Tribhuvan University

Faculty of Humanities and Social Sciences

WEB CHAT APPLICATION

A PROJECT REPORT

Submitted to

Department of Computer Application

Nagarjuna College of IT

Sankhamul, Lalitpur

In partial fulfillment of the requirements for the Bachelors in Computer Application

Submitted by

Bhuwan Paneru
BCA 8th Semester

Sandip Ghimire
BCA 8th Semester

Under the Supervision of

Mr. Ramesh Singh Saud

Tribhuvan University

Faculty of Humanities and Social Sciences

Nagarjuna College Of IT

Sankhamul, Lalitpur

Bachelor in Computer Application(BCA)

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by Bhuwan Paneru and Sandip Ghimire entitled **"Web Chat Application"** in the partial fulfillment of requirement for the degree of Bachelor in Computer Application is recommended for that final evaluation.

Ramesh Singh Saud

Project Supervisor

Nagarjuna College Of IT

Tribhuvan University

Faculty of Humanities and Social Sciences

Nagarjuna College Of IT

Sankhamul, Lalitpur

Bachelor in Computer Application(BCA)

LETTER OF APPROVAL

This is certify that this project prepared by Bhuwan Paneru and Sandip Ghimire entitled **"Web Chat Application"** in the partial fulfillment of requirement for the degree of Bachelor in Computer Application has been evaluated in our opinion it is satisfactory in the scope and quality as a project for the required degree.

Ramesh Singh Saud	Jay Mangal Gupta
Supervisor	Managing Director
Nagarjuna College Of IT	Nagarjuna College Of It
Sankhamul, Lalitpur	Sankhamul, Lalitpur

External Examiner	Internal Examiner

ACKNOWLEDGMENT

We would like to express our special thanks of gratitude to our supervisor and coordinator Mr. **Ramesh Singh Saud** who gave us the golden opportunity to do this wonderful project on the topic of Web based Chat Application, which also helped us in doing a lot of research and we came to know about so many new tools and technologies.

We would like to express our special thanks of gratitude to the Managing Director of Nagarjuna college **Mr. Jay Mangal Gupta** who gave us permission to do this Project.

I am highly indebted to Nagarjuna College for their guidance and constant supervision as well as for providing necessary information regarding the Project and support in the completion.

We would also like to express my gratitude towards the library and members of Nagarjuna College for their kind cooperation and encouragement which helped me in the completion of this Project.

We would also like to thank our parents and friends who helped us a lot in finalizing this project within the limited time frame.

I would like to thank the countless unnamed developers in github community and various other web portals, who have shared their experiences and knowledge freely for use by scholars likely us with unimaginable ease.

Yours sincerely,

Bhuwan Paneru / Sandip Ghimire

ABSTRACT

Chat applications are widely used for communication and social networking. However, not all chat applications provide adequate security and privacy for their users. End-to-end encryption (E2EE) is a technique that ensures that only the sender and receiver of a message can access its contents, preventing any third-party from intercepting or modifying it. Chat refers to the process of communicating, interacting and/or exchanging messages over the Internet. It involves two or more individuals that communicate through a chat-enabled service or software. A chat application has basic two components, server and client. A server is a computer program or a device that provides functionality for other programs or devices. Clients who want to chat with each other connect to the server. The chat application we are going to build will be like a peer-to-peer chat. So, this means that multiple users can connect to the chat server and send their messages. Every message is broadcasted to every connected chat user.

TABLE OF CONTENTS

SUPERVISOR'S RECOMMENDATION ..... **i**
LETTER OF APPROVAL ..... **ii**
ACKNOWLEDGMENT ..... **iii**
ABSTRACT ..... **iv**
TABLE OF CONTENTS ..... **v**
LIST OF ABBREVIATIONS ..... **vii**
LIST OF FIGURES ..... **viii**
LIST OF TABLES ..... **ix**
CHAPTER 1: INTRODUCTION **1**
1.1 Introduction 1
1.2 Problem Statement 1
1.3 Objectives 2
1.4 Scope and Limitation 2
1.4.1 Scope 2
1.4.2 Limitations 2
1.5 Report Organization 3
CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW **4**
2.1 Background Study 4
2.2. Literature Review 6
CHAPTER 3: SYSTEM ANALYSIS AND DESIGN **7**
3.1 System Analysis 7
3.1.1 Requirement Analysis 7
3.1.2 Feasibility Analysis 9
3.2.1 Data Modelling (Class and Object Diagram) 11
3.2.2 Process Modelling (State, Sequence and Activity Diagram) 13
3.2.3 Architectural Design 17
4.2 Algorithm Details 20
CHAPTER 4: IMPLEMENTATION AND TESTING **21**
4.1 Implementation 21
4.1.1 Tools used 21
4.1.2 Implementation Methodology 23

4.2 Testing 32
4.2.1 Test Cases for Unit Testing 32
4.2.2 Test Cases for System Testing.....33
CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS 34
5.1 Lesson Learnt / Outcome.....34
5.2 Conclusion 34
5.3 Future Recommendations 34
REFERENCES.....35
APPENDIX.....36

LIST OF ABBREVIATIONS

AES	Advance Encryption Standard
CSS	Cascading Style Sheets
CTR	Counter
DBMS	Database Management System
DFD	Data Flow Diagram
DHKE	Diffie Hellman Key Exchange
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I/O device	Input/output device
JS	JavaScript
OS	Operating System
ROM	Read Only Memory
SQL	Structure Query Language

LIST OF FIGURES

Figure 3.1 Waterfall Methodology 7
Figure 3.2: Use Case Diagram 9
Figure 3.3 Gantt chart for Web based Chat System.....11
Figure 3.5 Object Diagram 12
Figure 3. 6 State Diagram 13
Figure 3. 7 Sequence Diagram for Chat System.....14
Figure 3.8 Sequence Diagrams for Login and Signup System 15
Figure 3. 9 Activity Diagram 16
Figure 3.10 Architectural Design.....17
Figure 3.11 Component Diagram 18
Figure 3.12 Deployment Diagram 19
Figure 3.13 Interface Design.....19
Figure 3.14 End-to-End Encryption of Proposed System.....20

LIST OF TABLES

Table 4. 1 Unit Testing of User Registration 32
Table 4. 2 Testing of User Log In 33

CHAPTER 1: INTRODUCTION

1.1 Introduction

Messaging has become a part of our everyday lives in part due to its convenience for real-time chat communication and simple-to-use functionality. For instance, an iOS or text message on an iPhone or Android device from a friend, an email from a co-worker on Microsoft or Gmail, a team chat in a Slack or Microsoft Teams workspace, or even instant messaging through social media. These messaging and real-time chat applications play an important role in how the world interacts today, due to their immediacy and vast capabilities. A chat application makes it easy to communicate with people anywhere in the world by sending and receiving messages in real time. With a chat app, users are able to receive the same engaging and lively interactions through custom messaging features, just as they would in person. This also keeps users conversing on your platform instead of looking elsewhere for a messaging solution. Whether it's private chat, group chat, or large-scale chat, adding personalized chat features to your app can help ensure that your users have a memorable experience.

1.2 Problem Statement

This problem is to create a chat application with a server and users to enable users to chat with each other and develop an instant messaging solution to enable users to seamlessly communicate with one another.In an increasingly connected world, the demand for accessible and user-friendly communication tools has surged. Many existing chat platforms present barriers to entry, requiring users to navigate complex installations, privacy concerns, and compatibility issues. These challenges hinder the seamless flow of communication and limit accessibility, particularly for individuals seeking straightforward, cross-platform video chat experiences.

Ensuring data privacy and security during web chats is paramount, as users are increasingly cautious about sharing sensitive information online. Additionally, the scalability and performance of the chat applications become complex as user numbers grow. These challenges underscore the need for a comprehensive solution that provides a frictionless, secure, and accessible web chat communication experience.

Our project, "**Web Chat Application**" aims to address these challenges by creating a real time web-based chatting application with end-to-end encryption. Built on the Node JS for frontend and backend with mongo DB database, this application seeks to provide users with a user-friendly, secure, and scalable platform for real-time communication. By eliminating the need for complex installations and prioritizing data security, our project strives to bridge the gap between users, fostering meaningful and effortless chat conversations in an increasingly interconnected world.

1.3 Objectives

This project aims to create a new system which will focus on having a user-friendly interface and a simplistic design which can be easily understood by any user. The main objectives of this system are as follows:

- To develop a web-based chat application where users are able to communicate with each other with ease.
- To encourage user friendliness with interactive user interface such that it is very easy to use enabling even a novice person to use it.
- To build a secure chat application with end-to-end encryption where the messages communicated between the users are encrypted using encryption algorithm.

1.4 Scope and Limitation

1.4.1 Scope

- This chat application will enable communication between two users using point to point communication.
- Users can send messages to other users while maintaining optimum privacy with the help of end-to-end encryption.

1.4.2 Limitations

- This application does not support audio or video conversations.
- Transfer of files is not supported.

1.5 Report Organization

Chapter 1 includes the introduction of the project where problem statement, objective of the project along with the scopes and limitations of the project are listed which will provide a general concept of the project to the reader.

Chapter 2 includes background study and literature review of a similar project and concepts of the current project.

Chapter 3 covers the implementation method, feasibility analysis, database schema design, working mechanism, sequential diagram and flowcharts. It includes a detailed description of the working mechanism on how a chat system works.

Chapter 4 includes the testing and implementation phase of the project i.e., it includes designing of test cases to check if the system and its components work as intended during development. It also contains tools that were used in the construction of the project.

Chapter 5 contains the conclusion of the entire project. In this chapter, the outcome of the project is noted along with the things learnt from the creation of the project. In addition to these things the future recommendations of the project are also listed in the end of chapter

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

The emergence of computer networks and communication technologies allows people to communicate in a new way. Chatting is a method of using technology to bring people and ideas together despite geographical barriers. The technology has been available for years, but the acceptance was quite recent. The chat application will allow multiple users to connect to the server and chat with all other online users. This project will focus on providing security with high quality usability experiences to users. The application works in a broadcast fashion. This means that messages from a user are broadcast to other users. Messaging applications are growing in popularity. The past few years have brought applications like WhatsApp, Telegram, etc.

Study of Existing Systems

Telegram

Telegram uses end-to-end encryption for secret chats. When two users initiate a secret chat, the communication is encrypted from the user's device until it reaches the recipient's device. This means that the messages cannot be read by anyone else, not even Telegram. The encryption protocol used by Telegram is a custom implementation of the MTProto protocol. In addition to end-to-end encryption, Telegram also uses transport layer security (TLS) encryption to secure communication between the user's device and Telegram's servers. This ensures that the data is encrypted in transit and cannot be intercepted by third parties. [9][10]

Viber

Viber uses end-to-end encryption to secure user communications. This means that only the sender and intended recipient(s) can read the messages and nobody else, including Viber itself, can access the content of the messages. Viber uses a combination of RSA encryption for authentication and key exchange, and the AES encryption algorithm with a 256-bit key for message encryption. Each message sent between two Viber users is encrypted separately using a unique key that is derived from the users' shared secret key, which is generated during the initial setup process. [11]

Skype

Skype uses a combination of symmetric and asymmetric encryption to ensure the privacy and security of its users' communications. When two Skype users communicate, they use a unique encryption key that is generated by Skype for each session. This key is used to encrypt all communications between the two users.Skype uses a 256-bit Advanced Encryption Standard (AES) encryption algorithm for symmetric encryption, which is considered very secure. For asymmetric encryption, Skype uses a 2048-bit RSA key exchange and public key encryption. Skype also supports end-to-end encryption for voice calls, instant messages, and file transfers between two users. This means that the encryption and decryption of the communication happens only on the devices of the two communicating parties, and not on Skype servers. End-to-end encryption provides an additional layer of security and ensures that only the intended recipient can read the message or access the file.[12]

2.2. Literature Review

In today's rapidly evolving digital landscape, the profound impact of advanced computer networks and communication technologies on the way we connect and communicate with one another cannot be overstated. The world has witnessed a remarkable transformation in the realm of communication, transcending the geographical constraints that once limited human interaction. This evolution has been driven by the relentless march of technology, particularly the advent of the internet and the proliferation of various communication tools and platforms. One prominent mode of contemporary communication that has garnered widespread acceptance is chatting, which has evolved significantly over the years to become a cornerstone of our digital interconnectedness.

