**Chat App ( source : as a programmer )**

Install react :

Create vite@latest -> react -> js -> npm I -> npm run dev

Server :

Root : npm inti 🡪 this will generate a package.json file

Root : npm I express dotenv bcryptjs cookie-parser jsonwebtoken mongoose socket.io

Then create a server file inside the backend folder .

And update package.json

"server": "nodemon backend/server.js",

Then create server using express

In schema poperty enum : [“male” , “female”] 🡸 these value only come in this key from the forntend

//index :  
problem : instead of doing this

app.get('/api/auth/login' , (req ,res)=>{

  res.send("Home page")

})

app.get('/api/auth/signup' , (req ,res)=>{

  res.send("Home page")

})

app.get('/api/auth/logout' , (req ,res)=>{

  res.send("Home page")

})

**Solve :**

Create a new file : authRoute.js

import  express  from "express";

const router = express.Router();

router.get("/login" , (req ,res)=>{

    res.send("login page")

});

router.get("/signup" , (req ,res)=>{

    res.send("signup page")

});

*export* default router;

**add this in index.js :**

if url is this : /api/auth/login

starting are same , for check the endpoint , middleware is run and go to authRoutes file and check if any routes is present that match the endpoint then it will execute .

import authRoutes from './routes.js/authRoutes.js';

app.use('/api/auth' , authRoutes);

Problem : **why we write a big line of code in one route this is showing messay ?**

instead of doing the whole process in the routes file :

router.get("/login", (req , res)=>{ // assume that in one route if there is big line of code . it see massy

    res.send("signup page ")

    console.log("signup user");

}  );

router.get("/login", (req , res)=>{

    res.send("signup page ")

    console.log("signup user");

}  );

router.get("/login", (req , res)=>{

    res.send("signup page ")

    console.log("signup user");

}  );

**To solve this we create a controller file :**

*//authController.js*

*export* const login = (req , res)=>{ these are controllers

    res.send("login page ")

    console.log("login user");

}

*export* const signup = (req , res)=>{

    res.send("signup page ")

    console.log("signup user");

}

*export* const logout = (req , res)=>{

    res.send("logout page ")

    console.log("logout user");

}

// authRoutes.js

import  express  from "express";

import { login, logout, signup } from "../controllers/authController.js";

const router = express.Router();

router.get("/login", login );

router.get("/signup" , signup);

router.get("/logout" , logout);

*export* default router;

**now create folder that is use to connect from the mongodb :**

import mongoose from "mongoose";

const connectToMongoDB = *async*()=>{

    try{

        await mongoose.connect(process.env.MONGO\_DB\_URL);

        console.log("Connected to mongodb");

    }catch(err){

            console.log("Error in  Connecting to mongodb" , err.message);

    }

}

*export* default connectToMongoDB;

**//index.js**

app.listen(port , ()=>{

    console.log(`Server is running on port ${port}`);

    connectToMongoDB(); 🡸

})

mongodb+srv://ajaykushwaha626162:182460328@cluster0.49ae5ay.mongodb.net/**chat-app-db**?retryWrites=true&w=majority&appName=Cluster0

this is the name of database default is test.

**Now we create a userModel :**

**//userModel js**

import mongoose from "mongoose";

const userSchema = new mongoose.Schema({

    fullname : {

        type : String,

        required : true

    },

    username : {

        type : String,

        required : true,

        unique : true

    },

    password : {

        type : String,

        required : true,

        minlength : 6

    },

    gender : {

        type : String ,

        required : true,

        enum : ['male' , 'female']

    },

    profilePic:{

        type : String ,

        default : ""

    }

});

const userModel = new mongoose.model("User" ,userSchema);

*export* default userModel;

**//signupController.js**

*export* const signup = *async*(req , res)=>{

    try{

        const {fullname , username , password , confirmPassword , gender} = req.body; // req.body is work when we add **app.use(express.json()) in the index.file**

        if(password !== confirmPassword){

            return res.status(400).json({error : "Passwords don't match.."})

        }

        const user = await userModel.findOne({username});

        if(user){

            return res.status(400).json({error : "User already exist.."})

        }

//https://avatar-placeholder.iran.liara.run 🡺 to create a unique avatar

        const boyProfilePic = `https://avatar.iran.liara.run/public/boy?username=${username}`;

        const girlProfilePic = `https://avatar.iran.liara.run/public/girl?username=${username}`;

        const newUser = new userModel({ // this will return a obj that is including object\_id

            fullname ,

            username ,

            password ,

             gender,

             profilePic : gender == 'male' ? boyProfilePic : girlProfilePic

        });

        await newUser.save(); // save this model object

        res.status(201).json({ // response

                \_id : newUser.\_id,

                fullname : newUser.fullname,

                username : newUser.username,

                profilePic : newUser.profilePic

            })

    }catch(err){

        console.log("Error in  signup contoller" , err.message);

        res.status(500).json({error : "Internal server error.."})

    }

}

**Secure the password by converting them into hash code :**

For this we need a package that is npm I **bcrpytjs.**

**bcrypt has three methods :**

1. bcrypt.genSalt( 10 <value> );

2. bcrypt.hash(password , value return by genSalt);

3. bcrypt.compare(password , hashCode Password); //it is use to compare the password and input password. Because we save the password in the hashcodes so first parmater is convert into hashcode and after that it will compare and return boolean.

