CODE

Since it's not possible to include the entire codebase within 50 pages, we've highlighted the most important parts, focusing on key functionalities like data fetching, display, user authentication, and the overall lifecycle.

Key Files and Their Purpose:

- *index.html* – Base HTML file that loads the React app.
- *main.jsx* – Entry point; sets up providers and renders the app.
- *routes.jsx* – Manages routing and URL-to-component mapping.
- *search.jsx* – Implements PG search and filtering features.
- *login.jsx* – Handles user sign-in.
- *register.jsx* – Handles new user registration.
- *logout.jsx* – Logs out the user and clears session data.
- *messages.jsx* – Manages real-time chat between users.
- *submit-pg.jsx* – Form for submitting/updating PG listings.
- *room.jsx* – Displays detailed room info with owner contact.

/index.html

The **index.html**, from which it is calling the react app. main.jsx to render stuff in the DOM of index.html.

```
<!doctype html>
<html lang="en">
<head>
    <!-- All meta data tags
    and includes -->
  <title>MessFinder - Find Affordable Rooms for Rent</title>
</head>
<body class="relative bg-white bg-[url('/assets/rooms-cover.png')] bg-cover bg-center bg-no-repeat">
  <div id="root" class="relative z-10"></div>
  <!-- the main.jsx is calling the react to load the app. -->
  <script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

/src/main.jsx

The **main.jsx**, is the entry point of react app. which catch a DOM element called root, in which it renders all the component and perform all its functionalities.

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import './index.css';
import { createBrowserRouter, RouterProvider } from 'react-router-dom';
import { FirebaseProvider } from './context/firebase';
import { routes } from './routes';

const router = createBrowserRouter(routes);

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <FirebaseProvider>
      <RouterProvider router={router} />
    </FirebaseProvider>
  </StrictMode>
);
```

/src/routes.jsx

The **routes.jsx**, file defines all the application routes, mapping each URL path to its corresponding component. This enhances user experience by enabling intuitive navigation and clean, meaningful URLs, while also supporting smooth redirections.

```jsx
// routes.jsx
import { Route, createRoutesFromElements } from "react-router-dom";
// Layout & Error
import Layout from './Layout';
import { ErrorPage } from './components/error/';
// Pages: Home & Search
import { HomeSearch } from './pages/home';
import { OwnerPublicProfile } from './pages/profile';
import { SearchResult, RoomDetails } from './pages/rooms';
// Pages: Auth
import { AuthPage, Login, Logout, Register, ResettPassword } from './pages/auth';
// Pages: Info
import { Info, ReportOwner, Contact, Faqs, About, TermsAndConditions } from './pages/info';
// Dashboard Shared
import { Dashboard, ChatApp, Bookmarks, Profile, Settings, MyPGs, SubmitPG } from './pages/dashboard';

export const routes = createRoutesFromElements(
    <Route path="/" element={<Layout />} ErrorBoundary={ErrorPage}>
        {/* Home */}
        <Route index element={<HomeSearch />} />
        <Route path="search" element={<SearchResult />}>
            <Route path=":query" element={<SearchResult />} />
        </Route>
        <Route path="room/:roomId" element={<RoomDetails />} />
        <Route path="profile/:username" element={<OwnerPublicProfile />} />

        {/* Owner Dashboard */}
        <Route path="owner/" element={<Dashboard />}>
            <Route path="messages" element={<ChatApp />} />
            <Route path="bookmarks" element={<Bookmarks />} />
            <Route path="pgs" element={<MyPGs />} />
            <Route path="submit-pg" element={<SubmitPG />} />
            <Route path="submit-pg/:roomId" element={<SubmitPG />} />
            <Route path="profile" element={<Profile />} />
            <Route path="settings" element={<Settings />} />
            <Route path="logout" element={<Logout />} />
        </Route>

        {/* User Dashboard */}
        <Route path="user/" element={<Dashboard />}>
            <Route path="messages" element={<ChatApp />} />
            <Route path="bookmarks" element={<Bookmarks />} />
            <Route path="profile" element={<Profile />} />
            <Route path="settings" element={<Settings />} />
            <Route path="logout" element={<Logout />} />
        </Route>

        {/* Auth */}
        <Route path="auth/" element={<AuthPage />}>
            <Route path="login" element={<Login />} />
            <Route path="register" element={<Register />} />
            <Route path="forget-password" element={<ResettPassword />} />
            <Route path="reset-password" element={<ResettPassword />} />
            <Route path="logout" element={<Logout />} />
        </Route>

        {/* Info */}
        <Route path="info/" element={<Info />}>
            <Route path="contact" element={<Contact />} />
            <Route path="about" element={<About />} />
            <Route path="faqs" element={<Faqs />} />
            <Route path="terms" element={<TermsAndConditions />} />
            <Route path="report" element={<ReportOwner />} />
        </Route>

        {/* Catch-All */}
        <Route path="*" element={<ErrorPage />} />
    </Route>
);
```

/src/pages/rooms/search.jsx

The **search.jsx**, file get query from url or in page search and filters field. And fetch room details from database, with the given query and filters. And show them to user.

```
import { useEffect, useState, useRef } from "react";
import { useParams, useLocation, Link } from 'react-router-dom';
import { useFirebase } from "../../context/firebase";
import { userRTB, roomsRTB } from "../../context/firebase-rtb";
import { formatRelativeTime } from "../../module/js/relative-time";
import { Loader } from "../../components/ui";

import SearchBar from "./SearchBar";
import SearchResultCard from "./SearchResultCard";
import { filterRoomsByCriteria } from "./filter-rooms";

const SearchResult = () => {
    const [filters, setFilters] = useState({
        keyword: "",
        accommodationFor: "",
        suitableFor: "",
        maxPrice: "",
        shared: "",
        sortBy: 0,
    });

    const [locationFilter, setLocationFilter] = useState({
        state: "",
        dist: "",
        pin: ""
    });

    const [results, setResults] = useState([]);
    const [allRooms, setAllRooms] = useState([]); // all room data fetched once
    const [loading, setLoading] = useState(true);
    const [visibleCount, setVisibleCount] = useState(10);
    const [filteredAll, setFilteredAll] = useState([]);

    const params = useParams();
    const location = useLocation();
    const queryFilter = new URLSearchParams(location.search);

    const firebase = useFirebase();
    const { getAllRooms } = roomsRTB(firebase);
    const { getData } = userRTB(firebase);

    // Fetch room data only ONCE on URL change
    useEffect(() => {
        const fetchRooms = async () => {
            setLoading(true);
            setFilters(prev => ({ ...prev, keyword: params.query || "" }));
            setLocationFilter({
                state: queryFilter.get("state") || "",
                dist: queryFilter.get("dist") || "",
                pin: queryFilter.get("pin") || "",
            });
            setFilters((prev) => ({ ...prev, accommodationFor: queryFilter.get("accommodationFor") }));

            try {
                const res = await getAllRooms();
                if (!res) throw new Error("No data found");

                const roomEntries = Object.entries(res);
                const fetchedRooms = await Promise.all(
                    roomEntries.map(async ([roomId, room]) => {
                        try {
                            if (room.status !== "public") {
                                return null; // Exclude non-public rooms
                            }
                            const owner = await getData(room.ownerId);
                            return {
                                roomId,
                                thumbnail: room.images?.[0]?.preview || "",
                                href: `/room/${roomId}`,
                                name: room.name,
                                location: room.location,
                                accommodationFor: room.accommodationFor,
                                suitableFor: room.suitableFor,
                                shared: room.shared,
                                price: room.price,
```

```
                            owner: owner?.displayName || "Owner",
                            createdAt: formatRelativeTime(room.createdAt),
                            state: room.state || "",
                            district: room.district || "",
                            pincode: room.pincode || ""
                        };
                    } catch (err) {
                        console.error("Owner fetch error:", err);
                        return null;
                    }
                })
            );

            const validRooms = fetchedRooms.filter(r => r !== null); // Filters out non-public and failed fetches
            setAllRooms(validRooms);
        } catch (err) {
            console.error("Room fetch error:", err);
            setAllRooms([]);
        } finally {
            setLoading(false);
        }
    };

    fetchRooms();
}, [params.query, location.search]); // triggers on page load / URL change


// Filter only when filters change or rooms are loaded
useEffect(() => {
    const filtered = filterRoomsByCriteria(allRooms, filters, locationFilter);
    setFilteredAll(filtered);
    setResults(filtered.slice(0, 8));
    setVisibleCount(8);
}, [filters, locationFilter, allRooms]);

const handleSearch = () => {
    const filtered = filterRoomsByCriteria(allRooms, filters, locationFilter);
    setFilteredAll(filtered);
    setResults(filtered.slice(0, 8));
    setVisibleCount(8);
};

const handleLoadMore = () => {
    const nextCount = visibleCount + 8;
    const moreResults = filteredAll.slice(0, nextCount);
    setResults(moreResults);
    setVisibleCount(nextCount);
};

const allLoaded = results.length >= filteredAll.length;

if (loading) {
    return (
        <main className="flex flex-col min-h-[calc(100vh-72px)] bg-gray-100 px-4">
            <SearchBar onSearch={handleSearch} filters={filters} setFilters={setFilters} />
            <div className="pt-6 pb-6 flex flex-wrap gap-4 items-center justify-center">
                <Loader text="Searching your query to the database.. please wait.." />
            </div>
        </main>
    );
}

return (
    <main className="flex flex-col min-h-[calc(100vh-72px)] bg-gray-100 px-4 items-center">
        <SearchBar onSearch={handleSearch} filters={filters} setFilters={setFilters} />

        <div className="max-w-[1024px] pt-6 pb-6 flex flex-wrap gap-6 justify-center">
            {results.length > 0 ? (
                results.map((result, index) => (
                    <div key={index} className="w-full max-w-80">
                        <SearchResultCard result={result} />
                    </div>
                ))
            ) : (
                <p className="text-center text-gray-500 text-2xl w-full">No results found.</p>
            )}
        </div>
        <div className="pb-6">
            {results.length > 0 && !allLoaded && (
                <button
                    onClick={handleLoadMore}
```

```
                        className="mt-4 px-4 py-2 bg-blue-500 text-white rounded-md hover:bg-blue-600"
                    >
                        Load More
                    </button>
                )}

                {allLoaded && results.length > 0 && (
                    <p className="mt-4 text-gray-500 text-sm sm:text-lg italic">
                        You've listed all {filteredAll.length} rooms.
                    </p>
                )}
            </div>
        </main>
    );
};

export default SearchResult;
```

/src/pages/auth/login.jsx

The **login.jsx**, Handles user authentication by allowing users to securely log into their accounts using email and password credentials.

```
import { useState, useEffect } from "react";
import { Loader, SetRoleBox } from "../../components/ui";
import { useLocation, useNavigate } from "react-router-dom";
import { signInWithEmailAndPassword } from "firebase/auth";
import { useFirebase } from "../../context/firebase";
import { userRTB } from "../../context/firebase-rtb";
import useGoogleAuth from "../../context/useGoogleAuth";

import LoginForm from "./LoginForm";

const Login = () => {
    const [role, setRole] = useState("");
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [showPassword, setShowPassword] = useState(false);
    const [errorMessage, setErrorMessage] = useState("");
    const [loading, setLoading] = useState(false);

    const firebase = useFirebase();
    const navigate = useNavigate();
    const location = useLocation();

    const { handleGoogleSignIn, authLoading, authError } = useGoogleAuth();

    useEffect(() => {
        if (location.state?.role) setRole(location.state.role);
    }, [location.state]);

    useEffect(() => {
        setErrorMessage(authError || "");
    }, [authError]);

    const handleLogin = async () => {
        setLoading(true);
        setErrorMessage("");
        try {
            const res = await signInWithEmailAndPassword(firebase.auth, email, password);
            const { getData } = userRTB(firebase);
            const userData = await getData(res.user.uid);
            if (userData?.role) {
                navigate(`/${userData.role}/profile`, { state: { from: location } });
            } else {
                setErrorMessage("User role not found. Please contact support.");
            }
        } catch (error) {
            console.error(error);
            setErrorMessage("Invalid email or password. Please try again.");
        }
        setLoading(false);
    };

    const handleGoogleLogin = () => handleGoogleSignIn(role);

    const handleNavigateToRegister = () => {
        navigate("/auth/register", { state: { role } });
    };
```

```
    return (
        <main className="flex flex-col items-center justify-center min-h-[calc(100vh-72px)] px-4">
            {!role ? (
                <SetRoleBox setRole={setRole} />
            ) : (
                loading || authLoading ? (
                    <Loader text="trying to Logging you in. please wait.." />
                ) : (
                    <LoginForm
                        role={role}
                        setRole={setRole}
                        email={email}
                        password={password}
                        showPassword={showPassword}
                        errorMessage={errorMessage}
                        setEmail={setEmail}
                        setPassword={setPassword}
                        setShowPassword={setShowPassword}
                        onLogin={handleLogin}
                        onGoogleLogin={handleGoogleLogin}
                        onNavigateToRegister={handleNavigateToRegister}
                        authLoading={authLoading}
                    />
                )
            )}
        </main>
    );
};

export default Login;
```

/src/pages/auth/register.jsx

The **register.jsx**, Enables new users to create an account by submitting necessary registration details, integrating form validation and Firebase auth.

```
import { useEffect, useState } from "react";
import { useNavigate, useLocation } from "react-router-dom";

import { createUserWithEmailAndPassword } from "firebase/auth";
import { useFirebase } from "../../context/firebase";
import { userRTB } from "../../context/firebase-rtb";
import useGoogleAuth from "../../context/useGoogleAuth";

import RegisterForm from "./RegisterForm";
import SetRoleBox from "./SetRoleBox";

const Register = () => {
    const [role, setRole] = useState("");
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [confirmPassword, setConfirmPassword] = useState("");
    const [showPassword, setShowPassword] = useState(false);
    const [showConfirmPassword, setShowConfirmPassword] = useState(false);
    const [errorMessage, setErrorMessage] = useState("");
    const [loading, setLoading] = useState(false);

    const firebase = useFirebase();
    const navigate = useNavigate();
    const location = useLocation();

    const { handleGoogleSignIn, authLoading, authError } = useGoogleAuth();

    useEffect(() => {
        if (location.state?.role) setRole(location.state.role);
    }, [location.state]);

    useEffect(() => {
        setErrorMessage(authError || "");
    }, [authError]);

    const validateInputs = () => {
        if (!email.trim() || !/\S+@\S+\.\S+/.test(email)) return setError("Invalid Email Address.");
        if (!password) return setError("Password is required.");
        if (password.length < 6) return setError("Password must be at least 6 characters long.");
        if (password !== confirmPassword) return setError("Passwords do not match.");
        setErrorMessage("");
        return true;
    };
```

```
    const setError = (msg) => {
        setErrorMessage(msg);
        return false;
    };
    const handleRegister = async () => {
        setLoading(true);
        setErrorMessage("");
        if (!validateInputs()) return setLoading(false);
        if (!role) return setError("Please select a role (User or Owner) first.") && setLoading(false);

        createUserWithEmailAndPassword(firebase.auth, email, password)
            .then((res) => {
                if (res.user) {
                    const userData = {
                        id: res.user.uid,
                        displayName: res.user.displayName || "",
                        username: "",
                        email: res.user.email,
                        phoneNumber: "",
                        photoURL: res.user.photoURL || "",
                        dob: "",
                        role,
                        about: "",
                    };

                    userRTB(firebase).saveData(userData.id, userData)
                        .then(() => navigate(`/${role}/profile`, { state: { from: location } }))
                        .catch(() => setError("Failed to save user information. Please try again."))
                        .finally(() => setLoading(false));
                } else {
                    setError("Error with user, contact admin..");
                    setLoading(false);
                }
            })
            .catch((error) => {
                if (error.code === "auth/email-already-in-use") {
                    setError("User already exists. Please login.");
                } else {
                    setError("Error registering user. Please try again.");
                }
                setLoading(false);
            });
    };

    const handleGoogleRegister = () => handleGoogleSignIn(role);
    const handleNavigateToLogin = () => navigate('/auth/login', { state: { role } });

    return (
        <main className="flex flex-col items-center justify-center min-h-[calc(100vh-72px)] p-4">
            {!role ? (
                <SetRoleBox setRole={setRole} />
            ) : (
                <RegisterForm
                    role={role} setRole={setRole} email={email} setEmail={setEmail}
                    password={password} setPassword={setPassword}
                    confirmPassword={confirmPassword} setConfirmPassword={setConfirmPassword}
                    showPassword={showPassword} setShowPassword={setShowPassword}
                    showConfirmPassword={showConfirmPassword}
                    setShowConfirmPassword={setShowConfirmPassword}
                    errorMessage={errorMessage} loading={loading}
                    authLoading={authLoading} handleRegister={handleRegister}
                    handleGoogleRegister={handleGoogleRegister} handleNavigateToLogin={handleNavigateToLogin}
                />
            )}
        </main>
    );
};

export default Register;
```

/src/pages/auth/logout.jsx

The **logout.jsx**, Signs out the current user, clears session data, and redirects to the home or login page for secure access control.

```
import { useEffect } from "react";
import { useFirebase } from "../../context/firebase";

const Logout = () => {
    const firebase = useFirebase();
    const handleLogout = () => {
        firebase.auth.signOut();
    };
    useEffect(() => {
        handleLogout();
    }, []);
}

export default Logout;
```

/src/pages/dashboard/messages.jsx

The **messages.jsx**, Implements a real-time chat interface where users can send, receive, and manage messages with other users linked to PG listings.

```
import { useState, useRef, useEffect } from 'react';
import { Link, useLocation, useNavigate } from 'react-router-dom';
import { FaEllipsisV, FaTrash, FaCopy, FaArrowLeft, FaEllipsisH } from 'react-icons/fa';
import { chatRTB, userRTB } from '../../context/firebase-rtb';
import { useFirebase } from '../../context/firebase';

const ChatApp = () => {
    const firebase = useFirebase();
    const chat = chatRTB(firebase);
    const user = userRTB(firebase);
    const location = useLocation();
    const navigate = useNavigate();
    const chatWindowRef = useRef(null);

    const [chats, setChats] = useState([]);
    const [selectedChatId, setSelectedChatId] = useState(null);
    const [newMessage, setNewMessage] = useState('');
    const [showHeaderOptions, setShowHeaderOptions] = useState(false);
    const [popupMenu, setPopupMenu] = useState({ visible: false, x: 0, y: 0, chatId: null, messageIndex: null });
    const [hasAutoSelectedChat, setHasAutoSelectedChat] = useState(false);
    const [currentUserId, setCurrentUserId] = useState(null);

    const selectedChat = chats.find(chat => chat.id === selectedChatId);

    // Get the logged-in user ID
    useEffect(() => {
        const unsubscribe = firebase.auth.onAuthStateChanged(user => {
            setCurrentUserId(user?.uid || null);
        });
        return () => unsubscribe();
    }, []);

    // Fetch user's chats and listen for message updates
    useEffect(() => {
        if (!currentUserId) return;

        (async () => {
            try {
                const userChats = await chat.getUserChats(currentUserId);
                const chatEntries = Object.entries(userChats || {});

                const chatDetails = await Promise.all(chatEntries.map(async ([chatId, chatData]) => {
                    let chatDoc = await chat.getChat(chatId);
                    if (!chatDoc) {
                        await chat.createChat(chatData.ownerId, chatData.userId, chatId);
                        chatDoc = await chat.getChat(chatId);
                    }

                    const otherUserId = chatDoc.ownerId === currentUserId ? chatDoc.userId : chatDoc.ownerId;
                    const otherUser = await user.getData(otherUserId);

                    return {
                        id: chatId,
                        name: otherUser?.displayName || "Unknown",
```

```
                    username: otherUser?.username || "",
                    photoURL: otherUser?.photoURL || "/assets/avatar-default.svg",
                    messages: [],
                    ownerId: chatDoc.ownerId,
                    userId: chatDoc.userId,
                };
            }));

            setChats(chatDetails);

            // Subscribe to messages for each chat
            chatDetails.forEach(chatObj => {
                chat.onChatMessages(chatObj.id, messages => {
                    setChats(prevChats =>
                        prevChats.map(c => c.id === chatObj.id ? { ...c, messages } : c)
                    );
                });
            });
        } catch (error) {
            console.error("Failed to load chats:", error);
        }
    })();
}, [currentUserId]);

// Auto-select chat if chatId is provided via location.state
useEffect(() => {
    if (!hasAutoSelectedChat && location.state?.chatId && chats.length > 0) {
        const matchedChat = chats.find(c => c.id === location.state.chatId);
        if (matchedChat) {
            setSelectedChatId(matchedChat.id);
            setHasAutoSelectedChat(true);
        }
    }
}, [chats, location.state?.chatId, hasAutoSelectedChat]);

// Scroll chat window to bottom when new messages arrive
useEffect(() => {
    if (chatWindowRef.current) {
        chatWindowRef.current.scrollTop = chatWindowRef.current.scrollHeight;
    }
}, [selectedChat?.messages]);

// Hide popup menu on outside click
useEffect(() => {
    const hidePopup = () => {
        if (popupMenu.visible) {
            setPopupMenu({ visible: false, x: 0, y: 0, chatId: null, messageIndex: null });
        }
    };
    window.addEventListener('click', hidePopup);
    return () => window.removeEventListener('click', hidePopup);
}, [popupMenu.visible]);

// Chat handlers
const handleSelectChat = (id) => {
    setSelectedChatId(id);
    setShowHeaderOptions(false);
};

const handleBack = () => {
    setSelectedChatId(null);
    setShowHeaderOptions(false);
    setPopupMenu({ visible: false, x: 0, y: 0, chatId: null, messageIndex: null });
};

const handleSendMessage = async () => {
    if (!newMessage.trim() || !selectedChat) return;

    try {
        const isSenderOwner = selectedChat.ownerId !== currentUserId;
        await chat.sendMessage(selectedChat.id, newMessage.trim(), isSenderOwner);
        setNewMessage('');
    } catch (error) {
        console.error('Failed to send message:', error);
    }
};

const handleFormSubmit = (e) => {
    e.preventDefault();
    handleSendMessage();
};
```

```
    const toggleHeaderOptions = () => setShowHeaderOptions(prev => !prev);

    const handleReportChat = () => {
        if (!selectedChat || !currentUserId) return;
        const otherUserId = selectedChat.ownerId === currentUserId ? selectedChat.userId : selectedChat.ownerId;
        navigate(`/info/report/`, { state: { userId: otherUserId, username: selectedChat.username } });
        setShowHeaderOptions(false);
    };

    const handleMessageClick = (e, chatId, index) => {
        e.preventDefault();
        const chatObj = chats.find(c => c.chatId === chatId);
        const message = chatObj?.messages[index];
        const isCurrentUserOwner = chatObj?.ownerId === currentUserId;

        // Only allow popup for messages sent by the current user
        if ((isCurrentUserOwner && message?.sent) || (!isCurrentUserOwner && !message?.sent)) return;

        const rect = e.currentTarget.getBoundingClientRect();
        setPopupMenu({
            visible: true,
            x: rect.right + window.scrollX,
            y: rect.top + window.scrollY,
            chatId,
            messageIndex: index,
        });
    };

    const handleDeleteMessage = async () => {
        if (!popupMenu.visible) return;
        try {
            await chat.deleteMessage(popupMenu.chatId, popupMenu.messageIndex);
        } catch (error) {
            console.error("Failed to delete message:", error);
        }
        setPopupMenu({ visible: false, x: 0, y: 0, chatId: null, messageIndex: null });
    };

    const handleCopyMessage = () => {
        if (!popupMenu.visible) return;
        const chatObj = chats.find(c => c.id === popupMenu.chatId);
        const message = chatObj?.messages[popupMenu.messageIndex];
        if (message) navigator.clipboard.writeText(message.text);
        setPopupMenu({ visible: false, x: 0, y: 0, chatId: null, messageIndex: null });
    };

    return (
        <div className="flex flex-col h-[calc(100vh-120px)] w-full mx-auto rounded-lg shadow-md bg-white select-none
custom-scrollbar">
            {!selectedChat ? (
                <ChatList chats={chats} onSelectChat={handleSelectChat} />
            ) : (
                <ChatWindow
                    selectedChat={selectedChat}
                    currentUserId={currentUserId}
                    showHeaderOptions={showHeaderOptions}
                    toggleHeaderOptions={toggleHeaderOptions}
                    handleBack={handleBack}
                    handleReportChat={handleReportChat}
                    chatWindowRef={chatWindowRef}
                    handleMessageClick={handleMessageClick}
                    handleFormSubmit={handleFormSubmit}
                    newMessage={newMessage}
                    setNewMessage={setNewMessage}
                    handleDeleteMessage={handleDeleteMessage}
                    handleCopyMessage={handleCopyMessage}
                    popupMenu={popupMenu}
                />
            )}
        </div>
    );
};

export default ChatApp;
```

/src/pages/dashboard/owner/submit-pg.jsx

The **submit-pg.jsx**, Provides a dynamic form for PG owners to add or edit property listings, with support for image upload, validation, and preview.

```
import { useEffect, useState } from "react";
import { useLocation, useParams } from 'react-router-dom';
import { InputField, Dropdown, Alert, Loader } from "../../../components/ui";
import { isAgeAboveLimit, validateIndianPhoneNumber, validateUsername } from "../../../module/js/string";
import { onAuthStateChanged } from "firebase/auth";
import { statesAndDistricts } from '../../../module/js/district-pin';
import { useFirebase } from "../../../context/firebase"
import { roomStorage } from "../../../context/firebase-storage";
import { userRTB, roomsRTB } from "../../../context/firebase-rtb"
import { ImageUpload, AccommodationDetails, MessDetails, FormButtons, DescriptiveDetails } from
'../../../components/owner-form';

const defaultData = {
    name: '', location: '', state: '', district: '', pincode: '', accommodationFor: 'Boys',
    suitableFor: "Students", price: 0, shared: 1,
    messInfo: {
        messType: "Home", totalRooms: 1, totalBeds: 1, totalCRooms: 0,
        totalBathrooms: 1, canteenAvailability: "Near", totalFloors: 1,
    },
    facilities: "", services: "", rules: "", description: "", images: [], status: "draft",
}

const SubmitPG = () => {
    const [formData, setFormData] = useState({ ...defaultData });

    const params = useParams();

    const firebase = useFirebase();
    const location = useLocation();

    const [selectedState, setSelectedState] = useState('');
    const [selectedDistrict, setSelectedDistrict] = useState('');
    const [pincodeError, setPincodeError] = useState("");
    const [uid, setUID] = useState("");
    const [roomId, setRoomId] = useState("");
    const [authReady, setAuthReady] = useState(false);
    const [uploading, setUploading] = useState(false);
    const [loading, setLoading] = useState(true);
    const [updateDone, setUpdateDone] = useState(false);
    const [errorMessage, setErrorMessage] = useState(false);

    useEffect(() => {
        const id = params?.roomId || "";
        setRoomId(id);
        if (!id) {
            setFormData({ ...defaultData });
        }
        setErrorMessage(false)
        setUpdateDone(false)
    }, [params?.roomId, location.pathname]);

    useEffect(() => {
        const unregister = onAuthStateChanged(firebase.auth, async (user) => {
            const { getData } = userRTB(firebase);
            if (!user) return setErrorMessage("User not logged in. Please log in and try again.");

            setUID(user.uid);
            try {
                const res = await getData(user.uid);
                if (!validateIndianPhoneNumber(res.phoneNumber)) return setErrorMessage("Invalid or missing Indian
phone number.");
                if (!user.emailVerified) return setErrorMessage("Email not verified. Please verify your email
address.");
                if (!validateUsername(res.username)) return setErrorMessage("Invalid or missing username.");
                if (!isAgeAboveLimit(res.dob)) return setErrorMessage("You must be over 18 to submit a room.");

                setAuthReady(true);
            } catch {
                setErrorMessage("Error fetching user data. Please try again later.");
            }
        });

        return () => unregister(); // Cleanup on unmount
    }, [firebase.auth]);
```

```
    useEffect(() => {
        if (!authReady || !roomId) {
            setLoading(false);
            return;
        }
        setLoading(true);
        if (uploading) return;

        const fetchRoomDetails = async () => {
            try {
                const user = firebase.auth.currentUser;
                if (!user) return;

                const { getRoom } = roomsRTB(firebase);
                await getRoom(roomId)
                    .then((roomDetails) => {
                        if (roomDetails.ownerId == user.uid) {
                            setFormData(roomDetails);
                        } else {
                            setErrorMessage("You can't edit, someone else's data. please be real, and don't involve in
bad movement..")
                        }
                    }).catch(() => {
                        setErrorMessage("Invalid Room Id, plesae check your room id..")
                    })
            } catch (error) {
                console.log("Error fetching room:", error);
                setErrorMessage("Error fetching room.. Please report this to admin.");
            }
            setLoading(false);
        };

        fetchRoomDetails();
    }, [authReady, roomId, firebase]);

    // Other form handlers, can't show due to length limit of pdf (brevity).

    const { uploadRoomImages } = roomStorage();

    const submitFormData = async () => {
        setLoading(true);
        const validateInputs = verifyInputs(formData);
        if (validateInputs.status) {
            try {
                setUploading(true);
                const { saveRoom } = roomsRTB(firebase);

                let downloadUrls;
                let currentRoomId = roomId;

                if (!currentRoomId) {
                    const result = await saveRoom(formData, null);
                    currentRoomId = result.roomId;
                    setRoomId(currentRoomId); // update state for UI or future calls
                }

                // Now currentRoomId is always valid
                downloadUrls = await uploadRoomImages(currentRoomId, formData.images, (index, progress) => {
                });

                const updatedImages = downloadUrls.map((url) => ({ preview: url }));
                const updatedFormData = { ...formData, images: updatedImages };

                setFormData(updatedFormData);

                // Save updated formData with image URLs
                await saveRoom(updatedFormData, currentRoomId);

                setUpdateDone("You can see your room list, on mypgs..");
                setUploading(false);
            } catch (error) {
                console.error("Failed to save room:", error);
                setErrorMessage("Error saving room. Please try again.");
            }
        } else {
            setErrorMessage(validateInputs.message)
            setTimeout(() => {
                setErrorMessage("")
            }, 3000);
        }
        setLoading(false);
    };
```

```
    const handleSubmit = () => {
        formData.status = "public";
        setFormData((prev) => ({ ...prev, status: "public" }))
        submitFormData();
    };

    const handleDraft = () => {
        formData.status = "draft";
        setFormData((prev) => ({ ...prev, status: "draft" }))
        submitFormData();
    }

    const { deleteRoomImages } = roomStorage();

    const handleDelete = () => {
        const { deleteRoom } = roomsRTB(firebase);
        deleteRoom(roomId, uid)
            .then((res) => {
                if (res.status) {
                    deleteRoomImages(roomId)
                        .then(() => {
                            setUpdateDone("Deleted room successfully, enjoy..")
                        }).catch(() => {
                            setErrorMessage("Error: while deleting room images from storage, please contact admin..")
                        })
                } else {
                    setUpdateDone(res.message)
                }
            }).catch((error) => {
                console.log(error);
                setErrorMessage("Error: while deleting your room, please retry.")
            });
    }

    if (loading) return <Loader text="Loading, Please wait." />;
    if (updateDone) return <Alert type="success" header="Updating Room Successfully." message={updateDone} />;
    if (errorMessage) return <Alert type="error" header="Error Occured, Read below." message={errorMessage} />;

    return (
        <div className="w-full mx-auto p-4 bg-white shadow-md rounded-md">
            <h2 className="text-2xl font-bold mb-4">{roomId ? `Edit Your Room` : "Submit a New PG"}</h2>
            <div className="space-y-4">
                {/* ... All Other Input Fields with with calling handler function */}
                <FormButtons onDelete={roomId ? handleDelete : false} onDraft={handleDraft} onSubmit={handleSubmit} />
            </div>
        </div>
    );
};

export default SubmitPG;
```

/src/pages/rooms/room.jsx

The **room.jsx**, Displays full details of a selected PG room, including images, facilities, pricing, and owner contact options for interested users.

```
import { useState, useEffect } from "react";
import "react-responsive-carousel/lib/styles/carousel.min.css";
import { userRTB, roomsRTB, bookmarksRTB, chatRTB } from "../../context/firebase-rtb";
import { useParams } from "react-router-dom";
import { useFirebase } from "../../context/firebase";
import { Alert, Loader } from "../../components/ui"
import ImageCarousel from './ImageCarousel';
import RoomDetailsCard from './RoomDetailsCard';
import ContactButtons from './ContactButtons';
import ContactForm from './ContactForm';

// Main RoomDetails Component
const RoomDetails = () => {
    const [roomInfo, setRoomInfo] = useState({});
    const [ownerInfo, setOwnerInfo] = useState({})

    const params = useParams();

    const [errorMessage, setErrorMessage] = useState(false);
    const [loading, setLoading] = useState(true);
    const firebase = useFirebase();
    const { getRoom } = roomsRTB(firebase);
    const { getData } = userRTB(firebase);
```

```
    const [user, setUser] = useState(false);

    useEffect(() => {
        const roomId = params?.roomId;
        if (!roomId) {
            setErrorMessage("Invalid room ID.");
            return;
        }

        setLoading(true);
        getRoom(roomId)
            .then((res) => {
                const currentUser = firebase.auth.currentUser;
                setUser(currentUser);
                if (res.status === "draft") {
                    if (!currentUser || res.ownerId !== currentUser.uid) {
                        setErrorMessage("The Room is in draft mode, and you are not authorised to view this.");
                        return;
                    }
                }
                setRoomInfo(res);
            })
            .catch(() => {
                setErrorMessage("Error while fetching Room Details from server.");
            })
            .finally(() => {
                setLoading(false);
            });
    }, [firebase, params.roomId]);

    useEffect(() => {
        if (roomInfo.ownerId) {
            getData(roomInfo.ownerId)
                .then((owner) => {
                    setOwnerInfo({
                        id: owner.id, name: owner.displayName,
                        username: owner.username, photoURL: owner.photoURL,
                        phoneNumber: owner.phoneNumber, email: owner.email,
                    });
                });
        }
    }, [roomInfo.ownerId]);

    return (
        <div className="bg-gray-100 w-full min-h-[calc(100vh-80px)] mx-auto h-full px-4 sm:px-6 py-6 justify-center
flex">
            {errorMessage ? (
                <Alert type="error" message={errorMessage} />
            ) : loading ? (
                <Loader />
            ) : (
                <div className="max-w-[1080px] w-full flex flex-col sm:flex-row gap-6 justify-center">
                    {/* Left Section: Room Details */}
                    <div className="flex-1 sm:flex-[4] lg:flex-[6] w-full">
                        {/* Image Carousel */}
                        <div>
                            <ImageCarousel images={roomInfo.images} />
                        </div>

                        {/* Room Details */}
                        <RoomDetailsCard roomId={params.roomId} roomInfo={roomInfo} ownerInfo={ownerInfo} />
                    </div>

                    {/* Right Section: Contact Details */}
                    <div className="flex-1 sm:flex-2">
                        <ContactButtons ownerInfo={ownerInfo} />
                        <ContactForm ownerInfo={ownerInfo} userInfo={user} />
                    </div>
                </div>
            )}
        </div>
    );
};

export default RoomDetails;
```

BACKEND AND DATABASE VIA FIREBASE

/src/context/firebase-config.js

The **firebase-config.jsx**, It defines environment-specific Firebase project configuration using secure import.meta.env variables. This separates sensitive credentials from code and enables flexible deployment across environments (development, staging, production).

```
export const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  databaseURL: import.meta.env.VITE_FIREBASE_DATABASE_URL,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID,
  measurementId: import.meta.env.VITE_FIREBASE_MEASUREMENT_ID,
};
```

/src/context/firebase.jsx

The **firebase.jsx**, Initializes Firebase services (Auth, Realtime Database, Storage) and provides them via React Context (FirebaseContext) for global access throughout the app. It simplifies Firebase integration and promotes modular, reusable access through the useFirebase() hook.

```
import { createContext, useContext } from "react";
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getStorage } from "firebase/storage";
import { getDatabase } from "firebase/database";

import { firebaseConfig } from "./firebase-config"; // Import the firebaseConfig from the separate file

// Initialize Firebase
const firebaseApp = initializeApp(firebaseConfig);
const firebaseAuth = getAuth(firebaseApp);
const firebaseStorage = getStorage(firebaseApp);
const firebaseDB = getDatabase(firebaseApp)

// Create a Firebase context
const FirebaseContext = createContext();

export const FirebaseProvider = ({ children }) => {
    const auth = firebaseAuth;
    const storage = firebaseStorage;
    const db = firebaseDB;
    return (
        <FirebaseContext.Provider value={{
            auth,
            storage,
            db
        }}>
            {children}
        </FirebaseContext.Provider>
    );
};

// Custom hook to use Firebase
export const useFirebase = () => {
    return useContext(FirebaseContext);
};
```

/src/context/firebase-rtb.jsx

The **firebase-rtb.jsx**, Provides helper functions to interact with Firebase Realtime Database for both users and PG room listings. It includes methods to create, update, delete, and fetch data, as well as validate unique usernames and handle authentication-based operations securely.

```
import { ref, push, get, remove, update, serverTimestamp, onValue, set } from "firebase/database";
import { validateUsername } from "../module/js/string";

const userRTB = (firebase) => {
    const saveData = async (id, data) => {
        try {
            if (!id || !(data.email || data.id)) throw new Error("Missing required user information.");
            const dbRef = ref(firebase.db, `/mess-finder/users/${id}`);
            const snapshot = await get(dbRef);
            const exists = snapshot.exists();
            const timestampData = { updatedAt: serverTimestamp(), ...(exists ? {} : { createdAt: serverTimestamp()
}),};
            const newData = { ...data, ...timestampData, };
            await update(dbRef, newData); // use update to preserve existing data and apply server timestamps
            return { status: true, message: exists ? "Data updated." : "User created." };
        } catch (error) {
            throw error;
        }
    };
    const uploadPhoto = async (id, photoURL) => {
        try {
            if (!id || !photoURL) throw new Error("Missing user ID or photo URL.");
            const dbRef = ref(firebase.db, `/mess-finder/users/${id}`);
            const updateData = {
                photoURL: photoURL,
                updatedAt: serverTimestamp(),
            };
            await update(dbRef, updateData);
            return { status: true, message: "Profile photo updated." };
        } catch (error) {
            throw error;
        }
    };
    const deleteData = async (id) => {
        try {
            const dbRef = ref(firebase.db, `/mess-finder/users/${id}`);
            await remove(dbRef);
            return { status: true, message: "Deleted Successfully" };
        } catch (error) {
            throw error; // Throw the error to allow .catch() to handle it
        }
    };
    const getData = async (id) => {
        try {
            const dbRef = ref(firebase.db, `/mess-finder/users/${id}`);
            const snapshot = await get(dbRef);
            if (snapshot.exists())
                return snapshot.val();
            else
                return null;
        } catch (error) {
            throw error; // Throw the error to allow .catch() to handle it
        }
    };
    const getAllUsers = async () => {
        try {
            const dbRef = ref(firebase.db, "/mess-finder/users");
            const snapshot = await get(dbRef);
            return snapshot.exists() ? snapshot.val() : null;
        } catch (error) {
            throw error;
        }
    };
    const userExists = async (username) => {
        const valid = validateUsername(username);
        if (!valid.status) return { valid: false, available: false, reason: valid.message };
        try {
            const allUsersRef = ref(firebase.db, "/mess-finder/users");
            const snapshot = await get(allUsersRef);
            if (!snapshot.exists()) return { valid: true, available: true }; // No users yet, username is available
            const users = snapshot.val();
            for (const id in users) {
                if (users[id].username === username) return { valid: true, available: false, reason: "Username already
taken." };
```

```
            }
            return { valid: true, available: true };
        } catch (error) {
            console.error("Error checking username existence:", error);
            throw error;
        }
    };
    return { saveData, uploadPhoto, deleteData, getData, getAllUsers, userExists };
};
const roomsRTB = (firebase) => {
    const baseRef = ref(firebase.db, "/mess-finder/rooms");
    const saveRoom = async (roomData, roomId = null) => {
        try {
            const currentUser = firebase.auth.currentUser;
            if (!currentUser) throw new Error("User not authenticated.");
            const ownerId = currentUser.uid;
            const timestampData = { updatedAt: serverTimestamp(),
                ...(roomId ? {} : { createdAt: serverTimestamp() }),
            };
            const newData = { ...roomData, ownerId, ...timestampData,};
            let roomRef;
            if (roomId) {
                roomRef = ref(firebase.db, `/mess-finder/rooms/${roomId}`);
                await update(roomRef, newData);
                return { status: true, message: "Room updated.", roomId };
            } else {
                roomRef = push(baseRef); // generate auto ID
                await set(roomRef, newData);
                return { status: true, message: "Room created.", roomId: roomRef.key };
            }
        } catch (error) {
            throw error;
        }
    };

    const deleteRoom = async (roomId, ownerId) => {
        try {
            const roomRef = ref(firebase.db, `/mess-finder/rooms/${roomId}`);
            const snapshot = await get(roomRef);
            if (!snapshot.exists())
                return { status: false, message: "Room not found." };
            const roomData = snapshot.val();
            if (roomData.ownerId !== ownerId)
                return { status: false, message: "Unauthorized: Only the owner can delete the room." };
            await remove(roomRef);
            return { status: true, message: "Room deleted." };
        } catch (error) {
            throw error;
        }
    };


    const getRoom = async (roomId) => {
        try {
            const roomRef = ref(firebase.db, `/mess-finder/rooms/${roomId}`);
            const snapshot = await get(roomRef);
            return snapshot.exists() ? snapshot.val() : null;
        } catch (error) {
            throw error;
        }
    };
    const getAllRooms = async () => {
        try {
            const snapshot = await get(baseRef);
            return snapshot.exists() ? snapshot.val() : {};
        } catch (error) {
            throw error;
        }
    };
    return { saveRoom, deleteRoom, getRoom, getAllRooms };
};

export { userRTB, roomsRTB };
```

/src/context/firebase-storage.jsx

The **firebase-storage.jsx**, Provides utility functions to handle Firebase Storage operations related to user profile images and room listing images. It includes upload and delete functionalities with real-time progress tracking, making it easy to manage image assets for both users and PG rooms.

```
import {
    ref,
    uploadBytesResumable,
    getDownloadURL,
    deleteObject,
    listAll
} from "firebase/storage";
import { useFirebase } from "./firebase";

const userStorage = () => {
    const firebase = useFirebase();

    const uploadProfileImage = async (userId, file, onProgress) => {
        if (!file) {
            throw new Error("No file provided for upload.");
        }

        const storageRef = ref(firebase.storage, `mess-finder/users/${userId}/profile.png`);

        // Create a reference to the file to be uploaded
        const uploadTask = uploadBytesResumable(storageRef, file);

        return new Promise((resolve, reject) => {
            uploadTask.on(
                "state_changed",
                (snapshot) => {
                    // Calculate progress
                    const progress = (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
                    console.log(`Upload is ${progress}% done`);
                    if (onProgress) {
                        onProgress(progress); // Call the progress callback
                    }
                },
                (error) => {
                    // Handle unsuccessful uploads
                    reject(new Error("Upload failed: " + error.message));
                },
                async () => {
                    // Handle successful uploads
                    try {
                        const downloadURL = await getDownloadURL(uploadTask.snapshot.ref);
                        resolve(downloadURL);
                    } catch (error) {
                        reject(new Error("Failed to get download URL: " + error.message));
                    }
                }
            );
        });
    };

    const deleteProfileImage = async (userId) => {
        const storageRef = ref(firebase.storage, `mess-finder/users/${userId}/profile.png`); // Adjust extension if
needed

        return new Promise((resolve, reject) => {
            deleteObject(storageRef)
                .then(() => {
                    resolve("Profile image deleted successfully.");
                })
                .catch((error) => {
                    reject(new Error("Delete failed: " + error.message));
                });
        });
    };

    return {
        uploadProfileImage,
        deleteProfileImage,
    };
};
```

```
const roomStorage = () => {
    const firebase = useFirebase();

    const uploadRoomImages = async (roomId, files, onProgress) => {
        if (!files || files.length === 0) throw new Error("No files provided.");
        if (files.length > 7) throw new Error("Maximum 7 images allowed.");

        const uploadPromises = files.map((file, index) => {
            if (!file.file) {
                // No actual file to upload, use preview URL
                if (onProgress) onProgress(index, 100); // Simulate full progress
                return Promise.resolve(file.preview);
            }

            const filePath = `mess-finder/rooms/${roomId}/image_${index}.jpg`;
            const storageRef = ref(firebase.storage, filePath);
            const uploadTask = uploadBytesResumable(storageRef, file.file);

            return new Promise((resolve, reject) => {
                uploadTask.on(
                    "state_changed",
                    (snapshot) => {
                        const progress = (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
                        if (onProgress) onProgress(index, progress);
                    },
                    (error) => reject(new Error(`Upload failed: ${error.message}`)),
                    async () => {
                        try {
                            const downloadURL = await getDownloadURL(uploadTask.snapshot.ref);
                            resolve(downloadURL);
                        } catch (error) {
                            reject(new Error(`Failed to get URL: ${error.message}`));
                        }
                    }
                );
            });
        });

        return Promise.all(uploadPromises);
    };


    const deleteRoomImages = async (roomId) => {
        const roomFolderRef = ref(firebase.storage, `mess-finder/rooms/${roomId}`);

        try {
            const allItems = await listAll(roomFolderRef);
            const deletePromises = allItems.items.map(itemRef => deleteObject(itemRef));
            await Promise.all(deletePromises);
        } catch (error) {
            throw new Error(`Failed to delete images: ${error.message}`);
        }
    };

    return {
        uploadRoomImages,
        deleteRoomImages,
    };
};

export {
    userStorage,
    roomStorage,
};
```

/src/context/useGoogleAuth.jsx

The **useGoogleAuth.jsx**, Custom React hook for signing in users using Google authentication via Firebase. It manages authentication state, handles user onboarding or redirection based on role, and stores user information in Firebase Realtime Database if not already present.

```
import { useState } from "react";
import { GoogleAuthProvider, signInWithPopup } from "firebase/auth";
import { useFirebase } from "./firebase";
import { userRTB } from "./firebase-rtb";
import { useNavigate, useLocation } from "react-router-dom";

const useGoogleAuth = () => {
    const firebase = useFirebase();
    const navigate = useNavigate();
    const location = useLocation();
    const [authLoading, setAuthLoading] = useState(false);
    const [authError, setAuthError] = useState("");

    const handleGoogleSignIn = async (selectedRole) => {
        setAuthLoading(true);
        setAuthError(""); // Clear previous errors

        if (!selectedRole) {
            setAuthError("A role must be selected before signing in with Google.");
            setAuthLoading(false);
            return;
        }

        const provider = new GoogleAuthProvider();
        try {
            const result = await signInWithPopup(firebase.auth, provider);
            const user = result.user;

            if (user) {
                const { getData, saveData } = userRTB(firebase);

                // Check if user data already exists in Realtime Database
                const existingUserData = await getData(user.uid);

                if (existingUserData && existingUserData.role) {
                    setAuthError("");
                    navigate(`/${existingUserData.role}/profile`, { state: { from: location } });
                } else {
                    const userData = {
                        id: user.uid,
                        displayName: user.displayName || "",
                        username: "", // You might want to prompt for username later
                        email: user.email,
                        phoneNumber: "",
                        photoURL: user.photoURL || "",
                        dob: "",
                        role: selectedRole, // Use the role passed to the hook
                        about: "",
                    };
                    await saveData(userData.id, { ...userData });
                    setAuthError(""); // Clear any previous error
                    navigate(`/${selectedRole}/profile`, { state: { from: location } });
                }
            } else {
                setAuthError("Google sign-in failed: No user information received.");
            }
        } catch (error) {
            console.error("Google sign-in error:", error);
            if (error.code === "auth/popup-closed-by-user") {
                setAuthError("Google sign-in cancelled by user.");
            } else if (error.code === "auth/cancelled-popup-request") {
                setAuthError("Google sign-in request already in progress.");
            } else {
                setAuthError("Google sign-in failed. Please try again.");
            }
        } finally {
            setAuthLoading(false);
        }
    };

    return { handleGoogleSignIn, authLoading, authError };
};

export default useGoogleAuth;
```