A Major Project Report

on

DESIGNING A WEB BASED CHAT INTERFACE SERVER USING MERN STACK

Submitted in partial fulfilment of the requirements for the award of the degree

Of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

By

KODIDALA KOUSHIK KUMAR	**20EG105125**
SAI SPANDANA ECHAMBADI	**20EG105142**
SHAIK MOHAMMED KAIF	**20EG105145**
MARRI NITHIN REDDY	**20EG105153**

Under the guidance of

Mr. D. RAMANA KUMAR

Assistant Professor

Department of CSE

ANURAG UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Venkatapur(V), Ghatkeshar(M), Medchal(D) – 500088

Academic Year - 2023-2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the report entitled **"DESIGNING A WEB BASED CHAT INTERFACE SERVER USING MERN STACK"** that is being submitted by Kodidala Koushik Kumar (20EG105125), Sai Spandana Echambadi (20EG105142), Shaik Mohammed Kaif (20EG105145), Marri Nithin Reddy (20EG105153**)** in partial fulfilment for the award of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING** to the Anurag University, Hyderabad is a record of bonafide work carried out by them under my guidance and supervision.

The results presented in this report have been verified and found to be satisfactory. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Mr. D. Ramana Kumar	**Dr. G. Vishnu Murthy**
Assistant Professor, Dept. of CSE	**Professor &Dean, Dept. of CSE**

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Mr, D. Ramana Kumar** for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improvedour grasp of the subject and steered to the fruitful completion of the work. His patience, guidanceand encouragement made this project possible.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering,Anurag University, for his encouragement and timely support in our B. Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, Dean, Dept. of CSE, Anurag University. We also express our deep sense of gratitude to**Dr. V V S S S Balaram,** Academic coordinator, **Dr. Pallam Ravi**, Project in-Charge, **Dr. G. Prabhakar Raju** Project Coordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage our project work.

KODIDALA KODIDALA KUMAR

(20EG105125)

SAI SPANDANA ECHAMBADI

(20EG105142)

SHAIK MOHAMMED KAIF

(20EG105145)

MARRI NITHIN REDDY

(20EG105153)

DECLARATION

We, hereby declare that the Report entitled **“DESIGNING A WEB BASED CHAT INTERFACE SERVER USING MERN STACK”** submitted for the award of Bachelor of technology Degree is our original work and the Report has not formed the basis for the award of anydegree, diploma, associate ship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

KODIDALA KODIDALA KUMAR

(20EG105125)

SAI SPANDANA ECHAMBADI

(20EG105142)

SHAIK MOHAMMED KAIF

(20EG105145)

MARRI NITHIN REDDY

(20EG105153)

PLACE: HYDERABAD

ABSTRACT

A chat application is a type of software or technology that allows users to communicate with one other in real time over the internet. The article outlines an idea for messaging software that emphasizes security and usability above all else, with the goal of being adaptable across several platforms. The proposed method is starting with Node.js and building a server with the Express.js framework. Furthermore, the programme makes use of the Socket.io module to provide efficient user communication. MongoDB is used by the system for database administration. The solution uses bcrypt.js for password hashing and JSON Web Token to improve security measures.

TABLE OF CONTENTS

LIST OF IMAGES AND TABLES

1. INTRODUCTION

1.1 Introduction:

Chat applications allow you to stay connected with other people who may be using the application even on the other side of the world. In customer service, such applications are one of the most important communication channels. Working of Chat Applications: Chat applications typically run on centralized networks that are served by platform operator servers as opposed to peer-to-peer protocols such as XMPP. This allows two people to talk to each other in real time.[1][8]

At this time, the development of information and communication technology is growing. With the information and communication technologies to facilitate the public to obtain information and communicate from anywhere, anytime. Currently, there is a wide range of chat applications that have been developed by outsiders. A wide variety of chat application has its own advantages and disadvantages. But. The disadvantages over-powered then advantages in many cases like security, multi- device usage and many more.[3] Therefore, from the above problems, it became the background to design a web-based chat application.

Socket.io: Socket.IO is an event-driven library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It consists of two components: a client, and a server.

Node js: Node.js is a cross-platfrm, open-source JavaScript runtime environment that can run on Windows, Linux, Unix, macOS, and more. Node.js runs on the V8 JavaScript engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting.

Express js: Express.js, or simply Express, is a back-end web application framework for building RESTful APIs with Node.js, released as free and open-source software under the MIT License.

Json web token: JSON Web Token is an open standard for securely transferring data within parties using a JSON object. JWT is used for stateless authentication mechanisms for users and providers, this means maintaining sessions on the client side instead of storing sessions on the server. Here, we will implement the JWT authentication system in NodeJs.

Bcrypt js hashing: To avoid the sensitive data being visible to anyone, Node. js uses “bcryptjs.” This module enables storing passwords as hashed passwords instead of plaintext.

System is primarily developed with the ideology of mern stack. MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack.[7]

- MongoDB — document database
- Express(.js) — Node.js web framework
- React(.js) — a client-side JavaScript framework
- Node(.js) — the premier JavaScript web server

Express and Node make up the middle (application) tier. Express.js is a server-side web framework, and Node.js is the popular and powerful JavaScript server platform. Regardless of which variant you choose, ME(RVA)N is the ideal approach to working with JavaScript and JSON, all the way through.

1.2 Project Scope:

The scope of system should be declared before move advancing to the next step. System scope are as follows:

1. Design and construction of this chat application can be used on multiple devices using the login credentials.
2. This system was developed uses Node.js, Express.js and socket.io.
3. Database of this chat application using MongoDB.
4. For higher security, the design uses bcrypt.js hashing.

1.3 Objectives:

An objective then can be defined from the main problem above that is build a dedicated real-time multi-platform chat application which emphasises on high security and helps the users in keeping their credentials and chats secure.

2. LITERATURE SURVEY

The study delves into the historical challenges faced by early chat applications, highlighting vulnerabilities in security and limitations in language support. These early systems struggled to ensure robust security measures, leaving users vulnerable to privacy breaches and data leaks.[1] Moreover, language limitations hindered effective communication across diverse user bases, limiting the reach and usability of these platforms. These challenges underscored the need for innovation in the development of chat applications to address security concerns and enhance language capabilities for broader accessibility.[1][3]

However, subsequent advancements in technology have paved the way for more sophisticated solutions to these challenges. One such example is the utilization of modern frameworks like node.js and databases like MongoDB to develop chat applications. By leveraging these technologies, developers have been able to enhance security measures and improve the scalability of chat platforms. Furthermore, the integration of comprehensive stacks like MERN (MongoDB, Express.js, React, Node.js) has enabled the creation of feature-rich applications that offer a more seamless communication experience for users.[4][5][6]

Despite these strides, the journey towards creating an ideal chat application is not without its hurdles. Challenges such as interface design complexities, compatibility issues across multiple systems, and integration challenges persist. Projects like Lifeline Messenger and CHATTY serve as testaments to these ongoing struggles, with varying degrees of success in addressing these challenges. These endeavours highlight the importance of continually refining and adapting chat application development approaches to meet evolving user needs and technological advancements.[3][7][8]

while there has been significant progress in developing chat applications that prioritize security, usability, and comprehensive features, challenges persist in achieving the perfect balance. The evolution from early iterations with glaring security and language limitations to modern solutions employing advanced technologies underscores the iterative nature of software development. Moving forward, continued innovation and collaboration within the developer community will be crucial in overcoming remaining challenges and creating chat applications that truly cater to the diverse needs of users worldwide.[2][5][7]

Table 1

S.NO	METHODS	ADVANTAGES	DISADVANTAGES
1	Effective communication across diverse user bases and ensuring robust ecosystem [1]	1. It offers easy scalability: Node.js isn't more scalable than PHP or Ruby, but it is way easier to scale than other backend technologies. 2. Understand nodejs event-loop architecture, non-blocking I/O and event driven programming	1.Its performance is reduced with heavy computational tasks: Node.js is unable to process heavy CPU-bound tasks, and this is arguably one of the biggest drawbacks of Node.js. 2.It has an unstable API: Node.Js implements changes frequently connected to its API. The issue is that those changes are generally backward-incompatible with previous versions.
2	Vulnerabilities in security and limitations in language support [2]	1.Quick and live updating because of react.js and focuses on Native app development 2.Focus on the language: It is designed in a way that helps one chat in different languages.	1. Node.js relies heavily on callbacks. It is a function that runs after a task in the queue is finished, allowing other code to run in the meantime. However, with a number of tasks in the queue — each with its own callback —

		Code runs seamlessly on various operating systems.	it can lead to callback overloading. 2. Single-threaded nature 3.Does not encourage creating an application for a specific entity rather focuses on generalized features.
3	Utilization of modern frameworks like node.js and databases like MongoDB to develop chat applications [3]	1.Data binding, the code disappears, helping the developer concentrate on the actual application. 2.The driver automatically maps JavaScript objects to BSON documents, meaning that developers can easily work with their data. 3.Supports DOM manipulation. As AngularJS supports MVC architecture, it eliminates the need to manipulate the DOM directly from code inside the controller.	1.Not very in-depth: while including the basics of node js, it sacrifices the installation and details of some main elements. 2.Less Secure: Anyone can breach ones network by just entering the localhost details, hence is prone to network breach.

