

**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

**Samriddhi College**

**SUPERVISOR’S RECOMMENDATION**

I hereby recommend that this project prepared under my supervision by “**Sujal Munikar** and **Sandesh Bata**” entitled " **Chatline** " in partial fulfillment of the requirements for the degree of Bachelor of Computer Application is recommended for the final evaluation.

**……………………….**

**SIGNATURE of the Supervisor**

**Abhishek Dewan**

**SUPERVISOR**

**BCA (Humanities and Social Science)**

**Samriddhi College**

**Lokanthali, Bhaktapur**



**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

**Samriddhi College**

**LETTER OF APPROVAL**

This is to certify that this project prepared by ‘’**Sujal Munikar** and **Sandesh Bata**” entitled "**Chatline**" in partial fulfillment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

|  |  |
| --- | --- |
| **Signature of Supervisor**    **……………………….**  **Mr. Abhishek Dewan**  **Supervisor**  **Samriddhi College**  **Lokanthali, Bhaktapur** | **Signature of HOD/ Coordinator**    **……………………….**  **Mr. Jeeban Dhungel**  **HOD/ Coordinator**  **Samriddhi College**  **Lokanthali, Bhaktapur** |
| **……………………….**  **Signature of Internal Examiner**  **Internal Examiner** | **……………………….**  **Signature of External Examiner**  **External Examiner** |

# 

# ACKNOWLEDGEMENT

We would like to thank our supervisor **Mr. Abhishek Dewan** for his valuable guidance who advised us throughout all the phases of this project and gave us great insights for this work and helpful suggestions in making our Chatline-A Real Time Chat App project. The contributions from our friends were also significant and deeply appreciated. The opportunity to work on this project under their mentorship has been a rewarding experience.

We would like to acknowledge our friends at our college for their help and support throughout the duration of this project. The supportive learning environment provided by **Samriddhi College** has played a significant role in our learning journey and personal growth. We are grateful for the opportunities provided to us to enhance our skills and make use of proper knowledge to complete this project.

With Regards,

Sujal Munikar

Sandesh Bata

# ABSTRACT

Chatline is a real-time chat application developed using modern web technologies, including React, Tailwind, Node.js, Socket.io, and MySQL. The application is designed to provide an intuitive, responsive, and secure communication platform for users. It leverages React for a dynamic front-end, while Tailwind CSS ensures a customizable and visually appealing design. The backend is powered by Node.js and Socket.io, enabling real-time, bi-directional communication. MySQL handles efficient database interactions, ensuring data consistency.

To safeguard user data, the application employs the RSA algorithm for secure message encryption and decryption. Chatline is built to be cross-device compatible, ensuring users remain connected wherever they are. With features such as encrypted data transmission, secure authentication, and scalable architecture, the application guarantees privacy and reliability. The development process involved comprehensive testing and user interactive design, ensuring a high-performance product that meets modern user expectations. Future plans include the integration of media sharing, group chats, and message history management, positioning. Chatline as a versatile tool for both personal and professional communication.

***Keywords:*** *Real-time chat, React, Tailwind, Node.js, Socket.io, MySql, RSA Encryption*

## Table of Contents

**SUPERVISOR RECOMMENDATION i**

**LETTER OF APPROVAL ii**

**ACKNOWLEDGEMENT iii**

**ABSTRACT iv**

**TABLE OF FIGURES vii**

**TABLE OF TABLES viii**

**LIST OF ABBREVIATION ix**

**CHAPTER 1: INTRODUCTION 1**

1. [Introduction 1](#_bookmark0)
2. [Problem Statement 2](#_bookmark1)
3. [Objectives 2](#_bookmark2)
4. Scope and Limitation 3
5. Scope 3
6. Limitation 3
7. Development Methodology 4
8. Report Organization 5

**CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW 6**

1. Background Study 6
2. Literature Review 7

**CHAPTER 3: SYSTEM ANALYSIS AND DESIGN 8**

1. System Analysis 8
2. Requirement Analysis 8
3. Functional Requirements 8
4. Non-Functional Requirements 9
5. Feasibility Analysis 10
6. Economic Feasibility 10
7. Technical Feasibility 10
8. Operational Feasibility 10
9. Schedule Feasibility 10
10. Process Modelling (Structured Approach) 11
11. DFD (Data Flow Diagram) 11
12. ER Diagram 13
13. Flowchart 14
14. System Design 12
    1. Database Schema Design 15
15. Interface Design 16

**CHAPTER 4: IMPLEMENTATION AND TESTING 18**

1. Implementation 18
2. Tools Used 18
3. Implementation Details of Modules 18
4. Testing 23
5. Unit Testing 23
6. System Testing 25

**CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATION 29**

1. Conclusion 29
2. Future Recommendation 29

**REFERENCES 30**

**APPENDICES 31**

**LIST OF FIGURES**

**Figure 1.1 Agile Methodology [1] 4**

**Figure 3.1 Use Case Diagram of Chatline 9**

**Figure 3.2 Gantt Chart 10**

**Figure 3.3 DFD level 0 of Chatline 11**

**Figure 3.4 DFD level 1 of Chatline 12**

**Figure 3.5 ER Diagram of Chatline 13**

**Figure 3.6 Flowchart of Chatline 14**

**Figure 3.7 Database Schema Design of Chatline 15**

**Figure 3.8 Interface for Login Page 16**

**Figure 3.9 Interface for Chat Page 16**

**Figure 3.10 Interface for User Profile Page 17**

**Figure 3.11 Interface for Friends Page 17**

**Figure 4.1 Working of RSA Algorithm [5] 19**

**LIST OF TABLES**

**Table 4.1 User Login/Signup 23**

**Table 4.2 Bad Word Filtering 24**

**Table 4.3 RSA Encryption in Chat 24**

**Table 4.4 Socket.io Connection 25**

**Table 4.5 USER Registration and Login 25**

**Table 4.6 Chat Functionality 26**

**Table 4.7 Encryption and Decryption 26**

**Table 4.8 Socket.io Connectivity 27**

**Table 4.9 Username and Password Update 27**

**Table 4.10 Friend Request and Unfollow User 28**

**Table 4.11 System Non-Functional Testing 28**

**LIST OF ABBREVIATIONS**

**CSS** Cascading Style Sheet

**DFD** Data Flow Diagram

**ER** Entity Relationship

**JS** JavaScript

**RSA**  Rivest, Shamir and Adleman

**MYSQL** My Structured Query Language

**PHP** Hypertext Preprocessor

**WAMP** Windows, Apache, MySQL, and PHP

**ER Diagram** Entity Relationship Diagram

# CHAPTER I: INTRODUCTION

# Introduction

In today’s digital age, the security of online communication is paramount. Despite advancements in cybersecurity, no system can claim to be completely impervious to breaches. As cyber-attacks become increasingly sophisticated, it is crucial not only to secure user passwords through hashing but also to ensure that the messages exchanged between users are encrypted. Encryption plays a vital role in safeguarding sensitive information, ensuring that even if an attacker gains unauthorized access to a user’s account, they cannot decipher the contents of the communication.

Our project addresses these concerns by developing a real-time chat application that integrates RSA encryption to protect message exchanges. RSA, a widely recognized encryption algorithm, is employed to secure communication channels, thereby enhancing the confidentiality and integrity of user messages. This approach ensures that messages remain protected from potential breaches, mitigating risks associated with unauthorized data access.

Additionally, the application employs Socket.io to provide real-time messaging without the need for constant page refreshes or polling, enhancing user experience and efficiency. This technology ensures that users receive messages in real-time, enhancing the overall efficiency and responsiveness of the chat application.

# Problem Statement

Many existing chat applications still do not use strong encryption methods, which makes user messages vulnerable during transmission. Without proper security, attackers can intercept conversations, access sensitive information, or misuse personal data. Outdated or weak encryption systems increase these risks and reduce user trust in the platform. As user expectations for privacy and data security continue to rise, the failure to implement robust encryption mechanisms not only compromises user trust but also places these applications at a competitive disadvantage, if data privacy for users is not compromised as user expectation.

Our project aims to solve these issues by implementing secure message encryption and building an efficient real-time communication system that offers users both safety and smooth interaction.

# Objectives

The primary objective of A Real-time Chat Application is:

* To design and implement a responsive web-based chat application that allows users to send and receive messages instantly using Socket.io.
* To ensure user privacy and data protection by implementing secure login, encrypted password and secure messages with RSA encryption to ensure the confidentiality of conversations.
* To build a clean, intuitive, and modern chat interface that allows smooth communication.
  1. **Scope and Limitations**

Each website has its own features and limitations. The Chatline- Chat Application offers the following scopes and limitations.

* + 1. **Scope:**

The Chat Application is used to give access to following user using user information and verification and storing the data in a secure database for accessing the resources. It offers following scope given below:

* It provides instant real-time messaging using Socket.IO, including features like real-time presence updates.
* It ensures secure communication with end-to-end RSA encryption, secure key exchange mechanism, and protection of sensitive data such as chat histories.
* It enhances user experience and promotes a healthy communication environment by incorporating text filtering to detect and block toxic or inappropriate words in messages.
* It helps create connections by allowing users to send and accept friend requests, enabling them to build a trusted network for real-time conversations.

1. **Limitation:**

* User verification is limited to email-based OTP verification, with no support for mobile number verification access or reassessing the account if password is forgotten is not possible.
* The platform does not support sending images, videos, or audio files, restricting communication to text-only messages.
* Large conversational message are not supported, allowing only short conversational text messages to be sent.
* There are no indicator for knowing new message or requests being sent and user can only see if they are online or offline not when they were last present.

1. **Development Methodology**

The Agile methodology is applied throughout the project to facilitate flexibility and responsiveness to change. Agile focuses on iterative development, where the project is broken down into smaller, manageable units of work called sprints, typically lasting one to two weeks. Each sprint will result in a functional piece of the application that can be tested, reviewed, and refined based on user feedback.

**Figure 1.1: Agile Methodology [1]**

This iterative process allows for continuous improvements and ensures that the development stays aligned with user needs and project goals. Regular sprint reviews and retrospectives will provide opportunities for the team to assess progress, identify any challenges, and make necessary adjustments. Agile emphasis on collaboration and communication among team members and stakeholders ensures that the project remains on track and any issues are addressed promptly.

In this project, Agile enables the rapid incorporation of changes, particularly as the development involves integrating complex features like real-time messaging and encryption. By following Agile, the project team ensures that the final product meets user expectations while delivering high-quality software that is both functional and secure.

## Report Organization

The project is broken up into six isolated chapters, each of which describes a different stage of development. The following is a summary of the chapters:

**Chapter 1** is the overview of the project including the introduction, problem statement, objectives, scope, limitations, and development methodology.

**Chapter 2** focuses on the background study and literature review related to the project for reference in project development.

**Chapter 3** involves system analysis, including requirement analysis and feasibility analysis.

**Chapter 4** includes the project design and modeling.

**Chapter 5** focuses on the implementation and testing of the system.

**Chapter 6** is the conclusion of the report and includes future recommendations.

Furthermore, the reference section includes a list of all the sources cited in the report.

**CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW**

1. **Background Study**

Chatline is built on several modern technologies that together provide real-time communication, security, and a smooth user experience. For real-time messaging, the application uses Socket.io, which supports fast, two-way communication between clients and the server. To protect user data during transmission, RSA encryption is used so that messages can only be read by authorized parties. The system also includes text filtering to detect and block toxic or inappropriate content, helping maintain a safe environment for users. In addition, Chatline supports basic social networking features like sending and accepting friend requests, allowing users to build connections within the platform.

From a technological perspective, Chatline uses a modern full-stack approach for both the frontend and back-end. The front-end is built with React.js, delivering a dynamic and interactive user interface, styled efficiently using TailwindCSS. State management is streamlined with Redux Toolkit ensuring efficient data flow and predictable application state. The back-end, powered by Node.js, which provides a scalable and high-performance server infrastructure. MySQL is used to store data as it simplifies database interactions and ensures consistency in managing relational data. This blend of advanced technologies and fundamental concepts forms the backbone of Chatline, ensuring a secure, scalable, and user-friendly platform that meets the demands of a modern real-time communication application.

