Report for chat App

Abstract

The real-time chat application developed with the MERN stack (MongoDB, Express.js, React, Node.js) utilizes Socket.IO for instant communication and Cloudinary for file uploads. It provides an efficient, scalable solution for real-time messaging, allowing multiple users to communicate instantly. The application stores messages persistently in MongoDB and supports file sharing. The combination of technologies ensures a responsive, robust chat experience, suitable for diverse applications such as collaboration tools, support platforms, or social networking services.

Objective

The primary goal of this project is to create a high-performance real-time chat application that provides:

1. **Real-Time Messaging**: Through WebSocket communication (Socket.IO), ensuring instant message delivery between users.
2. **Data Persistence**: Using MongoDB to store chat histories, user details, and file metadata, allowing users to retrieve messages across sessions.
3. **File Sharing**: Efficient management and storage of user-uploaded files (images, documents) through Cloudinary.
4. **User-Friendly Interface**: Developing a dynamic, responsive interface using React, ensuring a seamless and intuitive user experience.
5. **Scalability**: Ensuring the system can handle multiple concurrent users without compromising performance.

Introduction

In modern web applications, real-time communication has become a cornerstone for user engagement and collaboration. This real-time chat application serves as a dynamic communication platform using WebSocket connections powered by Socket.IO. The back-end logic is handled by Node.js and Express.js, while MongoDB ensures persistent storage of user data and chat histories. Cloudinary is integrated for file storage, allowing users to share images and documents during conversations. The front-end is built with React, providing a highly interactive and responsive interface.

The combination of these technologies makes the application capable of handling real-time interactions efficiently, reducing latency, and offering smooth user experiences. The app is particularly relevant for building platforms such as team collaboration tools, customer service portals, and social media chat systems.

Methodology

1. **Backend (Node.js + Express):**
    - **Socket.IO** manages real-time, bidirectional WebSocket communication between the server and clients, ensuring low latency in message delivery.
    - **Express.js** provides the RESTful API infrastructure for handling routes related to users, chats, file uploads, and session management.
    - The server is optimized for scalability, capable of managing multiple simultaneous WebSocket connections without compromising performance.
2. **Database (MongoDB):**
    - **MongoDB** is used as the NoSQL database for storing chat histories, user profiles, and metadata. The database is designed to ensure data persistence, so users can retrieve chat data even after the session ends.
    - The document-based nature of MongoDB makes it ideal for managing dynamic data structures, such as messages, user data, and attachments, making it a flexible storage solution.
3. **Frontend (React):**
    - The user interface is built with **React**, ensuring a fast, responsive, and interactive environment. React components efficiently manage the real-time message flow, updating the chat screen instantly as new messages are received.
    - The UI design emphasizes simplicity and responsiveness, allowing it to adapt to various devices (desktop, mobile).
4. **Real-Time Messaging (Socket.IO):**
    - **Socket.IO** establishes a WebSocket connection between the client and server for real-time message exchange. When a user sends a message, it is immediately broadcast to all connected users, creating a live chat experience.
    - This low-latency architecture allows for instant messaging, making the platform highly interactive.
5. **File Uploads (Cloudinary):**
    - Users can share files during chat sessions, which are uploaded to **Cloudinary**. This cloud-based solution ensures secure storage and efficient file management, preventing large files from being stored locally, which would otherwise degrade performance.
    - Cloudinary also automatically optimizes the files, improving delivery times and reducing bandwidth usage for users with limited resources.
6. **Security & Authentication:**
    - The system incorporates authentication and authorization mechanisms to ensure that only registered users can participate in chats.
    - This enhances privacy and prevents unauthorized access to sensitive communication.
7. **Scalability:**
    - The application's architecture is designed for horizontal scaling, allowing the addition of more server instances to handle increased user traffic.

  - **MongoDB**'s distributed nature also supports scaling, making the application suitable for larger user bases.

By integrating these technologies, the application provides a reliable, efficient, and scalable platform for real-time communication with enhanced functionality like file sharing and persistent chat storage.

Conclusion

The real-time chat application, built using the MERN stack and leveraging Socket.IO for real-time communication, provides a scalable and efficient solution for instant messaging. By integrating MongoDB for data persistence and Cloudinary for secure file sharing, the application meets modern communication needs while maintaining responsiveness and performance. With a user-friendly React front-end, users can engage in seamless interactions, making it suitable for various applications, including team collaboration, social networking, and customer support platforms. The system's architecture ensures scalability, security, and flexibility for handling future growth.

Server Code

connection.js in DB folder

```
const mongoose=require('mongoose')
mongoose.set('strictQuery',false)
MONGO_URL=-_-_-_-_-_
const connectionToDB=async()=>{
  try{
    // it will proide a instance
    const {connection}=await mongoose.connect(
     MONGO_URL
    )
    if(connection){
      console.log(`Connected to mongo DB: ${connection.host}`);
    }

  }
  catch(e){
    console.log(e);
```

```
        // forcefully exit
        process.exit(1);
    }
}

module.exports =connectionToDB
```

Conversation.js in models

```
const mongoose=require('mongoose')
const conversationSchema=mongoose.Schema({
    members:{
        type:Array,
        required:true
    }
})

const Conversation=mongoose.model('Conversation',conversationSchema)
module.exports=Conversation
```

Messages.js in models

```
const mongoose=require('mongoose')
const messageSchema=mongoose.Schema({
    conversationId:{
        type:String,
    },
    senderId:{
        type:String
    },
    message:{
        type:String
    }

})
```

```
const Messages=mongoose.model('Message',messageSchema)
module.exports=Messages
```

