ChatCoat

Chatting Application

A major project report submitted in partial fulfillment of the requirement

for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering / Information Technology

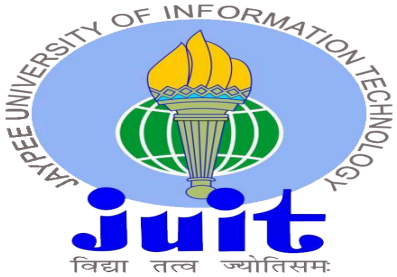
Submitted by

Tavishi Chauhan (201384)

Sejal Thakur (201377)

Under the guidance & supervision of

Dr. Kushal Kanwar



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

Candidate's Declaration

I thus certify that the work submitted in this report, titled **"ChatCoat Chatting Application"** partially satisfies the requirements for the award of a Bachelor of Technology in Computer Science & Engineering and is submitted to the Department of Computer Science & Engineering and Information Technology at Jaypee University of Information Technology, Waknaghat. This work was conducted under the supervision of **Dr. Kushal Kanwar**, Assistant Professor (SG), CSE, and spans the period from August 2023 to December 2023.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature with Date)
Student Name: Tavishi Chauhan
Roll No.: 201384
Student Name: Sejal Thakur
Roll No.: 201377

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature with Date)
Supervisor Name: Dr. Kushal Kanwar
Designation: Assistant Professor (SG)
Department: CSE
Dated:

Certificate

This certifies that the work being submitted in the project report titled **"ChatCoat Chatting Application,"** which was turned in to the Department of Computer Science and Engineering at Jaypee University of Information Technology, Waknaghat, in partial fulfillment of the requirements for the award of a B.Tech in Computer Science and Engineering, is an authentic record of the work completed by **Tavishi Chauhan (201384)** and **Sejal Thakur (201377**) between August 2023 and November 2023, under the direction of Dr. Kushal Kanwar of the Department of Computer Science and Engineering at Jaypee University of Information Technology, Waknaghat.

Tavishi Chauhan (201384)
Sejal Thakur (201377)

The above statement made is correct to the best of my knowledge.

Dr. Kushal Kanwar
Assistant Professor(SG)
Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat,

Acknowledgement

Our sincere gratitude and deepest debt of gratitude are extended to our supervisor, Dr. Kushal Kanwarf Jaypee University of Information Technology in Waknaghat. To complete this project, my supervisor must have deep knowledge of and a strong interest in the field of information security. This endeavor has been made possible by his unending patience, scholarly direction, constant encouragement, frequent and energetic supervision, constructive criticism, insightful advice, reviewing several subpar draughts and fixing them at every stage.

We would like to sincerely thank **Dr. Kushal Kanwar, Assistant Professor (SG)**, for all of his help and advice during this endeavor. We also thank Jaypee University of Information Technology for giving us access to all the tools we needed to

Tavishi Chauhan (201384)
Sejal Thakur (201377)

Table Of Contents

Title	Page No.
Abstract	VII
Chapter-1 (Introduction)	1
Chapter-2 (Literature Survey)	5
Chapter-3 (System Development)	10
Chapter-4 (Testing)	26
Chapter-5 (Results and Evaluation)	38
Chapter-6 (Conclusion and Future Scope)	30
References	31
Appendix	IX

List Of
Tables

Table Number	Page No.
Table (i)	5
Table(ii)	6
Table(iii)	7
Table(iv)	8
Table(v)	9

List Of Figures

Figure Number	Page No.
Figure(i)	13
Figure(ii)	14
Figure(ii)	14
Figure(iv)	15
Figure(v)	16
Figure(vi)	17
Figure(vii)	18
Figure(viii)	19
Figure(ix)	19
Figure(x)	20
Figure(xi)	21
Figure(xii)	22
Figure(xiii)	23
Figure(xiv)	24

List Of Abbreviations

Word	Abbreviation
MERN Stack	MongoDB, Express.js, React.js, Node.js
API	Application Programming Interface
UI	User Interface
UX	User Experience
JWT	JSON Web Token
TLS	Transport Layer Security
DB	Database
VCS	Version Control System

Abstract

In an era dominated by the need for instant connectivity, ChatCoat emerges as a dynamic and innovative Full Stack Chat application designed to revolutionize real-time communication. Leveraging advanced technologies such as Socket.io for instant messaging and MongoDB for secure data storage, ChatCoat not only solves the challenges of existing solutions, but also changes the face of messaging applications.

This project report explores the complexity of ChatCoat's development journey, starting with exploring the challenges facing modern communication platforms. The project aims to focus on real-time communication, robust security, and a feature-rich user experience. The importance and motivation behind the creation of ChatCoat lies in its ability to improve user interaction through enhanced privacy, versatility and innovation.

The report's systematic approach involves a literature review that provides insight into technologies, trends, and gaps available over the past five years. The system development department studied requirements, analysis, design principles, architecture, and data preparation, providing a comprehensive understanding of the technical basis of the project. Code snippets are strategically presented to highlight important features, including real-time messaging, user authentication, and group chat functionality.

The testing phase is complete and shows the complex testing strategy used and the results obtained from extensive test cases. The findings and insights from the pilot phase led to a comparative analysis with existing solutions, making ChatCoat a leader in user experience, security and innovative features.

Results and evaluations provide unique insight into project performance metrics, user feedback, and iterative improvements. The conclusion reflects the main findings, limitations and contributions of ChatCoat from conceptualization to implementation.

By looking to the future, exploring potential developments, emerging technologies, and continuous improvement strategies, it forms the basis for ChatCoat's evolution in the real communication industry.

ChatCoat is a testament to modern technology, user-centered design, and a commitment to reinventing the way individuals interact and communicate in the digital age.

Chapter 1: Introduction

1.1 Introduction

With its revolutionary Full Stack Chatting App, ChatCoat aims to transform real-time communication. ChatCoat appears as a solution to give consumers a seamless and safe instant messaging platform in a world where connectivity is essential. ChatCoat was developed in response to the growing need for effective communication solutions. It provides a feature-rich experience that goes beyond standard messaging apps.

The Need for ChatCoat

Instantaneous and dependable communication channels are essential for both individuals and enterprises in the fast-paced digital world of today. While useful, current messaging apps could be lacking in some areas or provide security risks. By combining the capabilities of MongoDB for safe user data storage and Socket.io for real-time communication, ChatCoat solves these problems.

Key Features

- **Real-Time Communication:** ChatCoat uses Socket.io to guarantee rapid message delivery, resulting in a responsive and dynamic user experience.
- **Authentication:**To protect user information and improve the general privacy of discussions, secure user authentication is used.
- **Group Chats and One-to-One Messaging:**Individual and group messaging ChatCoat meets a range of communication needs, from one-on-one discussions to group projects.
- **Search Functionality:** Search Functionality: Users may quickly locate and establish connections with other users, improving the platform's discoverability and accessibility.