*// hashpasswod code here*

        const salt = await bcrypt.genSalt(10); 🡸

        const hashpassword = await bcrypt.hash(password , salt); 🡸

        const newUser = new userModel({

            fullname ,

            username ,

            password :hashpassword, 🡸

             gender,

             profilePic : gender == 'male' ? boyProfilePic : girlProfilePic

        });

**Geneate token and setCookie :**import jwt from 'jsonwebtoken';

const geneateTokenAndSetCookie = (userId , res)=>{

*// generate token*

        const token = jwt.sign({userId} , process.env.JWT\_SECRET ,{

            expiresIn : '15d',

        })

*//set cookie*

        res.cookie("jwt" <cookie name> , token <cookie value > , {

            maxAge : 15\*24\*60\*60\*1000, // max age of cookie in ms

            httpOnly : true,

            sameSite :'strict',

            secure : process.env.NODE\_ENV !=="development"

        });

}

*export* default geneateTokenAndSetCookie;

**// call in signup controller**

  geneateTokenAndSetCookie(newUser , res);

**Login api :**

**bcrypt.compare(password , user.password);** it is use to compare the password and input password.

Because we save the password in the hashcodes so first parmater is convert into hashcode and after that it will compare and return boolean.

import userModel from "../models/userModel.js";

import bcrypt from 'bcryptjs';

import geneateTokenAndSetCookie from "../utils/generateToken.js";

*export* const login = *async*(req , res)=>{

   const {username , password} = req.body;

   const user = await userModel.findOne({username}); //find user

   const isPasswordCorrect = await bcrypt.compare(password , user?.password || “”); // compare input pass and user password

   if(!user || !isPasswordCorrect){

        return res.status(400).json({error : "invalid username or password!"})

   }

   geneateTokenAndSetCookie(user.\_id , res);

   res.status(200).json({

    \_id :user.\_id,

    fullname :user.fullname,

    username :user.username,

    profilePic :user.profilePic

   })

}

**Logout api :**

*export* const logout = (req , res)=>{

    try{

        res.cookie("jwt" ,"",{maxAge : 0}); // we set the cookie age to 0 means delete ( 1para : cookie name 2para:cookie value 3para : properties)

        res.status(200).json({message : "Logout Successfully"});

    }catch(err){

        console.log("Error in  login contoller" , err.message);

        res.status(500).json({error : "Internal server error.."})

    }

};

Message Model :

import mongoose from "mongoose";

const messageSchema = new mongoose.Schema({

   senderId : {

     type : mongoose.Schema.Types.ObjectId ,

     ref : User , *// this is collection name*

     required : true

   },

   receiverId : {

    type : mongoose.Schema.Types.ObjectId ,

    ref : User , *// this is collection name*

    required : true

   },

   message : {

    type :String,

    required : true

   }

//createdAt

// updatedAt

} ,

{timestamps : true} *// this  is second parameter of schema function it is add two field in obj that is “created At” and “updated At” that is contain time stemps.*

);

const messageModel = new mongoose.model("Message" , messageSchema);

*export* default messageModel;

**Conversation Model :**

import mongoose from "mongoose";

const conversationSchema  = new mongoose.Schema({

    participants : [{

        type : mongoose.Schema.Types.ObjectId,

        ref : 'User'

    }] ,

    messages : [

        {

            type : mongoose.Schema.Types.ObjectId,

            ref : 'Message',

            default :[]

        }

    ]

} , {timestamps : true});

const conversationModel = new mongoose.model('Conversation' , conversationSchema);

*export* default conversationModel;

**Send message api :**

//Server.js

Create a new route :

import cookieParser from 'cookie-parser'; // using this we can interact with cookies

app.use(cookieParser()); *// write this before routers  it is use to interact with cookies*

app.use('/api/message' , messageRoutes);

//messageRoutes.js

In this protectRoute is middleware that is use to confirm that the sender is valid or not .

If the creadential are valid then it will go in the sendmessage function .

import express from 'express';

import { sendMessage } from '../controllers/messageController.js';

import protectRoute from '../middleware/protectRoute.js';

const router = express.Router();

router.post('/send/:id' ,protectRoute ,sendMessage); // id is receverId

*export* default router;

//protectRoute.js

In this “.select("-password") “ means remove the password field from the obj that you are rerturning.

import jwt from 'jsonwebtoken';

import userModel from '../models/userModel.js';

const protectRoute = *async*(req , res , next)=>{

    try{

            const token = req.cookies.jwt; *// to use this import cookie parser*

            if(!token){

                return res.status(401).json({error : "unauthorized - No token provided"});

            }

            const decoded = jwt.verify(token ,process.env.JWT\_SECRET);*// it return true if this tokoen is created by this jwt secreate , this will return  userId that we  assign at the time of  token creation in generateToken.js*

            if(!decoded){

                return res.status(401).json({error : "unauthorized - invalid token"});

            }

            const user = await userModel.findById(decoded.userId).select("-password"); *// - means remove that from the obj*

            if(!user){

                return res.status(404).json({error : "User not found"});

