CSE471: System Analysis and Design
Project Report
Project Title: Project Title Here

Group No: 6, CSE471 Lab Section: 1, Fall 2024	
ID	**Name**
21301449	Farhan Abedin
21201147	Eusra Ibnat
20101184	MD Waliul Islam

Submission Date: 07. 05. 2025

Table of Contents

1. System Request

Business need:
The Real Estate Application is designed to simplify the property buying, selling, and renting process by offering users a centralized digital platform to browse available listings, communicate with property owners or agents, and post their own properties for sale or rent. It meets the growing demand for fast, transparent, and location-aware real estate services by integrating core features like map-based property viewing and built-in chat functionality. Users can also book slots to view the property.

Business requirements:
Using the Real Estate Application, users will be able to search for properties, connect with sellers or landlords, and list their own properties easily. The key functionalities the system should include are:

- Provide an intuitive, map-integrated interface for users to browse property listings by location.
- Allow users to communicate directly with property owners, agents, or potential buyers via a secure messaging system.
- Enable users to post properties for sale or rent with detailed information, including images, pricing, availability, and contact details.
- Offer account-based access where users can manage their listings, chat history, and saved properties.

Business value:
We expect the Real Estate Application to streamline the property discovery and transaction process, reducing the friction often encountered in traditional real estate platforms. By integrating real-time chat and geolocation tools, the app allows users to quickly act on listings, ask questions, and negotiate directly. For sellers and landlords, the app offers a fast and convenient way to reach potential buyers or renters without needing to rely on third-party agents.
The platform's clean design and ease of use will attract both experienced and first-time users looking for property solutions. As adoption increases, the application's scalable infrastructure will allow for expanded services such as smart filtering, AI-based recommendations, or integration with external mortgage providers. This creates long-term value through user retention and platform growth.

Special issues or constraints:

- The application should maintain simplicity and performance, especially on mobile devices, since many users are expected to access the platform while on the go.
- Real-time communication and location services must be secure and privacy-compliant, as users will be sharing sensitive personal and financial information.
- The platform must remain affordable and intuitive, particularly to appeal to users in markets where traditional real estate platforms are seen as too complex or costly.

2. Functional Requirements

Module 1:

- **Sign up/log in:** The user will have a login or sign-up option.
- **Property Listings:** Property owners and agents should be able to add, edit, and delete property listings with basic details.
- **Search Properties:** The app will allow users to search for properties based on location.
- **Filter Properties:** Users will be able to filter properties by price range, property type, and the number of bedrooms.

Module 2:

- **Save properties:** The app will allow users to save properties for easy access.
- **Admin panel:** The admin panel allows full control to manage, update, and delete user accounts and property listings.
- **Notifications:** Users will be notified if someone sends them texts.
- **Map integration:** Property locations will be displayed through the integration of maps.

Module 3:

Availability Status: The user can view the availability status of a listed property.

In-app chat system integration: A simple chat system will be included to help the users contact property owners.

Booking slots: Potential renters or buyers can schedule appointments for their preferred times.

Track payments: The app tracks renter payments, calculates installments, manages advance payments, displays due dates, and sends notifications for upcoming or overdue payments.

3. Technology (Framework, Languages)

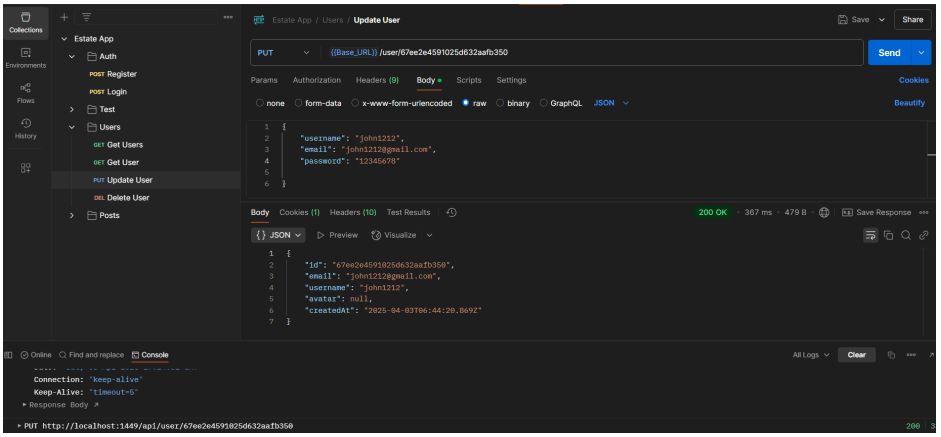
Javascript, MERN stack(MongoDB, ExpressJs, ReactJs, NodeJs)

4. Backend Development

1. Updating User Information:

```
export const updateUser = async (req, res) => {
  const id = req.params.id;
  const tokenUserId = req.userId;
  const { password, avatar, ...inputs } = req.body;
  let updatedPassword = null;
  try {
    if (password) {
      updatedPassword = await bcrypt.hash(password, 10);
    }
    const updatedUser = await prisma.user.update({
      where: { id },
      data: {
        ...inputs,
        ...(updatedPassword && { password: updatedPassword }),
        ...(avatar && { avatar }),
      },
    });
    const { password: userPassword, ...rest } = updatedUser;
    res.status(200).json(rest);
  } catch (err) {
    console.log(err);
    res.status(500).json({ message: "Failed to update users!" });
  }
};
```

This API is for updating the information of the user. A user can update their information even after signing up for the first time. From the Update profile in the profile page, one can update their username, password, and upload their picture or avatar. However, there is no change in UserIds and his posts. The particular user has to log in. Other than the user himself, the Admin can change the information of a user. But the Admin can only change and update the password of a user upon request.

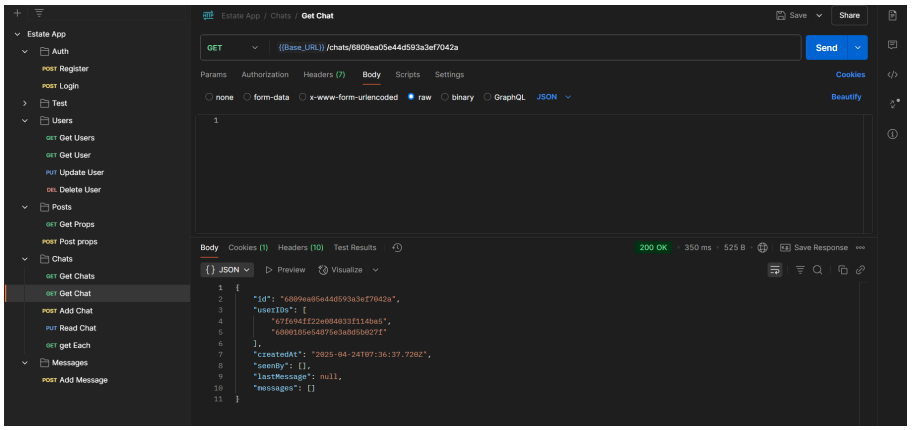


2. Chats Between Two Users

```
export const getChat = async (req, res) => {
  const tokenUserId = req.userId;

  try {
    const chat = await prisma.chat.findUnique({
      where: {
        id: req.params.id,
        userIDs: {
          hasSome: [tokenUserId],
        },
      },
      include: {
        messages: {
          orderBy: {
            createdAt: "asc",
          },
        },
      },
    });

    await prisma.chat.update({
      where: {
        id: req.params.id,
      },
      data: {
        seenBy: {
          push: [tokenUserId],
        },
      },
    });
    res.status(200).json(chat);
  } catch (err) {
    console.log(err);  " PrismaClientKnownRequestError { name: 'PrismaClientKnownRe
    res.status(500).json({ message: "Failed to get chat!" });
  }
};
```

This is used to view the chat texts of two users. When the add chat icon is clicked, it creates a unique ChatId having both of the UserIds. Then the whole chat system is used

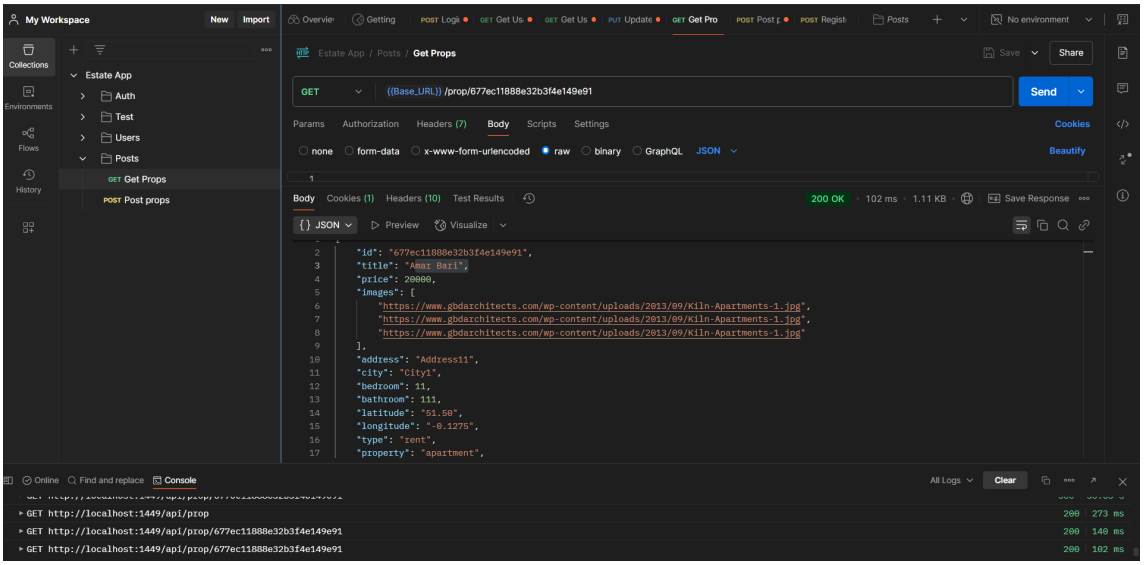
based on the ChatId to store the texts of the two users. The texts are kept in the messages list, and initially, there is no message between the users. The text messages are kept in ascending order, and the latest text is shown upon opening the chat. It also tracks the time upon sending and checks if the texts are seen or not.



3. Fetching a single post

```
export const getPost = async (req, res) =>{
    const id = req.params.id;
    try{
        const post = await prisma.post.findUnique({
            where:{id},
            include: {
                postDetail: true,
                user: {
                    select:{
                        username: true,
                        avatar:true
                    }
                },
            }
        })
        res.status(200).json(post)
    }catch(err){
        console.log(err)
        res.status(500).json({message:"Failed to get any post"})
    }
}
```

This API fetches the descriptions and related information of a single property that has been posted. Each property listing has its own unique property ID, and only a verified user can post a listing. When viewing a property, the pictures, images, title, additional infos, Longitude and latitude (to show on the map), type of the property are fetched. When a specific property card is clicked, all the information of a property is shown in the Property Page along with the User who posted it.

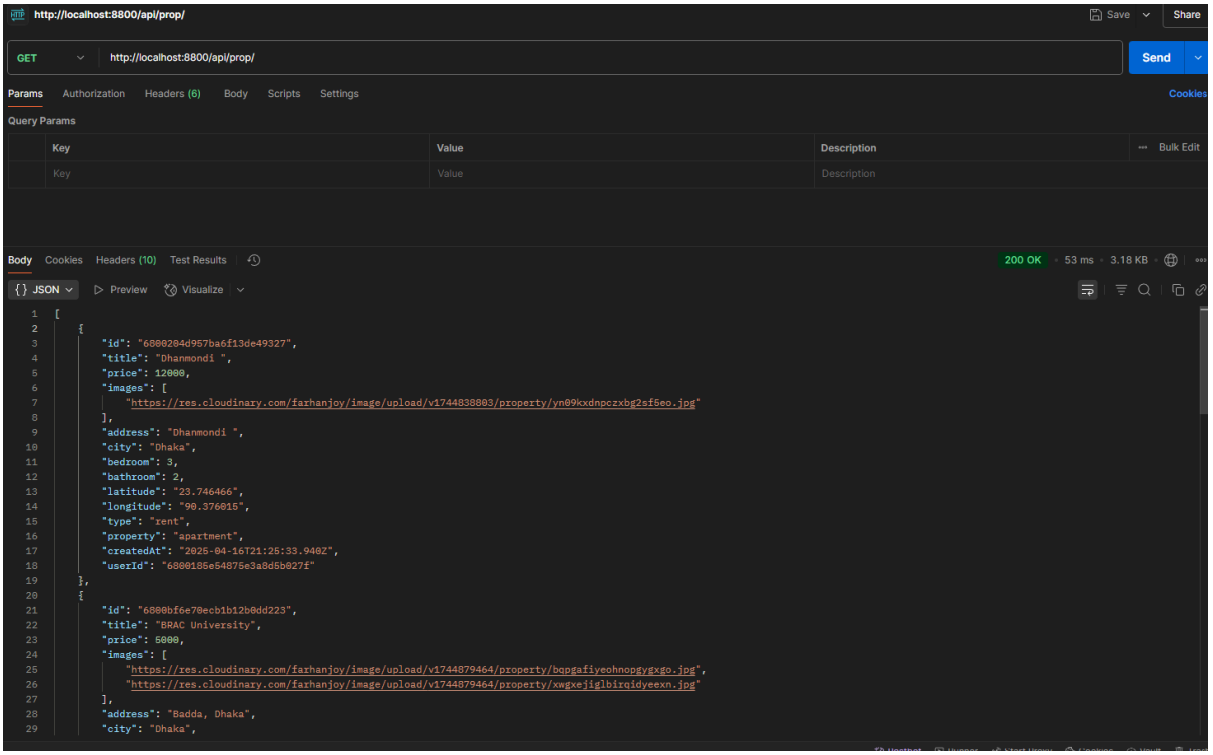


4. Fetching all posts

```
CSE471_Project > api > controllers > JS prop.controller.js > getPost > post > include > user
1   import prisma from "../lib/prisma.js";
2
3   export const getPosts = async (req, res) => {
4       const query = req.query;
5
6       try {
7         const posts = await prisma.post.findMany({
8           where: {
9             city: query.city || undefined,
10            type: query.type || undefined,
11            property: query.property || undefined,
12            bedroom: parseInt(query.bedroom) || undefined,
13            price: {
14              gte: parseInt(query.minPrice) || undefined,
15              lte: parseInt(query.maxPrice) || undefined,
16            },
17          },
18        });
19
20        res.status(200).json(posts);
21      } catch (err) {
22        console.log(err);
23        res.status(500).json({ message: "Failed to get posts" });
24      }
25  };
```

