CHAT APPLICATION

A PROJECT REPORT
for
Major Project (KCA451)
Session (2023-24)

Submitted by

Riya Khurrana
(University Roll No 2200290140128)
Riya Rai
(University Roll No 2200290140129)

Submitted in partial fulfillment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of

Ms Komal Salgotra
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(JUNE 2024)

CERTIFICATE

Certified that **Riya rai (2200290140129), Riya Khurrana (2200290140128)** have carried out the project work having "Chat Application" (**Major Project(KCA451**)) for **Master of Computer Application** from Dr.A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU),Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

Riya Khurana (2200290140128)

Riya Rai (2200290140129)

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

Date:

Miss Komal Salgotra
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Dr. Arun Tripathi
Head
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Chat Application

Riya Khurrana

Riya rai

ABSTRACT

The "Chat Application" project is an Angular-based chat application. It uses Angular CLI version 16.0.0 and includes features such as user authentication with credentials stored in `db.json`. To set up, clone the repository, install dependencies via `npm install`, and run the server using `npx json-server --watch db.json`. The application can be accessed locally at `http://localhost:4200/`. The project also supports code scaffolding, building, unit tests with Karma and end to end test. Angular CLI (Command Line Interface) is a tool that helps developers initialize, develop, scaffold, and maintain Angular applications. It simplifies the setup process, manages dependencies, and provides commands for building, testing, and deploying Angular projects. With Angular CLI, developers can quickly generate components, services, modules, and other Angular artifacts, ensuring a consistent project structure and development workflow. It also offers built-in support for best practices like TypeScript, Webpack, and Angular Material. Node.js is a crucial part of the ChatApp-Angular project, serving as the runtime environment for the server-side logic. It allows for the creation of a lightweight and efficient backend using JavaScript. In this chat application, Node.js, along with tools like `json-server`, is used to manage and serve the application's data, handle API requests, and ensure real-time data updates. This setup facilitates seamless communication between the client (Angular) and the server, ensuring a responsive and interactive user experience.

ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, Miss Komal Salgotra for her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to Dr. Arun Kumar Tripathi, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help on various occassions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Riya Rai

Riya Khurana

TABLE OF CONTENTS

Certificate	ii
Abstract	iii
Acknowledgement	iv
Table of Contents	v-v
List of figures	v
1.Introduction	1-8
1.1 Overview	1
1.2 Need	2-3
1.3 Project Description	3
1.4 Proposed System	3-5
1.5 Features of the system	5-6
1.6 Project scope	7-8
2.Literature review	**9-14**
2.1 Abstract	9
2.2 Introduction	9-10
2.3 Literature	10-13
2.3.1 Working mechanism	10
2.3.2 Issues in existing system	11-12
2.3.3 Future scope	12-13
2.4 Literature Survey	13-14
3.Feasibility study	**15-20**
3.1Technical feasibility	15-17

3.1.1Hardware feasibility 15

3.1.2Software feasibility 16

3.2 Economic feasibility 16

3.3 Legal feasibility 16

3.4 Operational feasibility 16

3.5 Behavioural feasibility 16

3.6 Chat Application feasibility study 16-19

3.6.1 Technical feasibility 16-17

3.6.1.1 Technology stack 16

3.6.1.2 Integration and compatability 17

3.6.2 Economic feasibility 17-18

3.6.1.3 Cost on analysis 17

3.6.1.4 Return on investment 18

3.6.3 Operational Feasibility 18

3.6.1.5 Development team skills 18

3.6.1.6 Implementation 18

3.6.1.7 User experience 18

3.6.4 Scheduling Feasibility 19

3.6.1.8 Project timeline 19

3.6.1.9 Milestones 19

3.7 Risk Analysis 20

4. Design **21-32**

4.1 Design goals 21

4.2 Use case diagram 21-22

4.3 Functional flow of the system 23

4.4 Entity-Relationship diagram 24

4.5 Flowchart 25-26

4.6 DFD 27-28

4.6 Structure chart 28-30

4.7 System Design 31-32

5. Requirement 33-35

5.1 User requirement 33-35

5.2 Hardware and software requirement 35

6.Testing and debugging 36-39

6.1 Unit testing 36

6.2 Integration testing 37

6.3 System testing 37-38

6.4 Acceptance testing 38

6.5 Debugging 39

7. Testing and test results 40-45

7.1 Types of testing 40

7.2 Testing tools 41

7.3 Example Test cases 42-45

8.Implementation 46-48

8.1 Modules 47-48

9. Screenshot 49-50

Conclusion 51

References 52

Source code 54-69

List of Figures

Fig 3.1	Feasibility study	15
Fig 4.2	Use case Diagram	22
Fig 4.3	Activity Diagram	23
Fig 4.4	E-R Diagram	24
Fig 4.1.1.1	Terminal	25
Fig 4.1.1.2	Input/Output	25
Fig 4.1.1.3	Processing	26
Fig 4.1.1.4	Design	26
Fig 4.1.1.5	Connectors	26
Fig 4.5	DFD	28
Fig 4.6.1	Module	29
Fig 4.6.2	Condition	29
Fig 4.6.3	Jump	30
Fig 4.6.4	Loop	30
Fig 4.6.5	Data Flow	31
Fig 4.6.6	Control Flow	31
Fig 4.8	System specification	32

Fig 9.1	JSON Server	48
Fig 9.2	Sign up page	49
Fig 9.3	Login page	50
Fig 9.4	Chat Window	51

CHAPTER 1

INTRODUCTION

1.1 Overview

The "ChatApp-Angular" project is an Angular-based chat application. It uses Angular CLI version 16.0.0 and includes features such as user authentication with credentials stored in `db.json`. To set up, clone the repository, install dependencies via `npm install`, and run the server using `npx json-server --watch db.json`. The application can be accessed locally at `http://localhost:4200/`. The project also supports code scaffolding, building, unit tests with Karma, and end-to-end tests.

Angular CLI (Command Line Interface) is a tool that helps developers initialize, develop, scaffold, and maintain Angular applications. It simplifies the setup process, manages dependencies, and provides commands for building, testing, and deploying Angular projects. With Angular CLI, developers can quickly generate components, services, modules, and other Angular artifacts, ensuring a consistent project structure and development workflow. It also offers built-in support for best practices like TypeScript, Webpack, and Angular Material.

Node.js is a crucial part of the ChatApp-Angular project, serving as the runtime environment for the server-side logic. It allows for the creation of a lightweight and efficient backend using JavaScript. In this chat application, Node.js, along with tools like `json-server`, is used to manage and serve the application's data, handle API requests, and ensure real-time data updates. This setup facilitates seamless communication between the client (Angular) and the server, ensuring a responsive and interactive user experience.

The app is designed to be user-friendly and accessible, with a clean and modern interface that adapt seamlessly to different devices and screen sizes. Whether users are

chatting on their smartphones, tablets, or computers, the chat App ensures a consistent and enjoyable experience.

1.2 Need

Building a chat application using Angular CLI and node.js offers several benefits and Fulfil specific needs.

Front end with Angular CLI:

1. Scalability: Angular's Component based architecture makes it easy to scale and maintain.
2. Performance: Efficient change detection and data binding enhance performance.
3. User experience: Angular provides a responsive, dynamic UI with built-in features like form validating and routing
4. Development efficiency: Angular CLI streamlines development with commands for generating components, services and other artifacts.

Back-End with Node.js:

1. Real-time communication: Node.js with libraries like Socket.IO, enables real time, bidirectional communication between the client and server.
2. High-Performance: Non-blocking, event driven architecture makes Node.js ideal for handling multiple connections simultaneously.
3. JavaScript throughput: Using JavaScript for both client and server simplifies the development process and reduce context switching.
4. Packet Ecosystem: NPM offers a vast range of packages to extend functionality easily.

Combined Benefits:

1. Full-stack development: Seamless integration between front-end and back-end, allowing for faster development and debugging.
2. Real-time features: Essential for chat applications to provide instant messaging, notifications, and updates.

3. Maintainability and testing: Both angular and Node.js support robust testing frameworks, ensuring code quality and reliability.

Typical features:

- User-Authentication: secure login and registration using JWT and OAuth.
- Message storage: Efficient data management for storing chat histories.
- Group chats: Capability to create and manage group conversation.
- Message Storage: Efficient database management for storing chat histories.
- Group Chats: Capability to create and manage group conversations.
- Media Sharing: Ability to send images, files, and other media. • Notifications: Real-time alerts for new messages and activities.

By using Angular CLI and Node.js, developers can create a powerful, scalable, and efficient chat application that meets modern communication needs

1.2 PROJECT DESCRIPTION

This chat application is built using Angular CLI for the front-end and Node.js for the backend, offering a robust, real-time communication platform. It features secure user authentication, real-time messaging, and a user-friendly interface designed with Angular. The back-end leverages Node.js and `json-server` for efficient data handling and real-time updates via WebSocket or similar technologies. This architecture ensures scalability, performance, and maintainability, catering to modern communication needs with capabilities like group chats, media sharing, and notifications.

1.3 PROPOSED SYSTEM

Proposed system architecture and workflow:

Frontend (Angular CLI):

1. Setup: Use Angular CLI to scaffold the frontend project.
2. UI Design: Design the chat interface using Angular Material or any UI framework of your choice. Include components for message display, user input, user authentication, etc.
3. Components: Create Angular components for different parts of the chat application like message list, message input box, user list, etc.

4. Services: Implement Angular services to handle data fetching, sending messages, user authentication, etc. Use Angular's HttpClient module to communicate with the backend.
5. Routing: Set up Angular routing for navigating between different pages or components like login, chat room, user profile, etc.
6. WebSocket Integration: Use libraries like `socket.io-client` to establish a WebSocket connection with the backend server for real-time communication.

Backend (Node.js):

1. Setup: Initialize a Node.js project and install necessary dependencies using npm or yarn.
2. Express.js: Use Express.js to create the backend server. It will handle HTTP requests from the frontend and manage WebSocket connections for real-time messaging.
3. User Authentication: Implement user authentication using libraries like Passport.js and JSON Web Tokens (JWT). Provide endpoints for user registration, login, logout, etc.
4. Database: Choose a database (MongoDB, MySQL, PostgreSQL, etc.) for storing user data, chat messages, etc. Use libraries like Mongoose (for MongoDB) or Sequelize (for SQL databases) to interact with the database.
5. WebSocket Server: Use libraries like `socket.io` to create a WebSocket server. This server will handle real-time messaging between users.
6. API Endpoints: Define API endpoints for functionalities like sending messages, fetching message history, retrieving user information, etc.
7. Security: Implement security measures such as input validation, sanitization, and authorization to prevent common web vulnerabilities like XSS, CSRF, etc.

Integration:

1. Authentication Flow: Integrate frontend authentication with backend authentication using JWT tokens. Secure API endpoints to ensure only authenticated users can access them.
2. Real-time Communication: Connect frontend WebSocket client with backend WebSocket server for real-time messaging.
3. Error Handling: Implement error handling mechanisms on both frontend and backend to gracefully handle errors and provide meaningful feedback to users.

Testing:

1. Unit Testing: Write unit tests for frontend and backend components using tools like Jasmine, Karma, Mocha, Chai, etc.

2. Integration Testing: Perform integration tests to ensure seamless communication between frontend and backend components.

3. End-to-End Testing: Test the entire application flow from user authentication to real-time messaging using tools like Protractor, Cypress, etc.

Deployment:

1. Frontend Deployment: Deploy the Angular frontend to a hosting service like Netlify, Vercel, Firebase Hosting, etc.

2. Backend Deployment: Deploy the Node.js backend to a cloud platform like Heroku, AWS, Google Cloud Platform, etc. Configure necessary environment variables and set up continuous integration and deployment pipelines if needed.

1.5 FEATURES OF THE SYSTEM

A chat application using Angular CLI and Node.js can have several features to make it functional, interactive, and user-friendly. Here are some common features you might want to include:

1. User Authentication: Allow users to sign up, sign in, and manage their profiles securely. Use techniques like JWT (JSON Web Tokens) for authentication.

2. Real-Time Messaging: Implement real-time communication using technologies like WebSockets (e.g., Socket.io) to enable instant messaging between users.

