**Real Estate Market Place**

#### **A PROJECT REPORT**

***Submitted by***

**Shrish Pandey - Q - 2115000974**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

**IN**

##### Computer Engineering and Application

**GLA University, Mathura**

#### **BONAFIDE CERTIFICATE**

Certified that this project report **Real Estate Market Place Website** is the bonafide work of **Shrish Pandey** who carried out the project work under my supervision.

|  |  |
| --- | --- |
|  |  |
| Signature of the HoD  **SIGNATURE**  Rohit Agrawal  **HEAD OF THE DEPARTMENT**  CEA Department | Signature of the Supervisor  **SIGNATURE**  Mr. Ankit Arora  **SUPERVISOR**  Technical Trainer  CEA Department |

**Table Contents**

1. Abstract
2. Introduction
3. Motivation
4. Problem Statement
5. Basic Introduction (What Is MERN)
6. Project Description
7. Working
8. Conclusion
9. Result

10. Refrences

**1.ABSTRACT**

Welcome to the future of real estate transactions — the Real Estate Marketplace, where buying, selling, and renting properties seamlessly converge in a digital realm. In an era defined by connectivity and convenience, our platform revolutionizes the way individuals, businesses, and investors engage with the real estate market. It is designed and built for the users to share buy and sell or rent the properties. I have used MERN. MERN stands for MongoDB, Express JS, ReactJS and NodeJS (the four key technologies that make the stack) and is used to built this fully responsive web application in conjugation with other APIs and tools. Index Terms—MongoDB, Express JS, React, Node, Backend, Frontend, Redux dependencies, APIs, SPA (Single Page Application).

**2.INTRODUCTION**

Our platform is designed to be an intuitive, secure, and user-friendly space where property owners, buyers, and renters can connect effortlessly. Whether you're looking for your dream home, seeking a lucrative investment opportunity, or trying to find the perfect tenant for your property, Real Estate Marketplace is your all-encompassing solution.

**Key Features:**

1. **List and Discover Properties:** Property owners can easily list their homes, apartments, commercial spaces, and more. Prospective buyers and renters can explore a diverse range of listings with detailed information, high-quality images, and virtual tours.
2. **Smart Search and Filters:** Our advanced search functionality allows users to narrow down their preferences, ensuring they find the properties that truly match their criteria. From location and size to amenities and price range, our smart filters streamline the search process.
3. **Secure Transactions:** Facilitating secure transactions is our top priority. Real Estate Marketplace employs cutting-edge encryption and authentication protocols to safeguard financial transactions, protecting both buyers and sellers throughout the process.
4. **Real-time Communication:** Connect directly with property owners or potential buyers/renters through our real-time messaging system. This feature fosters efficient communication, ensuring that all parties involved stay informed and engaged.
5. **Market Insights and Analytics:** Stay ahead of market trends with our comprehensive analytics tools. Access valuable insights into property values, market trends, and investment opportunities to make informed decisions.
6. **User Profiles and Ratings:** Build trust within the community through user profiles and ratings. Transparent feedback and reviews contribute to a reliable and credible marketplace environment.

**Join the Future of Real Estate:**

Real Estate Marketplace is not just a platform; it's a community reshaping the real estate landscape. Whether you're a first-time homebuyer, a seasoned investor, or a property owner, our platform is your gateway to a seamless and efficient real estate experience.

Embark on your real estate journey with us and discover the possibilities that the Real Estate Marketplace has to offer. Welcome to a new era of property transactions, where convenience meets innovation.

**3.MOTIVATION**

This report aims to conduct a comprehensive technical evaluation of the Real Estate Marketplace project. It will delve into the intricacies of the platform's architecture, design, functionality, performance, and scalability. Developers and technical stakeholders will find valuable insights to assess the robustness of the codebase and identify areas for enhancement.Focusing on the user-centric aspects, this report seeks to analyze the overall user experience provided by the Real Estate Marketplace. It will evaluate the platform's user-friendliness, engagement features, and its ability to meet the diverse needs of its target audience. Designers and user experience experts can leverage this information to enhance the usability and overall user satisfaction.This report aims to assess the business potential of the Real Estate Marketplace project. It will delve into market demand, revenue potential, and growth opportunities within the real estate sector. Entrepreneurs considering the launch of a real estate platform and investors seeking to evaluate the project's viability will find crucial information to inform their strategic decisions.In order to understand the competitive landscape, this report will conduct a comparative analysis of the Real Estate Marketplace project against other platforms in the real estate market. Features, performance, user base, and revenue streams will be scrutinized to identify competitive advantages and weaknesses. Marketers can then utilize this data to formulate effective strategies for attracting users and gaining a competitive edge in the market.The Real Estate Marketplace project report serves as a multi-faceted tool, addressing technical intricacies, user experience nuances, business viability, and market competitiveness. Whether you're a developer seeking to optimize the codebase, a designer aiming to enhance user interactions, an entrepreneur evaluating business potential, or a marketer strategizing for market entry, this report provides a holistic view of the Real Estate Marketplace project, empowering stakeholders to make informed decisions. Welcome to a new era of real estate transactions, where technology meets user satisfaction and business viability.