1. **Literature Review**

The Applied Cryptography: Protocols, Algorithms, and Source Code in C by Bruce Schneier is a foundational text that introduces cryptography, covering both theoretical concepts and practical implementations. The book details various cryptographic algorithms, including symmetric and public-key encryption, hash functions, and cryptographic protocols. It also includes C source code for many algorithms, making it a practical guide for developers. Schneier emphasizes the correct application of cryptography in real-world scenarios, discussing potential vulnerabilities and security considerations.[3]

Enhancing Real-Time Messaging Applications Using WebSocket and Socket.io is the journal posted in Journal of Web Engineering and Technology. This article examines the use of WebSocket and Socket.io to optimize the functionality of real-time messaging applications. It explains how WebSocket provides a full-duplex communication channel that allows for persistent connections between clients and servers, significantly reducing latency and improving message delivery speed. The article further explores how Socket.io, an abstraction layer over WebSocket, simplifies the development process by offering built-in features like automatic reconnection, broadcasting, and room-based communication, which are essential for creating interactive and scalable messaging systems.[4]

Secure and Efficient End-to-End Encrypted Messaging Framework for Real-Time Communication is the journal posted in IEEE Transactions on Information Forensics and Security. This article presents a framework designed to ensure both security and efficiency in real-time messaging systems. The study focuses on implementing end-to-end encryption to protect messages from unauthorized access, ensuring that only the communicating parties can decrypt the content. The proposed framework also addresses the challenge of maintaining low latency and high performance in encrypted communication by optimizing encryption algorithms and key management processes. The findings suggest that the framework successfully balances strong security with the need for real-time responsiveness, making it suitable for applications requiring secure and fast communication, such as in healthcare, finance, and private messaging services.[5]

**CHAPTER 3: SYSTEM ANALYSIS AND DESIGN**

## System Analysis

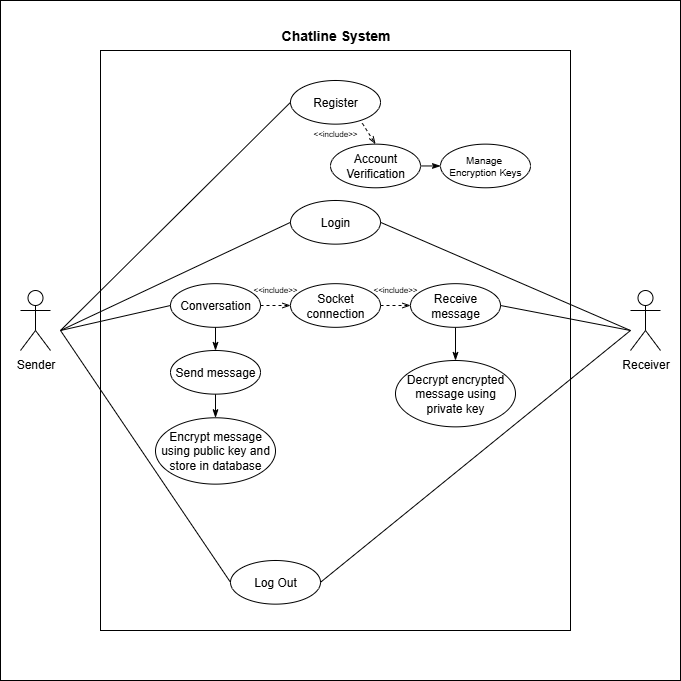
## The system analysis for the proposed project is a critical step in ensuring the project's success. This analysis involves collecting and interpreting facts, identifying problems, and decomposing the system into its components to achieve the project's objectives. Agile methodology is selected for this project because it supports iterative development and continuous feedback. Instead of strict sequential phases, the system is developed in short cycles, where each cycle enhances and refines the previous outputs.

1. **Requirement Analysis**

Requirement analysis serves as the foundation for guiding the design, development, and implementation of solutions to meet those requirements effectively. It is done while developing a system and before implementing, it is necessary to analyze the whole system requirement.

1. **Functional Requirements**

* The system must allow users to register and log in securely using an email and password.
* The application must support real-time messaging between users, ensuring instant delivery and receipt of messages.
* All messages must be encrypted using RSA encryption to ensure secure communication between users.
* The system should store chat histories for users, allowing them to view previous messages.



**Figure 3.1: Use Case Diagram of Chatline**

1. **Non-Functional Requirements**

* The application must be able to handle real-time message delivery with minimal latency, ensuring a smooth user experience.
* The application should support a light and dark mode switcher, allowing users to toggle between themes based on their preference.

1. **Feasibility Analysis**
2. **Economic Feasibility**

All required resources and tools are already available, so no additional cost is needed to complete the project.

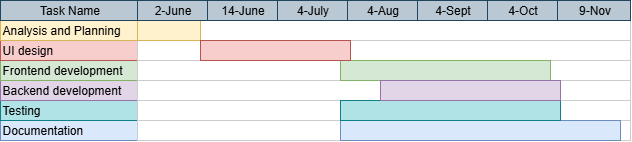
1. **Technical Feasibility**

The technologies required for development, such as Socket.io, RSA encryption, React, Node.js, and MySQL, are available and can be implemented successfully in the project.

1. **Operational Feasibility**

The project aligns with user requirements and can be smoothly integrated into existing systems for real-time communication.

1. **Schedule Feasibility**

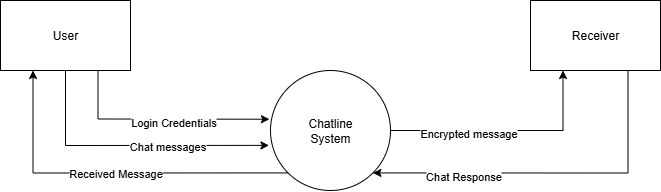
The project can be completed within the planned time frame as outlined in the Gantt chart.

**Figure 3.2: Gantt Chart (Working Schedule of Chatline)**

## Process Modelling (Structured Approach)

### DFD (Data Flow Diagram)

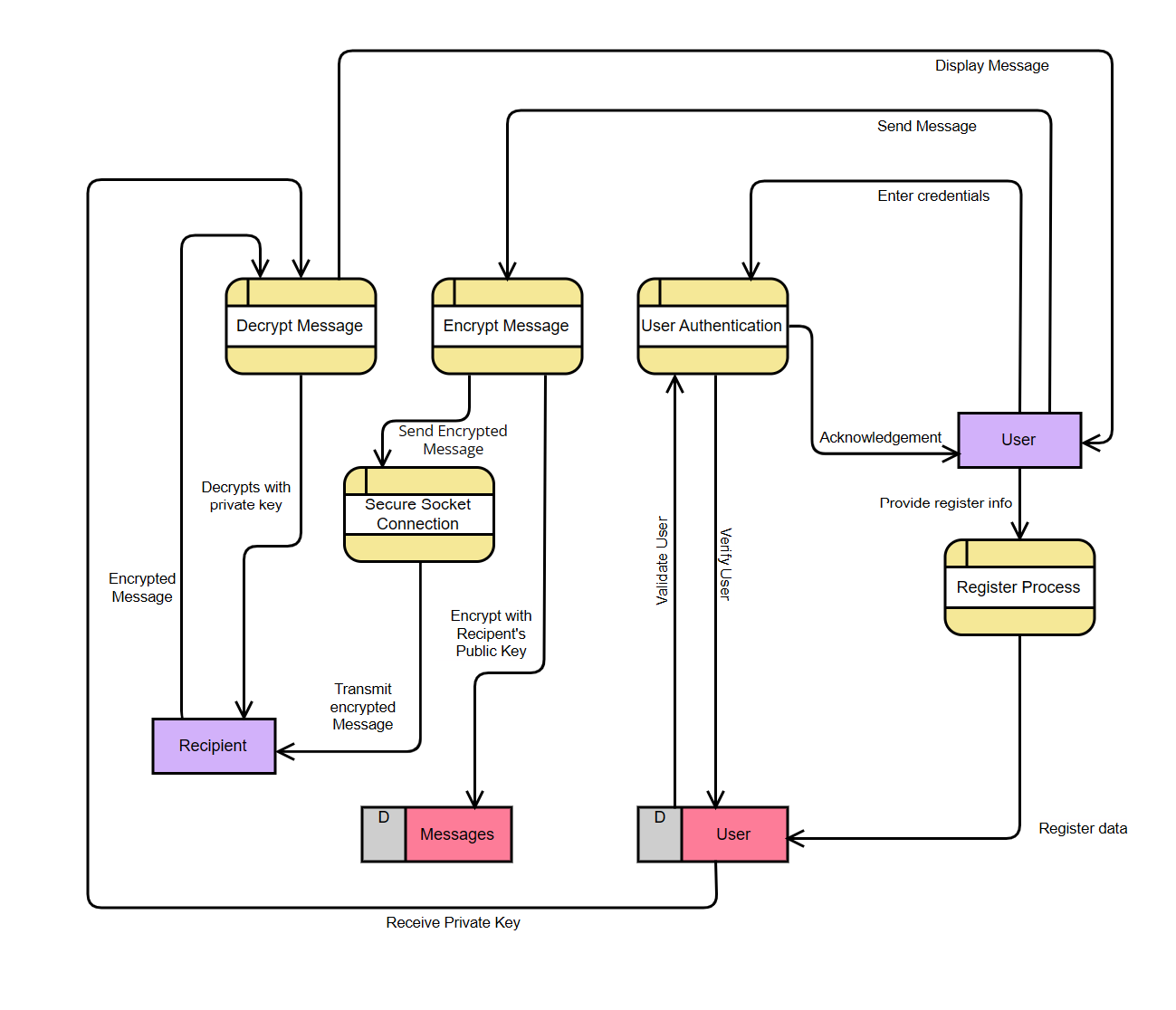
1. **Level-0 DFD**



**Figure 3.3: DFD level 0 of Chatline**

The Level 0 Data Flow Diagram (DFD) for the Chatline system provides an overview of the key processes and data flows in the system. It illustrates how users interact with the Chatline system by sending login credentials for authentication. Once authenticated, users can exchange chat messages, which are processed by the system and securely transmitted to recipients as encrypted messages. The system ensures that responses from recipients are received, decrypted, and delivered back to the original sender. The diagram emphasizes secure communication through encrypted messages and seamless handling of chat interactions, ensuring privacy and efficient messaging between users.

1. **Level-1 DFD**



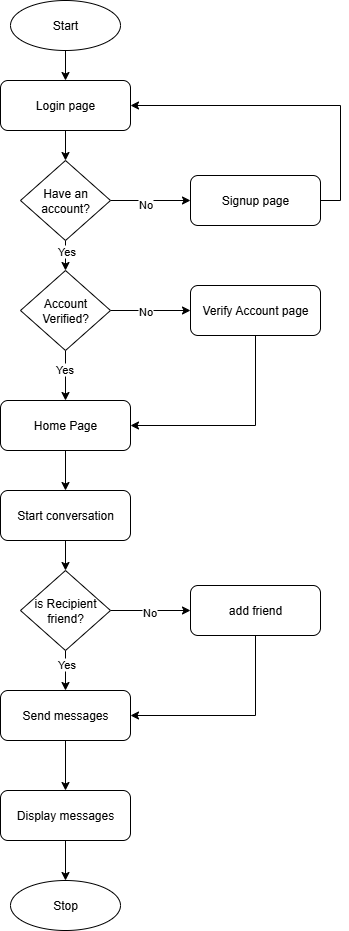
**Figure 3.4: DFD level 1 of Chatline**

The Level 1 Data Flow Diagram (DFD) for the Chatline system elaborates on the processes involved in secure communication. Initially, the user enters their credentials into the Authentication System, which verifies the information with the Authentication Server. Once validated, the system acknowledges the user and grants access. The user then sends a message, which is passed to the Encryption Module. This module encrypts the message using the recipient's public key to ensure security. The encrypted message is transmitted through a Secure Socket Connection to the recipient. Upon receipt, the Decryption Module decrypts the message using the recipient's private key, ensuring that only the intended recipient can read it. The decrypted message is then displayed to the recipient, completing the secure communication flow. This diagram emphasizes secure user authentication, message encryption, and decryption to safeguard data integrity and privacy.