This API fetches the descriptions and related information of all the properties that have been posted. Each property listing has its unique property ID, and only a verified user can post a listing. When viewing a property, their address, number of bedrooms, number of bathrooms,

longitude, latitude, type of the property(apartment, condo, etc), if the property is for rent or buy, etc is fetched. When a property card is clicked, all the info of a property is shown in the Property Page along with the User who posted.

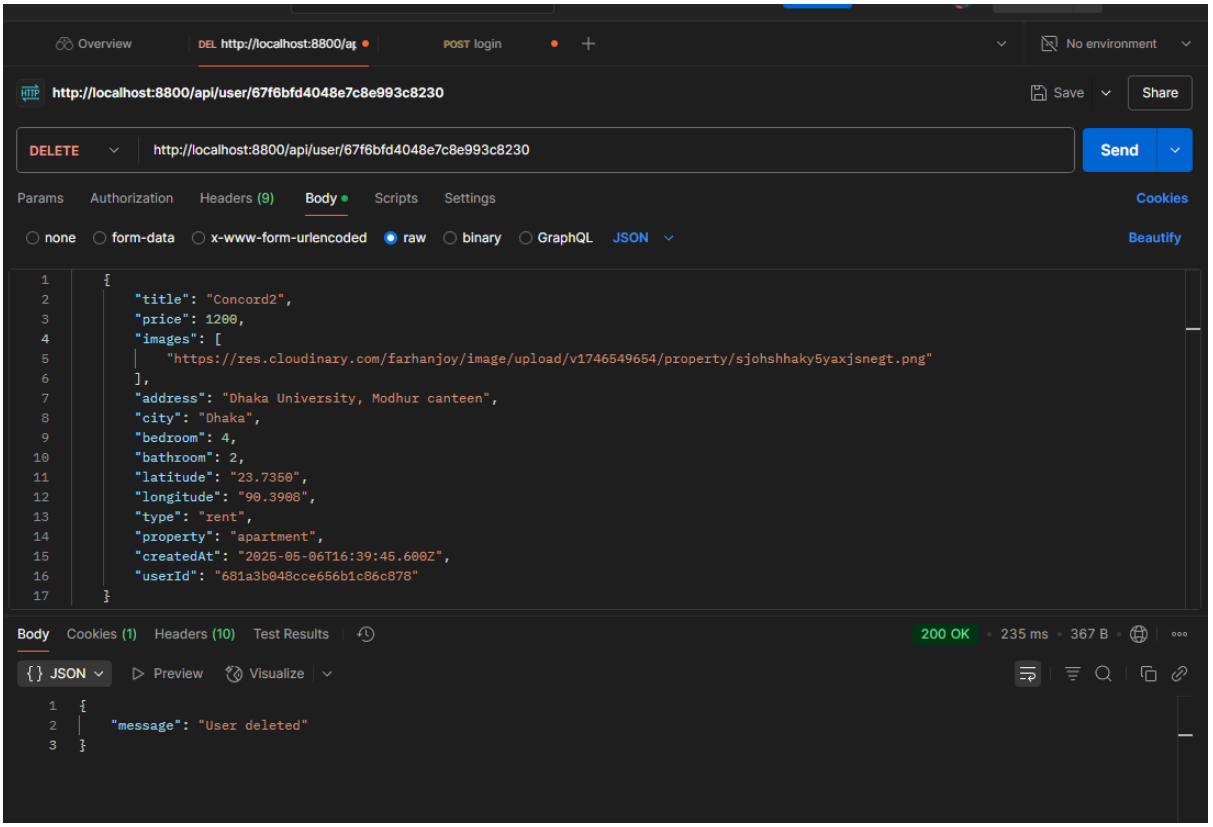


5. Delete a user

```
export const deleteUser = async (req, res) => {
  const id = req.params.id;
  const tokenUserId = req.userId;

  try {
    await prisma.user.delete({
      where: { id },
    });
    res.status(200).json({ message: "User deleted" });
  } catch (err) {
    console.log(err);
    res.status(500).json({ message: "Failed to delete users!" });
  }
};
```

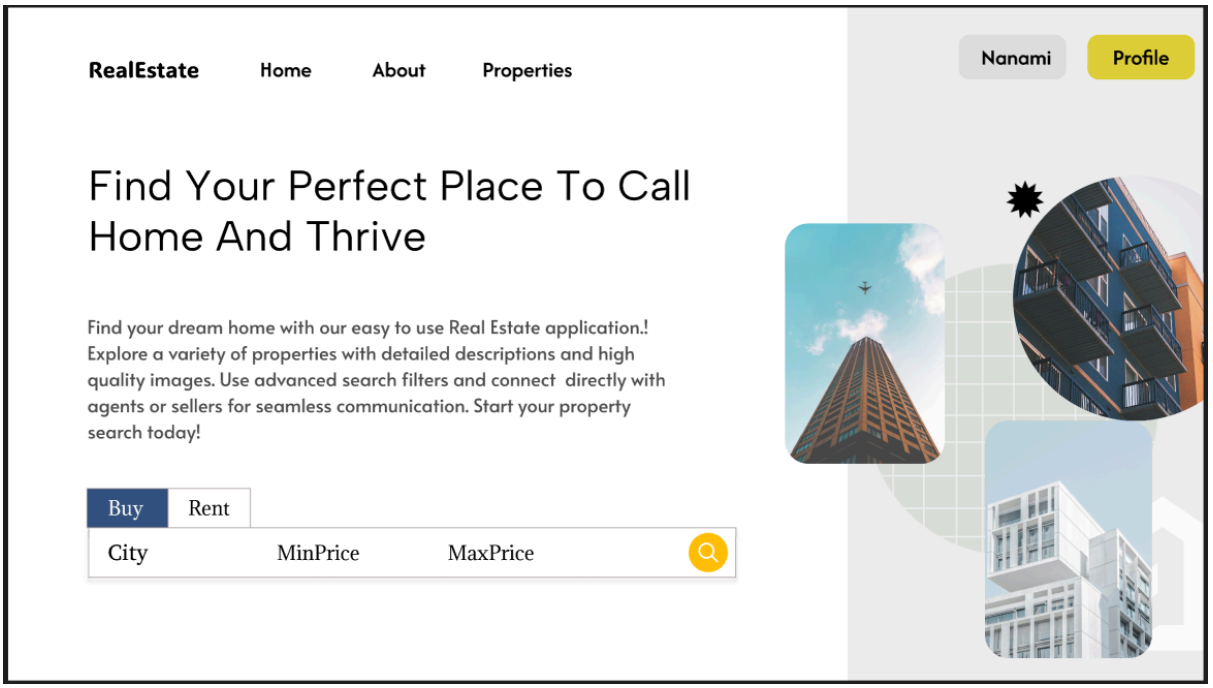
This API deletes a user from the system based on their unique user ID. Only a verified admin can perform this action. When a delete request is made, the API checks the ID of the user making the request and confirms if they have admin privileges. If authorized, the specified user is permanently removed from the database. This ensures user data management is secure

and controlled by admins only. If an unauthorized user tries to delete someone, the request is denied with a proper error message.

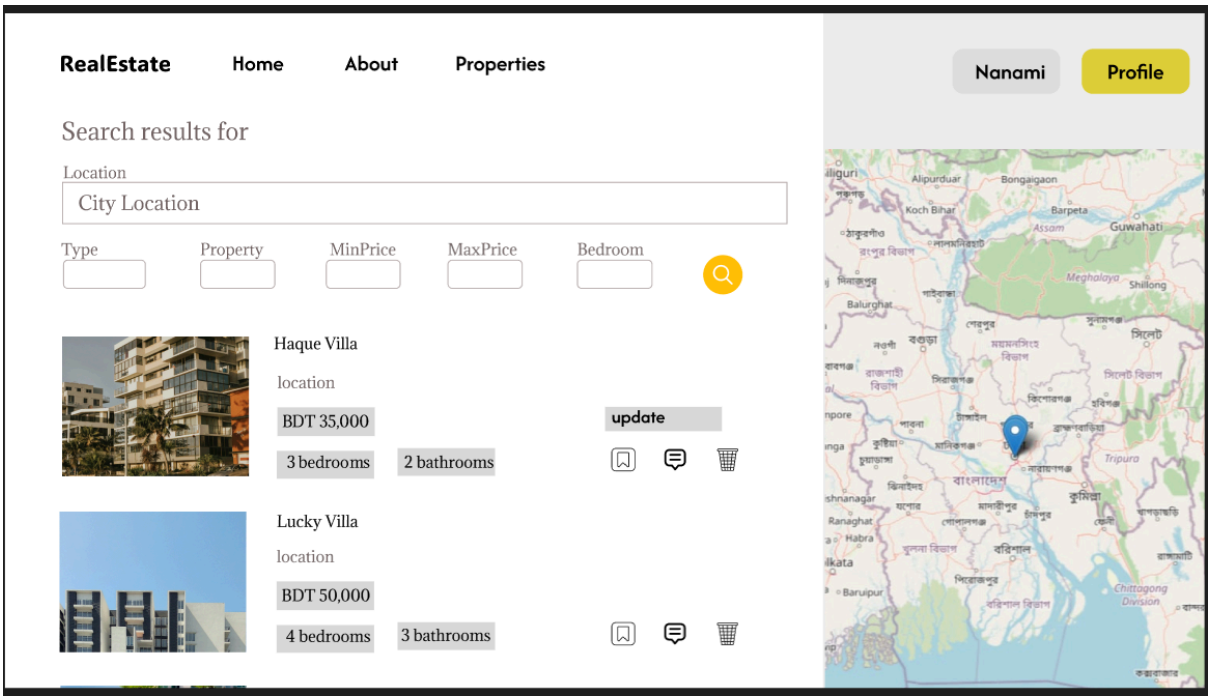


5. User Interface Design

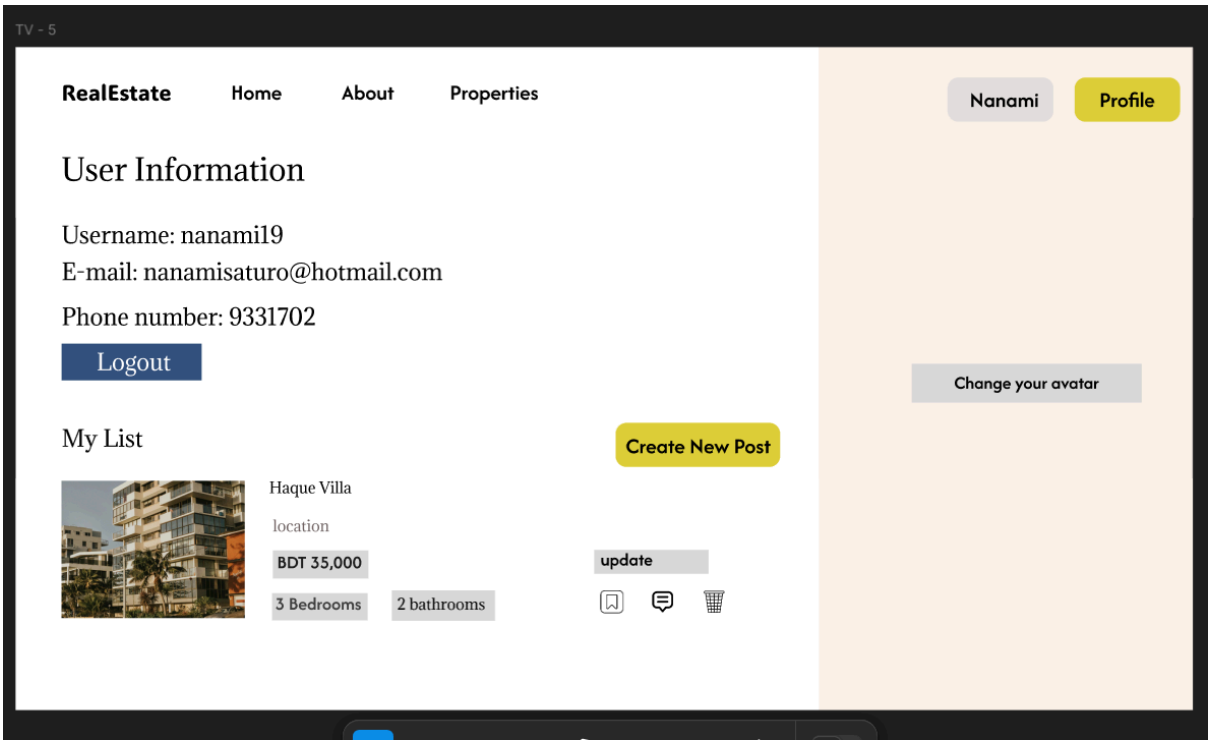
Homepage:



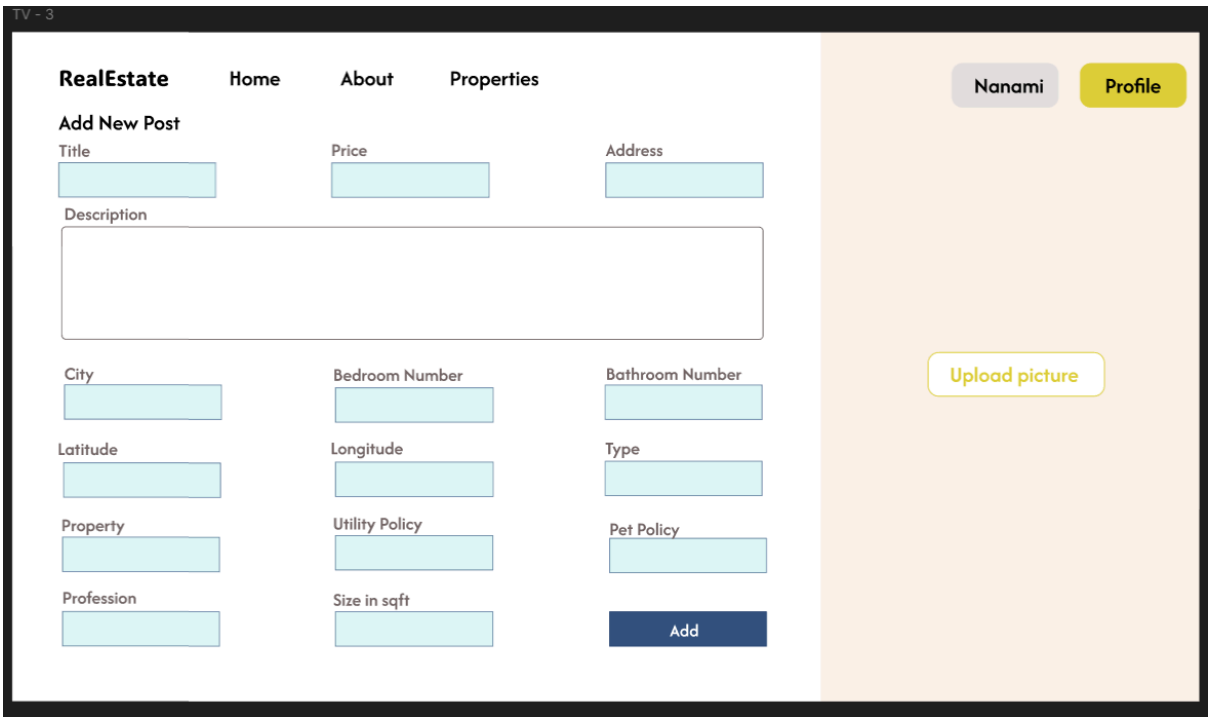
Property Listings:



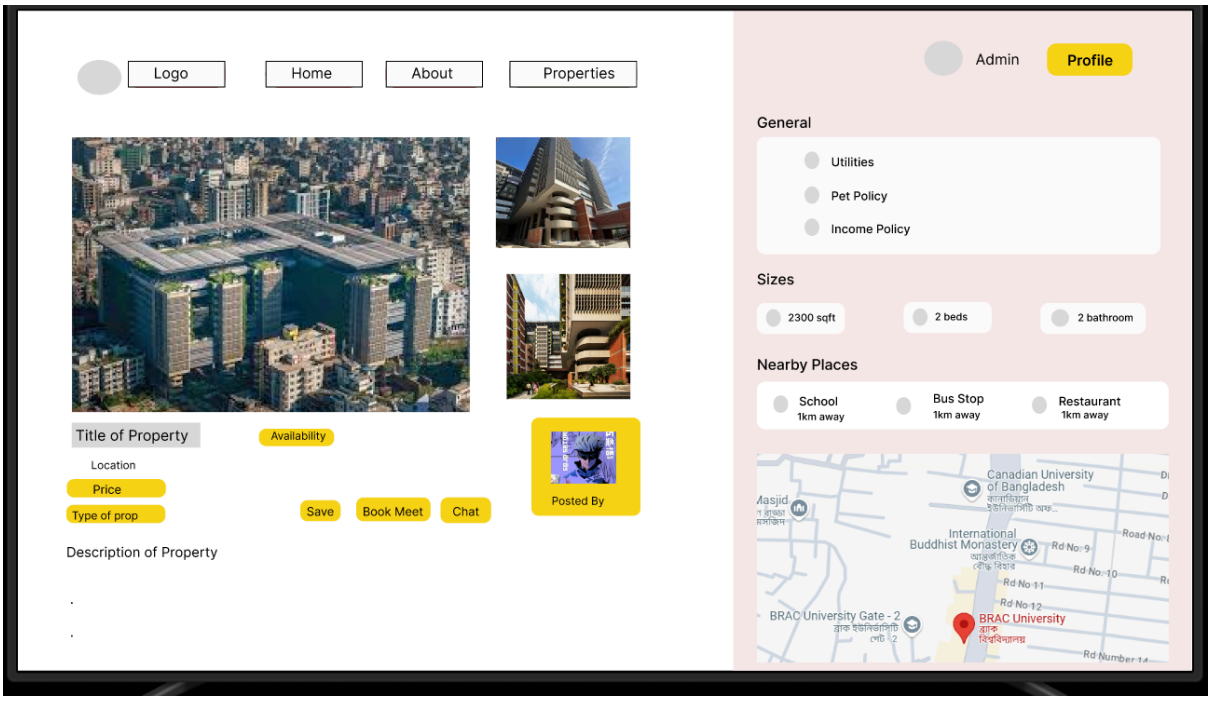
User Information:



Create Post:



Property Information:



Farhan:
https://www.figma.com/proto/nJKlnnBlO1m4QfOWGtaTwp/CSE471_Pro?node-id=29-67&t=KPCK32JPXctUHrEf-1&starting-point-node-id=1%3A3

Eusra
https://www.figma.com/proto/inKmGAvM4rasVlF68xGY4H/Untitled?node-id=35-1618&p=f&t=eAYu0AWJbEvbohzY-1&scaling=scale-down&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=5%3A41

6. Frontend Development

1. User Login

```jsx
import "./login.scss";
import { Link, useNavigate  } from "react-router-dom";
import { useContext, useState } from "react";
import apiRequest from "../../lib/apiRequest";
import { AuthContext } from "../../context/AuthContext";
import axios from "axios";
function Login() {
  const [error, setError]=useState("");
  const [isLoading, setIsLoading]=useState(false)
  const { updateUser } = useContext(AuthContext)

  const navigate = useNavigate()

  const handleSubmit= async (e) =>{
    e.preventDefault()
    setIsLoading(true)
    setError("")

    const formData = new FormData(e.target);

    const username = formData.get("username")
    const password = formData.get("password")

    try{
      const res = await apiRequest.post("/auth/login",{
        username, password
      })
      updateUser(res.data)

      navigate("/profile")

    }catch(err){
      console.log(err)
      setError(err.response.data.message);
    }finally{
      setIsLoading(false)
    }
  }
  return (
    <div className="login">
      <div className="formContainer">
        <form onSubmit={handleSubmit}>
          <h1>Welcome</h1>
          <input name="username" type="text" placeholder="Username" />
          <input name="password" type="password" placeholder="Password" />
          <button disabled={isLoading}>Login</button>
          {error && <span>{error}</span>}
          <p>
          Don't have an account? <Link to="/register">Register</Link>
          </p>
        </form>
      </div>
      <div className="imgContainer">
      <img src="/bg.png" alt="" />
        </div>
    </div>
  );
}

export default Login;
```

This code defines a login form component for a real estate application using React. It includes styled input fields for username and password, a login button, and error handling. On form submission, it sends a login request to the backend and updates the user context on success, redirecting to the profile page. The layout is visually split into a form section and an image section for better UI appeal. It uses cookies of the user, and upon successful authentication, the user is redirected to their profile. The component also includes a visual section with a background image to enhance the UI, and a link to navigate users to the registration page if they don't have an account.

2. Profile Page

client > src > routes > profilePage > profilePage.jsx > ProfilePage

```jsx
import List from "../../components/list/List";
import { Suspense, useContext, useEffect, useState } from "react";
import { Link, useLoaderData, useNavigate, Await } from "react-router-dom";
import { AuthContext } from "../../context/AuthContext";
import apiRequest from "../../lib/apiRequest";
import "./profilePage.scss";
import Chat from "../../components/chat/chat"


function ProfilePage() {
  const { updateUser, currentUser } = useContext(AuthContext);
  const [userPosts, setUserPosts] = useState([]);
  const navigate = useNavigate()
  const data = useLoaderData();

  const handleAdmin = () => {
    navigate("/admin");
  };

  useEffect(() => {
    const handlePosts = async () => {
      try {
        const response = await apiRequest.get("/prop/");
        const filteredData = response.data.filter(
          (item) => item.userId === currentUser.id
        );
        setUserPosts(filteredData);
      } catch (error) {
        console.error("Error fetching posts:", error);
      }
    };
    handlePosts();
  }, [currentUser.id]);

  const handleLogout = async () => {
    try {
      await apiRequest.post("/auth/logout");
      updateUser(null);
      navigate("/home");
    } catch (err) {
      console.log(err);
    }
  };
  console.log(userPosts)  ×106 ˅ [] ⌛ ˅ [] ⌛ ˅ [] ⌛ ˅ [] ⌛ ˅ [] ⌛ ˅ [
  return (
    <div className="profilePage">
      <div className="details">
        <div className="wrapper">
          <div className="title">
            <h1>User Information</h1>
            <Link to="/profile/update">
              <button>Update Profile</button>
            </Link>
          </div>
          <div className="info">
            <span>
              Avatar:
              <img src={currentUser.avatar || "noavatar.jpg"} alt="" />
            </span>
            <span>
```

```
            </span>
            <span>
              Username: <b>{currentUser.username}</b>
            </span>
            <span>
              E-mail: <b>{currentUser.email}</b>
            </span>
            <br />
            <button onClick={handleLogout}>Logout</button>
            {currentUser.email === "admin@gmail.com" && (
              <button onClick={handleAdmin}>Admin</button>
            )}
          </div>
          <div className="title">
            <h1>My List</h1>
            <Link to="/add">
              <button>Create New Post</button>
            </Link>
          </div>
          <List posts={userPosts} />
        </div>
      </div>
      <div className="chatContainer">
        <div className="wrapper">
          <Suspense fallback={<p>Loading...</p>}>
            <Await
              resolve={data.chatResponse}
              errorElement={<p>Error loading chats!</p>}
            >
              {(chatResponse) => <Chat chats={chatResponse.data} />}
            </Await>
          </Suspense>
        </div>
      </div>
    </div>
  );
}

export default ProfilePage;
```

This code defines the Profile Page UI of the real estate application, presenting user information alongside personal property listings and integrated chat functionality. This displays user details such as avatar, username, and email, with buttons for updating the profile, logging out, or accessing the admin panel (for admin users). Below that, the user's own property posts are listed, with the option to create new ones. The right panel includes a live chat section that loads conversations using. The layout is cleanly divided for easy navigation, supporting both user management and real-time communication from a single interface.