**4.PROBLEM STATEMENT**

Real Estate Marketplace is not just a platform; it's a community reshaping the real estate landscape. Whether you're a first-time homebuyer, a seasoned investor, or a property owner, our platform is your gateway to a seamless and efficient real estate experience.

Embark on your real estate journey with us and discover the possibilities that the Real Estate Marketplace has to offer. Welcome to a new era of property transactions, where convenience meets innovation.

The current real estate landscape, traditional methods of property transactions are often marred by inefficiencies, lack of transparency, and limited accessibility. Recognizing these challenges, the Real Estate Marketplace project aims to address the following key problems:

1. Fragmented Property Transactions:.
2. Limited Market Insight:
3. Security and Trust Concerns:
4. Inconsistent User Experience:
5. Limited Business Opportunities:
6. Competitive Analysis Gaps:

The Real Estate Marketplace project seeks to overcome these challenges by providing a centralized, secure, and user-centric platform that not only facilitates seamless property transactions but also offers valuable market insights, enhances trust, and opens new avenues for businesses and investors in the real estate sector. Through innovation and technology, this project aims to transform the way individuals engage with real estate, creating a more efficient and transparent marketplace for all stakeholders

**5.What is MERN Stack :-**

MERN stack is a framework used for creating websites (webapp development). MongoDB, Express JS, ReactJS, and NodeJS make up its functional components. The specific role of each of these elements while creating a web application are listed below:• MongoDB: The application data is stored in this document-oriented, No-SQL database. • NodeJS: This is the JavaScript runtime environment that is used to run the JavaScript code on the machine itself, instead of a browser. • Express JS: It is a framework that sits atop NodeJS and is used to create a website’s backend using NodeJS functions and structures. NodeJS was created to run JavaScript on computers, not to create websites, so Express JS was created to fill that gap. • ReactJS: It is a library that Facebook built. It is used to build the UI elements that go into a single page web application’s user interface. The user interacts with the ReactJS UI components in the front-end of the application, which is situated in the browser. The backend of this application, which is located on a server, is served by Express JS, which is built upon NodeJS. A request to change data is sent to the Express server, which is built on NodeJS, after any interaction. When necessary, Express fetches information from the MongoDB database and sends it to the application’s front end, where it is shown to the user

**6. PROJECT DESCRIPTION**

**1) FRONTEND (CLIENT-SIDE):-**

Client-side rendering, often known as front-end development, is a new style of site rendering that is employed in contemporary apps. JavaScript, which is now the de facto standard web language, is used to render the content on your computer as opposed to a distant web server in client side rendering. In actuality, this indicates that a browser is responsible for generating the HTML output of the web application and that a server is only needed to provide the raw web application. Additionally, it shows that a piece of the presentation logic—the reasoning used to create a webpage and display it to the user on the screen—is handled on the client-side. With the introduction of JavaScript libraries like React client-side rendering became more common.

**2)BACKEND (SERVER-SIDE):-**

Building websites and web apps has always been done using server-side rendering, also referred to as back-end web development. When we access a page, we send a request for data to the server, which processes it and sends back a response to the browser. All the activities required to build an HTML page that the web browser can understand are carried out on the remote server that houses the website or web application when a website renders server-side. This entails processing any required logic as well as information queries from databases for that web application. While it waits for the distant server to finish processing the request and provide the response, the web browser on the other end sits idle. When a response is sent, web browsers interpret it and show the material on the screen.

**Scope:**

The Real Estate Marketplace project aims to establish a centralized platform for streamlined property transactions, offering users a user-friendly interface, comprehensive market insights, and enhanced security. With a focus on improving the overall real estate experience, the project endeavors to create a space that fosters trust, efficiency, and innovation in the dynamic real estate market.

**Methodology:**

The project will employ the following methodologies, tools, and technologies:

• Programming Languages: Node JS (for Backend Development), React (for FrontEnd Development)

• Software: Vs Code, Firebase for backend services, web development tools.

• Hardware: System for testing, Web server for hosting web components.

**Proposed System:**

This web app will be intuitive and user-friendly with a well-designed user interface. Users can register(also using google ), create profiles, share real estate listings, buy and rent these listings . and can contact the landlords easily.

**Features:**

• User Registration (Using google also ) and Profile Management

• Update Profile and Delete Profile

• Create Real Estate listing

• Update Listing

• Search Listings and Buy Sell and Contact The Landlords

• User Authentication and Data Security

1. **Working**

The Real Estate MarketPlace Web app is very user friendly with the option to login using gmail and without it . A Registered User can create a listing of his/her property . He/she can update or delete it .

Anyone can view the created listings and Contact the landlord to buy or rent the property .