4	Glaring security and privacy to modern solutions [4]	1.Robust ecosystem through npm. 2.Maintains high secure 3.Is highly scalable and the frameworks installed are reliable to use	1.Does not support multiple user interfaces. 2.Cannot be used in multiple devices and cannot save the data 3. Lacks Library support: There is no monitoring system for accessing and removing the files from the Library.
5	Continued innovation and collaboration within the developer community [5]	1.A well-integrated system that supports chat application on multiple devices . 2.Provides multiply services like friend list management, chats, history management and many more.	1. The system needs to deployed timely as and when there is update in the Java Applet. 2. The application with designed using PHP consumes a lot of data and time.
6	Importance of continually refining and adapting chat application development [6]	1.Provides module-by-module code with clear explanation. 2.Analysis the different proposals to server-side solutions.	1.It does not cover complete details of Node API like fetching service schemes or creating middleware to keep the node active and solid. 2.It does not use any node frameworks like

			network traffic collection tools.
7	The integration of comprehensive stacks like MERN [7]	1.Media-sharing features enhances the overall user experience 2. prioritizes the security and privacy of its users 3.Revolutionizes communication by offering real-time messaging, eliminating unproductive meetings.	1. The updates made are not that efficient in working 2. The response time is higher but overall system's PI is more and is efficient.
8	Seeking comprehensive solutions for language limitations [8]	1.To develop a multi lingual chat application 2.To develop an interactive system that incorporated Pidgin English into chatting	1.Only registered users can use the system 2.Internet must be available to use the application 3.There must be minimum of two users per time for interactive chatting

3. PROPOSED METHOD

3.2. Problem-Identification:

In designing a real-time chat application interface, several critical problems often arise, impacting user experience and functionality. Firstly, one common issue is the lack of intuitive navigation and organization within the interface.[6] Users may struggle to find essential features or navigate between different chats efficiently, leading to frustration and decreased engagement. This problem becomes more pronounced as the number of conversations and participants increases, highlighting the need for a streamlined and user-friendly interface design.

Secondly, the challenge of maintaining context within conversations presents a significant hurdle. In dynamic group chats or conversations with multiple participants, users may find it challenging to track the flow of discussion, leading to misunderstandings, or missed messages. Without effective mechanisms for threading conversations or highlighting key messages, users can quickly become overwhelmed by the volume of information, hindering effective communication and collaboration.[2][4]

Finally, ensuring robust real-time communication without sacrificing data security and privacy remains a critical concern. Vulnerabilities in the application's infrastructure or encryption protocols can compromise sensitive user information or expose conversations to unauthorized access. Balancing the need for seamless communication with stringent security measures poses a complex challenge for developers, requiring continuous monitoring and updates to mitigate potential risks and maintain user trust.[6][4] Addressing these problems is crucial for creating a chat interface that delivers a seamless and secure communication experience for users across various platforms and devices.

3.2 Existing Model:

Mobile applications and standalone chatting applications have become increasingly popular in recent years.[7][8] These software tools allow users to communicate with one another through text messages or voice calls, all within the convenience of their handheld devices. The rise of smartphones has contributed to the widespread use of mobile apps, as they provide easy access to various functionalities and services.[5][3] Furthermore, standalone chatting applications offer a dedicated platform solely for communication purposes. They enable users to connect with friends, family, or colleagues and engage in seamless conversations.

3.3 Proposed Model:

A software application that operates on the internet, accessible through various platforms, designed to provide users with an easy-to-use interface. This application utilizes socket.io and express js as server modules, while employing mongodb for the database management aspect. For enhanced security measures, this software incorporates methods such as jwt (JSON Web Tokens) and password hashing.

A web-based software is created using Node.js to convert large image formats into smaller ones. This software is user-friendly and faster than other Node.js modules. It has features like text messaging, expressive emojis and stickers, group chats, notifications, and security measures.

3.3.1 Proposed Model Architecture:

Here is a proposed architecture diagram of chat application.

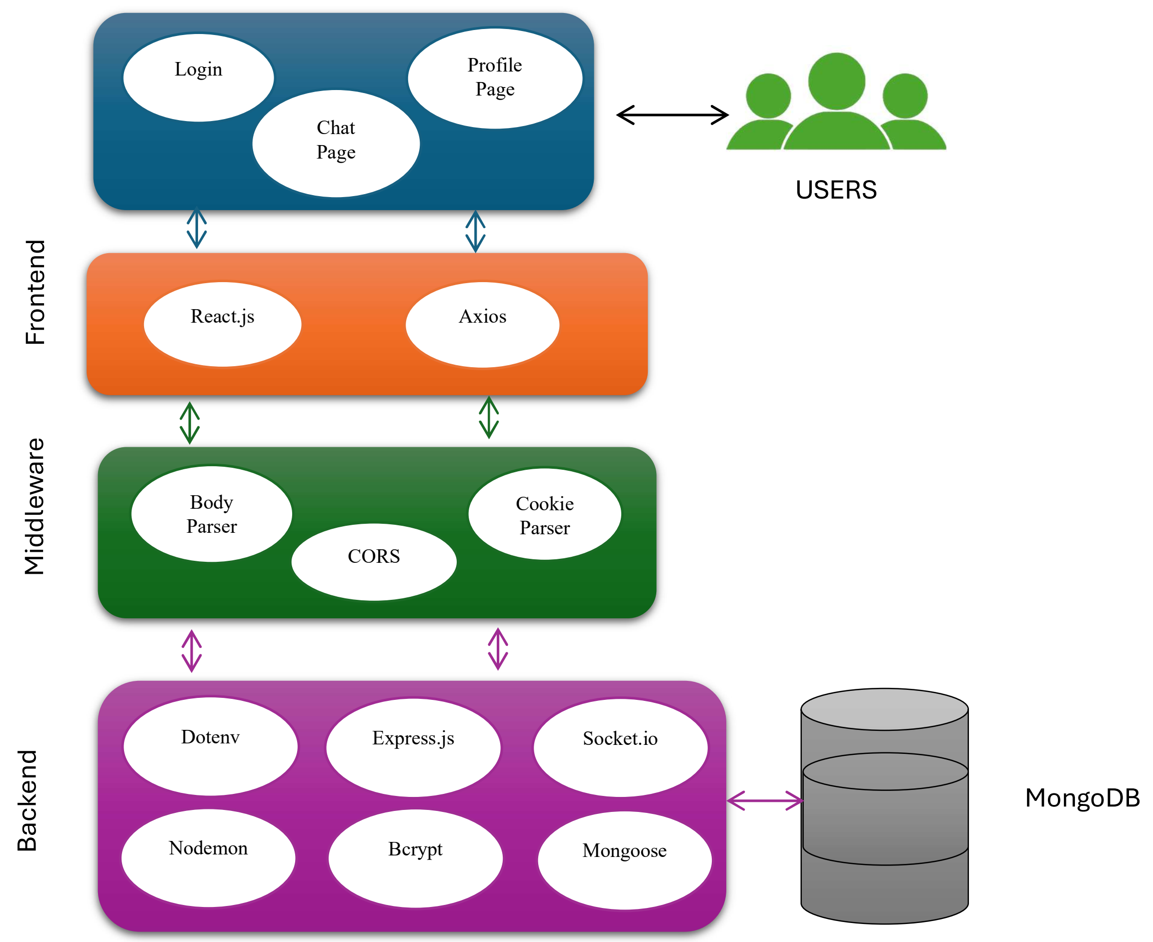


Figure 2:System Architecture

3.4 UML Diagram:

Unified Modeling Language (UML) diagrams are graphical representations widely used in software engineering to model software systems. They provide a standardized and visual way to describe various aspects of a system's architecture, structure, behavior, and interactions. UML diagrams serve as a common language for communication among stakeholders involved in software development, including developers, designers, project managers, and clients. These diagrams cover a wide range of perspectives, including architecture, structure, behavior, interactions, and deployment, enabling stakeholders to understand and communicate complex software concepts more effectively.

3.4.1 Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purpose of Use Case Diagrams

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

Following are the purposes of a use case diagram given below:

It gathers the system's needs.

It depicts the external view of the system.

It recognizes the internal as well as external factors that influence the system.

It represents the interaction between the actors.

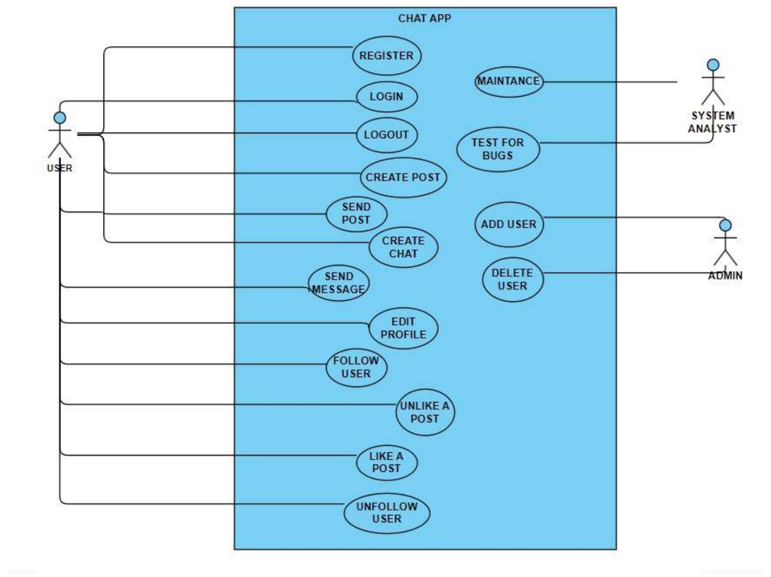


Figure 2: Use Case Diagram

3.4.2 Sequence Diagram

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

Sequence Diagrams captures:

- The interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams)
- high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)

Purpose of Sequence Diagram

Model high-level interaction between active objects in a system

Model the interaction between object instances within a collaboration that realizes a use case

Model the interaction between objects within a collaboration that realizes an operation

Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction)