3. SinglePage

singlePage.jsx

CSE471_Project > client > src > routes > singlePage > singlePage.jsx > SinglePage

```jsx
import DOMPurify from "dompurify";
import { useContext, useEffect, useState } from "react";
import { useLoaderData, useNavigate } from "react-router-dom";
import Map from "../../components/map/map";
import Slider from "../../components/slider/Slider";
import { AuthContext } from "../../context/AuthContext";
import "react-datepicker/dist/react-datepicker.css";
import "./singlePage.scss";
import Calendar from "../../components/calendar/calendar";



function SinglePage() {
  const post = useLoaderData();
  const { currentUser } = useContext(AuthContext);
  const navigate = useNavigate();

  return (
    <div className="singlePage">
      <div className="details">
        <div className="wrapper">
          <Slider images={post.images} />
          <div className="info">
            <div className="top">
              <div className="post">
                <h1>{post.title}</h1>
                <div className="address">
                  <img src="/pin.png" alt="" />
                  <span>{post.address}</span>
```

singlePage.jsx

CSE471_Project > client > src > routes > singlePage > singlePage.jsx > SinglePage

```jsx
function SinglePage() {
                  <span>{post.address}</span>
                </div>
                <div className="price">
                  BDT {post.price}
                </div>
              </div>
              <div className="user">
                <img src={post.user.avatar} alt="" />
                <span>{post.user.username}</span>
              </div>
            </div>
            <div
              className="bottom"
              dangerouslySetInnerHTML={{
                __html: DOMPurify.sanitize(post.postDetail.desc),
              }}
            ></div>
          </div>
        </div>
      </div>
      <div className="features">
        <div className="wrapper">
          <p className="title">General</p>
          <div className="listVertical">
            <div className="feature">
              <img src="/utility.png" alt="" />
              <div className="featureText">
                <span>Utilities</span>
                {post.postDetail.utilities === "owner" ? (
                  <p>Owner is responsible</p>
                ) : (
                  <p>Tenant is responsible</p>
                )}
              </div>
            </div>
            <div className="feature">
              <img src="/pet.png" alt="" />
              <div className="featureText">
                <span>Pet Policy</span>
                {post.postDetail.pet === "allowed" ? (
                  <p>Pets Allowed</p>
                ) : (
                  <p>Pets not Allowed</p>
                )}
```

singlePage.jsx

CSE471_Project > client > src > routes > singlePage > singlePage.jsx > SinglePage

```jsx
function SinglePage() {
                )}
              </div>
            </div>
            <div className="feature">
              <img src="/fee.png" alt="" />
              <div className="featureText">
                <span>Income Policy</span>
                <p>{post.postDetail.income}</p>
              </div>
            </div>
          </div>
          <p className="title">Sizes</p>
          <div className="sizes">
            <div className="size">
              <img src="/size.png" alt="" />
              <span>{post.postDetail.size} sqft</span>
            </div>
            <div className="size">
              <img src="/bed.png" alt="" />
              <span>{post.bedroom} beds</span>
            </div>
            <div className="size">
              <img src="/bath.png" alt="" />
              <span>{post.bathroom} bathroom</span>
            </div>
          </div>
          <p className="title">Nearby Places</p>
          <div className="listHorizontal">
            <div className="feature">
              <img src="/school.png" alt="" />
              <div className="featureText">
                <span>School</span>
                <p>
                  {post.postDetail.school > 999
                    ? post.postDetail.school / 1000 + "km"
                    : post.postDetail.school + "m"}{" "}
                  away
                </p>
              </div>
            </div>
            <div className="feature">
              <img src="/pet.png" alt="" />
              <div className="featureText">
                <span>Bus Stop</span>
```

```jsx
              </div>
            </div>
            <div className="feature">
              <img src="/pet.png" alt="" />
              <div className="featureText">
                <span>Bus Stop</span>
                <p>{post.postDetail.bus}m away</p>
              </div>
            </div>
            <div className="feature">
              <img src="/fee.png" alt="" />
              <div className="featureText">
                <span>Restaurant</span>
                <p>{post.postDetail.restaurant}m away</p>
              </div>
            </div>
          </div>
          <p className="title">Book your slot</p>
          <div className = "date-range-calender">
            {currentUser?.id !== post.userId &&
            <Calendar
            propertyId={post.id}  // Pass post.id as propertyId to the Calendar component
            userId={currentUser.id}  // Pass current user for reservation logic
          />}
          </div>
          <p className="title">Location</p>
          <div className="mapContainer">
            <Map items={[post]} />
          </div>
          <span></span>
        </div>
      </div>
    </div>
  );
}

export default SinglePage;
```

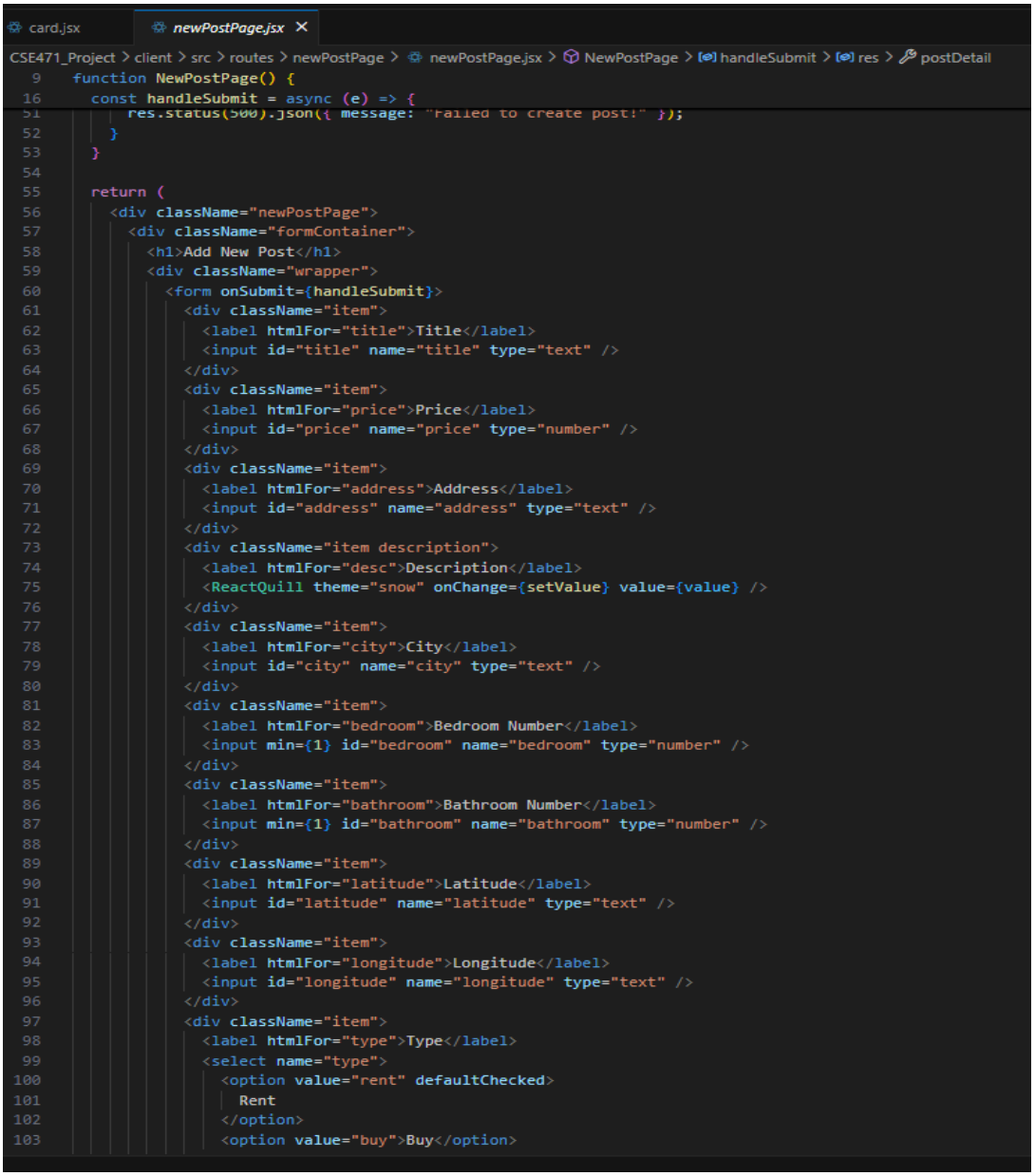
This code defines the Single Property Page UI, displaying detailed information about a selected property. It includes an image slider, property title, address, price, and the owner's user profile. Key property features like utilities, pet policy, income requirements, size, and proximity to nearby places (schools, restaurants) are shown. It also integrates a calendar component for booking appointments (visible only to users other than the property owner) and a map showing the property's location. This helps the user to know all the information about the property.

4. Property Creation

card.jsx | newPostPage.jsx

CSE471_Project > client > src > routes > newPostPage > newPostPage.jsx > NewPostPage > handleSubmit > res

```jsx
import "./newPostPage.scss";
import ReactQuill from "react-quill";
import "react-quill/dist/quill.snow.css";
import { useState } from "react";
import { useNavigate } from "react-router-dom";
import apiRequest from "../../lib/apiRequest";
import UploadWidget from "../../components/uploadWidget/UploadWidget";

function NewPostPage() {
  const [value, setValue] = useState("");
  const [error, setError] = useState("");
  const [images, setImages] = useState([]);

  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    const formData = new FormData(e.target);
    const inputs = Object.fromEntries(formData);

    try {
      const res = await apiRequest.post("/prop/", {
        postData: {
          title: inputs.title,
          price: parseInt(inputs.price),
          address: inputs.address,
          city: inputs.city,
          bedroom: parseInt(inputs.bedroom),
          bathroom: parseInt(inputs.bathroom),
          type: inputs.type,
          property: inputs.property,
          latitude: inputs.latitude,
          longitude: inputs.longitude,
          images: images,
        },
        postDetail: {
          desc: value,
          utilities: inputs.utilities,
          pet: inputs.pet,
          income: inputs.income,
          size: parseInt(inputs.size),
          school: parseInt(inputs.school),
          bus: parseInt(inputs.bus),
          restaurant: parseInt(inputs.restaurant),
        },
      });
      navigate("/" + res.data.id);
    } catch (err) {
      console.log(err);
      setError(err);
      res.status(500).json({ message: "Failed to create post!" });
    }
  }
```


card.jsx
newPostPage.jsx
CSE471_Project > client > src > routes > newPostPage > newPostPage.jsx > NewPostPage > handleSubmit > res > postDetail
function NewPostPage() {
const handleSubmit = async (e) => {
res.status(500).json({ message: "Failed to create post!" });
}
}
return (

<h1>Add New Post</h1>

<form onSubmit={handleSubmit}>

<label htmlFor="title">Title</label>
<input id="title" name="title" type="text" />

<label htmlFor="price">Price</label>
<input id="price" name="price" type="number" />

<label htmlFor="address">Address</label>
<input id="address" name="address" type="text" />

<label htmlFor="desc">Description</label>
<ReactQuill theme="snow" onChange={setValue} value={value} />

<label htmlFor="city">City</label>
<input id="city" name="city" type="text" />

<label htmlFor="bedroom">Bedroom Number</label>
<input min={1} id="bedroom" name="bedroom" type="number" />

<label htmlFor="bathroom">Bathroom Number</label>
<input min={1} id="bathroom" name="bathroom" type="number" />

<label htmlFor="latitude">Latitude</label>
<input id="latitude" name="latitude" type="text" />

<label htmlFor="longitude">Longitude</label>
<input id="longitude" name="longitude" type="text" />

<label htmlFor="type">Type</label>
<select name="type">
<option value="rent" defaultChecked>
Rent
</option>
<option value="buy">Buy</option>

card.jsx newPostPage.jsx

CSE471_Project > client > src > routes > newPostPage > newPostPage.jsx > NewPostPage > handleSubmit > res > postD

```jsx
function NewPostPage() {
            </div>
            <div className="item">
              <label htmlFor="type">Property</label>
              <select name="property">
                <option value="apartment">Apartment</option>
                <option value="house">House</option>
                <option value="condo">Condo</option>
                <option value="land">Land</option>
              </select>
            </div>

            <div className="item">
              <label htmlFor="utilities">Utilities Policy</label>
              <select name="utilities">
                <option value="owner">Owner is responsible</option>
                <option value="tenant">Tenant is responsible</option>
                <option value="shared">Shared</option>
              </select>
            </div>
            <div className="item">
              <label htmlFor="pet">Pet Policy</label>
              <select name="pet">
                <option value="allowed">Allowed</option>
                <option value="not-allowed">Not Allowed</option>
              </select>
            </div>
            <div className="item">
              <label htmlFor="income">Income Policy</label>
              <input
                id="income"
                name="income"
                type="text"
                placeholder="Income Policy"
              />
            </div>
            <div className="item">
              <label htmlFor="size">Total Size (sqft)</label>
              <input min={0} id="size" name="size" type="number" />
            </div>
            <div className="item">
              <label htmlFor="school">School</label>
              <input min={0} id="school" name="school" type="number" />
            </div>
            <div className="item">
              <label htmlFor="bus">bus</label>
              <input min={0} id="bus" name="bus" type="number" />
            </div>
            <div className="item">
              <label htmlFor="restaurant">Restaurant</label>
              <input min={0} id="restaurant" name="restaurant" type="number" />
            </div>
            <button className="sendButton">Add</button>
          </form>
        </div>
```

```jsx
            </div>
            <button className="sendButton">Add</button>
          </form>
        </div>
      </div>
      <div className="sideContainer">
        <div className="imageContainer">
          {images.map((image, index) => (
            <img src={image} key={index} alt="" />
          ))}
        </div>
        <UploadWidget
          className="uploadButton"
          uwConfig={{
            cloudName: "farhanjoy",
            uploadPreset: "estate",
            multiple: true,
            folder: "property",
          }}
          setState={setImages}
        />
      </div>
    </div>
  );
}

export default NewPostPage;
```

This code defines the New Post Creation Page UI for a real estate application, allowing users to create a property through a detailed form. Users can input property details such as title, price, address, city, type, property category, and location coordinates. The form also have additional specifications like bedrooms, bathrooms, size, utilities, pet policy, and nearby places. A text editor (ReactQuill) is used for writing property descriptions, and an image upload feature (via Cloudinary UploadWidget) enables users to attach multiple property photos. Upon creation, the listing is sent to the backend. The UI is intuitive and user-friendly, making it easy for both new and returning users to create posts seamlessly.