### ER Diagram

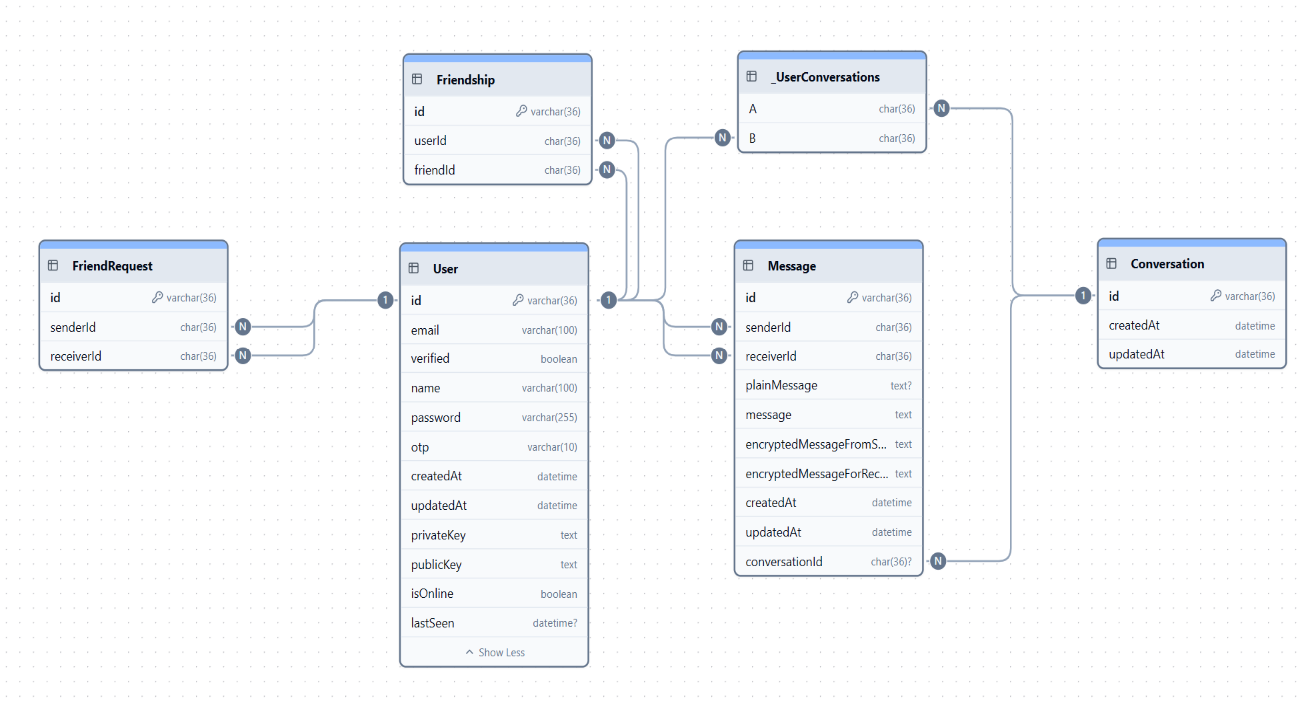
**Figure 3.5: ER diagram of Chatline**

The ER diagram for Chatline models a real-time chat application's database structure. The User entity stores essential details like id, name, email, password, and encryption keys (privateKey, publicKey). Users can send and receive Messages, which are encrypted and linked to Conversations. Users also engage in Friendships (M:M relationship) and can send Friend Requests. The Conversation entity organizes messages and tracks createdAt and updatedAt timestamps. Overall, the diagram captures the relationships between users, messages, conversations, and friendships to enable secure and efficient real-time communication.

1. **Flow Chart**
2. 

**Figure 3.6: Flowchart of Chatline**

The flowchart of Chatline outlines the user journey from starting the application to accessing the chat page. It begins at the Start point, where users are directed to the Login Page. If the user does not have an account, they are redirected to the Signup Page to create one. For existing users, the system checks if the account is verified. If the account is not verified, the user is directed to the Verify Account Page to complete the verification process. Once the account is verified, the user can access the Chatline Chat Page to begin communication. After this, user can send request and add friends to send message only to friends. The user who are friends can send encrypted messages and view them in chat window. The process ends at the Stop point. This flowchart ensures a logical sequence for user authentication, registration, and verification to access the chat functionalities.

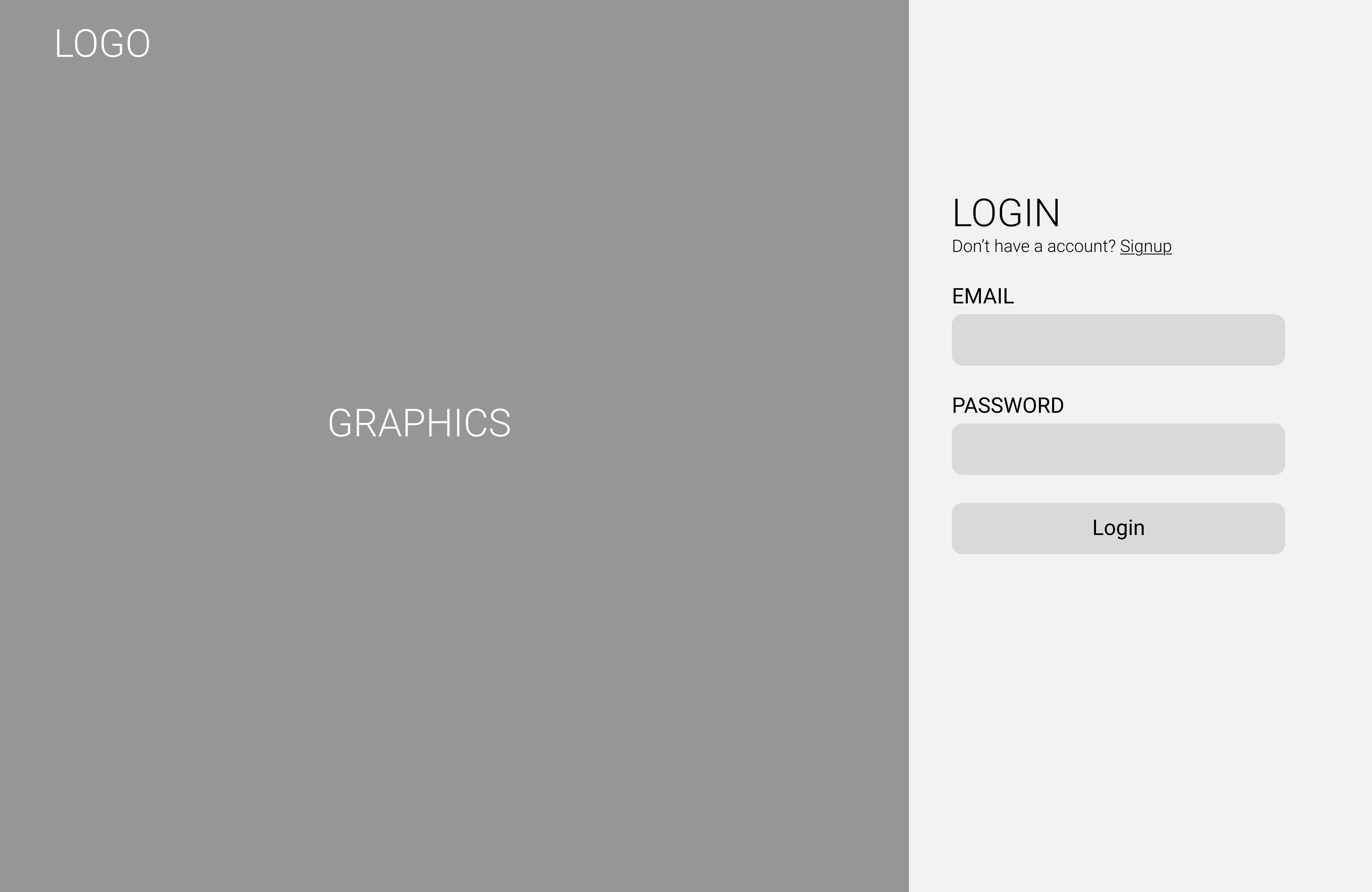
1. **System Design**
2. **Database Schema Design**

**Figure 3.7: Database Schema Design for Chatline**

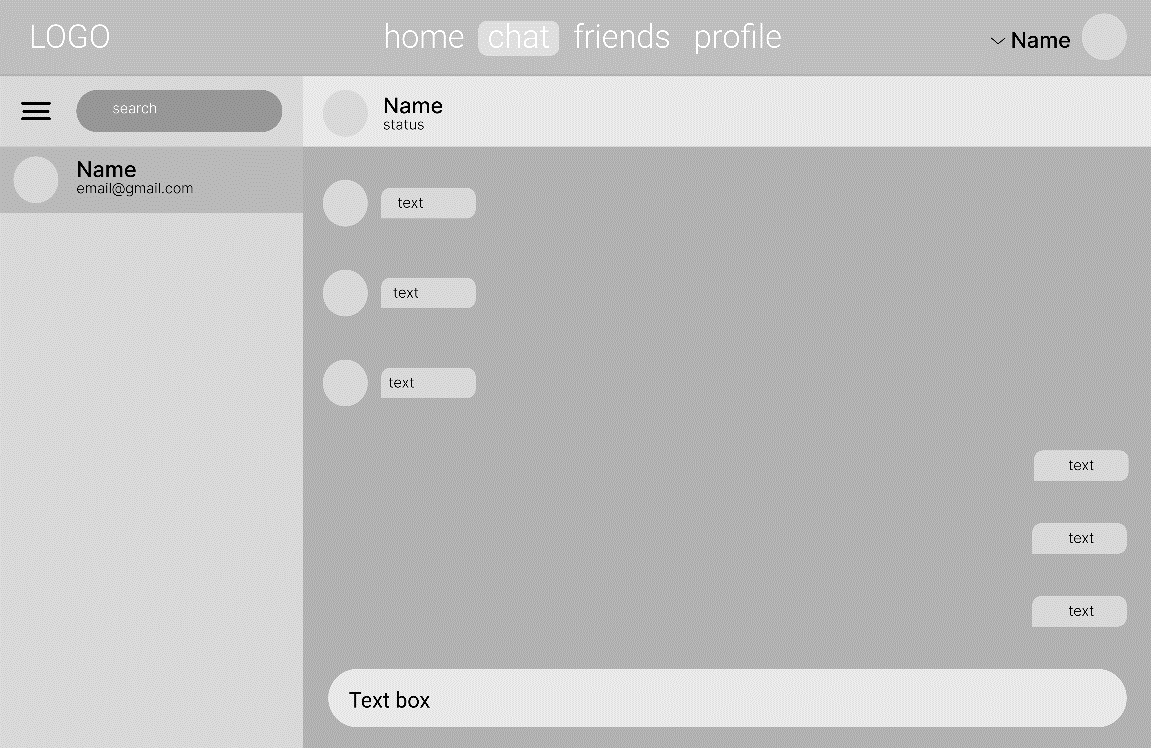
The database design for Chatlinedefines the structure for managing users, messages, conversations, and friendships. The User table stores essential user details, including authentication data and encryption keys. The Message table records encrypted messages exchanged between users, linking each message to a Conversation. The Conversation table tracks chat sessions with timestamps. The Friendshiptable manages relationships between users, while the FriendRequest table records pending connection requests. The HasMessages table establishes a many-to-many relationship between users and conversations. This design ensures secure, structured communication with encryption and user verification.