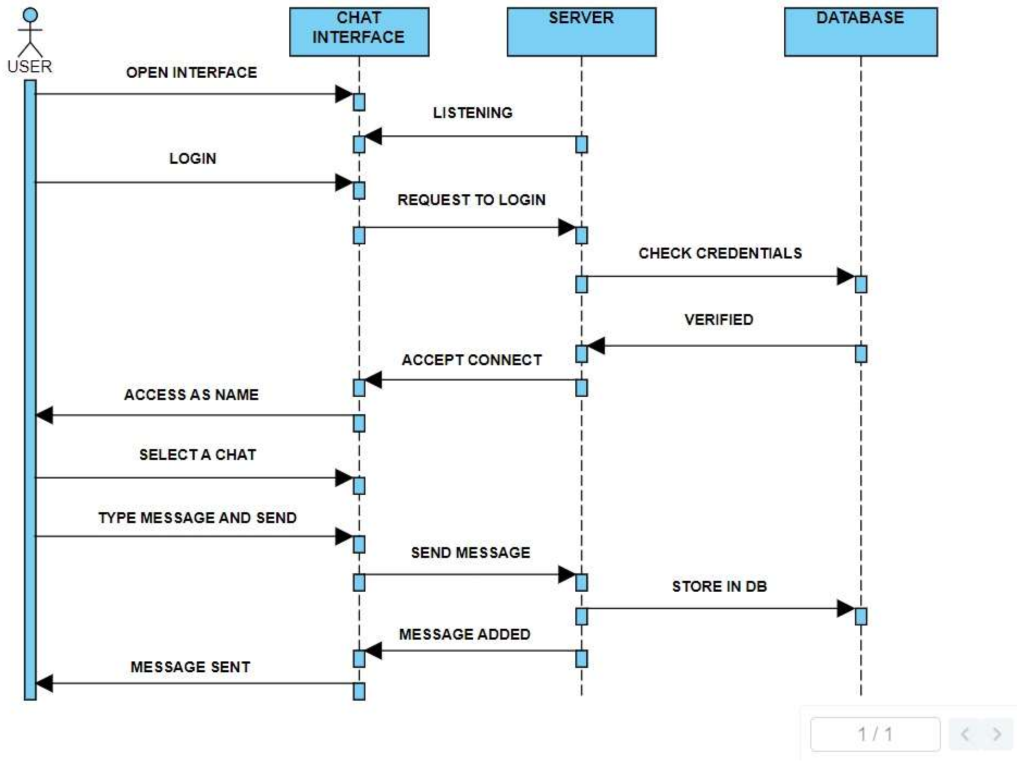


Figure3: Sequence Diagram

3.4.3 Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

It analyses and designs a static view of an application.

It describes the major responsibilities of a system.

It is a base for component and deployment diagrams.

It incorporates forward and reverse engineering.

Benefits of Class Diagrams

It can represent the object model for complex systems.

It reduces the maintenance time by providing an overview of how an application is structured before coding.

It provides a general schematic of an application for better understanding.

It represents a detailed chart by highlighting the desired code, which is to be programmed.

It is helpful for the stakeholders and the developers.

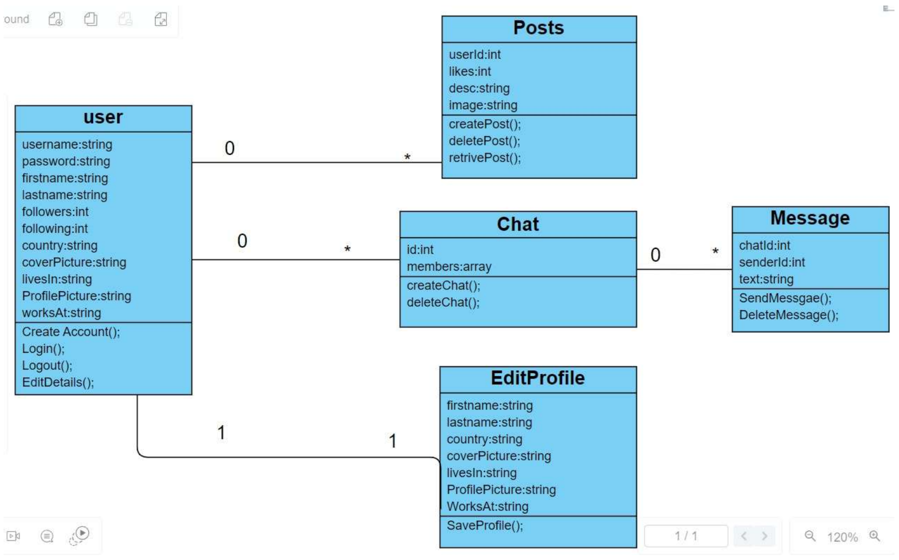


Figure 4: Use-Case Diagram

3.4.4 Component Diagram

UML Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.

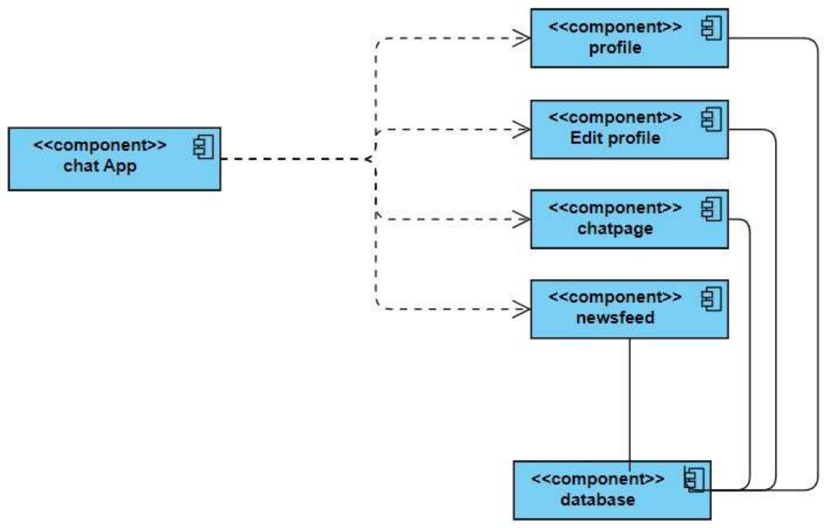


Figure 5: Component Diagram

Deployment Diagram

In the context of the Unified Modeling Language (UML), a deployment diagram falls under the structural diagramming family because it describes an aspect of the system itself. In this case, the deployment diagram describes the physical deployment of information generated by the software program on hardware components. The information that the software generates is called an artifact. This shouldn't be confused with the use of the term in other modeling approaches like BPMN.

Deployment diagrams are made up of several UML shapes. The three-dimensional boxes, known as nodes, represent the basic software or hardware elements, or nodes, in the system. Lines from node to node indicate relationships, and the smaller shapes contained within the boxes represent the software artifacts that are deployed.

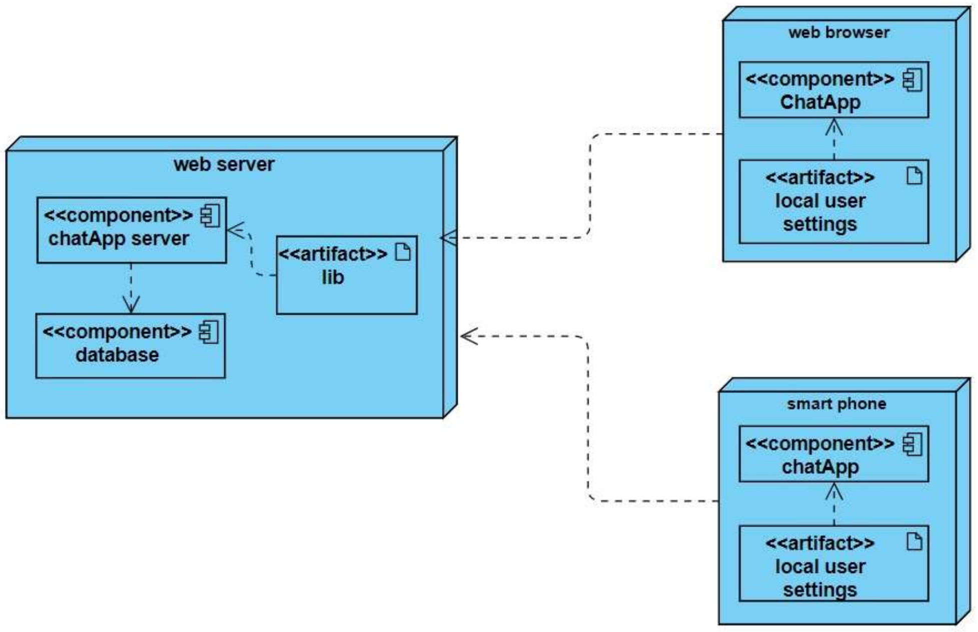


Figure 6: Deployment Diagram

3.5 System Design:

3.5.1 System Configuration:

Hardware Requirements:

Processor	: Any updated processor
RAM	: Minimum 4GB
Hard Disk	: Minimum 100GB

Software Requirements:

Operating System	: Windows Family
Technology	: Javascript
Front-End	: Node.js
Back-End	: Express.js, Socket.io, MongoDB
IDE	: VSCode

4. IMPLEMENTATION

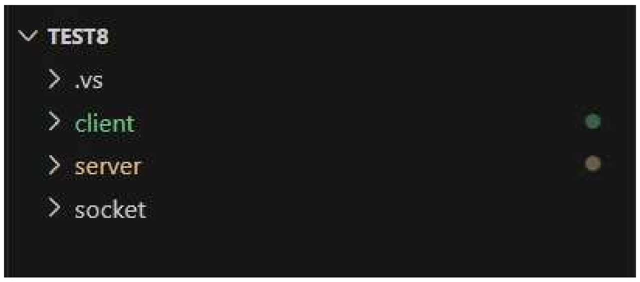
4.1 Program Files:

We have 3 folders for the project:

1.Client -- which is frontend. frontend is made with react Js to run client we navigate to client folder and type yarn start to start React APP

2.Server -- which is backend to run backend server navigate to server folder and type npm start to start backend server it consists of node Js Express Js

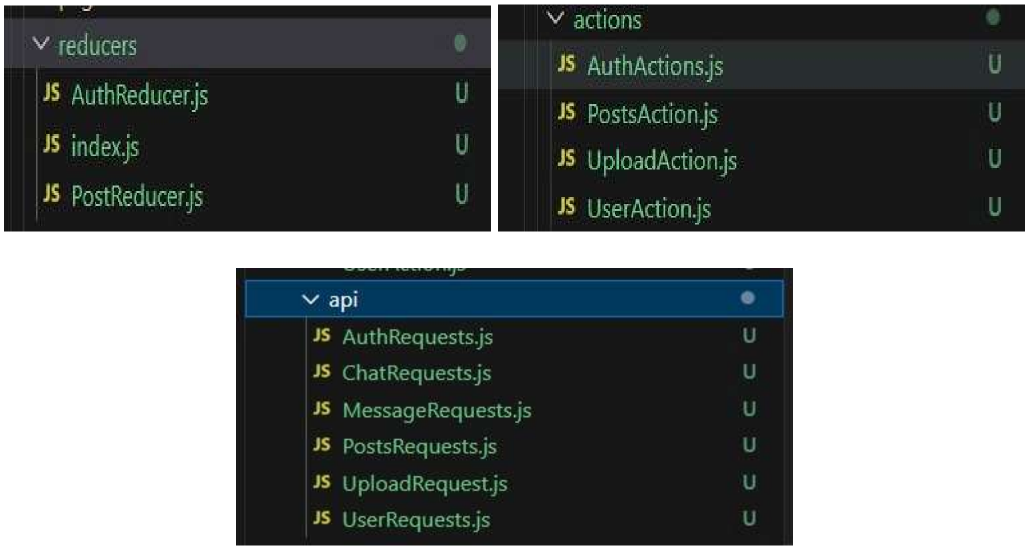
3.Socket -- for real time instant massage to run it we navigate to socket folder and type npm start



```
∨ TEST8
  > .vs
  > client
  > server
  > socket
```

Figure 7: Program Folders

4.1.1 Client Folder

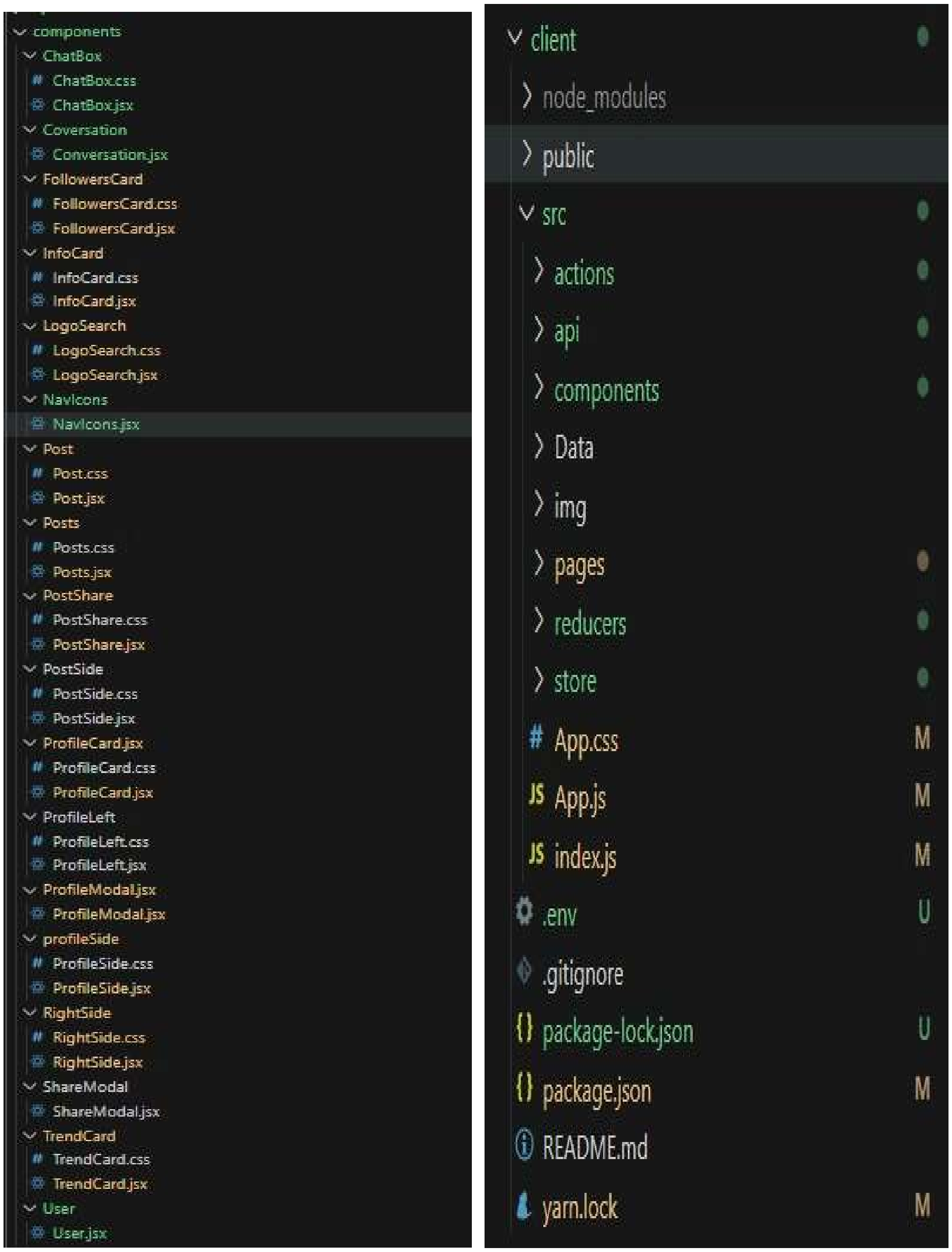


```
∨ reducers
  JS AuthReducer.js     U
  JS index.js           U
  JS PostReducer.js     U
```

```
∨ actions
  JS AuthActions.js     U
  JS PostsAction.js     U
  JS UploadAction.js    U
  JS UserAction.js      U
```

```
∨ api
  JS AuthRequests.js    U
  JS ChatRequests.js    U
  JS MessageRequests.js U
  JS PostsRequests.js   U
  JS UploadRequest.js   U
  JS UserRequests.js    U
```

Figure 8: Files in Client Folder



```
components
  ChatBox
    ChatBox.css
    ChatBox.jsx
  Coversation
    Conversation.jsx
  FollowersCard
    FollowersCard.css
    FollowersCard.jsx
  InfoCard
    InfoCard.css
    InfoCard.jsx
  LogoSearch
    LogoSearch.css
    LogoSearch.jsx
  NavIcons
    NavIcons.jsx
  Post
    Post.css
    Post.jsx
  Posts
    Posts.css
    Posts.jsx
  PostShare
    PostShare.css
    PostShare.jsx
  PostSide
    PostSide.css
    PostSide.jsx
  ProfileCard.jsx
    ProfileCard.css
    ProfileCard.jsx
  ProfileLeft
    ProfileLeft.css
    ProfileLeft.jsx
  ProfileModal.jsx
    ProfileModal.jsx
  profileSide
    ProfileSide.css
    ProfileSide.jsx
  RightSide
    RightSide.css
    RightSide.jsx
  ShareModal
    ShareModal.jsx
  TrendCard
    TrendCard.css
    TrendCard.jsx
  User
    User.jsx
```

```
client
  node_modules
  public
  src
    actions
    api
    components
    Data
    img
    pages
    reducers
    store
    App.css              M
    App.js               M
    index.js             M
  .env                   U
  .gitignore
  package-lock.json      U
  package.json           M
  README.md
  yarn.lock              M
```

Figure9: Files in Client Folder

public(folder): Root folder that gets served up as our react app.

favicon.ico(file): It’s an icon file that is used in index.html as favicon.

index.html(file): It is the template file which is served up when we run start script to launch our app. It is considered best practice not to create multiple html file in public folder instead use this file and inject react components in this file's root div container. Other css libraries, etc can be defined in this files.

manifest.json(file): It's used to define our app, mostly contains metadata. Used by mobile phones to add web app to the home-screen of a device.

src(folder): In simplest form it's our react app folder i.e. containing components, tests, css files etc. It's the mind of our app.

App.css(file): Contains styles of our react component(App.js)

App.js(file): This file has very basic react component defined which can be replaced by our own root component.

index.js(file): This files renders our component and registers service workers(unregistered by default)

Components (Folder): Components are the building blocks of any react project. This folder consists of a collection of UI components like buttons, modals, inputs, loader, etc., that can be used across various files in the project. Each component should consist of a test file to do a unit test as it will be widely used in the project.

Pages (Folder): The files in the pages folder indicate the route of the react application. Each file in this folder contains its route. A page can contain its sub folder. Each page has its state and is usually used to call an sync operation. It usually consists of various components grouped.

4.1.2 Server Folder

Controllers: Controllers folder contains the business logic and validations for the input data received by the client side and performs their business logic and sends it to database controllers, which contains logical files and folders.

Models: Models contain all the schemas of the database, like which kind of input will be received from client-side and server-side validations. This will contain all files of validations and data schemas that should exist.