Nowadays most chat applications use encryption algorithms which provide secure communication between two parties. WhatsApp is now one of the most popular messaging apps globally with more than two and a half billion active users. Designed by Open Whisper Systems, this Signal Protocol forms the basis of WhatsApp's end-to-end encryption algorithm. Telegram, which offers an open-source messaging service, uses its own cryptographic encryption protocol, MTProto. The encryption operations it uses are based on 256- bit symmetric AES encryption, 2048-bit RSA encryption, and Diffie-Hellman key exchange. [5]

Node provides an event-driven and asynchronous platform for server-side JavaScript. It brings JavaScript to the server in much the same way a browser brings JavaScript to the client, So it is more efficient to use NodeJS to create real time applications. [6]

The basic operation principle for a system of symmetric cryptographic communication is the use of a shared secret key that is used for both encryption and decryption. The secret key is the most important component of the encryption system, as it is the principle means that transforms clear messages to ciphertexts. [7]

CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

3.1 System Analysis

This project is based on the waterfall model. For this system, requirements are gathered first and then verified then design process is carried out according to the requirement. After the design process, the coding and development part is start then after integrating the system there is testing of the system. If the testing is positive, then system is implemented otherwise some maintenance is done and system come in operation.

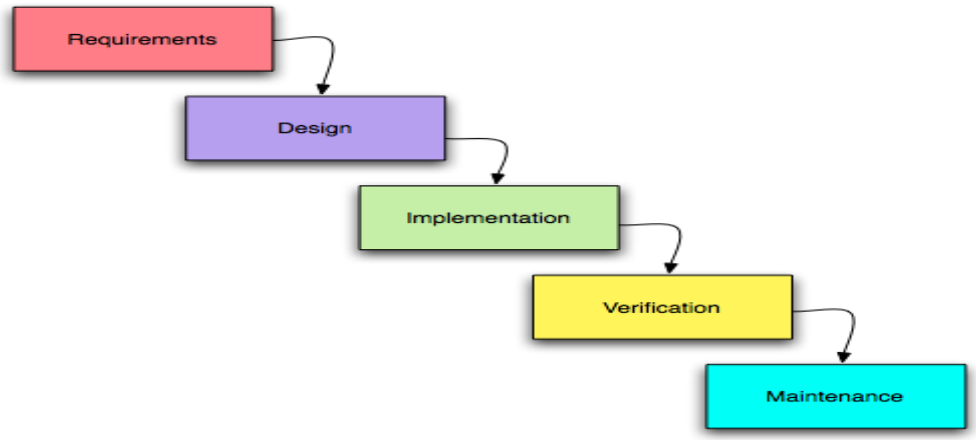


Figure 3.1 Waterfall Methodology

3.1.1 Requirement Analysis

i. Functional Requirements

Different functional requirement of system has been identified and are listed as below:

- **login and signup – authentication**
  Users must be able to register for the application through an Email, Username and Password, Image. On Opening the application, users must be able to register themselves or they can directly login if they have an account already. The user's email will be the unique identifier of his/her account on Chat Application.

- **Logout**
  After going to the user dashboard, the user should be able to log out from the chat application.

- **One to one chat - Realtime communication**
  In the chat application, one user can communicate with another directly.

- **View online and offline users.**
  Users should see the status of other users. Users can communicate with the users who are online at that moment.

- **Load old chats**
  The old chats between the two users should be loaded whenever the user clicks on the receiver.

- **Delete chats.**
  The user should be able to delete their own chats with other users by clicking the delete icon beside the chat message. When the user deletes the chat from the chat application, it should also reflect in the database.

ii. Non-functional Requirements

Different non-functional requirement has been studied and identified and are listed as below:

- **User friendly**
  The chat application should have an interactive and appropriate user interface for the users to use the application with ease.
- **Easy Accessibility**
  The chat application can be accessed easily from any place at any time using any browser.
- **Low latency**
  The chat application needs to have a low latency because it needs to look real-time so while you're sending a message, the other person should immediately be able to see that message.
- **Security**
  The chat application should provide security and optimum privacy to the messages being transmitted through the application.

USECASE DIAGRAM

In web chat applications, users are the actors. Users can register their account, login or logout from the system, select other users to chat, etc.

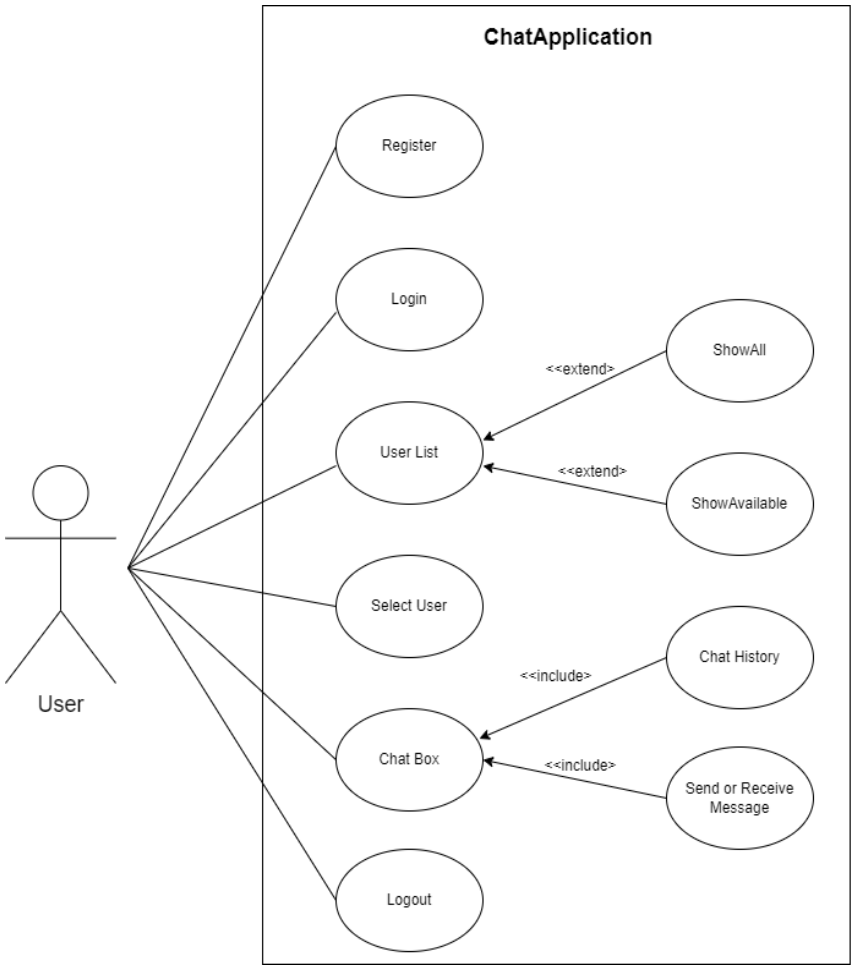


Figure 3.2: Use Case Diagram

3.1.2 Feasibility Analysis

Technical Feasibility

Technical feasibility centers around the existing computer system (hardware, software, etc.) and to what extent it can support the proposed addition. The chat application will be built using HTML, CSS, JavaScript in the front end. Node.js is a software platform that will be used to build server side more flexibly. Socket.IO will be used as a JavaScript library which is an implementation of web sockets protocol and various other required improvisation for real-time chat application. JavaScript is a scripting language that will be used to create applications, so the HTML document displayed in the browser becomes more

interactive and impressive for users. To save the user information and chats, we use MongoDB database which is a non-relational document database that provides support for JSON-like storage. All the required tools and software products are readily available on the web.

Operational Feasibility

People are inherently resistant to change, and computers have been known to facilitate change. An estimate should be made of how strong a reaction the user staff is likely to have toward the development of a computerized system. It is common knowledge that computer installations have something to do with turnover, transfers, retraining, and changes in employee job status. Therefore, it is understood that the introduction of a candidate system requires special effort to educate, sell and train the staff in new ways of conducting business. The interactive user interface of this chat application makes it easy for the users to easily understand and use this app without any hassle. Thus, this project is operationally feasible.

Economic Feasibility

More commonly known as cost benefit analysis, the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. If benefits outweigh costs, then the decision is made to design and implement the system. Otherwise, further justification or alterations in the proposed system will have to be made if it is to have a chance of being approved. This chat application doesn't require a huge amount of cost. Since the project is totally software related, it does not require extra hardware. Thus, it is feasible to build this chat application with little cost and is economically feasible.

Scheduling Feasibility

Since the project can be finished in the given period according to academic schedule, this project is feasible with respect to time also.

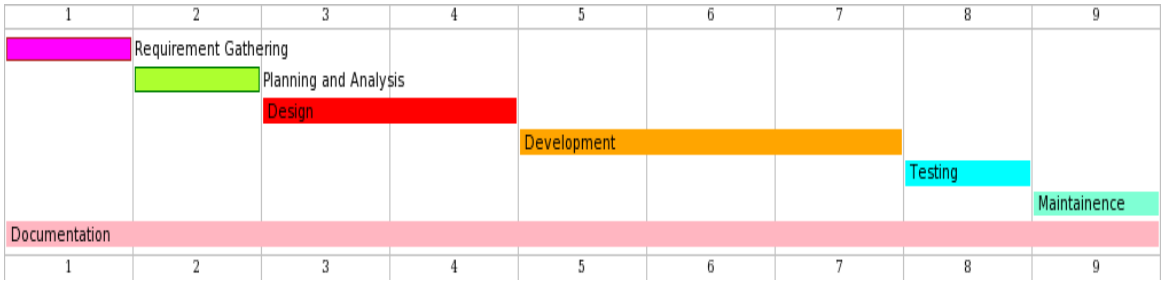


Figure 3.3 Gantt chart for Web based Chat System

3.2 System Design

3.2.1 Data Modelling (Class and Object Diagram)

Class Diagram

In this class diagram, we have two classes as users and chats where the users class includes attributes such as u_id , name , email, image, password and is_online. The chats class includes attributes such as m_id, sender_id who send the message, receiver_id who receive the message and the attribute message which includes the actual message that is sent from one user to another user.

