TAGUIG CITY UNIVERSITY

SAFE-on-CHAT: A MOBILE MESSAGING AND VIDEO CALL APPLICATION WITH SECURITY FEATURES USING RIVEST SHAMIR ADLEMAN AND ADVANCED ENCRYPTION STANDARD ALGORITHM

A Thesis Presented to the Course Specialists

of the College of Information and Communication Technology

of Taguig City University

In partial fulfillment of the requirements for the Degree of

Bachelor of Science in Computer Science

By:

Amparo, Paul Victor M.

Batarina, Randel B.

Pariado, Keem R.

Rodriguez, Clark F.

June 2025

APPROVAL SHEET

The thesis titled "**SAFE-on-CHAT: A MOBILE MESSAGING AND VIDEO CALL APPLICATION WITH SECURITY FEATURES USING RIVEST SHAMIR ADLEMAN AND ADVANCED ENCRYPTION STANDARD ALGORITHM",** prepared and submitted by **Amparo, Paul Victor M., Batarina, Randel B., Pariado, Keem R., and Rodriguez, Clark F.** In partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science, has been examined and is hereby recommended for oral examination.

PROF. EDMAR G. TAN, MSIT
Technical Adviser

PROF. JACKELYN T. TRINIDAD, MSIT
Subject Adviser

PANEL OF EXAMINERS

Approved by the Committee on Final Examination with a grade of _____.

DR. REYNALDO G. ALVEZ
Chairman

PROF. JERIC T. NUEZ
Member

PROF. CELINE DIANNE T. MONTANO
Member

Approved in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

_____
Date

DR. REYNALDO G. ALVEZ
Dean, CICT

ACKNOWLEDGEMENT

Foremost, the researchers would like to express their heartfelt gratitude to Almighty God for giving them strength to overcome their fears and helping them get through tough times during the accomplishment of their thesis paper. The researchers never doubted You and Your guidance.

The researchers would also like to give their warmest thanks to their adviser, **Prof. Jackelyn T. Trinidad, MSIT**, and to their technical adviser, **Prof. Edmar G. Tan, MSIT**, for always giving their best advices whenever the researchers ask them for guidance.

The researchers would also like to extend their appreciation to the panel members, **Prof. Jeric T. Nuez**, **Prof. Jesus N. Abalo,** and **Prof. Celine Dianne T. Montano**. Researchers would not have been able to get this far without their brilliant suggestions and comments for the betterment of their thesis paper.

The researchers would also like to express their deep appreciation to the dean of faculty, **Dean Reynaldo G. Alvez**, for this opportunity to showcase our potential and guided us to our journey from the beginning to the end.

The researchers would like to take this opportunity to give their sincere thanks to their friends and peers for always encouraging them and offering their helping hand when times are rough.

They would also like to express their deep gratitude to their family for always showing their support financially and emotionally during the process of writing the paper.

Finally, the researchers extend the recognition to **Taguig City University** for providing the environment and resources needed for this study. This thesis is the result of many people combined efforts, and they are thankful for helping them to reach this important academic milestone.

ABSTRACT

The increasing need for secure messaging applications in today's digital world has made the use of cryptography techniques. Safe-On-Chat is a messaging and video call application designed to protect user data using Rivest Shamir Adleman (RSA) and Advanced Encryption Standard (AES) encryption methods. This study aims to develop a secure, user-friendly messaging and video call application that mitigates the risk of cybersecurity threats.

Using a developmental research approach and ISO 25010 evaluation, the system received positive remarks for its functionality, usability, and security. The findings highlight Safe-On-Chat as a reliable and effective communication tool that addresses the growing demand for private messaging platforms.

Keywords: encryption, RSA, AES, secure communication, messaging application

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Project Context

In today's digital landscape, secured messaging applications have become increasingly important due to the growing demand for privacy and security. With the rise of digital communication in personal, business, and governmental domains, ensuring the security of sensitive information against unauthorized access and cyber threats has become of utmost importance.

One of the main methods to protect information security is the use of cryptography techniques. Along with keeping the information private, it also performs other tasks like system security, digital signature, authentication, and secret sub-storage. In order to prevent information from being altered, forged, or counterfeited, the encryption and decryption solution can guarantee the confidentiality of the information as well as its integrity and certainty. Encryption techniques, specifically the Rivest Shamir Adleman (RSA) and Advanced Encryption Standard (AES) algorithms, offer an appealing solution for protecting data.

To meet the rising demand for secure communication, Safe-on-Chat is a messaging application that aims to offer privacy and security features. The resulting application will give users an effective and clever way to communicate securely across various applications, where data security is an essential component. The study's main goal is to develop a messaging system that can effectively deal with the growing risks of unauthorized access and data breaches. The application strives to provide data security

by incorporating both RSA and AES encryption algorithms. By offering reliable communication solutions across various domains, the research's findings could advance the field of secured messaging.

Purpose and Description

Purpose of the Study

The purpose of this study was to develop and implement a messaging application for widespread usage that will not compromise the privacy and security features through the use of RSA and AES encryption algorithms with the aim of ensuring the confidentiality and integrity of shared data where information security is critical.

Description of the Study

Safe-on-Chat is a proposed messaging application that aims to provide users with overall data protection by integrating RSA and AES encryption algorithms, utilizing technologies and frameworks, and performing testing to confirm the application's integrity, reliability, and usability. The goal of this study was to significantly advance the field of free for commercial use secure messaging applications by employing RSA and AES encryption algorithms.

Objectives of the Study

Main Objective

The main objective of this thesis was to develop and implement a messaging application that employs the RSA and AES encryption algorithms to provide the confidentiality and integrity of shared data as well as to create a user-friendly interface

that is simple and easy to use. This thesis extended the goal of creating a free messaging system that works with different devices and operating systems.

Specific Objectives

Specifically, the researchers defined objectives to achieve their study's aims, guiding their investigations and contributing to the overall goal. These goals helped them focus their attention on what was crucial and addressed the important components of the study.

The specific objectives are as follows:

1. To offer data security and confidentiality in communication by implementing the RSA and AES encryption algorithm.
2. To provide security measures against internet threats and unauthorized access for secure communication.
3. To design and create a user-friendly interface that is simple to navigate that users of the application have a smooth and easy-to-use experience.
4. To offer user authentication mechanisms to verify the identities of message senders and receivers, preventing impersonation and unauthorized access.
5. To evaluate the following performance of the system in terms of ISO 25010. Through survey questionnaires as well as compiling suggestions and recommendations, forming a basis for improvements in future system development.

5.1 Functionality,

5.2 Usability,

5.3 Efficiency,

5.4 Security,

5.5 Compatibility,

5.6 Maintainability,

5.7 Portability, and

5.8 Efficiency

Conceptual Paradigm

In this section, the researchers outline the conceptual framework of the system. A conceptual framework is a structure that lists the main ideas, factors, connections, and expectations which guide academic studies. It offers a theoretical framework as well as a perspective for analyzing data. To comprehend research issues, it makes use of pre-existing ideas, models, or expertise. It provides all of the objectives of the study, identifies variables, formulates research questions, and leads the choice of methods and data analysis strategies. Conceptual frameworks can be mathematical, taxonomic, or literary.

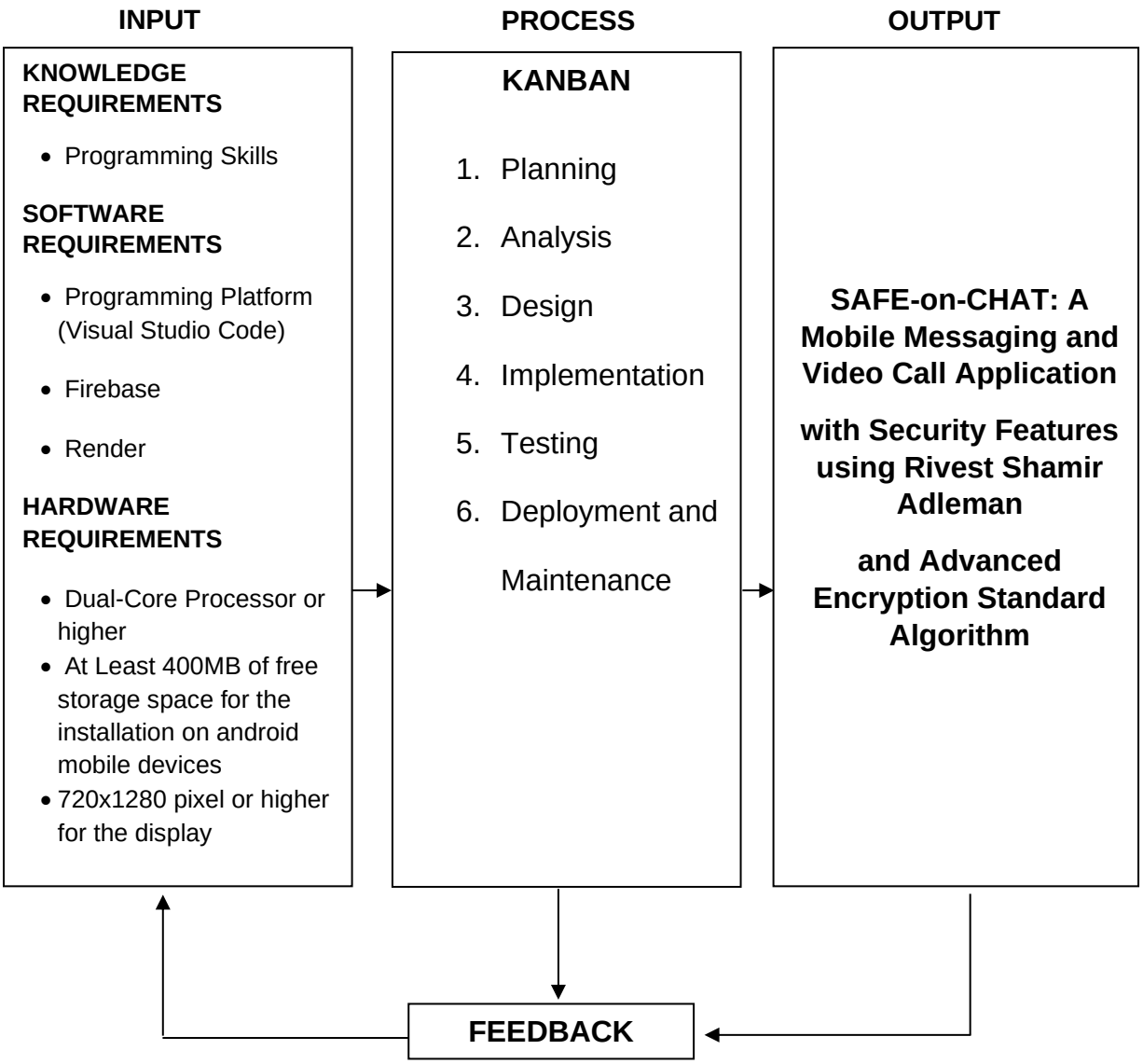


Figure 1. Conceptual Paradigm

The figure 1 above shows the conceptual paradigm. It describes the theoretical foundation and framework on which a study is built. It establishes the concepts, variables, relationships, and assumptions that form the basis of the research design. The figure above depicts the process and particular kanban method employed in this study, as well as the software and hardware requirements for the system that must be built.

Scope and Limitations

The scope of the study focused on creating a messaging application using RSA and AES encryption algorithms. It involves designing and implementing these algorithms for encryption and decryption, as well as incorporating a user-friendly interface and log-in options using both PIN and fingerprint, offering a more protected and private communication experience. The application can send files, videos, images, and text messages; create group chats; and make video and audio calls. The application also has an online indicator and pop-up notifications. Furthermore, the application disables screenshots inside the chat rooms.

The proposed messaging application also comes with some limitations. Only tablets and mobile phones can access it. Any other devices like desktops, laptops, smartwatches, and smart TVs cannot access the system. The application cannot customize the chat app's interface and cannot limit the number of users who want to access it.

Significance of the Study

The significance of creating a new messaging application using AES and RSA encryption algorithms lies in its potential to address a critical challenge in modern communication technology: security.

The use of the AES and RSA encryption algorithms provides the messaging application with reliable security. Given the increasing number of cyber-security threats and data breaches, using an encryption technique known for its strength and reliability

can offer users the trust that their messages are secure from interception and unauthorized access.

Therefore, the implementation of a messaging application with an encryption algorithm has the potential to improve communication security. Upon deploying this proposed system, there will be beneficiaries when using this proposed messaging application, which are the end users and the future researchers to use this study as their reference.

The **End Users** would be able to use a messaging application that offers data security.

And **future researchers** can use the study for providing insights about secured communication systems and privacy protection that can be relevant based on the results and findings of this study for their future references.

Definition of Terms

Advanced Encryption Standard (AES) Encryption Algorithm

is a highly trusted encryption algorithm used to secure data by converting it into an unreadable format without the proper key.

Communication defenses

are measures or strategies implemented to protect communication systems and networks from unauthorized access, data breaches, or other security threats.

Cyber attacks

are deliberate and malicious activities or attempts to compromise, disrupt, or gain unauthorized access to computer systems, networks, or digital infrastructure.

Cyber threats

are potential risks and dangers that target computer systems, networks, or digital infrastructure, including malicious activities like hacking, malware, phishing, and other cyber-attacks.

Data breaches

are incidents where unauthorized individuals gain access to sensitive or confidential data, potentially leading to data theft, loss, or misuse.

Encryption techniques

are methods used to convert plaintext into ciphertext to protect data confidentiality and integrity.

Hackers

are individuals with advanced knowledge and skills in computer systems and networks who seek to gain unauthorized access, exploit vulnerabilities, or disrupt operations for various purposes.

Rivest Shamir Adleman (RSA) Encryption Algorithm

is an asymmetric cryptography that employs both a public key and a private key. A public key is distributed publicly, as its name implies, whereas a private key is kept private and must never be disclosed.

Security assurance

includes measures, practices, and processes that ensure the confidentiality, integrity, availability, and reliability of systems, data, and information, providing confidence in their protection against unauthorized access or harm.

Symmetric key block cipher

is a type of encryption algorithm that uses a single key for both encryption and decryption. It operates on fixed-size blocks of data and transforms them into ciphertext.

Unauthorized access

is gaining access to data, systems, or networks without proper authorization or permission. It is a security breach and a significant concern in terms of data protection.

CHAPTER II

REVIEW OF RELATED LITERATURE

In order to support this study, the researchers gathered relevant studies and literature in messaging applications, security, and solutions, which are presented in this chapter, and added the study's synthesis as well to gain an understanding of this topic.

Message Applications

In accordance to Zarni S. et al. (2020), the significance of securing SMS messages due to the challenges in maintaining confidentiality and authenticity. It highlights SMS as a widely used method for text communication but emphasizes the difficulty in safeguarding its messages. It delves into the application of the RSA asymmetric key cryptography algorithm for encrypting and decrypting SMS data, specifically on Android platforms. Moreover, it touches on mobile data collection and the role of cryptography in ensuring privacy and security. The piece explains the RSA algorithm's role in secure data transmission, emphasizing its importance in maintaining confidentiality, authentication, integrity, and non-repudiation in mobile security systems. It suggests the need for further research to compare other public key cryptography systems.

Based on Dr. Priyanka Dubey (2022), chat applications have become one of the most important and popular applications on smart phones. It has the capability of exchanging text messages, images, and files, which is cost-free for the users to communicate with each other. All messages must be protected.

According to S. N. Alsaad (2020), among many types of social network applications, instant messaging is one of the applications that consider privacy and security to be two

crucial features due to the fact that the data exchanged between users is often private and not for public

According to A. Sahoo, P. Mohanty, and P.C. Sethi (2022). Information can be represented in multiple formats, including text, images, audio, and video. As the transmission of data over the internet continues to grow, concerns regarding security and authentication have become critical. To address these challenges, researchers have employed cryptographic algorithms that often utilize large-size keys to enhance data security. While larger keys provide a higher level of confidentiality, they introduce significant challenges in key management, making the process complex and resource-intensive.

Encryption techniques on Messaging Applications

The Armed Forces of the Philippines (AFP) are currently improving their cybersecurity by creating "end-to-end encryption" to safeguard classified communications, according to Priam Nepomuceno (2020). A report by Gapay states that these facilities can be closely watched over inside the camps, enabling recurring assessments and independent confirmation. This procedure is similar to the security guidelines that Globe and Smart set up for the locations of their towers on military installations. Gapay went on to say that the AFP reserves a more secure platform for classified data and uses commercial communication systems for unclassified information.

According to H. Bodur and R. Kara (2020). Service providers and software companies play a pivotal role in embedding security features, often implementing advanced measures without users' direct awareness. Among these security techniques, encryption algorithms stand out as one of the most vital tools for safeguarding

communication. These algorithms transform data into a secure format that can only be decrypted by the intended recipient, ensuring confidentiality and integrity throughout the communication process.

How RSA and AES algorithm work

As stated by Burgos Aeron B. et al. (2022), secured online communication requires the use of cryptography, particularly in data-centric apps like chats. Strong encryption techniques are necessary to stop hackers from accessing data. The suggested application is a web-based chat system that employs the AES algorithm for End-to-End encryption. AES is a symmetric encryption algorithm with bit lengths of 128, 192, and 256 that aims to provide users with a secure, real-time chat system.

Vulnerability of messaging application

According to A. Balapour, H. Reza Nikkaha, and R. Sabherwal (2020) investigates how privacy-related beliefs, such as privacy risk and the efficiency of privacy regulations, influence mobile app users' security views. To empirically test the suggested theoretical model, two surveys were conducted utilizing mobile apps that asked for less sensitive and more sensitive information, respectively.

Based on Y. Liu, B. Dantas Cruz, and E. Tilevich (2022) explains that modern mobile message-based communication is vulnerable to data leakage attacks, which allow untrustworthy apps to acquire conveyed message data. To improve the security of massage-based communication, they developed a model that includes two innovative mechanisms: hidden transmission and polymorphic delivery.

Threats of messaging applications

According to Quiazon, M. C. D., Manlutac, B. D., Besario, V. M., Centeno, C. J., & Casiw, G. M. (2024). The industry is seriously threatened by the unlicensed distribution, reproduction, and screen recording of copyrighted films. Industry management has to understand how the availability of digital platforms affects the demand for piracy in order to create practical solutions. The legal framework provided by intellectual property rights allows content creators to prevent unauthorized use of their creations, and illegal screen recording during playback seriously jeopardizes the integrity of copyrighted material. This study addresses these problems; the Android application "StreamShield," which prevents movie streaming through piracy, was created. The program actively involves users in the battle against piracy while addressing screen recording detection and content protection through the integration of technology and algorithms. Key features include an Anti-Screen Recording Mechanism, an Encrypted Media File system using the Advanced Encryption Standard (AES) algorithm, and a Picture Overlay Interference feature. StreamShield adheres to ISO-25010 guidelines, ensuring functionality, performance efficiency, reliability, security, and usability, contributing to a more secure and accountable digital entertainment ecosystem.

Comparison of Performance of RSA and AES Algorithm

According to R. Verma and A.K. Sharma (2020) Cryptography serves as a cornerstone for securing digital data, employing mathematical techniques to protect information from unauthorized access. Among the myriad cryptographic methods, symmetric and asymmetric algorithms stand out as two fundamental approaches.

Type of Cryptography

A.F Chernyavskiy, E.I Kozlova, and Yu. A. Chernyavskiy (2024) discusses the diverse types, functions, and key features of cryptosystems, along with various circuit design strategies aimed at enhancing their capabilities. It provides a comprehensive overview of key generation methods and ciphers typically used in cryptosystems with a standard structural framework. The study further outlines the structural diagrams for both symmetric and asymmetric cryptosystem organizations, highlighting their respective advantages and use cases. Additionally, a novel approach to implementing the decryption process within a threshold MIMA cryptomodule is proposed. This method is designed for secret sharing using a masking transformation, optimizing both time and hardware efficiency in reconstructing the original secret.

According to J. Williams, E.O. Bennett and V.I. Anireh (2024). In the current digital age, the need for secure communication is more critical than ever. Protecting sensitive information from unauthorized access while ensuring its integrity and authenticity has become a significant concern. Various encryption methods have been developed to address these challenges, with the most commonly used techniques being symmetric and asymmetric encryption. However, each of these techniques alone has limitations that can affect both security and performance. To overcome these limitations, hybrid encryption systems, which combine the strengths of multiple encryption algorithms, have been proposed.

According to Abhishek G. and Asha B., file encryption is an easy way to secure personal or commercial data. When used alone, the RSA and AES representative encryption algorithms fail to meet the criteria for file encryption dependability and security.

The experimental findings indicate that the RSA and AES hybrid encryption algorithm can not only encrypt files but also provide benefits in terms of algorithm efficiency and security.

Performance speed

According to M.H.M Baig, H. B. Haq and W. Habib (2023), in the digital era, ensuring information security has become paramount, with encryption serving as a cornerstone for safeguarding sensitive data against unauthorized access. Cryptographic algorithms, the backbone of encryption, are broadly categorized into symmetric, asymmetric, and hashing techniques. Symmetric cryptography operates using a single key for both encryption and decryption, while asymmetric cryptography employs a pair of distinct keys—a public key for encryption and a private key for decryption. Hashing, on the other hand, transforms input data into fixed-length bit strings, ensuring data integrity without reversibility.

Integration of RSA and AES in current applications

A secured chat application that uses the RSA encryption algorithm was developed, according to N. Namassivaya et al. (2023). Secured communication between users is ensured by the User/Server design, which prevents message decoding by the server. In addition to emphasizing the value of data security in communication, the paper suggests using cryptography to safeguard user data. The encrypted chat messages are stored on a cloud server, and the RSA algorithm is used for both encryption and decryption. The requirement for secured communication channels in order to guard against hacking and data loss.

An Improved RSA Algorithm for Enhanced Security

According Dr. K. Balasubramanian, M. Arun and Dr. K. R. Sekar (2024). The security of the Rivest-Shamir-Adleman (RSA) public-key algorithm relies heavily on the computational difficulty of factoring the modulus, which is obtained by multiplying two large prime numbers. The RSA cryptosystem's strength and utility stem from its asymmetric nature, where one key is used for encryption and another for decryption. However, the robustness of this system can be compromised if the chosen encryption and decryption keys are not sufficiently strong or well-designed.

Based on A. Kousalya and Nam-Kyun Baik (2023) The RSA algorithm, though widely used, faces challenges related to key management and computational efficiency. Improved RSA-based schemes incorporate optimizations such as key size reduction and modular arithmetic enhancements to address these issues (Al-Yasiri et al., 2020). When combined with access control models like RBAC, these methods offer a robust framework for securing sensitive information.

According to Edwin Kho et al. (2020), application is not limited to a single security technique but can also include information security data. The AES method is a cryptographic algorithm that can be used to secure data; the AES algorithm is a symmetric chipertext block that can encrypt and decrypt information, and the RSA method is the RSA Algorithm, which is a block cipher algorithm that divides plaintext into blocks before encryption to create ciphertext. This application aims to secure messages sent by the sender so that they are safe when received by the recipient. The sender will send a

message through the live chat website, and the message will be encrypted so that its security is guaranteed. The message stored in the database will be automatically.

According to I. Lukman, A. Arief, A.A. Ilham and Syafaruddin (2023). The Advanced Encryption Standard (AES) is a robust cryptographic algorithm widely adopted for securing sensitive data. AES-256, a block cipher symmetric encryption method, has been recognized for its high level of security and efficiency. Studies by Rijmen and Daemen (2001) demonstrated AES-256's resilience against brute force attacks, making it suitable for applications requiring strong encryption. Recent research has explored embedding AES-256 into communication protocols, such as LoRa, to enhance security at both the transmitter and receiver ends. This approach not only ensures encrypted message transmission but also aligns with the ITU-R standard for bandwidth usage.

Synthesis of the Gathered Literature

The amount of detail of the RSA and AES encryption techniques is highlighted in the literature as it examines encryption algorithms in messaging applications. Using these algorithms, researchers suggest creating a secured messaging application that protects the privacy, confidentiality, and security of user input data. Design, implementation, security features, modifications, and application-specific considerations are just a few of the topics covered in the study. The literature's contributions to secure messaging and encryption algorithms can greatly benefit researchers, practitioners, and developers in improving the security environment for messaging applications. It also addresses challenges in maintaining communication channels, contributing to the field of secure messaging and encryption algorithms.

CHAPTER III

RESEARCH METHODOLOGY

This section describes the research method, participants, population and sample, research instruments, data gathering procedures, and statistical analysis of the data. This is used to gather detailed information about user satisfaction, security concerns, and overall usability.

Research Method Used

The researchers used a developmental research approach that focuses on the design and development of the messaging application encrypted with the use of both Rivest Shamir Adleman (RSA) and Advanced Encryption Standard (AES) algorithms.

Participants

The participants of this study were individuals who use multiple existing messaging applications and are interested in using messaging applications that offer data security and confidentiality in communication.

Population

Table 1. Respondents, Frequency and Percentage

Respondents	Frequency	Percentage (%)
General Users	15	60
IT Professionals	10	40
Total	**25**	**100%**

Table 1 illustrated the target population of this study focuses on the end users. The researchers also prefer to conduct surveys with those individuals that have background about the development of applications and are actively users of different messaging tools.

The population includes 10 IT professionals and 15 general users, summing up to a total of 25 individuals.

Sample

This study's sample is composed of individuals that are preferably using messaging applications on a daily basis. The researchers used the purposive sampling method to guarantee the relevance of the data from the selected individuals.

Research Instruments

The researchers intended to use a quantitative research instrument, specifically with the use of survey questionnaires, to gather data from participants that was distributed through personal and online platform. The questionnaires were designed primarily for those individuals that are using existing messaging applications to evaluate the developed system.

The researchers also provided a section where they can comment on their feedback, experiences, perceptions, and suggestions about the developed messaging application.

The researchers used the ISO 25010 to evaluate the functionality, reliability, usability, maintainability, efficiency, portability, and security of the developed system.

Data Analysis and Procedure

To analyze the data gathered through our survey questionnaire, the researchers used descriptive statistics, such as frequency, mean, and standard deviation. The researchers used this analysis to identify the strengths and weaknesses of the developed messaging application and to make recommendations for improvement.

Validation and Distribution of the Instrument

To evaluate the reliability and accuracy of the survey instruments, the researchers validated it through an expert review, particularly with experience in software development and algorithms. Based on their feedback, the researchers revised and finalized the questionnaire before distributing it to all respondents through the survey.

Data Encoding and Formulation of the Solution

The researchers began the data encoding by using the descriptive statistics approach to analyze the information gathered from the survey. Additionally, the results can be used to develop and enhance the security and user interface of the application.

Evaluation of Data and Result

Descriptive statistics were used to analyze the collected data as part of the evaluation of the data and results. The survey involved analyzing and categorizing the responses based on the ISO 25010 characteristics. The results were analyzed to gain insights and conclusions about the application's security and user interface's strengths and weaknesses. Based on these findings, the researchers could offer recommendations for enhancements to the developed messaging application for future researchers.

Statistical Treatment of Data

Statistical Tools

Certain statistical tools were utilized by the researcher to gain insights from the users who have experience using the messaging app at Taguig City University. The answers to the questionnaire were analyzed by tallying.

Frequency

The number of respondents was identified and divided by a total number of respondents to acquire the correct percentage and appropriate description.

$$F = f/N \times 100$$

Where:

F = frequency

N = Sample Size / Total Population

Percentage

In order to make comparison of the responses possible, this statistical tool was employed since it reflects which items were favored most or favored least.

The formula is:

$$\% = f/n \times 100$$

Where:

% = percentage

f = frequency

n = total number of responses

Weighted Mean

The Arithmetic Mean is the summation of the scores divided by the number of respondents. A weight Arithmetic Mean is used which variables involved in this study, disaster, preparedness, response capability, quality of training, etc., are considered abstract and continuous and cannot be counted individually.

It is obtained by applying the formula:

$$Xw = \sum fw / \sum f$$

Where:

Xw = sum of all the products of f and w is the frequency of each weight and w is the weight as 4,3,2,1.

$\sum f$ = sum of all the responses of the sample size.

Technical Requirements

This section outlines hardware, software, and network requirements for a developed system. Hardware includes computing devices, servers, and peripherals for smooth operation. Software includes operating systems, frameworks, databases, and application dependencies. Network requirements outline data transmission, connectivity, and security for a stable, high-speed network.

The project design effectively integrates various components, ensuring a well-structured system architecture that aligns with project goals and improves user experience.

Hardware Requirements

Table 2. Hardware Requirements

HARDWARE	DESCRIPTION
MOBILE	
PROCESSOR	Dual-Core Processor or higher
STORAGE	At Least 400MB of free storage space
DISPLAY	720x1280 pixel or higher for the display
OPERATING SYSTEM	Android Version 9.0 to 14 (the most recent version)
LAPTOP	
RAM	Minimum of 4GB of RAM
ROM	At least 128gb of ROM
GPU	Intel ® UHD Graphics 605 or higher
CPU	Intel(R) Pentium(R) Silver N5030 CPU @ 1.10GHz 1.10 GHz or higher

Table 2 above outlines the minimum hardware requirements to access the messaging application. The device should run on Android Version 9.0 to 14 (the most recent version). The display resolution must be at least 720x1280 pixels. The device should have a dual-core processor, such as the MediaTek MT6737, or better. Additionally, the user must have a network speed of at least 15 Mbps or higher on Wi-Fi connection. At least 4G LTE and 5G on mobile networks.

Meanwhile, the minimum hardware requirements for the development of the system are that the device should have a quad-core processor, such as the Intel(R) Pentium(R)

Silver N5030, or better. The device should have a 4GB RAM capacity or higher. The device should have at least an Intel® UHD Graphics card or higher.

Software Requirements

The given figures below were the software requirements that researchers used to develop the proposed system, Safe-on-Chat. The following figures provide a details for the specific software requirements that were utilized in the development process.

Figure 2. JavaScript Logo

JavaScript plays the vital role in developing the proposed system, both frontend and backend. On the frontend, it creates an interactive and responsive user interface. It also deals with the UI design of the system. And for the backend development, JavaScript facilitates real-time data synchronization, user sign-ups, and identity verification. The researchers used the Node JS version 0.20.14.0.

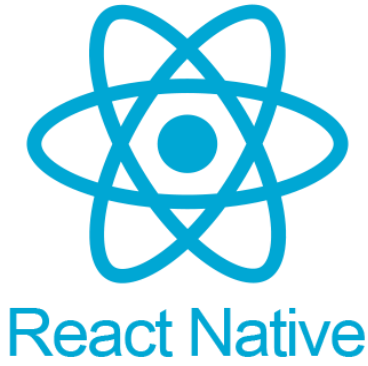


Figure 3. React Native Logo

React Native is the framework used by the researchers in developing the system. It enables the creation of a dynamic and responsive user interface. It supports real-time communication, instant message delivery, and typing indicators by integrating Firebase. React Native also supports file and media uploads. The researchers used React Native Version 0.74.5.

Figure 4. Firebase Logo

Firebase is a backend service platform commonly used for mobile and web development due to its scalability and ease of integration. It provides essential services like authentication, enabling user sign-up, sign-in, and identity management using methods such as email. Firebase offers real-time messaging and handles multimedia sharing, allowing users to upload images, videos, and documents securely. Basically, all

the features that the researchers are going to implement. Firebase simplifies backend development and supports real-time data synchronization, making it a great choice for building and maintaining the system. The researchers used Firebase Version 10.12.2.

Figure 5. Visual Studio Logo

Visual Studio Code (VS Code) is a lightweight code editor used for the development of the system. It offers advanced code editing features and an extensive library of extensions that are suitable for both the frontend and backend, as well as Firebase tools for integration with Firebase. The researchers used the VS Code version 1.96.4.

Figure 6. Render Logo

Render is a platform supporting calls and video calls for the messaging application. It can host real-time communication servers using WebRTC, which handle signaling, media streams, and peer connections for audio and video calls.

Network Requirements

Figure 7. Internet Connection

The proposed messaging application requires an internet connection to function, such as a mobile data connection or Wi-Fi. The recommended network speed for accessing the application is 10 Mbps or higher on a Wi-Fi connection. For cellular networks, the device should support at least 4G LTE or 5G.

Project Design

The given figures below are the initial design of the proposed system by the researchers. It consists of different parts for mobile devices. It contains log-in pages where users can put their credentials to log-in, a register page where new users can create their accounts, a home page, and a chat page where users can see the other users and interact with them.

Figure 8. Registration Page

The figure above shows the Registration page, which allows the user to create their account in order to the application. On this page, the user is required to provide some essential credentials. First, the user needs to choose a unique username that will represent their profile within the application. Next, a verified email address must be provided. The email address will be used to send a confirmation link ensure the authenticity of the user's account creation. Lastly, the user is asked to set a password that meets the specific security requirements outlined by the application. This ensures that the account remains secure and protected from unauthorized access. Once all these fields are filled out correctly, the user can proceed to complete the registration and gain access to the application.

Figure 9. Log-in Page

Figure 9 shows the Log-in Page. If the user has already created an account, they can proceed to this page to log in and access their account. To log in, the user is required to enter their username and password in the designated fields. The username should match the one they previously registered with, and the password must be the correct one associated with that account. If the user does not have an account yet, they can easily create one by clicking on the link labeled ‘Sign up here.’ This will redirect them to the Registration page, where they can provide the necessary credentials, such as a username, verified email address, and password, to create a new account and gain access to the application.

Figure 10. Home Interface

The figure shown above illustrates the Home Interface of the application, where the user can view their chat history. This interface is designed to provide an organized and easy-to-navigate overview of all past conversations the user has had. It displays a chronological list of messages, allowing the user to quickly review their interactions. The chat history is an essential feature, enabling users to refer back to previous exchanges, whether for personal reference or to continue ongoing discussions. On the lower right, you can see a call icon, which allows you to track your call history as well.

Figure 11. Chat Interface

Figure 11 above shows the Chat Interface of the application. This is where the sender and recipient can have conversations and exchange messages. All conversations are protected using RSA and AES algorithms, to keep the messages secure. Users can also send files and images through the chat. On the upper right side, there are icons for making voice calls and video calls. By clicking these icons, users can start a call or a video call with the recipient, allowing for more personal communication. This feature makes it easy for users to chat and connect in different ways, while keeping their conversations private.

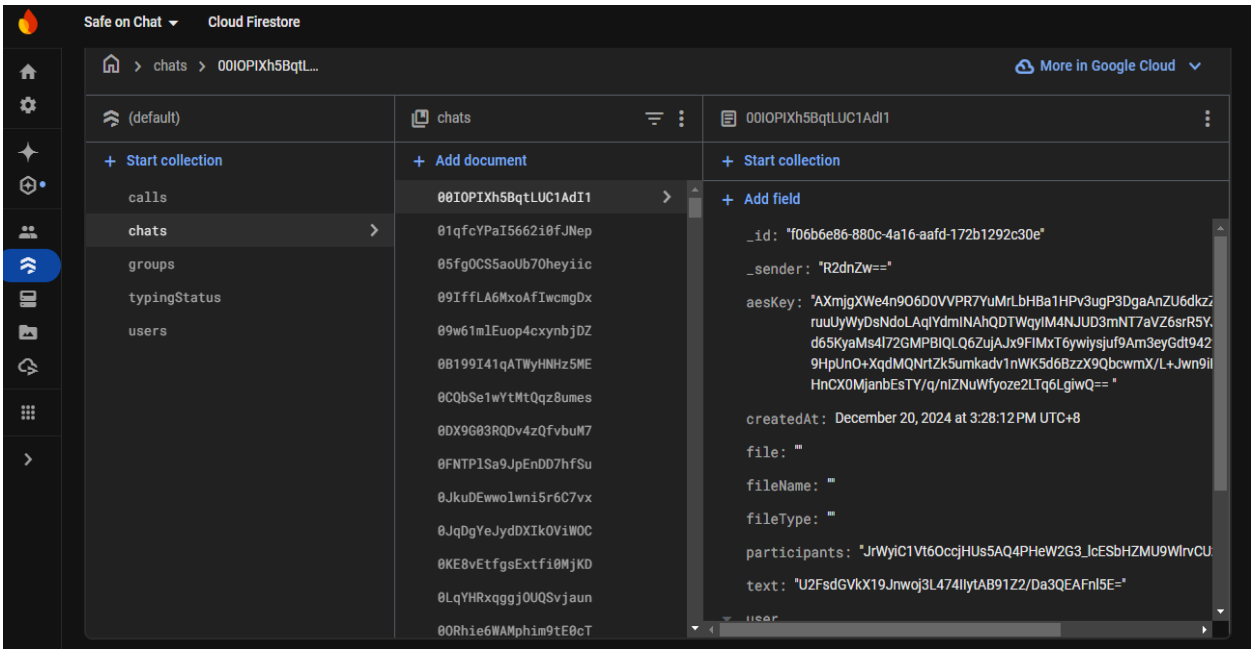


Figure 12. Hosting View

Figure 12 above shows what the hosting side looks like when the user sends and receives messages from different users. The system uses Firebase hosting to securely store messages in a cloud database, ensuring their privacy and quick access. The system uses security algorithms to encrypt messages, storing them as ciphertext on the hosting side. This process ensures that the contents of the messages remain private and cannot be easily accessed by unauthorized parties. The use of encryption and secure cloud storage is a critical feature of the proposed system, protecting users' data and ensuring their communication remains confidential.

Diagrams

Diagrams are the constructed visualization of the researchers to guide them on how to create the proposed system. This part consists of the System Architecture, Data Flow Diagram, Flowchart, Unified Modeling Language, Database Structure, and System

Development. Initial diagrams were created for each category to better understand the system's functionality.

