You are working on the FarmDirect (formerly FarmHarvest) Replit project, which uses React + Wouter + React-Query on the frontend and Express + Postgres on the backend. We have implemented an `AuthProvider` in `client/src/lib/auth.tsx` (cookie-based, with `signIn()`, `signOut()`, session rehydration, etc.) and a simple stubbed provider in `client/src/lib/auth-simple.tsx`. Currently, the wrong provider is being mounted, causing login state never to “stick” and components like `<Navbar>` to always see `isAuthenticated: false`. Additionally, some protected pages (e.g. `CreateListing.tsx`) call `setLocation()` directly during render, triggering React’s “cannot update during render” warning and interfering with context propagation.

Below is a single, self-contained, copy-and-paste prompt that instructs you (the AI agent) to make all the necessary changes—file by file—so that:

1. Only \*\*one\*\* `AuthProvider` (from `client/src/lib/auth.tsx`) is used throughout the entire app.

2. Every import of `useAuth()` and `AuthProvider` references `client/src/lib/auth.tsx`, never `auth-simple.tsx`.

3. All protected pages (e.g. `CreateListing.tsx`) move their `setLocation("/login")` calls into a `useEffect`, avoiding state updates during render.

4. The root render (`main.tsx`) wraps the entire `<App/>` tree in the correct `AuthProvider`.

5. Components like `<Navbar>` correctly see `auth.user` and `auth.isAuthenticated`.

Please update or create the following files exactly as shown. After applying these edits, run the app: you should see in the console that `<Navbar>` now logs `user: true isAuthenticated: true` once the session check completes, and there should be no “Cannot update a component while rendering” warnings.

---

## 1. Delete or rename the old `auth-simple.tsx`

\*\*File to remove\*\*:

client/src/lib/auth-simple.tsx

(Delete this file entirely. If you prefer not to delete it, rename it to `auth-simple-deprecated.tsx` so nothing imports it by accident.)

---

## 2. `client/src/main.tsx`

Replace any import of `auth-simple` with the real `auth.tsx` provider. The entire file should become:

// client/src/main.tsx

import React from "react";

import { createRoot } from "react-dom/client";

import App from "./App";

import "./index.css";

import { ThemeProvider } from "./components/ThemeProvider";

import { QueryClientProvider } from "@tanstack/react-query";

import { queryClient } from "@/lib/queryClient";

import { AuthProvider } from "@/lib/auth"; // ← use the real auth provider

import { HelmetProvider } from "react-helmet-async";

createRoot(document.getElementById("root")!).render(

<React.StrictMode>

<QueryClientProvider client={queryClient}>

<HelmetProvider>

<ThemeProvider>

<AuthProvider> {/\* Entire app is wrapped here \*/}

<App />

</AuthProvider>

</ThemeProvider>

</HelmetProvider>

</QueryClientProvider>

</React.StrictMode>

);

3. client/src/lib/auth.tsx

Ensure you have the full AuthContext logic here. If you do not yet have this exact content, replace your existing auth.tsx with:

// client/src/lib/auth.tsx

import React, { createContext, useContext, useEffect, useState, ReactNode } from "react";

import { apiRequest } from "./queryClient";

// ----- Types -----

export interface User {

id: number;

name: string | null;

email: string;

image?: string | null;

zip?: string | null;

about?: string | null;

authType: string;

authId: string;

}

interface AuthContextValue {

user: User | null;

token: string | null;

isInitializing: boolean;

isAuthenticated: boolean;

signIn: (email: string, password: string) => Promise<void>;

signOut: () => void;

}

// ----- Context -----

const AuthContext = createContext<AuthContextValue | undefined>(undefined);

export function useAuth() {

const ctx = useContext(AuthContext);

if (!ctx) {

throw new Error("useAuth must be used inside <AuthProvider>");

}

return ctx;

}

// ----- Provider -----

interface AuthProviderProps {

children: ReactNode;

}

export function AuthProvider({ children }: AuthProviderProps) {

const [user, setUser] = useState<User | null>(null);

const [token, setToken] = useState<string | null>(null);

const [isInitializing, setIsInitializing] = useState(true);

// On mount: check localStorage for token, validate session

useEffect(() => {

const savedToken = localStorage.getItem("fh\_token");

if (savedToken) {

setToken(savedToken);

(async () => {

try {

const res = await apiRequest("GET", "/api/auth/session", undefined, savedToken);

if (res.ok) {

const { user: existingUser } = await res.json();

if (existingUser) {

setUser(existingUser);

} else {

// Token invalid → clear

localStorage.removeItem("fh\_token");

setToken(null);

setUser(null);

}

} else {

localStorage.removeItem("fh\_token");

setToken(null);

setUser(null);

}

} catch (err) {

console.error("Error verifying session:", err);

localStorage.removeItem("fh\_token");

setToken(null);

setUser(null);

} finally {

setIsInitializing(false);

}

})();

} else {

setIsInitializing(false);

}

}, []);

// signIn: POST to /api/auth/login, store token + user

async function signIn(email: string, password: string) {

const res = await fetch("/api/auth/login", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ email, password }),

});

if (!res.ok) {

const text = (await res.text()) || res.statusText;

throw new Error(`Login failed: ${text}`);

}

const { token: newToken, user: loggedInUser } = await res.json();

localStorage.setItem("fh\_token", newToken);

setToken(newToken);

setUser(loggedInUser);

}

// signOut: clear token and user

function signOut() {

localStorage.removeItem("fh\_token");

setToken(null);

setUser(null);