            }

            req.user = user; *// we add the user in the request  so we can get that user in the next fuction parameter*

            next(); *// go to the next function*

    }catch(err){

        console.log('Error in protectRoute Middleware');

        res.status(500).josn({error : "Internal server error"})

    }

}

*export* default protectRoute;

// sendMessage form message controller

import conversationModel from "../models/conversationModel.js";

import messageModel from "../models/messageModel.js";

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

    try {

        const {message} = req.body;

        const {id:receiverId} = req.params; *// rename id to reciverId  using :*

        const senderId = req.user.\_id; *// we can now use this because be add the user in the req by using middleware(protectRoute)*

        let conversation = await conversationModel.findOne({

            participants :{$all : [senderId , receiverId]}

        });

        if(!conversation){

            conversation = await conversationModel.create({

                participants : [senderId , receiverId]

            })

        }

        const newMessage = new messageModel({

            senderId,

            receiverId ,

            message

        });

        if(newMessage){

            await conversation.messages.push(newMessage.\_id);

        }

*//await newMessage.save();*

*//await  conversation.save();  // there is a problem  it will  wait for previous work complete*

*//!    or write in promise way*

        await Promise.all([newMessage.save() , conversation.save()]); *//using this this will run in parallel*

        res.status(200).json(newMessage);

    } catch (error) {

        res.status(500).json({error : "Internal server error"});

        console.log("Error in  sendMessage Controller" ,error.message);

    }

}

**//getMessage api :**

First create a new route in message routes.

*//get message route*

router.get('/:id' ,protectRoute ,getMessage); *//here the id is second person id with we chat*

then create a getMessage in message controller .

*export* const getMessage = *async*(req , res)=>{

    try{

        const {id:userToChatId} = req.params;

        const senderId = req.user.\_id; // this is from the protect route

        const conversation = await conversationModel.findOne({

            participants : {$all : [senderId , userToChatId]}

        }).populate("messages"); // populate to populate the objectId of mongodb

        if(!conversation) return res.status(200).json([]); // if no conversation return blank array

        const messages = conversation.messages; //extrect the messages only

        res.status(200).json(messages);

    }catch(err){

            console.log("Error in getMessage Controller" , err.message);

            res.status(500).json({error : "Internal server error"});

    }

}

Get all users instead of us :

Create a new path : in server.js

app.use('/api/users' , usersRoutes); *// for get all the users insted of us*

Create a new route : in usersRoute.js

*//get all users insted of us*

router.get('/' , protectRoute , getUsersForSidebar);

Create a new contoller : in usersController.js

In this **$ne** means : not equal

*export* const getUsersForSidebar = *async*(req , res)=>{

    try{

        const loggedInUserId = req.user.\_id;

        const filterdUsers = await userModel.find({\_id : {$ne : loggedInUserId}})

        .select("-password"); // give user instead of us and remove password

        res.status(200).json(filterdUsers);

    }catch(err){

        console.log("Error in getAllUsersForSidebar Controller" , err.message);

        res.status(500).json({error : "Internal server error"});

    }

}

**FrontEnd Part is Started :**

**Tailwind installation for vite :**

Search : tailwindcss vite install

Click on first link

Copy this and run in frontend : **npm install -D tailwindcss postcss autoprefixer 🡪** this is for package installtion

**npx tailwindcss init –p 🡪** this is create 2 files 1.) tailwind.config.js 2.)postcss.config.js

copy index.css template and remove all the previous css and paste in index.css .

**For prebuild component we are using daisy ui website :**

Go in install and copy and install in frontend : **npm i -D daisyui@latest**

Copy plugins and paste into tailwind.config.js

**And again run the project**

**For better suggestions of tailwind install a extention :** tailwind css intelisense

Now all set…….

**For icon we use : react-icon**

1.) Search react icon

2.) click github link

3.) copy npm : npm install react-icons –save

Now all set …

To change the port of vite project :

Add server key in vite.config.js

import { defineConfig } from 'vite'

import react from '@vitejs/plugin-react'

*// https://vitejs.dev/config/*

*export* default defineConfig({

  plugins: [react()],

  server : {

    port : 3000

  }

})

**Now we install :**

React-router-dom for routing

For notification : search react-hot-toast

**Npm I react-hot-toast**

Import in the top component like app.jsx

 import {Toaster} from 'react-hot-toast';

<Toaster/>

To show the notifiaction : toast.error(<message>); // from anywhere in the project

If CORS error is coming :

Add proxy filed in vite.config.js and paste server port

*export* default defineConfig({

  plugins: [react()],

  server : {

    port : 3000,

    proxy : {

      "/api" : {

        target : "http://localhost:8000"

      }

    }

  }

})

Then

If we are fecthing data like this :

fetch('http://localhost:8000/api/auth/signup');

chanege to this :

fetch('/api/auth/signup') ;

CORS is like a prefix that add that target url in the fetching url .

Now the problem is solved..