User-Centric Approach

With an emphasis on usability, security, and simplicity, ChatCoat is created with the end user in mind. The programme seeks to give users a forum for effective communication, whether for group conversations, professional partnerships, or personal relationships.

Navigating the Introduction

This chapter gives a summary of the driving forces for ChatCoat's creation, the problems it seeks to solve, and an outline of its main characteristics. The next parts will go more deeply into the particulars of the project, providing a thorough overview of ChatCoat's development from idea to execution.

1.2 Problem Statement

The need for effective and secure real-time communication has become more than ever in the modern digital environment. The user experience of traditional messaging programmes may be hampered by issues including latency, a lack of sophisticated features, and privacy concerns. The idea behind ChatCoat came from the necessity for a complete solution.

Challenges in Existing Solutions

The following major issues may not be adequately addressed by current chat apps:

- **Latency Issues:** A lot of messaging applications have trouble sending messages instantly, which causes conversation to stall and take longer than expected.

- **Privacy Concerns:** Protecting the privacy of user data is crucial. Sensitive information may be exposed on some platforms since they may not have strong security mechanisms in place.

- **Limited Features:** Certain messaging apps don't have sophisticated features like cross-device seamless integration, group chat, or search capabilities.

- **Scalability:** This becomes an issue as user bases increase. Some platforms could find it challenging to manage a lot of users at once.

ChatCoat is a dependable, feature-rich, and secure platform for real-time communication that seeks to address these issues head-on.

1.3 Objectives

The main goals of ChatCoat are carefully designed to solve the problems found and provide a remarkable user experience:

Key Objectives

- **Real-Time Communication:**To guarantee immediate message delivery, put in place a reliable real-time messaging system utilizing Socket.io.

- **Security and Authentication:** Put user data security first by putting safe authentication systems in place to safeguard user accounts and chats.

- **Feature-Rich Interface:** Provide a feature-rich interface with user search capabilities, group chat features, one-to-one messaging, and an easy-to-use interface.

- **Scalability:**Create a scalable platform that can accommodate an increasing user base
- without sacrificing functionality.

- **Cross-Platform Compatibility:** Make sure ChatCoat is available on a range of platforms and devices, giving consumers a smooth experience on their chosen devices.

1.4 Significance and Motivation of the Project Work

The idea behind ChatCoat was the realization of how important communication is to day-to-day living. Successful and safe communication is essential in both personal and business contexts. In addition to addressing the shortcomings of current solutions, ChatCoat offers a platform that puts user privacy, responsiveness, and versatility first in an effort to improve communication.

Importance of ChatCoat

- **Enhanced User Experience:** ChatCoat prioritizes providing a user-centric experience by fusing simplicity and efficiency.

- **Privacy and Security:** ChatCoat offers a safe communication environment for users to interact without jeopardizing their sensitive information, in response to the growing concern over data privacy.

- **Versatility and Innovation:C** hatCoat seeks to surpass traditional messaging apps by integrating cutting-edge features, offering a varied and inventive

1.5 Organization of Project Report

The format of this project report is designed to give readers a thorough knowledge of ChatCoat's development process. Chapters that follow focus on particular facets, moving from the project's conception to its execution, testing, and assessment. The report's structure is intended to provide readers with a clear and logical flow of information as they progress through ChatCoat's development.

Report Structure Overview

- Introduction: The introduction provides a summary of the project, outlining its goals, significance, problem description, and report structure.

- Literature Survey: Examines the body of knowledge on real-time chat apps in the literature, pointing out any important gaps.

- System Development: System development includes information on the specifications, analysis, design, architecture, data preparation, implementation, and difficulties encountered.

- Testing: Talks about the methods, resources, test cases, and results of the testing process.

- Results and Evaluation:Summarizes the findings, provides an interpretation, and, if necessary, contrasts ChatCoat with other solutions already in place.

- Conclusions and Future Scope: Highlights major conclusions, restrictions, and contributions while outlining possible improvements in the future.

Readers can obtain a comprehensive grasp of ChatCoat's development processes and evolution by adhering to this well-organized framework.

Chapter 2: Literature Survey

2.1 Overview of Relevant Literature

S. No.	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitations
1.	Chat Application using Server-Side Scripting, Compression and End-to-End Encryption [1]	May, 2022	HTML, CSS, JavaScript, MySql, PHP	A chat application has essential parts - server and client.	Scalability and server load management may be challenging during high user traffic periods.
2.	Chat Application [2]	April, 2022	MEAN stack, web sockets, and Angular's 2 way binding	Python-based global chat/file-sharing app with client-server, TCP, and chat features.	Limited platform compatibility as Python may not be available or convenient on all devices.

Table (i)

S. No.	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitations
3.	Internet Chat Application [3]	2021	Forwarding data, point to point connection between browsers, and Threads from Multithreading.	The research paper presents the development of a browser-based real-time chatting tool that can be used directly through a web browser without requiring additional client software.	May lack advanced features and integrations available in dedicated chat clients.
4.	Group Chat Application [4]	2021	Java, multi threading, and client server concept	The research paper outlines a chat system implemented in Java, leveraging multi-threading and network concepts.	Java-based applications may require users to have Java Runtime Environment installed, which can be a barrier for some.

Table (ii)

S. No.	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitations
5.	Multi-User Chat Application [5]	2020	Java multithreading and network concept	The research paper describes a chat application where clients can enter their names, send messages to all users or specific users, and exit the chat.	User anonymity may be limited as clients must enter their names.
6.	Realtime Chat Application using Client-Server Architecture [6]	2019	TCP attachment, Socket, Client, GUI, Local Host, Tkinter	The research paper presents an application consisting of a server and multiple client connection points.	The attachment module may introduce complexity and potential bugs in the communication process.

Table (iii)

S. No.	Paper Title [Cite]	Journal/ Conference (Year)	Tools/ Techniques/ Dataset	Results	Limitations
7.	Professional chat application based on natural language processing [7]	2018	(NLP) techniques	The research paper describes the development of a professional chat application designed to prevent users from sending inappropriate or improper messages to other participants.	The effectiveness of the content filtering algorithms and AI in identifying inappropriate content is not detailed.
8.	A Secure Chat Application Based on Pure Peer-to-Peer Architecture [8]	2015	socket programming, user profile database, hash function, cryptographic algorithm.	The research paper presents a decentralized chat application that has undergone testing in a local area network.	User needs to register n number of times in order to warrant communication with n different peers and user needs to

Table (iv)

S. No.	Paper Title [Cite]	Conference (Year)	Tools Used	Results	Limitations
9.	Enhanced Chat Application [9]	2012	LAN, Client-Server architecture.	The research paper highlights a chat application with a primary focus on facilitating communication through the use of diagrams and figures.	Accessibility and ease of use for users who may not be familiar with diagrammatic communication methods.
10.	UDP based chat application [10]	2010	socket programming, User Datagram Protocol (UDP), and Java	The research paper introduces a method for creating a chat room using socket-based communication, specifically based on the User Datagram Protocol (UDP).	UDP, while lightweight and suitable for real-time communication, does not guarantee the order or reliability of message delivery, which can lead to message loss or out-of-order

Table (v)

Chapter 3: System Development

3.1 Requirements and Analysis

Navigating the Blueprint: ChatCoat's Development Journey

Requirement Gathering

An extensive investigation of the functional and non-functional requirements served as the foundation for the development of ChatCoat. This section delineates the multifarious requirements of users, spanning from the complexities of group chat functionality to real-time texting capabilities. Determining these needs set the foundation for a reliable and easy-to-use programme.