The login api registers the user in database during signup and uses that to login the user and it also has the functionality to login using google It is made possible using firebase services.

It also has the option to update the profile and upload profile picture (made possible using firebase services again) .

Now creating listing also has the option to upload the pictures using firebase.

The listing can also be updated all the listings are saved in database and can be viewed by anyone.

**Front End Code**

App.jsx

*import* React *from* "react";

*import* { BrowserRouter, Routes, Route } *from* "react-router-dom";

*import* Home *from* "./pages/Home";

*import* SignIn *from* "./pages/SignIn";

*import* SignUp *from* "./pages/SignUp";

*import* About *from* "./pages/About";

*import* Profile *from* "./pages/Profile";

*import* Header *from* "./Components/Header";

*import* PrivateRoute *from* "./Components/PrivateRoute";

*import* CreateListing *from* "./pages/CreateListing";

*import* UpdateListing *from* "./pages/UpdateListing";

*import* Listing *from* "./pages/Listing";

*import* Search *from* "./pages/Search";

const App = () => {

*return* (

    <BrowserRouter>

      <Header />

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="/sign-in" element={<SignIn />} />

        <Route path="/sign-up" element={<SignUp />} />

        <Route path="/about" element={<About />} />

        <Route path="/search" element={<Search />} />

        <Route path="/listing/:listingId" element={<Listing />} />

        <Route element={<PrivateRoute />}>

          <Route path="/profile" element={<Profile />} />

          <Route path="/create-listing" element={<CreateListing />} />

          <Route

            path="/update-listing/:listingId"

            element={<UpdateListing />}

          />

        </Route>

      </Routes>

    </BrowserRouter>

  );

};

*export* *default* App;

Home.JSX

*import* { useEffect, useState } *from* 'react';

*import* { Link } *from* 'react-router-dom';

*import* { Swiper, SwiperSlide } *from* 'swiper/react';

*import* { Navigation } *from* 'swiper/modules';

*import* SwiperCore *from* 'swiper';

*import* 'swiper/css/bundle';

*import* ListingItem *from* '../Components/ListingItem'

*export* *default* function Home() {

  const [offerListings, setOfferListings] = useState([]);

  const [saleListings, setSaleListings] = useState([]);

  const [rentListings, setRentListings] = useState([]);

  SwiperCore.use([Navigation]);

  console.log(offerListings);

  useEffect(() => {

    const fetchOfferListings = async () => {

*try* {

        const res = *await* fetch('/api/listing/get?offer=true&limit=4');

        const data = *await* res.json();

        setOfferListings(data);

        fetchRentListings();

      } *catch* (error) {

        console.log(error);

      }

    };

    const fetchRentListings = async () => {

*try* {

        const res = *await* fetch('/api/listing/get?type=rent&limit=4');

        const data = *await* res.json();

        setRentListings(data);

        fetchSaleListings();

      } *catch* (error) {

        console.log(error);

      }

    };

    const fetchSaleListings = async () => {

*try* {

        const res = *await* fetch('/api/listing/get?type=sale&limit=4');

        const data = *await* res.json();

        setSaleListings(data);

      } *catch* (error) {

        log(error);

      }

    };

    fetchOfferListings();

  }, []);

*return* (

    <div>

      {*/\* top \*/*}

      <div className='flex flex-col gap-6 p-28 px-3 max-w-6xl mx-auto'>

        <h1 className='text-slate-700 font-bold text-3xl lg:text-6xl'>

          Find your next <span className='text-slate-500'>perfect</span>

          <br />

          place with ease

        </h1>

        <div className='text-gray-400 text-xs sm:text-sm'>

          Real Estate marketplace is the best place to find your next perfect place to

          live.

          <br />

          We have a wide range of properties for you to choose from.

        </div>

        <Link

          to={'/search'}

          className='text-xs sm:text-sm text-blue-800 font-bold hover:underline'

        >

          Let's get started...

        </Link>

      </div>

      {*/\* swiper \*/*}

      <Swiper navigation>

        {offerListings &&

          offerListings.length > 0 &&

          offerListings.map((*listing*) => (

            <SwiperSlide>

              <div

                style={{

                  background: `url(${listing.imageUrls[0]}) center no-repeat`,

                  backgroundSize: 'cover',

                }}

                className='h-[500px]'

                key={listing.\_id}

              ></div>

            </SwiperSlide>

          ))}

      </Swiper>

      {*/\* listing results for offer, sale and rent \*/*}

      <div className='max-w-6xl mx-auto p-3 flex flex-col gap-8 my-10'>

        {offerListings && offerListings.length > 0 && (

          <div className=''>

            <div className='my-3'>

              <h2 className='text-2xl font-semibold text-slate-600'>Recent offers</h2>

              <Link className='text-sm text-blue-800 hover:underline' to={'/search?offer=true'}>Show more offers</Link>

