Property Finder

Discover your perfect home with ease.

Presented by Wing, Sam, Rorry

Our Idea

Our real estate app makes finding the perfect property easy and stress-free. It simplifies the search process by cutting through the clutter and complexity of other platforms.

With a user-friendly design, our app helps you quickly search and view properties without getting bogged down by irrelevant details. By focusing on essential features and presenting them clearly, we ensure a smooth and efficient property search experience.

Key Features

Visitors

- Can browse the landing page to understand the app's purpose and benefits.
- Can search and view basic property listings, including essential details like price, location, and property type.
- Have the option to sign up to become registered users.

Key Features

Registered Users

- Secure sign-up & sign-in with user authentication
- “My Properties”: Can save and remove shortlisted properties
- “My Listings”: Can create and manage their own property listings if they are agents

Admin Users

- Can monitor and moderate content, including property listings and user accounts through “Manage Users”

Live Demo

https://propertyfinderpro.netlify.app/

Code Demonstration

```
// Debounced search handler
const debouncedSearch = useCallback(
    debounce(() => {
        let filtered = allProperties;

        if (searchQuery) {
            filtered = filtered.filter(
                (property) =>
                    property.address.toLowerCase().includes(searchQuery) ||
                    property.suburb.toLowerCase().includes(searchQuery) ||
                    property.description.toLowerCase().includes(searchQuery)
            );
        }

        if (propertyType !== "all") {
            filtered = filtered.filter((property) => {
                return property.sellOrRent === propertyType;
            });
        }

        if (priceRange !== "all") {
            const [minPrice, maxPrice] = priceRange.split("-").map(Number);
            filtered = filtered.filter((property) => {
                const price = property.price;
                return price >= minPrice && (maxPrice ? price <= maxPrice : true);
            });
        }

        setFilteredProperties(filtered);
    }, 300),
    [allProperties, searchQuery, propertyType, priceRange]
);
```

The debounced search handler filters properties based on search terms, property type, and price range, updating the results with a delay to improve performance.

Code Demonstration

```
const handleFormSubmit = async (event) => {
    event.preventDefault();
    setIsLoading(true);

    try {
        const response = isEditMode ? await handleEditListing() : await handleCreateListing();

        if (!response.ok) {
            const errorData = await response.json();
            throw new Error(errorData.errorMessage || "Operation failed");
        }

        toast.success(`Property ${isEditMode ? "updated" : "added"} successfully!`);
        navigate("/my-listings");
    } catch (error) {
        toast.error(error.message || "An unexpected error occurred");
    } finally {
        setIsLoading(false);
    }
};
```

The **handleFormSubmit** function manages form submissions, handling both creation and editing of listings, and provides success or error feedback based on the outcome.

It also shows a loading state while processing.

Challenges

```
export const convertToBase64 = (file) => {
    return new Promise((resolve, reject) => {
        const fileReader = new FileReader();
        fileReader.readAsDataURL(file);
        fileReader.onload = () => {
            resolve(fileReader.result);
        };
        fileReader.onerror = (error) => {
            reject(error);
        };
    });
};

export const handleFileUpload = async (e, setter) => {
    const file = e.target.files[0];
    const base64 = await convertToBase64(file);
    setter(base64);
};
```

Struggled to work out a feasible way to get property and profile images uploaded and stored.

Looked into some external hosting options, however, those are either expensive or complicated.

Managed to leverage the built-in function to convert the image into base64 format and get them stored in our DB

Challenges

```
export const isTokenExpired = (token) => {
    if (!token) return true; // If there's no token, consider

    try {
        const decodedToken = jwtDecode(token);
        const currentTime = Date.now() / 1000; // Current tim

        return decodedToken.exp < currentTime; // Check if to
    } catch (error) {
        console.error("Failed to decode token:", error);
        return true; // Consider the token expired if there's
    }
};
```

```
    const token = loggedInUser?.token;

    if (token) {
        if (isTokenExpired(token)) {
            toast.error("Your session has expired. Please log in again.");
            setLoggedInUser(null);
            localStorage.removeItem("loggedInUser");
            sessionStorage.removeItem("loggedInUser");
            navigate("/login");
            return;
        }
    } else {
        toast.error("Unable to retrieve your session. Please log in again.");
        navigate("/login");
    }
}
```

There were some random errors while firing the api calls to the backend, and we figured out that it was due to the expired token. Hence, we decided to write a helper function to validate the token on every page transition

The End
Thank you!