A Project Report on

SECURE CONNECT MESSENGER

Submitted in partial fulfillment of the requirement for the award of the Degree in

Bachelor of technology

in

Computer Science and Engineering

Submitted By

MOGILICHERLA NANDINI	**21031A0532**
PINISETTI GAYATRI	**22035A0506**
KATHI VASANTH ABHISHEK	**21031A0529**
R. VEERA VENKATA SREERANGANADH	**22035A0508**
VEMULA PADMA PRIYA	**21031A0560**

Under the esteemed guidance of

Mrs. B. JYOTHI

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING NARASARAOPET

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

NARASARAOPET – 522601 PALNADU, ANDHRA PRADESH

2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING NARASARAOPET

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

NARASARAOPET – 522601 PALNADU, ANDHRA PRADESH

2024-2025

CERTIFICATE

This is to certify that the dissertation entitled "SECURE CONNECT MESSENGER" that is being submitted by in **MOGILICHERLA NANDINI (21031A0532), PINISETTI GAYATRI (22035A0506),KATHI VASANTH ABHISHEK (21031A0529), RAMESWARAPU VEERA VENKATA SREERANGANADH (22035A0508), VEMULA PADMA PRIYA(21031A0560)** partial fulfilment for the award of Bachelor of Technology in Computer Science and Engineering to the University College of Engineering Narasaraopet, Jawaharlal Nehru Technological Kakinada is a record of Bonafide work carried outby them under my guidance and supervision.

The results embedded in this dissertation have not been submitted to any other university/institute for the award of any degree/diploma.

PROJECT SUPERVISOR

Mrs. B. JYOTHI

Assistant Professor(c)
Department of CSE
UCEN JNTUK

HEAD OF DEPARTMENT

Dr. G. MADHAVI

Assistant Professor & HOD(i/c)
Department of CSE
UCEN JNTUK

External Examiner

DECLARATION

We hereby declare that the work described in the project report entitled "**SECURE CONNECT MESSENGER**" which is submitted by us in partial fulfillment of the requirements for the award of the degree Bachelor of Technology in the Department of **COMPUTER SCIENCE AND ENGINEERING** to the college, University College of Engineering Narasaraopet, Jawaharlal Nehru Technological University Kakinada, Andhra Pradesh is a record of original and independent research work done by us during the academic year 2024-2025 under the supervision of **Mrs. B. JYOTHI**, Assistant Professor (c), CSE Department. The work is original and has not been submitted for the award of any Degree diploma Associate Fellowship or other similar title to this or any other university.

Name of the Student	Roll No	Signature
MOGILICHERLA NANDINI	**21031A0532**	
PINISETTI GAYATRI	**22035A0506**	
KATHI VASANTH ABHISHEK	**21031A0529**	
RAMESWARAPU VEERA VENKATA SREERANGANADH	**22035A0508**	
VEMULA PADMA PRIYA	**21031A0560**	

PLACE:

DATE:

ACKNOWLEDGEMENT

It is needed with a great sense of pleasure and immense sense of gratitude that we acknowledge the help of these individuals. We express our sincere thanks and gratitude to many people who helped, supported, encouraged, and challenged us throughout our project and academic program. We take privilege to express our heartfelt gratitude to Project Supervisor and our Project Coordinator **Mrs. B. JYOTHI**, Assistant Professor (c) of CSE Department and our Head of the Department **Dr. G. MADHAVI**, Assistant Professor & HOD(i/c) Department of CSE, for his valuable suggestions and constant motivation that greatly helped us in successful completion of the project.

We express our sincere thanks to our beloved **Dr. Y. S. KISHORE BABU**, Vice-Principal (I/C), UCEN JNTUK and **Prof. CH. SRINIVASA RAO**, Principal, UCEN JNTUK for providing us with a good infrastructure and environment to carry out this project. We are thankful to all the faculty members for extending their kind cooperation and assistance. Finally, we are extremely thankful to our parents and friends for their constant help and moral support.

TEAM MEMBERS

MOGILICHERLA NANDINI	21031A0532
PINISETTI GAYATRI	22035A0506
KATHI VASANTH ABHISHEK	21031A0529
R. VEERAVENKATASREERANGANADH	22035A0508
VEMULA PADMA PRIYA	21031A0560

LIST OF CONTENTS

ABSTRACT

This project introduces a secure end-to-end chat application built using the MERN (MongoDB, Express.js, React, Node.js) stack. It prioritizes data security and user privacy by leveraging the AES (Advanced Encryption Standard) cryptographic library to encrypt and decrypt messages. Messages are encrypted on the sender's side and can only be decrypted by the recipient, ensuring that only the intended participants can access the content. The backend logic and API endpoints are handled using Express.js and Node.js, while React provides an intuitive and responsive user interface, ensuring seamless real-time communication. Additionally, WebSocket are employed to enable real-time message exchange, enhancing the user experience by reducing latency and ensuring instant communication. This SECURE CONNECT MESSENGER offers a practical solution for both individuals and organizations seeking a privacy-focused communication tool. It ensures data confidentiality, prevents unauthorized access, and avoids message interception while maintaining high performance and usability. Key features include encrypted messaging, user authentication, secure message storage, and scalability, making it a reliable and secure platform for private communication. Beyond standard messaging, the application provides real-time notifications for new messages, mentions, and group updates, enhancing user engagement. A secure login system ensures data integrity, while the group chat feature allows users to communicate in private or professional settings efficiently. The app includes audio and video calling capabilities, facilitating direct and high-quality communication. ChatGPT integration offers AI-powered automated responses for enhanced interaction and quick replies. The delete chat feature enables users to manage and control their message history effectively, while profile update functionality allows for personalization and user preferences. These features collectively provide a seamless, interactive, and secure communication experience. With a strong focus on security, performance, and scalability, this chat application is designed to cater to both personal and enterprise-level communication needs, ensuring reliability and efficiency in modern digital communication.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

S. NO	ABBREVATION	EXPANSION
1.	**E2EE**	End-to-End Encryption
2.	**AES**	Advanced Encryption Standard
3.	MFA	Multi-Factor Authentication
4.	OAuth	Open Authorization
5.	MITM	Man-in-the-Middle Attacks

CHAPTER-1

INTRODUCTION

1.INTRODUCTION:

In today's digital era, secure communication is crucial to protect sensitive information from cyber threats. The SECURE CONNECT MESSENGER is designed to provide end-to-end encrypted messaging, ensuring privacy, data integrity, and confidentiality. This application enables users to exchange text messages, multimedia files, and voice notes securely. It incorporates robust encryption algorithms, user authentication, and secure key exchange mechanisms to prevent unauthorized access and data breaches.
Lately, there is a lot of fuss around end-to-end encrypted chat applications. WhatsApp and Signal are two messaging apps dominating the headlines, let's take a look at why - WhatsApp recently updated its privacy policy, stating that the messaging platform will share user data with other Facebook-owned and third-party apps. This has prompted several users to look for alternative platforms, top among them is Signal. Signal is essentially an encrypted messaging app. Messages sent through Signal are said to be encrypted, meaning the platform cannot access private messages or media, or store them on their server. This is called end-to-end encryption. End-to-End Encryption(E2EE) is the most important feature in real-time chat applications. Our article will cover:

- Real-time Systems
- Web Sockets
- End-to-end Encryption
- Comparison of messaging applications

For any app to feel real-time, the user needs to be kept updated with any activity happening as soon as possible. The challenge arises in selecting and implementing a suitable development technique. With the traditional request-response model, we have few options:

Refresh Webpage

The user might refresh the web page time-to-time to check for message updates. But that is not an optimal solution. This may result in bad UX.

HTTP protocol

The concept of HTTP request-response is widely used. But this requires establishing a TCP connection every time data is sent to the server. Being a one-way synchronous communication protocol, this may result in a lot of overheads while creating and destroying a TCP connection every time a message is sent in real-time chat application.

1.1SIGNIFICANCE OF PROJECT

The primary objective of this SECURE CONNECT MESSENGER is to facilitate private and secure conversations for individuals and organizations. It aims to protect user data through advanced encryption techniques, preventing any unauthorized access or eavesdropping. The application is ideal for personal use, business communication, and confidential discussions.

Key Features

End-to-End Encryption (E2EE): Messages are encrypted on the sender's device and decrypted only on the receiver's device, ensuring data security.

User Authentication: Multi-factor authentication (MFA) enhances account security.

Secure File Sharing: Users can share files securely without compromising privacy.

Self-Destructing Messages: Optional feature allowing messages to be deleted after a specified period.

Cross-Platform Compatibility: Available on web, mobile, and desktop devices.

Real-Time Messaging: Instant messaging with a smooth and efficient user experience.

Group Chats: Secure group conversations with encrypted messages for multiple participants.

1.2 SUMMARY

The SECURE CONNECT MESSENGER is a versatile web-based messaging platform created to ensure secure communication within organizations. It is designed specifically for internal data exchange, offering encrypted, real-time messaging with a strong focus on data privacy and user security. Using Firebase as the backend, the application delivers reliable data management, secure authentication, and smooth synchronization between users. Its front end, developed with React and JavaScript, features a responsive and user-friendly interface, enabling efficient communication across multiple devices. The lightweight design ensures compatibility with standard web browsers, eliminating the need for additional software installations. Firebase Fire store serves as the database, providing secure storage for chat logs and enabling real-time updates, which help organizations manage and oversee communication channels effectively. User authentication is handled through Firebase Authentication, ensuring that only authorized personnel can access the platform, thereby safeguarding sensitive information. Built with scalability in mind, the SECURE CONNECT MESSENGER can easily adapt to the growing needs of an organization, accommodating a larger user base without compromising performance or security.

CHAPTER 2
LITERATURE SURVEY

2.LITERATURE SURVEY: -

A SECURE CONNECT MESSENGER is essential in today's digital landscape, where data privacy and security threats are prevalent. Various studies and technologies have been developed to ensure secure and encrypted communication. This literature survey explores different approaches, protocols, and existing solutions in the domain of secure messaging.

1. Introduction to Secure Messaging

Secure messaging applications provide users with confidential, authenticated, and tamper-proof communication. The primary focus is on end-to-end encryption (E2EE), secure authentication, and metadata protection to prevent unauthorized access and surveillance.

2. Existing Secure Messaging Protocols

Several cryptographic protocols have been developed for SECURE CONNECT MESSENGERs:

Signal Protocol (Used by Signal, WhatsApp, and Messenger)

Offers perfect forward secrecy (PFS) and self-healing properties.

Uses a combination of Double Ratchet Algorithm, X3DH Key Agreement, and AES-GCM encryption.

- OMEMO (Used by Conversations, Chat Secure)
  An extension of the Signal Protocol for XMPP-based messaging apps.
  Provides multi-device support with strong encryption.
- OTR (Off-the-Record Messaging)
  Ensures encryption, authentication, deniability, and perfect forward secrecy.
  Mostly replaced by Signal due to lack of multi-device support.
- Matrix Protocol (Used by Element, Fluffy Chat)
  Decentralized protocol with E2EE and support for federated messaging.

3. Security Challenges in Secure Messaging

Despite advanced encryption, secure messaging apps face the following challenges:

Metadata Leakage: Even if messages are encrypted, metadata (who you chat with, when, and how often) can be tracked.

Man-in-the-Middle Attacks (MITM): Weak authentication can lead to interception.

Compromised Endpoints: If a user’s device is compromised (e.g., via malware), encryption alone cannot protect messages.

Backdoors & Government Surveillance: Some governments push for "backdoors" in encryption for monitoring.

.

4.Research Trends & Future Directions

Recent research in secure messaging focuses on:

Post-Quantum Cryptography (PQC): Developing encryption resistant to quantum attacks.

Decentralized & Blockchain-Based Messaging: Eliminating centralized servers for better security.

Zero-Knowledge Proofs (ZKP): Enabling identity verification without exposing user data.

AI-Powered Threat Detection: Detecting anomalies in communication patterns to prevent cyber threats.

5.Review of Existing SECURE CONNECT MESSENGERs

Several SECURE CONNECT MESSENGERs exist, each implementing different encryption and security measures:

Application	Encryption Protocol	Key Features
Signal	Signal Protocol	E2EE, No metadata storage, Open-source
WhatsApp	Signal Protocol	E2EE, Metadata stored, Owned by Meta
Telegram (Secret Chats)	MTProto	E2EE (only in Secret Chats), Cloud-based
Threema	NaCl cryptography	No phone number required, Metadata-free
Wickr	AES-256, ECDH-521	Self-destructing messages, Used in enterprises

These applications highlight the importance of strong encryption but differ in metadata handling, usability, and decentralization.

2.2 MOTIVATION

2.3 PROBLEM STATEMENT

In today's digital landscape, secure communication is a growing concern due to increasing cyber threats, data breaches, and unauthorized access to personal and corporate conversations. Traditional messaging platforms often lack end-to-end encryption, leaving user data vulnerable to interception, hacking, and surveillance. The objective of this project is to develop a **Secure Messaging App** that ensures **confidentiality, integrity, and authenticity** of communications. The app must implement strong encryption protocols, protect user metadata, and provide features such as self-destructing messages, multi-factor authentication, and secure file sharing.

2.4 SUMMARY

The Secure Messaging Application will be designed for: Personal users who need secure conversations. Businesses and enterprises requiring confidential communication. Government agencies dealing with classified information. The app will support text messaging, voice calls, video calls, and file sharing, all secured with advanced encryption. Additionally, it will integrate features like anonymous sign-up, decentralized storage, and security audits to maintain high privacy standards. Secure Connect Messenger is a cutting-edge messaging platform designed to provide encrypted, real-time communication with advanced security features. It leverages end-to-end encryption, multi-factor authentication, and blockchain technology to prevent unauthorized access and data tampering. The platform is built with a user-friendly interface while ensuring compliance with industry security standards. Secure Connect Messenger caters to businesses, professionals, and security-conscious individuals who require a trustworthy messaging solution for confidential communication.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Currently, several messaging platforms provide some level of security for communications, but most have notable limitations when it comes to end-to-end encryption, data storage, and overall privacy. Below is an overview of some existing systems for secure chat, including their strengths and weaknesses:

1. WhatsApp

Security Features:

WhatsApp offers end-to-end encryption for messages, meaning only the sender and recipient can read the message content.

The app uses Signal Protocol, which is widely regarded as one of the most secure encryption protocols available.

Users can enable two-step verification for extra security.

Limitations:

WhatsApp stores metadata on its servers, such as phone numbers, timestamps, and message frequency, which can be accessed by third parties.

Although end-to-end encryption is present for messages, voice calls and video calls are not immune to vulnerabilities.

Being owned by Facebook (Meta) raises concerns over potential privacy compromises due to data collection and sharing policies.

2. Telegram

Security Features:

Telegram offers cloud-based encryption, which means messages are encrypted during transmission.

The platform provides Secret Chats, which use end-to-end encryption, and self-destructing messages.

Two-factor authentication (2FA) is available for user accounts.

Limitations:

Regular chats in Telegram are not end-to-end encrypted, leaving them potentially vulnerable to interception by hackers or unauthorized access from the platform.

Cloud storage may be a security risk, as messages are stored in Telegram’s data centers, making them susceptible to hacking.

User metadata can still be accessed by the Telegram servers, creating privacy concerns

Drawbacks of Existing system:

The existing secure messaging systems, although providing some level of security, have notable drawbacks that undermine the overall privacy and security of user communications.

One of the major limitations is the metadata exposure, as many platforms store user data such as sender/receiver information, timestamps, and message frequency, which can be accessed by third parties or used for surveillance. Even with encryption in place for message content, this metadata remains vulnerable, compromising the full privacy of communications.

Additionally, most platforms rely on centralized storage for messages and media, which increases the risk of data breaches and unauthorized access if the server is compromised. While some apps offer end-to-end encryption, such as WhatsApp and Signal, the lack of encryption for metadata and other types of communication (e.g., voice or video calls) leaves gaps in security.

These combined drawbacks highlight the need for a more secure, privacy-conscious, and user-friendly messaging solution.

3.1 PROPOSED SYSTEM

The proposed system is a secure end-to-end chat application designed to address the vulnerabilities of existing systems by implementing advanced encryption and robust security measures. It uses the AES cryptographic library to encrypt messages and files, ensuring that data remains confidential during transmission and storage. User authentication is strengthened with secure password hashing, token-based authentication (JWT), and optional multi-factor authentication (MFA). The application supports real-time communication via WebSocket, enabling seamless and secure message exchange between users. It also securely stores past messages and shared files in encrypted format, allowing users to access their history without compromising security. By integrating these features, the system offers a reliable, user-friendly platform that prioritizes data privacy and protection.

1. Introduction

The chat application provides secure, real-time messaging with end-to-end encryption (E2EE) using AES encryption for messages and files. It ensures data privacy, secure authentication, and real-time updates via WebSocket.

2. System Features

- User Authentication & Security.

- JWT-based Authentication: Secure login using JSON Web Tokens
- Password Hashing: Secure password storage using crypt.
- Multi-Factor Authentication (MFA) (Optional): Extra layer of security for login.

3.Real-Time Messaging

- WebSocket for Instant Messaging: Ensures real-time message delivery.
- Message Encryption: AES encryption for secure storage and transmission.
- Unread Message Count: Notifications for unread messages in the sidebar.

4.File Sharing & Encryption

- Secure File Uploads: Encrypt files before storage.
- AES-Encrypted Storage: Store files securely in MongoDB or cloud storage.

5.Notifications

- Notification Counter on Sidebar: Unread message count per user.
- Push Notifications (Optional): Real-time alerts for new messages.

6.User Management

- User Profiles: Basic details with profile pictures.
- Online/Offline Status: Show user availability.

7.Message Storage

- MongoDB Database: Store encrypted messages and files.
- Message History: Retrieve old conversations secure.