5. Property Listing

card.jsx | listPage.jsx

CSE471_Project > client > src > routes > listPage > listPage.jsx > ListPage

```jsx
//import { listData } from '../../lib/dummydata'
import { Suspense } from "react";
import { Await, useLoaderData } from "react-router-dom";
import './listPage.scss'
import Filter from '../../components/filter/Filter'
import Map from '../../components/map/map'
import Card from "../../components/card/card"

function ListPage() {
    const data = useLoaderData();

    return (
      <div className="listPage">
        <div className="listContainer">
          <div className="wrapper">
            <Filter />
            <Suspense fallback={<p>Loading...</p>}>
              <Await
                resolve={data.postResponse}
                errorElement={<p>Error loading posts!</p>}
              >
                {(postResponse) =>
                  postResponse.data.map((post) => (
                    <Card key={post.id} item={post} />
                  ))
                }
              </Await>
            </Suspense>
          </div>
        </div>
        <div className="mapContainer">
          <Suspense fallback={<p>Loading...</p>}>
            <Await
              resolve={data.postResponse}
              errorElement={<p>Error loading posts!</p>}
            >
              {(postResponse) => <Map items={postResponse.data} />}
            </Await>
          </Suspense>
        </div>
      </div>
    );
  }
```

card.jsx | listPage.jsx

CSE471_Project > client > src > components > card > card.jsx > ...

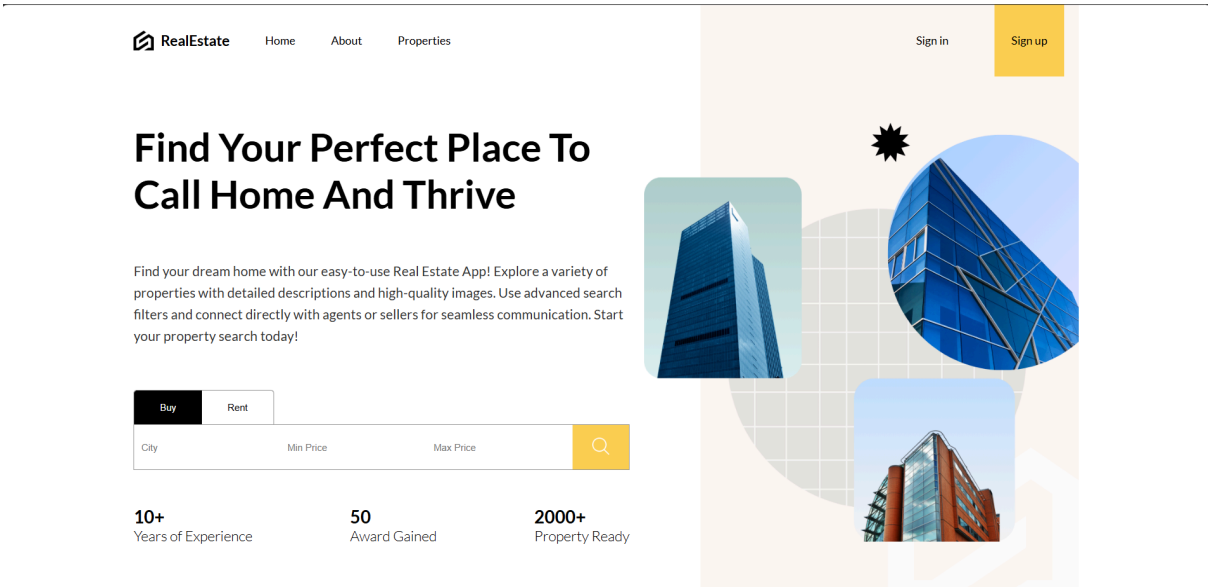
```jsx
import apiRequest from "../../lib/apiRequest";
import { useContext } from "react";
import { Link } from "react-router-dom";
import { AuthContext } from "../../context/AuthContext";
import { useNavigate } from "react-router-dom";
import axios from "axios";
import "./card.scss";

function Card({ item}) {
  const { currentUser } = useContext(AuthContext);
  const navigate = useNavigate();

  const handlePostDelete = async (id) => {
    try {
      const response = await axios.delete(`/api/prop/${id}`);
      if (response.status === 200) {
        console.log("successfully deleted post");
        window.location.reload();
      } else {
        console.error("Failed to delete post");
      }
    } catch (error) {
      console.error("Failed to delete post", error);
    }
  };

  const handleChatClick = async () => {
    try {
      const res = await apiRequest.post("/chats", {
        receiverId: item.userId,
      });

      const chatId = res.data?.id;
      if (!chatId) {
        console.error("chatId not returned from server");
        return;
      }

      await apiRequest.post(`/messages/${chatId}`, {
        text: `I am Interested to buy ${item.title}`,
      });
      navigate("/profile");

    } catch (error) {
      console.error("Failed to initiate chat:", error);
    }
  };

```

The ListPage component is responsible for displaying all property listings. It includes a filter section on the left and a map on the right, providing a responsive and interactive layout. Listings are loaded asynchronously using Suspense and Await, ensuring smooth data handling. Each listing is rendered using the Card component, which displays essential property information such as image, title, location, price, number of bedrooms and bathrooms. The card also includes interactive icons: users can initiate a chat with the property

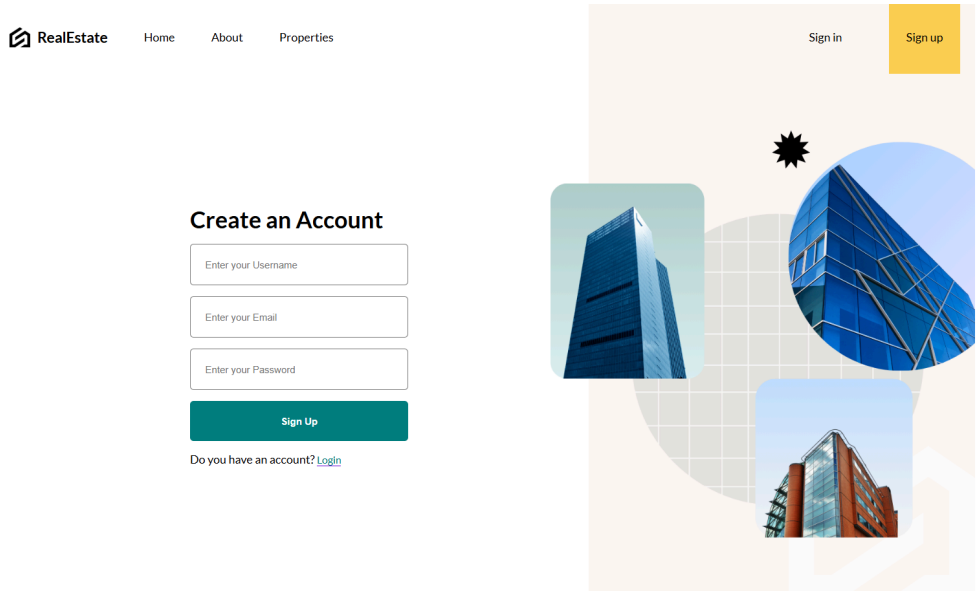
owner, view details, or, if they are the owner or admin, delete the post. The design adapts based on the current user's role, enhancing usability and control.

7. User Manual

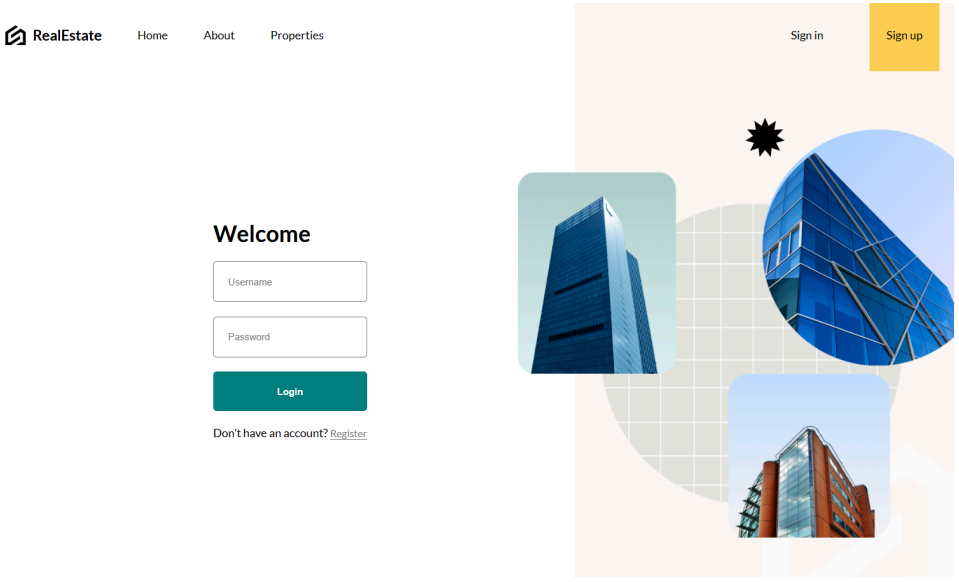
1. When users enter the app, they are directed to the Home page. However, they must be registered and logged in to view property listings or access other parts of the website. Clicking "Sign In" or "Sign Up" takes them to the login or registration page.



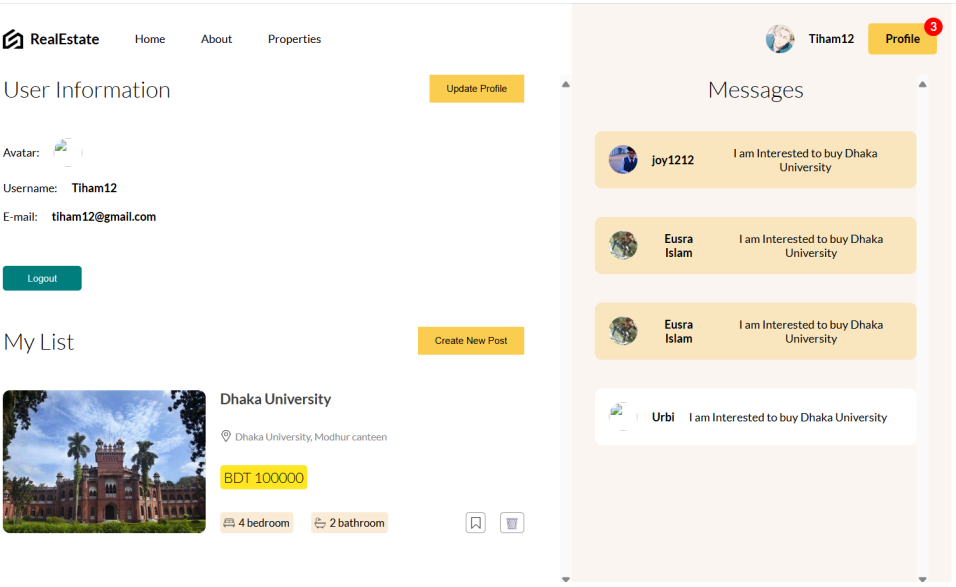
2. On the Sign Up page, users can enter their details to create an account. After clicking the "Sign Up" button, they are redirected to the Login page.



