**Milestone 4**

**Product Summary**

**Name of the product: ProClick**

**Motivation & Importance**

E-commerce is the fastest growing business during the past decade, considering the circumstances now, it is indeed one of the rapidly growing phenomena in Pandemic. **ProClick** aims to provides a platform which brings buyers and sellers together encouraging social e-commerce within the university. We have seen enormous potential and growth during pandemic for used products and services and how natives are more inclined towards buying through online channels.

**Functions & Services**

Our mission is to provide a platform where users can sell and purchase images. Having a distinct search filters that refines user experience, also boosts customer retention, and increases user sales. ClickPro will increase customer satisfaction when consumers deal with sellers directly via built-in-messaging system provided in our platform where consumers can negotiate or ask for product or service-related questions from sellers. Our team is focused to provide a technology that will have an immense upside potential and we strive to improve the overall image of the online merchant and therefore stimulate growth of online shopping.

**Major Committed Functions**

**1. Browse products**

Image Listing page that shows a list of images based on category or search criteria.

**2. Filter images**

Specific attribute a visitor/shopper can use to refine the search results or images searches of a particular image list page. e.g., by category, price range etc.

**3. Search for a image**

Allowing/Enabling the customers to find certain images that satisfy / fulfill a criterion.Also allow the voice search.

**4. Registration**

Allowing the sellers and customers to make a new account profile for themselves.

**5. Login**

Allowing the customers and sellers to log in to their account or dashboard.

**6. Post images**

Allowing the seller to upload a new image for selling purpose from their dashboard / account.

**7. Messaging**

Allowing the customers and sellers to communicate with each other through messages and also enables them to see the previous message history as well from their dashboard.

**8. Approve/Reject items**

Allowing the admins / owners to approve / reject different pending products of different sellers from their dashboard.

**URL of the website: ProClick (the link may change due to redeploying)**

**Usability Test Plan**

**• Test objectives**

The objective of this testing is to get the user feedback and user experience about searching for a specific product using searched keywords, price range filter or choosing specific product category

* 1. • **Test background and setup**
  2. **System setup**

ClickPro website have been published on AZURE to be tested by participants using the following technologies

• Operating System: Ubuntu 18.04 Server

• Database: MySQL v: 5.7

• Web Server: NGINX 1.18

* 1. • Server-Side Language: Node.js

Client-Side : ReactJS

* 1. **Starting point**

Participants will test ProClick website using their laptops or smartphones, and will focus on the search functionality in the home page

* 1. **The intended users :**   
     The test will be conducted with targeted category of users, which are the students and university employees of Fulda. Therefore, we will ask four users of the university to test the website   
       
     **URL of the system to be tested   
     MyClickPro** (the link may change due to redeploying)   
       
     **o What is to be measured**  
     We will be focusing on user satisfaction
  2. o **Tester 1**   
     **Phase: 1 Screening & Pretest**   
       
     **Q1. How old are you?**   
     28  
     **Q2. What is your profession?**   
     Student Service Consultant   
     **Q3. When was the last time you purchased an item online?**   
     Last week   
     **Q4. How often do you shop online?**   
     usually   
     **Q5. Which devices you usually you for online shopping?**   
     apparel   
       
     **Phase: 2 Usability Task description**   
      **Task 1:** Imagine tomorrow is your mother’s birthday and you are looking for some gift that ranges from 500 to 1000 $.?   
       
     **Task 2: Imagine** last week you.  
       
     **Task 3:** Imagine you have got bonus from your office for good work and wants to buy some beautiful pictures for your parents.   
       
     **Phase: 3 Post Test Questions**   
       
     **Q.1 What was your overall impression or experience when searching for a images that ranges from 50 to 100$**   
     It was a good experience, and I found the right one in the range of 50 to 100$.

**Q.2 What was the best or worst thing about searching for a specific product?**

The worst thing is that it is giving less choice to choose from and it is very difficult to know that whether the search works or not and it will be good if I will be able to use some other filters as well for searching. It will be good if I would be able to see sub-categories as well in searching option.