**Signup fuctionalities:**

For this we create a custom hook : for custom hook file extenstion always .js not the .JSX because this hook return value not the jsx code.

const useSignup = ()=>{ // this is a hook

      const [loading , setLoading] = useState(true);

    const signup = *async*({fullname , username , password , confirmPassword , gender})=>{

        const success =  handleInputErrors({fullname , username , password , confirmPassword , gender});

        if(!success) return;

        setLoading(true);

        try{

           const res = await fetch('/api/auth/signup' , {

            method : 'POST' ,

            headers : {"content-type" : "application/json"},

            body : JSON.stringify({fullname , username , password , confirmPassword , gender})

           });

           const data = await res.json();

           console.log(data);

        }catch(error){

            toast.error(error.message);

        }finally{

            setLoading(false);

        }

    }

    return {loading , signup};

}

function handleInputErrors({fullname , username , password , confirmPassword , gender}){

         if(!fullname || !username || !password || !confirmPassword || !gender){

            toast.error("Please fill in all fields");

            return false;

         }

         if(password !== confirmPassword){

            toast.error('Passwords do not match');

            return false;

         }

         if(password.length < 6){

            toast.error('Password must be atleast 6 characters');

            return false;

         }

         return true;

}

*export* default useSignup;

//use this hook in Signup.jsx

const {loading , signup} = useSignup();

   const handleSubmit = *async*(e)=>{

    e.preventDefault();

       await signup(inputs); *// this is from useSignup hook*

   };

04/03/24

*Automatic naviagate on logged or alredy loggedin :*

To do this we are using useContext and local.storage

We create a context : AuthContext.jsx

import { createContext, useContext, useState } from "react";

*export* const AuthContext = createContext(); // create a context

*export* const useAuthContext = ()=>{

    return useContext(AuthContext);

}

*export* const  AuthContextProvider = ({children})=>{ // for wrapp all the componnet we create a function . children props gets all the child component of the wrpper

    const [authUser , setAuthUser] = useState(JSON.parse(localStorage.getItem("chat-user")) || null); // parse for convert into obj

        return <AuthContext.Provider *value*={{authUser ,setAuthUser}}> // pass the value so each child can use this values

            {children}

        </AuthContext.Provider>

};

// useSignup.js

*//set user in local storge*

           localStorage.setItem("chat-user" , JSON.stringify(data)); // set data into local storage

*//context*

           setAuthUser(data); // set data into context

// to use this context value : we wrapp top componet with the provider function. Now each child can use the context value

 <AuthContextProvider>

    <App />

    </AuthContextProvider>

// use

import { useAuthContext } from './context/AuthContext'

function App() {

     const {authUser} = useAuthContext(); // import context values

  return (

    <>

    <div *className*='p-4 h-screen flex justify-center items-center'>

      <Routes>

        <Route *path*="/" *element*={authUser ? <Home/> : <Navigate *to*="/login"/>}/>

        <Route *path*="/login" *element*={authUser ? <Navigate *to*="/"/> : <Login/>}/>

        <Route *path*="/signup" *element*={authUser ? <Navigate *to*="/"/> : <SignUp/> }/>

      </Routes>

      <Toaster/>

    </div>

    </>

  )

}

// logout   
first we create a custom hook. useLogout.js

import { useState } from "react";

import { useAuthContext } from "../context/AuthContext";

const useLogout = ()=>{

    const [loading , setLoading] = useState(false);

    const {setAuthUser} = useAuthContext();

    const logout = *async*()=>{

        setLoading(true);

        try{

            const res = await fetch("/api/auth/logout" , {

                method : "POST",

                headers : {'Content-Type' : 'application/json'}

            });

            const data = res.json();

            if(data.error){

                throw new Error(data.error);

            }

            localStorage.removeItem("chat-user"); // remove data from LS

            setAuthUser(null);

        }catch(error){

        }finally{

            setLoading(false);

        }

    }

    return {loading , logout};

}

*export* default useLogout;

//use this on logout btn click

function Logout() {

  const {loading , logout} = useLogout();

  return (

    <div *className*='mt-auto'>

      {!loading ? ( // here we know the use of loading . show the loader until the hook’s work is not completed

        <BiLogOut *className*='w-6 h-6 text-white cursor-pointer'

*onClick*={logout}

        />

      ):(

          <span *className*='loading loading-spinner'></span> // loader when the loading state is false

      )}

    </div>

  )

}

**Get conversation users :**

For this we create a hook : useGetCoversations.js

import { useEffect, useState } from "react"

import toast from "react-hot-toast";

const useGetConversations = () => {

  const [loading , setLoading ] = useState(false);

  const [conversations ,SetConversations] = useState([]);

  useEffect(() => {

  const getConversations = *async*()=>{

    setLoading(true);

    try {

        const res = await fetch('/api/users');

        const data = await res.json();

        if(data.error){

            throw new Error(data.error);

        }

        SetConversations(data);

    } catch (error) {

        toast.error(error.message);

    }finally{

        setLoading(false);

    }

  }

  getConversations();

  }, [])

  return {loading ,conversations};

}

*export* default useGetConversations

//use this in coversations.jsx

function Conversations() {

   const {loading , conversations}= useGetConversations(); // use there

  return (

    <div *className*='py-2 flex flex-col overflow-auto'>

      {conversations.map((conversation , idx)=>{

          return <Conversation

*key*={conversation.\_id}

*conversation*={conversation}

*emoji*={getRandomEmoji()}

*lastIdx*={idx == conversations.length-1}

          />

      })}

     {loading ? <span *className*='loading loading-spinner'></span> : null}

    </div>

  )