## Interface Design

The User Interface (UI) and the User Experience (UX) plays a crucial role in the system, as it directly affects how easily new users can comprehend the various components and navigate them to accomplish their desired objectives. The system's interface will be displayed using the following representations:

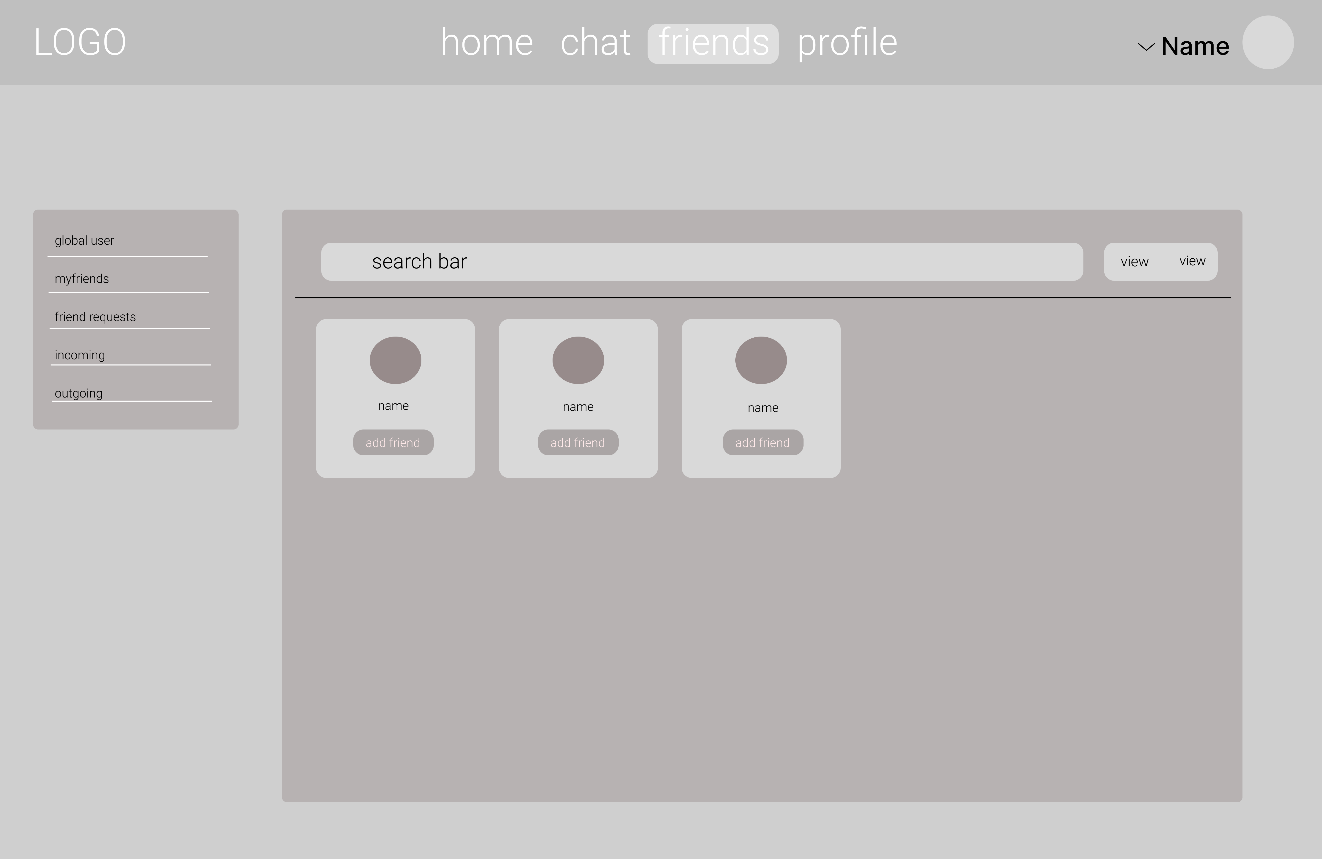


**Figure 3.8: Interface for Login Page**

****

**Figure 3.9: Interface for Chat Page**

**Figure 3.10: Interface for User Profile page**



**Figure 3.11: Interface for Friends Page**

# 

# CHAPTER 4: Implementation and Testing

## Implementation

The implementation phase involves executing the plan and deploying the system. For Chatline, this phase includes the development and deployment of a software system to manage real-time messaging, user authentication, secure communication, and chat data handling.

### Tools Used

Various tools were used while making this Chatline Application. Some of them are discussed below:

1. **Front-End**: The front-end of the application is built using React.js for efficient UI rendering, Redux Toolkit for state management, and Tailwind CSS for responsive and modern styling.
2. **Back-End**: The back-end utilizes Node.js and Express for handling server-side logic, with Socket.io for enabling real-time communication. MySQL is used for database formation and sql queries are used to interact with the database as needed.

### Server: WAMP is used as the local server environment to run the MySQL database. It provides SQL server support, client tools, administrative utilities, APIs, and a relational database system used to store and manage application data.

1. **Git**: For version control and change tracking, we used git along with GitHub. This ensures coordination among the programming members during app development.

### Implementation Details of Modules

**Socket.io:**

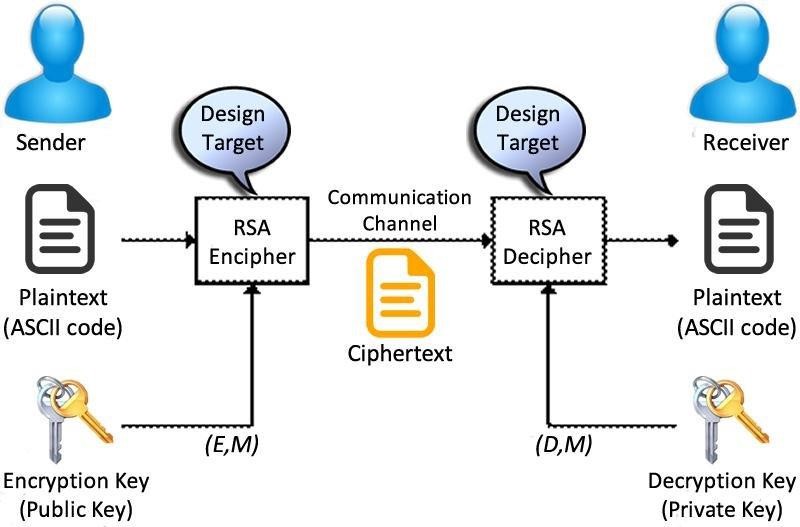
For the real-time messaging component, we use Socket.io, a powerful library that enables real-time, bidirectional communication between web clients and servers. Unlike traditional methods such as polling or periodic refreshing, Socket.io ensures that messages are delivered instantly, providing a seamless and responsive user experience. This technology is chosen specifically to enhance the efficiency and performance of real-time communication in our chat application.

**RSA Encryption:**

The RSA (Rivest–Shamir–Adleman) algorithm is utilized as the core encryption technique in the project to ensure the security of user communications. RSA is a public-key cryptography algorithm that relies on the mathematical properties of prime numbers to generate secure keys. In the context of this chat application, RSA is used to encrypt messages before they are transmitted over the network, ensuring that only the intended recipient can decrypt and read the messages.

RSA encryption involves two keys: a public key for encryption and a private key for decryption. When a user sends a message, the application will encrypt the message using the recipient's public key. The recipient will then use their private key to decrypt the message. This approach ensures that even if an attacker intercepts the encrypted message, they will not be able to decrypt it without the corresponding private key.

The choice of RSA is driven by its strong security guarantees and widespread use in secure communication protocols. RSA is well-suited for applications where confidentiality and integrity of data are paramount, making it an ideal choice for this real-time chat application. Additionally, the implementation of RSA in the project provides a robust layer of security, ensuring that user messages remain private and protected from unauthorized access.



**Figure 4.1: Working of RSA Algorithm [5]**

**RSA key generation:**

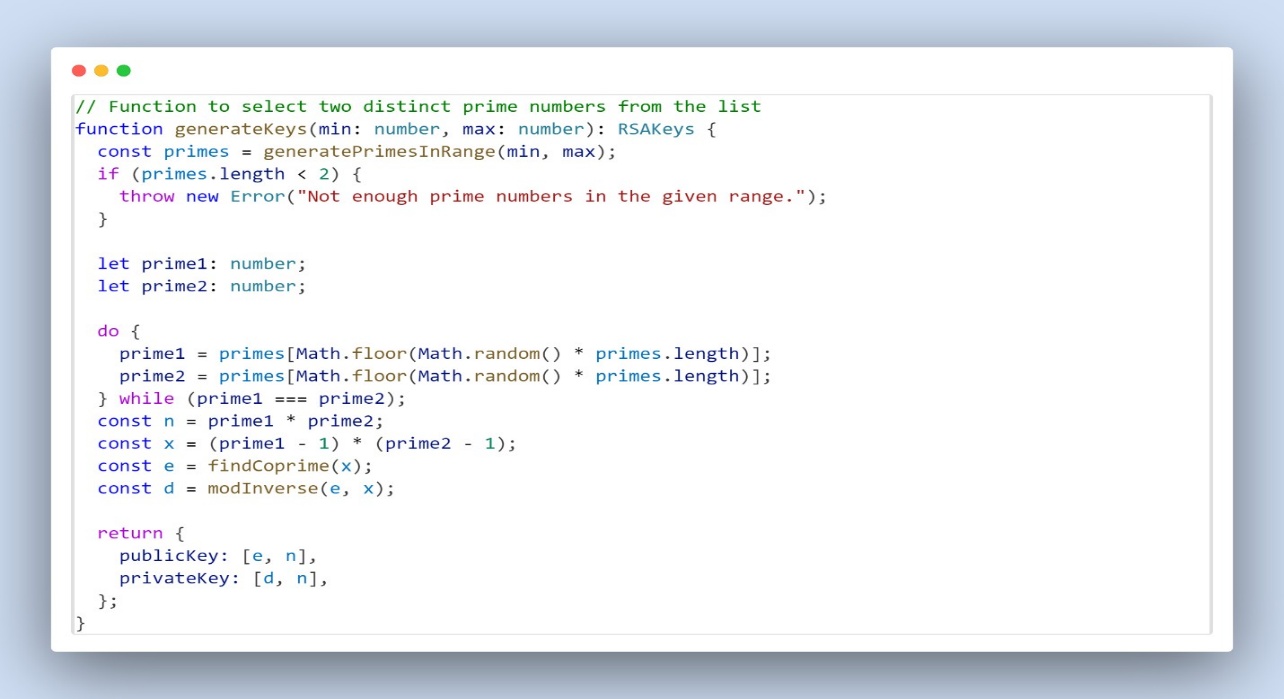
1. Choose two large prime numbers **p** and **q**.
2. Compute the modulus: **n = p × q** for both public and private keys.
3. Compute Euler’s Totient: **ϕ(n) = (p − 1)(q − 1)**.
4. Select a public exponent **e** such that **1 < e < ϕ(n)** and **gcd(e, ϕ(n)) = 1**  (meaning e is co-prime with ϕ(n)).
5. Compute the private exponent **d** such that:  
     **(e × d) ≡ 1 (mod ϕ(n))**
6. Public key is {e, n}.
7. Private key is {d, n}.

**Encryption:** To encrypt a message **m** using the public key

**c = mᵉ mod n**

**Decryption:** To decrypt cipher text **c** using the private key

**m = cᵈ mod n**

**Code Snippet:**

**Tracing:**

1. **Primes:** **p** = 61, **q** = 53
2. **n = p × q = 61×53 = 3233**
3. **ϕ(n) = (61 − 1)(53 − 1) = 3120**
4. **Public Key:** **e** = 17
5. **Private Key:** **d** = 2753

**Encryption Process**

**Steps:**

1. Convert each character in the plaintext to its ASCII value.
2. Encrypt each ASCII value using modular exponentiation:

cipherChar=(charCodee) mod n (∴ C=(Me) mod n )

**Code Snippet:**

**Tracing:**

1. Message: "Hello world"
2. ASCII values: H=72, e=101, l=108, l=108, o=111, (space)=32, w=119, o=111, r=114, l=108, d=100
3. Cipher text: 3000~1313~745~745~2183~32~1687~2183~1140~745~100

**Decryption Process**

**Steps:**

1. Split the ciphertext into individual numbers.
2. Decrypt each number using modular exponentiation:

plainChar=(cipherChard) mod n

1. Convert each decrypted number back to its ASCII character.

**Code Snippet:**

**Tracing:**

1. Ciphertext: “3000~1313~745~745~2183~32~1687~2183~1140~745~100”
2. Decrypted ASCII values: 72, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100
3. Decrypted Message: "Hello world”

## Testing

## The testing is done to check the system process in the Chat application. The System is tested to see if the can properly work programming and logical errors. It is tested to makes sure that all of the systems requirements are met and can be processed effectively.