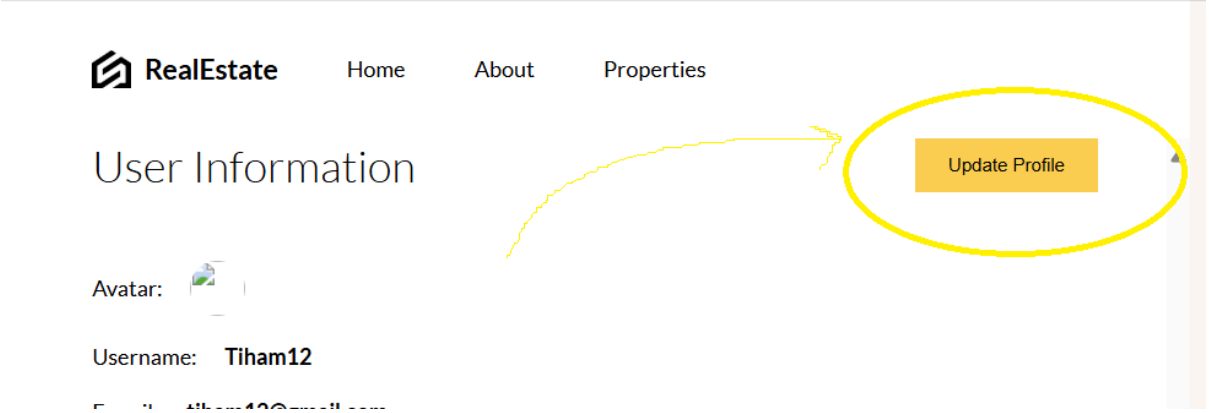
3. On the Login page, users can enter their credentials, and after clicking the "Login" button, they are navigated to the User Profile page.



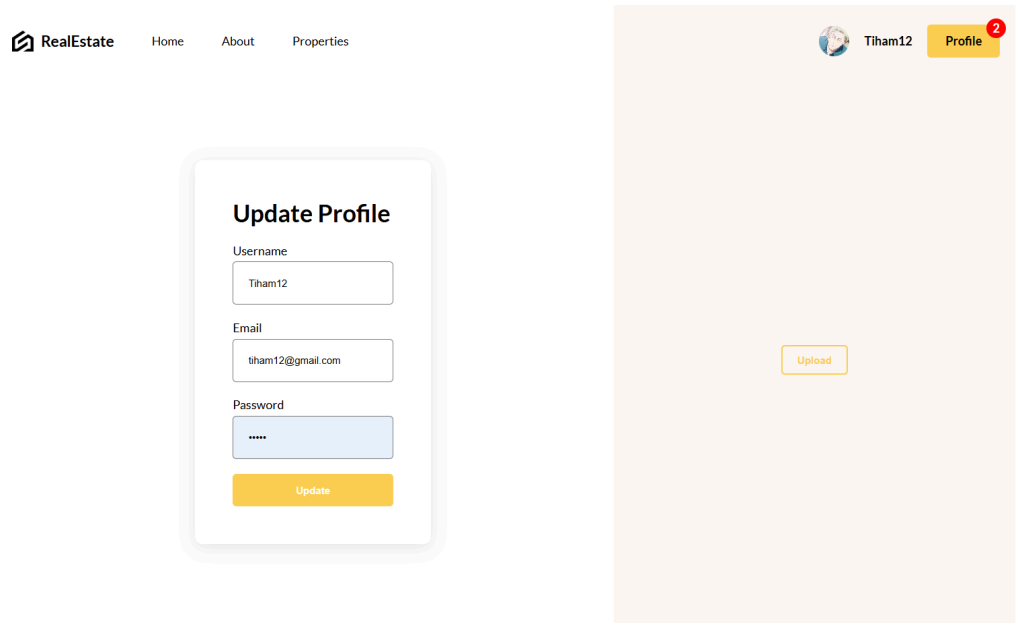
4. On the User Profile page, the user's information is displayed. Users can update their profile details, create property listings, message potential buyers or tenants, and also contact other property owners for buying or renting purposes.



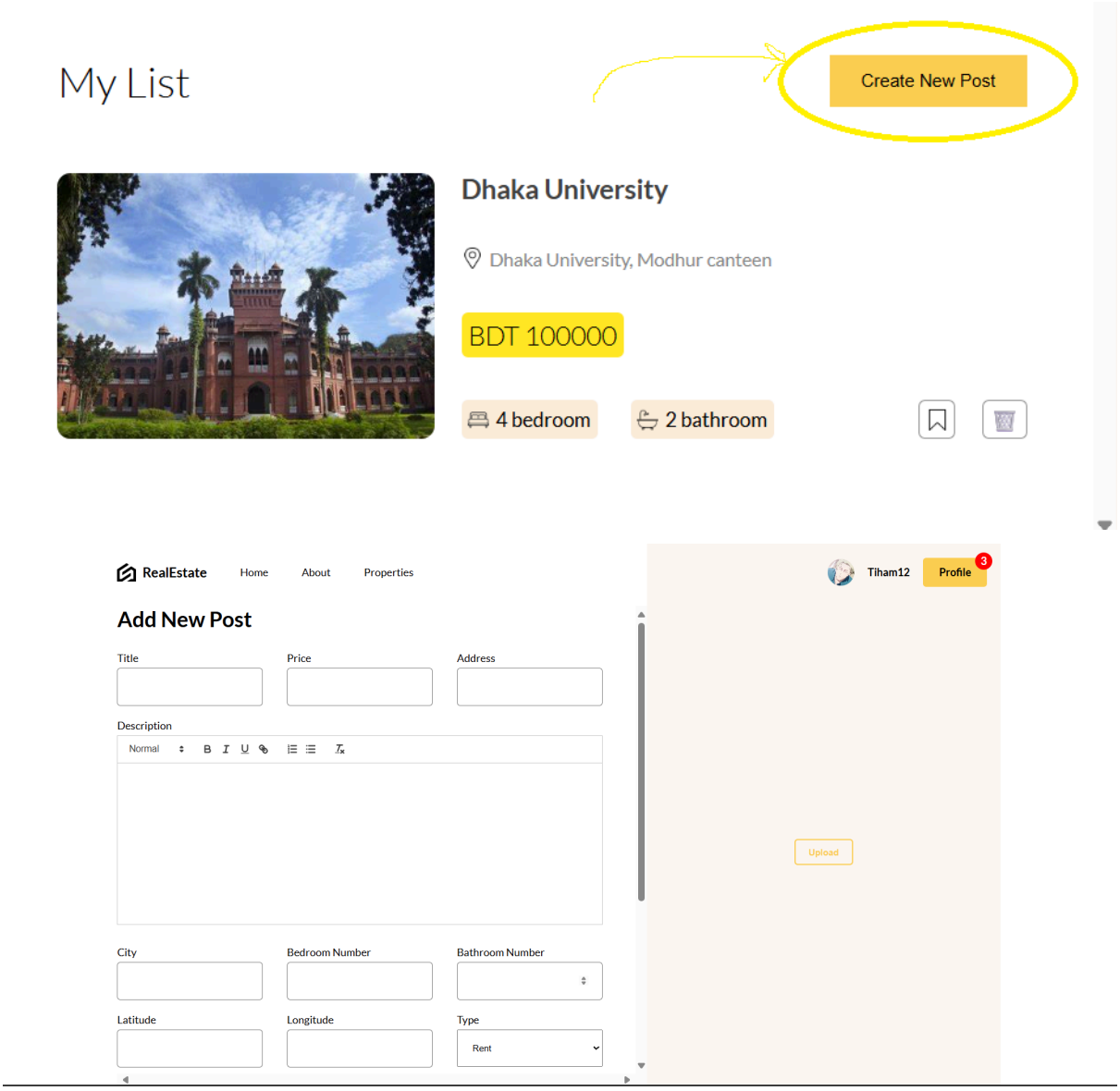
5. Clicking on the “Update Profile” button redirects the user to a form where they can update their profile information.

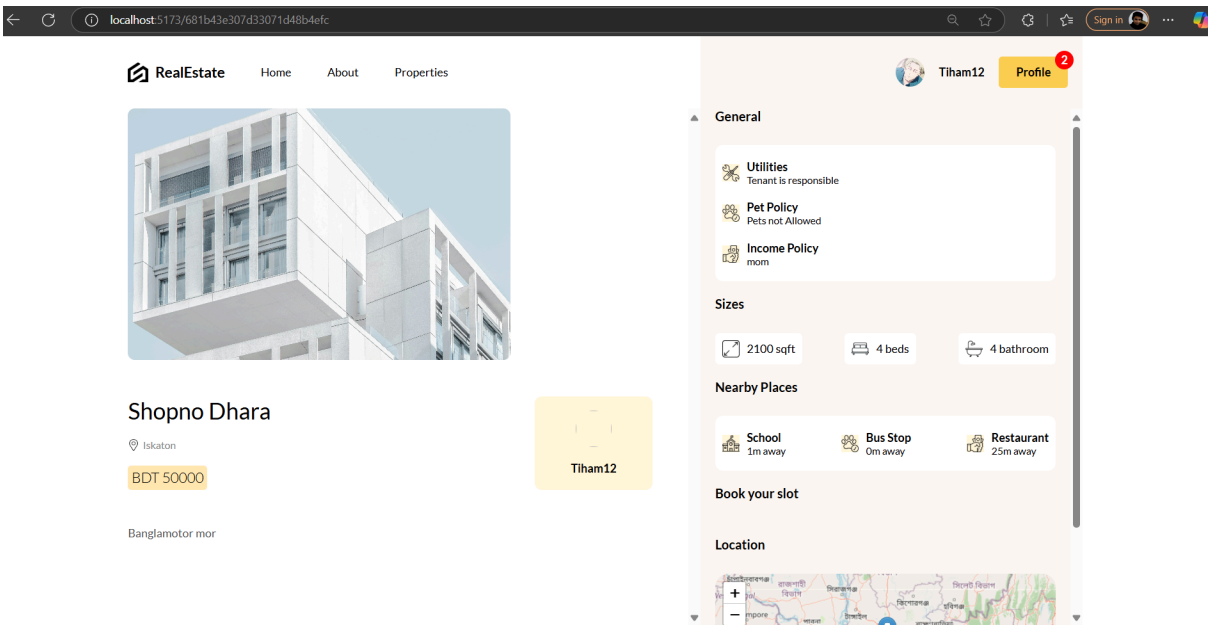


A user can change their picture and other information through this form and clicking on “update” will change the previous credentials.

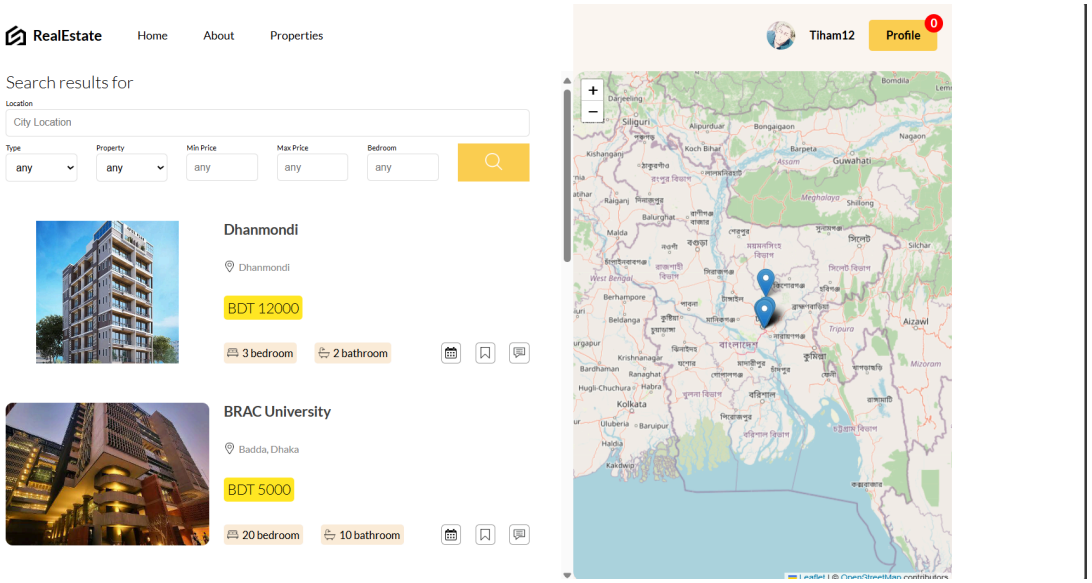


6. Clicking on “Create New Post” takes the user to a form where they can create a property listing. Here, they can enter details like the title, city, price, number of bedrooms, map coordinates, and more. Users can also upload multiple images of the property. After writing all the information, clicking on “Add” will add the property to the listings and redirect the page to the Property information page.