            </div>

            <div className='flex flex-wrap gap-4'>

              {offerListings.map((*listing*) => (

                <ListingItem listing={listing} key={listing.\_id} />

              ))}

            </div>

          </div>

        )}

        {rentListings && rentListings.length > 0 && (

          <div className=''>

            <div className='my-3'>

              <h2 className='text-2xl font-semibold text-slate-600'>Recent places for rent</h2>

              <Link className='text-sm text-blue-800 hover:underline' to={'/search?type=rent'}>Show more places for rent</Link>

            </div>

            <div className='flex flex-wrap gap-4'>

              {rentListings.map((*listing*) => (

                <ListingItem listing={listing} key={listing.\_id} />

              ))}

            </div>

          </div>

        )}

        {saleListings && saleListings.length > 0 && (

          <div className=''>

            <div className='my-3'>

              <h2 className='text-2xl font-semibold text-slate-600'>Recent places for sale</h2>

              <Link className='text-sm text-blue-800 hover:underline' to={'/search?type=sale'}>Show more places for sale</Link>

            </div>

            <div className='flex flex-wrap gap-4'>

              {saleListings.map((*listing*) => (

                <ListingItem listing={listing} key={listing.\_id} />

              ))}

            </div>

          </div>

        )}

      </div>

    </div>

  );

}

Redux Store

*import* { combineReducers, configureStore } *from* "@reduxjs/toolkit";

*import* userReducer *from* "./User/userSlice";

*import* storage *from* 'redux-persist/lib/storage'

*import* { persistReducer, persistStore } *from* 'redux-persist'

const rootReducer = combineReducers({user: userReducer})

const persistConfig = {

  key:'root',

  storage,

  version:1

}

const persistedReducer = persistReducer(persistConfig,rootReducer)

*export* const store = configureStore({

  reducer: persistedReducer,

  middleware: (*getDefaultMiddleware*) =>

    getDefaultMiddleware({

      serializableCheck: false,

    }),

});

*export* const persistor = persistStore(store)

User Slice

*import* { createSlice } *from* "@reduxjs/toolkit";

const initialState = {

  currentUser: null,

  loading: false,

  error: null,

};

const userSlice = createSlice({

  name: "user",

  initialState,

  reducers: {

    signInStart: (*state*) => {

      state.loading = true;

    },

    signInSuccess: (*state*, *action*) => {

      (state.currentUser = action.payload),

        (state.loading = false),

        (state.error = null);

    },

    signInFailure: (*state*, *action*) => {

      (state.error = action.payload), (state.loading = false);

    },

    updateUserStart: (*state*) => {

      state.loading = true;

    },

    updateUserSuccess: (*state*, *action*) => {

*state*.currentUser = *action*.payload;

*state*.loading = false;

*state*.error = null;

    },

    updateUserFailure: (*state*, *action*) => {

*state*.error = *action*.payload;

*state*.loading = false;

    },

    deleteUserStart: (*state*) => {

*state*.loading = true;

    },

    deleteUserSuccess: (*state*) => {

*state*.currentUser = null;

*state*.loading = false;

*state*.error = null;

    },

    deleteUserFailure: (*state*, *action*) => {

*state*.loading = false;

*state*.error = *action*.payload;

    },

    signOutUserStart: (*state*) => {

*state*.loading = true;

    },

    signOutUserSuccess: (*state*) => {

*state*.currentUser = null;

*state*.loading = false;

*state*.error = null;

    },

    signOutUserFailure: (*state*, *action*) => {

*state*.loading = false;

*state*.error = *action*.payload;

    },

  },

});

*export* const {

  signInStart,

  signInSuccess,

  signInFailure,

  updateUserFailure,

  updateUserStart,

  updateUserSuccess,

  deleteUserFailure,

  deleteUserStart,

  deleteUserSuccess,

  signOutUserFailure,

  signOutUserStart,

  signOutUserSuccess

} = userSlice.actions;

*export* *default* userSlice.reducer;

Profile.jsx

*import* { useSelector } *from* 'react-redux';

*import* { useRef, useState, useEffect } *from* 'react';

*import* {

  getDownloadURL,

  getStorage,

  ref,

  uploadBytesResumable,

} *from* 'firebase/storage';

*import* { app } *from* '../firebase';

*import* {

  updateUserStart,

  updateUserSuccess,

  updateUserFailure,

  deleteUserFailure,

  deleteUserStart,

  deleteUserSuccess,

  signOutUserStart,

} *from* '../redux/User/userSlice';

*import* { useDispatch } *from* 'react-redux';

*import* { Link } *from* 'react-router-dom';

*export* *default* function Profile() {

  const fileRef = useRef(null);

  const { currentUser, loading, error } = useSelector((*state*) => *state*.user);

  const [file, setFile] = useState(undefined);

  const [filePerc, setFilePerc] = useState(0);

  const [fileUploadError, setFileUploadError] = useState(false);

  const [formData, setFormData] = useState({});

  const [updateSuccess, setUpdateSuccess] = useState(false);

  const [showListingsError, setShowListingsError] = useState(false);

  const [userListings, setUserListings] = useState([]);

  const dispatch = useDispatch();

  useEffect(() => {

*if* (file) {

      handleFileUpload(file);