### Unit Testing

### Unit testing is a software development method in which the smallest modules or part of the system components, also known as unit are individually tested for functional operation.

### Table 4.1 User Login/Signup

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Description** | **Inputs** | **Expected result** | **Actual result** | **Status** |
| 1.1 | **Go to Login page** | http://localhost:5173/ |  |  |  |
| 1.2 | **Signup with invalid email** | Name:Sujal Munikar  Email: astro@.com  Password: p@ssword | Signup should fail with an error message: "Invalid email address." | Shows  Error message and fails the signup. | Pass |
| 1.3 | **Signup with valid data** | Name:Sujal Munikar  Email: astro@gmail.com  Password:p@ssword | User should be successfully registered in the system. | Signup success | Pass |
| 1.4 | **Login with incorrect password** | Email: astro@gmail.com  Password:p@sswor12 | Login should fail with an error message: "Invalid credentials." | Login failed | Pass |
| 1.5 | **Login with correct credentials** | Email: astro@gmail.com  Password:p@ssword | User should be successfully logged in and redirected to the homepage. | Login Success and redirected to  homepage | Pass |

### Table 4.2 Bad Word Filtering

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Description** | **Inputs** | **Expected Result** | **Actual Result** | **Status** |
| 2.1 | **Send message without bad words** | Message: "Hello, how are you?" | Message should be sent without any modifications. | Message: "Hello, how are you?" | Pass |
| 2.2 | **Send message with one bad word** | Message:  "You are a  fool!" | Message should be sent as: "You are a \*\*\*\*!" | Message:  "You are a fool!" | Fail |
| 2.3 | **Send message with multiple bad words** | Message: "This is a bad and  stupid app!" | Message should be sent as: "This is a \*\*\* and \*\*\* app!" | Message:  "This is a \*\*\* and \*\*\* app!" | Pass |
| 2.4 | **Send empty message** | Message:  "" | Message should not be sent, and an error message should display: "Message cannot be empty." | Did not allow empty message to empty. | Pass |
| 2.5 | **Send message with only bad words** | Message:  "Idiot, dumb,  stupid!" | Message should be sent as: "\*\*\*, \*\*\*, \*\*\*!" | Message:  "\*\*\*, \*\*\*, stupid!" | Fail |

### Table 4.3 RSA Encryption in Chat

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Description** | **Inputs** | **Expected Result** | **Actual Result** | **Status** |
| 3.1 | **Encrypt and decrypt message** | Message: "Hello, secure chat!" | The encrypted message should be successfully decrypted to its original form. | Message: "Hello, secure chat!" | Pass |
| 3.2 | **Send encrypted message over**  **Socket.IO** | Message:  "RSA is  working!" | The message should be encrypted, sent via  Socket.IO, and decrypted on the receiver's end to match the original message. | Message:  "RSA is  working!" | Pass |

### Table 4.4 Socket.io Connection

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **Description** | **Inputs** | **Expected Result** | **Actual Result** | **Status** |
| 4.1 | **Establish socket connection** | - | A connection should be successfully established with the server. | Connection Stablished | Pass |
| 4.2 | **Receive realtime messages** | Sender sends:  "Hi there!" | The receiver should immediately receive the message:  "Hi there!" | Receiver receives:  "Hi there!" | Pass |

### System Testing

### Table 4.5 User Registration and Login

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Description** | **Steps** | **Expected Result** |
| **ST-1** | **User**  **Registration with valid data** | 1. Open the app.  2. Go to the "Sign Up" page.   1. Enter valid details: Name, Email, Password and Confirm Password. 2. Submit the form. | User should be successfully registered and redirected to the login page. |
| **ST-2** | **User**  **Registration with invalid email** | 1. Open the app.  2. Go to the "Sign Up" page.   1. Enter valid Name, and Password, but an invalid email. 2. Submit the form. | An error message should display:  "Invalid email address." |
| **ST-3** | **User Login with valid credentials** | 1. Open the app.  2. Go to the "Login" page.   1. Enter a registered email and password. 2. Submit the form. | User should be successfully logged in and redirected to the dashboard if verified else go to opt verification page. |
| **ST-4** | **User Login with invalid credentials** | 1. Open the app.  2. Go to the "Login" page.   1. Enter an unregistered email or incorrect password. 2. Submit the form. | An error message should display:  "Invalid credentials." |

### Table 4.6 Chat Functionality

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Description** | **Steps** | **Expected Result** |
| **ST-5** | **Send a valid**  **message** | 1. Log in to the app.  2. Navigate to a chat window.   1. Enter a valid message: "Hello!". 2. Click "Send". | The message should be sent and appear in the chat window. |
| **ST-6** | **Send a message with bad words** | 1. Log in to the app.  2. Navigate to a chat window.   1. Enter a message:   "You are a fool!".   1. Click "Send". | The message should appear in the chat window with bad words replaced: "You are a \*\*\*!". |
| **ST-7** | **Receive a realtime message** | 1. Open a chat window with another user.  2. Have the other user send a message:  "Hi there!". | The message "Hi there!" should immediately appear in the chat window. |
| **ST-8** | **Online indicator functionality** | 1. Open a chat window. 2. Observe the other user's chat window. | The other user should see a online indicator in real-time. |

### Table 4.7 RSA Encryption and Decryption

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Description** | **Steps** | **Expected Result** |
| **ST-9** | **Encrypt and decrypt message** | 1. Log in and navigate to a chat window. 2. Send a message: "Hello!". | The message should be encrypted before transmission and decrypted on the receiver’s end. |

### Table 4.8 Socket.io Connectivity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Description** |  | **Steps** | **Expected Result** |
| **ST-10** | **Establish socket connection** |  | 1. Open the app.  2. Log in with valid credentials.  3. Check the console or logs for a successful Socket.IO connection. | A socket connection should be successfully established. |
| **ST-11** | **Reconnect after disconnect** | | 1. Open the app.  2. Log in with valid credentials.   1. Simulate a network disconnect. 2. Restore the network connection. | The app should automatically reconnect to the server. |
| **ST-12** | **Handle a serverside**  **socket event** | | 1. Log in to the app.  2. Trigger a server event, such as receiving a new message.  3. Observe the client-side response. | The client should handle the event and display the new message. |

### Table 4.9 Username & Password Update

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Description** |  | **Steps** | **Expected Result** |
| **ST-13** | **Change Username** |  | 1. Navigate to Profile Page.  2. Enter new Username and press ‘Update name’ button.  3. Confirm the update, if you want to change name by pressing yes in confirmation message. | The username for the User must have changed to the one they provided. |
| **ST-14** | **Change Password** | | 1. Navigate to Profile Page.  2. Enter current Password and new Password  3. Press the ‘Update Password’ button.  4. Confirm the update, if you want to change password by pressing yes in confirmation message. | The current password must have changed to the new Password when logging in next time. |

**Table 4.10 Friend Request and Unfollow User**

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Description** | **Steps** | **Expected Result** |
| **ST-15** | **Send Friend Request and Add Friend** | 1. Navigate to Friends Page.  2. Enter username or email in Search bar to find the friend.  3. Press ‘Add friend’ button to send request.  4. You can cancel request from outgoing request in left side bar.  5. After the Friend accepts the request, you can see them in MyFriends. | The User should be able to send requests and cancel them and when accepted they can see them in MyFriends. |
| **ST-16** | **Unfollow the user** | 1. Navigate to Friends Page.  2. In left sidebar, go to MyFriends and click ‘unfollow’ button.  3. A confirmation message will pops up, if you want to unfollow that user.  4. You can observe that the user is removed from Myfriends. | The User should be able to unfollow any user they have added as friends. |

**Table 4.11 System Non-Functional Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Description** | **Steps** | **Expected Result** |
| **ST-17** | **Light and dark mode functionality** | 1. Navigate to menu.  2. Toggle between light and dark mode.  3. Check the UI for consistency in both modes. | The UI should update correctly based on the selected mode. |

# Chapter 5: Conclusion And Future Recommendation

1. **Conclusion**

Chatline was developed to solve two major challenges found in many chat applications: secure communication and real-time message delivery. By using Socket.IO, the system is able to provide smooth, low-latency messaging, while RSA encryption ensures that user conversations remain private and protected from unauthorized access. Additional features such as toxic word filtering and a friend request system create a safer and more interactive communication environment.

With a modern technology stack including React.js, Redux Toolkit, Node.js, and SQL-based database integration, the application performs efficiently and is scalable for future growth. Overall, Chatline achieves its goal of providing a secure, reliable, and user-friendly chat platform, forming a strong foundation for further improvements and expansion.

## ****Future Recommendations****

To enhance the functionality and user experience of Chatline, the following improvements can be considered in future versions:

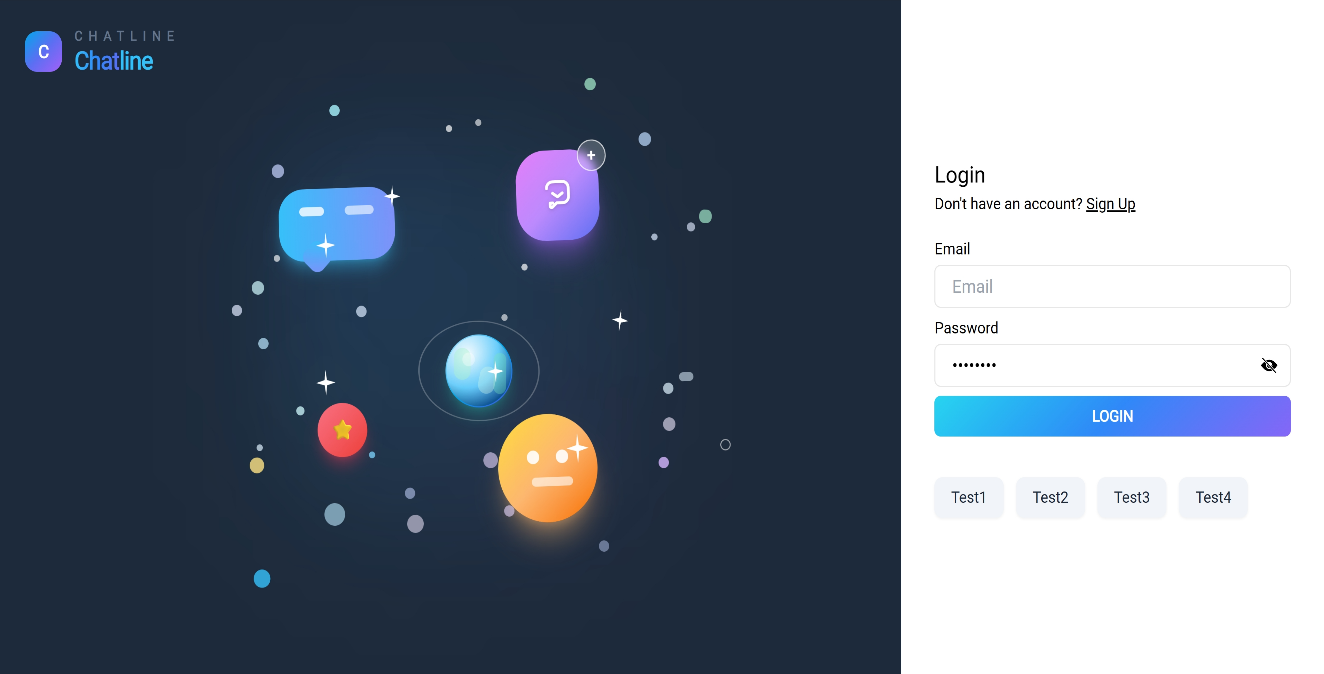
1. **Media File Sharing** – Allow users to send images, audio, and videos to make conversations more interactive.
2. **Mobile Application** – Develop an Android/iOS app to increase accessibility and convenience.
3. **Long-Form Message Support** – Enable sending longer text messages for more detailed communication.
4. **Phone Number OTP Verification** – Add phone-based OTP authentication for stronger user security and easier account recovery.
5. **Admin Dashboard** – Provide an admin panel to monitor system activity, manage users, and track performance.
6. **Website Integration** – Allow the chat system to be embedded into external websites, increasing its usability across different platforms.