Stakeholder Input and Collaboration

Working together with stakeholders was essential in determining ChatCoat's features. The development team made decisions based on feedback from administrators, other stakeholders, and potential users to make sure the end product met user expectations and industry standards.

Analysis Phase

During the analysis step, the requirements were broken down into particular details. Determining the scalability needs to support an expanding user population, creating communication protocols, and defining user responsibilities were all part of this process.

MERN Stack

- **MongoDB**
  The project's NoSQL database, MongoDB, offers a scalable and adaptable way to store user data. Making use of MongoDB's document-oriented structure enables effective administration and retrieval of data.

- **Express.js**
  The backend framework used to construct the server and manage HTTP requests is called Express.js. Because of its flexibility and small weight, it's the perfect option for managing server-side logic and building reliable APIs.

- **React.js**
  The client-side of ChatCoat is powered by React.js, which provides a declarative and effective user interface building tool. Because of its component-based architecture, creating dynamic and responsive user interfaces for smooth chat experiences is made easier.

- **Node.js**
  Server-side JavaScript can be executed since Node.js functions as the server's runtime environment. Its event-driven, non-blocking architecture guarantees peak performance while managing real-time communication using Socket.io.

Socket.io

- **Real communication**
  Socket.io is a critical component to enable real-time, two-way communication between client and server. It facilitates instant messaging, indexing and other real-time features, improving the overall user experience.

- **WebSocket protocol**
  Socket.io uses the WebSocket protocol to provide a low-latency and efficient communication channel. This ensures fast and reliable data exchange, making it ideal for applications that require real-time updates.

- **API testing and development**
  Mail becomes a comprehensive API development and testing tool that enables teams to efficiently design, test, and document APIs. An intuitive interface lets you create requests, verify endpoints, and validate API responses.

- **Automated testing**
  Mail also supports automated testing, which allows teams to create and run a batch of API requests to test the functionality of multiple endpoints. This ensures the reliability and consistency of the API during the development and testing phase.

  ○

Additional technology

- **Heroku (deployment)**
  Heroku is used to host the ChatCoat application on the Internet. The platform-as-a-service (PaaS) model makes it easy to deploy, scale, and manage applications by providing access to users over the Internet.

- **npm (node package manager)**
  npm is a package manager for Node.js that makes it easy to install and manage project dependencies. It manages adding, updating, and removing packages, contributing to project continuity.

- **Git and GitHub (version control)**
  Git is used for version control, which allows collaboration between team members and tracking changes to the code base. GitHub functions as a remote repository, providing a centralized platform for code deployment, collaboration, and tracking.

- This comprehensive suite of technologies provides the robustness, scalability and real-time capabilities required for ChatCoat projects. Each technology plays a specific role in creating rich and reliable chat applications.

- **ChakraUI**
  Chakra UI is an open source React component library that provides a set of accessible and customizable UI components for building web applications. It is designed to be highly modular, so developers can easily add or remove the components they need to create a custom UI design.

- **JWT**
  JWT stands for JSON Web Tokens, a popular open standard for securely transferring data between parties as JSON objects. JWT is often used for authentication and authorization in web applications.

Technical requirements (tools)

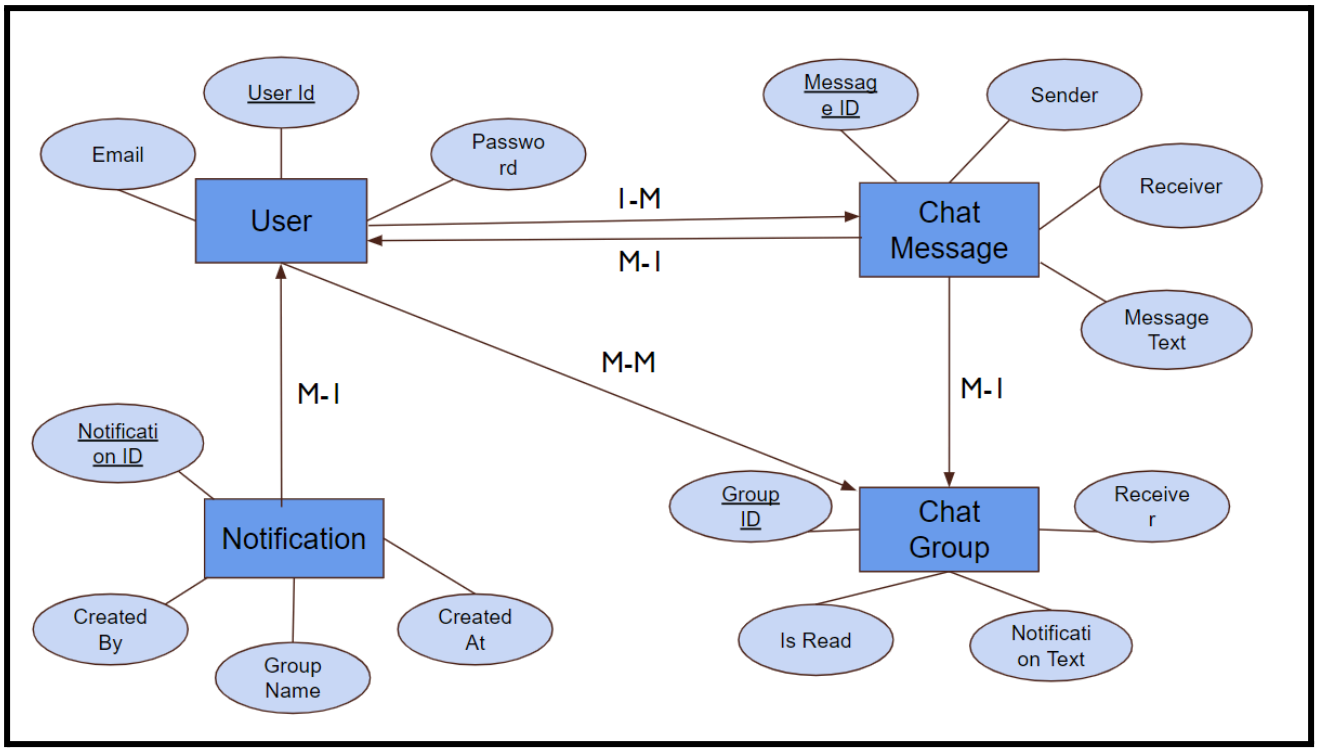
- CPU: Intel Core i3 or higher, 64-bit, dual-core, 1.30 GHz
- GPU: AMD Radeon R5 or higher, 64-bit, 320 bit, 0.86 GHz
- Ram: 4 GB or more
- Hard disk: 5 GB available space or more.
- Display: Dual XGA (1024 x 768) or high resolution monitor
- Operating system: Windows