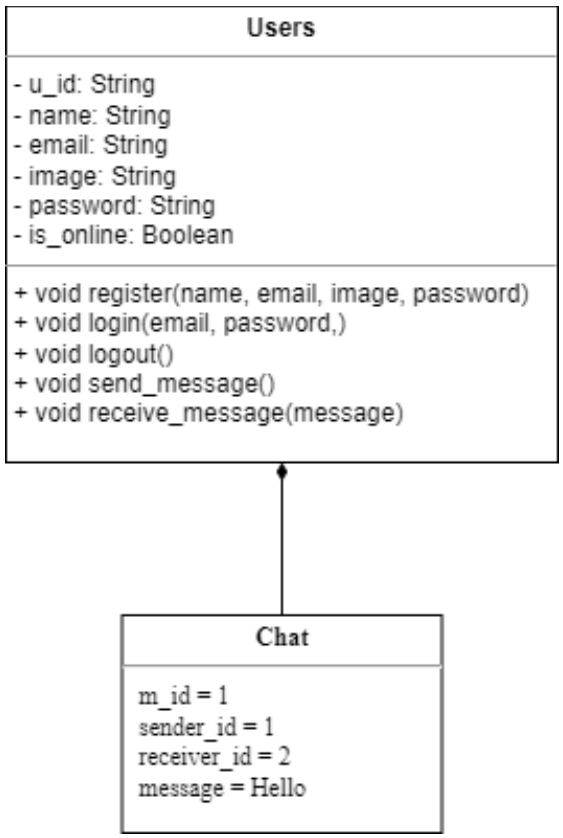


Figure 3.4 Class Diagram

Object Diagram

In this diagram, we have two objects representing the users having attributes u_id, name, email, password and is_online status, connected to a single object named as chat having attributes as m_id, sender_id, receiver_id and the messages representing the chat room where the conversation is taking place. The chat object represents the message hello was sent from user1 to user2.

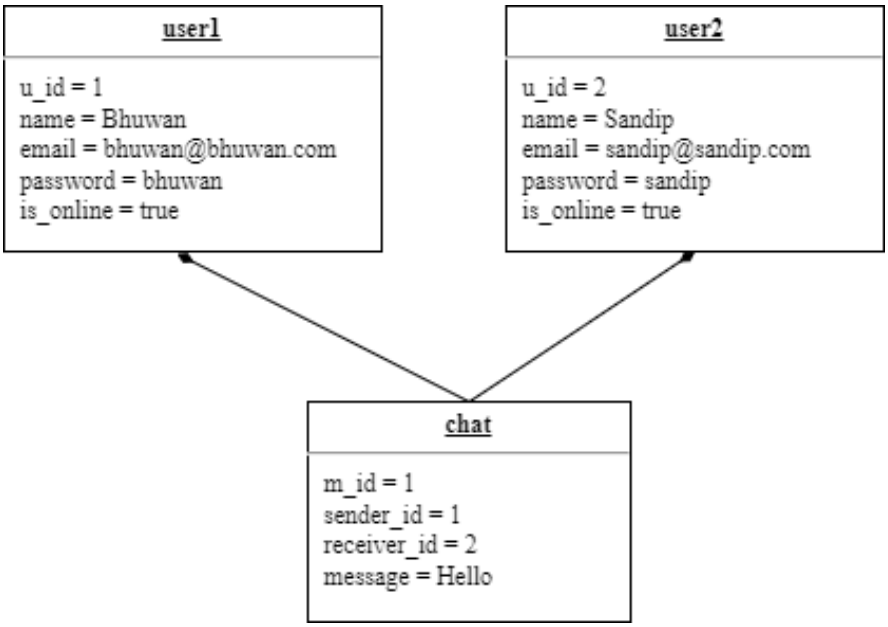


Figure 3.5 Object Diagram

3.2.2 Process Modelling (State, Sequence and Activity Diagram)

State Diagram

First, the user goes to the login page. If the user is new to the system, then the user needs to register and after that the user can login to the system dashboard, if the details are invalid it shows an error message, then the user needs to re-enter the valid information and log-in. After that, the user can see the list of other users in a dashboard where the user can select the user and can send the messages or also can receive the message from other users. At last, the user can also log out from the system.

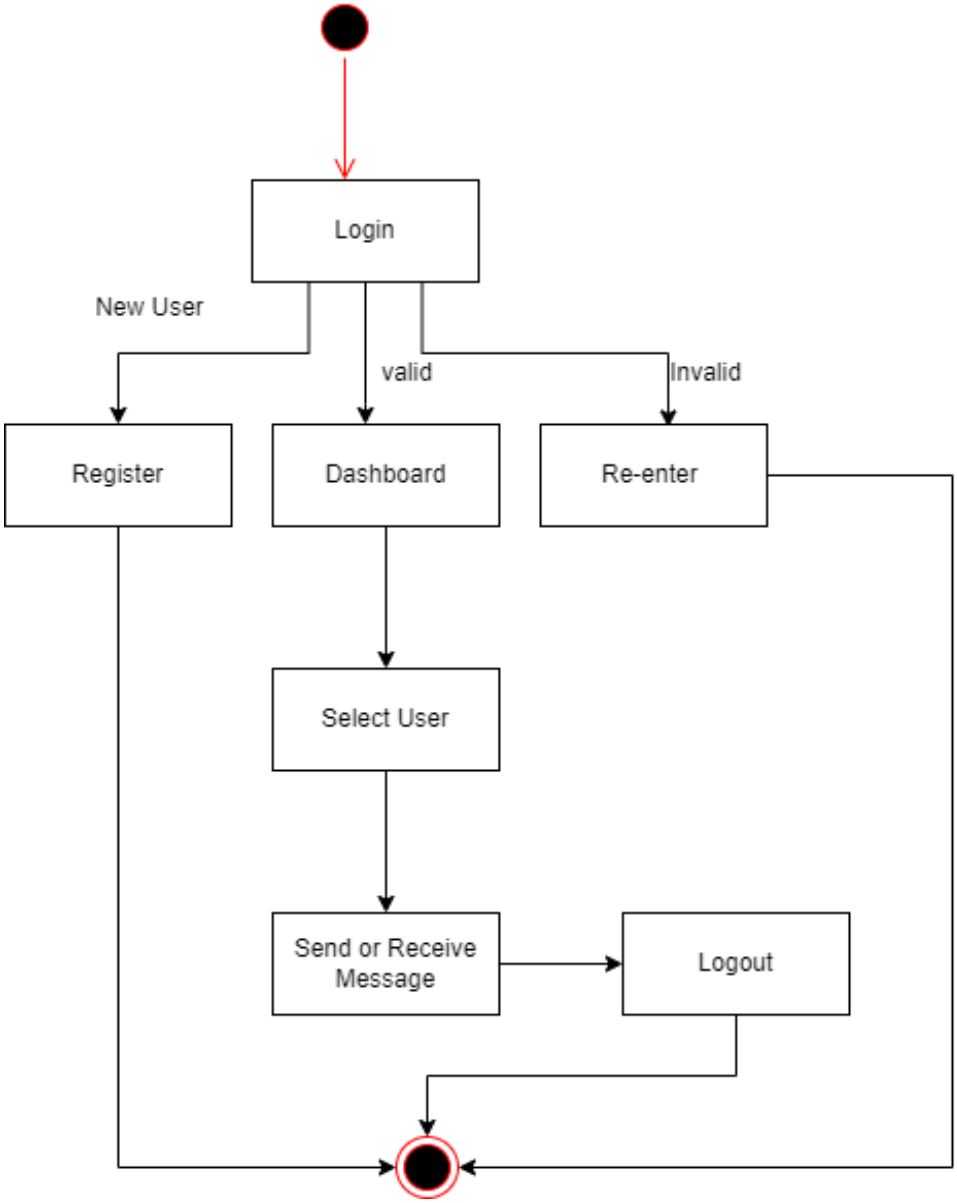


Figure 3. 6 State Diagram

Sequence Diagram

First, if the user is new to the system, the user needs to register by filling in the registration form. After registering, the user can login to the system with the valid information. After login, the user can see the list of other users with their status and can select the user to send a message. After sending the message, the message is encrypted with the encryption method and saved in the database and after that the encrypted message is received by the server from the database and decrypted, then the actual message is received by the user.

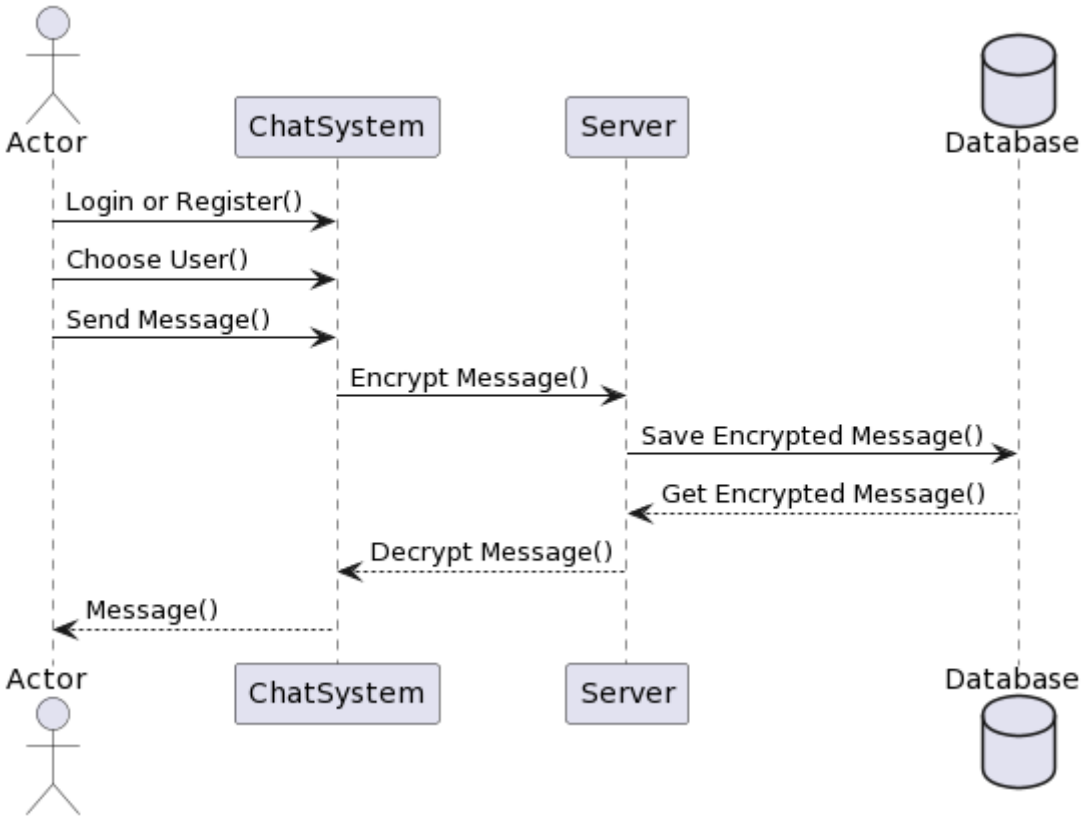


Figure 3. 7 Sequence Diagram for Chat System

When a user clicks on a register then a request is sent to a server and a server returns a register page to a user where the user can fill in the information and register. After registering, the information is sent to the server where the server checks whether the information is valid or not, if invalid it shows an error message. And if the information is valid then the information is saved in a database. After that, the user goes to the login page and enters the valid email and password and then the server checks the information from the user database if the information is correct then the user is sent to the home page.