# REFERENCES

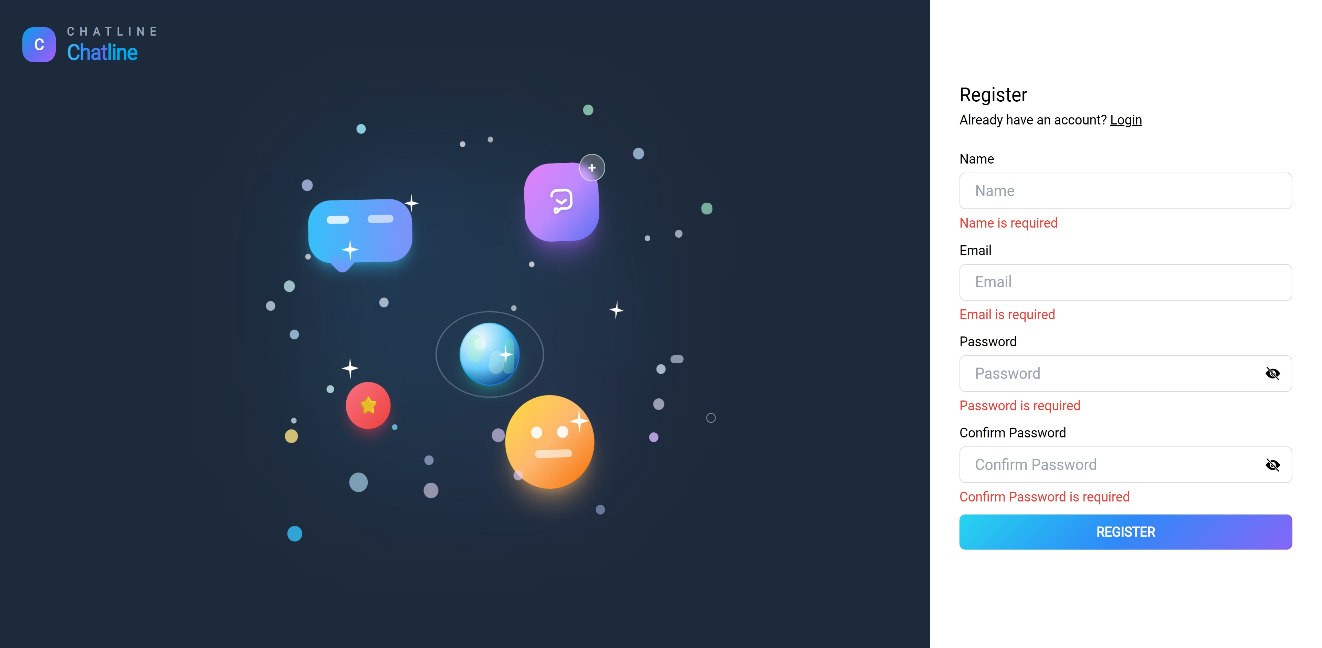
|  |  |
| --- | --- |
| [1] | "Vector Stock," [Online]. Available: https://www.vectorstock.com/royalty-free-vector/agile-methodology-icon-vector-35537764 |
| [2] | B. Schneier, "Comprehensive exploration of cryptographic protocols and algorithms, including RSA.," in *Bruce Schneier*, New York, 1996. |
| [3] | M. A. D. Laura T. Brown, "Enhancing Real-Time Messaging Applications Using WebSocket and Socket.io," *Journal of Web Engineering and Technology,* 2024. |
| [4] | E. R. J. John D. Smith, "Secure and Efficient End-to-End Encrypted Messaging Framework for Real-Time Communication," *IEEE Transactions on Information Forensics and Security,* 2023. |

[5] H. Bodur, "ResearchGate," [Online]. Available: https://www.researchgate.net/figure/RSA-algorithm-structure\_fig2\_298298027

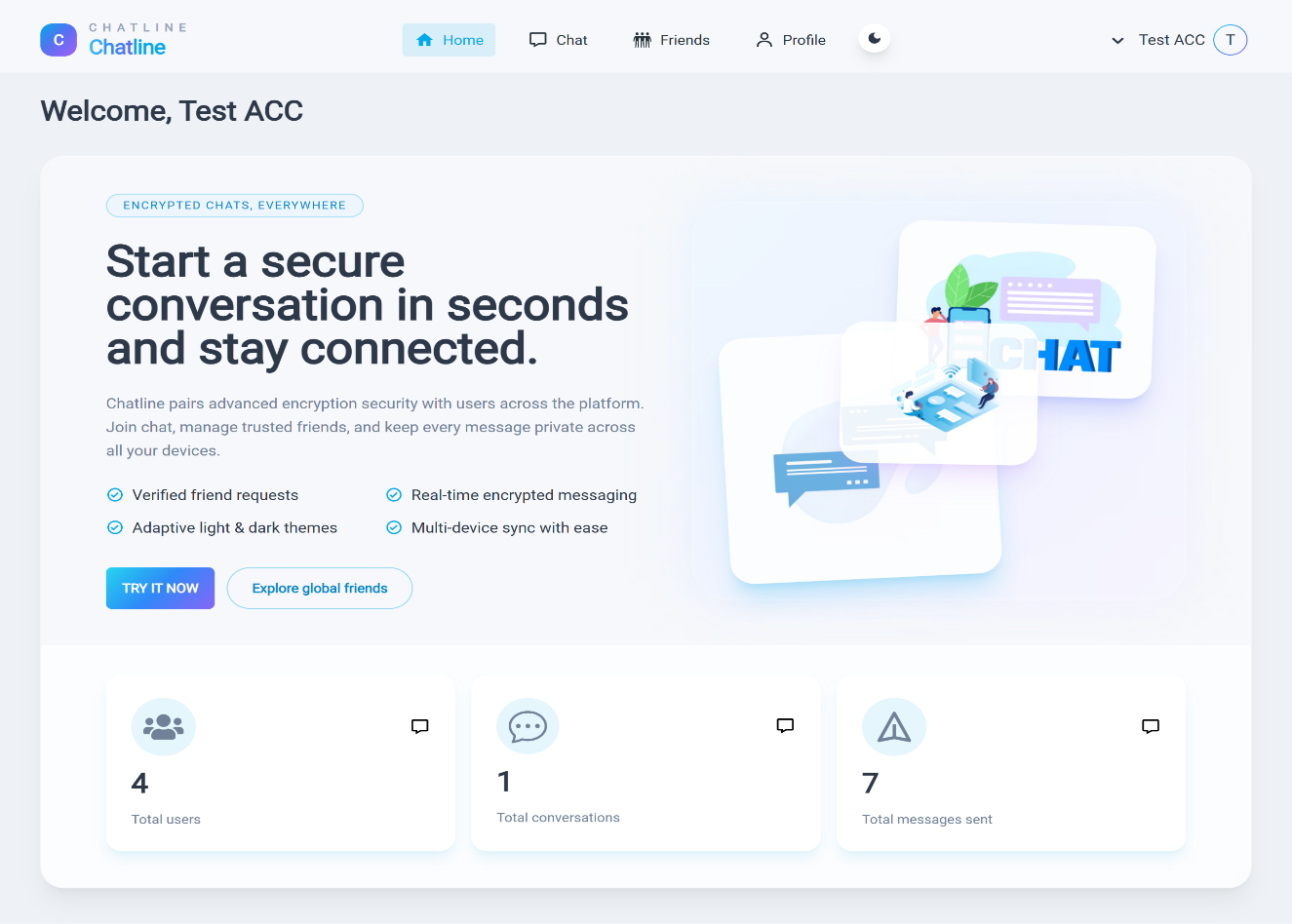
**APPENDICES**

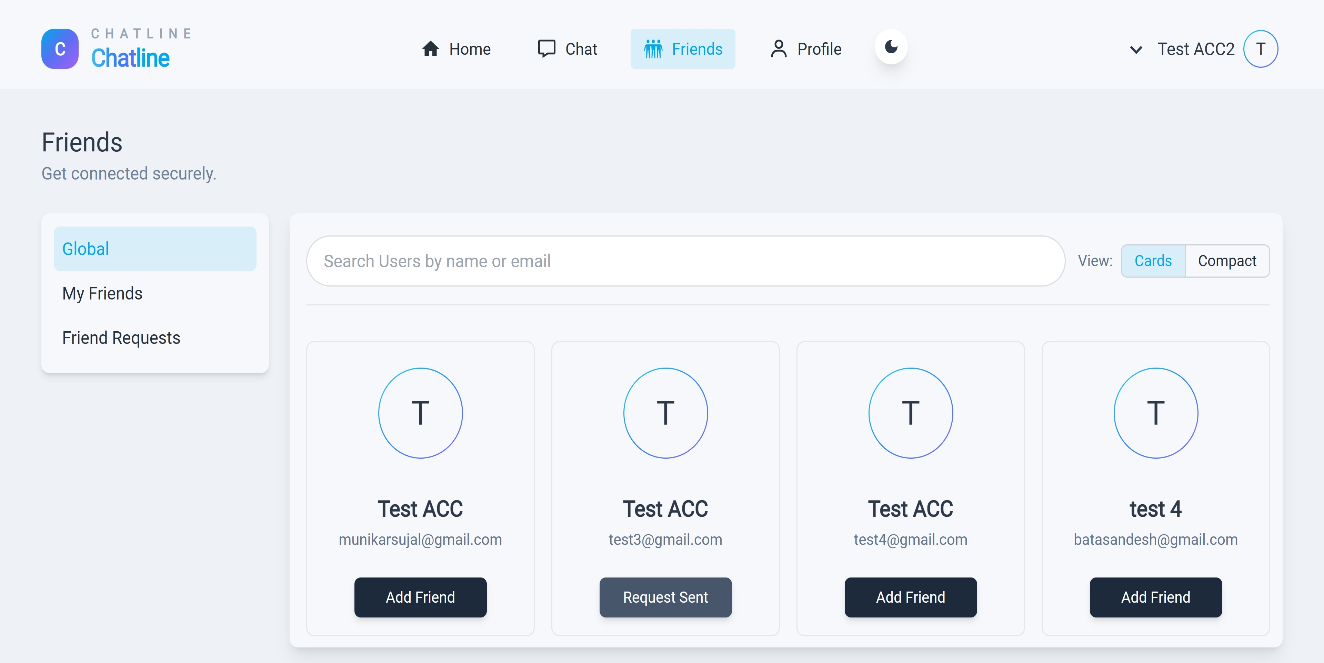
****

**Figure 1: Login Page**

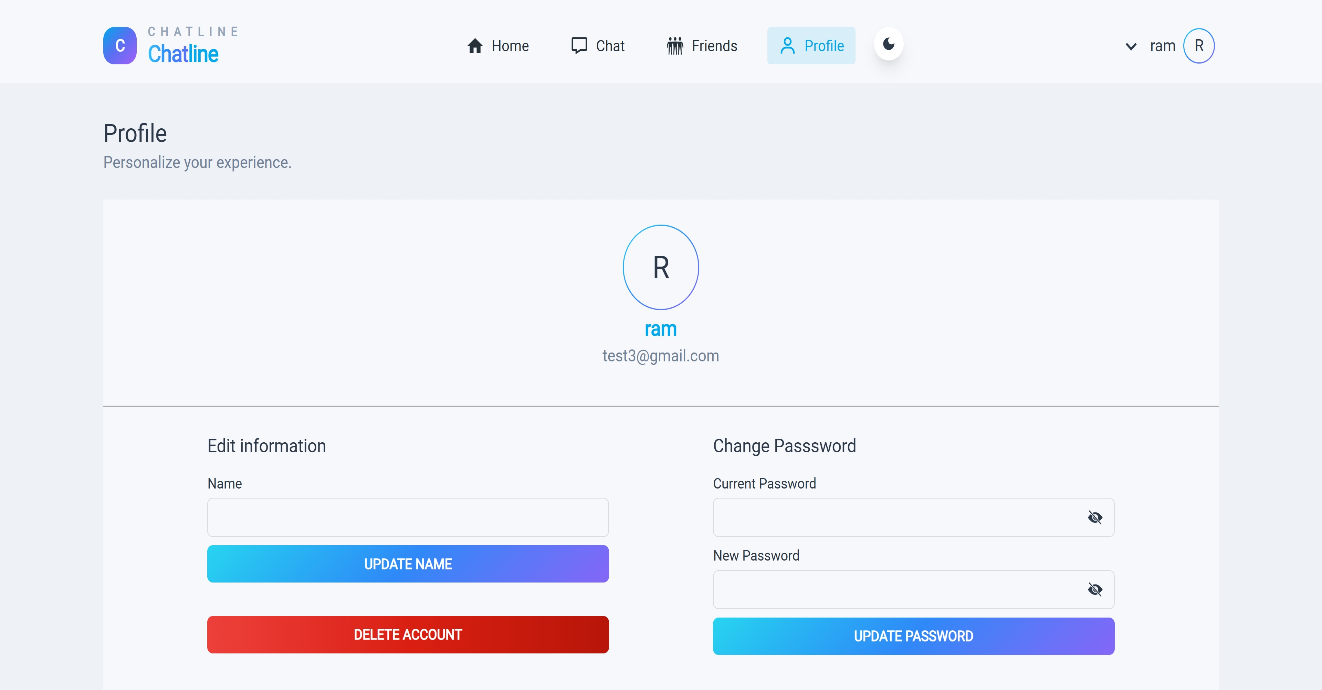


**Figure 2: Register Page**

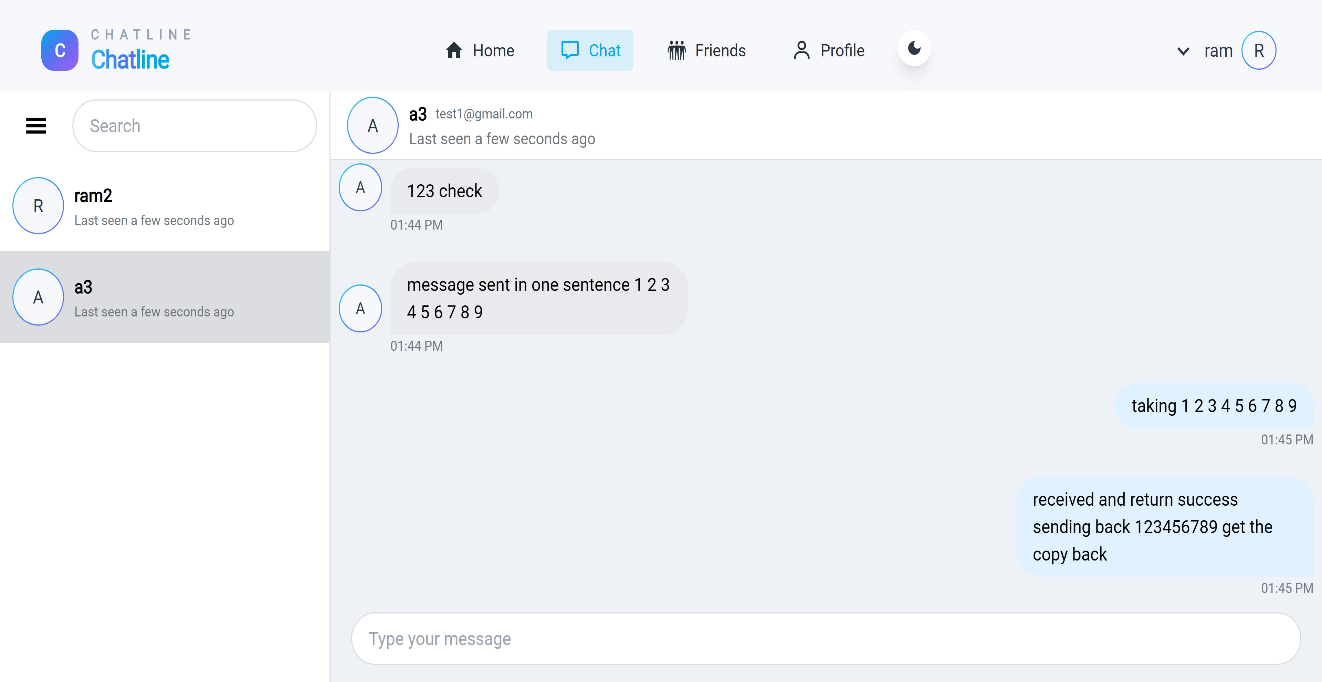
**Figure 3: Home Page**



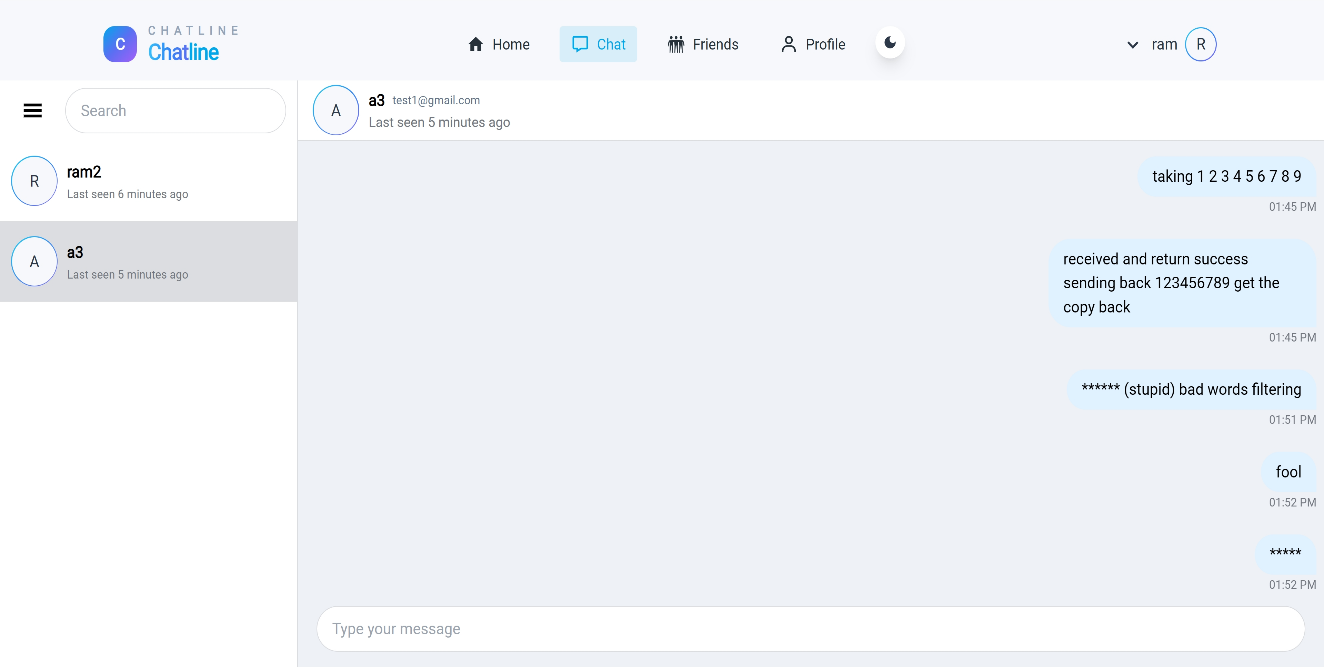
**Figure 4: Friends Page**



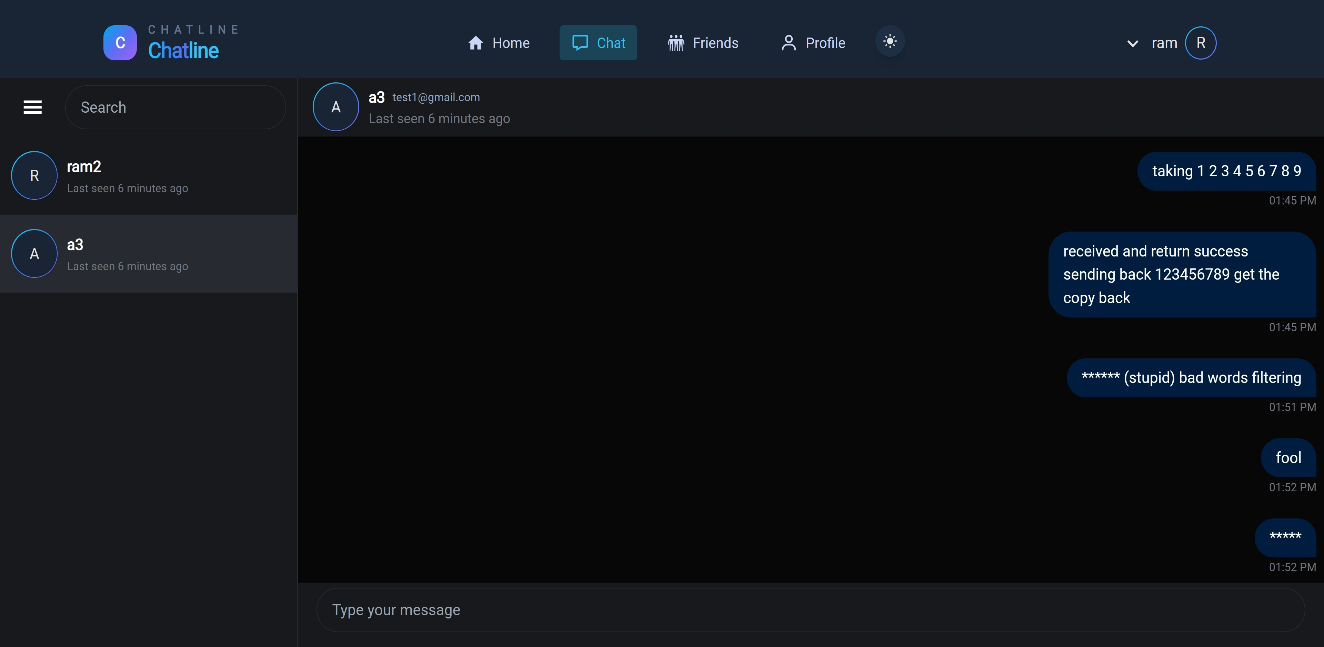
**Figure 5: Profile Page**



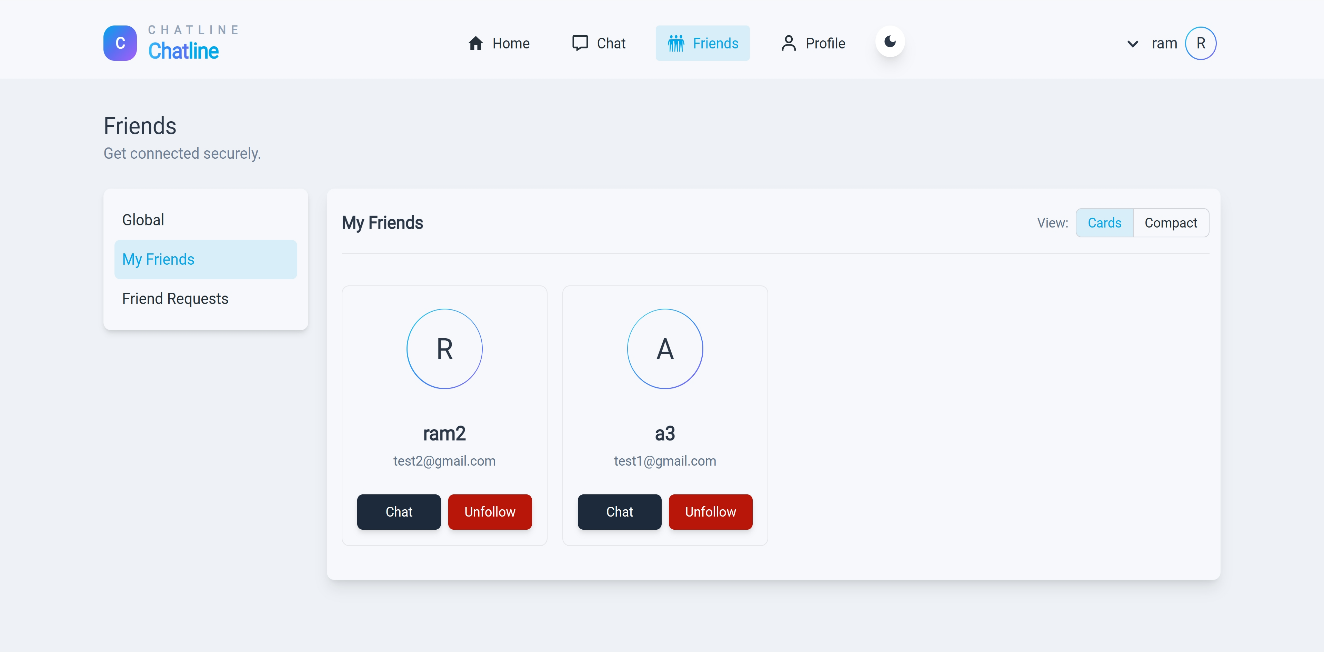
**Figure 6: Chat Page of Chatline**



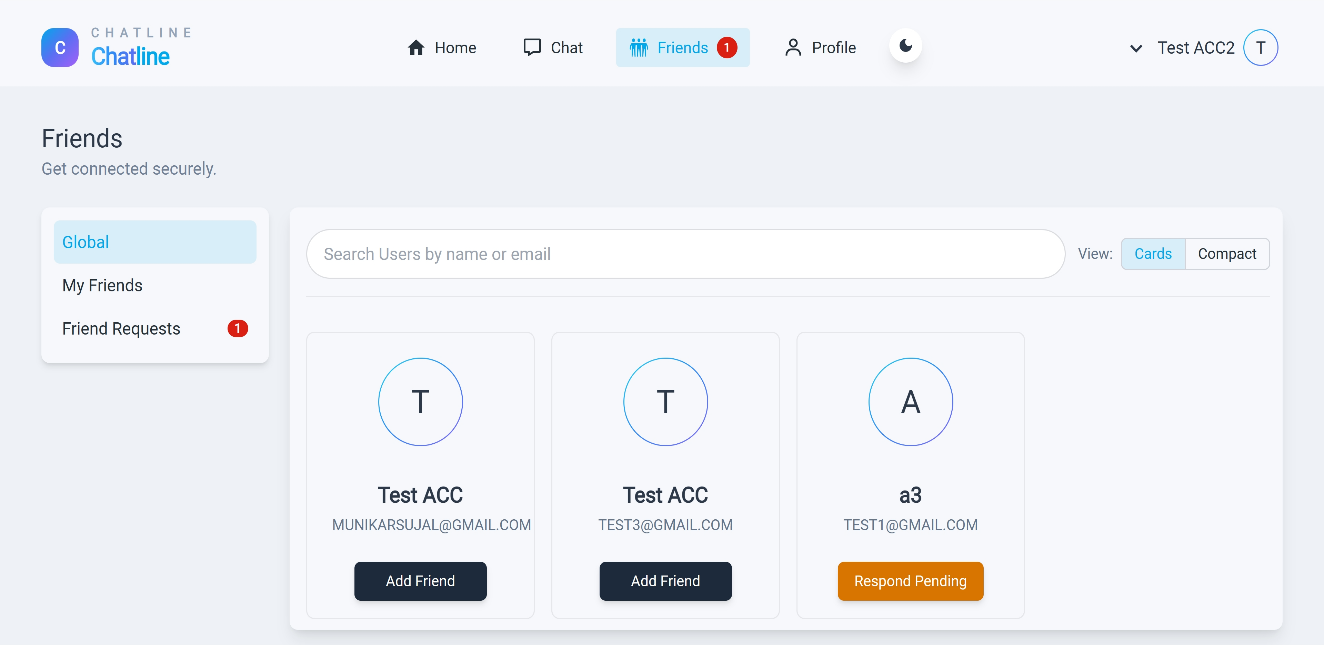
