



**CSE471: System Analysis and Design**

**Project Report**

**Project Title:** Project Title Here

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

**Submission Date: 07. 05. 2025**

## **Table of Contents**

1. System Request	3
Business need:	3
Business requirements:	3
Business value:	3
Special issues or constraints:	3
2. Functional Requirements	4
3. Technology (Framework, Languages)	5
4. Backend Development	5
5. User Interface Design	10
6. Frontend Development	14
7. User Manual	25
8. Performance and Network Analysis	33
9. Github Repo [Public] Link	35
10. Link of Deployed Project	35
11. Individual Contribution	35
12. References	36

# 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)

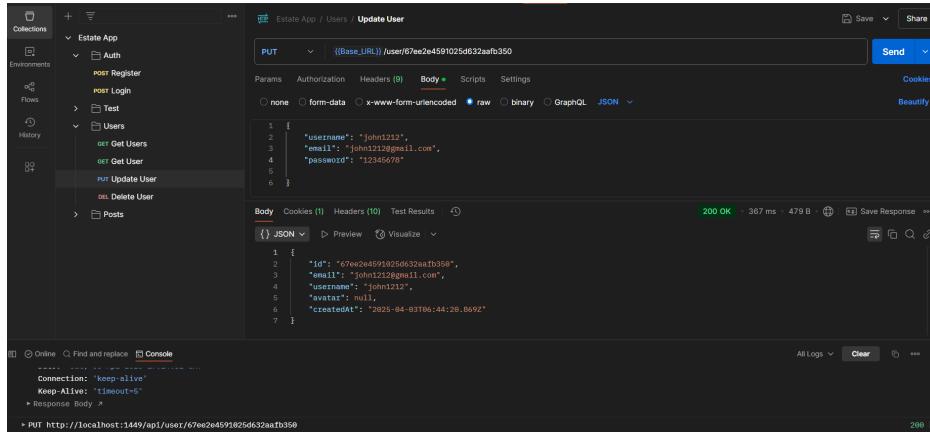
Javascript, MERN stack(MongoDB, ExpressJs, ReactJs, NodeJs)

## 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

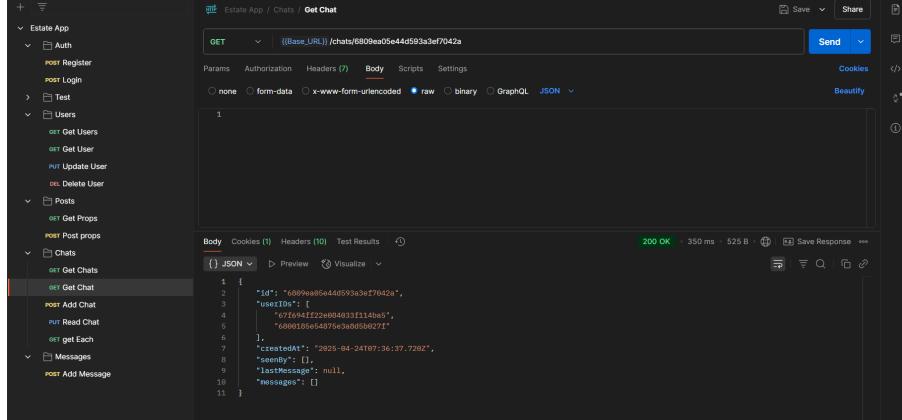
```

36  export const getChat = async (req, res) => {
37    const tokenUserId = req.userId;
38
39    try {
40      const chat = await prisma.chat.findUnique({
41        where: {
42          id: req.params.id,
43          userIDs: {
44            hasSome: [tokenUserId],
45          },
46        },
47        include: {
48          messages: {
49            orderBy: {
50              createdAt: "asc",
51            },
52          },
53        },
54      });
55
56      await prisma.chat.update({
57        where: {
58          id: req.params.id,
59        },
60        data: {
61          seenBy: {
62            push: [tokenUserId],
63          },
64        },
65      });
66      res.status(200).json(chat);
67    } catch (err) {
68      console.log(err); if (PrismaClientKnownRequestError { name: 'PrismaClientKnownRe
69      res.status(500).json({ message: "Failed to get chat!" });
70    }
71  };

```

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.

The screenshot shows the Postman application interface. On the left, the 'Collections' sidebar lists 'Estate App' with sub-folders 'Auth', 'Test', 'Users', and 'Posts'. Under 'Posts', there are two items: 'GET Get Props' and 'POST Post props'. The main workspace shows a 'Get Props' request with the URL `((Base_URL))/prop/677ec11888e32b3f4e149e91`. The 'Body' tab displays a JSON response for a single property listing:

```

1
2   "id": "677ec11888e32b3f4e149e91",
3   "title": "Amar Barri",
4   "price": 200000,
5   "images": [
6     "https://www.gbdarchitects.com/wp-content/uploads/2013/09/Kiln-Apartments-1.jpg",
7     "https://www.gbdarchitects.com/wp-content/uploads/2013/09/Kiln-Apartments-1.jpg",
8     "https://www.gbdarchitects.com/wp-content/uploads/2013/09/Kiln-Apartments-1.jpg"
9   ],
10  "address": "AddressIII",
11  "city": "CityI",
12  "bedroom": 11,
13  "bathroom": 11,
14  "latitude": "51.58",
15  "longitude": "-0.1275",
16  "type": "rent",
17  "property": "apartment"

```

The status bar at the bottom indicates a 200 OK response with 102 ms and 1.11 KB.

#### 4. Fetching all posts

```

CSE471_Project > api > controllers > 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
26

```

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.

The screenshot shows a Postman interface with a GET request to `http://localhost:8800/api/prop/`. The response is a 200 OK status with 53 ms latency and 3.18 KB size. The JSON body contains two property objects:

```

1 [
2   {
3     "id": "6000204d957baaf13de49327",
4     "title": "Diamondi",
5     "price": 12000,
6     "images": [
7       "https://res.cloudinary.com/farhanjoy/image/upload/v174483803/property/yn09kxndnpxzbxg2sf6eo.jpg"
8     ],
9     "address": "Diamondi",
10    "city": "Dhaka",
11    "bedroom": 3,
12    "bathroom": 2,
13    "latitude": "23.746466",
14    "longitude": "90.376015",
15    "type": "rent",
16    "property": "apartment",
17    "createdAt": "2025-04-15T21:25:33.940Z",
18    "userId": "6000105e54875e3a0d5b027f"
19  },
20  {
21    "id": "6000bf6e70ecbb12b0dd223",
22    "title": "BRAC University",
23    "price": 6000,
24    "images": [
25      "https://res.cloudinary.com/farhanjoy/image/upload/v1744879464/property/bcpgafiyehnopgyxgo.jpg",
26      "https://res.cloudinary.com/farhanjoy/image/upload/v1744879464/property/xwgxejiglbraqidyeexn.jpg"
27    ],
28    "address": "Badda, Dhaka",
29    "city": "Dhaka",
30  }
]

```

## 5. Delete a user

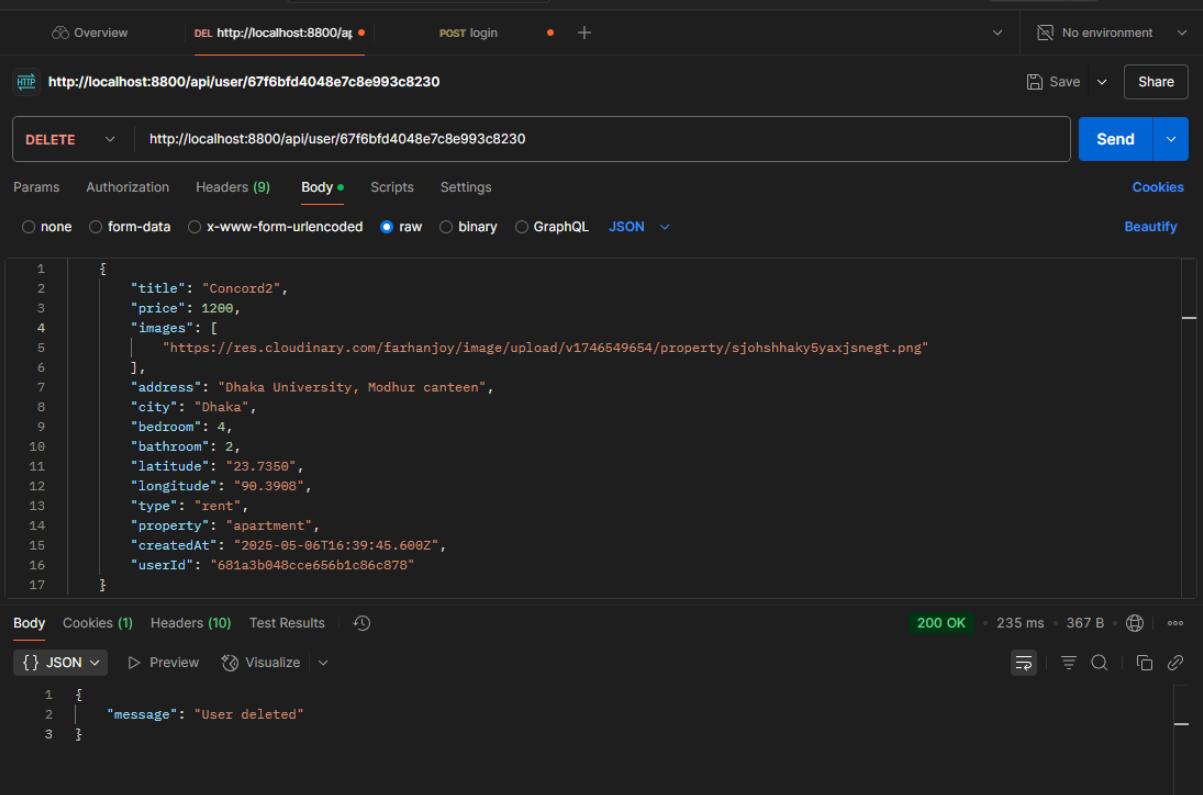
```

50
51  export const deleteUser = async (req, res) => {
52    const id = req.params.id;
53    const tokenUserId = req.userId;
54
55    try {
56      await prisma.user.delete({
57        where: { id },
58      });
59      res.status(200).json({ message: "User deleted" });
60    } catch (err) {
61      console.log(err);
62      res.status(500).json({ message: "Failed to delete users!" });
63    }
64  };
65

```

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.



The screenshot shows a Postman interface with the following details:

- URL: `http://localhost:8800/api/user/67f6bfd4048e7c8e993c8230`
- Method: `DELETE`
- Body tab selected, showing JSON data for a user:

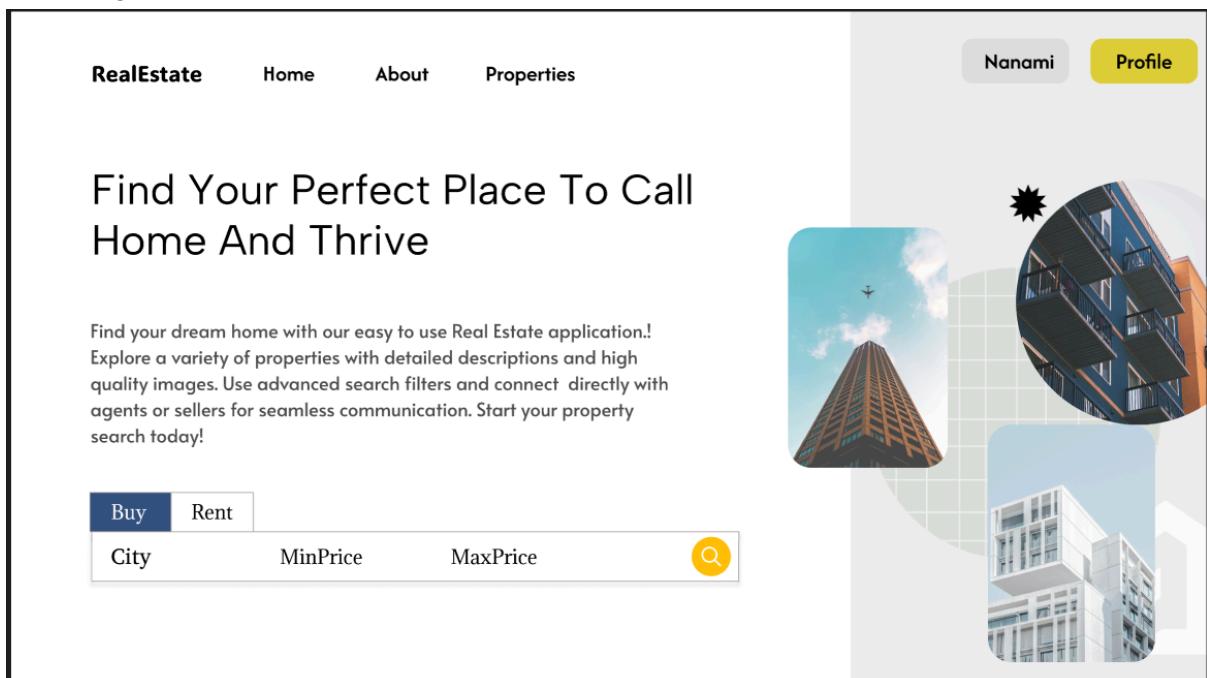
```
1  {
2   "title": "Concord2",
3   "price": 1200,
4   "images": [
5     "https://res.cloudinary.com/fazhanjoy/image/upload/v1746549654/property/sjohshhaky5yaxjsnegt.png"
6   ],
7   "address": "Dhaka University, Modhui canteen",
8   "city": "Dhaka",
9   "bedroom": 4,
10  "bathroom": 2,
11  "latitude": "23.7350",
12  "longitude": "90.3900",
13  "type": "rent",
14  "property": "apartment",
15  "createdAt": "2025-05-06T16:39:45.600Z",
16  "userId": "681a3b048cce656b1c86c878"
17 }
```

- Response status: `200 OK`
- Response time: `235 ms`
- Response size: `367 B`
- Response body (JSON):

```
1  {
2   "message": "User deleted"
3 }
```

## 5. User Interface Design

Homepage:



## Property Listings:

**RealEstate** Home About Properties

Nanami Profile

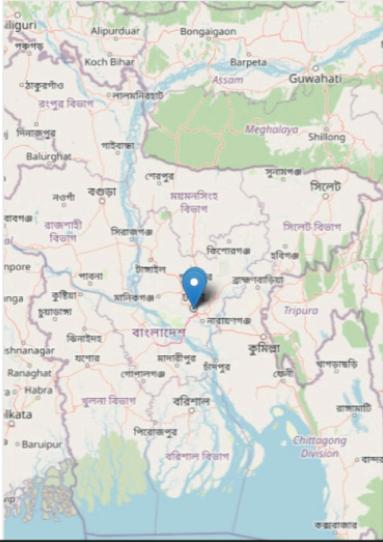
Search results for

Location

Type  Property  MinPrice  MaxPrice  Bedroom

 Haque Villa  
location

 Lucky Villa  
location



## User Information:

TV - 5

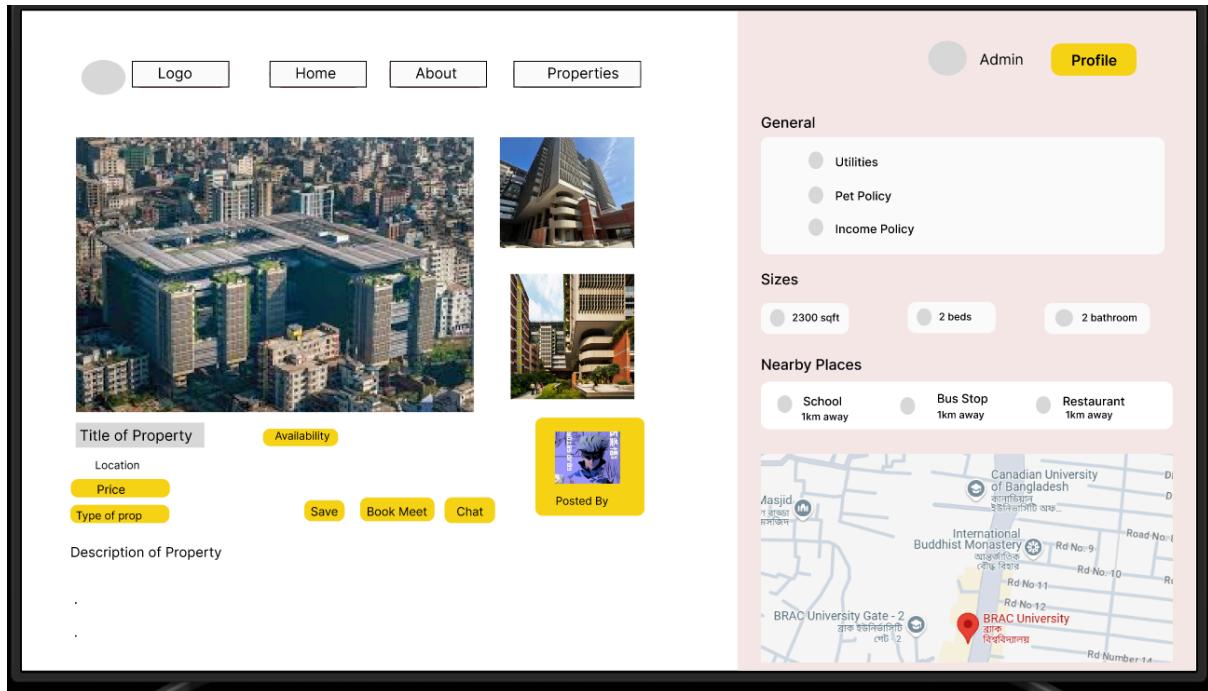
The screen shows a user profile for "RealEstate". The top navigation bar includes "RealEstate", "Home", "About", "Properties", "Nanami", and a yellow "Profile" button. Below the navigation, the title "User Information" is displayed. The user's details are listed: Username: nanamili9, E-mail: nanamisaturo@hotmail.com, and Phone number: 9331702. A "Logout" button is located at the bottom left. On the right side, there is a "Change your avatar" button. The main content area displays a property listing for "Haque Villa" located in a unspecified location, priced at BDT 35,000, featuring 3 bedrooms and 2 bathrooms. There are icons for update, bookmark, message, and delete.

## Create Post:

TV - 3

The screen shows a form titled "Add New Post" under the "RealEstate" section. The top navigation bar includes "RealEstate", "Home", "About", "Properties", "Nanami", and a yellow "Profile" button. The form fields include "Title", "Price", and "Address" (all in light blue boxes). Below these is a large "Description" text area. To the right, there is a "Upload picture" button. The next row contains "City", "Bedroom Number", and "Bathroom Number" fields. The following row contains "Latitude", "Longitude", and "Type" fields. The next row contains "Property", "Utility Policy", and "Pet Policy" fields. The final row contains "Profession" and "Size in sqft" fields, followed by a large "Add" button.

## Property Information:



Farhan:

[https://www.figma.com/proto/nJKlnnBlO1m4QfOWGtaTwp/CSE471\\_Pro?node-id=29-67&t=KPCK32JPXctUHrEf-1&starting-point-node-id=1%3A3](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

```
1 import "./login.scss";
2 import { Link, useNavigate } from "react-router-dom";
3 import { useContext, useState } from "react";
4 import apiRequest from "../../lib/apiRequest";
5 import { AuthContext } from "../../../context/AuthContext";
6 import axios from "axios";
7 function Login() {
8   const [error, setError] = useState("");
9   const [isLoading, setIsLoading] = useState(false)
10  const { updateUser } = useContext(AuthContext)
11
12  const navigate = useNavigate()
13
14  const handleSubmit = async (e) =>{
15    e.preventDefault()
16    setIsLoading(true)
17    setError("")
18
19    const formData = new FormData(e.target);
20
21    const username = formData.get("username")
22    const password = formData.get("password")
23
24    try{
25      const res = await apiRequest.post("/auth/login",{
26        | username, password
27      })
28      updateUser(res.data)
29
30      navigate("/profile")
31
32    }catch(err){
33      console.log(err)
34      setError(err.response.data.message);
35    }finally{
36      setIsLoading(false)
37    }
38  }
39  return (
40    <div className="login">
41      <div className="formContainer">
42        <form onSubmit={handleSubmit}>
43          <h1>Welcome</h1>
44          <input name="username" type="text" placeholder="Username" />
45          <input name="password" type="password" placeholder="Password" />
46          <button disabled={isLoading}>Login</button>
47          {error && <span>{error}</span>}
48          <p>
49            Don't have an account? <Link to="/register">Register</Link>
50          </p>
51        </form>
52      </div>
53      <div className="imgContainer">
54        
55      </div>
56    </div>
57  );
58}
59
60 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
1 import List from "../../components/list/List";
2 import { Suspense, useContext, useEffect, useState } from "react";
3 import { Link, useLoaderData, useNavigate, Await } from "react-router-dom";
4 import { AuthContext } from "../../context/AuthContext";
5 import apiRequest from "../../lib/apiRequest";
6 import "./profilePage.scss";
7 import Chat from "../../components/chat/chat"
8
9
10 function ProfilePage() {
11   const { updateUser, currentUser } = useContext(AuthContext);
12   const [userPosts, setUserPosts] = useState([]);
13   const navigate = useNavigate();
14   const data = useLoaderData();
15
16   const handleAdmin = () => {
17     navigate("/admin");
18   };
19
20   useEffect(() => {
21     const handlePosts = async () => {
22       try {
23         const response = await apiRequest.get("/prop/");
24         const filteredData = response.data.filter(
25           (item) => item.userId === currentUser.id
26         );
27         setUserPosts(filteredData);
28       } catch (error) {
29         console.error("Error fetching posts:", error);
30       }
31     };
32     handlePosts();
33   }, [currentUser.id]);
34
35   const handleLogout = async () => {
36     try {
37       await apiRequest.post("/auth/logout");
38       updateUser(null);
39       navigate("/home");
40     } catch (err) {
41       console.log(err);
42     }
43   };
44   console.log(userPosts) ×106 × []
45   return (
46     <div className="profilePage">
47       <div className="details">
48         <div className="wrapper">
49           <div className="title">
50             <h1>User Information</h1>
51             <Link to="/profile/update">
52               <button>Update Profile</button>
53             </Link>
54           </div>
55           <div className="info">
56             <span>
57               Avatar:
58               <img src={currentUser.avatar || "noavatar.jpg"} alt="" />
59             </span>
60             <span>
```

```

59         </span>
60         <span>
61           | Username: <b>{currentUser.username}</b>
62         </span>
63         <span>
64           | E-mail: <b>{currentUser.email}</b>
65         </span>
66         <br />
67         <button onClick={handleLogout}>Logout</button>
68       {currentUser.email === "admin@gmail.com" && (
69         <button onClick={handleAdmin}>Admin</button>
70       )}
71     </div>
72     <div className="title">
73       <h1>My List</h1>
74       <Link to="/add">
75         <button>Create New Post</button>
76       </Link>
77     </div>
78     <List posts={userPosts} />
79   </div>
80 </div>
81 <div className="chatContainer">
82   <div className="wrapper">
83     <Suspense fallback={(<p>Loading...</p>)}>
84       <Await
85         resolve={data.chatResponse}
86         errorElement={(<p>Error loading chats!</p>)}
87       >
88         {(<chatResponse>) => <Chat chats={chatResponse.data} />}
89       </Await>
90     </Suspense>
91   </div>
92 </div>
93 </div>
94 );
95 }
96
97 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

1 import DOMPurify from "dompurify";
2 import { useContext, useEffect, useState } from "react";
3 import { useLoaderData, useNavigate } from "react-router-dom";
4 import Map from "../../components/map/map";
5 import Slider from "../../components/slider/Slider";
6 import { AuthContext } from "../../context/AuthContext";
7 import "react-datepicker/dist/react-datepicker.css";
8 import "./singlePage.scss";
9 import Calendar from "../../components/calendar/calendar";
10
11
12
13 function SinglePage() {
14   const post = useLoaderData();
15   const { currentUser } = useContext(AuthContext);
16   const navigate = useNavigate();
17
18   return (
19     <div className="singlePage">
20       <div className="details">
21         <div className="wrapper">
22           <Slider images={post.images} />
23           <div className="info">
24             <div className="top">
25               <div className="post">
26                 <h1>{post.title}</h1>
27                 <div className="address">
28                   
29                   <span>{post.address}</span>
30                 </div>
31               </div>
32             </div>
33           </div>
34           <div className="user">
35             <img src={post.user.avatar} alt="" />
36             <span>{post.user.username}</span>
37           </div>
38         </div>
39       </div>
40       <div className="bottom" dangerouslySetInnerHTML={{ _html: DOMPurify.sanitize(post.postDetail.desc), }}>
41     </div>
42   </div>
43 </div>
44 <div className="features">
45   <div className="wrapper">
46     <p>General</p>
47     <div className="listVertical">
48       <div className="feature">
49         
50         <div className="featureText">
51           <span>Utilities</span>
52           {post.postDetail.utilities === "owner" ? (
53             <p>Owner is responsible</p>
54           ) : (
55             <p>Tenant is responsible</p>
56           )}
57         </div>
58       </div>
59       <div className="feature">
60         
61         <div className="featureText">
62           <span>Pet Policy</span>
63           {post.postDetail.pet === "allowed" ? (
64             <p>Pets Allowed</p>
65           ) : (
66             <p>Pets not Allowed</p>
67           )}
68     </div>
69   </div>
70 </div>
71 </div>
72 </div>
```

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

13 function SinglePage() {
29   <span>{post.address}</span>
30   </div>
31   <div className="price">
32     <BDT>{post.price}
33   </div>
34   </div>
35   <div className="user">
36     <img src={post.user.avatar} alt="" />
37     <span>{post.user.username}</span>
38   </div>
39   </div>
40   <div className="bottom" dangerouslySetInnerHTML={{ _html: DOMPurify.sanitize(post.postDetail.desc), }}>
41   </div>
42 </div>
43 <div className="features">
44   <div className="wrapper">
45     <p>General</p>
46     <div className="listVertical">
47       <div className="feature">
48         
49         <div className="featureText">
50           <span>Utilities</span>
51           {post.postDetail.utilities === "owner" ? (
52             <p>Owner is responsible</p>
53           ) : (
54             <p>Tenant is responsible</p>
55           )}
56         </div>
57       </div>
58       <div className="feature">
59         
60         <div className="featureText">
61           <span>Pet Policy</span>
62           {post.postDetail.pet === "allowed" ? (
63             <p>Pets Allowed</p>
64           ) : (
65             <p>Pets not Allowed</p>
66           )}
67         </div>
68       </div>
69     </div>
70   </div>
71 </div>
72 </div>
```

```

singlePage.jsx ×
CSF471_Project > client > src > routes > singlePage > singlePage.jsx > SinglePage
13  function SinglePage() {
72    |   }
73    |   </div>
74    |   </div>
75    |   <div className="feature">
76    |     
77    |     <div className="featureText">
78    |       <span>Income Policy</span>
79    |       <p>{post.postDetail.income}</p>
80    |     </div>
81    |   </div>
82    |   </div>
83    |   <p className="title">Sizes</p>
84    |   <div className="sizes">
85    |     <div className="size">
86    |       
87    |       <span>{post.postDetail.size} sqft</span>
88    |     </div>
89    |     <div className="size">
90    |       
91    |       <span>{post.bedroom} beds</span>
92    |     </div>
93    |     <div className="size">
94    |       
95    |       <span>{post.bathroom} bathroom</span>
96    |     </div>
97    |   </div>
98    |   <p className="title">Nearby Places</p>
99    |   <div className="listHorizontal">
100   |     <div className="feature">
101   |       
102   |       <div className="featureText">
103   |         <span>School</span>
104   |         <p>
105   |           {post.postDetail.school > 999
106   |             ? post.postDetail.school / 1000 + "km"
107   |             : post.postDetail.school + "m"}{" "}
108   |         away
109   |       </p>
110   |     </div>
111   |   </div>
112   |   <div className="feature">
113   |     
114   |     <div className="featureText">
115   |       <span>Bus Stop</span>

```

```

</div>
<div className="feature">

<div className="featureText">
<span>Bus Stop</span>
<p>{post.postDetail.bus}m away</p>
</div>
</div>
<div className="feature">

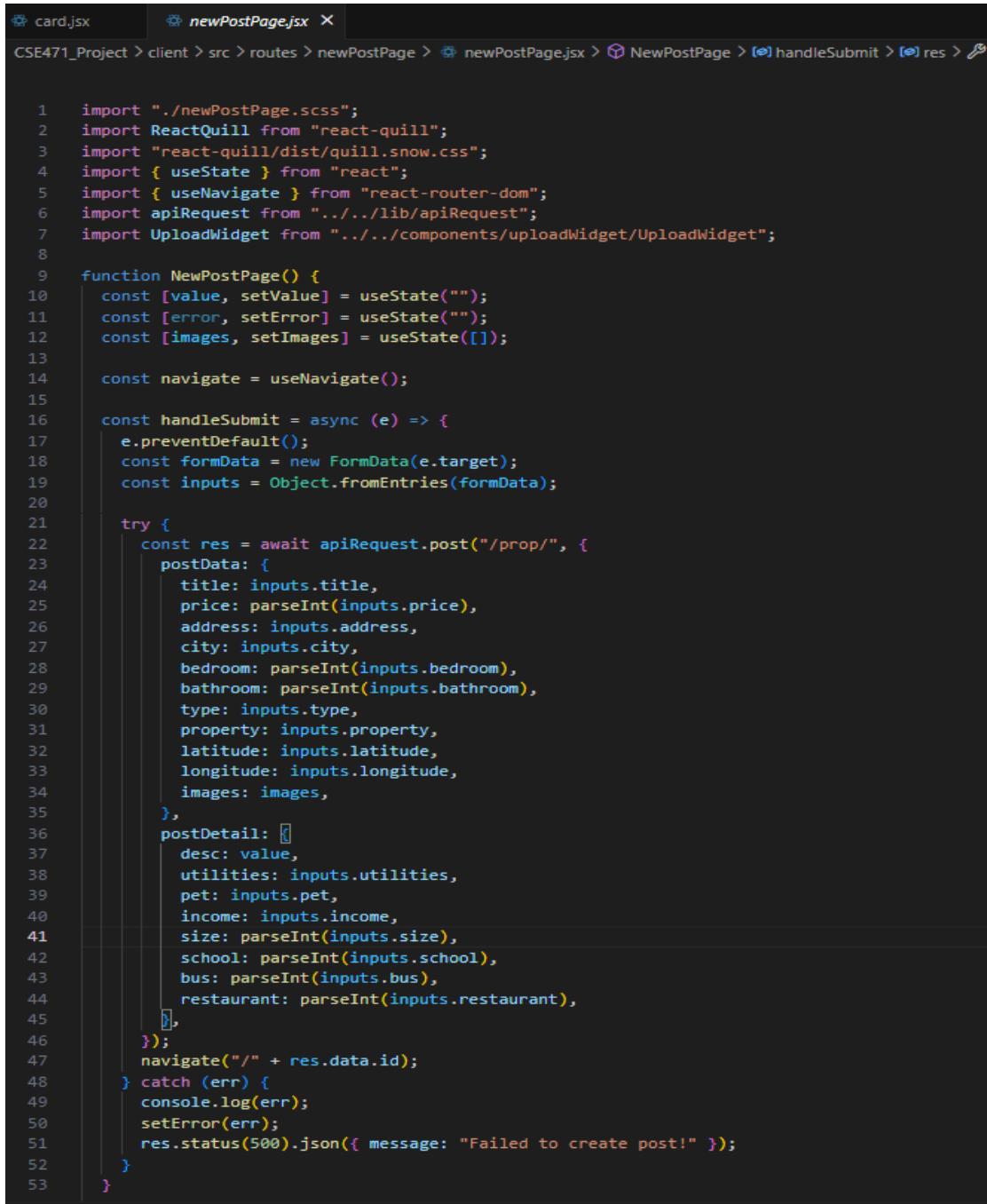
<div className="featureText">
<span>Restaurant</span>
<p>{post.postDetail.restaurant}m away</p>
</div>
</div>
</div>
<p className="title">Book your slot</p>
<div className = "date-range-calender">
  {currentUser?.id !== post.userId &&
  <Calendar
    propertyId={post.id} // Pass post.id as propertyId 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>
);
}

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



```

1  import "./newPostPage.scss";
2  import ReactQuill from "react-quill";
3  import "react-quill/dist/quill.snow.css";
4  import { useState } from "react";
5  import { useNavigate } from "react-router-dom";
6  import apiRequest from "../../lib/apiRequest";
7  import UploadWidget from "../../components/uploadWidget/UploadWidget";
8
9  function NewPostPage() {
10    const [value, setValue] = useState("");
11    const [error, setError] = useState("");
12    const [images, setImages] = useState([]);
13
14    const navigate = useNavigate();
15
16    const handleSubmit = async (e) => {
17      e.preventDefault();
18      const formData = new FormData(e.target);
19      const inputs = Object.fromEntries(formData);
20
21      try {
22        const res = await apiRequest.post("/prop/", {
23          postData: {
24            title: inputs.title,
25            price: parseInt(inputs.price),
26            address: inputs.address,
27            city: inputs.city,
28            bedroom: parseInt(inputs.bedroom),
29            bathroom: parseInt(inputs.bathroom),
30            type: inputs.type,
31            property: inputs.property,
32            latitude: inputs.latitude,
33            longitude: inputs.longitude,
34            images: images,
35          },
36          postDetail: [
37            desc: value,
38            utilities: inputs.utilities,
39            pet: inputs.pet,
40            income: inputs.income,
41            size: parseInt(inputs.size),
42            school: parseInt(inputs.school),
43            bus: parseInt(inputs.bus),
44            restaurant: parseInt(inputs.restaurant),
45          ],
46        });
47        navigate从根本路径 + res.data.id);
48      } catch (err) {
49        console.log(err);
50        setError(err);
51        res.status(500).json({ message: "Failed to create post!" });
52      }
53    }
}

```

```
❶ card.jsx ❷ newPostPage.jsx X
CSE471_Project > client > src > routes > newPostPage > ❸ newPostPage.jsx > ❹ NewPostPage > ❺ handleSubmit > ❻ res > ❼ postDetail
  9  function NewPostPage() {
16    const handleSubmit = async (e) => {
51      res.status(500).json({ message: "Failed to create post!" });
52    }
53  }
54
55  return (
56    <div className="newPostPage">
57      <div className="formContainer">
58        <h1>Add New Post</h1>
59        <div className="wrapper">
60          <form onSubmit={handleSubmit}>
61            <div className="item">
62              <label htmlFor="title">Title</label>
63              <input id="title" name="title" type="text" />
64            </div>
65            <div className="item">
66              <label htmlFor="price">Price</label>
67              <input id="price" name="price" type="number" />
68            </div>
69            <div className="item">
70              <label htmlFor="address">Address</label>
71              <input id="address" name="address" type="text" />
72            </div>
73            <div className="item description">
74              <label htmlFor="desc">Description</label>
75              <ReactQuill theme="snow" onChange={setValue} value={value} />
76            </div>
77            <div className="item">
78              <label htmlFor="city">City</label>
79              <input id="city" name="city" type="text" />
80            </div>
81            <div className="item">
82              <label htmlFor="bedroom">Bedroom Number</label>
83              <input min={1} id="bedroom" name="bedroom" type="number" />
84            </div>
85            <div className="item">
86              <label htmlFor="bathroom">Bathroom Number</label>
87              <input min={1} id="bathroom" name="bathroom" type="number" />
88            </div>
89            <div className="item">
90              <label htmlFor="latitude">Latitude</label>
91              <input id="latitude" name="latitude" type="text" />
92            </div>
93            <div className="item">
94              <label htmlFor="longitude">Longitude</label>
95              <input id="longitude" name="longitude" type="text" />
96            </div>
97            <div className="item">
98              <label htmlFor="type">Type</label>
99              <select name="type">
100                <option value="rent" defaultChecked>
101                  Rent
102                </option>
103                <option value="buy">Buy</option>

```

```

  card.jsx      newPostPage.jsx
CSE471_Project > client > src > routes > newPostPage > newPostPage.jsx > NewPostPage > handleSubmit > res > postD
  9   function NewPostPage() {
105
106     </div>
107     <div className="item">
108       <label htmlFor="type">Property</label>
109       <select name="property">
110         <option value="apartment">Apartment</option>
111         <option value="house">House</option>
112         <option value="condo">Condo</option>
113         <option value="land">Land</option>
114       </select>
115     </div>
116
117     <div className="item">
118       <label htmlFor="utilities">Utilities Policy</label>
119       <select name="utilities">
120         <option value="owner">Owner is responsible</option>
121         <option value="tenant">Tenant is responsible</option>
122         <option value="shared">Shared</option>
123       </select>
124     </div>
125     <div className="item">
126       <label htmlFor="pet">Pet Policy</label>
127       <select name="pet">
128         <option value="allowed">Allowed</option>
129         <option value="not-allowed">Not Allowed</option>
130       </select>
131     </div>
132     <div className="item">
133       <label htmlFor="income">Income Policy</label>
134       <input
135         id="income"
136         name="income"
137         type="text"
138         placeholder="Income Policy"
139       />
140     </div>
141     <div className="item">
142       <label htmlFor="size">Total Size (sqft)</label>
143       <input min={0} id="size" name="size" type="number" />
144     </div>
145     <div className="item">
146       <label htmlFor="school">School</label>
147       <input min={0} id="school" name="school" type="number" />
148     </div>
149     <div className="item">
150       <label htmlFor="bus">bus</label>
151       <input min={0} id="bus" name="bus" type="number" />
152     </div>
153     <div className="item">
154       <label htmlFor="restaurant">Restaurant</label>
155       <input min={0} id="restaurant" name="restaurant" type="number" />
156     </div>
157     <button className="sendButton">Add</button>
158   </form>
159 </div>

```

```

155   </div>
156   <button className="sendButton">Add</button>
157 </form>
158 </div>
159 </div>
160 <div className="sideContainer">
161   <div className="imageContainer">
162     {images.map((image, index) => (
163       <img src={image} key={index} alt="" />
164     ))}
165   </div>
166   <UploadWidget
167     className="uploadButton"
168     uwConfig={{
169       cloudName: "farhanjoy",
170       uploadPreset: "estate",
171       multiple: true,
172       folder: "property",
173     }}
174     setState={setImages}
175   />
176 </div>
177 </div>
178 );
179 }
180
181 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

```

1 //import { postData } from '../../../../../lib/dummydata'
2 import { Suspense } from "react";
3 import { Await, useLoaderData } from "react-router-dom";
4 import './listPage.scss'
5 import Filter from '../../../../../components/filter/Filter'
6 import Map from '../../../../../components/map/map'
7 import Card from '../../../../../components/card/card"
8
9 function ListPage() {
10   const data = useLoaderData();
11
12   return (
13     <div className="listPage">
14       <div className="listContainer">
15         <div className="wrapper">
16           <Filter />
17           <Suspense fallback={<p>Loading...</p>}>
18             <Await
19               resolve={data.postResponse}
20               errorElement={<p>Error loading posts!</p>}
21             >
22               {(postResponse) =>
23                 postResponse.data.map((post) => (
24                   <Card key={post.id} item={post} />
25                 ))
26               }
27             </Await>
28           </Suspense>
29         </div>
30       <div className="mapContainer">
31         <Suspense fallback={<p>Loading...</p>}>
32           <Await
33             resolve={data.postResponse}
34             errorElement={<p>Error loading posts!</p>}
35           >
36             {(postResponse) => <Map items={postResponse.data} />}
37           </Await>
38         </Suspense>
39       </div>
40     </div>
41   );
42 }
43

```

```
1 import apiRequest from "../../lib/apiRequest";
2 import { useContext } from "react";
3 import { Link } from "react-router-dom";
4 import { AuthContext } from "../../context/AuthContext";
5 import { useNavigate } from "react-router-dom";
6 import axios from "axios";
7 import "./card.scss";
8
9 function Card({ item }) {
10   const { currentUser } = useContext(AuthContext);
11   const navigate = useNavigate();
12
13   const handlePostDelete = async (id) => {
14     try {
15       const response = await axios.delete(`api/prop/${id}`);
16       if (response.status === 200) {
17         console.log("successfully deleted post");
18         window.location.reload();
19       } else {
20         console.error("Failed to delete post");
21       }
22     } catch (error) {
23       console.error("Failed to delete post", error);
24     }
25   };
26
27   const handleChatClick = async () => {
28     try {
29       const res = await apiRequest.post("/chats", {
30         receiverId: item.userId,
31       });
32
33       const chatId = res.data?.id;
34       if (!chatId) {
35         console.error("chatId not returned from server");
36         return;
37       }
38
39       await apiRequest.post(`messages/${chatId}`, {
40         text: `I am Interested to buy ${item.title}`,
41       });
42       navigate("/profile");
43
44     } catch (error) {
45       console.error("Failed to initiate chat:", error);
46     }
47   };
48 }
```

```

card.jsx X listPage.jsx
CSE471_Project > client > src > components > card > card.jsx > Card
9  function Card({ item }) {
49
50    return (
51      <div className="card">
52        <Link to={`/ ${item.id}`}> className="imageContainer">
53          <img src={item.images[0]} alt="" />
54        </Link>
55        <div className="textContainer">
56          <h2 className="title">
57            <Link to={`/ ${item.id}`}>{item.title}</Link>
58          </h2>
59          <p className="address">
60            
61            {item.address}</p>
62          </p>
63          <p className="price">BDT {item.price}</p>
64          <div className="bottom">
65            <div className="features">
66              <div className="feature">
67                
68                {item.bedroom} bedroom</div>
69              </div>
70              <div className="feature">
71                
72                {item.bathroom} bathroom</div>
73            </div>
74            <div className="icons">
75              {currentUser.id !== item.userId && (
76                <Link to={`/ ${item.id}`}> className="icon">
77                  
78                </Link>
79              )}
80              <div className="icon">
81                
82              </div>
83              {currentUser.id !== item.userId && (
84                <div className="icon" onClick={() => handleChatClick(item.userId)}>
85                  
86                </div>
87              )}
88              {currentUser.email === "admin@gmail.com" ||
89               currentUser.id === item.userId && (
90                <div className="icon" onClick={() => handlePostDelete(item.id)}>
91                  <p>X</p>
92                </div>
93              )}
94            </div>
95          </div>
96        </div>
97      </div>
98    </div>
99  );
100}
101
102 export default Card;

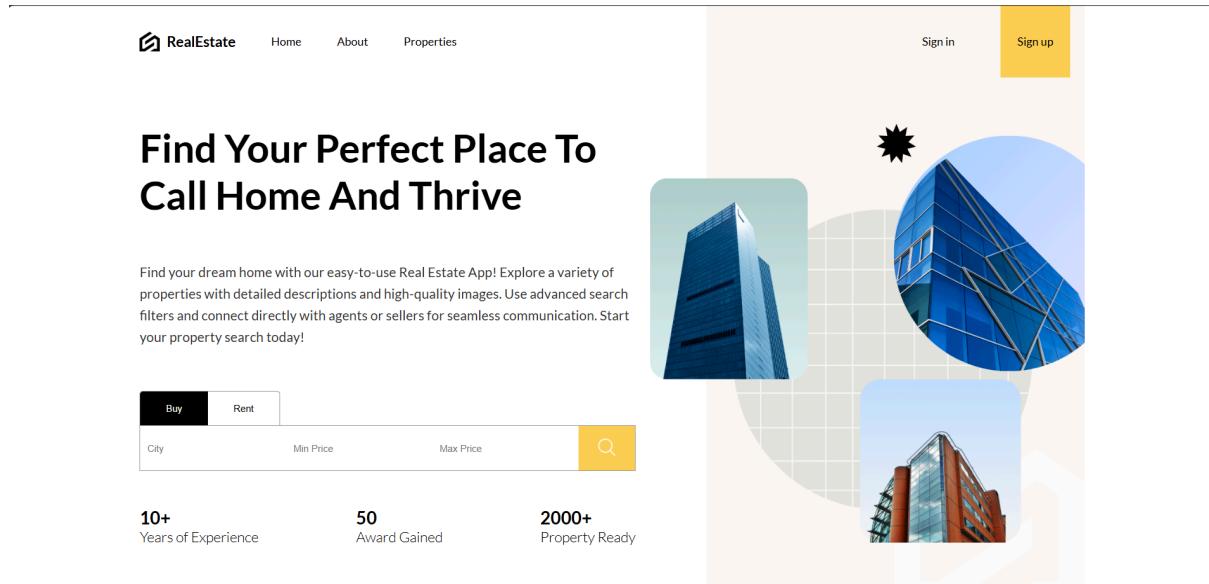
```

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

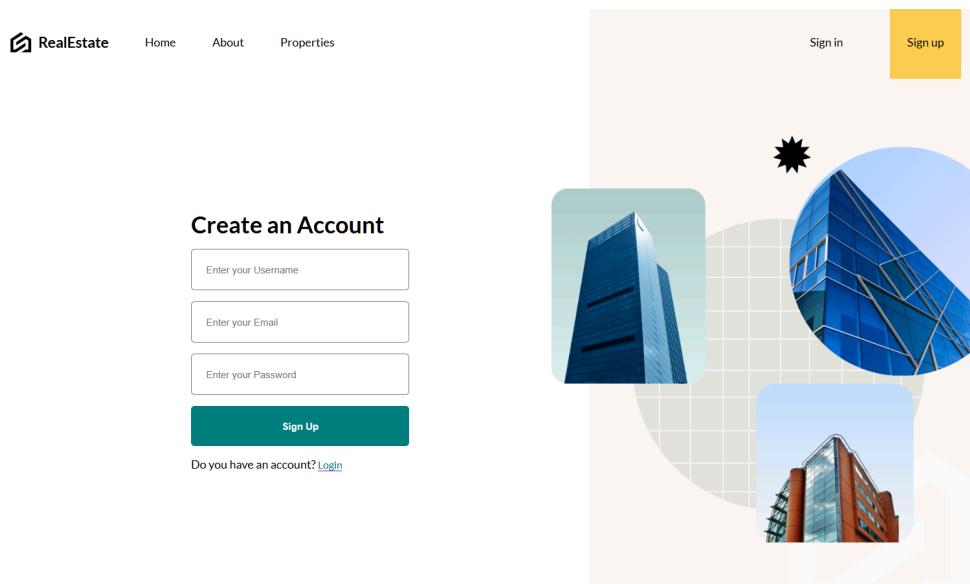
owner, view details, or, if they are the owner or admin, delete the post. The design adapts based on the current user's role, enhancing usability and control.

## 7. User Manual

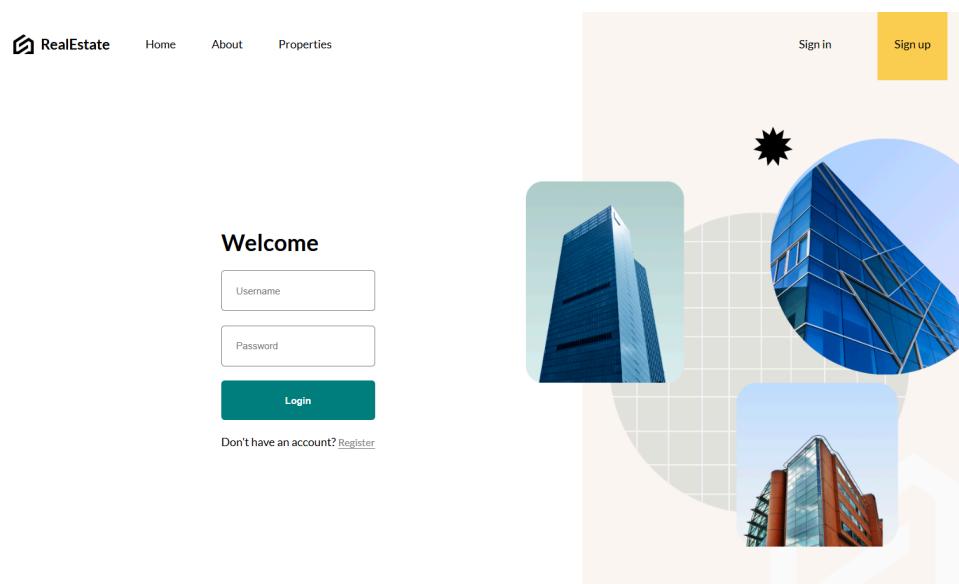
- When users enter the app, they are directed to the Home page. However, they must be registered and logged in to view property listings or access other parts of the website. Clicking "Sign In" or "Sign Up" takes them to the login or registration page.



- 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.

The image contains two screenshots. The left screenshot shows a modal dialog titled 'Update Profile' with three input fields: 'Username' (containing 'Tiham12'), 'Email' (containing 'tiham12@gmail.com'), and 'Password' (containing '.....'). Below the fields is a yellow 'Update' button. The right screenshot shows the user's profile page. It features a circular profile picture, the username 'Tiham12', and a yellow 'Profile' button. A red circle with the number '2' is positioned above the 'Profile' button, indicating pending updates or notifications.

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.

The image shows two screenshots of a real estate application. The top screenshot displays a property listing titled "Dhaka University" located at "Dhaka University, Modhur canteen". It shows a red brick building with a prominent tower and palm trees. The price is listed as "BDT 100000". Below the listing are buttons for "4 bedroom", "2 bathroom", and two icons (a bookmark and a trash). A yellow oval highlights the "Create New Post" button in the top right corner of the listing card. The bottom screenshot shows the "Add New Post" form. It includes fields for Title, Price, Address, Description (with rich text editor), City, Bedroom Number, Bathroom Number, Latitude, Longitude, and Type (set to Rent). There is also an "Upload" button and a sidebar with a user profile for "Tiham12" and a notification count of 3.

The screenshot shows a property listing for "Shopno Dhara". The main image is a modern, multi-story building. Below it, the property name "Shopno Dhara" is displayed, along with its location "Iskaton" and price "BDT 50000". To the right, there is a yellow callout box containing the user's profile information: "Tiham12". The right side of the screen shows the user's profile page for "Tiham12", which includes sections for General (Utilities: Tenant is responsible, Pet Policy: Pets not allowed, Income Policy: mom), Sizes (2100 sqft, 4 beds, 4 bathroom), Nearby Places (School 1m away, Bus Stop 0m away, Restaurant 25m away), and Location (a map of the area).

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

The screenshot shows the property listing page. At the top, there is a search bar with the placeholder "Search results for" and a "Location" dropdown set to "City Location". Below the search bar, there are filters for "Type" (any), "Property" (any), "Min Price" (any), "Max Price" (any), and "Bedroom" (any). A search button is located to the right of the filters. The main content area displays two property listings: "Dhamondi" (a modern building, BDT 12000, 3 bedroom, 2 bathroom) and "BRAC University" (a large, modern building at night, BDT 5000, 20 bedroom, 10 bathroom). To the right, there is a map of the region, with two blue markers indicating specific locations. The user's profile "Tiham12" is visible at the top right of the map.

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.



## Search results for Dhaka

Location

Type

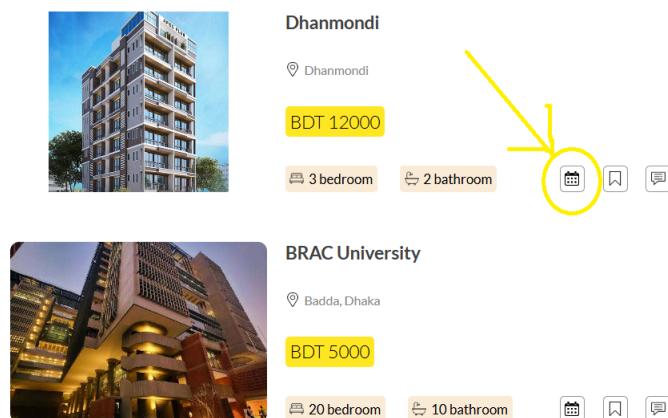
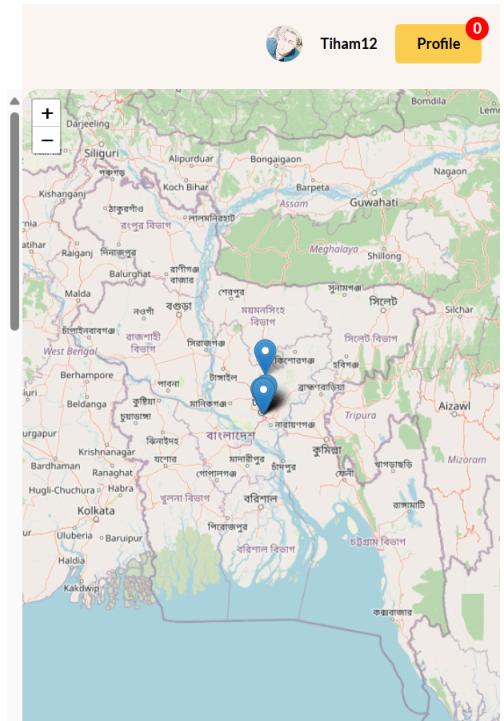
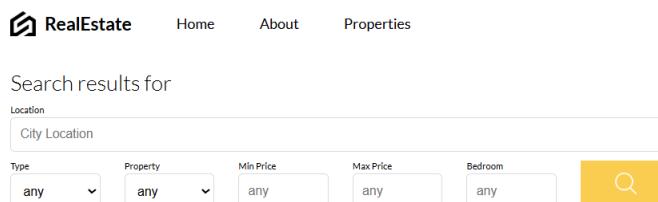
Property

Min Price

Max Price

Bedroom

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.



Concord

Dhaka University, Modhur canteen

BDT 1200

Urbi

Tiham12 Profile 0

General

- Utilities: Owner is responsible
- Pet Policy: Pets Allowed
- Income Policy

Sizes

2000

	May 2025						Time	
	Su	Mo	Tu	We	Th	Fr	Sa	
27	28	29	30	1	2	3		02:00 PM
4	5	6	7	8	9	10		03:00 PM
11	12	13	14	15	16	17		04:00 PM
18	19	20	21	22	23	24		05:00 PM
25	26	27	28	29	30	31		06:00 PM

Book you

May 7, 2025 5:30 PM

Book Now

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.

RealEstate Home About Properties

Search results for

Location: City Location

Type: any Property: any Min Price: any Max Price: any Bedroom: any

BDT 12000

3 bedroom 2 bathroom

Dhanmondi

Dhanmondi

BDT 12000

3 bedroom 2 bathroom

BRAC University

Badda, Dhaka

BDT 5000

20 bedroom 10 bathroom

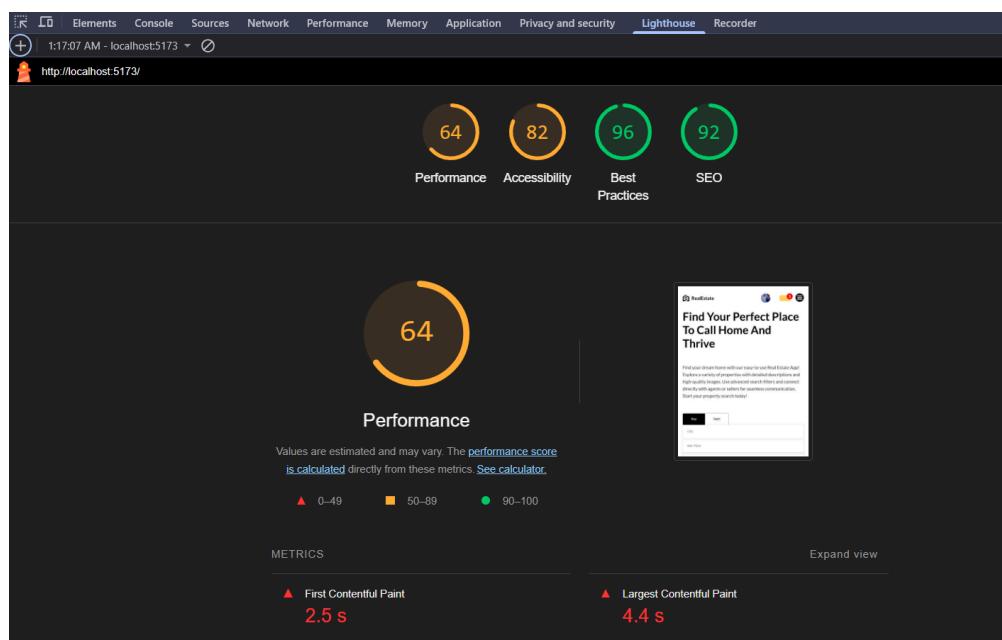
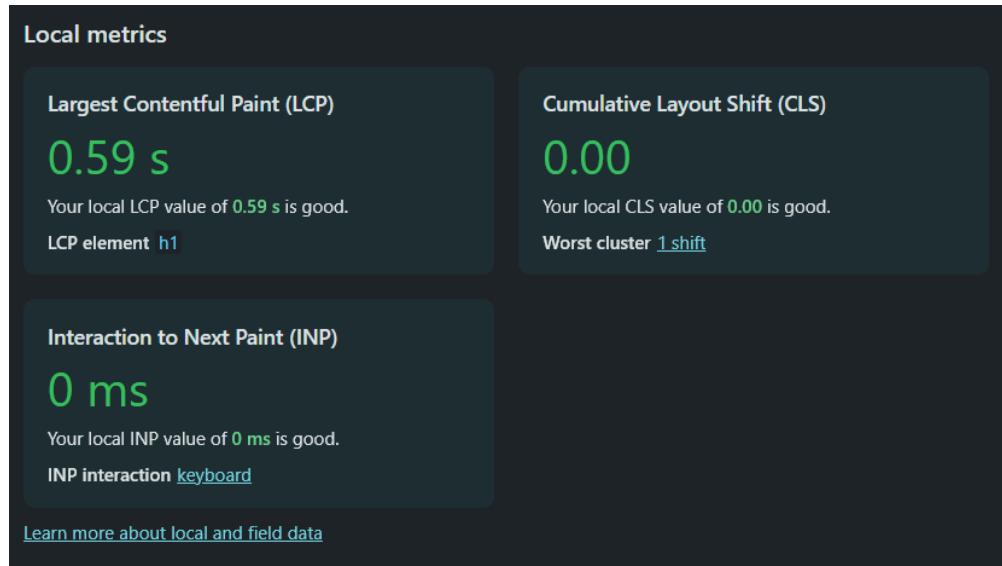
The screenshot shows a user profile page for 'Tiham12'. The top navigation bar includes links for Home, About, Properties, and Profile. The Profile link has a red notification badge with the number '3'. Below the navigation, the 'User Information' section displays the user's avatar, username 'Tiham12', and email 'tiham12@gmail.com'. A 'Logout' button is also present. To the right, a 'Messages' sidebar lists three notifications from users 'joy1212', 'Eusra Islam', and 'Urbi', all expressing interest in buying Dhaka University. A 'Create New Post' button is located at the top of the messages sidebar.

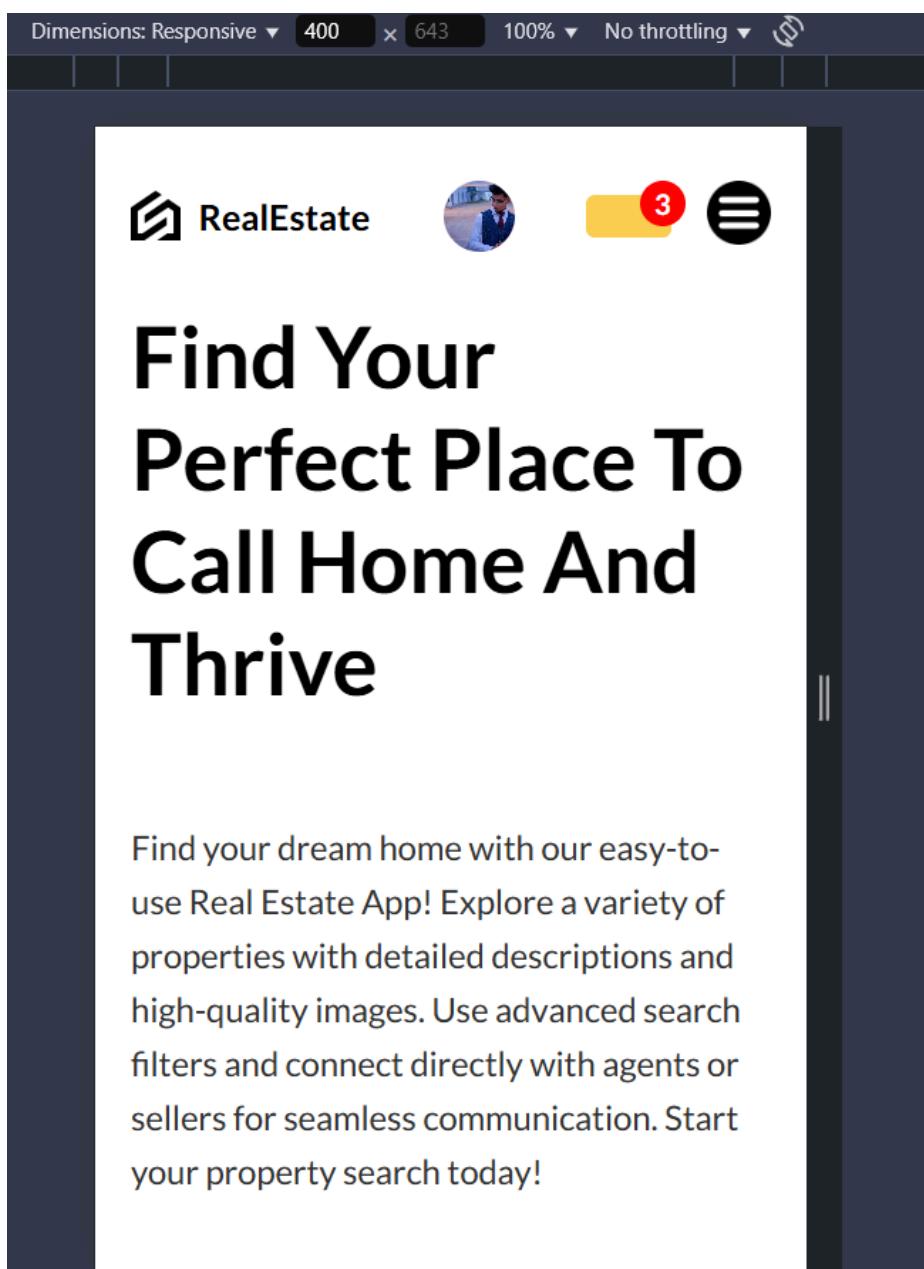
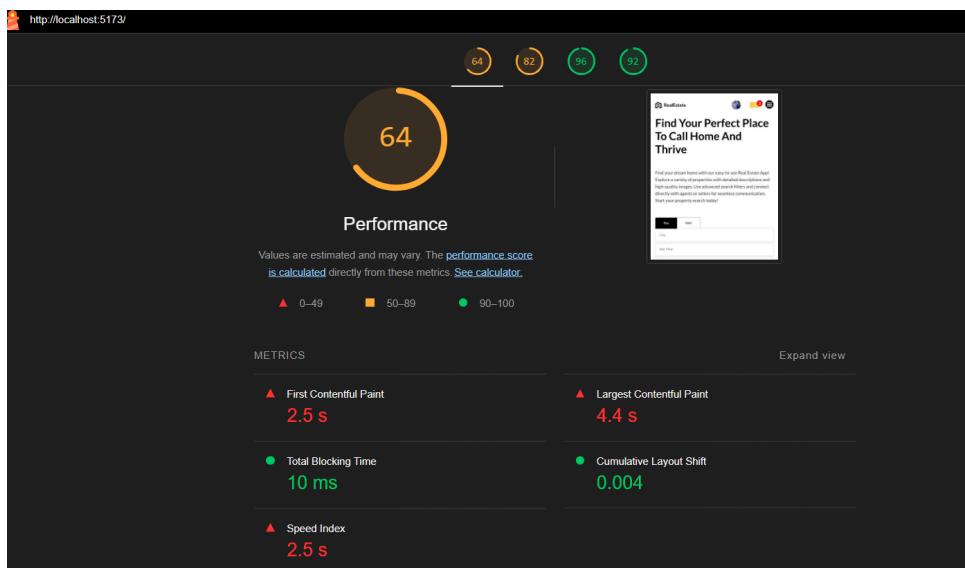
Here, after reading a message, the notification is changed from 3 to 2.

This screenshot shows the same user profile page for 'Tiham12' after one message has been read. The 'Profile' link now has a red notification badge with the number '2'. In the 'Messages' sidebar, the first message from 'joy1212' remains, while the messages from 'Eusra Islam' and 'Urbi' have been removed, indicated by an 'X' icon. A new message from 'Eusra Islam' has been added, with the timestamp '18 hours ago' and the text 'I am Interested to buy Dhaka University'. A 'Send' button is visible at the bottom of the messages sidebar.

## 8. Performance and Network Analysis

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





## **9. Github Repo [Public] Link**

[https://github.com/FarhanAbedin12/CSE471\\_Project](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
<b>Functional Requirements which are developed by this member:</b>	
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
<b>Functional Requirements which are developed by this member:</b>	
1. Property Listings	
2. Map integration	
3. Slot Reservation	
4. Notification of chats	

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

## **References:**