System Architecture

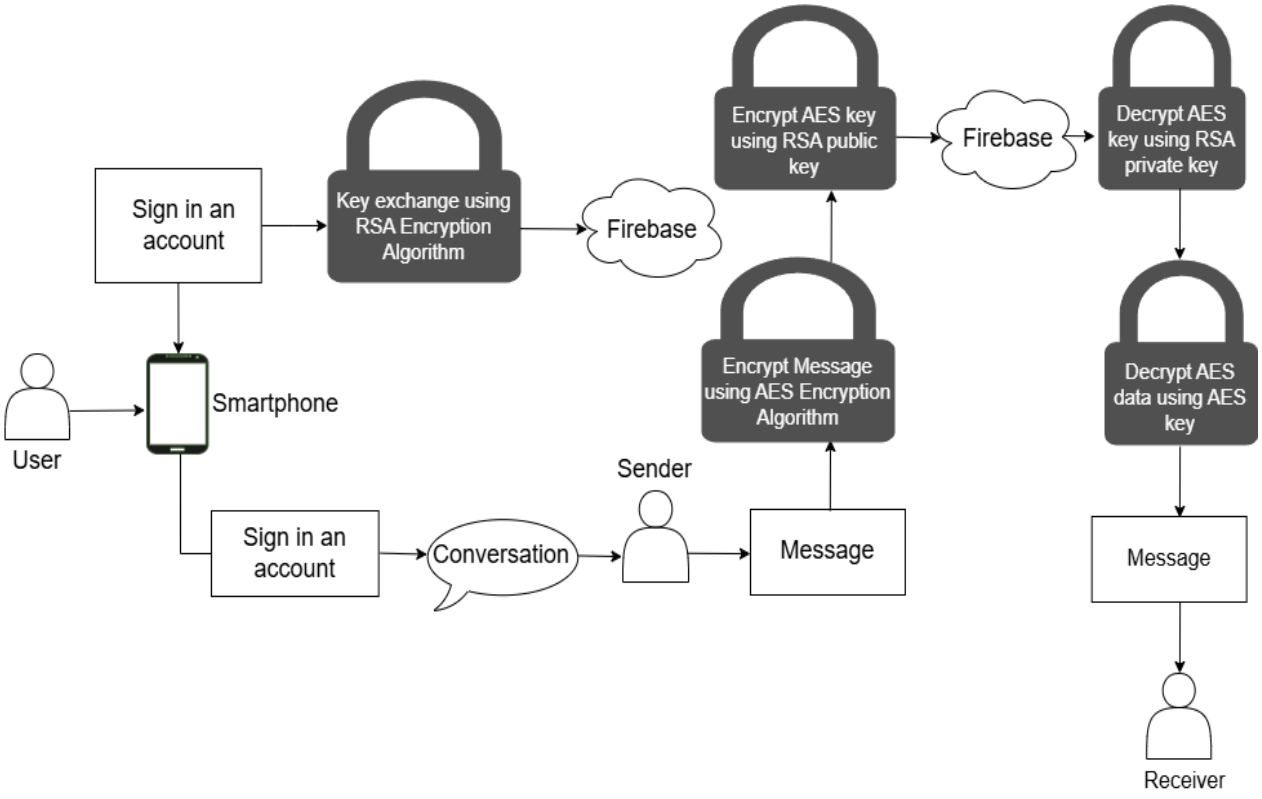


Figure 13. System Architecture of Messaging Application

Figure 13 above shows the System Architecture of the messaging application. When a user creates an account, a public and private key will be generated for that specific user. When a conversation begins, the messages will be encrypted with an AES key, and that AES key will be encrypted using the RSA public key. The encrypted data will be stored in Firebase and will be decrypted before being received by the recipient. This process ensures a secure and confidential exchange of information throughout the messaging system.

Data Flow Diagram

Figure 14. Data Flow Diagram Context Level 0

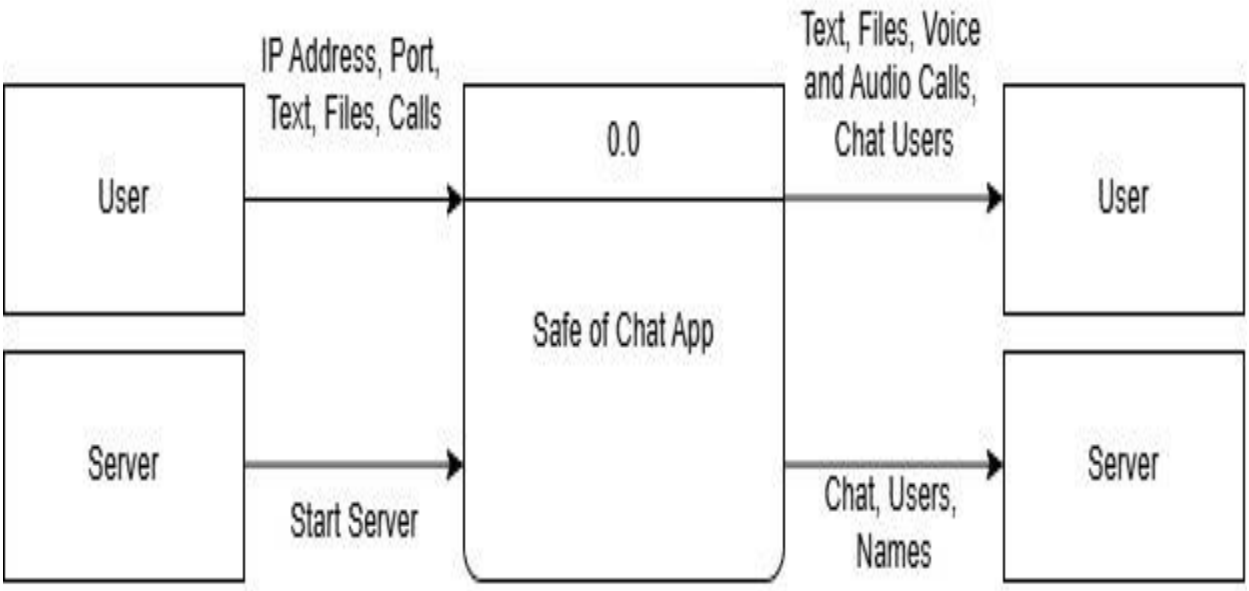


Figure 14 illustrates the Data Flow Diagram Level 0. It consists only of the main process of the developed messaging application by the researchers. The user will send data to the messaging application, which will be sign-in, authentication, chats, and calls. Safe of Chat App shows how users and servers interact with the system. Users send messages, files, and calls, while the app processes and returns the data for communication. The server helps manage the system by starting the app and storing chat and user data. This diagram provides a simple overview of how data flows within the chat application.

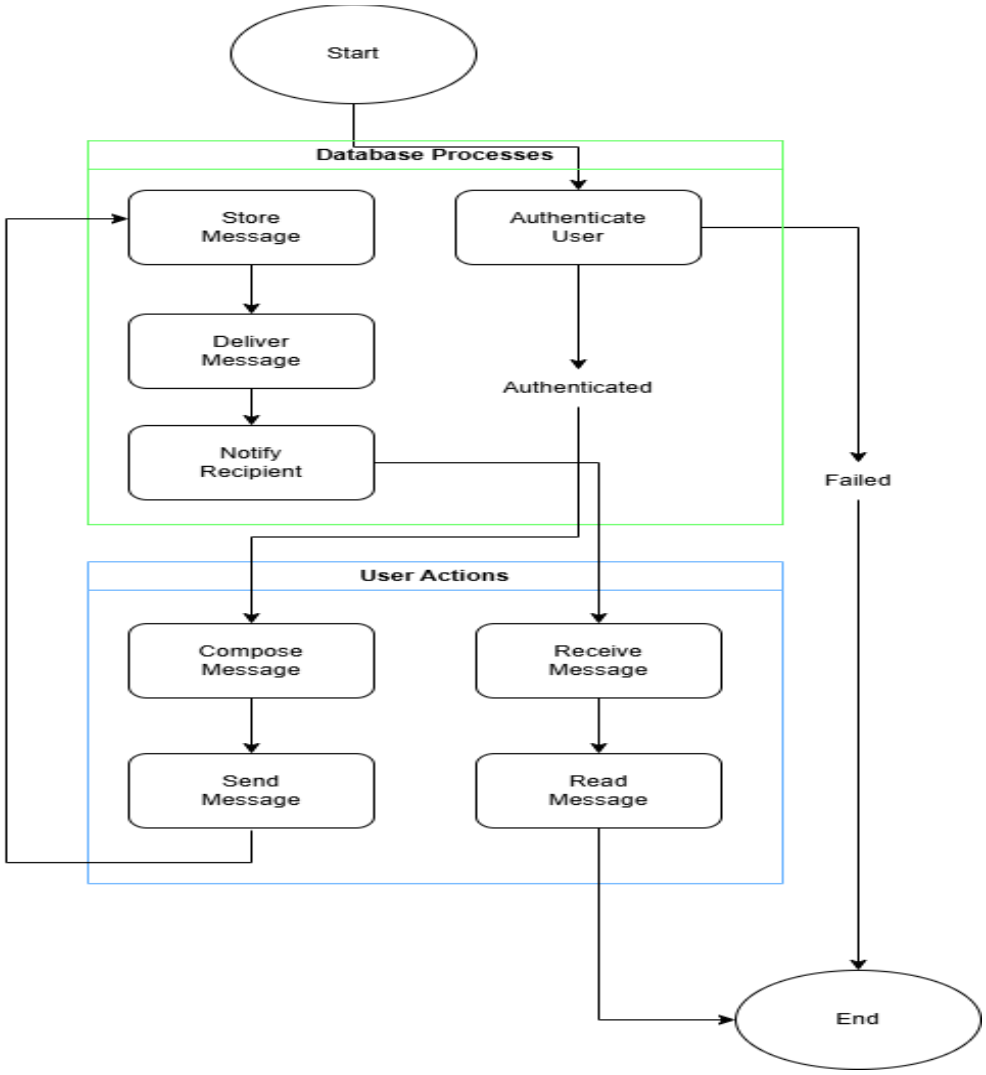


Figure 15. Data Flow Diagram Context Level 1

Figure 15 illustrates the Data Flow Diagram Context Level 1. The illustration consists of a slightly deeper understanding of how the system works, also including the encryption method. The user will input their credentials, and it will store them directly in Firebase. The user information will serve as an inventory of new and old users of the system. Next, the user will send chats and calls, create group chats, and send images and videos. The data that will be sent to Firebase will go through the encryption phase so that the data will be secured when it is sent to the receiver of the data.

Proposed Flowchart

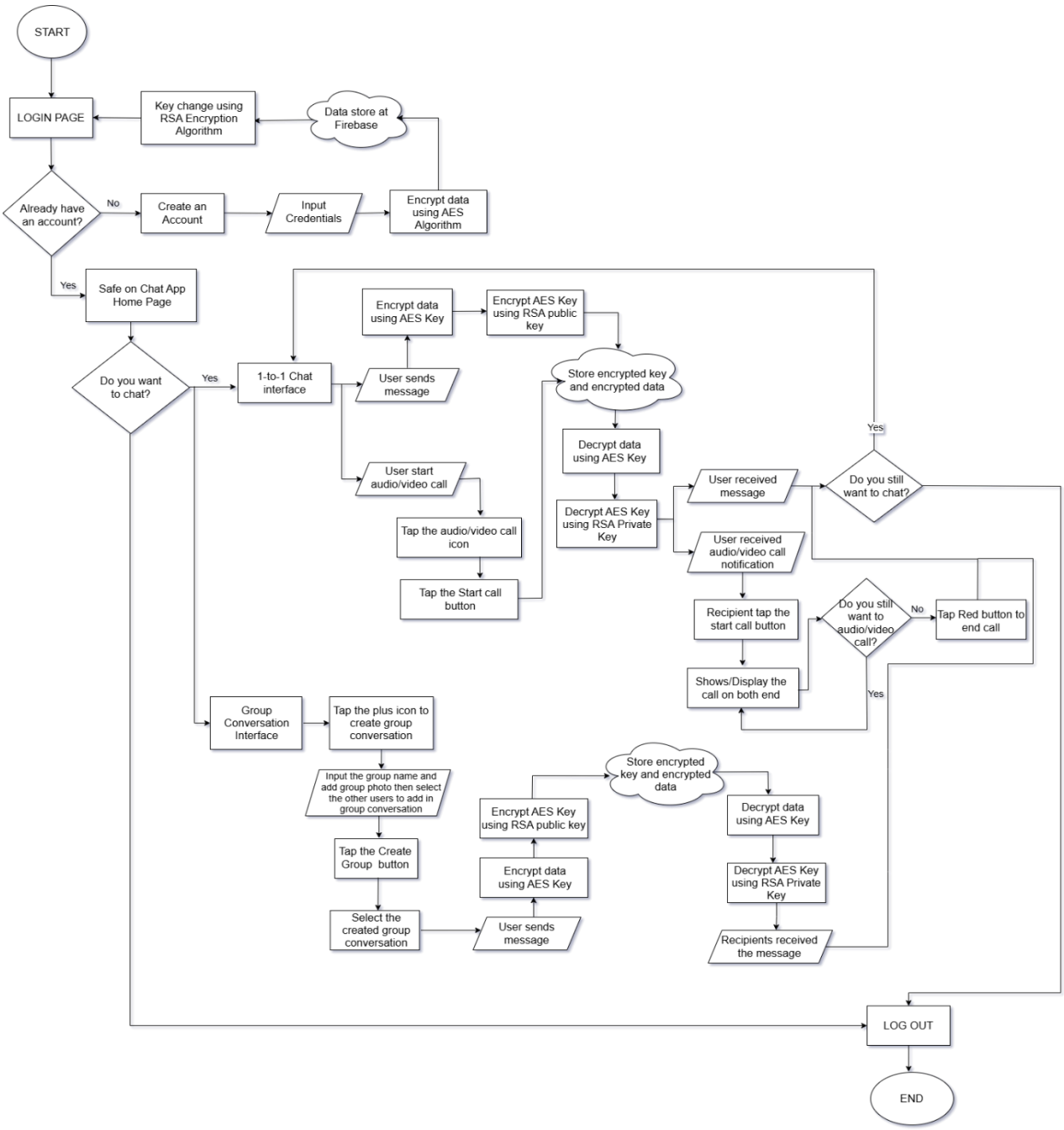


Figure 16. System Flowchart

Figure 16 above illustrates the flowchart of the system. The application requires users to have an account. If a user does not have an existing account, they need to sign up by providing all the necessary information to successfully create an account.

Afterward, the user must authenticate using their fingerprint and PIN from the local device. Once the account is successfully created, the application will display the home screen.

The user can then send messages to other users. Inside the chatroom, messages sent will be encrypted using an AES key. The AES key itself will be encrypted using an RSA public key and stored in Firestore. To deliver a message to the recipient, the encrypted message will be decrypted. This process involves decrypting the AES key using the RSA private key, and the decrypted AES key is then used to decrypt the encrypted message. Once decrypted, the message will be available to the recipient. This process continues as long as the conversation is active. If not, the user has the option to log out of their account.

Unified Modeling Language

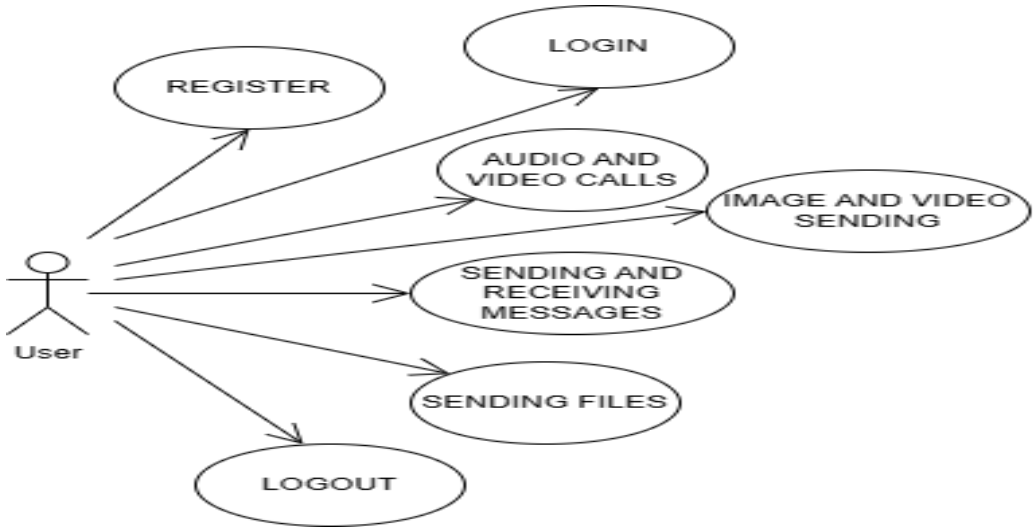


Figure 17. User View

Figure 17 shows the User View, allowing users to register, make calls, send images, photos, messages, and files, and log out after using the messaging app.

Database Structure

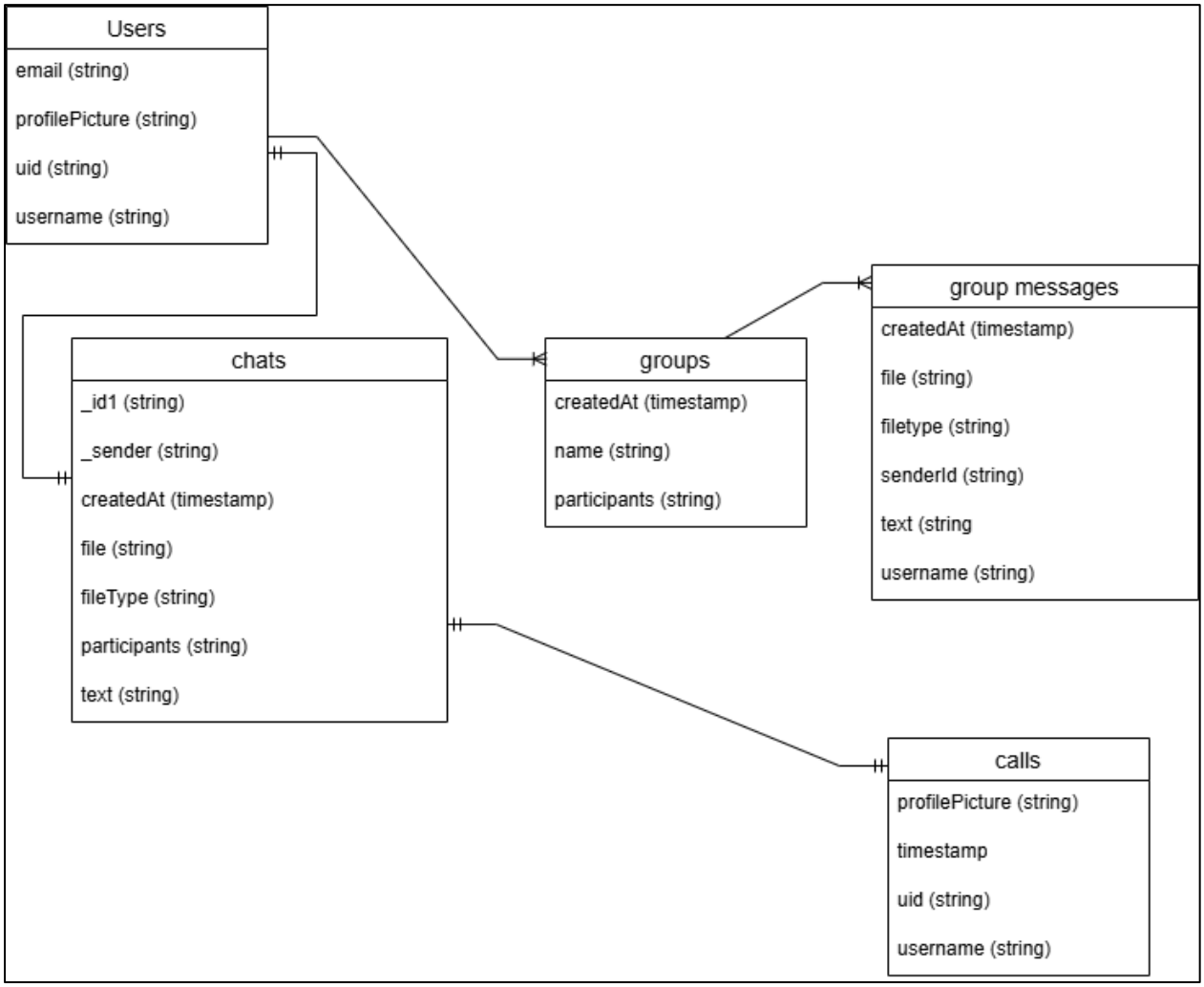


Figure 18. Database Structure

The figure 18 shows the database structure of the messaging application in Firebase. It is represented by this database schema, which consists of five primary entities: Users, Chats, Groups, Group Messages, and Calls. User information, including email, username, and profile picture, is stored in the Users table. Private messages, including text, files, and participants, are handled by Chats. Group Messages stores individual messages within groups, including sender information and shared files, while

Groups controls group conversations with attributes like group name, creation time, and members. And Calls records call information, including timestamps and participants.

System Development

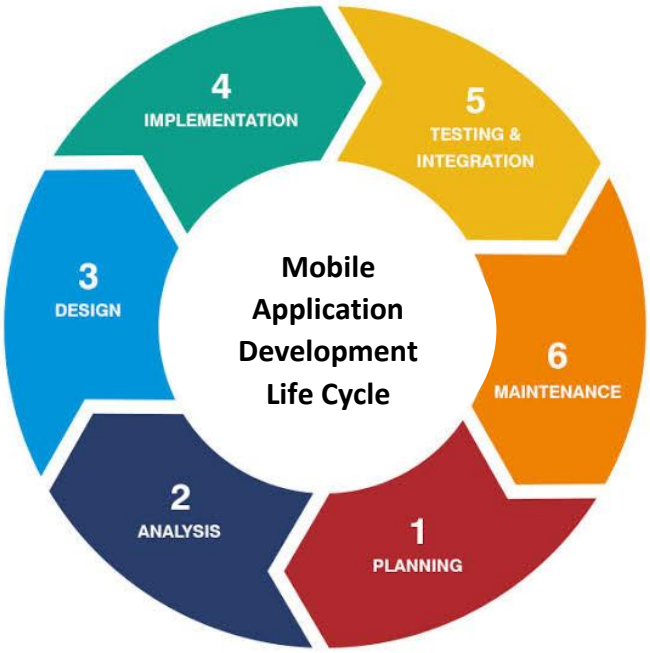


Figure 19: Mobile Application Development Life Cycle

Figure 19 shows the Mobile Application Development Life Cycle. It is the process used to be more efficient when developing a system. SLDC has different models and one of these is the Kanban. It is a method used to organize the flow of work and promote continuous improvement. It helps the team visualize tasks, with work items "pulled" from a product backlog into a steady workflow. Kanban is represented by a Kanban Board, which typically includes columns such as To-do, In Progress, and Done. Allowing the team to track the progress of a project.

The following phases outline the stages of development as defined by the SDLC.

Planning

In this phase, researchers plan the objectives and scope of the proposed application. They conceptualize a solution to address data privacy issues related to the gathered data of a messaging application. In real-life scenarios, data breaches frequently occur and are often exploited for scamming. The proposed system employs two algorithms, RSA and AES, which serve as the security measures for encrypting the exchange of data within the messaging application.

Analysis

After the planning stage, researchers analyze the features that can enhance security, the approaches to be used, and the requirements needed. They decided to implement features such as disabling screen capture and screen recording in chat rooms. They also determined the tools and programming languages to be used during the implementation phase.

Design

In this phase, researchers create a detailed structure of the application. A system design is developed to help visualize the application. Possible changes, such as theme colors, icon alignment, and the overall user interface (UI), are identified for improvement.

Implementation

This stage involves writing the program code. It includes building the application, connecting it to Firebase, and integrating the required servers.

Testing

At this stage, users test the application to identify bugs, necessary fixes, adjustments, or enhancements.

Deployment & Update

This phase involves deploying the application, making it ready for users. Researchers also monitor the overall system performance of the application and identify areas for improvement to enhance the messaging application's functionality.

Algorithm Discussion

Comparison of the Algorithm

Since the proposed system will use two different encryption methods, the researchers decided to compare the algorithms that will be used in the proposed system, to the most used encryption algorithm in other messaging applications.

This comparison aims to evaluate the security, efficiency, and practicality of the selected encryption methods in relation to algorithms. By analyzing factors such as key length, resistance to attacks, and computational performance, the researchers can determine the most suitable encryption approach for ensuring secure communication within the system. Additionally, this comparison highlights the trade-offs between different encryption techniques, allowing for an informed decision that balances security and system performance.

Advanced Encryption Standard (AES)	Data Encryption Standard (DES)
Strengths: AES supports key sizes from 128-bit, up to 256-bit. The larger key sizes make it also very secure against attacks. AES is known for its efficient performance and resource usage, making it suitable for diverse devices like computers, mobile devices, and embedded systems. Its implementation in both hardware and software minimizes the computational overhead, ensuring a smooth and optimized encryption process across different applications and devices. **Weaknesses:** AES implementations can be vulnerable to side-channel attacks, which exploit information leaked through physical characteristics of the system such as power consumption, electromagnetic emissions, or timing measurement. While AES is highly resistant to known cryptographic attacks, it still susceptible to future attacks that exploit yet undiscovered vulnerabilities or weaknesses.	**Strengths:** Its 64-bit block size and 56-bit key length allowed it to run efficiently on the computers of the time. DES's design is straightforward and well-understood, making it relatively easy to implement in hardware. DES was popular in embedded systems and hardware-based encryption before more advanced algorithms. **Weaknesses:** DES has the key size of only 56-bits, which makes it more vulnerable to brute-force attack. DES uses a 64-bit block size, which is relatively small by modern standards. When encrypting large volumes of data, this can lead to attack like cyphertext - only analysis.

RSA Algorithm (Rivest-Shamir-Adleman)	DSA Algorithm (Digital Signature Algorithm)
Strengths: RSA's security is based on the mathematical properties of prime numbers,	**Strengths:** DSA's security is based on the difficulty of solving the discrete logarithm

providing a solid theoretical foundation. The algorithm relies on the difficulty of factoring the product of two large prime numbers, making it resistant to certain cryptographic attacks.

RSA is widely supported across different platforms, operating systems, and cryptographic libraries. This broad support enhances its interoperability, making it easy to implement in various systems and applications.

RSA is often used in key exchange protocols, such as in the initial key exchange phase of the TLS (Transport Layer Security) protocol used to secure internet communication. Its ability to securely exchange keys for symmetric encryption contributes to the security of communication channels.

Weaknesses:

RSA is vulnerable to attacks by quantum computers. The potential development of practical quantum computers could compromise the security of RSA by factoring large numbers exponentially faster than classical computers.

The security of RSA is highly dependent on key size. Shorter key lengths, like 1024 bits, are now considered vulnerable, necessitating the use of longer key lengths (2048 bits or higher) for stronger security.

problem in a finite field. This mathematical foundation provides a strong level of security, especially when implemented with appropriate key lengths.

DSA is part of the Digital SignaturE Standard (DSS), established by NIST. Standardization enhances interoperability, ensuring consistent rules and specifications for DSA implementations. This contributes to a more unified and widely accepted use of the algorithm.

DSA is specifically designed for digital signatures, making it a focused and effective choice for authentication, data integrity verification, and non-repudiation in secure communications.

Weaknesses:

DSA implementations may be susceptible to timing attacks, which exploit variations in cryptographic operations, necessitating the use of constant-time algorithms for mitigation.

DSA is designed specifically for digital signatures and lacks versatility compared to algorithms like RSA, which can be used for both encryption and digital signatures.

Features

RSA

1. Asymmetric encryption algorithm
2. Provides encryption, digital signatures, and key exchange functionalities
3. Uses public and private key pairs for encryption and decryption
4. Based on mathematical operations involving large prime numbers and modular arithmetic
5. Key strength and security depend on the size of the keys used
6. Widely used in secure communication, digital signatures, and certificate-based authentication

AES

1. Symmetric encryption algorithm and a block cipher
2. Supports key sizes of 128, 192, or 256 bits
3. Provides strong resistance against cryptographic attacks
4. Offers high-speed encryption and decryption
5. Suitable for secure communication, file encryption, and disk encryption.

Function

RSA is a versatile cryptographic algorithm that encompasses encryption, digital signatures, key exchange, and certificate-based authentication. It encrypts data using the

recipient's public key, generates and verifies digital signatures for authentication, enables secure key exchange, and supports the infrastructure for trusted certificates. Additionally, RSA can be combined with symmetric encryption for efficient and secure communication. Its widespread use highlights its ability to provide confidentiality, integrity, authentication, and secure key management in various cryptographic applications.

AES is a secure symmetric encryption algorithm used in various applications, including wireless security, database encryption, secure communications, data storage, VPNs, secure password storage, and file and disk encryption. It operates on fixed-size 128-bit blocks of data and offers strong resistance against cryptographic attacks. AES is commonly used in protocols like internet communications, email, instant messaging, and voice/video calls and is also used in VPNs to ensure data privacy. Its high-speed performance, especially with hardware acceleration, makes it suitable for scenarios requiring robust and efficient data protection.

Uses

AES encryption algorithm can be used for file encryption, network security, database encryption, and other works that need encryption or security. While RSA is often used in a much more secure environment, such as digital signatures and certified-based authentication, and also for encryption.

Security Strength Techniques of the Algorithms

AES

1. **Substitution-Permutation Network (SPN)**: AES employs a Substitution-Permutation Network structure, replacing byte blocks with non-linear substitution tables, cyclically

shifting data matrix rows, mixing columns with fixed polynomials, and XORing the block with a round key derived from the main key.

2. **Key Length**: AES supports three key lengths: 128 bits, 192 bits, and 256 bits. Longer key lengths provide higher levels of security.

3. **Key Schedule:** AES uses a key expansion process to generate a series of round keys from the main encryption key. The round keys are used in each round of encryption and decryption, ensuring the key is distributed and diffused across the entire block.

4. **S-Box Design**: The S-Box design used in AES to be non-linear and to have strong cryptographic properties such as low correlation and hign nonlinearity, which protects against life differential power analysis and side-channel attacks.

5. **Confusion and Diffusion:** AES effectively employs confusion and diffusion principles to create a complex relationship between ciphertext and encryption key, thereby making it difficult to deduce input patterns.

RSA

1. **Modular Arithmetic:** RSA relies heavily on modular arithmetic operations, such as modular exponentiation and modular multiplication. These operations play a key role in the encryption, decryption, and key generation processes.

2. **Public and Private Key Pair:** RSA employs the concept of a public and private key pair. The public key is used for encryption and digital signature verification, while the private key is kept secret and used for decryption and digital signature generation.

3. **Large Prime Number Generation:** RSA requires the generation of large prime numbers during key generation. The security of RSA relies on the difficulty of factoring large composite numbers into their prime factors.

4. **Exponentiation**: RSA employs exponentiation to encrypt and decrypt data. The encryption operation involves raising the plaintext to the power of the public key exponent, modulo the public key modulus. The decryption operation involves raising the cipher text to the power of the private key exponent, modulo the private key modulus.

5. **Padding Schemes:** RSA employs padding schemes, such as PKCS#1 (Public-Key Cryptography Standards #1), to add additional security and randomness to the plain-text before encryption. Padding helps prevent certain cryptographic attacks and ensures proper message recovery during decryption.

6. Random Number Generation: RSA requires the generation of random numbers for key generation and padding. Proper random number generation is crucial to ensure the security of RSA operations.

Application Comparison

In this table, the researchers gathered different messaging applications that exist today compared to the developed system. The researchers listed down their unique features, platform support, and the algorithm used.

Application	**Features**	**Platform Support**	**Algorithm/s**
Safe-On-Chat	✓ Real-time chat ✓ File Send ✓ Group chats	Android	AES RSA

	✓ Data Encryption ✓ Multimedia ✓ Pin and Fingerprint log in ✓ Disable ScreenShots ✓ Encrypted Messages		
Messenger	✓ Real-Time Chat ✓ File Send ✓ Group Chats ✓ Voice Message ✓ Video Message ✓ Encrypted Conversation ✓ Send Map	Android Web Iphone	End-to-End (TLS) to encrypt messages in transit over the network. RSA AES
Telegram	✓ Alerts/Notifications ✓ Audio Calls ✓ Chat/ Messaging ✓ Collaboration Tools ✓ File Sharing ✓ Real-Time Chat ✓ Video Conferencing ✓ Video Support	Android Web Iphone	MTProtocol (Combination of Symmetric and Asymmetric), it include AES for encrypting message content and Diffie Hellman Key exchange for establishing secure communication channel.
Viber	✓ Batch Communication ✓ Chat/ Messaging ✓ Communication Management ✓ Customer History ✓ Group Management ✓ Interactive Content ✓ Live Chat ✓ Video Support	Android Web Iphone	End-to-End AES RSA

CHAPTER IV

RESULTS AND DISCUSSION

This chapter presents data, statistical analysis, and interpretation of the study, aiming to clarify its significance and provide insights for future research. The researchers aim to answer questions about the developed system and provide insights for future studies. The assessment includes levels of respondents' evaluation of the system, including functionality, usability, efficiency, security, compatibility, maintainability, and portability.

Evaluation and Scoring

System performance and quality can be assessed using ratings ranging from poor to excellent. The levels of scoring are as follows:

Table 3. Level of Scoring and Verbal Interpretation

Levels of Scoring	Range of Average Weighted Mean	Categorical Response	Verbal Interpretation
5	4.21 - 5.00	Excellent (E)	Excellent
4	3.41 – 4.20	Very Good (VG)	Very Good
3	2.61 – 3.40	Good (G)	Good
2	1.81 – 2.60	Satisfactory (S)	Satisfactory
1	1.00 – 1.80	Poor (P)	Poor

Table 4. Respondents Frequency and Percentage based on Gender

Sex	Frequency	Percentage (%)
Male	16	64
Female	9	36
Total	**25**	**100%**

Table 5. Age of Respondents

Age Range	Frequency	Percentage (%)
18 – 22	11	44
23 – 27	12	48
28 – 32	0	0
33 & above	2	8
Total	**25**	**100%**

Table 6. General Users and IT Professionals

Respondents	Frequency	Percentage (%)
General Users	15	60
IT Professionals	10	40
Total	**25**	**100%**

Table 7. Respondents Level of Acceptance Based on Functional Suitability

Functional Suitability	IT Professionals		General Users	
	Mean	Verbal interpretation	Mean	Verbal interpretation
Functional Completeness	4.70	Excellent	4.33	Excellent
Functional Correctness	4.80	Excellent	4.47	Excellent
Functional Appropriateness	4.90	Excellent	4.67	Excellent
Overall Weighted Mean	**4.80**	**Excellent**	**4.49**	**Excellent**

Table 7 presents the evaluation of the system's functional suitability by IT Professionals and General Users. The overall weighted means are 4.80 and 4.49, respectively, which are both verbally interpreted as "Excellent (E)." The data indicate that the system effectively meets functional expectations, with no significant differences between the two groups' assessments.

Table 8. Respondents Level of Acceptance Based on Usability

Usability	IT Professionals		General Users	
	Mean	Verbal interpretation	mean	Verbal interpretation
Appropriateness Recognizability	4.70	Excellent	4.8	Excellent
Learnability	4.80	Excellent	4.53	Excellent
User Interface Aesthetics	4.80	Excellent	4.47	Excellent
Operability	4.70	Excellent	4.53	Excellent
User Error Protection	4.60	Excellent	4.60	Excellent
Overall Weighted Mean	**4.72**	**Excellent**	**4.59**	**Excellent**

Table 8 presents the evaluation of the system's usability by IT Professionals and General Users. IT Professionals rated the system with an overall weighted mean of 4.72, while General Users gave an overall weighted mean of 4.59, both verbally interpreted as

“Excellent." All usability aspects received the highest ratings, classified as "Excellent." These results highlight the system's usability strengths, particularly in ease of learning and recognizability.

Table 9. Respondents Level of Acceptance Based on Efficiency

Efficiency	IT Professionals		General Users	
	Mean	Verbal interpretation	Mean	Verbal interpretation
Time Behavior	4.80	Excellent	4.53	Excellent
Resource Utilization	4.80	Excellent	4.27	E Excellen
Capacity	4.60	Excellent	4.20	Very Good
Overall Weighted Mean	**4.73**	**Excellent**	**4.33**	**Excellent**

Table 9 shows the evaluation of the system's efficiency by IT Professionals and General Users. IT Professionals gave a mean score of 4.73, while General Users gave a mean score of 4.33, both deemed "Excellent." The highest score was given to time behavior, indicating strong system performance in handling time behavior requirements.

Table 10. Respondents Level of Acceptance Based on Security

Security	IT Professionals		General Users	
	Mean	Verbal interpretation	Mean	Verbal interpretation
Confidentiality	4.70	Excellent	4.33	Excellent
Integrity	4.90	Excellent	4.80	Excellent
Authenticity	4.70	Excellent	4.67	Excellent
Accountability	4.90	Excellent	4.33	Excellent
Overall Weighted Mean	**4.80**	**Excellent**	**4.53**	**Excellent**

Table 10 presents the evaluation of the system's security by IT Professionals and General Users. The overall weighted means are 4.80 and 4.53, respectively, both verbally interpreted as "Excellent." Accountability and Integrity received the same highest score

from IT Professionals (4.90, Excellent), while General Users rated Integrity (4.80, Excellent) as the top criteria, highlighting reliable security.

Table 11. Respondents Level of Acceptance Based on Compatibility

Compatibility	IT Professionals		General Users	
	Mean	Verbal interpretation	Mean	Verbal interpretation
Co-Existence	4.50	Excellent	4.47	Excellent
Interoperability	4.80	Excellent	4.67	Excellent
Overall Weighted Mean	**4.65**	**Excellent**	**4.57**	**Excellent**

Table 11 outlines the evaluation of the system's compatibility by IT Professionals and General Users. IT Professionals provided an overall weighted mean of 4.65, and General Users gave 4.57, both verbally interpreted as "Excellent." These results indicate that the system demonstrates solid compatibility in co-existence and interoperability across both groups.

Table 12. Respondents Level of Acceptance Based on Maintainability

Maintainability	IT Professionals		General Users	
	Mean	Verbal interpretation	Mean	Verbal interpretation
Modifiable	4.50	Excellent	4.27	Excellent
Testability	4.80	Excellent	4.47	Excellent
Overall Weighted Mean	**4.65**	**Excellent**	**4.37**	**Excellent**

Table 12 presents the evaluation of the system's maintainability by IT Professionals and General Users. The overall weighted means are 4.65 and 4.37, respectively, both verbally interpreted as "Excellent." The results highlight the system's strong maintainability, particularly in testability, with consistent assessments from both groups.

Table 13. Respondents Level of Acceptance Based on Portability

Portability	IT Professionals		General Users	
	Mean	Verbal interpretation	Mean	Verbal interpretation
Adaptability	4.60	Excellent	4.33	Excellent
Installability	4.90	Excellent	4.73	Excellent
Replaceability	4.60	Excellent	4.27	Excellent
Overall Weighted Mean	**4.70**	**Excellent**	**4.44**	**Excellent**

Table 13 provides the evaluation of the system's portability by IT Professionals and General Users. The overall weighted means are 4.70 and 4.44, respectively, both verbally interpreted as "Excellent." Installability received the highest score from IT Professionals (4.90, Excellent) and 4.73 remarked also as Excellent from General Users, highlighting reliable portability.

CHAPTER V

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

This section summarizes the main findings of the study, offering a thorough synopsis of the results, their implications, and useful recommendations for further enhancement of the developed system.

Summary of Findings

The system was evaluated using the ISO 25010 questionnaire by IT professionals (40%) and general users (60%).