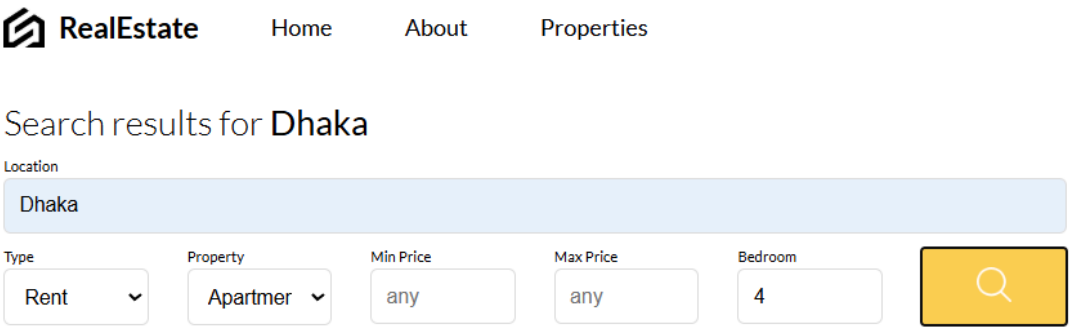




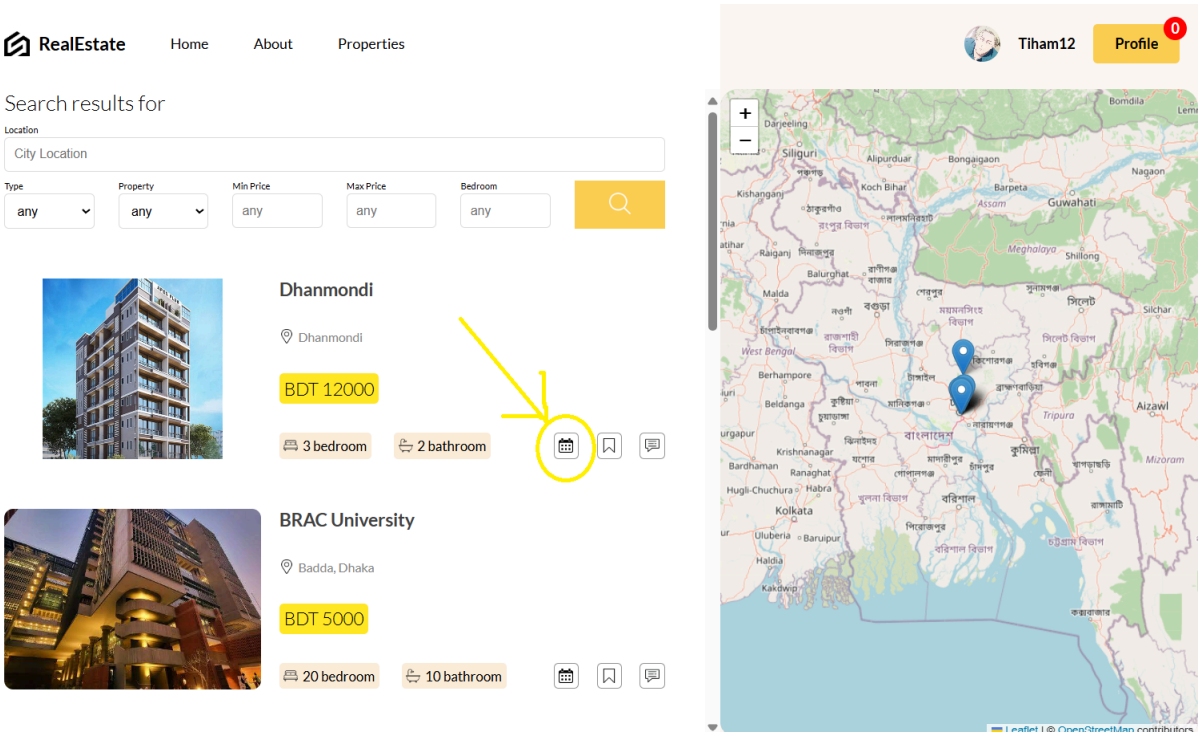
7. Tapping on the “Properties” section will navigate to the property listing page, where all properties on the website are shown.

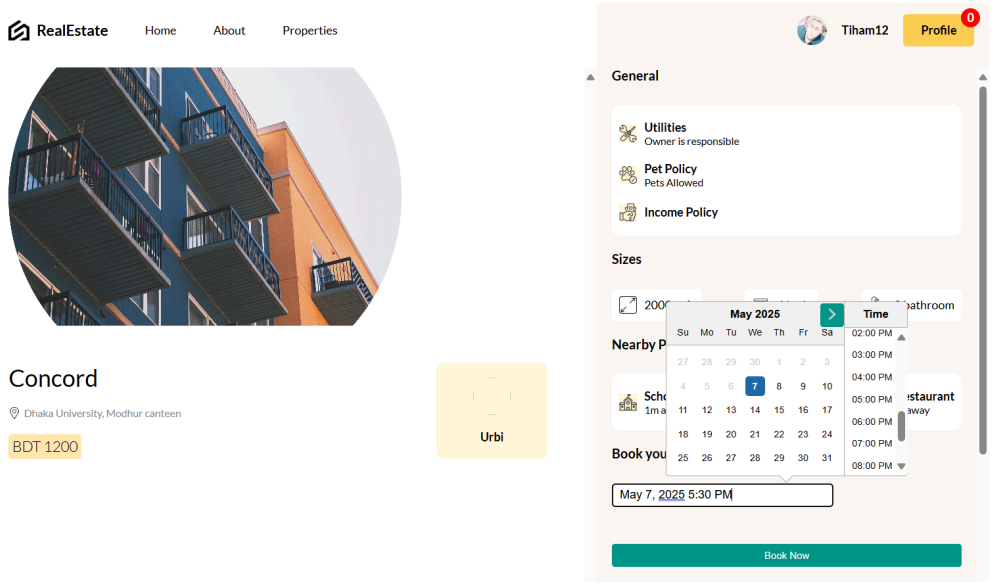


8. Users can search for properties by entering a location and applying various filters based on their preferences. For example, they can set a price range to view only properties within that budget. Additionally, users can specify whether they are looking to rent or buy, and choose the type of property they are interested in, such as an apartment, condo, or land.

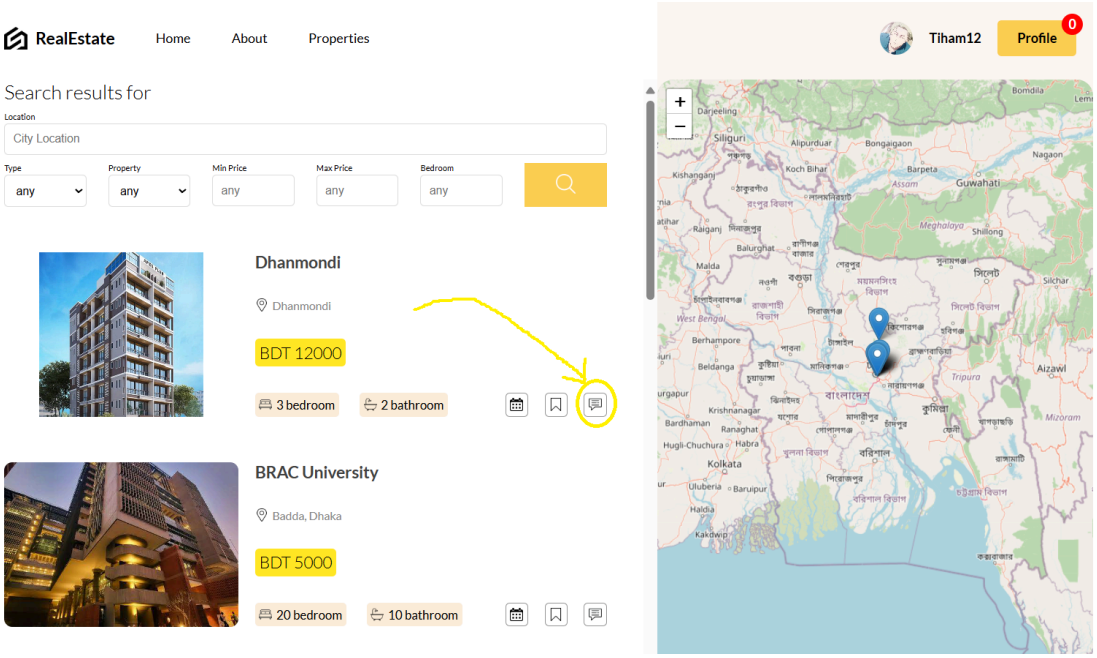


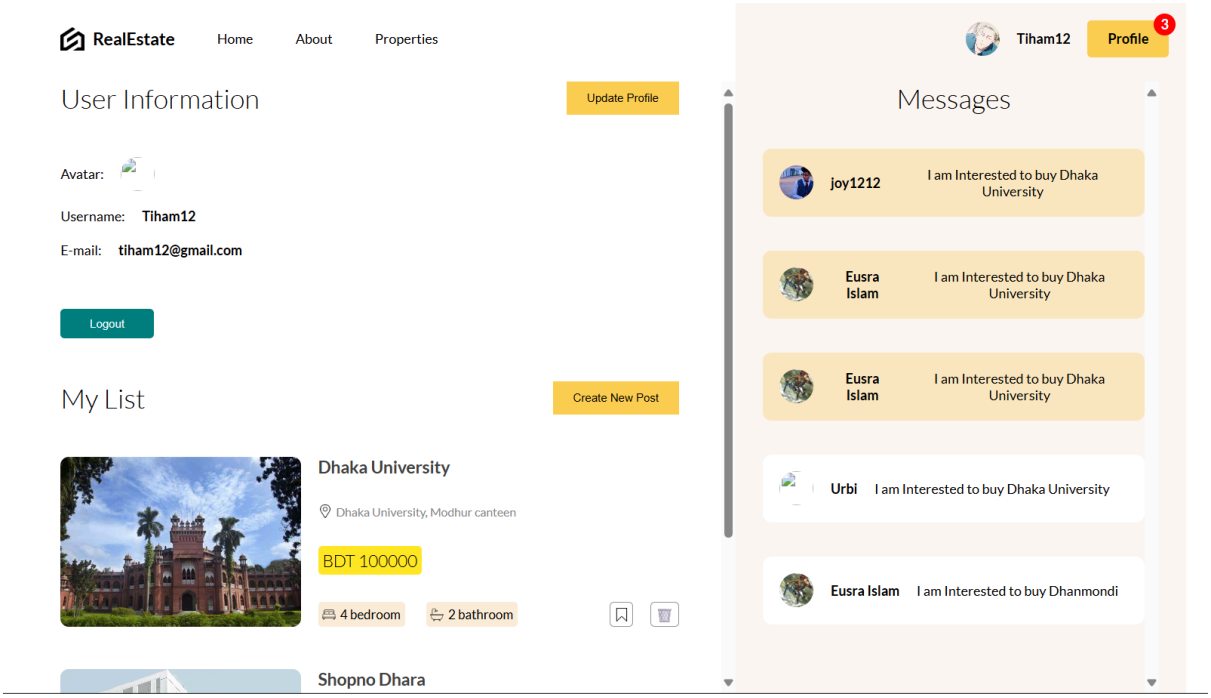
9. If a user is interested in viewing an apartment, they can reserve a time slot. By clicking the calendar icon, the user will be redirected to the property information page, where they can select their preferred date and time for the visit. Each slot can only be booked once.



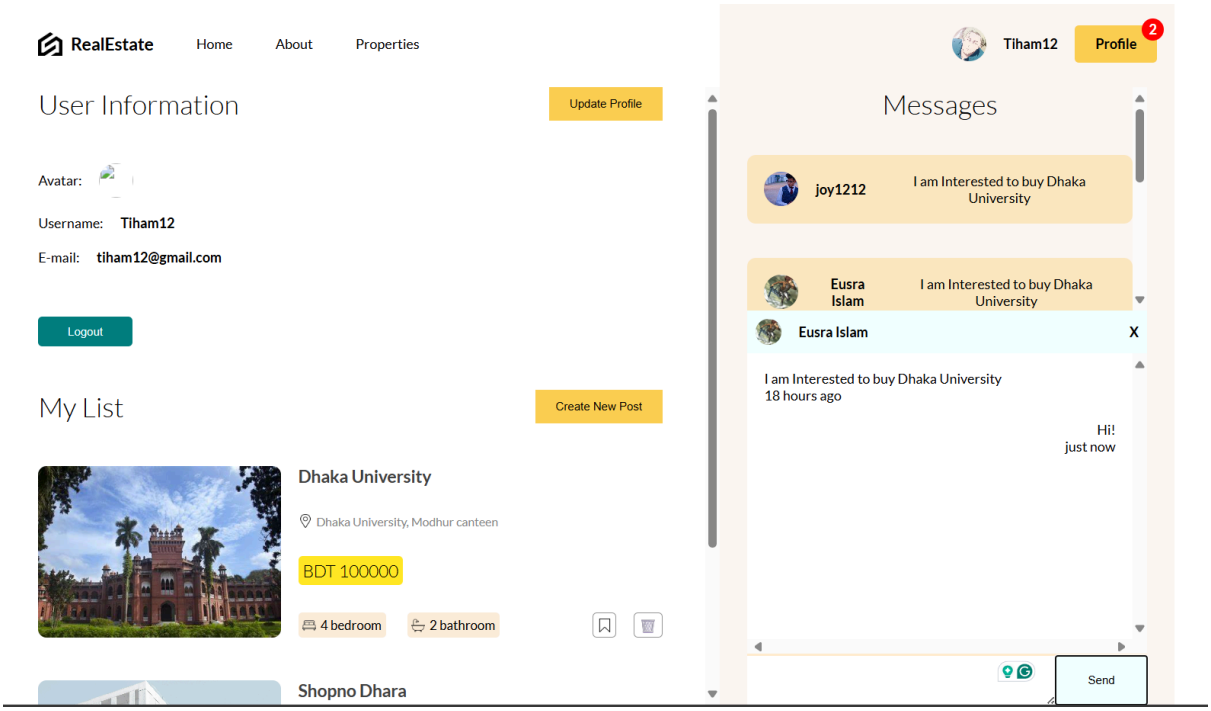


10. Users can also communicate directly with the property owner if they are interested in buying or renting a property. By clicking the message icon, they will be redirected to the profile page, where the chatbox is located. Unread messages are highlighted in yellow, while read messages appear in white. The number of unread messages is displayed in the top corner of the profile, and this count updates automatically as messages are read.