Routes: This will contains all the routes and endpoints required for the server and also thier required files like for authentication routes we can create a file and setup routes.

env: The env file stands for Environmental Variables of Application. This file contains details about environment variables like API keys, the private salt of a Payment Gateway, and many other details that should be kept private.

gitignore: Git ignore is a file that contains files and folders that should not be pushed to the server. The git ignore file is used to stop pushing files from the server, like node module files, which should not be pushed to the server because they can easily be installed with the package.json file.

AuthController.js : Itprovides functionalities for user authentication: registration and login.

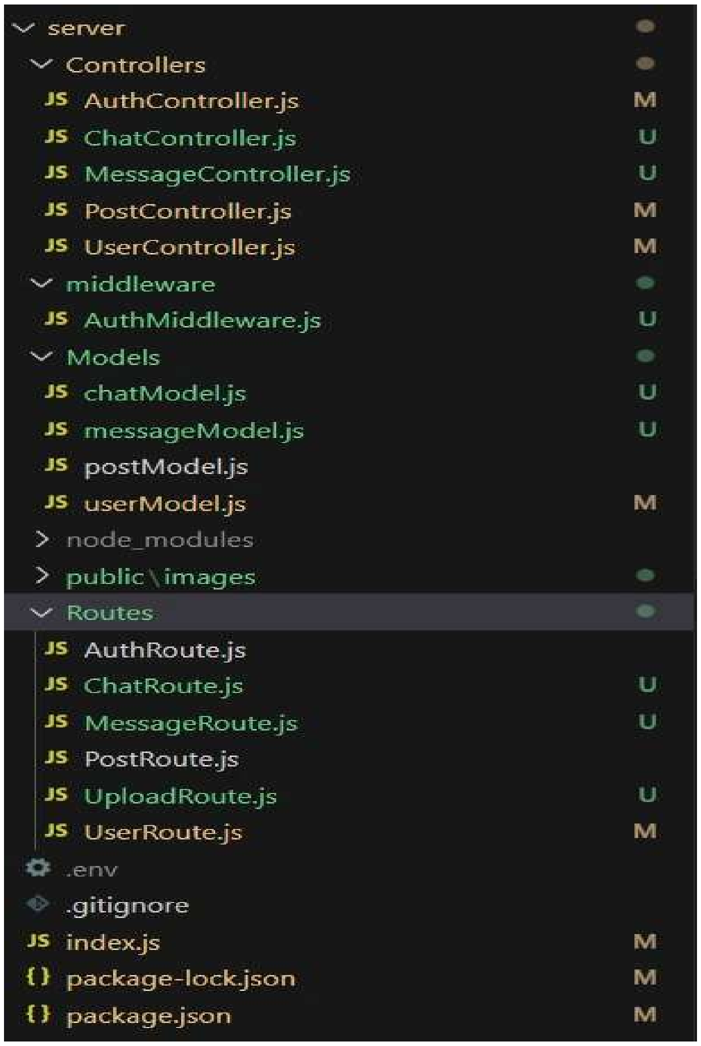
registerUser: This function handles the registration of a new user. It first generates a salt using bcrypt, then hashes the user's password with the generated salt. It checks if a user with the same username already exists in the database. If not, it saves the new user to the database along with the hashed password. Finally, it generates a JWT token containing the user's username and ID, signs it with the JWT secret key, and sends it along with the user information in the response.

loginUser: This function handles the login of a user. It takes the username and password from the request body and tries to find a user with the provided username in the database. If the user exists, it compares the provided password with the hashed password stored in the database using bcrypt. If the passwords match, it generates a JWT token containing the user's username and ID, signs it with the JWT secret key, and sends it along with the user information in the response. If the passwords don't match, it returns a "wrong password" error message. If the user is not found in the database, it returns a "User not found" error message.

ChatController.js: It provides functionalities related to creating and retrieving chats between users.

createChat: This function creates a new chat between two users. It expects the IDs of the two users who will participate in the chat as parameters (firstId and secondId). It creates a new instance of the ChatModel with the provided member IDs and saves it to the database. Upon successful creation, it returns the newly created chat object in the response.

ChatController.js: It provides functionalities related to creating and retrieving chats between users.



```
server                                  ●
  Controllers                           ●
    AuthController.js                   M
    ChatController.js                   U
    MessageController.js                U
    PostController.js                   M
    UserController.js                   M
  middleware                            ●
    AuthMiddleware.js                   U
  Models                                ●
    chatModel.js                        U
    messageModel.js                     U
    postModel.js
    userModel.js                        M
  > node_modules
  > public\images                       ●
  Routes                                ●
    AuthRoute.js
    ChatRoute.js                        U
    MessageRoute.js                     U
    PostRoute.js
    UploadRoute.js                      U
    UserRoute.js                        M
  .env
  .gitignore
  index.js                              M
  package-lock.json                     M
  package.json                          M
```

Figure 10: Files in Server Folder

createChat: This function creates a new chat between two users. It expects the IDs of the two users who will participate in the chat as parameters (firstId and secondId). It creates a new instance of the ChatModel with the provided member IDs and saves it to the database. Upon successful creation, it returns the newly created chat object in the response.

userChats: This function retrieves chats that involve a specific user. It queries the ChatModel to find chats where the provided user ID (userId) is present in the members array. It returns an array of chats involving the user in the response.

findChat: This function finds a specific chat between two users. It expects the IDs of the two users involved in the chat (firstId and secondId) as parameters. It queries the ChatModel to find a chat where both user IDs are present in the members array. If such a chat exists, it returns it in the response. Otherwise, it returns null.

MessageController.js provides functionalities for managing messages within a chat.

addMessage: This function adds a new message to a chat. It expects the chat ID (chatId), sender ID (senderId), and message text (text) to be included in the request body. It creates a new instance of the MessageModel with the provided data and saves it to the database. Upon successful saving, it returns the newly created message object in the response.

getMessages: This function retrieves all messages belonging to a specific chat. It expects the chat ID (chatId) to be included in the request parameters. It queries the MessageModel to find all messages where the provided chat ID matches. It returns an array of messages belonging to the chat in the response.

PostController.js provides functionalities for managing posts in the application.

createPost: This function creates a new post. It expects post data to be included in the request body. It creates a new instance of the PostModel with the provided data and saves it to the database. Upon successful saving, it returns the newly created post object in the response.

getPost: This function retrieves a specific post by its ID. It expects the post ID to be included in the request parameters. It queries the PostModel to find the post with the provided ID and returns it in the response.

updatePost: This function updates an existing post. It expects the post ID in the request parameters and the user ID in the request body. It checks if the user owns the post and updates it accordingly. It returns "Post Updated" if successful, otherwise "Action forbidden" if the user is not authorized.

deletePost: This function deletes a post. It expects the post ID in the request parameters and the user ID in the request body. It checks if the user owns the post and deletes it accordingly. It returns "Post deleted successfully" if successful, otherwise "Action forbidden" if the user is not authorized.

likePost: This function allows users to like or dislike a post. It expects the post ID in the request parameters and the user ID in the request body. If the user hasn't liked the post, it adds their ID to the likes array; otherwise, it removes it. It returns "Post liked" or "Post Unliked" depending on the action.

getTimelinePosts: This function retrieves posts for a user's timeline. It expects the user ID in the request parameters. It fetches posts created by the user and posts from users they follow. It returns a sorted array of timeline posts based on the creation time.

UserController.js: It provides functionalities for managing user-related operations in the application

getAllUsers: Retrieves all users from the database. It hides the password field from each user document and returns only the necessary details.

getUser: Retrieves a specific user by their ID. It hides the password field from the user document and returns only the necessary details. If the user doesn't exist, it returns a "No such user exists" error message.

updateUser: Updates a user's profile. It checks if the user is updating their own profile by comparing the provided user ID with the ID in the request parameters. If the user is authorized, it updates the user's details, including hashing the password if provided, and returns the updated user along with a new JWT token for authentication.

deleteUser: Deletes a user's profile. It checks if the user is either deleting their own profile or an administrator is performing the action. If authorized, it deletes the user's profile from the database.

followUser: Allows a user to follow another user. It checks if the user is attempting to follow themselves and if they are already following the target user. If not, it updates both the follower's and the target user's documents to reflect the follow relationship.

UnFollowUser: Allows a user to unfollow another user. It checks if the user is attempting to unfollow themselves and if they are already following the target user. If so, it updates both the follower's and the target user's documents to remove the follow relationship.

This AuthMiddleware.js: This file implements an authentication middleware using JWT (JSON Web Token).

Import Dependencies: It imports the necessary packages, jsonwebtoken for handling JWTs and dotenv for loading environment variables.

Load JWT Secret: It loads the JWT secret key from the environment variables using dotenv.

Middleware Function: It defines an asynchronous middleware function named authMiddleWare. This function takes three parameters: req, res, and next.

Try-Catch Block: It wraps the middleware logic in a try-catch block to catch any errors that may occur during token verification.

Token Verification: It attempts to extract the JWT token from the Authorization header in the request. It splits the header value by spaces and retrieves the token from the resulting array.

Token Decoding: If a token is found, it verifies and decodes the token using the JWT secret key. The decoded payload typically contains information like the user ID (id).

Attach User ID to Request: If the token is successfully decoded, it attaches the user ID (id) from the decoded payload to the req.body._id. This allows subsequent middleware functions or route handlers to access the user ID.

Call Next Middleware or Route Handler: Regardless of whether the token is decoded successfully or not, it calls the next() function to pass control to the next middleware function or route handler in the request processing pipeline.

Error Handling: If any error occurs during token verification or decoding, it catches the error and logs it to the console. However, it doesn't send any response back to the client, leaving the handling of errors to subsequent error-handling middleware or the global error handler.

chatModel.js: defines a Mongoose schema for a chat entity and creates a corresponding model .It provides a basic structure for representing chat entities in a MongoDB database using Mongoose, including fields for storing member IDs and timestamps for tracking when chats are created or updated.

messageModel.js: This file provides a structure for representing message entities in a MongoDB database using Mongoose, including fields for storing chat IDs, sender IDs, message text.

postModel.js: It provides a structure for representing post entities in a MongoDB database using Mongoose, including fields for storing user IDs, post descriptions, likes, images.

userModel.js: It provides a structure for representing user entities in a MongoDB database using Mongoose, including various fields to store user information and relationships.

AuthRoute.js: It sets up routes for user registration and login, directing incoming requests to the appropriate controller functions for processing.

ChatRoute.js: It sets up routes for creating chats, finding chats, and retrieving user chats.

MessageRoute.js: It sets up routes for adding messages to a chat and retrieving messages for a chat.

PostRoute.js: It sets up routes for various post-related operations.

UploadRoute.js: It enables the server to handle file uploads by configuring multer middleware and defining a route to receive and process uploaded files.

UserRoute.js: It sets up routes for various user-related operations, ensuring authentication using the middleware before accessing routes that require authorization.

Index.js: This file sets up a backend server using Express.js, with various middleware for handling requests.

Overall, this code sets up a comprehensive backend server for handling various functionalities of a web application, including user authentication, user management, posting, file uploading, chatting, and messaging, with appropriate middleware and database connectivity.

4.1.3 Socket folder:

node_modules folder: It is used to store the dependencies (external libraries or packages) that your Node. js project relies on.

Index.js file : This file sets up a Socket.IO server that listens on port 8800. It allows clients to connect and communicate in real-time.

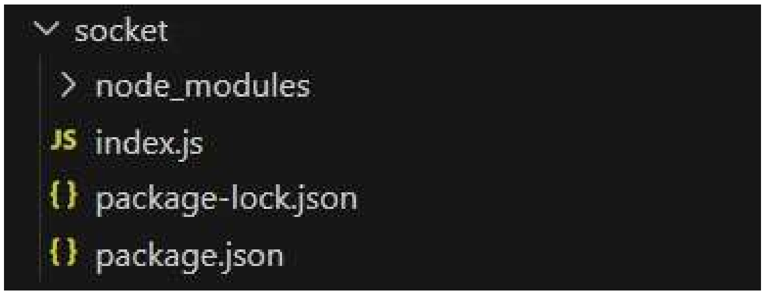


Figure 11: Files in Socket Folder

The Socket.IO server is created with CORS configured to allow requests from "http://localhost:3000".

It maintains a list of active users (activeUsers array) consisting of their user IDs and corresponding socket IDs.

When a client connects (connection event), it adds the user to the activeUsers array and emits the list of active users to all clients.

When a client disconnects (disconnect event), it removes the user from the activeUsers array and emits the updated list of active users to all clients.

It allows clients to send messages to specific users (send-message event). It finds the recipient's socket ID based on their user ID and emits the message to that socket.

It emits various events like get-users to send the list of active users and recieve-message to deliver messages to clients.

Overall, this code creates a basic real-time chat application server where users can connect, disconnect, and exchange messages with each other.

The JSON files that contains metadata about a Node.js project, such as its name, version, and dependencies.

4.2 Datasets:

The proposed chat application does not rely on external datasets as it primarily interacts with real-time data captured from the user interaction. Messaging, post chat data and user data are stored directly as and when the user uses the application.

Here is database table data structure for each table user and chatroom.

MESSAGE DATA STRUCTUER

Table 2

S.No	Field Name	Data Type	Explanation
1	Chat Id	String	Id for Chat Room
2	Sender Id	String	Id of the Sender
3	Text	String	Messages

POST DATA STRUCTURE

Table 3

S.No	Field Name	Data Type	Explanation
1	userid	String	User ID
2	Desc	String	About the Post
3	Likes	Int	No of people liked the most
4	Image	String	Post Image

USER DATA STRUCTURE

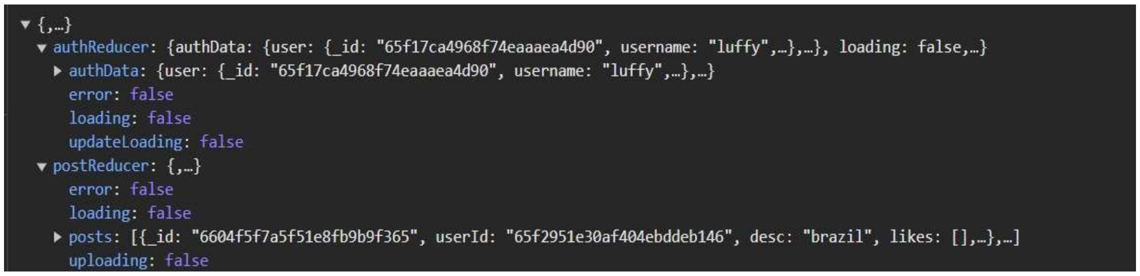
Table 4

S.No	Field Name	Data Type	Explanation
1	username	String	Username
2	password	String	Password
3	first name	String	First name
4	last name	String	Lastname
5	profile picture	String	Profile picture of user
6	cover picture	String	Cover picture of user
7	about	String	Description of user
8	Lives in	String	Place the user lives in
9	Works At	String	Company/firm user works at
10	followers	Int	People user follows
11	following	Int	People followed by the user

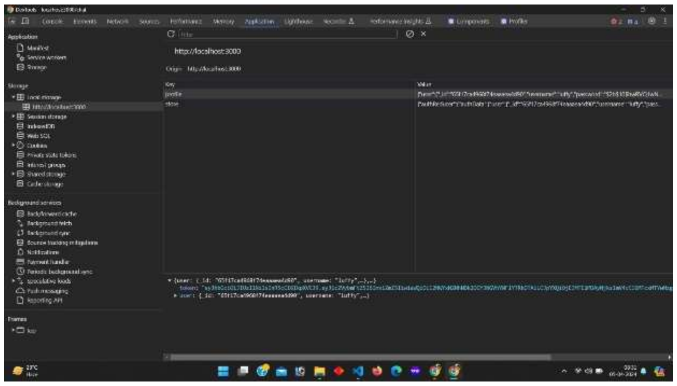
4.3 LocalStorage:

We use local storage in order to store the login credentials of the user.

LocalStorage is a browser-based storage mechanism that allows you to store key-value pairs locally within the user's browser. This storage persists even after the browser is closed and reopened, unlike session storage which persists only until the browser is closed. Used to store user preferences, authentication tokens, or any other data that needs to persist across sessions.



```
▼ {,…}
  ▼ authReducer: {authData: {user: {_id: "65f17ca4968f74eaaaea4d90", username: "luffy",…},…}, loading: false,…}
    ▶ authData: {user: {_id: "65f17ca4968f74eaaaea4d90", username: "luffy",…},…}
      error: false
      loading: false
      updateLoading: false
  ▼ postReducer: {,…}
      error: false
      loading: false
    ▶ posts: [{_id: "6604f5f7a5f51e8fb9b9f365", userId: "65f2951e30af404ebddeb146", desc: "brazil", likes: [],…},…]
      uploading: false
```



```
http://localhost:3000
Origin  http://localhost:3000
Key       Value
profile   [illegible]
store     [illegible]
```

Figure 12: Local Storage

4.4 Redux:

Redux is a predictable state container for JavaScript applications, including React. A reducer in Redux is a pure function that takes the previous state and an action and returns the next state. We are using redux for authentication and actions are AUTH_START AUTH_SUCCESS AUTH_FAIL and uploading posts actions are UPLOAD_START UPLOAD_SUCCESS UPLOAD_FAIL and retrieving posts actions are RETREIVING_START RETREIVING_SUCCESS RETREIVING_FAIL.

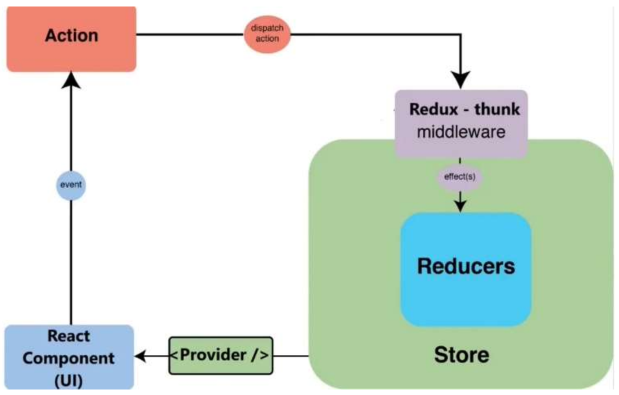


Figure 13: Redux

5. EXPERIMENT ENVIRONMENT

5.1 Experimental Setup

Text Editors/IDEs:

- **Visual Studio:**

Visual Studio is an integrated development environment (IDE) created by Microsoft. It provides a comprehensive set of tools and services for software development, making it easier for developers to create, debug, test, and deploy applications. Visual Studio supports a wide range of programming languages, including C++, C#, Visual Basic, F#, Python, and more.

- **Postman API:**

Postman is an API development environment that allows developers to design, test, and document APIs. It's a popular tool among developers for its user-friendly interface and robust features. Postman offers both a desktop application and a web version, enabling collaboration among team members.

Here's a brief overview of some key features and functionalities of the Postman API:

API Testing: Postman allows you to create and execute automated tests for your APIs. You can define test scripts using JavaScript, which can validate responses, check status codes, and more.

API Documentation: With Postman, you can generate interactive API documentation automatically from your API requests, making it easier for developers to understand and use your APIs.

Mock Servers: Postman allows you to create mock servers for your APIs, enabling you to simulate API responses without needing a backend server. This is useful for testing or for providing a sandbox environment for developers.

API Monitoring: Postman offers monitoring capabilities to keep track of the performance and uptime of your APIs. You can set up monitors to periodically send requests to your APIs and receive alerts if there are any issues.

Collaboration: Postman provides features for team collaboration, allowing team members to share collections, collaborate on API development, and comment on API requests.

Environment Variables: Postman lets you define environment variables, which can be used to parameterize your requests and make them more flexible. This is useful for working with different environments such as development, staging, and production.

Collections: Postman allows you to organize your API requests into collections, making it easy to manage and share them. Collections can be exported and imported, allowing for easy sharing among team members or with the broader community.

Overall, Postman simplifies the process of API development and testing, making it an essential tool for developers working with APIs.

5.2 Technology Used

Node.js: Node.js is an open source, cross-platform runtime environment and library that is used for running web applications outside the client's browser.
It is used for server-side programming, and primarily deployed for non-blocking, event-driven servers, such as traditional web sites and back-end API services, but was originally designed with real-time, push-based architectures in mind. Every browser has its own version of a JS engine, and node.js is built on Google Chrome's V8 JavaScript engine.

In simple terms, what this means is that entire sites can be run using a unified 'stack', which makes development and maintenance quick and easy, allowing you to focus on meeting the business goals of the project.

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

1. Simplicity and Minimalism: Express JS has a very simple design, that makes it easy to learn and use. With its simple structure you can quickly set up a server, define routes, and handle HTTP requests which makes it an excellent choice for building web applications efficiently.
2. Flexibility and Customization**:** Express JS is a flexible framework that allows you to structure the application based on your preferences. It does have a strict application architecture so you can organize your code according to your preference.

3. Middleware Ecosystem**:** Express JS has a large numbers of middleware that can be easily integrated into applications. Middleware functions increases the functionality of Express by allowing you to handle various tasks such as authentication, logging, and error handling.
4. Scalability: Express JS is designed to be lightweight and scalable, which makes it suitable for building both small projects and large-scale applications. It is asynchronous and has event-driven architecture which allows you to handle a large number of requests.
5. Active Community Support: Express JS has a large active community who contribute to its growth and improvement. Because of them the framework is regularly updated and well-documented.

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

MongoDB: MongoDB is an open source NOSQL database management program. NoSQL (Not only SQL) is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.

MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.

Structured Query Language (SQL) is a standardized programming language that is used to manage relational databases. SQL normalizes data as schemas and tables, and every table has a fixed structure.

Instead of using tables and rows as in relational databases, as a NoSQL database, the MongoDB architecture is made up of collections and documents. Documents are made up of Key-value pairs -- MongoDB's basic unit of data. Collections, the equivalent of SQL tables, contain document sets. MongoDB offers support for many programming languages, such as C, C++, C#, Go, Java, Python, Ruby and Swift.

Axios: Axios, which is a popular library is mainly used to send asynchronous HTTP requests to REST endpoints. This library is very useful to perform CRUD operations.

1. This popular library is used to communicate with the backend. Axios supports the Promise API, native to JS ES6.
2. Using Axios we make API requests in our application. Once the request is made we get the data in Return, and then we use this data in our project.
3. This library is very popular among developers.

Axios allows you to communicate with the APIs in your React project. The same tasks can also be performed by using AJAX, but Axios provide you more functionality and features and that helps you in building your application quickly. Axios is a promise-based library, so you need to implement some promise-based asynchronous HTTP requests. jQuery and AJAX also perform the same job but in React project React handles each and everything in its own virtual DOM, so there is no need to use jQuery at all.

6. RESULTS

6.1 User Interface:

The web-based chat application has many features but consist of three main features in the user interface that are Login Page, Public Page and Profile accessibility page.

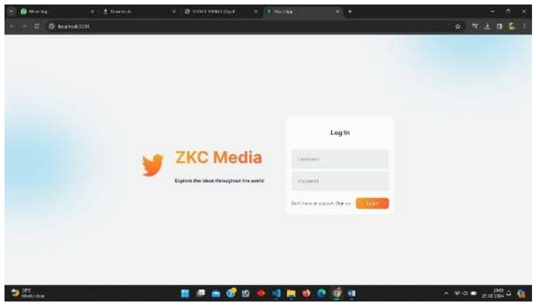


Figure 3:Login Page

Here, Figure 14 illustrates the login page where the user can login with their credentials or can register and login if he/she is a new user. At the stage the system will generate a web-token to maintain a time for the usage.

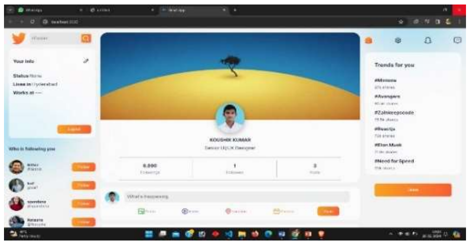


Figure 4:Profile Page

The image (Figure 15) is the Profile Page, where the user can view their profile details and edit to make changes when required.

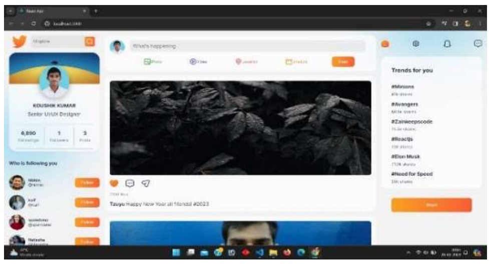


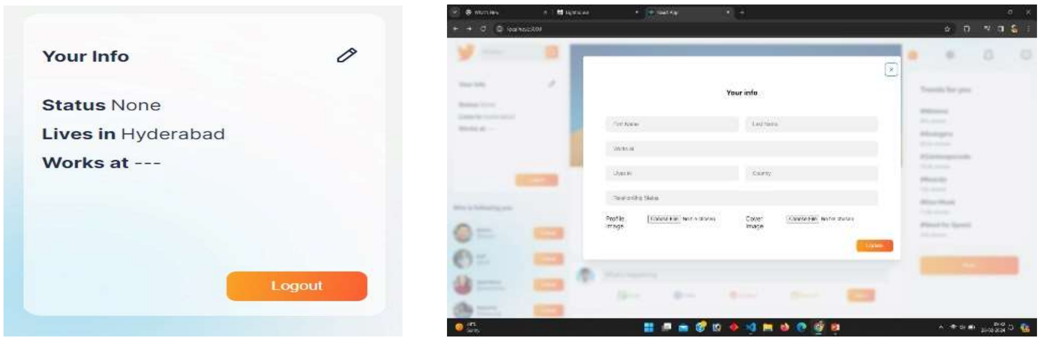
Figure 5: Home Page

The above image is the main page or the home page which caters all the requirements go-to. In the home page, the user can make a post, follow people, start a conversation (chat room) and see the latest trends

6.2 User Interaction Functions and Testing:

User interaction testing is conducted to evaluate the responsiveness of the tools to different input methods. Edge cases, involving extreme conditions, are considered to assess the robustness of the drawing tools. Testing is performed for real-time drawing scenarios, and debugging tools are utilized to identify any underlying issues. User feedback is collected to gain insights into the overall user experience. The testing process is iterative, with continuous improvement based on identified issues, ensuring that the drawing tools meet high standards of precision and functionality. After all the continuous improvements and testing we could see that the drawing tools, such as line, circle, and rectangle are working accurately.

6.2.1 Info Card:


Your Info

Status None
Lives in Hyderabad
Works at ---

Figure 6: Info Card

The above image (Figure 17) is the info card which contains general information about the user and this information is visible to others who interact with the user and image (right side) is the info card editing page which contains general information can be edited by the user.

6.2.2 Edit Card (for posts):

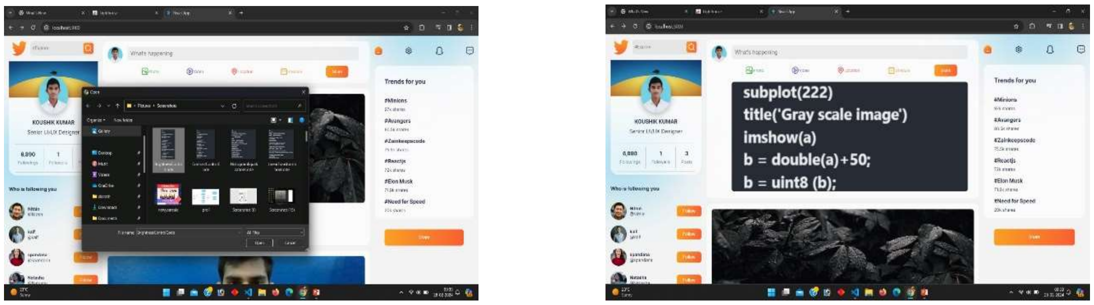

```
subplot(222)
title('Gray scale image')
imshow(a)
b = double(a)+50;
b = uint8 (b);
```

Figure 7: Editing and Sharing Posts

The above figure (Fig 18) is where the user can make a post after selecting an image, or by writing some text and the residing image is the interface after sharing a post.

6.2.3 Chatting Interface:

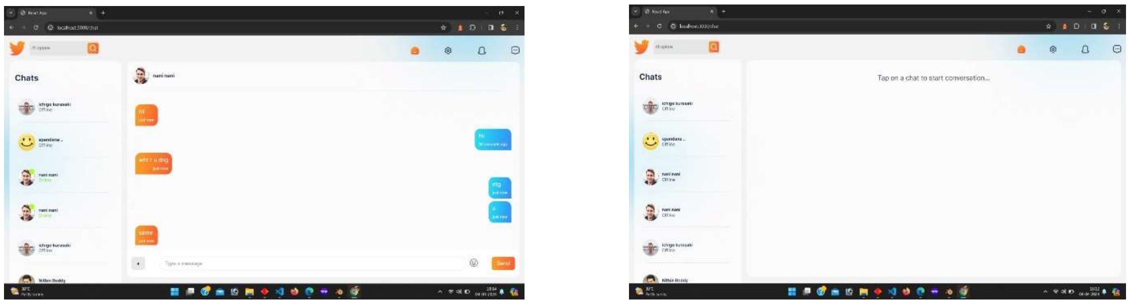


Figure 8:Chatting Interface

The image above (Figure 19) is the illustration of Chatting Page where we can start a conversation. Here, we can chat with multiple individuals and check if the other user is online or offline in using the application. The chat interface supports chats of the people you follow and the people who follow you.

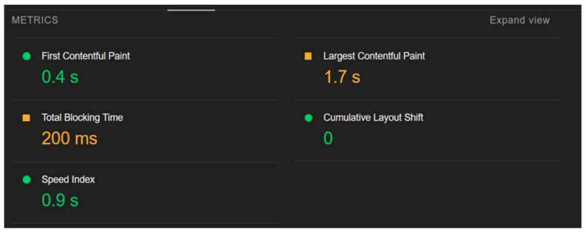
6.3 Test Results:

As test result, the main features of these chat application send and receives the messages very well and can help us make posts and connect to different people. Chat application built with Node.js, MongoDB and Socket.io running much faster in order to achieve real-time chat applications. Test result of the chat application shows the same results when tested on localhost, in the network or when in hosting.

For the model authentication, we have refered a general simple chat application using PHP and have compared the parameters which proves the accuracy and efficiency of the system[5][1]

Chat Application using MERN Stack

(Proposed Model)


METRICS Expand view

● First Contentful Paint
0.4 s

■ Largest Contentful Paint
1.7 s

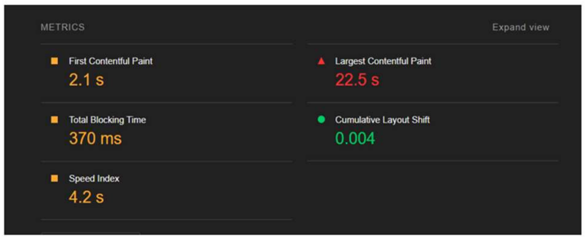
■ Total Blocking Time
200 ms

● Cumulative Layout Shift
0

● Speed Index
0.9 s

Chat Application using PHP

(Reference Model)


METRICS Expand view

■ First Contentful Paint
2.1 s

▲ Largest Contentful Paint
22.5 s

■ Total Blocking Time
370 ms

● Cumulative Layout Shift
0.004

■ Speed Index
4.2 s

Figure 20: Test Result Comparison

The following table shows the results of the tests conducted:

Table 5

S.No	Metrics	Chat Application using PHP	Chat Application using MERN Stack-Proposed	Metric Description
1	First Contentful Paint	2.1 s	0.4 s	First Contentful Paint marks the time at which the first text or image is painted.
2	Largest Contentful Paint	22.5 s	1.7 s	Largest Contentful Paint marks the time at which the largest text or image is painted.

3	Total Blocking Time	370 ms	200 ms	Sum of all time periods between FCP and Time to Interactive, when task length exceeded 50ms, expressed in milliseconds.
4	Cumulative Layout Shift	0.004	0	Cumulative Layout Shift measures the movement of visible elements within the viewport.
5	Speed Index	4.2 s	0.9 s	Speed index shows how quickly the contents of a page are visibly populated

7. CONCLUSION, FINDINGS AND FUTURE SCOPE

7.1 Conclusion:

Development of a chat application using the MERN (MongoDB, Express.js, React.js, Node.js) stack offers a robust and efficient solution for real-time communication needs. Throughout this paper, we have explored the key components and advantages of the MERN stack in building such applications.

Firstly, MongoDB provides a flexible and scalable database solution, allowing for seamless storage and retrieval of chat data. Its NoSQL nature accommodates dynamic data structures, making it ideal for handling the diverse nature of chat messages.

Secondly, Express.js serves as a powerful backend framework, simplifying the development of RESTful APIs for handling user authentication, message storage, and retrieval. Its lightweight nature and extensive middleware support streamline the development process.

Thirdly, React.js offers a modern and component-based frontend framework, enabling the creation of dynamic and interactive user interfaces. Its virtual DOM ensures efficient rendering, while its component reusability fosters code maintainability and scalability.

Lastly, Node.js acts as the runtime environment for executing JavaScript code on the server-side, facilitating seamless communication between the frontend and backend components. Its event-driven architecture and non-blocking I/O operations make it well-suited for handling concurrent connections in real-time chat applications.

By leveraging the MERN stack, developers can create feature-rich chat applications with responsive interfaces, real-time messaging capabilities, and scalable architectures. However, it's essential to consider factors such as security, performance optimization, and scalability when building production-ready applications.

On the contrary, the application created using PHP, this is a Messaging Web Application in PHP and MySQL Database. This web-based messaging or chat platform where users can communicate with each other online. This project has multiple features that automatically load messages or chat updates. With the help of JavaScript, Ajax, and jQuery, users can have a better experience while using the application.

In summary, the MERN stack provides a comprehensive solution for developing modern chat applications, empowering developers to create engaging and efficient communication platforms tailored to the needs of users than the other applications.

7.1 Findings:

<u>Chat Application using PHP Performance Index</u>

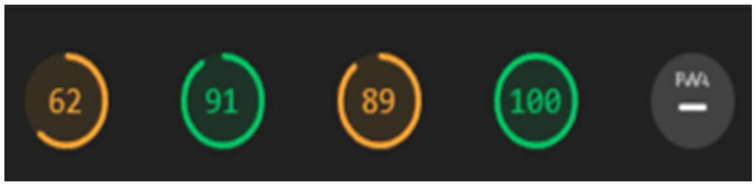


Figure 21: PI: Chat Application using PHP

<u>Chat Application using MERN stack Performance Index</u>

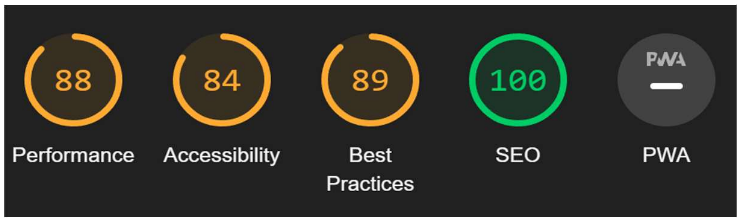


Figure 9: PI : Proposed Chat Application using MERN Stack

Based on the results of tests performed, it can produce a conclusion that:

1. The overall response time of the system is 4 times better than the application created using PHP.
2. Though the performance indexes like SEO and Best Practices are the same and the accessibility of the general application is better than that of proposed, the overall performance and efficiency is higher by 26% which proves that the proposed model is accurate and efficient than other models.

7.3 Future Scope:

Looking ahead, the future scope for chat applications like our application is brimming with opportunities for innovation and expansion. One promising avenue is the integration of artificial intelligence and machine learning algorithms to enhance user experience. By leveraging AI-powered chatbots, these applications can offer personalized assistance, automate routine tasks, and provide relevant recommendations based on user preferences and past interactions. This could revolutionize how users engage with the platform, making interactions more efficient and enjoyable.

Furthermore, the integration of augmented reality (AR) and virtual reality (VR) technologies holds immense potential for transforming the chat experience. Imagine being able to share immersive AR experiences or participate in VR meetings directly within the chat application. These advancements could redefine communication by adding a new dimension of interactivity and collaboration, bridging the gap between virtual and physical interactions.

Another area ripe for exploration is the expansion of multi-platform capabilities. As users increasingly rely on a variety of devices for communication, from smartphones and tablets to smartwatches and smart speakers, chat applications must adapt to seamlessly synchronize conversations across all these platforms. By offering a cohesive and synchronized experience regardless of the device being used, chat applications can enhance convenience and accessibility for users[7].

Moreover, the evolution of chat applications may extend beyond traditional text-based communication to include richer media formats and integrations.[1] Features such as high-definition video calling, interactive multimedia messaging, and seamless integration with third-party services could further enrich the user experience, making chat applications not just a means of communication, but also a central hub for entertainment, productivity, and social interaction.

In summary, the future of chat applications is characterized by a convergence of cutting-edge technologies, enhanced user experiences, and expanded functionalities. By embracing innovation and staying attuned to evolving user needs, these platforms can continue to play a central role in how people connect and communicate in an increasingly digital world.

8. REFERENCES

[1]Croucher, T. H., & Wilson, M. (2012). Node: Up and Running. United States.

[2]Sidik, B. (2011). JavaScipt. Bandung: Informatika.

[3]Kiessling, Manuel. 2012. The Node Beginner Book. lulu.com, United Stated.

[4] Mardan, Azat. 2012. Practical Node.js: Building Real-World Scalable Web Apps. Appress.

[5] Diotra Henriyan. 2016 Design and Implementation of Web Based Real Time Chat Interfacing Server, Bandung: Informatika

[6] Teixeira, Pedro. 2012. Hands-on Node.js. Wrox.

[7] Abhishek Bedare 2023, Lifeline Messenger Real-Time Chat Application: Using Mern Stack

[8] Real Time Chat Application, which is submitted by Eric Obadjere Nyerhovwo 2020 [8]