3.2 Project Design and Architecture

Blueprints of Innovation: Creating Chat Coats Design

Design Principles

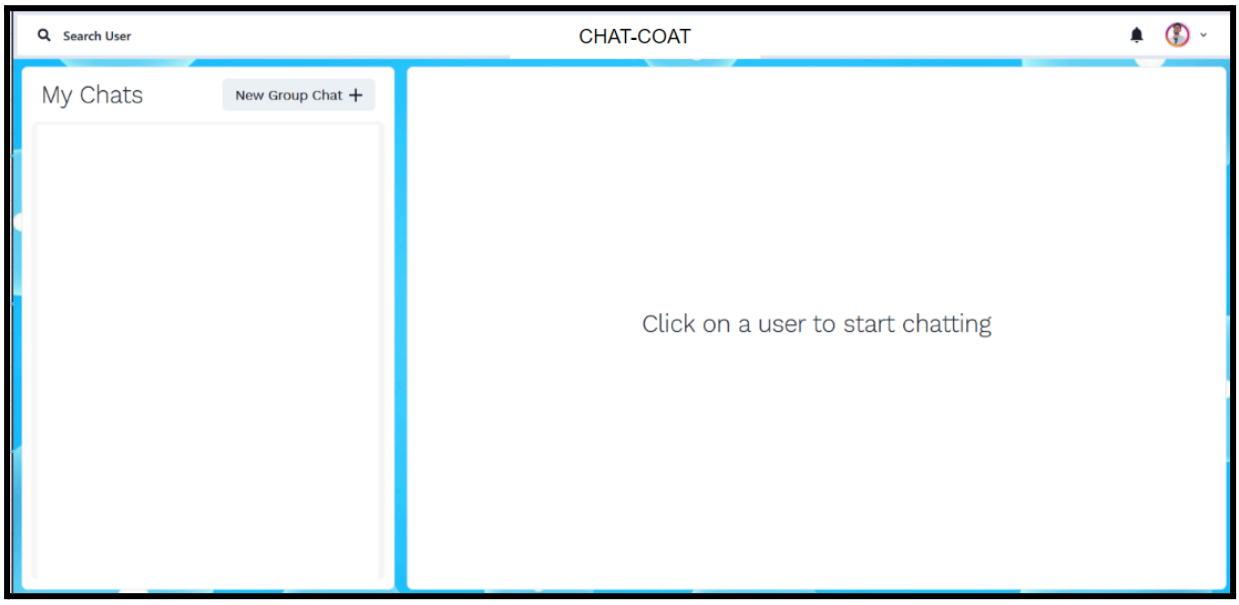
The requirements for an aesthetically beautiful layout, smooth navigation, and an intuitive user interface served as the foundation for ChatCoat's design philosophy. This section explores the design concepts that influenced the interactive and visual components of the programme.



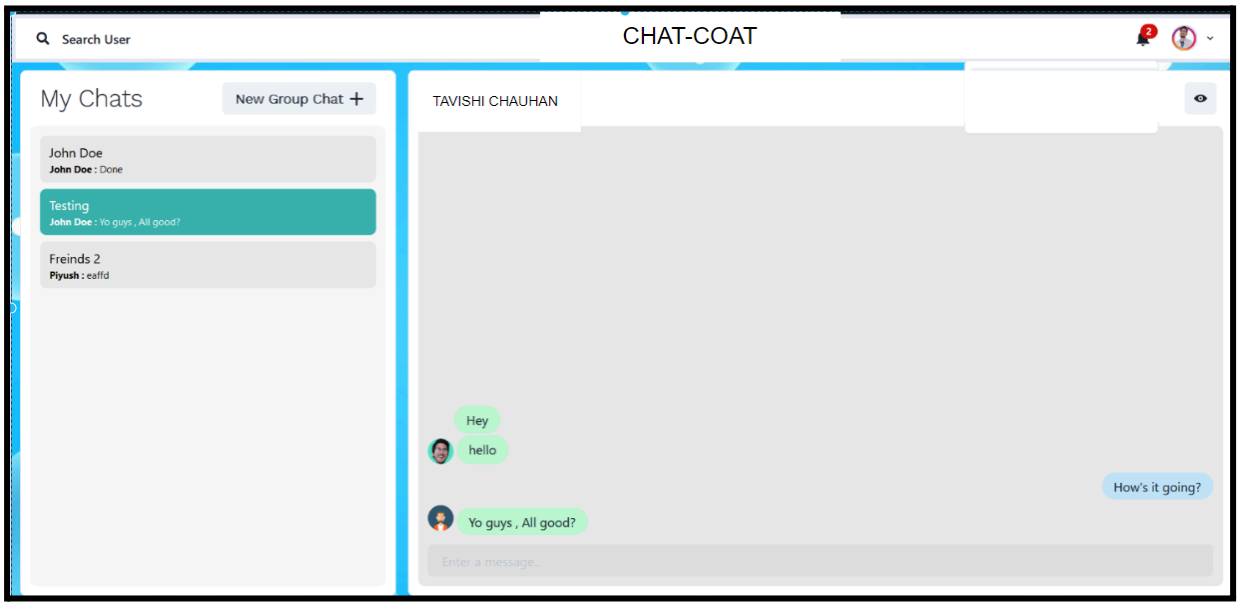
Figure(i)

User Experience (UX) Design

The design choices were made with the user experience in mind. ChatCoat's UX design placed a high priority on accessibility and usability, from deliberate feature placement to straightforward navigation, ensuring that users could interact with the programme with ease.