**Q.3 How would you compare it with another competitor?**

I think it is a simple app but if I compare it with most of the online shopping platforms,they provides many filters and most of them don’t work properly but the good thing is that so far you have limited filters but at least they are working correctly.

**Q.4 How did you find the experience of using this website to complete this task?**

It was a good experience, and I found the image I was thinking to buy easily.

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Task Objective** | **Scale 0/5** |
| **TASK 1 (Tester1)** | Search for Product using image filter | 4/5 |
| **TASK 2(Tester1)** | Search for Image using searched Keywords | 3/5 |
| **TASK 3(Tester1)** | Search for Image by choosing specific  category | 3/5 |

* 1. **Tester 2**

**Phase: 1 Screening & Pretest**

**Q1. How old are you?**

23

**Q2. What is your profession?**

Software Engineer

**Q3. When was the last time you purchased an item online?**

Two days ago

**Q4. How often do you shop online?**

usually

**Q5. Which devices you usually use for online shopping?**

Laptop, mobile

**Phase: 2 *Usability Task description:***

**Task 1:** Imagine that After two weeks is the valentine day and you want to buy a gift for your girlfriend, and you are searching for some random gifts.

**Task 2:** Imagine that you found on the social media a very interesting book about Engineers and you want to buy it.

**Task 3:** Imagine that you want to practice some sports and you want to look for a bike.

**Phase: 3 Post Test Questions**

**Q.1 What was experience when you were searching for a random product?**

I could not find the product without selecting a category which I realized that I get all the products if my search is not available, but it is better to do something like an alert or error to let the user know what happened after he/she did the search

**Q.2 What was the best or the bad thing about searching for a specific image?**

The worst thing that I could not understand what happened after clicking on search button and as I mentioned before it should be an alert for the user to know what happened after any search, he/she did. The best part is that I got my image, and it was displayed in a good way.

**Q.3 How would you compare it with other competitors?**

It is till now a very basic website, but what you achieved in this specific and limited time is very impressive by implementing all the functionality and the best part is the search functionality even with limited filtering was working, and If I want to compare it with other competitors it should be fully functional and also voice search make it more efficient.

**Q.4 How did you find the experience of using this website to complete this task?**

My experience was good regarding the search functionality and the result that I got, but you can do better guys.

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Task Objective** | **Scale 0/5** |
| **TASK 1 (Tester2)** | Search for a random Image without selecting a  category | 2/5 |
| **TASK 2(Tester2)** | Search for Image using price filter | 3/5 |
| **TASK 3(Tester2)** | Search for Image by using search keywords | 3/5 |

* 1. **Tester 3**

**Phase: 1 Screening & Pretest**

**Q1. How old are you?**

24

**Q2. What is your profession?**

Master Student at Fulda University

**Q3. When was the last time you purchased an item online?**

Month ago ***ProClick Project Documentation***

**Q4. How often do you shop online?**

usually

**Q5. Which devices you usually you for online shopping?**

mobile

**Q6. Have you ever used the similar site before?**

yes

**Phase: 2 *Usability Task description:***

**Task 1:** Imagine that After two weeks is the valentine day and you want to buy a gift for your boyfriend, and you are searching for some random gifts.

**Task 2:** Imagine that you are tired of your old wallpapers and now looking for a new for your room

**Task 3:** Imagine that you got your salary slip and wants to buy a new wallpapers.

**Phase: 3 Post Test Questions**

**Q.1 What was experience when you were searching for a random images?**

No feedback or animation shows that the search is working especially when there is no result and return everything (like page refresh). It feels like the search button is not working. At least it should show at the header “no matching result found for the search Criteria”

**Q.2 What was the best or the bad thing about searching for a specific image?**

The best part is that the interface is user friendly and easy to use, but needs to improve the search fields and would it lovely to have auto complete which helps in search, also It’s better to show “Please Select Category” instead “No Category Selected” , and the price filter should be improved like using a slider component

**Q.3 How would you compare it with other competitors?**

***ProClick Project Documentation***

It is a simple website, but what you achieved in this specific and limited time is very impressive.