CHAPTER 4
SOFTWARE REQUIREMENTS SPECIFICATION

4.SOFTWARE REQUIREMENTS SPECIFICATIONS

4.1 FUNCTIONAL REUIREMENTS

Secure Connect Messenger must provide end-to-end encrypted messaging to ensure secure communication between users. It should support real-time text messaging, voice and video calls, and file sharing with encryption. User authentication must include multi-factor authentication (MFA) to prevent unauthorized access. The application should allow secure group chats, message self-destruction, and offline message storage with automatic synchronization upon reconnection. Additionally, it must offer contact verification methods to prevent identity spoofing and include admin-controlled access for enterprise users. A secure login/logout mechanism, activity monitoring, and compliance with data protection regulations are also essential functional requirements.

Secure Connect Messenger is poised to become a leading solution for secure and private communication, addressing growing concerns over data security and unauthorized access. With its robust end-to-end encryption, multi-factor authentication, and advanced security features, the platform ensures confidentiality and integrity in messaging, voice calls, and file sharing. Its user-friendly interface, combined with enterprise-level controls, makes it an ideal choice for businesses, professionals, and individuals seeking a trustworthy communication tool. As cybersecurity threats continue to evolve, Secure Connect Messenger is expected to gain widespread adoption as a reliable and secure messaging platform.

Final Prediction

The system should be able to classify the input lung sound recording into different respiratory disease categories based on the trained deep learning model's predictions. The system should provide performance evaluation metrics, such as recall, precision, accuracy, and F1 score, to assess the model's effectiveness and reliability for the given input. The system should have a user-friendly interface that displays the predicted disease category and relevant performance metrics to healthcare professionals or users.

4.2. NON-FUNCTIONAL REQUIREMENTS

A system may be required to represent the user with a display of the number of records in a database which is a functional requirement. How up to date this number needs to be is a non– functional requirement.

Usability

The user interface should be intuitive and user-friendly, allowing healthcare professionals or users to easily navigate and interact with the system. The system should provide clear and understandable output, including the predicted disease category and relevant performance metrics.

Performance

The system should provide real-time or near-real-time classification of respiratory diseases to support timely diagnosis and intervention. The system should be able to handle a high volume of audio data and concurrent requests without significant performance degradation.

Security

The system should implement robust security measures to protect patient data and ensure compliance with relevant data privacy regulations (e.g., HIPAA, GDPR).

Access to the system should be restricted and controlled through appropriate authentication and authorization mechanisms.

Portability

The system should be platform-independent and able to run on different operating systems and hardware configurations, ensuring wide compatibility and ease of deployment.

Maintainability

The system should be designed with modular and well-documented code to facilitate future updates, bug fixes, and enhancements. The system should support easy integration of new or updated deep learning models and audio preprocessing techniques.

Interoperability

The system should be able to seamlessly integrate with existing healthcare information systems, electronic medical records, and other relevant software or hardware components. The system should support standard data formats and protocols for data exchange and communication.

4.4 HARDWARE REQUIREMENTS

Any software application defines the most common requirements known as the physical computer resources or hardware, A hardware requirements list regularly come with hardware compatibility list (HCL), particularly in case of operating systems. An HCL contains tested, compatible, and sometimes incompatible hardware devices for a particular operating system application.

Audio Recording Devices:

Microphones or digital stethoscopes capable of capturing high-quality lung sound recordings. Compatibility with various audio formats (e.g., WAV, MP3, FLAC) and sampling rates.

High-performance servers or workstations with powerful CPUs and GPUs for efficient processing of audio data and running deep learning models.

RAM: Sufficient RAM (32 GB or higher) to handle large audio datasets and model training.

Peripherals: Headphones or speakers for audio playback and analysis. External storage devices (USB drives, portable hard drives) for data transfer or backup.

SSD: For fast data access and retrieval.

4.3 SOFTWARE REQUIREMENTS

Software requirements involve defining software resource requirements that need to be installed on a computer to provide optimal functioning of an application. These requirements are usually not included in the software installation package and required to be installed separately before the software is install.

CPU: Dual-core processor (Intel i3/AMD Ryzen 3)

RAM: 2GB

Storage: 10GB SSD (for MongoDB and logs)

OS: Linux (Ubuntu 20.04+), Windows Server, or macOS

Database: MongoDB (3.6+ recommended)

Network: Stable internet connection with 1 Mbps+ upload speed

4.4 SUMMARY

In this chapter we discussed about the system requirements like what hardware is required and what are the software requirements one must have to know to use our model along with functional and non-functional requirements.

CHAPTER-5

SYSTEM DESIGN

5. SYSTEM DESIGN

System design is an essential step in the software development life cycle, which involves designing the system architecture, modules, components, and interfaces to satisfy the specified requirements. The system design in a project is critical to ensure that the system meets the specified requirements, is scalable, and is maintainable in the long run

SYSTEM ARCHITECTURE

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. It shows the connections between the various components of the system and indicates what functions are to be performed. The below figure shows system architecture of our project. This depicts the model building process.

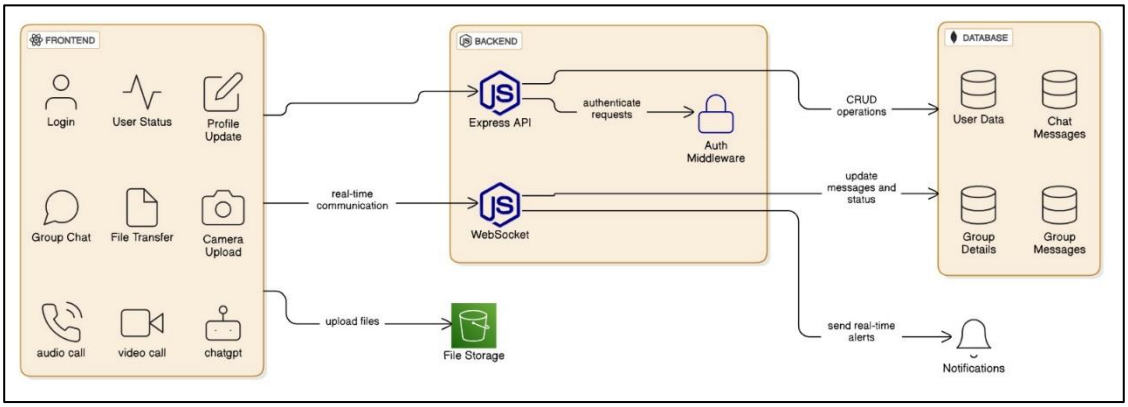


Fig 5.1 System Architecture

5.2 MODULES

User Authentication & Access Control Module

This module ensures that only authorized users can access the chat platform. It supports multi-factor authentication (MFA), OAuth 2.0, and JWT-based session management. During user registration, it verifies the identity via email/SMS OTP verification. It also handles user permissions, preventing unauthorized access to sensitive information. Additionally, biometric authentication (fingerprint/Face ID) can be integrated for enhanced security.

End-to-End Encryption (E2EE) Module

Security is a top priority, and this module ensures that messages remain encrypted from sender to receiver. Using AES-256 for symmetric encryption and RSA-4096 for asymmetric encryption, it encrypts all messages before transmission. A secure key exchange mechanism (Diffie-Hellman or Elliptic Curve Cryptography) ensures that only the intended recipient can decrypt messages. Even the chat server cannot read the messages, following a zero-knowledge encryption approach.

Real-Time Messaging Module

This module enables seamless real-time communication using WebSocket, MQTT, or Socket.io. It ensures low-latency message delivery with features like message acknowledgment, retries, and offline storage. The system queues messages using Kafka or RabbitMQ, so that even if a user is offline, the message is delivered when they reconnect. This module also supports read receipts, typing indicators, and presence status updates.

Secure File & Media Sharing Module

To allow users to send images, videos, or documents securely, this module encrypts all files before uploading them to a secure cloud storage (AWS S3, Firebase, or IPFS for decentralized storage). It also provides expiry-based file deletion, ensuring that sensitive media doesn't persist indefinitely. Additionally, this module scans files for malware before processing them.

User Profile & Contact Management Module

Users can manage their profiles, update personal information, and control privacy settings (who can see their profile, last seen, etc.). The contact management feature allows users to add/block/report contacts and synchronize contacts securely without exposing their address book to the server. This module also supports user status updates and profile pictures.

Notifications & Alerts Module

To keep users engaged, this module sends push notifications (Firebase Cloud Messaging for Android/iOS, Web Push for browsers) when a new message arrives. It ensures end-to-end encrypted notifications where message content is not exposed in the notification preview. Users can customize notification settings for different contacts or group chats.

AI-Powered Moderation & Spam Filtering Module

To maintain a safe chat environment, this module uses AI and machine learning to detect and block spam, phishing links, and abusive content. It flags suspicious messages and enables user-based reporting of inappropriate content. For enterprise chat apps, this module can include compliance monitoring for corporate regulations (GDPR, HIPAA, etc.)

5.1 UML DIAGRAMS USED IN DESIGN

UML is a standardized modelling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems. UML is used for describing the system architecture in detail using the blueprint. The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software project.

5.3 USE CASE DIAGRAM

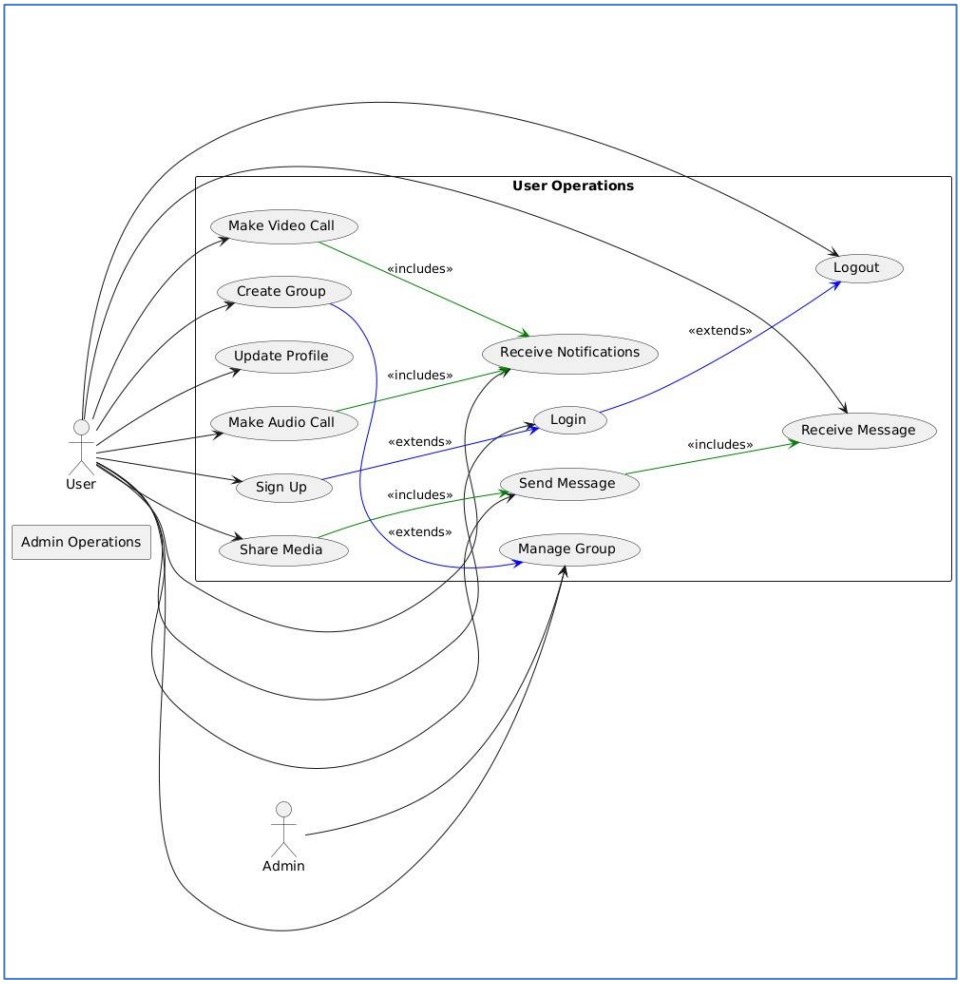


Fig 5.3.1 Use case Diagram

Key Actors:

Registered User: A user who has successfully created an account and logged in.

Guest User (Optional): A user who can access limited features without creating an account.

System Administrator: A user with administrative privileges to manage the application.

Use Cases:

Register User:

- Create a new user account.
- Set up profile information (username, display name, etc.).
- Choose a secure password.

Login User:

- Authenticate user credentials.
- Initiate a secure session.

Send Message:

- Compose and encrypt a message.
- Send the message to another registered user.

Receive Message:

- Receive and decrypt incoming messages.
- Display received messages in the chat interface.

View Chat History:

- Access and view past conversations.

Manage Contacts:

- Add, remove, and block contacts.
- View contact information.

Change Settings:

- Modify user profile settings (e.g., display name, profile picture).
- Adjust notification preferences.

Relationships:

- Include: Some use cases may include smaller, more specific use cases. For example,
- "Send Message" might include "Encrypt Message" and "Send Encrypted Message."
- Extend: Optional extensions to use cases can be added, such as "Send File" as an extension of "Send Message."

Usage:

The use case diagram helps stakeholders visualize the interactions between users and the system. It aids in understanding the system's functionalities and requirements, facilitating effective communication among project stakeholders.

Example Scenarios:

A user logs into the secure chat app, sends an encrypted message to a friend, and later deletes it using the self-destruct feature

5.1.1 CLASS DIAGRAM

Classes and Relationships

User (Represents registered users of the app)

Attributes: userID, username, email, phone, passwordHash, publicKey

Methods: register (), login(), updateProfile(), encryptMessage()

Relationships:

One-to-Many with Message (A user can send/receive multiple messages)

One-to-Many with Group (A user can be in multiple groups)

Message (Stores messages with encryption applied)

Attributes: messageID, content, timestamp, senderID, receiverID, encryptedFlag

Methods: encrypt(), decrypt(), deleteMessage()

Relationships:

Many-to-One with User (Each message belongs to a sender and a receiver)

One-to-Many with Attachment (A message can have multiple attachments)

Group (Handles group chats and participants)

Attributes: groupID, groupName, createdAt

Methods: addUser(), removeUser(), sendGroupMessage()

Relationships:

Many-to-Many with User (Users can be in multiple groups)

One-to-Many with Message (A group chat contains many messages)

Attachment (Manages file sharing within chats)

Attributes: attachmentID, fileName, fileSize, fileType, encryptionKey

Methods: encryptFile(), decryptFile(), deleteFile()

Relationships:

Many-to-One with Message (Each attachment is linked to a message)

Notification (Sends real-time message alerts to users)

Attributes: notificationID, userID, message, timestamp

Methods: sendNotification(), markAsRead()

Relationships:

Many-to-One with User (Each notification is associated with a user)

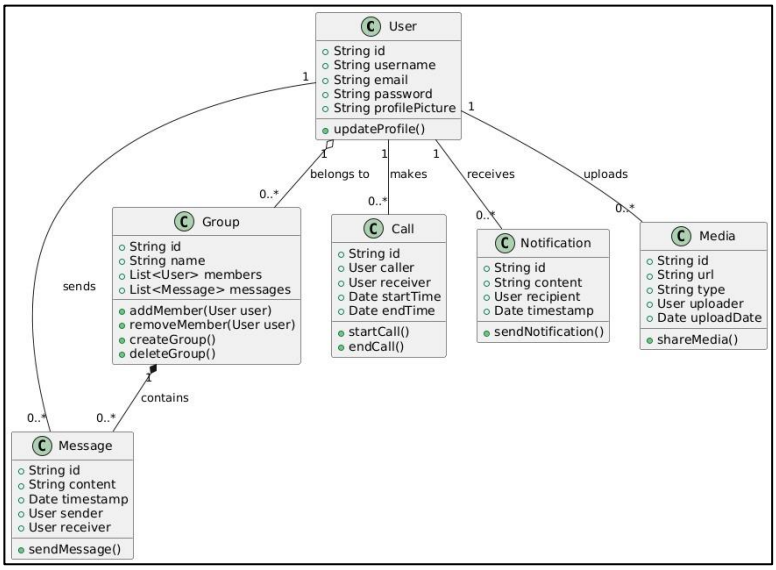


Fig 5.3.2 Class Diagram

5.2.2 SEQUENCE DIAGRAM:

These diagrams represent the interaction between user and the system. Sequence diagram explains about how the operations are carried out between elements as they interact over time, and they are organized according to object dimension (horizontal) and time dimension(vertical)

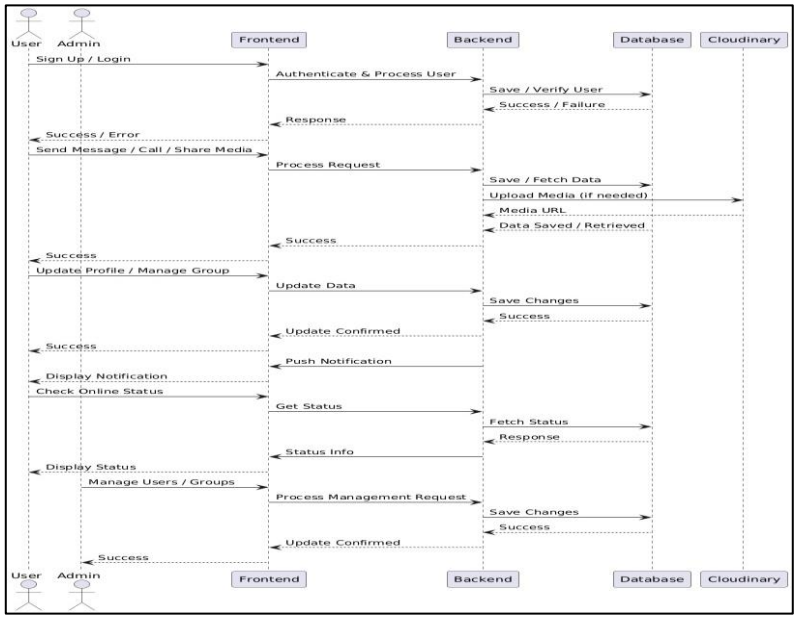


Fig 3.3 Sequence Diagram

Actors & Components:

User – The person using the chat application.