User.js in models

```
const mongoose=require('mongoose')
const userSchema=mongoose.Schema({
    fullName:{
        type:String,
        required:true
    },
    email:{
        type:String,
        required:true,
        unique:true
    },
    password:{
        type:String,
        required:true
    },
    token:{
        type:String
    }
})

const Users=mongoose.model('User',userSchema)
module.exports=Users
```

App.js in models

```
const express =require('express')
const app=express()
const port=process.env.PORT||8000
const bcryptjs=require('bcryptjs')
const jwt=require('jsonwebtoken')
const morgan=require('morgan')
// require('./db/connection.js')
const connectionToDB=require('./db/connection')
const Users=require('./models/User')
const Conversation=require('./models/Conversation')
const Messages=require('./models/Messages')
const cors=require('cors')
const { Socket } = require('socket.io')
const io=require('socket.io')(8080,{
    cors:{
        origin:'http://localhost:3000'
    }
})
app.use(express.json())
app.use(express.urlencoded({extended:false}))
app.use(cors())
app.use(morgan('dev'))
app.get('/',(req,res)=>{
    res.send('welcome')
})
// socket io
let users=[]
io.on('connection',socket=>{
    // receivingg on backend use on
    console.log('user Conneced',socket.id);
    socket.on('addUser',userId=>{
        const isUserExist=users.find(user=>user.userId===userId)
        if(!isUserExist){
            const user={userId,socketId:socket.id}
            users.push(user)
            io.emit('getUser',users)
        }
        socket.userId=userId
    });
```

```
socket.on('sendMessage',async({senderId,receiverId,message,conversationId})=
>{
        // console.log('Userass',users);

        // console.log('Usersss',user);
        const receiver=users.find(user=>user.userId===receiverId)
        const sender=users.find(user=>user.userId===senderId)
        const user=await Users.findById(senderId)
        // console.log('sender',sender,receiver);
        if(receiver){
            io.to(receiver.socketId).to(sender.socketId).emit('getMessage',{
                senderId,
                message,
                conversationId,
                receiverId,
                user:{id:user._id,fullName:user.fullName,email:user.email}
            })
        }
        // else{
        //     io.to(sender.socketId).emit('getMessage',{
        //         senderId,
        //         message,
        //         conversationId,
        //         receiverId,
        //         user:{id:user._id,fullName:user.fullName,email:user.email}
        //     })
        // }
    })
    // sending data from backend to frontend
    // io.emit('getUsers',socket.userId)
    socket.on('disconnect',()=>{
        users=users.filter(user=>user.socketId!==socket.id)
        // io.emit is used to send deatils from backend
        io.emit('getUser',users)
    })
```

```
})
// app.use(morgan('dev'))
app.post('/api/register',async (req,res,next)=>{
    try{
        const {fullName,email,password}=req.body
        if(!fullName || !email || !password){
            res.status(400).send('Please fill all entries')
        }
        else{
            const isExist=await Users.findOne({email})
            if(isExist){
                res.status(400).send('User already Exist')
            }
            else{
                const newUser=new Users({
                    fullName,
                    email
                })
                bcryptjs.hash(password,10,(err,hashedPasword)=>{
                    newUser.set('password',hashedPasword)
                    newUser.save()
                    next();
                })

                return res.status(200).send('User is registered Successfully')
                // res.status(200).json({
                //     message:'User is registered Successfully',

                // })
            }
        }
    }
    catch(e){
        console.log(e.message);
    }
})
```

```
app.post('/api/login', async (req,res,next)=>{
    try{
        const {email,password}=req.body
        if(!email || !password){
            res.status(400).send('Please fill all entries')
        }
        else{
            const user=await Users.findOne({email})
            if(!user){
                res.status(400).send('Please enter valid email Id')
            }
            else{
               const validateUser=await bcryptjs.compare(password,user.password)
               if(!validateUser){
                    res.status(400).send('Enter Valid password')
               }
               else{
                    const payload={
                        userId:user._id,
                        email:user.email
                    }
                    const JWT_SECRET_KEY=process.env.JWT_SECRET_KEY||'534'


jwt.sign(payload,JWT_SECRET_KEY,{expiresIn:84600},async(err,token)=>{
                        await Users.updateOne({_id:user._id},{
                            $set:{token}
                        })
                        user.save()
                        return
res.status(200).json({user:{id:user._id,email:user.email,fullName:user.fullName},t
oken:token})

                    })

               }
            }
        }
    }
    catch(e){
        console.log(e.message);
    }
})

app.post('/api/conversation',async(req,res,next)=>{
    try{
        const {senderId,receiverId}=req.body
        const newConversation=new
Conversation({members:[senderId,receiverId]})
        await newConversation.save()
        res.status(200).send('Conversation Created successfully')
    }
    catch(e){
        console.log(e.message);
    }
})

app.get('/api/conversation/:userId',async(req,res,next)=>{
    try{
        const userId=req.params.userId
        const conversation=await Conversation.find({members:{$in:[userId]}})
        // console.log('conversation',conversation);
        const
conversationUserData=Promise.all(conversation.map(async(conversation)=>{
            const
receiverId=conversation.members.find((member)=>member!==userId)
            const user= await Users.findById(receiverId)
            return
{user:{receiverId:user._id,email:user.email,fullName:user.fullName},conversationI
d:conversation._id}
        }))
        res.status(200).json(await conversationUserData)
    }
    catch(e){
        console.log(e.message);
    }
})
app.post('/api/message',async(req,res,next)=>{
    try{
        const {conversationId,senderId,message,receiverId=''}=req.body
        if(!senderId || !message )  return res.status(200).send('Please fill all entries')
        if(conversationId==='new' && receiverId) {
            const newConversation=new
Conversation({members:[senderId,receiverId]})
            await newConversation.save()
            const newMessage=new
Messages({conversationId:newConversation._id,senderId,message})
            await newMessage.save()
            return res.status(200).send('Message sent successfully')

        }
        else if(!conversationId && !receiverId){
            return res.status(400).send('Please fill all entries')
        }
        const newMessage=new Messages({conversationId,senderId,message})
        await newMessage.save();
        res.status(200).send('Messages sent successfully');

    }
    catch(e){
        console.log(e);
    }
})

app.get('/api/message/:conversationId',async(req,res)=>{
    try{
        const checkMessage=async(conversationId)=>{
            const message=await Messages.find({conversationId})
            const messageUserData=Promise.all(message.map(async(message)=>{
                console.log('sender id',message.senderId);
            const user=await Users.findById(message.senderId)
            return
{user:{id:user._id,email:user.email,fullName:user.fullName},message:message.m
essage}
            // return user
            // console.log('user',user);
            // console.log('message',message);
            //  {(!user) return message:message.message,}
         }))
         res.status(200).json(await messageUserData)

      }
      const conversationId=req.params.conversationId

      if(conversationId==='new') {
         const checkConversation=await
Conversation.find({members:{$all:[req.query.senderId,req.query.receiverId]}})
         if(checkConversation.length>0) checkMessage(checkConversation[0]._id)
         else return res.status(200).json([])
      }else{
         checkMessage(conversationId)
      }

   }

   catch(e){
      console.log(e);
   }
})


app.get('/api/users/:userId',async(req,res,next)=>{
   try{
      const userId=req.params.userId
      // $ne=not equal to
      const users=await Users.find({_id:{$ne:userId}})
      const userData=Promise.all(users.map(async(user)=>{
            return {user:
{email:user.email,fullName:user.fullName,receiverId:user._id}}
        }))
        res.status(200).json(await userData)
    }
    catch(e){
        console.log(e);
    }
})


app.listen(port,async()=>{
    await connectionToDB()
    console.log('listening on port ' +port);
})
```