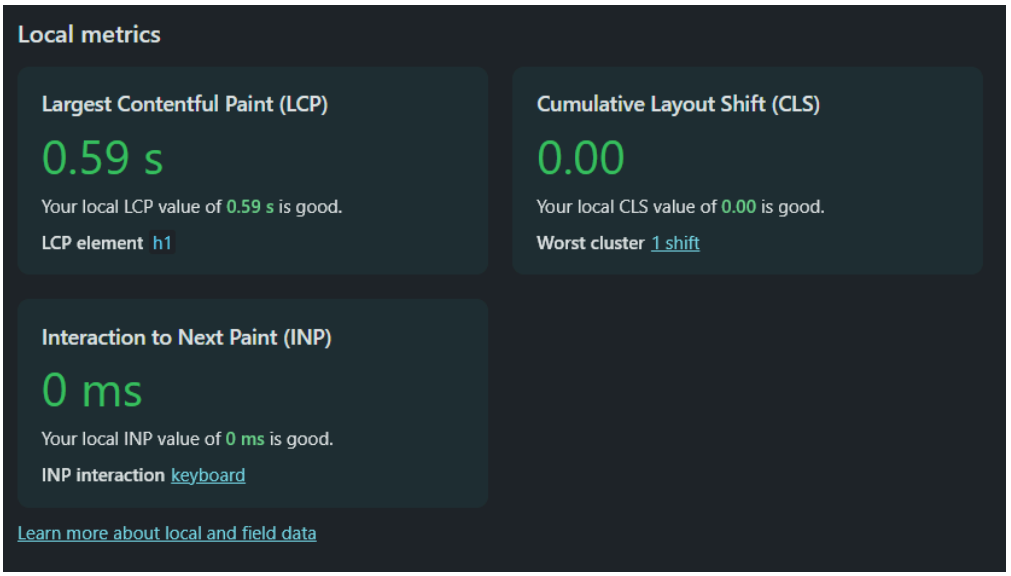


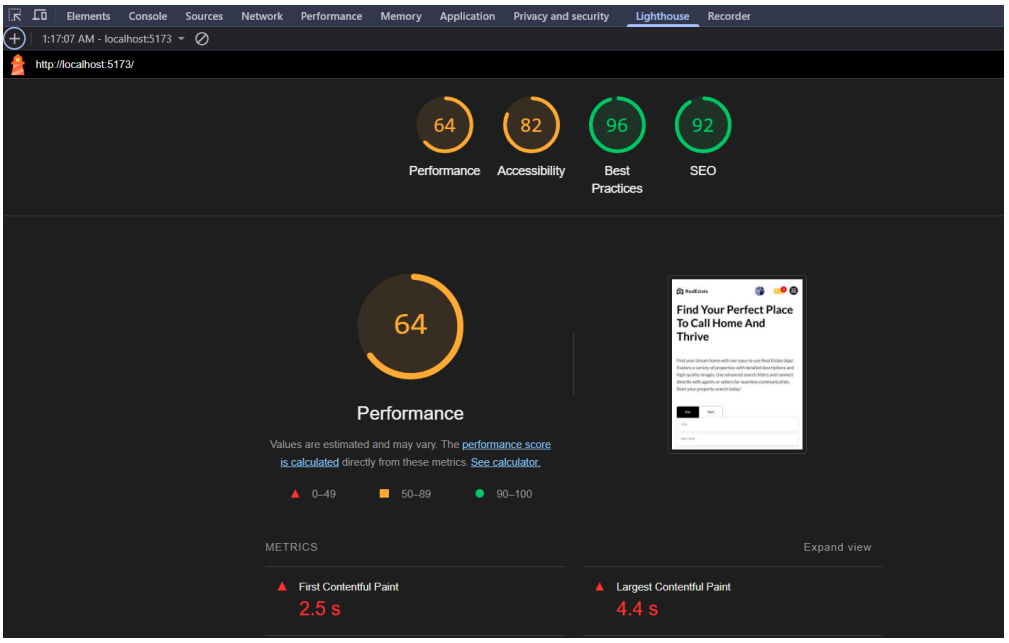
Here, after reading a message, the notification is changed from 3 to 2.

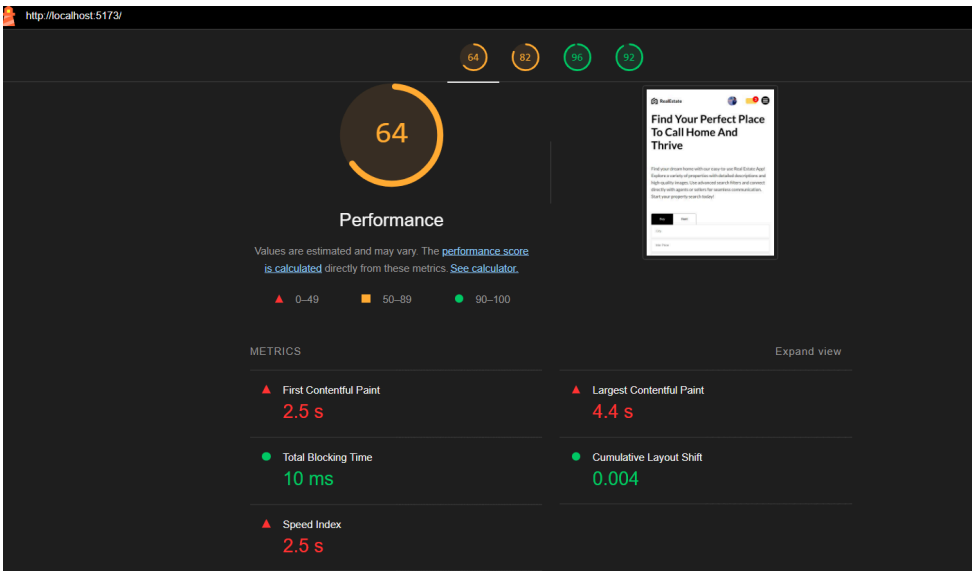


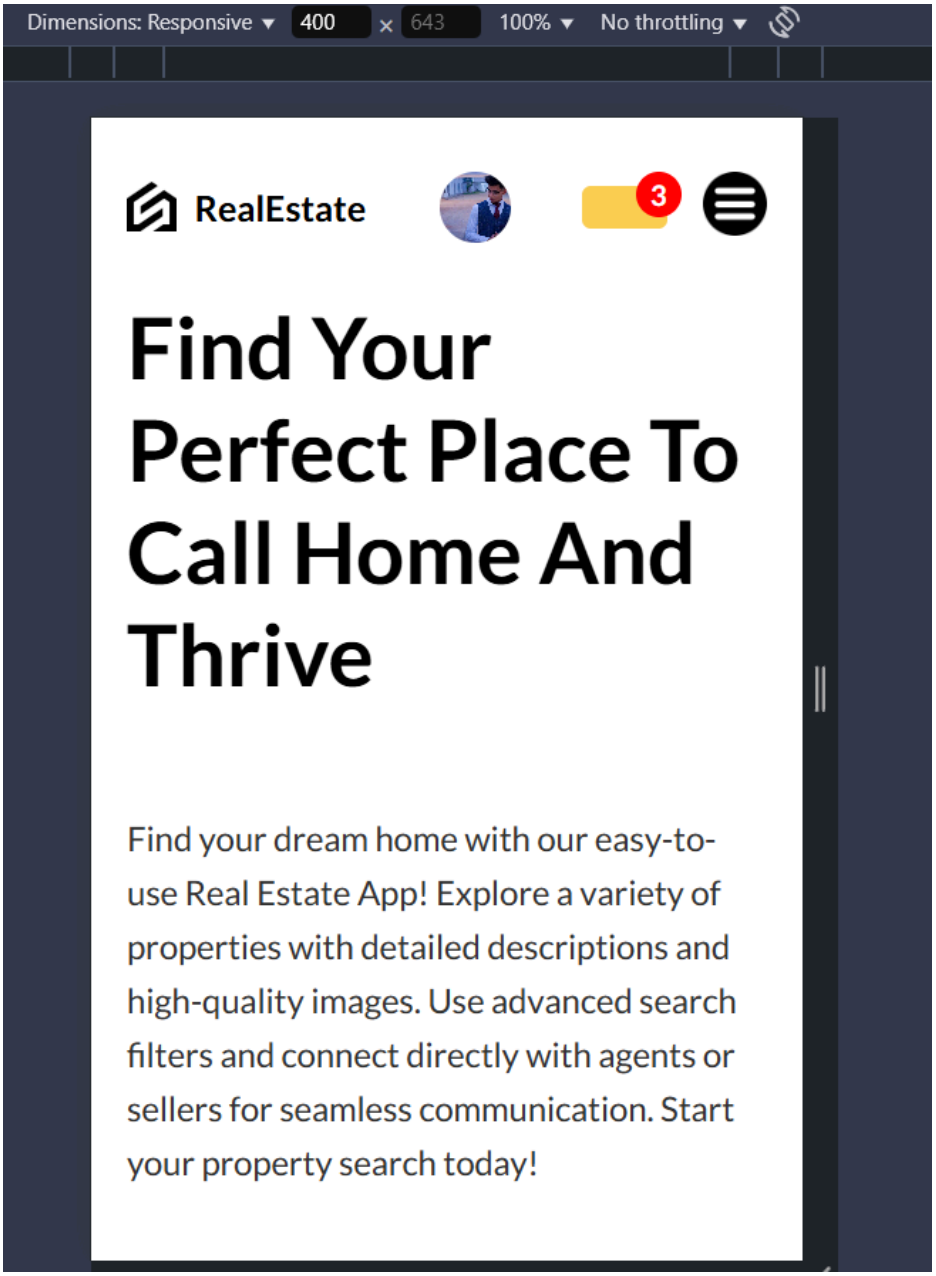
8. Performance and Network Analysis

Performance and Network analysis report using lighthouse DevTool. Add screenshots of the report.






http://localhost:5173/
64
82
96
92
64
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
0–49
50–89
90–100
METRICS
Expand view
First Contentful Paint
2.5 s
Largest Contentful Paint
4.4 s
Total Blocking Time
10 ms
Cumulative Layout Shift
0.004
Speed Index
2.5 s
Find Your Perfect Place To Call Home And Thrive


Dimensions: Responsive
400
643
100%
No throttling
RealEstate
3
Find Your Perfect Place To Call Home And Thrive
Find your dream home with our easy-to-use Real Estate App! Explore a variety of properties with detailed descriptions and high-quality images. Use advanced search filters and connect directly with agents or sellers for seamless communication. Start your property search today!

9. Github Repo [Public] Link

https://github.com/FarhanAbedin12/CSE471_Project

10. Link of Deployed Project

https://cse-471-project-oagw.vercel.app/

This project has been developed using the MERN stack. Hence, it can not be deployed using Vercel, Netlify, or any other free hosting service as the API can not be connected. The best approach to deploy this particular project is by containerizing the services and deploying to cloud services such as AWS.

11. Individual Contribution

Group member - 01	
Name: Farhan Abedin	Student ID: 21301449
Functional Requirements which are developed by this member:	
1. Login and Register as a user	
2. Filtering and Upload post	
3. Admin panel and the task of the admin	
4. Chat system integration	

Group member - 02	
Name: Eusra Ibnat	Student ID: 21201147
Functional Requirements which are developed by this member:	
1. Property Listings	
2. Map integration	
3. Slot Reservation	
4. Notification of chats	

Group member - 03	
Name: MD Waliul Islam	Student ID: 20101184
Functional Requirements which are developed by this member:	
1. Search bar	
2. Save Property	
3. Check Availability Status	
4. Payment integration	

References: