

How to set up and run your prototype??

Step 1: Fetching the files required to run the prototype using command “ git clone <https://github.com/Ashwin454/IBY-s-Chatmaster> ”. This command will get a folder named “IBY-s-Chatmaster”.

Step 2: Get inside the “IBY-s-Chatmaster” folder and there you see 2 folders inside it

- 1) Backend
- 2) Chatmaster

Step 3: First get inside backend folder and run command “ **npm install** “. This will install all the dependencies required to run backend.

Step 4: Then get inside chatmaster folder and run command “ **npm install** “. This will install all the dependencies required to run frontend.

Step 5: It would be helpful if these folders are opened in two different terminals.

Step 6: Ensure that there are no processes running on ports 27017(for MongoDB), 8000(for backend) and 3000(for frontend).

Step 7: Now we run command “**nodemon server.js** ” in backend folder and first “**npm run build** ” and then “**npm start**” in chatmaster folder.

Step 8: Now we type “ **localhost:3000** ” on the browser and we get to see the application.

What dependencies I used and why?

1. I used basic frameworks of javascript like express.js, node.js, mongoose, next.js to make the app.
2. On top of that, I used socket.io for real time message transfer.
3. I used jwt, bcrypt, crypto, nodemailer for making authentication sytem stronger.
4. I used cloundinary and express-fileupload for uploading profile pics and showing them to the use.
5. I used a very cool UI library called ChakraUI and I used various components of that.
6. I used redux, react-redux, axios and Context for making communication between frontend and backend and maintaining states.

System Design Document

1. Overview

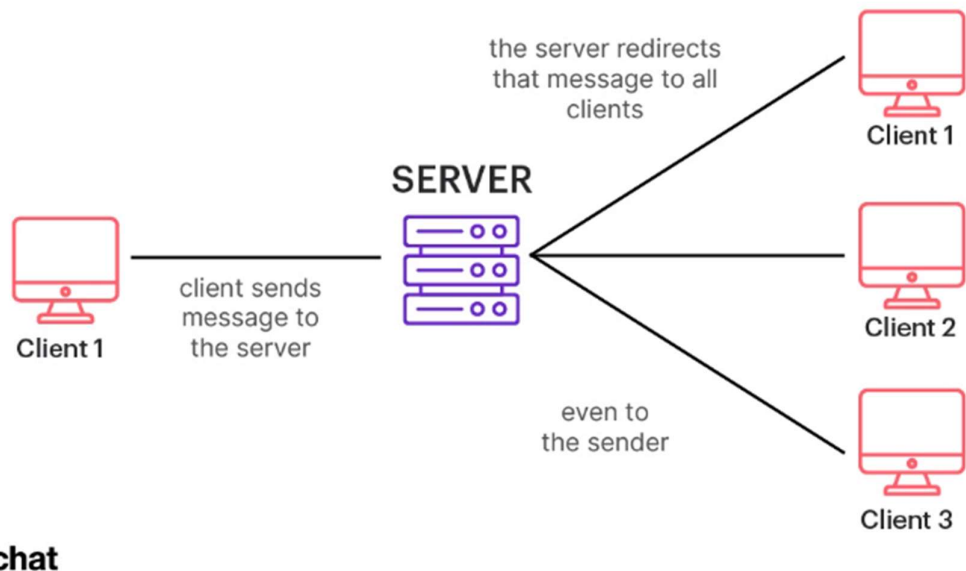
System Name: IBY's ChatMaster

Purpose: To provide a real-time chat platform where users can sign up, log in using their email via private chats and Group chats. It supports features like user authentication, chat history, email verification, and more.

Technology used:

1. **Frontend:** React (Next.js)
2. **Backend:** Node.js with Express
3. **Database:** MongoDB
4. **Authentication:** JWT, Nodemailer and encryption libraries
5. **Real time Chat:** Socket.io
6. **User Interface:** ChakraUI

2. System Architecture



As we can see in the diagram above, we simply use a server to establish real time communication between different clients. But how??

