Authentication and login for lesson-app -setup

July 11, 2025 log

Today: React Context and auth components, keeping accessibility at the forefront.

**Current State Analysis:**

* Supabase backend configured
* Authentication service layer ready
* Need: UI components for Login/Signup with multi-role support (admin/teacher/student)

**Accessibility-First Auth Design**

For users who can't type, we should implement multiple input methods:

**Voice Input Options:**

* Speech-to-text for form fields
* Voice commands for navigation
* Audio prompts and feedback

**Alternative Input Methods:**

* QR code login (scan with phone/assistive device)
* Biometric authentication (Face ID, Touch ID on mobile)
* Switch navigation support
* Eye-tracking compatibility hooks

**Visual Accessibility:**

* High contrast mode
* Screen reader optimization
* Keyboard-only navigation
* Focus indicators

**Next Implementation Steps**

**Step 1: Auth Context Setup**

* Create ‘AuthContext’ with role-based state management
* Implement session persistence
* Add loading states and error handling

**Step 2: Accessible Login Components**

* Multi-input method login forms
* Voice input integration
* QR code fallback option
* Screen reader announcements

**Step 3: Role-Based UI Components**

* Admin dashboard entry
* Teacher interface
* Student interface
* Progressive enhancement based on capabilities

**Key Questions Before We Start:**

1. **Input Priority**: Which alternative input method should we implement first - voice input, QR codes, or biometric?
2. **Role Flow**: Should role selection happen before or after authentication?
3. **Offline Support**: Do you want offline login capabilities for PWA?
4. **Device Testing**: Do you have access to assistive technology for testing, or should we focus on web standards compliance?

Let's start with Step 1 - the AuthContext. Should I create the context structure first, or would you prefer to discuss the accessibility implementation approach more deeply?

**Next Steps Summary:**

1. Set up AuthContext with role management
2. Create accessible login components with voice/alternative inputs
3. Implement role-based routing
4. Add biometric/QR fallback options
5. Test with screen readers and keyboard navigation
6. Document accessibility features in architecture.md

Let’s begin with the AuthContext implementation:

First create a comprehensive, accessible auth context that handles our multi-role system.

AuthContext.tsx - Multi-Role Authentication Context

import React, { createContext, useContext, useEffect, useState, ReactNode } from 'react'; import { User, Session } from '@supabase/supabase-js'; import { supabase } from '../lib/supabase'; import { authService } from '../services/authService'; // Ty

Ready: a comprehensive **AuthContext** that handles the multi-role system with accessibility built-in. Here's what it includes:

**AuthContext** **Key Features:**

* **Multi-role support**: Admin, Teacher, Student with role-based access control
* **Accessibility-first**: Screen reader announcements, high contrast support, voice settings
* **Session management**: Automatic session persistence and state management
* **Error handling**: Comprehensive error states with user feedback
* **OAuth support**: Google/Apple sign-in ready
* **HOC for role protection**: withRoleAccess component wrapper

**Accessibility Features:**

* Screen reader announcements for all auth state changes
* ARIA live regions for dynamic content
* Keyboard navigation support hooks
* User accessibility preferences storage
* High contrast and font size options

**Next Step Options:**

**A)** Create the Login component with voice input and alternative access methods   
**B)** Set up the role-based routing system  
**C)** Add the AuthProvider to your App.tsx root

Which would you like to implement next?   
I'm thinking we should add the AuthProvider to your app root first, then build the accessible Login component.

Also, check the current authService to ensure compatibility?