}

To get the selected convesation we use a libaray :

**ZUSTAND**  this is use to create a global state

To install visit : <https://docs.pmnd.rs/zustand/getting-started/introduction>

Copy : npm I zustand

Then create a function that is like global state :

import {create} from 'zustand'; // import this

const useConversation = create((set)=>({

   selectedConversation : null ,

   setSelectedConversation : (selectedConversation)=> set({selectedConversation}), //make sure that both name are same , green is data that we pass when we use this funcion and gary is key of this obj in which set the passed data

   message : [],

   setMessage : (message)=> set({message})

}));

*export* default useConversation;

now we can use this state’s value anywhere .

// Conversation.jsx

* First we use this to change the bg of selected chat user

import useConversation from '../../zustand/useConversation'

function Conversation({conversation , emoji , lastIdx}) {

    const {selectedConversation , setSelectedConversation} = useConversation(); // use here ZUSTAND

    const isSelected = selectedConversation?.\_id === conversation.\_id;

  return (

    <>

    <div *className*={`flex gap-2 items-center hover:bg-sky-500 rounded p-2 py-1 cursor-pointer

    ${ isSelected && 'bg-sky-500'}`}

*onClick*={()=>setSelectedConversation(conversation)} // update the values

    >

* Second we use this to show the chat when a user is selected

import useConversation from '../../zustand/useConversation';

function MessageContainer() {

  const {selectedConversation , setSelectedConversation} = useConversation(); // use here ZUSTAND

  useEffect(()=>{ *// to reset the state of selected conversation*

*// run on component unmount*

    return ()=>{

      setSelectedConversation(null);

    }

  }, [setSelectedConversation])

  return (

    <div *className*='md:min-w-[450px] flex flex-col'>

      {

         !selectedConversation ? (<NoChatSelected/>) :

         (<>

         <div *className*='bg-slate-500 px-2 py-2 mb-2 '>

         <span *className*='label-text'>To:</span><span *className*="text-gray-900 font-bold"> {selectedConversation.fullname}</span>

       </div>

         <Messages/>

        <MessageInput/>

        </>)

    }

    </div>

  )

}

05/03/2024

Today we create send message functionalites :

For this we create a hook that is : useSendMessage.js

import { useState } from "react"

import useConversation from "../zustand/useConversation";

import toast from 'react-hot-toast';

const useSendMessage = () => {

    const [loading , setLoading ] = useState(false);

    const {messages ,setMessages ,selectedConversation} = useConversation(); // 🡸 we need two things 1. Current selected user 2. Where we set the message in frontend

    const sendMessage = *async*(message)=>{ // we create a function

            setLoading(true);

            try{

                const res = await fetch(`/api/message/send/${selectedConversation.\_id}` , { // create a req

                    method : 'POST' ,

                    headers : {'Content-Type' : 'application/json'},

                    body : JSON.stringify({message})

                });

                const data = await res.json();

                if(data.error) throw new Error(data.error);

                setMessages([...messages , data]); // set the message in the messages array

            }catch(error){

                toast.error(error.message);

            }finally{

                    setLoading(false);

            }

    }

    return {loading , sendMessage}; // return the function

}

*export* default useSendMessage

we use that hook in the messageInput.jsx

function MessageInput() {

  const [message , setMessage] = useState("");

  const { loading , sendMessage } = useSendMessage(); // here we use

  const handleSubmit=*async*(e)=>{

    e.preventDefault();

      if(!message) return;

      await sendMessage(message); // call the function make sure that await is include because be call to hook function it takes some time

      setMessage("");

  }

Get message functionalities :

First we create a hook : useGetMessage.jsx

import {useEffect, useState} from 'react';

import toast from 'react-hot-toast';

import useConversation from '../zustand/useConversation.js'

const useGetMessages = ()=>{

   const [loading , setLoading] = useState(false);

   const {messages , setMessages , selectedConversation } = useConversation(); 🡸

   useEffect(()=>{

    const getMessage = *async*()=>{ // create a funcition

        setLoading(true);

        try{

            const res = await fetch(`/api/message/${selectedConversation.\_id}`); // create a req

            const data = await res.json();

            if(data.error) throw new Error(data.error);

            setMessages(data); // set into messages array

        }catch(error){

            toast.error(error.message);

        }finally{

            setLoading(false);

        }

    }

    if(selectedConversation?.\_id) getMessage(); // this will run when a conversation is selected

   } ,[selectedConversation?.\_id , setMessages]) // It is run on every conversation change

   return {loading , messages}; // now return a array not a function

}

Use this hook in the Messages.jsx

import React , {useRef , useEffect}from 'react'

import Message from './Message'

import useGetMessages from '../../hooks/useGetMessages'

import MessageSkeleton from '../skeleton/MessageSkeleton'

import ReactScrollToBottom from 'react-scroll-to-bottom'

function Messages() {

  const {loading , messages} = useGetMessages(); 🡸 here use

*// this is for scroll to bottom : for this we  create a div and take their reference using useRef*

  const lastMessageRef = useRef(); //4. for scroll to bottom when a new msg or chat select

  useEffect(()=>{

    setTimeout(()=>{

      lastMessageRef.current?.scrollIntoView({behavior : "smooth"}); // 5 . and apply a method that take evey msg into viewing state