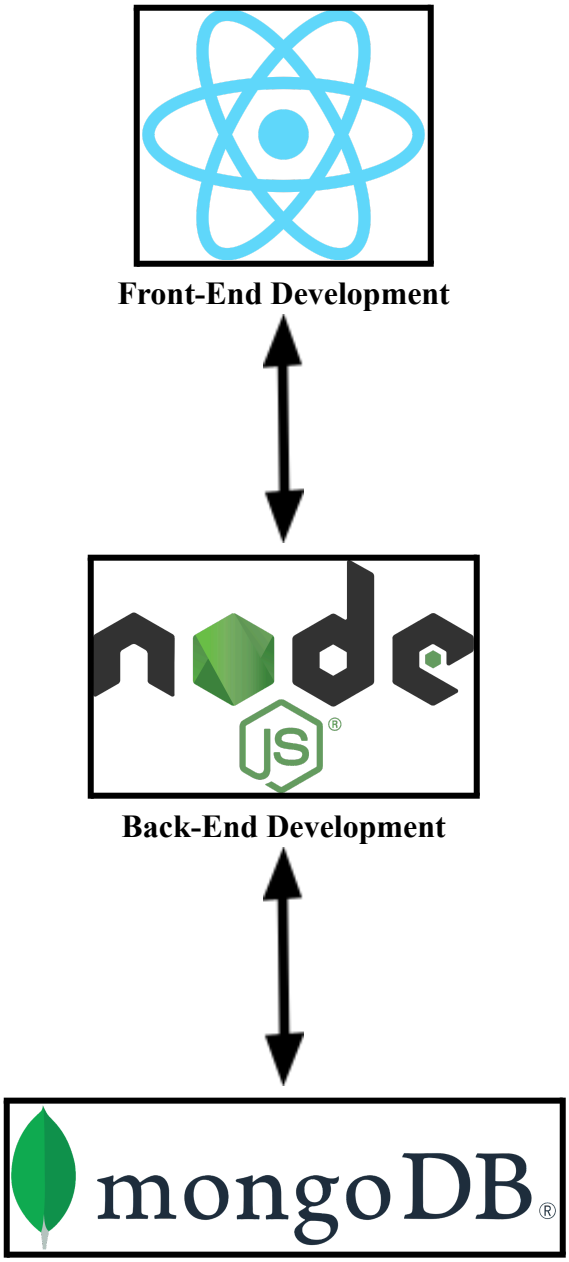
Figure(ii) Main Screen



Figure(iii) Chat Page Interface

System Architecture

This section explores the nuances of ChatCoat's system architecture and describes the interactions between the client (React JS), server (Node JS, Express JS), and database (MongoDB). The selection of technologies and their functions in developing an application that is both responsive and scalable are covered.



Figure(iv) Database Management

3.3 Data Preparation

Property protection: user data on ChatCoat

Data storage and encryption

A description of how user data is processed and stored is integral to understanding ChatCoat's commitment to user privacy. This section describes the encryption methods used to protect sensitive user data, highlighting the importance of data security.

user profile management

The process of creating and managing user profiles is explored, shedding light on the strategy used to ensure data accuracy, integrity, and user autonomy in controlling profile data.

3.4 Implementation (code snippets, algorithms, tools and methods, etc.)

Symphony of Code: Behind the Scenes of ChatCoat

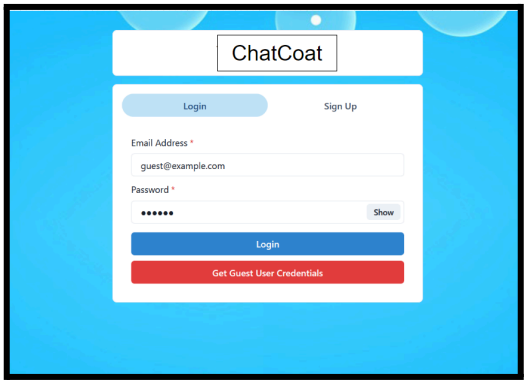
Technology suite

A deep dive into the technologies used in the ChatCoat implementation are React JS for the client, Node JS and Express JS for the server, and MongoDB for the database. This section provides a high-level overview of the role each technology plays in practice.

code snippet

Key sections of the codebase provide insight into the implementation of key features such as real-time messaging, user authentication, and group chat functionality. Code snippets are accompanied by comments to make them easy to understand.

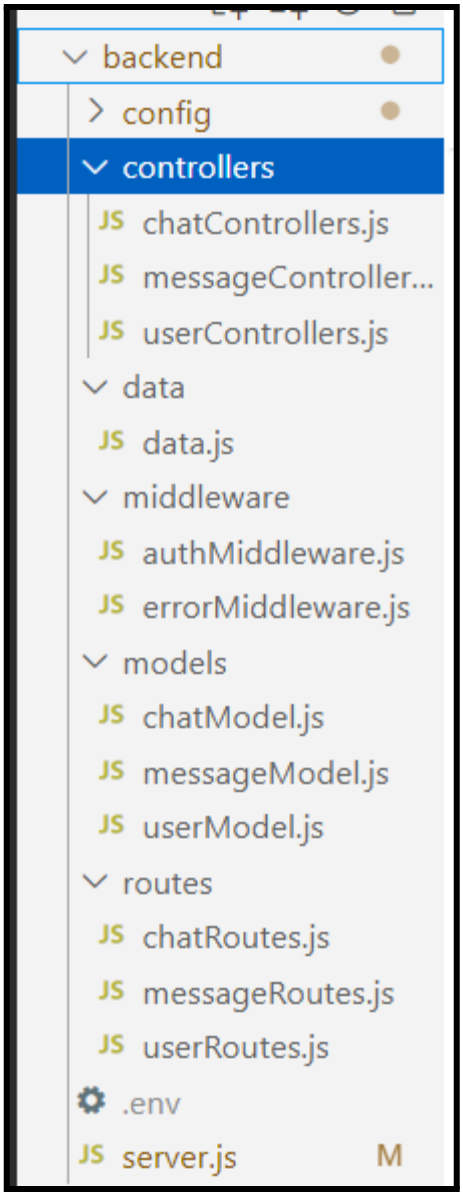
```
const Chatpage = () => {
  const [fetchAgain, setFetchAgain] = useState(false);
  const { user } = ChatState();

  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>
  );
};

```



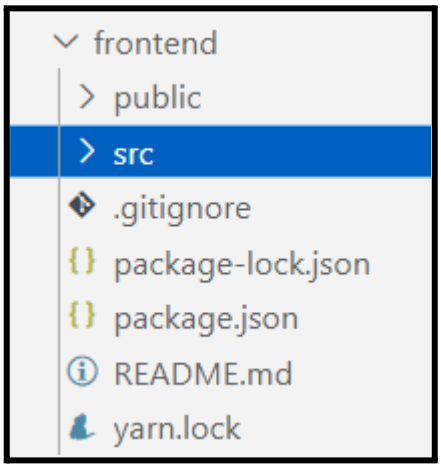
Figure(v) Login and Sign Up form

```
const mongoose = require("mongoose");
const colors = require("colors");

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(
      "mongodb+srv://tavisshiChauhan:tavisshiChauhan@cluster0.mz8vc3l.mongodb.net/",
      {
        useNewUrlParser: true,
        useUnifiedTopology: true,
      }
    );

    console.log(`MongoDB Connected: ${conn.connection.host}`.cyan.underline);
  } catch (error) {
    console.error(`Error: ${error.message}`.red.bold);
    process.exit(1); // Exit with a non-zero status code to indicate an error
  }
};

module.exports = connectDB;
```