Frontend Code

Index.html in public folder

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

All the things are in src folder

index.js in button (in components folder)

```
import React from "react";
const Button = ({
    label='button',
    type='button',
    className='',
    disabled=''
})=>{
    return(
        <button type={type} className={`text-white bg-primary hover:bg-blue-800 
focus:ring-4 focus:outline-none
        focus:ring-blue-300 font-medium rounded-lg text-sm px-5 py-2.5 text-center 
${className}`}
```

```
        disabled={disabled}>{label}</button>
    )
}
export default Button
```

index.js in input(in components folder)

```
import React from "react";
const Input = ({
    label='',
    name='',
    type='text',
    className='',
    inputclassName='',
    isRequired=false,
    placeholder='',
    value='',
    onChange=()=>{}
})=>{

    return(
    <div className={`w-1/2 ${className}`}>
            <label htmlFor={name} className="block mb-2 text-sm font-medium
">{label}</label>
            <input type={type} id={name} className={`border bg-gray-50
border-gray-300 text-gray-900 text-sm rounded-lg focus:ring-blue-500
            focus:border-blue-500 block w-full p-2.5 ${inputclassName}`}
value={value} placeholder={placeholder} required={isRequired}
            onChange={onChange}/>
        </div>
    )

}
export default Input
```

index.js in Dashboard(in modules folder)

```
import React, { useEffect, useRef, useState } from "react";
import Avatar from '../../Assets/avatar.webp'
import Input from '../../components/Input/index.js'
import { io } from 'socket.io-client'
import { useNavigate  } from "react-router-dom";

const DashBoard = ()=>{
   let userName=''
   const messageRef=useRef(null)
   const [socket,setSocket]=useState(null)
   const [message,setMessage]=useState({})
   const[people,setPeople]=useState({})
   const
[user,setUser]=useState(JSON.parse(localStorage.getItem('user:detail')))
   const [conversation,setConversation]=useState([])
   const [formsendMessage,setformSendMessage]=useState('');
   const [value,setValue]=useState('abhay')
   // const[status,setStatus]=useState({})
   const[showPeople,setToShowPeople]=useState([])
   useEffect(()=>{
      const loggedInUser=JSON.parse(localStorage.getItem('user:detail'))
      const fetchConversation=async()=>{
         const res=await
fetch(`http://localhost:8000/api/conversation/${loggedInUser?.id}`,{
            method:'GET',
            headers:{
               'Content-Type':'application/json',
            }
         })
         // console.log('gg');
         const resdata=await res.json()
         // console.log('resData',resdata);
         setConversation(resdata)
      }
      fetchConversation()
   },[])
```

```
useEffect(()=>{
    // backend url
    setSocket(io('http://localhost:8080'))
},[])
useEffect(()=>{
    socket?.emit('addUser',user?.id)
    socket?.on('getUser',users=>{
        console.log('ckd');
        console.log('active users',users);
    })
    socket?.on('getMessage',data=>{
        console.log('getMeaagesss',data);
        setMessage(prev=>({
            ...prev,
            messages:[...prev.messages,{user:data.user,message:data.message}]
        }))
    })
},[socket])

useEffect(()=>{
    console.log('user fetched');
    console.log('users',people);
    const fetchUsers=async()=>{
        const res=await fetch(`http://localhost:8000/api/users/${user?.id}`,{
            method:'GET',
            headers:{
                'Content-Type':'application/json',
            }
        })
        const resData=await res.json();
        console.log('resData',resData);
        setPeople(resData)
        // if(resData.token!="")   setStatus()
    }
    fetchUsers()
},[])
```

```
    useEffect(()=>{
        messageRef?.current?.scrollIntoView({behaviour:'smooth'})
    },[message?.messages])
    const navigate=useNavigate()
    const logout=async(id)=>{
        console.log('called');
        const res=await fetch(`http://localhost:8000/api/logout/${id}`,{
            method:'GET',
            headers:{
                'Content-Type':'application/json',
            }
        })
        console.log('res',res.status);
        if(res?.status==200){
            // localStorage.setItem('user:token',null)
            // // array of object and object thatswhy json strigify
            // localStorage.setItem('user:detail',null)
            localStorage.removeItem('user:token')
            localStorage.removeItem('user:detail')
            navigate('/')
        }
    }

    const fetchMessages=async(conversationId,receiver)=>{
            const res=await
fetch(`http://localhost:8000/api/message/${conversationId}?senderId=${user?.id}
&&receiverId=${receiver?.receiverId}`,{
                method:'GET',

                headers:{
                    'Content-Type':'application/json',
                }
            })
            const resData=await res.json()
            console.log('Fetching messages',resData);
            console.log('userrr',user);
        userName=user?.fullName
        // console.log();
        console.log('userName',userName);
        setMessage({messages:resData,receiver,conversationId})
}

// {people.length>0 ?
//                 people.map(({userId,user}) => {

// )}
let c=0;

const sendMessage=async(e)=>{
    // console.log('checkkkkk');
    // console.log(message?.conversationId);
    // console.log(user?.id);
    // console.log(formsendMessage);
    // console.log(message?.receiver?.receiverId)
    setformSendMessage('')
    // setValue('')
    const res=await fetch('http://localhost:8000/api/message',{
        method:'POST',
        headers:{
            'Content-Type':'application/json',
        },
        body:JSON.stringify({
            conversationId:message?.conversationId,
            senderId:user?.id,
            message:formsendMessage,
            receiverId:message?.receiver?.receiverId,
            id:user?.id
        })
    })

    socket?.emit('sendMessage',{
        senderId:user?.id,
        receiverId:message?.receiver?.receiverId,
        message,
        conversationId:message?.conversationId
    })
    // const resData=await res.json();

}

return(
    <div className="w-screen flex">
        <div className="w-[25%] h-screen bg-secondary overflow-scroll">
            <div className="flex justify-center items-center my-8">
                <div className="border border-primary p-[2px] rounded-full">
                     <img src={Avatar} width={50} height={75}/>
                </div>

                <div className="ml-8">
                    <h3 className="text-2xl">
                        {user.fullName}
                    </h3>
                    <p className="text-lg font-light ">My Account</p>
                    <button onClick={()=>{
