Authentication System Implementation

Overview

The Focus Hub authentication system is built on Supabase Auth with custom user management, role-based access control, and profile management.

Core Components

1. Supabase Client Configuration

File: `src/integrations/supabase/client.ts`

```
import { createClient } from '@supabase/supabase-js';
import type { Database } from './types';

const SUPABASE_URL = "https://hfiltwodcwlqwxrwfjyp.supabase.co";
const SUPABASE_PUBLISHABLE_KEY = "your_anon_key";

export const supabase = createClient<Database>(SUPABASE_URL,
SUPABASE_PUBLISHABLE_KEY);
```

2. Authentication Context

File: `src/contexts/AuthContext.tsx`

Context Interface

```
interface AuthContextType {
  user: User | null;
  session: Session | null;
  profile: any | null;
  userRole: string | null;
  loading: boolean;
  signUp: (email: string, password: string, fullName: string, memberType: string) =>
Promise<{ error: any }>;
  signIn: (email: string, password: string) => Promise<{ error: any }>;
  signOut: () => Promise<void>;
  isAdmin: boolean;
}
```

AuthProvider Implementation

```
export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) =>
{
  const [user, setUser] = useState<User | null>(null);
  const [session, setSession] = useState<Session | null>(null);
  const [profile, setProfile] = useState<any | null>(null);
  const [userRole, setUserRole] = useState<string | null>(null);
  const [loading, setLoading] = useState(true);
  const { toast } = useToast();
  const navigate = useNavigate();
```

```
useEffect(() => {
  // Set up auth state listener
  const { data: { subscription } } = supabase.auth.onAuthStateChange(
    async (event, session) => {
      setSession(session);
      setUser(session?.user ?? null);

      if (session?.user) {
        setTimeout(async () => {
          await fetchUserData(session.user.id);
        }, 0);
      } else {
        setProfile(null);
        setUserRole(null);
      }
      setLoading(false);
    }
  );

  // Check for existing session
  supabase.auth.getSession().then(({ data: { session } }) => {
    setSession(session);
    setUser(session?.user ?? null);
    if (session?.user) {
      fetchUserData(session.user.id);
    }
    setLoading(false);
  });

  return () => subscription.unsubscribe();
}, []);

const fetchUserData = async (userId: string) => {
  try {
    // Fetch profile
    const { data: profileData, error: profileError } = await supabase
      .from('profiles')
      .select('*')
      .eq('id', userId)
      .single();

    if (profileError && profileError.code !== 'PGRST116') {
      console.error('Error fetching profile:', profileError);
    } else {
      // Check for banned or inactive status
      if (profileData?.status === 'banned' || profileData?.status === 'inactive') {
        toast({
          title: 'Account Disabled',
          description: 'Your account has been banned or deactivated.',
          variant: 'destructive',
        });
```

```
          await supabase.auth.signOut();
          setProfile(null);
          setUserRole(null);
          setUser(null);
          setSession(null);
          navigate('/login');
          return;
        }
        setProfile(profileData);
      }

      // Fetch user role
      const { data: roleData, error: roleError } = await supabase
        .from('user_roles')
        .select('role')
        .eq('user_id', userId)
        .single();

      if (roleError && roleError.code !== 'PGRST116') {
        console.error('Error fetching role:', roleError);
      } else {
        setUserRole(roleData?.role || 'user');
      }
    } catch (error) {
      console.error('Error fetching user data:', error);
    }
  };

  const signUp = async (email: string, password: string, fullName: string, memberType:
string) => {
    try {
      const redirectUrl = `${window.location.origin}/app`;

      const { error } = await supabase.auth.signUp({
        email,
        password,
        options: {
          emailRedirectTo: redirectUrl,
          data: {
            full_name: fullName,
            member_type: memberType
          }
        }
      });

      if (error) {
        toast({
          title: "Sign up failed",
          description: error.message,
          variant: "destructive",
        });
      } else {
        toast({
          title: "Sign up successful!",
          description: "Please check your email to confirm your account.",
        });
      }

      return { error };
    } catch (error: any) {
      toast({
        title: "Sign up failed",
        description: error.message,
        variant: "destructive",
      });
      return { error };
    }
  };

  const signIn = async (email: string, password: string) => {
    try {
      const { error } = await supabase.auth.signInWithPassword({
        email,
        password,
      });

      if (error) {
        toast({
          title: "Sign in failed",
          description: error.message,
          variant: "destructive",
        });
      } else {
        toast({
          title: "Welcome back!",
          description: "You have been signed in successfully.",
        });
      }

      return { error };
    } catch (error: any) {
      toast({
        title: "Sign in failed",
        description: error.message,
        variant: "destructive",
      });
      return { error };
    }
  };

  const signOut = async () => {
    try {
      const { error } = await supabase.auth.signOut();
      if (error) {
        toast({
          title: "Sign out failed",
          description: error.message,
          variant: "destructive",
        });
      } else {
        toast({
          title: "Signed out",
          description: "You have been signed out successfully.",
        });
        navigate('/');
      }
    } catch (error: any) {
      toast({
        title: "Sign out failed",
        description: error.message,
        variant: "destructive",
      });
    }
  };

  const isAdmin = userRole === 'admin';

  return (
    <AuthContext.Provider value={{
      user,
      session,
      profile,
      userRole,
      loading,
      signUp,
      signIn,
      signOut,
      isAdmin
    }}>
      {children}
    </AuthContext.Provider>
  );
};
```