    }

  }, [file]);

  const handleFileUpload = (*file*) => {

    const storage = getStorage(app);

    const fileName = new Date().getTime() + *file*.name;

    const storageRef = ref(storage, fileName);

    const uploadTask = uploadBytesResumable(storageRef, *file*);

    uploadTask.on(

      'state\_changed',

      (*snapshot*) => {

        const progress =

          (*snapshot*.bytesTransferred / *snapshot*.totalBytes) \* 100;

        setFilePerc(Math.round(progress));

      },

      (*error*) => {

        setFileUploadError(true);

      },

      () => {

        getDownloadURL(uploadTask.snapshot.ref).then((*downloadURL*) =>

          setFormData({ ...formData, avatar: *downloadURL* })

        );

      }

    );

  };

  const handleChange = (*e*) => {

    setFormData({ ...formData, [*e*.target.id]: *e*.target.value });

  };

  const handleSubmit = async (*e*) => {

*e*.preventDefault();

*try* {

      dispatch(updateUserStart());

      const res = *await* fetch(`/api/user/update/${currentUser.\_id}`, {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify(formData),

      });

      const data = *await* res.json();

*if* (data.success === false) {

        dispatch(updateUserFailure(data.message));

*return*;

      }

      dispatch(updateUserSuccess(data));

      setUpdateSuccess(true);

    } *catch* (error) {

      dispatch(updateUserFailure(error.message));

    }

  };

  const handleDeleteUser = async () => {

*try* {

      dispatch(deleteUserStart());

      const res = *await* fetch(`/api/user/delete/${currentUser.\_id}`, {

        method: 'DELETE',

      });

      const data = *await* res.json();

*if* (data.success === false) {

        dispatch(deleteUserFailure(data.message));

*return*;

      }

      dispatch(deleteUserSuccess(data));

    } *catch* (error) {

      dispatch(deleteUserFailure(error.message));

    }

  };

  const handleSignOut = async () => {

*try* {

      dispatch(signOutUserStart());

      const res = *await* fetch('/api/auth/signout');

      const data = *await* res.json();

*if* (data.success === false) {

        dispatch(deleteUserFailure(data.message));

*return*;

      }

      dispatch(deleteUserSuccess(data));

    } *catch* (error) {

      dispatch(deleteUserFailure(data.message));

    }

  };

  const handleShowListings = async () => {

*try* {

      setShowListingsError(false);

      const res = *await* fetch(`/api/user/listings/${currentUser.\_id}`);

      const data = *await* res.json();

*if* (data.success === false) {

        setShowListingsError(true);

*return*;

      }

      setUserListings(data);

    } *catch* (error) {

      setShowListingsError(true);

    }

  };

  const handleListingDelete = async (*listingId*) => {

*try* {

      const res = *await* fetch(`/api/listing/delete/${listingId}`, {

        method: 'DELETE',

      });

      const data = *await* res.json();

*if* (data.success === false) {

        console.log(data.message);

*return*;

      }

      setUserListings((*prev*) =>

        prev.filter((*listing*) => listing.\_id !== listingId)

      );

    } *catch* (error) {

      console.log(error.message);

    }

  };

*return* (

    <div className='p-3 max-w-lg mx-auto'>

      <h1 className='text-3xl font-semibold text-center my-7'>Profile</h1>

      <form onSubmit={handleSubmit} className='flex flex-col gap-4'>

        <input

          onChange={(*e*) => setFile(e.target.files[0])}

          type='file'

          ref={fileRef}

          hidden

          accept='image/\*'

        />

        <img

          onClick={() => fileRef.current.click()}

          src={formData.avatar || currentUser.avatar}

          alt='profile'

          className='rounded-full h-24 w-24 object-cover cursor-pointer self-center mt-2'

        />

        <p className='text-sm self-center'>

          {fileUploadError ? (

            <span className='text-red-700'>

              Error Image upload (image must be less than 2 mb)

            </span>

          ) : filePerc > 0 && filePerc < 100 ? (

            <span className='text-slate-700'>{`Uploading ${filePerc}%`}</span>

          ) : filePerc === 100 ? (

            <span className='text-green-700'>Image successfully uploaded!</span>

          ) : (

            ''

          )}

        </p>

        <input

          type='text'

          placeholder='username'

          defaultValue={currentUser.username}

          id='username'

          className='border p-3 rounded-lg'

          onChange={handleChange}

        />

        <input

          type='email'

          placeholder='email'

          id='email'

          defaultValue={currentUser.email}

          className='border p-3 rounded-lg'

          onChange={handleChange}

        />

        <input

          type='password'

          placeholder='password'

          onChange={handleChange}

          id='password'

          className='border p-3 rounded-lg'