Table 14. Overall Weighted Mean

Categories	Overall Weighted Mean	Ranking
Security	4.67	1
Usability	4.65	2
Functionality	4.64	3
Compatibility	4.61	4
Portability	4.57	5
Efficiency	4.53	6
Maintainability	4.51	7

Table 14 illustrates the overall weighted mean. Upon conducting the survey by using the ISO 25010 questionnaire, results showed high ratings across various attributes: functionality (4.64), usability (4.65), efficiency (4.53), security (4.67), compatibility (4.61), maintainability (4.51), and portability (4.57). These overall weighted means indicate the system's capabilities and performance in all attributes as well as meeting user requirements.

Conclusions

Upon concluding the study, the researchers came up with these conclusions that further support the viability of the study:

1. The study successfully ensured data security and confidentiality in communication by implementing the RSA and AES encryption algorithms. These algorithms achieved positive scores for confidentiality, integrity, authenticity, and accountability while offering security against data threats. The system effectively protected user data by utilizing these encryption algorithms, demonstrating its ability to handle security issues.
2. To further secure communication, the system incorporated measures against internet threats and unauthorized access. These measures proved highly effective in mitigating risks. With its security features, the system demonstrated a high degree of reliability, signifying defense against potential cyber threats.
3. A key objective of the study was to design a user-friendly interface that ensures ease of navigation and an enjoyable user experience. The interface received positive ratings for its usability. These results confirmed that the application

successfully delivered a smooth and intuitive user experience and met user expectations.

4. The implementation of user authentication mechanisms further strengthened the system's security by accurately verifying the identities of message senders and receivers. This prevented impersonation and unauthorized access, significantly improving the integrity of communications.
5. The system's performance was assessed using the ISO 25010 based questionnaires. Evaluations by IT professionals and general users revealed consistently high ratings across all attributes, including functionality, usability, efficiency, compatibility, maintainability, and portability. These findings validated the system's capabilities while also highlighting its potential for further improvement. Overall, the study successfully fulfilled its objectives, offering a secure, efficient, and user-friendly communication platform.
6. By following the Kanban methodology, the system development adhered to established standards, facilitating iterative improvements. The findings of this study highlight the system's overall effectiveness and provide a foundation for future enhancements, ensuring continued compliance with evolving user and technological needs.

Recommendations

The following recommendations are based on this study.

1. Future versions should focus on creating a more intuitive and customizable user interface, allowing users to personalize their experience. Adding fun features, such as games, can also enhance engagement.
2. Implementing advanced security features like face unlock will improve account security while offering a more convenient way for users to access their accounts.
3. Adding a 'show password' option during sign-up will help users verify their passwords, reducing input errors and making the registration process smoother.
4. The app should be available on iOS to reach a broader audience and provide a consistent experience for Apple device users.

BIBLIOGRAPHY

Dr. Priyanka Dubey. (2022). *Secure Messaging Application*.

https://www.ijset.in/wp-content/uploads/IJSET_V10_issue3_245.pdf

Saad Najim Alsaad. (2020). *Instant messaging security and privacy secure instant messenger design*.

https://www.researchgate.net/publication/343586226_Instant_messaging_security_and_privacy_secure_instant_messenger_design.

A. Balapour, H.R Nikkhah, R. Sabherwal. (2020). *Mobile application security: Role of perceived privacy as the predictor of security perceptions.*

https://www.researchgate.net/publication/338520954_Mobile_application_security_Role_of_perceived_privacy_as_the_predictor_of_security_perceptions

A.F Chernyavskiy, E.I Kozlova, and Yu. A. Chernyavskiy (2024*) Features of Structural Hardware Transformation of Information in Cryptosystems.*

https://www.researchgate.net/publication/385327901

Y. Liu, B. Dantas Cruz, and E. Tilevich (2022). *Secure and Flexible Message-Based Communication for Mobile Apps within and Across Devices.*

https://people.cs.vt.edu/~tilevich/papers/HTPD_Journal.pdf?utm

Quiazon, M. C. D., Manlutac, B. D. (2024). *Stream shield: An Anti-Piracy Movie Streaming Android Application with Screen Recording detection and Integrated Media Content Protection using Advance Encryption Standard (AES) Algorithm.*

https://doi.org/10.52783/jes.2422

Rohit Verma, Aman Kumar Sharma (2020). *Cryptography: Avalanche Effect of AES and RSA*

https://www.ijsrp.org/research-paper-0420/ijsrp-p10013.pdf

Edwin Kho et al. (2020). *Web-based Live Chat Application uses Advanced Encryption Standard Methods and Rivest Shamir Adleman*

https://iopscience.iop.org/article/10.1088/1757-899X/1007/1/012153/pdf

Abhishek G. and Asha B. (2023). *Hybrid Multiple Cryptography for Data Encryption*

https://ieeexplore.ieee.org/document/10192838

A. Kousalya and Nam-Kyun Baik (2023). *Enhance cloud security and effectiveness using improved RSA-based RBAC with XACML technique*

researchgate.net/publication/369427796

I. Lukman, A. Arief, A.A. Ilham and Syafaruddin (2023). *Improved Data Security using Advanced Encryption Standard (AES) Algorithm on long-range communication system at smart grid*

https://www.researchgate.net/publication/369453664

N. Namassivaya et al. (2023). *Encrypted chat application using RSA Algorithm*

www.ijetms.in/Vol-7-issue-2/Vol-7-Issue-2-95.html

Zarni S. et al (2020). *Secure SMS System using RSA Encryption Based on Android platform*

https://www.researchgate.net/publication/341517681

Burgos Aeron B. et al. (2022). *Web-based Chat Application uses Advanced Encryption Standard Methods*

https://tinyurl.com/rbd7b6pd

M.H.M Baig, H. B. Haq and W. Habib (2023). *A Comparative Analysis of AES, RSA, and 3DES Encryption Standards based on Speed and Performance*

https://msa-journal.org/journal/article/view/4

Sahoo, P. Mohanty, and P.C. Sethi (2022). *Image Encryption Using RSA Algorithm*

https://link.springer.com/chapter/10.1007/978-981-19-0901-6_56

Dr. K. Balasubramanian, M. Arun and Dr. K. R. Sekar (2024). *An Improved RSA Algorithm for Enhanced Security*

https://www.ijcns.latticescipub.com/portfolio-item/b1421112222/

H. Bodur and R. Kara (2020). *Secure SMS Encryption Using RSA Encryption Algorithm on Android Message Application*

https://tinyurl.com/4m5yd75a

J. Williams, E.O. Bennett and V.I. Anireh (2024*). An efficient algorithm for text encryption on android devices*

researchgate.net/publication/382188066

Priam Nepomuceno (2020). *AFP developing encryption capability for classified comms*

https://shorturl.at/v3ePT

APPENDICES

Approved Proposed Title

Taguig City University
Gen. Santos Ave., Centra Bicutan, Taguig City
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY

COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY

THESIS 101

A.Y. 2024-2025

Date: February 20, 2025

Degree: **BACHELOR OF SCIENCE IN COMPUTER SCIENCE**

Name of Proponents:		
	Amparo, Paul Victor M.	Student ID No.: 20-00994
	Batarina, Randel B.	Student ID No.: 20-00940
	Pariado, Keem R.	Student ID No.: 20-01015
	Rodriguez, Clark F.	Student ID No.: 20-02524

Proposed Title:

Safe-on-Chat: A Mobile Messaging and Video Call Application With Security Features Using Rivest Shamir Adleman And Advance Encryption Standard Algorithm

Endorsed By:

PROF. JACKELYN T. TRINIDAD, MSIT
Subject Adviser

PROF. EDMAR G. TAN, MSIT
Technical Adviser

Recommending Approval:

PROF. HAIDEE A. VIÑALON, MIT
Research Coordinator

PROF. RODELIO G. DELA FUENTE, MSIT
Program Chair

DR. REYNALDO G. ALVEZ
Dean, College of Information and Communication Technology

Letter to the Technical Adviser

Taguig City University
Gen. Santos Ave., Centra Bicutan, Taguig City
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY

February 20, 2025

PROF. EDMAR G. TAN, MSIT
Professor, College of Information and Communication Technology
Taguig City University
Taguig City, Philippines

Dear Sir/Ma'am:

Greetings!

We, the fourth-year students of Taguig City University pursuing a Bachelor of Science degree in Computer Science, are currently enrolled in Thesis Writing 2 with the proposed thesis entitled "**Safe-on-Chat: A Mobile Messaging and Video Call Application With Security Features Using Rivest Shamir Adleman And Advance Encryption Standard Algorithm**".

We request your service and expertise to serve as an adviser for our thesis. Your knowledge and insights will be valuable and will greatly enrich our work.

We appreciate your consideration, and we hope you can fulfill our request.

Respectfully yours,

The Proponents

PAUL VICTOR M. AMPARO
KEEM R. PARIADO
RANDEL B. BATARINA
CLARK F. RODRIGUEZ

Noted by:

PROF. JACKELYN T. TRINIDAD, MSIT
Subject Adviser

Conforme:

PROF. EDMAR G. TAN, MSIT
Professor, College of Information and Communication Technology

Letter of Interview

Taguig City University
Gen. Santos Ave., Centra Bicutan, Taguig City
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY

February 20, 2025

PROF. RODELIO O DELA FUENTE, MSIT

Program Chair, Bachelor of Science in Computer Science

Taguig City University

Taguig City, Philippines

Dear Sir Rodelio,

Greetings!

We, the fourth-year students of Taguig City University pursuing a Bachelor of Science degree in Computer Science, are currently enrolled in Thesis Writing 2 with the proposed thesis entitled "**Safe-on-Chat: A Mobile Messaging and Video Call Application With Security Features Using Rivest Shamir Adleman And Advance Encryption Standard Algorithm**" would like to request for your approval in commencing an educational survey or interview to all respective respondents from all categories.

We would also like to request your approval in commencing educational interview or survey to IT personnel. By giving approval to our requests, we can move forward to our study, modify the problems that the user may get and strengthen our knowledge when it comes to security handling.

Thank you for your careful consideration and we hope you will be able to fulfill our humble requests.

Respectfully yours,

The Proponents

PAUL VICTOR M. AMPARO
KEEM R. PARIADO
RANDEL B. BATARINA
CLARK F. RODRIGUEZ

Noted by:

PROF. JACKELYN T. TRINIDAD, MSIT
Subject Adviser

Survey Questionnaire

The researchers will use the ISO 25010 questionnaire format via likert scale to gather data, to identify what will be the successful side of the proposed system and what are the downside of the proposed system. The given table below will be the initial format for the possible questionnaires that will be used to gather accurate data.

Table 14. Likert Scale for ISO 25010 Questionnaire

5 – Excellent, **4** – Very Good, **3** - Good, **2** – Satisfactory, **1** – Poor

1. FUNCTIONALITY SUTABILITY		**5**	**4**	**3**	**2**	**1**
1.1 Completeness	The degree to which the messaging application system has the correct features and functions.					
1.2 Correctness	The degree to which the messaging application system's ability to meet the necessary requirements for its intended functions.					
1.3 Appropriateness	The degree to which the messaging application's capacity to offer essential features and functions to meet user needs and perform assigned tasks.					

2. USABILITY		**5**	**4**	**3**	**2**	**1**
2.1 Appropriateness Recognizability	The degree to which the user interface of messaging apps is valued for its clarity, usability, and accessibility, as well as for having icons and symbols that are perfect for messaging settings.					

2.2 Learnability	The degree to which users find the messaging app simple to learn and use—even those who are unfamiliar with messaging apps.					
2.3 User Interface Aesthetics	The degree to which the messaging application offers an aesthetically pleasing, simple-to-use, and trendy user interface.					
2.4 Operability	The degree to which the users find the messaging app simple to use and manage with easy-to-comprehend error messages and helpful advice when problems occur.					
2.5User Error Protection	The degree to which the messaging application protects users from user error by preventing users from making mistakes and helping them in fixing the errors they make.					

3. EFFICIENCY		**5**	**4**	**3**	**2**	**1**
3.1 Time Behavior	The degree to which the messaging application responds quickly to user actions and delivers messages on time.					
3.2Resource Utilization	The degree to which users consider the messaging application effectively utilizes the resources without slowing down the system.					
3.3 Capacity	The degree to which the app's user perception of its ability to handle multiple users and large messages without slowing down or crashing.					

4. SECURITY		**5**	**4**	**3**	**2**	**1**
4.1 Confidentiality	The degree to which the messaging application secures the confidentiality of user messages and personal data.					
4.2 Integrity	The degree to which the messaging application protects the integrity of user messages and personal information.					
4.3 Authenticity	The degree to which the messaging application offers authenticity features so that users can confirm the sender of messages and make sure that messages weren't change while being sent and received.					
4.4 Accountability	The degree to which the messaging application offers accountability features to make sure users can keep track of their activity as well as keep an eye on the actions and activities they perform with application					

5. COMPATIBILITY		**5**	**4**	**3**	**2**	**1**
5.1 Co-existence	The degree to which the messaging application is compatible with other software and systems, such as operating systems, hardware, and other messaging applications.					

5.2 Interoperability	The degree to which the messaging application is able to communicate information using typical formats and protocols with different applications and systems.					

6. MAINTAINABILITY		**5**	**4**	**3**	**2**	**1**
6.1 Modifiable	The degree to which the messaging application is easy to modify or extend to meet changing requirements.					
6.2 Testability	The degree to which the messaging application is easy to test and validate to ensure its functionality and quality.					

7. PORTABILITY		**5**	**4**	**3**	**2**	**1**
7.1 Adaptability	The degree to which the messaging application is adaptable to changes in its operating environment, such as new devices or platforms.					
7.2 Installability	The degree to which the messaging application is easy to install and set up on a user's device.					
7.3 Replaceability	The degree to which the messaging application is easily replaceable by another similar application without disrupting the user's workflow.					

User's Manual/ Screenshots of the System

This user manual serves as a guide to users on how to properly use the system. Researchers provide screenshots for clearer instructions to ensure the system's maximum functionality.

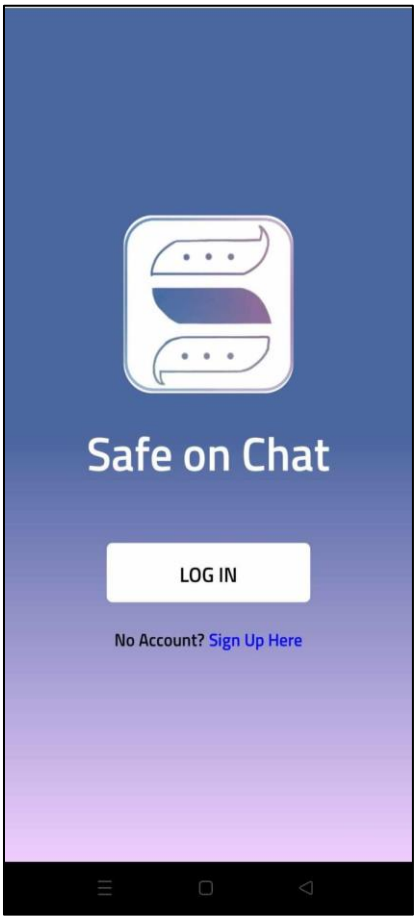


Figure 20. Start Up Page

Figure 20 shows the Start-Up page. This is the first page that appears when the user opens the application. There is a login button that allows the user to log in to their account. If the user does not have an account yet, they can create one by clicking the "Sign Up Here" link, which will direct them to the Sign-Up page.

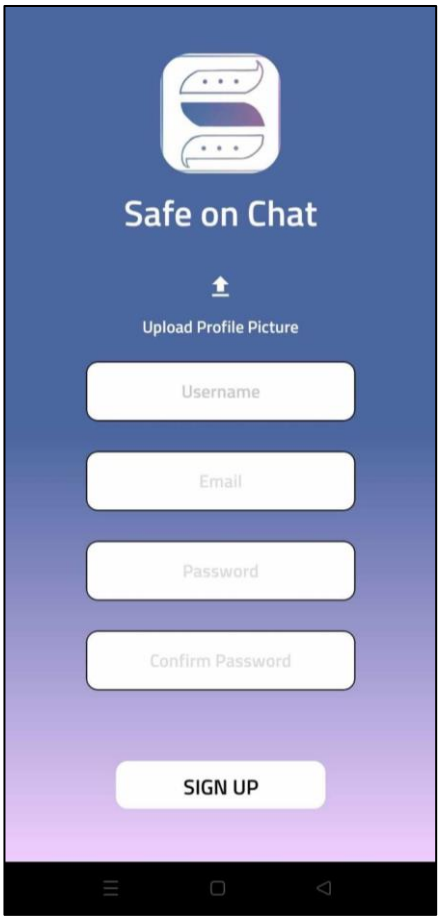


Figure 21. Sign Up Page

Figure 21 shows the Sign-Up page, where users can create an account by providing the credentials. To successfully create an account, the user must complete several steps. First, they need to upload a profile picture, which will serve as their visual identity within the application. Next, they must choose a unique username that will distinguish them from other users. Additionally, a verified email address is required, as a verification link will be sent to confirm the authenticity of the account. Lastly, the user must create a strong password that meets the application's password requirements. Once all the required information is provided and verified, the user can proceed with the registration process and gain access to the application's features.

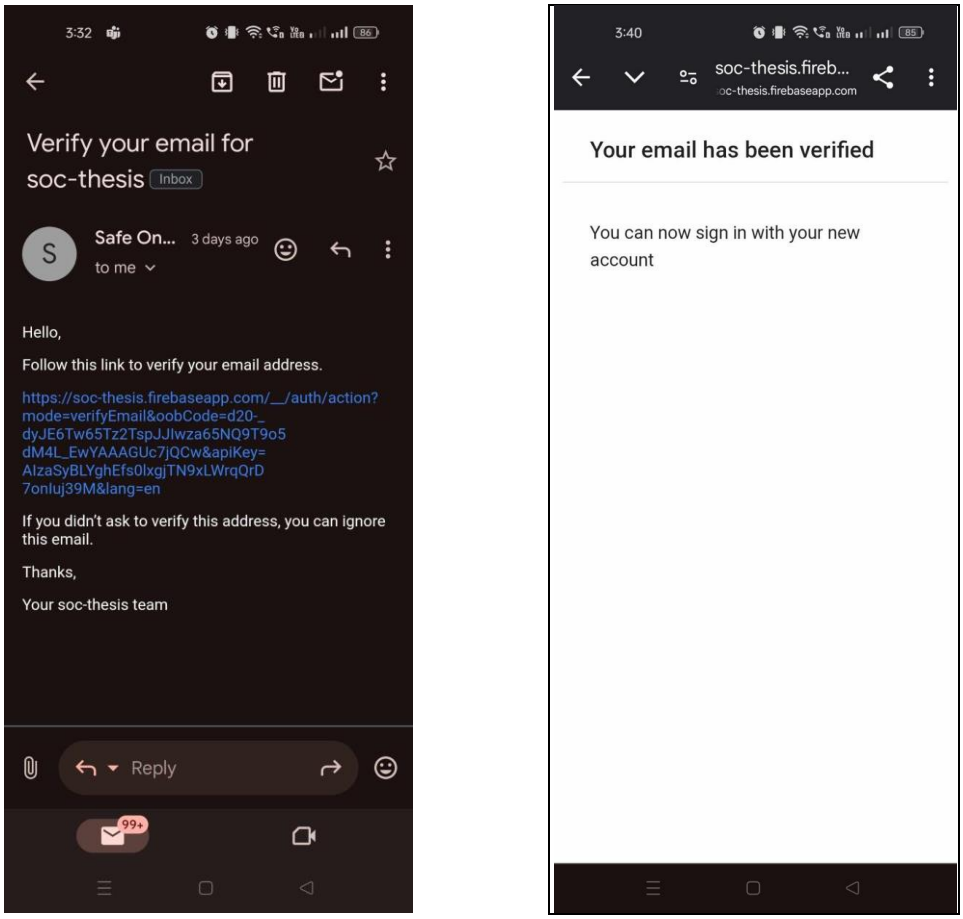


Figure 22. Email Verification Page

Figure 22 shows the email verification page. This will only be accessible if the user's email is fully registered in the application. Upon successful registration, the user will receive an email confirmation containing a verification link. By clicking the link, the user can verify their email address and activate their account. This process ensures that only valid and accessible email addresses are used, enhancing the security of the application. If the user does not receive the email, they can check their spam folder or request a new verification link.

Figure 23. Security Authentication Page

Figure 23 shows the security authentication page. This page allows users to authenticate their identity using a PIN or biometric access, such as fingerprint. After successfully creating an account, the application will prompt the user to set up their preferred authentication method. Once configured, the application will request the user's fingerprint or PIN each time they open the app to ensure secure access. This additional layer of protection helps safeguard user accounts from unauthorized access and enhances overall security.

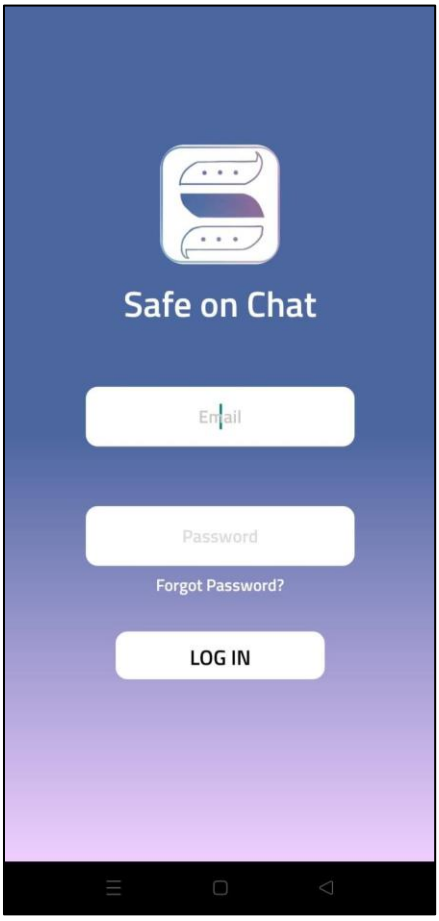


Figure 24. Log-in Page

Figure 24 shows the log-in page. This page is where users can log in to their accounts if they have been fully verified. They can enter their registered email address and password to access the application. If users forget their password, they can click the "Forgot Password" link to initiate the password recovery process. Additionally, if their account has not been verified, they may need to complete the email verification step before gaining full access.

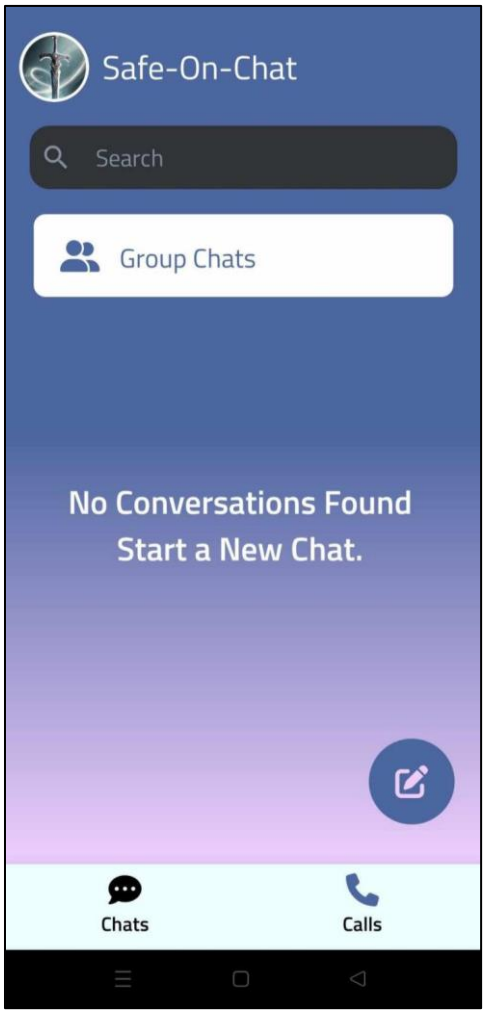


Figure 25. Home Page

Figure 25 shows the home page. After successfully creating an account and completing the authentication process, users will be directed to this page. This is where the chat history is displayed, allowing users to search for past conversations. On the lower side of the screen, there is a pen icon, which enables users to start a new conversation. Additionally, there is a call icon that provides access to the call history, allowing users to review their past calls.

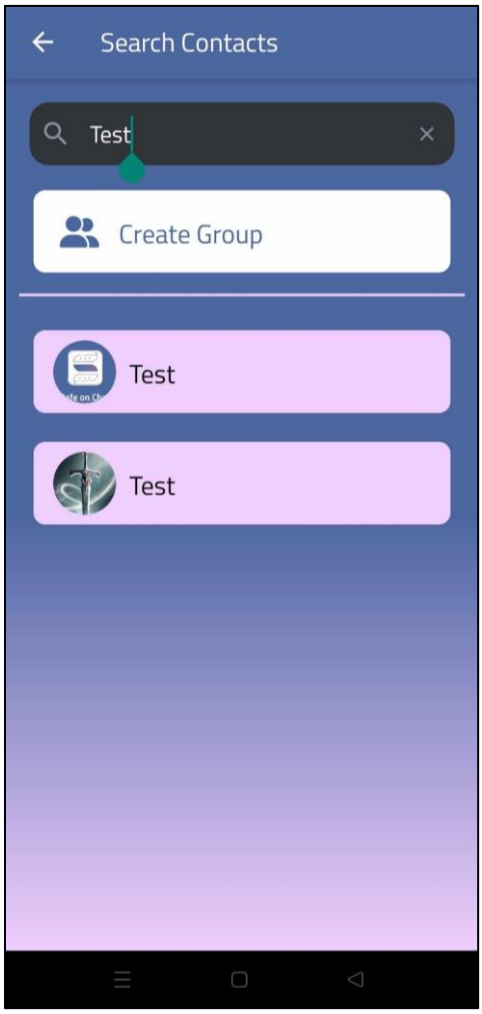


Figure 26. Search User

Figure 26 shows the search user. Users can type the username of other users in the search bar to be able to start a conversation with them. The search results will display the matching users along with their usernames and profile pictures, making it easier to identify the correct person. This feature helps users of the proposed application to quickly connect with friends, colleagues, or new contacts within the application

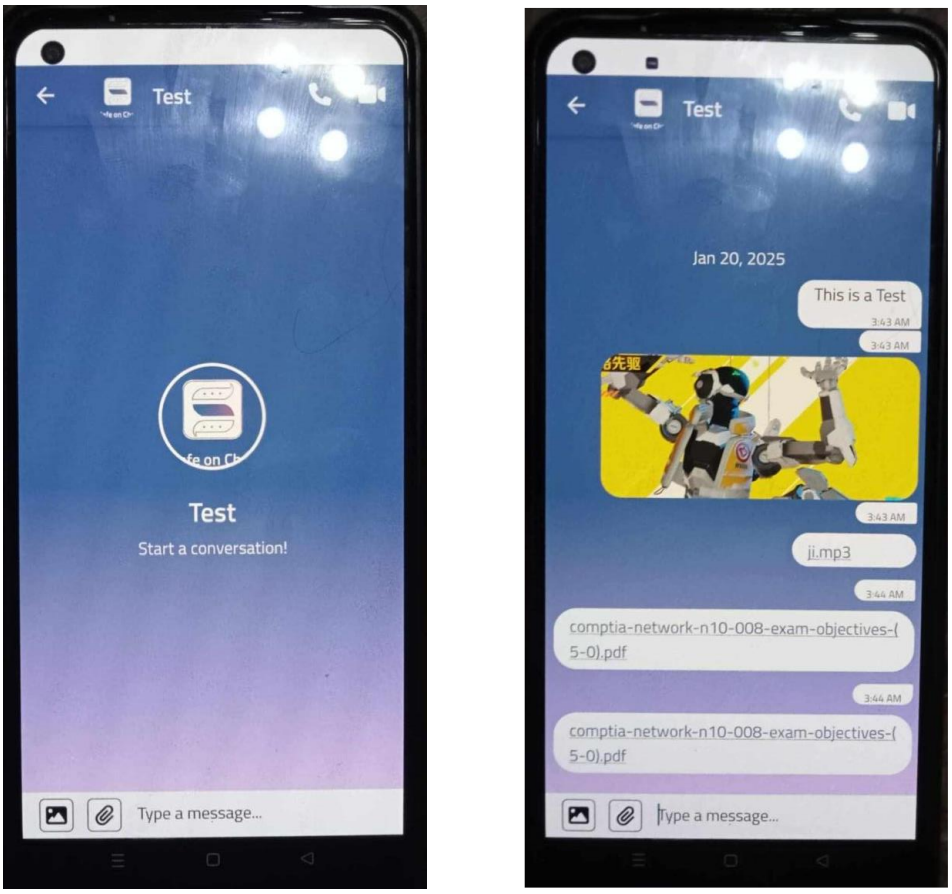


Figure 27: Chat Page

Figure 27 shows the chat page. This is where conversations are exchanged between the sender and the recipient. Each conversation within the chat room is encrypted using RSA and AES algorithms to ensure secure communication. On this page, screen capture and screen recording are disabled to protect the confidentiality of the conversation. In addition to text messages, the sender can also share images and files. On the upper right side, there are call and video call icons, allowing users to initiate voice and video calls

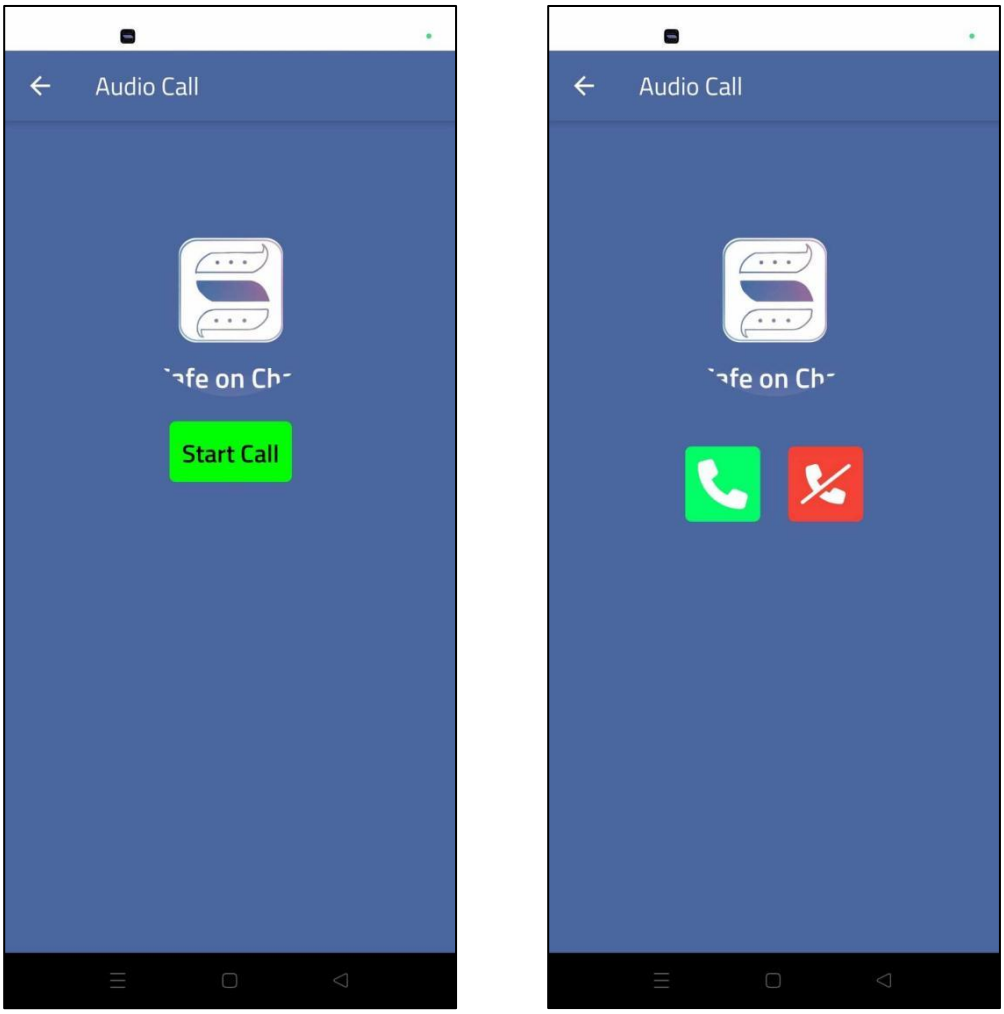


Figure 28. Audio Call

Figure 28 shows how a user can initiate an audio call with another user. To access the audio call feature, the user must click on the call icon and press the "Start Call" button to begin the call. The recipient must then also press the call icon and tap the "Start Call" button to initiate their connection. To accept the call, the recipient must press the green button to connect with the other user. If the user wants to exit the call, they can press the red button to automatically end the conversation.

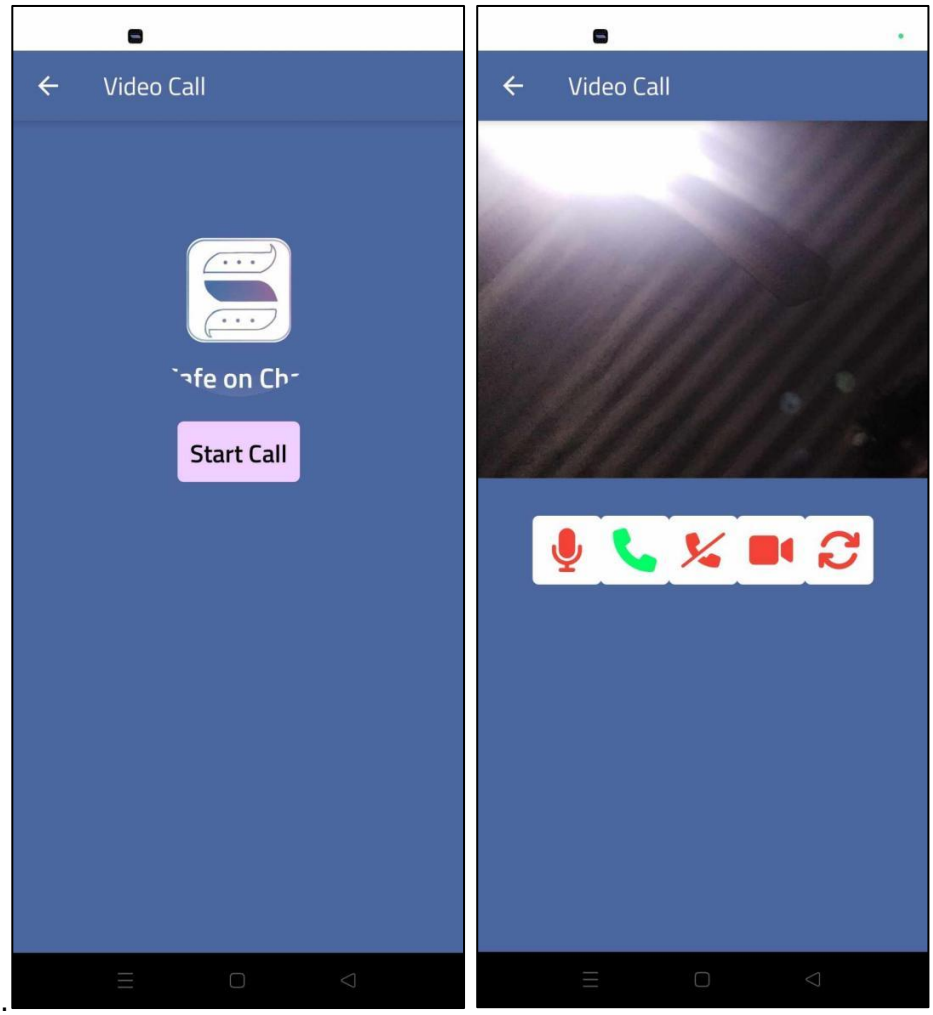


Figure 29. Video Call

Figure 29 shows the video call page. To access the video call feature, the user must click on the camera icon and press the "Start Call" button to begin the call. The recipient must then also press the camera icon and tap the "Start Call" button to initiate their connection. Either user must press the green button to establish the connection. During the call, users have the option to mute their microphone, turn off their camera, or switch between front and rear cameras. If a user wants to exit the call, they can press the red button to automatically end the conversation.

Figure 30. Call History

Figure 30 shows the call history page. This where the user of the application can see all the call history they made with the other user of the application. Whenever a user calls another user, it will automatically track it and it will be displayed in the calls history. The user will tap the Calls icon at the lower bottom right side of the screen. This feature helps users keep track of their call activity by providing a simple and accessible way to view their call history within the application.

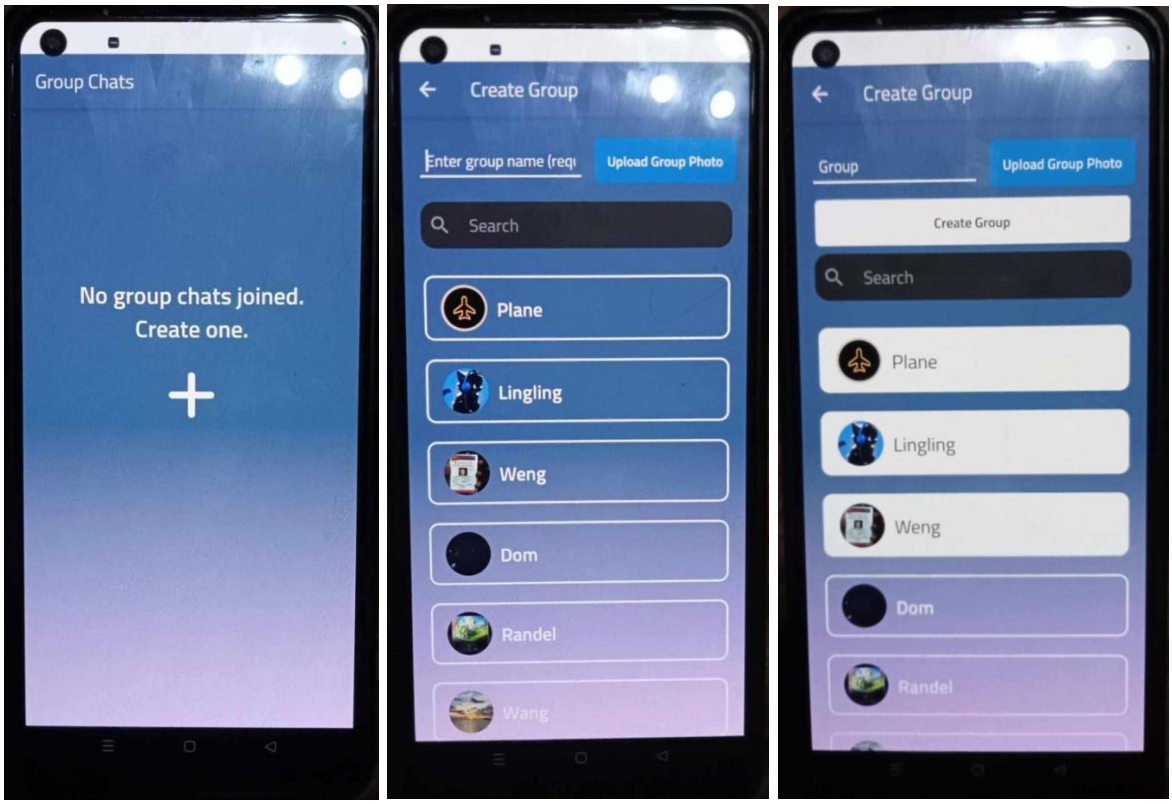


Figure 31. Creating a Group Chat

Figure 31 shows the creation of a Group Chat. The application allows users to have conversations with multiple members in a single chat. To start a group conversation, a few simple steps must be followed. First, click the plus icon. This will direct the user to a list of registered users within the application. The user must then select the members they want to add to the group. For convenience, a search bar is available to quickly find specific users. Next, enter a name for the group chat and upload a profile picture for the group. After completing these steps, tap the "Create" button to finalize the group chat setup.