**Q.4 How did you find the experience of using this website to complete this task?**

My experience was very good regarding the search function and it is very well placed at the top and have clear criteria separation

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Task Objective** | **Scale 0/5** |
| **TASK 1 (Tester2)** | Search for a random Image without selecting a  category | 3/5 |
| **TASK 2(Tester2)** | Search for Image using price filter | 2/5 |
| **TASK 3(Tester2)** | Search for Image by using search keywords | 3/5 |

**QA Test Plan**

**• Test Objectives**

The test objectives are to verify the Functionality of website ProClick, the project should focus on testing the search for images to guarantee this functionality can work normally in real business environment and to ensure that the bugs/issues are identified and fixed before going live

**• HW and SW setup**

Laptop and smartphones using Firefox and chrome web browsers

• **Feature to be tested**

All the functionalities of the website should be tested and verified, but will concentrate on the “search for a images” feature

**• QA Test plan**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test #** | **Test Title** | **Test Description** | **Test Input** | **Expected Result** | **Test Results/Chrome** | **Test Results/Firefox** |
| **1** | Search for a specific Image | Search for a specific image by its name | In the search fields, enter “Cat” in the name field | Retrieves all the active images that have Cats. | **** | **** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **2** | Search for products of a specific category | Search for products of the furniture category | In the search fields, choose category “Furniture” | Retrieves all the furniture active products | **** | **** |
| **3** | Search for products within a price range | Search for products of the furniture category within (200-  1500) euro | In the search fields, choose min price 200 and max price 1500 | Retrieves all the active products that its price is within the specified  range | **** | **** |
| **4** | Search for unavailable product | Search for a product that doesn’t exist in the ProClick | In the search fields, enter “Cat” in the name field. | Retrieves all the active products as there is no product available with the specified criteria | **** | **** |
| **5** | Clear Search Criteria | Remove all the search criteria | Press the clear button | Retrieves all the active products without any filtration | **** | **** |

**Code Review**

Chat Functionality

* Done by Hamza Mazhar
* Reviewed by Ahmad Hassan

Frontend Code in ReactJS

/\*

 \*Contributor : Hamza Mazhar

 \*/

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

import axios from "axios";

import MessageBox from "./MessageBox";

// eslint-disable-next-line no-undef

const APIENDPOINT = process.env.REACT\_APP\_API\_BASE\_URL;

function Conversation() {

  const [buyerList, setBuyerList] = useState([]);

  useEffect(() => {

    const sellerId = JSON.parse(localStorage.getItem("user"))?.user.id;

    // prettier-ignore

    const getConversations = async () => {

      // prettier-ignore

        const bodyParameters = {

          // prettier-ignore

          seller\_id: sellerId,

        };

        // prettier-ignore

      const res = await axios.post(

        APIENDPOINT + `/conversation/uniqueConversation`, bodyParameters

      );

      setBuyerList(res.data.result.record);

      console.log(res.data);

    };

    getConversations();

  }, []);

  const handleRouteChange = (data) => {

    console.log(data);

    let userData = JSON.parse(localStorage.getItem("user"));

    if (userData) {

      let path = `/chat/?name=${userData.user.first\_name}&room=${data.id}&senderId=${userData.user.id}&conversationId=${data.id}`;

      var win = window.open(path, "\_blank");

      win.focus();

    }

  };

  return (

    <div>

      {console.log(buyerList)}

      {buyerList?.map((data, idx) => {

        return (

          <div key={idx}>

            <button onClick={() => handleRouteChange(data)}>

              <MessageBox name={data.buyerData} />

            </button>

          </div>

        );

      })}

    </div>

  );

}

export default Conversation;

import \* as React from "react";

import Box from "@mui/material/Box";

import Card from "@mui/material/Card";

import CardActions from "@mui/material/CardActions";

import CardContent from "@mui/material/CardContent";

import Button from "@mui/material/Button";

import Typography from "@mui/material/Typography";

import MessageIcon from "@mui/icons-material/Message";

export default function MessageBox({ name }) {

  return (

    <Box sx={{ minWidth: 275 }}>

      <Card variant="outlined">

        <React.Fragment>

          <CardContent>

            <Typography variant="h5" component="div">

              {name}

            </Typography>

          </CardContent>

          <CardActions>

            <Button size="small">

              <MessageIcon />

            </Button>

          </CardActions>

        </React.Fragment>

      </Card>

    </Box>

  );

}

import React from "react";

import ReactEmoji from "react-emoji";

import "./Message.css";

import Stc from "string-to-color";

//Contributor : Hamza Mazhar

export const Message = ({ message, name }) => {

  let { user, text, time } = message;

  let myName = name.trim();

  let trimmedName = name.trim().toLowerCase();

  // cheepi need to defined the proper logic in future

  if (user === undefined) {

    user = message.name;

    myName = message.name.trim();

    trimmedName = message.name.trim().toLowerCase();

  }

  // if (user !== undefined) {

  //   chatterName = message?.name | name;

  // }

  let isSentByCurrentUser = false;

  let isAdminMsg = false;

  let isDate = false;

  if (user === "admin") {

    isAdminMsg = true;

  } else if (user === "date") {

    isDate = true;

  } else {

    if (user === trimmedName) {

      isSentByCurrentUser = true;

    }

  }

  function getColor(value) {

    return Stc(value).replace("#", "");

  }

  function checkRTL(s) {

    let ltrChars =

        "A-Za-z\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u02B8\u0300-\u0590\u0800-\u1FFF" +

        "\u2C00-\uFB1C\uFDFE-\uFE6F\uFEFD-\uFFFF",

      rtlChars = "\u0591-\u07FF\uFB1D-\uFDFD\uFE70-\uFEFC",

      rtlDirCheck = new RegExp("^[^" + ltrChars + "]\*[" + rtlChars + "]");

    return rtlDirCheck.test(s);

  }

  function formatTime(time) {

    let date = new Date(time);

    // console.log(date.toUTCString());

    return (

      ("0" + date.getHours()).slice(-2) +

      ":" +

      ("0" + date.getMinutes()).slice(-2)

    );

  }

  let isRTL = checkRTL(text);

  let timeText = formatTime(time);

  return isAdminMsg ? (

    <div className="messageContainer justifyCenter">

      <div className="messageBox backgroundYellow">

        <p className="messageText">{text}</p>

      </div>

    </div>

  ) : isDate ? (

    <div className="messageContainer justifyCenter">

      <div className="messageBox backgroundLightBlue">

        <p className="messageText">{text}</p>

      </div>

    </div>

  ) : isSentByCurrentUser ? (

    <div className="messageContainer justifyEnd">

      <div className="messageBox background">

        <p className={isRTL ? `messageTextRTL` : `messageText`}>

          {ReactEmoji.emojify(text)}

        </p>

        <span className={isRTL ? `msgTimeStampRTL` : `msgTimeStamp`}>

          {timeText}

        </span>

      </div>

      <span className="bubbleTailOther">

        <svg

          xmlns="http://www.w3.org/2000/svg"

          viewBox="0 0 8 13"

          width="8"

          height="13"

        >

          <path

            opacity=".13"

            d="M5.188 1H0v11.193l6.467-8.625C7.526 2.156 6.958 1 5.188 1z"

          />

          <path

            fill="currentColor"

            d="M5.188 0H0v11.193l6.467-8.625C7.526 1.156 6.958 0 5.188 0z"

          />

        </svg>

      </span>

      <img

        src={`https://ui-avatars.com/api/?name=${myName}&rounded=true&size=40&background=${getColor(

          myName

        )}&color=fff&font-size=0.4&bold=true`}

        className="userAvatar"

        alt=""

      />

    </div>

  ) : (

    <div className="messageContainer justifyStart">

      <img

        src={`https://ui-avatars.com/api/?name=${user}&rounded=true&size=40&background=${getColor(

          user

        )}&color=fff&font-size=0.4&bold=true`}

        className="userAvatar"

        alt=""

      />

      <span className="bubbleTail">

        <svg

          xmlns="http://www.w3.org/2000/svg"

          viewBox="0 0 8 13"

          width="8"

          height="13"

        >

          <path

            opacity=".13"

            d="M5.188 1H0v11.193l6.467-8.625C7.526 2.156 6.958 1 5.188 1z"

          />

          <path

            fill="currentColor"

            d="M5.188 0H0v11.193l6.467-8.625C7.526 1.156 6.958 0 5.188 0z"

          />

        </svg>

      </span>

      <div className="messageBox backgroundLight">

        <p className={isRTL ? `messageTextRTL` : `messageText`}>

          <span className="userName" style={{ color: `#${getColor(user)}` }}>

            {user}

          </span>

          {ReactEmoji.emojify(text)}

        </p>

        <span className={isRTL ? `msgTimeStampRTL` : `msgTimeStamp`}>

          {timeText}

        </span>

      </div>

    </div>

  );

};

import "./messenger.css";

import Topbar from "../../components/topbar/Topbar";

import Conversation from "../../components/conversations/Conversation";

import Message from "../../components/message/Message";

import ChatOnline from "../../components/chatOnline/ChatOnline";

import React, { useContext, useEffect, useRef, useState } from "react";

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

import axios from "axios";

import { io } from "socket.io-client";

//Contributor : Hamza Mazhar

export default function Messenger() {

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

  const [currentChat, setCurrentChat] = useState(null);

  const [messages, setMessages] = useState([]);

  const [newMessage, setNewMessage] = useState("");

  const [arrivalMessage, setArrivalMessage] = useState(null);

  const [onlineUsers, setOnlineUsers] = useState([]);

  const socket = useRef();

  const { user } = useContext(AuthContext);

  const scrollRef = useRef();

  useEffect(() => {

    // socket.current = io("ws://localhost:8900");

    socket.current = io("http://localhost:8900", {

      autoConnect: true,

    });

    socket.current.on("getMessage", (data) => {

      setArrivalMessage({

        sender: data.senderId,

        text: data.text,

        createdAt: Date.now(),

      });

    });

  }, []);

  useEffect(() => {

    arrivalMessage &&

      currentChat?.members.includes(arrivalMessage.sender) &&

      setMessages((prev) => [...prev, arrivalMessage]);

  }, [arrivalMessage, currentChat]);

  useEffect(() => {

    socket.current.emit("addUser", user.\_id);

    socket.current.on("getUsers", (users) => {

      setOnlineUsers(

        user.followings.filter((f) => users.some((u) => u.userId === f))

      );

    });

  }, [user]);

  useEffect(() => {

    const getConversations = async () => {

      try {

        const res = await axios.get("/conversations/" + user.\_id);

        setConversations(res.data);

      } catch (err) {

        console.log(err);

      }

    };

    getConversations();

  }, [user.\_id]);

  useEffect(() => {

    const getMessages = async () => {

      try {

        const res = await axios.get("/messages/" + currentChat?.\_id);

        setMessages(res.data);

      } catch (err) {

        console.log(err);

      }

    };

    getMessages();

  }, [currentChat]);

  const handleSubmit = async (e) => {

    e.preventDefault();

    const message = {

      sender: user.\_id,

      text: newMessage,

      conversationId: currentChat.\_id,

    };

    const receiverId = currentChat.members.find(

      (member) => member !== user.\_id

    );

    socket.current.emit("sendMessage", {

      senderId: user.\_id,

      receiverId,

      text: newMessage,

    });

    try {

      const res = await axios.post("/messages", message);

      setMessages([...messages, res.data]);

      setNewMessage("");

    } catch (err) {

      console.log(err);

    }

  };

  useEffect(() => {

    scrollRef.current?.scrollIntoView({ behavior: "smooth" });

  }, [messages]);

  return (

    <>

      <Topbar />

      <div className="messenger">

        <div className="chatMenu">

          <div className="chatMenuWrapper">

            <input placeholder="Search for friends" className="chatMenuInput" />

            {conversations?.map((c, index) => (

              <div onClick={() => setCurrentChat(c)} key={index}>

                <Conversation conversation={c} currentUser={user} />

              </div>

            ))}

          </div>

        </div>

        <div className="chatBox">

          <div className="chatBoxWrapper">

            {currentChat ? (

              <>

                <div className="chatBoxTop">

                  {messages?.map((m, index) => (

                    <div ref={scrollRef} key={index}>

                      <Message message={m} own={m.sender === user.\_id} />

                    </div>

                  ))}

                </div>

                <div className="chatBoxBottom">

                  <textarea

                    className="chatMessageInput"

                    placeholder="write something..."

                    onChange={(e) => setNewMessage(e.target.value)}

                    value={newMessage}

                  ></textarea>

                  <button className="chatSubmitButton" onClick={handleSubmit}>

                    Send

                  </button>

                </div>

              </>

            ) : (

              <span className="noConversationText">

                Open a conversation to start a chat.

              </span>

            )}

          </div>

        </div>

        <div className="chatOnline">

          <div className="chatOnlineWrapper">

            <ChatOnline

              onlineUsers={onlineUsers}

              currentId={user.\_id}

              setCurrentChat={setCurrentChat}

            />

          </div>

        </div>

      </div>

    </>

  );

}

Backend Code:

//author: hamza\_mazhar

"use strict";

var debug = require("debug");

var express = require("express");

var path = require("path");

var logger = require("morgan");

var cookieParser = require("cookie-parser");

var bodyParser = require("body-parser");

const cors = require("cors");

var routes = require("./routes/index");

const imageDataRoutes = require("./routes/image-data.routes");

const UserRoutes = require("./routes/user.routes");

const ReviewsRoutes = require("./routes/reviews.routes");

const SalesRoutes = require("./routes/sales.routes");

const socketio = require("socket.io");

const ChatRoutes = require("./routes/chat.routes");

const ConversationRoutes = require("./routes/conversation.routes");

// const server = http.createServer(app);

const Pool = require("pg").Pool;

const {

  addUser,

  removeUser,

  getUser,

  getUsersInRoom,

} = require("./routes/socketUser");

const pool = new Pool({

  user: "username",

  host: "host",

  database: "db",

  password: "password",

  port: 5432,

});

const PORT = process.env.PORT || 3333;

var app = express();

// view engine setup

app.set("views", path.join(\_\_dirname, "views"));

app.set("view engine", "pug");

// uncomment after placing your favicon in /public

//app.use(favicon(\_\_dirname + '/public/favicon.ico'));

app.use(logger("dev"));

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, "uploads")));

//app.use(express.static(path.join(\_\_dirname, "public")));

//app.get('/\*', function (req, res) {

//    res.sendFile(path.join(\_\_dirname, '/public/index.html'));

//});

//app.use(express.static('public'));

//app.use('/images', express.static('images'));

//app.get('/\*', function (req, res) {

//    res.sendFile(path.join(\_\_dirname, '/public/index.html'));

//});

app.use(cors());

app.use((req, res, next) => {

  res.header("Access-Control-Allow-Origin", "\*");

  res.header(

    "Access-Control-Allow-Headers",

    "Origin, X-Requested-With, Content-Type, Accept, Authorization"

  );

  if (req.method === "OPTIONS") {

    res.header("Access-Control-Allow-Methods", "PUT, POST, PATCH, DELETE, GET");

    return res.status(200).json({});

  }

  next();

});

// using as middleware

app.use("/", routes);

app.use("/api/media", imageDataRoutes);

app.use("/api/user", UserRoutes);

app.use("/api/review", ReviewsRoutes);

app.use("/api/chat", ChatRoutes);

app.use("/api/conversation", ConversationRoutes);

app.use("/api/sales", SalesRoutes);

// catch 404 and forward to error handler

app.use(function (req, res, next) {

  var err = new Error("Not Found");

  err.status = 404;

  next(err);

});

// error handlers

// development error handler

// will print stacktrace

if (app.get("env") === "development") {

  app.use(function (err, req, res, next) {

    res.status(err.status || 500);

    res.render("error", {

      message: err.message,

      error: err,

    });

  });

}

// production error handler

// no stacktraces leaked to user

app.use(function (err, req, res, next) {

  res.status(err.status || 500);

  res.render("error", {

    message: err.message,

    error: {},

  });

});

app.set("port", process.env.PORT || 3000);

var server = app.listen(app.get("port"), function () {

  debug("Express server listening on port " + server.address().port);

});

//socket programming for the chat

//TODO need to change the logic

// Static Data checking without db

// end socket here

const io = socketio(server);

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

  let address = socket.request.connection.remoteAddress;

  socket.on("join", ({ name, room, senderId }, callback) => {

    if (room === undefined || room == null || room === "") {

      room = "public"; //Default

    }

    if (name && room) {

      const { error, user } = addUser({

        id: socket.id,

        name,

        room,

        ip: address,

      });

      if (error) return callback(error);

      if (!!user && user.room) {

        socket.join(user.room);

        socket.emit("message", {

          user: "admin",

          text: `Hi ${user.name}, welcome to ${user.room} chat room`,

          name: user.name,

          socket: socket.id,

        });

        if (user) {

          socket.broadcast.to(user.room).emit("message", {

            user: "admin",

            text: `${user.name} has joined`,

            name: user.name,

            socket: socket.id,

          });

          io.to(user.room).emit("roomData", {

            room: user.room,

            users: getUsersInRoom(user.room),

          });

        }

      }

    }

    callback();

  });

  socket.on("sendMessage", async (message, callback) => {

    const user = await getUser(socket.id);

    if (user !== undefined) {

      io.to(user.room).emit("message", {

        user: user.name,

        text: message.text,

        time: message.time,

        socket: socket.id,

      });

    }

    callback();

  });

  socket.on("sendMessageNotification", (message, callback) => {

    const user = getUser(message.name);

    if (user !== undefined) {

      io.to(user.room).emit("message", {

        user: user.name,

        text: message.text,

        socket: socket.id,

      });

    }

    callback();

  });

  socket.on("disconnect", () => {

    const user = removeUser(socket.id);

    if (user) {

      io.to(user.room).emit("message", {

        user: "admin",

        text: `${user.name} has left.`,

      });

      io.to(user.room).emit("roomData", {

        room: user.room,

        users: getUsersInRoom(user.room),

      });

    }

  });

});

//author: hamza\_mazhar

require("dotenv").config();

// Load model

const { Chat } = require("../config/mysqldatabase.config");

// create conversation

module.exports.Create = async (req, res, next) => {

  try {

    const record = await Chat.create(req.body);

    return res.json({

      status: "success",

      result: {

        record: record,

      },

    });

  } catch (err) {

    return next(

      res.status(500).json({

        error: {

          message: "Internal error occurred",

          reason: err,

        },

      })

    );

  }

};

//// fetch conversation by userID

module.exports.fetch = async (req, res, next) => {

  try {

    const conversationId = req.body.conversationId;

    const record = await Chat.findAll({

      where: {

        conversationId: conversationId,

      },

    });

    return res.json({

      status: "success",

      result: {

        record: record,

      },

    });

  } catch (err) {

    return next(

      res.status(500).json({

        error: {

          message: "Internal error occurred",

          reason: err,

        },

      })

    );

  }

};

//author: hamza\_mazhar

require("dotenv").config();

// Load model

const { Conversation, User } = require("../config/mysqldatabase.config");

const { Op } = require("sequelize");

const Sequelize = require("sequelize");

// create conversation

module.exports.Create = async (req, res, next) => {

  try {

    const buyerId = req.body.buyerId;

    const sellerId = req.body.sellerId;

    const recordExist = await Conversation.findAll({

      where: {

        [Op.or]: [

          {

            [Op.and]: [

              {

                sellerId: sellerId,

              },

              {

                buyerId: buyerId,

              },

            ],

          },

          {

            [Op.and]: [

              {

                sellerId: buyerId,

              },

              {

                buyerId: sellerId,

              },

            ],

          },

        ],

      },

    });

    if (recordExist.length > 0) {

      return next(

        res.status(500).json({

          error: {

            message: "record already exist",

            reason: "record already exist",

          },

        })

      );

    } else {

      const record = await Conversation.create({

        buyerId: buyerId,

        sellerId: sellerId,

      });

      return res.json({

        status: "success",

        result: {

          record: record,

        },

      });

    }

  } catch (err) {

    return next(

      res.status(500).json({

        error: {

          message: "Internal error occurred",

          reason: err,

        },

      })

    );

  }

};

//// fetch conversation by userID

module.exports.fetch = async (req, res, next) => {

  try {

    const userId = req.params.userId;

    const record = await Conversation.findAll({

      where: {

        [Op.or]: [{ sellerId: userId }, { buyerId: userId }],

      },

    });

    return res.json({

      status: "success",

      result: {

        record: record,

      },

    });

  } catch (err) {

    return next(

      res.status(500).json({

        error: {

          message: "Internal error occurred",

          reason: err,

        },

      })

    );

  }

};

// fetch conversation id  by user and reciver if not create new conversation and return the new obj

//// fetch conversation by userID

// this end point used in the buyer dashboard fetch the conversaiton id or existing conversation

module.exports.fetchByUserAndReciver = async (req, res, next) => {

  try {

    const buyerId = req.body.buyer\_Id;

    const sellerId = req.body.seller\_Id;

    const record = await Conversation.findAll({

      where: {

        [Op.or]: [

          {

            [Op.and]: [

              {

                sellerId: sellerId,

              },

              {

                buyerId: buyerId,

              },

            ],

          },

          {

            [Op.and]: [

              {

                sellerId: buyerId,

              },

              {

                buyerId: sellerId,

              },

            ],

          },

        ],

      },

    });

    if (record.length > 0) {

      return res.json({

        status: "success",

        result: {

          record: record[0],

        },

      });

    } else {

      const record = await Conversation.create({

        buyerId: buyerId,

        sellerId: sellerId,

      });

      return res.json({

        status: "success",

        result: {

          record: record,

        },

      });

    }

  } catch (err) {

    return next(

      res.status(500).json({

        error: {

          message: "Internal error occurred",

          reason: err,

        },

      })

    );

  }

};

// this end point is used for the seller dashboard to get the unique conversation

//// fetch unique conversation by sellerId

module.exports.fetchUniqueConversation = async (req, res, next) => {

  try {

    const sellerId = req.body.seller\_id;

    const record = await Conversation.findAll({

      // include: [

      //   {

      //     model: User,

      //     as: "conversation",

      //     required: true,

      //     where: { id: Conversation.buyerId },

      //   },

      // ],

      where: { sellerId: sellerId },

      attributes: [

        [Sequelize.fn("DISTINCT", Sequelize.col("buyerId")), "buyerId"],

      ].concat(Object.keys(Conversation.rawAttributes)),

    });

    let finalData = [];

    if (record.length > 0) {

      for (let i = 0; i < record.length; i++) {

        let userData = await User.findAll({

          where: { id: record[i].buyerId },

        });

        let obj = {

          buyerData: userData[0].first\_name,

          ...record[i].dataValues,

        };

        finalData.push(obj);

      }

    }

    return res.json({

      status: "success",

      result: {

        record: finalData,

      },

    });

  } catch (err) {

    console.log("++++++++++jere is error++", err);

    return next(

      res.status(500).json({

        error: {

          message: "Internal error occurred",

          reason: err,

        },

      })

    );

  }

};