        />

        <button

          disabled={loading}

          className='bg-slate-700 text-white rounded-lg p-3 uppercase hover:opacity-95 disabled:opacity-80'

        >

          {loading ? 'Loading...' : 'Update'}

        </button>

        <Link

          className='bg-green-700 text-white p-3 rounded-lg uppercase text-center hover:opacity-95'

          to={'/create-listing'}

        >

          Create Listing

        </Link>

      </form>

      <div className='flex justify-between mt-5'>

        <span

          onClick={handleDeleteUser}

          className='text-red-700 cursor-pointer'

        >

          Delete account

        </span>

        <span onClick={handleSignOut} className='text-red-700 cursor-pointer'>

          Sign out

        </span>

      </div>

      <p className='text-red-700 mt-5'>{error ? error : ''}</p>

      <p className='text-green-700 mt-5'>

        {updateSuccess ? 'User is updated successfully!' : ''}

      </p>

      <button onClick={handleShowListings} className='text-green-700 w-full'>

        Show Listings

      </button>

      <p className='text-red-700 mt-5'>

        {showListingsError ? 'Error showing listings' : ''}

      </p>

      {userListings && userListings.length > 0 && (

        <div className='flex flex-col gap-4'>

          <h1 className='text-center mt-7 text-2xl font-semibold'>

            Your Listings

          </h1>

          {userListings.map((*listing*) => (

            <div

              key={listing.\_id}

              className='border rounded-lg p-3 flex justify-between items-center gap-4'

            >

              <Link to={`/listing/${listing.\_id}`}>

                <img

                  src={listing.imageUrls[0]}

                  alt='listing cover'

                  className='h-16 w-16 object-contain'

                />

              </Link>

              <Link

                className='text-slate-700 font-semibold  hover:underline truncate flex-1'

                to={`/listing/${listing.\_id}`}

              >

                <p>{listing.name}</p>

              </Link>

              <div className='flex flex-col item-center'>

                <button

                  onClick={() => handleListingDelete(listing.\_id)}

                  className='text-red-700 uppercase'

                >

                  Delete

                </button>

                <Link to={`/update-listing/${listing.\_id}`}>

                  <button className='text-green-700 uppercase'>Edit</button>

                </Link>

              </div>

            </div>

          ))}

        </div>

      )}

    </div>

  );

}

Back End Code

Index.js

*import* express *from* "express";

*import* mongoose *from* "mongoose";

*import* dotenv *from* "dotenv";

*import* userRoute *from* './routes/userRoute.js'

*import* authRoute *from* './routes/authRoute.js'

*import* listingRoute *from* './routes/listingRoute.js'

*import* cookieParser *from* "cookie-parser";

*import* path *from* 'path'

dotenv.config();

mongoose

.connect(process.env.MONGO\_URI)

.then(() => {

  console.log("Connected to DB");

})

.catch((*err*) => {

  console.log(*err*);

});

const \_\_dirname = path.resolve()

const app = express();

app.use(express.json())

app.use(cookieParser())

app.listen(3000, () => {

  console.log("Server is running on port 3000");

});

app.use('/api/user',userRoute)

app.use('/api/auth',authRoute)

app.use('/api/listing',listingRoute)

app.use(express.static(path.join(\_\_dirname,'/client/dist')))

app.get('\*',(*req*,*res*)=>{

*res*.sendFile(path.join(\_\_dirname,'client','dist','index.html'))

})

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

  const statusCode = *err*.statusCode || 500

  const message = *err*.message || 'Internal Server Error'

*return* *res*.status(statusCode).json({

    success:false,

    statusCode,

    message,

  })

})

Authentication API:

authRoute.js

*import* express *from* 'express'

*import* { google, signIn, signOut, signUp } *from* '../controllers/authController.js'

const router = express.Router()

router.post("/signup",signUp)

router.post("/signin",signIn)

router.post("/google",google)

router.get('/signout',signOut)

*export* *default* router

authController.js

*import* User *from* "../models/User.js";

*import* bcryptjs *from* "bcryptjs";

*import* { errorHandler } *from* "../utils/error.js";

*import* jwt *from* "jsonwebtoken";

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

  const { username, email, password } = *req*.body;

  const hashedPassword = bcryptjs.hashSync(password, 10);

  const newUser = new User({ username, email, password: hashedPassword });

*try* {

*await* newUser.save();

*res*.status(201).json("user Created Successfully");

  } *catch* (error) {

    next(errorHandler(550, error.message));

  }

};

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

  const { email, password } = *req*.body;

*try* {

    const validUser = *await* User.findOne({ email });

*if* (!validUser) *return* next(errorHandler(404, "User Not found"));

    const validPassword = bcryptjs.compareSync(password, validUser.password);

