

# **REALTIME CHAT APPLICATION**

## **A PROJECT REPORT**

*Submitted by :*

Abhik Gupta (09514902021)

John P Varghese (04714902021)

Yatin Hooda (02914902021)

Nishchay Chandok (35914902021)

*In partial fulfillment for the award of the degree  
of*

## **BACHELOR OF COMPUTER APPLICATION**



## **MAHARAJA SURAJMAL INSTITUTE**

**C4, JANAKPURI, NEW DELHI – 110058**

**DECEMBER 2023**

# DECLARATION

We affirm that this submission is the result of our individual effort, and we attest that, to the best of our knowledge and belief, it does not incorporate any material previously published or authored by another individual. Furthermore, we confirm that this work has not been utilized in whole or in part for the completion of any other academic degree or diploma from any university or educational institution. Any instances where external contributions have been acknowledged are duly cited within the text.

**Name:** Abhik Gupta

**Enrollment Number:** 09514902021

**Signature:**

**Date:**

**Name:** Yatin Hooda

**Enrollment Number:** 02914902021

**Signature:**

**Date:**

**Name:** John P Varghese

**Enrollment Number:** 04714902021

**Signature:**

**Date:**

**Name:** Nishchay Chandok

**Enrollment Number:** 35914902021

**Signature:**

**Date:**

# **BONAFIDE CERTIFICATE**

I certify that the project report titled "**REALTIME CHAT APPLICATION**" is a bonafide, authentic and genuine work and hard work of the team comprising “Abhik Gupta, John P Varghese, Yatin Hooda, and Nishchay Chandok”. This team diligently conducted the project under my supervision, and I can attest to the integrity and originality of their work.

## **SUPERVISOR**

Mr. Sundeep Kumar

Assistant professor

Dept. of Computer Application

**SIGNATURE OF SUPERVISOR**

# SELF CERTIFICATE

This is to certify that the dissertation/project report entitled “**REALTIME CHAT APPLICATION**” is done by me is an authentic work carried out for the partial fulfilment of the requirements for the award of the degree of Bachelor of Computer Applications under the guidance of **Mr. Sundeep Kumar**. The matter embodied in this project work has not been submitted earlier for award of any degree or diploma to the best of my knowledge and belief.

ABHIK GUPTA

09514902021

**(TEAM LEADER)**

# ACKNOWLEDGEMENT

Certified We would like to take this opportunity to acknowledge everyone who has helped us in every stage of this project.

We extend our heartfelt appreciation to all those who contributed to the successful realization of this project. A special acknowledgment goes to **Mr. Sundeep Kumar** from the **Department of Computer Science** for his invaluable guidance and insightful suggestions, which played a pivotal role in bringing this project to fruition. We are sincerely grateful for his unwavering support.

Furthermore, our sincere thanks go to our esteemed **Director, Dr. Harish Singh**, whose provision of essential facilities significantly contributed to the seamless completion of this project. We express our gratitude for his support and encouragement throughout the entire process.

ABHIK GUPTA

JOHN P VARGHESE

YATIN HOODA

NISHCHAY CHANDOK

**BCA 5A (MORNING)**

# TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NUMBER
1.	Synopsis <ul style="list-style-type: none"><li>• Objective</li><li>• Scope</li><li>• What is Convoverse?</li><li>• Why did we choose Convoverse?</li></ul>	1 - 3
2.	Requirements <ul style="list-style-type: none"><li>• Hardware Requirements</li><li>• Software Requirements</li></ul>	4-5
3.	Abstract of the project.	6
4.	Working of the application	7 - 8
5.	Tech Stack being used <ul style="list-style-type: none"><li>• Frontend</li><li>• Backend</li></ul>	9 - 15
6.	SRS <ul style="list-style-type: none"><li>• Functional Requirements</li><li>• Interface Requirements</li><li>• Performance Requirements</li><li>• Design Constraints</li><li>• Non-Functional Attributes</li><li>• Schedule and Plan</li></ul>	16 - 19

<b>7.</b>	Software Design <ul style="list-style-type: none"><li>• Level-0 DFD of project</li><li>• Level-1 DFD of project</li><li>• Use Case Diagram of project</li><li>• ER Diagram of database structure</li></ul>	20 - 28
<b>8.</b>	Team Description	29
<b>9.</b>	Coding and Implementation	30 - 124
<b>10.</b>	Output Screenshots	125 - 135
<b>11.</b>	Version Control and Collaboration	136 - 139
<b>12.</b>	Conclusion	140 – 141
<b>13.</b>	Future Scope	142 – 143
<b>14.</b>	References	144

# **SYNOPSIS**

## **OBJECTIVE**

Convoverse aims to revolutionize online communication by providing a feature-rich and real-time chat experience built on the MERN (MongoDB, Express.js, React.js, Node.js) stack. The primary objective is to offer users a platform that transcends traditional messaging applications, prioritizing not only the immediacy of communication but also the versatility and depth of interactions. With a focus on user friendly design and seamless navigation, Convoverse aims to facilitate both professional collaborations and personal connections. The application seeks to enhance the way individuals engage in conversations, providing a dynamic space that adapts to diverse communication needs. Through the MERN stack's robust capabilities, Convoverse aspires to set a new standard for online dialogue, ensuring a responsive and engaging experience for users across various contexts.

## **SCOPE**

Convoverse's scope extends beyond mere text-based conversations, encompassing a broad spectrum of communication methods within its MERN-based architecture. From real-time multimedia sharing to the integration of collaborative tools, the application aims to be a comprehensive hub for diverse communication needs. The MERN stack's flexibility allows Convoverse to scale efficiently, accommodating an expanding user base while maintaining optimal performance. The scope also includes a commitment to continuous improvement, with plans for regular updates and the integration of emerging technologies to enhance user experience. Convoverse envisions becoming a go-to platform for individuals seeking a modern,



responsive, and feature-rich chat application, solidifying its place as a dynamic solution within the evolving landscape of online communication.

## **SUMMARY**

Convoverse is a groundbreaking real-time chat web application developed on the MERN (MongoDB, Express.js, React.js, Node.js) stack, aiming to redefine online communication. With a user-friendly interface and cutting-edge features, Convoverse seamlessly connects individuals globally, fostering dynamic conversations and facilitating instant connections. The application prioritizes a smooth and responsive user experience, accommodating both professional collaborations and personal connections. Leveraging the power of MongoDB for efficient data storage, Mongoose for streamlined interactions, and Socket.io for real-time communication, Convoverse introduces a new era of interactive and secure online dialogue. The project's comprehensive hardware and software requirements ensure optimal performance, scalability, and user accessibility, making Convoverse a versatile and robust platform for diverse communication needs.

## **WHAT IS CONVOVERSE ?**

Convoverse is a cutting-edge real-time chat web application designed to redefine the way people connect and communicate online. With its sleek interface and robust features, Convoverse offers users a seamless platform to engage in dynamic conversations instantaneously. This application goes beyond conventional messaging, providing a space where users can effortlessly navigate between group chats and private discussions, fostering both professional collaborations and personal connections. Convoverse is characterized by its

user-friendly design, ensuring a smooth and enjoyable experience for individuals looking to connect with others in real time. Whether it's for work, socializing, or staying in touch with loved ones, Convoverse introduces a new era of interactive and responsive online communication.

## **WHY DID WE CHOOSE CONVOVERSE?**

The choice of developing Convoverse on the MERN stack was driven by a strategic blend of efficiency, flexibility, and scalability. The MERN stack, comprising MongoDB for database management, Express.js for server-side development, React.js for the user interface, and Node.js for server-side execution, presented an ideal combination for creating a responsive and dynamic real-time chat application. MongoDB's NoSQL database structure accommodates the varied data formats inherent in chat applications, while Express.js streamlines server-side development. React.js, renowned for its component-based architecture, ensures a smooth and interactive user experience, and Node.js provides a non-blocking, event-driven architecture, enabling scalability to handle concurrent connections efficiently. This comprehensive synergy within the MERN stack aligns seamlessly with Convoverse's vision to deliver a cutting-edge chat platform that not only meets current demands but is also primed for future enhancements and technological evolution.

# REQUIREMENTS

## HARDWARE REQUIREMENTS

1. Processor: Dual-core processor or higher to ensure smooth execution of Convoverse's real-time features.
2. RAM: 4 GB or higher for optimal performance, allowing the application to handle concurrent user interactions efficiently.
3. Storage: At least 20 GB of free disk space to accommodate data storage and ensure the seamless operation of Convoverse.
4. Network: A stable internet connection with sufficient bandwidth for real-time communication, enabling users to experience uninterrupted and responsive interactions.
5. Web Hosting: A reliable web hosting service capable of supporting Node.js applications, ensuring server accessibility to users and providing a solid foundation for hosting Convoverse.
6. SSL Certificate: An SSL certificate for secure, encrypted data transmission, enhancing Convoverse's overall security and ensuring user privacy during online interactions.

## SOFTWARE REQUIREMENTS

1. Operating System: Compatible with Windows 7 or later, macOS, or Linux distributions, ensuring flexibility for a diverse user base.
2. Web Browser: The latest versions of popular browsers such as Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge for optimal user experience and compatibility.

3. Node.js: The latest version of Node.js is essential for server-side JavaScript execution, powering the backend of Convoverse.
4. MongoDB: A running instance of MongoDB, serving as the database for Convoverse, providing a scalable and efficient data storage solution.
5. MongoDB Atlas: A cloud-based database service for MongoDB, ensuring scalable and secure database hosting, aligning with Convoverse's growth requirements.
6. Mongoose: An ODM library facilitating interactions between Convoverse and the MongoDB database, enhancing data modeling and management. Mainly used for creating schemes and models for out databases.
7. Express.js: The backend server framework for handling HTTP requests and responses, enabling the seamless operation of Convoverse's server-side functionalities.
8. React.js: The front-end library for building modular and efficient user interfaces, contributing to Convoverse's user-friendly design.
9. Node Package Manager (NPM): Used to manage and install project dependencies, ensuring the smooth integration of libraries and modules.
10. Text Editor/IDE: Any preferred text editor or integrated development environment (IDE), such as Visual Studio Code or Atom, for code development and maintenance.
11. Testing: Thoroughly test your website to ensure it functions as expected. Manual testing is sufficient for our project.

# ABSTRACT

Convoverse stands as a sophisticated web chatting application meticulously crafted on the MERN (MongoDB, Express.js, React, Node.js) stack, seamlessly merging frontend and backend technologies. This innovative platform offers users a contemporary communication experience, enabling effortless registration and login through phone numbers. The application boasts an interactive and visually appealing user interface, ensuring an engaging space for diverse conversations. Convoverse's scope extends beyond traditional chat applications by leveraging the power of the MERN stack, providing a robust and scalable foundation for potential future enhancements. The intuitive UI, developed in React, facilitates a seamless user experience, while the backend, powered by Node.js and Express.js, efficiently manages data flow and interactions with MongoDB.

In an era where digital communication has become an integral part of daily life, the need for a sophisticated chatting application like Convoverse is paramount. This application addresses the growing demand for seamless, secure, and user-friendly platforms for interpersonal communication. The ability to register and log in using phone numbers enhances accessibility, catering to a diverse user base. Convoverse not only meets the contemporary requirements of instant communication but also anticipates future needs by incorporating scalable technologies. Whether for personal or professional use, the demand for an interactive and feature-rich chatting application is ever-present, and Convoverse rises to the occasion by offering a comprehensive solution that aligns with the evolving dynamics of modern communication.

# WORKING OF THE APPLICATION

1. Convoverse operates on a user-centric model, starting with a seamless sign-up process.
2. Users are required to provide essential details such as their name, phone number, a secure password, and a URL to their profile picture.
3. Once successfully registered, users gain access to the platform's login page, where they can enter their phone number and password to log in securely.
4. Upon logging in, the user interface is intuitively designed.
5. The left portion features a contact list, providing an overview of the user's connections. The list displays the name, display pic and last text sent from either side.
6. A search box is also prominently displayed, allowing the logged-in user to quickly search for a contact by entering their phone number. This feature simplifies the initiation of new chats, especially with individuals already registered on Convoverse.
7. Clicking on a contact in the list opens up the right half of the interface, dedicated to the chat functionality. This ultimately creates a channel between 2 users where they can message each other. Here, users can engage in real-time conversations, send and receive messages using the input box at the bottom of the screen.
8. The top of the right half displays the selected person's profile picture and name, providing context and personalization to the chat experience.
9. The chat interface is enriched with a user-friendly emoji picker, enhancing the expressiveness of the conversations.

10. To maintain a secure and user-controlled environment, Convoverse offers a straightforward logout option. The logout button is conveniently located in the left half of the interface, positioned above the contact list and the search box, ensuring that users can easily disconnect from their accounts whenever needed.
11. This well-organized and user-friendly interface design ensures a smooth and enjoyable experience for Convoverse users throughout their interactions on the platform.

# TECH STACK USED FOR DEVELOPMENT

## PROJECT STRUCTURE:

### 1. Client-Side (Frontend):

- a. Developed using React, React Router DOM, Styled Components, and various UI libraries for enhanced functionality and user experience.

### 2. Server-Side (Backend):

- a. Built on Node.js and Express, facilitating the creation of a scalable and efficient server.
- b. Utilizes MongoDB as the database, with Mongoose for data modeling and interactions.
- c. Employs Babel for ECMAScript compatibility and provides additional middleware for features like compression, CORS, and validation.

### 3. Note:

- a. The technology stack combines the strengths of React for the dynamic frontend, Node.js and Express for the robust backend, MongoDB for flexible and scalable data storage, and various libraries and tools to enhance functionality, security, and development efficiency. This stack ensures a modern, efficient, and scalable architecture for the Convoverse project.



## **FRONTEND (CLIENT):**

### **1. React:**

- a. Version: ^18.2.0
- b. Description: React is a JavaScript library for building user interfaces. It facilitates the creation of interactive and dynamic components, ensuring a seamless user experience.

### **2. React Router DOM:**

- a. Version: ^6.20.0
- b. Description: React Router DOM enables navigation and routing in React applications, allowing for the creation of single-page applications with multiple views.

### **3. Styled Components:**

- a. Version: ^6.0.8
- b. Description: Styled Components is a CSS-in-JS library that enables styling React components with tagged template literals. It promotes the creation of maintainable and reusable styles.

### **4. Emoji Picker React:**

- a. Version: ^4.5.2
- b. Description: Emoji Picker React is a library for adding emoji functionality to React applications, enhancing the expressive nature of conversations.

## **5. React Toastify:**

- a. Version: ^9.1.3
- b. Description: React Toastify provides a simple and customizable way to display toast notifications in React applications, enhancing user feedback.

## **6. React Icons:**

- a. Version: ^4.11.0
- b. Description: React Icons provides a collection of popular icons as React components, facilitating the integration of visually appealing icons in the application.

## **7. Web Vitals:**

- a. Version: ^2.1.4
- b. Description: Web Vitals is a library for tracking essential performance metrics, allowing developers to monitor and improve the user experience.

## **8. @LeeCheuk/React-Google-Login:**

- a. Version: ^5.4.1
- b. Description: This library provides a convenient way to integrate Google login functionality into React applications, facilitating user authentication.

## **BACKEND (SERVER):**

### **1. Node.js:**

- a. Version: (Not specified)
- b. Description: Node.js is a runtime environment that allows the execution of JavaScript on the server-side. It provides a non-blocking, event-driven architecture, making it well-suited for scalable and real-time applications.

### **2. Express:**

- a. Version: ^4.18.2
- b. Description: Express is a minimal and flexible Node.js web application framework that facilitates the creation of robust APIs. It simplifies the handling of HTTP requests and responses.

### **3. MongoDB:**

- a. Version: ^7.6.2
- b. Description: MongoDB is a NoSQL database that stores data in JSON-like documents. It provides flexibility and scalability for data storage in the project.

#### **4. Mongoose:**

- a. Version: ^7.6.2
- b. Description: Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js. It simplifies interactions with MongoDB by providing a schema-based solution.

#### **5. Babel:**

- a. Version: ^6.23.0
- b. Description: Babel is a JavaScript compiler that enables the use of next-generation JavaScript features in the project by transforming code to a backward-compatible version.

#### **6. Yup:**

- a. Version: ^1.3.2
- b. Description: Yup is a schema validation library for JavaScript objects. It is commonly used for form validation in applications.

#### **7. Cors:**

- a. Version: ^2.8.5
- b. Description: CORS (Cross-Origin Resource Sharing) middleware for Express, facilitating secure communication between the frontend and backend.

## **8. Moment:**

- a. Version: ^2.29.4
- b. Description: Moment is a JavaScript library for parsing, validating, manipulating, and formatting dates. It simplifies date and time handling in the application.

## **9. Compression:**

- a. Version: ^1.7.4
- b. Description: Compression middleware for Express, which compresses HTTP responses, reducing response time and improving overall application performance.

## **GENERAL DEPENDENCIES:**

### **1. ESLint:**

- a. ESLintConfig: React-app, React-app/jest
- b. Description: ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code. The specified configurations enforce consistent coding styles and practices.

## **2. Browserslist:**

- a. Production: ">0.2%", "not dead", "not op\_mini all"
- b. Development: "last 1 chrome version", "last 1 firefox version", "last 1 safari version"
- c. Description: Browserslist is a tool used to share target browsers and Node.js versions between different front-end tools. It ensures consistency in browser support.

## **3. DevDependencies:**

- a. @babel/core: ^7.23.2, @babel/node: ^7.22.19
- b. Description: Additional development dependencies, including Babel core and Babel node, to support ECMAScript features and Node.js server-side scripting during development.

# SOFTWARE REQUIREMENT SPECIFICATIONS

## (SRS)

### Functional Requirements:

1. User Registration:
  - a. Define fields for user registration: phone number, name, password, and profile picture URL.
  - b. Implement a mechanism to validate the uniqueness of phone numbers during registration.
2. User Authentication:
  - a. Develop a secure login mechanism using phone numbers and passwords.
  - b. Ensure robust authentication protocols to protect user accounts.
3. Contact Management:
  - a. Create a visually appealing contact list for users upon login.
  - b. Implement a search box for users to easily find and connect with contacts.
4. Chat Functionality:
  - a. Enable real-time messaging functionality between users.
  - b. Integrate an emoji picker to enhance the expressive nature of conversations.
  - c. Implement a seamless process for initiating new chats by selecting contacts from the list.
5. User Profile Display:
  - a. Design a layout to display the profile picture and name of the selected contact during a chat.

6. Logout Feature:

- a. Develop a secure logout mechanism that ensures user account protection.

## **Interface Requirements:**

1. Intuitive UI Design:

- a. Design a user-friendly interface with clear navigation between the contact list, search box, and chat interface.
- b. Ensure a visually cohesive and intuitive layout to enhance the overall user experience.

2. Responsive Design:

- a. Optimize the application for responsiveness across various devices, ensuring a consistent and user-friendly experience on desktops, tablets, and mobile devices.

3. Profile Picture Display:

- a. Implement an effective display of profile pictures and names during chats, providing context and personalization.

## **Performance Requirements:**

1. Real-time Messaging:

- a. Optimize the messaging system for low latency, ensuring real-time communication between users.

2. Scalability:



- a. Design the application to handle a growing user base and increasing message loads by implementing scalable architecture and efficient data management.
- 3. Search Efficiency:
  - a. Develop an efficient algorithm for the contact search functionality, ensuring quick and accurate results.

### **Design Constraints:**

- 1. Browser Compatibility:
  - a. Ensure compatibility with popular web browsers, such as Chrome, Firefox, Safari, and Edge.
- 2. Data Security:
  - a. Implement robust security measures, including encryption and secure authentication, to protect user data and maintain privacy.

### **Non-functional Attributes:**

- 1. Security:
  - a. Implement SSL encryption for secure data transmission.
  - b. Ensure secure user authentication protocols to prevent unauthorized access.
- 2. User Experience:
  - a. Focus on an intuitive design that provides a positive and engaging user experience, promoting user satisfaction and retention.

## **Project Plan:**

1. Project Phases:
  - a. Define clear phases, including Planning, Design, Implementation, Testing, and Deployment.
2. Timeline:
  - a. Allocate realistic timeframes for each phase, considering the complexity of the tasks and potential challenges.
3. Testing:
  - a. Conduct thorough testing to ensure the functionality, security, and performance of the application.
4. Deployment:
  - a. Plan a gradual rollout to monitor and address any issues, ensuring a smooth launch.
5. Maintenance:
  - a. Establish a plan for ongoing updates, maintenance, and support post-deployment to address user feedback and evolving needs.