We have three database collections

1. User: Stores details of individual users and like name, email, password and picture.

```
▶ {
  _id: ObjectId('66e679c979277f6d802b513b')
  name: "Ashwin Jaiswal"
  email: "ashwin.aj4545@gmail.com"
  phone: 9125981226
  password: "$2a$10$w[redacted]a/S9jcRFDcgN6o5hITLiz/jPL5Vqce"
  is_verified: true
  roles: Array (1)
  __v: 0
  pic: "https://via.placeholder.com/150/000000/FFFFFF/?text=Profile"
  updatedAt: 2024-09-18T05:25:04.672+00:00
}
```

2. Chat: Stores every type of chat whether group or private and contains list of users in the chat and if chat is group chat, it contains information like admin information and chat name.

```
▶ {
  _id: ObjectId('66ebb75ec2e4e8d83394badb')
  chatName: "single"
  isGroupChat: false
  users: Array (2)
  createdAt: 2024-09-19T05:32:14.911+00:00
  updatedAt: 2024-09-20T06:22:00.280+00:00
  __v: 0
  latestMessage: ObjectId('66ed1488d01c6a3aab06f5cb')
}
```

3. Message: Store sender, content and Chat in which this message is sent.

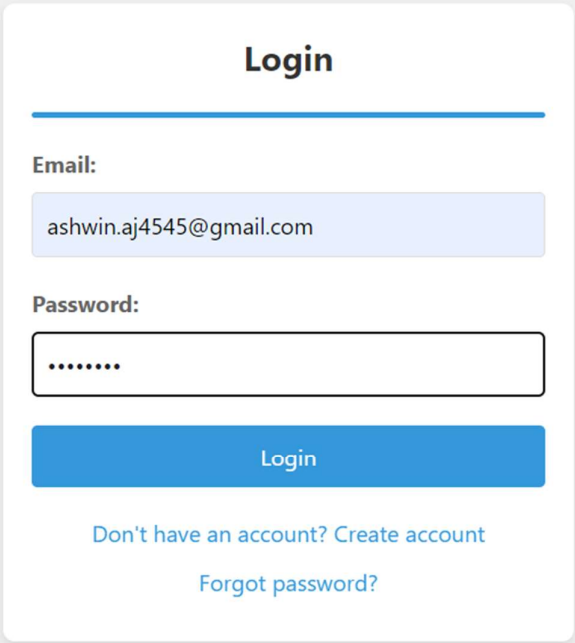
```
{
  _id: ObjectId('66ebc6bdf8ce580363f33677')
  sender: ObjectId('66e679c979277f6d802b513b')
  content: "Hi, I am Ashwin"
  chat: ObjectId('66ebb75ec2e4e8d83394badb')
  createdAt: 2024-09-19T06:37:49.142+00:00
  updatedAt: 2024-09-19T06:37:49.142+00:00
  __v: 0
}
```

Using this architecture we build an app that helps user do real time chatting.

3. APIs

User APIs: This app contains APIs for

1. registering/creating a user,
2. authenticate the user using email verification by sending OTP,
3. making a user log in and keeping the user logged in for 5 days even if the user closes a session,
4. helping a user set new password through sending OTP if he/she forgets.



A login form UI mockup centered on a light gray background. The form is a white rounded rectangle with a blue header bar containing the word "Login" in white. Below the header, there are two input fields: "Email:" with a light blue border and "Password:" with a white border. The email field contains the text "ashwin.aj4545@gmail.com" and the password field contains seven dots. Below the password field is a blue "Login" button. At the bottom of the form, there are two links: "Don't have an account? Create account" and "Forgot password?", both in blue text.

Login

Email:

ashwin.aj4545@gmail.com

Password:

.....