    } , 100)

  } , [messages]);

*// this is for scroll to bottom*

  return (

   <div *className*='px-4 flex-1 overflow-auto pb-3'>

      {!loading && messages.length>0 &&  messages.map((message)=>

      <div *key*={message.\_id} *ref*={lastMessageRef}> //for take a reference of msg we create a div and take there reference

      <Message  *message*={message}/> // 3. Send one msg data to Message.jsx

      </div>

      )}

      {!loading && messages.length===0 && (

        <p *className*='text-center'>Send a message to start the conversation</p> //2. Set a default msg when there is no conversation

        )}

      {loading && [...Array(3)].map((\_ , idx)=> <MessageSkeleton *key*={idx}/>)} // 1. Create a loading skeleton until the msg is not fetched

   </div>

  )

}

*export* default Messages

then we update a msg according to there msg : Message.jsx

import {useAuthContext} from '../../context/AuthContext'

import useConversation from '../../zustand/useConversation';

import { extractTime }  from '../../utils/extractTime.js';

function Message({message}) {

  const {authUser} = useAuthContext(); // for current login user

  const {selectedConversation} = useConversation(); // selected user

  const fromMe = authUser.\_id === message.senderId; // this msg is send by me or not

  const chatClassname = fromMe ? 'chat-end' : 'chat-start'; // change class on user type (sender or reciver)

  const profilePic = fromMe ? authUser.profilePic : selectedConversation.profilePic; // change dp on user type (sender or reciver)

  const bubbleBgColor = fromMe && 'bg-blue-500'; // change bg of msg on user type (sender or reciver)

  const formatedTime = extractTime(message.createdAt); *// to convert into normal time*

  return (

    <div *className*={`chat ${chatClassname}`}>

    <div *className*='chat-image avatar'>

        <div *className*='w-10 rounded-full'>

           <img *src*={profilePic} />

        </div>

    </div>

    <div *className*={`chat-bubble text-white ${bubbleBgColor}`}>{message.message}</div>

    <div *className*='chat-footer opacity-50 text-xs flex gap-1 items-center'>{formatedTime}</div>

    </div>

  )

}

We create a function to format the time of mongodb(createdAt) to normal time :

*// this is copy from github*

*export* function extractTime(dateString) {

    const date = new Date(dateString);

    const hours = padZero(date.getHours());

    const minutes = padZero(date.getMinutes());

    return `${hours}:${minutes}`;

}

*// Helper function to pad single-digit numbers with a leading zero*

function padZero(number) {

    return number.toString().padStart(2, "0");

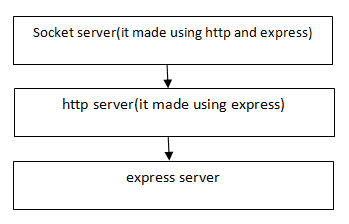
}

And use this function into Message.jsx that is written on above of this function .

06/03/2024

Implement socket.io for realtime communication :

How we can implemet a soket server :



**Backend**

Install : npm I socket.io in backend

First we create a file :soket.js

And create a soket.io server that contain http server that is made by express.

import {Server} from 'socket.io'; 🡨 need server

import http from 'http'; 🡨 need http

import  express  from 'express'; 🡨 need express

const app = express();

const server = http.createServer(app); 🡨 now we create a http server

const io = new Server(server , { 🡨 now we pass the http server to the socket server

and this obj for handle cors error

    cors : {

        origin : ["http://localhost:3000"], 🡨 the front end url

        methods : ['GET' , 'POST'] 🡨 methods that we use in the frontend

    }

});

io.on("connection" ,(socket)=>{ 🡨 this is run when a connection

  console.log("new user connected " , socket.id);

    socket.on("disconnect" , ()=>{ 🡨 for detect the connection

 console.log("user disconnected " , socket.id);

    })

})

*export* {app , io , server}; // export these methods

after that make sure the server is listening on http server(here is server that is made by http ) .

**Frontend**

For client side install : npm I socket.io-client

For provide the values to all the components we create a context that is SocketContext.jsx

import { createContext, useContext, useEffect, useState } from "react";

import io from 'socket.io-client'; 🡸 import io from soket.io-client

import {useAuthContext} from './AuthContext';

 const SocketContext = createContext(); 🡨 create a context

*export*  const useSocketContext=()=>{

    return useContext(SocketContext);

 }

*export* const SocketContextProvider = ({children})=>{

    const [socket ,setSoket] = useState(null);

    const [onlineUsers , setOnlineUser] = useState();

    const {authUser} = useAuthContext();

    useEffect(()=>{

        if(authUser){ 🡨 if user then create a socket

            const socket = io('http://localhost:8000' , { 🡨 this soket we get in the server

                query : { 🡨 pass the data to the server on socket created

                    userId : authUser.\_id

                }

            });

            setSoket(socket);

            return ()=> socket.close(); 🡨 close socket on unmount

        }else{

            if(socket){

                socket.close(); 🡨 if already a soket then close

                setSoket(null);

            }

        }

    } , [authUser])

    return <SocketContext.Provider *value*={{socket , onlineUsers}}>

       {children}

    </SocketContext.Provider>

}

And wrapp the app with this provider function.

If the socket connection is successful we get socket.id in the server console which we console.log().

Now we create a functionality for online users status:

//socket.js (server side)

import {Server} from 'socket.io';

import http from 'http';

import  express  from 'express';

const app = express();

const server = http.createServer(app);

const io = new Server(server , {

    cors : {

        origin : ["http://localhost:3000"],

        methods : ['GET' , 'POST']

    }

});

const userSocketMap = {}; *// it contain userId : socketId 🡸 1. We create a obj that contains all the current socket*

io.on("connection" ,(socket)=>{

    const userId = socket.handshake.query.userId; *// using this we can get the data that we pass from the client side 🡸 3. We get the userId that is pass from the frontend*

    if(userId !== 'undefined') userSocketMap[userId] = socket.id; 🡸 4. And add this in the obj that we create

*//io.emit() is used to send the events to all the connected clients*

    io.emit("getOnlineUsers" , Object.keys(userSocketMap)); 🡸 5. To send that data to the fronend we emit a event

*//socket.on() is used to listen to the events . can be used both client and server side*

    socket.on("disconnect" , ()=>{

        console.log("user disconnected " , socket.id);

        delete userSocketMap[userId]; 🡸 6. If the user is disconnect then we have to remove that id from the obj that store online users id

        io.emit('getOnlineUsers' , Object.keys(userSocketMap)); 🡸 7. Then again send updated data to client side

    })

})

*export* {app , io , server};

//SocketContext.jsx (client )

import { createContext, useContext, useEffect, useState } from "react";

import io from 'socket.io-client';

import {useAuthContext} from './AuthContext';

 const SocketContext = createContext();

*export*  const useSocketContext=()=>{

    return useContext(SocketContext);

 }

*export* const SocketContextProvider = ({children})=>{

    const [socket ,setSoket] = useState(null);

    const [onlineUsers , setOnlineUser] = useState();

    const {authUser} = useAuthContext();

    useEffect(()=>{

        if(authUser){

            const socket = io('http://localhost:8000' , {

                query : { 🡸2. on server create we pass the userId of current authuser to the backend

                    userId : authUser.\_id

                }

            });

            setSoket(socket);

*//socket.on() is used to listen to the events . can be used both client and server side*

            socket.on("getOnlineUsers" , (users)=>{ 🡸 8. To get that value we use socket.on() and pass the event name and callback fun take that values

                setOnlineUser(users); 🡸 9. and save that user in the frontend side

            })

            return ()=> socket.close();

        }else{

            if(socket){

                socket.close();

                setSoket(null);

            }

        }

    } , [authUser])

    return <SocketContext.Provider *value*={{socket , onlineUsers}}>

       {children}

    </SocketContext.Provider>

}

//To implement in the ui :

import useConversation from '../../zustand/useConversation'

import { useSocketContext } from '../../context/SocketContext';

function Conversation({conversation , emoji , lastIdx}) {

    const {selectedConversation , setSelectedConversation} = useConversation();

    const {onlineUsers } = useSocketContext(); 🡸 10. Get the onlineUsers using context

    const isOnline = onlineUsers.includes(conversation.\_id); 🡸11. Return true if this conversation id is present in the online users obj .

  return (

    <>

    <div *className*={`flex gap-2 items-center hover:bg-sky-500 rounded p-2 py-1 cursor-pointer

    ${ isSelected && 'bg-sky-500'}`}

*onClick*={()=>setSelectedConversation(conversation)}

    >

    <div *className*={`avatar ${isOnline ? "online" : ''}`}> 🡸 12 . use that boolean value

        <div *className*="w-12 rounded-full">

            <img *src*={conversation.profilePic}/>

        </div>

   </div>

   <div *className*='flex flex-col flex-1 '>

    <div *className*='flex gap-3 justify-between'>

        <p *className*='font-bold text-gray-200'>{conversation.fullname}</p>

        <span *className*='text-xl'>{emoji}</span>

    </div>

   </div>

  </div>

    {!lastIdx && <div *className*='divider my-0 py-0 h-1'/>}

    </>)

}

*export* default Conversation

now the functionality is created ……

Realtime message functionalities :

Socket.js(server side )

import {Server} from 'socket.io';

import http from 'http';

import  express  from 'express';

const app = express();

const server = http.createServer(app);

const io = new Server(server , {

    cors : {

        origin : ["http://localhost:3000"],

        methods : ['GET' , 'POST']

    }

});

const userSocketMap = {}; *// it contain userId : socketId*

*export* const getReceiverSoketId = (receiverId)=>{ 🡸 1. For this we create a function that return the socket id of the reciever

    return userSocketMap[receiverId];

}

io.on("connection" ,(socket)=>{

    console.log("new user connected " , socket.id);

*//socket.on() is used to listen to the events . can be used both client and server side*

    socket.on("disconnect" , ()=>{

    })

})

*export* {app , io , server};

use this function in the messageController.js(server side)

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

    try {

        const {message} = req.body;

        const {id:receiverId} = req.params; *// rename id to reciverId  using :*

        const senderId = req.user.\_id; *// we can now use this because be add the user in the req by using middleware(protectRoute)*

        let conversation = await conversationModel.findOne({

            participants :{$all : [senderId , receiverId]}

        });

        if(!conversation){

            conversation = await conversationModel.create({

                participants : [senderId , receiverId]

            })

        }

        const newMessage = new messageModel({

            senderId,

            receiverId ,

            message

        });

        if(newMessage){

            await conversation.messages.push(newMessage.\_id);

        }

*//await newMessage.save();*

*//await  conversation.save();  // there is a problem  it will  wait for previous work complete*

*//!    or write in promise way*

        await Promise.all([newMessage.save() , conversation.save()]); *//using this this will run in parallel*

*// socket.io functionality is here*

        const receiverSocketId = getReceiverSoketId(receiverId); 🡸 2.use that function and pass recever id in this

        if(receiverSocketId){

*// io.to(<soket\_id>).emit() is use to send event to a specific client*

            io.to(receiverSocketId).emit("newMessage" , newMessage); 🡸 3. If receverSocketid then emit a event only those specific user

        }

        res.status(200).json(newMessage);

    } catch (error) {

        res.status(500).json({error : "Internal server error"});

        console.log("Error in  sendMessage Controller" ,error.message);

    }

}

Fronend :-- now create a hook – useListenMessage.js

import { useSocketContext } from "../context/SocketContext"

import useConversation from '../zustand/useConversation.js'

const useListenMessage = () => {

     const {socket}  = useSocketContext();

     const {messages , setMessages}  = useConversation();

     useEffect(()=>{

        socket?.on("newMessage" , (newMessage)=>{ 🡸 4. to get that new message we use socket.on() and pass that event name ans callback fun get that values

            setMessages([...messages , newMessage]); 🡸 5. add that new message to the messages array

        });

        return ()=> socket.off("newMessage");

     },[socket , messages , setMessages]);

}

*export* default useListenMessage

and use this function in the messages.jsx

function Messages() {

  const {loading , messages} = useGetMessages();

  useListenMessage(); 🡸 6. Use there

now the functionalitiy is completed ….

Shake new msg feature :

Add animation css in index.js(client )

.shake {

    animation: shake 0.82s cubic-bezier(0.36, 0.07, 0.19, 0.97) 0.2s both;

    transform: translate3d(0, 0, 0);

    backface-visibility: hidden;

    perspective: 1000px;

}

@keyframes shake {

    10%, 90% {transform: translate3d(-1px, 0, 0);}

20%,80% {transform: translate3d(2px, 0, 0);

30%,50%,70% {transform: translate3d(-4px, 0, 0);}

40%,60% {transform: translate3d(4px, 0, 0); }

}

//useListenMessage.js(client)

import { useSocketContext } from "../context/SocketContext"

import useConversation from '../zustand/useConversation.js'

const useListenMessage = () => {

     const {socket}  = useSocketContext();

     const {messages , setMessages}  = useConversation();

     useEffect(()=>{

        socket?.on("newMessage" , (newMessage)=>{

            newMessage.shouldShake = true; 🡸 1. we add a property in the new message

            setMessages([...messages , newMessage]); 🡸 2. add that msg in the msg array

        });

        return ()=> socket.off("newMessage");

     },[socket , messages , setMessages]);

}

*export* default useListenMessage

//messages.jsx

import React from 'react'

import {useAuthContext} from '../../context/AuthContext'

import useConversation from '../../zustand/useConversation';

import { extractTime }  from '../../utils/extractTime.js';

function Message({message}) {

  const shakeClass = message.shouldShake ? "shake" : ''; 🡸3. check that property

  return (

    <div *className*={`chat ${chatClassname}`}>

    <div *className*='chat-image avatar'>

        <div *className*='w-10 rounded-full'>

           <img *src*={profilePic} />

        </div>

    </div>

    <div *className*={`chat-bubble break-words text-white ${bubbleBgColor} ${shakeClass} 🡸 4. use here

`}>{message.message}</div>

    <div *className*='chat-footer opacity-50 text-xs flex gap-1 items-center'>{formatedTime}</div>

    </div>

  )

}

*export* default Message;

feature Completed ….

Add sound on new msg feature :

For this we save a audio file in the assets .

import notificatioSound from '../assets/sound/notification.mp3'; 🡸1. import that file

const useListenMessage = () => {

     const {socket}  = useSocketContext();

     const {messages , setMessages}  = useConversation();

     useEffect(()=>{

        socket?.on("newMessage" , (newMessage)=>{

            newMessage.shouldShake = true;

            const sound = new Audio(notificatioSound); 🡸 2. Use the Audio function

            sound.play(); 🡸 3. And play the sound

            setMessages([...messages , newMessage]);

        });

        return ()=> socket.off("newMessage"); 🡸 4. make sure that soket is off after every ring

     },[socket , messages , setMessages]);

}

*export* default useListenMessage

feature completed ………..