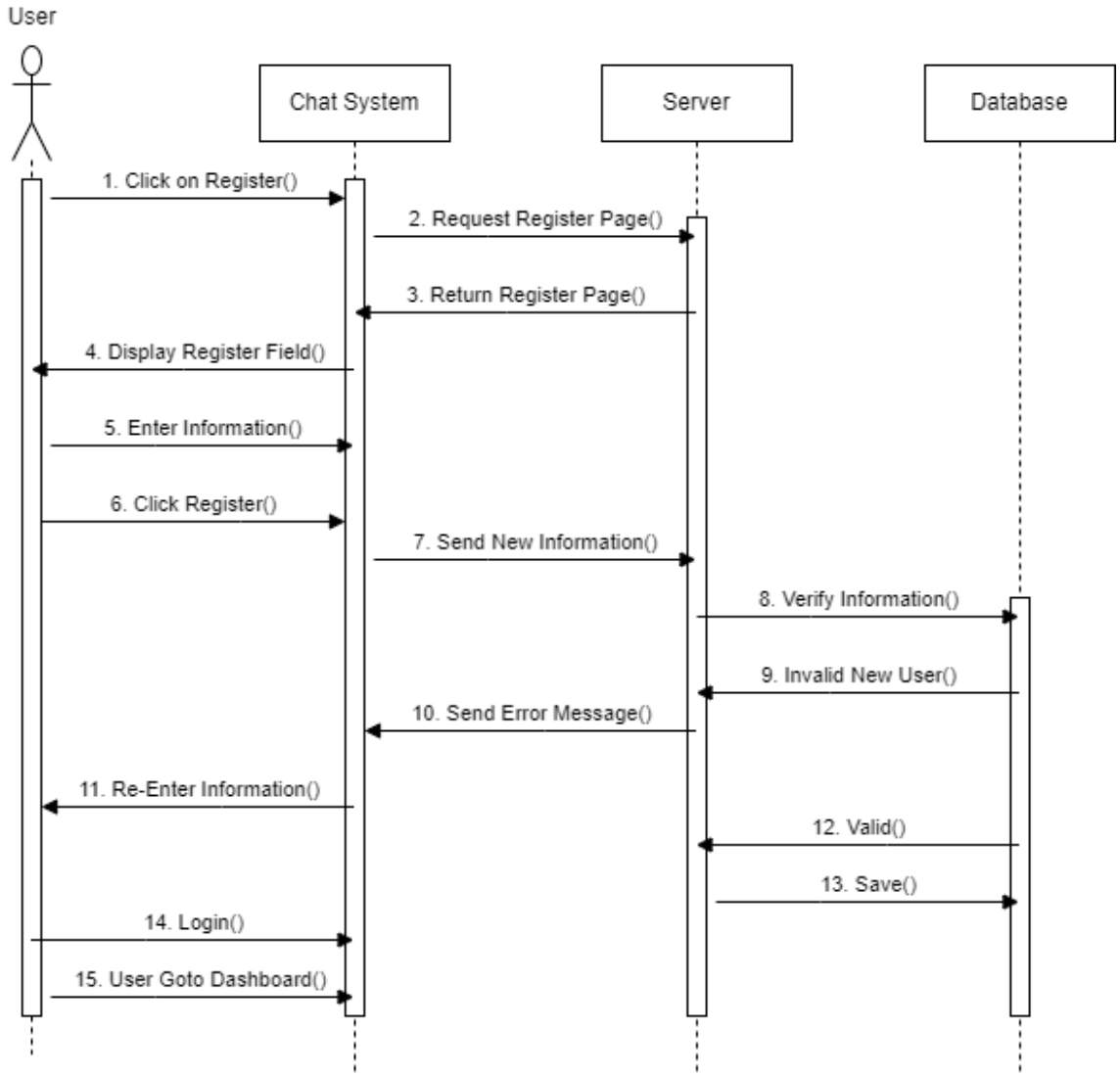


Figure 3.8 Sequence Diagrams for Login and Signup System

Activity Diagram

A user can login to the system by filling the login form or also can sign up if the user is new to the system. If the login information is valid then the user is sent to the dashboard or if the information is invalid, then the user needs to re-enter the details. In the dashboard page the user can see the list of other users with their online/offline status where the user can select the other users to send or receive messages and also the user can logout from the system.

Figure 3. 9 Activity Diagram

3.2.3 Architectural Design

The architectural design of a Web Chat System can be visualized as follows.

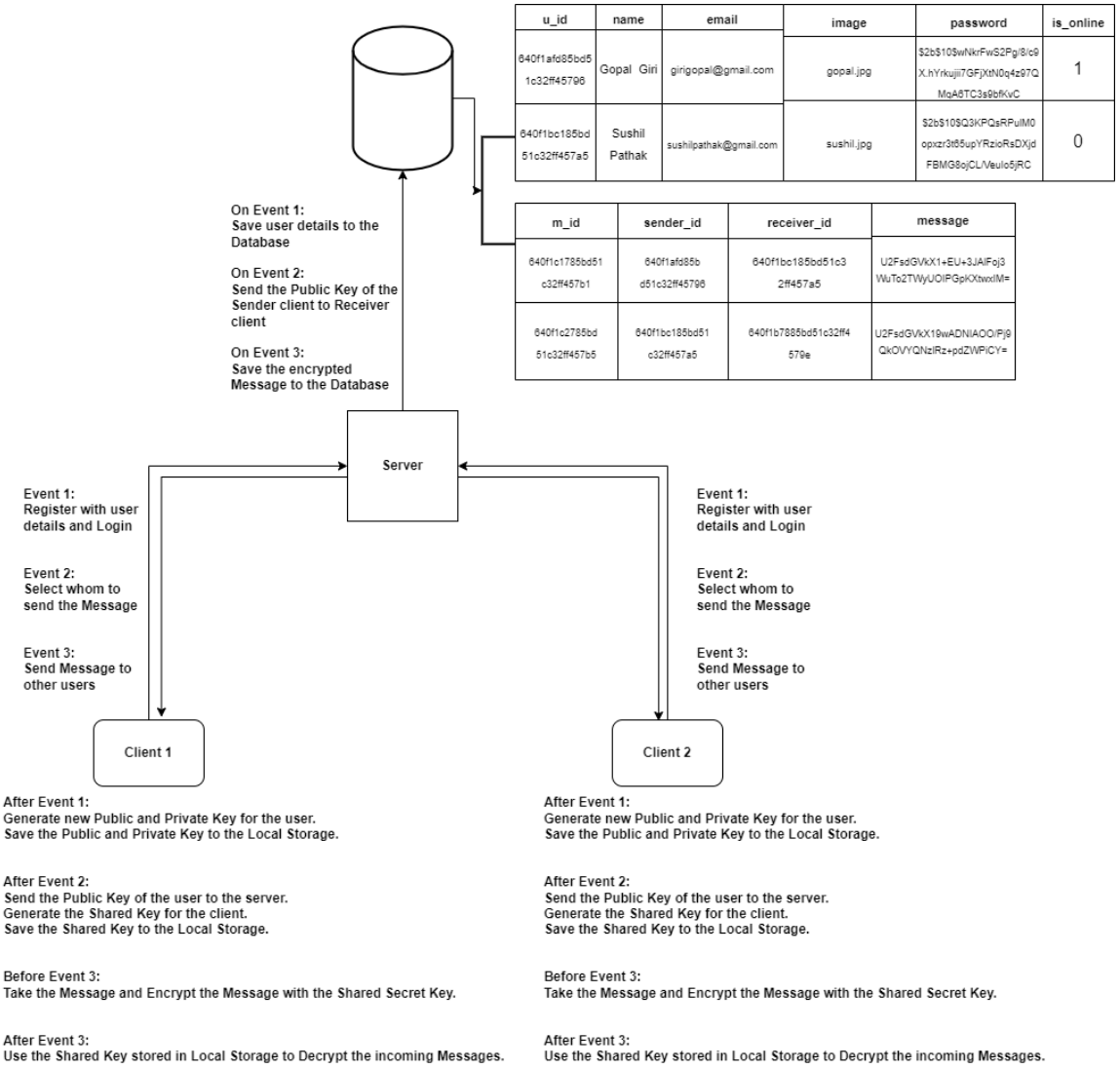


Figure 3.10 Architectural Design

3.2.3 Component Diagram

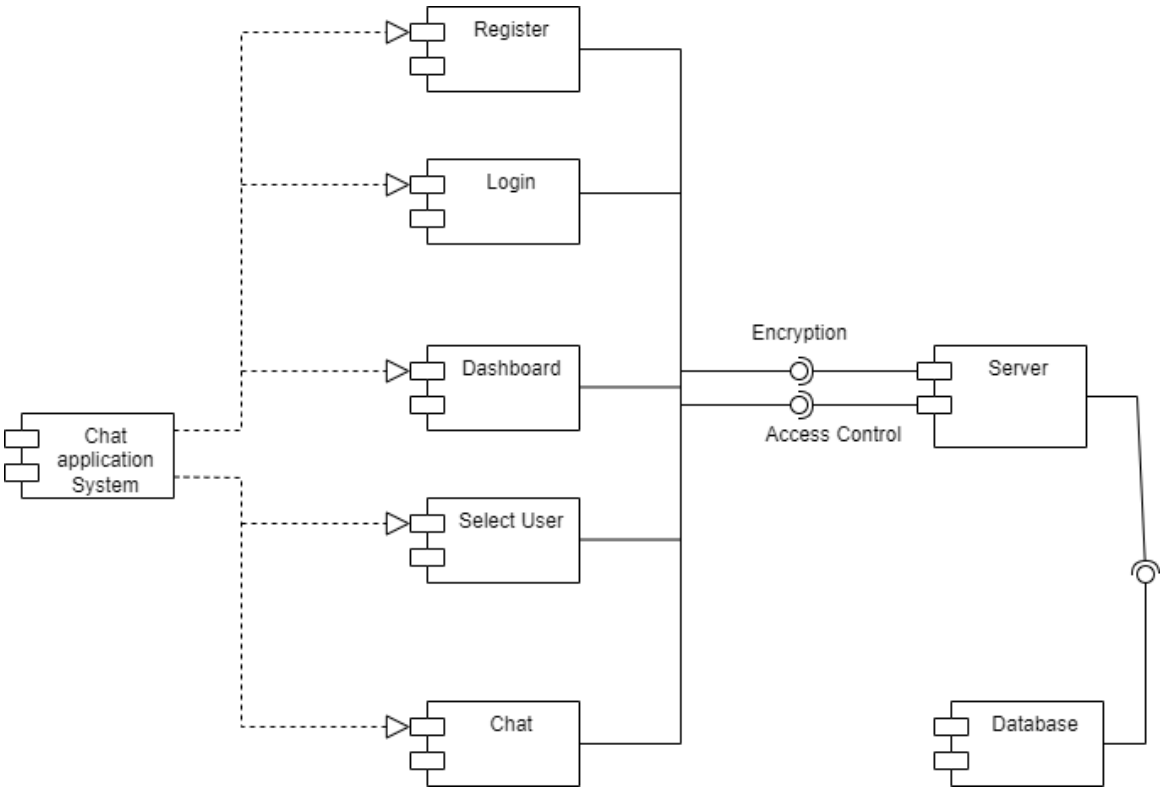


Figure 3.11 Component Diagram

In this diagram, we have two main terms: -

- Client: represents the front-end of the chat application, responsible for displaying the user interface and handling user input. The client component includes register, login, dashboard, select user, chat.
- Server: represents the back-end of the chat application, responsible for processing user requests and managing the chat sessions. The server component includes server and database.

3.2.4 Deployment Diagram

In this diagram, there is a one node named chat application having two attributes as user interface and database server where user interface includes login, register, dashboard, select user and send or receive message, and the database server includes user details, database interface and MongoDB server.

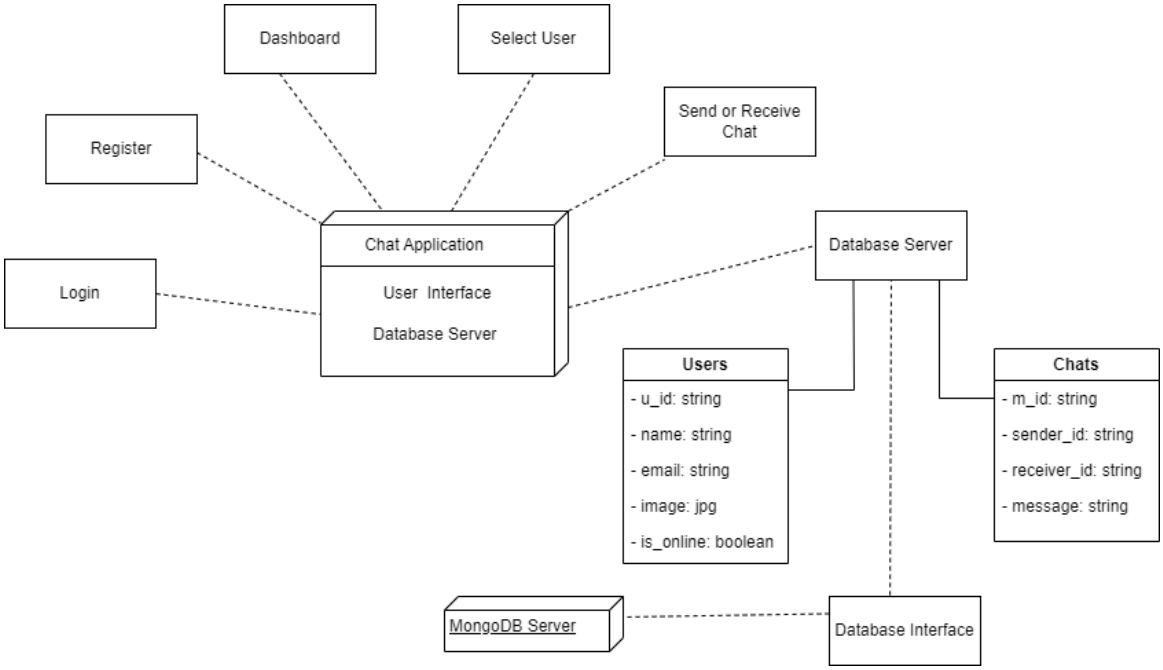


Figure 3.12 Deployment Diagram

3.2.5 Interface Design

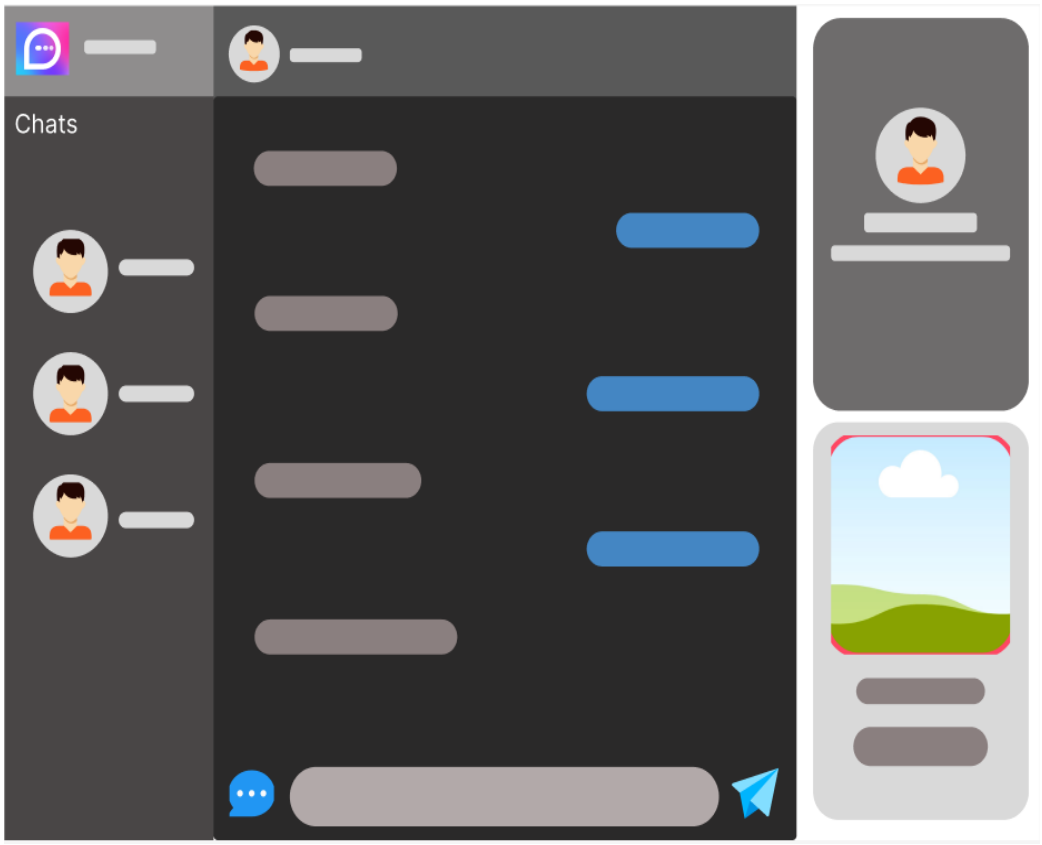


Figure 3.13 Interface Design

4.2 Algorithm Details

End to End Encryption in this chat application is achieved by involving Diffie-Hellman Key Exchange (DHKE) to provide the key pair, which will be exchanged between the two parties to generate the secure shared key that will be used as a key for the encryption algorithms. The proposed secure chatting application furnishes confidentiality, privacy, and integrity. Users can be granted that nobody, even the provider of the service, cannot read their messages. The exchanged data is stored only on the server, and none of it is stored at the physical memory of the phone. The algorithm used for encrypting text messages is the AES standard, which is slower than other block cipher, but it provides higher security

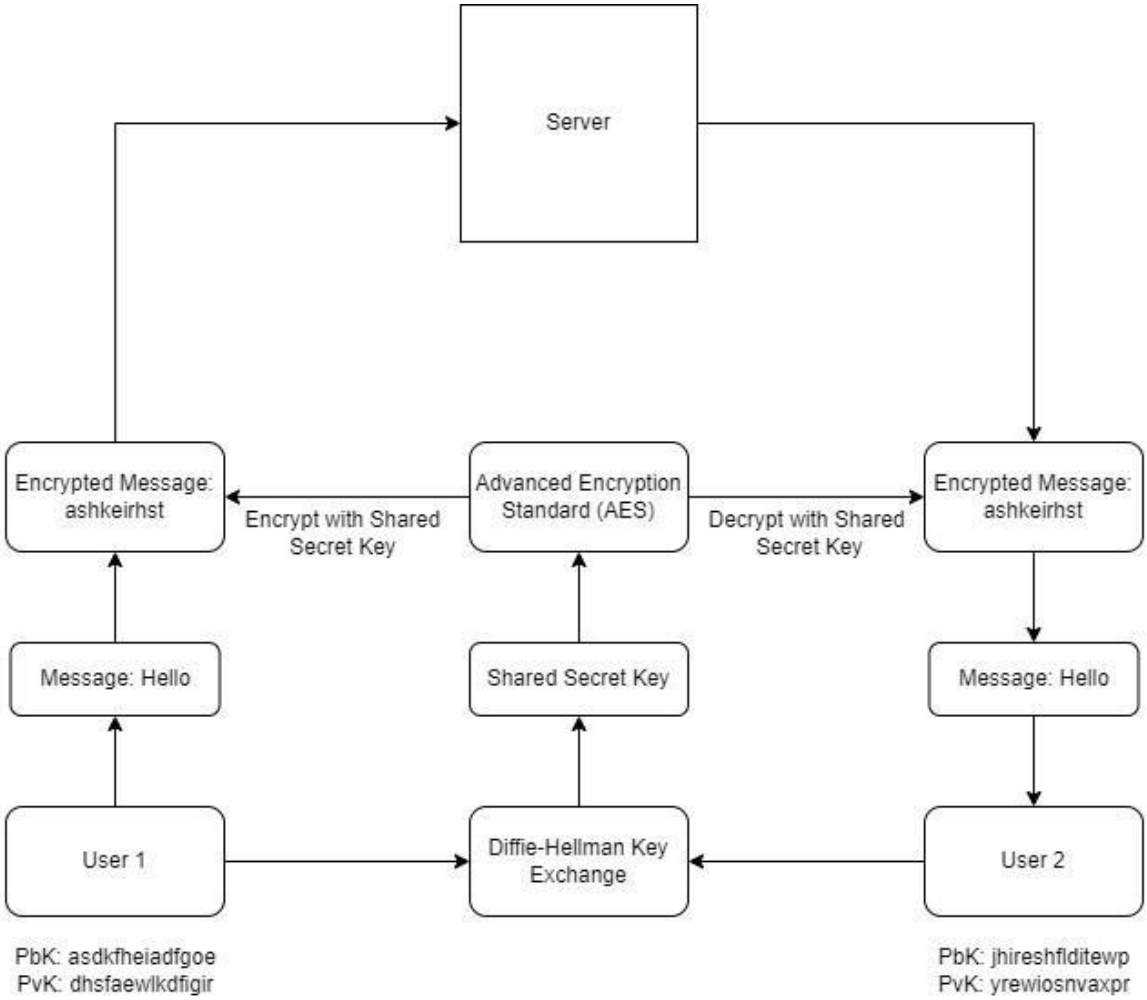


Figure 3.14 End-to-End Encryption of Proposed System

CHAPTER 4: IMPLEMENTATION AND TESTING

4.1 Implementation

In this chapter, the tools used in developing the prototype and the developed system are described.

4.1.1 Tools used

Following are the tools and framework used for the accomplishment of this project:

- **Draw.io**

  Draw.io is a popular free and open-source web-based diagramming software that allows users to create a wide range of diagrams and charts, including flowcharts, process diagrams, mind maps, UML diagrams, and more. Draw.io provides a simple and user-friendly interface for creating and editing diagrams, with a wide range of shapes and symbols available for customization.

- **Git**

  Git is a popular and widely used open-source version control system that allows developers to track changes to source code during software development. Git enables developers to manage their codebase, collaborate with other developers, and maintain multiple versions of their code.

- **HTML5**

  HTML (Hypertext Markup Language) is a standardized language used to create and design web pages. It is the primary markup language used to structure content on the World Wide Web and defines the structure and layout of web pages by using a series of tags and attributes. HTML allows developers to create text, images, hyperlinks, forms, and other interactive elements on web pages, making it possible to create a rich and dynamic user experience on the web.

- **CSS3**

  CSS (Cascading Style Sheets) is a stylesheet language used to describe the presentation and formatting of HTML and XML documents. CSS allows developers to separate the presentation of a document from its structure, making it easier to create visually appealing and consistent designs across multiple web pages. With CSS, developers can define styles for various elements such as font size, color, layout, and positioning, and apply them to HTML elements on a web page.

- **Bootstrap**

  Bootstrap is a popular open-source front-end web development framework that provides developers with a range of pre-designed HTML, CSS, and JavaScript components and tools for creating responsive and mobile-first web pages and web applications.

- **JavaScript**

  JavaScript (often abbreviated as JS) is a high-level programming language used primarily for creating interactive and dynamic web content. It is one of the three core technologies used in web development, along with HTML and CSS. JavaScript can be used to add interactive elements to web pages, such as animations, form validation, and user interface components, and to create more complex web applications that interact with web servers and databases.

- **Node.js**

  Node.js is a popular open-source, cross-platform JavaScript runtime environment that enables developers to execute JavaScript code on the server-side. Node.js is built on top of Google's V8 JavaScript engine and offers a range of features that allow developers to build scalable and high-performance web applications. Node.js provides a non-blocking I/O model that enables developers to write asynchronous code that can handle a large number of concurrent connections without blocking the event loop. This makes it possible to build real-time web applications and APIs that can handle a large number of requests. Node.js also comes with a built-in package manager, npm, which makes it easy to install and manage third-party libraries and modules.

- **MongoDB**

  MongoDB is a popular open-source NoSQL document-oriented database system that uses a flexible schema and JSON-like documents for data storage. MongoDB allows developers to store and manage large amounts of unstructured data, making it ideal for building scalable and high-performance web applications.

  Compass is a graphical user interface (GUI) for MongoDB that provides developers with a user-friendly way to interact with and manage MongoDB databases. Compass offers a range of features, including a visual schema explorer, a query builder, and an aggregation pipeline builder, that make it easy for developers to create, modify, and query MongoDB databases.

- **Socket.IO**

  Socket.IO is a popular open-source JavaScript library that enables real-time bidirectional communication between a client (such as a web browser) and a server. Socket.IO uses WebSockets, a protocol for two-way communication between a client and a server over a single TCP connection, to enable real-time communication. It falls back to other communication protocols, such as AJAX long polling, if WebSockets are not available in the client's browser.

- **Integrated Development Environment (IDE)**

  VS Code (Visual Studio Code) is a popular free and open-source code editor developed by Microsoft for Windows, macOS, and Linux. VS Code provides a range of features and extensions that make it a powerful and flexible tool for developers to write and edit code. Some of the key features of VS Code include code highlighting, auto-completion, debugging, code refactoring, and Git integration. The editor also supports multiple programming languages and file formats, making it a versatile tool for a wide range of development tasks.

4.1.2 Implementation Methodology

To build the proposed Chat Application, we have implemented the MVC pattern. MVC (Model-View-Controller) is a design pattern that separates the concerns of an application into three distinct components: the Model, the View, and the Controller. The Model represents the data and business logic of the application, the View represents the presentation layer, and the Controller acts as an intermediary between the two, handling user input and coordinating updates to the Model and View.

In Node.js, the MVC pattern can be implemented using a variety of frameworks, such as Express.js. Here is a brief overview of how the MVC pattern can be implemented in Node.js using Express:

- **Model:** The Model represents the data and business logic of the application. In a Node.js application, this could be implemented using a database such as MongoDB or MySQL, or an ORM (Object-Relational Mapping) library such as Sequelize or Mongoose.

- **View:** The View represents the presentation layer of the application. In Node.js, this could be implemented using a template engine such as EJS, Pug, or Handlebars,

which allows for dynamic generation of HTML and other content based on data from the Model.

- **Controller:** The Controller acts as an intermediary between the Model and the View, handling user input and coordinating updates to the Model and View. In Node.js, this could be implemented using a router such as Express Router, which maps incoming requests to the appropriate controller function.

By implementing the MVC pattern in Node.js, developers can create applications that are easier to maintain, more modular, and more scalable. Separating concerns into distinct components makes it easier to make changes to one part of the application without affecting other parts, and allows for easier testing and debugging.

Setting up the Application

Run **npm init -y** on your terminal. This creates a package.json file.

Now for this application, we will need to install some packages to get started.

npm install express mongoose ejs bycrypt body-parser multer express-session dotenv socket.io

These packages provide is with the following:

- **Express** is an Express application, which is necessary for our Express server
- **EJS** (Embedded JavaScript) is a popular templating engine for Node.js that allows developers to create dynamic HTML pages by embedding JavaScript code directly into the HTML.
- **Mongoose** is an Object Data Modeling (ODM) library for Node.js and MongoDB that connects our application to MongoDB.
- **Bycrptjs** handles encrypting passwords
- **Body-parser** is Node.js body parsing middleware. It parses incoming request bodies in a middleware before your handlers, available under the req.body property.

- **Multer** is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of busboy for maximum efficiency.

- **Express-session** is a middleware for the Express.js web framework that provides session management functionality. Sessions are a way to store data between HTTP requests in a stateless web application.

- **dotenv** is a popular npm package for managing environment variables in Node.js applications. Environment variables are key-value pairs that are stored outside the application code and can be accessed by the application at runtime. They are used to store sensitive information such as passwords, API keys, and database credentials.

- **Socket.IO** enables real-time bidirectional event-based communication. It consists of: a Node.js server (this repository) and a Javascript client library for the browser (or a Node.js client)

After this is complete, a node_module folder is created where all the packages are downloaded to.

Now create three folders to represent MVC: models, views, and controllers.

Setting up the Server

While we've created our folders, they can't do anything without a server. To create our server, let's create an app.js folder in our root directory. We can call this file whatever we want, provided we state so in the package.json file.

Create a express server using the code below.

```
const app = require('express')();
const http = require('http').Server(app);
http.listen(3000, function () {
   console.log('server is running');
});
```

To confirm that our server is now up and running, run the following:

nodemon app

Creating Routes, Views and Models

Creating Views

With our server up and running, let's create some .ejs files in our view folder. Since we are following the MVC pattern, we need all our views, that is, what the end users see, to be in one folder.

Inside the views folder, create the following files: login.ejs, register.ejs, dashboard.ejs

Create partials for dashboard.ejs inside layouts folder with files header.ejs and footer.ejs

- login.ejs code renders the Login page.
- register.ejs code render the Register page.
- dashboard.ejs code renders the Dashboard page.

Creating Routes

Next, create a folder called routes; this is technically part of the controller, but it is nice to have all the routing in one, separate folder.

Here, we have created userRoute.js to define all the routing in the chat application.

```
const express = require('express')
const user_route = express();
const userController = require('../controllers/userController')
const auth = require('../middlewares/auth')
user_route.get('/register', auth.isLogout, userController.registerLoad)
user_route.post('/register', upload.single('image'), userController.register)
user_route.get('/', auth.isLogout, userController.loadLogin)
user_route.post('/', userController.login)
user_route.get('/logout', auth.isLogin, userController.logout)
user_route.get('/dashboard', auth.isLogin, userController.loadDashboard)
user_route.post('/save-chat', userController.saveChat)
user_route.get('*', function(req, res){
```

```
    res.redirect('/')
})
```

To render the different view according to the defined route, we have created userController.js file inside controllers folder. This userController.js file accessed in userRoute.js by the help of require() method.

- registerLoad in userController.js is used to render register.ejs view.
- loadLogin is used to render login.ejs view.
- loadDashbaord is used to render the dashboard.ejs view.

Creating Model

Models are what communicate directly to our database.

Here we have created two models in our models folder.

userModel.js for userRegistration and chatModel.js for save the user chat.

userModel.js

```
const mongoose = require('mongoose')
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    image: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
```

```
        is_online: {
            type: String,
            default: 0
        }
    },
    {timestamps: true}
)
module.exports = mongoose.model('User', userSchema);
```

These are the fields we want to insert into the database whenever a new user registers through the Registration page. We can store a name, password, email address, is_online status and include a timestamp when the registration completes.

chatModel.js

```
const mongoose = require('mongoose')
const chatSchema = new mongoose.Schema(
    {
        sender_id:{
            type: mongoose.Schema.Types.ObjectId,
            ref: 'User'
        },
        receiver_id:{
            type: mongoose.Schema.Types.ObjectId,
            ref: 'User'
        },
        message:{
            type: String,
            required: true
        }
    },
    {timestamps: true}
)
module.exports = mongoose.model('Chat', chatSchema);
```

These are the fields we want to insert into the database whenever a user submits the message from dashboard Page. We can store a sender_id, receiver_id, message and include a timestamp.

Both models are exported by using model() method.