3. Chat Rooms/Channels: Support multiple chat rooms or channels where users can join and engage in group conversations.

4. Message History: Store and display message history so users can view past conversations when they join a chat room.

5. User Status Indicators: Show online/offline status indicators for users and display when users are typing.

6. Message Formatting: Allow users to format their messages using features like bold, italics, lists, etc., for better communication.

7. File Sharing: Enable users to share files (images, documents, etc.) with each other within the chat interface.

8. Notifications: Implement notifications to alert users about new messages, mentions, or other relevant activities even when they are not actively using the application.

9. Search Functionality: Provide search functionality to allow users to search for specific messages or users within the chat application.

10. Emojis and Reactions: Allow users to express themselves using emojis and reactions to messages.

11. User Mentions: Enable users to mention other users in messages, notifying them directly.

12. Security: Implement security measures to prevent common threats like XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), and ensure data encryption for sensitive information.

13. Responsive Design: Ensure the application is responsive and works well on various devices and screen sizes.

14. Profile Customization: Allow users to customize their profiles by adding avatars, bios, etc.

15. Moderation Tools: Provide tools for moderators/administrators to manage users, messages, and chat rooms, including features like banning users or deleting messages.

16. Localization: Support multiple languages to cater to users from different regions.

17. Analytics and Reporting: Include analytics to track user engagement, popular chat rooms, etc., and reporting features for moderators to monitor the activity within the application.

18. Integration with Other Services: Integrate with other services like third-party authentication providers (Google, Facebook, etc.) or other communication tools for enhanced functionality.

When building such an application, it's crucial to focus on both frontend (Angular CLI) and backend (Node.js) development, ensuring seamless communication between the two and robust architecture to handle real-time messaging efficiently.

1.6 PROJECT SCOPE

The project scope for a chat application using Angular CLI and Node.js could include several components and features. Here's a breakdown of potential aspects to consider:

1. User Authentication and Authorization:
    - Implement user registration and login functionality.
    - Authenticate users securely.
    - Authorize access to certain features based on user roles.

2. Real-Time Communication:
    - Develop real-time messaging using technologies like WebSockets or libraries like Socket.IO.
    - Support instant messaging between users.
    - Implement features like typing indicators, read receipts, and message history.

3. User Interface:
    - Design an intuitive and responsive user interface using Angular CLI.
    - Include features such as user profiles, contact lists, and message threads.
    - Ensure cross-browser compatibility and mobile responsiveness.

4.Data Management:
    - Set up a database to store user information, chat messages, and other relevant data.
    - Utilize Node.js for server-side logic and database interactions.
    - Implement CRUD (Create, Read, Update, Delete) operations for managing user data and messages.

5. Security:
    - Implement security measures to protect against common web vulnerabilities like XSS (Cross-Site Scripting) and CSRF (Cross-Site Request Forgery).
    - Securely handle user authentication tokens and sensitive data.
    - Use HTTPS for secure communication between the client and server.

6. Notifications:
    - Implement push notifications to alert users of new messages or other relevant updates.
    - Allow users to customize notification preferences.

7. Additional Features:
    - Support multimedia messaging (e.g., images, files).
    - Implement search functionality to easily find past messages or users.
    - Include features like emoji reactions, message editing, and deleting.

8. Testing and Deployment:
    - Write unit tests and end-to-end tests to ensure application reliability.
    - Deploy the application to a production environment, considering scalability and performance requirements.
    - Set up monitoring and logging to track application performance and detect issues.

9. Documentation and Support:
    - Provide comprehensive documentation for developers to understand the application's architecture, APIs, and deployment process.
    - Offer user documentation to guide users through the application features and usage.
    - Establish a support system to address user inquiries and troubleshoot issues.

10. Future Considerations:
    - Plan for scalability and potential future enhancements, such as supporting group chats or integrating with other services.
    - Consider internationalization and localization for a diverse user base.
    - Stay updated with Angular CLI and Node.js releases for security patches and new features.

CHAPTER-2

Literature Review

2.1 Abstract

The "Chat Application" project is an Angular-based chat application. It uses Angular CLI version 16.0.0 and includes features such as user authentication with credentials stored in `db.json`. To set up, clone the repository, install dependencies via `npm install`, and run the server using `npx json-server --watch db.json`. The application can be accessed locally at `http://localhost:4200/`. The project also supports code scaffolding, building, unit tests with Karma and end to end test. Angular CLI (Command Line Interface) is a tool that helps developers initialize, develop, scaffold, and maintain Angular applications. It simplifies the setup process, manages dependencies, and provides commands for building, testing, and deploying Angular projects. With Angular CLI, developers can quickly generate components, services, modules, and other Angular artifacts, ensuring a consistent project structure and development workflow. It also offers built-in support for best practices like TypeScript, Webpack, and Angular Material. Node.js is a crucial part of the ChatApp-Angular project, serving as the runtime environment for the server-side logic. It allows for the creation of a lightweight and efficient backend using JavaScript. In this chat application, Node.js, along with tools like `json-server`, is used to manage and serve the application's data, handle API requests, and ensure real-time data updates. This setup facilitates seamless communication between the client (Angular) and the server, ensuring a responsive and interactive user.

2.2 Introduction

The "ChatApp-Angular" project is an Angular-based chat application. It uses Angular CLI version 16.0.0 and includes features such as user authentication with credentials stored in `db.json`. To set up, clone the repository, install dependencies via `npm install`, and run the server using `npx json-server --watch db.json`. The application can be accessed locally at `http://localhost:4200/`. The project also supports code scaffolding, building, unit tests with Karma, and end-to-end tests.

Angular CLI (Command Line Interface) is a tool that helps developers initialize, develop, scaffold, and maintain Angular applications. It simplifies the setup process, manages

dependencies, and provides commands for building, testing, and deploying Angular projects. With Angular CLI, developers can quickly generate components, services, modules, and other Angular artifacts, ensuring a consistent project structure and development workflow. It also offers built-in support for best practices like TypeScript, Webpack, and Angular Material.

Node.js is a crucial part of the ChatApp-Angular project, serving as the runtime environment for the server-side logic. It allows for the creation of a lightweight and efficient backend using JavaScript. In this chat application, Node.js, along with tools like `json-server`, is used to manage and serve the application's data, handle API requests, and ensure real-time data updates. This setup facilitates seamless communication between the client (Angular) and the server, ensuring a responsive and interactive user experience.

The app is designed to be user-friendly and accessible, with a clean and modern interface that adapt seamlessly to different devices and screen sizes. Whether users are chatting on their smartphones, tablets, or computers, the chat App ensures a consistent and enjoyable experience.

There are several research papers available on this topic. For example, a research paper titled "A Machine Learning Based Chatbot Song Recommender System" proposes a chatbot song recommender system that recommends songs based on the user's current emotion or mood. The proposed system uses IBM Tone Analyzer API to check the text tone of the user and to predict the mood based on the text of the user, and Last.FM API to recommend songs based on the mood of the user [1]. Another research paper titled "Chatbot with Song Recommendation based on Emotion" proposes a chatbot that interacts with the user, analyzes the emotions of chats, and recommends a song playlist based on the user's emotions. The objective of the application is to identify the emotion perceived by the user, and once the emotion is identified, a list of songs is suggested based on the emotion.

2.3 Literature review

2.3.1 Working mechanism

1. **Client-Side (Angular)**

- Components and Services: The Angular application is structured into various components and services. Components like `ChatComponent` handle the user interface, displaying messages and user interactions. Services like `ChatService`

manage the WebSocket connection to the server and provide methods to send and receive messages.

- Data Binding: Angular's two-way data binding allows real-time updates of the UI based on user inputs and incoming messages.
- Routing: Angular's routing module enables navigation between different views, such as login and chat screens.
- State Management: Tools like NgRx can be used for state management, ensuring a consistent state across the application.

2. Server-side (Node.js):

- Express Server: The server is built using Express, a lightweight web application framework for Node.js, which handles HTTP requests and serves static files.
- WebSocket Communication: Using Socket.IO, the server establishes real-time bidirectional communication with connected clients. This is essential for sending and receiving messages in real-time.
- Authentication: The server handles user authentication, typically using JWT (JSON Web Tokens) to ensure secure access. Tokens are verified for each WebSocket connection to authenticate users.

3. Real-Time Messaging

- Message Handling: When a user sends a message, it is emitted from the client via WebSocket to the server. The server then broadcasts this message to all connected clients, ensuring all users receive the message instantly.
- Persistence: Messages can be stored in a database (like MongoDB) for historical purposes, allowing users to see past messages when they reconnect.

2.3.2 Issues in existing system

1. Scalability

- Performance Bottlenecks: Traditional chat applications may face performance issues when the number of concurrent users increases. Handling a large number of WebSocket connections and broadcasting messages efficiently requires robust infrastructure and optimization techniques.

2. Security Concerns

- Data Breaches: Inadequate security measures can lead to unauthorized access and data breaches. Without proper encryption and secure authentication methods, user data can be compromised.
- Injection Attacks: Chat applications are vulnerable to various injection attacks (like XSS and SQL injection) if inputs are not properly sanitized.

3. Reliability

- Connection Stability: Maintaining stable WebSocket connections can be challenging, especially with intermittent network issues or server crashes. Ensuring seamless reconnections and data consistency is crucial.
- Load Balancing: Distributing the load across multiple servers effectively is necessary to prevent downtime and ensure high availability.

4. User Experience

- Latency: High latency can degrade the user experience, causing delays in message delivery. Optimizing real-time communication to minimize latency is essential.
- Interface Complexity: Overly complex user interfaces can be daunting for users. Ensuring a simple, intuitive UI design is important for user engagement.

2.3.3 Future scope

1. Advanced Features

- Rich Media Support: Integrating support for rich media (images, videos, files) can enhance user interaction.
- Voice and Video Calls: Implementing VoIP and video calling features can provide a comprehensive communication solution.

2. Improved Security

- End-to-End Encryption: Implementing end-to-end encryption to ensure that only communicating users can read the messages.
- Multi-Factor Authentication: Adding multi-factor authentication for an added layer of security.

3. Scalability Enhancements

- Microservices Architecture: Adopting a microservices architecture can help in scaling different components independently, improving overall scalability.
- Serverless Architectures: Leveraging serverless computing for certain functionalities to enhance scalability and reduce infrastructure management overhead.

4. AI and Machine Learning

- Chatbots and Automation: Integrating AI-driven chatbots for automated responses and support.
- Content Moderation: Using machine learning algorithms for real-time content moderation to filter inappropriate messages.

2.4 Literature survey

1. Real-Time Communication Protocols

- WebSocket vs. HTTP: Studies comparing WebSocket and HTTP long polling for realtime applications highlight the efficiency and low latency of WebSocket for chat applications.
- Socket.IO Implementation: Research on Socket.IO showcases its ability to handle realtime bidirectional communication efficiently, making it suitable for chat applications.

2. Front-End Frameworks

- Angular Performance: Various papers and articles evaluate Angular's performance and its suitability for building dynamic, single-page applications (SPAs). Angular's powerful CLI and extensive ecosystem support rapid development and maintainability.
- State Management: Studies on state management solutions like NgRx emphasize the importance of predictable state management and its impact on application performance and scalability.

3. Back-End Technologies

- Node.js Scalability: Research on Node.js highlights its non-blocking, event-driven architecture, which makes it highly suitable for I/O-bound applications like chat systems.

- Express Framework: The simplicity and extensibility of Express for building web applications are well-documented, making it a popular choice for server-side logic in chat applications.

4. Security Practices

- JWT Authentication: Literature on JWT provides insights into its effectiveness in securing APIs and real-time applications through token-based authentication.
- WebSocket Security: Research on securing WebSocket communications underscores the importance of using WSS and implementing robust authentication mechanisms.

5. User Experience

- UI/UX Design Principles: Studies on user interface design for chat applications emphasize the need for simplicity, responsiveness, and accessibility to enhance user satisfaction and engagement.
- Performance Optimization: Research on performance optimization techniques for Angular applications, such as lazy loading and AOT compilation, highlights their impact on improving application responsiveness.

A chat application using Angular CLI and Node.js combines the strengths of these technologies to deliver a scalable, real-time communication platform. The working mechanism involves seamless integration of front-end and back-end components, leveraging WebSocket for real-time messaging. Addressing issues like scalability, security, and user experience in existing systems is crucial. The future scope includes incorporating advanced features, enhancing security, improving scalability, and leveraging AI. A thorough literature survey provides insights into best practices and technological advancements, guiding the development of robust and efficient chat applications.

CHAPTER-3

FEASIBILITY STUDY

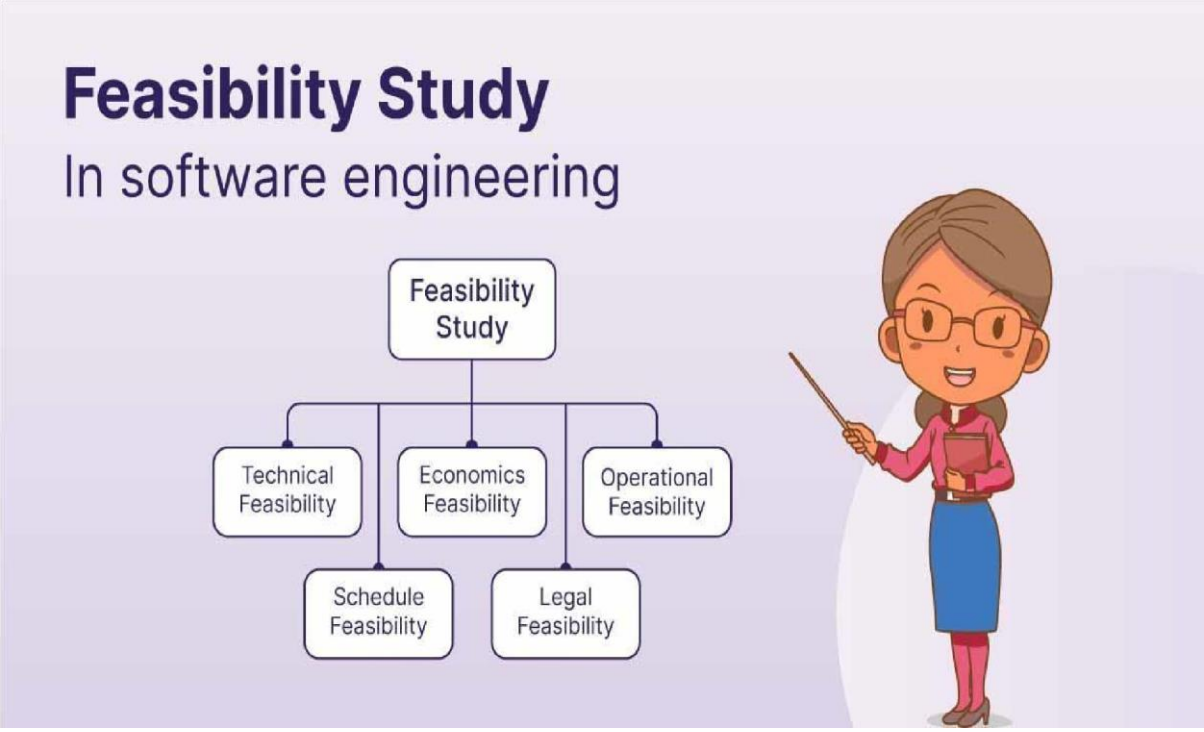

Fig 3.1 feasibility study

The feasibility study for the proposed Chat application reveals promising outcomes across technical, economic, operational, and scheduling dimensions. From a technical standpoint, the required software, hardware, and expertise are readily available, facilitating seamless development and integration. Economic feasibility is supported by a favourable cost-benefit analysis, showcasing a reasonable return on investment over time. Operationally, the study indicates a positive outlook, with users displaying openness to the system, and its alignment with organizational goals is evident. The scheduling feasibility is reinforced by a wellstructured project timeline, accounting for dependencies and potential risks. The project's regulatory and legal aspects are deemed feasible, with a commitment to compliance and data privacy measures. The system exhibits scalability and flexibility, ensuring adaptability to

future demands. Overall, the feasibility study indicates a strong foundation for the successful development and implementation of the Chat Application.

3.1 Technical feasibility

A technical feasibility study is a standard practice for companies to conduct feasibility studies before commencing work on a project. The study assesses the practicality and viability of a product or service before launching it. Technical feasibility helps determine the efficacy of the proposed plan by analyzing the process, including tools, technology, material, labor, and logistics. The study identifies potential challenges and uncovers ways to overcome them. It also helps in long-term planning, as it can serve as a flowchart for how products and services evolve before they reach the market .In technical feasibility current resources both hardware software along with required technology are analyzed/assessed to develop the project. This technical feasibility study reports whether there exists correct required resources and technologies which will be used for project development.

The technical feasibility study in software engineering is conducted in various ways depending on the company. Some people may do it in a precise and organized method, while others may do it as needed. However, you must have the following resources -

- Hardware
- Software
- Technology
- Skills and knowledge
- Time and budget for development
- Specialists
- Software development tools

3.1.1 Hardware feasibility

Hardware feasibility is a part of technical feasibility that assesses the ability of computer hardware to handle workloads adequately. It involves evaluating the hardware requirements of the project and ensuring that the available hardware is capable of meeting those requirements .

3.1.2 Software feasibility

A feasibility study is performed on a software project to understand the viability of the product. Understanding a project's feasibility has a lot to do with how it will perform in the market, what will work, what competitors have created and how will this product survive.

3.2 Economic feasibility

In economic feasibility study the cost and benefit of the project is analyzed. It means under this feasibility study a detailed analysis is carried out of what will be the cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. After that it is analyzed whether the project will be beneficial in terms of finance for the organization or not. It offers reduced costs, improved efficiency, increased profits with transparency and flexibility.

3.3 Legal feasibility

In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. Overall it can be said that Legal Feasibility Study is study to know if proposed project conform legal and ethical requirements.

3.4 Operational feasibility

In operational feasibility, the degree of providing service to requirements is analyzed along with how easy the product will be to operate and maintain after deployment. Along with this other operational scopes are determining usability of product, determining suggested solution by software development team is acceptable or not etc.

3.5 Behavioural feasibility

Behavioural feasibility is studied in order to check, whether the human or employees in the business will use it or not. Operational feasibility relies on human resources and analyses whether the software will operate after it is developed properly or not.

Behavioural feasibility is a scale of how the proposed system solves the problem, to what extent it takes the advantage of the opportunities identified during scope definition and how much it satisfies the requirements identified in the requirement analysis of the software development.

3.6 Chat Application feasibility study

3.6.1. Technical Feasibility

3.6.1.1 Technology Stack

Angular CLI:

- Benefits: Angular CLI offers a powerful set of tools for scaffolding, building, and testing Angular applications. It supports modular development, making the application scalable and maintainable.
- Features: Two-way data binding, dependency injection, and a robust componentbased architecture.

Node.js:

- Benefits: Node.js is designed for building scalable network applications. Its nonblocking I/O model is suitable for real-time applications like chat systems.
- Features: Event-driven architecture, rich ecosystem with npm, and high performance for I/O-bound tasks.

Socket.IO:

- Benefits: Provides real-time bidirectional event-based communication. It's a popular choice for real-time web applications.
- Features: WebSocket support, automatic reconnection, and easy integration with both Node.js and Angular.

3.6.1.2. Integration and Compatibility

- Frontend-Backend Communication: Angular can easily communicate with the Node.js backend using RESTful APIs and WebSockets (via Socket.IO).
- Development Tools: Both Angular and Node.js have extensive documentation and community support, which facilitates development and troubleshooting.
- Scalability: The chosen stack can handle scaling up as the user base grows. Node.js, in particular, is known for handling many simultaneous connections efficiently.

3.6.2. Economic Feasibility

3.6.2.1. Cost Analysis

- Development Costs: The primary costs involve salaries for developers skilled in Angular and Node.js, which are relatively common and hence cost-effective.
- Infrastructure Costs: Hosting on cloud platforms (like AWS, Azure, or Heroku) can be scaled according to demand. Initial costs are low and can grow with the user base.
- Maintenance Costs: Ongoing costs include server maintenance, updates, and bug fixes. Using managed services can reduce these costs.

3.6.2.2. Return on Investment (ROI)

- Revenue Streams: Potential revenue can come from subscription models, in-app purchases, advertisements, and premium features.
- Market Demand: The demand for real-time communication tools is high, particularly for businesses and social interactions, indicating a positive market response.

3.6.3. Operational Feasibility

3.6.3.1. Development Team Skills

- Skill Requirements: Developers need to be proficient in Angular, Node.js, and WebSocket protocols. Skills in database management (e.g., MongoDB) and security practices are also necessary.
- Training: Minimal training is required due to the extensive documentation and community support for Angular and Node.js.

3.6.3.2. Implementation

- Development Workflow: The workflow involves setting up the Angular frontend and Node.js backend, integrating Socket.IO for real-time communication, and implementing security measures.
- Testing: Comprehensive testing (unit, integration, and end-to-end) is essential to ensure reliability and performance.

3.6.3.3. User Experience

- UI/UX Design: Focusing on intuitive design and responsiveness is critical. Angular provides robust tools for building a user-friendly interface.

- Performance: Ensuring low latency and high responsiveness is crucial for user satisfaction. Techniques like lazy loading and Ahead-of-Time (AOT) compilation in Angular can enhance performance.

3.6.4. Scheduling Feasibility

3.6.4.1. Project Timeline

- Planning and Requirements Gathering: 2-3 weeks
- Design and Architecture: 3-4 weeks
- Frontend Development (Angular): 4-6 weeks
- Backend Development (Node.js): 4-6 weeks
- Integration and Testing: 3-4 weeks
- Deployment and Maintenance Setup: 2-3 weeks

3.6.4.2. Milestones

- Prototype Development: Initial working version with basic chat functionality.
- Beta Release: Feature-complete version with all planned functionalities and thorough testing.
- Official Launch: After addressing feedback from beta testing and ensuring stability.

3.7 Risk analysis

Risk Analysis in project management is a sequence of processes to identify the factors that may affect a project's success. These processes include risk identification, analysis of risks, risk management and control, etc. Proper risk analysis helps to control possible future events that may harm the overall project. It is more of a pro-active than a reactive process.

It is the process of prioritizing risks for further analysis of project risk or action by combining and assessing their probability of occurrence and impact. It helps managers to lessen the uncertainty level and concentrate on high priority risks.

Plan risk management should take place early in the project, it can impact on various aspects for example: cost, time, scope, quality and procurement.

The inputs for qualitative Project Risk Analysis and Management includes

- Risk management plan
- Scope baseline
- Risk register
- Enterprise environmental factors

- Organizational process assets

The output of this stage would be

- Project documents updates

CHAPTER-4

DESIGN

Designing is the most important phase of software development. It requires a careful planning and thinking on the part of the system designer. Designing software means to plan how the various parts of the software are going to achieve the desired goal. It should be done with utmost care because if the phase contains any error, then that will affect the performance of the system, as a result it may take more processing time, more response time, extra coding workload etc.

Software design sits at the technical kernel of the software engineering process and is applied regardless of the software process model that is used. After the software requirements have been analysed and specified, software design is the first of the three technical activities Designing, Coding and Testing that are required to build and verify the software. Each activity transforms information in such a manner that ultimately results in validated computer software.

4.1 Design goals

The following goals were kept in mind while designing the system:

- Make system user-friendly. This was necessary so that system could be used efficiently and system could act as catalyst in achieving objectives.
- Make system compatible i.e. It should fit in the total integrated system. Future maintenance and enhancement must be less.
- Make the system compatible so that it could integrate other modules of system into itself.
- Make the system reliable, understandable and cost-effective.

4.2 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 diagram

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:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

Use case diagram for chat Application:

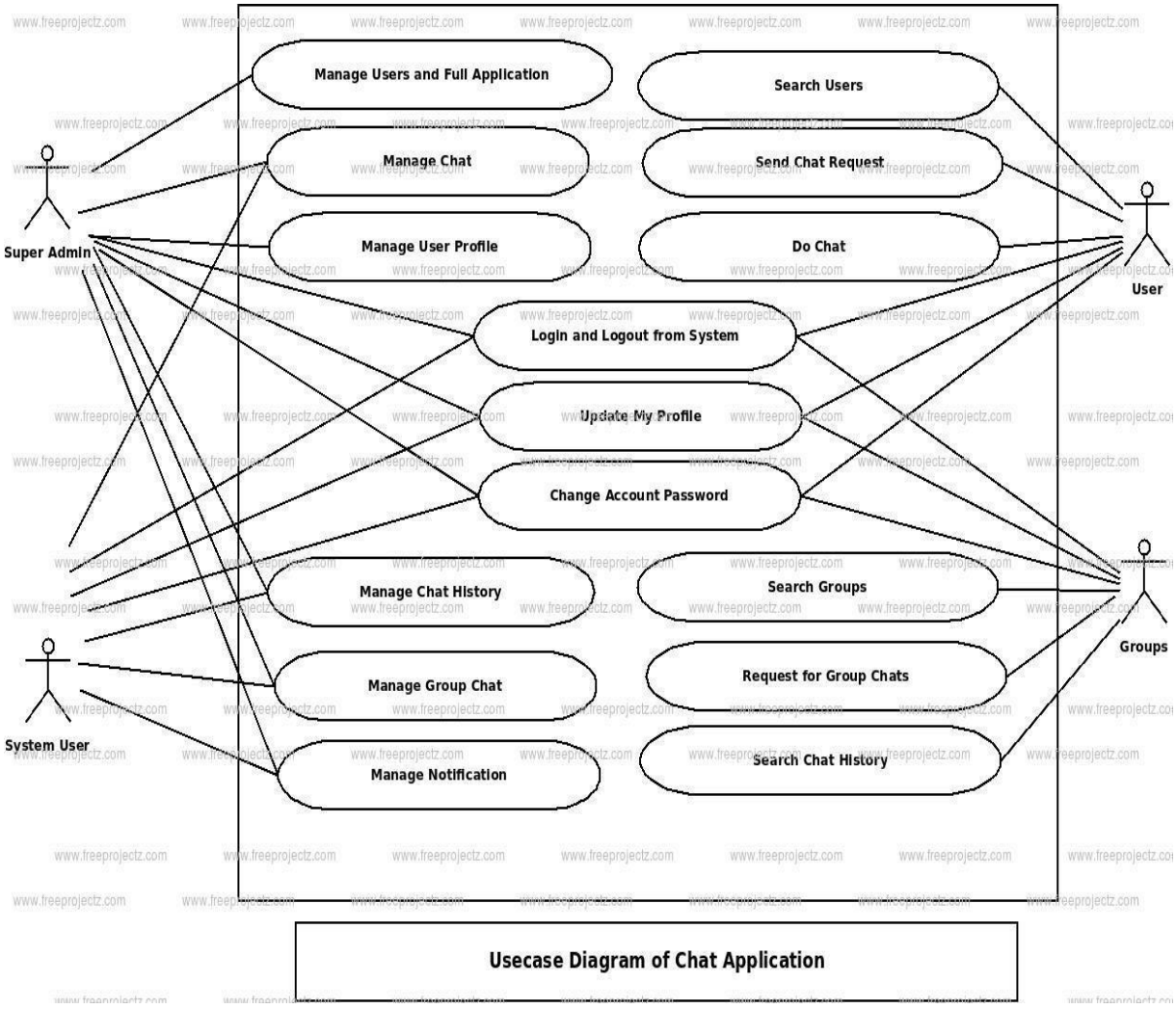

Fig 4.2 use case diagram

4.3 Functional flow of the system

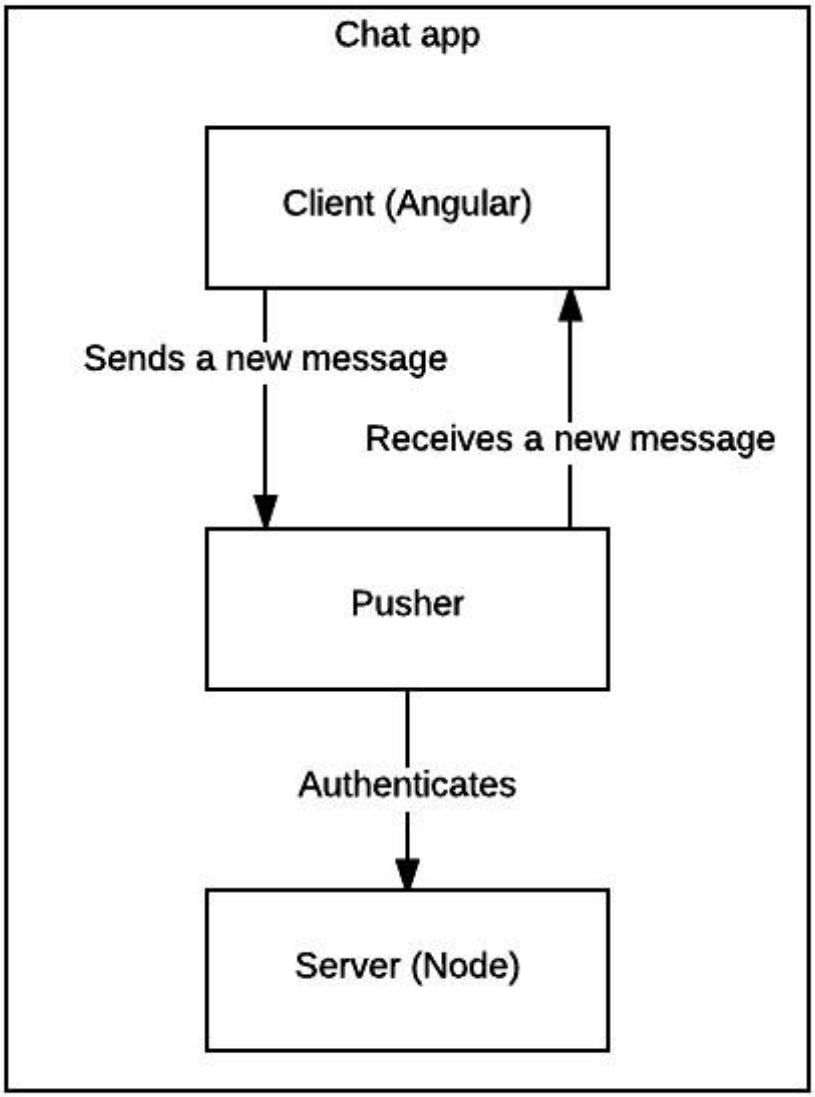

Fig 4.3 Activity diagram

4.4 Entity relationship diagram

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system. ○ It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modelling, the database structure is portrayed as a diagram called an entity-relationship diagram.

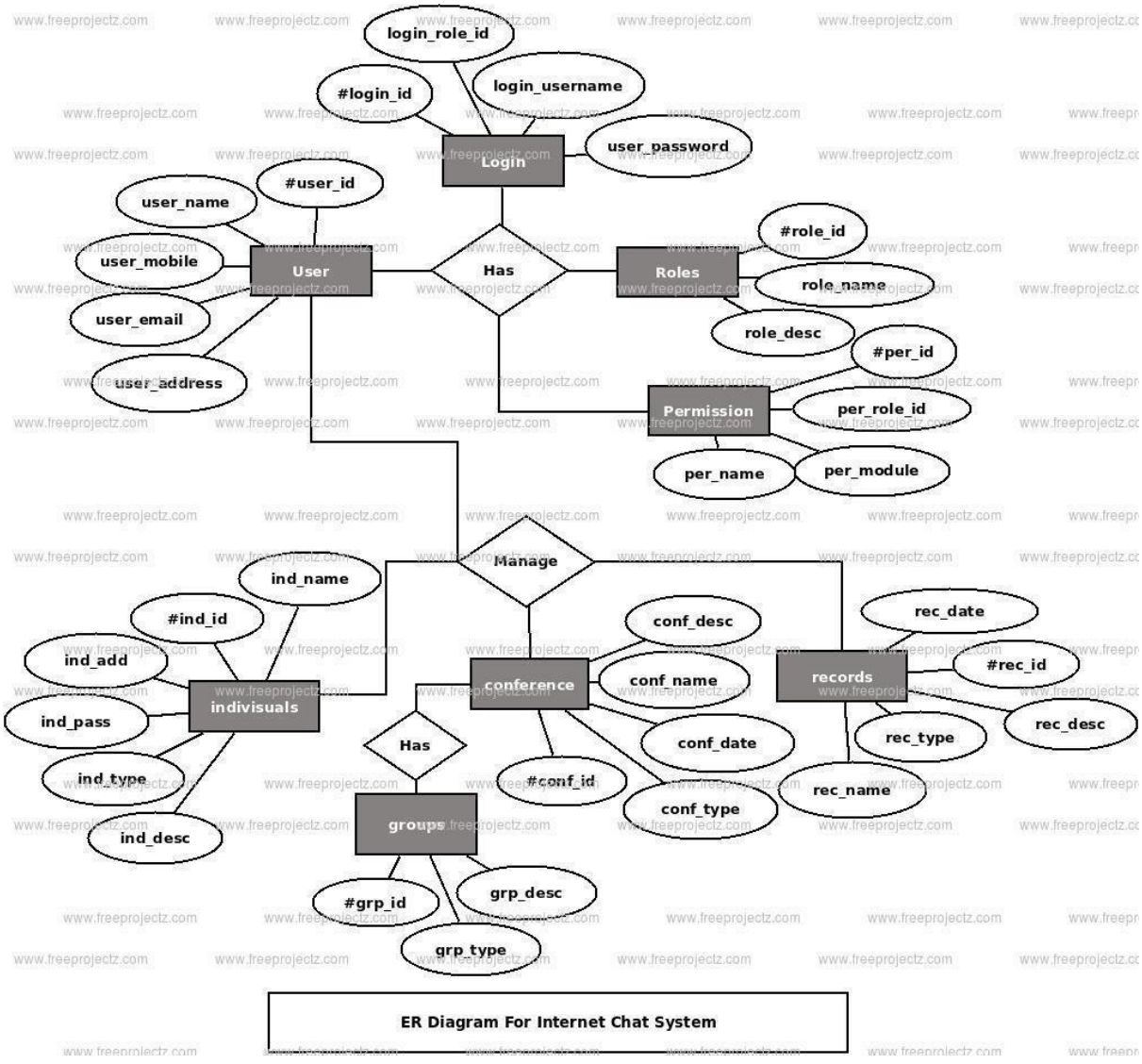

Fig:4.4 ER-diagram

4.5 FLOWCHART

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing.

The process of drawing a flowchart for an algorithm is known as “flowcharting”.

4.1.1 Basic Symbols used in flowchart

1. **Terminal:** The oval symbol indicates Start, Stop and Halt in a program’s logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.

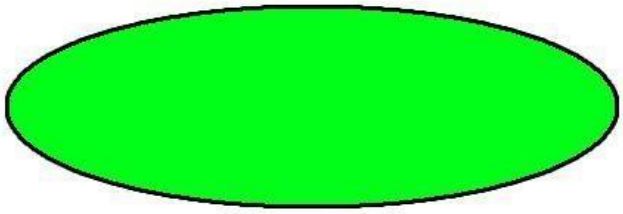

Fig 4.1.1.1 Terminal

- **Input/Output:** A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.

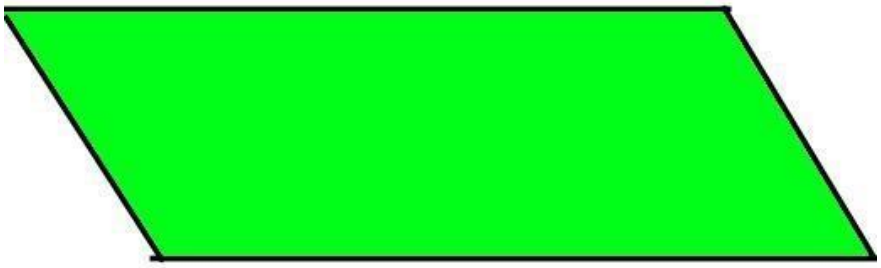

Fig 4.1.1.2 Input/output

- **Processing:** A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.

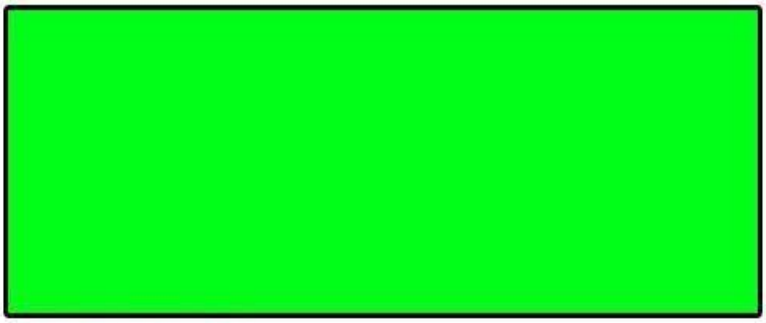

Fig 4.1.1.3 Processing

- **Decision** Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.

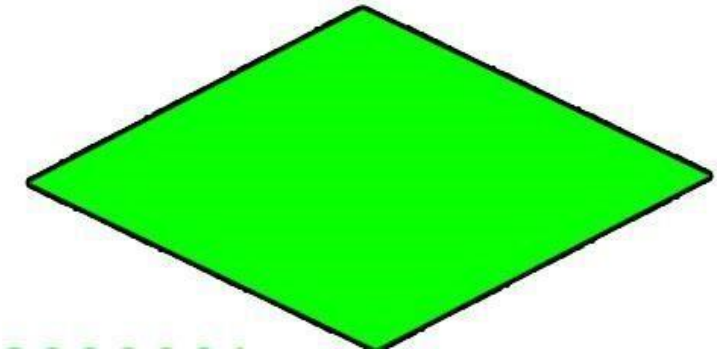

Fig 4.1.1.4 Decision

- **Connectors:** Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.

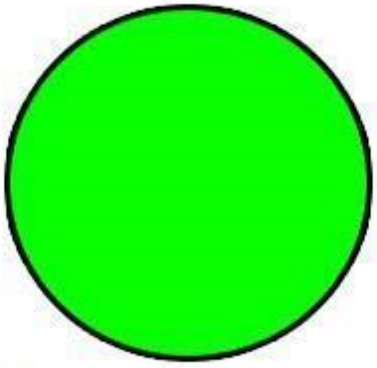

Fig 4.1.1.5 Connectors

- **Flow lines:** Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.

4.6 Data flow diagrams

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.
There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD

Data Flow Diagrams are either Logical or Physical.

- Logical DFD - This type of DFD concentrates on the system process, and flow of data in the system. For example in a Banking software system, how data is moved between different entities.
- Physical DFD - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

DFD Components

DFD can represent Source, destination, storage and flow of data using the following set of components:

- Entities - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- Process - Activities and action taken on the data are represented by Circle or Roundedged rectangles.
- Data Storage - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- Data Flow - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD

- Level 0 - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.

- Level 1 - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.
- Level 2 - At this level, DFD shows how data flows inside the modules mentioned in Level 1. Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Fig 4.6 DFD

4.7 Structure Charts

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and subfunctions of each module of the system to a greater detail than DFD.

Structure chart represents hierarchical structure of modules. At each layer a specific task is performed. Here are the symbols used in construction of structure charts -

- Module - It represents process or subroutine or task. A control module branches to more than one submodule. Library Modules are re-usable and invokable from any module.

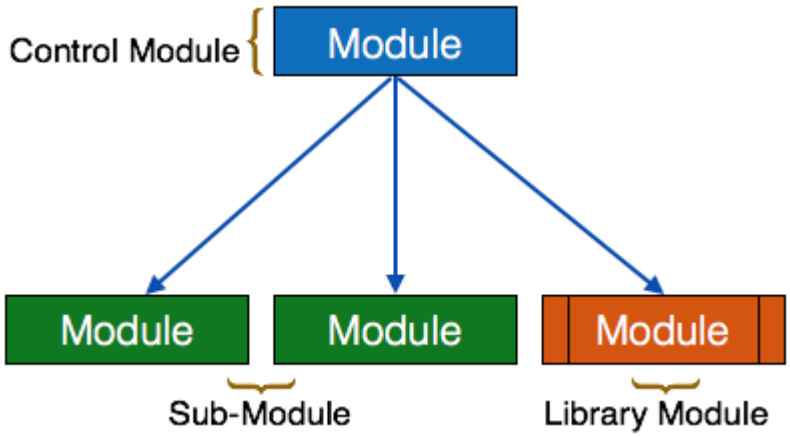

Fig 4.6.1 Module

- Condition - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.

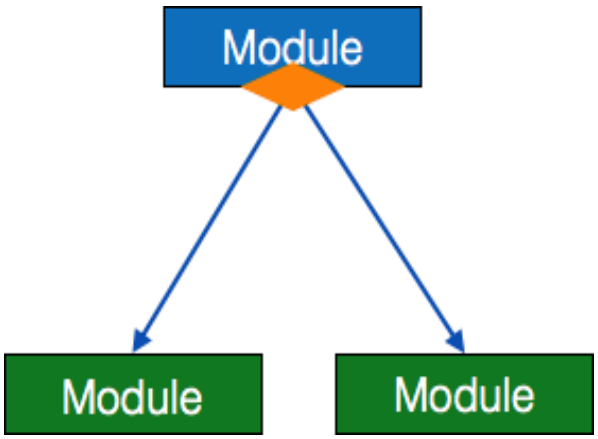

Fig 4.6.2 Condition

- Jump - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.

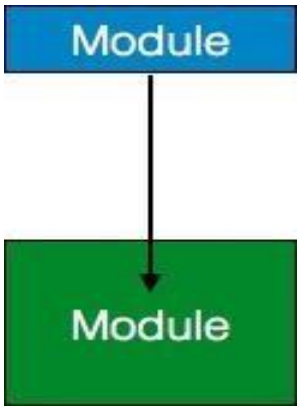

Fig4.6.3Jump

- Loop – A Loop is a control structure.

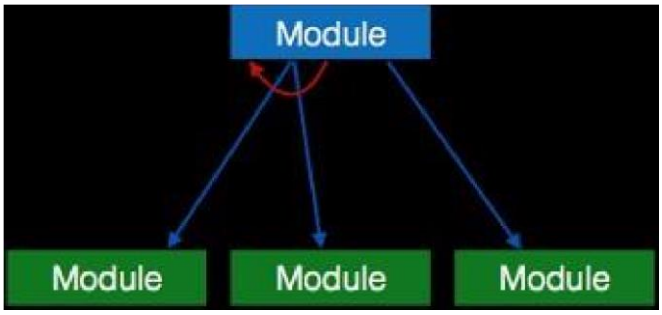

Fig 4.6.4 Loop

- Data flow - A directed arrow with empty circle at the end represents data flow.

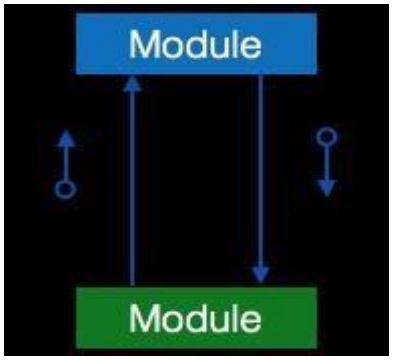

Fig 4.6.5 Data flow

- Control flow - A directed arrow with filled circle at the end represents control flow.

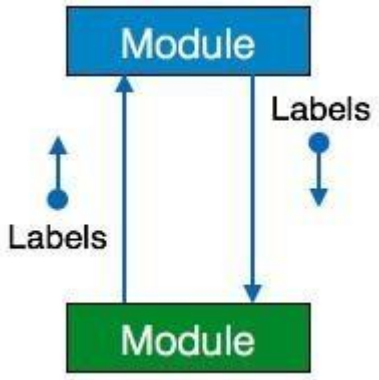

Fig 4.6.6 Control flow

4.8 System Design

System Specification

Process model used: Agile Model

The meaning of Agile is swift or versatile. Agile process model refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

Fig 4.8 System specification

Phases of Agile Model:

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design and requirements: When you have identified the project, work with stakeholders to define requirements. You can use flow diagrams or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction / iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

CHAPTER-5

REQUIREMENTS

5.1 USER REQUIREMENTS

User Requirements for Chat Application:

1. User Registration and Authentication

    - Sign Up: Users should be able to create an account using an email address and password.
    - Login: Users should be able to log in using their email address and password.
    - Authentication: Secure authentication using JWT (JSON Web Tokens).

2. User Profile

    - Profile Management: Users should be able to view and edit their profile information, including username, profile picture, and status message.
    - Online Status: Users should be able to see the online/offline status of their contacts.

3. Contacts and Friends

    - Add Contacts: Users should be able to search for and add contacts.
    - Contact List: Users should have a list of their contacts/friends.

4. Chat Functionality

    - One-on-One Chat: Users should be able to send and receive messages in real-time with their contacts.
    - Group Chat: Users should be able to create groups and chat with multiple users simultaneously.
    - Message Status: Users should be able to see message status (sent, delivered, read).
    - Typing Indicator: Users should see when the other party is typing.

5. Real-Time Updates

    - Web Sockets: Use Web Sockets for real-time message updates.
    - Notifications: In-app notifications for new messages and other events.

6. Media Sharing
    - File Upload: Users should be able to share images, videos, and other files.
    - File Preview: Preview shared media files within the chat.

7. Search and Filter

    - Search Messages: Users should be able to search through their message history.
    - Filter Contacts: Users should be able to filter contacts by name or status.

8. User Interface and Experience

    - Responsive Design: The application should be responsive and work on both desktop and mobile devices.
    - User-Friendly Interface: An intuitive and easy-to-use interface.

9. Security and Privacy

    - Data Encryption: Messages and media should be encrypted end-to-end.
    - Privacy Settings: Users should be able to control their privacy settings (e.g., who can see their status or profile picture).

10. Additional Features

    - Emojis and Stickers: Support for emojis and stickers in messages.
    - Voice and Video Calls: Optional feature for making voice and video calls.

5.2 Hardware and Software Requirement

Hardware Requirements

- Processor: Minimum dual-core CPU
- Memory: At least 4 GB of RAM
- Storage: 100 MB of available disk space

Software Requirements

- Operating System: Windows, macOS, or Linux
- Node.js: Version 16.0.0 or higher
- Angular CLI: Version 16.0.0
- npm: Node Package Manager, included with Node.js installation
- Code Editor: Visual Studio Code or any preferred ID

CHAPTER-6

Testing and Debugging

Software testing is a critical element of the ultimate review of specification design and coding. Testing of software leads to the uncovering of errors in the software functional and performance requirements are met. Testing also provides a good indication of software reliability and software quality as a whole. The result of different phases of testing are evaluated and then compared with the expected results. If the errors are uncovered, they are debugged and corrected. A strategy approach to software testing has the generic characteristics:

- Testing begins at the module level and works "outwards" towards the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points of time.
- Testing and debugging are different activities, but debugging must be accommodated in the testing strategy.

6.1 Unit testing

The module interface is tested to ensure that information properly flows into and out of the program unit under test. The unit testing is normally considered as an adjunct step to coding step. Because modules are not a standalone program, drivers and/or stubs software must be developed for each unit. A driver is nothing more than a "main program" that accepts test cases data and passes it to the module. A stub serves to replace the modules that are subordinate to the modules to be tested. A stub may do minimal data manipulation, prints verification of entry and returns.

Approaches used for Unit Testing were:

Functional Test: Each part of the code was tested individually and the panels were tested individually on all platforms to see if they are working properly.

Performance Test: These determined the amount of execution time spent on various parts of units and the resulting throughput, response time given by the module.

Stress Test: A lot of test files were made to work at the same time in order to check how much workloads can the unit bear.

Structure Test: These tests were made to check the internal logic of the program and traversing particular execution paths.

6.2 Integrartion testing

If they all work individually, they should work when we put them together. The problem of course is "putting them together". This can be done in two ways:

Top-down integration: Modules are integrated by moving downwards through the control hierarchy, beginning with main control module are incorporated into the structure in either a depth first or breadth first manner.

Bottom-up integration: It begins with construction and testing with atomic modules i.e. modules at the lowest level of the program structure. Because modules are integrated from the bottom up, processing required for the modules subordinate to a given level is always available and the need of stubs is eliminated.

Testing includes Verification and Validation

Verification:-is a process of confirming that software meets its specification.

Validation:- is the process of confirming that software meets the customer's requirements.

6.3 System Testing

System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behaviour of a component or a system when it is tested.

System testing process:

System Testing is performed in the following steps:

- Test Environment Setup: Create testing environment for the better quality testing.
- Create Test Case: Generate test case for the testing process.
- Create Test Data: Generate the data that is to be tested.
- Execute Test Case: After the generation of the test case and the test data, test cases are executed.
- Defect Reporting: Defects in the system are detected.
- Regression Testing: It is carried out to test the side effects of the testing process.
- Log Defects: Defects are fixed in this step.
- Retest: If the test is not successful then again test is performed

6.4 ACCEPTANCE TESTING

It is formal testing according to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers, or other authorized entities to determine whether to accept the system or not. Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.

6.5 Debugging

Debugging occurs as a consequence of successful testing i.e. when a test case uncovers an error, debugging is the process that results in identifying the location of error ad the removal of error. The poorly understood mental process that connects a symptom to cause is debugging.

This process will always have one of the two outcomes.

- The cause will be found, corrected and then removed or
- The cause will not be found. In the latter case the person performing debugging may suspect a cause, design a test case to help validate his suspicion, and then work towards the correction of errors in the interactive fashion.

Following three approaches of debugging were used:

- Debugging by Induction
- Debugging by Deduction
- Backtracking

CHAPTER-7

TESTING AND TEST RESULTS

Testing a chat application built with Angular and Node.js involves several types of tests to ensure the application is functional, secure, and performs well. Here's a comprehensive plan for testing, including different testing methods and potential test cases:

7.1 Types of Testing

1. Unit Testing

- Focuses on individual components or services.
- Ensures that each part of the application behaves as expected.

2. Integration Testing

- Tests the interactions between different parts of the application.
- Ensures that components and services work together correctly.

3. End-to-End (E2E) Testing

- Simulates real user scenarios.
- Tests the entire flow of the application from start to finish.

4. Performance Testing

- Assesses the application's performance under various conditions.
- Ensures the application can handle a large number of users and messages.

5. Security Testing

- Identifies vulnerabilities and ensures the application is secure.
- Includes tests for authentication, authorization, and data encryption.

6. Usability Testing

- Evaluates the user interface and user experience.
- Ensures the application is intuitive and easy to use.

7.2 Testing Tools

- Unit Testing: Jasmine, Karma (Angular), Mocha, Chai (Node.js)
- Integration Testing: Jasmine, Karma, Mocha, Chai
- End-to-End Testing: Protractor (Angular), Cypress
- Performance Testing: Artillery, JMeter
- Security Testing: OWASP ZAP, Nessus
- Usability Testing: Manual testing, User feedback tools like Hotjar

7.3 Example Test Cases

Unit Tests

Angular Frontend:

1. Authentication Service:

- Test login method with valid and invalid credentials.
- Test token storage and retrieval.

2. Chat Service:

- Test sendMessage method.
- Test getMessages method to ensure it retrieves the correct messages.

3. User Profile Component:

- Test profile data binding.
- Test profile update method.

Node.js Backend:

1. User Model:

- Test user creation with valid and invalid data.
- Test password hashing and validation.

2. Message Model:

- Test message creation and retrieval.
- Test message schema validation.

3. Auth Middleware:

- Test JWT validation.
- Test route protection.

Integration Tests

1. User Registration and Login:

- Test the complete flow from user registration to login and token generation.

2. Message Sending and Receiving:

- Test the interaction between the frontend sending a message and the backend storing it.
- Verify real-time message updates using Socket.io.

End-to-End Tests

1. User Registration and Login Flow:

- Test the entire process from visiting the registration page, creating an account, and logging in.

2. Chat Functionality:

- Simulate a user sending a message and verify the message appears in another user's chat window.
- Test group chat creation and messaging.

3. Profile Update:

- Test the user updating their profile information and verifying the changes.

Performance Tests

1. Load Testing:

- Simulate a large number of users logging in simultaneously.
- Simulate a high volume of messages being sent and received.

2. Stress Testing:

- Gradually increase the load until the system fails to determine its breaking point.

Security Tests

1. Authentication:

- Test for vulnerabilities in the login process, such as SQL injection or brute force attacks.

2. Authorization:

- Ensure users cannot access restricted routes without proper tokens.

3. Data Encryption:

- Verify that messages and sensitive data are encrypted end-to-end.

Example Test Results

Unit Test Results

Authentication Service:

- Test login method with valid credentials: Passed
- Test login method with invalid credentials: Passed
- Test token storage and retrieval: Passed

Chat Service:

- Test sendMessage method: Passed
- Test getMessages method: Passed

Integration Test Results

User Registration and Login:

- User registration flow: Passed
- User login flow: Passed

Message Sending and Receiving:

- Message sending from frontend to backend: Passed
- Real-time message updates: Passed

End-to-End Test Results

User Registration and Login Flow:

- User can register and log in successfully: Passed Chat Functionality:
- User can send and receive messages in one-on-one chat: Passed
- User can create and message in a group chat: Passed

Performance Test Results

Load Testing:

- 1000 concurrent users: Passed with average response time of 200ms
- 5000 concurrent users: Passed with average response time of 500ms Stress Testing:
- System breaking point identified at 7000 concurrent users

Security Test Results Authentication:

- SQL injection vulnerability: Not found
- Brute force attack protection: Passed Authorization:
- Unauthorized access to restricted routes: Not possible Data Encryption:
- End-to-end encryption verification: Passed

Conclusion

Conducting comprehensive testing ensures that the chat application is robust, secure, and user-friendly. Each type of test provides valuable insights into different aspects of the application, helping to identify and resolve issues before deployment. Regular testing and updates are crucial for maintaining the quality and security of the application as it evolves.

CHAPTER-8

IMPLEMENTATION

Once the system was tested, the implementation phase started. A crucial phase in the system development life cycle is successful implementation of new system design. Implementations simply mean converting new system design into operation. This is the moment of truth the first question that strikes in every one's mind that whether the system will be able to give all the desires results as expected from system. The implementation phase is concerned with user training and file conversion. The term implementation has different meanings, ranging from the conversion of a basic application to a complete replacement of computer system Implementation is used here to mean the process of converting a new or revised system design into an operational one. Conversion is one aspect of implementation. The other aspects are the post implementation review and software maintenance. There are three types of implementations:

- Implementation of a computer system to replace a manual system
- Implementation of a new computer system to replace an existing one
- Implementation of a modified application to replace an existing one.

8.1 Modules

In computer software, a module is an extension to a main program dedicated to a specific function. In programming, a module is a section of code that is added in as a whole or is designed for easy reusability

The proposed system of "Chat Application" has the following modules:

Angular Frontend Modules

1. App Module (`App Module`) • Purpose: The root module

bootstraps the application.

- Components: App Component Services: Shared services that need to be available applicationwide.
- Imports: Core Module, Shared Module, Auth Module, Chat Module, User Module, Angular Material modules, etc.

2. Core Module (`Core Module`) • Purpose: Contains singleton

services used across the entire application.

- Services: Auth Service, Socket Service, Api Service, etc.
- Providers: Global providers for interceptors, guards, and error handling.

3. Shared Module (`Shared Module`) • Purpose: Contains shared

components, directives, and pipes.

- Components: Header Component, Footer Component, Loader Component, etc.
- Directives and Pipes: Commonly used directives and pipes.
- Exports: Shared components, directives, and pipes.

4. Auth Module (`Auth Module`)

- Purpose: Manages user authentication and authorization.
- Components: Login Component, Register Component, Forgot Password
- Component Services: Auth Service (if not provided in Core Module)
- Routes: Routes related to authentication (e.g., /login, /register)

5. Chat Module (`Chat Module`)

- Purpose: Handles chat functionalities.

• Components: Chat Window Component, Chat List

• Component, Message Component, Group Chat Component

• Services: Chat Service, Message Service Routes: Routes for chat features (e.g., /chat, /chat/:id)

6. User Module (`User Module`) • Purpose: Manages user profile

and settings.

- Components: User Profile Component, User Settings Component, Contact List Component
- Services: User Service
- Routes: Routes related to user profiles and settings (e.g., /profile, /settings)

Node.js Backend Modules

1. App Module (`app.js`)

- Purpose: The entry point of the application.
- Middlewares: Global middlewares for parsing JSON, handling CORS, etc.
- Routes: Import and use different route modules.

2. Database Module (`db.js`)

- Purpose: Handles database connections and configurations.
- Libraries: Mongoose for MongoDB.

3. Auth Module (`auth`) • Purpose: Manages authentication and authorization.

- Routes: Routes for login, register, and token validation.
- Controllers: AuthController for handling auth-related logic.
- Middlewares: AuthMiddleware for protecting routes.
- Utilities: Token generation and validation.

4. User Module (`user`)

- Purpose: Manages user-related functionalities.

- Routes: Routes for user profile, user settings, and contacts.
- Controllers: User Controller for handling user logic.
- Models: User Model for JSON server.

5. Chat Module (`chat`)

- Purpose: Handles chat functionalities.
- Routes: Routes for messaging, retrieving chats, etc.
- Controllers: Chat Controller for handling chat logic.

6. Socket Module (`socket`)

- Purpose: Manages WebSocket connections and real-time communication.
- Libraries: Socket.io for WebSocket communication.
- Handlers: Event handlers for connection, disconnection, message sending, etc.

CHAPTER-9

SCREENSHOTS

9.1 JSON SERVER

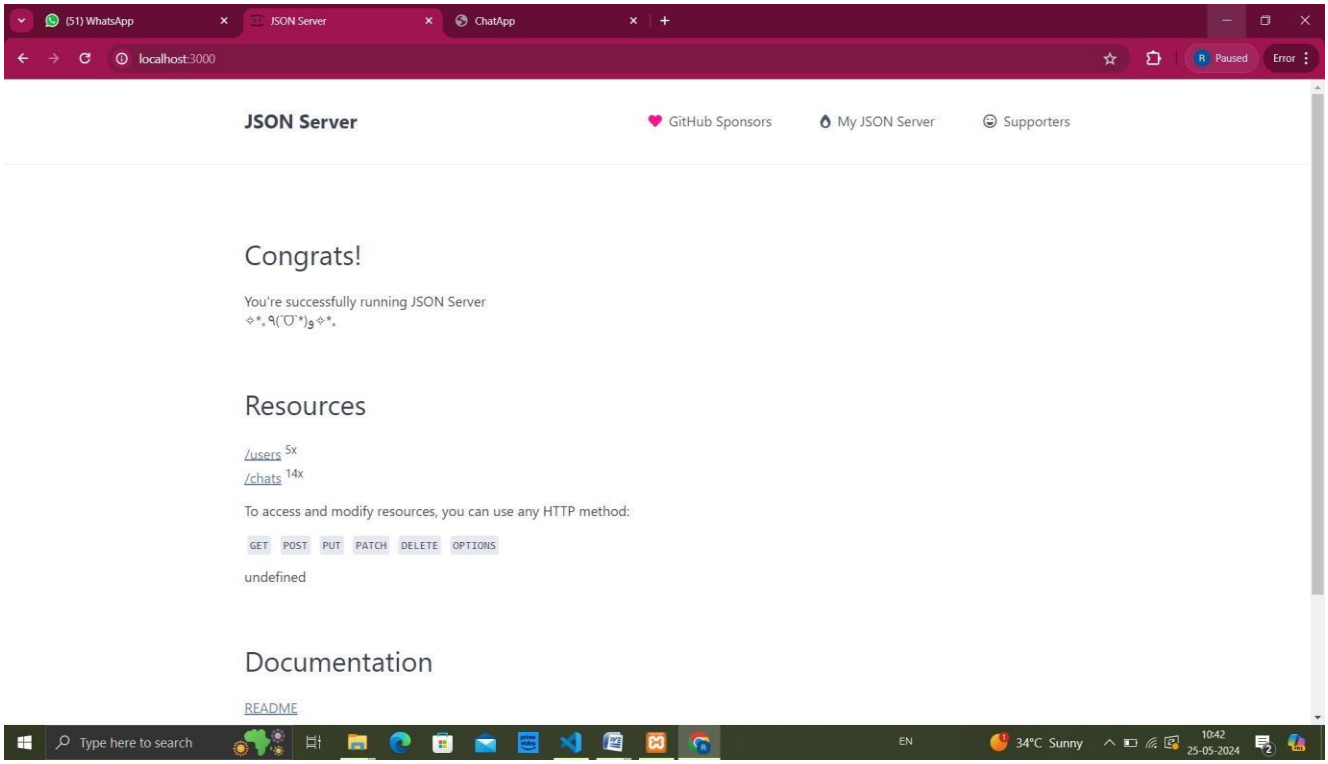

Fig 9.1 JSON Server

This page is to load all the json functionalities .

JSON Server is a popular tool for creating a simple and quick mock REST API using a JSON file. It is especially useful during the development phase of a project for testing and prototyping purposes without the need for setting up a full backend.JSON Server is a Node.js module that allows you to create a full fake REST API with zero coding required. It uses a JSON file to serve data and supports standard RESTful routes for creating, reading, updating, and deleting (CRUD) data.

9.2 Sign up page

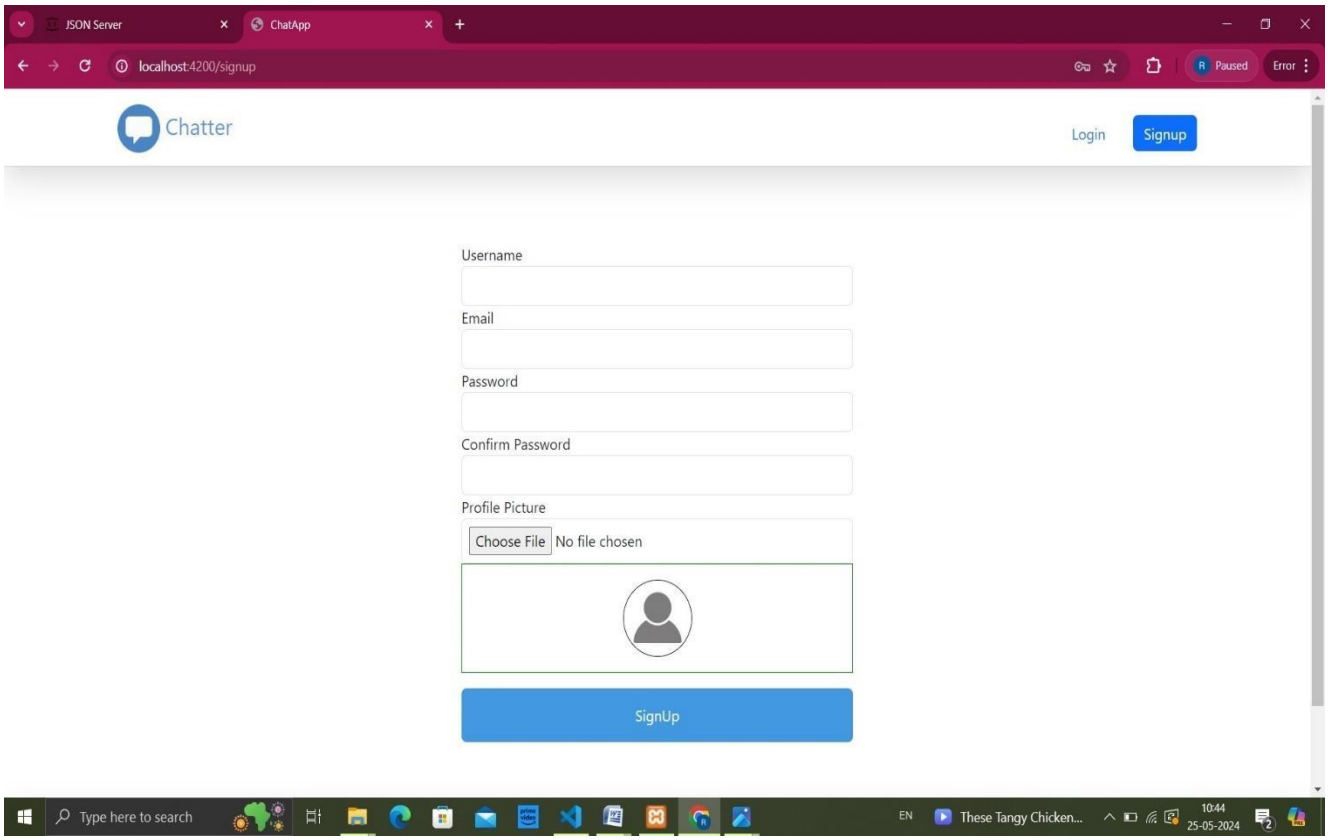

Fig 9.2 Sign up page

This is our Chatter sign up page for registering the user.

It has following parameters like username, email, password, confirm password, profile picture and click on the sign up page to register. These all details get stored in db.json file.

Components of the sign up Form:

1. Username field
2. Email field
3. Password filed
4. Confirm Password field
5. Profile picture field
6. Sign up field

9.3 Login page

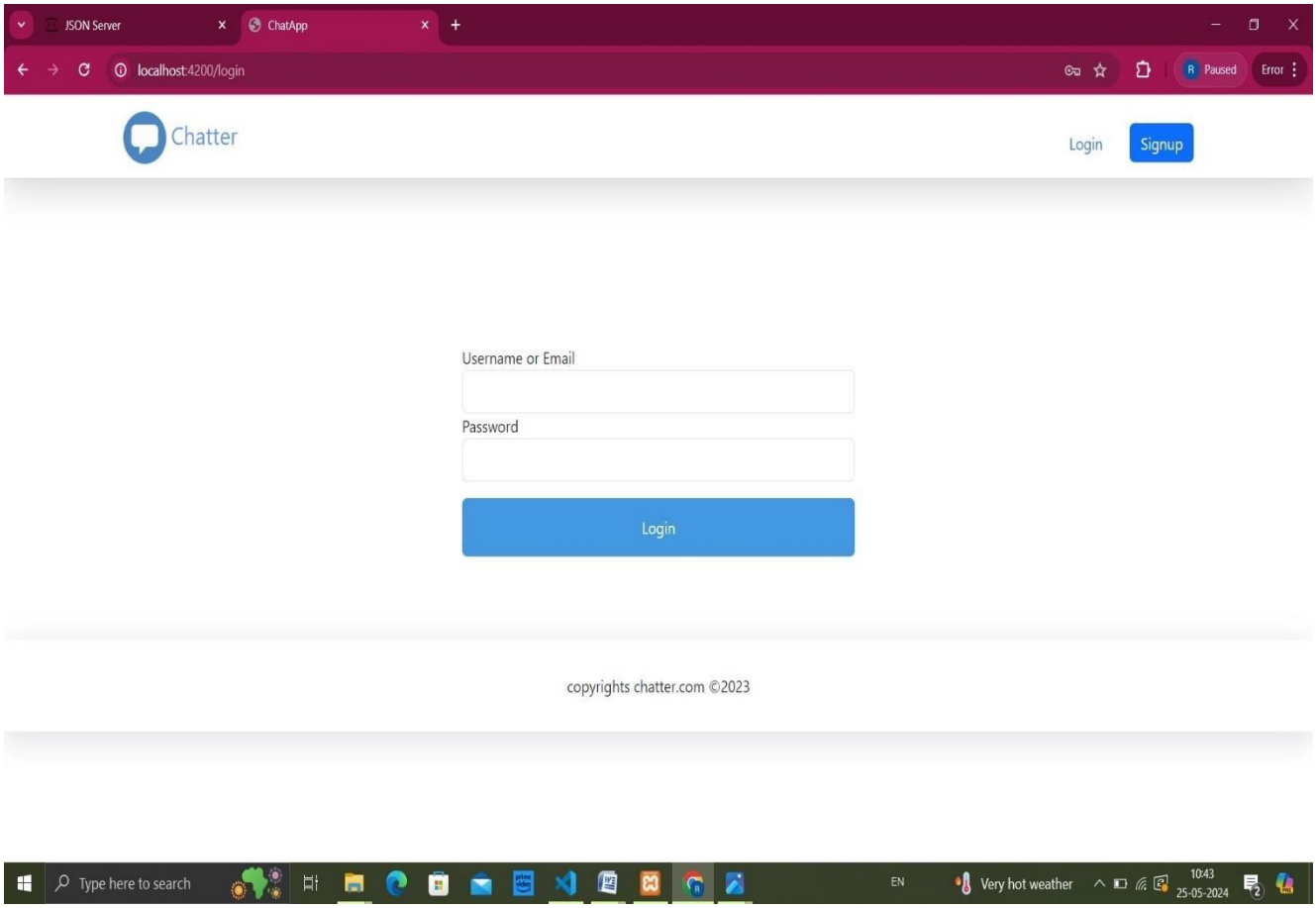

Fig 9.3 Login page

The screenshot shows a login page for a chat application called "Chatter," built using Angular. The page contains fields for entering the username or email and password, as well as a login button.

Components of the login page:

1. Username or email field
2. Password field
3. Login button

9.4 Chat window

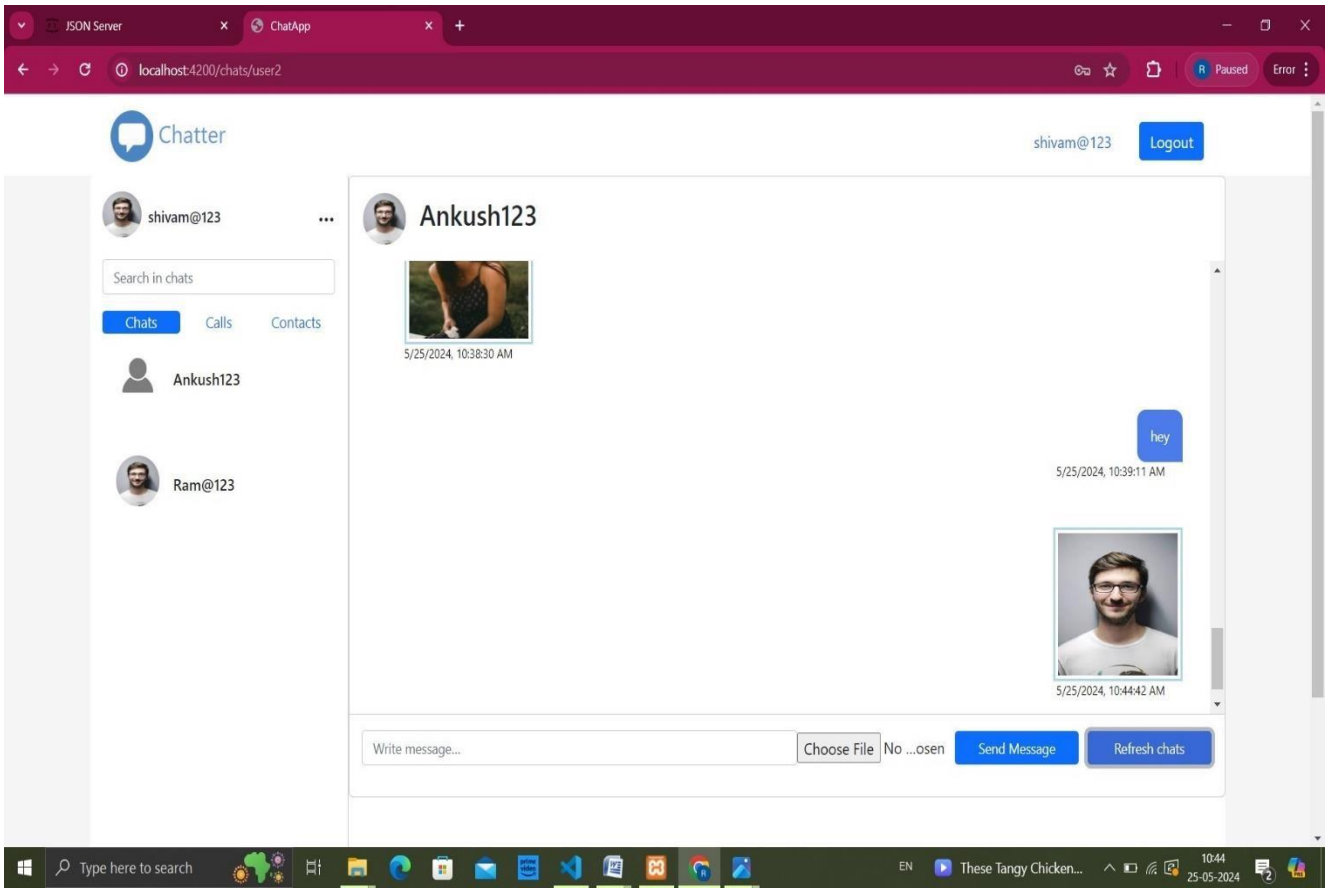

Fig 9.4 Chat Window

The screenshot shows the main chat interface of a chat application named "Chatter," built using Angular.

Components of the chat interface:

1. User profile section
2. Navigation tab
    - Chats: Lists all chat conversations.
    - Calls: Would list all recent calls, though this section is not shown in detail.
    - Contacts: Would list contacts, though this section is not shown in detail.

CONCLUSION

The chat application built using Angular and Nodejs demonstrates the powerful combination of frontend and back-end technologies to create a dynamic, real-time communication platform.

Overall, the chat application serves as a robust, user-friendly platform for real-time communication. The use of Angular CLI for the front-end and Node.js for the back-end showcases the strength of modern web development frameworks in building scalable and maintainable applications. With room for further enhancements and features, this project lays a strong foundation for a comprehensive communication tool.

The project demonstrates the integration of Angular for the front-end development and Node.js for the back-end services, creating a full-stack web application capable of real-time messaging. This combination offers several benefits in terms of performance, scalability, and maintainability.

In conclusion, the chat application built with Angular CLI and Node.js serves as a robust, scalable, and maintainable solution for real-time communication. It leverages the strengths of both technologies to provide a seamless user experience, efficient back-end processing, and secure data management. This project exemplifies the effective use of modern web development frameworks to build comprehensive applications capable of meeting current and future communication needs.

References

- https://www.w3schools.com/php/
- https://www.javatpoint.com/html-tutorial
- https://www.canva.com/graphs/er-diagrams/
- https://ijcrt.org/papers/IJCRT22A6549.pdf
- https://github.com/
- https://nodejs.org/en
- https://getstream.io/chat/angular/tutorial/
- https://www.pubnub.com/blog/building-chat-application-using-angular/

SOURCE CODE

Db.json


```
{
  "users": [
    {
      "id": 1,
      "username": "ankush123",
      "password": "Ankush@123",
      "email": "ankushpanchal144@gmail.com",
      "profile": "../../assets/images/profile1.jpg",
      "mobile": "+91 9636933013",
      "friends": [
        "2",
        "3"
      ]
    },
    {
      "id": 2,
      "username": "shivam@123",
      "password": "shivam@123",
      "email": "shivam123@gmail.com",
      "profile": "../../assets/images/profile2.jpg",
      "mobile": "+91 9636933012",
      "friends": [
        "1",
        "3"
      ]
    },
    {
      "id": 3,
      "username": "ram@123",
      "password": "ram@123",
      "email": "ram123@gmail.com",
      "profile": "../../assets/images/profile2.jpg",
      "mobile": "+91 9636933013",
      "friends": [
        "1",
        "2"
      ]
    },
```

```
    {
      "username": "riya_mathur",
      "email": "riyamathur121@gmail.com",
      "password": "Riya@123",
      "cpassword": "Riya@123",
      "profile": "../../assets/images/profile3.webp",
      "mobile": "+91 9636933014",
      "chats": {},
      "id": 4
    },
    {
      "username": "riya97948@gmail.com",
      "email": "riya97948@gmail.com",
      "password": "Riyaa@123",
      "cpassword": "Riyaa@123",
      "profile": "../../assets/images/151.jpg",
      "chats": {},
      "id": 5
    }
  ],
  "chats": [
    {
      "id": 1,
      "sender": "1",
      "receiver": "2",
      "timestamp": "11/01/2023, 11:00:37 PM",
      "message": "Hi",
      "type": "msg"
    },
    {
      "id": 2,
      "sender": "2",
      "receiver": "1",
      "timestamp": "11/01/2023, 11:01:00 AM",
      "message": "Hi,How are you?",
      "type": "msg"
    },
    {
      "id": 3,
      "sender": "1",
      "receiver": "2",
      "timestamp": "11/01/2023, 11:01:37 PM",
      "message": "I am fine! What about you?",
      "type": "msg"
```

```
},
{
  "id": 4,
  "sender": "2",
  "receiver": "1",
  "timestamp": "11/01/2023, 11:02:00 PM",
  "message": "I am also fine",
  "type": "msg"
},
{
  "id": 5,
  "sender": "1",
  "receiver": "3",
  "timestamp": "12/01/2023, 11:00:37 PM",
  "message": "Hi,kese ho?",
  "type": "msg"
},
{
  "id": 6,
  "sender": "3",
  "receiver": "1",
  "timestamp": "12/01/2023, 11:01:00 PM",
  "message": "Hi,I am Majama,How are you?",
  "type": "msg"
},
{
  "id": 7,
  "sender": "1",
  "receiver": "3",
  "timestamp": "12/01/2023, 11:01:37 PM",
  "message": "ekdum majama",
  "type": "msg"
},
{
  "id": 8,
  "sender": "3",
  "receiver": "1",
  "timestamp": "12/01/2023, 11:02:00 PM",
  "message": "OK",
  "type": "msg"
},
{
  "sender": 1,
  "receiver": "2",
```

```
  "timestamp": "5/15/2023, 5:26:50 PM",
  "message": "../../assets/images/profile2.jpg",
  "id": 29,
  "type": "image"
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 5:30:06 PM",
  "message": "123213",
  "type": "msg",
  "id": 30
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 5:30:12 PM",
  "message": "Hello",
  "type": "msg",
  "id": 31
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 5:30:26 PM",
  "message": "../../assets/images/profile1.jpg",
  "type": "image",
  "id": 32
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 5:40:08 PM",
  "message": "../../assets/images/profile2.jpg",
  "type": "image",
  "id": 33
},
{
  "sender": 1,
  "receiver": 3,
  "timestamp": "5/15/2023, 5:50:49 PM",
  "message": "../../assets/images/profile1.jpg",
  "type": "image",
  "id": 34
```

```
},
{
  "sender": 1,
  "receiver": 3,
  "timestamp": "5/15/2023, 5:54:16 PM",
  "message": "so this my photo",
  "type": "msg",
  "id": 35
},
{
  "sender": 1,
  "receiver": 3,
  "timestamp": "5/15/2023, 5:54:37 PM",
  "message": "\t🤪 ",
  "type": "msg",
  "id": 36
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 5:58:20 PM",
  "message": "../../assets/images/profile3.webp",
  "type": "image",
  "id": 37
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 5:59:21 PM",
  "message": "../../assets/images/profile1.jpg",
  "type": "image",
  "id": 38
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 5:59:37 PM",
  "message": "../../assets/images/profile3.webp",
  "type": "image",
  "id": 39
},
{
  "sender": 1,
  "receiver": 3,
```

```
  "timestamp": "5/15/2023, 5:59:51 PM",
  "message": "../../assets/images/profile1.jpg",
  "type": "image",
  "id": 40
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 6:19:46 PM",
  "message": "../../assets/images/profile1.jpg",
  "type": "image",
  "id": 41
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/15/2023, 6:46:46 PM",
  "message": "../../assets/images/profile3.webp",
  "type": "image",
  "id": 42
},
{
  "sender": 1,
  "receiver": 2,
  "timestamp": "5/16/2023, 11:37:11 AM",
  "message": "../../assets/images/profile1.jpg",
  "type": "image",
  "id": 43
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:02:53 AM",
  "message": "ok",
  "type": "msg",
  "id": 44
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:02:59 AM",
  "message": "yeah",
  "type": "msg",
  "id": 45
```

```
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:08:17 AM",
  "message": "121212",
  "type": "msg",
  "id": 46
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:10:05 AM",
  "message": "121212",
  "type": "msg",
  "id": 47
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:10:09 AM",
  "message": "asas",
  "type": "msg",
  "id": 48
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:12:45 AM",
  "message": "hii",
  "type": "msg",
  "id": 49
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:21:31 AM",
  "message": "we",
  "type": "msg",
  "id": 50
},
{
  "sender": 1,
  "receiver": "2",
```

```
  "timestamp": "5/17/2023, 10:21:43 AM",
  "message": "we",
  "type": "msg",
  "id": 51
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:23:04 AM",
  "message": "that is funny",
  "type": "msg",
  "id": 52
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:24:04 AM",
  "message": "werew",
  "type": "msg",
  "id": 53
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:24:09 AM",
  "message": "tere",
  "type": "msg",
  "id": 54
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:24:17 AM",
  "message": "asdsadsad",
  "type": "msg",
  "id": 55
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:24:21 AM",
  "message": "asdadadsdasdsd",
  "type": "msg",
  "id": 56
```

```
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:24:27 AM",
  "message": "asdfsafsfsafasfsadf",
  "type": "msg",
  "id": 57
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:24:33 AM",
  "message": "asfsadfasfdasf",
  "type": "msg",
  "id": 58
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:24:49 AM",
  "message": "yeah that is fine but ",
  "type": "msg",
  "id": 59
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:27:43 AM",
  "message": "qwqwqw",
  "type": "msg",
  "id": 60
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:28:26 AM",
  "message": "yeah",
  "type": "msg",
  "id": 61
},
{
  "sender": 1,
  "receiver": "2",
```

```
  "timestamp": "5/17/2023, 10:29:23 AM",
  "message": "we are",
  "type": "msg",
  "id": 62
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:29:29 AM",
  "message": "we are",
  "type": "msg",
  "id": 63
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:29:33 AM",
  "message": "we are",
  "type": "msg",
  "id": 64
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:29:40 AM",
  "message": "we are",
  "type": "msg",
  "id": 65
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:30:03 AM",
  "message": "hii",
  "type": "msg",
  "id": 66
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:30:12 AM",
  "message": "how's life",
  "type": "msg",
  "id": 67
```

```
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:30:50 AM",
  "message": "../../assets/images/profile3.webp",
  "type": "image",
  "id": 68
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:37:24 AM",
  "message": "../../assets/images/profile3.webp",
  "type": "image",
  "id": 69
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 10:37:59 AM",
  "message": "hey what is going on ?",
  "type": "msg",
  "id": 70
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 2:06:52 PM",
  "message": "what is going on ?",
  "type": "msg",
  "id": 71
},
{
  "sender": 1,
  "receiver": "2",
  "timestamp": "5/17/2023, 2:08:24 PM",
  "message": "../../assets/images/profile3.webp",
  "type": "image",
  "id": 72
},
{
  "sender": 2,
  "receiver": "1",
```

```
  "timestamp": "14/5/2024, 11:59:34 am",
  "message": "hi",
  "type": "msg",
  "id": 73
},
{
  "sender": 2,
  "receiver": "1",
  "timestamp": "14/5/2024, 11:59:48 am",
  "message": "hiii",
  "type": "msg",
  "id": 74
},
{
  "sender": 2,
  "receiver": 3,
  "timestamp": "14/5/2024, 12:00:43 pm",
  "message": "hlo",
  "type": "msg",
  "id": 75
},
{
  "sender": 2,
  "receiver": 1,
  "timestamp": "14/5/2024, 12:03:17 pm",
  "message": "hlo",
  "type": "msg",
  "id": 76
},
{
  "sender": 5,
  "timestamp": "14/5/2024, 12:08:29 pm",
  "message": "hlo",
  "type": "msg",
  "id": 77
},
{
  "sender": 5,
  "timestamp": "14/5/2024, 12:16:14 pm",
  "message": "hello",
  "type": "msg",
  "id": 78
},
{
```

```
      "sender": 2,
      "receiver": "1",
      "timestamp": "14/5/2024, 12:20:31 pm",
      "message": "hi",
      "type": "msg",
      "id": 79
    },
    {
      "sender": 5,
      "timestamp": "25/5/2024, 11:22:56 am",
      "message": "hi",
      "type": "msg",
      "id": 80
    }
  ]
}
```

Index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>ChatApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Tsconfig.json

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
```

```
    "outDir": "./dist/out-tsc",
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "noImplicitOverride": true,
    "noPropertyAccessFromIndexSignature": true,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": true,
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "moduleResolution": "node",
    "importHelpers": true,
    "target": "ES2022",
    "module": "ES2022",
    "useDefineForClassFields": false,
    "lib": [
      "ES2022",
      "dom"
    ]
  },
  "angularCompilerOptions": {
    "enableI18nLegacyMessageIdFormat": false,
    "strictInjectionParameters": true,
    "strictInputAccessModifiers": true,
    "strictTemplates": true
  }
}
```

Tsconfig.spec.json

```
{
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "outDir": "./out-tsc/spec",
    "types": [
      "jasmine",
      "@angular/localize"
    ]
  },
  "include": [
```

```
    "src/**/*.spec.ts",
    "src/**/*.d.ts"
  ]
}
```