                        // logout(id)
                        logout(user.id)
                        // console.log(user.id);
                    }} className="hover:underline color-blue">Logout</button>
                </div>
            </div>
            <hr/>
            <div className="ml-10 mt-10">
                <div className="text-primary text-lg">
                    Messages
                </div>
                <div>
                {conversation.length>0 ?
                conversation.map(({conversationId,user}) => {
                    // console.log('conversation is',c);
                        return(
                                <div className="flex items-center py-5 border-b
border-b-gray-300">
                                    <div className="cursor-pointer flex items-center
justify-center" onClick={()=>fetchMessages(conversationId,user)}>
                                        <div>
                                            <img src={Avatar} width={50} height={50}/>
                                        </div>

                                        <div className="ml-8">
                                            <h3 className="text-lg font-semibold">
                                                {/* {user.fullName} */}
                                                {user.fullName}
                                            </h3>
                                            <p className="text-sm font-light
text-gray-600">{user.token!='' ? 'Online':'Offline'}</p>
                                        </div>
                                    </div>
                                </div>
                            )
                        }):<div className="text-center text-lg ">No Conversation</div>}
                    </div>
                </div>

            </div>
            <div className="w-[50%] h-screen bg-white flex flex-col items-center" >
                {message?.receiver?.fullName &&
                    <div className="w-[75%] bg-secondary h-[60px] my-4
rounded-full flex items-center px-14">
                    <div className="cursor-pointer">
                        <img src={Avatar} width={40} height={40}/>
                    </div>

                    <div className="ml-6">
                        <h3 className="text-lg">
                            {/* {userName==''?'Name':{userName}} */}
                            {message?.receiver?.fullName}
                        </h3>
                    <p className="text-sm font-light   text-gray-600">
                    {message?.receiver?.email}
                    {/* {user.token!='' ? 'Online':'Offline'} */}
                    </p>
                  </div>
            </div>
            }
              <div className="h-[75%] w-full overflow-scroll border-b">
                  <div className="p-14">
                    {
                     message?.messages?.length>0 ?
                     message?.messages?.map(({message,user:{id}={}})=>{
                         if(id===user?.id){
                             return(
                                 <div>
                                         <div className="p-4 text-white max-w-[50%]
bg-primary rounded-b-xl rounded-tl-xl ml-auto mb-6">
                                             {message}
                                         </div>
                                         <div ref={messageRef}>

                                         </div>
                                 </div>
                             )
                         }
                         else{
                            return(
                             <div className="max-w-[50%] bg-secondary
rounded-b-xl rounded-tr-xl p-4 mb-6">
                             {message}
                         </div>
                            )
                         }
                     }):<div className="
                     text-center text-lg font-semibold mt-24">No message</div>
                    }
                  </div>
```

```
                </div>
                {
                    message?.receiver?.fullName &&
                    <div className="p-14 w-full flex items-center">
                        <Input placeholder='Type a message .... '
value={formsendMessage}
onChange={(e)=>setformSendMessage(e.target.value)} className="w-full"
inputclassName="p-2 px-4 shadow-lg rounded-md bg-secondary focus:ring-0
focus:border-0 outline-none"/>
                        {/* <i class="fa-solid fa-paper-plane"></i> */}
                        <div className={`ml-3 p-2 cursor-pointer bg-secondary
rounded-full ${!formsendMessage && 'pointer-events-none'}`}
onClick={()=>sendMessage()}>
                        <svg  xmlns="http://www.w3.org/2000/svg"  width="20"
height="20"  viewBox="0 0 24 24"  fill="none"  stroke="currentColor"
stroke-width="2"  stroke-linecap="round"  stroke-linejoin="round"  class="icon
icon-tabler icons-tabler-outline icon-tabler-send"><path stroke="none" d="M0
0h24v24H0z" fill="none"/><path d="M10 14l11 -11" /><path d="M21 3l-6.5 18a.55
.55 0 0 1 -1 0l-3.5 -7l-7 -3.5a.55 .55 0 0 1 0 -1l18 -6.5" /></svg>

                        </div>
                    </div>
                }

            </div>

            <div className="w-[25%] h-screen bg-light px-8 py-16 overflow-scroll">
                <div className="text-primary text-lg">People</div>
                {people.length>0 ?
                    people.map(({userId,user}) => {
                        // console.log('conversation is',c);
                            return(
                                <div className="flex items-center py-5 border-b
border-b-gray-300">
                                    <div className="cursor-pointer flex items-center
justify-center" onClick={()=>fetchMessages('new',user)}>
                                        <div>
                                            <img src={Avatar} width={50} height={50}/>
                                        </div>

                                        <div className="ml-8">
                                            <h3 className="text-lg font-semibold">
                                                {/* {user.fullName} */}
                                                {user.fullName}
                                            </h3>
                                            <p className="text-sm font-light
text-gray-600">{user.token!='' ? 'Online':'Offline'}</p>
                                        </div>
                                    </div>
                                </div>
                            )
                        }):<div className="text-center text-lg ">No Conversation</div>}
            </div>
        </div>
      )

}
export default DashBoard
```

index.js in form(in modules folder)

```
import { useState } from "react"
import Button from "../../components/Button"
import Input from "../../components/Input"
import {  useNavigate } from "react-router-dom"
const Form=({
  isSignInPage=true
})=>{
  const [data,setData]=useState({
    ...(!isSignInPage && {
      fullName:''
```

```
        }),
        email:'',
        password:''
    })
    // console.log('data ',data);
    const navigate=useNavigate()
    const handleSubmit=async(e)=>{
        e.preventDefault()
        console.log(data);
        console.log('issignin',isSignInPage);
        const res= await fetch(`http://localhost:8000/api/${isSignInPage ? 'login' :
'register'}`,{
            method:'POST',
            headers:{
                'Content-Type':'application/json'
            },
            body:JSON.stringify(data)
        })
        console.log('res.satatus',res.status);
        if(res.status===400){
            alert('Please fill correct entry')
        }
      else{
            console.log('enter',res);
            if(!isSignInPage) navigate('/users/sign_in')
            const resdata=await res.json()
            console.log('check');
            console.log("res",resdata);
            if(resdata.token){
                localStorage.setItem('user:token',resdata.token)
                // array of object and object thatswhy json strigify
                localStorage.setItem('user:detail',JSON.stringify(resdata.user))
                navigate('/')
            }
      }
    }
    return(
        <div className="bg-screen h-screen flex items-center justify-center">
          <div className="bg-white w-[600px] h-[600px] shadow-lg rounded-3xl
flex items-center justify-center flex-col">
            {/* flex col se indiviual each div inside it come in new row */}
            <div className="text-4xl font-bold">
              Welcome {isSignInPage && 'Back'}
            </div>

            <div className="text-xl font-light mb-10">{isSignInPage ? 'Sign In to
get Explored ' : 'Sign Up to Start'}</div>
            <form onSubmit={(e) => handleSubmit(e) } className="flex flex-col
justify-center w-full items-center w-1/2">
              {!isSignInPage &&<Input label="Full Name" name="name"
placeholder="Enter your name" value={data.fullName}
onChange={(e)=>setData({...data,fullName:e.target.value})}/>}
              <Input label="Email" name="email" placeholder="Enter your Email"
value={data.email}  onChange={(e)=>setData({...data,email:e.target.value})}/>
              <Input label="password" name="password" placeholder="Enter your
password" type="password" className="mb-10" value={data.password}
onChange={(e)=>setData({...data,password:e.target.value})}/>
              <Button label={isSignInPage ? "Sign In":"Sign Up"}
className='w-[75%] mb-3' type="submit"/>
            </form>
            <div className="gap-1  flex">
              <div> {isSignInPage ? "Did't have an Account" :
              "Already Have an Account?"
              }</div>
              <span className="text-primary cursor-pointer underline"
onClick={()=>navigate(`/users/${isSignInPage ? 'sign_up':'sign_in'}`)}>
                {!isSignInPage ? "Sign In" :"Sign Up"}
              </span>
            </div>
          </div>
        </div>
    )
}
export default Form
```

App.js in src

```
import './App.css';
import DashBoard from './modules/Dashboard/Index';
import Form from './modules/Form';
import { Routes ,Route, Navigate, useNavigate } from 'react-router-dom';
const ProtectedRoutes=({children,auth=false}) =>{
  const isLoggedIn=localStorage.getItem('user:token') !==null || false
  console.log('isLoggedIn',isLoggedIn);
  const navigate=useNavigate()
  if(!isLoggedIn && auth){
    return <Navigate to={'/users/sign_in'}/>
  }
  else if(isLoggedIn &&
['/users/sign_in','/users/sign_up'].includes(window.location.pathname)){
     return <Navigate to={'/'}/>
  }
  return children
}
function App() {
  return(
    <Routes>
      <Route path='/' element={
        <ProtectedRoutes auth={true}>
          <DashBoard/>
        </ProtectedRoutes>
      }/>
      <Route path='/users/sign_in' element={
        <ProtectedRoutes>
              <Form isSignInPage={true}/>
       </ProtectedRoutes>
      }/>
      <Route path='/users/sign_up' element={
       <ProtectedRoutes>
            <Form isSignInPage={false}/>
      </ProtectedRoutes>}/>
   </Routes>

  )
}

export default App;
```

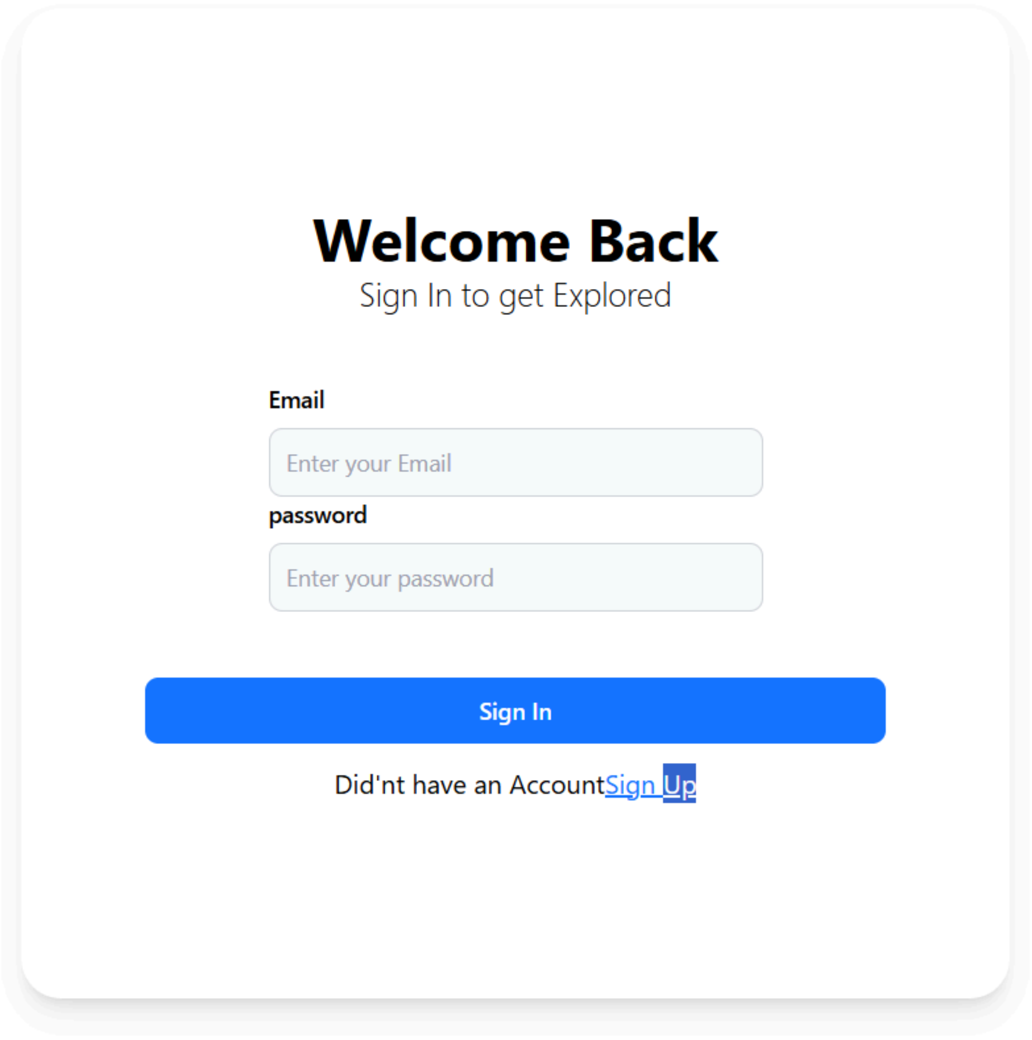
Index.js in src

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
   <BrowserRouter>
      <App />
   </BrowserRouter>