Login

[Don't have an account? Create account](#)

[Forgot password?](#)

5. Search for a user on the basis of name/ email

Search Users



Arpan Goswami
Email : arpang@iitbhlai.ac.in



Sarthak Jaiswal
Email : ashwin.aj8008@gmail.com



Sarthak Jaiswal
Email : ashwin.505@gmail.com



Sarthak Jaiswal
Email : ashwin1@gmail.com



Sarthak Jaiswal
Email : ashwin2@gmail.com



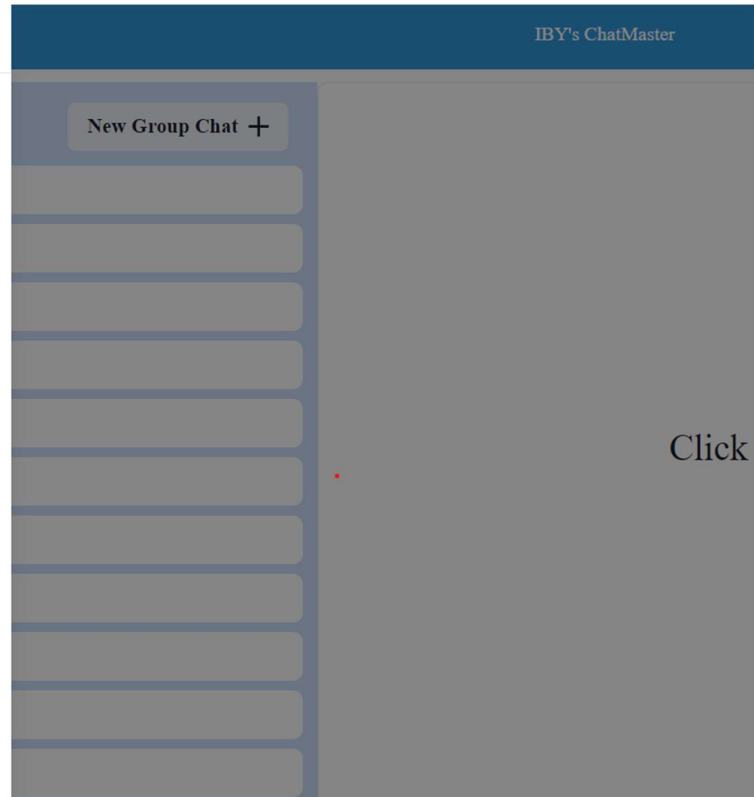
Sarthak Jaiswal
Email : ashwin3@gmail.com



