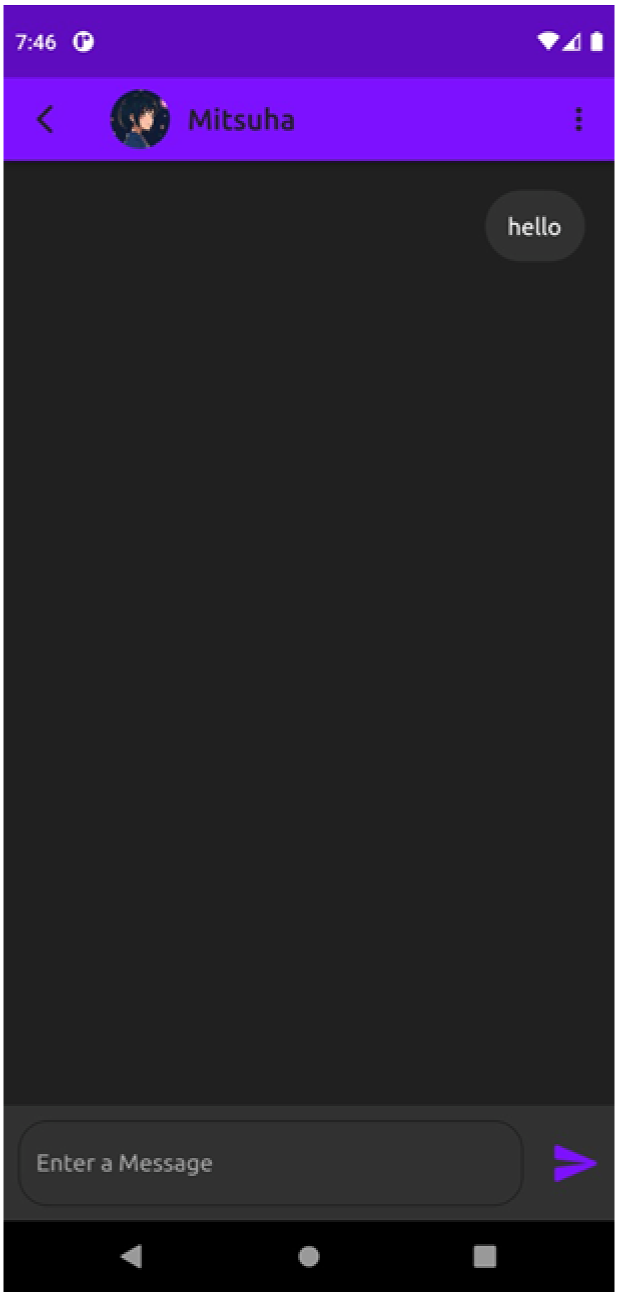
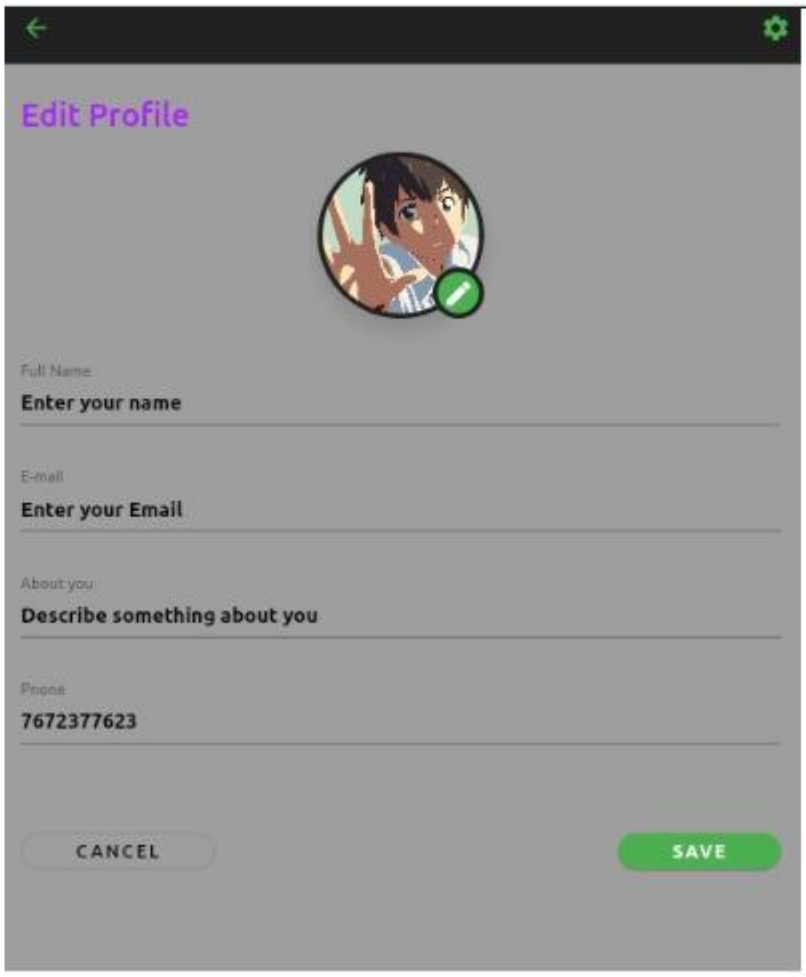