);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Login Page



Welcome Back

Sign In to get Explored

Email

Enter your Email

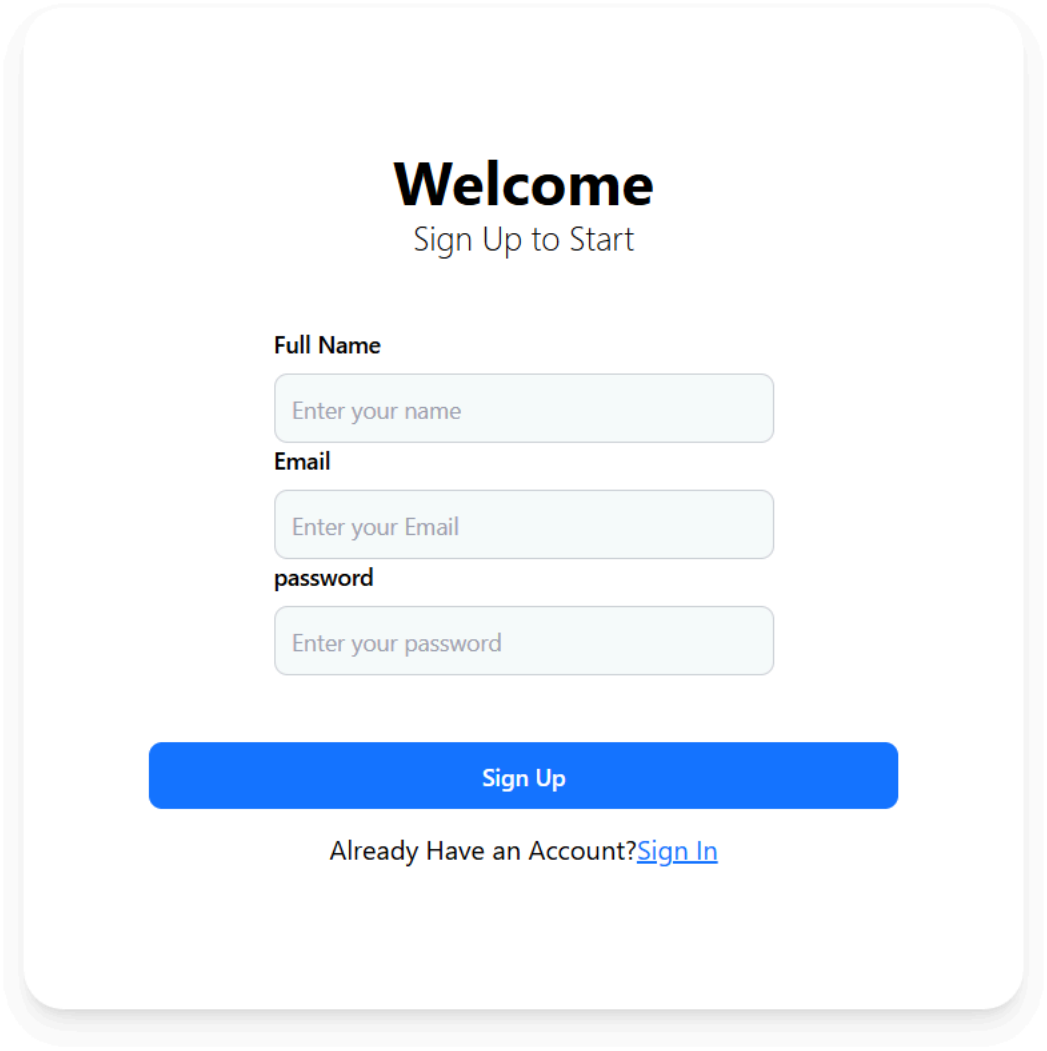
password

Enter your password

Sign In

Did'nt have an AccountSign Up

Sign Up Page



Welcome

Sign Up to Start

Full Name

Enter your name

Email

Enter your Email

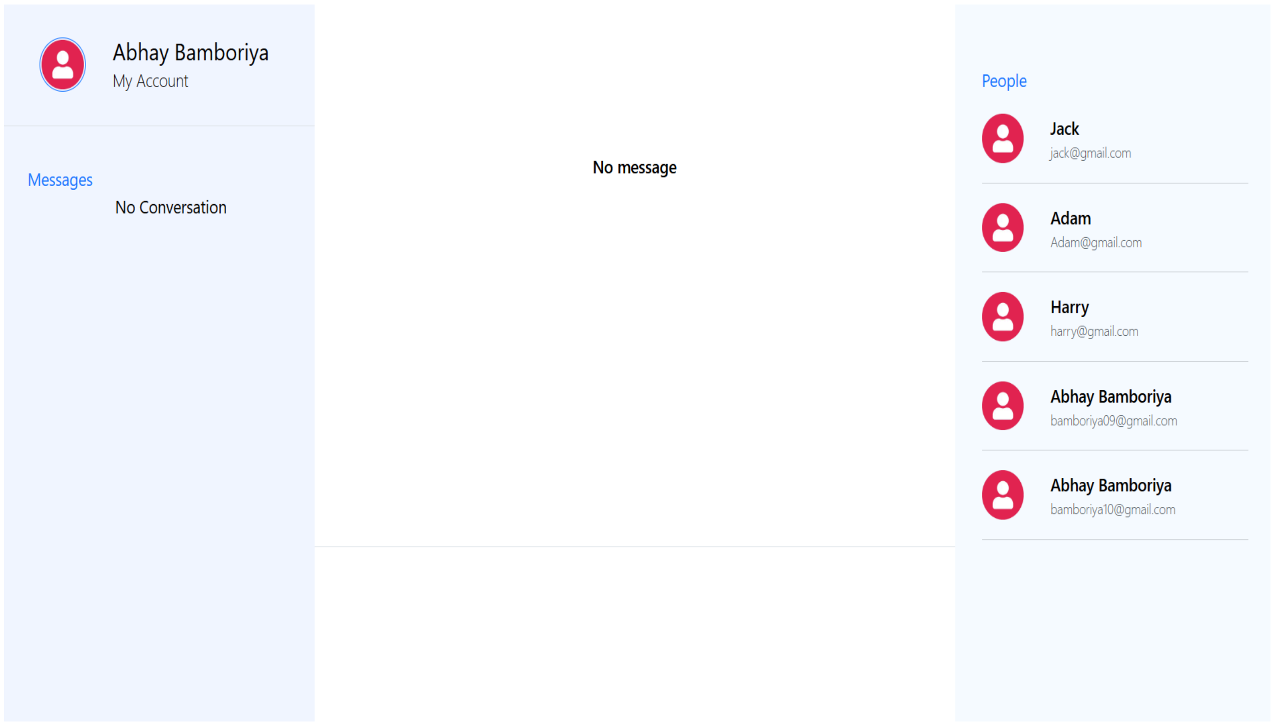
password

Enter your password

Sign Up

Already Have an Account?Sign In

Main DashBoard



Abhay Bamboriya
My Account

Messages
No Conversation

No message

People

Jack
jack@gmail.com

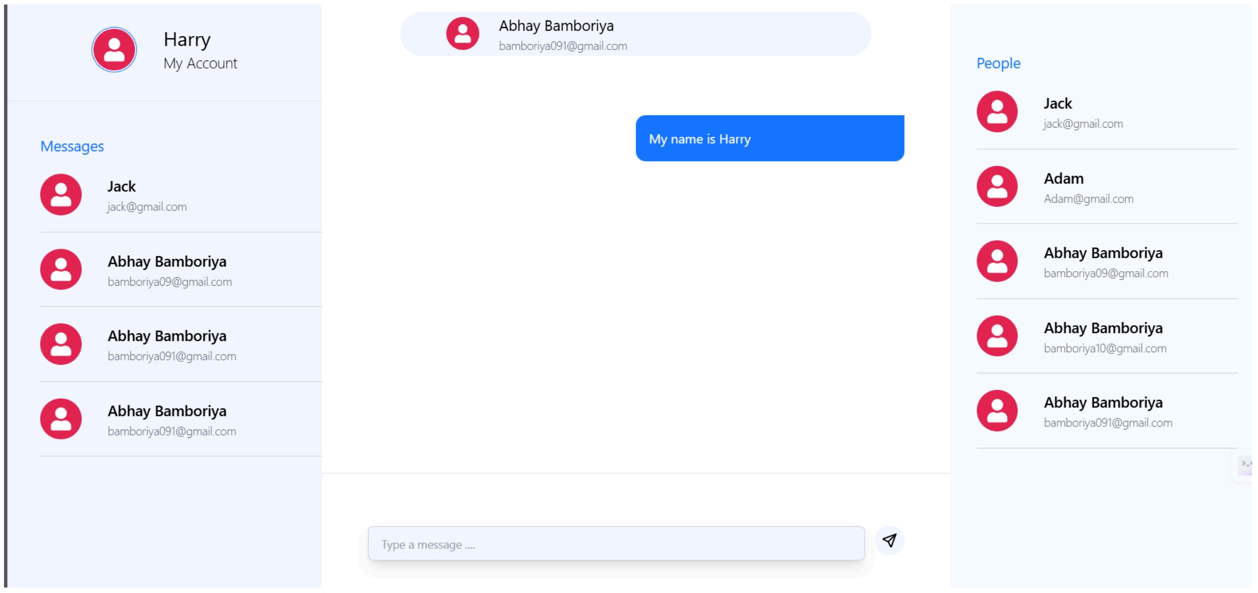
Adam
Adam@gmail.com

Harry
harry@gmail.com

Abhay Bamboriya
bamboriya09@gmail.com

Abhay Bamboriya
bamboriya10@gmail.com

Message Sent



Harry