Sarthak Jaiswal
Email : ashwin4@gmail.com

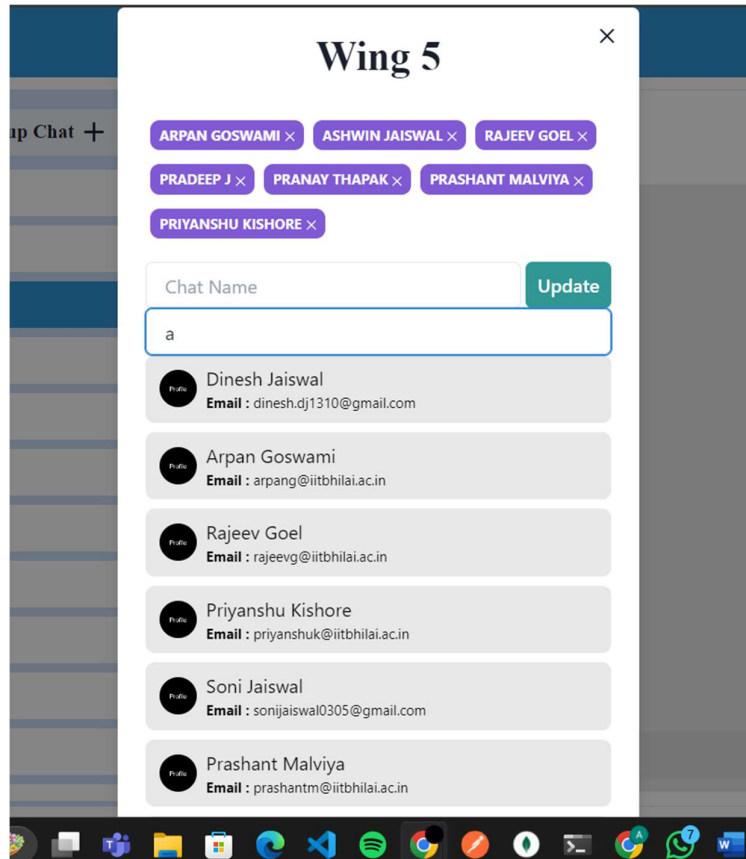


Rajesh kumar
Email : ibyschatmaster@gmail.com



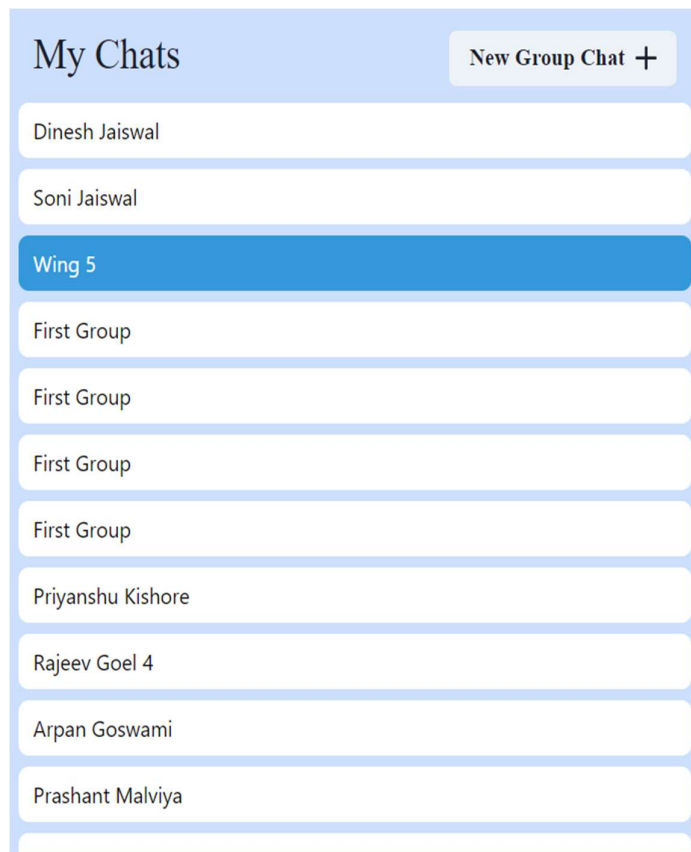
Chat APIs:

1. for creating a private chat or Group chat



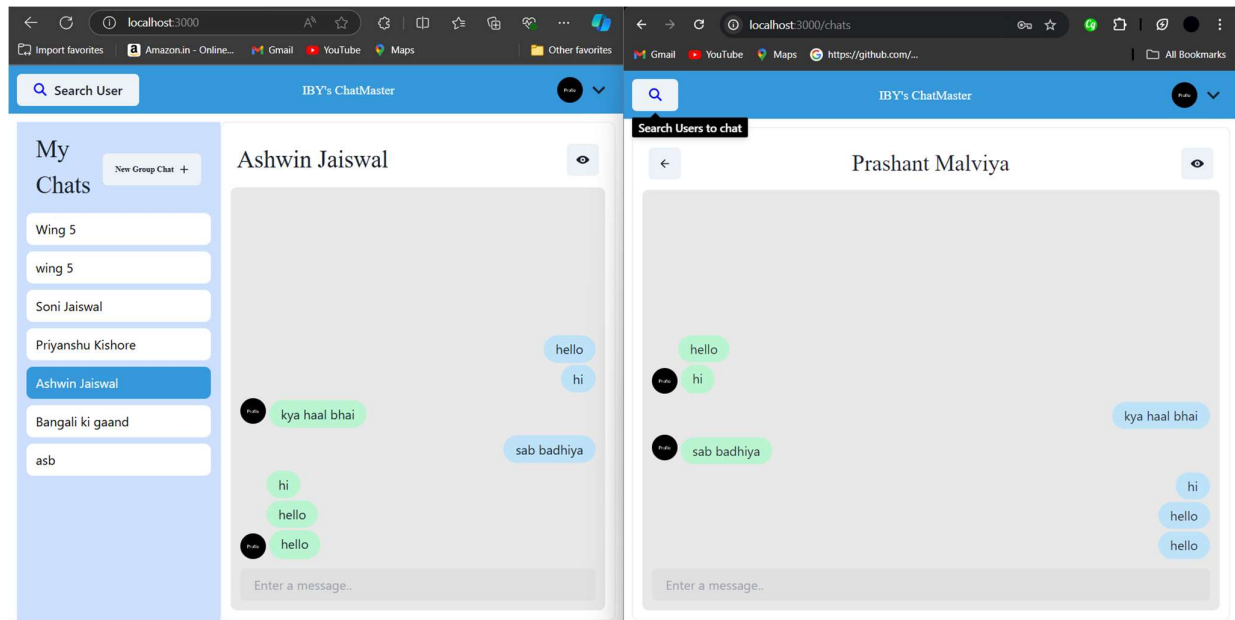
2. adding/removing someone to Group chat

3. displaying all the chats which have been accessed before.



Message APIs

1. sending messages
2. receiving whole list of messages that have been exchanged ever before.



Why socket.io?

I have used Socket.IO for real-time communication systems because of the following key reasons:

1. **Bi-Directional Communication:** Socket.IO provides real-time, two-way communication between the client and server, making it ideal for my application
2. **Cross-Browser Support:** Socket.IO automatically uses the best transport mechanism available
3. **Reconnection Support:** It handles reconnections automatically. If the connection drops, Socket.IO attempts to reconnect the client to the server without the user having to refresh or take action.
4. **Room Support:** It allowed me to create "rooms" which group clients together and allow broadcasting messages

to multiple clients which was useful in group chat functionality.

5. **Event-Driven Architecture:** Socket.IO uses an event-based approach, where events like "message," "join," or custom events can be emitted and listened to.

Dependencies:

For backend:

```
"bcrypt": "^5.1.1",  
"cookie-parser": "^1.4.6",  
"cors": "^2.8.5",  
"dotenv": "^16.4.5",  
"express": "^4.21.0",  
"mongoose": "^8.6.2",  
"nodemailer": "^6.9.15",  
"nodemon": "^3.1.4",  
"cloudinary": "^2.5.0",  
"express-fileupload": "^1.5.1",  
"socket.io": "^4.7.5"
```

For frontend

```
"@chakra-ui/icons": "^2.1.1",  
"@chakra-ui/react": "^2.8.2",  
"@emotion/react": "^11.13.3",
```

"@emotion/styled": "^11.13.0",
"@fortawesome/fontawesome-svg-core": "^6.6.0",
"@fortawesome/free-solid-svg-icons": "^6.6.0",
"@fortawesome/react-fontawesome": "^0.2.2",
"@mui/material": "^6.1.1",
"@reduxjs/toolkit": "^2.2.7",
"axios": "^1.7.7",
"formik": "^2.4.6",
"framer-motion": "^11.5.4",
"js-cookie": "^3.0.5",
"next": "^14.2.11",
"react": "^18",
"react-dom": "^18",
"react-redux": "^9.1.2",
"react-scrollable-feed": "^2.0.2",
"react-toastify": "^10.0.5",
"socket.io-client": "^4.7.5",
"yop": "^0.0.0",
"yup": "^1.4.0"