Great! Let me first check your current authService to ensure compatibility, then we'll add the AuthProvider to your app root.

favicon https://github.com/rituzangle/lesson-plan-app/blob/main/services/authService.ts

I need to search for your authService structure first to ensure compatibility.

site:github.com rituzangle lesson-plan-app authService

favicon ells-controlcentre-web/app/services/authService.ts at master · habibialkaff/ells-controlcentre-web

favicon GitHub - rolling-scopes/rsschool-app: An application for the RS School education process

favicon GitHub - istio-ecosystem/authservice: Move OIDC token acquisition out of your app code and into the Istio mesh

faviconGitHub - sushidev-team/AngularJS---AuthService

favicon GitHub - goat000/android-lesson-plans: Handouts for the Android classes I taught at Montgomery College, and the code we generated in class

faviconlesson-plans · GitHub Topics · GitHub

favicon spring-boot-security/src/main/java/com/sambi/app/rest/services/AuthService.java at master · Webenius/spring-boot-security

 Authentication and Authorization - Azure App Service | Microsoft Learn

 [Planbook](http://planbook.com/) - The Leader in Lesson Planning

 Quickstart - Add app authentication to a web app - Azure App Service | Microsoft Learn

faviconAuth | chalk.com



OAuth 2.0 for Mobile & Desktop Apps & Google for Developers

favicon GitHub - openid/AppAuth-Android: Android client SDK for communicating with OAuth 2.0 and OpenID Connect providers.

favicon [5 Minute Lesson Plan](5minutelessonplan.co.uk) - Free for 2 weeks!

Authentication & User Management Software – FusionAuth: fusionauth.io

Firebase Authentication by google.com

Proceed with creating a compatible authService structure and then update: App.tsx.

Let me create the authService that matches the AuthContext expectations:

authService.ts - Supabase Authentication Service

import { supabase } from '../lib/supabase'; import { AuthError, User } from '@supabase/supabase-js'; import { UserProfile, UserRole } from '../contexts/AuthContext'; interface SignUpData { email: string; password: string; firstName: string;

Now let's update your App.tsx to include the AuthProvider:

App.tsx - Updated with AuthProvider

import React from 'react'; import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom'; import { AuthProvider, useAuth } from './contexts/AuthContext'; import { ErrorBoundary } from './components/ErrorBoundary'; import { Loadi

Next create the essential supporting components referenced in App.tsx:

Essential Components - ErrorBoundary, LoadingSpinner, Layouts

// components/ErrorBoundary.tsx import React, { Component, ErrorInfo, ReactNode } from 'react'; interface Props { children: ReactNode; fallback?: ReactNode; } interface State { hasError: boolean; error?: Error; } export class ErrorBoundary

Now you have a complete auth system setup. Here's what we've implemented:

**✅ What's Ready:**

1. **AuthContext** - Complete multi-role authentication with accessibility features
2. **AuthService** - Supabase integration with role-based permissions
3. **App.tsx** - Route protection and role-based navigation
4. **Essential Components** - Error handling, loading states, layouts
5. **Accessibility Features** - Screen reader support, keyboard navigation, high contrast

**🚀 Next Steps:**

**Step 1: Create the Database Schema** You'll need these Supabase tables:

sql

*-- User profiles table*

CREATE TABLE user\_profiles (

id UUID REFERENCES auth.users(id) PRIMARY KEY,

email TEXT NOT NULL,

first\_name TEXT,

last\_name TEXT,

role TEXT CHECK (role IN ('admin', 'teacher', 'student')),

school\_id UUID,

class\_ids TEXT[],

accessibility\_settings JSONB,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

**Step 2: Test the Setup**

1. Add the AuthProvider to your existing App.tsx
2. Create placeholder page components that are referenced
3. Test the authentication flow

**🎯 Ready to Implement:**

**A)** The accessible Login component with voice input and alternative authentication methods?   
**B)** The placeholder page components to test the routing?

The Login component will be the most complex since it needs to handle multiple input methods for accessibility.

supabase -schema -all done (<https://github.com/rituzangle/lesson-plan-app/tree/main/suppa>).

Let's create the accessible Login component and the placeholder page components.

First, let's create the placeholder page components to test routing:

Placeholder Page Components

// pages/DashboardPage.tsx

import React from 'react'; import { useAuth } from '../contexts/AuthContext'; export const DashboardPage: React.FC = () => { const { userProfile } = useAuth(); return ( <div className="space-y-6"> <div class

Now let's create the accessible Login component with multiple input methods:

LoginPage.tsx - Accessible Login with Voice & Alternative Inputs

Code