*if* (!validPassword) *return* next(errorHandler(401, "wrong credentials"));

    const token = jwt.sign({ id: validUser.\_id }, process.env.JWT\_SECRET);

    const { password: pass, ...rest } = validUser.\_doc;

*res*

      .cookie("access\_token", token, { httpOnly: true })

      .status(200)

      .json(rest);

  } *catch* (error) {

    next(error);

  }

};

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

*try* {

    const user = *await* User.findOne({ email :*req*.body.email});

*if* (user) {

      const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET);

      const { password: pass, ...rest } = user.\_doc;

*res*

        .cookie("access\_token", token, { httpOnly: true })

        .status(200)

        .json(rest);

    } *else* {

      const generatedPassword = Math.random().toString(36).slice(-8);

      const hashedPassword = bcryptjs.hashSync(generatedPassword, 10);

      const newUser = new User({

        username:

*req*.body.name.split(" ").join("").toLowerCase() +

          Math.random().toString(36).slice(-4),

        email: *req*.body.email,

        password: hashedPassword,

        avatar: *req*.body.photo,

      });

*await* newUser.save();

      const token = jwt.sign(

        {

          id: newUser.\_id,

        },

        process.env.JWT\_SECRET

      );

      const { password: pass, ...rest } = newUser.\_doc;

      res

        .cookie("access\_token", token, { httpOnly: true })

        .status(200)

        .json(rest);

    }

  } *catch* (error) {

    next(error);

  }

};

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

*try*{

    res.clearCookie('access\_token')

    res.status(200).json("User has been logged out")

  }*catch*(err){

    next(err);

  }

}

User Profile API

userRoute.js

*import* express *from* 'express'

*import* { deleteUser, getUser, getUserListings, test, updateUser } *from* '../controllers/userController.js';

*import* { verifyToken } *from* '../utils/verifyUser.js';

const router = express.Router();

router.get('/test',test);

router.post('/update/:id',verifyToken,updateUser)

router.delete('/delete/:id',verifyToken,deleteUser)

router.get('/listings/:id',verifyToken,getUserListings)

router.get('/:id', verifyToken, getUser)

*export* *default* router;

userController.js

*import* Listing *from* "../models/Listing.js";

*import* User *from* "../models/User.js";

*import* { errorHandler } *from* "../utils/error.js";

*import* bcryptjs *from* "bcryptjs";

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

*res*.json({ message: "hello world" });

};

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

*if* (*req*.user.id !== *req*.params.id)

*return* next(errorHandler(401, "You can only update your own account"));

*try* {

*if* (*req*.body.password) {

*req*.body.password = bcryptjs.hashSync(*req*.body.password, 10);

    }

    const updateUser = *await* User.findByIdAndUpdate(

*req*.params.id,

      {

        $set: {

          username: *req*.body.username,

          email: *req*.body.email,

          password: *req*.body.password,

          avatar: *req*.body.avatar,

        },

      },

      { new: true }

    );

    const { password, ...rest } = updateUser.\_doc;

*res*.status(200).json(rest);

  } *catch* (err) {

    next(err);

  }

};

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

*if* (*req*.user.id !== *req*.params.id)

*return* next(errorHandler(401, "you can only delete your own account"));

*try* {

*await* User.findByIdAndDelete(*req*.params.id);

*res*.clearCookie("access\_token");

*res*.status(200).json("User has been Deleted");

  } *catch* (error) {

    next(error);

  }

};

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

*if* (*req*.user.id === *req*.params.id) {

*try* {

      const listings = *await* Listing.find({userRef:*req*.params.id})

*res*.status(200).json(listings)

    } *catch* (error) {

      next(error);

    }

  } *else* {

*return* next(errorHandler(401, "You can only view your own listings"));

  }

};

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

*try* {

    const user = *await* User.findById(*req*.params.id);

*if* (!user) *return* next(errorHandler(404, 'User not found!'));

    const { password: pass, ...rest } = user.\_doc;

*res*.status(200).json(rest);

  } *catch* (error) {

    next(error);

  }

};

Listing API

listingRoute.js

*import* express *from* 'express'

*import* { createListing, deleteListing, getListing, getListings, updateListing } *from* '../controllers/listingController.js'

*import* {verifyToken }*from* '../utils/verifyUser.js'

const router = express.Router()

router.post('/create', verifyToken, createListing);

router.delete('/delete/:id', verifyToken, deleteListing);

router.post('/update/:id', verifyToken, updateListing);

router.get('/get/:id', getListing);

router.get('/get', getListings);

*export* *default* router

ListingController.js

*import* Listing *from* '../models/Listing.js';

*import* { errorHandler } *from* '../utils/error.js';

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

*try* {

    const listing = *await* Listing.create(*req*.body);

*return* *res*.status(201).json(listing);

  } *catch* (error) {

    next(error);

  }

};

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

  const listing = *await* Listing.findById(*req*.params.id);

*if* (!listing) {

*return* next(errorHandler(404, 'Listing not found!'));