**Figure 7: Explict Filtering in Chat Page**



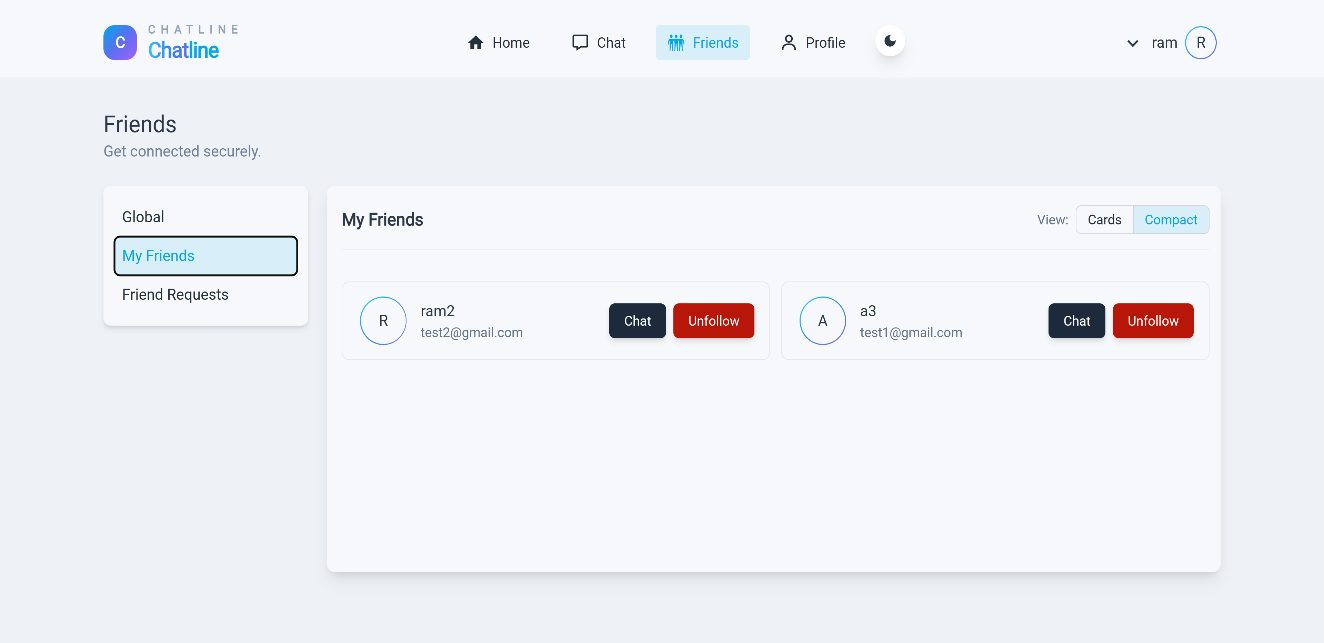
**Figure 8: Dark Mode in Chat Page**



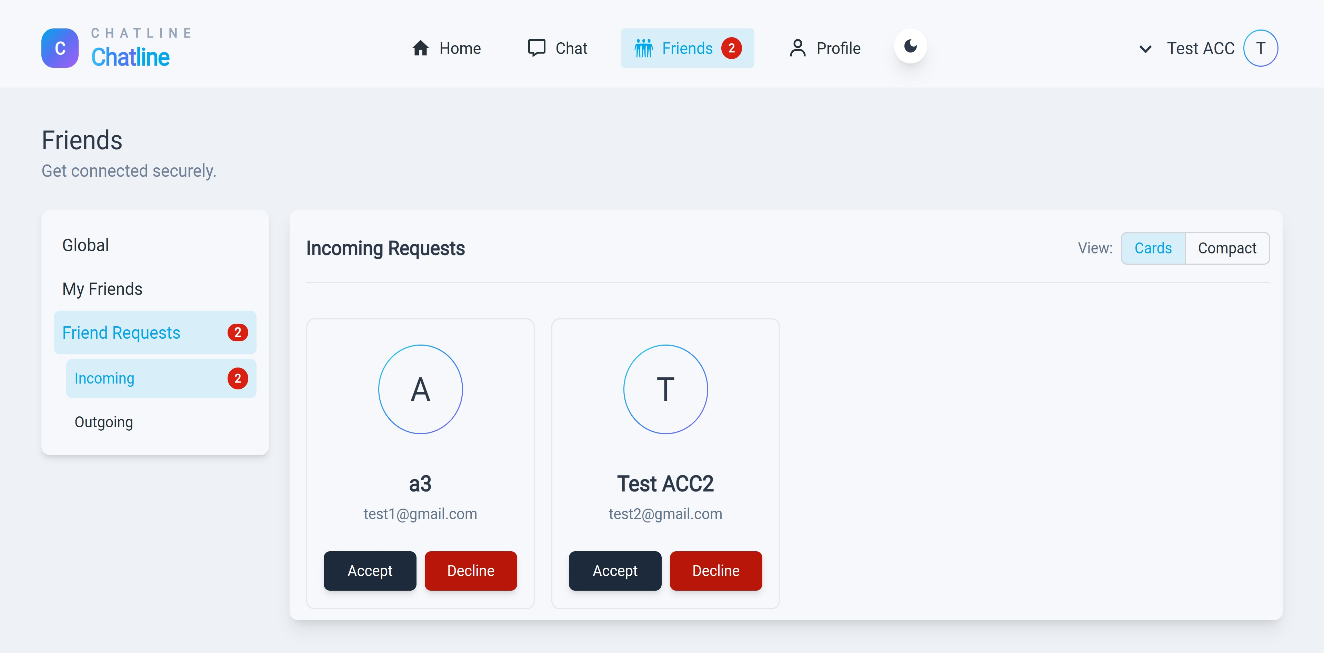
**Figure 9: MyFriends when both are friends**



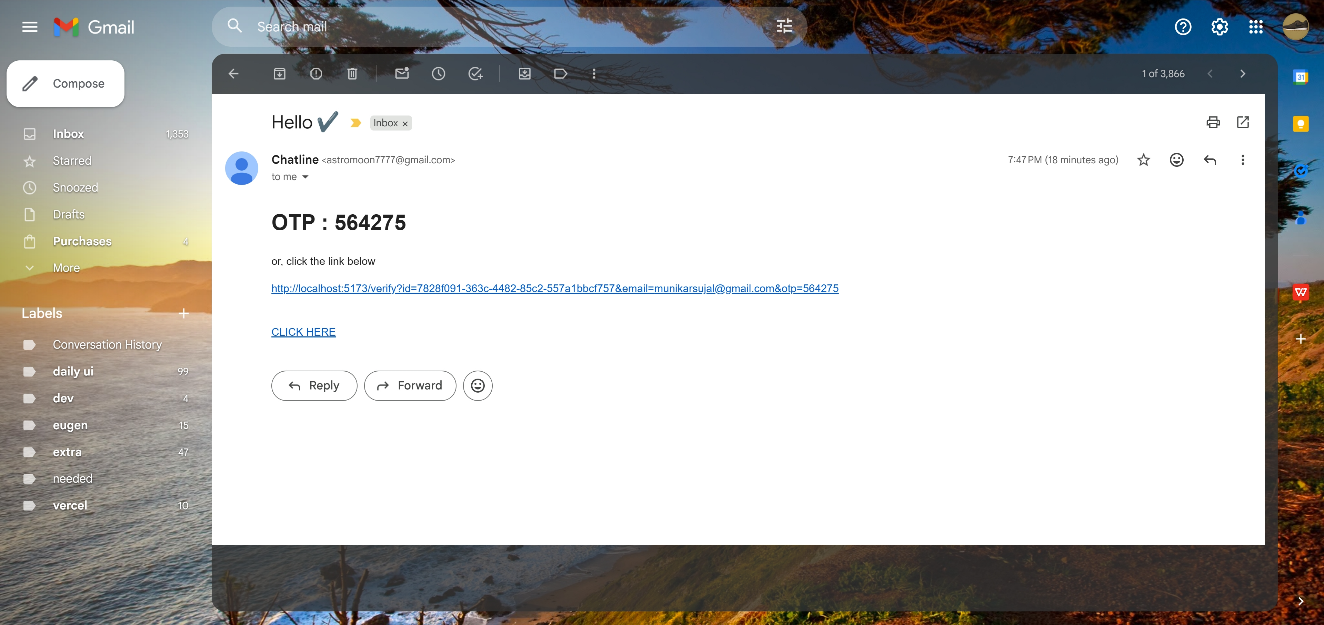
**Figure 10: Global Friends when request is being sent**



**Figure 11: MyFriends in Compact View**



**Figure 12: Incoming Requests when friend request is being sent**



**Figure 13: Gmail when Otp is sent for verification**