Figure A.4 Chat Screen

5. Edit Profile Screen


Edit Profile

Full Name
Enter your name

E-mail
Enter your Email

About you
Describe something about you

Phone
7672377623

Figure A.5 Edit Profile Screen

6. Settings Screen

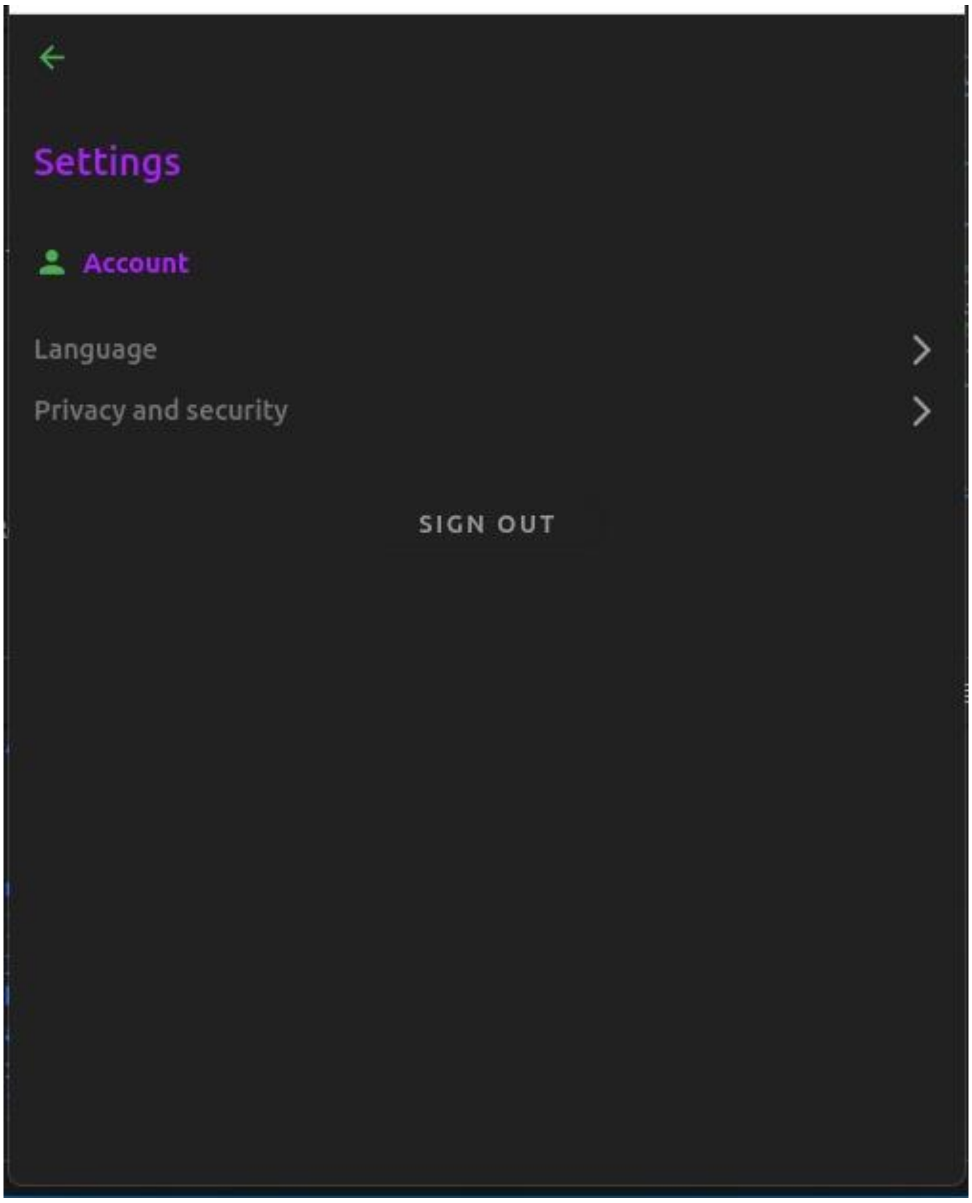


Figure A.6 Settings Screen

REFERENCES

1. W. Pan, F. Luo and L. Xu, "Research and design of chatting room system based on Android Bluetooth" *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, 2012, pp. 3390-3393, doi: 10.1109/CECNet.2012.6201484.

2. El-Seoud, Samir & Taj-Eddin, Islam. (2016). Developing an Android Mobile Bluetooth Chat Messenger as an Interactive and Collaborative Learning Aid. 10.1007/978-3-319-50340-0_1.

3. Deb, Amrita & Sinha, Swarnabha. (2014). Bluetooth Messenger: an Android Messenger app based on Bluetooth Connectivity. IOSR Journal of Computer Engineering. 16. 61-66. 10.9790/0661-16336166.

4. http://ethesis.nitrkl.ac.in/6762/1/Bluetooth_banerjee_2015.pdf

5. Deb, Amrita and Swarnabha Sinha. “Bluetooth Messenger: an Android Messenger app based on Bluetooth Connectivity.” *IOSR Journal of Computer Engineering* 16 (2014): 61-66.

6. https://in.mashable.com/tech/9554/firechat-will-be-your-go-to-messaging-app-when-the-internet-is-blocked

7. https://indianexpress.com/article/technology/tech-news-technology/what-is-bridgefy-the-offline-messaging-app-with-over-1-million-installs-in-48-hours-7172736/