  }

*if* (*req*.user.id !== listing.userRef) {

*return* next(errorHandler(401, 'You can only delete your own listings!'));

  }

*try* {

*await* Listing.findByIdAndDelete(*req*.params.id);

*res*.status(200).json('Listing has been deleted!');

  } *catch* (error) {

    next(error);

  }

};

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

  const listing = *await* Listing.findById(*req*.params.id);

*if* (!listing) {

*return* next(errorHandler(404, 'Listing not found!'));

  }

*if* (*req*.user.id !== listing.userRef) {

*return* next(errorHandler(401, 'You can only update your own listings!'));

  }

*try* {

    const updatedListing = *await* Listing.findByIdAndUpdate(

*req*.params.id,

*req*.body,

      { new: true }

    );

*res*.status(200).json(updatedListing);

  } *catch* (error) {

    next(error);

  }

};

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

*try* {

    const listing = *await* Listing.findById(*req*.params.id);

*if* (!listing) {

*return* next(errorHandler(404, 'Listing not found!'));

    }

*res*.status(200).json(listing);

  } *catch* (error) {

    next(error);

  }

};

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

*try* {

    const limit = parseInt(*req*.query.limit) || 9;

    const startIndex = parseInt(*req*.query.startIndex) || 0;

    let offer = *req*.query.offer;

*if* (offer === undefined || offer === 'false') {

      offer = { $in: [false, true] };

    }

    let furnished = *req*.query.furnished;

*if* (furnished === undefined || furnished === 'false') {

      furnished = { $in: [false, true] };

    }

    let parking = *req*.query.parking;

*if* (parking === undefined || parking === 'false') {

      parking = { $in: [false, true] };

    }

    let type = *req*.query.type;

*if* (type === undefined || type === 'all') {

      type = { $in: ['sale', 'rent'] };

    }

    const searchTerm = *req*.query.searchTerm || '';

    const sort = *req*.query.sort || 'createdAt';

    const order = *req*.query.order || 'desc';

    const listings = *await* Listing.find({

      address: { $regex: searchTerm, $options: 'i' },

      offer,

      furnished,

      parking,

      type,

    })

      .sort({ [sort]: order })

      .limit(limit)

      .skip(startIndex);

*return* *res*.status(200).json(listings);

  } *catch* (error) {

    next(error);

  }

};

Listing.js (Listing Schema)

*import* mongoose *from* 'mongoose';

const listingSchema = new mongoose.Schema(

  {

    name: {

      type: String,

      required: true,

    },

    description: {

      type: String,

      required: true,

    },

    address: {

      type: String,

      required: true,

    },

    regularPrice: {

      type: Number,

      required: true,

    },

    discountPrice: {

      type: Number,

      required: true,

    },

    bathrooms: {

      type: Number,

      required: true,

    },

    bedrooms: {

      type: Number,

      required: true,

    },

    furnished: {

      type: Boolean,

      required: true,

    },

    parking: {

      type: Boolean,

      required: true,

    },

    type: {

      type: String,

      required: true,

    },

    offer: {

      type: Boolean,

      required: true,

    },

    imageUrls: {

      type: Array,

      required: true,

    },

    userRef: {

      type: String,

      required: true,

    },

  },

  { timestamps: true }

);

const Listing = mongoose.model('Listing', listingSchema);

*export* *default* Listing;

User Schema (User.js)

*import* mongoose *from* "mongoose";

const userSchema = new mongoose.Schema(

  {

    username: {

      type: String,

      required: true,

      unique: true,

    },

    email: {

      type: String,

      required: true,

      unique: true,

    },

    password: {

      type: String,

      required: true,

    },

    avatar: {

      type: String,

      default:

        "https://img.freepik.com/premium-vector/account-icon-user-icon-vector-graphics\_292645-552.jpg",

    },

  },

  {

    timestamps: true,

  }

);

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

*export* *default* User;

**8. Conclusion**

In conclusion, the Real Estate Marketplace project stands as a transformative force in property transactions. Combining technical excellence with user-centric design and a robust business model, it aims to redefine the real estate landscape. Meticulous technical evaluation ensures a high-quality codebase, while a focus on user experience positions the platform as intuitive and engaging. The business assessment provides entrepreneurs and investors with key market insights, fostering strategic decision-making. The comparative analysis propels effective marketing strategies. The Real Estate Marketplace isn't just a platform; it's a comprehensive solution poised to reshape real estate into a more accessible, transparent, and efficient ecosystem. Welcome to the future of property transactions.

**9. Result : -**

* Create Listing: The app lets you create listing of the property you want to sell or give on rent.
* Update Listing: The app lets you update listing of the property you want to sell or give on rent.
* Contact Landlords: The app lets you contact the landlord of the property you want to buy or take on rent using in built mail service of your system.

**10.References:**

• MDN Documentation

• Express Documentation

• Mongoose Documentation