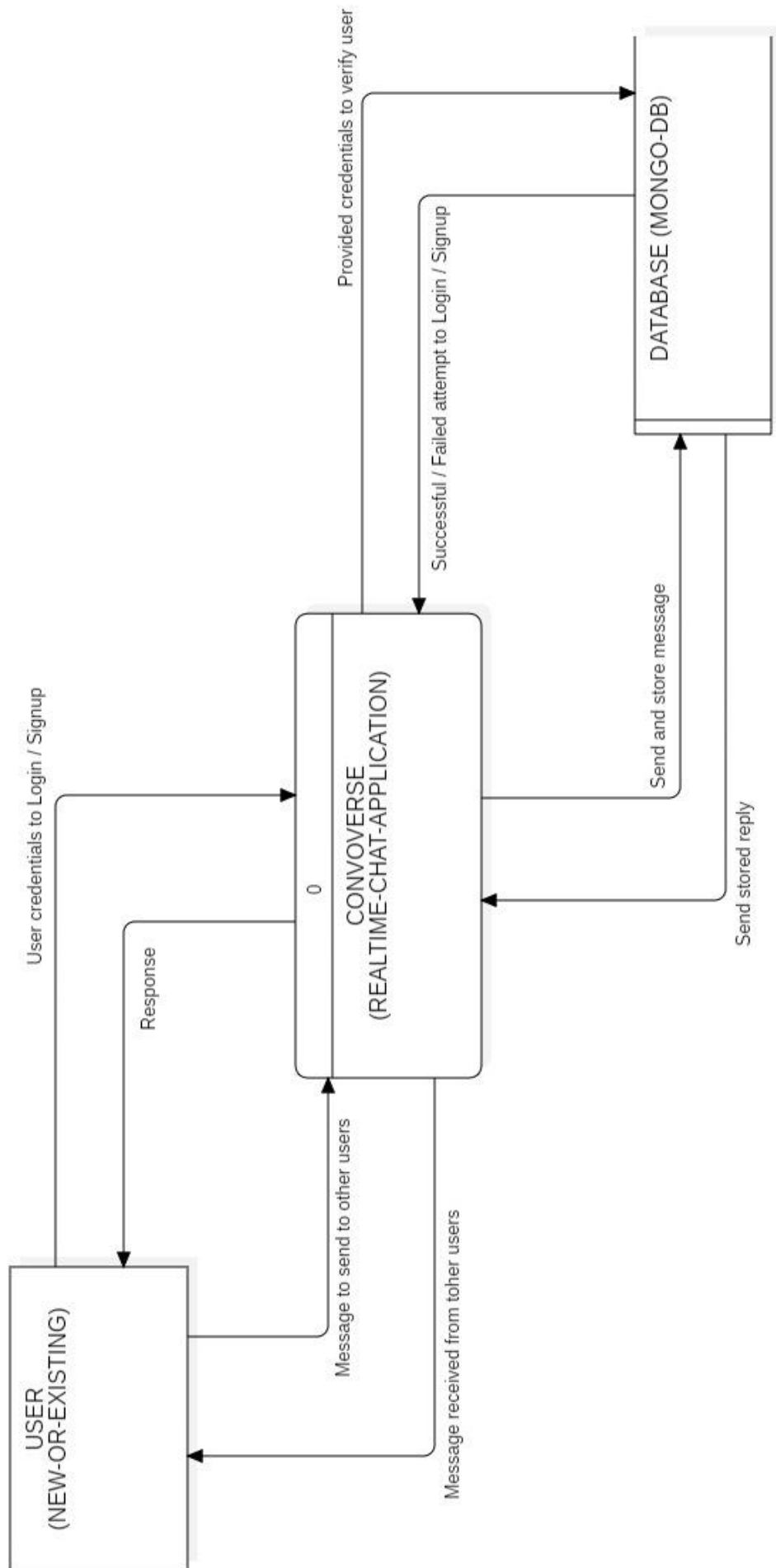
## **Project Schedule:**

1. Planning and Requirement Gathering: 1-2 weeks
2. Designing and developing frontend: 2 weeks
3. Designing and developing backend: 2 weeks
4. Integrating frontend and backend: 1 week
5. Updates and maintenance: Ongoing

# SOFTWARE DESIGN

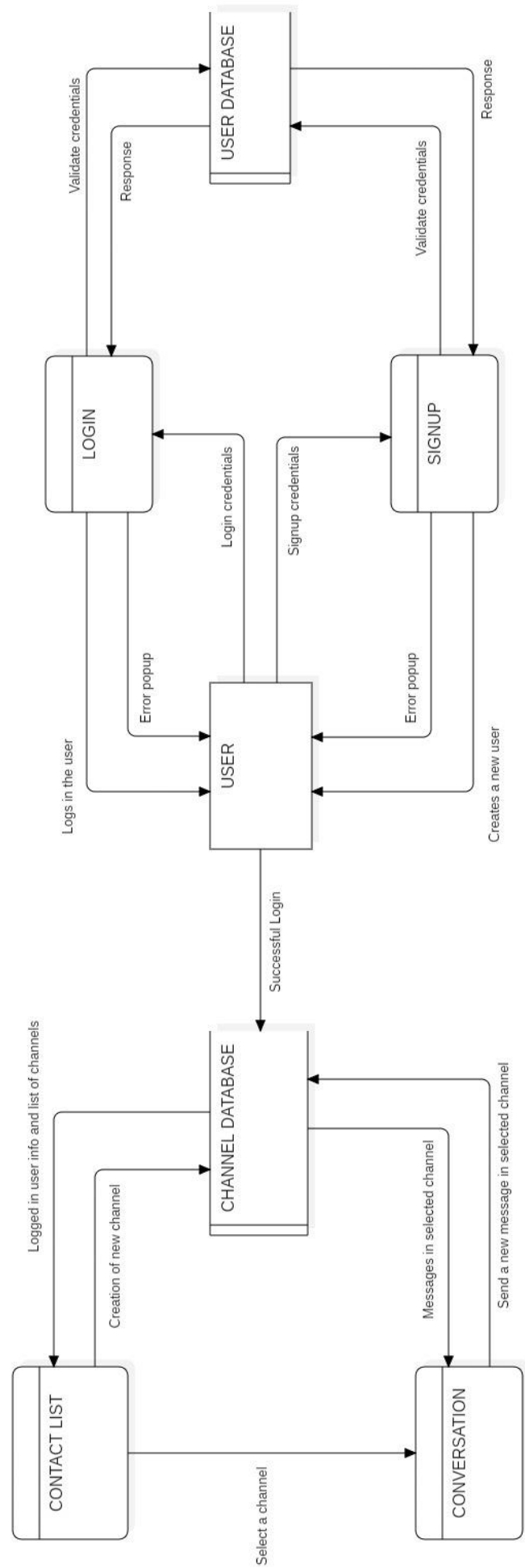
## Level – 0 Data Flow Diagram of the project

1. Users either log in or sign up using dedicated forms.
2. The application validates entered credentials against stored user data.
3. Upon successful login, users are redirected to the homepage, while a successful signup redirects them to the login page.
4. Users can compose and send messages through the application.
5. The application securely stores these messages in the database.
6. Messages are then forwarded to the designated recipients.
7. Incoming messages are stored and promptly displayed to the logged-in user.
8. Users can securely log out by clicking the logout button.
9. Logout ensures account protection and terminates active sessions.
10. The application seamlessly integrates with the database to manage user credentials and messages.
11. It facilitates bidirectional messaging for efficient communication.
12. The application prioritizes a user-focused interaction to enhance the overall experience.
13. Messages are securely stored to maintain data confidentiality.
14. The user-friendly flow includes login/signup, authentication, messaging, and the logout process.



## **Level – 1 Data Flow Diagram of the project**

1. Existing users log in through the Login component, which validates credentials against the User Database.
2. A backend response to successful validation results in user access, while unsuccessful attempts trigger an error popup.
3. New users access the Signup component to fill in required details for account creation.
4. The Signup component validates user credentials against the user schema.
5. Backend response leads to either successful account creation or displays an error message for failed validation.
6. After a successful account creation, users are seamlessly navigated to the Login component for login access.
7. Upon successful login, the backend fetches the channel database.
8. The user is presented with a comprehensive list of existing channels.
9. Users can create new channels from a designated section.
10. Users select a specific chat from the channel list, redirecting them to the Conversation component.
11. The Conversation component dynamically retrieves all existing messages from the channel database.
12. Users can send new messages and engage in conversations using the send message functionality.
13. Users can finally log out by clicking on the logout button.



## Use Case Diagram of the project

### 1. Actors:

- a) User: Represents a system user, categorized as either New or Existing.
- b) User Database (External): Stores user information and interacts with Login and Signup components.
- c) Channel Database (External): Stores contact information and messages, interacts with Contact Item, Contact List, Message, and Conversation components.

### 2. Use Cases:

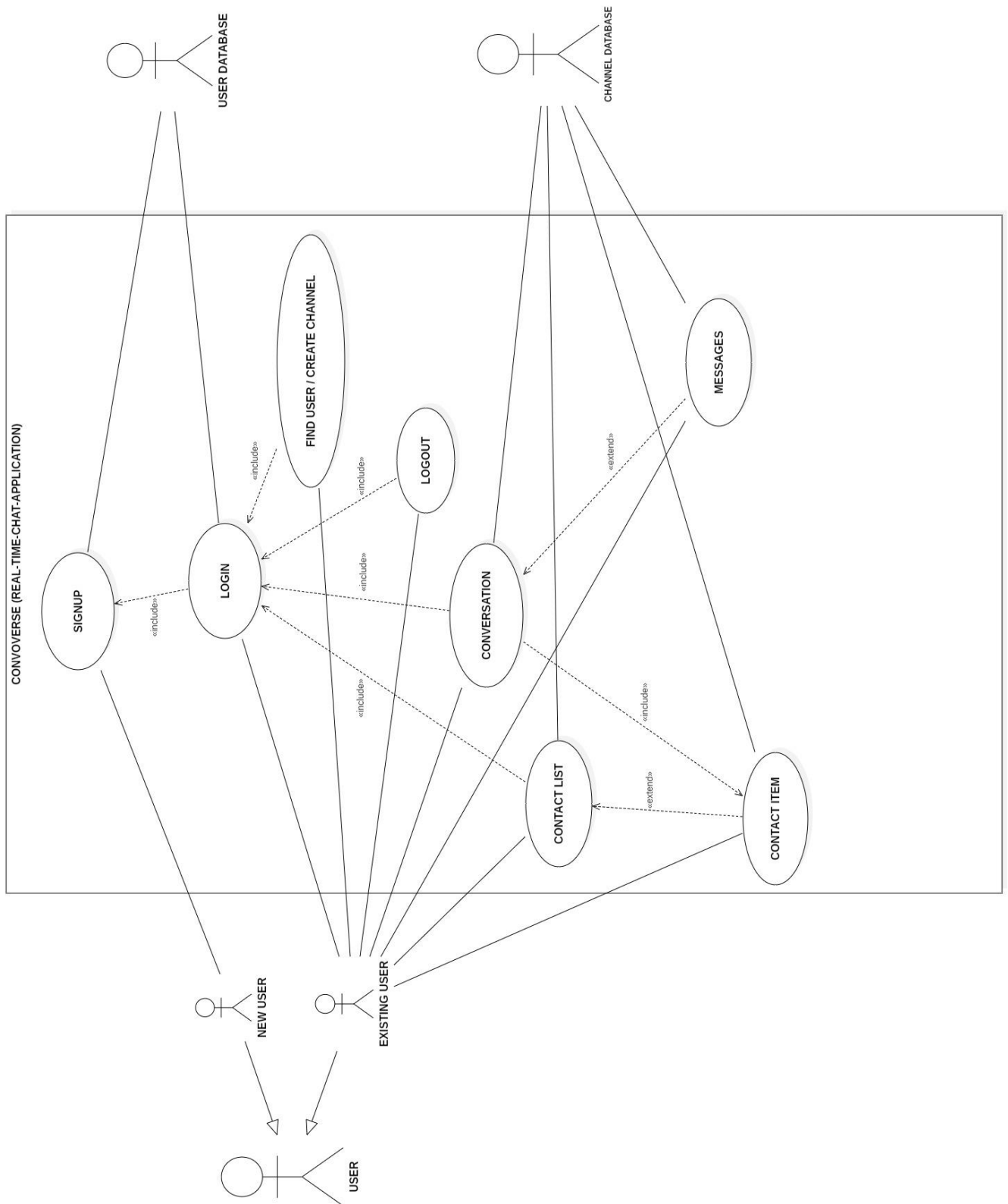
- a) Signup: Allows New Users to register and create their accounts.
- b) Login: Enables Existing Users to authenticate and access the system.
- c) Contact List: Displays a list of all contacts for a Logged In User.
- d) Contact Item: Represents an individual contact within the Contact List. Clicking a Contact Item opens the Conversation component.
- e) Conversation: Shows the communication history with a specific contact.
- f) Message: Represents individual messages within the Conversation.

### 3. Relationships:

- a) Generalization: User is generalized into New User and Existing User.
- b) Extension: Contact Item is an extension of Contact List, meaning it can be empty or contain Contact Items.
- c) Extension: Message is an extension of Conversation, meaning it can be empty or contain Messages.
- d) Include: To view Contact List or Conversation, User must be Logged In.

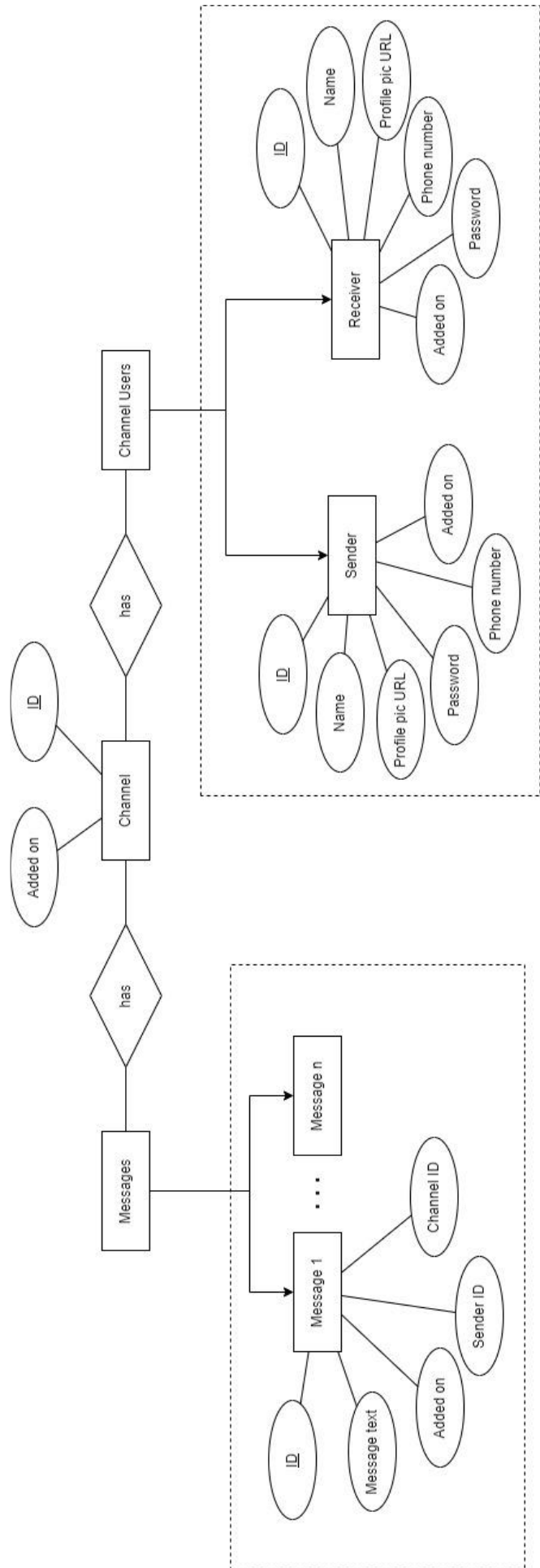
- e) Include: Login includes Signup functionality, meaning users must be signed up to log in.
- f) Include: User can only logout if Logged In.
- g) Include: Logged In Users can create new contacts and search for other users.
- h) Association: User Database interacts with Login and Signup components.
- i) Association: Channel Database interacts with Contact Item, Contact List, Message, and Conversation components.





## Entity relationship Diagram for the Database structure

1. The major objective of our chatting application called "Convoverse" was to create users first and then to create channels to send messages between users.
2. Each channel mainly has an attribute called "\_id" that is unique for each channel, and an attribute called "addedOn" for storing the time when the was channel created.
3. Each channel also has an array of users called "channelUsers" which only consists of 2 users, the sender and receiver.
4. Each of these users has attributes as "\_id" that is unique, "name" for storing the user's name, "phoneNumber" for storing the user's phone number, "profilePicURL" for storing the link to user's profile pic, "password" for storing the user's password and "addedOn" for storing the time when the user was created.
5. Each channel also has an array called "messages" which contains all the messages in the channel whether it is send by sender or reciever.
6. There can be unlimited number of messages, and each message has attributes as "\_id" that is unique, "messageText" that contains the actual message being sent, "addedOn" for storing the time when the message was sent.
7. Each channel also has 2 additional attributes, first being "channelID" which tells which channel this message is a part of, and "senderID" which tells out of the 2 users present in the channel, who sent this message.



# MEET THE TEAM

Presenting our dynamic team: Abhik Gupta leads with full-stack expertise, Nishchay Chandok weaves frontend magic, John P Varghese shines in backend brilliance, and Yatin Hooda brings database expertise. Together, we craft unforgettable experiences.



## ABHIK GUPTA

Full Stack Web Developer

Leads the project as a full-stack web developer, playing a key role in developing both frontend and backend components using React and Node.js. Responsibilities include seamless integration and design contributions to UI and backend structure.

## JOHN P VARGHESE

Back end Web Developer

Has played a pivotal role in crafting all API calls using Express. Additionally, he is responsible for setting up the actual server of the project using Node.js. His efforts also encompass the integration of the backend with the frontend.



## YATIN HOODA

Database Engineer

Has made substantial contributions to database management. His role involves efficiently managing and incorporating all data into the actual database for testing purposes. He also has made major efforts in the actual development of backend for the project.

## NISHCHAY CHANDOK

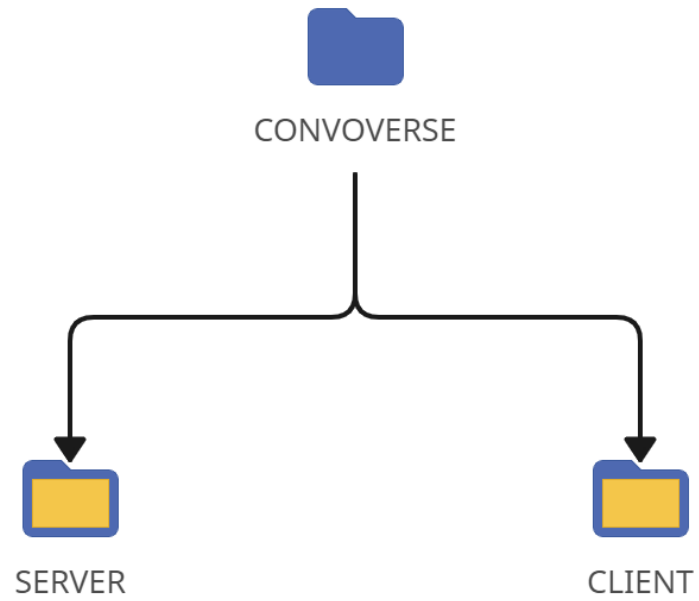
Front End Web Developer

Has been instrumental and really effective in the creation of the actual React app that is both effective and good looking, taking charge of a significant portion of the frontend development.

