CHIT-CHAT

A Project Report

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

BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)

By

MANISH JHA

Under the esteemed guidance of

MRS. DIVYA SHETTY

Lecturer

DEPARTMENT OF INFORMATION TECHNOLOGY

VIVEKANAND EDUCATION SOCIETY'S
COLLEGE OF ARTS, SCIENCE AND COMMERCE
(Autonomous)
Affiliated to University of Mumbai

CHEMBUR MUMBAI – 400071

MAHARASHTRA

2023-2024

Proforma for the Approval Project Proposal

PNR No.: 2021016401551792 Roll No. 23

1. Name of the Student

   MANISH JHA

2. Title of the Project

   CHIT - CHAT

3. Name of the Guide

   MRS. DIVYA SHETTY

4. Teaching Experience of the Guide

   16

5. Is this your first submission? [✓ Yes] No ☐

Signature of Student

Date: ……………..

Signature of Guide

Date: ……………..

Signature of the Coordinator

Date: …………….

VIVEKANAND EDUCATION SOCIETY'S
COLLEGE OF ARTS, SCIENCE AND COMMERCE
(Autonomous)

Affiliated to University of Mumbai

CHEMBUR-MAHARASHTRA-400071

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project entitled, **" CHIT-CHAT "** , is bonafide work of **MANISH JHA** submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

Internal Guide **Coordinator**

External Examiner

Date: **College Seal**

ABSTRACT

Chatting, is a method of using technology to bring people and ideas together despite of the geographical barriers. The technology has been available for years but the acceptance it was quit recent. My project is an example of chat server. It is made up to two phase the client phase, which runs on the users machine and the server phase, which runs on the network. To start chatting client should get connected to server where they can do private and group chat security measures were taken during the last one. My aim is to develop a modern and secure chat application system for both individual users and businesses. This project utilizes cutting-edge encryption algorithms and a user-friendly interface to ensure the confidentiality and integrity of messages. Key features include end-to-end encryption, multi-platform support, group chat functionality, and real-time notifications. This documentation outlines the project's methodology, discusses the benefits of the system, and provides insights into the development process.

ACKNOWLEDGEMENT

We thank the people who were a part of this project in numerous ways, people who gave their unending support right from the stage the project idea was conceived.

The four things that go on to make a successful endeavour are dedication, hard work, patience and correct guidance.

We would like to thank our Principal **Dr. Mrs. Anita Kanwar** who has always been the source of inspiration.
We are also thankful to **Mrs. Shital Patil, our** In-charge coordinator who was very much kind enough to give us an idea and guide us throughout our project work.

We take this opportunity to offer sincere thanks to **Mrs. Divya Shetty .**

We are thankful to all the teaching staff (I.T) who shared their experience and gave their suggestions for developing our project in a better way.

We are also thankful to **Parents and Friends** for helping us out in Project Documentation.

DECLARATION

I hereby declare that the project entitled, “**CHIT - CHAT**” done at the place where the project is done, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university. The project is done in partial fulfilment of the requirements for the award of the degree of BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY) to be submitted as a final semester project as part of our curriculum.

Name and Signature of the Student

TABLE OF CONTENTS

CHAPTER:- 1 INTRODUCTION

1.1 BACKGROUND

Certainly! A chat application, also known as a messaging app or instant messaging app, is a software platform that allows users to communicate with each other in real-time via text messages. These applications have become increasingly popular with the widespread use of smartphones and the need for quick and convenient communication.

Here is some background information on chat applications : -

1. Early Beginnings :- Chat applications have been around since the early days of the internet. The first widely used chat protocol was Internet Relay Chat (IRC), which was developed in 1988.

2. **Instant Messaging** :- The concept of instant messaging gained popularity in the 1990s with the rise of services like ICQ, AOL Instant Messenger (AIM), and MSN Messenger.

3. **Mobile Messaging** :- With the advent of smartphones, messaging apps evolved to support mobile platforms. WhatsApp, launched in 2009, became one of the pioneers of mobile messaging by providing a cross-platform instant messaging service that utilized phone numbers as unique identifiers.

4. **Multimedia Messaging** :- As technology advanced, chat applications started supporting multimedia features. Users could send not only text messages but also photos, videos, voice messages, and even documents.

5. **End-to-End Encryption :-** Privacy and security became significant concerns for messaging app users. End-to-end encryption became a desirable feature, ensuring that only the intended recipients can read the messages.

6. **Group Chats and Channels** :- Chat applications expanded to include group chats, where multiple users could communicate in a single conversation. Channels or public chats allowed users to follow and engage with specific topics, communities, or celebrities.

7. **Social Integration** :- Some messaging apps have integrated social networking features, blurring the lines between messaging and social media platforms. Facebook Messenger, for instance, allows users to share stories, posts, and other social media content alongside their conversations.

1.2 OBJECTIVES

The primary objectives of a chat application are to facilitate communication and foster real-time interaction between users. Here are the key objectives of a chat application:

1. Real-Time Communication :- The main objective of a chat application is to enable users to exchange messages in real time. It allows individuals or groups to have instant conversations, overcoming the delays associated with traditional communication methods like email or postal mail.

2**. Convenience and Accessibility** :- Chat applications aim to provide a convenient and accessible means of communication. Users can quickly send and receive messages anytime, anywhere, as long as they have an internet connection.

3. **Instant Notifications** :- Chat applications typically provide notifications to users, alerting them about new messages or activities. This objective ensures that users stay informed and can respond promptly, even when they are not actively using the app.

4. **Multimedia Sharing** :- A chat application allows users to share various forms of media, including photos, videos, audio recordings, documents, and links. The objective is to enable rich and interactive conversations by accommodating different types of content.

5. **Privacy and Security** :- Ensuring the privacy and security of user conversations is a crucial objective for chat applications.

6. **Group Collaboration** :- Many chat applications support group chats or channels, allowing users to collaborate, share information, and work together on projects.

7. **Additional Features and Integrations** :- Chat applications may offer additional features and integrations to enhance user experience and productivity. This can include features like chatbot assistance, file sharing, task management, calendar integration, and integration with other services or platforms.

8. **Cross-Platform Compatibility** :- The objective of cross-platform compatibility is to allow users to seamlessly transition between devices while maintaining their conversations and message history.

9. **User Engagement and Retention :-** Chat applications aim to engage users and keep them actively using the platform. This can be achieved through features like emojis, stickers, GIFs, reactions, status updates, and social integration, fostering a dynamic and enjoyable user experience.

Overall, the objectives of a chat application revolve around facilitating real-time communication, providing convenience and accessibility, ensuring privacy and security, and offering a range of features to enhance user interactions and engagement.

1.3 PURPOSE, SCOPE & APPLICABILITY

1.3.1 PURPOSE

The purpose of a chat application is to facilitate communication and enable individuals or groups to engage in real-time conversations. Here are the main purposes of a chat application:

1. Real-Time Communication :- It provides a platform for instant messaging, allowing users to send and receive messages instantly, fostering quick and efficient communication.

2. Connectivity and Relationship Building :- Chat applications help people connect with each other, regardless of geographical distance. They bridge the gap between individuals, enabling them to communicate and build relationships, whether for personal, social, or professional purposes.

3. Convenience and Accessibility :- Chat applications offer a convenient and accessible mode of communication. This purpose emphasizes the ease of use and availability of chat platforms.

4. Collaboration and Teamwork :- Chat applications often support group chats or channels, making them ideal for collaboration and teamwork. Users can discuss ideas, share information, coordinate tasks, and work together on projects, enhancing productivity and efficiency.

5. Information Sharing :- Chat applications facilitate the sharing of information and knowledge. Users can exchange text messages, multimedia content, documents, links, and other relevant data.

6. Customer Support and Service :- This are widely used in customer support and service scenarios. They enable businesses to interact with customers in real time, providing assistance, answering queries, and resolving issues promptly.

7. Networking and Social Interaction :- It provide a platform for networking and social interaction. They allow users to connect with friends, family, colleagues, and even new acquaintances, facilitating socialization and fostering a sense of community.

8. Entertainment and Expressiveness :- Chat applications often incorporate features like emojis, stickers, GIFs, and multimedia sharing, adding an element of fun, expressiveness, and entertainment to conversations.

9. Privacy and Security :- Chat applications prioritize user privacy and security. They implement measures like end-to-end encryption, user authentication, and privacy settings to ensure that conversations are protected and users can communicate with confidence.

10. Business Communication and Collaboration :- Chat applications play a vital role in business communication and collaboration. They enable teams to communicate internally, share updates, discuss projects, and coordinate tasks in real time. This purpose emphasizes the importance of chat apps in enhancing workplace communication and productivity.

Overall, the purpose of a chat application is to facilitate instant communication, connect individuals or groups, enable collaboration and information sharing, provide convenience and accessibility, and enhance social interaction and networking.

1.3.2 SCOPE

The basic scope of a chat application includes the fundamental features and functionalities that allow users to engage in real-time communication. Here are the key components within the scope of a chat application:

1. User Registration and Authentication :- The chat application should provide a user registration process that allows individuals to create an account and authenticate their identity. This ensures that users have a unique identifier within the system.

2. Real-Time Messaging :- The core functionality of a chat application is the ability to exchange text messages in real time. Users should be able to send, receive, and view messages instantly within individual or group conversations.

3. Multimedia Messaging :- Chat applications often support the sharing of multimedia content, such as photos, videos, audio files, and documents. Users should be able to send and receive these types of files within their conversations.

4. Notifications :- The application should provide notifications to alert users about new messages, friend requests, or other relevant activities. Notifications help users stay informed and respond promptly to incoming messages.

5. Presence and Status :- Users should have the ability to set their availability status (e.g., online, offline, away) and view the status of their contacts. This feature allows users to know when their contacts are active or unavailable for communication.

6. Group Chats and Channels :- Chat applications often support group conversations, allowing multiple users to participate in a single chat. Users should be able to create, join, and manage group chats, as well as follow or join public channels on specific topics or interests.

7. Emojis, Stickers, and Reactions :- Many chat applications provide a library of emojis, stickers, or reactions that users can include in their messages to express emotions or convey additional meaning. These elements enhance the communication experience and add a touch of fun and personalization.

8. Search and Message History :- Users should be able to search their message history to find specific conversations or keywords. The chat application should store and retrieve past messages, allowing users to refer back to previous discussions.

9. Privacy and Security Settings :-The application should include privacy and security settings that allow users to control their visibility, manage friend requests, block or report users, and customize their privacy preferences.

1.3.3 APPLICABILITY

Chat applications have a wide range of applicability across various domains and Industries. Here are some common areas where chat applications find applicability:

1. Personal Communication :- Chat applications are extensively used for personal communication between friends, family members, and acquaintances. They provide a convenient and instant way to stay connected, exchange messages, and share updates.

2. Social Networking :- Many social networking platforms incorporate chat functionality to allow users to communicate with their connections. Chat applications within social networks enable users to chat with their followers, engage in group conversations, and share content.

3. Business Communication :- Chat applications have become essential tools for business communication. They enable employees to collaborate, share information, and coordinate tasks in real time. Businesses often use chat applications to facilitate internal communication within teams and departments.

4. Customer Support and Service :- Chat applications are commonly used for customer support and service interactions. They provide a direct channel for customers to reach out to businesses, ask questions, seek assistance, and resolve issues.

5. E-commerce and Online Marketplaces :- Chat applications are integrated into e-commerce platforms and online marketplaces to enable direct communication between buyers and sellers. Users can inquire about products, negotiate prices, and finalize transactions within the chat interface.

6. Education and E-Learning :- Chat applications are utilized in educational settings for student-teacher communication, group discussions, and collaborative learning. They allow students to ask questions, seek clarification, and engage in virtual classrooms or study groups.

7. Community Building :- Chat applications are utilized to create and nurture online communities. They provide a platform for like-minded individuals to connect, share ideas, and engage in discussions on specific topics of interest.

8. Collaborative Projects and Remote Work :- Chat applications support collaboration among remote teams and facilitate project management. They enable team members to communicate, share files, and coordinate tasks in real time, regardless of their physical location.

The versatility and flexibility of chat applications make them suitable for various communication needs across different industries and contexts.

1.4 ACHIEVEMENTS

Chat applications have achieved significant milestones and made notable contributions in the realm of communication and technology.

Here are some key achievements of chat applications:

1. Revolutionized Communication :- Chat applications have revolutionized the way people communicate. They have replaced traditional communication methods like letters, faxes, and even phone calls for many users.

2. Global Reach and Connectivity :- Chat applications have facilitated global connectivity, allowing individuals from different parts of the world to connect and communicate easily.

3. Mobile Messaging Dominance :- Chat applications have become dominant in the mobile messaging space. Apps like WhatsApp, WeChat, and Messenger have amassed billions of users worldwide, becoming a primary mode of communication for smartphone users.

4. Multimedia Messaging :- Chat applications have introduced the capability to share multimedia content within conversations.

5. Real-Time Collaboration :- They allow real-time discussions, file sharing, and coordination, enabling seamless teamwork and enhancing productivity in various industries.

6. End-to-End Encryption :- Several chat applications have implemented end-to-end encryption, ensuring that messages are securely transmitted and only accessible to the intended recipients.

7. Integration of Chatbots and AI :- Chat applications have integrated chatbots and artificial intelligence capabilities, enabling automated responses, personalized recommendations, and task automation.

8. Customer Support and Service :- Chat applications have been adopted as primary channels for customer support and service interactions. Businesses leverage chat applications to provide instant assistance, resolve queries, and enhance customer satisfaction.

9. Social and Community Engagement :- They have provided platforms for individuals to connect, share common interests, and engage in discussions or groups centered around specific topics or hobbies.

10. Cross-Platform Compatibility :- Chat applications have achieved cross-platform compatibility, allowing users to seamlessly transition between devices while maintaining their conversations and message history.

1.5 ORGANISATION OF REPORT

Chapter 1:

Introducing the structure of the thesis paper.

Chapter 2:

Explaining the technologies and tools, programming languages and methodology used in this project.

Chapter 3:

Explore the system requirement analysis (functional and non-functional requirements) and requirement specifications

Chapter 4:

Describing the system design of OSS and the topics are use-case diagrams, class diagram, sequence diagrams and database.

Chapter 5:

System implementation discusses the process of coding and implementation of the system.

Chapter 6:

System Testing discusses a practical example of electronic marking through the use of the Generic Interface System.

Chapter 7:-

Conclusion. recommendations and future work.

CHAPTER:- 2 SURVEY OF TECHNOLOGY

FRONT END/GUI TOOLS	**.Net Technologies, Java, HTML/CSS**
DBMS/BACK END	**Oracle, SQL Plus, MySQL, SQL Server**
LANGUAGES	**C, C++, Java, VC++, C#, R, Python**
SCRIPTING LANGUAGES	**PHP, JSP, JavaScript**
.NET Platform	**C# .Net, Visual C# .Net, ASP .Net**

FRONT END/GUI TOOLS :-

i. **.Net Technologies :-** . NET is an open-source platform for building desktop, web, and mobile applications that can run natively on any operating system. The . NET system includes tools, libraries, and languages that support modern, scalable, and high-performance software development.

ii. **Java :-** Java is a widely used object-oriented programming language and software platform that runs on billions of devices, including notebook computers, mobile devices, gaming consoles, medical devices and many others. The rules and syntax of Java are based on the C and C++ languages.

iii. **HTML/CSS :-** The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It defines the meaning and structure of web content. It is often assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

DBMS/BACK END :-

i. **Oracle :-** Oracle database is a relational database management system. It is also called OracleDB, or simply Oracle. It is produced and marketed by Oracle Corporation. It was created in 1977 by Lawrence Ellison and other engineers. It is one of the most popular relational database engines in the IT market for storing, organizing, and retrieving data.

ii. **SQL Plus :-** SQL Plus is the most basic Oracle Database utility, with a basic command-line interface, commonly used by users, administrators, and programmers.

iii. **MySQL :-** MySQL is an Oracle-backed open source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX and Windows. Although it can be used in a wide range of applications, MySQL is most often associated with web applications and online publishing.

iv. **SQL Server :-** SQL Server is a proprietary relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

LANGUAGES :-

i. **C :-** C is a general-purpose computer programming language. It was created in the 1970s by Dennis Ritchie, and remains very widely used and influential. C is commonly used on computer architectures that range from the largest supercomputers to the smallest microcontrollers and embedded systems.

ii. **C++ :-** C++ is a cross-platform language that can be used to create high-performance applications, was developed by Bjarne Stroustrup, as an extension to the C language. It is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

iii. **Java :-** Java is a multi-platform, object-oriented, and network-centric language that can be used as a platform in itself. It is a fast, secure, reliable programming language for coding everything from mobile apps and enterprise software to big data applications and server-side technologies.

iv. **VC++ :-** Visual C++ is part of the general "C suite" of programming languages often used for many different types of development projects. The full name of the product is Microsoft Visual C++, which is sometimes abbreviated as MSVC or VC++.

v. **C# :-** C# (C Sharp) is an element and type safe object Oriented Programming Language that enables developers to build a variety of secure and robust applications that run on .NET Framework.

vi. **R :-** R is a language and environment for statistical computing and graphics. It is a GNU Project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.

vii. **Python :-** Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

SCRIPTING LANGUAGES :-

i. **PHP :-** PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages. It is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

ii. **JSP :-** Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

iii. **JavaScript :-** JavaScript is a lightweight, cross-platform, single-threaded, and interpreted compiled programming language which is also known as the scripting language for webpages.

.NET Platform :-

i. **C# .NET :-** C# (C-Sharp) is a programming language developed by Microsoft that runs on the .NET Framework. It is used to develop web apps, desktop apps, mobile apps, games and much more.

ii. **Visual C#.NET :-** Visual C#.NET is just C#. We can build any kind of .NET application using C# and Visual Studio makes it easier test and debug your application.

iii. **ASP.NET :-** ASP.NET is an open-source, server-side web-application framework designed for web development to produce dynamic web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, applications and services. The name stands for Active Server Pages Network Enabled Technologies.

FRONT END/GUI TOOLS	**REACT,CSS**
DBMS/BACK END	**MongoDB**
LANGUAGES	**MongoDB, Express.js, Node.js, React.js**
SCRIPTING LANGUAGES	**JavaScript**
.NET Platform	**ASP.NET**

FRONT END/GUI TOOLS

React :- React is a free and open-source front-end JavaScript library for building user interfaces based on components. React can be used to develop single-page, mobile, or server-rendered applications with frameworks like Next.js.

CSS :- Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

DBMS/BACK END

MongoDB :- MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and current versions are licensed under the Server Side Public License (SSPL) which is considered non-free by some organizations and distributions. MongoDB is a member of the MACH Alliance.

LANGUAGES

MongoDB :- MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

Express.js :- Express.js is a small framework that works on top of Node.js web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middleware and routing. It adds helpful utilities to Node.js HTTP objects and facilitates the rendering of dynamic HTTP objects.

Node.js :- Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.Node.js has an event-driven architecture capable of asynchronous I/O.

React.js :- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

SCRIPTING LANGUAGES

JavaScript :- JavaScript is an object-based scripting language which is lightweight and cross-platform .JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

.NET Platform

ASP.NET :- ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites. It allows you to use a full featured programming language such as C# or VB.NET to build web applications easily.

JUSTIFICATION

FRONT END/GUI TOOLS

React.js :- I use ReactJS in my project for its component-based architecture, virtual DOM for efficient rendering, declarative syntax, and vast community support. It simplifies UI development, enhances performance, and enables code reusability, making it an ideal choice for building interactive and scalable web applications.

CSS :- I use CSS in my project to control the presentation and layout of web pages. It enables me to customize colors, fonts, spacing, and other design aspects, enhancing the overall user experience.

DBMS/BACK END

MongoDB :- I use MongoDB in my project for its NoSQL database capabilities. MongoDB's flexible document-based data model allows us to store and manage unstructured or semi-structured data efficiently. It enables scalability, high performance, and seamless integration with modern web applications, making it ideal for handling diverse data requirements.

LANGUAGES

MongoDB :- . It enables scalability, high performance, and seamless integration with modern web applications, making it ideal for handling diverse data requirements.

Express.js :- I use Express.js in my project as it is a fast and minimalist web framework for Node.js. It simplifies the creation of server-side applications, handles routing, and provides middleware support.

Node.js :- It is a powerful and efficient server-side JavaScript runtime. It allows us to build scalable, high-performance applications, handle concurrent connections, and utilize a single language (JavaScript) for both frontend and backend development, leading to improved productivity and reduced development complexities.

React.js :- Its component-based architecture, virtual DOM, and declarative syntax, which simplify UI development and enhance performance. Its vast community and ecosystem provide ready-made solutions, making it an excellent choice for building interactive and responsive web applications efficiently.

SCRIPTING LANGUAGES

JavaScript :- It is a versatile and widely supported programming language. It runs on the client and server sides, enabling dynamic and interactive web applications. JavaScript's rich ecosystem, frameworks, and libraries make it suitable for a wide range of projects, enhancing development speed and flexibility.

CHAPTER 3 :- SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing system required for a chat application includes:

A user-friendly interface that allows users to send, receive, and view messages easily. This may involve chat bubbles, conversation threads, and user avatars. Backend servers to manage message sending, receiving, and storage. These servers handle real-time message delivery and synchronization across devices. User authentication mechanisms to ensure that only authorized users can access the chat application. Encryption is crucial to secure the communication and protect user data. A database to store messages, user profiles, and other relevant information. This enables message history and retrieval. Technologies like WebSocket's for establishing persistent connections, enabling instant message delivery and real-time updates. A notification system to inform users about new messages even when the app is not actively open. Support for emojis, emoticons, and multimedia sharing like images, videos, and documents. Functionality to create and manage group chats, enabling multiple users to communicate in a single conversation. Displaying the online/offline status of users and indicating when someone is typing. Allowing users to create and customize their profiles with names, profile pictures, and status messages. Options for users to block or report others for inappropriate behaviour. Capability to search for and retrieve past messages and conversations. Ensuring the application works seamlessly across different devices and platforms (web, mobile, desktop). Architecture that can handle increasing numbers of users and messages without a significant drop in performance. Regular data backups and mechanisms to restore data in case of data loss. Tools to gather insights on user behaviour, engagement, and usage patterns. Possibility to integrate with other platforms, services, or APIs for enhanced functionality. Resources to provide technical support and continuous maintenance to address bugs, updates, and improvements.

3.2 PROPOSED SYSTEM

The proposed chat application system will feature end-to-end encryption for secure messaging, multi-platform compatibility across iOS, Android, and desktop, voice and video calling, AI-powered suggestions, multimedia sharing, message reactions, and threaded replies. Additionally, it will offer location sharing, advanced search, smart notifications, chatbot integration, language translation, data synchronization, blockchain data integrity, third-party service integration, accessibility features, user data control, and a commitment to continuous improvement based on user feedback. This system aims to provide a dynamic, secure, and user-centric communication experience.

Advantage of Proposed System :-

- ➢ Enhanced Security: End-to-end encryption and blockchain data integrity ensure messages are private and tamper-proof, safeguarding user privacy and data.
- ➢ Versatile Accessibility: Multi-platform compatibility enables users to communicate seamlessly across different devices and operating systems, enhancing convenience and accessibility.

- Rich Communication: Voice and video calling, multimedia sharing, message reactions, and threaded replies enable diverse and engaging communication experiences.

- Efficient Organization: Advanced search, location sharing, and smart notifications improve organization and contextual relevance within conversations.

- Global Interaction: Language translation bridges language barriers, allowing users to communicate effectively with individuals from different linguistic backgrounds.

3.3 REQUIREMENT ANALYSIS

3.3.1 PROBLEM DEFINITION :-

- Current chat apps struggle with organization, making it challenging for users to locate specific messages within cluttered conversations, especially in group chats.

- Inadequate notification systems overwhelm users with irrelevant alerts, causing missed essential messages and user frustration.

- Language barriers remain a challenge, as real-time translation features are often missing, hampering effective communication between speakers of different languages.

- Data privacy concerns persist due to insufficient safeguards for user data, leading to worries about data leaks and misuse.

- Basic chatbots lack meaningful automation, failing to provide the advanced assistance users need for efficient interactions.

3.3.2 REQUIREMENT SPECIFICATION :-

User Registration: Ability for users to create accounts and log in securely.

Messaging: Real-time one-on-one and group chat functionality with text, emojis, and multimedia support.

Notifications: Push notifications for new messages and mentions.

User Profiles: User profiles with display pictures and status updates.

Privacy and Security: End-to-end encryption, report/block functionality, and password recovery.

Search: Search for users, messages, and groups.
File Sharing: Ability to share images, videos, documents, and other files.

Voice and Video Calls: Optional audio and video call features.
Offline Messaging: Store and deliver messages when users are offline.

Multi-Platform: Availability on multiple devices (web, mobile, desktop).

Accessibility: User-friendly interface with accessibility features.

Scalability: Handle a growing number of users and messages efficiently.

3.4 HARDWARE REQUIREMENTS

In hardware requirement I required all those components which will provide me the platform for the development of the project. The minimum hardware required for the development of this project is as follows :

RAM – minimum 4 or 8 GB.
HARD DISK – minimum 20 GB.
PROCESSOR -- minimum Intel i3 or i5.
SYSTEM TYPE -- 64-bit Operating System, x64-based processor.
MONITOR -- Acer Display Pannel 17(inches).
CONNECTION -- Local Broadband Connection, Internet.

3.5 SOFTWARE REQUIREMENTS

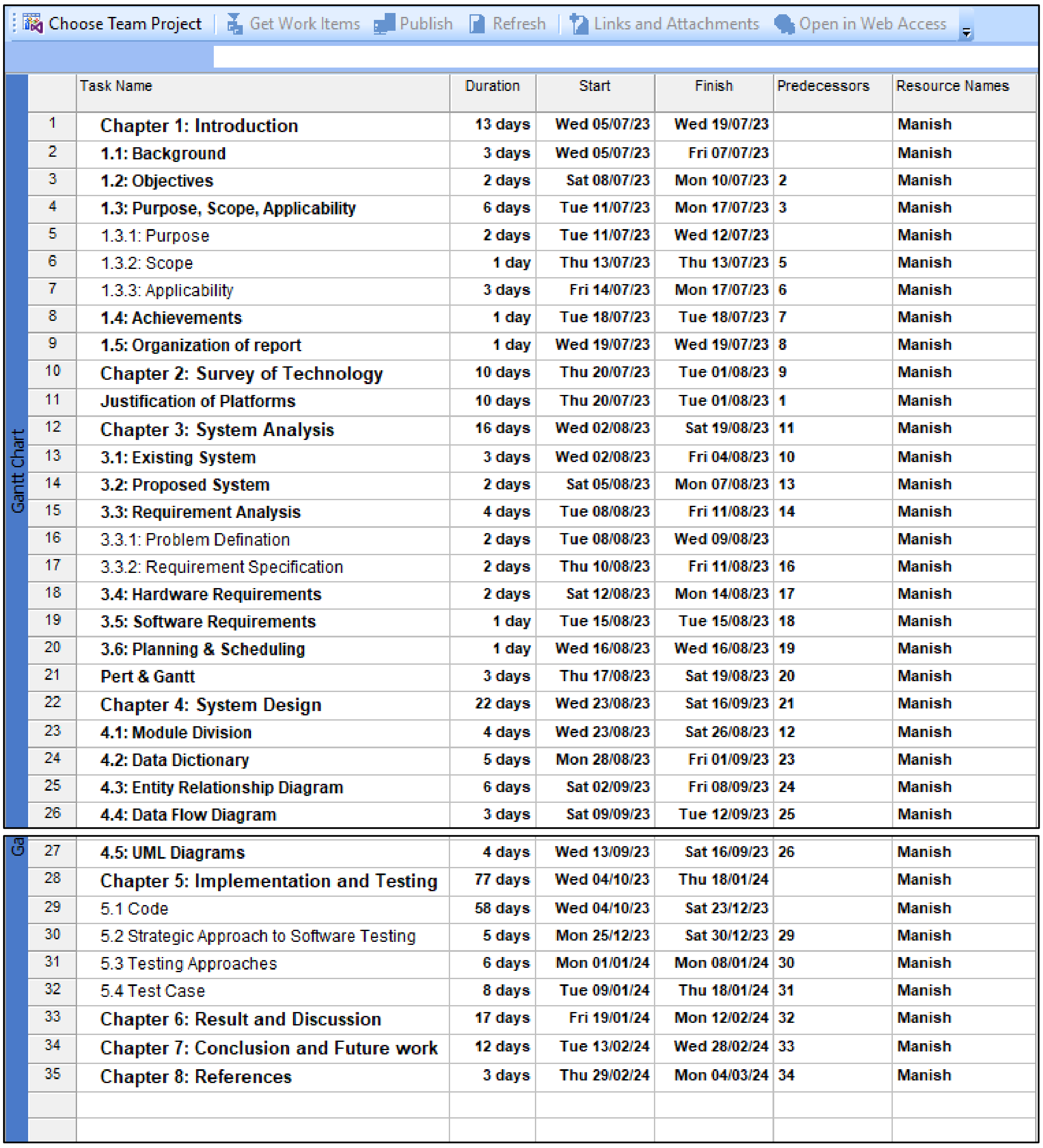
Software’s can be defined as programs which run on my computer. Various software’s are needed in this project for its development.
Which are as follows --

OPERATING SYSTEM -- Windows 10 / 11
OTHERS -- Visual Studio Code,
Node,Js, Chrome, WBS Chart Pro,
Microsoft Project.

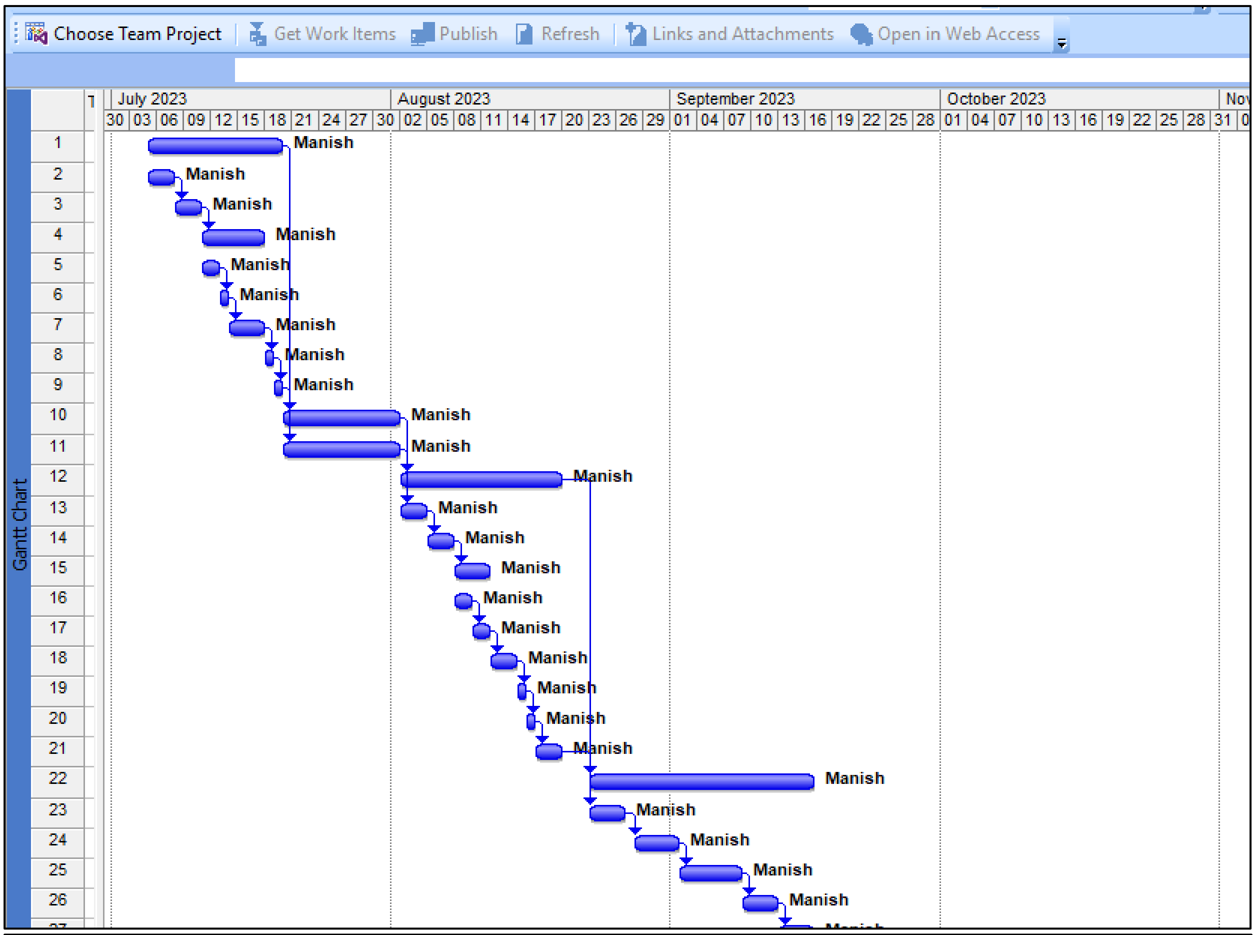
I will be using VS code as my front hand because it is easier to use and provide features or Various extensions to the users for the development of the project.

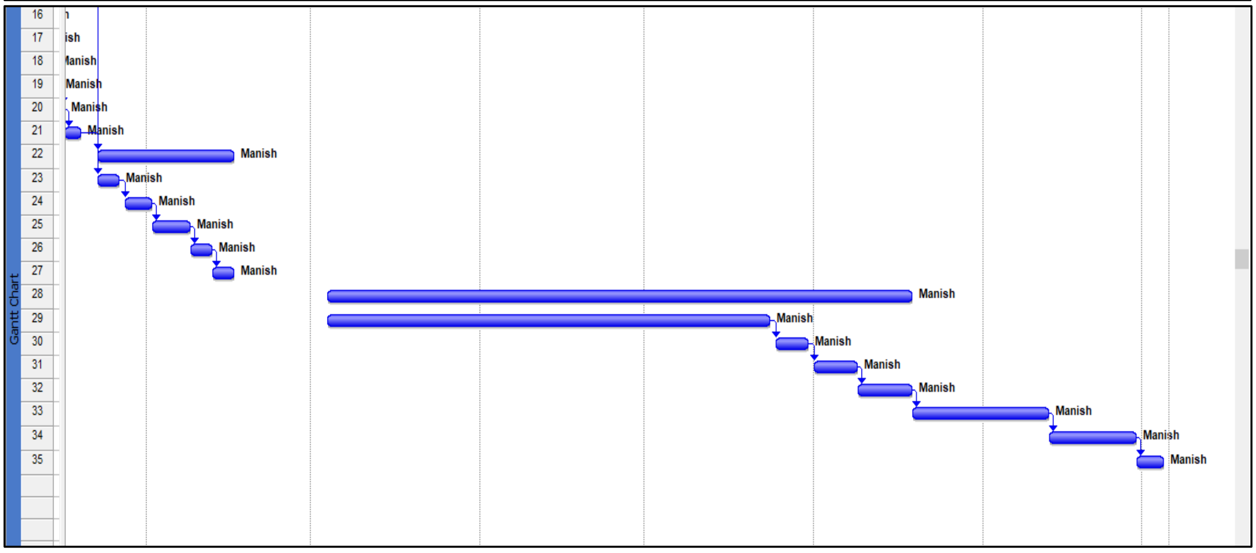
3.6 PLANNING & SCHEDULING

Gantt Chart :

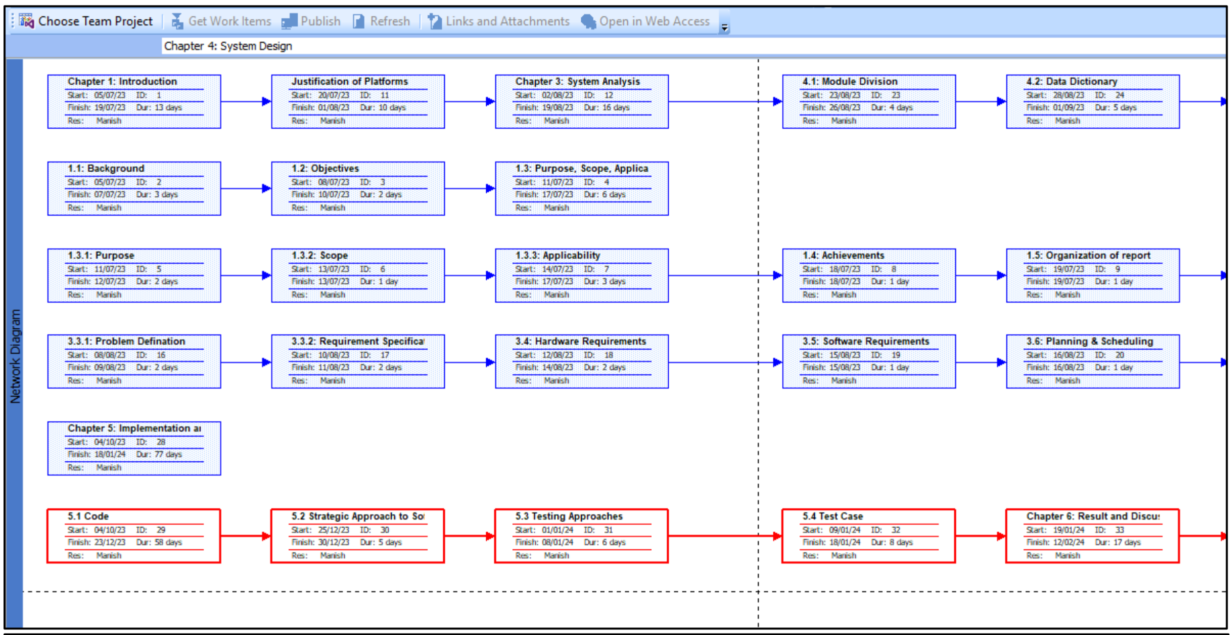


	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	Chapter 1: Introduction	13 days	Wed 05/07/23	Wed 19/07/23		Manish
2	1.1: Background	3 days	Wed 05/07/23	Fri 07/07/23		Manish
3	1.2: Objectives	2 days	Sat 08/07/23	Mon 10/07/23	2	Manish
4	1.3: Purpose, Scope, Applicability	6 days	Tue 11/07/23	Mon 17/07/23	3	Manish
5	1.3.1: Purpose	2 days	Tue 11/07/23	Wed 12/07/23		Manish
6	1.3.2: Scope	1 day	Thu 13/07/23	Thu 13/07/23	5	Manish
7	1.3.3: Applicability	3 days	Fri 14/07/23	Mon 17/07/23	6	Manish
8	1.4: Achievements	1 day	Tue 18/07/23	Tue 18/07/23	7	Manish
9	1.5: Organization of report	1 day	Wed 19/07/23	Wed 19/07/23	8	Manish
10	Chapter 2: Survey of Technology	10 days	Thu 20/07/23	Tue 01/08/23	9	Manish
11	Justification of Platforms	10 days	Thu 20/07/23	Tue 01/08/23	1	Manish
12	Chapter 3: System Analysis	16 days	Wed 02/08/23	Sat 19/08/23	11	Manish
13	3.1: Existing System	3 days	Wed 02/08/23	Fri 04/08/23	10	Manish
14	3.2: Proposed System	2 days	Sat 05/08/23	Mon 07/08/23	13	Manish
15	3.3: Requirement Analysis	4 days	Tue 08/08/23	Fri 11/08/23	14	Manish
16	3.3.1: Problem Defination	2 days	Tue 08/08/23	Wed 09/08/23		Manish
17	3.3.2: Requirement Specification	2 days	Thu 10/08/23	Fri 11/08/23	16	Manish
18	3.4: Hardware Requirements	2 days	Sat 12/08/23	Mon 14/08/23	17	Manish
19	3.5: Software Requirements	1 day	Tue 15/08/23	Tue 15/08/23	18	Manish
20	3.6: Planning & Scheduling	1 day	Wed 16/08/23	Wed 16/08/23	19	Manish
21	Pert & Gantt	3 days	Thu 17/08/23	Sat 19/08/23	20	Manish
22	Chapter 4: System Design	22 days	Wed 23/08/23	Sat 16/09/23	21	Manish
23	4.1: Module Division	4 days	Wed 23/08/23	Sat 26/08/23	12	Manish
24	4.2: Data Dictionary	5 days	Mon 28/08/23	Fri 01/09/23	23	Manish
25	4.3: Entity Relationship Diagram	6 days	Sat 02/09/23	Fri 08/09/23	24	Manish
26	4.4: Data Flow Diagram	3 days	Sat 09/09/23	Tue 12/09/23	25	Manish
27	4.5: UML Diagrams	4 days	Wed 13/09/23	Sat 16/09/23	26	Manish
28	Chapter 5: Implementation and Testing	77 days	Wed 04/10/23	Thu 18/01/24		Manish
29	5.1 Code	58 days	Wed 04/10/23	Sat 23/12/23		Manish
30	5.2 Strategic Approach to Software Testing	5 days	Mon 25/12/23	Sat 30/12/23	29	Manish
31	5.3 Testing Approaches	6 days	Mon 01/01/24	Mon 08/01/24	30	Manish
32	5.4 Test Case	8 days	Tue 09/01/24	Thu 18/01/24	31	Manish
33	Chapter 6: Result and Discussion	17 days	Fri 19/01/24	Mon 12/02/24	32	Manish
34	Chapter 7: Conclusion and Future work	12 days	Tue 13/02/24	Wed 28/02/24	33	Manish
35	Chapter 8: References	3 days	Thu 29/02/24	Mon 04/03/24	34	Manish


Choose Team Project
Get Work Items
Publish
Refresh
Links and Attachments
Open in Web Access
Gantt Chart
July 2023
August 2023
September 2023
October 2023
Manish


Gantt Chart
Manish

Pert Chart :



Chapter 4: System Design

Chapter 1: Introduction
Start: 05/07/23 ID: 1
Finish: 19/07/23 Dur: 13 days
Res: Manish

Justification of Platforms
Start: 20/07/23 ID: 11
Finish: 01/08/23 Dur: 10 days
Res: Manish

Chapter 3: System Analysis
Start: 02/08/23 ID: 12
Finish: 19/08/23 Dur: 16 days
Res: Manish

4.1: Module Division
Start: 23/08/23 ID: 23
Finish: 26/08/23 Dur: 4 days
Res: Manish

4.2: Data Dictionary
Start: 28/08/23 ID: 24
Finish: 01/09/23 Dur: 5 days
Res: Manish

1.1: Background
Start: 05/07/23 ID: 2
Finish: 07/07/23 Dur: 3 days
Res: Manish

1.2: Objectives
Start: 08/07/23 ID: 3
Finish: 10/07/23 Dur: 2 days
Res: Manish

1.3: Purpose, Scope, Applica
Start: 11/07/23 ID: 4
Finish: 17/07/23 Dur: 6 days
Res: Manish

1.3.1: Purpose
Start: 11/07/23 ID: 5
Finish: 12/07/23 Dur: 2 days
Res: Manish

1.3.2: Scope
Start: 13/07/23 ID: 6
Finish: 13/07/23 Dur: 1 day
Res: Manish

1.3.3: Applicability
Start: 14/07/23 ID: 7
Finish: 17/07/23 Dur: 3 days
Res: Manish

1.4: Achievements
Start: 18/07/23 ID: 8
Finish: 18/07/23 Dur: 1 day
Res: Manish

1.5: Organization of report
Start: 19/07/23 ID: 9
Finish: 19/07/23 Dur: 1 day
Res: Manish

3.3.1: Problem Defination
Start: 08/08/23 ID: 16
Finish: 09/08/23 Dur: 2 days
Res: Manish

3.3.2: Requirement Specifica
Start: 10/08/23 ID: 17
Finish: 11/08/23 Dur: 2 days
Res: Manish

3.4: Hardware Requirements
Start: 12/08/23 ID: 18
Finish: 14/08/23 Dur: 2 days
Res: Manish

3.5: Software Requirements
Start: 15/08/23 ID: 19
Finish: 15/08/23 Dur: 1 day
Res: Manish

3.6: Planning & Scheduling
Start: 16/08/23 ID: 20
Finish: 16/08/23 Dur: 1 day
Res: Manish

Chapter 5: Implementation ai
Start: 04/10/23 ID: 28
Finish: 18/01/24 Dur: 77 days
Res: Manish

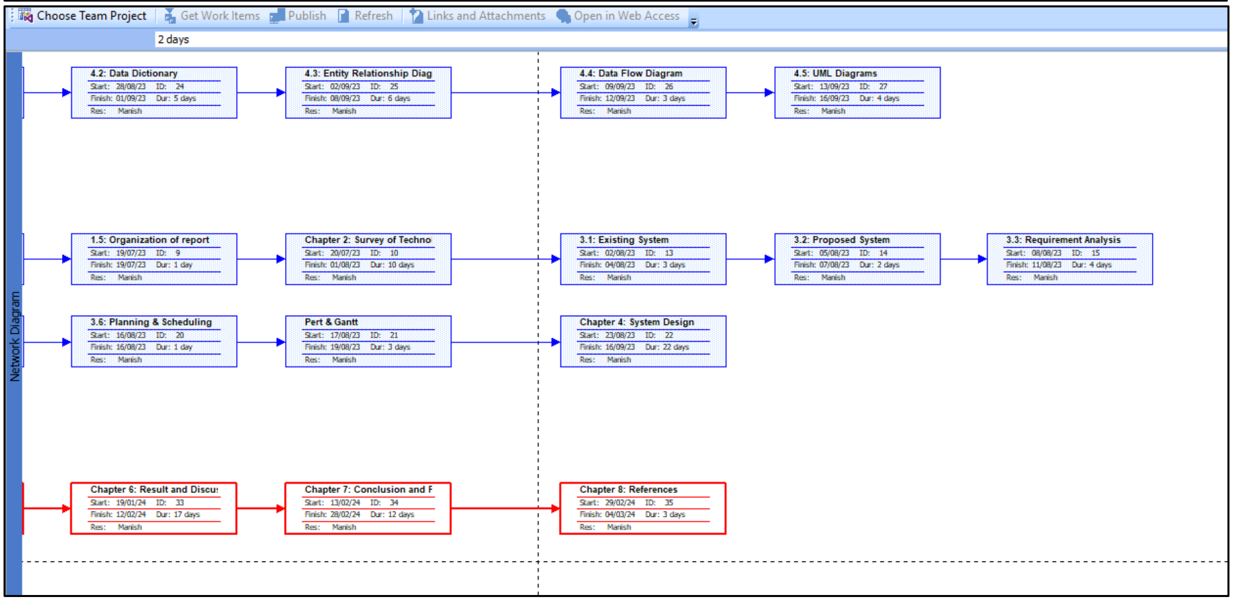
5.1 Code
Start: 04/10/23 ID: 29
Finish: 23/12/23 Dur: 58 days
Res: Manish

5.2 Strategic Approach to So
Start: 25/12/23 ID: 30
Finish: 30/12/23 Dur: 5 days
Res: Manish

5.3 Testing Approaches
Start: 01/01/24 ID: 31
Finish: 08/01/24 Dur: 6 days
Res: Manish

5.4 Test Case
Start: 09/01/24 ID: 32
Finish: 18/01/24 Dur: 8 days
Res: Manish

Chapter 6: Result and Discu
Start: 19/01/24 ID: 33
Finish: 12/02/24 Dur: 17 days
Res: Manish



2 days

4.2: Data Dictionary
Start: 28/08/23 ID: 24
Finish: 01/09/23 Dur: 5 days
Res: Manish

4.3: Entity Relationship Diag
Start: 02/09/23 ID: 25
Finish: 08/09/23 Dur: 6 days
Res: Manish

4.4: Data Flow Diagram
Start: 09/09/23 ID: 26
Finish: 12/09/23 Dur: 3 days
Res: Manish

4.5: UML Diagrams
Start: 13/09/23 ID: 27
Finish: 16/09/23 Dur: 4 days
Res: Manish

1.5: Organization of report
Start: 19/07/23 ID: 9
Finish: 19/07/23 Dur: 1 day
Res: Manish

Chapter 2: Survey of Techno
Start: 20/07/23 ID: 10
Finish: 01/08/23 Dur: 10 days
Res: Manish

3.1: Existing System
Start: 02/08/23 ID: 13
Finish: 04/08/23 Dur: 3 days
Res: Manish

3.2: Proposed System
Start: 05/08/23 ID: 14
Finish: 07/08/23 Dur: 2 days
Res: Manish

3.3: Requirement Analysis
Start: 08/08/23 ID: 15
Finish: 11/08/23 Dur: 4 days
Res: Manish

3.6: Planning & Scheduling
Start: 16/08/23 ID: 20
Finish: 16/08/23 Dur: 1 day
Res: Manish

Pert & Gantt
Start: 17/08/23 ID: 21
Finish: 19/08/23 Dur: 3 days
Res: Manish

Chapter 4: System Design
Start: 23/08/23 ID: 22
Finish: 16/09/23 Dur: 22 days
Res: Manish

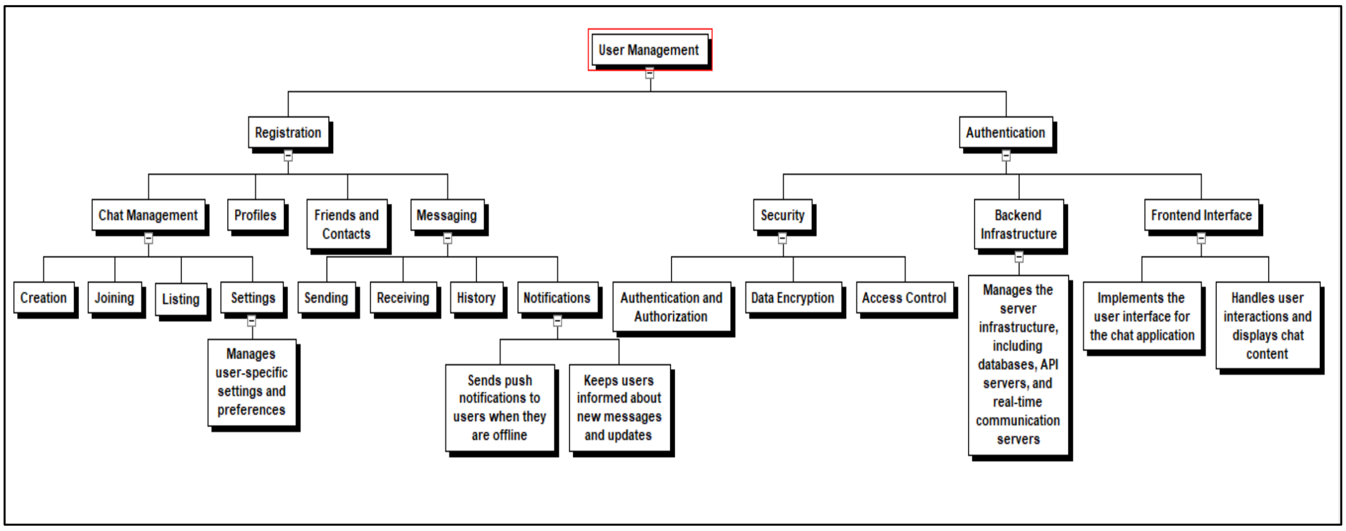
Chapter 6: Result and Discu
Start: 19/01/24 ID: 33
Finish: 12/02/24 Dur: 17 days
Res: Manish

Chapter 7: Conclusion and F
Start: 13/02/24 ID: 34
Finish: 28/02/24 Dur: 12 days
Res: Manish

Chapter 8: References
Start: 29/02/24 ID: 35
Finish: 04/03/24 Dur: 3 days
Res: Manish

CHAPTER 4 : SYSTEM DESIGN

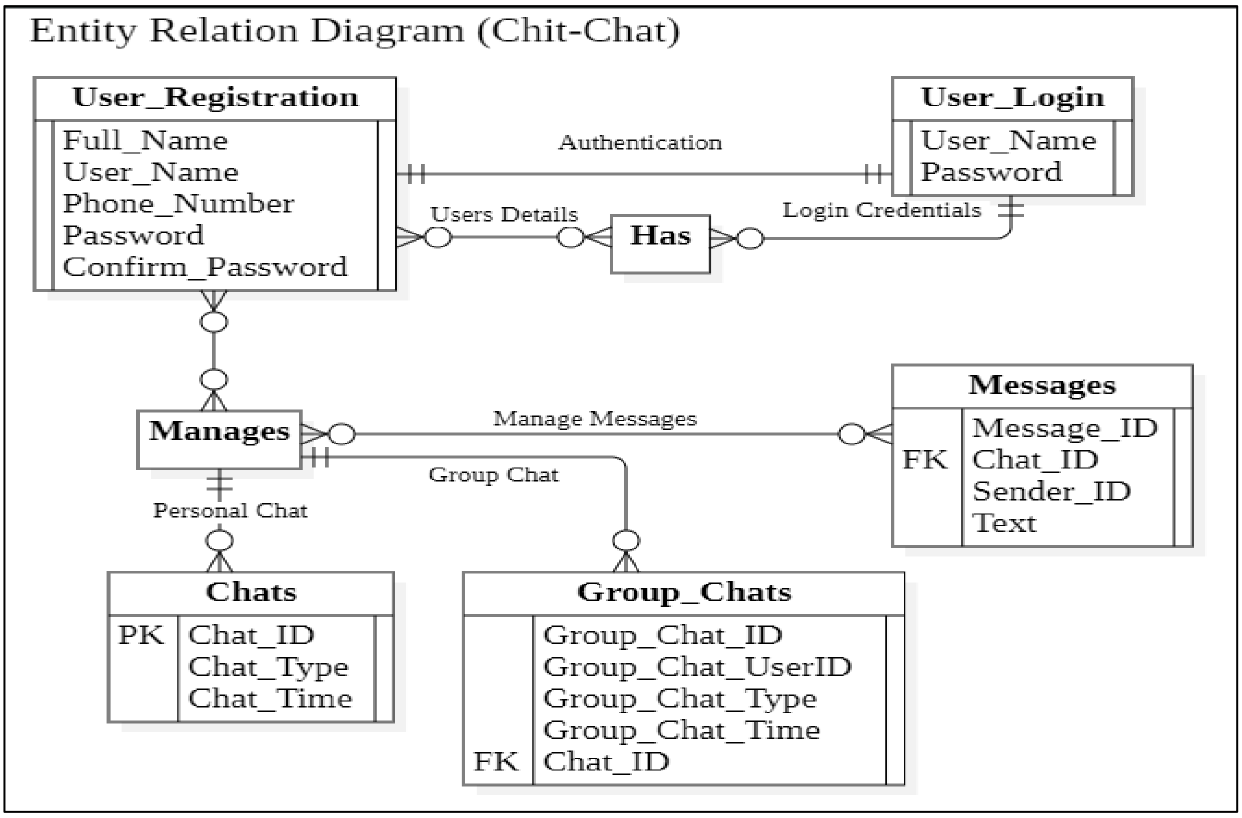
4.1 MODULE DIVISION


User Management
Registration
Authentication
Chat Management
Profiles
Friends and Contacts
Messaging
Security
Backend Infrastructure
Frontend Interface
Creation
Joining
Listing
Settings
Sending
Receiving
History
Notifications
Authentication and Authorization
Data Encryption
Access Control
Manages the server infrastructure, including databases, API servers, and real-time communication servers
Implements the user interface for the chat application
Handles user interactions and displays chat content
Manages user-specific settings and preferences
Sends push notifications to users when they are offline
Keeps users informed about new messages and updates

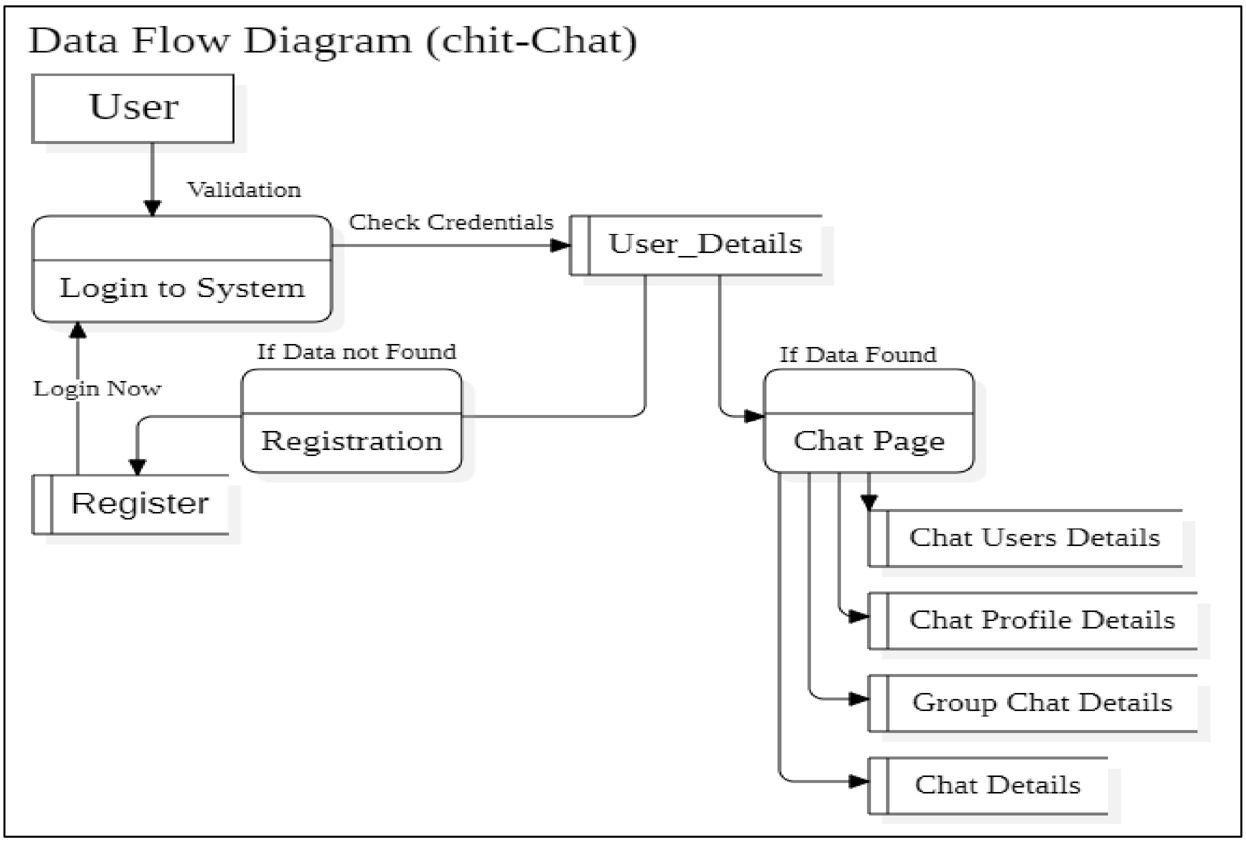
4.2 DATA DICTIONARY

Column Name	Data Type	Description
Full Name	Varchar(50)	Full name of the User.
Username	Varchar(50)	Username of the User by their choice.
Password	Varchar(8)	Password of the user.
Confirm Password	Varchar(8)	Password given by User that would be Confirmed.
Phone Number	Varchar(10)	Phone Number of the User.
Message	Varchar	Messages of the Users.
Chat	Varchar	Chats between Users.

4.3 ENTITY RELATIONSHIP DAIGRAM

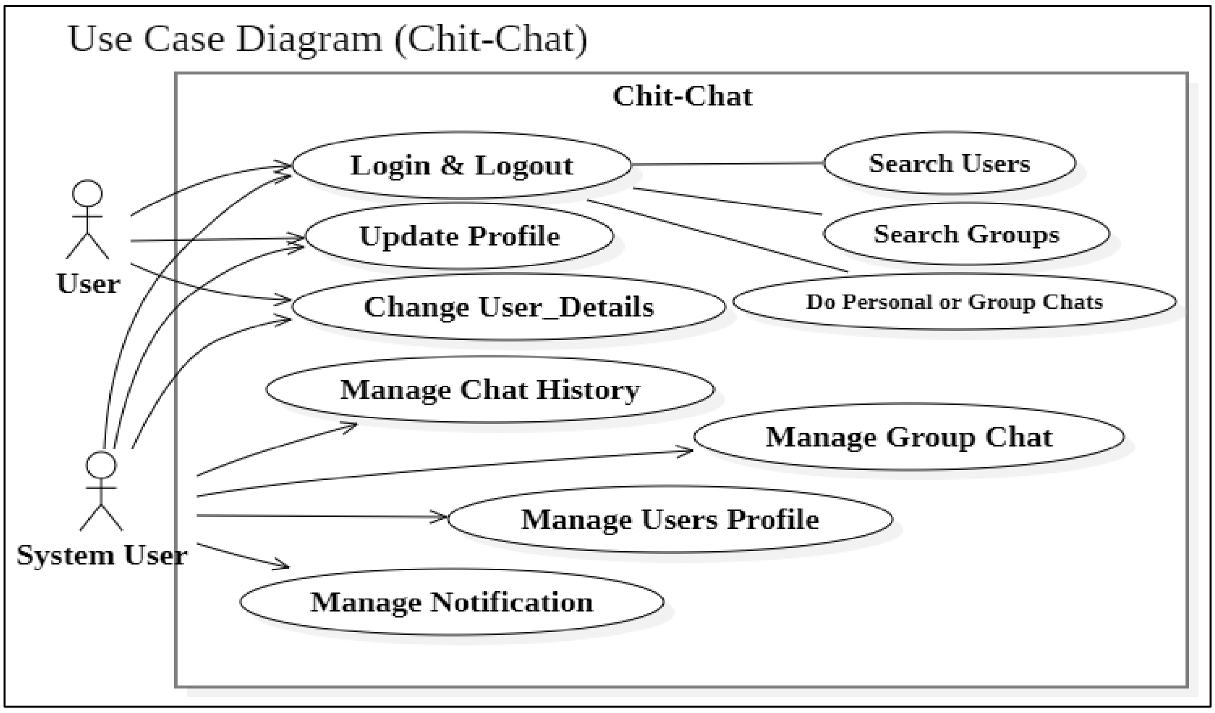


4.4 DATA FLOW DAIGRAM

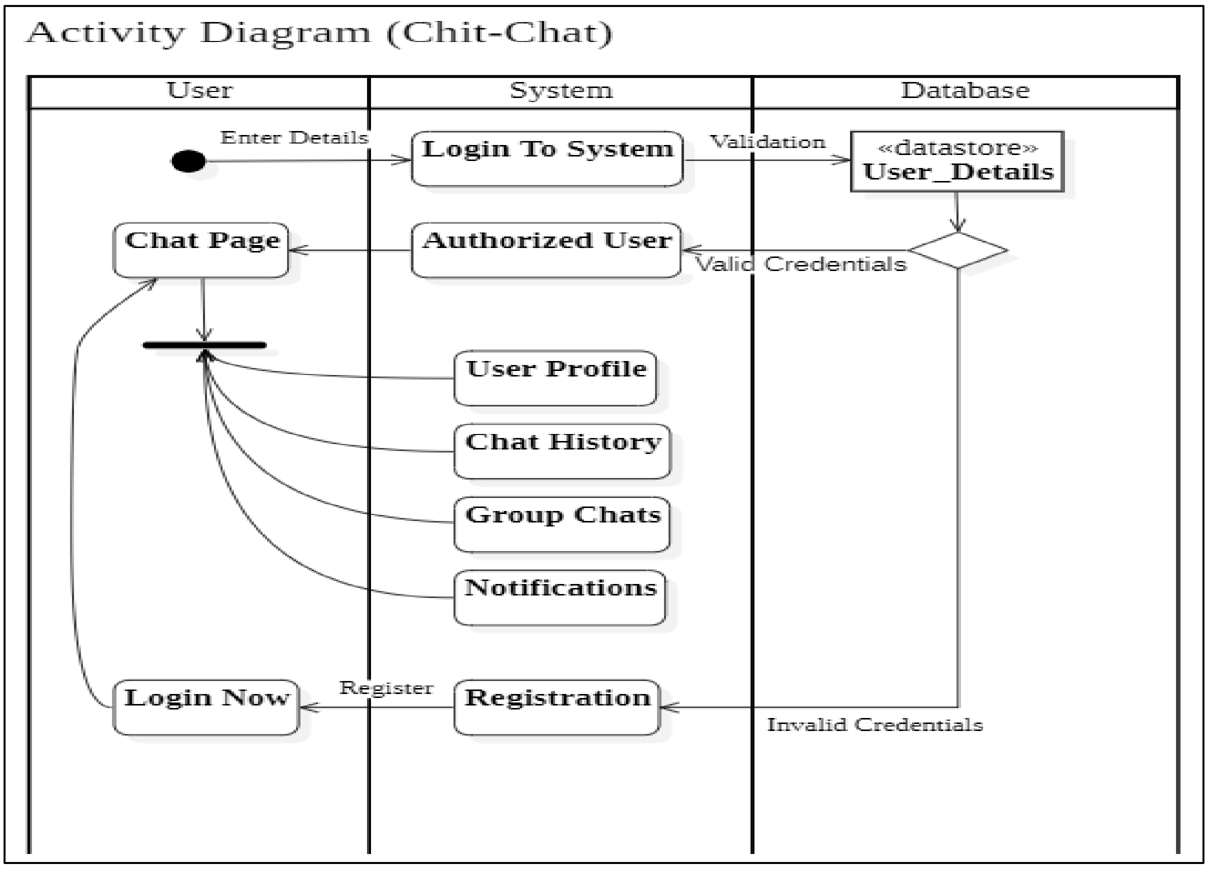


4.5 UML DAIGRAMS

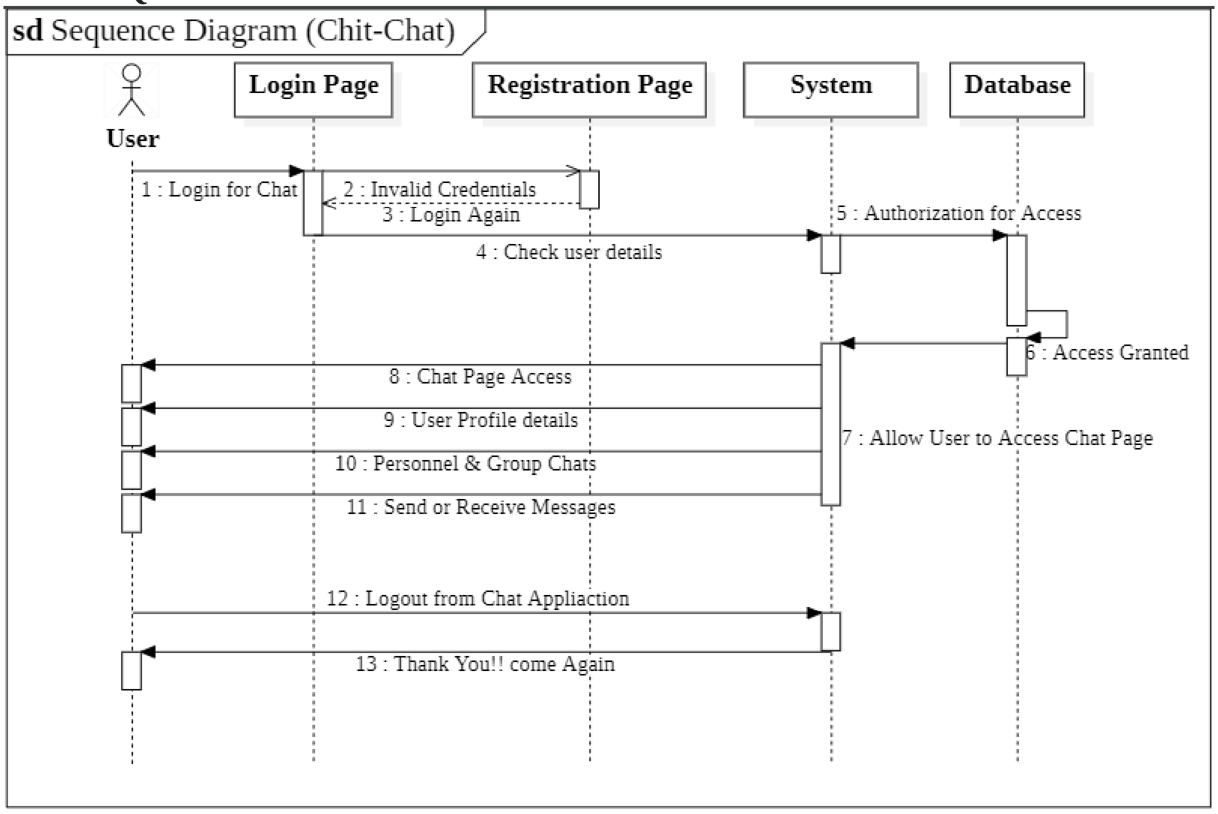
4.5.1 USE CASE DAIGRAM



4.5.2 ACTIVITY DAIGRAM

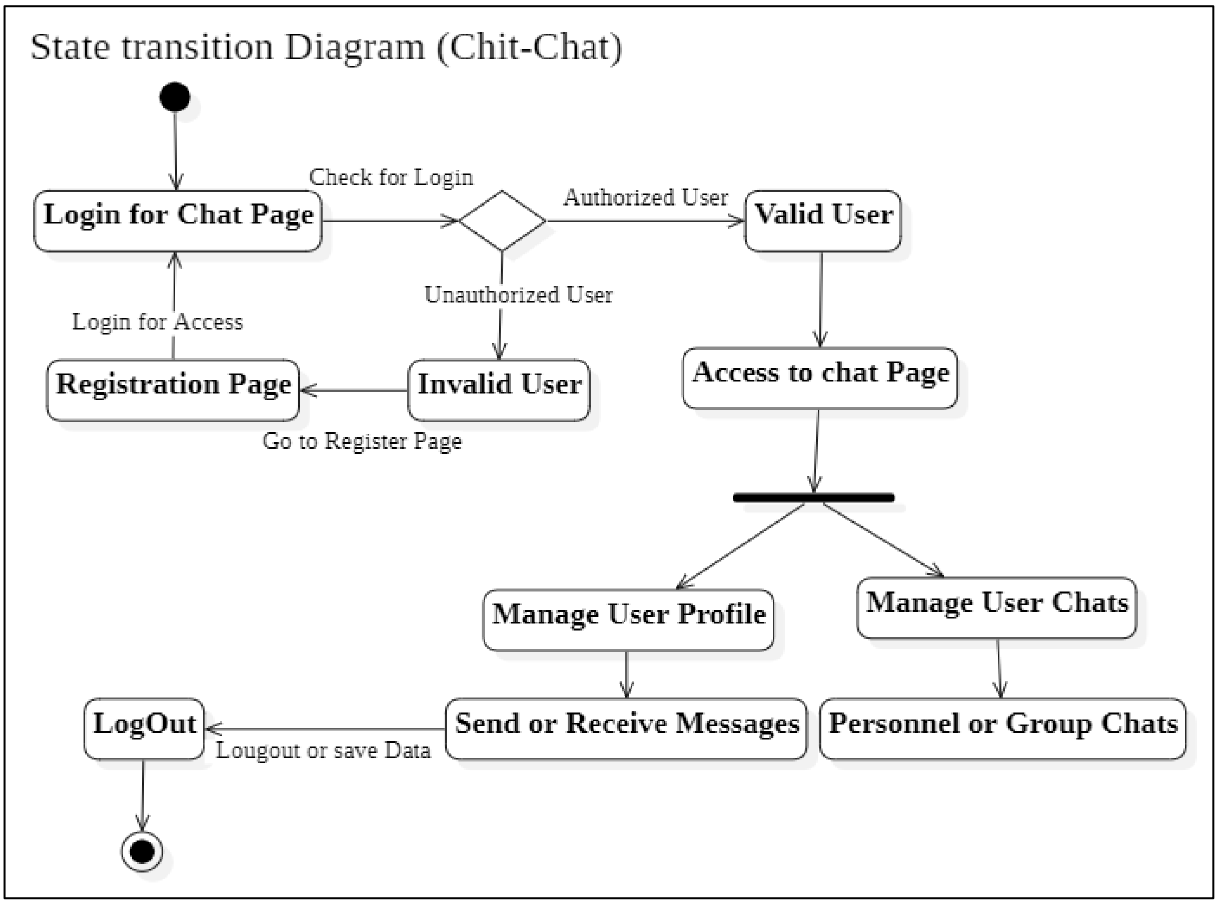


4.5.3 SEQUENCE DAIGRAM



sd Sequence Diagram (Chit-Chat)
User
Login Page
Registration Page
System
Database
1 : Login for Chat
2 : Invalid Credentials
3 : Login Again
4 : Check user details
5 : Authorization for Access
6 : Access Granted
7 : Allow User to Access Chat Page
8 : Chat Page Access
9 : User Profile details
10 : Personnel & Group Chats
11 : Send or Receive Messages
12 : Logout from Chat Appliaction
13 : Thank You!! come Again

4.5.4 STATE TRANSITION DAIGRAM



Chapter 5: IMPLEMENTATION & TESTING

5.1 Code

Frontend :-

(App.js)

```
import './App.css';
import HomePage from './Pages/HomePage';
import { Route, Routes } from "react-router-dom";
import Chatpage from './Pages/Chatpage';
function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<HomePage />} exact />
        <Route path="/chats" element={<Chatpage />} />
      </Routes>
    </div>
  );
}
export default App;
```

(Signup.js)

```
import { Button } from "@chakra-ui/button";
import { FormControl, FormLabel } from "@chakra-ui/form-control";
import { Input, InputGroup, InputRightElement } from "@chakra-ui/input";
import { VStack } from "@chakra-ui/layout";
import { useToast } from "@chakra-ui/toast";
import axios from "axios";
import { useState } from "react";
import { useNavigate } from "react-router";

const Signup = () => {
    const [show, setShow] = useState(false);
    const handleClick = () => setShow(!show);
    const toast = useToast();
    const navigate = useNavigate();
    const [name, setName] = useState();
    const [email, setEmail] = useState();
    const [confirmpassword, setConfirmpassword] = useState();
    const [password, setPassword] = useState();
    const [pic, setPic] = useState();
    const [picLoading, setPicLoading] = useState(false);

    const submitHandler = async () => {
        setPicLoading(true);
              if (!name.match("[a-zA-Z]+")) {
          toast({
            title: "Please enter valid name",
            status: "warning",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          setPicLoading(false);
          return;
        }
        if (!name || !email || !password || !confirmpassword) {
          toast({
            title: "Please Fill all the Feilds",
            status: "warning",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          setPicLoading(false);
          return;
        }
        if (password !== confirmpassword) {
          toast({
            title: "Passwords Do Not Match",
            status: "warning",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          return;
        }
        console.log(name, email, password, pic);
        try {
          const config = {
            headers: {
              "Content-type": "application/json",
            },
          } const { data } = await axios.post(
            "/api/user",
            {
              name,
              email,
              password,
              pic,
            },
            config
          );
          console.log(data);
          toast({
            title: "Registration Successful",
            status: "success",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          localStorage.setItem("userInfo", JSON.stringify(data));
          setPicLoading(false);
          navigate("/chats");
        } catch (error) {
          toast({
            title: "Error Occured!",
            description: error.response.data.message,
            status: "error",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          setPicLoading(false);
        }
    };

    const postDetails = (pics) => {
        setPicLoading(true);
        if (pics === undefined) {
          toast({
            title: "Please Select an Image!",
            status: "warning",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          return;
        }
        console.log(pics);
        if (pics.type === "image/jpeg" || pics.type === "image/png") {
          const data = new FormData();
          data.append("file", pics);
          data.append("upload_preset", "chit-chat");
          data.append("cloud_name", "drcpulcqt");

          const CLOUDINARY_URL = "https://api.cloudinary.com/v1_1/drcpulcqt/image/upload";
          fetch(CLOUDINARY_URL, {
            method: "post",
            body: data,
          })
            .then((res) => res.json())
            .then((data) => {
              setPic(data.url.toString());
              console.log(data.url.toString());
              setPicLoading(false);
            })
            .catch((err) => {
              console.log(err);
              setPicLoading(false);
            });
        } else {
          toast({
            title: "Please Select an Image!",
            status: "warning",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          setPicLoading(false);
          return;
        }
    };

    return (
        <VStack spacing="5px" fontFamily="cursive">
          <FormControl id="first-name" isRequired >
            <FormLabel>Name</FormLabel>
            <Input
              autoComplete="off"
              placeholder="Enter Your Name"
              onChange={(e) => setName(e.target.value)}
              color="#f4e409"
            />
          </FormControl>
          <FormControl id="email" isRequired>
            <FormLabel>Email Address</FormLabel>
            <Input
              autoComplete="off"
        type="email"
        placeholder="Enter Your Email Address"
        onChange={(e) => setEmail(e.target.value)}
        color="#f4e409"
    />
</FormControl>
<FormControl id="password" isRequired>
  <FormLabel>Password</FormLabel>
  <InputGroup size="md">
    <Input
      autoComplete="off"
      type={show ? "text" : "password"}
      placeholder="Enter Password"
      onChange={(e) => setPassword(e.target.value)}
      color="#f4e409"
    />
    <InputRightElement width="4.5rem">
      <Button h="1.75rem" size="sm" onClick={handleClick}>
        {show ? "Hide" : "Show"}
      </Button>
    </InputRightElement>
  </InputGroup>
</FormControl>
<FormControl id="password" isRequired>
  <FormLabel>Confirm Password</FormLabel>
  <InputGroup size="md">
    <Input
      autoComplete="off"
      type={show ? "text" : "password"}
      placeholder="Confirm password"
      onChange={(e) => setConfirmpassword(e.target.value)}
      color="#f4e409"
    />
    <InputRightElement width="4.5rem">
      <Button h="1.75rem" size="sm" onClick={handleClick}>
        {show ? "Hide" : "Show"}
      </Button>
    </InputRightElement>
  </InputGroup>
</FormControl>
<FormControl id="pic">
  <FormLabel>Upload your Picture</FormLabel>
  <Input
    type="file"
    p={1.5}
    accept="image/*"
    onChange={(e) => postDetails(e.target.files[0])}
  />
</FormControl>
<Button
  bg="#ffbc42"
  width="100%"
  fontSize="1.5rem"
  borderRadius="15px"
  style={{ marginTop: 15 }}
  onClick={submitHandler}
  isLoading={picLoading}
>
  Sign Up
</Button>
```

```
        </VStack>
    );
};
export default Signup;
```

(Login.js)

```
import { Button } from "@chakra-ui/button";
import { FormControl, FormLabel } from "@chakra-ui/form-control";
import { Input, InputGroup, InputRightElement } from "@chakra-ui/input";
import { VStack } from "@chakra-ui/layout";
import { useState } from "react";
import { useToast } from "@chakra-ui/react";
import { useNavigate } from "react-router-dom";
import { ChatState } from "../../Context/ChatProvider";
const Login = () => {
    const [show, setShow] = useState(false);
    const handleClick = () => setShow(!show);
    const toast = useToast();
    const [email, setEmail] = useState();
    const [password, setPassword] = useState();
    const [loading, setLoading] = useState(false);
    const navigate = useNavigate();
    const { setUser } = ChatState();

    const submitHandler = async () => {
        setLoading(true);
        if (!email || !password) {
          toast({
            title: "Please Fill all the Feilds",
            status: "warning",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          setLoading(false);
          return;
        }
   try {
          const config = {
            headers: {
              "Content-type": "application/json",
            },
          }  const { data } = await axios.post(
              "api/user/login",
            { email, password },
            config
          );
          toast({
            title: "Login Successful",
            status: "success",
            duration: 5000,
            isClosable: true,
            position: "bottom",
          });
          setUser(data);
          localStorage.setItem("userInfo", JSON.stringify(data));
          setLoading(false);
          navigate("/chats");

        } catch (error) {
```

```
      toast({
        title: "Error Occured!",
        description: error.response.data.message,
        status: "error",
        duration: 5000,
        isClosable: true,
        position: "bottom",
      });
      setLoading(false);
    }
  };

  return (
    <VStack spacing="10px" fontFamily="cursive">
      <FormControl id="email" isRequired>
        <FormLabel>Email Address</FormLabel>
        <Input
          autoComplete="off"
          value={email}
          type="email"
          placeholder="Enter Your Email Address"
          onChange={(e) => setEmail(e.target.value)}
          color="#f4e409"
        />
      </FormControl>
      <FormControl id="password" isRequired>
        <FormLabel>Password</FormLabel>
        <InputGroup size="md">
          <Input
            autoComplete="off"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            type={show ? "text" : "password"}
            placeholder="Enter password"
            color="#f4e409"
          />
          <InputRightElement width="4.5rem">
            <Button h="1.75rem" size="sm" onClick={handleClick}>
              {show ? "Hide" : "Show"}
            </Button>
          </InputRightElement>
        </InputGroup>
      </FormControl>
      <Button
        bg="#ffbc42"
        width="100%"
        fontFamily="cursive"
        fontSize="1.5rem"
        borderRadius="15px"
        style={{ marginTop: 15 }}
        onClick={submitHandler}
        isLoading={loading}
      >
        Login
      </Button>
      <Button
        variant="solid"
        bg="#fbfbfb"
        width="100%"
        fontSize="1.1rem"
```

```
            borderRadius="15px"
            fontFamily="cursive"
            onClick={() => {
              setEmail("guest@example.com");
              setPassword("123456");
            }}
          >
            Get Guest User Credentials
          </Button>
        </VStack>
      );
};
export default Login;
```

(HomePage.js)

```
import { Box, Container, Tab, TabList, TabPanel,  TabPanels, Tabs, Text } from "@chakra-ui/react";
import { useEffect } from "react";
import { useNavigate } from "react-router";
import Login from "../Components/Authentication/Login";
import Signup from "../Components/Authentication/Signup";

function HomePage() {
    const history = useNavigate();
    useEffect(() => {
        const user = JSON.parse(localStorage.getItem("userInfo"));

        if (user) history.push("/chats");
    }, [history]);

    return (
        <Container maxW="xl" centerContent>
            <Box
                d="flex"
                justifyContent="center"
                p={3}
                bg="#b5e48c"
                w="100%"
                m="40px 0 15px 0"
                borderRadius="20px"
            >
                <Text fontSize="4xl" fontFamily="Cursive" align="center">
                    Chit-Chat
                </Text>
            </Box>
            <Box bg="#000000" color="#b5e48c" w="100%" p={4} borderRadius="20px" borderWidth="8px"
borderColor="#b5e48c">
                <Tabs isFitted variant="soft-rounded" colorScheme="orange" color="#a5ffd6">
                    <TabList mb="1rem">
                        <Tab>Login</Tab>
                        <Tab>Signup</Tab>
                    </TabList>
                    <TabPanels>
                        <TabPanel>
                            <Login />
                        </TabPanel>
                        <TabPanel>
                            <Signup />
                        </TabPanel>
                    </TabPanels>
                </Tabs>
```

```
          </Box>
        </Container>
    );
}
export default HomePage;
```

(Chatpage.js)

```
import { Box } from '@chakra-ui/layout';
import { useState } from 'react';
import Chatbox from "../Components/Chatbox";
import  MyChats from "../Components/MyChats";
import SideDrawer from "../Components/miscellaneous/SideDrawer";
import { ChatState } from "../Context/ChatProvider";

const Chatpage = () => {
    const [fetchAgain, setFetchAgain] = useState(false);
    const { user } = ChatState();

    return (
        <div style={{ width: "100% "}}>
            {user && <SideDrawer />}
            <Box d="flex" justifyContent="space-between" w="100%" h="91.5vh" p="10px">
                {user && <MyChats fetchAgain={fetchAgain} />}
                {user && (
                    <Chatbox fetchAgain={fetchAgain} setFetchAgain={setFetchAgain} />
                )}
            </Box>
        </div>
    );
};
export default Chatpage;
```

(MyChat.js)

```
import { AddIcon } from "@chakra-ui/icons";
import { Box, Stack, Text } from "@chakra-ui/layout";
import { useToast } from "@chakra-ui/toast";
import { useEffect, useState } from "react";
import { getSender } from "../config/ChatLogics";
import ChatLoading from "./ChatLoading";
import GroupChatModal from "./miscellaneous/GroupChatModel";
import { Button } from "@chakra-ui/react";
import { ChatState } from "../Context/ChatProvider";

const MyChats = ({ fetchAgain }) => {
    const [loggedUser, setLoggedUser] = useState();
    const { selectedChat, setSelectedChat, user, chats, setChats } = ChatState();
    const toast = useToast();
    const fetchChats = async () => {
      // console.log(user._id);
      try {
        const config = {
              headers: {
                          Authorization: `Bearer ${user.token}`,
          },
        };
        const { data } = await axios.get("/api/chat", config);
        setChats(data);
      } catch (error) {
        toast({
          title: "Error Occured!",
          description: "Failed to Load the chats",
          status: "error",
          duration: 5000,
          isClosable: true,
          position: "bottom-left",
        });
      }
    };
    useEffect(() => {
      setLoggedUser(JSON.parse(localStorage.getItem("userInfo")));
      fetchChats();
      // eslint-disable-next-line
    }, [fetchAgain]);

    return (
      <Box
        d={{ base: selectedChat ? "none" : "flex", md: "flex" }}
        flexDir="column"
        alignItems="center"
        p={3}
        bg="white"
        w={{ base: "100%", md: "31%" }}
        borderRadius="lg"
        borderWidth="1px"
      >
        <Box
          pb={3}
          px={3}
          fontSize={{ base: "28px", md: "30px" }}
          fontFamily="Work sans"
          d="flex"
          w="100%"
          justifyContent="space-between"
          alignItems="center"
        >
          My Chats
          <GroupChatModal>
            <Button
              d="flex"
              fontSize={{ base: "17px", md: "10px", lg: "17px" }}
              rightIcon={<AddIcon />}
            >
              New Group Chat
            </Button>
          </GroupChatModal>
        </Box>
        <Box
          d="flex"
          flexDir="column"
          p={3}
          bg="#F8F8F8"
          w="100%"
          h="100%"
          borderRadius="lg"
          overflowY="hidden"
        >
        {chats ? (
          <Stack overflowY="scroll">
            {chats.map((chat) => (
              <Box
                onClick={() => setSelectedChat(chat)}
                cursor="pointer"
                bg={selectedChat === chat ? "#38B2AC" : "#E8E8E8"}
                color={selectedChat === chat ? "white" : "black"}
                px={3}
                py={2}
                borderRadius="lg"
                key={chat._id}
              >
                <Text>
                  {!chat.isGroupChat
                    ? getSender(loggedUser, chat.users)
                    : chat.chatName}
                </Text>
                {chat.latestMessage && (
                  <Text fontSize="xs">
                    <b>{chat.latestMessage.sender.name} : </b>
                    {chat.latestMessage.content.length > 50
                      ? chat.latestMessage.content.substring(0, 51) + "..."
                      : chat.latestMessage.content}
                  </Text>
                )}
              </Box>
            ))}
          </Stack>
        ) : (
          <ChatLoading />
        )}
      </Box>
    </Box>
   );
};
export default MyChats;
```

(ChatLoading.js)

```
import { Stack } from "@chakra-ui/layout";
import { Skeleton } from "@chakra-ui/skeleton";
const ChatLoading = () => {
  return (
    <Stack>
      <Skeleton height="45px" />    <Skeleton height="45px" />
      <Skeleton height="45px" />    <Skeleton height="45px" />
      <Skeleton height="45px" />    <Skeleton height="45px" />
      <Skeleton height="45px" />    <Skeleton height="45px" />
      <Skeleton height="45px" />    <Skeleton height="45px" />
      <Skeleton height="45px" />    <Skeleton height="45px" / </Stack>
  );
};
export default ChatLoading
```

(Chatbox.js)

```
import { Box } from "@chakra-ui/layout";
import "./styles.css";
import SingleChat from "./SingleChat";
import { ChatState } from "../Context/ChatProvider";
const Chatbox = ({ fetchAgain, setFetchAgain }) => {
  const { selectedChat } = ChatState();

  return (
    <Box
      d={{ base: selectedChat ? "flex" : "none", md: "flex" }}
      alignItems="center"
      flexDir="column"
      p={3}
      bg="white"
      w={{ base: "100%", md: "68%" }}
      borderRadius="lg"
      borderWidth="1px"
    >
      <SingleChat fetchAgain={fetchAgain} setFetchAgain={setFetchAgain} />
    </Box>
  );
};
export default Chatbox;
```

(ScrollableChat.js)

```
import { Avatar } from "@chakra-ui/avatar";
import { Tooltip } from "@chakra-ui/tooltip";
import ScrollableFeed from "react-scrollable-feed";
import {
  isLastMessage,
  isSameSender,
  isSameSendMargin,
  isSameUser,
} from "../config/ChatLogics";
import { ChatState } from "../Context/ChatProvider";

const ScrollableChat = ({ messages }) => {
    const { user } = ChatState();

    return (
      <ScrollableFeed>
        {messages &&
          messages.map((m, i) => (
            <div style={{ display: "flex" }} key={m._id}>
              {(isSameSender(messages, m, i, user._id) ||
                isLastMessage(messages, i, user._id)) && (
                <Tooltip label={m.sender.name} placement="bottom-start" hasArrow>
                  <Avatar
                    mt="7px"
                    mr={1}
                    size="sm"
                    cursor="pointer"
                    name={m.sender.name}
                    src={m.sender.pic}
                  />
                </Tooltip>
              )}
              <span
                style={{
                  backgroundColor: `${
                    m.sender._id === user._id ? "#BEE3F8" : "#B9F5D0"
                  }`,
                  marginLeft: isSameSendMargin(messages, m, i, user._id),
                  marginTop: isSameUser(messages, m, i, user._id) ? 3 : 10,
                  borderRadius: "20px",
                  padding: "5px 15px",
                  maxWidth: "75%",
                }}
              >
                {m.content}
              </span>
            </div>
          ))}
      </ScrollableFeed>
    );
};

export default ScrollableChat;
```

(ChatProvider.js)

```
import React, { createContext, useContext, useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
const ChatContext = createContext();

const ChatProvider = ({ children }) => {
  const [selectedChat, setSelectedChat] = useState();
  const [user, setUser] = useState();
  const [notification, setNotification] = useState([]);
  const [chats, setChats] = useState();
  const navigate = useNavigate();

  useEffect(() => {
    const userInfo = JSON.parse(localStorage.getItem("userInfo"));
    setUser(userInfo);

    if (!userInfo) navigate("/");
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [navigate]);

  return (
    <ChatContext.Provider
      value={{
        selectedChat,
        setSelectedChat,
        user,
        setUser,
        notification,
        setNotification,
        chats,
        setChats,
      }}
    >
      {children}
    </ChatContext.Provider>
  );
};
export const ChatState = () => {
  return useContext(ChatContext);
};
export default ChatProvider;
```

(ChatLogics.js)

```
export const isSameSendMargin = (messages, m, i, userId) => {
  if (
    i < messages.length -1 &&
      messages[i + 1].sender._id === m.sender._id &&
      messages[i].sender._id !== userId
  )
      return 33;
  else if (
      (i < messages.length - 1 &&
        messages[i + 1].sender._id !== m.sender._id &&
        messages[i].sender._id !== userId) ||
      (i === messages.length - 1 && messages[i].sender._id !== userId)
  )
      return 0;
  else return "auto";
};

export const isSameSender = (messages, m, i, userId) => {
  return (
    i < messages.length - 1 &&
    (messages[i + 1].sender._id !== m.sender._id ||
      messages[i + 1].sender._id === undefined) &&
    messages[i].sender._id !== userId
  );
};

export const isLastMessage = (messages, i, userId) => {
  return (
    i === messages.length - 1 &&
    messages[messages.length - 1].sender._id !== userId &&
    messages[messages.length - 1].sender._id
  );
};

export const isSameUser = (messages, m, i) => {
  return i > 0 && messages[i - 1].sender._id === m.sender._id;
};

export const getSender = (loggedUser, users) => {
  return users[0]?._id === loggedUser?._id ? users[1].name : users[0].name;
};

export const getSenderFull = (loggedUser, users) => {
  return users[0]._id === loggedUser._id ? users[1] : users[0];
};
```

(index.js)

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

import { ChakraProvider } from '@chakra-ui/react';
import ChatProvider from './Context/ChatProvider';
import { BrowserRouter } from "react-router-dom"

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <ChakraProvider>
    <BrowserRouter>
      <ChatProvider>
        <App />
    </ChatProvider>
  </BrowserRouter>
 </ChakraProvider>,
 );
reportWebVitals();
```

Backend

(server.js)

```
const express = require("express");
const connectDB = require("./config/db");
const dotenv = require("dotenv");
const userRoutes = require("./routes/userRoutes");
const chatRoutes = require("./routes/chatRoutes");
const messageRoutes = require("./routes/messageRoutes");
const { notFound, errorHandler } = require("./middleware/errorMiddleware");
const path = require("path");

dotenv.config();
connectDB();
const app = express();

app.use(express.json()); // to accept json data

app.use("/api/user", userRoutes);
app.use("/api/chat", chatRoutes);
app.use("/api/message", messageRoutes);
// --------------------------deployment------------------------------
const __dirname1 = path.resolve();

if (process.env.NODE_ENV === "production") {
  app.use(express.static(path.join(__dirname1, "/frontend/build")));

  app.get("*", (req, res) =>
    res.sendFile(path.resolve(__dirname1, "frontend", "build", "index.html"))
  );
} else {
  app.get("/", (req, res) => {
    res.send("API is running..");
  });
}
// --------------------------deployment------------------------------

// Error Handling middlewares
app.use(notFound);
app.use(errorHandler);

const PORT = process.env.PORT || 5000;

const server = app.listen(
  PORT,
  console.log(`Server running on PORT ${PORT}... http://localhost:5000`.yellow.bold)
);

const io = require("socket.io")(server, {
  pingTimeout: 60000,
  cors: {
    origin: "http://localhost:3000",
    // credentials: true,
  },
});

io.on("connection", (socket) => {
  console.log("Connected to socket.io");
  socket.on("setup", (userData) => {
    socket.join(userData._id);
    socket.emit("connected");
  });

  socket.on("join chat", (room) => {
    socket.join(room);
    console.log("User Joined Room: " + room);
  });
  socket.on("typing", (room) => socket.in(room).emit("typing"));
  socket.on("stop typing", (room) => socket.in(room).emit("stop typing"));

  socket.on("new message", (newMessageRecieved) => {
    var chat = newMessageRecieved.chat;

    if (!chat.users) return console.log("chat.users not defined");

    chat.users.forEach((user) => {
      if (user._id == newMessageRecieved.sender._id) return;

      socket.in(user._id).emit("message recieved", newMessageRecieved);
    });
  });

  socket.off("setup", () => {
    console.log("USER DISCONNECTED");
    socket.leave(userData._id);
  });
});
```

(userRoutes.js)

```
const express = require("express");
const {
  registerUser,
  authUser,
  allUsers,
} = require("../controllers/userControllers");
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/").get(protect, allUsers);
router.route("/").post(registerUser);
router.post("/login", authUser);

module.exports = router;
```

(messageRoutes.js)

```
const express = require("express");
const {
  allMessages,
  sendMessage,
} = require("../controllers/messageControllers");
```

```
const { protect } = require("../middleware/authMiddleware");

const router = express.Router();

router.route("/:chatId").get(protect, allMessages);
router.route("/").post(protect, sendMessage);

module.exports = router;
```

(chatRoutes.js)

```
const express = require("express");
const { accessChat, fetchChats, createGroupChat, removeFromGroup, addToGroup, renameGroup, } =
require("../controllers/chatControllers");
const { protect } = require("../middleware/authMiddleware");
const router = express.Router();

router.route("/").post(protect, accessChat);
router.route("/").get(protect, fetchChats);
router.route("/group").post(protect, createGroupChat);
router.route("/rename").put(protect, renameGroup);
router.route("/groupremove").put(protect, removeFromGroup);
router.route("/groupadd").put(protect, addToGroup);

module.exports = router;
```

(userController.js)

```
const asyncHandler = require("express-async-handler");
const User = require("../models/userModel");
const generateToken = require("../config/generateToken");
const allUsers = asyncHandler(async (req, res) => {
  const keyword = req.query.search
    ? {
        $or: [
          { name: { $regex: req.query.search, $options: "i" } },
          { email: { $regex: req.query.search, $options: "i" } },
        ],
      }
    : {};

  const users = await User.find(keyword).find({ _id: { $ne: req.user._id } });
  res.send(users);
});
const registerUser = asyncHandler(async (req, res) => {
  const { name, email, password, pic } = req.body;
  console.log(req.body);
  if (!name || !email || !password) {
    res.status(400);
    throw new Error("Please Enter all the Feilds");
  }

  const userExists = await User.findOne({ email });

  if (userExists) {
    res.status(400);
    throw new Error("User already exists"); }
  const user = await User.create({
    name,
    email,
    password,
    pic,
  });

  if (user) {
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      //isAdmin: user.isAdmin,
      pic: user.pic,
      token: generateToken(user._id),
    });
  } else {
    res.status(400);
    throw new Error("User not found");
  }
});

const authUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

  const user = await User.findOne({ email });

  if (user && (await user.matchPassword(password))) {
    res.json({
      _id: user._id,
      name: user.name,
      email: user.email,
      //isAdmin: user.isAdmin,
      pic: user.pic,
      token: generateToken(user._id),
    });
  } else {
    res.status(401);
    throw new Error("Invalid Email or Password");
  }
});

module.exports = { allUsers, registerUser, authUser };
```

(messageController.js)

```
const asyncHandler = require("express-async-handler");
const Message = require("../models/messageModel");
const User = require("../models/userModel");
const Chat = require("../models/chatModel");

const allMessages = asyncHandler(async (req, res) => {
  try {
    const messages = await Message.find({ chat: req.params.chatId })
      .populate("sender", "name pic email")
      .populate("chat");
    res.json(messages);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});

const sendMessage = asyncHandler(async (req, res) => {
  const { content, chatId } = req.body;
  if (!content || !chatId) {
    console.log("Invalid data passed into request");
    return res.sendStatus(400);
  }

  var newMessage = {
    sender: req.user._id,
    content: content,
    chat: chatId,
  };

  try {
    var message = await Message.create(newMessage);

    message = await message.populate("sender", "name pic").execPopulate();
    message = await message.populate("chat").execPopulate();
    message = await User.populate(message, {
      path: "chat.users",
      select: "name pic email",
    });

    await Chat.findByIdAndUpdate(req.body.chatId, { latestMessage: message });

    res.json(message);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});
module.exports = { allMessages, sendMessage };
```

(chatcontroller.js)

```
const asyncHandler = require("express-async-handler");
const Chat = require("../models/chatModel");
const User = require("../models/userModel");

const accessChat = asyncHandler(async (req, res) => {
  const { userId } = req.body;
  if (!userId) {
    console.log("UserId param not sent with request");
    return res.sendStatus(400);
  }

  var isChat = await Chat.find({
    isGroupChat: false,
    $and: [
      { users: { $elemMatch: { $eq: req.user._id } } },
      { users: { $elemMatch: { $eq: userId } } },
    ],
  })
    .populate("users", "-password")
    .populate("latestMessage");

  isChat = await User.populate(isChat, {
    path: "latestMessage.sender",
    select: "name pic email",
  });

  if (isChat.length > 0) {
    res.send(isChat[0]);
  } else {
    var chatData = {
      chatName: "sender",
      isGroupChat: false,
      users: [req.user._id, userId],
    };

    try {
      const createdChat = await Chat.create(chatData);
      const FullChat = await Chat.findOne({ _id: createdChat._id }).populate(
        "users",
        "-password"
      );
      res.status(200).json(FullChat);
    } catch (error) {
      res.status(400);
      throw new Error(error.message);
    }
  }
});
const fetchChats = asyncHandler(async (req, res) => {
  try {
    Chat.find({ users: { $elemMatch: { $eq: req.user._id } } })
      .populate("users", "-password")
      .populate("groupAdmin", "-password")
      .populate("latestMessage")
      .sort({ updatedAt: -1 })
      .then(async (results) => {
        results = await User.populate(results, {
          path: "latestMessage.sender",
          select: "name pic email",
        });
        res.status(200).send(results);
      });
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});
const createGroupChat = asyncHandler(async (req, res) => {
  if (!req.body.users || !req.body.name) {
    return res.status(400).send({ message: "Please Fill all the feilds" });
  }

  var users = JSON.parse(req.body.users);

  if (users.length < 2) {
    return res
      .status(400)
      .send("More than 2 users are required to form a group chat");
  }
  users.push(req.user);
  try {
    const groupChat = await Chat.create({
      chatName: req.body.name,
      users: users,
      isGroupChat: true,
      groupAdmin: req.user,
    });
    const fullGroupChat = await Chat.findOne({ _id: groupChat._id })
```

```
      .populate("users", "-password")
      .populate("groupAdmin", "-password");

    res.status(200).json(fullGroupChat);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});
const renameGroup = asyncHandler(async (req, res) => {
  const { chatId, chatName } = req.body;
  const updatedChat = await Chat.findByIdAndUpdate(
    chatId,
    {
      chatName: chatName,
    },
    {
      new: true,
    }
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!updatedChat) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(updatedChat);
  }
});
const removeFromGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;
  const removed = await Chat.findByIdAndUpdate(
    chatId,
    {  $pull: { users: userId }, },
    { new: true,}
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");
  if (!removed) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(removed);
  }
});
const addToGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;
  const added = await Chat.findByIdAndUpdate(
    chatId,
    { $push: { users: userId }, },
    { new: true,}
  )
    .populate("users", "-password")
    .populate("groupAdmin", "-password");

  if (!added) {
    res.status(404);
    throw new Error("Chat Not Found");
  } else {
    res.json(added);
  }
});

module.exports = { accessChat, fetchChats, createGroupChat, renameGroup, addToGroup,
removeFromGroup,};
```

(authMiddleware.js)

```
const jwt = require("jsonwebtoken");
const User = require("../models/userModel.js");
const asyncHandler = require("express-async-handler");

const protect = asyncHandler(async (req, res, next) => {
  let token;
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith("Bearer")
  ) {
    try {
      token = req.headers.authorization.split(" ")[1];

      //decodes token id
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.id).select("-password");
      next();
    } catch (error) {
      res.status(401);
      throw new Error("Not authorized, token failed");
    }
  }
  if (!token) {
    res.status(401);
    throw new Error("Not authorized, no token");
  }
});
module.exports = { protect };
```

(errorMiddleware.js)

```
const notFound = (req, res, next) => {
    const error = new Error(`Not Found - ${req.originalUrl}`);
    res.status(404);
    next(error);
  };
  const errorHandler = (err, req, res, next) => {
    const statusCode = res.statusCode === 200 ? 500 : res.statusCode;
    res.status(statusCode);
    res.json({
      message: err.message,
      stack: process.env.NODE_ENV === "production" ? null : err.stack,
    });
  };
  module.exports = { notFound, errorHandler };
```

(db.js)

```
const notFound = (req, res, next) => {
    const error = new Error(`Not Found - ${req.originalUrl}`);
    res.status(404);
    next(error);
```

```
  };
  const errorHandler = (err, req, res, next) => {
    const statusCode = res.statusCode === 200 ? 500 : res.statusCode;
    res.status(statusCode);
    res.json({
      message: err.message,
      stack: process.env.NODE_ENV === "production" ? null : err.stack,
    });
  };
  module.exports = { notFound, errorHandler };
```

(generateTowken.js)

```
const jwt = require("jsonwebtoken");
const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: "30d",
  });
};
module.exports = generateToken;
```

5.2 Strategic approach to software testing

Software testing is the process of evaluating a software application to identify if it meets specified requirements and to identify any defects. The following are common testing strategies:

Black box testing – Tests the functionality of the software without looking at the internal code structure. The following are the several categories of black box testing:

1. Functional Testing
2. Regression Testing
3. Nonfunctional Testing (NFT)

White box testing – Tests the internal code structure and logic of the software. It is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

Unit testing – Tests individual units or components of the software to ensure they are functioning as intended.

Integration testing – Tests the integration of different components of the software to ensure they work together as a system.

Functional testing – Tests functional requirements of software to ensure they are met.

System testing – Tests the complete software system to ensure it meets the specified requirements.

Acceptance testing – Tests the software to ensure it meets the customer’s or end-user’s expectations.

Regression testing – Tests the software after changes or modifications have been made to ensure the changes have not introduced new defects. It is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made. Regression means the return of something and in the software field, it refers to the return of a bug.

Performance testing – Tests the software to determine its performance characteristics such as speed, scalability, and stability. The goal of performance testing is to identify bottlenecks, measure system performance under various loads and conditions, and ensure that the system can handle the expected number of users or transactions.

Security testing – Tests the software to identify vulnerabilities and ensure it meets security requirements. It ensures that the software system and application are free from any threats or risks that can cause a loss.

5.3 Testing Approaches

Boundary Value Analysis(BVA)

Boundary Value Analysis(BVA) is based on testing the boundary values of valid and invalid partitions. The behavior at the edge of the equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects. It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.

Cyclomatic Complexicity

Cyclomatic complexity is a measurement developed by Thomas McCabe to determine the stability and level of confidence in a program. It measures the number of linearly-independent paths through a program module. Programs with lower Cyclomatic complexity are easier to understand and less risky to modify.
So, **cyclomatic complexity M** would be defined as,

$M = E – N + 2P$
where
E = the number of edges in the control flow graph
N = the number of nodes in the control flow graph
P = the number of connected components

In case, when exit point is directly connected back to the entry point. Here, the graph is strongly connected, and cyclometric complexity is defined as:

$\mathbf{M = E – N + P}$

In the case of a single method, P is equal to 1. So, for a single subroutine, the formula can be defined as:

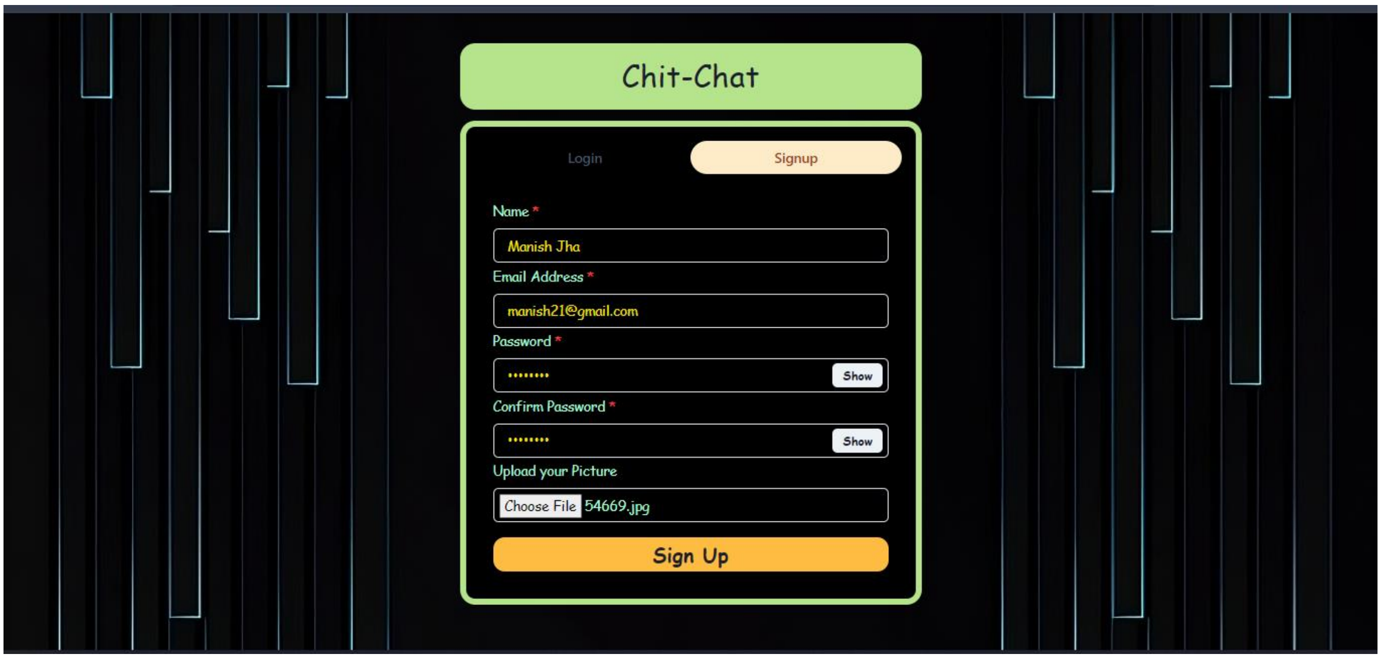
$\mathbf{M = E – N + 2}$

5.4 Test Case

Test ID	Test Objectives	Tester Name	Steps	Module	Pre-Conditions	Results
T1	Check for Mandatory Validation	Manish Jha	Click on “Sign Up” -> Keep all fields blank -> Click Sign Up	Authentic-ation	Sign up successful	Displays the message for “Please Fill all the Fields”.
T2	Check for Mandatory Validation	Manish Jha	Enter Number instead of Name -> Click Sign up	Validation	Sign up successful	Displays the message for “Please enter valid name”.
T3	Check for Mandatory Validation	Manish Jha	Enter different password in “ConfirmPassword” -> Sign Up	Validation	Sign up successful	Displays the message for “Passwords Do Not match”.
T4	Enter Sign Up Details which does not exist in database	Manish Jha	Enter Existing data -> Click Sign Up	Authentic-ation	Sign up successful	Displays the message for “User already exists! Go with Login”.
T5	Enter all valid data	Manish Jha	Fill all fields with proper data -> Click Sign Up	Authentic-ation	Sign up successful	Displays the message for “Sign Up Successfull”.
T6	Check for Mandatory Validation	Manish Jha	Go to login page -> Keep all fields blank -> Click LogIn	Authentic-ation	Login successful	Displays the message for “Please Fill all the Fields”.
T7	Enter login Details which does not exist in database	Manish Jha	Fill non-existing data -> Click Log In	Authentic-ation	Login successful	Displays the message “Invalid Credential!.”
T8	Enter correct Credentials in login which exists in database	Manish Jha	Enter details of existing user -> Click Log In	Authentic-ation	Login successful	Redirects to The Chat page.

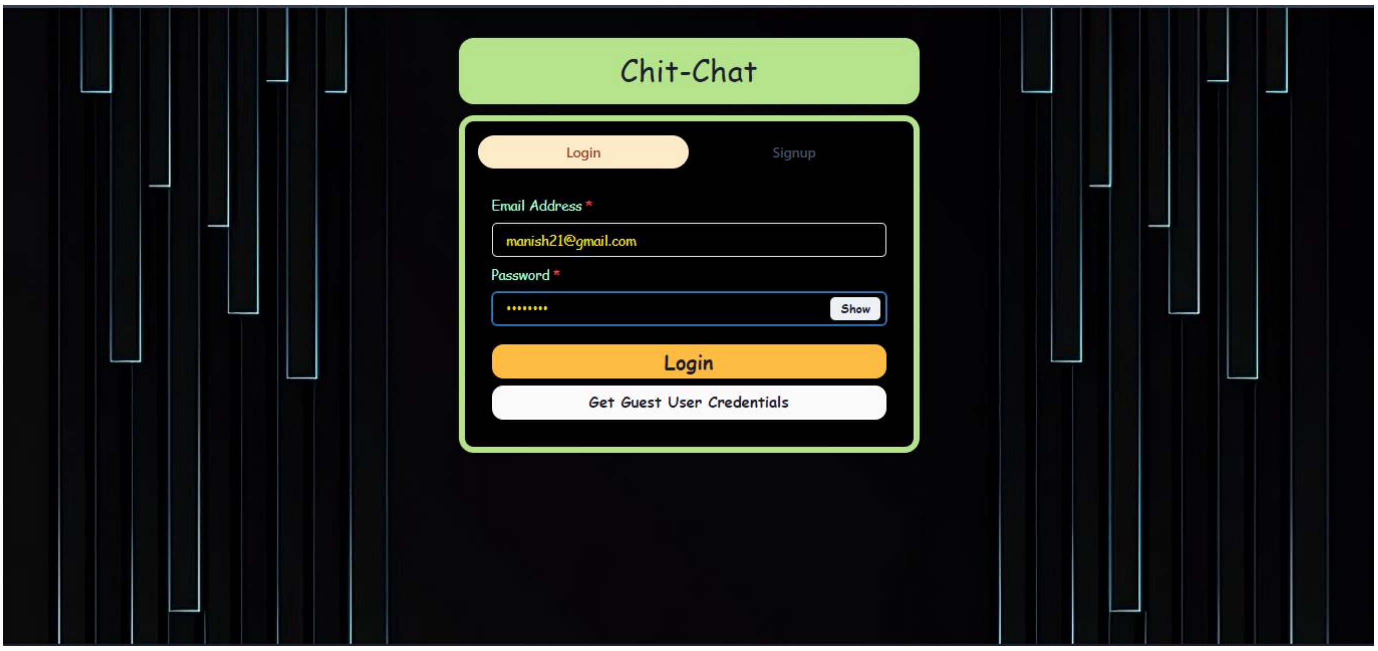
Chapter 6: RESULT & DISCUSSION

Signing up to a new account.



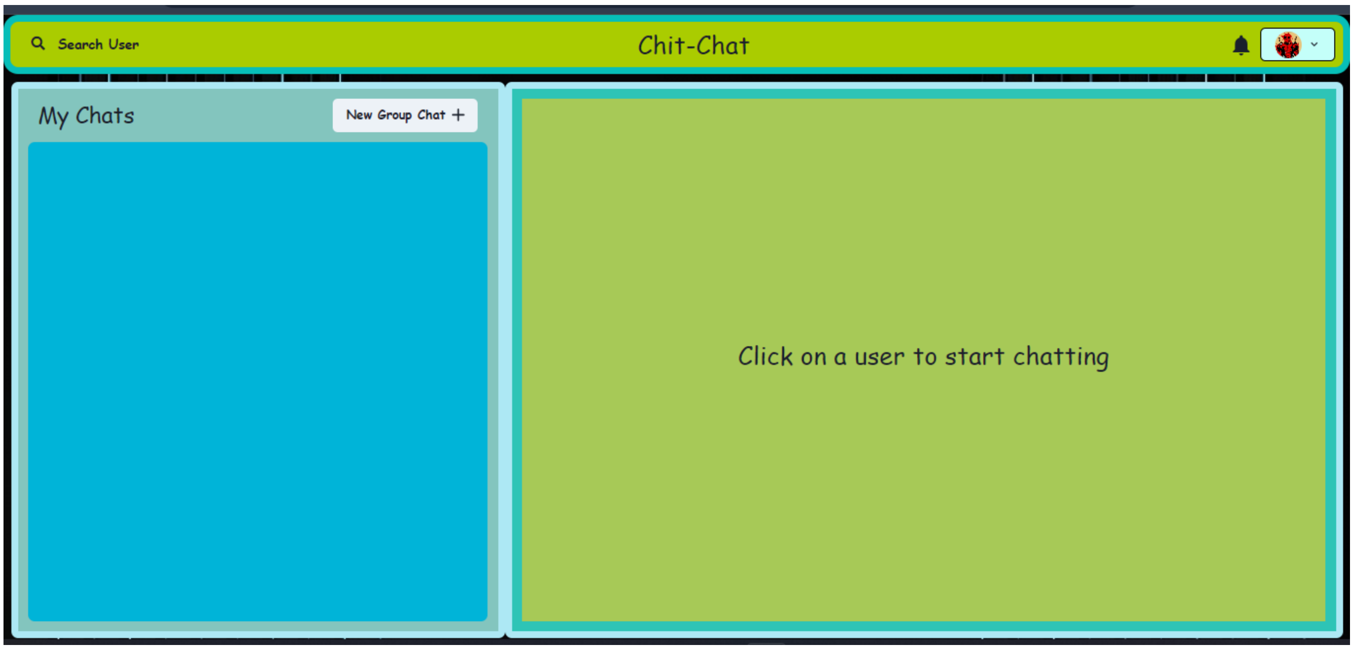
Name *
Manish Jha
Email Address *
manish21@gmail.com
Password *
••••••••
Show
Confirm Password *
••••••••
Show
Upload your Picture
Choose File 54669.jpg
Sign Up

Log in to the account that just created.

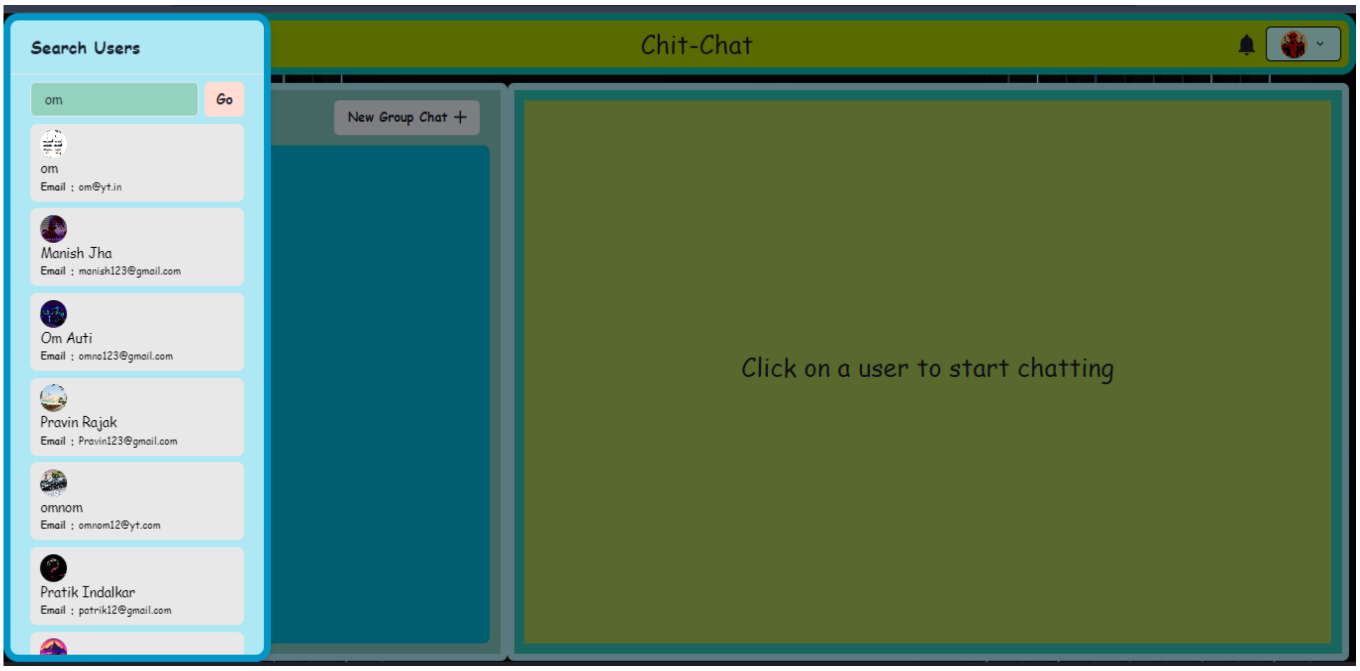


Email Address *
manish21@gmail.com
Password *
••••••••
Show
Login
Get Guest User Credentials

Chat Page of Chit-Chat (Default Chatting Page)



Search Users on clicking the “Search User” which is in Top-Left corner.



om

om
Email : om@yt.in

Manish Jha
Email : manish123@gmail.com

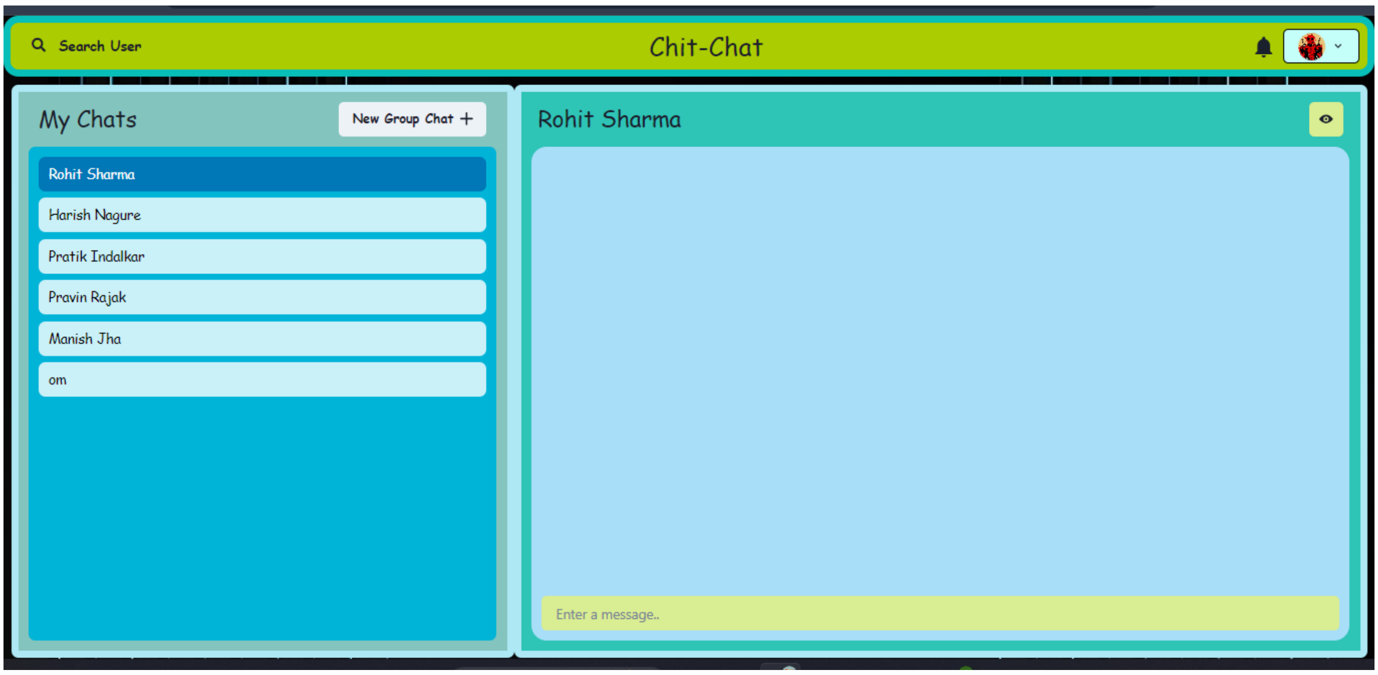
Om Auti
Email : omno123@gmail.com

Pravin Rajak
Email : Pravin123@gmail.com

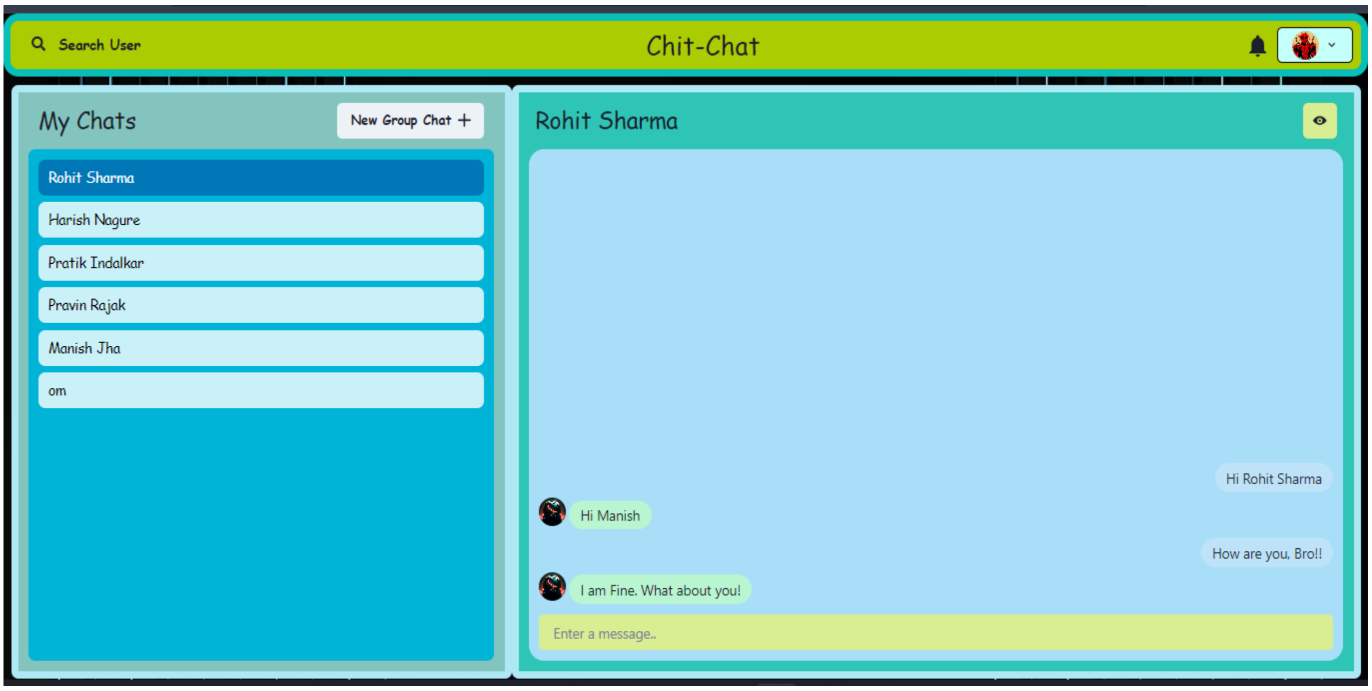
omnom
Email : omnom12@yt.com

Pratik Indalkar
Email : patrik12@gmail.com

Select Users with those you want to Chat.



Type the message you want to Send with.



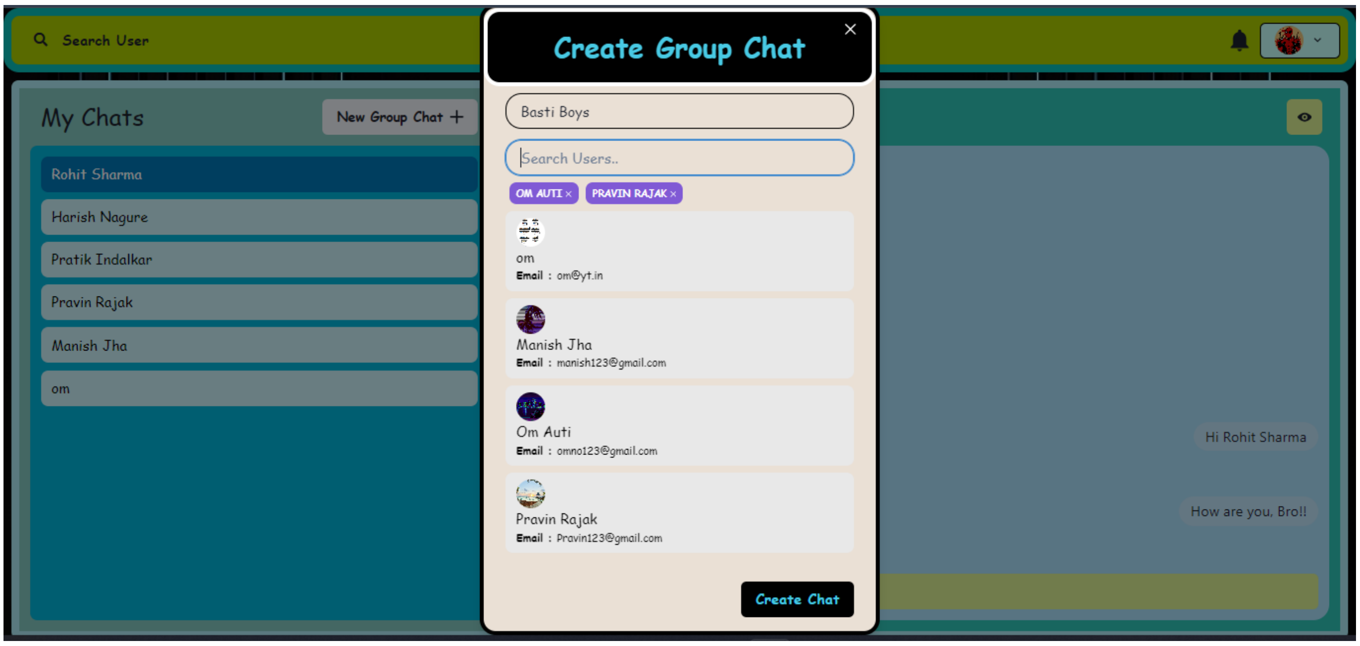
Hi Rohit Sharma

Hi Manish

How are you, Bro!!

I am Fine. What about you!

You can also Create Group by Clicking the "New Group Chat + " Button.



Basti Boys

Search Users..

OM AUTI × PRAVIN RAJAK ×

om
Email : om@yt.in

Manish Jha
Email : manish123@gmail.com

Om Auti
Email : omno123@gmail.com

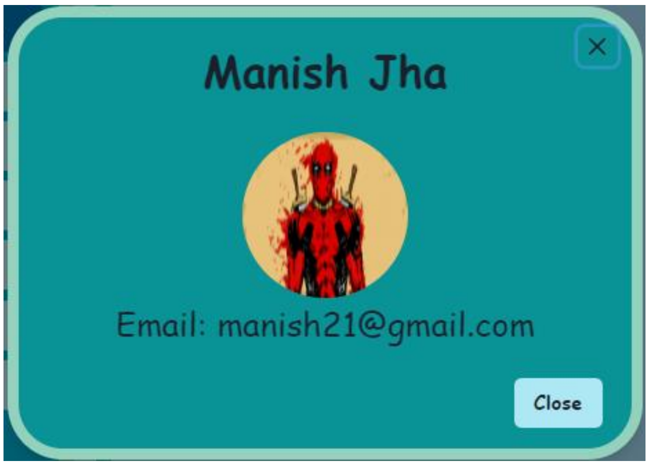
Pravin Rajak
Email : Pravin123@gmail.com

Rohit Sharma
Harish Nagure
Pratik Indalkar
Pravin Rajak
Manish Jha
om

Hi Rohit Sharma

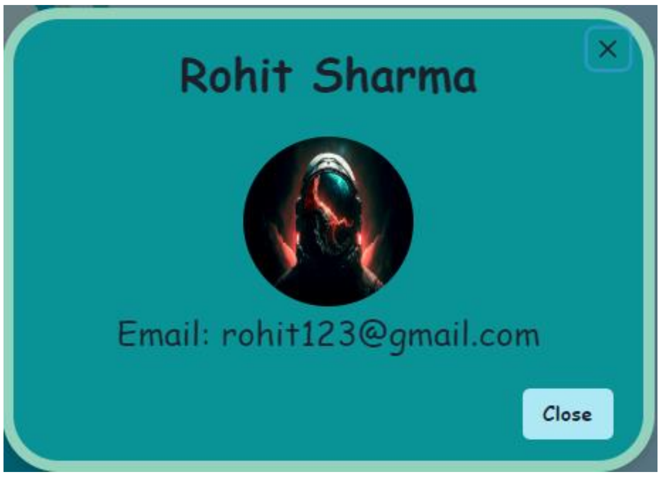
How are you, Bro!!

My Profile



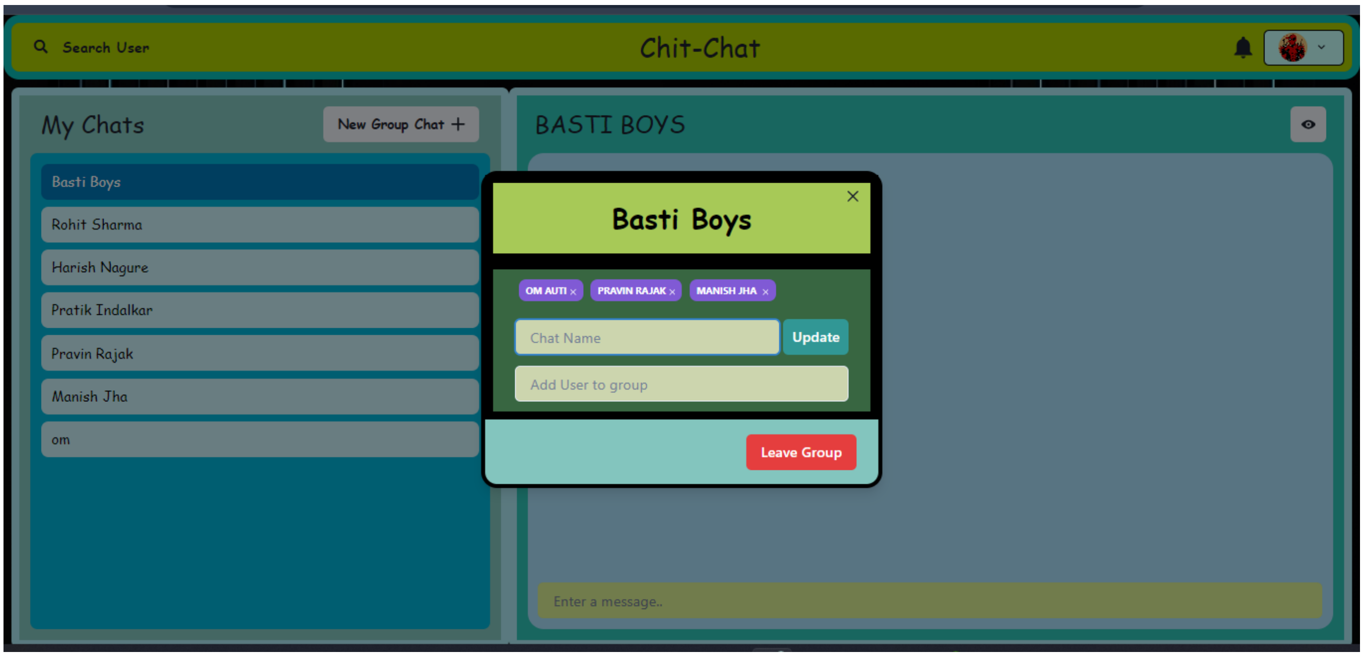
Email: manish21@gmail.com

Friend Profile with I am messaging.

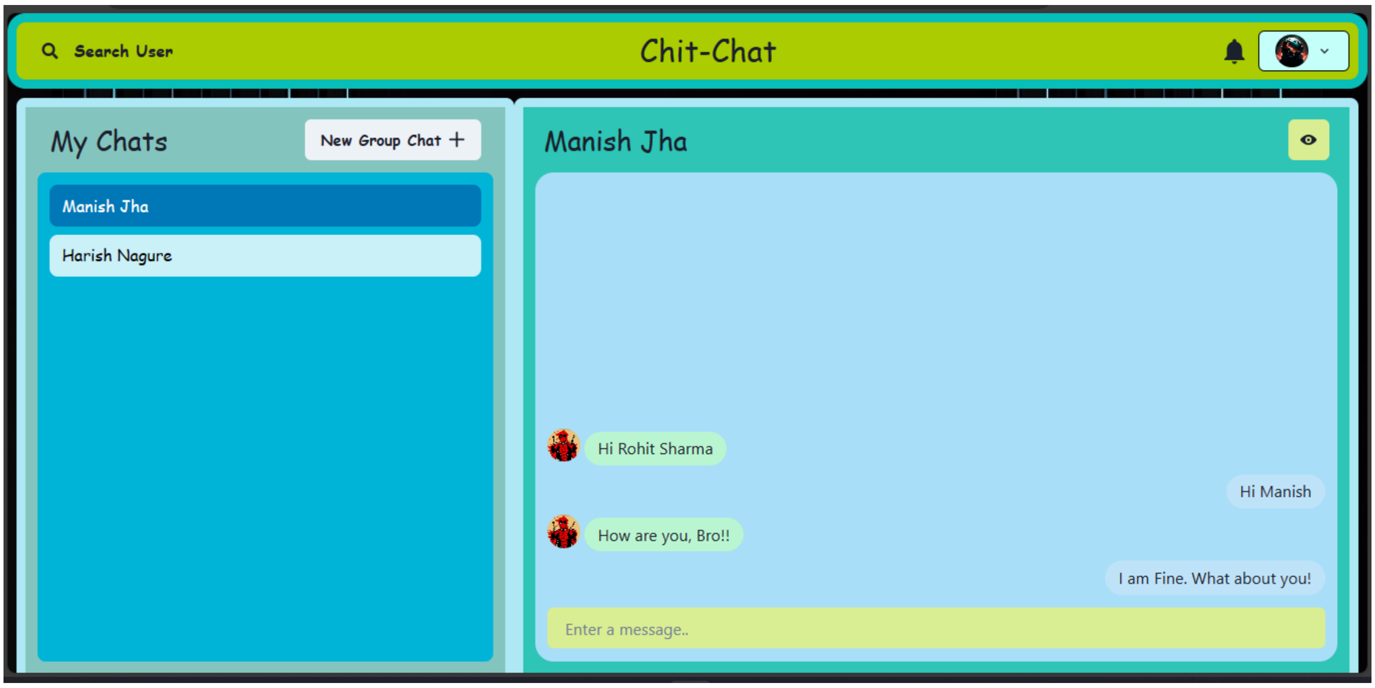


Email: rohit123@gmail.com

We can see that who is in the Group or we can Remove or Add new users in the Group.



User Rohit with I am chatting.



Hi Rohit Sharma

Hi Manish

How are you, Bro!!

I am Fine. What about you!

Chapter 7: CONCLUSION & FUTURE WORK

Conclusion :

In conclusion, a well-designed web chat application can significantly enhance communication, collaboration, and customer support. Regular updates, attention to security, and responsiveness to user needs are key factors in the ongoing success of such applications. Keep in mind that the landscape of technology is dynamic, and staying abreast of emerging trends is essential for continued relevance and effectiveness.

some key conclusions and considerations regarding web chat applications:

- The primary strength of web chat applications lies in their ability to enable real-time communication, fostering instant interaction and collaboration among users.
- Chat applications enhance user engagement, offering a dynamic and interactive way for users to connect, share information, and collaborate.

Future Work :

The future of web chat applications is likely to see several advancements and trends, driven by evolving technologies and user expectations. Here are some potential areas of future work for web chat applications:

- Integrating ChatOps principles for collaborative work, allowing users to perform tasks and access information directly within the chat interface, bridging communication and productivity.
- Implementing robust data analytics tools to gather insights into user behavior, preferences, and trends. This information can be used to improve user experience and tailor services to meet evolving needs.
- Integration of advanced language translation capabilities to facilitate global communication, breaking down language barriers and enabling users from different regions to interact seamlessly.

Web chat applications will likely continue to evolve in response to technological advancements, user demands, and the changing landscape of communication. Continuous innovation, a focus on user experience, and adaptability to emerging technologies will be crucial for the success of future web chat applications.

Chapter 8: REFERENCES

React References

React (https://legacy.reactjs.org/docs/getting-started.html)

Express(https://expressjs.com/en/guide/routing.html)

Nodejs (https://nodejs.org/docs/latest/api/)

Socket.io (https://socket.io/docs/v4/)

YouTube References

https://www.youtube.com/watch?v=9tn82yBrX90&list=PLUVqY59GNZQMcLXlrBo4T557kWjbKHMLS&ab_channel=TutorialsWebsite

For Designing

https://react-icons.github.io/react-icons/

https://chakra-ui.com/getting-started