}

const value: AuthContextValue = {

user,

token,

isInitializing,

isAuthenticated: !!user,

signIn,

signOut,

};

return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;

}

4. client/src/lib/queryClient.ts

Make sure apiRequest includes the token header. Replace with:

ts

Copy

Edit

// client/src/lib/queryClient.ts

import { QueryClient, QueryFunction } from "@tanstack/react-query";

async function throwIfResNotOk(res: Response) {

if (!res.ok) {

const text = (await res.text()) || res.statusText;

throw new Error(`${res.status}: ${text}`);

}

}

export async function apiRequest(

method: string,

url: string,

data?: unknown,

token?: string

): Promise<Response> {

const headers: Record<string, string> = {

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

};

if (token) {

headers["Authorization"] = `Bearer ${token}`;

}

const res = await fetch(url, {

method,

headers,

body: data ? JSON.stringify(data) : undefined,

});

await throwIfResNotOk(res);

return res;

}

function getQueryFn(): QueryFunction {

return async ({ queryKey }) => {

const [method, url, data] = queryKey as [string, string, any];

const token = localStorage.getItem("fh\_token") || undefined;

const res = await apiRequest(method, url, data, token);

return res.json();

};

}

export const queryClient = new QueryClient({

defaultOptions: {

queries: {

queryFn: getQueryFn(),

refetchOnWindowFocus: false,

staleTime: Infinity,

retry: false,

},

mutations: { retry: false },

},

});

5. Update every useAuth() import

Search for all occurrences of useAuth() in your client/src directory. They must import from @/lib/auth. For example, in client/src/components/Navbar.tsx:

- import { useAuth } from "@/lib/auth-simple"; // ❌ wrong

+ import { useAuth } from "@/lib/auth"; // ✔ correct

export default function Navbar() {

const { user, isAuthenticated, isInitializing, signOut } = useAuth();

// …rest of Navbar code…

}

Do the same in all pages/components:

client/src/pages/CreateListing.tsx

client/src/pages/Dashboard.tsx (if exists)

client/src/pages/SomeOtherPage.tsx

Etc.

6. Refactor CreateListing.tsx (and any other protected page)

Move any setLocation("/login") calls into a useEffect. Replace the old render-phase redirect with this pattern:

// client/src/pages/CreateListing.tsx

import React, { useEffect } from "react";

import { useLocation } from "wouter";

import { useAuth } from "@/lib/auth";

export default function CreateListing() {

const auth = useAuth();

const [\_, setLocation] = useLocation();

// 1) While we’re still verifying the session cookie, show a loading placeholder:

if (auth.isInitializing) {

return <div>Loading…</div>;

}

// 2) Once init is finished, if not authenticated, redirect via useEffect:

useEffect(() => {

if (!auth.isAuthenticated) {

setLocation("/login");

}

}, [auth.isAuthenticated, setLocation]);

// 3) If not authenticated, bail out (router will change path):

if (!auth.isAuthenticated) {

return null;

}

// 4) Authenticated → render the create-listing form:

return (

<div>

<h1>Create New Listing</h1>

{/\* …your existing form JSX here… \*/}

</div>

);

}

If you have other pages with similar logic (e.g. Dashboard.tsx), apply the same pattern:

Check auth.isInitializing, return loading.

In useEffect, redirect if !auth.isAuthenticated.

If still !auth.isAuthenticated, return null.

Otherwise, render protected content.

7. Verify App.tsx routing & hierarchy

Ensure your App.tsx (or wherever routes are declared) looks like this—do not wrap another AuthProvider inside App:

tsx

Copy

Edit

// client/src/App.tsx

import React from "react";

import { Route, Switch } from "wouter";

import Navbar from "@/components/Navbar";

import Home from "@/pages/Home";

import CreateListing from "@/pages/CreateListing";

import SimpleAuth from "@/pages/SimpleAuth";

// …other imports…

export default function App() {

return (

<div className="min-h-screen flex flex-col">

<Navbar />

<main className="flex-1">

<Switch>

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

<Route path="/login" component={SimpleAuth} />

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

{/\* …other routes… \*/}

</Switch>

</main>

</div>

);

}

Note: <Navbar> must be inside the single <AuthProvider> from main.tsx. Do not add or wrap any new provider in App.tsx.

8. Summary of what to hand Replit AI agent

Please implement the above changes in one pass. That means:

Delete client/src/lib/auth-simple.tsx.

Replace client/src/main.tsx exactly with the code in section 2.

Replace client/src/lib/auth.tsx exactly with the code in section 3.

Replace client/src/lib/queryClient.ts exactly with the code in section 4.

Search & replace all imports of useAuth to import from @/lib/auth (fix any existing references to auth-simple.tsx).

Refactor client/src/pages/CreateListing.tsx (and any other protected page) to move setLocation("/login") into a useEffect as shown in section 6.

Verify that App.tsx is not wrapping any extra AuthProvider—only the single provider from main.tsx.

Restart the dev server and confirm:

No more “Cannot update a component while rendering” warnings.

Console logs show Navbar render – user: true isAuthenticated: true after session check.

Protected routes redirect to /login without warnings.

Once signed in, <Navbar> and other components see auth.user and auth.isAuthenticated = true.

Copy the entire content of this prompt, paste it into the Replit AI agent prompt window, and ask it to apply these edits exactly. This will ensure the frontend login state is stored and rehydrated correctly, and all components receive the updated auth context.