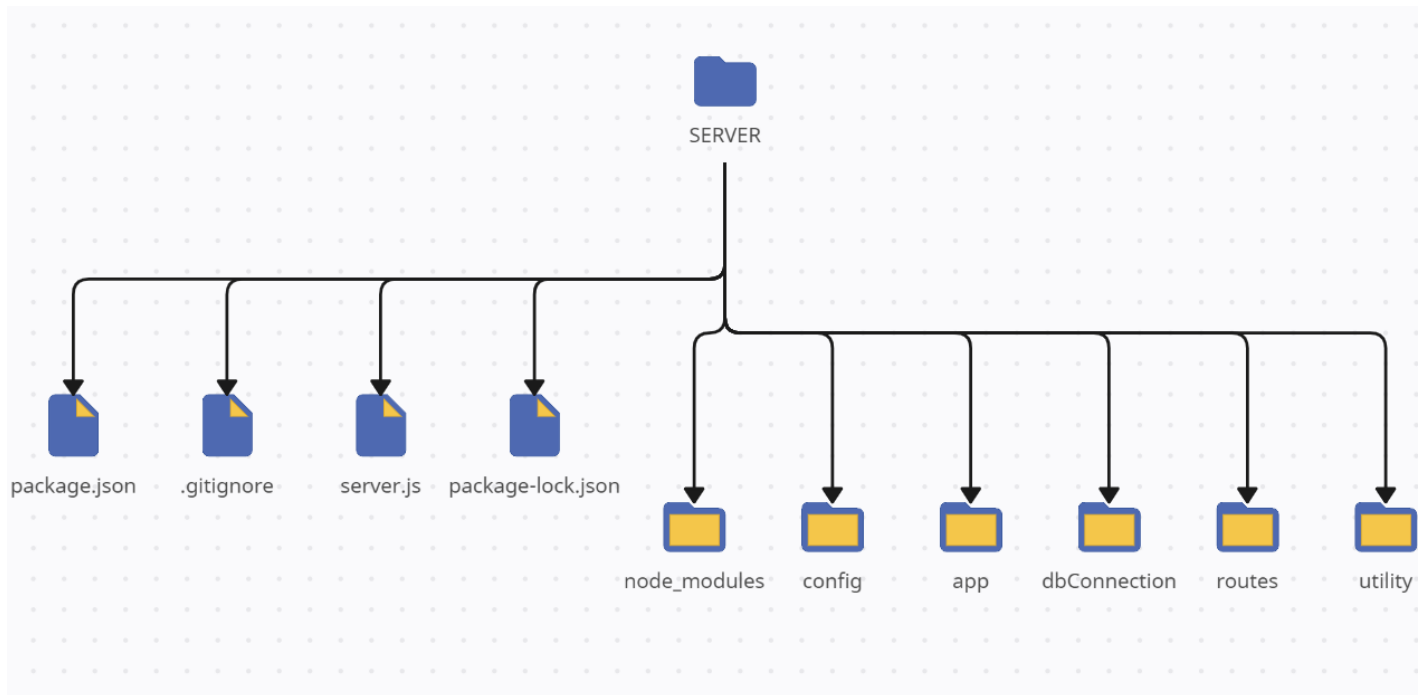


# CODING AND IMPLEMENTATION

## CONVOVERSE PROJECT



# SERVER



## package.json (server)

```
{  
  "type": "module",  
  "name": "convoverse",  
  "version": "1.0.0",  
  "description": "A real time chat application for our minor project",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js",  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
}
```

```

"author": "AbhikGupta",

"license": "ISC",

"dependencies": {
  "babel": "^6.23.0",
  "babel-cli": "^6.26.0",
  "babel-preset-node8": "^1.2.0",
  "body-parser": "^1.20.2",
  "compression": "^1.7.4",
  "cors": "^2.8.5",
  "express": "^4.18.2",
  "moment": "^2.29.4",
  "moment-timezone": "^0.5.43",
  "mongoose": "^7.6.7",
  "yup": "^1.3.2"
},

"devDependencies": {
  "@babel/core": "^7.23.2",
  "@babel/node": "^7.22.19"
}
}

```

## **.gitignore (server)**

```
./node_modules
```

## server.js (server)

```
import APP from "express";

import connectDB from "../dbConnection/index.js";

import configureExpressApp from "../config/index.js";

import applyRoutes from "../routes/index.js";


// Creating a new instance of the Express application

const app = new APP();


// Configuring the Express application using the imported function

configureExpressApp(app);


// Defining the port on which the server will run

const PORT = 3005;


// Function to start the server

const startServer = () => {

  Promise.all([connectDB()]).then(() => {

    app.listen(PORT);

    console.log(`-- Server started at port ${PORT} --`);

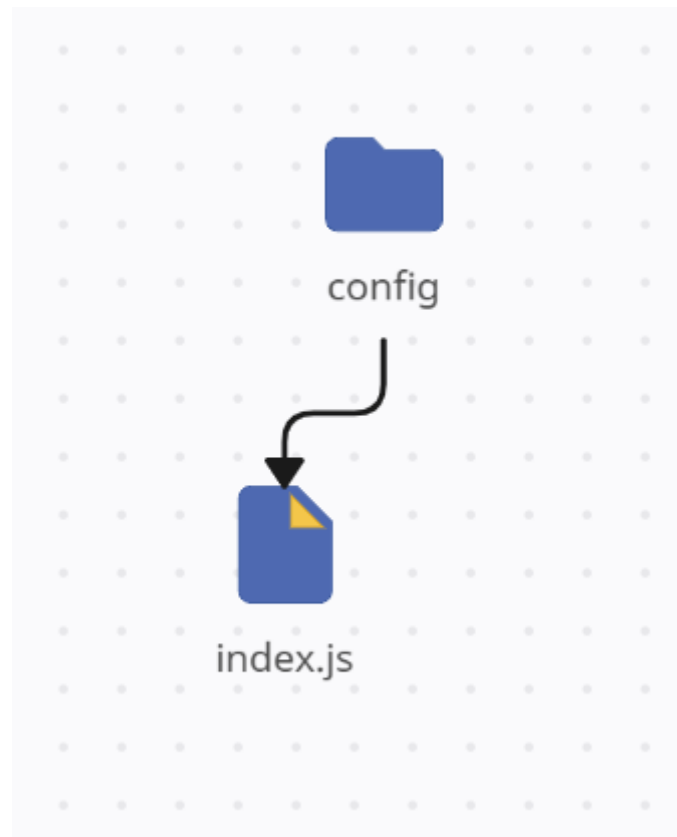
    applyRoutes(app);

  })

}

startServer();
```





## **index.js (config)**

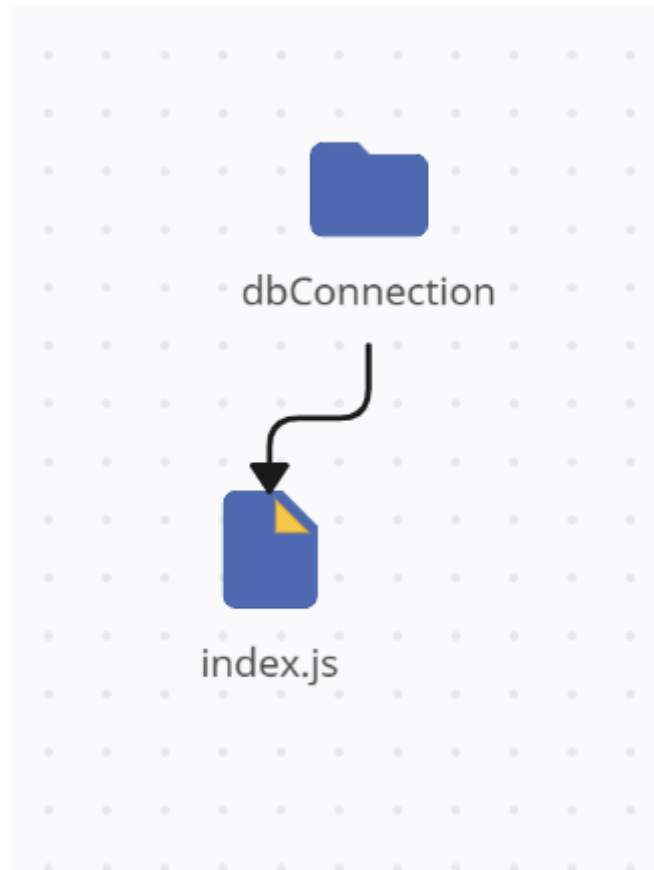
```
import bodyParser from "body-parser";  
  
import compression from "compression";  
  
import cors from "cors";  
  
const configureExpressApp = (app) => {  
  // Parse URL-encoded data and populate req.body  
  app.use(bodyParser.urlencoded({extended:true}));  
}
```

```
// Parse JSON data and populate req.body
app.use(bodyParser.json());

// Enable response compression with compression level 9
app.use(compression(9));

// Enable Cross-Origin Resource Sharing (CORS)
app.use(cors());
}

export default configureExpressApp;
```



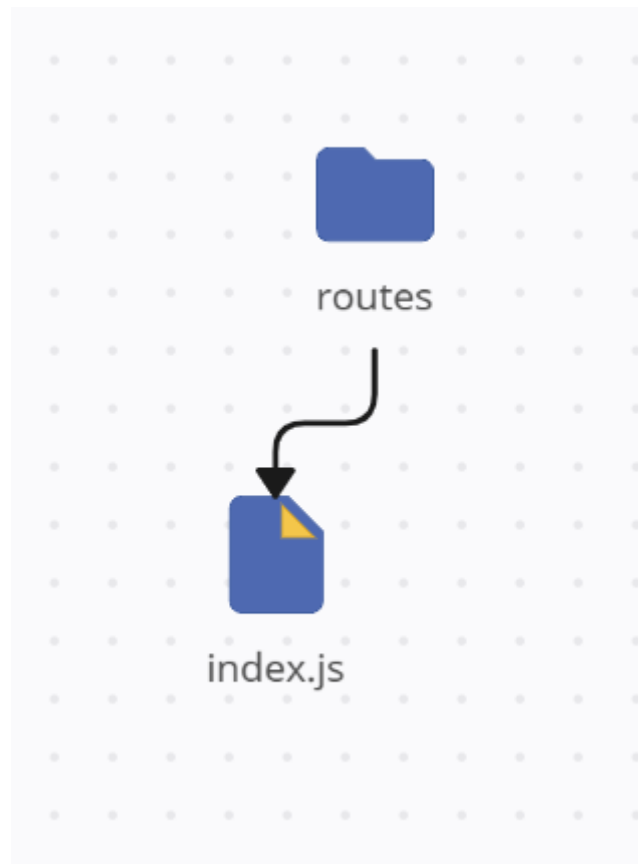
## **index.js (dbConnection)**

```
import mongoose from "mongoose";

// MongoDB connection URL
const DB_CONNECTION_URL = 'mongodb://127.0.0.1:27017/convoverseDB';

// Function to connect to the MongoDB database
const connectDB = () => {
  console.log(`-- Database is trying to connect on ${new Date()} --`);
  const options = {
```

```
    keepAlive: true,  
    autoReconnect: true,  
    poolSize: 10,  
    useUrlParser: true,  
    useUnifiedTopology: true,  
  }  
  return mongoose.connect(DB_CONNECTION_URL, options);  
}  
  
export default connectDB;
```



## **index.js (routes)**

```
import * as Controller from "../app/controllers/index.js";  
  
import * as Validation from "../utility/validations.js";  
  
const applyRoutes = (app) => {  
  // Default route to check if the API is running  
  app.get('/', (req, res) => res.json('API is running!'));
```

```

    // Route to create a new user, with validation and controller handling
    app.post('/user', Validation.validateCreateUser,
Controller.createUser);

    // Route for user login, with validation and controller handling
    app.post('/login', Validation.validateLogin, Controller.loginUser);

    // Route to create a new channel, with validation and controller
handling
    app.post('/channel', Validation.validateCreateChannel,
Controller.createChannel);

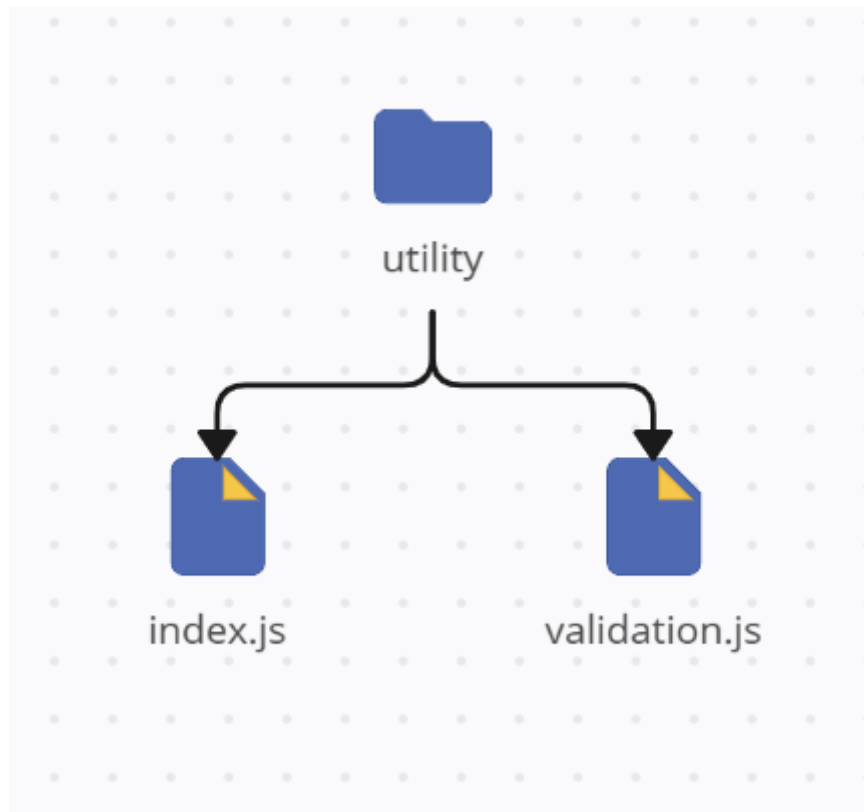
    // Route to get a list of channels, with validation and controller
handling
    app.get('/channel-list', Validation.validateGetChannels,
Controller.getChannels);

    // Route to search for a user, with validation and controller handling
    app.get('/search-user', Validation.validateSearchUser,
Controller.searchUser);

    // Route to send a message, with validation and controller handling
    app.post('/message', Validation.validateAddMessage,
Controller.sendMessage);
}

export default applyRoutes;

```



## **index.js (utility)**

```
// Function to send a formatted success response

const sendResponse = (res, data, msg, success, code) => {

  // Creating a response object with specified properties

  const responseObj = {

    responseData: data,

    message: msg,

    success: success,
```

```

        responseCode: code
    }

    // Sending the response in JSON format
    res.format({
        json: () => {
            res.send(responseObj);
        }
    })
}

// Function to send a formatted error response
const sendError = (res, data, msg) => {
    // Calling the sendResponse function with default error values
    sendResponse(res, data, msg || "Request failed!", false, 400);
}

export { sendResponse, sendError };

```

## **validation.js (utility)**

```

import { sendError } from "../index.js";

import * as yup from "yup";

```



```
// Validation middleware for creating a user

export const validateCreateUser = async (req, res, next) => {

  const schema = yup.object().shape({

    phoneNumber: yup.number().required(),

    name: yup.string().min(3, 'Name must be at least 3
characters!').required(),

    password: yup.string().min(8, 'Password must be at least 8
characters!').required(),

    profilePic: yup.string().url()

  });

  await validate(schema, req.body, res, next);

};
```

```
// Validation middleware for user login

export const validateLogin = async (req, res, next) => {

  const schema = yup.object().shape({

    phoneNumber: yup.number().required(),

    password: yup.string().required()

  });

  await validate(schema, req.body, res, next);

};
```

```
// Validation middleware for creating a channel

export const validateCreateChannel = async (req, res, next) => {

  const schema = yup.object().shape({

    channelUsers: yup.array().of(

      yup.object().shape({

        _id: yup.string().required(),

        name: yup.string().required(),
```

```

        profilePic: yup.string()

    })

    ).length(2).required()

  });

  await validate(schema, req.body, res, next);
};

// Validation middleware for getting channels
export const validateGetChannels = async (req, res, next) => {

  const schema = yup.object().shape({

    userId: yup.string().required()

  });

  await validate(schema, req.query, res, next);
};

// Validation middleware for searching for a user
export const validateSearchUser = async (req, res, next) => {

  const schema = yup.object().shape({

    phone: yup.number().required()

  });

  await validate(schema, req.query, res, next);
};

// Validation middleware for adding a message to a channel
export const validateAddMessage = async (req, res, next) => {

  const schema = yup.object().shape({

    channelId: yup.string().required(),

    messages: yup.object().shape({

```

```

        senderID: yup.string().required(),

        message: yup.string().required()

    })

});

await validate(schema, req.body, res, next);

};

// Common validation function used by other validation middlewares
const validate = async (schema, reqData, res, next) => {

    try {

        await schema.validate(reqData, { abortEarly: false });

        next();

    }

    catch (e) {

        const errors = e.inner.map(({ path, message, value }) => ({

            path,

            message,

            value

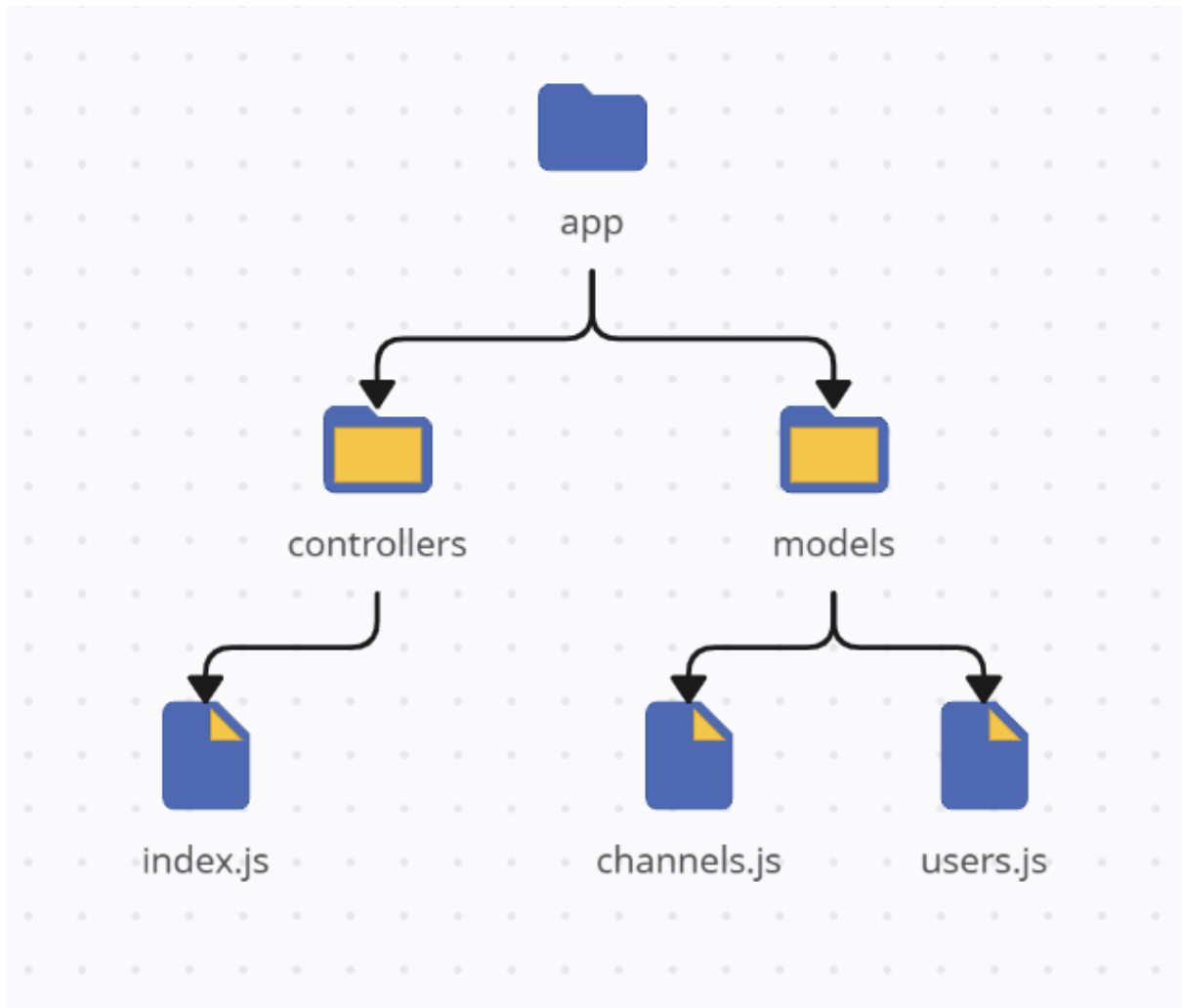
        })));

        sendError(res, errors, "Invalid request!");

    }

};

```



### **index.js (controllers)**

```
import UserModel from "../models/users.js";

import ChannelModel from "../models/channels.js";

import { sendResponse, sendError } from "../../utility/index.js";

// Creating a new user

export const createUser = async (req, res) => {
```

```

    const userObj = new UserModel(req.body);

    await userObj.saveData();

    sendResponse(res, userObj, "User created successfully!", true, 200);
};

// Logging in a user
export const loginUser = async (req, res) => {

    const requestData = req.body;

    const isUserExist = await UserModel.findOneData({

        phoneNumber: requestData.phoneNumber,

        password: requestData.password

    });

    // delete isUserExist.password;

    if (!isUserExist) {

        return sendError(res, {}, "User not found with those
credentials!");

    }

    sendResponse(res, isUserExist, "User logged in successfully!", true,
200);

};

// Creating a new channel
export const createChannel = async (req, res) => {

    const channelObj = new ChannelModel(req.body);

    await channelObj.saveData();

    sendResponse(res, channelObj, "Channel created successfully!", true,
200);

};

```

```

// Getting a list of channels for a user

export const getChannels = async (req, res) => {

  const requestData = req.query;

  const channelList = await ChannelModel.findData({

    "channelUsers._id": requestData.userId

  });

  sendResponse(res, channelList, "Channel list fetched successfully!",
true, 200);

};


// Searching for a user by phone number

export const searchUser = async (req, res) => {

  const requestData = req.query;

  const isUserExist = await UserModel.findOneData({

    phoneNumber: requestData.phone

  });

  if (!isUserExist) {

    return sendError(res, {}, "No user found!");

  }

  sendResponse(res, isUserExist, "User found successfully!", true, 200);

};


// Sending a message to a channel

export const sendMessage = async (req, res) => {

  const requestData = req.body;

  await ChannelModel.findOneAndUpdateData(

    { _id: requestData.channelId },

```

```

        { $push: { messages: requestData.messages } }
    );

    sendResponse(res, {}, "Message sent successfully!", true, 200);
};

```

## **channels.js (models)**

```

import mongoose from "mongoose";

// Defining the schema for the 'channels' collection
const channelSchema = new mongoose.Schema({

    // Array of channel users with their information
    channelUsers: [

        {

            _id: { type: String, default: "" },

            name: { type: String, default: "" },

            profilePic: { type: String, default: "" }

        }

    ],

    // Array of messages in the channel
    messages: [

        {

            senderID: { type: String, default: "" },

```

```

        message: { type: String, default: "" },
        addedOn: { type: Date, default: Date.now() }
    }
],
// Timestamp indicating when the channel was added
addedOn: { type: Date, default: Date.now() }
});

// Defining instance methods for the channel schema
channelSchema.method({
    // Method to save channel data to the database
    saveData: async function() {
        return this.save()
    }
})

// Defining static methods for the channel schema
channelSchema.static({
    // Method to find multiple channels based on a query object
    findData: function(findObj) {
        return this.find(findObj)
    },

    // Method to find a single channel based on a query object
    findOneData: function(findObj) {
        return this.findOne(findObj)
    },

```



```

        // Method to find and update a channel based on query and update
objects

        findOneAndUpdateData: function(findObj, updateObj) {

            return this.findOneAndUpdate(findObj, updateObj, {

                upsert: true,

                new: true,

                setDefaultsOnInsert: true

            })

        }

    })

export default mongoose.model('channelSchema', channelSchema);

```

## users.js (models)

```

import mongoose from "mongoose";

// Defining the schema for the 'users' collection

const userSchema = new mongoose.Schema({

    name: { type: String, default: "" },

    phoneNumber: { type: Number, default: "" },

    password: { type: String, default: "" },

    profilePic: { type: String, default: "" },

    addedOn: { type: Date, default: Date.now() }

```

```

});

// Defining instance methods for the user schema
userSchema.method({

  // Method to save user data to the database
  saveData: async function() {

    return this.save()

  }

})

// Defining static methods for the user schema
userSchema.static({

  // Method to find multiple users based on a query object
  findData: function(findObj) {

    return this.find(findObj)

  },

  // Method to find a single user based on a query object
  findOneData: function(findObj) {

    return this.findOne(findObj)

  },

  // Method to find and update a user based on query and update objects
  findOneAndUpdateData: function(findObj, updateObj) {

    return this.findOneAndUpdate(findObj, updateObj, {

      upsert: true,

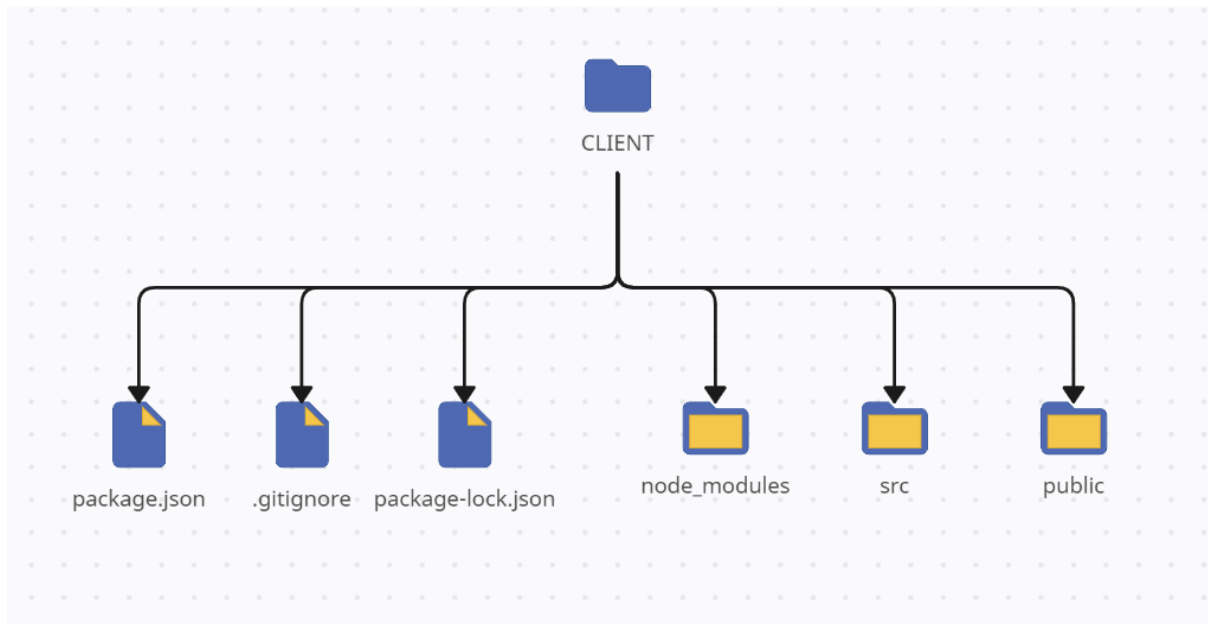
      new: true,

      setDefaultsOnInsert: true
    })
  }
})

```

```
        })  
    }  
})  
  
export default mongoose.model('úserSchema', userSchema);
```

## CLIENT



### package.json (client)

```
{  
  "name": "whatsapp-clone",  
  "version": "0.1.0",  
  "private": true,  
  "dependencies": {  
    "@leecheuk/react-google-login": "^5.4.1",  
    "@testing-library/jest-dom": "^5.17.0",  
    "@testing-library/react": "^13.4.0",  
    "@testing-library/user-event": "^13.5.0",  
    "emoji-picker-react": "^4.5.2",  
    "react": "^18.2.0",
```

```

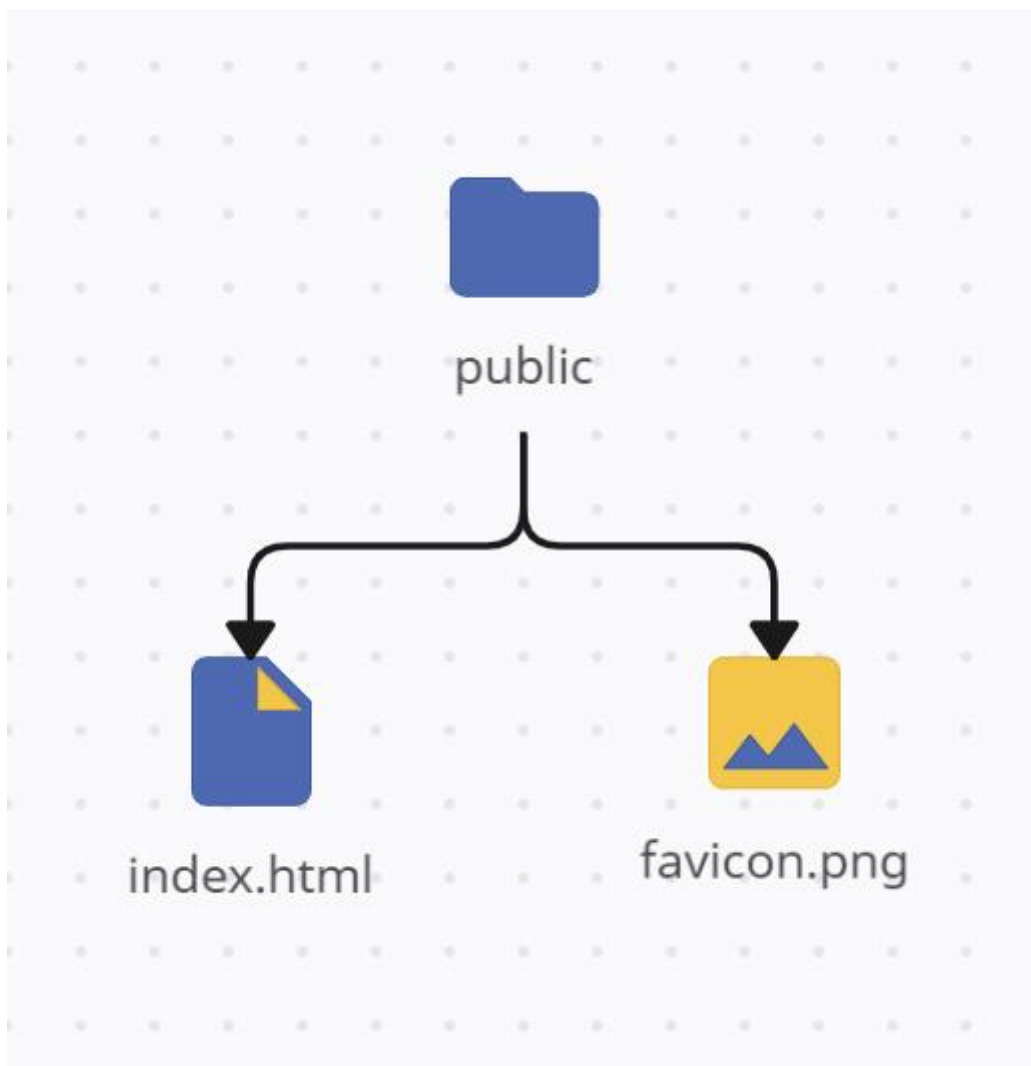
    "react-dom": "^18.2.0",
    "react-icons": "^4.11.0",
    "react-router-dom": "^6.20.0",
    "react-scripts": "5.0.1",
    "react-toastify": "^9.1.3",
    "styled-components": "^6.0.8",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",

```

```
        "last 1 firefox version",  
        "last 1 safari version"  
    ]  
}  
}
```

## **.gitignore (client)**

```
/node_modules  
/.pnp  
.pnp.js  
/coverage  
/build  
  
.DS_Store  
.env.local  
.env.development.local  
.env.test.local  
.env.production.local  
  
npm-debug.log*  
yarn-debug.log*  
yarn-error.log*
```



### **index.html (public)**

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="utf-8" />

  <link rel="icon" href="favicon.png" />
```

```
<meta name="viewport" content="width=device-width, initial-scale=1" />

<meta name="theme-color" content="#000000" />

<meta name="description" content="Convoverse is a cutting-edge real-time
chat web application designed to redefine the way people connect and
communicate online. " />

<title>CONVOVERSE - Send messages quick and securely!</title>

</head>

<body>

  <noscript>You need to enable JavaScript to run this app.</noscript>

  <div id="root"></div>

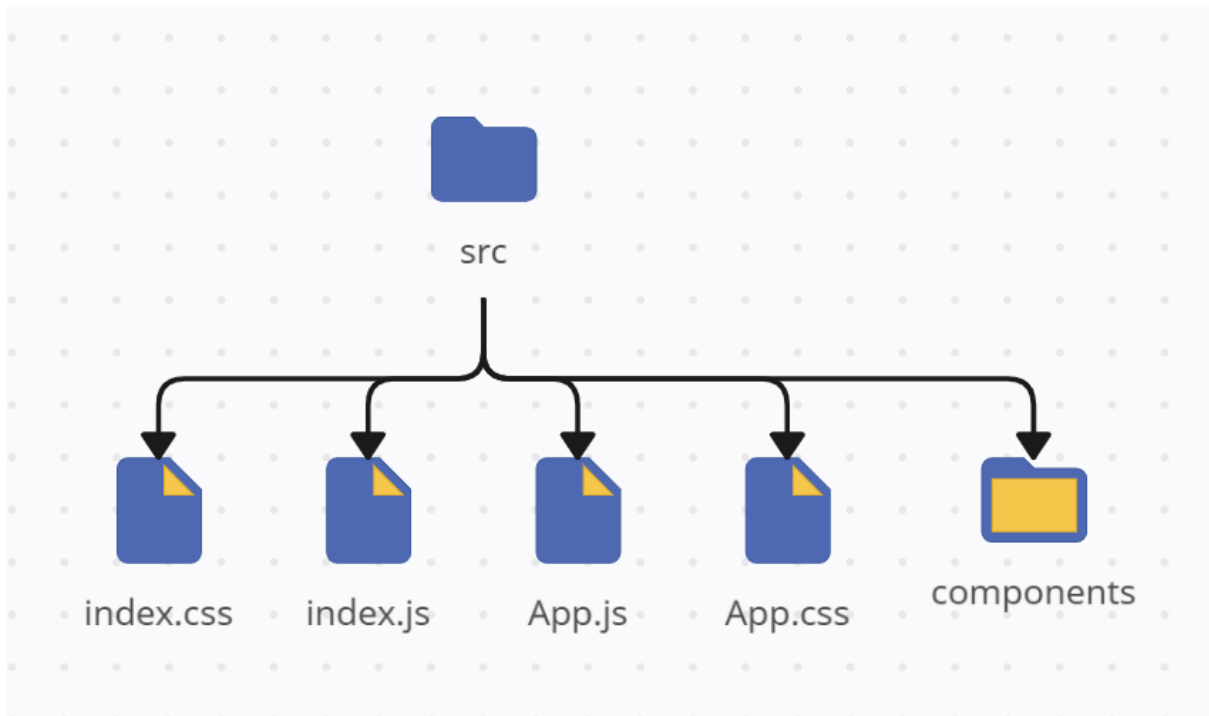
</body>

</html>
```

### **favicon.png (public)**







## index.js (src)

```
import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import { BrowserRouter } from 'react-router-dom';

// Creating a React root and rendering the App component within it
const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    { /* Providing BrowserRouter to enable routing */ }
```

```

    <BrowserRouter>

      {/* Rendering the main App component */}

      <App />

    </BrowserRouter>

  </React.StrictMode>

);

```

## index.css (src)

```

@import
url('https://fonts.googleapis.com/css2?family=Noto+Sans:wght@100;200;300;400;500;600;700;800;900&display=swap');

* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

  font-family: 'Noto Sans', sans-serif;

}

html,

body {

  height: 100%;

  width: 100%;

}

```

## app.js (src)

```
import React, { useState } from 'react';

import './App.css';

import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

import styled from 'styled-components';

import ContactList from './components/ContactList';
import Conversation from './components/Conversation';
import ChatSectionPlaceholder from './components/ChatSectionPlaceholder';
import Login from './components/Login';
import Signup from './components/Signup';

import { Route, Routes } from 'react-router-dom';

// Main App component
const App = () => {

  // State to manage the display of placeholder and selected chat
  const [ chatPlaceHolder, setChatPlaceHolder ] = useState(true);

  const [ selectedChat, setSelectedChat ] = useState();

  return (

    <>

      <Routes>

        { /* Route for the login page */ }

        <Route path="/" element={ <Login /> } />

      </>

    </>

  )
}
```

```

    { /* Route for the signup page */ }

    <Route path='/signup' element={<Signup />} />

    { /* Route for the home page */ }

    <Route path='/home' element={
      <div className='app'>

        { /* ContactList component */ }

        <ContactList setChatPlaceholder={setChatPlaceholder}
setSelectedChat={setSelectedChat} />

        { /* Conditional rendering based on 'chatPlaceholder' state */ }

        { chatPlaceholder ? (<ChatSectionPlaceholder />) :
(<Conversation selectedChat={selectedChat} />) }

      </div>

    } />

  </Routes>

  <ToastContainer />

</>

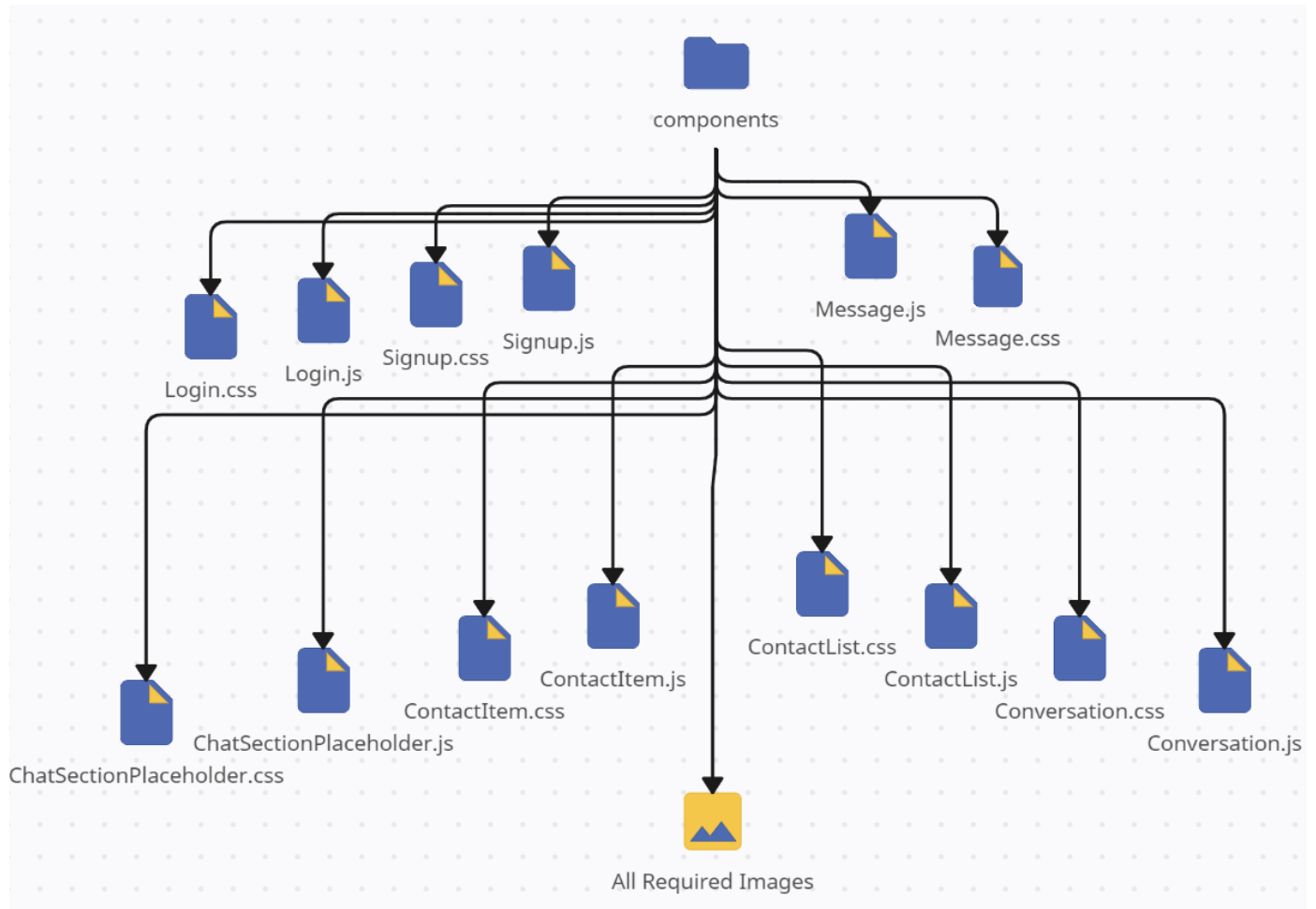
)
}

export default App;

```

## **app.css (src)**

```
.app {  
  
    display: flex;  
  
    flex-direction: row;  
  
    height: 100vh;  
  
    width: 100vw;  
  
    background-color: #f8f9fb;  
  
}
```



## ChatSectionPlaceholder.js (components)

```

import React from 'react';

import './ChatSectionPlaceholder.css';

import placeholderImage from './ChatSectionPlaceholderImage.png';

// Functional component for displaying a placeholder in the chat section
const ChatSectionPlaceholder = () => {

```

```

return (
  <div className='placeholderContainer'>
    <div className="imageDiv">
      <img src={placeholderImage} alt="Placeholder" />
    </div>
    <h1>Welcome to Convoverse!</h1>
    <p>Whatsapp connects to your phone to sync messages<br></br>and
provide you a seamless experience</p>
  </div>
)
}

export default ChatSectionPlaceholder;

```

## **ChatSectionPlaceholder.css (components)**

```

.placeholderContainer {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100%;
  width: 70%;

```

```
        background-color: rgb(255, 252, 252);
    }
}
```

```
.imageDiv {
    padding: 1vmax;
    border-radius: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
}
```

```
.imageDiv > img {
    width: 12vmax;
    transition: 0.5s;
}
```

```
.imageDiv > img:hover {
    cursor: grab;
    transform: scale(1.1);
}
```

```
.placeholderContainer > h1 {
    font-size: 1.6vmax;
    font-weight: 600;
    color: #00000088;
}
```

```
.placeholderContainer > p {
```



```
font-size: 0.8vmax;

font-weight: 400;

color: #00000088;

text-align: center;

}
```

## ContactItem.js (components)

```
import React, { useEffect, useState } from 'react';

import './ContactItem.css';

import { FaCheckCircle } from "react-icons/fa";

import { FaEnvelope } from "react-icons/fa";

// Functional component representing a contact item in the chat list

const ContactItem = (props) => {

  // Destructuring props to extract relevant data and functions

  const { userInfo, setChatPlaceholder, setSelectedChat } = props;

  // State to manage display data for the user

  const [userDisplayData, setUserDisplayData] = useState({
    userDisplayPic: '', userDisplayName: '' });

  // States for the last message and its sender indicator

  const [lastMessage, setLastMessage] = useState('');
```

```

const [lastMessageSign, setLastMessageSign] = useState();

// Function to fetch and set user display data based on the channel
users

function fetchContactData() {

    if (localStorage.getItem('convoverseUserLoginId') ===
userInfo.channelUsers[0]._id) {

        setUserDisplayData({ userDisplayPic:
userInfo.channelUsers[1].profilePic, userDisplayName:
userInfo.channelUsers[1].name })

    }

    else {

        setUserDisplayData({ userDisplayPic:
userInfo.channelUsers[0].profilePic, userDisplayName:
userInfo.channelUsers[0].name })

    }

}

// Function to fetch and set the last message and its sender indicator

function fetchLastMessage() {

    if (userInfo.messages.length > 0) {

        const length = userInfo.messages.length;

        const lastMsg = userInfo.messages[length - 1];

        if (lastMsg.senderID ===
localStorage.getItem('convoverseUserLoginId')) {

            setLastMessageSign(0);

        }

        else {

            setLastMessageSign(1);

        }

    }

}

```

```

        const lastMsgText = lastMsg.message.slice(0, 40);

        setLastMessage(lastMsgText);

    }

}

// useEffect to fetch user display data and the initial last message on
component mount

useEffect(() => {

    fetchContactData();

    fetchLastMessage();

    // eslint-disable-next-line react-hooks/exhaustive-deps

}, []);

// useEffect to update the last message when there's a change in
userInfo.messages

useEffect(() => {

    fetchLastMessage();

    // eslint-disable-next-line react-hooks/exhaustive-deps

}, [userInfo.messages]);

return (

    <div className='contactItemContainer' onClick={() => {

        setChatPlaceholder(false);

        setSelectedChat(userInfo);

    }}>

        <div className="leftSection">

            <div className="chatDP">

                <img src={userDisplayData.userDisplayPic} alt="DP" />

            </div>

```

```

        <div className="chatContent">

            <h1
className='contactName'>{userDisplayData.userDisplayName}</h1>

            <p className='lastText'>{lastMessage}</p>

        </div>

    </div>

    <div className="rightSection">

        {/* Conditional rendering based on the lastMessageSign */}

        {lastMessageSign === 0 ? <p
className='sentSign'><FaCheckCircle /></p> : <p
className='recievedSign'><FaEnvelope /></p>}

    </div>

</div>

)

}

export default ContactItem;

```

## ContactItem.css (components)

```

.contactItemContainer {

    height: 5vmax;

    border-bottom: 2px solid #b3b3b3;

    display: flex;

    flex-direction: row;

```

```

    align-items: center;

    background-color: #ffffffdc;

    padding: 0px 0.4vmax;

    transition: 0.3s;
}

.contactItemContainer:hover {

    background-color: #00977e;

    cursor: grab;
}

.contactItemContainer:active {

    cursor: grabbing;
}

.contactItemContainer:hover .chatContent > h1 {

    color: #fff;
}

.contactItemContainer:hover .chatContent > p {

    color: #ffffffcc;
}

.contactItemContainer:hover .rightSection > p {

    color: #ffffffcc;
}

.leftSection {

```

```
width: 100%;

height: 100%;

display: flex;

flex-direction: row;

flex: 1;
}

.chatDP {

width: 20%;

height: 100%;

display: flex;

align-items: center;

justify-content: center;
}

.chatDP > img {

width: 75%;

height: 75%;

object-fit: cover;

object-position: center;

border-radius: 50%;
}

.chatContent {

width: 75%;

display: flex;

flex-direction: column;

justify-content: center;
```

```
    gap: 0px;

    padding-left: 0.3vmax;
}
```

```
.chatContent > h1 {

    font-size: 1vmax;

    transition: 0.3s;

    color: #000;

    font-weight: 800;
}
```

```
.chatContent > p {

    font-size: 0.8vmax;

    transition: 0.3s;

    color: #000;

    font-weight: 400;
}
```

```
.rightSection {

    height: 100%;

    width: 15%;

    display: flex;

    align-items: center;

    justify-content: center;

    font-size: 0.8vmax;
}
```

```
.rightSection > p {
```

```

        font-size: 1vmax;

        font-weight: 500;

        transition: 0.3s;
    }

    .sentSign {

        color: #00000062;
    }

    .recievedSign {

        color: #00b395;
    }

```

## ContactList.js (components)

```

import React, { useEffect, useState } from 'react';

import './ContactList.css';

import { ImSearch } from "react-icons/im";

import ContactItem from './ContactItem';

import { useNavigate } from 'react-router-dom';

import { toast } from 'react-toastify';

// Functional component representing the list of contacts in the chat
application

const ContactList = (props) => {

```



```

// Destructuring props to extract relevant data and functions

const { setChatPlaceholder, setSelectedChat } = props;

// States to manage user information, search text, and navigation
const [userInfo, setUserInfo] = useState([]);
const [searchText, setSearchText] = useState('');
const navigate = useNavigate();

// Function to logout the user and navigate to the login page
function logoutUser() {
    localStorage.removeItem('convoverseUserLoginId');
    localStorage.removeItem('convoverseUserLoginName');
    localStorage.removeItem('convoverseUserLoginProfilePic');
    setChatPlaceholder(true);

    // Toast notification for successful user logout
    toast.success('User logged out successfully!', {
        position: "bottom-right",
        autoClose: 2000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
        theme: "colored",
    });

    // Navigating to the login page

```

```

        navigate('/');
    }

    // Function to fetch user channels

    async function fetchChannels() {

        const loggedInUserId =
localStorage.getItem('convoverseUserLoginId');

        const response = await fetch(`http://localhost:3005/channel-
list?userId=${loggedInUserId}`, {

            method: 'GET',

            headers: {

                'Content-Type': 'application/json'

            }

        });

        const jsonResponse = await response.json();

        setUserInfo(jsonResponse.responseData);

    }

    // Function to refresh user channels

    async function refreshChannels() {

        const loggedInUserId =
localStorage.getItem('convoverseUserLoginId');

        const response = await fetch(`http://localhost:3005/channel-
list?userId=${loggedInUserId}`, {

            method: 'GET',

            headers: {

                'Content-Type': 'application/json'

            }

        });
    }

```

```

    const jsonResponse = await response.json();

    setUserInfo(jsonResponse.responseData);

    setSearchText('');
}

// Event handler for text input change
function onChange(e) {

    setSearchText(e.target.value);
}

// Function to search for a user and initiate a chat
async function searchUser() {

    const searchResponse = await fetch(`http://localhost:3005/search-
user?phone=${searchText}`, {

        method: 'GET',

        headers: {

            'Content-Type': 'application/json'

        }

    });

    const searchResponseJSON = await searchResponse.json();

    if (searchResponseJSON.responseData &&
searchResponseJSON.responseData._id) {

        const searchResult = searchResponseJSON.responseData;

        // Fetching user channels

        const getChannelResponse = await
fetch(`http://localhost:3005/channel-
list?userId=${localStorage.getItem('convoverseUserLoginId')}`, {

```

```

        method: 'GET',

        headers: {

            'Content-Type': 'application/json'

        }

    });

    const getChannelResponseJSON = await getChannelResponse.json();

    // Checking if a channel already exists with the searched user

    const existingChannel =
getChannelResponseJSON.responseData.find(channel =>

        channel.channelUsers.some(user => user._id ===
searchResult._id)

    );

    // Creating a new channel if it doesn't exist

    if (!existingChannel) {

        const channelResponse = await
fetch('http://localhost:3005/channel', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({

                channelUsers: [

                    {

                        _id:
localStorage.getItem('convoverseUserLoginId'),

                        name:
localStorage.getItem('convoverseUserLoginName'),

```

```

        profilePic:
localStorage.getItem('convoverseUserLoginProfilePic')

        },

        {

            _id: searchResult._id,
            name: searchResult.name,
            profilePic: searchResult.profilePic

        }

    ]

    })

});

const channelResponseJSON = await channelResponse.json();

if(channelResponseJSON){

    // Toast notification for successful channel creation

    toast.success('A new chat created successfully!', {

        position: "bottom-right",

        autoClose: 2000,

        hideProgressBar: false,

        closeOnClick: true,

        pauseOnHover: true,

        draggable: true,

        progress: undefined,

        theme: "colored",

    });

}

else{

    // Toast notification for unsuccessful channel creation

```

```

        toast.error('Something went wrong!', {
            position: "bottom-right",
            autoClose: 2000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
            theme: "colored",
        });
    }
}

else {
    // Toast notification if a channel already exists with the
user
    toast.info('You already have a chat with this user!', {
        position: "bottom-right",
        autoClose: 2000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
        theme: "colored",
    });
}

// Refreshing user channels after the operation

```

```

        refreshChannels();
    }

    else {
        // Toast notification if the user is not found
        toast.error('Sorry could not find the user!', {
            position: "bottom-right",
            autoClose: 2000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
            theme: "colored",
        });
    }
}

// Event handler for the "Enter" key press
function onEnterPress(e) {
    if (e.key === "Enter") {
        searchUser();
    }
}

// useEffect to fetch user channels on component mount
useEffect(() => {
    fetchChannels();

```

```

    }, []);

    return (
      <div className='contactListContainer'>
        <div className='contactListHeader'>
          <div className="profileInfo">
            <div className="profilePic">
              <img
src={localStorage.getItem('convoverseUserLoginProfilePic')} alt='Profile'
/>
            </div>
            <div className="profileName">
              <p>{localStorage.getItem('convoverseUserLoginName')}</p>
            </div>
          </div>
          <div className="logoutDiv">
            <button className="logoutButton"
onClick={logoutUser}>Logout</button>
          </div>
        </div>

        <div className="searchDiv">
          <div className="searchLogo">
            <ImSearch className='ImSearch' onClick={searchUser} />
          </div>
          <div className="searchBox">
            <input type="text" placeholder='Start a new chat by
phone number' value={searchText} onChange={onChange}
onKeyDown={onEnterPress} />
          </div>
        </div>
      </div>
    );
  }
}

```



```

        </div>

    </div>

    <div className="listOfContacts">

        { /* Mapping through user channels and rendering ContactItem
        component for each */}

        {userInfo.map((item) =>

            <ContactItem key={item._id} userInfo={item}
            setChatPlaceholder={setChatPlaceholder} setSelectedChat={setSelectedChat}
            />

            )}

    </div>

</div>

)

}

export default ContactList;

```

## ContactList.css (components)

```

.contactListContainer {

    display: flex;

    flex-direction: column;

    height: 100%;

    background-color: #004e41;

```

```

        width: 30%;
    }

    .listOfContacts {
        overflow: scroll;
        overflow-x: hidden;
    }

    .listOfContacts::-webkit-scrollbar {
        width: 0;
    }

    .contactListHeader {
        display: flex;
        flex-direction: row;
        align-items: center;
        justify-content: space-between;
        background: linear-gradient(to bottom right, #022423, #004136, #00725f,
#00977e);
        color: #fff;
        padding: 0.5vmax;
        height: 8%;
    }

    .profileInfo {
        display: flex;
        flex-direction: row;
        align-items: center;

```

```
    gap: 0.5vmax;

    height: 100%;

    width: 80%;

    margin-top: 1vmax;

    padding-left: 0.3vmax;
}
```

```
.profilePic {

    height: 3vmax;

    width: 3vmax;
}
```

```
.profilePic > img {

    width: 100%;

    height: 100%;

    object-fit: cover;

    object-position: center;

    border-radius: 50%;
}
```

```
.profileName > p {

    font-weight: 800;

    font-size: 1.3vmax;
}
```

```
.logoutDiv {

    height: 100%;

    width: 20%;
}
```

```
display: flex;

align-items: center;

justify-content: flex-end;

margin-top: 1vmax;

}
```

```
.logoutButton {

border: none;

outline: none;

padding: 0.4vmax;

background-color: #ffffff9d;

color: #006454;

font-size: 1vmax;

font-weight: 700;

border-radius: 10px;

transition: 0.3s;

}
```

```
.logoutButton:hover {

cursor: grab;

background-color: rgba(255, 255, 255, 0.76);

}
```

```
.logoutButton:active {

cursor: grabbing;

}
```

```
.searchDiv {
```

```

    height: 8%;

    display: flex;

    flex-direction: row;

    align-items: center;

    justify-content: center;

    background: linear-gradient(to top right, #022423, #004136, #00725f,
#00977e);

    padding: 0.5vmax;
}

```

```

.searchLogo {

    width: 10%;

    height: 75%;

    display: flex;

    align-items: center;

    justify-content: center;

    background-color: #ffffffdc;

    border-top-left-radius: 25px;

    border-bottom-left-radius: 25px;
}

```

```

.searchBox {

    width: 90%;

    height: 75%;

    display: flex;

    align-items: center;

    justify-content: flex-start;

    border-top-right-radius: 25px;
}

```

```

        border-bottom-right-radius: 25px;

        overflow: hidden;
    }

    .searchBox > input {
        background-color: #ffffffdc;

        height: 100%;

        width: 100%;

        border: none;

        outline: none;

        font-size: 0.8vmax;
    }

    .ImSearch {
        font-size: 1.1vmax;
    }

    .listOfContacts {
        height: 84%;

        background: linear-gradient(to bottom right, #022423, #004136, #00725f,
#00977e);
    }

```

## Conversation.js (components)

```
import React, { useEffect, useState } from 'react';

import './Conversation.css';

import { HiOutlineEmojiHappy } from "react-icons/hi";
import { AiOutlineSend } from "react-icons/ai";

import Message from './Message';

import EmojiPicker from 'emoji-picker-react';

import ConversationWallpaper from './ConversationWallpaper.jpg';

// Functional component representing a conversation in the chat application
const Conversation = (props) => {

  // Destructuring props to extract relevant data

  const { selectedChat } = props;

  // States to manage message input, emoji picker visibility, message
  list, and active user data

  const [messageText, setMessageText] = useState('');

  const [pickerVisible, setPickerVisible] = useState(false);

  const [messageList, setMessageList] = useState([]);

  const [activeUserData, setActiveUserData] = useState({ activeUserName:
'', activeUserProfilePic: '' });

  // Event handler for handling changes in the message input

  function handleChange(e) {

    setMessageText(e.target.value);

  }

  // Event handler for emoji click in the emoji picker
```

```

function onEmojiClick(emojiObject) {

    let emoji = emojiObject.emoji;

    console.log(messageText);

    console.log(emoji);

    setMessageText(messageText => messageText + emoji);

    // setPickerVisible(false);

}

// Function to toggle visibility of the emoji picker

function toggleEmojiPicker() {

    setPickerVisible(!pickerVisible);

}

// Function to refresh messages in the current conversation

async function refreshMessages() {

    const loggedInUserID =
localStorage.getItem('convoverseUserLoginId');

    const newResponse = await fetch(`http://localhost:3005/channel-
list?userId=${loggedInUserID}`, {

        method: 'GET',

        headers: {

            'Content-Type': 'application/json'

        }

    });

    const newResponseJSON = await newResponse.json();

    const requiredChat = newResponseJSON.responseData.filter((item) =>
item._id === selectedChat._id);

    setMessageList(requiredChat[0].messages);

}

```



```

// Function to send a message

async function sendMessage() {

    if (messageText !== '') {

        await fetch('http://localhost:3005/message', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json'

            },

            body: JSON.stringify({

                channelId: selectedChat._id, messages: {

                    senderID:

localStorage.getItem('convoverseUserLoginId'),

                    message: messageText

                }

            })

        });

        setMessageText('');

        setPickerVisible(false);

        if (selectedChat.messages.length !== 0) {

            refreshMessages();

        }

    }

}

// Event handler for the "Enter" key press

function onEnterPress(e) {

    if (e.key === "Enter") {

```

```

        sendMessage();
    }
}

// Function to fetch active user data for the conversation
function fetchActiveUserData() {
    if (localStorage.getItem('convoverseUserLoginId') ===
selectedChat.channelUsers[0]._id) {
        setActiveUserData({ activeUserName:
selectedChat.channelUsers[1].name, activeUserProfilePic:
selectedChat.channelUsers[1].profilePic });
    }
    else {
        setActiveUserData({ activeUserName:
selectedChat.channelUsers[0].name, activeUserProfilePic:
selectedChat.channelUsers[0].profilePic });
    }
}

// Function to fetch all messages in the conversation
function fetchAllMessages() {
    const allMessages = selectedChat.messages;
    console.log(allMessages);
    setMessageList(allMessages);
}

// useEffect to fetch active user data and all messages on component
mount and when the selectedChat changes
useEffect(() => {
    fetchActiveUserData();

```

```

        fetchAllMessages();

        // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [selectedChat]);

// useEffect to refresh messages when the messageList changes
useEffect(() => {
    refreshMessages();

    // eslint-disable-next-line react-hooks/exhaustive-deps
}, [messageList]);

// Styling for the conversation wallpaper
const conversationWallpaperStyling = {
    backgroundImage: `url(${ConversationWallpaper})`,
    backgroundPosition: 'center',
    backgroundSize: 'cover',
    backgroundRepeat: 'no-repeat',
    backgroundAttachment: 'fixed'
}

return (
    <div className='conversationContainer'>
        <div className="conversationHeader">
            <div className="chatPerson">
                <div className="personDP">
                    <img src={activeUserData.activeUserProfilePic}
alt="PERSON DP" />
                </div>
                <div className="personName">

```

```

        <p>{activeUserData.activeUserName}</p>

    </div>

</div>

</div>

    <div className="chatSection"
style={conversationWallpaperStyling}>

        {/* Mapping through messages and rendering Message
component for each */}

        {messageList.map((msg) =>

            <Message key={msg._id} messageContent={msg.message}
senderIsMe={msg.senderID === localStorage.getItem('convoverseUserLoginId')}
timestamp={msg.addedOn} />

        )}

    </div>

    <div className="sendMessageDiv">

        <div className="emojiLogo">

            <div className='emojiContainer'>

                {pickerVisible && <EmojiPicker height={500}
width={400} onEmojiClick={onEmojiClick} />}

            </div>

            <HiOutlineEmojiHappy className='HiOutlineEmojiHappy'
onClick={toggleEmojiPicker} />

        </div>

        <div className="messageInput">

            <input type="text" placeholder="Type a message"
onKeyDown={onEnterPress} value={messageText} onChange={handleChange} />

        </div>

        <div className="sendLogo">

```

```

        <AiOutlineSend className='AiOutlineSend'
onClick={sendMessage} />

      </div>

    </div>

  </div>

)

}

```

```
export default Conversation;
```

## Conversation.css (components)

```

.conversationContainer {

  display: flex;

  flex-direction: column;

  height: 100%;

  width: 70%;

  background-color: #f8f9fb;

  position: relative;

}

.conversationHeader {

  background: linear-gradient(to bottom right, #022423, #004136, #00725f,
#00977e);

```

```

    height: 8%;

    width: 100%;

    display: flex;

    align-items: center;

    justify-content: space-between;

    flex-direction: row;

    padding: 0px 1vmax;

    color: #fff;
}

```

```

.chatPerson {

    display: flex;

    align-items: center;

    flex-direction: row;

    height: 100%;

    width: 100%;

    gap: 0.8vmax;
}

```

```

.personDP {

    display: flex;

    align-items: center;

    justify-content: center;

    width: 3.5vmax;

    height: 3.5vmax;

    border: 2px solid #00BFA0;

    border-radius: 50%;
}

```

```

.personDP > img {
    height: 100%;
    width: 100%;
    object-fit: cover;
    border-radius: 50%;
    padding: 0.1vmax;
}

.personName > p {
    font-size: 1.4vmax;
    font-weight: 700;
}

.chatSection {
    height: 84%;
    width: 100%;
    display: flex;
    flex-direction: column;
    gap: 0.6vmax;
    padding: 1vmax;
    overflow: scroll;
    overflow-x: hidden;
}

.emojiContainer {
    position: absolute;
    bottom: 4.12vmax;

```

```

        left: 1vmax;
    }

.chatSection::-webkit-scrollbar {

    width: 0;
}

.sendMessageDiv {

    width: 100%;

    height: 8%;

    background: linear-gradient(to top right, #022423, #004136, #00725f,
#00977e);

    display: flex;

    flex-direction: row;

    align-items: center;

    padding: 0px 1vmax;
}

.emojiLogo {

    height: 60%;

    width: 5%;

    background-color: #ffffffbe;

    display: flex;

    align-items: center;

    justify-content: center;

    border-top-left-radius: 50px;

    border-bottom-left-radius: 50px;

    cursor: pointer;
}

```



```
}
```

```
.HiOutlineEmojiHappy {  
    font-size: 1.5vmax;  
}
```

```
.messageInput {  
    height: 60%;  
    width: 90%;  
}
```

```
.messageInput > input {  
    height: 100%;  
    width: 100%;  
    border: none;  
    outline: none;  
    font-size: 1vmax;  
    background-color: #ffffffbe;  
}
```

```
.sendLogo {  
    height: 60%;  
    width: 5%;  
    background-color: #ffffffbe;  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    border-top-right-radius: 50px;  
    border-bottom-right-radius: 50px;
```

```

        cursor: pointer;
    }

    .AiOutlineSend {
        font-size: 1.5vmax;
    }

    .startTalkingDiv {
        width: 35%;
        height: 20%;
        background-color: #000000b6;
        border-radius: 30px;
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
        display: flex;
        align-items: center;
        justify-content: center;
        color: #fff;
        font-size: 1.2vmax;
        box-shadow: 0px 0px 30px #000;
    }

```

## Message.js (components)

```
import React from 'react';

import './Message.css';

// Functional component representing a chat message

const Message = (props) => {

  // Destructuring props to extract message content and sender
  information

  const { messageContent, senderIsMe, timestamp } = props;

  const tooltipContent = timestamp.slice(11,19) + " | " +
timestamp.slice(0,10);

  // Conditional rendering based on whether the message is sent by the
  user or received from others

  if(senderIsMe){

    return (

      <div className='sentMessageDiv'>

        <div className="sentMessage msg" title={tooltipContent}>

          {messageContent}

        </div>

      </div>

    )

  }

  else{

    return (

      <div className='receivedMessageDiv'>

        <div className="receivedMessage msg"
title={tooltipContent}>

          {messageContent}


```

```

        </div>
    </div>
)
}

}

export default Message;

```

## Message.css (components)

```

.sendMessageDiv {
    display: flex;
    justify-content: flex-end;
}

.receiveMessageDiv {
    display: flex;
    justify-content: flex-start;
}

.msg {
    color: #000;
    max-width: 60%;
}

```

```

padding: 0.5vmax;

font-size: 0.9vmax;

border-radius: 6px;

font-weight: 500;

transition: 0.3s;
}

.msg:hover {

    cursor: grab;

    transform: scale(1.02);

    box-shadow: 0px 0px 40px #000;
}

.sentMessage {

    /* background-color: #00d8b4; */

    background-color: #00f3cb;
}

.receivedMessage {

    background-color: #FFFFFF;
}

```

## Login.js (components)

```
import React, { useState } from 'react';

import './Login.css';

import qrCode from './LoginQRcode.png';

import loginLogo from './LoginLogo.png';

import { Link, useNavigate } from 'react-router-dom';

import { toast } from 'react-toastify';

// Functional component representing the login page

const Login = () => {

  // State to manage user credentials

  const [credentials, setCredentials] = useState({ phoneNumber: '',
password: '', profilePic: '' });

  const navigate = useNavigate();

  // Event handler for input changes

  function onChange(e) {

    setCredentials({ ...credentials, [e.target.name]: e.target.value
});

  }

  // Async function to handle form submission

  async function handleSubmit(e) {

    e.preventDefault();

    const response = await fetch('http://localhost:3005/login', {

      method: 'POST',

      headers: {

        'Content-Type': 'application/json'
```

```

    },

    body: JSON.stringify({ phoneNumber: credentials.phoneNumber,
password: credentials.password, profilePic: credentials.profilePic })

  });

  const jsonResponse = await response.json();

  // Handling the response from the server

  if (jsonResponse.success) {

    localStorage.setItem('convoverseUserLoginId',
jsonResponse.responseData._id);

    localStorage.setItem('convoverseUserLoginName',
jsonResponse.responseData.name);

    localStorage.setItem('convoverseUserLoginProfilePic',
jsonResponse.responseData.profilePic);

    // Navigating to home page and setting fields back to empty

    setCredentials({ phoneNumber: '', password: '', profilePic: ''

  });

  navigate('/home');

  // Displaying a success toast message

  toast.success(jsonResponse.message, {

    position: "bottom-right",

    autoClose: 2000,

    hideProgressBar: false,

    closeOnClick: true,

    pauseOnHover: true,

    draggable: true,

    progress: undefined,

    theme: "colored",

  });

```

```

    }

    else {

        // Displaying an error toast message
        toast.error(jsonResponse.message, {
            position: "bottom-right",
            autoClose: 2000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
            theme: "colored",
        });
    }
}

```

```

return (
    <div className='loginPage'>
        <div className="loginTop">
            <h1>CONVOVERSE</h1>
            <h3>Chat with ease!</h3>
        </div>

        <div className="loginBottom">

        </div>

        <div className="loginCard">

```



```

    <div className="cardLeft">

        <h1>LOG IN</h1>

        <div className="line"></div>

        <form onSubmit={handleSubmit}>

            <label htmlFor="phoneNumber">Phone Number</label>

<br />

            <input type="text" name="phoneNumber"
id="phoneNumber" value={credentials.phoneNumber} onChange={onChange}
placeholder='Ex: 989162XXXX' minLength="10" maxLength="12" required /><br
/>

            <label htmlFor="password">Password</label> <br />

            <input type="password" name="password"
id="password" value={credentials.password} onChange={onChange}
placeholder='Enter your password' minLength="8" maxLength="32" required />

            <button type='submit' className='loginButton'><img
src={loginLogo} alt="Google logo" />Log in</button>

            {/* Link to navigate to the signup page */}

            <Link to='/signup' className='signupText'>Not a
user? Sign up</Link>

        </form>

    </div>

    <div className="cardRight">

        <img src={qrCode} alt="QR code" />

    </div>

</div>

</div>

)
}

```

```
export default Login;
```

## Login.css (components)

```
.loginPage {  
    height: 100vh;  
    width: 100vw;  
    background-color: #000;  
    display: flex;  
    flex-direction: column;  
    position: relative;  
}
```

```
.loginTop {  
    background: linear-gradient(to bottom right, #022423, #004136, #00725f,  
#00977e);  
    height: 40%;  
    width: 100%;  
    padding: 5vmax;  
}
```

```
.loginTop > h1 {  
    color: #fff;  
    font-size: 1.6vmax;
```

```

}

.loginTop > h3 {
    color: #fff;
    font-size: 0.8vmax;
}

.loginBottom {
    background-color: #141414;
    height: 60%;
    width: 100%;
}

.loginCard {
    position: absolute;
    top: 48%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 40%;
    height: 45%;
    background-color: #fff;
    border-radius: 15px;
    box-shadow: 0px 0px 30px #0000006b;
    display: flex;
    overflow: hidden;
}

.cardLeft {

```

```
    width: 50%;  
  
    height: 100%;  
  
    padding: 2vmax;  
}
```

```
.cardLeft > h1 {  
  
    font-size: 2vmax;  
  
    font-weight: 700;  
}
```

```
.line {  
  
    height: 1%;  
  
    width: 100%;  
  
    border-radius: 10px;  
  
    background-color: #00000094;  
  
    margin-bottom: 1vmax;  
}
```

```
form {  
  
    height: 80%;  
  
    width: 100%;  
}
```

```
.cardLeft > form > label {  
  
    font-size: 1.5vmax;  
}
```

```
.cardLeft > form > input {
```

```

    font-size: 1vmax;

    background-color: rgba(231, 231, 231, 0.678);

    width: 100%;

    outline: none;

    border: none;

    padding: 0.5vmax;

    margin-bottom: 0.5vmax;
}

```

```

.cardLeft > form > .loginButton {

    height: 20%;

    width: 50%;

    border-radius: 50px;

    border: none;

    outline: none;

    background-color: #fff;

    color: #000;

    display: flex;

    align-items: center;

    justify-content: center;

    gap: 0.5vmax;

    box-shadow: 0px 0px 10px #0000004d;

    margin-top: 0.8vmax;

    margin-bottom: 1vmax;

    font-size: 1vmax;

    font-weight: 700;

    transition: 0.3s;
}

```

```
.cardLeft > form > .loginButton:hover {  
    cursor: grab;  
    box-shadow: 0px 0px 10px #000000c5;  
}
```

```
.cardLeft > form > .loginButton:active {  
    cursor: grabbing;  
}
```

```
.cardLeft > form > .loginButton > img {  
    height: 50%;  
    width: 20%;  
    object-fit: contain;  
}
```

```
.signupText {  
    color: #00977e;  
    font-size: 1vmax;  
    font-weight: 600;  
    text-decoration: none;  
    transition: 0.3s;  
}
```

```
.signupText:hover {  
    color: #006b59;  
    cursor: pointer;  
}
```

```
.cardRight {  
  
  width: 50%;  
  
  height: 100%;  
  
  background-color: #fff;  
  
}
```

```
.cardRight > img {  
  
  width: 100%;  
  
  height: 100%;  
  
  object-fit: contain;  
  
}
```

## Signup.js (components)

```
import React, { useState } from 'react';  
  
import './Signup.css';  
  
import qrCode from './LoginQRcode.png';  
  
import signupLogo from './SignupLogo.png';  
  
import { Link, useNavigate } from 'react-router-dom';  
  
import { toast } from 'react-toastify';  
  
  
// Functional component representing the signup page  
  
const Signup = () => {
```

```

// State to manage user credentials

const [credentials, setCredentials] = useState({ phoneNumber: '', name:
'', password: '', profilePic:'' });

const navigate = useNavigate();

// Event handler for input changes

function onChange(e) {

    setCredentials({ ...credentials, [e.target.name]: e.target.value
});

}

// Async function to handle form submission

async function handleSubmit(e) {

    e.preventDefault();

    const response = await fetch('http://localhost:3005/user', {

        method: 'POST',

        headers: {

            'Content-Type': 'application/json'

        },

        body: JSON.stringify({ phoneNumber: credentials.phoneNumber,
name: credentials.name, password: credentials.password, profilePic:
credentials.profilePic })

    });

    const jsonResponse = await response.json();

    setCredentials({ phoneNumber: '', name: '', password: '',
profilePic:'' });

// Handling the response from the server

if (jsonResponse.success) {

```



```

        // Displaying a success toast message and navigating to the
login page

        toast.success(jsonResponse.message, {

            position: "bottom-right",

            autoClose: 2000,

            hideProgressBar: false,

            closeOnClick: true,

            pauseOnHover: true,

            draggable: true,

            progress: undefined,

            theme: "colored",

        });

        // Navigating back to login page

        navigate('/');

    }

    else {

        // Displaying a warning toast message if there are validation
errors

        toast.warning(jsonResponse.responseData[0].message, {

            position: "bottom-right",

            autoClose: 2000,

            hideProgressBar: false,

            closeOnClick: true,

            pauseOnHover: true,

            draggable: true,

            progress: undefined,

            theme: "colored",

        });

    }

```

```

    }

    return (
      <div className='signupPage'>
        <div className="signupTop">
          <h1>CONVOVERSE</h1>
          <h3>Chat with ease!</h3>
        </div>

        <div className="signupBottom">

        </div>

        <div className="signupCard">
          <div className="cardLeft">
            <h1>SIGN UP</h1>
            <div className="line"></div>
            <form onSubmit={handleSubmit}>
              <label htmlFor="phoneNumber">Phone Number</label>
<br />
              <input type="text" name="phoneNumber"
id="phoneNumber" value={credentials.phoneNumber} onChange={onChange}
placeholder='Ex: 989162XXXX' minLength="10" maxLength="12" required /><br
/>

              <label htmlFor="name">Name</label> <br />
              <input type="text" name="name" id="name"
value={credentials.name} onChange={onChange} placeholder='Ex: George Smith'
required /><br />
              <label htmlFor="password">Password</label> <br />

```

```

        <input type="password" name="password"
id="password" value={credentials.password} onChange={onChange}
placeholder='Must be 8 characters' required /><br />

        <label htmlFor="profilePic">Profile Pic</label> <br
/>

        <input type="text" name="profilePic"
id="profilePic" value={credentials.profilePic} onChange={onChange}
placeholder='URL to Profile Pic' required />

        <button type='submit' className='signupButton'><img
src={signupLogo} alt="Google logo" />Sign up</button>

        {/* Link to navigate to the login page */}

        <Link to="/" className='loginText'>Already a user?
Log in</Link>

    </form>

</div>

<div className="cardRight">

    <img src={qrCode} alt="QR code" />

</div>

</div>

</div>

)

}

export default Signup;

```

## Signup.css (components)

```
.signupPage {  
    height: 100vh;  
    width: 100vw;  
    background-color: #000;  
    display: flex;  
    flex-direction: column;  
    position: relative;  
}
```

```
.signupTop {  
    background: linear-gradient(to bottom right, #022423, #004136, #00725f,  
#00977e);  
    height: 40%;  
    width: 100%;  
    padding: 5vmax;  
}
```

```
.signupTop > h1 {  
    color: #fff;  
    font-size: 1.6vmax;  
}
```

```
.signupTop > h3 {  
    color: #fff;  
    font-size: 0.8vmax;  
}
```

```

.signupBottom {

    background-color: #141414;

    height: 60%;

    width: 100%;

}

.signupCard {

    position: absolute;

    top: 48%;

    left: 50%;

    transform: translate(-50%, -50%);

    width: 40%;

    height: 62%;

    background-color: #fff;

    border-radius: 15px;

    box-shadow: 0px 0px 30px #0000006b;

    display: flex;

    overflow: hidden;

}

.cardLeft {

    width: 50%;

    height: 100%;

    padding: 2vmax;

}

.cardLeft > h1 {

    font-size: 2vmax;

```

```

    font-weight: 700;
}

.line {
    height: 1%;
    width: 100%;
    border-radius: 10px;
    background-color: #00000094;
    margin-bottom: 1vmax;
}

.cardLeft > form > label {
    font-size: 1.2vmax;
}

.cardLeft > form > input {
    font-size: 0.8vmax;
    background-color: rgba(231, 231, 231, 0.678);
    width: 100%;
    outline: none;
    border: none;
    padding: 0.5vmax;
    margin-bottom: 0.5vmax;
}

.cardLeft > form > .signupButton {
    height: 14%;
    width: 50%;

```

```

border-radius: 50px;

border: none;

outline: none;

background-color: #fff;

color: #000;

display: flex;

align-items: center;

justify-content: center;

gap: 0.5vmax;

box-shadow: 0px 0px 10px #0000004d;

margin-top: 0.8vmax;

margin-bottom: 1vmax;

font-size: 1vmax;

font-weight: 700;

transition: 0.3s;
}

.cardLeft > form > .signupButton:hover {

  cursor: grab;

  box-shadow: 0px 0px 10px #000000c5;
}

.cardLeft > form > .signupButton:active {

  cursor: grabbing;
}

.cardLeft > form > .signupButton > img {

  height: 50%;
}

```

```

        width: 20%;

        object-fit: contain;
    }

    .loginText {
        color: #00977e;

        font-size: 1vmax;

        font-weight: 600;

        text-decoration: none;

        transition: 0.3s;
    }

    .loginText:hover {
        color: #006b59;

        cursor: pointer;
    }

    .cardRight {
        width: 50%;

        height: 100%;

        background-color: #fff;
    }

    .cardRight > img {
        width: 100%;

        height: 100%;

        object-fit: contain;
    }

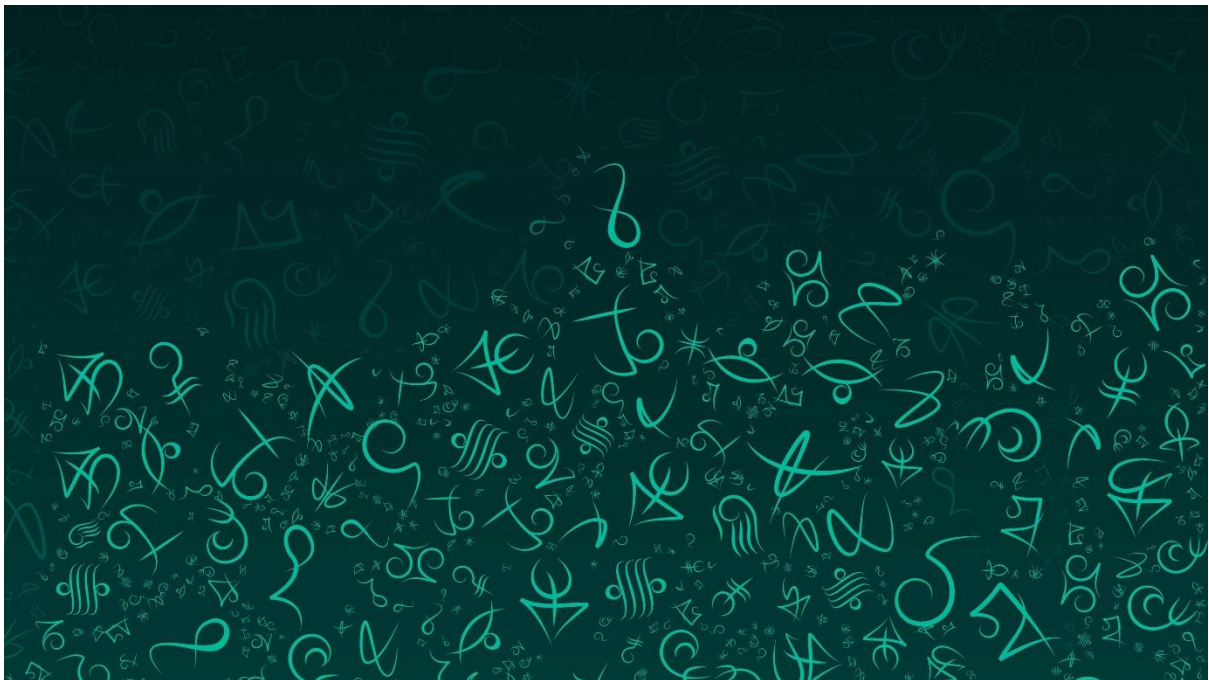
```



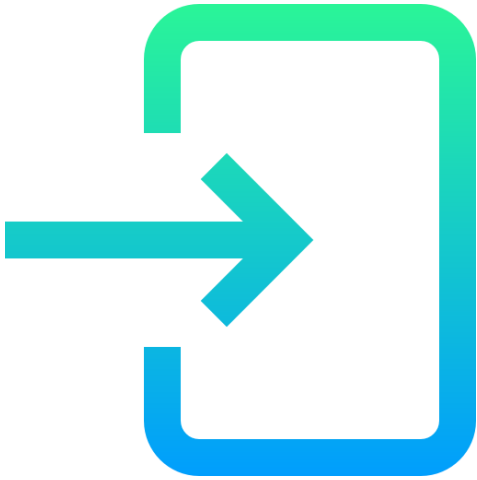
### **ChatSectionPlaceholderImage.png (components)**



### **ConversationWallpaper.png (components)**



**LoginLogo.png (components)**



**SignupLogo.png (components)**

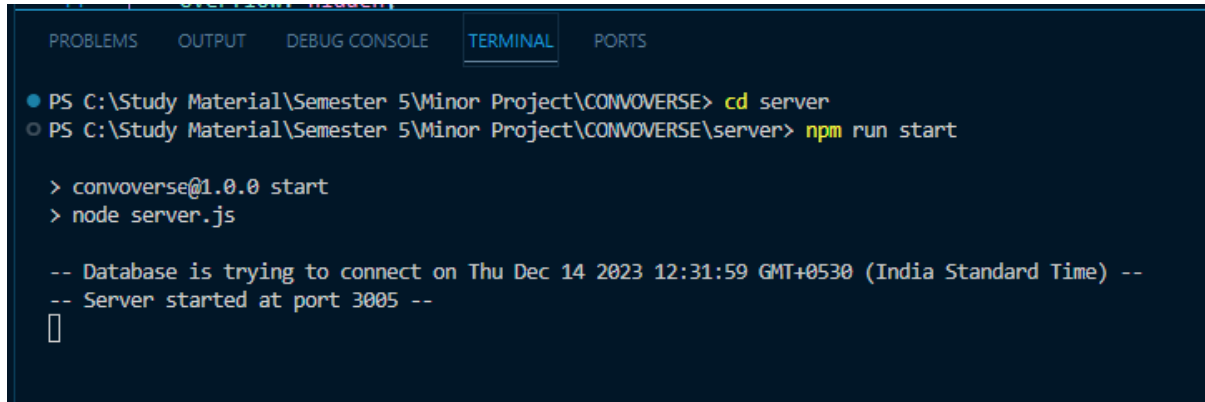


QR.png (components)



# OUTPUT SCREENSHOTS

## Running the application

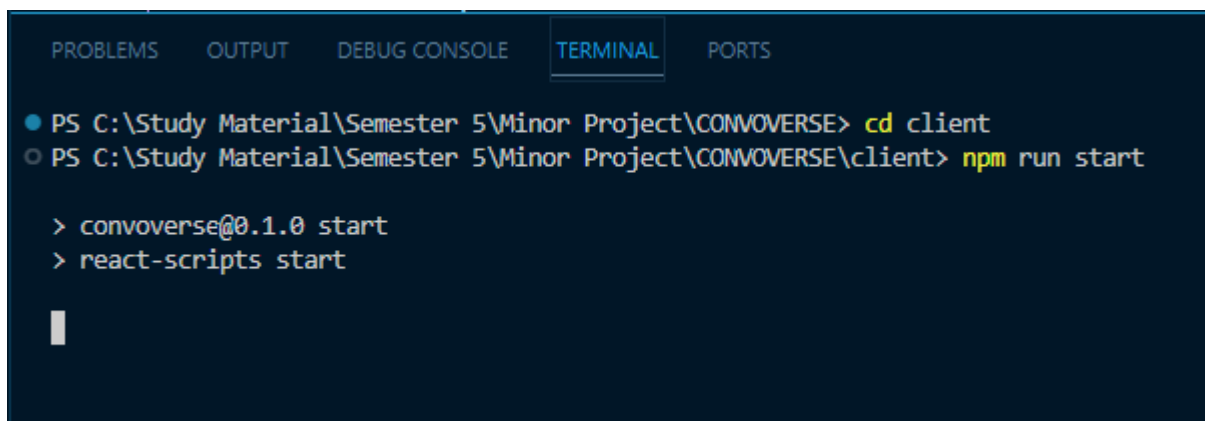


This screenshot shows the VS Code terminal interface with the 'TERMINAL' tab selected. The terminal displays the following commands and output:

```
PS C:\Study Material\Semester 5\Minor Project\CONVOVERSE> cd server
PS C:\Study Material\Semester 5\Minor Project\CONVOVERSE\server> npm run start

> convoverse@1.0.0 start
> node server.js

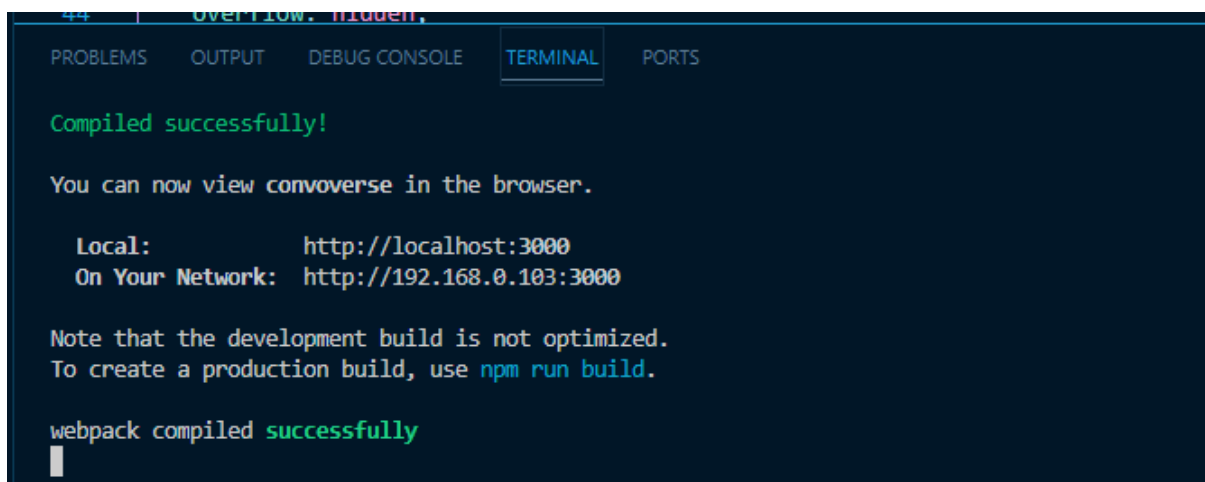
-- Database is trying to connect on Thu Dec 14 2023 12:31:59 GMT+0530 (India Standard Time) --
-- Server started at port 3005 --
```



This screenshot shows the VS Code terminal interface with the 'TERMINAL' tab selected. The terminal displays the following commands and output:

```
PS C:\Study Material\Semester 5\Minor Project\CONVOVERSE> cd client
PS C:\Study Material\Semester 5\Minor Project\CONVOVERSE\client> npm run start

> convoverse@0.1.0 start
> react-scripts start
```



This screenshot shows the VS Code terminal interface with the 'TERMINAL' tab selected. The terminal displays the following output:

```
Compiled successfully!

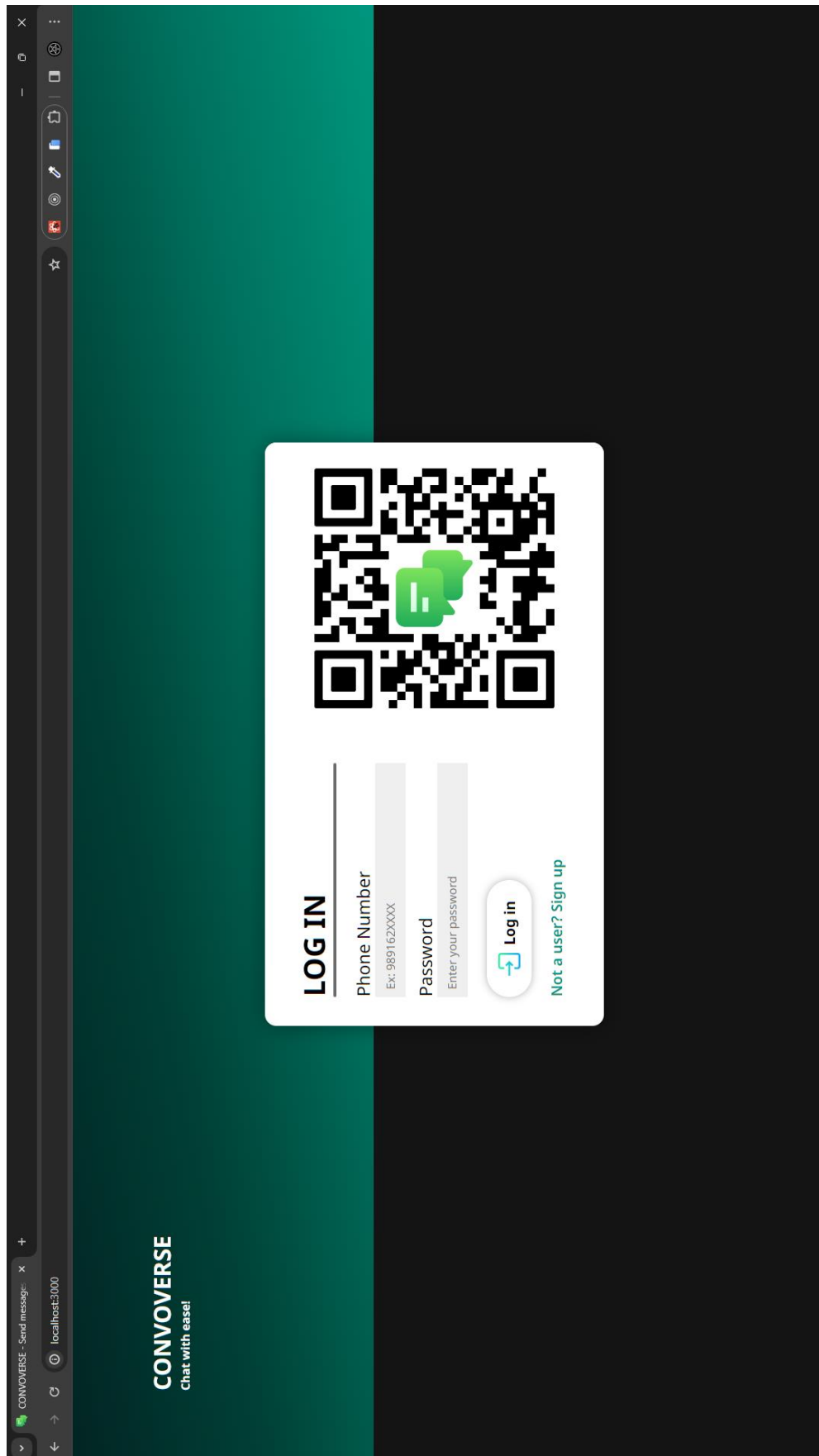
You can now view convoverse in the browser.

Local:      http://localhost:3000
On Your Network: http://192.168.0.103:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

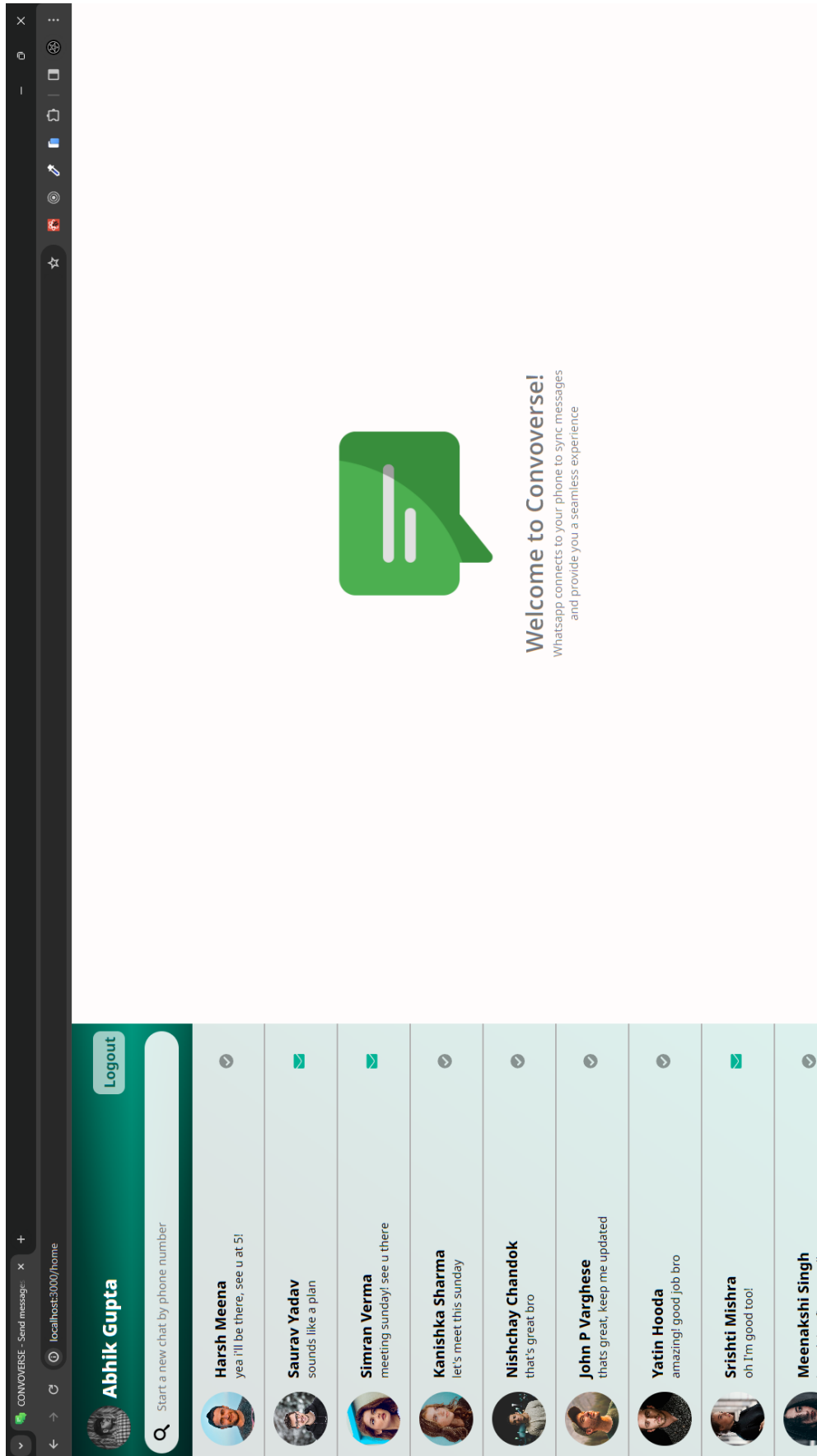
## Login Page



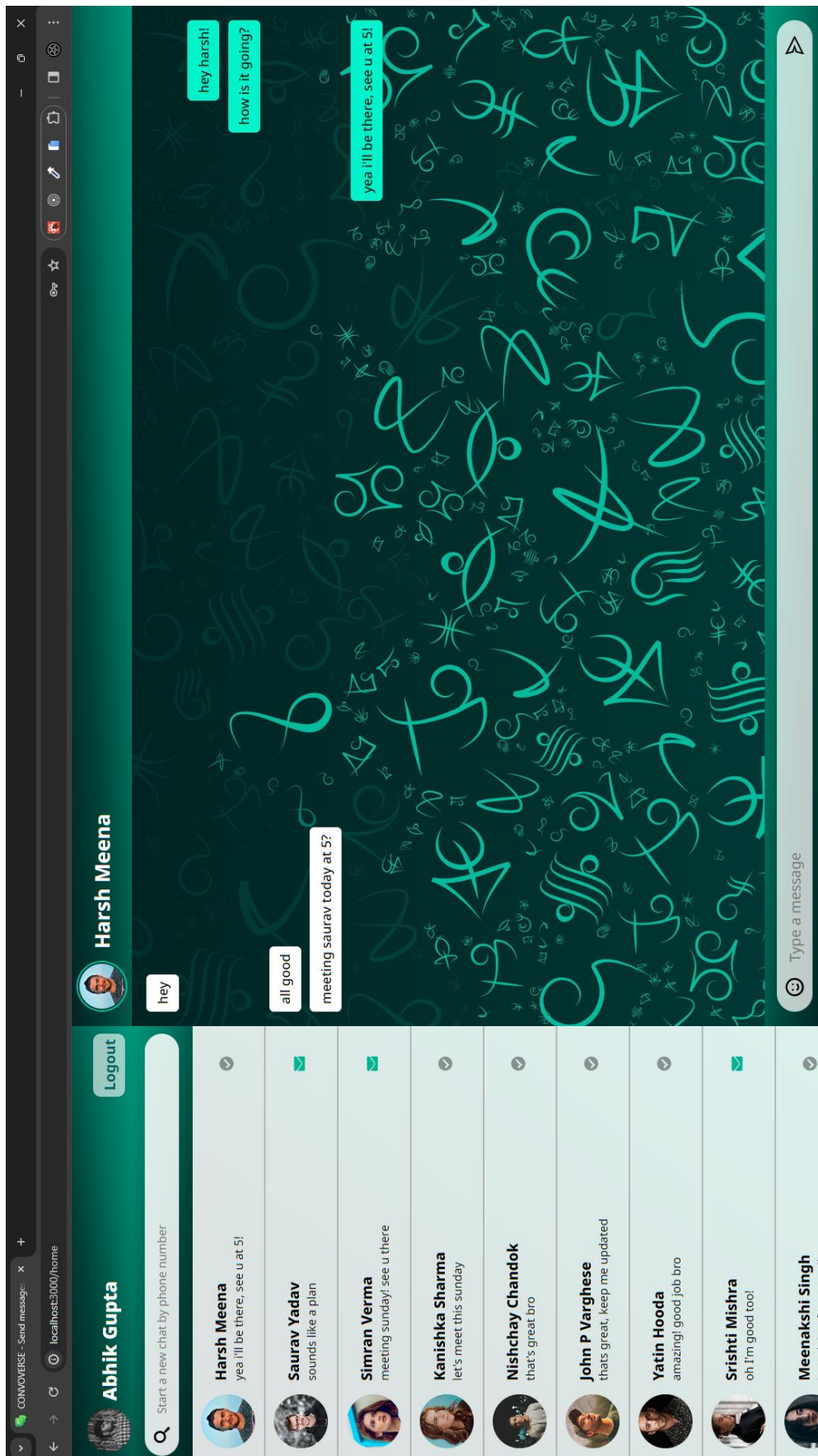
## Signup Page



## Home Page

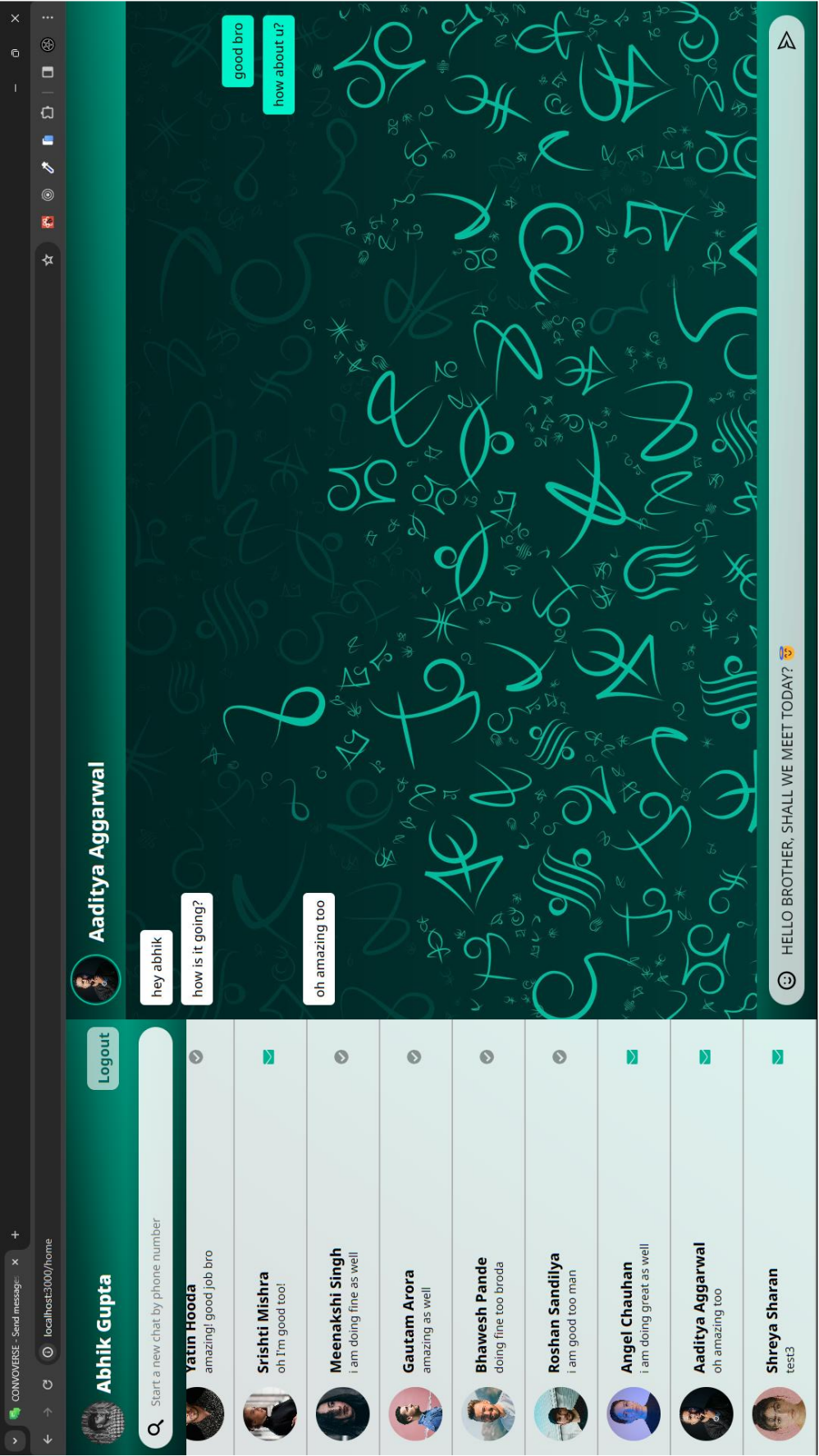


## After opening up a chat

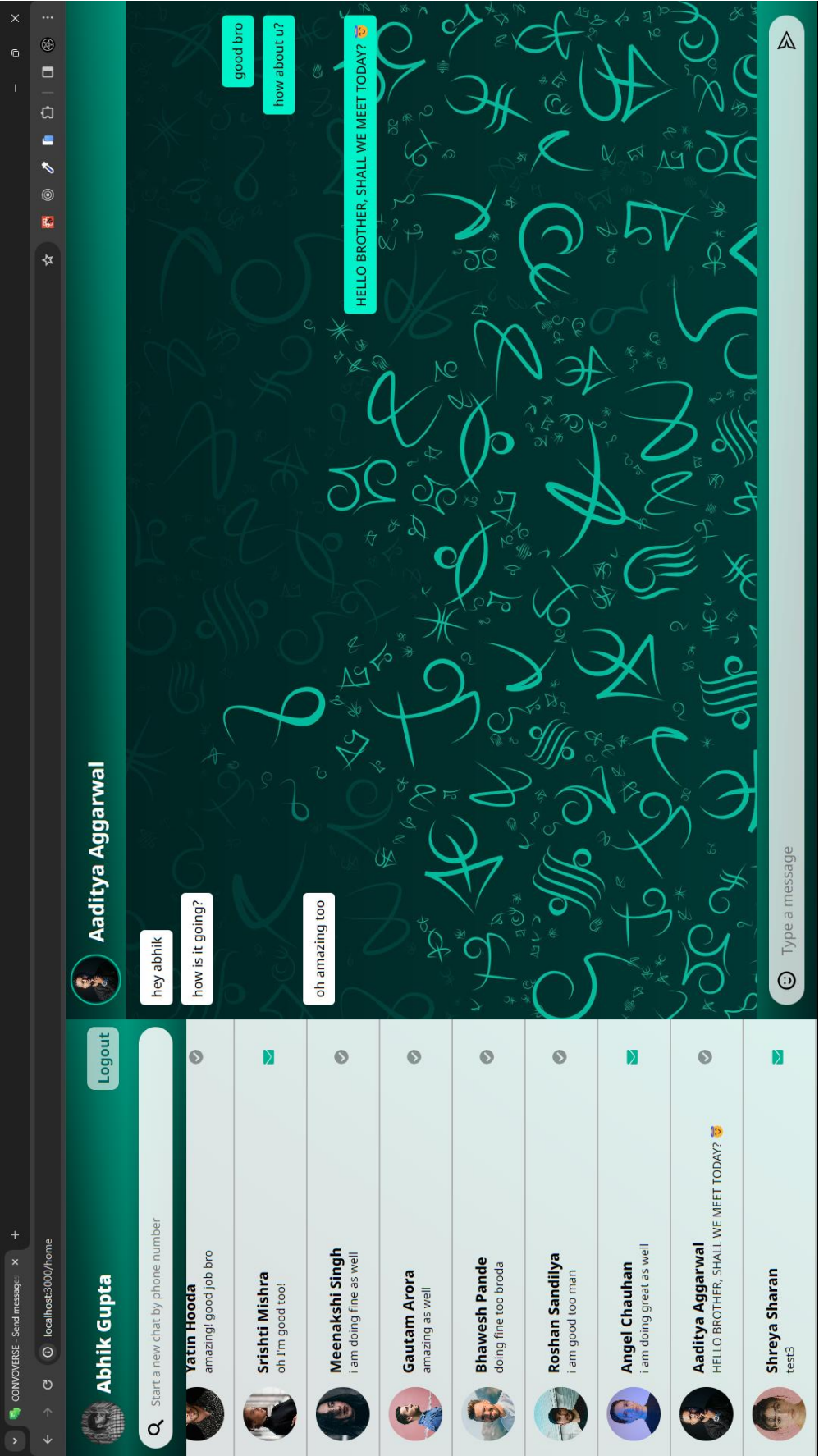




Writing a new message



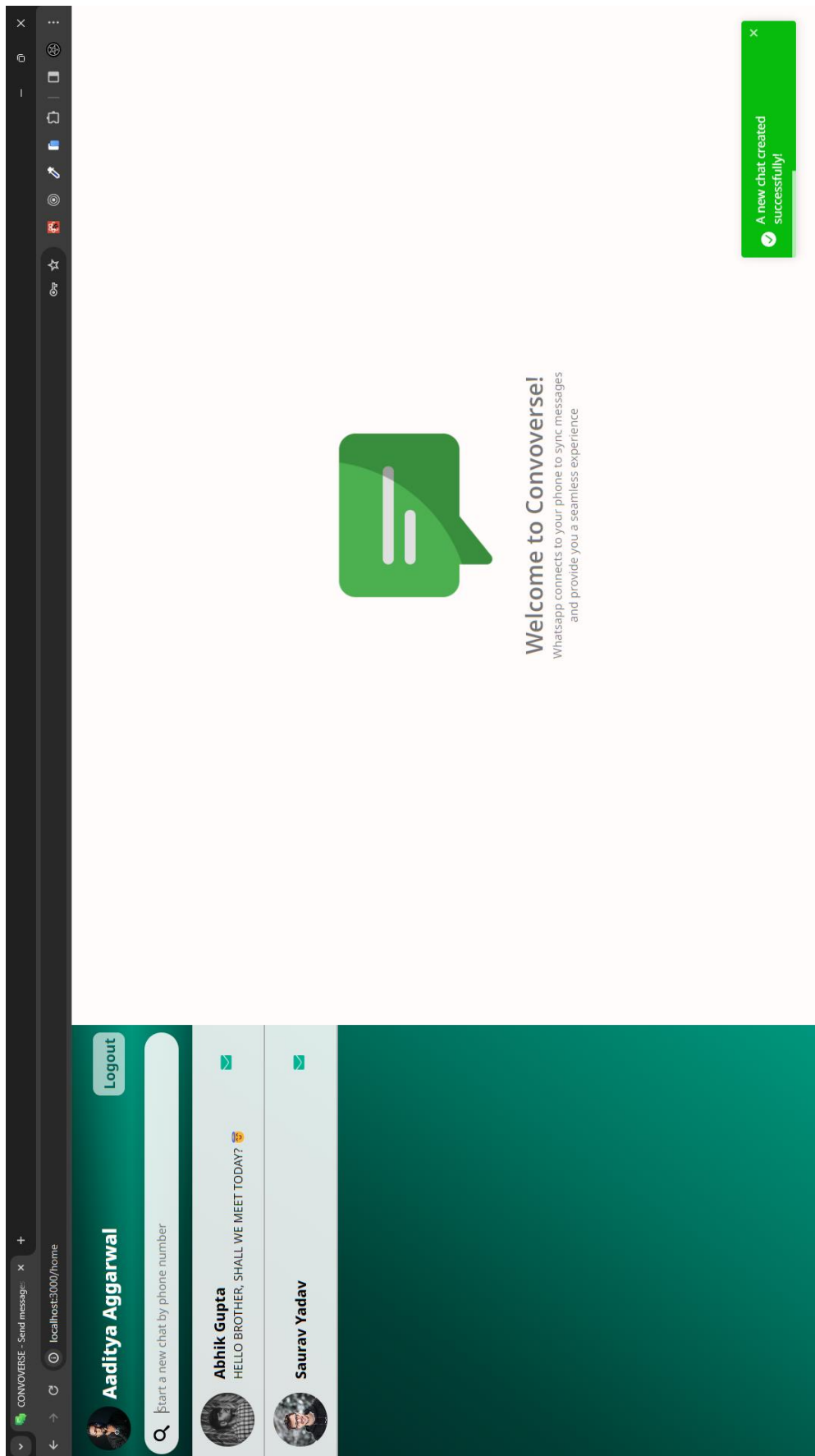
Sending the new message



## Searching for a new user

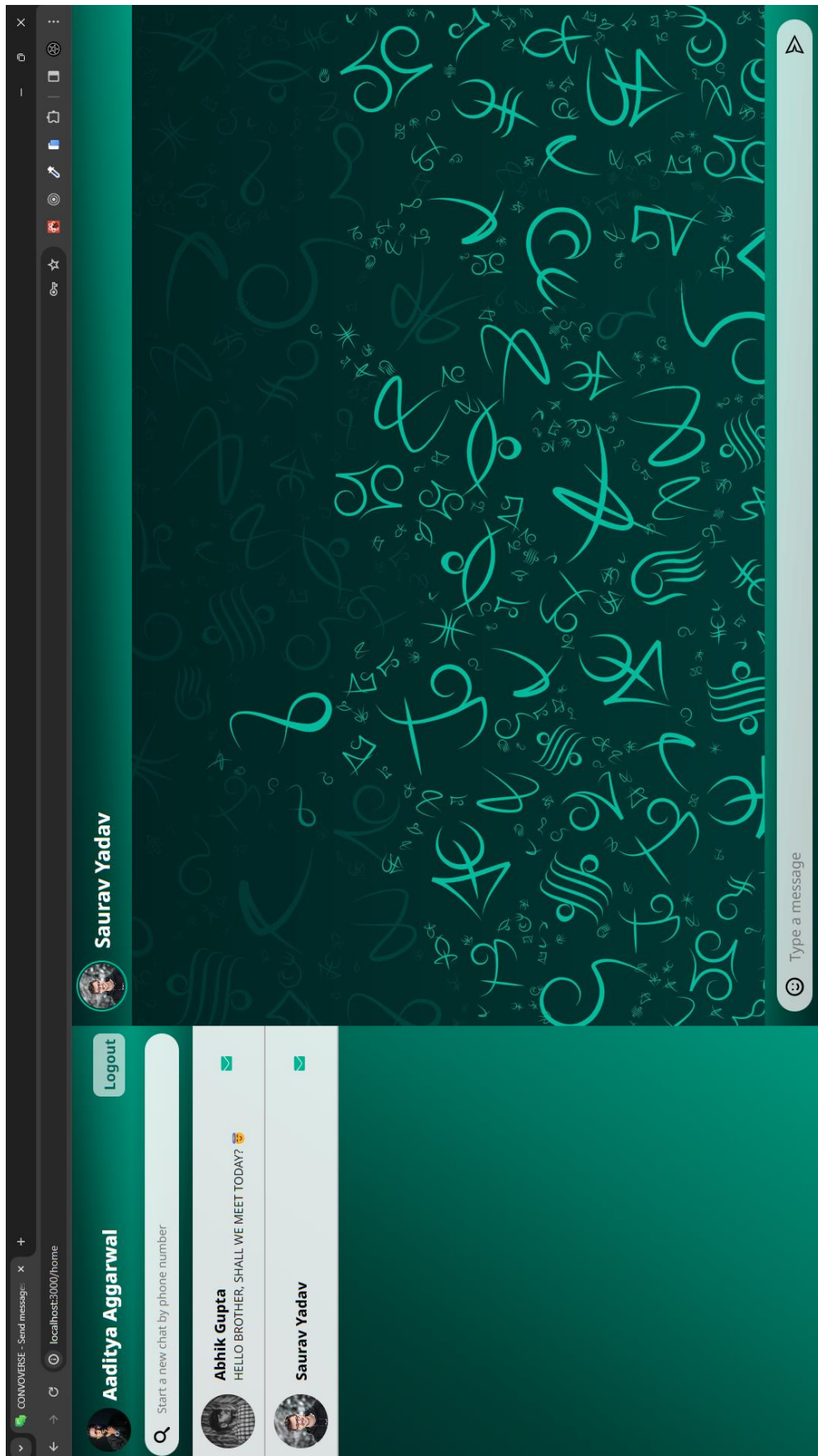


## Creating a new chat

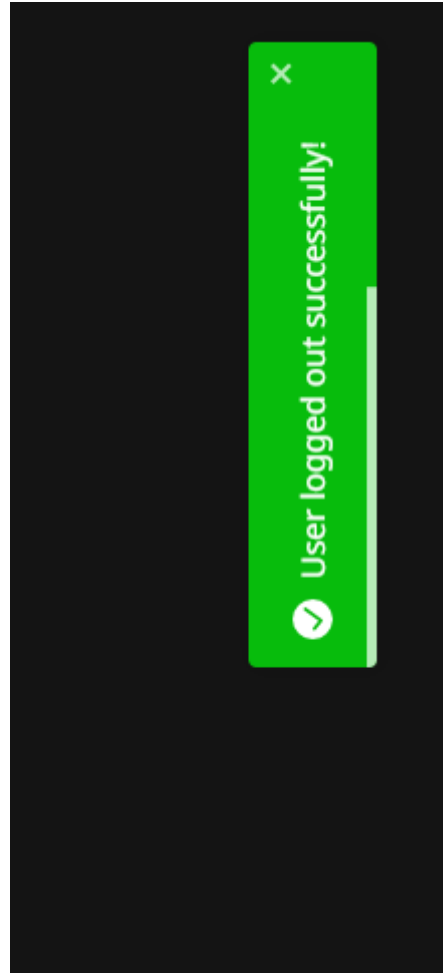
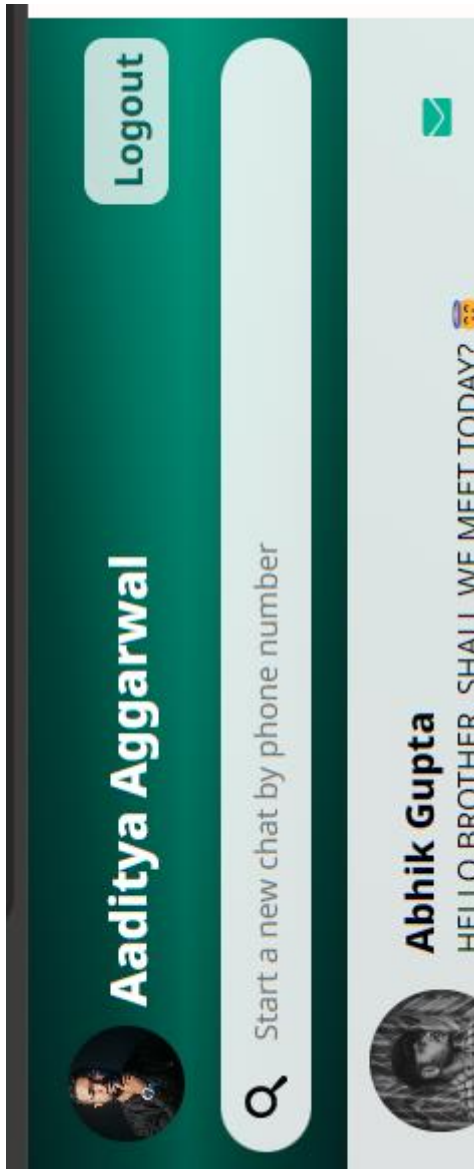




## An empty chat created



## Logging out



# VERSION CONTROL AND COLLABORATION

Convoverse, a collaborative effort by Abhik Gupta, Yatin Hooda, John P. Varghese, and Nishchay Chandok, takes the world of chat web applications by storm. Built using the powerful MERN stack, Convoverse provides a seamless and engaging chat experience.

From its humble beginnings as a mock-data-driven frontend in version 1.0, Convoverse has evolved into a fully functional chat platform. The team meticulously crafted each version, adding functionalities and refining the user experience.

Version 2.0 laid the foundation for real-time communication by introducing mock APIs. While conversations weren't happening yet, the infrastructure was in place for future development.

Version 3.0 marked a major turning point. The team implemented the actual controller and API logic, enabling users to send and receive real messages. This version brought Convoverse one step closer to its final form.

The meticulous integration between the frontend and backend took place between version 3.1 and 3.6. This phased approach ensured a smooth and stable integration, avoiding potential complications.

Version 4.0 focused on documentation and code comments. This step was crucial for future maintenance and allowed others to understand the app's inner workings.

Finally, version 5.0 saw the culmination of the team's efforts. This version boasts a polished UI, bug fixes, and enhanced functionality, making Convoverse the user-friendly chat platform it is today.

Convoverse serves as a testament to the power of collaboration and innovation. The team's dedication and perseverance resulted in a compelling chat application that stands out in the digital landscape.

[Visit the Github Repository](#)



# Our Github Repository

CONVERSE - Send message

abhi2207/CONVERSE

github.com/abhi2207/CONVERSE

abhi2207 / CONVERSE

Code

Issues

Pull requests

Actions

Wiki

Projects

Insights

Security

Settings

CONVERSE

Public

master 1 Branch 0 Tags

Unpin

Unwatch 1

Fork 0

Star 0

20 Commits

ae58bc4 · 11 hours ago

Go to file

Add file

Code

CONVERSE is a real time chat application, and is submitted as our 5th semester minor project.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

JavaScript 71.6%

CSS 27.0%

version-1.0

Tenth commit

last week

version-2.0

Tenth commit

last week

version-3.0

Tenth commit

last week

version-3.1

Fifth commit

2 weeks ago

version-3.2

Seventh commit

2 weeks ago

version-3.3

Eighth commit

2 weeks ago

version-3.4

Ninth commit

last week

version-3.5

Tenth commit

last week

version-3.6

Thirteenth commit

2 days ago

version-4.0

Fourteenth commit

14 hours ago

version-5.0

Eighteenth commit

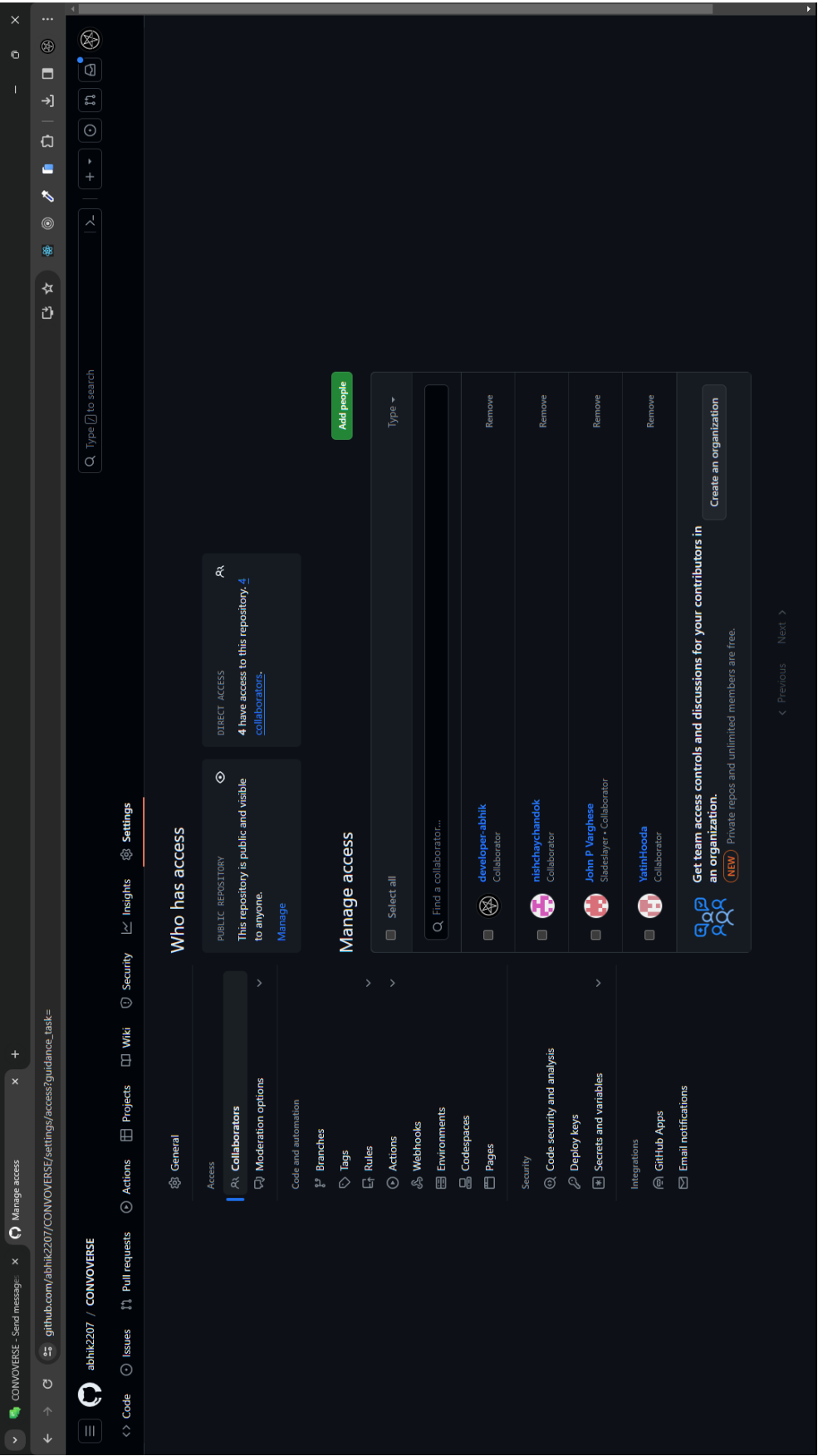
11 hours ago

FINAL REPORT.docx

Twentieth commit

11 hours ago

# All collaborators



# CONCLUSION

Convoverse, a groundbreaking real-time chat application developed on the MERN stack, signifies a paradigm shift in online communication. Rooted in the ambition to transcend conventional messaging platforms, Convoverse prioritizes not only the immediacy of communication but also the versatility and depth of interactions. Its user-centric design, seamless navigation, and commitment to facilitating both professional collaborations and personal connections distinguish it as a comprehensive platform for diverse communication needs. Leveraging the robust capabilities of MongoDB, Express.js, React.js, and Node.js, Convoverse aspires to set a new standard for online dialogue, ensuring a responsive and engaging experience across various contexts.

The scope of Convoverse extends beyond mere text-based conversations, encompassing a broad spectrum of communication methods within its MERN-based architecture. Real-time multimedia sharing and the integration of collaborative tools showcase its commitment to providing a holistic communication hub. The flexibility of the MERN stack enables Convoverse to scale efficiently, accommodating an expanding user base while maintaining optimal performance. With continuous improvement at its core, Convoverse plans regular updates and the integration of emerging technologies, positioning itself as a go-to platform for individuals seeking a modern, responsive, and feature-rich chat application.

Convoverse, with its sleek interface and robust features, introduces a new era of interactive and secure online dialogue. Beyond conventional messaging, it seamlessly connects individuals

globally, fostering dynamic conversations and facilitating instant connections. The project's comprehensive hardware and software requirements ensure optimal performance, scalability, and user accessibility, making Convoverse a versatile and robust platform for diverse communication needs. Whether for work-related collaborations or personal interactions, Convoverse stands as a testament to innovation and adaptability in the realm of web application development, setting the stage for a transformative and user-centric communication experience.

# FUTURE SCOPE

As Convoverse continues to evolve, several exciting avenues open up for enhancing the user experience and expanding the functionality of the application:

1. **Real-Time Communication with WebSockets:** Implementing WebSockets will elevate Convoverse to a truly real-time communication platform. This feature will ensure instantaneous message delivery and a more interactive user experience.
2. **Group Chats in Addition to Individual Chats:** Introducing group chat functionality will enable users to engage in collaborative conversations with multiple participants. This expansion of chat capabilities will cater to both personal and professional communication needs.
3. **Website Hosting:** Taking Convoverse live on the internet by hosting the website will make it accessible to users worldwide. This step opens up possibilities for a broader user base and increased engagement.
4. **Migration to a Remote Database:** Moving from a local database to a remote database offers advantages in terms of scalability, reliability, and accessibility. Cloud-based databases, such as MongoDB Atlas, could be considered for seamless integration.

5. **Enhanced Mobile-Friendly Design:** Optimizing the user interface for mobile devices ensures a consistent and user-friendly experience across various platforms. Responsive design improvements will make Convoverse more accessible to users on smartphones and tablets.
6. **Implementing End-to-End Encryption:** Prioritizing user security, the implementation of end-to-end encryption for messages will provide an additional layer of privacy. This feature ensures that only the intended recipients can decrypt and read the messages, enhancing overall data security.

# REFERENCES

1. MongoDB: MongoDB. (n.d.). Retrieved from <https://www.mongodb.com/>
2. Express.js: Express.js. (n.d.). Retrieved from <https://expressjs.com/>
3. React.js: React.js. (n.d.). Retrieved from <https://reactjs.org/>
4. Node.js: Node.js. (n.d.). Retrieved from <https://nodejs.org/>
5. Mongoose: Mongoose. (n.d.). Retrieved from <https://mongoosejs.com/>
6. Stack Overflow: Stack Overflow. (n.d.). Retrieved from <https://stackoverflow.com/>
7. ChatGPT: ChatGPT Documentation. (n.d.). Retrieved from <https://platform.openai.com/docs/guides/chat>
8. JavaScript: JavaScript. (n.d.). Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
9. React router dom: React router dom documentation retrieved from <https://reactrouter.com/en/main/start/tutorial>
10. React toastify: React's toastify library's documentation retrieved from <https://fkhadra.github.io/react-toastify/introduction/>
11. Styled components: Styled components library's documentation retrieved from <https://styled-components.com/>
12. React icons: React icons library's documentation retrieved from <https://react-icons.github.io/react-icons/>