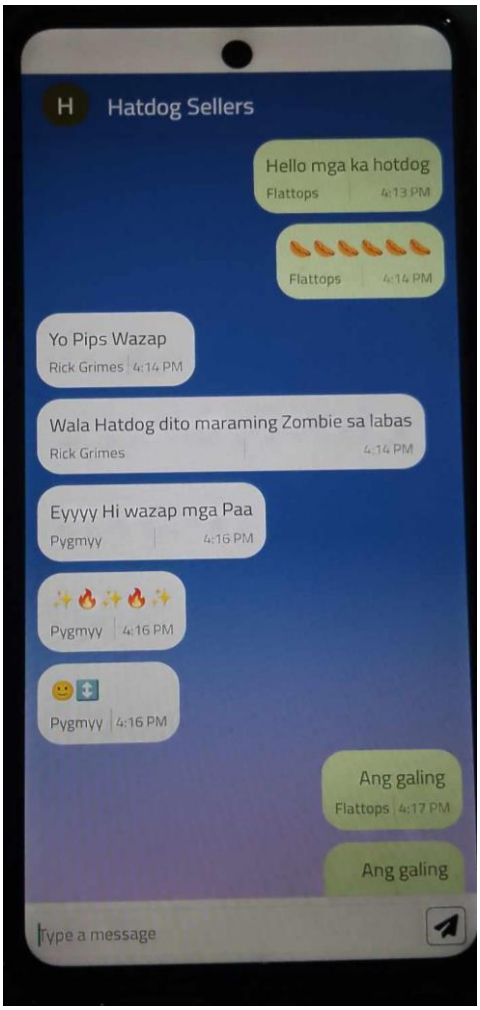


Figure 32. Group Chat Page

Figure 32 shows the Group Chat Page. After successfully creating a group user can start a conversation with other added group chat members. Users can send their messages, and the other users will be able to see the message inside the group chat. The system ensures that messages are delivered in real-time, allowing for smooth and continuous communication among group members. Additionally, screen capture and screen recording are disabled on this page to protect the privacy and security of the conversations.

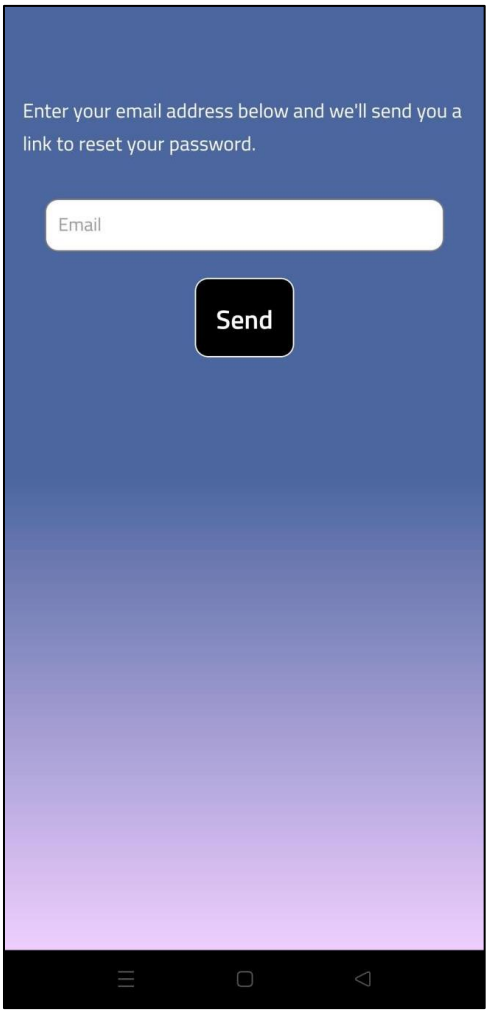


Figure 33. Reset Password

Figure 33 shows the user can reset their password by tapping forgot password found below the password field, in the case that they forgot their password. Users are prompted to input their verified email to be able to reset user password. User must go to his verified email to check if he gets link to reset their password, if the user gets the link click the link and to proceed to password reset.

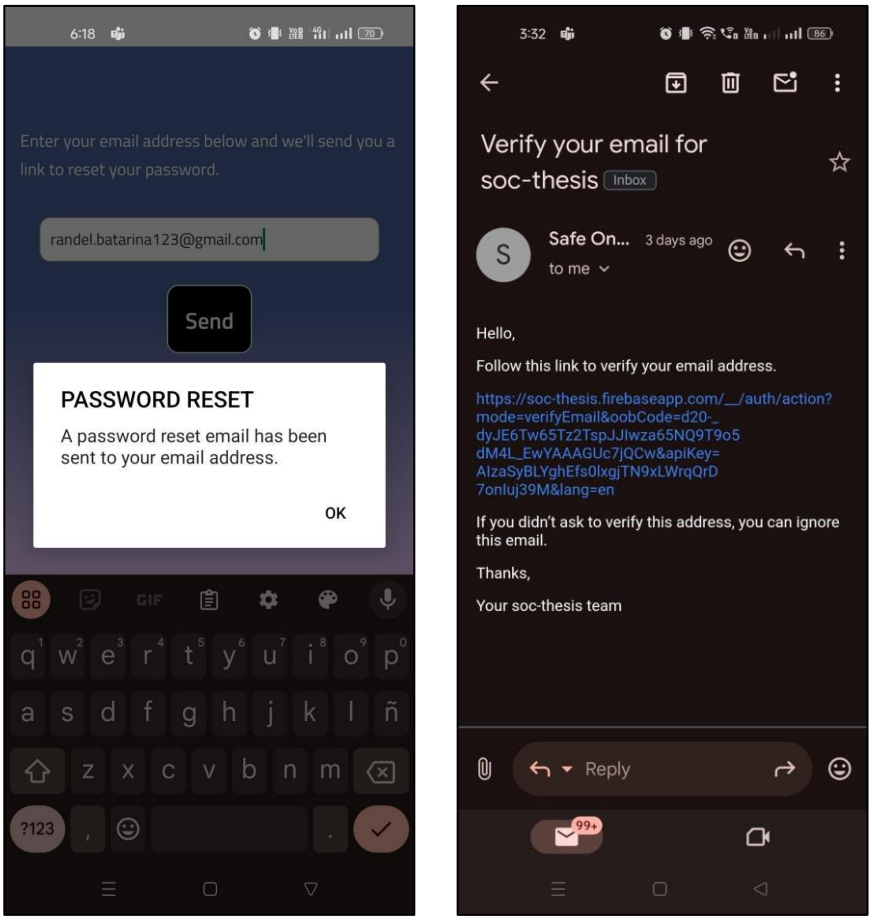


Figure 34. Reset Password Verification

Figure 34 shows the registered email page, where users can enter their verified email address to initiate the password reset process. Once the user submits their verified email, they will receive an email confirmation containing a password reset link. By clicking the link, they will be redirected to a page where they can create a new password. This ensures that only authorized users can regain access to their accounts. If the email does not arrive within a few minutes, users should check their spam or junk folder.