Client App – The mobile/desktop application used for chatting.

Server – The backend handling authentication and message relay.

Database – Stores user credentials, encryption keys, and chat history.

Sequence Flow

User Registration

- User enters details (username, password).
- The client encrypts the password and sends it to the server.
- The server verifies and stores the encrypted password in the database.

User Login

- User enters credentials.
- Client encrypts and sends credentials to the server.
- Server verifies and sends a session token.

Secure Message Exchange

- User A sends an encrypted message.
- The client encrypts the message using end-to-end encryption (E2EE).
- The encrypted message is sent to the server.

5.1.2 ACTIVITY DIAGRAM

A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioral diagram that illustrates the flow of activities through a system. This diagram depicts the workflow of the system by modeling the flow from one activity to another activity. Activity diagram also describes the parallel, branched, and concurrent flow of the system

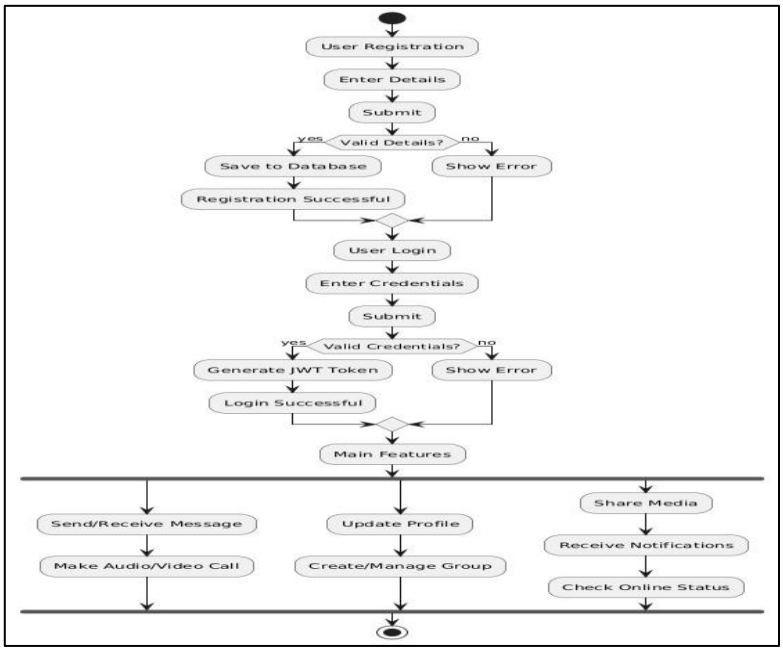


Fig 5.3.4 Activity Diagram

User Registration/Login

- User enters credentials.
- System verifies authentication.
- If authentication fails, retry or exit.

Chat Session Initialization

- User selects a contact.
- System establishes a secure connection.

Message Encryption & Sending

- User composes a message.
- Message is encrypted using an encryption algorithm.
- Message is transmitted to the recipient.

Message Decryption & Display

- The recipient's device decrypts the message.
- The decrypted message is displayed in the chat.

Logout Process

- User can log out securely.
- Session is terminated.

Sequence of Interactions:

- The user uploads file data through the user interface**.**

CHAPTER-6

SYSTEM IMPLEMENTATION

6. SYSTEM IMPLEMENTATION

System implementation is the process of putting the designed system into action or making it functional. It involves developing the system in a way that meets the requirements and objectives specified during the system analysis and design phase. It involves writing the code for the system based on the design specifications. In this chapter all the required libraries are explained briefly.

6.1 LIBRARIES

1.Server Requirements (Backend - Node.js, Express, MongoDB, WebSocket)

Minimum Requirements

- CPU: Dual-core processor (Intel i3/AMD Ryzen 3)
- RAM: 2GB
- Storage:10GB SSD (for MongoDB and logs)
- OS: Linux (Ubuntu 20.04+), Windows Server, or macOS
- Database: MongoDB (3.6+ recommended)
- Network: Stable internet connection with 1 Mbps+ upload speed

Recommended Requirements (Production-Ready)

- CPU: Quad-core processor (Intel i5/AMD Ryzen 5 or higher)
- RAM: 8GB+ (to handle concurrent users)
- Storage: 50GB SSD+ (for messages, files, and logs)
- OS: Linux (Ubuntu 22.04 LTS recommended)
- Database: MongoDB 6.x+ (with proper indexing and sharding if needed)
- WebSocket: Socket.io or native WebSocket implementation
- Security: TLS/SSL enabled for encrypted connections

2. Client Requirements (Frontend - React, Web App)

Minimum Requirements:

- CPU: Dual-core processor
- RAM: 2GB
- Browser: Chrome, Firefox, Edge (latest versions)
- OS: Windows, macOS, Linux, or Mobile (Android/iOS)
- Storage: 100MB (for caching and local storage)

Recommended Requirements:

- CPU: Quad-core processor
- RAM: 4GB+ (for smoother performance)
- Browser: Chrome/Edge (latest versions, optimized for React)
- Storage: 500MB (for cache and offline data).

6.2 SAMPLE SOURCE CODE

```
export const protectRoute = async (req, res, next) => {
  try {
    const token = req.cookies.jwt;

    if (!token) {
      return res.status(401).json({ message: "Unauthorized - No Token Provided" });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    if (!decoded) {
      return res.status(401).json({ message: "Unauthorized - Invalid Token" });
    }

    const user = await User.findById(decoded.userId).select("-password");

    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }

    req.user = user;

    next();
  } catch (error) {
    console.log("Error in protectRoute middleware: ", error.message);
    res.status(500).json({ message: "Internal server error" });
  }
};
```

Fig. 6.2.1 User Authentication using JWT code

```
  return (
    <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center">
      <div className="bg-white p-5 rounded-lg shadow-lg">
        <h2 className="text-lg font-medium mb-4">Update Group Profile</h2>
        <input
          type="text"
          placeholder="Enter group name"
          value={groupName}
          onChange={(e) => setGroupName(e.target.value)}
          className="input input-bordered w-full mb-4"
        />
        <input
          type="file"
          onChange={handleFileChange}
          className="input input-bordered w-full mb-4"
        />
        <div className="flex justify-end gap-2">
          <button onClick={onClose} className="btn btn-secondary">Cancel</button>
          <button onClick={handleUpdateProfile} className="btn btn-primary">Update</button>
        </div>
      </div>
    </div>
  );
};

export default UpdateProfileModal;
```

Fig. 6.2.2 Group Chat code

```
  return (
    <div className="call-container">
      <video ref={localVideoRef} autoPlay playsInline muted />
      <video ref={remoteVideoRef} autoPlay playsInline />
      {isReceivingCall && !callAccepted && (
        <div className="fixed inset-0 flex items-center justify-center bg-black/50 z-50">
          <div className="bg-white p-5 rounded-lg shadow-lg text-center">
            <h2 className="text-lg font-semibold">{authUser.fullName || "Unknown"} is calling...</h2>
            <div className="flex justify-center gap-4 mt-4">
              <button onClick={acceptCall} className="bg-green-500 text-white px-4 py-2 rounded-lg">
                Accept
              </button>
              <button onClick={handleRejectCall} className="bg-red-500 text-white px-4 py-2 rounded-lg">
                Reject
              </button>
            </div>
          </div>
        </div>
      )}
      {callAccepted && <button onClick={handleCallEnded}>End Call</button>}
    </div>
  );
};

export default CallContainer;
```

Fig. 6.2.3 Video call code

```
app.use(cors({
  origin: 'http://localhost:5173', // Replace with your frontend URL
  methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
  credentials: true, // Allow cookies to be sent
  allowedHeaders: 'Content-Type,Authorization'
}));
app.options('*', cors()); // Enable pre-flight for all routes
app.use(express.json({ limit: "500mb" })); // Increase JSON payload limit
app.use(express.urlencoded({ limit: "500mb", extended: true }));
app.use(cookieParser());

// Route setup
app.use("/api/auth", authRoutes);
app.use("/api/messages", messageRoutes);
app.use('/api/groups', groupRoutes);

// Serve static files in production
if (process.env.NODE_ENV === "production") {
  app.use(express.static(path.join(__dirname, "../frontend/dist")));

  app.get("*", (req, res) => {
    res.sendFile(path.join(__dirname, "../frontend", "dist", "index.html"));
  });
}

// Start the server
server.listen(PORT, () => {
  console.log("server is running on PORT:" + PORT);
  connectDB();
});
```

Fig. 6.2.4 Index code

```
return (
  <div className="call-container">
    <audio ref={localAudioRef} autoPlay playsInline muted />
    <audio ref={remoteAudioRef} autoPlay playsInline />
    {audioCallAccepted && (
      <div className="flex flex-col items-center justify-center h-full">
        <div className="flex items-center gap-4">
          <div className="relative">
            <h1>{authUser.fullName}</h1>
            <img
              src={authUser?.profilePic || "/avatar.png"}
              alt={authUser?.fullName}
              className="w-24 h-24 rounded-full"
            />
            <div className="absolute inset-0 flex items-center justify-center">
              <div className="animate-ping w-24 h-24 rounded-full bg-blue-500 opacity-75"></div>
            </div>
          </div>
          <div className="relative">
          <h1>{selectedUser?.fullName}</h1>
            <img
              src={selectedUser?.profilePic || "/avatar.png"}
              alt={selectedUser?.fullName}
              className="w-24 h-24 rounded-full"
            />
            <div className="absolute inset-0 flex items-center justify-center">
              <div className="animate-ping w-24 h-24 rounded-full bg-green-500 opacity-75"></div>
            </div>
          </div>
        </div>
        <button onClick={handleCallEnded} className="mt-4 px-4 py-2 bg-red-500 text-white rounded">
          End Call
        </button>
      </div>
    )}
```

Fig. 6.2.5 Audio call code

```
import mongoose from "mongoose";

export const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGODB_URI);
    console.log(`MongoDB connected: ${conn.connection.host}`);
  } catch (error) {
    console.log("MongoDB connection error:", error);
  }
};
```

Fig. 6.2.6 Ds code

```
const userSchema = new mongoose.Schema(
  {
    email: {
      type: String,
      required: true,
      unique: true,
    },
    fullName: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
      minlength: 6,
    },
    profilePic: {
      type: String,
      default: "",
    },
    notificationCounts: {
      type: Map,
      of: Number,
      default: {},
    },
  },
  { timestamps: true }
);
```

Fig. 6.2.7 User code

```
export const encryptFile = (fileBuffer) => {
  if (!fileBuffer) return null;
  try {
    const base64String = fileBuffer.toString("base64");
    return aes256.encrypt(secret_key, base64String);
  } catch (error) {
    console.error("❌ Error encrypting file:", error.message);
    return null;
  }
};

// ✅ Decrypt files (returns Buffer)
export const decryptFile = (encryptedFile) => {
  if (!encryptedFile) return null;
  try {
    const decryptedBase64 = aes256.decrypt(secret_key, encryptedFile);
    return Buffer.from(decryptedBase64, "base64");
  } catch (error) {
    console.error("❌ Error decrypting file:", error.message);
    return null;
  }
};
```

Fig. 6.2.8 File Encryption code

```
// ✅ Encrypt text messages
export const to_Encrypt = (text) => {
  if (!text) return null; // Prevent encrypting empty values
  try {
    return aes256.encrypt(secret_key, text);
  } catch (error) {
    console.error("❌ Error encrypting text:", error.message);
    return null;
  }
};

// ✅ Decrypt text messages
export const to_Decrypt = (cipher) => {
  if (!cipher) return null;
  try {
    return aes256.decrypt(secret_key, cipher);
  } catch (error) {
    console.error("❌ Error decrypting text:", error.message);
    return cipher; // Return encrypted text if decryption fails
  }
};
```

Fig. 6.2.9 Message Encryption code

```
<Routes>
  <Route path="/" element={authUser ? <HomePage /> : <Navigate to="/login" />} />
  <Route path="/signup" element={!authUser ? <SignUpPage /> : <Navigate to="/" />} />
  <Route path="/login" element={!authUser ? <LoginPage /> : <Navigate to="/" />} />
  <Route path="/settings" element={authUser ? <SettingsPage /> : <Navigate to="/login" />} />
  <Route path="/profile" element={authUser ? <ProfilePage /> : <Navigate to="/login" />} />
  <Route path="*" element={<Navigate to="/" />} />
</Routes>
```

Fig. 6.2.10 Route code

```
// Handle file download
const handleDownload = async (fileUrl, fileName) => {
  try {
    const response = await fetch(fileUrl);
    if (!response.ok) throw new Error("Failed to fetch file");

    const blob = await response.blob();
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = fileName || "download";
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
    window.URL.revokeObjectURL(url);
  } catch (error) {
    console.error("Error downloading file:", error);
    alert("Failed to download file. Please try again.");
  }
};
```

Fig. 6.2.11 File code

CHAPTER 7

SYSTEM TESTING

7. SYSTEM TESTING

System testing is the process of testing our software or application regarding its performance, technical requirements, any technical glitches, and whether it is meeting the user requirements effectively and efficiently. Software testing not only focuses on finding faults but also helps in improving efficiency, accuracy, and usability.

7.1 INTRODUCTION

Software bugs and glitches are inevitable in any software application. The bugs can be architecture, functions, the code itself, etc. The more complex the application the more bugs it will have. Unlike hardware, software that does not suffer from corrosion, wear-and-tear generally will not change until upgrades. So, once the software is shipped the design defects or bugs will be buried in and remain until activation.

Software testing can be divided into two steps:

Verification: It checks whether the software correctly implements a specific function

Validation: It checks whether the software build is reaching the customer's requirements.

7.2 PURPOSE OF TESTING

Identifies defects early: Developing complex applications can leave room for errors.

Software testing identifies any issues and defects in the code so they can be fixed before the delivery of the software product.

Improves product quality: Software testing helps the product pass quality assurance and meet the criteria and specifications defined by the user.

Increase customer trust and satisfaction: Testing a product throughout its development lifecycle builds customer trust and satisfaction, as it provides visibility into the product's strong and weak points. By the time customers receive the product, it has been tried and tested multiple times and delivers on quality.

Detects security vulnerabilities: Insecure application code can leave vulnerabilities that attackers can exploit. Since most applications are online today, they can be a leading vector for cyber- attack and should be tested thoroughly during various stages of application development.

Helps with scalability: Scalability test helps in increasing workload such as traffic, data volume, and transaction counts. It can also identify the point where an application might stop functioning and the reasons behind it, which may include meeting or exceeding a certain threshold, such as the total number of concurrent app users.

7.3 TYPES OF TESTING

UNIT TESTING

Unit testing is a method of testing individual units or components of a software application. It is typically done by developers and is used to ensure that the individual units of the software are working as intended. Unit testing is usually automated and is designed for specific parts of the code, such as a particular function or method. Unit testing is done at the lowest level of the software development process, where individual units of code are tested in isolation.

INTEGRATION TESTING

Integration testing mainly focuses on how different units of an application interact with each other. It is used to resolve any issues that arise when different components of the software application are combined. Integration testing is typically done after unit testing and before functional testing.

FUNCTIONAL TESTING

The purpose of Functional tests is to test each function of the software application, by providing appropriate input, and verifying the output against the Functional requirements. Functional testing mainly involves black box testing and it is not concerned with the source code of the application. This testing checks the User Interface, APIs, Database, Security, Client/Server communication, and other functionality of the Application Under Test. The testing can be done either manually or using automation.

BETA TESTING

Beta Testing is performed by “real users” of the software application in “real environment” and it can be considered as a form of external user acceptance testing. It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test products in customer’s environment. Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.

ALPHA TESTING

Alpha Testing is a type of acceptance testing; performed to identify all possible issues and bugs before releasing the final product to the end users. Alpha testing is carried out by the testers who are internal employees of the organization. The main goal is to identify the tasks that a typical user might perform and test them. This testing comes under validation testing. It is a type of acceptance testing that is done before the product is released to customers. It is typically done by QA people.

SYSTEM TESTING

System testing is a level of software testing that evaluates a complete and integrated system to ensure that it meets specified requirements. It is a black-box testing technique where the system is tested as a whole, without knowledge of the internal workings of the system. The main objective of system testing is to identify and resolve any defects or issues that may exist in the system before it is deployed in the production environment. System testing can include functional, non-functional, regression, security, and performance testing to ensure the system meets all necessary requirements and performs as expected in various scenarios.

ACCEPTANCE TESTING

Acceptance testing is a type of testing that determines whether a software product meets the customer's requirements and expectations. It is also known as user acceptance testing (UAT), customer acceptance testing (CAT), or end-user testing. The purpose of acceptance testing is to ensure that the software product meets the functional, usability, performance, and compatibility requirements as specified by the customer.

7.4 TESTING METHODS

1.BLACK BOX TESTING

Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides input and observes the output generated by the system under test. This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues, and reliability issues. Black box testing is a powerful testing technique because it exercises a system end-to- end. Just like end-users “don’t care” how a system is coded or architected, and expect to receive an appropriate response to their requests, a tester can simulate user activity and see if the system

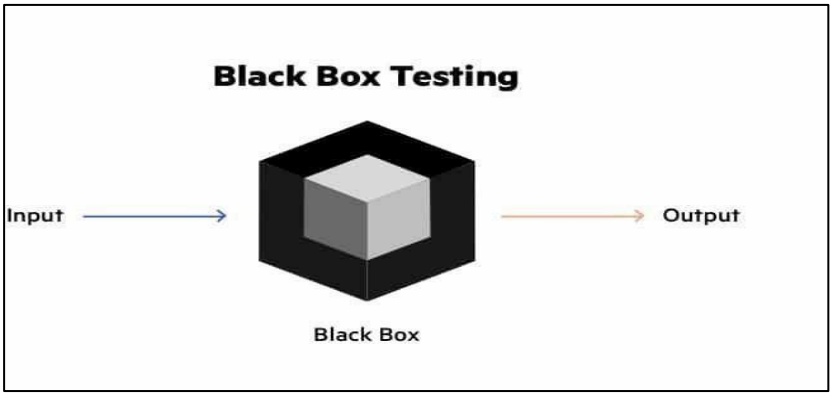


Fig 7.4.1 Black Box Testing

2.WHITE BOX TEST

White box testing is an approach that allows testers to inspect and verify the inner workings of a software system. Its code, infrastructure, and integrations with external systems. White box testing is an essential part of automated build processes in a modern Continuous Integration/Continuous Delivery (CI/CD) development pipeline. White box testing is often referenced in the context of Static Application Security Testing (SAST), an approach that checks source code or binaries automatically and provides feedback on bugs and possible vulnerabilities. White box testing provides inputs and examines outputs, considering the inner workings of the code.

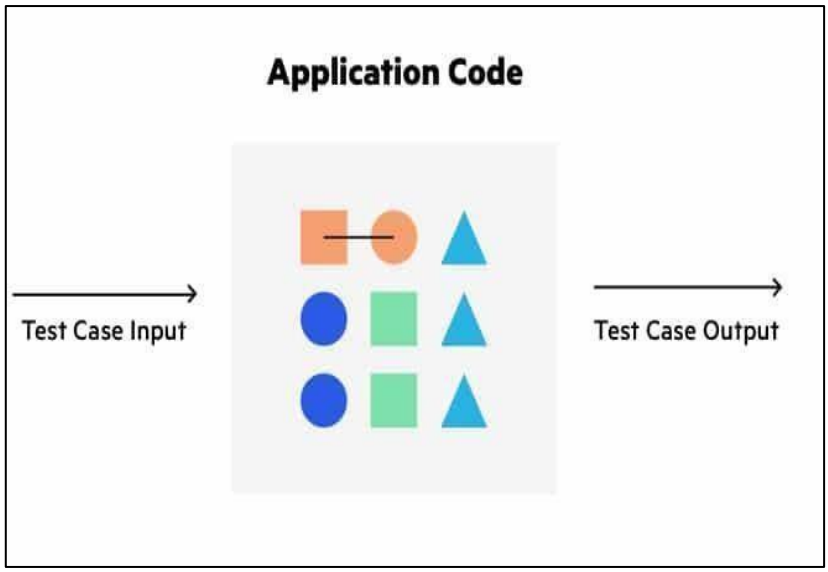


Fig 7.4.2 White Box Testing

7.5 TEST CASES

Test cases are a format of software testing to check if a particular application of the software is working or not. A test case consists of a certain set of conditions that need to be checked to test an application or software. If all the conditions are satisfied then the test case is passed if not the case is failed. The process of writing test cases can also help reveal errors or defects within the system.

System testing for a project involving the analysis of lung sound recordings for respiratory disease classification would aim to validate the entire system's functionality, performance, and reliability. Here are some examples of system testing test cases:

Authorization Testing:

Only the valid user with correct username and password are authenticated and logged in to know the patient details and detect.

7.5.1 TEST CASES

Test cases are a format of software testing to check if a particular application of the software is working or not. A test case consists of a certain set of conditions that need to be checked to test an application or software. If all the conditions are satisfied then the test case is passed if not the case is failed. The process of writing test cases can also help reveal errors or defects within the system.

Test Case 1:

In this scenario, we verify whether the user can successfully able to login or not.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is giving valid credentials to check if user can login or not.	User should login to system successfully.	Logged in to system	PASS

Table. 7.5.1 Test Case 1

Input:

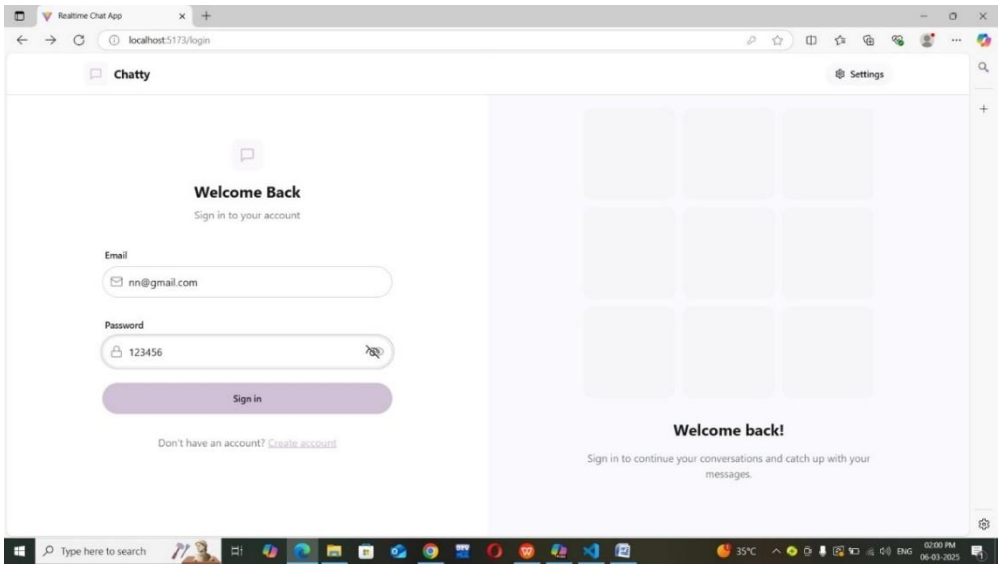


Fig. 7.5.1 Test Case 1 Input

Output:

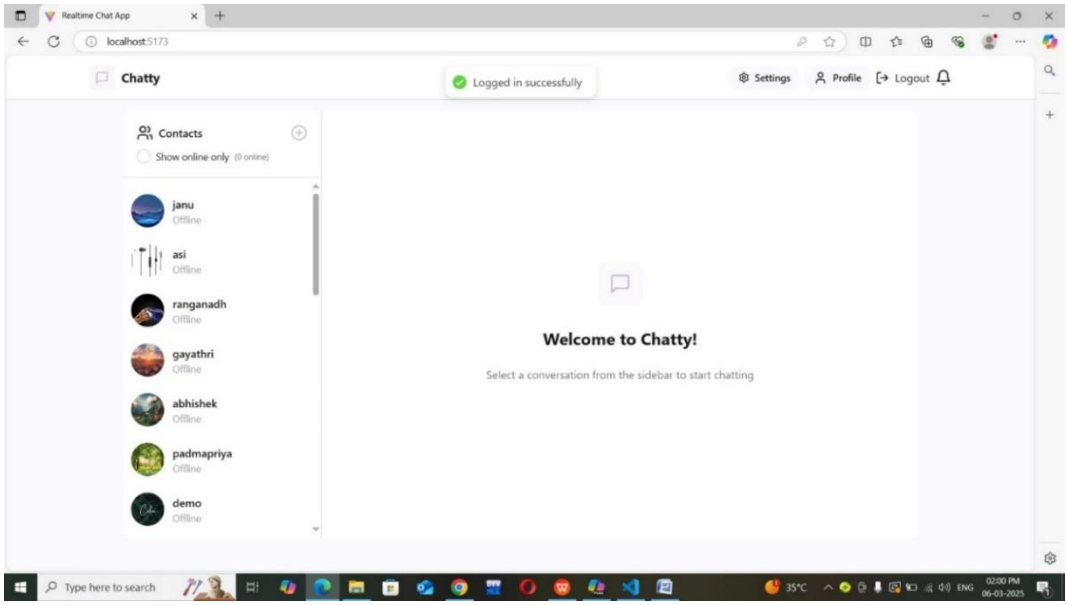


Fig. 7.5.2 Test Case 1 Output

Test Case 2:

In this scenario, we verify whether the application displays invalid credentials when user gives invalid credentials.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is giving invalid credentials.	It should display "invalid credentials" error.	Displayed "Invalid credentials" message	PASS

Table. 7.5.2 Test Case 2

Input:

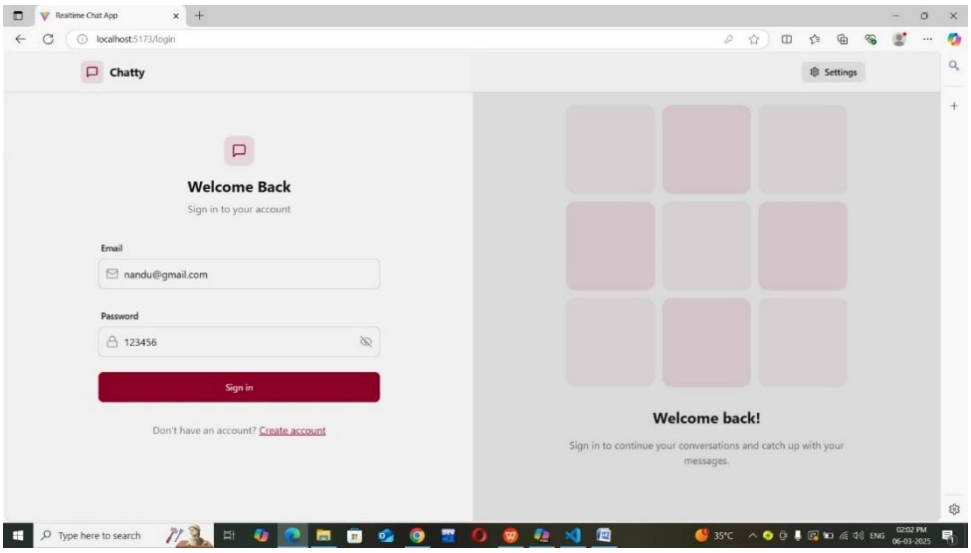


Fig. 7.5.3 Test Case 2 Input

Output:

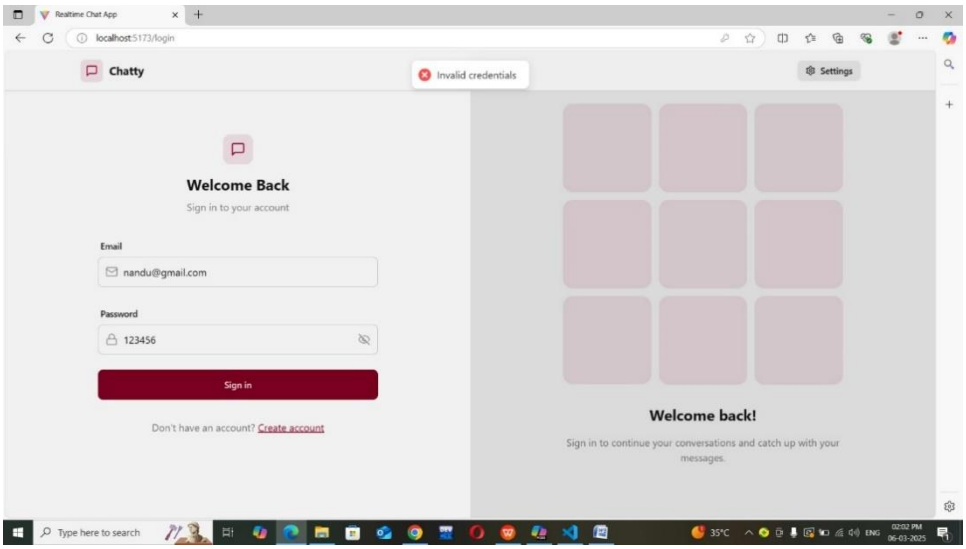


Fig. 7.5.4 Test Case 2 Output

Test Case 3:

In this scenario, we verify whether the Profile pic updated or not.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is Select the Image from the Local System.	Profile pic updated Successfully.	Profile pic updated Successfully.	PASS

Table. 7.5.3 Test Case 3

Input:

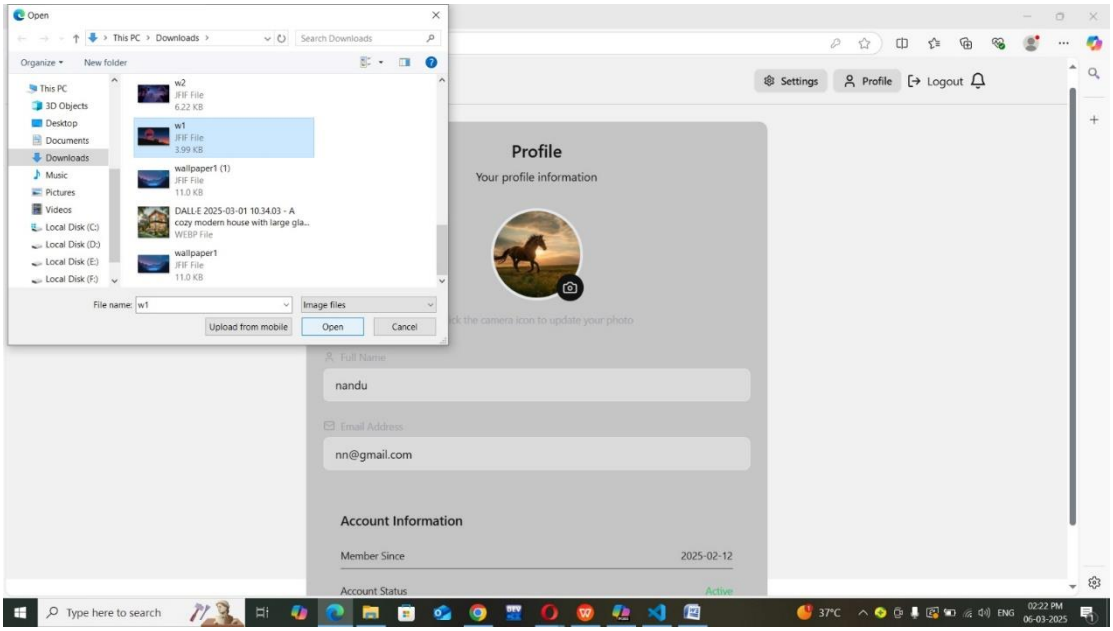


Fig. 7.5.5 Test Case 3 Input

Output:

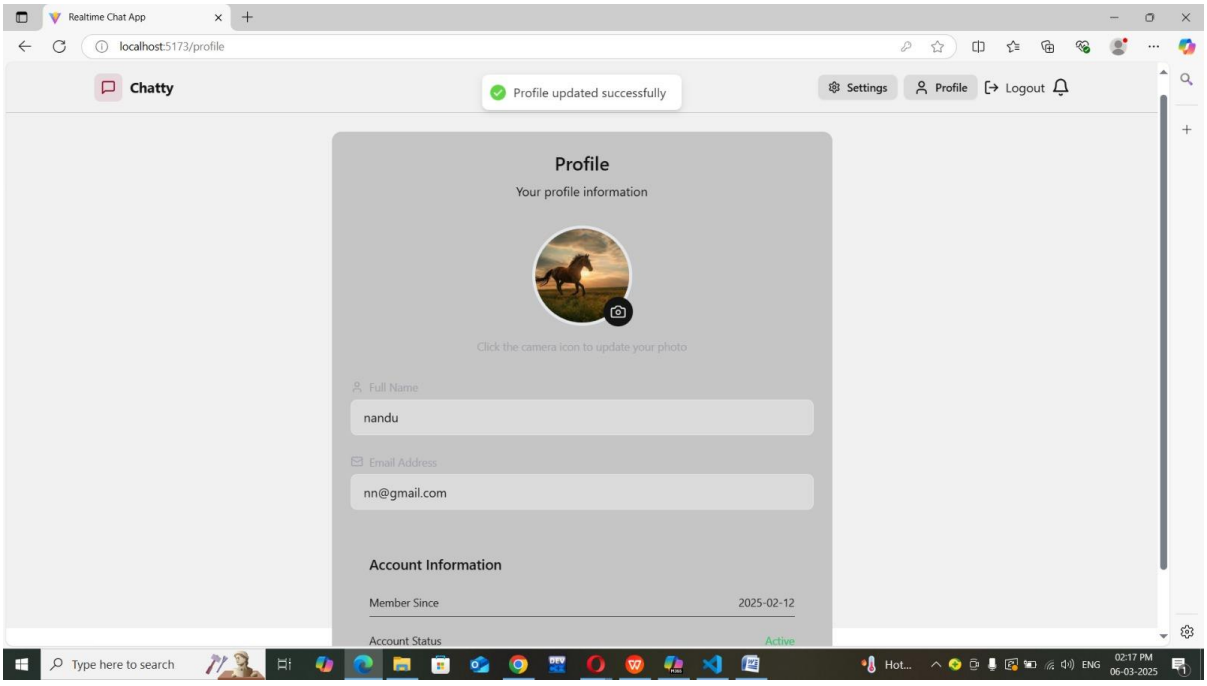


Fig. 7.5.6 Test Case 3 Output

Test Case 4:

In this scenario, we verify whether the Message send to One User to Another User or not.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is Sending Message	Message sent	Message sent.	PASS

Table. 7.5.4 Test Case 4

Input:

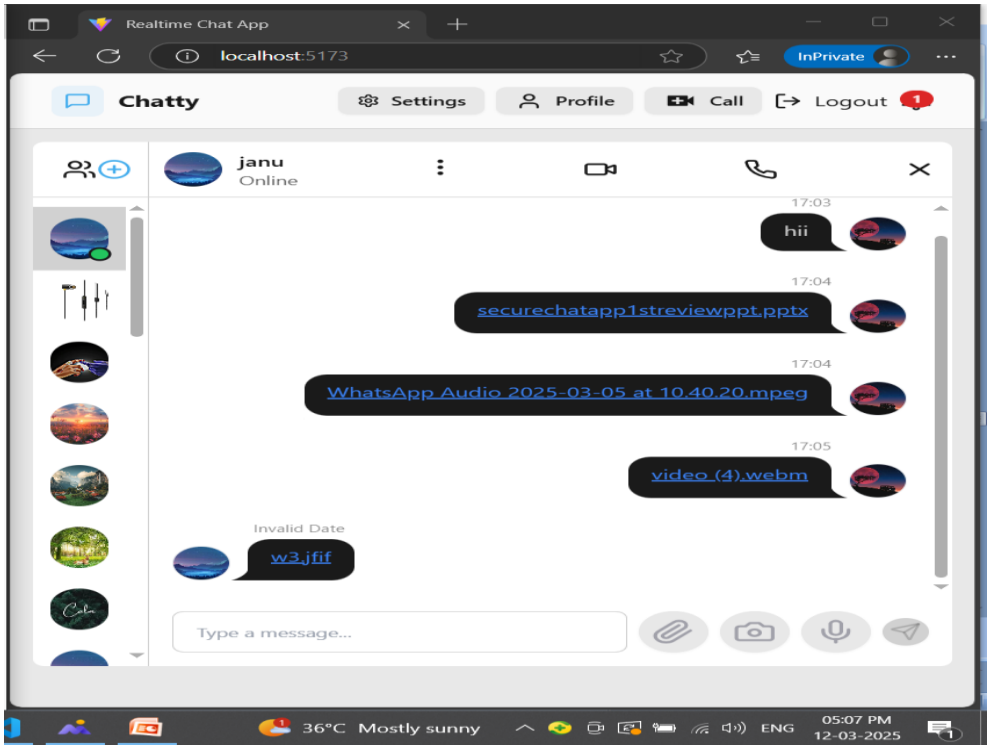


Fig. 7.5.7 Test Case 4 Input

Output:

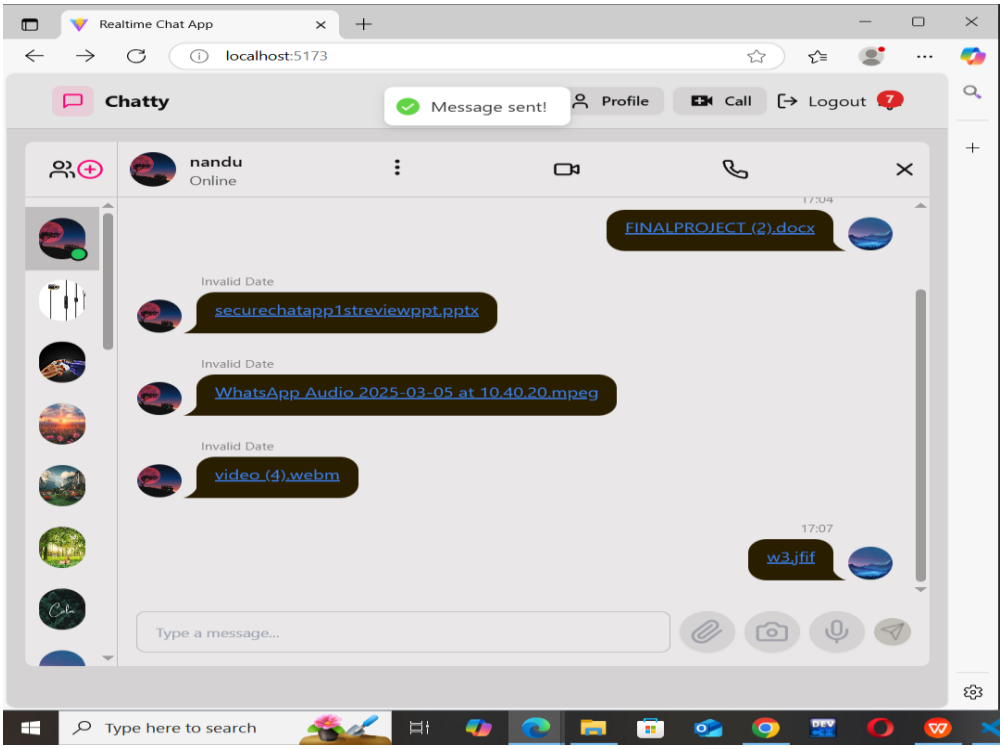


Fig. 7.5.8 Test Case 4 Output

Test case 5:

In this scenario, we verify whether the user can send image from camera

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is capturing the image from the camera	Image captured and send to user	Image captured and send to user.	PASS

Table. 7.5.5 Test Case 5

Input:

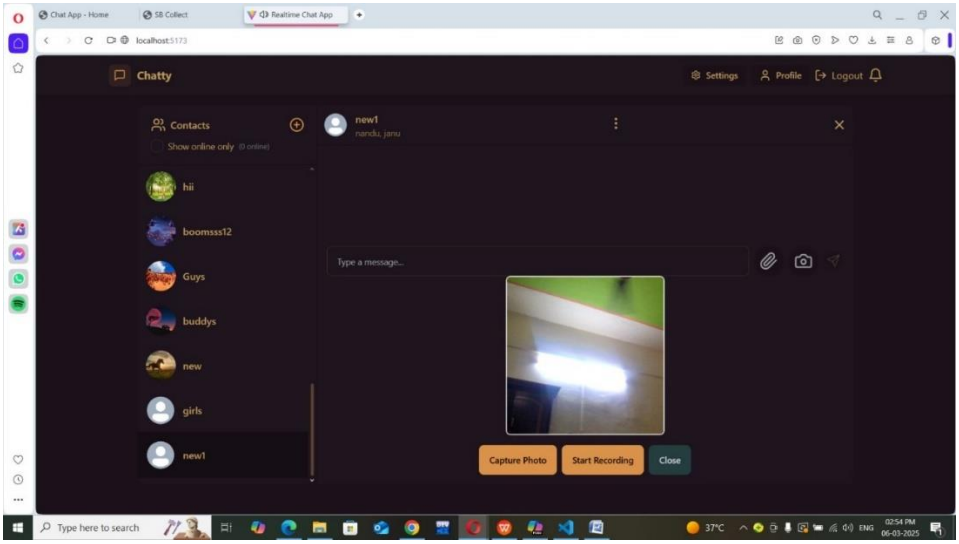


Fig. 7.5.9 Test Case5 Input

Output:

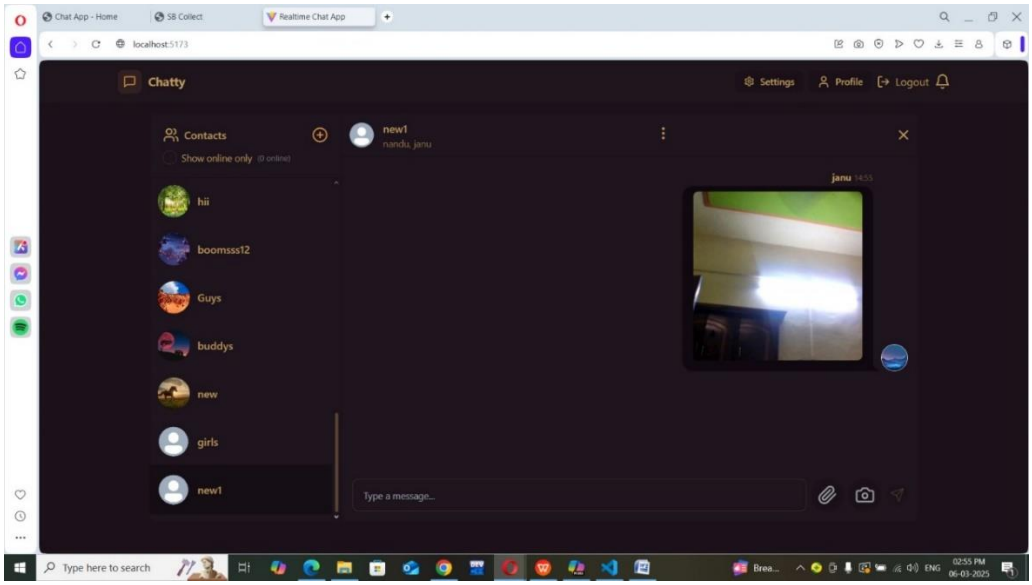


Fig. 7.5.10 Test Case 5 Output

Test case 6:
In this scenario, we verify whether the user can send video message or not

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Recording the video from the camera	Video is recorded and send to the user.	Video is recorded and send to the user.	PASS

Table. 7.5.6 Test Case 6

Input:

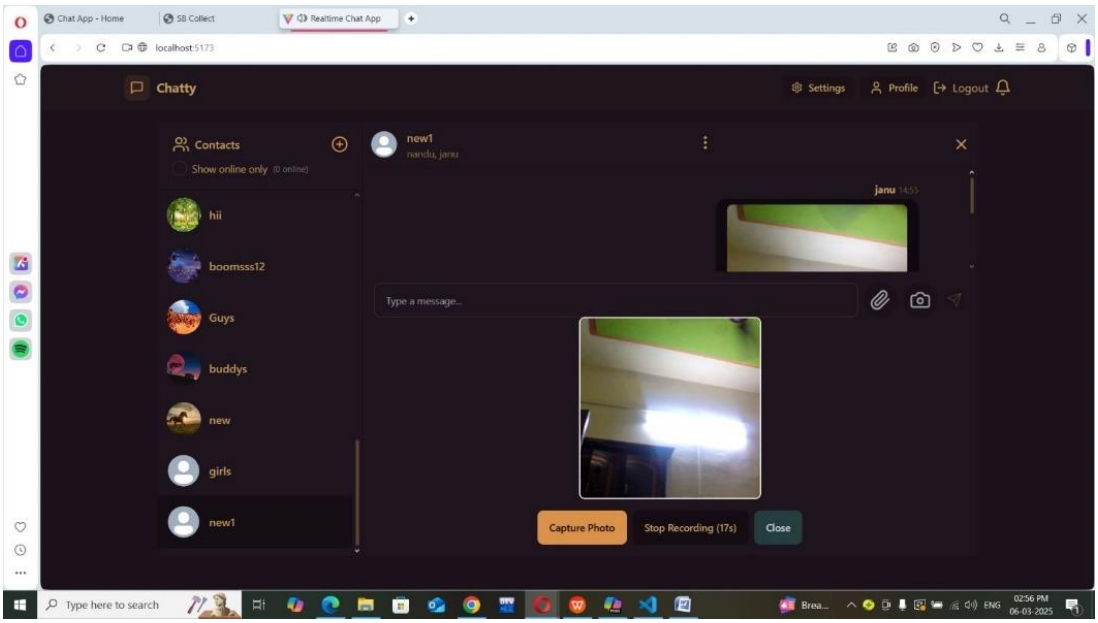


Fig. 7.5.11 Test Case 6 Input

Output:

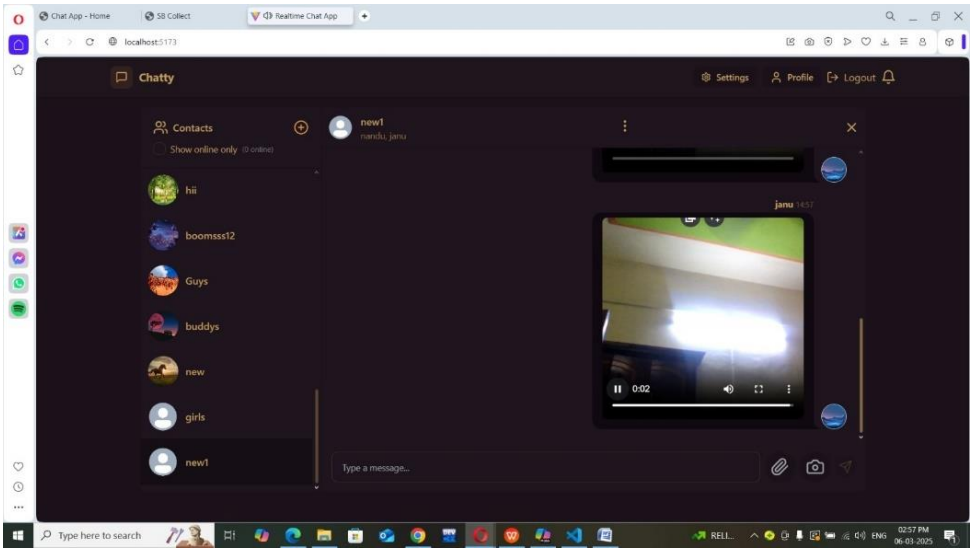


Fig. 7.5.12 Test Case 6 Output

Test case 7:

In this scenario, we verify whether the user can create a group or not.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is creating a group	Group is Created and Shows in contacts	Group is Created and Shows in contacts	PASS

Table. 7.5.7 Test Case 7

Input:

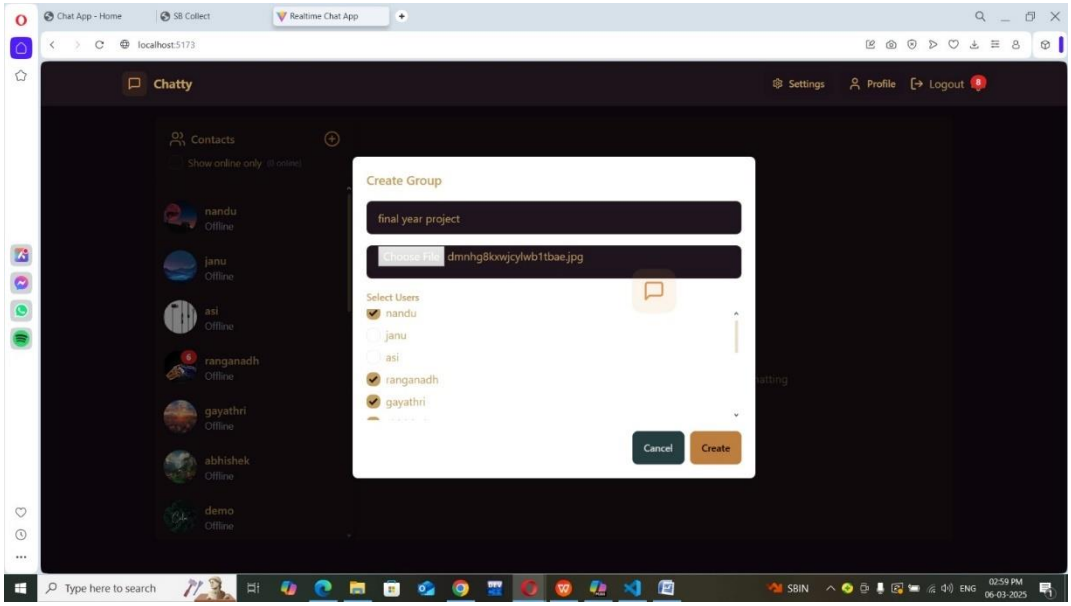


Fig. 7.5.13 Test Case 7 Input

Output:

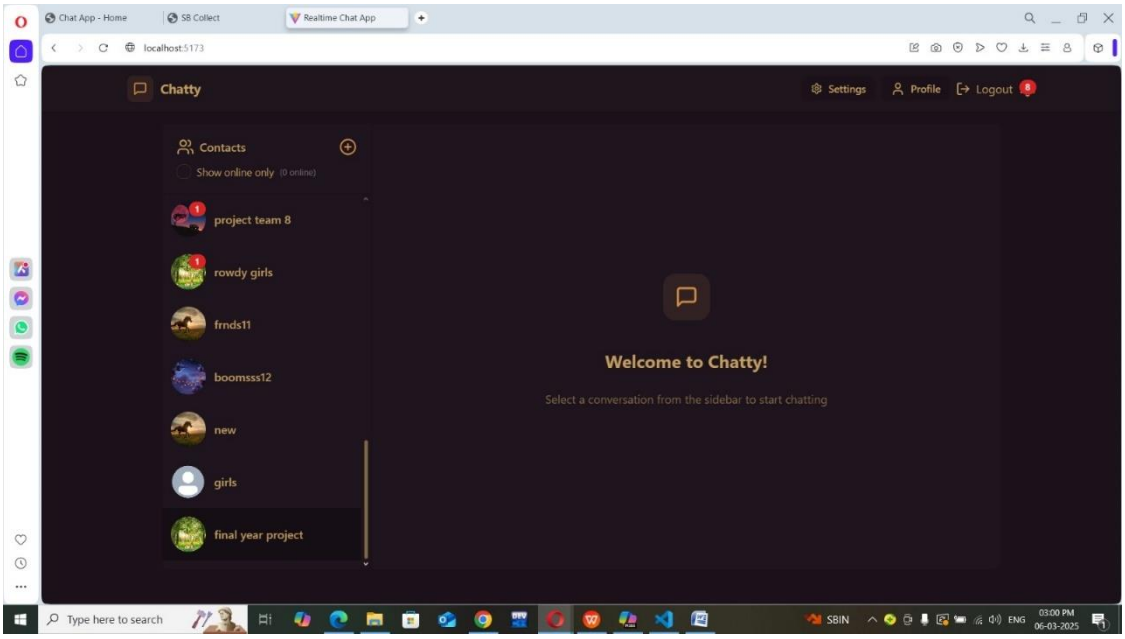


Fig. 7.5.14 Test Case 7 Output

Output2:

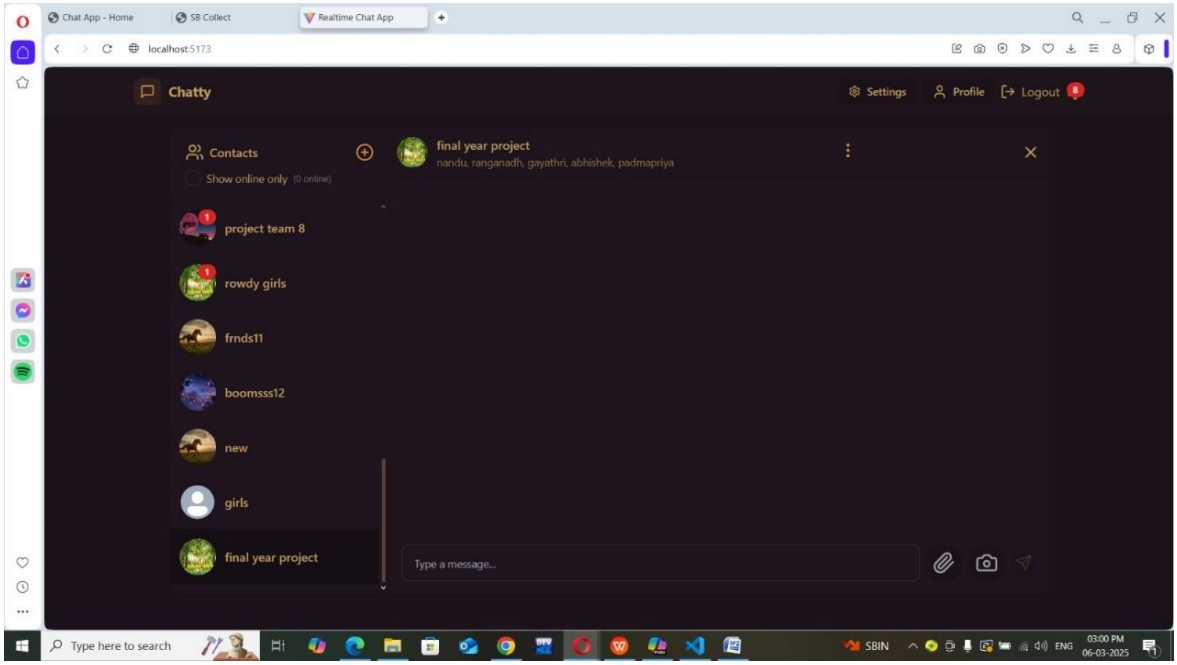


Fig. 7.5.14 Test Case 7 Output

Test case 8:

In this scenario, we verify whether the admin can send voice message or not.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is sending a voice message	Message sent	Message sent	PASS

Table. 7.5.8 Test Case 8

Input:

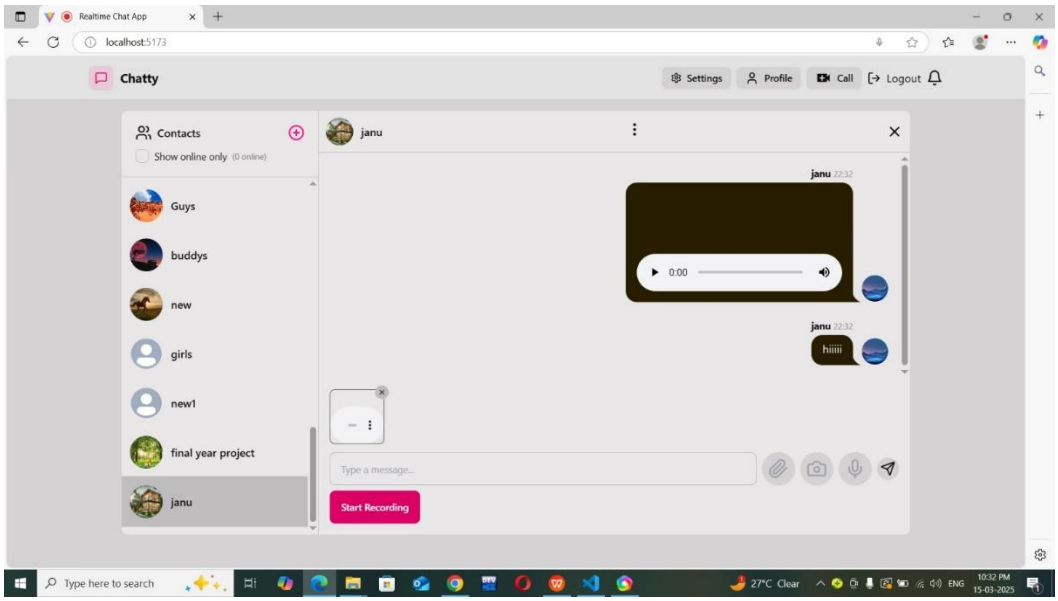


Fig. 7.5.15 Test Case 8 input

Output:

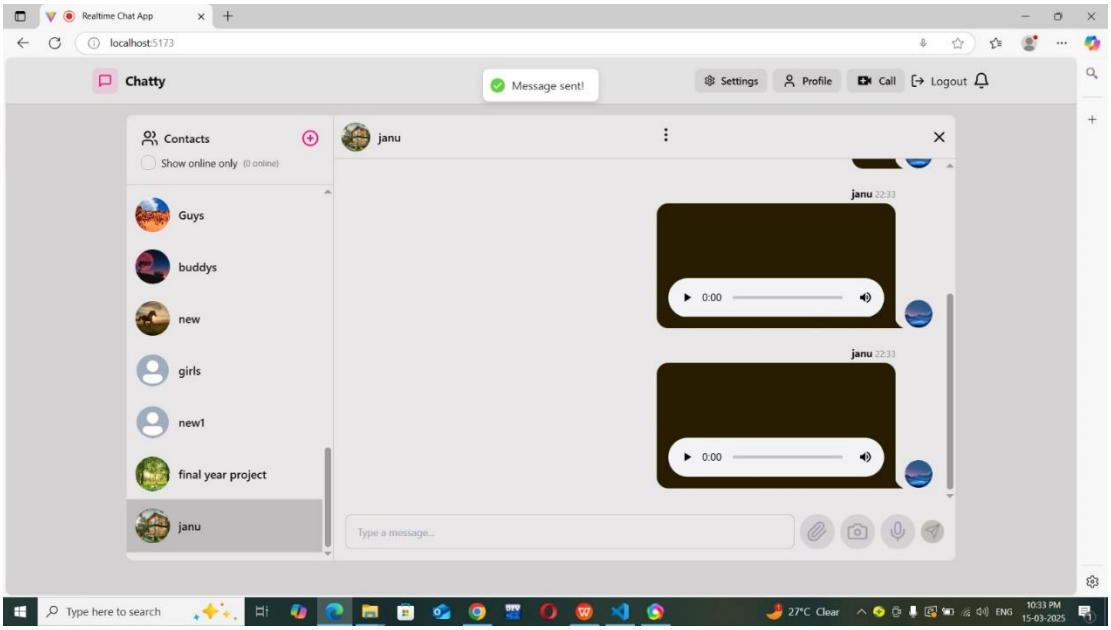


Fig. 7.5.16 Test Case 8 Output

Test case 9:

In this scenario, we verify whether the application is providing Notification of Messages or not.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is calling a person videocall	Its connect to another user	Its connect to another user	PASS

Table. 7.5.9 Test Case 9

Input:

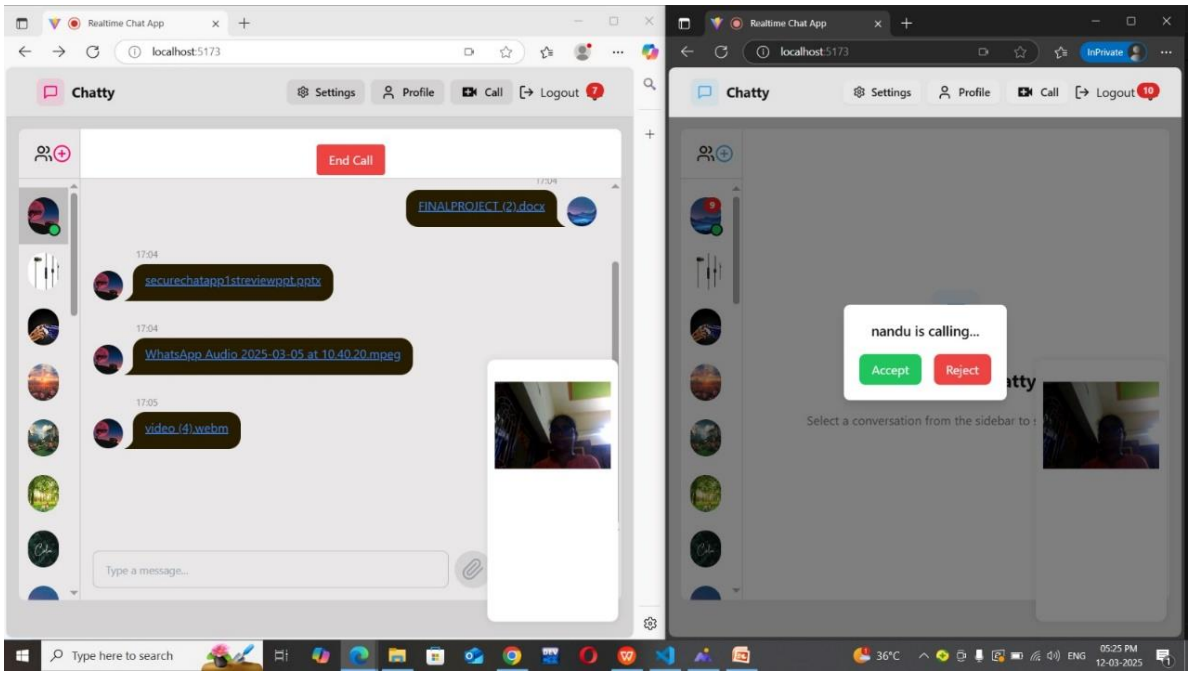


Fig 7.5.17 Test Case 9 Input

Output:

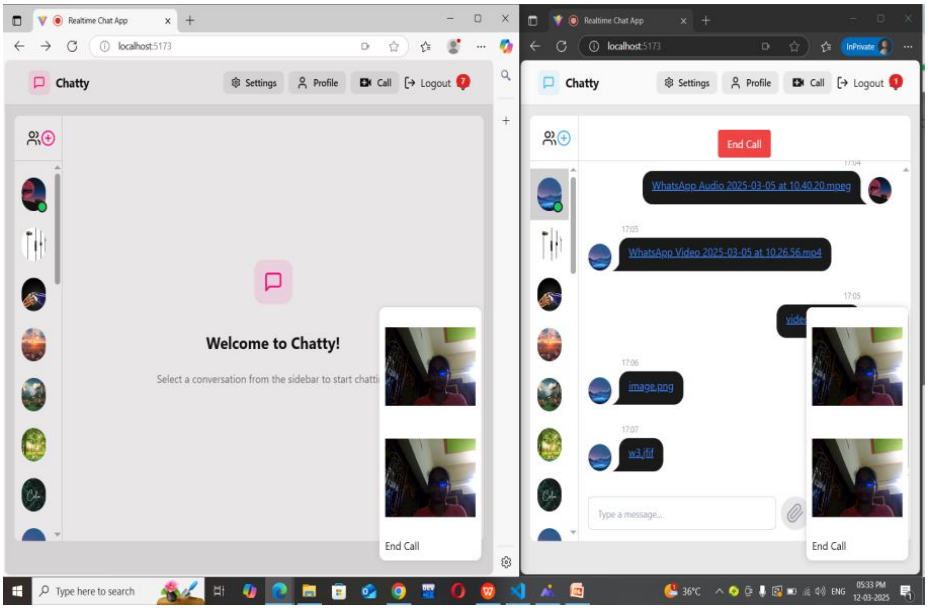


Fig 7.5.18 Testcase 9 Output

Test case 10:

In this scenario, we verify whether the application is providing Notification of Messages or not.

S.NO	INPUT	EXPECTED OUTPUT	OBSERVED OUTPUT	STATUS
1.	Input is calling a person audio call	It connects to another user	It connects to another user	PASS

Table. 7.5.10 Test Case 10

Input:

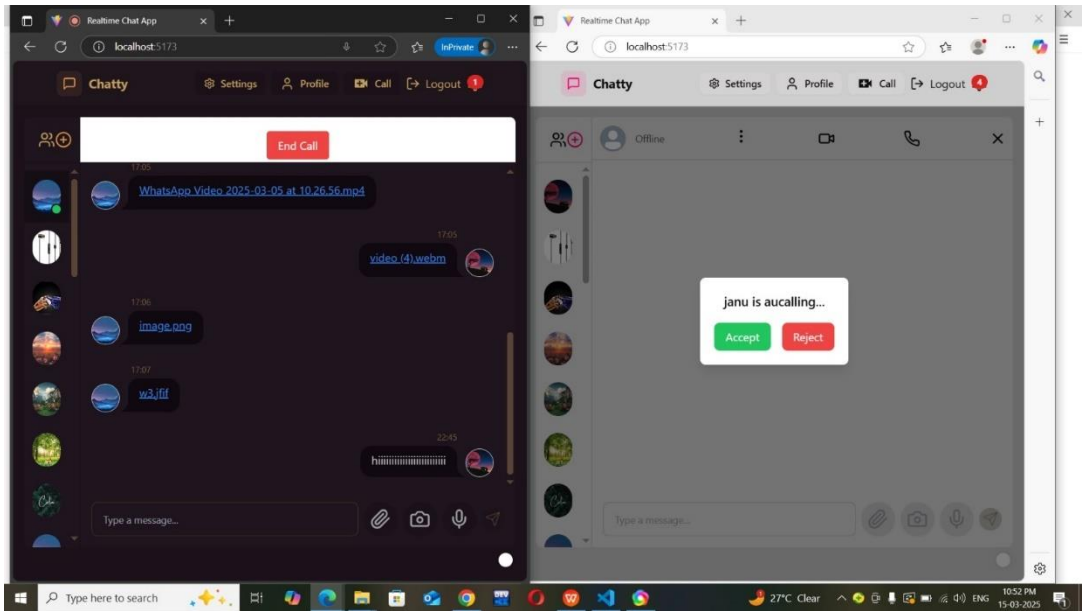


Fig. 7.5.19 Testcase 10 Input

Output:

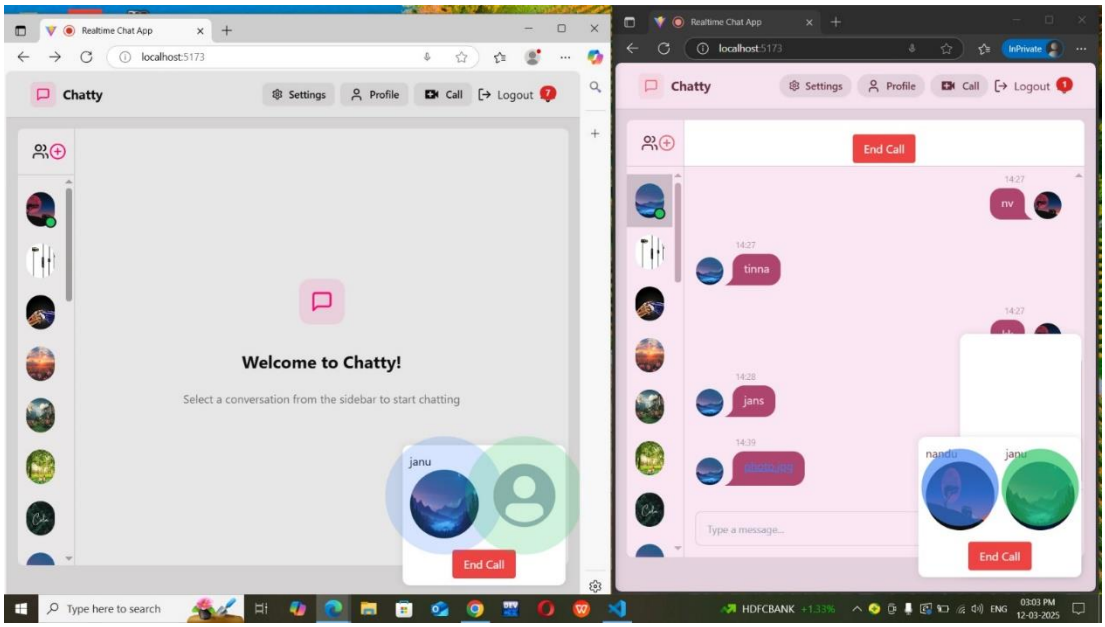


Fig 7.5.20 Test case 10 Output

CHAPTER 8

SYSTEM RESULTS

8.SYSTEM RESULTS

System results are the outcome of the whole process of software testing life cycle. The results thus produced, offer an insight into the deliverables of a software project, significant in representing the status of the project to the stakeholders. The analysis of system results is an important part of system evaluation, as it provides insight into the effectiveness and efficiency of the system and helps to identify areas for improvement.

8.1 User Interface

The user interface of the **SECURE CONNECT MESSENGER** system features a straightforward design with options for both home access and login functionalities.

- Home
- Login

The home option provides immediate access to the system's features, while **the login** option allows registered users to securely access personalized settings and data. This intuitive interface ensures ease of navigation and enhances user experience for Communication.

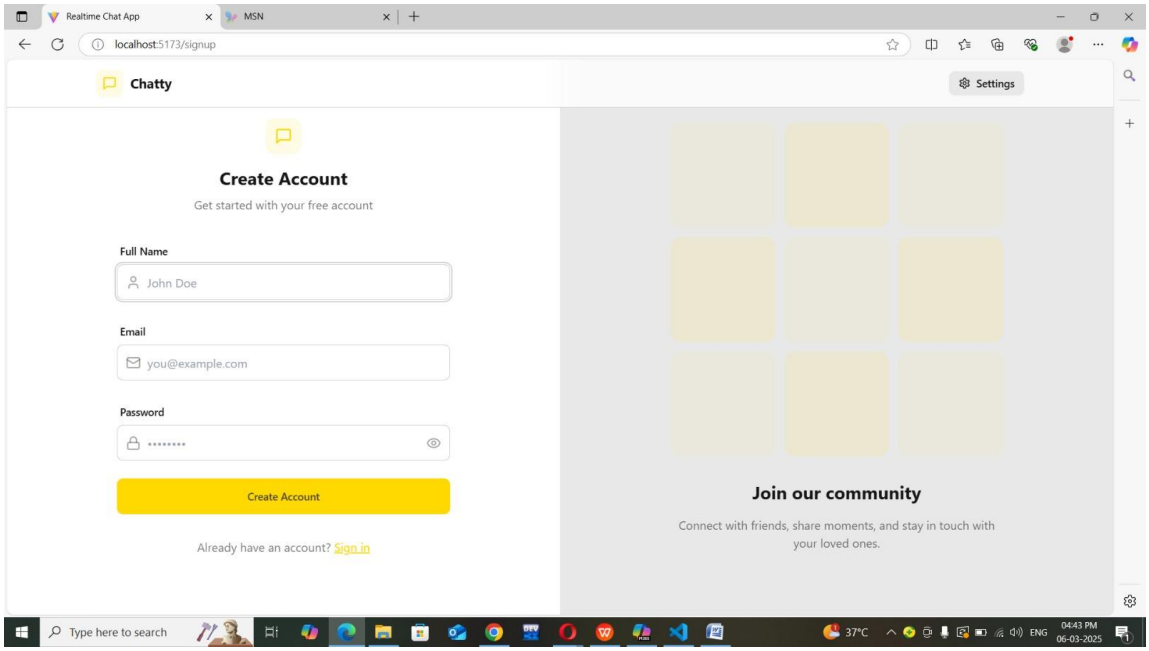


Fig. 8.1 User Account Creating

8.2 User Login Page

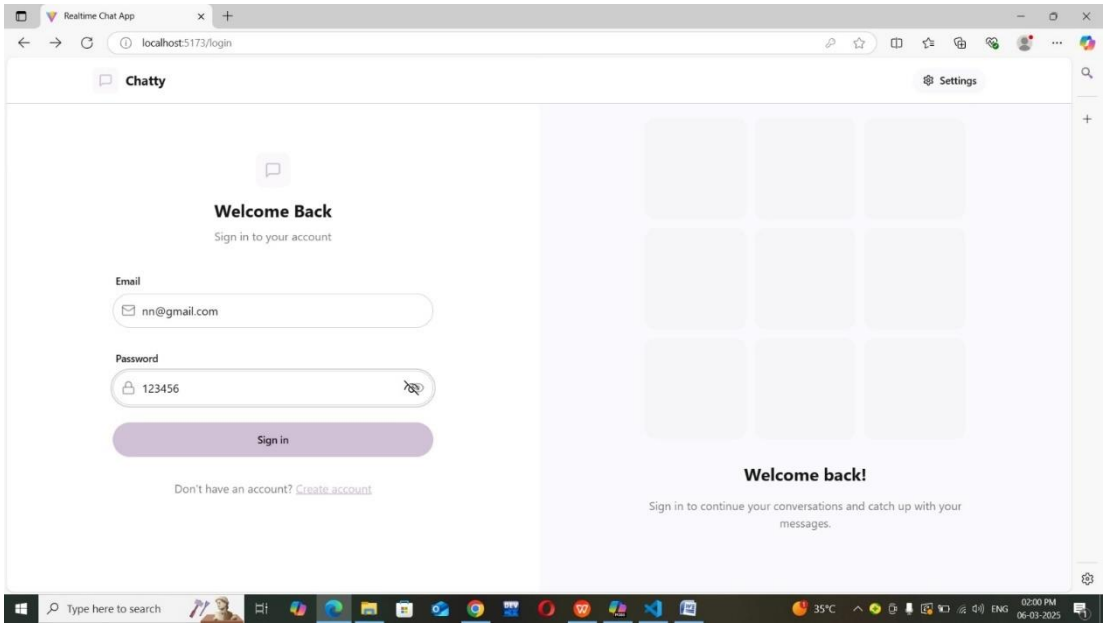


Fig. 8.2 User Login page

8.3 User Dashboard

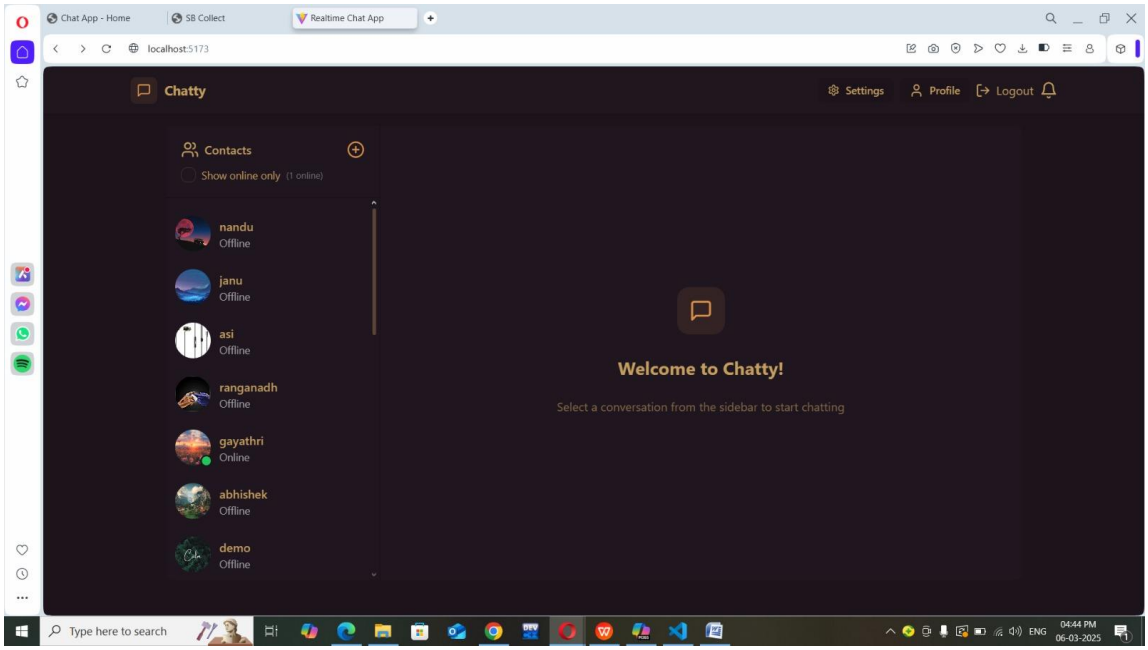


Fig. 8.3 User Dashboard

8.4 User Dashboard Theme

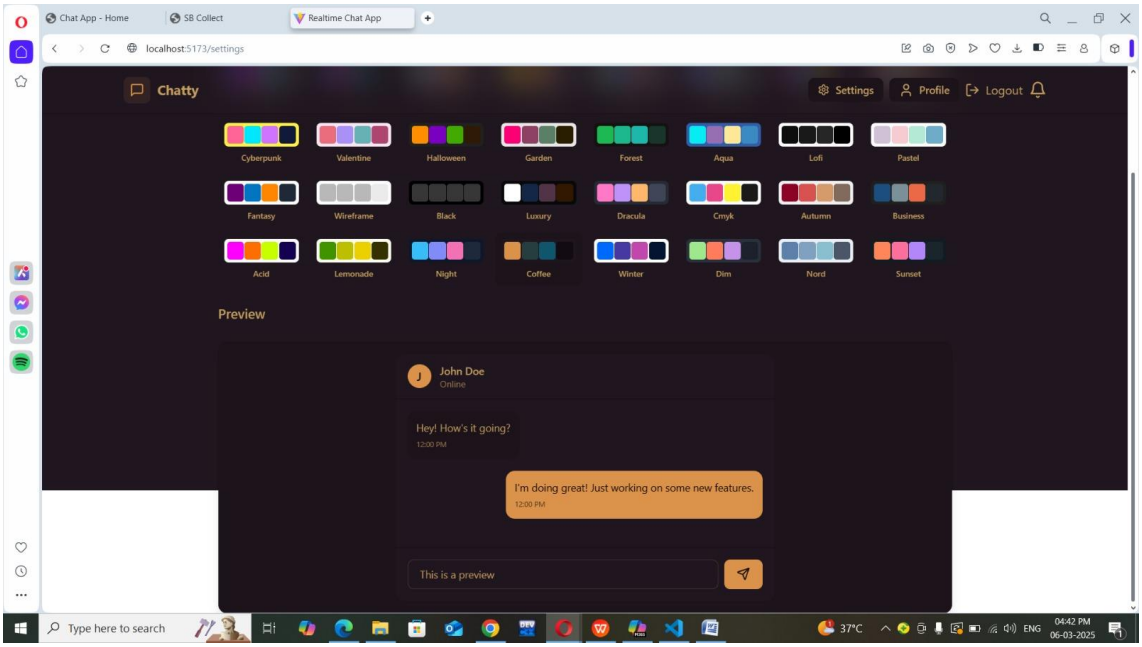


Fig. 8.4 User Dashboard Theme

8.5 User Profile Page

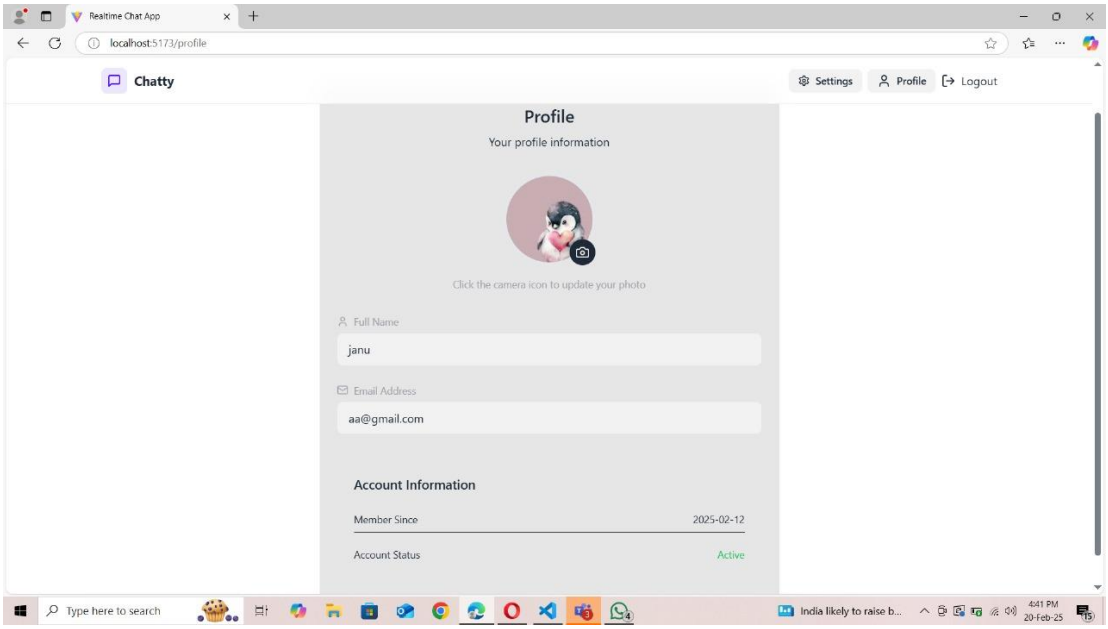


Fig. 8.5 User Profile Page

8.6 User Message

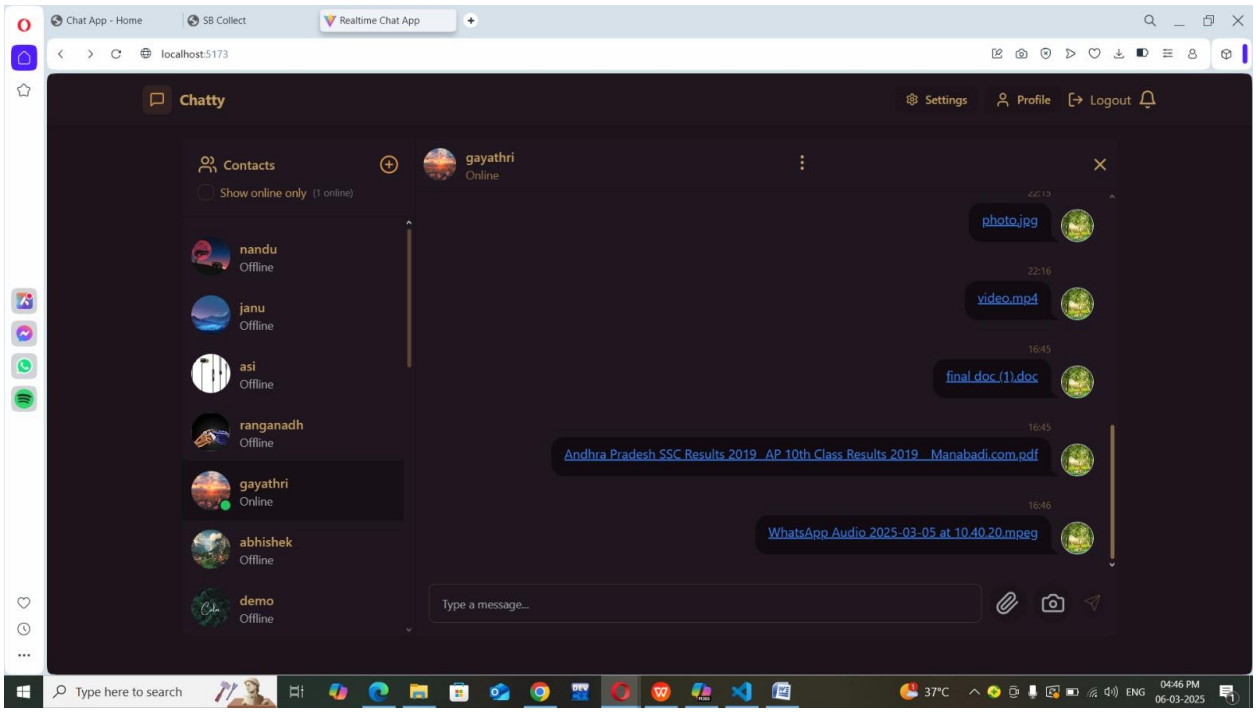


Fig. 8.6 User message

8.7 Group Chat

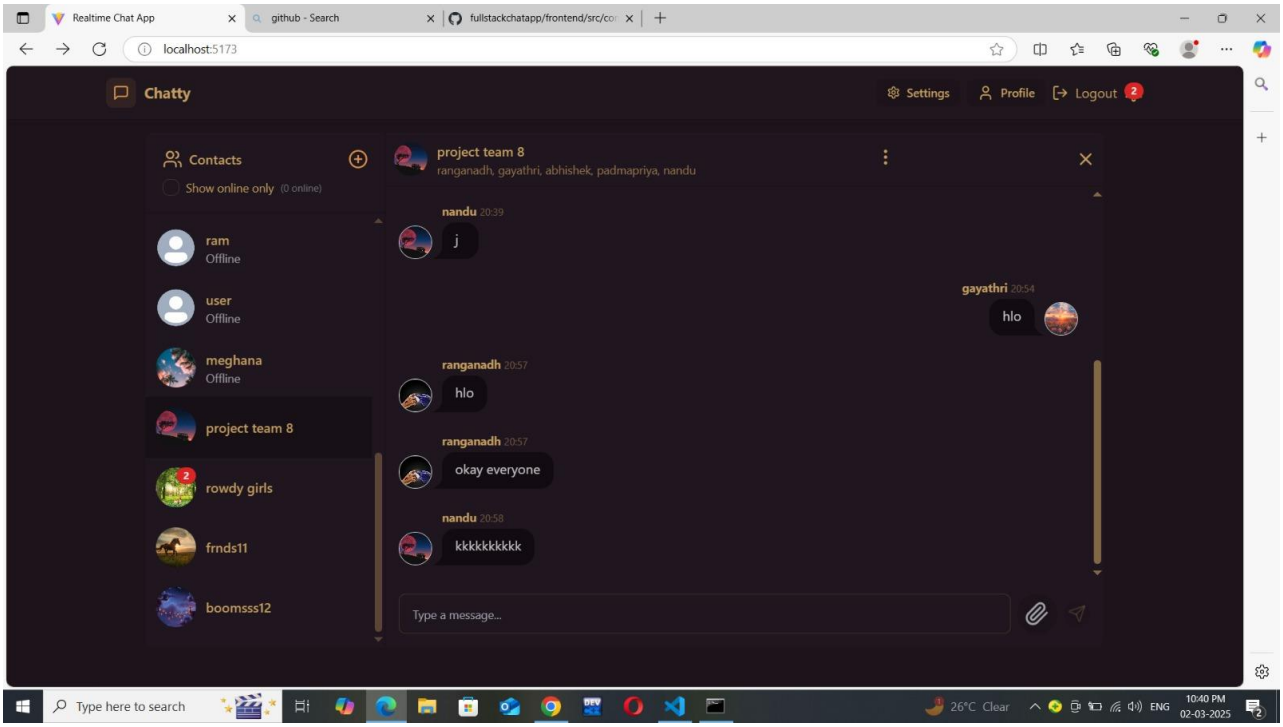
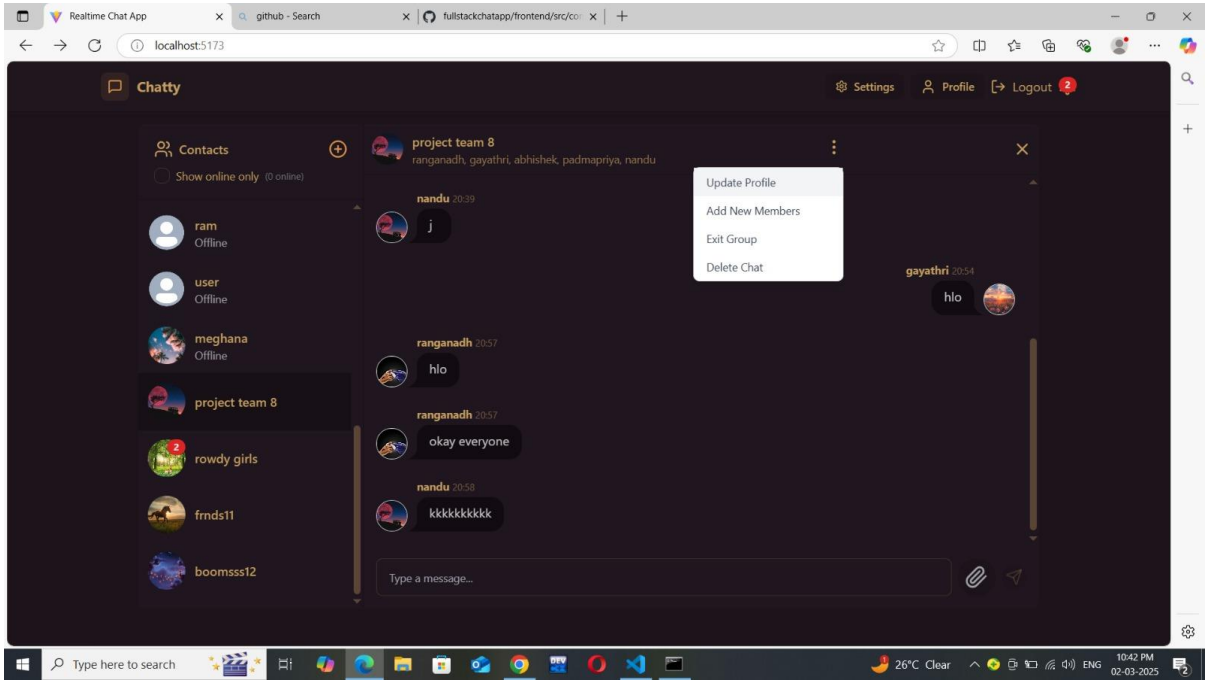


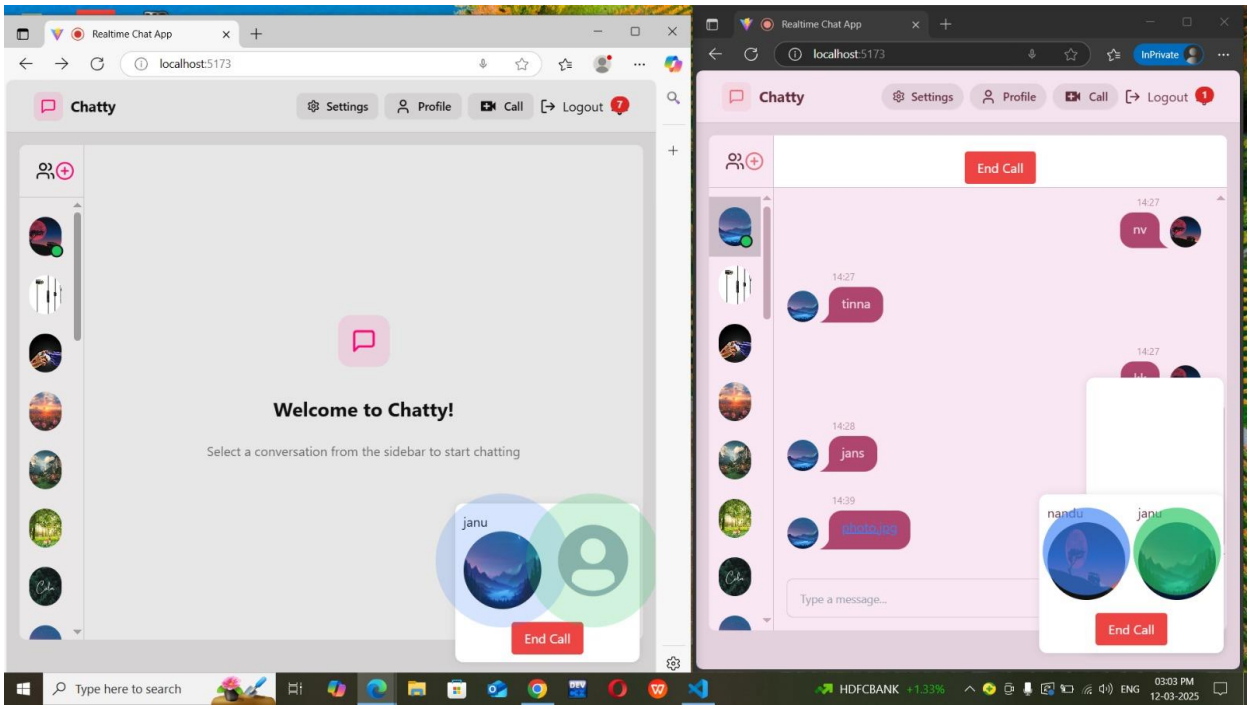
Fig. 8.7 Group chat

8.8 User Options in group chat



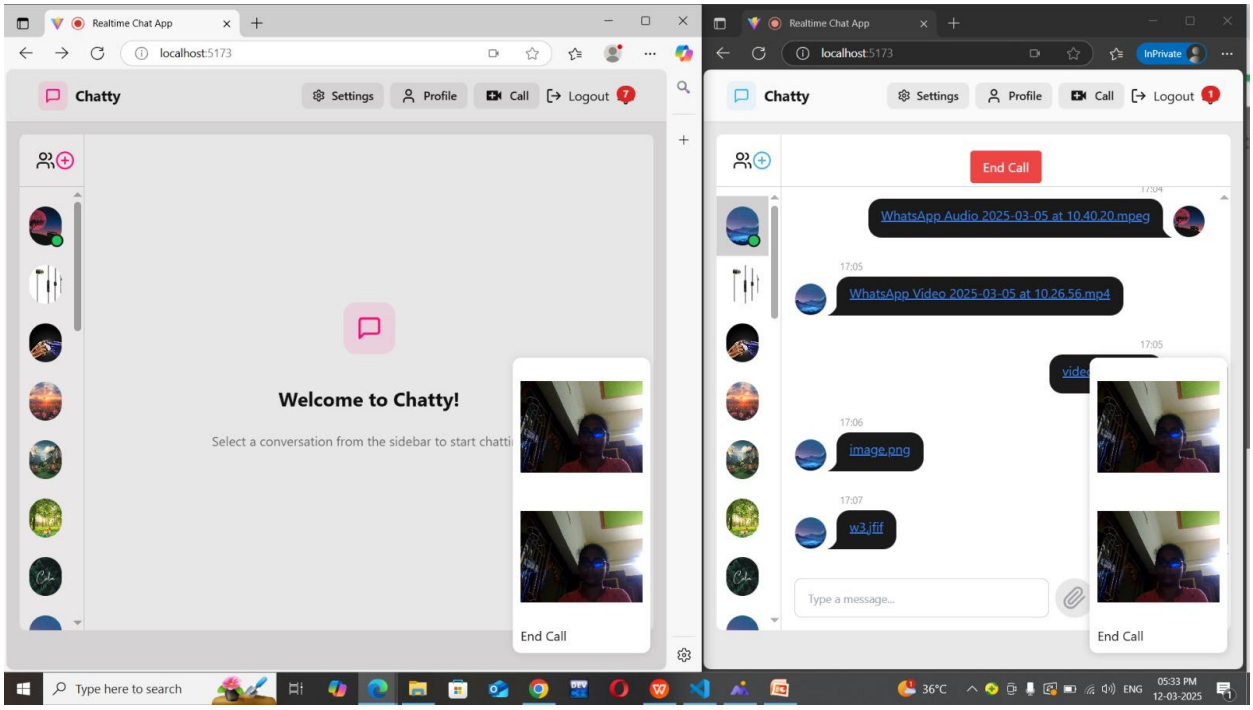
8.8 User options in group chat

8.9 Audio call



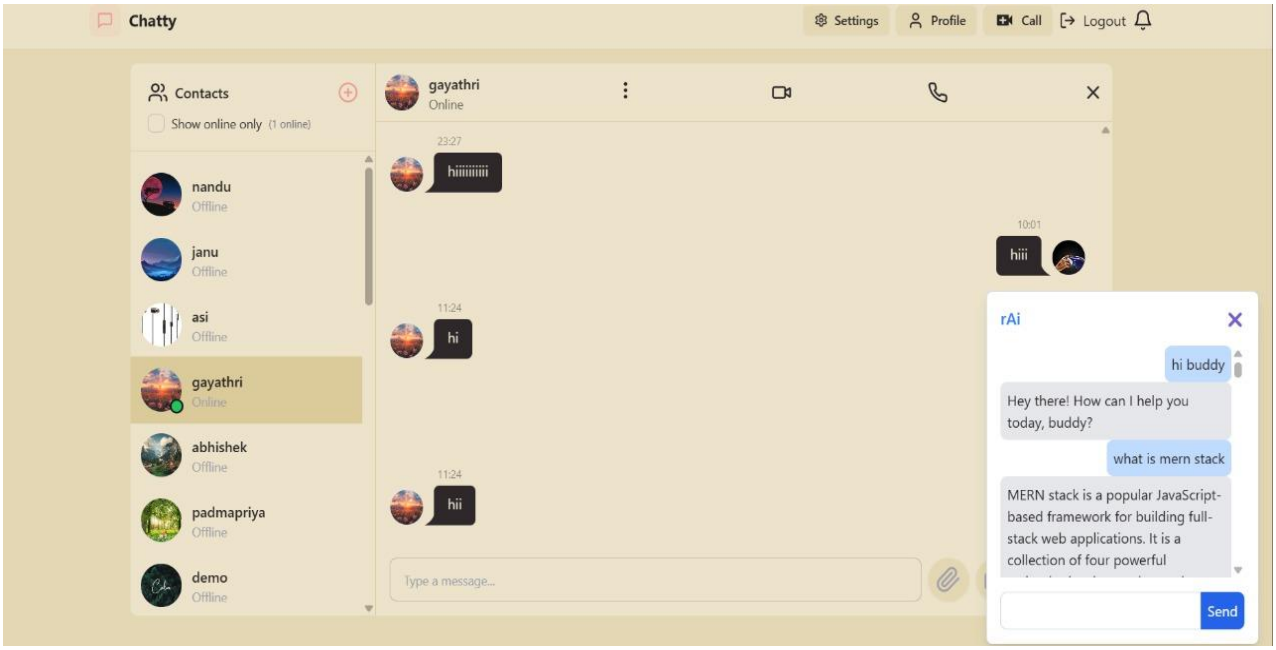
8.9 Audio call

8.10 Video Call



8.10 Video call

8.11 Chatbot



8.11 Chatbot

CHAPTER 9

CONCLUSION AND FUTUREWORK

9.1 CONCLUSION

The Full-Stack Chat Application is a robust and feature-rich communication platform designed to facilitate seamless real-time messaging, group chats, audio/video calls, and secure file transfers. The application successfully integrates WebSocket for instant message delivery, ensuring a smooth and interactive user experience. With an intuitive user interface and efficient backend architecture built using Express.js and MongoDB, the system is both scalable and highly responsive, making it suitable for a wide range of users.

Security has been a primary focus, with AES encryption implemented to protect messages and files, ensuring that user data remains private and secure. Additional security measures, such as token-based authentication (JWT) and optional multi-factor authentication (MFA), enhance account safety.

The admin controls allow for effective group management, ensuring better moderation and control over discussions. The notification system is designed to provide real-time updates for new messages, group activity, and incoming calls, enhancing user engagement.

The application also prioritizes user convenience with features like dark mode, message editing, search functionality, and pinned messages, making communication more efficient. Performance optimization ensures that even large group chats function smoothly without significant delays, and the system is built to handle high traffic with optimized database queries. Additionally, the integration of audio and video calls enhances communication beyond just text, making the application a comprehensive messaging solution.

9.2FUTURE WORK

Future enhancements for the chat application include secure cloud storage for encrypted file sharing, ensuring safe and private data transfers. Voice-to-text conversion will improve accessibility, while real-time language translation will enable seamless global communication. The application will also support integration with third-party services, enhancing its functionality. Blockchain-based identity verification will provide enhanced security, while an improved AI chatbot will offer more intuitive interactions. Additionally, offline functionality using Progressive Web App (PWA) features will allow users to access chats without an active internet connection.

REFERENCES

Project GitHub link:

https://github.com/MogilicherlaNandini/fullstackchatapp.git

REFERENCES:

[1]. https://www.researchgate.net/publication/370516068
_Real-Time_Chat_Application

[2]. https://www.ijrti.org/papers/IJRTI2206316.pdf

[3]. https://avestia.com/EECSS2024_Proceedings/files/paper/CIST/CIST_140.pdf

[4]. https://www.iieta.org/download/file/fid/143006

[5]. https://www.ijcrt.org/papers/IJCRT2411648.pdf

[6]. https://www.researchgate.net/publication/322509087_Developing_an_End-to
End_Secure_Chat_Application

BASE PAPER

International Journal of Scientific Research & Engineering Trends
Volume 8, Issue 1, Jan-Feb-2022, ISSN (Online): 2395-566X

Webvibe: A Secure Webchat Application

Prof. Amrita. A. Shirode, Riya Demapure, Atharva Katurde, Mohit Dhande, Awantika Jadhav
Dept. of Computer Science
AISSMS Polytechnic,Pune,MH,India

Abstract- Instant messaging is a set of communication technologies used for text-based communication between two (private messaging) or more (chatroom) participants over the Internet or other types of networks, byproviding an alternative to telephone and email conversations. The number of users using Messenger products like WhatsApp, Telegram has been increasing over recent years. Messenger services allow effective and efficient communication, allowing immediate receipt of replies. People of all ages log into messenger services to spend time chatting with known and unknown persons. Similarly, our project aims to develop a client-server java chat application with security in mind. The use of Encryption and decryption algorithms forms part of this project whereby messages exchanged between client and server are encrypted and decrypted with asymmetric private keys.The project overview that shows the structure of the project design is depicted in the introduction part of this document.

Keywords- Chatting, Security, Application, Networking.

I. INTRODUCTION

"Messaging is one of those things that people do more than social networking." At present more than three million people are using numerous chat apps installed on their smartphones. Today's messengers are not just about sending and receiving messages; instead, they allow users to share photos, videos, gifs, emoticons, voice messages, and a lot more. With many messenger applications already available in the market, making our chat messenger popular is not a cakewalk. However, by implementing it on the web with advanced features into it, we are trying to make this journey smooth and easy. People are more likely to be anonymous, which gives them more strength to express themselves anytime anywhere. Thissame ideais used by most of the popular forums on the internet. So, we came up with some of the basic requirements and by adding some advanced features.

II.METHODOLOGIES

As we came up with some of the basic requirements and features that will fulfill users' requirements, we have some of the features like:

1. Users will register by giving a handle, which will be unique to every user. Only the handle will be revealed to other users to whom they will chat. So, people are free to choose any handle so they stay anonymous.
2. A member can see other online members.
3. The sender should first send a request to the other member for private messages. On accepting the request of other members,they can have private chat with them.
4. To develop a client-server java chat application that will enable instant messaging between host computers on a network
5. To provide authentication of the message to know whether the message has been sent by the genuine or intended user

III. FUTURE SCOPE

There is always a possibility and extent for improvement in any application. Right now, we are just dealing with text-based communication. Several chat apps serve similar purposes as this project, but these apps were rather difficult to use and provide confusing interfaces. A positive first impression is essential in a human relationship as well as in human-computer interaction. This project hopes to develop a chat service Web app with a high-quality user interface.With the knowledge we have gained by developing this application, we are confident that in the future we can make the application more effective by adding these services.

- File Transfer
- Video Message
- Audio Call
- Video Call
- Group Call
- Extending this application by providing Authorization service.
- Creating Database and maintaining users.
- Extending it to Web Support.
- Increasing the effectiveness of the application by providing Voice-based chat.

IV.CONCLUSION

The main goal of the project is to develop a Secure Chat Application. We had taken a wide range of literature reviewsto achieve all the tasks, where we came to know about some of the products that are existing in the market. We made detailed research in that path to cover the loopholes that existing systems are facing and to eradicate them in our application. In the process of research, we came to know about the technologies used to develop any chat architecture and different encryption-decryption algorithms. We analyzed various encryption algorithms (DES, AES, IDEA…), Integrity algorithms (MD5, SHA), key-exchange algorithms, authentication and we had implemented those functionalities in our application. We had gone through core and security concepts of java (JSSE, JCA) packages and for developing GUI we had implemented java swings.

ACKNOWLEDGMENT

We would like to thank the teacher and friends for supporting us throughout the making and preparation of this paper presentation.

BIBLIOGRAPHY

[1] ElGamal, T. "A Public-Key Cryptosystem and a Signature Scheme Based on DiscreteLogarithms." IEEE Transactions on Information Theory, July 19858.
[2] Jueneman, R.; Matyas, S.; and Meyer, C. "Message Authentication." IEEE CommunicationsMagazine, September 1988
[3] Design and realization of chatting tool based on the web by IEEE https://ieeexplore.ieee.org/document/6703312 (Referred as base paper)
[4] Kohnfelder, L. Towards a Practical Public-Key Cryptosystem. Bachelor's Thesis, M.I.T.,May 1978
[5] Mohammed, M. SC May S. "Online Chatting Protection system." Iraqi Journal of Information Technology 9, no. 4 (2019): 120-134.
[6] Ogundeyi, K. E., and C. Yinka-Banjo. "WebSocket in real-time application." Nigerian Journal of Technology 38, no. 4 (2019): 1010-1020.
[7] Bellare, M.; Kilian, J.; and Rogaway, P. "The Security of the Cipher Block Chaining Message Authentication Code." Journal of Computer and System Sciences, December 2000.4.
[8] A project report on chat Application by SlideShare https://www.slideshare.net/CrGaurav/a-project-report-on-chat-application
[9] Design and implementation of a real-time chat application.https://portal.bazeuniversity.edu.ng/student/assets/thesis/20210215120658149063642.pdf (referred base paper)
[10] web-based chat application minor project report computer science &engineering
[11] https://www.academia.edu/40977586/web_based_chat_application_minor_project_report_computer_science_and_engineering