Connecting to the database

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/<database_name>');
```

End to End Encryption using DHKE and AES

Diffie-Hellman key exchange

Diffie-Hellman key exchange (DHKE) is a cryptographic algorithm that allows two parties to establish a shared secret over an insecure communication channel without any prior knowledge of each other's secret key. Here is a brief description of the algorithm:

- Choose parameters: The first step is to agree on a set of parameters that will be used for the key exchange. These parameters typically include a large prime number p and a primitive root g modulo p.
- Generate keys: Each party generates a random secret number, a and b, respectively. These secret numbers are kept private and are not shared with the other party.
- Exchange public keys: Each party calculates a public key by raising the agreed-upon primitive root g to the power of their secret number a or b, respectively, modulo the prime number p. The resulting public keys, A and B, are exchanged over the insecure communication channel.
- Compute shared secret: Each party takes the public key they received from the other party and raises it to the power of their own secret number. This results in a shared secret key that is the same for both parties.

Algorithm:

Alice	Bob
Public Keys available = P, G	Public Keys available = P, G
Private Key Selected = a	Private Key Selected = b
Key generated = X = $G^a P$	Key generated = Y = $G^b P$
Exchange of generated keys takes place	
Key received = Y	key received = X
Generated Secret Key = k_a = Y^amod P	Generated Secret Key = k_b= X^bmod P

Algebraically, it can be shown that

$k_a = k_b$

Users now have a symmetric secret key to encrypt

Advanced Encryption Standard

AES (Advanced Encryption Standard) is a symmetric encryption algorithm that is widely used to protect sensitive data. It was developed to replace the outdated DES (Data Encryption Standard) algorithm and is now considered the industry standard for symmetric encryption.

AES works by taking plaintext data and transforming it into ciphertext using a secret key. The ciphertext can then be transmitted over an insecure communication channel, and the recipient can use the same secret key to decrypt the cipher text and recover the original plaintext. AES operates on fixed block sizes of 128 bits and supports three key sizes: 128 bits, 192 bits, and 256 bits. The larger the key size, the stronger the encryption. AES is considered to be extremely secure, and it is used to protect a wide range of sensitive data, including financial transactions, government communications, and personal data.

One of the advantages of AES is its speed and efficiency. It can encrypt and decrypt data quickly, which makes it ideal for use in applications that require fast encryption and decryption, such as data storage and transmission.

Overall, AES is a highly secure and efficient encryption algorithm that is widely used to protect sensitive data. Its widespread adoption and high level of security make it an essential tool for protecting data in a variety of applications.

CryptoJS

CryptoJS is a JavaScript library that provides cryptographic functionality, including AES encryption and decryption. The library provides a simple API for encrypting and decrypting data using AES.

To encrypt data using AES with CryptoJS, you first generate a random secret key and convert the plaintext data into a WordArray object. You can then use the CryptoJS.AES.encrypt() method to encrypt the data, passing in the plaintext WordArray and the secret key. The encrypted data will be returned as a CipherParams object, which you can convert to a string using the toString() method.

Here, we have performed encryption in the following way:

let encryptedMessage = CryptoJS.AES.encrypt(message, sharedSecret.toString()).toString();

To decrypt data using AES with CryptoJS, you first convert the encrypted data back to a WordArray using the CryptoJS.enc.Hex.parse() method. You can then use the CryptoJS.AES.decrypt() method to decrypt the data, passing in the encrypted WordArray and the secret key. The decrypted data will be returned as a WordArray, which you can convert to a string using the CryptoJS.enc.Utf8.stringify() method.

Here, we have performed decryption in the following way:

let decryptedMessage = CryptoJS.AES.decrypt(data.message, sharedSecret.toString()).toString(CryptoJS.enc.Utf8);

Overall, encrypting and decrypting data using AES with CryptoJS is a simple and effective way to protect sensitive data in a web application.

4.2 Testing

System testing is done by evaluating output regarding different input. This test is done to evaluate whether the system is providing accurate summary or not. During the development phase of the system it is tested time to time. The series of testing conducted are as follow:

4.2.1 Test Cases for Unit Testing

Test Case	Test Data	Expected Outcome	Test Result
Registration	Any input field: null	Registration not successful	Pass
	Name: Bhuwan Email: bhuwan@bhuwan.com Choose file: 1.jpg Password: bhuwan	Registration successful	Pass
	Name: Sandip Email: sandip@sandip.com Choose file: 2.jpg Password: sandip	Registration successful	Pass

Table 4. 1 Unit Testing of User Registration

Test Case	Test Data	Expected Outcome	Test Result
Login	Email: null Password : null	Login not successful	Pass
	Email: bhuwan@bhuwan.com Password: bhuwan	Login successful	Pass

Table 4. 2 Testing of User Log In

4.2.2 Test Cases for System Testing

Scenario 1: User Log In

- First User move to log in page.
- In log in page, he/she write email and password which is correct one.
- If they insert the right email and password, they will be to home page else they will be unable to go to home page

Scenario 2: Chatting

- After user are redirected to home page, they will choose the available user.
- Now user can send and receive message from or to available users.

CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS

5.1 Lesson Learnt / Outcome

Every project makes us learn and gain knowledge in different aspects. In the following project, we learn things like teamwork, finding the solution on our own, proper use of guidelines, communication and writing skills and management of team. Throughout this project, we got confident with JS, HTML, CSS, MongoDB.

5.2 Conclusion

Overall, the project has achieved its objectives. The chat application provides a better, flexible, and secure system for chatting. It is developed with recent advanced technologies in a way to provide a reliable system. This application can find better need in the market for most of the organizations aim at having private applications for them. Chat application is one such social instant messaging application that brings you closer to your close friends and family.

5.3 Future Recommendations

There is always room for improvements in any software package, however good and efficient it may be made. But the most important thing should be flexible to accept further modification. Right now, we are just dealing with text communication. In future this software may be extended to include features such as:

- File transfer: This will enable the user to send files of different formats to others via the chat application.
- Group Chat: This will enable the user to create chat rooms.
- Update and delete chat messages.
- Voice chat: This will enhance the application to a higher level where communication will be possible via calling as in telephone.
- Video chat: This will further enhance the feature of calling into video communication.

REFERENCES

[1] Ban N.Dhannoon, Noor Sabah "Developing an End-to-End Secure Chat Appication", https://www.researchgate.net/profile/Ban-N-Dhannoon/publication/322509087_Developing_an_End-to-End_Secure_Chat_Application/links/5a5d1134458515c03ede7ead/Developing-an-End-to-End-Secure-Chat-Application.pdf

[2] Ruba Abu-Salma, Kat krol, Simon Parkin and Victoria Koh "The Security Blanket of the Chat World: An Analytic Evaluation and a User Study of Telegram", https://www.researchgate.net/publication/319622415_The_Security_Blanket_of_the_Chat_World_An_Analytic_Evaluation_and_a_User_Study_of_Telegram

[3] https://www.viber.com/app/uploads/viber-encryption-overview.pdf

[4] https://support.skype.com/en/faq/FA31/does-skype-use-encryption

[5] KILIÇ, Muhammed Burak. "Encryption methods and comparison of popular chat applications." *Advances in Artificial Intelligence Research* 1.2 (2021): 52-59.

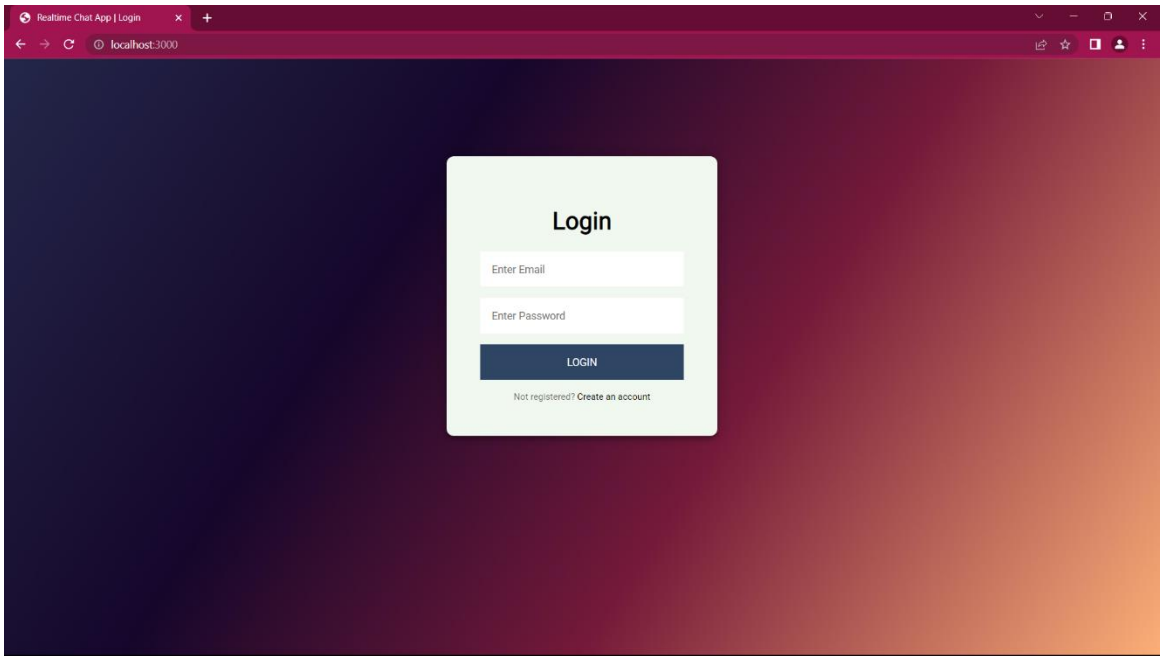
[6] Cantelon, Mike, et al. *Node. js in Action*. Greenwich: Manning, 2014.

[7] Bardis, Nikolaos G., and Konstantinos Ntaikos. "Design of a secure chat application based on AES cryptographic algorithm and key management." *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*. No. 10. WSEAS, 2008.

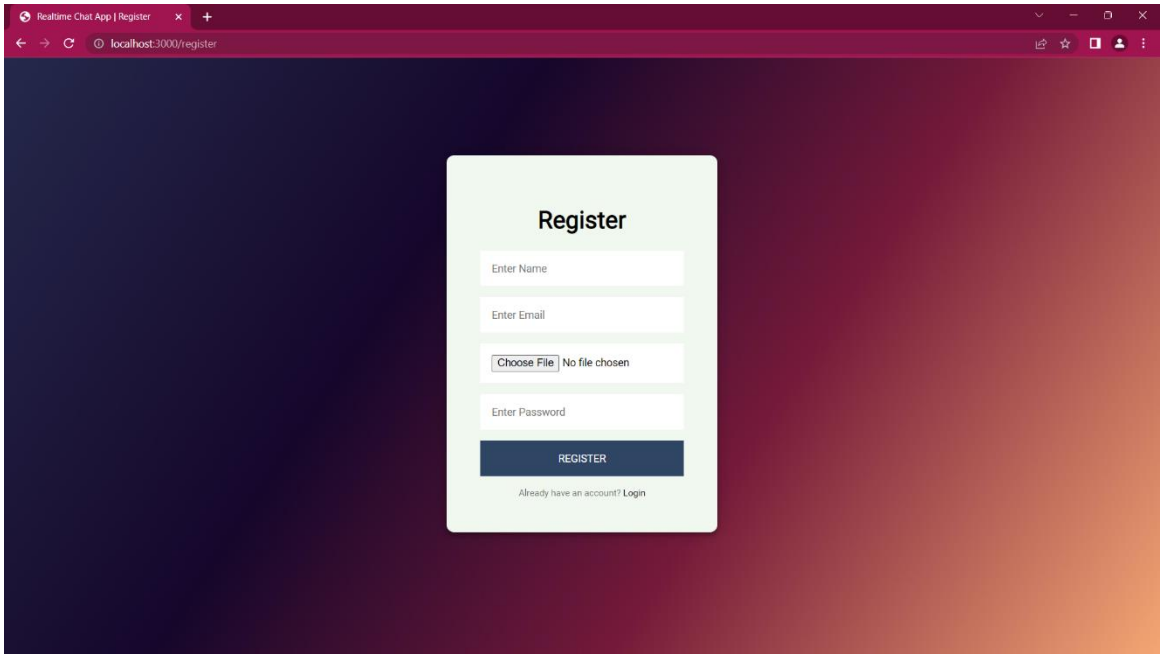
APPENDIX

Website Screenshots

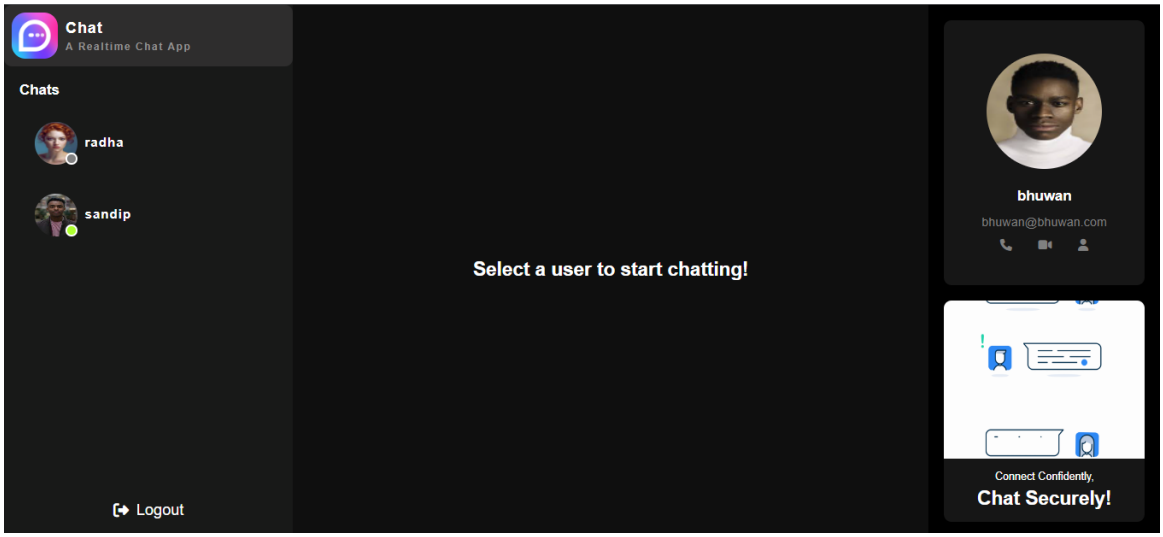
Here are the few screenshots of our project.



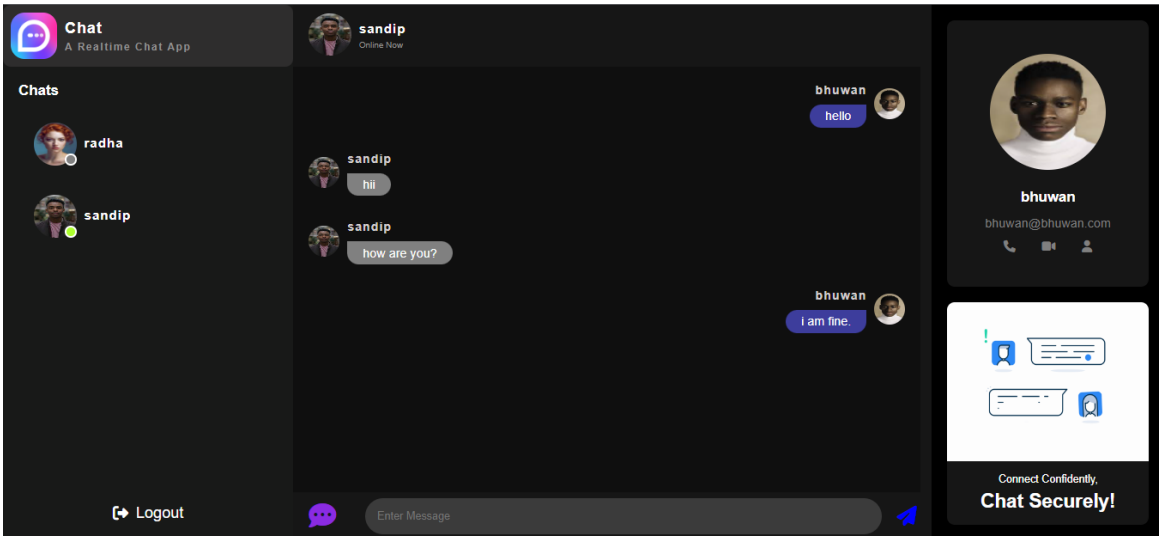
Snapshot of the Login Page



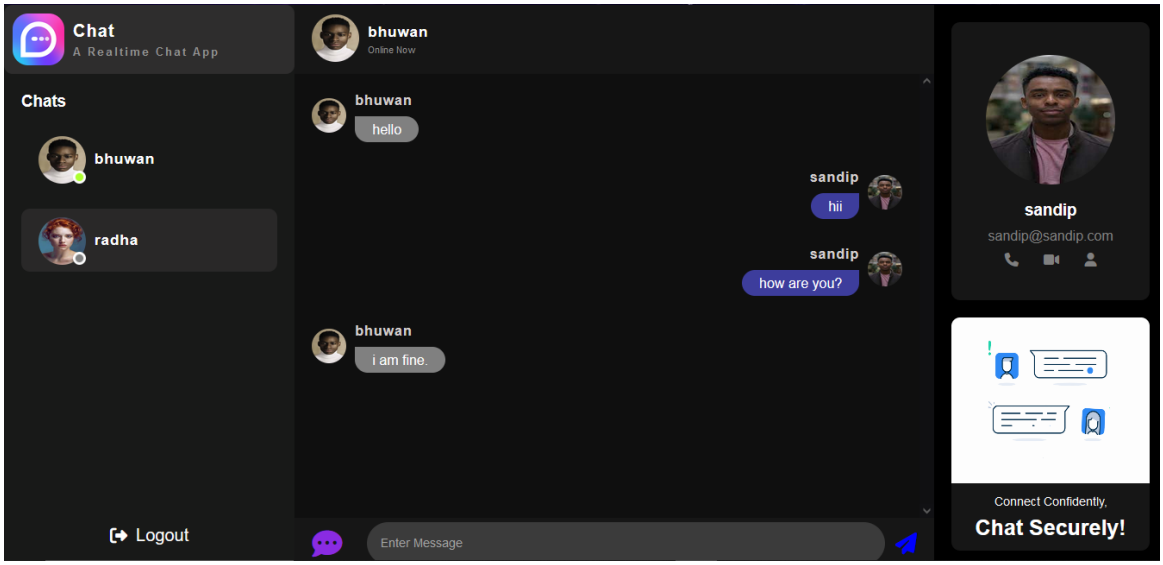
Snapshot of the Registration Page



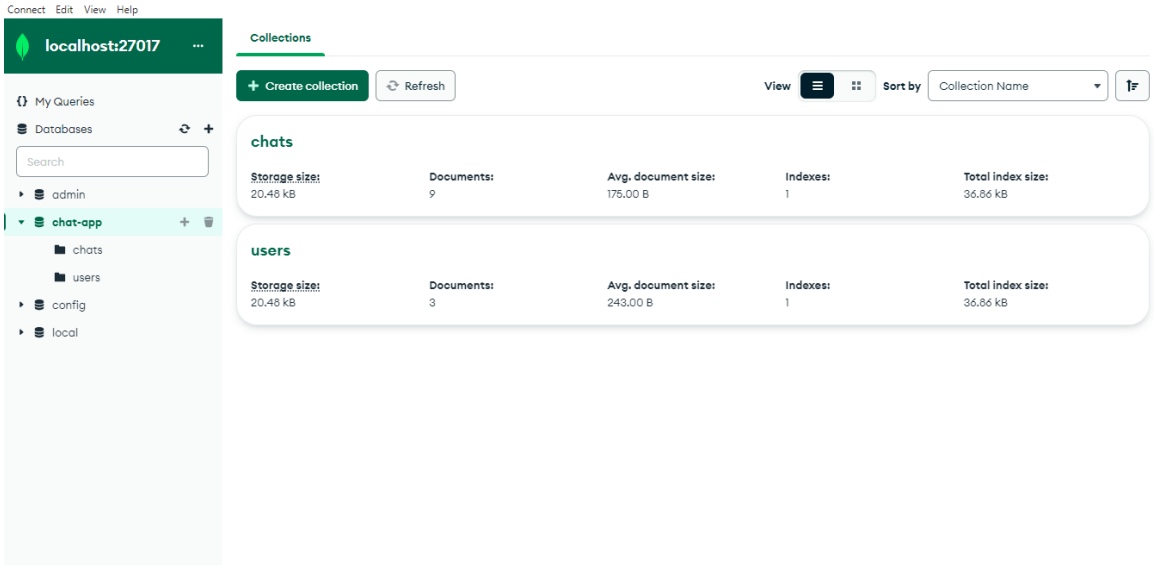
Snapshot of the Dashboard Page



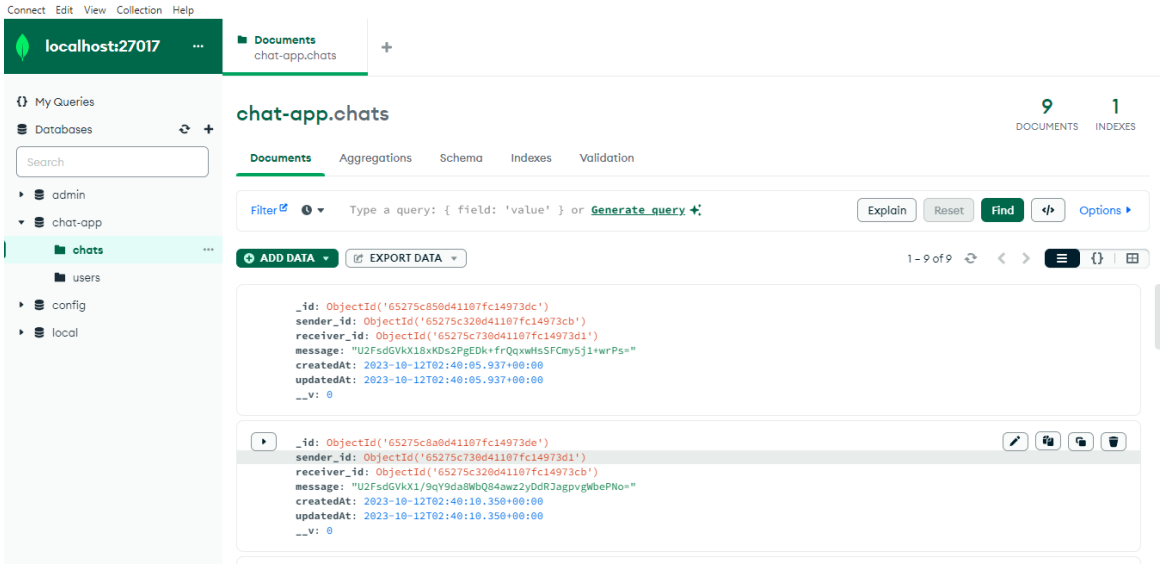
Snapshot of user's inbox



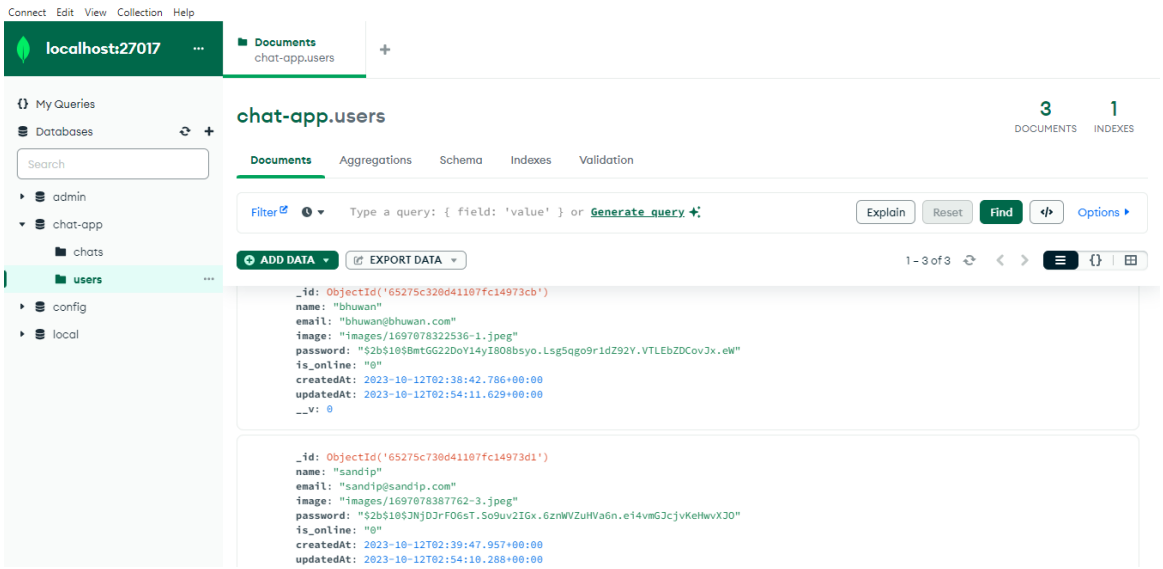
Snapshot of the One-to-One chat between two users



Snapshot of the database collection for the chat application in MongoDB Compass



Snapshot of the Chats Collection



Snapshot of the Users Collection