My Account

Messages

Jack
jack@gmail.com

Abhay Bamboriya
bamboriya09@gmail.com

Abhay Bamboriya
bamboriya091@gmail.com

Abhay Bamboriya
bamboriya091@gmail.com

Abhay Bamboriya
bamboriya091@gmail.com

My name is Harry

Type a message ....

People

Jack
jack@gmail.com

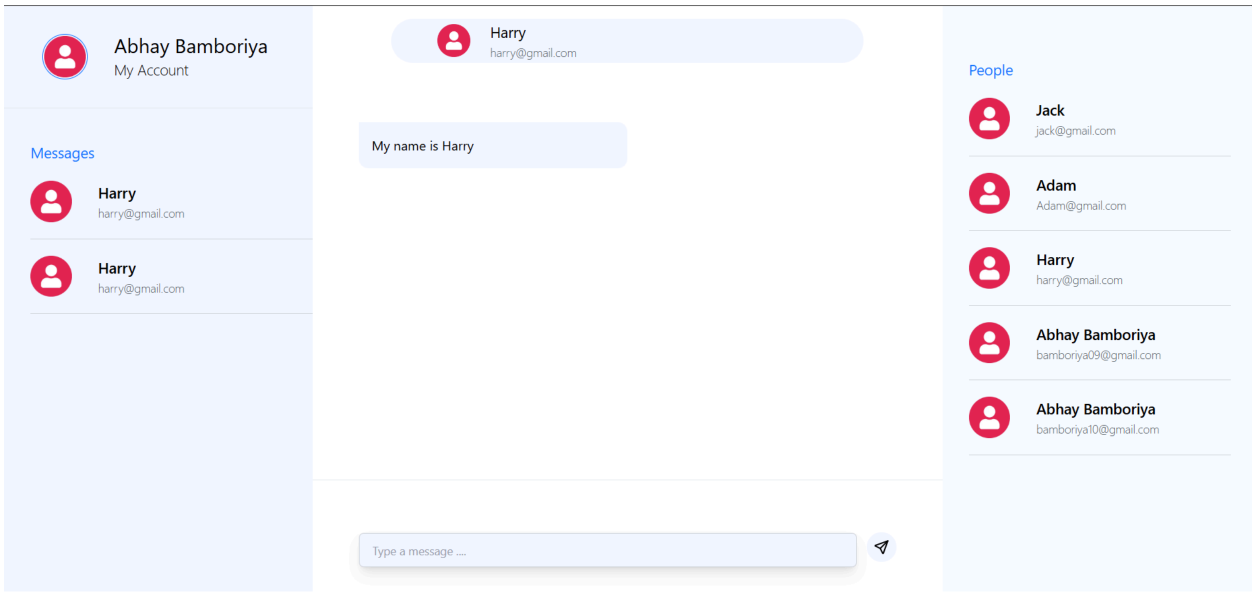
Adam
Adam@gmail.com

Abhay Bamboriya
bamboriya09@gmail.com

Abhay Bamboriya
bamboriya10@gmail.com

Abhay Bamboriya
bamboriya091@gmail.com

Message Received



Abhay Bamboriya
My Account

Messages

Harry
harry@gmail.com

Harry
harry@gmail.com

Harry
harry@gmail.com

My name is Harry

Type a message ....

People

Jack
jack@gmail.com

Adam
Adam@gmail.com

Harry
harry@gmail.com

Abhay Bamboriya
bamboriya09@gmail.com

Abhay Bamboriya
bamboriya10@gmail.com