Figure(vi) Encryption of password before sending it to the database



Figure(vii) Backend Directory



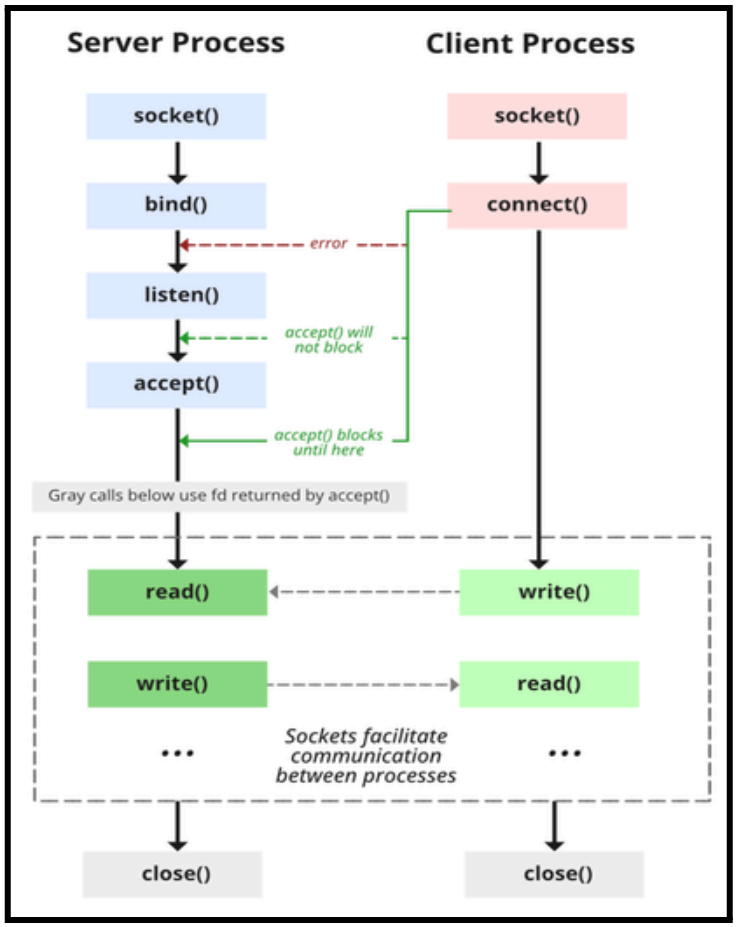
Figure(viii) Frontend Directory

```
10  dotenv.config();
11  connectDB();
12  const app = express();
13
14  app.use(express.json()); // to accept json data
15
16  // app.get("/", (req, res) => {
17  //   res.send("API Running!");
18  // });
19
20  app.use("/api/user", userRoutes);
21  app.use("/api/chat", chatRoutes);
22  app.use("/api/message", messageRoutes);
23
```

Figure(ix) Server.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;

```

Figure(x) chatRoute.js



Figure(xii) State Diagram for server and client model of socket

CODE

Directory: Controllers/chatController

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

//@description     Create or fetch One to One Chat
//@route           POST /api/chat/
//@access          Protected
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");

```

```
54  //@description     Fetch all chats for a user
55  //@route           GET /api/chat/
56  //@access          Protected
57  const fetchChats = asyncHandler(async (req, res) => {
58    try {
59      Chat.find({ users: { $elemMatch: { $eq: req.user._id } } })
60        .populate("users", "-password")
61        .populate("groupAdmin", "-password")
62        .populate("latestMessage")
63        .sort({ updatedAt: -1 })
64        .then(async (results) => {
65          results = await User.populate(results, {
66            path: "latestMessage.sender",
67            select: "name pic email",
68          });
69          res.status(200).send(results);
70        });
71    } catch (error) {
72      res.status(400);
73      throw new Error(error.message);
74    }
75  });
76
77  //@description     Create New Group Chat
78  //@route           POST /api/chat/group
79  //@access          Protected
80  const createGroupChat = asyncHandler(async (req, res) => {
81    if (!req.body.users || !req.body.name) {
82      return res.status(400).send({ message: "Please Fill all the feilds" });
83    }
84
85    var users = JSON.parse(req.body.users);
```

```
  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);
  }
});

```

```
117  const renameGroup = asyncHandler(async (req, res) => {
118    const { chatId, chatName } = req.body;
119
120    const updatedChat = await Chat.findByIdAndUpdate(
121      chatId,
122      {
123        chatName: chatName,
124      },
125      {
126        new: true,
127      }
128    )
129      .populate("users", "-password")
130      .populate("groupAdmin", "-password");
131
132    if (!updatedChat) {
133      res.status(404);
134      throw new Error("Chat Not Found");
135    } else {
136      res.json(updatedChat);
137    }
138  });
```

```
143  const removeFromGroup = asyncHandler(async (req, res) => {
144    const { chatId, userId } = req.body;
145
146    // check if the requester is admin
147
148    const removed = await Chat.findByIdAndUpdate(
149      chatId,
150      {
151        $pull: { users: userId },
152      },
153      {
154        new: true,
155      }
156    )
157      .populate("users", "-password")
158      .populate("groupAdmin", "-password");
159
160    if (!removed) {
161      res.status(404);
162      throw new Error("Chat Not Found");
163    } else {
164      res.json(removed);
165    }
166  });
```

```
const addToGroup = asyncHandler(async (req, res) => {
  const { chatId, userId } = req.body;

  // check if the requester is admin

  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);
  }
});
```

```
  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);
    }
  }
});
```

Directory: Controllers/messageController

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

//@description     Get all Messages
//@route           GET /api/Message/:chatId
//@access          Protected
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);
  }
});

//@description     Create New Message
//@route           POST /api/Message/
//@access          Protected
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 };

```

Directory: Controllers/userController

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

//@description     Get or Search all users
//@route           GET /api/user?search=
//@access          Public
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);
});

//@description     Register new user
//@route           POST /api/user/
//@access          Public
const registerUser = asyncHandler(async (req, res) => {
  const { name, email, password, pic } = req.body;

  if (!name || !email || !password) {
    res.status(400);
    throw new Error("Please Enter all the Feilds");
  }
```

```
35    if (userExists) {
36      res.status(400);
37      throw new Error("User already exists");
38    }
39
40    const user = await User.create({
41      name,
42      email,
43      password,
44      pic,
45    });
46
47    if (user) {
48      res.status(201).json({
49        _id: user._id,
50        name: user.name,
51        email: user.email,
52        isAdmin: user.isAdmin,
53        pic: user.pic,
54        token: generateToken(user._id),
55      });
56    } else {
57      res.status(400);
58      throw new Error("User not found");
59    }
60  });
```

```
62  //@description     Auth the user
63  //@route           POST /api/users/login
64  //@access          Public
65  const authUser = asyncHandler(async (req, res) => {
66    const { email, password } = req.body;
67
68    const user = await User.findOne({ email });
69
70    if (user && (await user.matchPassword(password))) {
71      res.json({
72        _id: user._id,
73        name: user.name,
74        email: user.email,
75        isAdmin: user.isAdmin,
76        pic: user.pic,
77        token: generateToken(user._id),
78      });
79    } else {
80      res.status(401);
81      throw new Error("Invalid Email or Password");
82    }
83  });
84
85  module.exports = { allUsers, registerUser, authUser };
86
```

Directory: 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.get("/", (req, res) => {
//   res.send("API Running!");
// });