Program Code

```
import React, { useCallback, useEffect, useRef, useState } from 'react';
import { View, Image, Text, Button, StyleSheet, Pressable } from 'react-native';
import { mediaDevices, RTCPeerConnection, RTCSessionDescription } from 'react-native-webrtc';
import io from 'socket.io-client';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faPhoneSlash, faPhone } from '@fortawesome/free-solid-svg-icons';
import { sendAudioCallNotification, getPushTokenForUser, getUserDetailsForNotification } from './Notifications';
import { useFocusEffect } from '@react-navigation/native';
import { app } from '../firebaseConfig';
import { getAuth } from 'firebase/auth';

const AudioCallScreen = ({ route, navigation }) => {
  const { user, username, profilePicture, callerInfo } = route.params;
    const [localStream, setLocalStream] = useState(null);
    const [remoteStream, setRemoteStream] = useState(null);
    const [isConnected, setIsConnected] = useState(false);
    const [callStarted, setCallStarted] = useState(false);
    const socketRef = useRef(null);
  const pcRef = useRef(null);
  const auth = getAuth(app);

  useFocusEffect(
    useCallback(() => {
      if (callerInfo) {
        initializeSocket();
      }
      if (route.params.autoStart) {
        startLocalStream();
      }
    }, [callerInfo, route.params.autoStart])
  );


  useEffect(() => {
    return () => {
      endCall();
    }
  }, []);



  const initializeSocket = () => {
    const socket = io('https://soc-thesis.onrender.com');
    socketRef.current = socket;

    socket.on('connect', () => {
      console.log('Connected to signaling server');
      setIsConnected(true);
    });

    socket.on('offer', async (offer) => {
      console.log('Received offer:', offer);
      if (!pcRef.current) {
        pcRef.current = createPeerConnection();
      }
      try {
        await pcRef.current.setRemoteDescription(new RTCSessionDescription(offer));
        const answer = await pcRef.current.createAnswer();
        await pcRef.current.setLocalDescription(answer);
        socket.emit('answer', answer);
      } catch (error) {
        console.error('Error handling offer:', error);
      }
    });

    socket.on('answer', async (answer) => {
      console.log('Received answer:', answer);
      if (pcRef.current) {
        try {
          await pcRef.current.setRemoteDescription(new RTCSessionDescription(answer));
        } catch (error) {
          console.error('Error setting remote description:', error);
        }
      }
    });

    socket.on('ice-candidate', async (candidate) => {
      console.log('Received ICE candidate:', candidate);
      if (pcRef.current) {
        try {
          await pcRef.current.addIceCandidate(candidate);
        } catch (error) {
          console.error('Error adding ICE candidate:', error);
        }
      }
    });

    return () => {
      socket.disconnect();
      if (pcRef.current) {
```

```
      pcRef.current.close();
      pcRef.current = null;
    }
  };
}

 const createPeerConnection = () => {
      const pc = new RTCPeerConnection({
    iceServers: [
      {
                urls: 'stun:stun.l.google.com:19302'
      },
      {
                urls: "turn:asia.relay.metered.ca:80",
                username: "c1a44fd70f84e2d8ef05b4ac",
                credential: "XO88wG9Y0hTvj0XD",
      },
      {
                urls: "turn:asia.relay.metered.ca:80?transport=tcp",
                username: "c1a44fd70f84e2d8ef05b4ac",
                credential: "XO88wG9Y0hTvj0XD",
      },
      {
                urls: "turn:asia.relay.metered.ca:443",
                username: "c1a44fd70f84e2d8ef05b4ac",
                credential: "XO88wG9Y0hTvj0XD",
      },
      {
                urls: "turns:asia.relay.metered.ca:443?transport=tcp",
                username: "c1a44fd70f84e2d8ef05b4ac",
                credential: "XO88wG9Y0hTvj0XD",
      },
    ],
  });

  pc.onicecandidate = (event) => {
    if (event.candidate) {
      socketRef.current.emit('ice-candidate', event.candidate);
    }
  };

  pc.ontrack = (event) => {
      if (event.streams && event.streams[0]) {
      setRemoteStream(event.streams[0]);
    }
  };

    pc.onconnectionstatechange = () => {
      console.log('Connection state changed:', pc.connectionState);
    if (pc.connectionState === 'closed') {
              console.log('Peer connection is closed');
    }
  };

  return pc;
 };

  const startLocalStream = async () => {
  if (!socketRef.current) {
    initializeSocket();
  }
  try {
      const stream = await mediaDevices.getUserMedia({
      audio: true,
    });
    setLocalStream(stream);
    if (!pcRef.current) {
              pcRef.current = createPeerConnection();
    }
     stream.getTracks().forEach(track => {
      pcRef.current.addTrack(track, stream);
    });
    setCallStarted(true);

      const callerInfo = await getUserDetailsForNotification(auth.currentUser.uid);
      const recipientDetails = await getUserDetailsForNotification(user.uid);
       if (recipientDetails && recipientDetails.expoPushToken && callerInfo) {
          console.log('Sending video call notification to:', user.username);
                    await sendAudioCallNotification(recipientDetails.expoPushToken, callerInfo);
    }
  } catch (error) {
          console.error('Error starting local stream:', error);
  }
 };

 const createOffer = async () => {
      if (!pcRef.current || pcRef.current.connectionState === 'closed') {
          console.warn('Peer connection is not available or closed');
    return;
  }

  try {
      const offer = await pcRef.current.createOffer();
                    await pcRef.current.setLocalDescription(offer);
    socketRef.current.emit('offer', offer);
  } catch (error) {
          console.error('Error creating offer:', error);
  }
 };

 const endCall = () => {
```

```
    if (pcRef.current) {
      pcRef.current.close();
      pcRef.current = null;
    }
    if (localStream) {
       localStream.getTracks().f
orEach(track => track.stop());
      setLocalStream(null);
    }
    if (remoteStream) {
       remoteStream.getTracks(
).forEach(track                 =>
track.stop());
      setRemoteStream(null);
    }

    if (socketRef.current) {
       socketRef.current.discon
nect();
      socketRef.current = null;
      setIsConnected(false);
    }
    setCallStarted(false);
  };

  return (
                              <View
style={styles.container}>
      <View style={{
        flexDirection: 'row',
          justifyContent:  'space-
around',
        marginTop: 20,
        width: '50%',
      }}>
        {!callStarted ? (
          <View
            style={{
              alignItems: 'center',
                    justifyContent:
'center',
              marginTop: 50,
            }}
          >
            <Image
                    source={{  uri:
profilePicture }}
              style={{
                width: 150,
                height: 150,
                borderRadius: 75,
                marginBottom: 10,
              }}
            />
                              <Pressable
style={styles.button}
onPress={startLocalStream}>
                                  <Text
style={styles.buttonText}>Sta
rt Call</Text>
            </Pressable>
          </View>
        ) : (
          <>
            <View
              style={{
                alignItems: 'center',
                      justifyContent:
'center',
                marginTop: 50,
              }}
            >
              <Image
                      source={{  uri:
profilePicture }}
                style={{
                  width: 150,
                  height: 150,
                  borderRadius: 75,
                  marginBottom: 10,
                }}
              />
              <View
                style={{
                  flexDirection: 'row',
                      justifyContent:
'space-around',
                  marginTop: 20,
                  width: '100%',
                }}
              >
                <Pressable style={{
                  ...styles.button,
                   backgroundColor:
'#00ff66',
                                    }}
onPress={createOffer}>
                    <FontAwesomeIc
on icon={faPhone} size={40}
color="#fff" />
                </Pressable>
                <Pressable style={{
                  ...styles.button,
                   backgroundColor:
'#f44336',
                                    }}
onPress={endCall}>
                    <FontAwesomeIc
on        icon={faPhoneSlash}
size={40} color="#fff" />
                </Pressable>
              </View>
            </View>
          </>
        )}
      </View>
    </View>
  );
};

const          styles          =
StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
                backgroundColor:
'#4c669f',
    position: 'relative',
  },
  button: {
    padding: 10,
    borderRadius: 5,
    backgroundColor: '#00ff00',
    marginTop: 10,
  },
  buttonText: {
    color: '#000',
    fontSize: 20,
                          fontFamily:
'TitilliumWeb_600SemiBold',
  },
});

export                    default
AudioCallScreen;

import React, { useEffect,
useState, useCallback } from
'react'
import    {    BackHandler,
FlatList, Image, Pressable,
StyleSheet, Text, View } from
'react-native'
import    {    useFonts,
TitilliumWeb_400Regular,
TitilliumWeb_600SemiBold  }
from          '@expo-google-
fonts/titillium-web';
import    {    useNavigation,
useFocusEffect    }    from
'@react-navigation/native';
```

```
import { Avatar } from 'react-native-elements';
import { getAuth } from 'firebase/auth';
import { getFirestore, doc, getDoc, collection, getDocs } from 'firebase/firestore';
import { app } from '../firebaseConfig';
import { SearchBar } from '@rneui/themed';

const Item = ({ user }) => (
  <View style={styles.item}>
    <View style={{
      flexDirection: 'row',
      paddingVertical: 2.5,
      paddingHorizontal: 5,
    }}>
      <View>
            <Avatar size={48} rounded
source={user.profilePicture ? { uri: user.profilePicture } : require('../assets/profilepic.jpg')} />
      </View>
      <View>
        <Text style={{
                    fontFamily: 'TitilliumWeb_400Regular',
          fontSize: 20,
          paddingLeft: 10,
          paddingVertical: 10,
                textAlignVertical: 'center',
        }}>{user.username}</Text>
      </View>
    </View>
  </View>
);

const Calls = () => {
       const [profilePicture, setProfilePicture] = useState('');
  const auth = getAuth(app);
       const firestore = getFirestore(app);
         const [userInput, setUserInput] = useState('');
    const [users, setUsers] = useState([]);
        const [filteredUsers, setFilteredUsers] = useState([]);

  const fetchCalls = async () => {
    try {
        const callsCollection = collection(firestore, 'calls');
      const callSnapshot = await getDocs(callsCollection);
              const callList = callSnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
         const currentUserId = auth.currentUser.uid;
            const userCalls = callList.filter(call => call.uid === currentUserId);
      setUsers(userCalls);
       setFilteredUsers(userCalls);
    } catch (error) {
              console.error('Error fetching calls: ', error);
    }
  };

  const fetchProfilePicture = async () => {
    try {
                const user = auth.currentUser;
      if (user) {
                const userDoc = doc(firestore, 'users', user.uid);
        const userSnap = await getDoc(userDoc);
        if (userSnap.exists()) {
                const userData = userSnap.data();
          setProfilePicture(userData.profilePicture || '')
        } else {
            console.log('No such document!');
        }
      }
    } catch (error) {
              console.error('Error fetching profile picture: ', error);
    }
  };

  useFocusEffect(
    useCallback(() => {
      fetchProfilePicture();
      fetchCalls();
      const onBackPress = () => {
        BackHandler.exitApp();
        return true;
      };

      BackHandler.addEventListener('hardwareBackPress', onBackPress);

      return () =>
        BackHandler.removeEventListener('hardwareBackPress', onBackPress);
    }, [])
  );

       const navigation = useNavigation();

  let [fontsLoaded, fontError] = useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });


      if (!fontsLoaded && !fontError) {
    return null;
  }

  return (
                <View style={styles.container}>
                <View style={styles.content}>
                <View style={styles.header}>
          <Pressable
            onPress={() => {
              navigation.openDrawer();
            }}
            style={({ pressed }) => [
              {
                opacity: pressed ? 0.5 : 1,
```

```
          }, {
            borderRadius: 50,
            borderWidth: 2.5,
             borderColor: 'hsl(0,
0%, 100%)',
          }
         ]}
        >
          <Avatar
            size={48}
            rounded
          source={profilePicture
? { uri: profilePicture } :
require('../assets/profilepic.jp
g')}
          />
        </Pressable>
                              <Text
style={styles.textheader}>
           Safe-On-Chat
        </Text>
      </View>
      <View styles={{
        flexDirection: 'row',
        justifyContent: 'center',
        alignItems: 'center',
        margin: 50,
      }}>

      </View>
      {filteredUsers.length ===
0 ? (
        <View style={{
          flex: 1,
          marginTop: 125,
        }}>
                              <Text
style={styles.temp_text}>You
r Call History Is Empty.
</Text>
                              <Text
style={styles.temp_text}>Initi
ate Your First Call. </Text>
        </View>
      ) : (
        <FlatList
          showsVerticalScrollIn
dicator={false}
          data={filteredUsers}
          renderItem={({ item })
=> <Item user={item} />}
          keyExtractor={item =>
item.id}
          style={{ marginTop: 10,
paddingBottom: 10 }}
          />
        )}
      </View>
    </View>
  )
}

const styles =
StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  content: {
    flex: 1,
    paddingTop: 10,
    paddingBottom: 0,
    paddingLeft: 10,
    padding: 10,
  },
  textheader: {
                              fontFamily:
'TitilliumWeb_400Regular',
    fontSize: 25,
    color: 'hsl(0, 0%, 100%)',
    textAlignVertical: 'center',
    marginLeft: 10,
  },
  profilepic: {
    width: 50,
    height: 50,
    borderRadius: 40,
  },
  header: {
    flexDirection: 'row',
    paddingTop: 15,
    paddingBottom: 10,
  },
  temp_text: {
                              fontFamily:
'TitilliumWeb_600SemiBold',
    fontSize: 25,
    color: '#fff',
    textAlign: 'center',
  },

  item: {
    flexDirection: 'row',
    backgroundColor: '#eef',
    paddingLeft: 10,
    padding: 5,
    marginVertical: 10,
    marginHorizontal: 5,
    borderRadius: 30,
  },
  title: {
    fontSize: 20,
                              fontFamily:
'TitilliumWeb_400Regular',
    paddingLeft: 10,
    textAlignVertical: 'center',
  },
})

export default Calls;

import React, { createContext,
useContext, useState } from
'react';

const ChatContext =
createContext();

export const useChats = () =>
useContext(ChatContext);

export const ChatProvider = ({
children }) => {
  const [chats, setChats] =
useState([]);

  const addChat = (chat) => {
    setChats((prevChats) =>
[...prevChats, chat]);
  };

  return (
        <ChatContext.Provider
value={{ chats, addChat }}>
    {children}
  </ChatContext.Provider>
  );
};

import React, { useEffect,
useState, useCallback } from
'react'
import { BackHandler,
FlatList, Image, Pressable,
RefreshControl, StyleSheet,
Text, View } from 'react-native'
import { useFonts,
TitilliumWeb_400Regular,
TitilliumWeb_600SemiBold }
from '@expo-google-
fonts/titillium-web';
import { useNavigation,
useFocusEffect } from
'@react-navigation/native';
```

```
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faPenToSquare, faUserGroup } from '@fortawesome/free-solid-svg-icons';
import { Avatar } from 'react-native-elements';
import { getDatabase, ref, get, child, onValue, onDisconnect, set } from 'firebase/database';
import { getAuth } from 'firebase/auth';
import { getFirestore, doc, getDoc, collection, getDocs, query, where, limit, orderBy } from 'firebase/firestore';
import { app } from '../firebaseConfig';
import { SearchBar } from '@rneui/themed';
// import RSA from 'react-native-rsa-native';
// import * as SecureStore from 'expo-secure-store';
global.Buffer = require('buffer').Buffer;

const Item = ({ user, auth, onPress }) => (
  <Pressable onPress={() => onPress(user)}>
    <View style={styles.item}>
      <View style={{
        flexDirection: 'row',
        paddingVertical: 2.5,
        paddingHorizontal: 5,
      }}>
                        <View style={styles.imageContainer}>
          <Avatar size={48} rounded source={user.profilePicture ? { uri: user.profilePicture } : require('../assets/profilepic.jpg')} />
          {user.isOnline && <View style={styles.onlineIndicator} />}
        </View>
        <View>
          <Text style={{
                        fontFamily: 'TitilliumWeb_400Regular',
            fontSize: 20,
            paddingLeft: 10,
            paddingTop: 5,
                        textAlignVertical: 'center',
          }}>{user.username}</Text>
          <Text style={{
                        fontFamily: 'TitilliumWeb_400Regular',
            fontSize: 15,
            paddingLeft: 10,
            paddingBottom: 5,
                        textAlignVertical: 'center',
            color: '#777',
          }}>
            {user.recentMessage ? (user.isSentByCurrentUser ? 'You: ' : '') + user.recentMessage : 'No messages yet'}
          </Text>
        </View>
      </View>
    </View>
  </Pressable >
);

const Chats = () => {
  const [profilePicture, setProfilePicture] = useState('');
  const auth = getAuth(app);
  const database = getDatabase(app);
  const firestore = getFirestore(app);
  const [userInput, setUserInput] = useState('');
  const [users, setUsers] = useState([]);
  const [lastClickedUser, setLastClickedUser] = useState('');
  const [filteredUsers, setFilteredUsers] = useState([]);
  const [refreshing, setRefreshing] = useState(false);
  const onRefresh = useCallback(() => {
    setRefreshing(true);
    setTimeout(() => {
      setRefreshing(false);
    }, 2000);
  }, []);

  const fetchUsersWithRecentMessages = async () => {
    try {

      const usersCollection = collection(firestore, 'users');
      const userSnapshot = await getDocs(usersCollection);
      const userList = await Promise.all(
        userSnapshot.docs.map(async (doc) => {
          const userData = doc.data();
          const participantIds = [auth.currentUser.uid, doc.id].sort().join('_');
          const recentMessageQuery = query(
            collection(firestore, 'chats'),
            where('participants', '==', participantIds),
            orderBy('createdAt', 'desc'),
            limit(1)
          );

          const recentMessageSnapshot = await getDocs(recentMessageQuery);
          if (recentMessageSnapshot.docs.length === 0) {
            return null;
          }

          const recentMessageData = recentMessageSnapshot.docs[0].data()
```

```
        let decryptedMessage = '';
                                if (recentMessageData._sender) {
          // decryptedMessage = await decryptMessage(recentMessageData._sender, privateKey);
            decryptedMessage = Buffer.from(recentMessageData._sender, 'base64').toString('utf-8');
          }

          const userStatusRef = ref(database, `/status/${doc.id}`);
                                const userStatusSnapshot = await get(userStatusRef);
                const isOnline = userStatusSnapshot.val()?.state === 'online';

          return {
            id: doc.id,
            ...userData,
                  recentMessage: decryptedMessage,
            isSentByCurrentUser: recentMessageData.sender === auth.currentUser.uid,
            isOnline,
          };
        })
      );
        const filteredUserList = userList.filter(user => user != null);
      setUsers(filteredUserList);
       setFilteredUsers(filteredUserList);
    } catch (error) {
              console.error('Error fetching users with recent messages: ', error);
    }
  };

  const fetchProfilePicture = async () => {
    try {
                const user = auth.currentUser;
      if (user) {
                const userDoc = doc(firestore, 'users', user.uid);
         const userSnap = await getDoc(userDoc);
        if (userSnap.exists()) {
                const userData = userSnap.data();
          setProfilePicture(userData.profilePicture || '')
        } else {
            console.log('No such document!');
        }
      }
    } catch (error) {
              console.error('Error fetching profile picture: ', error);
    }
  };

  useEffect(() => {
              const user = auth.currentUser;
    if (user) {
                                const userStatusDatabaseRef = ref(database, `status/${user.uid}`);

                                const isOfflineForDatabase = {
        state: 'offline',
              last_changed: new Date().toISOString(),
      }

                                const isOnlineForDatabase = {
        state: 'online',
              last_changed: new Date().toISOString(),
      }

          onValue(ref(database, '.info/connected'), (snapshot) => {
          if (snapshot.val() === false) {
            return;
          }
          onDisconnect(userStatusDatabaseRef).set(isOfflineForDatabase).then(() => {
            set(userStatusDatabaseRef, isOnlineForDatabase);
          });
        });
      }
    }, []);

    useFocusEffect(
      useCallback(() => {
        fetchProfilePicture();
        // fetchUsers();
        fetchUsersWithRecentMessages();
        const onBackPress = () => {
          BackHandler.exitApp();
          return true;
        };

        BackHandler.addEventListener('hardwareBackPress', onBackPress);

        return () =>
          BackHandler.removeEventListener('hardwareBackPress', onBackPress);
      }, [])
    );

    useEffect(() => {
      setFilteredUsers(
              users.filter(user => user.username.toLowerCase().includes(userInput.toLowerCase()))
      );
    }, [userInput, users]);

        const navigation = useNavigation();

      const handleUserPress = (user) => {
      if (user.uid) {
        setLastClickedUser(user.id);
        navigation.navigate('Chat Screen', { user, username: user.username, profilePicture:
```

```
user.profilePicture, uid: user.uid });
    } else {
      console.error('User does not have a valid UID');
    }
  };

  let [fontsLoaded, fontError] = useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });


      if (!fontsLoaded && !fontError) {
    return null;
  }

      const sortedUsers = filteredUsers.slice().sort((a, b) => {
    if (a.id === lastClickedUser) return -1;
    if (b.id === lastClickedUser) return 1;
    return 0;
  });

  return (
                              <View style={styles.container}>
                              <View style={styles.content}>
                              <View style={styles.header}>
          <Pressable
            onPress={() => {
              navigation.openDrawer();
            }}
            style={({ pressed }) => [
              {
                opacity: pressed ? 0.5 : 1,
              }, {
                borderRadius: 50,
                borderWidth: 2.5,
                borderColor: 'hsl(0, 0%, 100%)',
              }
            ]}
          >
            <Avatar
              size={48}
              rounded
              source={profilePicture ? { uri: profilePicture } : require('../assets/profilepic.jpg')}
            />
          </Pressable>
                                        <Text style={styles.textheader}>
            Safe-On-Chat
          </Text>
        </View>
        <View styles={{
          flexDirection: 'row',
          justifyContent: 'center',
          alignItems: 'center',
          margin: 50,
        }}>
          <SearchBar
            round
            searchIcon={{ size: 24 }}
            placeholder=" Search"
            onChangeText={(text) => setUserInput(text)}
            value={userInput}
            containerStyle={{
                  backgroundColor: 'transparent',
                borderBottomWidth: 0,
              borderTopWidth: 0,
            }}
            inputStyle={{
              color: '#fff',
                              fontFamily: 'TitilliumWeb_400Regular',
            }}
            underlineColorAndroid={'transparent'}
            cursorColor={'#fff'}
          />
        </View>
        <View>
          <Pressable
            onPress={() =>
              navigation.navigate('GroupChats')
            }
            style={({ pressed }) => [
              {
                backgroundColor: pressed
                  ? '#4c669f'
                  : 'white',
                        borderColor: pressed
                  ? 'white'
                  : '#4c669f',
                borderWidth: 1,
              },
              {
                paddingHorizontal: 10,
                paddingVertical: 5,
                marginVertical: 10,
                  marginHorizontal: 10,
                borderRadius: 10,
              }
            ]}
          >
            {({ pressed }) => (
              <View style={{
                flexDirection: 'row',
                paddingVertical: 2.5,
                  paddingHorizontal: 5,
              }}>
                <View
                  style={{
                        justifyContent: 'center',
                              alignItems: 'center',
                    paddingVertical: 2.5,
                    paddingHorizontal: 5,
                  }}
                >
                  <FontAwesomeIcon          icon={faUserGroup} size={30}  color={pressed ? 'white' : '#4c669f'} />
                </View>
                <View>
                  <Text style={{
                              fontFamily: 'TitilliumWeb_400Regular',
                    fontSize: 20,
                    paddingLeft: 10,
                      paddingVertical: 10,
                    textAlignVertical: 'center',
```

```
                color: pressed
                  ? '#fff'
                  : '#4c669f',
                        }}>Group
Chats</Text>
              </View>
            </View>
          )}
        </Pressable>
      </View>
      {filteredUsers.length ===
0 ? (
        <View style={{
          flex: 1,
          marginTop: 125,
        }}>
                              <Text
style={styles.temp_text}>No
Conversations Found </Text>
                              <Text
style={styles.temp_text}>Star
t a New Chat. </Text>
        </View>
      ) : (
        <FlatList
          refreshControl={
            <RefreshControl
              refreshing={refreshi
ng}
              onRefresh={onRefr
esh}
              tintColor={'#f0ceff'}
              titleColor={'#f0ceff'}
              title={'Loading...'}
              colors={['#4c669f']}
              progressBackgroun
dColor={'#f0ceff'}
              progressViewOffset
={20}
            />
          }
          showsVerticalScrollIn
dicator={false}
          data={sortedUsers}
          renderItem={({ item })
=>       <Item       user={item}
onPress={handleUserPress}
/>}
          keyExtractor={item =>
item.id}
          style={{ marginTop: 10,
paddingBottom: 10 }}
        />
      )}
      <View
        style={{
          position: 'absolute',
          bottom: 10,
          right: 0,
          margin: 20,
                backgroundColor:
'#4c669f',
          borderRadius: 50,
          padding: 20,
        }}
      >
        <Pressable
          style={({ pressed }) =>
[
            {
              opacity: pressed ?
0.5 : 1,
            }
          ]}
          onPress={() => {
            navigation.navigate("
SearchChat")
          }
          }>
            <FontAwesomeIcon
icon={faPenToSquare}
color="#f0ceff"         size={25}
style={{ alignContent: 'center'
}} />
        </Pressable>
      </View>
    </View>
  )
}

const          styles          =
StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  content: {
    flex: 1,
    paddingTop: 10,
    paddingBottom: 0,
    paddingLeft: 10,
    padding: 10,
  },
  textheader: {
                              fontFamily:
'TitilliumWeb_400Regular',
    fontSize: 25,
    color: 'hsl(0, 0%, 100%)',
    textAlignVertical: 'center',
    marginLeft: 10,
  },
  profilepic: {
    width: 50,
    height: 50,
    borderRadius: 40,
  },
  header: {
    flexDirection: 'row',
    paddingTop: 15,
    paddingBottom: 10,
  },
  temp_text: {
                              fontFamily:
'TitilliumWeb_600SemiBold',
    fontSize: 25,
    color: '#fff',
    textAlign: 'center',
  },
  //!  FOR THE FLATLIST
  item: {
    flexDirection: 'row',
    backgroundColor: '#eef',
    paddingLeft: 10,
    padding: 5,
    marginVertical: 10,
    marginHorizontal: 5,
    borderRadius: 30,
  },
  title: {
    fontSize: 20,
                              fontFamily:
'TitilliumWeb_400Regular',
    paddingLeft: 10,
    textAlignVertical: 'center',
  },
  onlineIndicator: {
    position: 'absolute',
    right: 1,
    bottom: 3,
    width: 12.5,
    height: 12.5,
    borderRadius: 7.5,
                backgroundColor:
'#00dd00',
  },
})

export default Chats;
import React, { useState,
useEffect,    useLayoutEffect,
useCallback } from 'react';
import { BackHandler, Image,
Text,     View,     StyleSheet,
Pressable,              Linking,
```

```
TouchableOpacity } from 'react-native';
import { app } from '../firebaseConfig';
import { getDatabase, ref, onValue } from 'firebase/database';
import { getAuth } from 'firebase/auth';
import { getFirestore, collection, addDoc, orderBy, getDoc, doc, updateDoc, setDoc, serverTimestamp, query, onSnapshot, where } from 'firebase/firestore';
import { getStorage, uploadBytes, getDownloadURL, ref as sRef } from 'firebase/storage';
import { useRoute, useNavigation, useIsFocused } from '@react-navigation/native';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faPaperPlane, faPaperclip, faImage, faVideo, faPhone } from '@fortawesome/free-solid-svg-icons';
import { Composer, GiftedChat, Bubble, MessageText, InputToolbar, Send, Day } from 'react-native-gifted-chat';
import { LinearGradient } from 'expo-linear-gradient';
import { getPushTokenForUser, sendPushNotification } from './Notifications';
import * as ScreenCapture from 'expo-screen-capture';
import * as DocumentPicker from 'expo-document-picker';
import * as ImagePicker from 'expo-image-picker';
import * as SecureStore from 'expo-secure-store';
import { RSA } from 'react-native-rsa-native';
import CryptoJS from 'react-native-crypto-js';
import { LogBox } from 'react-native';
global.Buffer = require('buffer').Buffer;

LogBox.ignoreLogs(['Warning ...']);
LogBox.ignoreAllLogs();

const ChatScreen = () => {
  const navigation = useNavigation();
  const isFocused = useIsFocused();
  const [isUserOnline, setIsUserOnline] = useState(false);
  const [messages, setMessages] = useState([]);
  const [isTyping, setIsTyping] = useState(false);
  const [publicKey, setPublicKey] = useState('');
  const [privateKey, setPrivateKey] = useState('');
  const [aesKey, setAesKey] = useState('');
  const [selectedFileName, setSelectedFileName] = useState(null);
  const auth = getAuth(app);
  const firestore = getFirestore(app);
  const route = useRoute();
  const { user, profilePicture, username } = route.params;
  const participantIds = [auth.currentUser.uid, user.uid].sort().join('_');

  const VideoC = () => {
    navigation.navigate('Video Call', { user, profilePicture });
  };

  const AudioC = () => {
    navigation.navigate('Audio Call', { user, profilePicture });
  }

  useEffect(() => {
    const database = getDatabase(app);
    const userStatusRef = ref(database, 'status/' + user.uid);

    const unsubscribe = onValue(userStatusRef, (snapshot) => {
      const status = snapshot.val();
      setIsUserOnline(status?.state === 'online');
    });

    return () => {
      unsubscribe();
    };
  }, [user.uid]);

  useEffect(() => {
    const fetchKeys = async () => {
      try {
        const publicKeyDoc = await getDoc(doc(firestore, 'users', user.uid));
        if (publicKeyDoc.exists()) {
          setPublicKey(publicKeyDoc.data().publicKey);

        } else {
          console.error('Public key not found');
        }

        const privateKey = await SecureStore.getItemAsync('privateKey');
        if (privateKey) {
          setPrivateKey(privateKey);
        } else {
          console.error('Private key not found');
        }
      } catch (error) {
        console.error('Error fetching keys:', error);
      }
    };

    fetchKeys();
  }, []);

  useEffect(() => {
```

```
                const activateScreenCapture = async () => {
                    await ScreenCapture.preventScreenCaptureAsync();
                };
                const deactivateScreenCapture = async () => {
                    await ScreenCapture.allowScreenCaptureAsync();
                };

                if (isFocused) {
                    activateScreenCapture();
                } else {
                    deactivateScreenCapture();
                }
            }, [isFocused]);

            useEffect(() => {
                const backAction = () => {
                    navigation.navigate("Chats");
                    return true;
                };

                const backHandler = BackHandler.addEventListener(
                    "hardwareBackPress",
                    backAction
                );

                return () => backHandler.remove();
            }, [navigation]);

            useLayoutEffect(() => {
                navigation.setOptions({
                    headerTitle: () => <HeaderWithPicture username={username} profilePicture={profilePicture} isUserOnline={isUserOnline} />,
                });

                if (auth.currentUser && user && user.uid) {
                    const q = query(
                        collection(firestore, 'chats'),
                        where('participants', '==', participantIds),
                        orderBy('createdAt', 'desc')
                    );

                    const unsubscribe = onSnapshot(q, async (snapshot) => {
                        const messagesFirestore = await Promise.all(snapshot.docs.map(async (doc) => {
                            const data = doc.data();

                            let decryptedText = '';
                            if (data.text && data.aesKey) {
                                if (data.user._id !== auth.currentUser.uid) {
                                    const decryptedAesKeyBase64 = await RSA.decrypt(data.aesKey, privateKey);
                                    console.log('Decrypted AES key:', decryptedAesKeyBase64);
                                    const decryptedAesKey = Buffer.from(decryptedAesKeyBase64, 'base64').toString('hex');
                                    console.log('Decrypted AES key buffer:', decryptedAesKey);
                                    const decryptedTextBytes = CryptoJS.AES.decrypt(data.text, decryptedAesKey);
                                    console.log('Decrypted text bytes:', decryptedTextBytes);
                                    decryptedText = decryptedTextBytes.toString(CryptoJS.enc.Utf8);
                                    console.log('Decrypted text:', decryptedText);
                                    console.log('Decryption successful');
                                } else {
                                    decryptedText = Buffer.from(data._sender, 'base64').toString('utf8');
                                }

                                return {
                                    _id: doc.id,
                                    text: decryptedText,
                                    createdAt: data.createdAt.toDate(),
                                    user: {
                                        _id: data.user._id,
                                        name: data.user._id === auth.currentUser.uid
                                            ? username
                                            : user.username,
                                        avatar: data.user._id === auth.currentUser.uid
                                            ? (profilePicture || './assets/profilepic.jpg')
                                            : user.profilePicture,
                                    },
                                    file: data.file || '',
                                    fileType: data.fileType || '',
                                };
                            }));
                            setMessages(messagesFirestore);
                        });

                        return () => {
                            unsubscribe();
                        };
                    } else {
                        console.error('Current user or chat participant is missing a UID');
                    }
                }, [firestore, auth.currentUser, user, participantIds, privateKey]);

                useLayoutEffect(() => {
                    if (auth.currentUser && user && user.uid) {
                        const typingDocRef = doc(firestore, 'typingStatus', participantIds);

                        const unsubscribe = onSnapshot(typingDocRef, (doc) => {
                            const data = doc.data();
                            if (data) {
```

```
        setIsTyping(!!data.typing && data.typing !== auth.currentUser.uid);
      }
    });

    return () => {
      unsubscribe();
    };
  }
        }, [firestore, auth.currentUser, user, participantIds]);

      const onSend = useCallback(async (messages = [], fileURL = null, fileType = null, fileName = null) => {
        const message = messages[0];

        if (!message || !message._id || !message.createdAt || (!message.text && !fileURL) || !message.user) {
          console.error('Invalid message format:', message);
          return;
        }

        const { _id, createdAt, text, user: sender } = message;

        if (!auth.currentUser.uid || !user.uid) {
          console.error('Either the current user or the chat participant does not have a valid UID');
          return;
        }

        try {
          const recipientToken = await getPushTokenForUser(user.uid);

          if (recipientToken) {
            await sendPushNotification(recipientToken, {
              title: "New message sent to you.",
              body: message.text || `Attachment sent to you.`,
              data: {
                screen: 'ChatScreen',
                userId: auth.currentUser.uid,
                userName: username,
                profilePicture: profilePicture || './assets/profilepic.jpg',
                recipieintId: user.uid,
                recipientUserName: user.username,
              },
            })
          } else {
            console.error('Recipient push token not found');
          }

          const aesKey = CryptoJS.lib.WordArray.random(16).toString(); console.log('AES key:', aesKey);

          let encryptedText = '';
          if (text) {
            encryptedText = CryptoJS.AES.encrypt(text, aesKey).toString(); //! Encrypt text using AES key
            console.log('Encrypted text:', encryptedText);
          }

          const aeseKeyBuffer = Buffer.from(aesKey, 'hex'); //! Convert AES key to buffer
          console.log('AES key buffer:', aeseKeyBuffer);
          const encryptedAesKey = await RSA.encrypt(aeseKeyBuffer.toString('base64'), publicKey); console.log('Encrypted AES key:', encryptedAesKey);

          const encryptedFileName = fileName ? CryptoJS.AES.encrypt(fileName, aesKey).toString() : '';
          console.log('Encrypted file name:', encryptedFileName);

          const messageData = {
            _id,
            createdAt: new Date(),
            text: fileURL ? '' : encryptedText, //! Encrypted text
            aesKey: encryptedAesKey, //! Encrypted AES key
            user: {
              _id: sender._id,
              name: sender._id === auth.currentUser.uid ? username : user.username,
              avatar: sender._id === auth.currentUser.uid ? (profilePicture || './assets/profilepic.jpg') : user.profilePicture,
            },
            participants: participantIds,
            file: fileURL || '',
            fileType: fileType || '',
            // fileName: message.fileName,
            fileName: encryptedFileName,
            _sender: Buffer.from(text, 'utf8').toString('base64'),
          };

          await addDoc(collection(firestore, 'chats'), messageData);
          // console.log('Message sent successfully!');
          await setDoc(doc(firestore, 'typingStatus', participantIds), {
            typing: '',
            lastTyped: serverTimestamp(),
          }, { merge: true });
        } catch (error) {
          console.error('Error sending message: ', error);
        }
```

```
    }, [auth.currentUser.uid, user.uid, firestore, participantIds, publicKey]);

                                const handleInputTextChanged = async (text) => {
      const typingDocRef = doc(firestore, 'typingStatus', participantIds);

   if (text) {
                                await setDoc(typingDocRef, {
                                typing: auth.currentUser.uid,
                                lastTyped: serverTimestamp(),
    }, { merge: true });
   } else {
                                await updateDoc(typingDocRef, {
      typing: '',
                                lastTyped: serverTimestamp(),
    });
   }
 };

  const pickImage = async () => {
   try {
       const result = await ImagePicker.launchImageLibraryAsync({
                    mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      quality: 1,
    });

         // console.log('Image picker result:', result);

       if (!result.canceled && result.assets && result.assets.length > 0) {
            const imageUri = result.assets[0].uri;
            const fileName = result.assets[0].fileName || 'image.jpg';
         // console.log('Image picked:', imageUri);

       try {
          const fileURL = await uploadFile(imageUri, 'images');
        const encryptedFileURL = CryptoJS.AES.encrypt(fileURL, aesKey).toString(); //!

         const message = {
                                _id: new Date().getTime().toString(),
          createdAt: new Date(),
          user: {
                                _id: auth.currentUser.uid,
           name: username,
           avatar: profilePicture || './assets/profilepic.jpg',
          },
          text: '',
         file: encryptedFileURL, //! Encrypted file URL
          fileName: fileName,
          fileType: 'image',
         };
             onSend([message], encryptedFileURL, 'image');
       } catch (uploadError) {
             console.error('Error uploading file:', uploadError);
       }
     } else {
         // console.log('Image picking canceled or assets are missing');
     }
   } catch (error) {
             console.error('Error picking image:', error);
   }
  };

  const pickDocument = async () => {
   try {
        // console.log('Picking document...');
        const result = await DocumentPicker.getDocumentAsync(
       {
        type: '*/*',
        copyToCacheDirectory: true,
        multiple: false,
       }
      );

       if (!result.canceled && result.assets && result.assets.length > 0) {
            const fileUri = result.assets[0].uri;
            const fileName = result.assets[0].name;
       setSelectedFileName(fileName);
        console.log('Document picked:', fileUri);
        console.log('Document name:', fileName);

       try {

         const fileURL = await uploadFile(fileUri, 'documents');
       const encryptedFileURL = CryptoJS.AES.encrypt(fileURL, aesKey).toString(); //! Encrypt file URL using AES key
        // console.log('File URL obtained:', fileURL);
        const message = {
                                _id: new Date().getTime().toString(),
          createdAt: new Date(),
          user: {
                                _id: auth.currentUser.uid,
           name: username,
           avatar: profilePicture || './assets/profilepic.jpg',
          },
          text: '',
         file: encryptedFileURL, //! Encrypted file URL
          fileName: fileName,
          fileType: 'document',
         };
             onSend([message], encryptedFileURL, 'document', fileName);
```

```
                                    //
console.log('Document
message sent');
      } catch (uploadError) {
            console.error('Error
uploading
document:',
uploadError);
      }
    } else {
      // console.log('Document
picking canceled or assets are
missing');
    }
  } catch (error) {
            console.error('Error
picking document:', error);
  }
 };

  const uploadFile = async
(uri, fileType) => {
  try {
   // console.log('Fetching file
from URI:', uri);
      const response = await
fetch(uri);

          // console.log('Fetch
response
status:',
response.status);

    if (!response.ok) {
       throw new Error(`Fetch
failed
with
status:
${response.status}`);
    }

          const blob = await
response.blob();
           const storage =
getStorage(app);
            const fileRef =
sRef(storage,
`${fileType}/${new
Date().getTime()}_${auth.curr
entUser.uid}`);

    await uploadBytes(fileRef,
blob);
      const downloadURL =
await
getDownloadURL(fileRef);

           // console.log('File
uploaded
successfully,
download
URL:',
downloadURL);
    return downloadURL;
  } catch (error) {
        console.error('Error
in
uploadFile:', error);
    throw error;
  }
 };

 const HeaderWithPicture = ({
username,
profilePicture,
isUserOnline }) => {
  return (
   <View style={{
    flexDirection: 'row',
    alignItems: 'center',
    width: '100%',
    position: 'relative',
   }}>
                                <View
style={styles.imageContainer
}>
         <Image source={{ uri:
profilePicture }} style={{
          width: 45,
          height: 45,
          borderRadius: 25,
          marginRight: 10,
         }} />
        {isUserOnline && <View
style={styles.onlineIndicator}
/>}
      </View>
      <Text style={{
       color: '#fff',
       fontSize: 20,
                        fontFamily:
'TitilliumWeb_600SemiBold',
      }}>{username}</Text>
      <Pressable
       style={{
        position: 'absolute',
        right: 140,
       }}
       onPress={AudioC}
      >
       <FontAwesomeIcon
        icon={faPhone}
        size={21}
        color='#fff'
       />
      </Pressable>
      <Pressable
       style={{
        position: 'absolute',
        right: 90,
       }}
       onPress={VideoC}
      >
       <FontAwesomeIcon
        icon={faVideo}
        size={25}
        color='#fff'
       />
      </Pressable>
     </View>
    );
 };

  const CustomMessageText
= (props) => {
   return (
    <MessageText
     {...props}
     textStyle={{
              left: [styles.text,
styles.textLeft],
             right: [styles.text,
styles.textRight],
     }}
    />
   );
 };

    const CustomBubble =
(props) => {
     const [imageDimensions,
setImageDimensions] =
useState({ width: 0, height: 0
});
   const MAX_WIDTH = 300;
   const MAX_HEIGHT = 300;

       const decryptedUri =
CryptoJS.AES.decrypt(props.
currentMessage.file,
aesKey).toString(CryptoJS.e
nc.Utf8); //! Decrypt file URL
using AES key

   let decryptedFileName = '';
                                if
(props.currentMessage.fileNa
me) {
     try {
          decryptedFileName =
CryptoJS.AES.decrypt(props.
currentMessage.fileName,
aesKey).toString(CryptoJS.e
```

```
nc.Utf8); //! Decrypt file name using AES key
    } catch (error) {
      console.error('Error decrypting file name:', error);
    }
  }

  useEffect(() => {
    if (props.currentMessage.fileType === 'image' && decryptedUri) {
      Image.getSize(decryptedUri, (width, height) => {
        let newWidth = width;
        let newHeight = height;

        if (width > MAX_WIDTH || height > MAX_HEIGHT) {
          const aspectRatio = width / height;

          if (width > height) {
            newWidth = MAX_WIDTH;
            newHeight = MAX_WIDTH / aspectRatio;
          } else {
            newHeight = MAX_HEIGHT;
            newWidth = MAX_HEIGHT * aspectRatio;
          }

        }
        setImageDimensions({ width: newWidth, height: newHeight });
      });
    }
  }, [decryptedUri]);

  return (
    <View>
      <Bubble
        {...props}
        wrapperStyle={{
          right: {
            backgroundColor: '#fff',
            paddingHorizontal: 5,
          },
          left: {
            backgroundColor: '#fff',
            paddingHorizontal: 5,
          },
        }}
        renderTime={() => (
          <Text style={[
            props.position === 'left' ? styles.timeLeft : styles.timeRight,
            styles.timeText,
          ]}>
            {props.currentMessage.createdAt.toLocaleTimeString([], {
              hour: '2-digit',
              minute: '2-digit',
            })}
          </Text>
        )}
      />
      {props.currentMessage.fileType === 'image' && (
        <Image
          source={{ uri: decryptedUri }}
          style={{
            width: imageDimensions.width,
            height: imageDimensions.height,
            borderRadius: 20,
            marginTop: 5,
          }}
        />
      )}
      {props.currentMessage.fileType === 'document' && (
        <Text
          style={{
            color: '#4c669f',
            marginVertical: 10,
            backgroundColor: '#fff',
            textDecorationLine: 'underline',
            border: 1,
            borderColor: '#000',
            borderRadius: 22,
            paddingHorizontal: 15,
            paddingBottom: 5,
            paddingTop: 5,
            fontSize: 16,
            fontFamily: 'TitilliumWeb_400Regular',
          }}
          onPress={() => Linking.openURL(decryptedUri)}
        >
          {selectedFileName || 'Document'}
        </Text>
      )}
    </View>
  );
};

const CustomInputToolbar = (props) => {
  return (
    <InputToolbar
      {...props}
      containerStyle={{
        // backgroundColor: '#4c669f',
        maxHeight: 60,
        overflow: 'hidden',
      }}
      renderComposer={(composerprops) => (
        <View style={{ flex: 1, flexDirection: 'row', alignItems: 'center', paddingLeft: 15 }}>
          <Pressable
            onPress={pickImage}
            style={{
              marginLeft: 1,
              marginRight: 10,
              borderWidth: 1,
              borderColor: '#000',
              borderRadius: 5,
              padding: 5,
            }}
          >
            <FontAwesomeIcon icon={faImage} size={20} color='#000' />
          </Pressable>
          <Pressable
            onPress={pickDocument}
            style={{
              marginLeft: 3,
              marginRight: 5,
              borderWidth: 1,
              borderColor: '#000',
```

```
            borderRadius: 5,
            padding: 5,
          }}
        >
          <FontAwesomeIcon
icon={faPaperclip}  size={20}
color='#000' />
        </Pressable>
        <Composer
          {...composerprops}
          textInputStyle={{
            color: '#000',
                    fontFamily:
'TitilliumWeb_400Regular',
            flex: 1,
            multiline: true,
          }}
          placeholderTextColo
r='#000'
        />
      </View>
    )}
  />
 );
};

const renderSend = (props)
=> {
  return (
    <Send {...props}>
      <View style={{
        marginRight: 10,
        marginBottom: 5,
        borderWidth: 1,
        borderColor: '#000',
        borderRadius: 5,
        padding: 5,
      }}>
          <FontAwesomeIcon
icon={faPaperPlane}
size={20} color='#000' />
      </View>
    </Send>
  );
};

return (
  <LinearGradient
            colors={['#4c669f',
'#f0ceff']}
    style={{ flex: 1 }}
    start={[0.5, 0.5]}
  >
    <GiftedChat
      messages={messages}
      onSend={messages =>
onSend(messages)}
      user={{
                    _id:
auth.currentUser.uid,
        name: username,
        avatar: profilePicture ||
'./assets/profilepic.jpg',
      }}
      renderBubble={props =>
<CustomBubble {...props} />}
      isTyping={isTyping}
      onInputTextChanged={h
andleInputTextChanged}
      renderMessageText={C
ustomMessageText}
      renderInputToolbar={Cu
stomInputToolbar}
      renderAvatarOnTop={fal
se}
      renderSend={renderSen
d}
      showAvatarForEveryMe
ssage={false}
        renderDay={props =>
<Day {...props} textStyle={{
        color: '#fff',
                    fontFamily:
'TitilliumWeb_400Regular',
        fontSize: 16,
      }}
      />}
                    //
renderUsernameOnMessage
={true}
      renderChatEmpty={() =>
(
        <View style={{
          flex: 1,
          justifyContent: 'center',
          alignItems: 'center',
            transform: [{ rotate:
'180deg' }],
        }}>
          <Image
                source={{  uri:
user.profilePicture }}
            style={{
              width: 100,
              height: 100,
              borderRadius: 50,
              marginBottom: 20,
              borderWidth: 3,
              borderColor: '#fff',
            }}
          />
          <Text style={{
                    fontFamily:
'TitilliumWeb_600SemiBold',
            fontSize: 25,
            color: '#fff',
          }}>{user.username}</
Text>
          <Text style={{
                    fontFamily:
'TitilliumWeb_400Regular',
            fontSize: 16,
            color: '#fff',
            marginTop: 5,
                    }}>Start  a
conversation!</Text>
        </View>
      )}
    />
  </LinearGradient>
 );
};

const          styles          =
StyleSheet.create({
  text: {
                    fontFamily:
'TitilliumWeb_400Regular'
  },
  textLeft: {
    color: '#000',
  },
  textRight: {
    color: '#000',
  },
  timeText: {
    fontSize: 10,
                    fontFamily:
'TitilliumWeb_400Regular',
    marginHorizontal: 6,
  },
  timeLeft: {
    color: '#555',
  },
  timeRight: {
    color: '#555',
  },
  imageContainer: {
    position: 'relative',
  },
  onlineIndicator: {
    position: 'absolute',
    right: 8,
    bottom: 3,
    width: 12.5,
```

```
    height: 12.5,
    borderRadius: 7.5,
    backgroundColor: '#00dd00',
  },
});

export default ChatScreen;

import React, { useEffect, useState } from 'react';
import { ActivityIndicator, BackHandler, View, Text, TextInput, Pressable, FlatList, StyleSheet, ToastAndroid } from 'react-native';
import { LinearGradient } from 'expo-linear-gradient';
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import { SearchBar } from '@rneui/themed';
import { Avatar, Button } from 'react-native-elements';
import { getAuth } from 'firebase/auth';
import { getFirestore, doc, getDoc, collection, getDocs, addDoc } from 'firebase/firestore';
import { getStorage, ref, uploadString, getDownloadURL, uploadBytes } from 'firebase/storage';
import { app } from '../firebaseConfig';
import { RSA } from 'react-native-rsa-native';
import CryptoJS from 'react-native-crypto-js';
import * as ImagePicker from 'expo-image-picker';
import { getPushTokenForUser } from './Notifications';

const Item = ({ user, onPress, isSelected }) => (
  <Pressable
    onPress={() => onPress(user)}
  >
    <View
      onPress={() => onPress(user)}
      style={
        {
          backgroundColor: isSelected ? '#fff' : 'transparent',
          borderColor: '#fff',
          borderWidth: 2,
          paddingHorizontal: 10,
          paddingVertical: 5,
          marginVertical: 10,
          marginHorizontal: 10,
          borderRadius: 10,
        }
      }
    >
      <View style={{
        flexDirection: 'row',
        paddingVertical: 2.5,
        paddingHorizontal: 5,
      }}>
        <View>
          <Avatar size={48} rounded source={user.profilePicture ? { uri: user.profilePicture } : require('../assets/profilepic.jpg')} />
        </View>
        <View>
          <Text style={{
            fontFamily: isSelected ? 'TitilliumWeb_400Regular' : 'TitilliumWeb_600SemiBold',
            fontSize: 20,
            paddingLeft: 10,
            paddingVertical: 10,
            textAlignVertical: 'center',
            color: isSelected ? '#000' : '#fff',
          }}>{user.username}</Text>
        </View>
      </View>
    </View>
  </Pressable>
);

const CreateGroupScreen = ({ navigation }) => {
  const [groupName, setGroupName] = useState('');
  const [selectedUsers, setSelectedUsers] = useState([]);
  const [userInput, setUserInput] = useState('');
  const [showCreateButton, setShowCreateButton] = useState(false);
  const [isLoading, setIsLoading] = useState(false);
  const [filteredUsers, setFilteredUsers] = useState([]);
  const [profilePicture, setProfilePicture] = useState('');
  const [groupPhotoUri, setGroupPhotoUri] = useState('');
  const auth = getAuth(app);
  const [users, setUsers] = useState([]); // Fetch the list of users from Firestore

  const firestore = getFirestore(app);

  const createGroup = async () => {
    if (!groupName || selectedUsers.length === 0) {
      ToastAndroid.show('Group name and participants are required!', ToastAndroid.SHORT);
      console.log('Photo selected:', result.uri);
      return; // Handle validation
    }

    let photoURL = '';
    if (groupPhotoUri) {

      setIsLoading(true);
      const filename = groupPhotoUri.substring(groupPhotoUri.lastIndexOf('/') + 1);
      const response = await fetch(groupPhotoUri);
```

```
        const blob = await response.blob();

        const storage = getStorage(app);
        const storageRef = ref(storage, `group-photos/${filename}`);

    try {
                    await uploadBytes(storageRef, blob);
          photoURL = await getDownloadURL(storageRef);
           console.log('Photo uploaded:', photoURL); // Add this line to check if the photo is uploaded
      ToastAndroid.show('Photo uploaded successfully!', ToastAndroid.SHORT);
    } catch (e) {
           console.error('Error uploading photo: ', e);

      ToastAndroid.show('Error uploading photo!', ToastAndroid.SHORT);
    } finally {
      setIsLoading(false);
    }
  }

  try {
          const aesKey = CryptoJS.lib.WordArray.random(16).toString();
    const encryptedKeys = {};
    const pushTokens = {};

        for (const userId of selectedUsers) {
       const userDoc = await getDoc(doc(firestore, "users", userId));
          const publicKey = userDoc.data().publicKey;
       const encryptedAesKey = await RSA.encrypt(aesKey, publicKey);
      encryptedKeys[userId] = encryptedAesKey;
      const pushToken = await getPushTokenForUser(userId);
      if (pushToken) {
        pushTokens[userId] = pushToken;
      }
    }

       const groupDocRef = await addDoc(collection(firestore, 'groups'), {
      name: groupName,
                participants: selectedUsers,
      createdAt: new Date(),
      photoURL: photoURL,
              encryptedKeys: encryptedKeys,
      pushToken: pushTokens,
    });

    navigation.navigate('GroupChats', { groupId: groupDocRef.id, groupName, photoURL }); // Navigate to the chat screen or group chat list
  } catch (error) {
          console.error('Error creating group:', error);
    ToastAndroid.show('Error creating group!', ToastAndroid.SHORT);
  }
 };

 const fetchUsers = async () => {
  try {
    const usersCollection = collection(firestore, 'users');
       const userSnapshot = await getDocs(usersCollection);
          const userList = userSnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
    setUsers(userList);
    setFilteredUsers(userList);
  } catch (error) {
          console.error('Error fetching users: ', error);
  }
 };

 const fetchProfilePicture = async () => {
  try {
          const user = auth.currentUser;
    if (user) {
          const userDoc = doc(firestore, 'users', user.uid);
      const userSnap = await getDoc(userDoc);
      if (userSnap.exists()) {
          const userData = userSnap.data();
        setProfilePicture(userData.profilePicture || '')
      } else {
        console.log('No such document!');
      }
    }
  } catch (error) {
          console.error('Error fetching profile picture: ', error);
  }
 };

 useEffect(() => {
  setShowCreateButton(groupName.length > 0);
 }, [groupName]);

 useEffect(() => {
  fetchProfilePicture();
  fetchUsers();
  const backAction = () => {
   navigation.navigate("SearchChat");
   return true;
  };

      const backHandler = BackHandler.addEventListener(
   "hardwareBackPress",
   backAction
  );

          return () => backHandler.remove();
 }, []);
```

```
  useEffect(() => {
    setFilteredUsers(
            users.filter(user =>
user.username.toLowerCase(
).includes(userInput.toLower
Case()))
    );
  }, [userInput, users]);

    const handleUserPress =
(user) => {
                              if
(selectedUsers.includes(user.
id)) {
      setSelectedUsers(selecte
dUsers.filter(id => id !==
user.id));
    } else {
      setSelectedUsers([...sele
ctedUsers, user.id]);
    }
  };

  const uploadPhoto = async ()
=> {
     const { status } = await
ImagePicker.requestMediaLi
braryPermissionsAsync();
    if (status !== 'granted') {
      console.error('Permission
to access camera roll is
required!');
      return;
    }

       const result = await
ImagePicker.launchImageLib
raryAsync({
                    mediaTypes:
ImagePicker.MediaTypeOptio
ns.Images,
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    })

    console.log(result);

       if (!result.canceled &&
result.assets &&
result.assets.length > 0) {
                const uri =
result.assets[0].uri;
                console.log('Photo
selected:',
uri);      setGroupPhotoUri(uri)
;
    }
  }

  let [fontsLoaded, fontError] =
useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });

      if (!fontsLoaded &&
!fontError) {
    return null;
  }

  return (
                              <View
style={styles.container}>
      <LinearGradient
              colors={['#4c669f',
'#f0ceff']}
        style={styles.linearGradi
ent}
        start={[0.5, 0.5]}
      >
                              <View
style={styles.content}>
          <View>
            <View
              style={{
                flexDirection: 'row',
                paddingHorizontal:
5,
                alignItems: 'center',
              }}>
              <TextInput
                placeholder="Enter
group name (required)"
                placeholderTextCol
or={'#fff'}
                underlineColorAndr
oid={'#fff'}
              value={groupName}
                onChangeText={se
tGroupName}
                style={{
                  flex: 1,
                              fontFamily:
'TitilliumWeb_400Regular',
                  fontSize: 16,
                  padding: 10,
                  borderRadius: 5,
                  marginBottom: 10,
                  marginRight: 10,
                marginVertical: 10,
                }}
                cursorColor={'#fff'}
                color={'#fff'}
                autoCapitalize={'wo
rds'}
                autoFocus={true}
              />
              <Pressable
                onPress={uploadP
hoto}
              >
                {({ pressed }) => (
                  <Text style={{
                              fontFamily:
'TitilliumWeb_600SemiBold',
                    fontSize: 14,
                  backgroundColor:
pressed    ?    "#005f99"    :
"#007acc",
                    color: '#fff',
                    textAlign: 'center',
                    padding: 13,
                    borderRadius: 5,
                    }}>Upload Group
Photo</Text>
                )}
              </Pressable>
            </View>
          {showCreateButton &&
(
            <Pressable
              onPress={createGr
oup}
              style={({ pressed })
=> ({
                backgroundColor:
pressed ? '#f0ceff' : '#fff',
                borderRadius: 5,
                marginHorizontal:
10,
              })}>
              {isLoading ? (
                <ActivityIndicator
size="large"
color="#000"              style={{
padding: 10 }} />
              ) : (
                <Text
                  style={{
                              fontFamily:
'TitilliumWeb_600SemiBold',
                    fontSize: 14,
```

```
                color: '#000',
                padding: 13,
                borderRadius: 5,
                textAlign: 'center',
                }}>Create
Group</Text>
              )}
            </Pressable>
          )}
        </View>
        <SearchBar
          round={true}
          platform='default'
          searchIcon={{ size: 24
}}
          placeholder=" Search"
          onChangeText={(text)
=> setUserInput(text)}
          value={userInput}
          containerStyle={{
              backgroundColor:
'transparent',
            borderBottomWidth:
0,
            borderTopWidth: 0,
          }}
          inputStyle={{
            color: '#fff',
                    fontFamily:
'TitilliumWeb_400Regular',
          }}
          underlineColorAndroi
d={'transparent'}
          cursorColor={'#fff'}
        />
          {filteredUsers.length
=== 0 ? (
          <View style={{
            flex: 1,
            marginTop: 125,
          }}>
                          <Text
style={styles.temp_text}>No
Results Available. </Text>
          </View>
        ) : (
          <FlatList
            showsVerticalScrollI
ndicator={false}
            data={filteredUsers}
            renderItem={({ item })
=> <Item
              user={item}
              onPress={handleU
serPress}
              isSelected={selecte
dUsers.includes(item.id)}
            />}
            keyExtractor={item =>
item.id}
            style={{ marginTop:
10, paddingBottom: 10 }}
          />
        )}
        {/* <FlatList
          data={users}
            keyExtractor={(item)
=> item.id}
          renderItem={({ item })
=> (
            <TouchableOpacity
onPress={() =>
setSelectedUsers([...selected
Users, item.id])}>
            <Text>{item.userna
me}</Text>
          </TouchableOpacity>
        )}
        /> */}
      </View>
    </LinearGradient>
  </View>
 );
};

const styles =
StyleSheet.create({
 container: {
  flex: 1,
  justifyContent: 'center',
 },
 content: {
  flex: 1,
  paddingTop: 10,
  paddingBottom: 0,
  padding: 10,
 },
 linearGradient: {
  flex: 1,
 },
 temp_text: {
                  fontFamily:
'TitilliumWeb_600SemiBold',
  fontSize: 25,
  color: '#fff',
  textAlign: 'center',
 },
 loadingIndicator: {
  position: 'absolute',
  top: "50%",
  left: "50%",
  transform: [{ translateX: -25
}, { translateY: -25 }],
 },
});

export default
CreateGroupScreen;


import React, { useEffect,
useState } from "react";
import {
 Alert,
 BackHandler,
 StyleSheet,
 Text,
 TextInput,
 TouchableOpacity,
 View,
} from "react-native";
import { useFonts,
TitilliumWeb_400Regular,
TitilliumWeb_600SemiBold }
from '@expo-google-
fonts/titillium-web';
import { useNavigation } from
"@react-navigation/native";
import { app } from
"../firebaseConfig";
import { getAuth,
sendPasswordResetEmail }
from "firebase/auth";
import { LinearGradient } from
'expo-linear-gradient';

function ForgotPassword() {
 const auth = getAuth(app);
   const navigation =
useNavigation();
  const [email, setEmail] =
useState("");
   const [isEmailsent,
setIsEmailsent] =
useState("Send Email");

                  const
handdleForgotPassword =
async () => {
  try {
                  await
sendPasswordResetEmail(au
th, email);
    setIsEmailsent(true);
    Alert.alert(
```

```
      "PASSWORD RESET",
       "A password reset email has been sent to your email address."
     );
   } catch (error) {
        const errorsCode = error.code;
       const errorsMessage = error.message;

     switch (errorsCode) {
          case "auth/missing-email":
              Alert.alert("Forgot Password", "Input email for password reset.");
         break;
           case "auth/user-not-found":
              Alert.alert("Forgot Password", "User not found.");
         break;
       case "auth/invalid-email":
              Alert.alert("Forgot Password", "The email address is not valid.");
         break;
       default:
         Alert.alert(
           "Forgot Password",
              `Account creation error: ${errorsMessage} (Error Code: ${errorsCode})`
         );
         break;
     }
   }
 };

 useEffect(() => {
   const backAction = () => {
     navigation.navigate("Login");
     return true;
   };

       const backHandler = BackHandler.addEventListener(
     "hardwareBackPress",
     backAction
   );
              return () => backHandler.remove();
 }, []);

 let [fontsLoaded, fontError] = useFonts({
   TitilliumWeb_400Regular,
   TitilliumWeb_600SemiBold,
 });

     if (!fontsLoaded && !fontError) {
   return null;
 }

 return (
                                <View style={styles.container}>

     <LinearGradient
              colors={['#4c669f', '#f0ceff']}
       style={
         styles.linearg
       }
       start={[0.5, 0.5]}
     >
                                <View style={styles.content}>
         <Text
           style={{
                              fontFamily: "TitilliumWeb_400Regular",
             fontSize: 15,
             marginTop: 50,
             marginBottom: 10,
             padding: 10,
             color: "#fff",
             lineHeight: 25,
           }}
         >
              Enter your email address below and we'll send you a link to reset your
           password.
         </Text>
         <TextInput
           style={{
             height: 40,
             width: 300,
                              fontFamily: "TitilliumWeb_400Regular",
             borderColor: "gray",
             borderWidth: 1,
             borderRadius: 10,
             padding: 10,
                 backgroundColor: "#fff",
             marginTop: 10,
             marginBottom: 10,
           }}
           placeholder="Email"
           placeholderTextColor={"#999"}
           onChangeText={(text) => setEmail(text)}
         />
         <TouchableOpacity
           onPress={handdleForgotPassword}
           style={{
                 backgroundColor: "#000",
             borderWidth: 1,
             borderColor: "#ffffff",
             paddingLeft: 10,
             paddingRight: 10,
             borderRadius: 10,
             marginTop: 10,
             marginBottom: 10,
             opacity: isEmailsent ? 1 : 0.5,
           }}
           disabled={!isEmailsent}
         >
           <Text
             style={{
                              fontFamily: "TitilliumWeb_600SemiBold",
               fontSize: 20,
               marginTop: 10,
               marginBottom: 10,
               padding: 5,
               color: "#fff",
             }}
           >
             {isEmailsent ? "Send" : "Sent"}
           </Text>
         </TouchableOpacity>
       </View>
     </LinearGradient>
   </View>
 );
}

const styles = StyleSheet.create({
  container: {
```

```
    flex: 1,
  },
  linearg: {
    position: 'absolute',
    left: 0,
    right: 0,
    top: 0,
    height: '100%',
  },
  content: {
    justifyContent: "center",
    alignItems: "center",
    marginTop: 10,
  },
});
export default ForgotPassword;

import React, { useEffect, useState, useCallback } from 'react';
import { BackHandler, View, Text, FlatList, Pressable, StyleSheet, LogBox, RefreshControl, ScrollView } from 'react-native';
import { getFirestore, collection, getDocs } from 'firebase/firestore';
import { LinearGradient } from 'expo-linear-gradient';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faPlus } from '@fortawesome/free-solid-svg-icons';
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import { Avatar } from 'react-native-elements';
import { getAuth } from 'firebase/auth';
import { useNavigation } from '@react-navigation/native';
import { app } from '../firebaseConfig';

LogBox.ignoreLogs(['Warning ...']);
LogBox.ignoreAllLogs();

const GroupChatsScreen = () => {
  const [groupChats, setGroupChats] = useState([]);
  const firestore = getFirestore(app);
  const auth = getAuth(app);
  const navigation = useNavigation();
  const [refreshing, setRefreshing] = useState(false);

  const onRefresh = useCallback(() => {
    setRefreshing(true);
    fetchGroupChats().finally(() => setRefreshing(false));
  }, []);

  useEffect(() => {
    navigation.setOptions({
      headerBackVisible: false,
    })
  })

  useEffect(() => {
    const backAction = () => {
      navigation.navigate("Chats");
      return true;
    };

    const backHandler = BackHandler.addEventListener(
      "hardwareBackPress",
      backAction
    );

    return () => backHandler.remove();
  }, []);

  const fetchGroupChats = async () => {
    try {
      const groupChatsCollection = collection(firestore, 'groups');
      const groupChatsSnapshot = await getDocs(groupChatsCollection);
      const groupChatsList = groupChatsSnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
      const currentUserId = auth.currentUser.uid;
      const userGroupChats = groupChatsList.filter(group => group.participants.includes(currentUserId));
      setGroupChats(userGroupChats);
    } catch (error) {
      console.error('Error fetching group chats: ', error);
    }
  };

  useEffect(() => {
    fetchGroupChats();
  }, []);


  const handleGroupChatPress = (group) => {
    navigation.navigate('GroupChatScreen', { groupId: group.id, groupName: group.name, photoURL: group.photoURL });
  };

  const getRandomColor = () => {
    const letters = '0123456789ABCDEF';
    let color = '#';
    for (let i = 0; i < 6; i++) {
      color += letters[Math.floor(Math.random() * 16)];
    }
    return color;
  };

  let [fontsLoaded, fontError] = useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });


  if (!fontsLoaded && !fontError) {
```

```
  return null;
 }

 return (
                        <View
style={styles.container}>
    <LinearGradient
            colors={['#4c669f',
'#f0ceff']}
      style={{ flex: 1 }}
      start={[0.5, 0.5]}
    >
      <View
       style={styles.content}
      >
       {
}
         <FlatList
          refreshControl={
           <RefreshControl
            refreshing={refres
hing}
            onRefresh={onRe
fresh}
           tintColor={'#f0ceff'}
           titleColor={'#f0ceff'}
            title={'Loading...'}
           colors={['#4c669f']}
            progressBackgrou
ndColor={'#f0ceff'}
            progressViewOffs
et={20}
          />
          }
          ListEmptyComponen
t={
           <View style={{
            flex: 1,
            marginTop: 40,
           }}>
                        <Text
style={styles.temp_text}>No
group chats joined.</Text>
            <Text style={{
                  fontFamily:
'TitilliumWeb_600SemiBold',
             fontSize: 25,
             color: '#fff',
             textAlign: 'center',
                  }}>Create
one.</Text>
           </View>
          }
          data={groupChats}
          renderItem={({ item })
=> (
           <Pressable
            onPress={() =>
handleGroupChatPress(item)
           }>
                        <View
style={styles.groupChatItem}
>
            {item.photoURL ?
(
             <Avatar rounded
source={{ uri: item.photoURL
}} size={40} containerStyle={{
                  marginRight:
20,
             }} />
            ) : (
             <Avatar rounded
title={item.name[0]} size={40}
containerStyle={{
                  backgroundCo
lor: getRandomColor(),
                  marginRight:
20,
             }} />
            )}
                  {/* <Avatar
rounded  title={item.name[0]}
size={40} containerStyle={{
                  backgroundCol
or: getRandomColor(),
                  marginRight: 20,
                 }} /> */}
                        <Text
style={styles.groupChatName
}>{item.name}</Text>
           </View>
          </Pressable>
         )}
          keyExtractor={item =>
item.id}
         />
        {/* )} */}
        <Pressable
         onPress={
                        () =>
navigation.navigate('CreateG
roupChat')
         }
        >
           <FontAwesomeIcon
icon={faPlus}          size={50}
color="#fff" style={{
            alignSelf: 'center',
            marginTop: 25,
            padding: 30,
            borderRadius: 50,
           }} />
         </Pressable>
        </View>
      </LinearGradient>
    </View>
  );
};

const          styles          =
StyleSheet.create({
 container: {
  flex: 1,
 },
 content: {
  padding: 10,
 },
 header: {
  fontSize: 24,
  fontWeight: 'bold',
  textAlign: 'center',
  marginVertical: 10,
 },
 groupChatItem: {
  alignItems: 'center',
  flexDirection: 'row',
  padding: 15,
  paddingLeft: 20,
  backgroundColor: '#f0ceff',
  borderColor: '#000',
  borderWidth: 1,
  borderRadius: 30,
  marginBottom: 10,
 },
 groupChatName: {
  fontSize: 18,
                fontFamily:
'TitilliumWeb_600SemiBold',
  color: '#000',
 },
 temp_text: {
                fontFamily:
'TitilliumWeb_600SemiBold',
  fontSize: 25,
  color: '#fff',
  textAlign: 'center',
  marginTop: 125,
 },
});

export                default
GroupChatsScreen;
```

```
import React, { useEffect, useRef, useState } from 'react';
import { Image, BackHandler, View, Text, TextInput, Pressable, FlatList, StyleSheet, Linking } from 'react-native';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faPaperPlane, faPaperclip, faImage, faVideo, faPhone } from '@fortawesome/free-solid-svg-icons';
import { getAuth } from 'firebase/auth';
import { getFirestore, doc, collection, query, where, orderBy, onSnapshot, addDoc, Timestamp, getDoc } from 'firebase/firestore';
import { getStorage, uploadBytes, getDownloadURL, ref as sRef } from 'firebase/storage';
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import * as ScreenCapture from "expo-screen-capture";
import { app } from '../firebaseConfig';
import { Avatar, Divider } from 'react-native-elements';
import { LinearGradient } from 'expo-linear-gradient';
import { useIsFocused } from '@react-navigation/native';
import { RSA } from 'react-native-rsa-native';
import { getPushTokenForUser, sendPushNotification } from './Notifications';
import * as SecureStore from 'expo-secure-store';
import * as DocumentPicker from 'expo-document-picker';
import * as ImagePicker from 'expo-image-picker';
import CryptoJS from 'react-native-crypto-js';

const GroupChatScreen = ({ route, navigation }) => {
    const isFocused = useIsFocused();
  const { groupId, groupName, photoURL } = route.params; // Pass groupId and groupName through route params
        const [messages, setMessages] = useState([]);
      const [messageText, setMessageText] = useState('');
  const auth = getAuth(app);
      const firestore = getFirestore(app);
      const flatlistRef = useRef(null);
  const [aesKey, setAesKey] = useState('');

  useEffect(() => {
                                const activateScreenCapture = async () => {
                                await ScreenCapture.preventScreenCaptureAsync();
    };
                                const deactivateScreenCapture = async () => {
                                await ScreenCapture.allowScreenCaptureAsync();
    };

    if (isFocused) {
      activateScreenCapture();
    } else {
      deactivateScreenCapture();
    }
  }, [isFocused]);

  const getRandomColor = () => {
          const letters = '0123456789ABCDEF';
    let color = '#';
    for (let i = 0; i < 6; i++) {
                                color += letters[Math.floor(Math.random() * 16)];
    }
    return color;
  };

  useEffect(() => {
    navigation.setOptions({
      headerLeft: () (

        <Avatar
          rounded
          source={photoURL ? { uri: photoURL } : null}
          title={groupName[0]}
          size={40}
          containerStyle={{
                    backgroundColor: photoURL ? 'transparent' : getRandomColor(),
            marginRight: 15,
          }}
        />
      ),
      headerTitle: groupName,
      headerBackVisible: false,
    })
  }, [navigation, groupName, photoURL]);

  useEffect(() => {
    const backAction = () => {
      navigation.navigate("GroupChats");
      return true;
    };

        const backHandler = BackHandler.addEventListener(
      "hardwareBackPress",
      backAction
    );

                return () => backHandler.remove();
  }, []);

  useEffect(() => {
                                const fetchMessagesWithUsernames = async () => {
```

```
        const messagesRef = collection(firestore, 'groups', groupId, 'messages');
                const q = query(messagesRef, orderBy('createdAt', 'asc'));

      const groupDoc = await getDoc(doc(firestore, 'groups', groupId));

     const encryptedAesKey = groupDoc.data().encryptedKeys[auth.currentUser.uid];
          console.log('Encrypted AES key:', encryptedAesKey);
      const privateKey = await SecureStore.getItemAsync('privateKey');
     console.log('Private key:', privateKey);
        const aesKey = await RSA.decrypt(encryptedAesKey, privateKey);
        console.log('AES key:', aesKey);
     setAesKey(aesKey);

         const unsubscribe = onSnapshot(q, async (snapshot) => {
         const messagesList = await Promise.all(snapshot.docs.map(async (msgDoc) => {
          const messageData = msgDoc.data();
              const userRef = doc(firestore, 'users', messageData.senderId);
         const userSnap = await getDoc(userRef);
              const username = userSnap.data().username;

                        const decryptedTextBytes = CryptoJS.AES.decrypt(messageData.text, aesKey);
          console.log('Decrypted text bytes:', decryptedTextBytes);
          const decryptedText = decryptedTextBytes.toString(CryptoJS.enc.Utf8);
           console.log('Decrypted text:', decryptedText);
           console.log('Decryption successful');

           return {
             id: msgDoc.id,
             ...messageData,
             text: decryptedText,
             username: username,
           }
         }));
         setMessages(messagesList);
         flatlistRef.current?.scrollToEnd({ animated: true });
       });

       return () => unsubscribe();
     }
     fetchMessagesWithUsernames();
   }, [groupId]);

   const sendMessage = async (messageData = {}) => {
     if (!messageText.trim() && !messageData.file) return;

     try {
           const userRef = doc(firestore, 'users', auth.currentUser.uid);
         const userSnap = await getDoc(userRef);

       if (!userSnap.exists()) {
          console.error('User not found');
         return;
       }

           const username = userSnap.data().username;

         const groupDoc = await getDoc(doc(firestore, 'groups', groupId));

           const groupData = groupDoc.data();
       const encryptedAesKey = groupDoc.data().encryptedKeys[auth.currentUser.uid];
          console.log('Encrypted AES key:', encryptedAesKey);
       const privateKey = await SecureStore.getItemAsync('privateKey');
       console.log('Private key:', privateKey);
          const aesKey = await RSA.decrypt(encryptedAesKey, privateKey);
          console.log('AES key:', aesKey);

          const encryptedText = messageText ? CryptoJS.AES.encrypt(messageText, aesKey).toString() : "";
           console.log('Encrypted text:', encryptedText);

       const encryptedFileName = messageData.fileName ? CryptoJS.AES.encrypt(messageData.fileName, aesKey).toString() : '';
      console.log('Encrypted file name:', encryptedFileName);

          const messagesRef = collection(firestore, 'groups', groupId, 'messages');
             console.log('Sending message:', encryptedText);
         console.log('Sender ID:', auth.currentUser.uid);
          console.log('Username:', username);
         console.log('Created at:', Timestamp.now());
             console.log('Message data:', messageData);

                                 await addDoc(messagesRef, {
         text: encryptedText,
                                senderId: auth.currentUser.uid,
         username: username,
                                createdAt: Timestamp.now(),
         file: messageData.file || '',
                                // fileName: messageData.fileName || '',
```

```
                fileName: encryptedFileName,
                fileType: messageData.fileType || '',
            });

            const pushTokens = groupData.pushToken;
            if (pushTokens) {
                for (const memberId in pushTokens) {
                    const recipientToken = pushTokens[memberId];
                    if (recipientToken) {
                        console.log('Sending push notification to:', memberId + " with token " + recipientToken);
                        const response = await sendPushNotification(recipientToken, {
                            title: `New messages in ${groupName}`,
                            body: messageText || `Sent an attachment in ${groupName}`,
                            data: {
                                screen: 'GroupChatScreen',
                                sender: auth.currentUser.uid,
                                groupId: groupId,
                                groupName: groupName,
                            },
                        });
                        console.log('Push notification sent:', response);
                    } else {
                        console.log('No push token found for:', memberId);
                    }
                }
            } else {
                console.log('No push tokens found');
            }

            setMessageText('');
            console.log('Message sent');
        } catch (error) {
            console.error('Error sending message:', error);
        }
    };

    const pickImage = async () => {
        try {
            const result = await ImagePicker.launchImageLibraryAsync({
                mediaTypes: ImagePicker.MediaTypeOptions.Images,
                allowsEditing: true,
                quality: 1,
            })

            if (!result.canceled && result.assets && result.assets.length > 0) {
                const imageUri = result.assets[0].uri;
                const fileName = result.assets[0].fileName || 'image.jpg';

                try {
                    const fileURL = await uploadFile(imageUri, "images");
                    const encryptedFileURL = CryptoJS.AES.encrypt(fileURL, aesKey).toString();
                    const message = {
                        text: '',
                        file: encryptedFileURL,
                        fileName: fileName,
                        fileType: 'image',
                    };
                    sendMessage(message);
                } catch (uploadError) {
                    console.log(aesKey);
                    console.error('Error uploading image:', uploadError);
                }
            }
        } catch (error) {
            console.error('Error picking image:', error);
        }
    }

    const pickDocument = async () => {
        console.log('Picking document');
        try {
            const result = await DocumentPicker.getDocumentAsync({
                type: '*/*',
                copyToCacheDirectory: true,
                multiple: false,
            });

            console.log('Document result:', result);

            if (!result.canceled && result.assets && result.assets.length > 0) {
                const fileUri = result.assets[0].uri;
                const fileName = result.assets[0].name;
                console.log('Document URI:', fileUri);
                console.log('Document name:', fileName);

                try {
                    const fileURL = await uploadFile(fileUri, 'documents');
                    const encryptedFileURL = CryptoJS.AES.encrypt(fileURL, aesKey).toString();
                    console.log('Encrypted document URL:', encryptedFileURL);
                    const message = {
                        text: '',
                        file: encryptedFileURL,
                        fileName: fileName,
                        fileType: 'document',
                    };
                    sendMessage(message);
                } catch (uploadError) {
                    console.error('Error uploading document:', uploadError);
                }
            }
        } catch (error) {
            console.error('Error picking document:', error);
```

```
  }
 };

  const uploadFile = async (uri, fileType) => {
   try {
     const response = await fetch(uri);
    if (!response.ok) {
      throw new Error(`Fetch failed with status: ${response.status}`);
    }

        const blob = await response.blob();
          const storage = getStorage(app);
          const fileRef = sRef(storage, `${fileType}/${new Date().getTime()}_${auth.currentUser.uid}`);

    await uploadBytes(fileRef, blob);
      const downloadURL = await getDownloadURL(fileRef);
    return downloadURL;
   } catch (error) {
          console.error('Error uploading file:', error);
   }
 }

 let [fontsLoaded, fontError] = useFonts({
   TitilliumWeb_400Regular,
  TitilliumWeb_600SemiBold,
 });

    if (!fontsLoaded && !fontError) {
   return null;
 }

 return (
                    <View style={styles.container}>
    <LinearGradient
          colors={['#4c669f', '#f0ceff']}
      style={{ flex: 1 }}
      start={[0.5, 0.5]}
      >
       <FlatList
        ref={flatlistRef}
        data={messages}
         renderItem={({ item }) => {

        let decryptedFileName = '';
        if (item.fileName) {
         try {
          decryptedFileName = CryptoJS.AES.decrypt(item.fileName, aesKey).toString(CryptoJS.enc.Utf8);
           console.log("Decrypted file name: ", decryptedFileName);
          } catch (error) {
           console.error("Decryption failed for fileName: ", error);
          }
         }

        return (
         <View style={[
          styles.messageContainer,
           item.senderId === auth.currentUser.uid ? styles.currentUserMessage : styles.otherUserMessage
          ]}>
           <Text style={{
            fontSize: 16,
                fontFamily: 'TitilliumWeb_400Regular',
                alignSelf: item.senderId === auth.currentUser.uid ? 'flex-end' : 'flex-start',
           }}>{item.text}</Text>
           {item.file && (
             item.fileType === 'image' ? (
              <ImageComponent file={item.file} aesKey={aesKey} />
             ) : (
              <Text
               style={{
               color: "#4c669f",
                marginVertical: 10,
                backgroundColor: "#fff",
                textDecorationLine: "underline",
                border: 1,
                 borderColor: "#000",
                borderRadius: 22,
                paddingHorizontal: 15,
               paddingBottom: 5,
                paddingTop: 5,
                fontSize: 16,
                 fontFamily: 'TitilliumWeb_400Regular',
               }}
                onPress={() => Linking.openURL(CryptoJS.AES.decrypt(item.file, aesKey).toString(CryptoJS.enc.Utf8))}
              >
               {/* {item.fileName || "View Document"} */}
                {decryptedFileName || "View Document"}
              </Text>
            ))}
           <View
            style={{
            flexDirection: 'row',
              justifyContent: 'space-between',
                alignItems: 'center',
             marginTop: 5,
            }}
           >
                <Text style={styles.messageText}>{item.username}</Text>
             <Divider
              orientation="vertical"
             width={1}
                 style={{ backgroundColor: 'grey', marginHorizontal: 3 }}
             />
                <Text style={styles.messageTime}>
```

```
{new Date(item.createdAt.seconds * 1000).toLocaleTimeString([], { hour: '2-digit', minute: '2-digit' })}</Text>
            </View>
          </View>
        )
      }}
      keyExtractor={item => item.id}
      style={styles.messagesList}
      showsVerticalScrollIndicator={false}
      onContentSizeChange={() => flatlistRef.current?.scrollToEnd({ animated: true })}
    />
                    <View style={styles.inputContainer}>
      <Pressable
        onPress={pickImage}
        style={styles.sendButton}
      >
        <FontAwesomeIcon icon={faImage} size={20} style={{ color: '#000' }} />
      </Pressable>
      <Pressable
        onPress={pickDocument}
        style={styles.sendButton}
      >
        <FontAwesomeIcon icon={faPaperclip} size={20} style={{ color: '#000' }} />
      </Pressable>
      <TextInput
        value={messageText}
        onChangeText={setMessageText}
        style={styles.input}
        placeholder="Type a message"
      />
                <Pressable onPress={sendMessage} style={styles.sendButton}>
        <FontAwesomeIcon icon={faPaperPlane} size={20} style={{ color: '#000' }} />
      </Pressable>
    </View>
  </LinearGradient>
 </View>
 );
};

const ImageComponent = ({ file, aesKey }) => {
    const [dimensions, setDimensions] = useState({ width: 0, height: 0 });
 const MAX_WIDTH = 300;
 const MAX_HEIGHT = 300;

   const decryptedUri = CryptoJS.AES.decrypt(file, aesKey).toString(CryptoJS.enc.Utf8);

 useEffect(() => {
   Image.getSize(decryptedUri, (width, height) => {
    let newWidth = width;
    let newHeight = height;

    if (width > MAX_WIDTH || height > MAX_HEIGHT) {
     const aspectRatio = width / height;

      if (width > height) {
        newWidth = MAX_WIDTH;
        newHeight = MAX_WIDTH / aspectRatio;
      } else {
        newHeight = MAX_HEIGHT;
        newWidth = MAX_HEIGHT * aspectRatio;
      }

    }
      setDimensions({ width: newWidth, height: newHeight });
   });
 }, [decryptedUri]);

 return (
   <Image
    source={{ uri: decryptedUri }}
    style={{
      width: dimensions.width,
      height: dimensions.height,
      // marginVertical: 10,
      borderRadius: 10,
    }}
   />
 )
}

const styles = StyleSheet.create({
 container: {
  flex: 1,
 },
 groupName: {
  fontSize: 20,
  fontFamily: 'TitilliumWeb_600SemiBold',
  textAlign: 'center',
  marginVertical: 10,
 },
 messagesList: {
  flex: 1,
 },
 messageContainer: {
  flexDirection: 'column',
  marginVertical: 5,
  marginHorizontal: 10,
  padding: 10,
  borderRadius: 20,
 },
 messageText: {
  color: '#666',
  fontSize: 12.5,
  fontFamily: 'TitilliumWeb_400Regular',
 },
 messageTime: {
  fontSize: 12,
  fontFamily: 'TitilliumWeb_400Regular',
  textAlign: 'right',
  color: 'grey',
 },
 inputContainer: {
  flexDirection: 'row',
  alignItems: 'center',
  borderTopWidth: 1,
  backgroundColor: '#fff',
  borderColor: '#ddd',
```

```
    paddingVertical: 5,
    paddingHorizontal: 5,
  },
  input: {
    flex: 1,
    padding: 5,
    borderWidth: 0.5,
    borderColor: '#ccc',
    borderRadius: 5,
                    fontFamily:
'TitilliumWeb_400Regular',
  },
  sendButton: {
    padding: 5,
    backgroundColor: '#fff',
    borderWidth: 1,
    borderRadius: 5,
    marginLeft: 5,
    marginRight: 5,
    marginVertical: 5,
  },
  currentUserMessage: {
    alignSelf: 'flex-end',
            backgroundColor:
'#dcf8c6',
  },
  otherUserMessage: {
    alignSelf: 'flex-start',
    backgroundColor: '#f0f0f0',
  },
});

export                default
GroupChatScreen;


import * as React from 'react'
import { Image, Pressable,
StyleSheet, Text, View } from
'react-native'
import       { useFonts,
TitilliumWeb_400Regular,
TitilliumWeb_600SemiBold }
from          '@expo-google-
fonts/titillium-web';
import { useNavigation } from
'@react-navigation/native';

const Item = ({ title, image })
=> (
  <View style={styles.item}>
      <Image source={image}
style={styles.image} />
                    <Text
style={styles.title}>{title}</Tex
t>
  </View>
)

const Header = () => {
  let [fontsLoaded, fontError] =
useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });

      const      navigation      =
useNavigation();

      if     (!fontsLoaded     &&
!fontError) {
    return null;
  }

  return (
                    <View
style={styles.container}>
                    <View
style={styles.content}>
                    <View
style={styles.header}>
          <Pressable
            onPress={() => {
              console.log('Profile
Picture Pressed');
              navigation.openDra
wer();
            }}
            style={({{ pressed })
=> [
              {
                opacity: pressed ?
0.5 : 1,
              }
            ]}
          >
            <Image
              style={styles.profile
pic}
              source={require('../
assets/profilepic.jpeg')}
            />
          </Pressable>
                    <Text
style={styles.textheader}>
            Safe-on-chat
          </Text>
        </View>
      </View>
    </View>
  )
}

const          styles          =
StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  content: {
    flex: 1,
    paddingTop: 40,
    paddingBottom: 0,
    padding: 10,
  },
  textheader: {
                    fontFamily:
'TitilliumWeb_400Regular',
    fontSize: 25,
    color: 'hsl(0, 0%, 100%)',
    // marginTop: 6,
    textAlignVertical: 'center',
    marginLeft: 10,
  },
  profilepic: {
    width: 50,
    height: 50,
    borderRadius: 40,
  },
  header: {
    flexDirection: 'row',
    paddingTop: 5,
    paddingBottom: 10,
  },
})

export default Header;

import React, { createContext,
useContext, useState } from
"react";

const      ImageContext      =
createContext();

export const ImageProvider =
({ children }) => {
  const [image, setImage] =
useState(null);

  return (
      <ImageContext.Provider
value={{ image, setImage }}>
```

```
    {children}
   </ImageContext.Provider>
  );
};

export const useImage = () =>
{
      const context =
useContext(ImageContext);
  if (!context) {
   throw new Error("useImage
must be used within an
ImageProvider");
  }
  return context;
};

import * as React from 'react';
import { Image, Pressable,
Text, View, StyleSheet } from
'react-native';
import { LinearGradient } from
'expo-linear-gradient';
import { useFonts,
TitilliumWeb_400Regular,
TitilliumWeb_600SemiBold }
from '@expo-google-
fonts/titillium-web';
import { useNavigation } from
'@react-navigation/native';

const Landingpage = () => {
      const navigation =
useNavigation();

  const pressSignUp = () => {
    navigation.navigate('Regist
er');
  }

  const pressLogin = () => {
    navigation.navigate('Login'
);
  }
  let [fontsLoaded, fontError] =
useFonts({
    TitilliumWeb_400Regular,
   TitilliumWeb_600SemiBold,
  });

      if (!fontsLoaded &&
!fontError) {
   return null;
  }
  return (
                        <View
style={styles.container}>
     <LinearGradient
              colors={['#4c669f',
'#f0ceff']}
       style={
         styles.linearg
       }
       start={[0.5, 0.5]}
      >
       <Image
         style={styles.logo}
          source={require('../ass
ets/soc.png')}
       />
       <Text style={{
                        fontFamily:
'TitilliumWeb_600SemiBold',
         fontSize: 40,
         alignSelf: 'center',
         color: '#fff',
         marginBottom: 30,
       }}>Safe on Chat</Text>
       <Pressable
         onPress={pressLogin}
         style={({ pressed }) => [
          {
              backgroundColor:
pressed
            ? 'rgb(210, 230, 255)'
             : 'white',
          },
          styles.loginbutton
         ]}
       >
                        <Text
style={styles.logintext}>LOG
IN</Text>
       </Pressable>
                        <View
style={styles.groupedtext}>
                        <Text
style={styles.randomtext}>No
Account?</Text>
            <Text  style={[{color:
'#0000ff'},  styles.randomtext]}
onPress={pressSignUp}>
Sign Up Here</Text>
       </View>
     </LinearGradient>
    </View>
  )
}

const          styles          =
StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  linearg: {
    position: 'absolute',
    left: 0,
    right: 0,
    top: 0,
    height: '100%',
  },
  logo: {
    marginTop: 175,
    marginBottom: 20,
    backgroundColor: '#fff',
    borderRadius: 25,
    width: 150,
    height: 150,
    alignSelf: 'center',
  },
  loginbutton: {
    marginTop: 20,
    width: 180,
    height: 50,
    alignSelf: 'center',
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 5,
  },
  logintext: {
                        fontFamily:
'TitilliumWeb_600SemiBold',
    fontSize: 17.5,
  },
  randomtext: {
    marginTop: 20,
                        fontFamily:
'TitilliumWeb_600SemiBold',
    fontSize: 15,
    alignSelf: 'center',
  },
  groupedtext: {
    flexDirection: 'row',
    justifyContent: 'center',
  }
});

export default Landingpage;

// Loading.js
import  React,  {  useEffect  }
from "react";
```

```
import { Image, View, Text, ActivityIndicator } from "react-native";
import { getAuth, onAuthStateChanged } from "firebase/auth";
import { useNavigation } from "@react-navigation/native";
import { LinearGradient } from "expo-linear-gradient";

const Loading = () => {
    const navigation = useNavigation();
  const auth = getAuth();

  useEffect(() => {
      const unsubscribe = onAuthStateChanged(auth, (user) => {
    if (user) {
      navigation.replace("PinandFingerprint");
    } else {
      navigation.replace("Land");
    }
  });

  return () => unsubscribe();
 }, []);

 return (
  <View
   style={{
    flex: 1,
    justifyContent: 'center',
   }}
  >
   <LinearGradient
            colors={['#4c669f', '#f0ceff']}
     style={{
      position: 'absolute',
      left: 0,
      right: 0,
      top: 0,
      height: '100%',
     }}
     start={[0.5, 0.5]}
    >
     <Image
      style={{
       marginTop: 150,
       width: 150,
       height: 150,
       backgroundColor: '#fff',
       borderRadius: 30,
       alignSelf: 'center',
      }}
      source={require('../assets/soc.png')}
     />
     <Text style={{
                fontFamily: 'TitilliumWeb_600SemiBold',
      fontSize: 40,
      alignSelf: 'center',
      color: '#fff',
      marginTop: 10,
      marginBottom: 30,
     }}>Safe on Chat</Text>
            <ActivityIndicator size="large" color={"#FFFFFF"} />
    </LinearGradient>
   </View>
  );
};

export default Loading;

import React, { useState, useEffect } from 'react';
import { BackHandler, Image, KeyboardAvoidingView, Pressable, Text, TextInput, View, StyleSheet, TouchableOpacity } from 'react-native';
import { LinearGradient } from 'expo-linear-gradient';
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import { useNavigation } from '@react-navigation/native';
import AsyncStorage from "@react-native-async-storage/async-storage";
import { app } from '../firebaseConfig';
import {
  getAuth,
  onAuthStateChanged,
  signInWithEmailAndPassword,
} from "firebase/auth";

const LoginScreen = () => {
    const navigation = useNavigation();
  const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
  const [loading, setLoading] = useState(false);
    const [authError, setAuthError] = useState("");
  const auth = getAuth(app);

  useEffect(() => {
   const backAction = () => {
    navigation.navigate("Land");
    return true;
   };

    const backHandler = BackHandler.addEventListener(
    "hardwareBackPress",
    backAction
   );

            return () => backHandler.remove();
  }, []);

  const pressLogin = () => {
   setLoading(true);


    signInWithEmailAndPassword(auth, email, password)
     .then((userCredential) => {
      // Signed in
            const user = userCredential.user;
       AsyncStorage.setItem("lastemail", email);
          setAuthError("Log in successful.");
      setTimeout(() => {
       navigation.navigate("SemiApp");
      }, 1000);
     })
     .catch((error) => {
          const errorCode = error.code;
```

```
        const errorMessage = error.message;
      switch (errorCode) {
       case "auth/invalid-login-credentials":
        setAuthError("Account doesn't exist.");
        break;
          case "auth/invalid-credential":
         setAuthError("Invalid Credentials");
        break;
         case "auth/user-not-found":
        setAuthError("Account doesn't exist.");
        break;
          case "auth/invalid-email":
         setAuthError("Please provide a valid email.");
        break;
           case "auth/weak-password":
        setAuthError("Password is too weak. Please provide a stronger password.");
        break;
          case "auth/wrong-password":
       setAuthError("Incorrect password.");
        break;
         case "auth/missing-password":
         setAuthError("Please provide a password.");
        break;
        case "auth/too-many-requests":
           setAuthError("Too many attempts. Please try again later.");
        break;
       default:
           setAuthError("An unexpected error occurred. Please try again later.");
        break;
      }
    })
    .finally(() => {
     setLoading(false);
    });
  };


  let [fontsLoaded, fontError] = useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });

    if (!fontsLoaded && !fontError) {
    return null;
  }

  return (
                              <View style={styles.container}>

      <LinearGradient
              colors={['#4c669f', '#f0ceff']}
        style={
          styles.linearg
        }
        start={[0.5, 0.5]}
      >
        <KeyboardAvoidingView
          style={styles.container}
          behavior='padding'
        >
          <Image
            style={styles.logo}
            source={require('../assets/soc.png')}
          />
          <Text style={{
            marginTop: 10,
                        fontFamily: 'TitilliumWeb_600SemiBold',
            fontSize: 30,
            alignSelf: 'center',
            color: '#fff',
          }}>Safe on Chat</Text>
          <TextInput
            placeholderTextColor={'rgb(200, 200, 200)'}
            placeholder='Email'
            autoCapitalize='none'
            style={styles.input}
            autoFocus={true}
            inputMode='email'
            value={email}
            onChangeText={(text) => {
              setEmail(text);
            }}
          />
          <TextInput
            placeholderTextColor={'rgb(220, 220, 220)'}
            placeholder='Password'
            autoCapitalize='none'
            style={styles.input}
            secureTextEntry={true}
            contextMenuHidden={true}
            value={password}
            onChangeText={(text) => {
              setPassword(text);
            }}
          />
          <TouchableOpacity
            onPress={() => {
              navigation.navigate('ForgotPassword');
            }}
          >
            <Text
              style={{
                color: '#fff',
                marginTop: 5,
                        fontFamily: 'TitilliumWeb_600SemiBold',
                textAlign: 'center',
              }}
            >
              Forgot Password?
            </Text>
          </TouchableOpacity>
          {authError ? (
            <Text style={{
                              color: authError.includes('Log in successful') ? '#00FF00' : 'red',
              marginTop: 20,
                        fontFamily: 'TitilliumWeb_400Regular',
              textAlign: 'center',
            }}
            >
              {authError}
            </Text>
          ) : null}
          <Pressable
            style={({ pressed }) => [
              {
```

```
            backgroundColor:
pressed
              ? 'rgb(210, 230,
255)'
              : 'rgb(255, 255,
255)',
          },
          styles.loginbutton
        ]}
        onPress={pressLogin}
      >
                          <Text
style={styles.logintext}>LOG
IN</Text>
      </Pressable>
      </KeyboardAvoidingVie
w>
    </LinearGradient>
  </View>
 )
}

const styles =
StyleSheet.create({
 container: {
  flex: 1,
  // justifyContent: 'center',
 },
 linearg: {
  position: 'absolute',
  left: 0,
  right: 0,
  top: 0,
  height: '100%',
 },
 input: {
  marginTop: 50,
  textAlign: 'center',
  width: 225,
  height: 50,
  backgroundColor: '#fff',
  alignSelf: 'center',
  justifyContent: 'center',
  alignItems: 'center',
  border: '1px solid #000000',
  borderRadius: 10,
                    fontFamily:
'TitilliumWeb_600SemiBold',
  fontSize: 15,
 },
 logo: {
  marginTop: 120,
  width: 100,
  height: 100,
  backgroundColor: '#fff',
  borderRadius: 20,
  alignSelf: 'center'
 },
 loginbutton: {
  marginTop: 30,
  width: 175,
  height: 40,
  alignSelf: 'center',
  justifyContent: 'center',
  alignItems: 'center',
  borderRadius: 10,
 },
 logintext: {
                    fontFamily:
'TitilliumWeb_600SemiBold',
  fontSize: 17.5,
 },
});

export default LoginScreen;

import { getFirestore, doc,
setDoc, getDoc } from
'firebase/firestore';
import { app } from
'../firebaseConfig';
import * as Notifications from
'expo-notifications';
import Constants from 'expo-
constants';
import * as Device from 'expo-
device';

const firestore =
getFirestore(app);

// Configure how notifications
are handled
Notifications.setNotificationH
andler({
 handleNotification: async ()
=> ({
  shouldShowAlert: true,
  shouldPlaySound: true,
  shouldSetBadge: true,
 }),
});

function
handleRegistrationError(error
Message) {
 alert(errorMessage);
          throw new
Error(errorMessage);
}

export const
registerForPushNotifications
Async = async (userId) => {
 console.log('Registering for
notifications...');
 if (!Device.isDevice) {
  console.warn('Must use a
physical device for push
notifications.');
  return null;
 }

 const { status: existingStatus
}      =      await
Notifications.getPermissions
Async();
  // let finalStatus =
existingStatus;

  if (existingStatus !==
'granted') {
  const { status } = await
Notifications.requestPermissi
onsAsync();
       console.log('Updated
notification permissions
status:', status);
  if (status !== 'granted') {
   handleRegistrationError('
Failed to get push token for
push notifications!');
    console.warn('Failed to
get push token for push
notifications!');
  return null;
  }
 }
   const projectId =
Constants?.expoConfig?.extr
a?.eas?.projectId ??
Constants?.easConfig?.proje
ctId;
  console.log('Project ID:',
projectId);
 if (!projectId) {
  handleRegistrationError('F
ailed to get project ID for push
notifications!');
 }

 try {
  const token = (
                    await
Notifications.getExpoPushTo
kenAsync(
```

```
      {
        projectId,
      }
    )
  ).data;
        console.log("Token:", token);
  return token;
 } catch (e) {
  handleRegistrationError(`${e}`);
  console.error(e);
  throw e;
  // return null;
 }
}

// Retrieve the token for a user from Firestore
export const getPushTokenForUser = async (userId) => {
 if (!userId) {
  console.warn('User ID is required to fetch push token.');
  return null;
 }

    const userDoc = doc(firestore, 'users', userId);
 const docSnapshot = await getDoc(userDoc);

 if (docSnapshot.exists()) {
      const data = docSnapshot.data();
      return data.expoPushToken || null;
 } else {
      console.warn('No document found for the specified user ID.');
  return null;
 }
};

export const getUserDetailsForNotification = async (userId) => {
 if (!userId) {
  console.warn('User ID is required to fetch user details.');
  return null;
 }

    const userDoc = doc(firestore, 'users', userId);
 const docSnapshot = await getDoc(userDoc);

 if (docSnapshot.exists()) {
      const data = docSnapshot.data();
  return {
       expoPushToken: data.expoPushToken || null,
       username: data.username || 'Unknown User',
       profilePicture: data.profilePicture || './assets/defaultProfile.png',
       userId, // Include userId as part of the result
  };
 } else {
      console.warn('No document found for the specified user ID.');
  return null;
 }
};


// Send a push notification
export const sendPushNotification = async (expoPushToken, message) => {
 if (!expoPushToken) {
  console.warn('Expo push token is required to send notifications.');
  return;
 }

 const messagePayload = {
  to: expoPushToken,
      sound: './assets/notif-sound/notif.wav',
      title: message.title || 'Notification',
  body: message.body || 'You have a new message.',
  data: message.data,
 };

 try {
  const response = await fetch('https://exp.host/--/api/v2/push/send', {
   method: 'POST',
   headers: {
    Accept: 'application/json',
    'Accept-encoding': 'gzip, deflate',
    'Content-Type': 'application/json',
   },
   body: JSON.stringify(messagePayload),
  });

  const result = await response.json();
      console.log('Push notification sent successfully:', result);
  return result;
 } catch (error) {
      console.error('Error sending push notification:', error);
  throw error;
 }
};

export const sendVideoCallNotification = async (expoPushToken, callerInfo) => {
 if (!expoPushToken) {
  console.warn('Expo push token is required to send notifications.');
  return;
 }

 const messagePayload = {
  to: expoPushToken,
      sound: './assets/notif-sound/notif.wav',
  title: 'Incoming Video Call',
      body: `You have an incoming video call from ${callerInfo.username}.`,
  data: {
   type: 'videocall',
   callerInfo,
  },
 };
```

```
  try {
      const response = await fetch('https://exp.host/--/api/v2/push/send', {
      method: 'POST',
      headers: {
       Accept: 'application/json',
        'Accept-encoding': 'gzip, deflate',
                'Content-Type': 'application/json',
      },
                        body: JSON.stringify(messagePayload),
    });

       const result = await response.json();
            console.log('Push notification sent successfully:', result);
    return result;
  } catch (error) {
            console.error('Error sending push notification:', error);
    throw error;
  }
}

export const sendAudioCallNotification = async (expoPushToken, callerInfo) => {
  if (!expoPushToken) {
     console.warn('Expo push token is required to send notifications.');
    return;
  }

  const messagePayload = {
    to: expoPushToken,
        sound: './assets/notif-sound/notif.wav',
    title: 'Incoming Audio Call',
       body: `You have an incoming audio call from ${callerInfo.username}.`,
    data: {
      type: 'audiocall',
      callerInfo,
    },
  };
```

```
  try {
      const response = await fetch('https://exp.host/--/api/v2/push/send', {
      method: 'POST',
      headers: {
       Accept: 'application/json',
        'Accept-encoding': 'gzip, deflate',
                'Content-Type': 'application/json',
      },
                        body: JSON.stringify(messagePayload),
    });

       const result = await response.json();
            console.log('Push notification sent successfully:', result);
    return result;
  } catch (error) {
            console.error('Error sending push notification:', error);
    throw error;
  }
}

import React, { useState, useEffect, useRef } from 'react';
import { View, Text, StyleSheet, Pressable } from 'react-native';
import { LinearGradient } from 'expo-linear-gradient';
import * as LocalAuthentication from 'expo-local-authentication';
import ReactNativePinView from 'react-native-pin-view';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faUnlock, faDeleteLeft, faFingerprint } from '@fortawesome/free-solid-svg-icons';
```

```
import { useFonts, TitilliumWeb_400Regular } from '@expo-google-fonts/titillium-web';

const AuthScreen = ({ navigation }) => {
      const pinView = useRef(null);
  const [isBiometricSupported, setIsBiometricSupported] = useState(false);
      const [storedPin, setStoredPin] = useState('');
      const [enteredPin, setEnteredPin] = useState('');
  const [showRemoveButton, setShowRemoveButton] = useState(false);
                        const [showCompletedButton, setShowCompletedButton] = useState(false);
      const [authError, setAuthError] = useState('');

  useEffect(() => {
    checkBiometricSupport();
    retrieveStoredPin();
  }, []);

                        const checkBiometricSupport = async () => {
     const compatible = await LocalAuthentication.hasHardwareAsync();
    setIsBiometricSupported(compatible);
    if (compatible) {
       const result = await LocalAuthentication.authenticateAsync({
            promptMessage: 'Authenticate',
          fallbackLabel: 'Enter passcode',
      });
      if (result.success) {
        setAuthError('Authentication successful');
       setTimeout(() => {
         navigation.navigate('SemiApp');
       }, 1000)
```

```
    } else {
      setAuthError('Fingerprint Authentication Failed');
    }
  }
 };

  const retrieveStoredPin = async () => {
      const pin = await AsyncStorage.getItem('user_pin');
  setStoredPin(pin);
 };

 const handlePinComplete = async (pin, clear) => {
  if (pin === storedPin) {
    setAuthError('Authentication successful');
    clear();
    setTimeout(() => {
      navigation.navigate('SemiApp');
    }, 1000)
  } else {
      setAuthError('Incorrect PIN');
    clear();
  }
 };

 useEffect(() => {
  if (enteredPin.length > 0) {
    setShowRemoveButton(true);
  } else {
    setShowRemoveButton(false);
  }
  if (enteredPin.length === 4) {
    setShowCompletedButton(true);
  } else {
    setShowCompletedButton(false);
  }
 }, [enteredPin]);

 let [fontsLoaded, fontError] = useFonts({
  TitilliumWeb_400Regular,
 });

    if (!fontsLoaded && !fontError) {
  return null;
 }

 return (
                <View style={styles.container}>
    <LinearGradient
            colors={['#4c669f', '#f0ceff']}
      style={styles.gradient}
      start={[0.5, 0.5]}
    >
                <Text style={styles.title}>Authentication</Text>
        {isBiometricSupported && (
          <FontAwesomeIcon icon={faFingerprint} size={150} style={{ color: "#fff", marginBottom: 20 }} />
        )}
       {isBiometricSupported ? (
        <Pressable
          style={styles.fingerprintButton}
          onPress={checkBiometricSupport}
        >
                <Text style={styles.fingerprintButtonText}>Authenticate</Text>
        </Pressable>
       ) : (
        <ReactNativePinView
         ref={pinView}
         onComplete={(inputtedPin, clear) => handlePinComplete(inputtedPin, clear)}
          pinLength={4}
          inputSize={32}
          buttonSize={60}
          onValueChange={value => setEnteredPin(value)}
          buttonAreaStyle={styles.buttonArea}
          inputAreaStyle={styles.inputArea}
          inputViewEmptyStyle={styles.inputViewEmpty}
          inputViewFilledStyle={styles.inputViewFilled}
          buttonViewStyle={styles.buttonView}
          buttonTextStyle={styles.buttonText}
          onButtonPress={key => {
            if (key === "custom_left") {
              pinView.current.clear();
            } else if (key === "custom_right") {
              handlePinComplete(enteredPin, pinView.current.clear);
            }
          }}
          customLeftButton={showRemoveButton ? <FontAwesomeIcon icon={faDeleteLeft} size={48} color={"#000"} /> : undefined}
          customRightButton={showCompletedButton ? <FontAwesomeIcon icon={faUnlock} size={48} color={"#000"} /> : undefined}
        />
       )}
       {authError ? (
          <Text style={{ color: authError === 'Authentication successful' ? '#00FF00' : 'red', marginTop: 20, fontFamily: 'TitilliumWeb_400Regular' }}>
            {authError}
          </Text>
       ) : null}
     </LinearGradient>
   </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  gradient: {
    position: 'absolute',
    left: 0,
```

```
    right: 0,
    top: 0,
    height: '100%',
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  title: {
    fontSize: 24,
    color: '#fff',
    marginBottom: 50,
                    fontFamily:
'TitilliumWeb_400Regular',
  },
  fingerprintButton: {
    backgroundColor: '#fff',
    padding: 10,
    borderRadius: 5,
    marginTop: 20,
  },
  fingerprintButtonText: {
                    fontFamily:
'TitilliumWeb_400Regular',
  },
  buttonArea: { marginTop: 24
},
   inputArea: { marginBottom:
25 },
        inputViewEmpty:        {
backgroundColor:
'transparent', borderWidth: 3,
borderColor: '#fff' },
         inputViewFilled:       {
backgroundColor: '#fff' },
           buttonView:          {
backgroundColor:       '#000',
borderWidth: 3, borderColor:
'#fff' },
  buttonText: { color: '#fff' },
});

export default AuthScreen;

import React, { useState,
useEffect, useRef } from
'react';
import { View, Text,
StyleSheet, Pressable } from
'react-native';
import { LinearGradient } from
'expo-linear-gradient';
import *
LocalAuthentication from
'expo-local-authentication'; //!
import ReactNativePinView
from 'react-native-pin-view';
import AsyncStorage from
'@react-native-async-
storage/async-storage';
import { FontAwesomeIcon }
from '@fortawesome/react-
native-fontawesome';
import { faUnlock,
faDeleteLeft, faFingerprint }
from '@fortawesome/free-
solid-svg-icons';
import { useFonts,
TitilliumWeb_400Regular }
from '@expo-google-
fonts/titillium-web';
const SignUpScreen = ({
navigation }) => {
  const [isBiometricSupported,
setIsBiometricSupported] =
useState(false);
        const [enteredPin,
setEnteredPin] = useState('');
       const pinView =
useRef(null);
   const [showRemoveButton,
setShowRemoveButton] =
useState(false);
                        const
[showCompletedButton,
setShowCompletedButton] =
useState(false);
         const [authError,
setAuthError] = useState('');
       // const db =
getFirestore(app);

  useEffect(() => {
    checkBiometricSupport();
  }, []);

                        const
checkBiometricSupport =
async () => {
     const compatible = await
LocalAuthentication.hasHard
wareAsync();
     setIsBiometricSupported(c
ompatible);
  };

   const handlePinComplete =
async (pin) => {
    if (pin.length === 4) {
                        await
AsyncStorage.setItem('user_
pin', pin);
      setAuthError('Authenticati
on Set Successfully');
      setTimeout(() => {
        navigation.navigate('Se
miApp');
      }, 1000)
    }
  };
                        const
handleFingerprintSetup =
async () => {
       const result = await
LocalAuthentication.authentic
ateAsync({
       promptMessage: 'Set up
fingerprint',
    });
    if (result.success) {
      setAuthError('Authenticati
on Set Successfully');
      setTimeout(() => {
        navigation.navigate('Se
miApp');
      }, 1000)
    } else {
      setAuthError('Fingerprint
Setup Failed');
    }
  };

  useEffect(() => {
    if (enteredPin.length > 0) {
      setShowRemoveButton(tr
ue);
    } else {
      setShowRemoveButton(f
alse);
    }
    if (enteredPin.length === 4)
{
      setShowCompletedButto
n(true);
    } else {
      setShowCompletedButto
n(false);
    }
  }, [enteredPin]);

  let [fontsLoaded, fontError] =
useFonts({
    TitilliumWeb_400Regular,
  });
```

```
    if (!fontsLoaded && !fontError) {
   return null;
  }

  return (
                <View style={styles.container}>
     <LinearGradient
              colors={['#4c669f', '#f0ceff']}
       style={styles.gradient}
       start={[0.5, 0.5]}
     >
                <Text style={styles.title}>Set Up Authentication</Text>
          {isBiometricSupported && (
            <FontAwesomeIcon icon={faFingerprint} size={150} style={{ color: "#fff", marginBottom: 20 }} />
       )}
       {isBiometricSupported ? (
        <Pressable
          style={styles.fingerprintButton}
          onPress={handleFingerprintSetup}
        >
                <Text style={styles.fingerprintButtonText}>Authenticate</Text>
        </Pressable>
       ) : (
        <ReactNativePinView
          onComplete={handlePinComplete}
          ref={pinView}
          pinLength={4}
          inputSize={32}
          buttonSize={60}
          onValueChange={value => setEnteredPin(value)}
          buttonAreaStyle={styles.buttonArea}
          inputAreaStyle={styles.inputArea}
          inputViewEmptyStyle={styles.inputViewEmpty}
          inputViewFilledStyle={styles.inputViewFilled}
          buttonViewStyle={styles.buttonView}
          buttonTextStyle={styles.buttonText}
          onButtonPress={key => {
            if (key === "custom_left") {
              pinView.current.clear();
            } else if (key === "custom_right") {
              handlePinComplete(enteredPin, pinView.current.clear);
            }
          }}
          customLeftButton={showRemoveButton ? <FontAwesomeIcon icon={faDeleteLeft} size={48} color={"#000"} /> : undefined}
          customRightButton={showCompletedButton ? <FontAwesomeIcon icon={faUnlock} size={48} color={"#000"} /> : undefined}
        />
       )}
       {authError ? (
          <Text style={{ color: authError === 'Authentication Set Successfully' ? '#00FF00' : 'red', marginTop: 20, fontFamily: 'TitilliumWeb_400Regular' }}>
            {authError}
          </Text>
       ) : null}
     </LinearGradient>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  gradient: {
    position: 'absolute',
    left: 0,
    right: 0,
    top: 0,
    height: '100%',
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  title: {
    fontSize: 24,
    color: '#fff',
    marginBottom: 50,
                fontFamily: 'TitilliumWeb_400Regular',
  },
  fingerprintButton: {
    backgroundColor: '#fff',
    padding: 10,
    borderRadius: 5,
    marginTop: 20,
  },
  fingerprintButtonText: {
                fontFamily: 'TitilliumWeb_400Regular',
  },
  buttonArea: {
    marginTop: 24
  },
  inputArea: {
    marginBottom: 25
  },
  inputViewEmpty: {
              backgroundColor: 'transparent',
    borderWidth: 3,
    borderColor: '#fff'
  },
  inputViewFilled: {
    backgroundColor: '#fff'
  },
  buttonView: {
    backgroundColor: '#000',
    borderWidth: 3,
    borderColor: '#fff'
  },
  buttonText: {
    color: '#fff'
  },
});

export default SignUpScreen;

import React, { useState, useEffect } from 'react';
import { ToastAndroid, Alert, BackHandler, TouchableOpacity,
```

```
KeyboardAvoidingView, Image, Pressable, Text, TextInput, View, StyleSheet } from 'react-native';
import { LinearGradient } from 'expo-linear-gradient';
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import { useNavigation } from '@react-navigation/native';
import { MaterialCommunityIcons } from 'react-native-vector-icons';
import * as ImagePicker from 'expo-image-picker';
import { ImageProvider, useImage } from './ImageContext';
import { app } from '../firebaseConfig';
import { getAuth, createUserWithEmailAndPassword, sendEmailVerification } from "firebase/auth";
import { getFirestore, doc, setDoc } from "firebase/firestore";
import { getStorage, ref, uploadBytes, getDownloadURL } from "firebase/storage";
import { registerForPushNotificationsAsync } from './Notifications';
import AsyncStorage from '@react-native-async-storage/async-storage';
import * as SecureStore from 'expo-secure-store';
import RSA from 'react-native-rsa-native';

const RegisterScreen = () => {
    const navigation = useNavigation();

    const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [confirmPassword, setConfirmPassword] = useState("");
  const [error, setError] = useState("");
    const [authError, setAuthError] = useState("");
  const [matched, setMatched] = useState("");
  const [loading, setLoading] = useState(false);
    const imageProps = useImage() || {};
    const image = imageProps.image || null;
    const setImage = imageProps.setImage || (() => { });
  const auth = getAuth(app);
    const firestore = getFirestore(app);
    const storage = getStorage(app);

  const validatePassword = (password) => {
    const errors = [];
    if (password.length < 8) {
      errors.push("Password must be at least 8 characters long.");
    }
    if (!password.match(/[A-Z]/)) {
      errors.push("Password must contain at least one uppercase letter.");
    }
    if (!password.match(/[a-z]/)) {
      errors.push("Password must contain at least one lowercase letter.");
    }
    if (!password.match(/[0-9]/)) {
      errors.push("Password must contain at least one number.");
    }
    if (!password.match(/[\W_]/)) {
      errors.push("Password must contain at least one special character.");
    }
    return errors;
  };
  const generateKeyPair = async () => {
    const keys = await RSA.generateKeys(2048);
    const { public: publicKey, private: privateKey } = keys;

    await SecureStore.setItemAsync('privateKey', privateKey);
    return publicKey;
  }

  const pressSignup = async () => {
    const formattedUsername = username.trim().charAt(0).toUpperCase() + username.slice(1);
    if (!image) {
      setError('Profile picture is required');
      return;
    }
    if (!username.trim()) {
      setError('Username is required');
      return;
    }

    if (!password || !confirmPassword) {
      setError('Fill in the required fields.');
      setMatched('');
      return;
    }

    if (password !== confirmPassword) {
      setError('Passwords do not match');
      setMatched('');
      return;
    } else {
      setError('');
      setMatched('Passwords matched');
```

```
  }

    const passwordErrors = validatePassword(password);
  if (passwordErrors.length > 0) {
    setError(passwordErrors.join("\n"));
    return;
  }

  setLoading(true);

  try {
    const userCredential = await createUserWithEmailAndPassword(auth, email, password);
          const user = userCredential.user;

                        await sendEmailVerification(user);
    ToastAndroid.show("Email verification sent!", ToastAndroid.SHORT);

    const checkEmailVerified = async () => {
      await user.reload();
      return user.emailVerified;
    }

    let emailVerified = await checkEmailVerified();
    const maxAttempts = 10;
    let attempts = 0;

    while (!emailVerified && attempts < maxAttempts) {
                await new Promise(resolve => setTimeout(resolve, 3000));
        emailVerified = await checkEmailVerified();
      attempts++;
    }

    if (!emailVerified) {
      throw new Error("Email verification failed. Please try again later.");
    }

    let imageUrl = '';
    if (image) {
      const response = await fetch(image);
          const blob = await response.blob();
          const storageRef = ref(storage, `profilePictures/${user.uid}`);
                        await uploadBytes(storageRef, blob);
          imageUrl = await getDownloadURL(storageRef);
    }

        const token = await registerForPushNotificationsAsync(user.uid);
    const publicKey = await generateKeyPair();

          const userDoc = doc(firestore, "users", user.uid);
    await setDoc(userDoc, {
      uid: user.uid,
                        username: formattedUsername,
      email: email,
      profilePicture: imageUrl,
      publicKey: publicKey,
      expoPushToken: token,
    });

        setAuthError("Account created successfully!");
    setTimeout(() => {
      navigation.navigate("SignUpAuth");
    }, 1000);
  } catch (error) {
        const errorCode = error.code;
      const errorMessage = error.message;

    switch (errorCode) {
     case "auth/invalid-email":
      setAuthError("Fill in the required fields.");
      break;
        case "auth/missing-password":
      setAuthError("Fill in your password.");
      break;
        case "auth/missing-email":
      setAuthError("Fill in your email address.");
      break;
          case "auth/weak-password":
      setAuthError("Password is too weak. It must be at least 6 characters long.");
      break;
    case "auth/email-already-in-use":
      setAuthError("The email address is already in use by another account.");
      break;
     default:
      Alert.alert(
       "SAFE-ON-CHAT",
       `Account creation error occurred. Please try again later. Error: ${errorMessage}`
      );
      break;
    }
  } finally {
    setLoading(false);
  }
 };

 useEffect(() => {
  const backAction = () => {
    navigation.navigate("Land");
    return true;
  };

    const backHandler = BackHandler.addEventListener(
    "hardwareBackPress",
    backAction
  );

        return () => backHandler.remove();
 }, []);

 let [fontsLoaded, fontError] = useFonts({
  TitilliumWeb_400Regular,
```

```
  TitilliumWeb_600SemiBold,
 });

     if (!fontsLoaded &&
!fontError) {
   return null;
 }

  const pickImage = async ()
=> {
       let result = await
ImagePicker.launchImageLib
raryAsync({
                  mediaTypes:
ImagePicker.MediaTypeOptio
ns.Images,
    allowsEditing: true,
    aspect: [4, 3],
    quality: 1,
         }).catch((error) =>
console.log(error));

     if (!result.canceled &&
result.assets) {
    setImage(result.assets[0].
uri);
   }
 };

 return (
                        <View
style={styles.container}>
    <LinearGradient
            colors={['#4c669f',
'#f0ceff']}
      style={styles.linearg}
      start={[0.5, 0.5]}
    >
      <KeyboardAvoidingView
       style={styles.container}
       behavior='padding'
      >
       <Image
        style={styles.logo}
         source={require('../as
sets/soc.png')}
       />
       <Text style={{
                  fontFamily:
'TitilliumWeb_600SemiBold',
        fontSize: 30,
        alignSelf: 'center',
        color: '#fff',
        marginTop: 10,
        marginBottom: 30,
      }}>Safe on Chat</Text>
          <TouchableOpacity
onPress={pickImage}>
        {!image && (
         <View
          style={{
                 flexDirection:
"column",
           alignItems: "center"
          }}
         >
          <MaterialCommunit
yIcons
           name="upload"
           size={24}
           color="#ffffff"
          />
          <Text
           style={{
            marginTop: 10,
           marginBottom: 20,
            color: "#ffffff",
                 fontFamily:
"TitilliumWeb_600SemiBold",
           }}
          >
              Upload Profile
Picture
          </Text>
         </View>
        )}
        {image && (
         <Image
          source={{ uri: image
}}
          style={{
           marginBottom: 40,
           width: 75,
           height: 75,
           borderRadius: 50,
           borderWidth: 2,
                borderColor:
"#ffffff",
          }}
         />
        )}
       </TouchableOpacity>
                          <View
style={styles.inputs}>
        <TextInput
         placeholderTextColo
r={'rgb(200, 200, 200)'}
         placeholder='Userna
me'
          style={[styles.input, {
marginTop: 0 }]}
          autoCapitalize='none'
          value={username}
         onChangeText={(text)
=> setUsername(text)}
        />
        <TextInput
         placeholderTextColo
r={'rgb(220, 220, 220)'}
          placeholder='Email'
          style={styles.input}
         autoCapitalize='none'
          keyboardType="ema
il-address"
          value={email}
         onChangeText={(text)
=> setEmail(text)}
        />
        <TextInput
         placeholderTextColo
r={'rgb(220, 220, 220)'}
          placeholder='Passw
ord'
          style={styles.input}
         autoCapitalize='none'
          secureTextEntry={tr
ue}
          value={password}
         onChangeText={(text)
=> setPassword(text)}
        />
        <TextInput
         placeholderTextColo
r={'rgb(220, 220, 220)'}
          placeholder='Confirm
Password'
          style={styles.input}
         autoCapitalize='none'
          secureTextEntry={tr
ue}
          value={confirmPass
word}
        onChangeText={(text)
=>
setConfirmPassword(text)}
        />
              {error ? <Text
style={styles.errorText}>{erro
r}</Text>          :          <Text
style={styles.matchedpass}>{
matched}</Text>}
        {authError ? (
         <Text style={{
```

```
                color: authError.includes('Account created successfully!') ? '#00FF00' : 'red',
                marginTop: 20,
                fontFamily: 'TitilliumWeb_400Regular',
                textAlign: 'center',
              }}
              >
                {authError}
              </Text>
            ) : null}
          </View>
          <Pressable
            style={({ pressed }) => [
              {
                backgroundColor: pressed
                  ? 'rgb(210, 230, 255)'
                  : 'rgb(255, 255, 255)',
              },
              styles.signupbutton
            ]}
            onPress={pressSignup}
            disabled={loading}
          >
                <Text style={styles.signuptext}>SIGN UP</Text>
          </Pressable>
          </KeyboardAvoidingView>
      </LinearGradient>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  content: {
    justifyContent: "center",
    alignItems: "center",
  },
  linearg: {
    position: 'absolute',
    left: 0,
    right: 0,
    top: 0,
    height: '100%',
  },
  inputs: {
    flexDirection: 'column',
    justifyContent: 'center',
    alignItems: 'center',
  },
  input: {
    marginTop: 25,
    textAlign: 'center',
    width: 225,
    height: 50,
    backgroundColor: '#fff',
    alignSelf: 'center',
    justifyContent: 'center',
    alignItems: 'center',
    borderWidth: 1,
    borderColor: '#000000',
    borderRadius: 10,
    fontFamily: 'TitilliumWeb_600SemiBold',
    fontSize: 15,
  },
  logo: {
    width: 100,
    height: 100,
    backgroundColor: '#fff',
    borderRadius: 25,
    alignSelf: 'center',
  },
  signupbutton: {
    marginTop: 30,
    width: 175,
    height: 40,
    alignSelf: 'center',
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 10,
  },
  signuptext: {
    fontFamily: 'TitilliumWeb_600SemiBold',
    fontSize: 17.5,
  },
  errorText: {
    color: '#ff0000',
    marginTop: 10,
    fontFamily: 'TitilliumWeb_600SemiBold',
  },
  matchedpass: {
    color: '#00ff00',
    marginTop: 10,
    fontFamily: 'TitilliumWeb_600SemiBold',
  }
});

export default function AppWrapper() {
  return (
    <ImageProvider>
      <RegisterScreen />
    </ImageProvider>
  )
};

import React, { useEffect, useState } from 'react'
import { BackHandler, FlatList, Image, TouchableOpacity, Pressable, StyleSheet, Text, View } from 'react-native'
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import { useNavigation } from '@react-navigation/native';
import { LinearGradient } from 'expo-linear-gradient';
import { SearchBar } from '@rneui/themed';
import { Divider, Avatar } from 'react-native-elements';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faUserGroup } from '@fortawesome/free-solid-svg-icons';
import { getAuth } from 'firebase/auth';
import { getFirestore, doc, getDoc, collection, getDocs } from 'firebase/firestore';
import { app } from '../firebaseConfig';

const Item = ({ user, onPress }) => (
  <Pressable
    onPress={() => onPress(user)}
    style={({ pressed }) => [
```

```
    {
              backgroundColor:
pressed
        ? '#4c669f'
        : '#f0ceff',
      borderColor: pressed
        ? '#f0ceff'
        : '#4c669f',
      borderWidth: 1,
    },
    {
      paddingHorizontal: 10,
      paddingVertical: 5,
      marginVertical: 10,
      marginHorizontal: 10,
      borderRadius: 10,
    }
  ]}
 >
  {({ pressed }) => (
    <View style={{
      flexDirection: 'row',
      paddingVertical: 2.5,
      paddingHorizontal: 5,
    }}>
      <View>
            <Avatar  size={48}
rounded
source={user.profilePicture  ?
{  uri:  user.profilePicture  }  :
require('../assets/profilepic.jp
g')} />
      </View>
      <View>
        <Text style={{
                    fontFamily:
'TitilliumWeb_400Regular',
          fontSize: 20,
          paddingLeft: 10,
          paddingVertical: 10,
                textAlignVertical:
'center',
          color: pressed
            ? '#fff'
            : '#000',
        }}>{user.username}</T
ext>
      </View>
    </View>
  )}
 </Pressable>
);

const SearchChat = () => {
      const      [profilePicture,
setProfilePicture]          =
useState('');
  const  [users,  setUsers]  =
useState([]);
      const      [filteredUsers,
setFilteredUsers]          =
useState([]);
 const auth = getAuth(app);
      const      firestore      =
getFirestore(app);
     const      navigation      =
useNavigation();
       const       [userInput,
setUserInput] = useState('');

 const fetchUsers = async ()
=> {
   try {
     const usersCollection =
collection(firestore, 'users');
      const  userSnapshot  =
await
getDocs(usersCollection);
          const  userList  =
userSnapshot.docs.map(doc
=> ({ id: doc.id, ...doc.data()
}));
    setUsers(userList);
    setFilteredUsers(userList);
   } catch (error) {
          console.error('Error
fetching users: ', error);
   }
 };

  const fetchProfilePicture =
async () => {
   try {
          const  user  =
auth.currentUser;
    if (user) {
          const  userDoc  =
doc(firestore,          'users',
user.uid);
      const userSnap = await
getDoc(userDoc);
      if (userSnap.exists()) {
          const  userData  =
userSnap.data();
       setProfilePicture(userD
ata.profilePicture || '')
      } else {
        console.log('No  such
document!');
      }
    }
   } catch (error) {
          console.error('Error
fetching  profile  picture:  ',
error);
   }
 };

 useEffect(() => {
  fetchProfilePicture();
  fetchUsers();
  const backAction = () => {
   navigation.navigate("Chat
s");
    return true;
  };

      const  backHandler  =
BackHandler.addEventListen
er(
    "hardwareBackPress",
    backAction
  );

          return      ()      =>
backHandler.remove();
 }, []);

 useEffect(() => {
  setFilteredUsers(
         users.filter(user  =>
user.username.toLowerCase(
).includes(userInput.toLower
Case()))
  );
 }, [userInput, users]);

  const handleUserPress =
(user) => {
  if (user.uid) {
    navigation.navigate('Chat
Screen', { user, username:
user.username,
profilePicture:
user.profilePicture,        uid:
user.uid });
  } else {
    console.error('User does
not have a valid UID');
  }
 };

 let [fontsLoaded, fontError] =
useFonts({
```

```
  TitilliumWeb_400Regular,
  TitilliumWeb_600SemiBold,
 });

    if (!fontsLoaded && !fontError) {
  return null;
 }

 return (
                        <View style={styles.container}>
    <LinearGradient
              colors={['#4c669f', '#f0ceff']}
       style={styles.linearGradient}
      start={[0.5, 0.5]}
    >
                        <View style={styles.content}>
       <View styles={{
        flexDirection: 'row',
        justifyContent: 'center',
        alignItems: 'center',
        margin: 50,
       }}>
        <SearchBar
         round
           searchIcon={{ size: 24 }}
                placeholder="Search"
         onChangeText={(text) => setUserInput(text)}
          value={userInput}
          containerStyle={{
              backgroundColor: 'transparent',
           borderBottomWidth: 0,
            borderTopWidth: 0,
          }}
          inputStyle={{
           color: '#fff',
                  fontFamily: 'TitilliumWeb_400Regular',
          }}
          underlineColorAndroid={'transparent'}
          cursorColor={'#fff'}
         />
       </View>
       <View>
        <Pressable
          onPress={() =>
           navigation.navigate('CreateGroupChat')
          }
           style={({ pressed }) => [
            {
             backgroundColor: pressed
              ? '#4c669f'
              : 'white',
                 borderColor: pressed
              ? 'white'
              : '#4c669f',
             borderWidth: 1,
            },
            {
             paddingHorizontal: 10,
             paddingVertical: 5,
             marginVertical: 10,
              marginHorizontal: 10,
             borderRadius: 10,
            }
           ]}
          >
           {({ pressed }) => (
            <View style={{
             flexDirection: 'row',
               paddingVertical: 2.5,
             paddingHorizontal: 5,
            }}>
             <View
              style={{
                 justifyContent: 'center',
                    alignItems: 'center',
               paddingVertical: 2.5,
                paddingHorizontal: 5,
              }}
              >
               <FontAwesomeIcon icon={faUserGroup} size={30} color={pressed ? 'white' : '#4c669f'} />
              </View>
              <View>
               <Text style={{
                  fontFamily: 'TitilliumWeb_400Regular',
                 fontSize: 20,
                 paddingLeft: 10,
                 paddingVertical: 10,
                textAlignVertical: 'center',
                 color: pressed
                  ? '#fff'
                  : '#4c669f',
                }}>Create Group</Text>
              </View>
             </View>
            )}
           </Pressable>
         </View>
         <Divider
          style={{
              backgroundColor: "#f0ceff",
           marginVertical: 5,
           width: '100%',
           height: 2,
          }} />
         {userInput === '' ? (
          <View style={{
           flex: 1,
           marginTop: 125,
          }}>
                        <Text style={styles.temp_text}>Search Contacts.</Text>
          </View>
         ) : (filteredUsers.length === 0 ? (
          <View style={{
           flex: 1,
           marginTop: 125,
          }}>
                        <Text style={styles.temp_text}>No Results Available. </Text>
          </View>
         ) : (
          <FlatList
            showsVerticalScrollIndicator={false}
           data={filteredUsers}
           renderItem={({ item }) => <Item user={item} onPress={handleUserPress} />}
```

```
          keyExtractor={item => item.id}
          style={{ marginTop: 10, paddingBottom: 10 }}
        />
      )
      )}
    </View>
   </LinearGradient>
  </View>
 )
}

const styles = StyleSheet.create({
 container: {
  flex: 1,
  justifyContent: 'center',
 },
 content: {
  flex: 1,
  paddingTop: 10,
  paddingBottom: 0,
  padding: 10,
 },
 textheader: {
                fontFamily: 'TitilliumWeb_400Regular',
  fontSize: 25,
  color: 'hsl(0, 0%, 100%)',
  textAlignVertical: 'center',
  marginLeft: 10,
 },
 profilepic: {
  width: 50,
  height: 50,
  borderRadius: 40,
 },
 header: {
  flexDirection: 'row',
  paddingTop: 15,
  paddingBottom: 10,
 },
 temp_text: {
                fontFamily: 'TitilliumWeb_600SemiBold',
  fontSize: 25,
  color: '#fff',
  textAlign: 'center',
 },
 title: {
  fontSize: 20,
                fontFamily: 'TitilliumWeb_400Regular',
  paddingLeft: 10,
  textAlignVertical: 'center',
 },
 linearGradient: {
  flex: 1,
 }
})

export default SearchChat;

import React, { useState } from 'react';
import { Alert, Text, View, Image, Pressable, StyleSheet, ToastAndroid } from 'react-native';
import { UserProvider, useUser } from './UserContext';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { createDrawerNavigator, DrawerContentScrollView, DrawerItemList } from '@react-navigation/drawer';
import Chats from './Chats';
import Calls from './Calls';
import SettingsScreen from './SettingsScreen';
import { LinearGradient } from 'expo-linear-gradient';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faCommentDots, faPhone, faArrowRightFromBracket } from '@fortawesome/free-solid-svg-icons';
import { getAuth, signOut } from 'firebase/auth';
import { useNavigation } from '@react-navigation/native';
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';

const Tab = createBottomTabNavigator();
const Drawer = createDrawerNavigator();

const CustomDrawerContent = (props) => {
 const auth = getAuth();
    const navigator = useNavigation();
 const { user } = useUser();

  // Function to handle the actual logout
 const handleLogout = async () => {
   Alert.alert(
    "LOG OUT",
    "Are you sure you want to log out?",
    [
     {
      text: "Cancel",
      onPress: () => {
       ToastAndroid.show("Logout cancelled", ToastAndroid.SHORT);
      },
      style: "cancel"
     },
     {
      text: "OK", onPress: async () => {
       await signOut(auth);
       navigator.navigate("Land");
      }
     }
    ]
   )
 };

 let [fontsLoaded, fontError] = useFonts({
   TitilliumWeb_400Regular,
  TitilliumWeb_600SemiBold,
 });

    if (!fontsLoaded && !fontError) {
  return null;
 }

 return (
  <DrawerContentScrollView {...props}>
   <View style={{ flex: 1 }}>
                <View style={styles.drawerHeader}>
```

```
        <Image
         style={styles.profilepic}
                                  //
source={require('../assets/pro
filepic.jpg')}
                source={user &&
user.profilePicture ? { uri:
user.profilePicture } :
require('../assets/profilepic.jp
g')}
        />
                {user && <Text
style={styles.drawerHeaderT
ext}>{user.username}</Text>
}
       </View>
                 <DrawerItemList
{...props} />
     </View>
     <Pressable
       style={{
         marginBottom: 20,
         paddingVertical: 15,
         paddingHorizontal: 10,
         borderRadius: 5,
       }}
        onPress={handleLogout
}>
       <Text style={{
                        fontFamily:
'TitilliumWeb_600SemiBold',
         fontSize: 16,
         color: '#fff',
                backgroundColor:
'#ff0000',
         textAlign: 'center',
         borderRadius: 10,
         padding: 10,
       }}>
         {/* <FontAwesomeIco
          */}
         Log Out
       </Text>
     </Pressable>
     </DrawerContentScrollVie
w>
  );
};

const ChatsScreen = () => {
  return (
                              <View
style={styles.container}>
     <LinearGradient
                colors={['#4c669f',
'#f0ceff']}
         style={styles.linearGradi
ent}
       start={[0.5, 0.5]}
     >
       <Chats />
     </LinearGradient>
   </View>
  );
};

const CallsScreen = () => {
  return (
                                  <View
style={styles.container}>
     <LinearGradient
                colors={['#4c669f',
'#f0ceff']}
         style={styles.linearGradi
ent}
       start={[0.5, 0.5]}
     >
       <Calls />
     </LinearGradient>
   </View>
  );
};

// const Settings = () => {
//   return (

const TabNavigator = () => {
  let [fontsLoaded, fontError] =
useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });

      if     (!fontsLoaded     &&
!fontError) {
    return null;
  }

  return (
    <Tab.Navigator
      screenOptions={{
        tabBarStyle: {
          height: 65,
          paddingBottom: 10,
          paddingTop: 20,
          backgroundColor: '#eff',
          borderTopWidth: 0,
        },
        tabBarLabelStyle: {
                                  fontFamily:
'TitilliumWeb_600SemiBold',
          fontSize: 15,
          color: ({ focused }) => (
                focused  ?  '#00ff'  :
'#000'
          )
        },
      }}
    >
                    <Tab.Screen
name="Chats"
component={ChatsScreen}
options={{
        headerShown: false,
        tabBarIcon: ({ focused })
=> (
          <FontAwesomeIcon
            icon={faCommentDot
s}
                  color={focused  ?
"#000" : "#4c669f"}
            size={25}
            style={{
              marginBottom: 15,
            }}
          />
        ),
      }}
      />
                    <Tab.Screen
name="Calls"
component={CallsScreen}
options={{
        headerShown: false,
        tabBarIcon: ({ focused })
=> (
          <FontAwesomeIcon
            icon={faPhone}
                  color={focused  ?
"#000" : "#4c669f"}
            size={25}
            style={{
              marginBottom: 15,
            }}
          />
        ),
      }} />
    </Tab.Navigator>
  );
};

const DrawerNavigator = () =>
{
  return (
```

```
    <Drawer.Navigator
      drawerContent={(props) => <CustomDrawerContent {...props} />}
      screenOptions={{
        drawerStyle: {
          backgroundColor: '#f0ceff',
          width: 250,
        },
        drawerType: 'front',
        drawerLabelStyle: {
          fontFamily: 'TitilliumWeb_600SemiBold',
        },
        gestureEnabled: false,
      }}
    >
      <Drawer.Screen
        name="Home"
        component={TabNavigator}
        options={{
          headerShown: false,
          drawerLabel: 'Chats',
        }}
      />
      {/
      /> */}
    </Drawer.Navigator>
  );
};

const App = () => {
  return (
    <UserProvider>
      <DrawerNavigator />
    </UserProvider>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  linearGradient: {
    flex: 1,
  },
  drawerHeader: {
    flexDirection: 'row',
    alignItems: 'center',
    paddingVertical: 20,
    paddingHorizontal: 15,
    borderBottomWidth: 1,
    borderBottomColor: '#ccc',
    backgroundColor: "#4c669f",
  },
  profilepic: {
    width: 50,
    height: 50,
    borderRadius: 40,
    marginRight: 10,
  },
  drawerHeaderText: {
    fontFamily: 'TitilliumWeb_400Regular',
    fontSize: 20,
    color: '#fff',
  },
});

export default App;



import React, { useEffect, useState, useCallback } from 'react'
import { BackHandler, FlatList, Image, Pressable, StyleSheet, Text, View } from 'react-native'
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import { useNavigation, useFocusEffect } from '@react-navigation/native';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faPenToSquare } from '@fortawesome/free-solid-svg-icons';
import { Avatar } from 'react-native-elements';
import { getAuth } from 'firebase/auth';
import { getFirestore, doc, getDoc } from 'firebase/firestore';
import { app } from '../firebaseConfig';

const Settings = () => {
    const [profilePicture, setProfilePicture] = useState('');
  const auth = getAuth(app);
    const firestore = getFirestore(app);

   const fetchProfilePicture = async () => {
     try {
       const user = auth.currentUser;
       if (user) {
         const userDoc = doc(firestore, 'users', user.uid);
         const userSnap = await getDoc(userDoc);
         if (userSnap.exists()) {
           const userData = userSnap.data();
           setProfilePicture(userData.profilePicture || '')
         } else {
           console.log('No such document!');
         }
       }
     } catch (error) {
       console.error('Error fetching profile picture: ', error);
     }
   };

  useFocusEffect(
    useCallback(() => {
      fetchProfilePicture();
      const onBackPress = () => {
        BackHandler.exitApp();
        return true;
      };

      BackHandler.addEventListener('hardwareBackPress', onBackPress);

      return () =>
        BackHandler.removeEventListener('hardwareBackPress', onBackPress);
    }, [])
  );
```

```
      const navigation = useNavigation();

  let [fontsLoaded, fontError] = useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });


     if (!fontsLoaded && !fontError) {
    return null;
  }

  return (
                              <View style={styles.container}>
                              <View style={styles.content}>
                              <View style={styles.header}>
          <Pressable
            onPress={() => {
               navigation.openDrawer();
            }}
            style={({ pressed }) => [
              {
                 opacity: pressed ? 0.5 : 1,
              }, {
                borderRadius: 50,
                borderWidth: 2.5,
                 borderColor: 'hsl(0, 0%, 100%)',
              }
            ]}
          >
            <Avatar
              size={48}
              rounded
             source={profilePicture ? { uri: profilePicture } : require('../assets/profilepic.jpg')}
            />
          </Pressable>
                              <Text style={styles.textheader}>
            Safe-on-chat
          </Text>
        </View>
        </View>
      )
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  content: {

    paddingTop: 40,
    paddingBottom: 0,
    paddingLeft: 10,
    padding: 10,
  },
  textheader: {
                              fontFamily: 'TitilliumWeb_400Regular',
    fontSize: 25,
    color: 'hsl(0, 0%, 100%)',
    textAlignVertical: 'center',
    marginLeft: 10,
  },
  profilepic: {
    width: 50,
    height: 50,
    borderRadius: 40,
  },
  header: {
    flexDirection: 'row',
    paddingTop: 15,
    paddingBottom: 10,
  },
  temp_text: {
                              fontFamily: 'TitilliumWeb_600SemiBold',
    fontSize: 25,
    color: '#fff',
    textAlign: 'center',
  },
})

export default Settings;

// UserContext.js
import React, { createContext, useContext, useState, useEffect } from 'react';
import { app } from '../firebaseConfig';
import { getAuth, onAuthStateChanged } from "firebase/auth";
import { getFirestore, doc, getDoc } from "firebase/firestore";

const UserContext = createContext();

export const useUser = () => useContext(UserContext);

export const UserProvider = ({ children }) => {
   const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const auth = getAuth(app);
      const firestore = getFirestore(app);

  useEffect(() => {
       const unsubscribe = onAuthStateChanged(auth, async (currentUser) => {
     if (currentUser) {
        const userDoc = await getDoc(doc(firestore, "users", currentUser.uid));
       if (userDoc.exists()) {
                    setUser({  uid: currentUser.uid, ...userDoc.data() });
       }
     } else {
       setUser(null);
     }
     setLoading(false);
   });

   return () => unsubscribe();
  }, []);

  return (
        <UserContext.Provider value={{ user, loading }}>
     {children}
   </UserContext.Provider>
  );
};
```

```
import React, { useEffect, useRef, useState, useCallback } from 'react';
import { View, Image, Text, StyleSheet, Pressable, Animated } from 'react-native';
import { RTCView, mediaDevices, RTCPeerConnection, RTCSessionDescription } from 'react-native-webrtc';
import io from 'socket.io-client';
import { app } from '../firebaseConfig';
import { getFirestore, addDoc, collection, getDoc, doc } from 'firebase/firestore';
import { FontAwesomeIcon } from '@fortawesome/react-native-fontawesome';
import { faPhoneSlash, faPhone, faSync, faMicrophone, faMicrophoneSlash, faVideo, faVideoSlash } from '@fortawesome/free-solid-svg-icons';
import { sendVideoCallNotification, getPushTokenForUser, getUserDetailsForNotification } from './Notifications';
import { useFocusEffect } from '@react-navigation/native';
import { getAuth } from 'firebase/auth';

const VideoCallScreen = ({ route, navigation }) => {
  const { user, profilePicture, callerInfo } = route.params;
    const [localStream, setLocalStream] = useState(null);
    const [remoteStream, setRemoteStream] = useState(null);
    const [isConnected, setIsConnected] = useState(false);
    const [callStarted, setCallStarted] = useState(false);
  const [isMicOn, setIsMicOn] = useState(true);
  const [isUsingFrontCamera, setIsUsingFrontCamera] = useState(true);
    const [isCameraOn, setIsCameraOn] = useState(true);
    const [isCameraOff, setIcCameraOff] = useState(false);
  const [callerProfilePicture, setCallerProfilePicture] = useState(profilePicture || null);
    const socketRef = useRef(null);
  const pcRef = useRef(null);
    const firestore = getFirestore(app);
  const auth = getAuth(app);

  console.log("Route Params:", route.params);
  useFocusEffect(
    useCallback(() => {
      if (callerInfo) {
        initializeSocket();
      }
      if (route.params.autoStart) {
        startLocalStream();
      }
    }, [callerInfo, route.params.autoStart])
  );

  useEffect(() => {
    const initializeProfilePicture = async () => {
      if (!callerProfilePicture) {
        const profilePictureUrl = await fetchUserProfilePicture(callerInfo.uid);
        setCallerProfilePicture(profilePictureUrl);
      }
    };

    initializeProfilePicture();
  }, [callerProfilePicture, user]);

  const fetchUserProfilePicture = async (userId) => {
    try {
      const firestore = getFirestore(app);
      const userDoc = await getDoc(doc(firestore, 'users', userId));

      if (userDoc.exists()) {
        return userDoc.data().profilePicture || './assets/profilepic.jpg';
      } else {
        console.warn('User not found in Firestore.');
        return './assets/profilepic.jpg'; // Default profile picture
      }
    } catch (error) {
      console.error('Error fetching user profile picture:', error);
      return './assets/profilepic.jpg'; // Default profile picture
    }
  };

  const AnimatedPressable = ({ onPress, style, children }) => {
    const animatedScale = useRef(new Animated.Value(1)).current;

    const handlePressIn = () => {
      Animated.spring(animatedScale, {
        toValue: 0.9,
        useNativeDriver: true,
      }).start();
    }

    const handlePressOut = () => {
      Animated.spring(animatedScale, {
        toValue: 1,
        friction: 3,
        useNativeDriver: true,
      }).start();
```

```
  }
  return (
    <Pressable
      onPressIn={handlePressIn}
      onPressOut={handlePressOut}
      onPress={onPress}
    >
      <Animated.View style={[style, { transform: [{ scale: animatedScale }] }]}>
        {children}
      </Animated.View>
    </Pressable>
  );
}

const initializeSocket = () => {
    const socket = io('https://soc-thesis.onrender.com'); //! Change this to your server URL
  socketRef.current = socket;

  socket.on('connect', () => {
    console.log('Connected to signaling server');
    setIsConnected(true);
  });

    socket.on('offer', async (offer) => {
      console.log('Received offer:', offer);
    if (!pcRef.current) {
      pcRef.current = createPeerConnection();
    }
    try {
      await pcRef.current.setRemoteDescription(new RTCSessionDescription(offer));
      const answer = await pcRef.current.createAnswer();
      await pcRef.current.setLocalDescription(answer);
      socket.emit('answer', answer);
    } catch (error) {
      console.error('Error handling offer:', error);
    }
  });

    socket.on('answer', async (answer) => {
      console.log('Received answer:', answer);
    if (pcRef.current) {
      try {
        await pcRef.current.setRemoteDescription(new RTCSessionDescription(answer));
      } catch (error) {
        console.error('Error setting remote description:', error);
      }
    }
  });

    socket.on('ice-candidate', async (candidate) => {
    console.log('Received ICE candidate:', candidate);
    if (pcRef.current) {
      try {
        await pcRef.current.addIceCandidate(candidate);
      } catch (error) {
        console.error('Error adding ICE candidate:', error);
      }
    }
  });

  return () => {
    if (socketRef.current) {
      socketRef.current.disconnect();
      socketRef.current = null;
    }
    if (pcRef.current) {
      pcRef.current.close();
      pcRef.current = null;
    }
  };
};

// useEffect(() => {
//   initializeSocket();
// }, []);

const createPeerConnection = () => {
    const pc = new RTCPeerConnection({
    iceServers: [
      //? Google's public STUN server
      {
        urls: 'stun:stun.l.google.com:19302'
      },
      //? Metered TURN server
      {
        urls: "turn:asia.relay.metered.ca:80",
        username: "c1a44fd70f84e2d8ef05b4ac",
        credential: "XO88wG9Y0hTvj0XD",
      },
      {
        urls: "turn:asia.relay.metered.ca:80?transport=tcp",
        username: "c1a44fd70f84e2d8ef05b4ac",
        credential: "XO88wG9Y0hTvj0XD",
      },
      {
        urls: "turn:asia.relay.metered.ca:443",
        username: "c1a44fd70f84e2d8ef05b4ac",
        credential: "XO88wG9Y0hTvj0XD",
      },
      {
        urls: "turns:asia.relay.metered.ca:443?transport=tcp",
        username: "c1a44fd70f84e2d8ef05b4ac",
```

```
                credential: "XO88wG9Y0hTvj0XD",
      },
    });

    pc.onicecandidate = (event) => {
      if (event.candidate && socketRef.current) {
        socketRef.current.emit('ice-candidate', event.candidate);
      }
    };

    pc.ontrack = (event) => {
      if (event.streams && event.streams[0]) {
        setRemoteStream(event.streams[0]);
      }
    };

    pc.onconnectionstatechange = () => {
      console.log('Connection state changed:', pc.connectionState);
      if (pc.connectionState === 'closed' || pc.connectionState === 'failed') {
        console.log('Peer connection is closed');
        endCall();
      }
    };

    return pc;
  };

  const startLocalStream = async () => {
    if (!socketRef.current) {
      initializeSocket();
    }

    try {
      const stream = await mediaDevices.getUserMedia({
        audio: true,
        video: {
          facingMode: isUsingFrontCamera ? 'user' : 'environment',
        },
      });
      setLocalStream(stream);
      if (!pcRef.current) {
        pcRef.current = createPeerConnection();
      }
      stream.getTracks().forEach(track => {
        pcRef.current.addTrack(track, stream);
      });
      setCallStarted(true);
      await saveCallDetails();

      const callerInfo = await getUserDetailsForNotification(auth.currentUser.uid);
      const recipientDetails = await getUserDetailsForNotification(user.uid);
      if (recipientDetails && recipientDetails.expoPushToken && callerInfo) {
        console.log('Sending video call notification to:', user.username);
        await sendVideoCallNotification(recipientDetails.expoPushToken, callerInfo);
      }
    } catch (error) {
      console.error('Error starting local stream:', error);
    }
  };

  const saveCallDetails = async () => {
    try {
      const callDetails = {
        username: user.username,
        profilePicture: user.profilePicture,
        timestamp: new Date(),
        uid: user.uid,
      };
      await addDoc(collection(firestore, 'calls'), callDetails);
    } catch (error) {
      console.error('Error saving call details:', error);
    }
  };

  const createOffer = async () => {
    if (!pcRef.current || pcRef.current.connectionState === 'closed') {
      console.warn('Peer connection is not available or closed');
      return;
    }

    if (!socketRef.current) {
      console.warn('Socket connection is not available');
      return;
    }

    try {
      const offer = await pcRef.current.createOffer();
      await pcRef.current.setLocalDescription(offer);
      socketRef.current.emit('offer', offer);
    } catch (error) {
      console.error('Error creating offer:', error);
    }
  };

  const endCall = () => {
    if (pcRef.current) {
      pcRef.current.close();
      pcRef.current = null;
    }
    if (localStream) {
      localStream.getTracks().forEach(track => track.stop());
      setLocalStream(null);
    }
    if (remoteStream) {
      remoteStream.getTracks().forEach(track => track.stop());
      setRemoteStream(null);
    }
    if (socketRef.current) {
      socketRef.current.disconnect();
```

```
    socketRef.current = null;
    setIsConnected(false);
   }
   setCallStarted(false);
   setIsMicOn(true);
   setIsUsingFrontCamera(true);
   setIsCameraOn(true);
   setIcCameraOff(false);
  };

  useEffect(() => {
   return () => {
    endCall();
   }
  }, []);

  const toggleMicrophone = () => {
   if (localStream) {
    localStream.getAudioTracks().forEach(track => {
                track.enabled = !track.enabled;
      setIsMicOn(track.enabled);
    });
   }
  };

  const toggleCameraOnOff = () => {
   if (localStream) {
    localStream.getVideoTracks().forEach(track => {
                track.enabled = !track.enabled;
      setIsCameraOn(track.enabled);
      setIcCameraOff(!track.enabled);
       console.log('Camera is on:', track.enabled);
      if (socketRef.current) {
       socketRef.current.emit('toggle-camera', { isCameraOn: track.enabled });
      }
    });
   }
  };

  useEffect(() => {
   if (socketRef.current) {
      socketRef.current.on('toggle-camera', ({ isCameraOn }) => {
          console.log('Received camera toggle:', isCameraOn);
       setIsCameraOn(isCameraOn);
       setIcCameraOff(!isCameraOn);
      })
    }
  }, []);

  const switchCamera = async () => {
    setIsUsingFrontCamera(prevState => !prevState);
   if (localStream) {
     localStream.getTracks().forEach(track => track.stop());
    setLocalStream(null);
   }

   try {
      const stream = await mediaDevices.getUserMedia({
      audio: true,
      video: {
                facingMode: isUsingFrontCamera ? 'user' : 'environment',
      },
     });
    setLocalStream(stream);

    if (pcRef.current) {
         const videoTrack = stream.getVideoTracks()[0];
            const sender = pcRef.current.getSenders().find(s => s.track.kind === 'video');
      if (sender) {
       sender.replaceTrack(videoTrack);
      } else {
       pcRef.current.addTrack(videoTrack, stream);
      }
     }
   } catch (error) {
          console.error('Error switching camera:', error);
   }
  };

  return (
                <View style={styles.container}>
    {localStream && (
     <RTCView
      streamURL={localStream.toURL()}
      style={styles.rtcView}
      objectFit="cover"
     />
    )}
    {isCameraOff && (
                <View style={styles.blackScreen} />
    )}
    {remoteStream && (
     <RTCView
      streamURL={remoteStream.toURL()}
      style={styles.rtcView}
      objectFit="cover"
     />
    )}
    {isCameraOff && (
                <View style={styles.blackScreen} />
    )}
    <View style={{
     flexDirection: 'row',
      justifyContent: 'space-around',
     marginTop: 20,
     width: '70%',
    }}>
     {!callStarted ? (
      <View
       style={{
        alignItems: 'center',
            justifyContent: 'center',
        marginTop: 50,
       }}
      >
       <Image
        source={callerProfilePicture ? { uri: callerProfilePicture } : require('../assets/profilepic.jpg')}
        style={{
         width: 150,
         height: 150,
```

```
            borderRadius: 75,
            marginBottom: 10,
          }}
        />
                    <Pressable
style={styles.button}
onPress={startLocalStream}>
                      <Text
style={styles.buttonText}>Sta
rt Call</Text>
          </Pressable>
        </View>
      ) : (
        <>
          <AnimatedPressable
style={{
            ...styles.button,
              backgroundColor:
isMicOn ? '#fff' : '#f44336',
            marginRight: 20,
          }}
            onPress={toggleMicr
ophone}
          >
            <FontAwesomeIcon
icon={isMicOn ?
faMicrophone :
faMicrophoneSlash}
size={35} color={isMicOn ?
'#f44336' : "#fff"} />
          </AnimatedPressable>
          <AnimatedPressable
style={{
            ...styles.button,
              backgroundColor:
'#fff',
            marginRight: 20,
                                }}
onPress={createOffer}>
            <FontAwesomeIcon
icon={faPhone}      size={35}
color="#00ff66" />
          </AnimatedPressable>
          <AnimatedPressable
style={{
            ...styles.button,
              backgroundColor:
'#fff',
            marginRight: 20,
          }} onPress={endCall}>
            <FontAwesomeIcon
icon={faPhoneSlash}
size={35} color="#f44336" />
          </AnimatedPressable>
          <AnimatedPressable
style={{
            ...styles.button,
              backgroundColor:
isCameraOn ? '#fff' :
'#f44336',
            marginRight: 20,
          }}
            onPress={toggleCa
meraOnOff}
          >
            <FontAwesomeIcon
icon={isCameraOn ? faVideo
: faVideoSlash} size={35}
color={isCameraOn ?
"#f44336" : "#fff"} />
          </AnimatedPressable>
          <AnimatedPressable
style={{
            ...styles.button,
              backgroundColor:
'#fff',
                                }}
onPress={switchCamera}>
            <FontAwesomeIcon
icon={faSync}      size={35}
color="#f44336" />
          </AnimatedPressable>
        </>
      )}
    </View>
  </View>
 );
};

const      styles      =
StyleSheet.create({
 container: {
  flex: 1,
  alignItems: 'center',
              backgroundColor:
'#4c669f',
  position: 'relative',
 },
 rtcView: {
  width: '100%',
  height: '43%',
  backgroundColor: '#000',
  zIndex: 0,
 },
 blackScreen: {
  position: 'absolute',
  width: '100%',
  height: '43%',
  backgroundColor: '#000',
  zIndex: 1,
 },
 buttonContainer: {
  margin: 10,
 },
 button: {
  padding: 10,
  borderRadius: 5,
  backgroundColor: '#f0ceff',
  marginTop: 10,
 },
 buttonText: {
  color: '#000',
  fontSize: 20,
                fontFamily:
'TitilliumWeb_600SemiBold',
 },
});

export                default
VideoCallScreen;

import 'dotenv/config';

export default {
 "expo": {
  "name": "Safe on Chat",
  "slug": "socsysteeemm",
  "version": "1.0.0",
  "orientation": "portrait",
  "icon": "./assets/icon.png",
  "userInterfaceStyle": "light",
  "splash": {
                "image":
"./assets/splash.png",
    "resizeMode": "contain",
          "backgroundColor":
"#ffffff"
  },
  "ios": {
   "supportsTablet": true
  },
  "android": {
                "package":
"com.example.SOC",
    "adaptiveIcon": {
          "foregroundImage":
"./assets/adaptive-icon.png",
          "backgroundColor":
"#ffffff"
    },
    "permissions": [
     "CAMERA",
      "READ_EXTERNAL_ST
ORAGE",
```

```
      "WRITE_EXTERNAL_STORAGE",
      "INTERNET"
    ],
          "googleServicesFile": "./google-services.json"
  },
  "web": {
                  "favicon": "./assets/favicon.png"
  },
  "plugins": [
      "@config-plugins/react-native-webrtc",
    [
      "expo-notifications", {
                        "icon": "./assets/socdark.jpg",
        "color": "#ffffff",
              "defaultChannel": "default",
        "sounds": [
                "./assets/notif-sound/notif.wav",
        ]
      }
    ]
  ],
  extra: {
                    apiKey: process.env.API_KEY,
                authDomain: process.env.AUTH_DOMAIN,
                    projectId: process.env.PROJECT_ID,
                storageBucket: process.env.STORAGE_BUCKET,
          messagingSenderId: process.env.MESSAGING_SENDER_ID,
                    appId: process.env.APP_ID,
              measurementId: process.env.MEASUREMENT_ID,
    "eas": {
      "projectId": "82439ae0-c4b7-42bd-94db-91d341f6d613"
    }
  }
 }
}
```

```
import React, { useEffect, useRef } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import * as Notifications from 'expo-notifications';
import { registerForPushNotificationsAsync } from './screens/Notifications';
import LoadingScreen from './screens/Loading';
import Landingpage from './screens/Landingpage';
import LoginScreen from './screens/LoginScreen';
import RegisterScreen from './screens/RegisterScreen';
import SettingsScreen from './screens/SettingsScreen';
import SemiApp from './screens/SemiApp';
import PinandFingerPrint from './screens/PinandFingerprint';
import SignUpAuth from './screens/PinandFingerprintSignUp';
import SearchChat from './screens/SearchChat';
import ChatScreen from './screens/ChatScreen';
import VideoCall from './screens/VideoCallScreen';
import AudioCall from './screens/AudioCallScreen';
import CreateGroupChat from './screens/CreateGroupChatScreen';
import GroupChatScreen from './screens/GroupChatScreen';
import ForgotPassword from './screens/ForgotPassword';
import GroupChats from './screens/GroupChats';
import { useFonts, TitilliumWeb_400Regular, TitilliumWeb_600SemiBold } from '@expo-google-fonts/titillium-web';
import 'react-native-get-random-values';
import { v4 as uuidv4 } from 'uuid';

const Stack = createNativeStackNavigator();
const uuid = uuidv4();

export default function App() {
  const navigationRef = useRef();

  useEffect(() => {
    registerForPushNotificationsAsync();

    const subscription = Notifications.addNotificationResponseReceivedListener(response => {
      const data = response.notification.request.content.data;
      if (data.type === "videocall") {
        navigationRef.current?.navigate('VideoCall', { callerInfo: data.callerInfo, autoStart: true});
      } else if (data.type === "audiocall") {
        navigationRef.current?.navigate('AudioCall', { callerInfo: data.callerInfo, autoStart: true }); // autoStart: true
      } else if (data.screen === 'GroupChatScreen') {
        navigationRef.current?.navigate('GroupChatScreen', {
          groupId: data.groupId,
          groupName: data.groupName,
        });
      }
    });
    return () => subscription.remove();
  }, []);
```

```
  let [fontsLoaded, fontError] = useFonts({
    TitilliumWeb_400Regular,
    TitilliumWeb_600SemiBold,
  });

      if (!fontsLoaded && !fontError) {
    return null;
  }

  return (
          <NavigationContainer ref={navigationRef}>
      <Stack.Navigator
      initialRouteName="Load"
      screenOptions={{
        headerStyle: {
                backgroundColor: '#4c669f',
        },
        headerTintColor: '#fff',
        headerTitleStyle: {
                        fontFamily: 'TitilliumWeb_400Regular',
        }
      }}
      >
                    <Stack.Screen name="Load" component={LoadingScreen} options={{ headerShown: false }} />
                    <Stack.Screen name="Land" component={Landingpage} options={{ headerShown: false }} />
                    <Stack.Screen name="Login" component={LoginScreen} options={{ headerShown: false }} />
                    <Stack.Screen name="ForgotPassword" component={ForgotPassword} options={{ headerShown: false }} />
                    <Stack.Screen name="Register" component={RegisterScreen} options={{ headerShown: false }} />
                    <Stack.Screen name="Settings" component={SettingsScreen} options={{ headerShown: false }} />
                    <Stack.Screen name="SemiApp" component={SemiApp} options={{ headerShown: false }} />
                    <Stack.Screen name="PinandFingerprint" component={PinandFingerPrint} options={{ headerShown: false }} />
                    <Stack.Screen name="SignUpAuth" component={SignUpAuth} options={{ headerShown: false }} />
                    <Stack.Screen name="ChatScreen" component={ChatScreen} options={{ headerShown: true }}/>
                    <Stack.Screen name="CreateGroupChat" component={CreateGroupChat} options={{ title: 'Create Group' }}/>
                    <Stack.Screen name="GroupChatScreen" component={GroupChatScreen} options={{ headerShown: true }}/>
                    <Stack.Screen name="GroupChats" component={GroupChats} options={{ title: 'Group Chats' }}/>
                    <Stack.Screen name="SearchChat" component={SearchChat} options={{ title: 'Search Contacts' }}/>
                    <Stack.Screen name="VideoCall" component={VideoCall} options={{ title: 'Video Call' }}/>
                    <Stack.Screen name="AudioCall" component={AudioCall} options={{ title: 'Audio Call' }}/>
      </Stack.Navigator>
    </NavigationContainer>
  );
}

module.exports = function (api) {
  api.cache(true);
  return {
        presets: ['babel-preset-expo'],
    plugins: [
          ['module:react-native-dotenv', {
        moduleName: '@env',
        path: '.env',
        blacklist: null,
        whitelist: null,
        safe: false,
        allowUndefined: true,
      }],
    ],
  };
};

{
  "cli": {
    "version": ">= 10.0.2",
    "requireCommit": true
  },
  "build": {
    "development": {
      "developmentClient": true,
      "distribution": "internal"
    },
    "preview": {
      "android": {
        "buildType": "apk"
      }
    },
    "preview2": {
      "android": {
                "gradleCommand": ":app:assembleRelease"
      }
    },
    "preview3": {
      "developmentClient": true
    },
    "preview4": {
      "distribution": "internal"
    },
    "production": {}
  }
}
import { initializeApp } from "firebase/app";
```

```
import { initializeAuth, getReactNativePersistence } from "firebase/auth";
import AsyncStorage from "@react-native-async-storage/async-storage";
const firebaseConfig = {
  apiKey: "AIzaSyBLYghEfs0lxgjTN9xLWrqQrD7onIuj39M",
  authDomain: "soc-thesis.firebaseapp.com",
  projectId: "soc-thesis",
  storageBucket: "soc-thesis.appspot.com",
  messagingSenderId: "151656028112",
  appId: "1:151656028112:web:df9c5556490a6309924bd5",
  measurementId: "G-B1ZFNSLQJG"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

initializeAuth(app, {
  persistence: getReactNativePersistence(AsyncStorage),
})

export default app;


{
  "project_info": {
    "project_number": "151656028112",
    "firebase_url": "https://soc-thesis-default-rtdb.firebaseio.com",
    "project_id": "soc-thesis",
    "storage_bucket": "soc-thesis.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:151656028112:android:da09a901c6e7a03c924bd5",
});
      "android_client_info": {
        "package_name": "com.example.SOC"
      }
    },
    "oauth_client": [],
    "api_key": [
      {
        "current_key": "AIzaSyBeW-3PDBqBpi7IlhlSGIu12By6WVbC9YE"
      }
    ],
    "services": {
      "appinvite_service": {
        "other_platform_oauth_client": []
      }
    }
  },
  "configuration_version": "1"
}


{

  }

},

  "node_modules/zod-validation-error": {

    "version": "2.1.0",

    "resolved": "https://registry.npmjs.org/zod-validation-error/-/zod-validation-error-2.1.0.tgz",

    "integrity": "sha512-VJh93e2wb4c3tWtGgTa0OF/dTt/zoPCPzXq4V11ZjxmEAFaPi/Zss1xIZdEB5RD8GD00U0/iVXgqkF77RV7pdQ==",

    "license": "MIT",

    "engines": {

      "node": ">=18.0.0"

    },

    "peerDependencies": {
      "zod": "^3.18.0"

    }

  }

}

const express = require('express');
const { createServer } = require('http');
const socketIo = require('socket.io');

const app = express();
const server = createServer(app);
const io = socketIo(server);

io.on('connection', (socket) => {
  console.log('User connected:', socket.id);

  socket.on('offer', (offer) => {
    socket.broadcast.emit('offer', offer);
  });

  socket.on('answer', (answer) => {
    socket.broadcast.emit('answer', answer);
  });

  socket.on('ice-candidate', (candidate) => {
    socket.broadcast.emit('ice-candidate', candidate);
  });

  socket.on('disconnect', () => {
    console.log('User disconnected:', socket.id);
  });
});

server.listen(3000, () => {
  console.log('Signaling server running on port 3000');
```

Certificate of Grammarian

CERTIFICATION OF ENGLISH CRITIQUE AND GRAMMAR EDITING

January 23, 2025

To whom it may concern,

This is to certify that the researchers with the thesis title

"SAFE-ON-CHAT: A MOBILE MESSAGING AND VIDEO CALL APPLICATION WITH SECURITY FEATURES USING RIVEST SHAMIR ADLEMAN AND ADVANCED ENCRYPTION STANDARD ALGORITHM."

in partial fulfillment of the requirements of

Amparo, Paul Victor M.
Batarina, Randel B.
Pariado, Keem R.
Rodriguez, Clark F.

Has been EDITED for language consistency and grammar coherence. The manuscript has been found to be thorough and acceptable concerning spelling, grammar, composition, and overall style.

Neither the manuscript's contents nor the author's intention were altered in any way during the process of editing and proofreading.

AYUMI ASHLEY S. DELA CRUZ, LPT
Bachelor of Secondary Education Major in English
PRC License. No. 1991159

09128735320
ayumiashley0@gmail.com

Plagiarism Report

Turnitin Originality Report

Document Viewer

Processed on: 23-Jan-2025 09:11 EAT
ID: 2569614810
Word Count: 12019
Submitted: 3

THS-101_41_SOC_WGRAMMARIAN&PLAGIARISMCHECKER By Keem Pariado

Similarity Index	Similarity by Source	
5%	Internet Sources:	5%
	Publications:	3%
	Student Papers:	3%

turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Keem Pariado
Assignment title:	ASSIGNMENT 96
Submission title:	THS-101_41_SOC_WGRAMMARIAN&PLAGIARISMCHECKER
File name:	THS-101_41_SOC_WGRAMMARIAN.docx
File size:	5.44M
Page count:	111
Word count:	12,019
Character count:	68,485
Submission date:	23-Jan-2025 09:10AM (UTC+0300)
Submission ID:	2569614810

TAGUIG CITY UNIVERSITY

SAFE-co-CHAT: A Mobile Messaging and Video Call Application with Security Features using Rivest Shamir Adleman and Advanced Encryption Standard Algorithm

A Thesis Presented to the Faculty of the Computer Science Taguig City University, General Santos Avenue, Central Bicutan, Taguig City

In partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science

By:

Amparo, Paul Victor M.
Batarina, Randel B.
Pariado, Keem R.
Rodriguez, Clark F.

June 2025

Curriculum Vitae

PAUL VICTOR M. AMPARO

09150881057

Herrera Compound Santa Anna Taguig City

paulvictoramparo04@gmail.com

Objective

Seeking a job that aligns with my present skill set, where I can utilize my abilities and knowledge, gain hands-on experience, and contribute to a dynamic team. I am ready to learn, improve, and provide excellent technical help.

Skills

- Computer Literate
- Proficient in MS Office
- Computer Troubleshooting
- Basic Knowledge in Canva
- Basic Knowledge in Wix
- Basic Knowledge in Switch Configuration (Huawei)
- CCTV Setup & Configuration(HikVision)
- Basic Knowledge in Photoshop

References

Jericho Rosario
IT Team Lead/IT Specialist
Phone: 09184547866
Email: jerrosario1@gmail.com

Jayce Polintan
IT Team Lead/IT Specialist
Phone: 09327986323
Email: polintan.jayce9@gmail.com

Experience

IT SUPPORT INTERN — February 2024 - May 2024
Makati Development Corporation

ENCODER (WORK IMMERSION) — October 2019
Medical Center Taguig

Education

Bachelor of Science in Computer Science — 2020 - 2025
Taguig City University

Information Communication and Technology — 2018 - 2020
Gen. Ricardo G. Papa Sr. Memorial High School

Junior High School — 2014 - 2018
Gen. Ricardo G. Papa Sr. Memorial High School

Achievements

Dean's Lister — 2021 - 2022
Taguig City University

National Certificate (NCII) — 2020
Computer System Servicing (CSS)

Top 3 and Complete Computer Literacy Program — 2015
St. Therese School of Technology

Keem R. Pariado

OBJECTIVE

To seek an opportunity that will utilize my skills and ability to work effectively with others, gain more experience, and expand my knowledge in which will allow me to continuously develop both personally and professionally. With the dedication to contributing to the success of the organization while further developing my abilities.

CONTACT

pariadokeem0@gmail.com
09104217001
#4B Ballecer St., Brgy. Central
Signal Village, Taguig City, Metro
Manila 1630

EDUCATION

TAGUIG CITY UNIVERSITY
Bachelor of Science in Computer Science

GENERAL RICARDO PAPA SR. MEMORIAL HIGHSCHOOL
TechVoc ICT Graduate
With Honors, Best in Research

GENERAL RICARDO PAPA SR. MEMORIAL HIGHSCHOOL
Junior High School

CERTIFICATION

Dean's Lister (2022-2023) & (2021-2022)
Electrical Installation and Maintenance NC II

CHARACTER REFERENCES

Elton John P. Legion
Logistics Analyst - SRT
09560873240

Mar B. Ochea
Customer Service Representative II
09851061986

WORK EXPERIENCE

DATA ENCODER CRUZ-RABE MATERNITY AND GENERAL HOSPITAL
October 2019 - November 2019

Encodes patients' health status with accuracy while also assisting them to fill out their papers correctly and carry out any tasks given.

SERVICE CREW AT MCDONALD'S 861 DRAGONFISH RESTAURANT (PUREGOLD TUKTUKAN)
July 2023 – January 2025

Taking customers' orders, handling cash transactions and reports, and working in a fast-paced environment.
Proficiency in delivering products quickly and precisely. Ability to maintain good relationship towards customers and couriers.

IT ANALYST INTERN PAMANA PROPERTIES CORPORATION
March 2024 - May 2024

Developed an interactive and responsive website for the company.

SKILLS

Proficient in using MS Office tools with basic knowledge of computer hardware.
Basic programming skills in HTML, CSS and Java
Capable of using WIX and WordPress
Knowledgeable in Figma, Canva, and Inkscape
Able to learn quickly about POS system.
Effective communication. Involves resolving customers concerns with a positive attitude. Can work autonomously. Adaptability. High attention to detail.

About Me

I am passionate about learning, growing professionally, and applying my skills to real-world challenges. I'm open to exploring new opportunities to broaden my expertise and contribute meaningfully.

Contact

09705195641
rodriguez.clark011@gmail.com
Rodriguez Comp. Sta Ana
Taguig City

Skills

- Computer Network
- Computer Literate
- Good Communication Skill
- Basic Computer Trouble Shooting

Language

- Filipino
- Englsih

CAHARACER REFERENCE

Julius Rodland De Luna
09122103259
Service Engineer

Ivy Charry F. Opeña
09092240667
SHS Teacher

Clark Rodriguez

Education

(Present)
TAGUIG CITY UNIVERSITY
BS Computer Science
2nd place Networking Competition
CICT College Days 2023

(2014 -2020)
GEN. RICARDO PAPA SR. MEMORIAL HIGHSCHOOL
CSS NC2 Holder

(2008 -2014)
TAGUIG INTEGRATED SCHOOL

Experience

(2019)
ENCODER
CRUZ RABE HOSPITAL

- Encoder of patient information and health status Carries out any task given by the superiors

Clark F. Rodriguez
Applicant's Signature

'I hereby certify that the above information is true and correct to the best of my knowledge'

RANDEL B. BATARINA

CONTACT

+63-961-561-2830

randelbbatarina@gmail.com

EDUCATION

2020-PRESENT
TAGUIG CITY UNIVERSITY
- Bachelor of Science in Computer Science

2018-2020
SRCCMSTHS
- K-12 / ICT-CSS

2014-2018
PDMHS
- Junior High School

2008-2014
KERMES
- Elementary

TECHNICAL SKILLS

- Networking Troubleshooting
- Hardware Troubleshooting
- Graphic Design
- Cyber Security
- Back-End Programming
- Front-End Programming

LANGUAGES

- Python
- JavaScript
- HTML/CSS
- C++

PROFILE

As a graduating student eager to embark on my professional journey, I bring a strong commitment to achieving results and a readiness to learn and adapt. While I am new to the field, my academic experiences have fostered skills in teamwork, problem-solving, and effective communication. I am enthusiastic about the opportunity to be trained and to contribute positively to your team. I am confident that my dedication and willingness to learn will enable me to quickly gain the necessary skills and make a meaningful impact in your organization.

WORK EXPERIENCE

- **Harte Hanks Manila** — 2024-PRESENT
  Technical Support Representative
- **Scientific Standard Services Inc.** — 2024
  Encoder (OJT - 500 hours)
- **Western Bicutan Barangay Office** — 2020
  Encoder (Work Immersion - 96 hours)

LEADERSHIP EXPERIENCE

- **Supreme Student Council** — 2023-2024
  Board of Directors
- **ICT-Student Federation** — 2023-2024
  Student Director
- **Computer Science Society** — 2022-2023
  President
- **Information Technology Society** — 2021-2022
  Committee

REFERENCE

Nicole Aramil
Phone: +63-961-703-4231
Email :aramilnicole999@gmail.com

Cherrylyn M. Tatoy
Phone: +63-961-702-2526
Email : tatoycherrylyn12@gmail.com

Certificate of Enrollment

Republic of the Philippines
City of Taguig

TAGUIG CITY UNIVERSITY

CERTIFICATE OF ENROLLMENT
ACADEMIC YEAR: 2023 - 2024
FIRST SEMESTER

picture

Student No: 20-00994
Name: AMPARO, PAUL VICTOR MAGNO
Year Level/ Section: 4th / BSCS B2020 / Regular
Course: Bachelor of Science in Computer Science (BSCS)
College: College of Information and Communication Technology (CICT)

ENROLLED SUBJECT(S)

NO.	SUBJECT CODE	SUBJECT DESCRIPTION	UNITS	LEC / TIME; LAB / TIME	ROOM
1	THS 102	CS Thesis Writing 2	6	Tu Th /2:30:PM-5:30:PM	419
2	ELEC 5	System Fundamentals	3	M W /7:30:AM-9:30:AM	LAB 110
3	HCI 102	Technopreneurship/ E-Commerce	6	F /2:00:PM-8:00:PM	418
4	ELEC 4	Graphics and Visual Computing	3	M W /9:30:AM-11:30:AM	LAB 110
5	IT102	Social Media and Presentation	3	Tu Th /1:00:PM-2:30:PM	419
		TOTAL	21		

VALIDATED
BY:
DATE: 2023-08-03

ENROLLED
BY:
DATE: 2023-08-03

Approved by:
FRANCISCO C. BARRAMEDA, JR.
UNIVERSITY REGISTRAR
Note: This signature is system generated.

=>> Student's Copy

Republic of the Philippines
City of Taguig

TAGUIG CITY UNIVERSITY

CERTIFICATE OF ENROLLMENT
ACADEMIC YEAR: 2023 - 2024
FIRST SEMESTER

picture

Student No: **20-01015**
Name: **PARIADO, KEEM RUELAN**
Year Level/ Section: **4th / BSCS B2020 / Regular**
Course: **Bachelor of Science in Computer Science (BSCS)**
College : **College of Information and Communication Technology (CICT)**

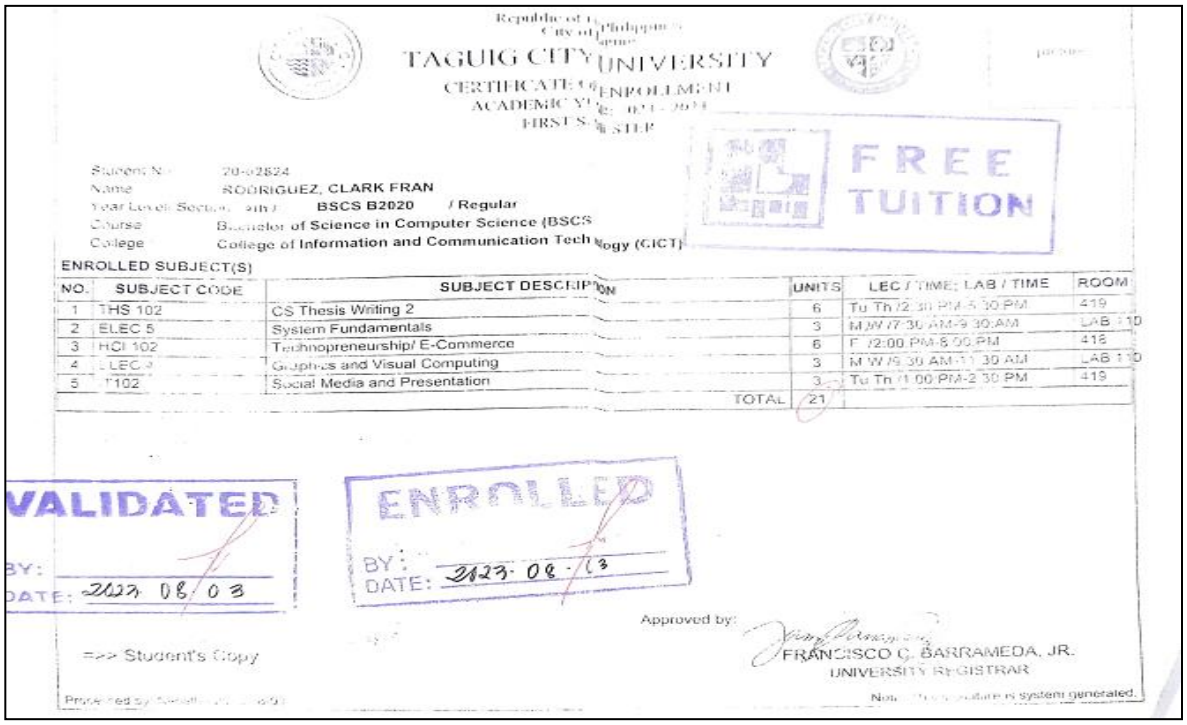
ENROLLED SUBJECT(S)

NO.	SUBJECT CODE	SUBJECT DESCRIPTION	UNITS	LEC / TIME; LAB / TIME	ROOM
1	THS 102	CS Thesis Writing 2	6	Tu Th /2:30:PM-5:30:PM	419
2	ELEC 5	System Fundamentals	3	M W /7:30:AM-9:30:AM	LAB 110
3	HCI 102	Technopreneurship/ E-Commerce	6	F /2:00:PM-8:00:PM	418
4	ELEC 4	Graphics and Visual Computing	3	M W /9:30:AM-11:30:AM	LAB 110
5	IT102	Social Media and Presentation	3	Tu Th /1:00:PM-2:30:PM	419
		TOTAL	21		

Approved by:
FRANCISCO C. BARRAMEDA, JR.
UNIVERSITY REGISTRAR
Note: This signature is system generated.

=>> Registrar's Copy

Processed by: Neneth - 2023-08-03



Republic of the Philippines
City of Taguig
TAGUIG CITY UNIVERSITY
CERTIFICATE OF ENROLLMENT
ACADEMIC YEAR 2023 - 2024
FIRST SEMESTER

FREE TUITION

Student No: 20-02824
Name: RODRIGUEZ, CLARK FRAN
Year Level/ Section: 4th / BSCS B2020 / Regular
Course: Bachelor of Science in Computer Science (BSCS
College: College of Information and Communication Technology (CICT)

ENROLLED SUBJECT(S)

NO.	SUBJECT CODE	SUBJECT DESCRIPTION	UNITS	LEC / TIME; LAB / TIME	ROOM
1	THS 102	CS Thesis Writing 2	6	Tu Th /2:30 PM-5:30 PM	419
2	ELEC 5	System Fundamentals	3	M W /7:30 AM-9:30 AM	LAB 110
3	HCI 102	Technopreneurship/ E-Commerce	6	F /2:00 PM-8:00 PM	418
4	ELEC 4	Graphics and Visual Computing	3	M W /9:30 AM-11:30 AM	LAB 110
5	IT102	Social Media and Presentation	3	Tu Th /1:00 PM-2:30 PM	419
		TOTAL	21		

VALIDATED
BY:
DATE: 2023 08 03

ENROLLED
BY:
DATE: 2023 08 03

Approved by:
FRANCISCO C. BARRAMEDA, JR.
UNIVERSITY REGISTRAR

==> Student's Copy

Processed by: [illegible]

Note: This document is system generated.

Republic of the Philippines
City of Taguig
TAGUIG CITY UNIVERSITY
CERTIFICATE OF ENROLLMENT
ACADEMIC YEAR 2023 - 2024
FIRST SEMESTER

picture

Student No: 20-00940
Name: BATARINA, RANDEL BALDERAMA
Year Level/ Section: 4th / BSCS B2020 / Regular
Course: Bachelor of Science in Computer Science (BSCS)
College: College of Information and Communication Technology (CICT)

ENROLLED SUBJECT(S)

NO.	SUBJECT CODE	SUBJECT DESCRIPTION	UNITS	LEC / TIME; LAB / TIME	ROOM
1	THS 102	CS Thesis Writing 2	6	Tu Th /2:30:PM-5:30:PM	419
2	ELEC 5	System Fundamentals	3	M W /7:30:AM-9:30:AM	LAB 110
3	HCI 102	Technopreneurship/ E-Commerce	6	F /2:00:PM-8:00:PM	418
4	ELEC 4	Graphics and Visual Computing	3	M W /9:30:AM-11:30:AM	LAB 110
5	IT102	Social Media and Presentation	3	Tu Th /1:00:PM-2:30:PM	419
		TOTAL	21		