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..");
  });
}
```

```
42  // Error Handling middlewares
43  app.use(notFound);
44  app.use(errorHandler);
45
46  const PORT = 9092;
47
48  const server = app.listen(
49    PORT,
50    console.log(`Server running on PORT ${PORT}...`.yellow.bold)
51  );
52
53  const io = require("socket.io")(server, {
54    pingTimeout: 60000,
55    cors: {
56      origin: "http://localhost:3000",
57      // credentials: true,
58    },
59  });
60
61  io.on("connection", (socket) => {
62    console.log("Connected to socket.io");
63    socket.on("setup", (userData) => {
64      socket.join(userData._id);
65      socket.emit("connected");
66    });
67
```

```
61  io.on("connection", (socket) => {
62    console.log("Connected to socket.io");
63    socket.on("setup", (userData) => {
64      socket.join(userData._id);
65      socket.emit("connected");
66    });
67
68    socket.on("join chat", (room) => {
69      socket.join(room);
70      console.log("User Joined Room: " + room);
71    });
72    socket.on("typing", (room) => socket.in(room).emit("typing"));
73    socket.on("stop typing", (room) => socket.in(room).emit("stop typing"));
74
75    socket.on("new message", (newMessageRecieved) => {
76      var chat = newMessageRecieved.chat;
77
78      if (!chat.users) return console.log("chat.users not defined");
79
80      chat.users.forEach((user) => {
81        if (user._id == newMessageRecieved.sender._id) return;
82
83        socket.in(user._id).emit("message recieved", newMessageRecieved);
84      });
85    });
86
87    socket.off("setup", () => {
88      console.log("USER DISCONNECTED");
89      socket.leave(userData._id);
90    });
91  });
92
```

Algorithms and Techniques
This section explores the technological details that make ChatCoat a reliable and effective programme, from the algorithms controlling message delivery to the methods used for user identification.

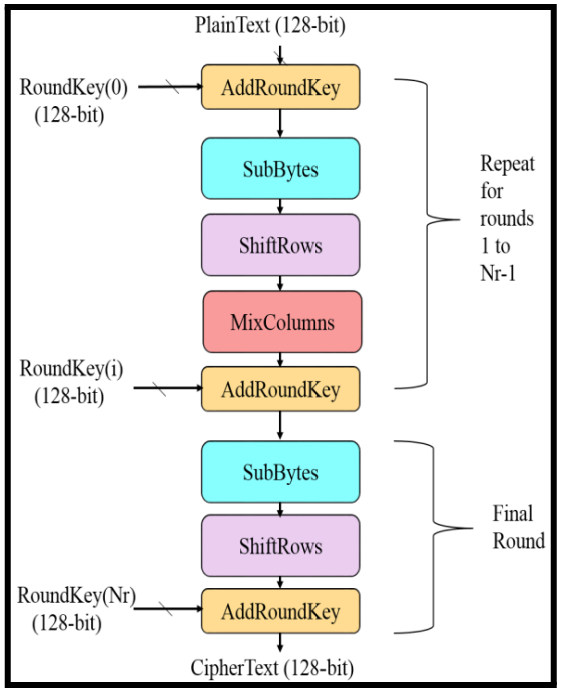
AES: Essentially, AES works by organizing each plaintext block into a 4x4 matrix and applying a series of operations on it repeatedly. We do 10, 12, or 14 rounds, depending on the key length (an additional parameter selected by NIST), and refer to each iteration as a round:
With ten rounds, a 128-bit key
12 cycles for a key of 196 bits
A 256-bit key requires 14 rounds.

We create a round key for every round based on the main key.
The 4x4 matrix has four operations.

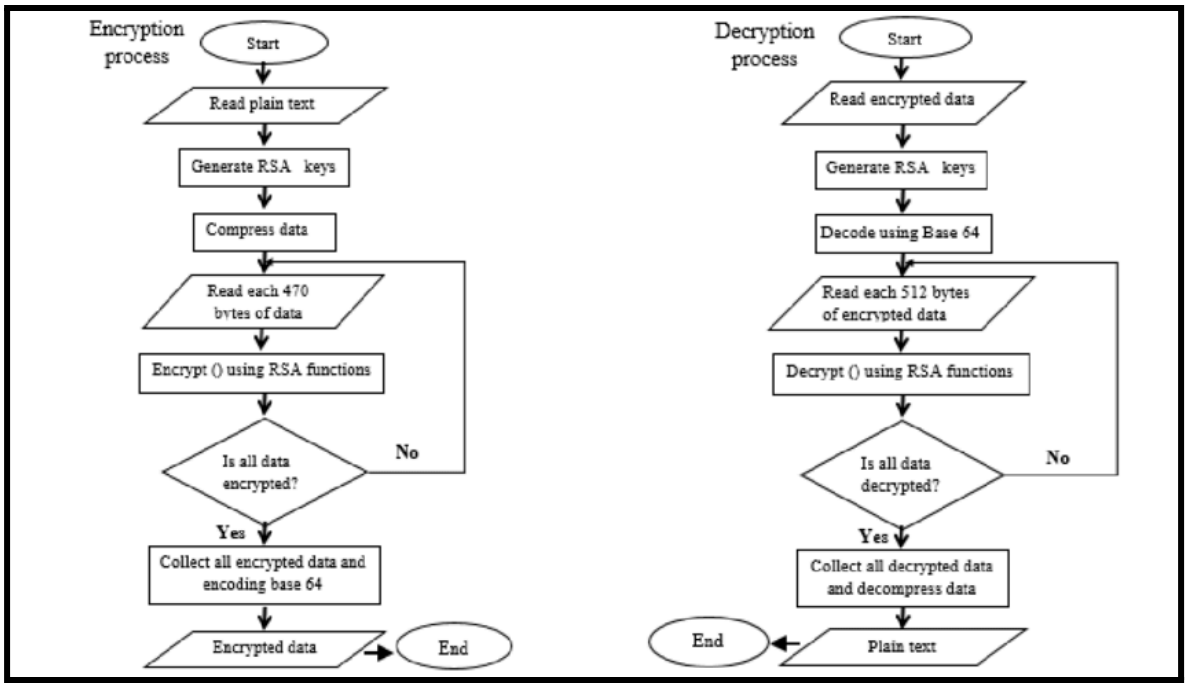
We shall define the following four operations on the 4x4 matrix:

subBytes()
moveRows()
combineColumns()
includeRoundKey()
Making a Public Key:
Choose two prime numbers.
Assume that Q = 59 and P = 53.
The public key's first portion is now as follows: n = P*Q = 3127.
We additionally require a tiny exponent, let's say e:
But an integer must be exist.
Not modified to $\Phi(n)$.
$1 < e < \Phi(n)$
We use n and e to create our public key.

To generate the private key, we must compute $\Phi(n)$:
such that $(P-1)(Q-1) = \Phi(n)$
Consequently, $\Phi(n) = 3016$
Proceed to compute the Private Key, d: $d = (k*\Phi(n) + 1) / e$, where k is an integer.
2011 is the value of d for k = 2.



Figure(xiii) AES Encryption Flowchart



Figure(xiv) Flowchart for RSA encryption and decryption

3.5 Key Challenges (discuss the challenges faced during the development process and how these are addressed)

Navigating Challenges: Challenges in ChatCoat development

The challenge of expansion
It discusses the challenges of scaling, and how the development team anticipated and addressed potential problems with the growing number of users.

The challenge of integration
Challenges faced during the integration of various components, such as integrating Socket.io for real-time communication and ensuring smooth compatibility across different devices and platforms.

security challenges
It explores the challenges of ensuring the highest security standards, especially in the management of user data. Measures taken to reduce security risks are discussed.

Chapter 4: Testing

4.1 Testing Strategy (discuss the testing strategy/tools used in the project)

- **Unit Testing:**
  - Objective: Confirm that each of the application's sections or parts functions as intended.

- **Components to Test:**

  - Server-side operations (such as Socket.io event handlers and Express.js routes).
  - To render UI elements, use React components.
  - Tools: Jest and Enzyme can be used for testing Node Js
  -

- **Integration Testing:**
  - Objective: Make sure that the application's many sections or modules function as a cohesive whole.

- **Components to Test:**

  - communication between the client and server via API calls.
  - integrating Socket.io to facilitate communication in real time.
  - Tools:Socket.io testing tools for real-time communication, and Supertest for API integration testing.

- **End-to-End Testing:**
  - Objective: Test the application's whole flow by modeling actual user situations.

- **Scenarios to Test:**

  - Authentication and user registration.
  - one-on-one conversation features.
  - Creating and interacting in group chats.
  - Cypress or Selenium are good tools for end-to-end

- **Performance Testing:**
  - Objective: Goal: Evaluate the application's stability, scalability, and responsiveness under various load scenarios.

- **Scenarios to Test:**

  - Assume a sizable number of people conversing in real time at the same time.
  - Measure reaction times when there are several demands.
  - Tools: artillery.io or Apache JMeter.

- **Security Testing:**
    - Objective: The goal is to locate and fix any possible security holes in the programme.

- **Areas to Test:**

    - Validate input to stop injection attacks.
    - mechanisms for authentication.
    - secure data transfer via HTTPS.
    - Tools: SonarQube or OWASP ZAP.

4.2 Test Cases and Outcomes

- **Test Case Design:**
  Create test cases for every scenario that the testing strategy identifies. To cover a variety of eventualities, include test cases that are both positive and negative.

- **Test Execution:**
  Utilize the appropriate testing frameworks and tools to carry out the test cases. Keep track of and document every test case's results, including any errors or strange behavior.

Outcomes and Adjustments:

- Examine the test data to find and address any problems.

- Make changes to the codebase in light of the testing's lessons learned.

This thorough testing approach ensures that the ChatCoat application is secure, dependable, and resilient by examining many aspects of it. Adapt the details to the special attributes and capabilities of your application.

Chapter 5: Results and Evaluation

5.1 Results (presentation of findings, interpretation of the results, etc.)

Unlocking Concepts: ChatCoat Results and Conclusions

Presentation of results

This section presents the results obtained from extensive testing and evaluation of ChatCoat. Metrics, statistics, and user feedback are combined to provide a comprehensive picture of program performance.

Interpretation of results

Explore the interpretation of results, performance metrics results, user satisfaction scores, and identified areas for improvement. This section aims to offer a nuanced understanding of the data generated during testing.

Analysis of user feedback

Positive and constructive user feedback unlocks the user experience. This section shows how user feedback and preferences align with ChatCoat's intended goals and the adjustments made based on this valuable input.

Iterative development

Discuss the iterative addition to ChatCoat based on the results obtained. From minor UI tweaks to significant back-end optimization, this section shows the evolution of the application in response to the testing phase.

5.2 Comparison with Existing Solutions (if applicable)

Navigating the landscape: ChatCoat in messaging apps

Check available solutions

Take a look at today's messaging app landscape, identifying the key players and their key features. This section sets the stage for a comprehensive comparison with ChatCoat.

Attribute analysis

Conduct a feature-by-feature analysis comparing ChatCoat to existing solutions. Learn how ChatCoat is different in terms of user experience, security, scalability, and all the innovative features that set it apart.

Compare user experience
Explore the user experience offered by ChatCoat compared to other messaging apps. Evaluate intuitive design, responsiveness, and overall user satisfaction, draw conflicts, and identify areas for improvement.

Security and privacy considerations
Evaluate the security measures implemented in ChatCoat compared to other messaging applications. Note any innovative security features and their impact on protecting user data.

Chapter 6: Future Scope

The possibility
The project can be de[ployed on Heroku this can be considered as a future scope

Emerging technologies
We can also integrate use of emojis and gifs while chatting the make the experience more personalized

User engagement strategy
Explore strategies to engage users more, such as gamification elements, multimedia sharing capabilities, or collaboration features that go beyond traditional messaging apps.

Continuous improvement
Emphasizes the importance of continuous improvement in the rapidly evolving landscape of communications technology. Discuss how ChatCoat aims to stay at the forefront of innovation and adapt to the changing needs and expectations of its users.

References

[1] Patel, A., Khanna, M. S., & Shyam, R. (2022). Chat Application using Server-Side Scripting, Compression and End-to-End Encryption. Journal of Web Development and Web Designing, 7(2), 8-12.

[2] Kolambe, M., Sable, S., Kashivale, V., & Khaire, P. Chat Application.

[3] Kumar, T. S., Reddy, V., DL, S., & Rananavare, L. (2021). INTERNET CHAT APPLICATION. International Journal of Advanced Research in Computer Science,

[4] Tsai, T. C., Liu, T. S., & Han, C. C. (2017). WaterChat: A Group Chat Application Based on Opportunistic Mobile Social Networks. J. Commun., 12(7), 405-411.

[5]Gayathri, R., & Kalieswari, C. (2020). Multi-User Chat Application.

[6] Henriyan, D., Subiyanti, D. P., Fauzian, R., Anggraini, D., Aziz, M. V. G., & Prihatmanto, A. S. (2016, October). Design and implementation of web based real time chat interfacing server. In 2016 6th International Conference on System Engineering and Technology (ICSET) (pp. 83-87). IEEE.

[7] Karthick, S., Victor, R. J., Manikandan, S., & Goswami, B. (2018, February). Professional chat application based on natural language processing. In 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC) (pp. 1-4). IEEE.

[8] Mohamad Afendee, M., Abdullah, M., & Mustafa, M. (2015). A secure chat application based on pure peer-to-peer architecture. Journal of Computer Science, 11(5), 723-729.

[9]. Bamane, A., Bhoyar, P., Dugar, A., & Antony, L. (2012). Enhanced Chat Application. Global Journal of Computer Science and Technology Network, Web & Security, 12(11), 1-7.

[10] Malhotra, A., Sharma, V., Gandhi, P., & Purohit, N. (2010, April). UDP based chat application. In 2010 2nd International Conference on Computer Engineering and Technology (Vol. 6, pp. V6-374). IEEE.