3. Protected Route Component

File: `src/components/ProtectedRoute.tsx`

```
import { useAuth } from '@/contexts/AuthContext';
import { Navigate } from 'react-router-dom';

interface ProtectedRouteProps {
  children: React.ReactNode;
  requireAdmin?: boolean;
}

export const ProtectedRoute: React.FC<ProtectedRouteProps> = ({
  children,
  requireAdmin = false
}) => {
  const { user, loading, isAdmin } = useAuth();

  if (loading) {
    return <div>Loading...</div>;
  }

  if (!user) {
    return <Navigate to="/login" replace />;
  }

  if (requireAdmin && !isAdmin) {
    return <Navigate to="/app" replace />;
  }

  return <>{children}</>;
};
```

Authentication Pages

1. Login Page

File: `src/pages/Login.tsx`

```
import { useState } from 'react';
import { useAuth } from '@/contexts/AuthContext';
import { useNavigate, Link } from 'react-router-dom';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
'@/components/ui/card';
import { Label } from '@/components/ui/label';

const Login = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [loading, setLoading] = useState(false);
  const { signIn } = useAuth();
  const navigate = useNavigate();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);

    const { error } = await signIn(email, password);

    if (!error) {
      navigate('/app');
    }
    setLoading(false);
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-background">
      <Card className="w-full max-w-md">
        <CardHeader>
          <CardTitle>Welcome Back</CardTitle>
          <CardDescription>Sign in to your Focus Hub account</CardDescription>
        </CardHeader>
        <CardContent>
          <form onSubmit={handleSubmit} className="space-y-4">
            <div className="space-y-2">
              <Label htmlFor="email">Email</Label>
              <Input
                id="email"
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                required
              />
            </div>
            <div className="space-y-2">
              <Label htmlFor="password">Password</Label>
              <Input
                id="password"
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                required
              />
            </div>
            <Button type="submit" className="w-full" disabled={loading}>
              {loading ? 'Signing in...' : 'Sign In'}
            </Button>
          </form>
          <div className="mt-4 text-center">
            <Link to="/forgot-password" className="text-sm text-muted-foreground
hover:underline">
              Forgot password?
            </Link>
          </div>
          <div className="mt-4 text-center">
            <span className="text-sm text-muted-foreground">
              Don't have an account?{' '}
            </span>
            <Link to="/register" className="text-sm hover:underline">
              Sign up
            </Link>
          </div>
        </CardContent>
      </Card>
    </div>
  );
};

export default Login;
```

2. Register Page

File: `src/pages/Register.tsx`

```
import { useState } from 'react';
import { useAuth } from '@/contexts/AuthContext';
import { Link } from 'react-router-dom';
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from
'@/components/ui/card';
import { Label } from '@/components/ui/label';
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from
'@/components/ui/select';

const Register = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [fullName, setFullName] = useState('');
  const [memberType, setMemberType] = useState('student');
  const [loading, setLoading] = useState(false);
  const { signUp } = useAuth();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);

    const { error } = await signUp(email, password, fullName, memberType);

    if (!error) {
      // User will be redirected to email confirmation
    }

    setLoading(false);
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-background">
      <Card className="w-full max-w-md">
        <CardHeader>
          <CardTitle>Create Account</CardTitle>
          <CardDescription>Join Focus Hub to connect with
professionals</CardDescription>
        </CardHeader>
        <CardContent>
          <form onSubmit={handleSubmit} className="space-y-4">
            <div className="space-y-2">
```

```
              <Label htmlFor="fullName">Full Name</Label>
              <Input
                id="fullName"
                value={fullName}
                onChange={(e) => setFullName(e.target.value)}
                required
              />
            </div>
            <div className="space-y-2">
              <Label htmlFor="email">Email</Label>
              <Input
                id="email"
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                required
              />
            </div>
            <div className="space-y-2">
              <Label htmlFor="memberType">Member Type</Label>
              <Select value={memberType} onValueChange={setMemberType}>
                <SelectTrigger>
                  <SelectValue />
                </SelectTrigger>
                <SelectContent>
                  <SelectItem value="student">Student</SelectItem>
                  <SelectItem value="alumni">Alumni</SelectItem>
                </SelectContent>
              </Select>
            </div>
            <div className="space-y-2">
              <Label htmlFor="password">Password</Label>
              <Input
                id="password"
                type="password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                required
              />
            </div>
            <Button type="submit" className="w-full" disabled={loading}>
              {loading ? 'Creating account...' : 'Create Account'}
            </Button>
          </form>
          <div className="mt-4 text-center">
            <span className="text-sm text-muted-foreground">
              Already have an account?{' '}
            </span>
            <Link to="/login" className="text-sm hover:underline">
              Sign in
            </Link>
          </div>
```

```
        </CardContent>
      </Card>
    </div>
  );
};

export default Register;
```

Role-Based Access Control

1. Admin Check Hook

File: `src/hooks/use-admin.ts`

```
import { useAuth } from '@/contexts/AuthContext';

export const useAdmin = () => {
  const { isAdmin, userRole } = useAuth();

  return {
    isAdmin,
    userRole,
    requireAdmin: (callback: () => void) => {
      if (isAdmin) {
        callback();
      }
    }
  };
};
```

2. Database Role Functions

```
-- Check if user has specific role
CREATE OR REPLACE FUNCTION has_role(_user_id UUID, _role app_role)
RETURNS BOOLEAN AS $$
  SELECT EXISTS (
    SELECT 1
    FROM public.user_roles
    WHERE user_id = _user_id
      AND role = _role
  )
$$ LANGUAGE sql STABLE SECURITY DEFINER;

-- RLS policy for admin-only operations
CREATE POLICY "Admin only operations" ON table_name
  FOR ALL USING (
    has_role(auth.uid(), 'admin')
  );
```

User Profile Management

1. Profile Update Function

```
const updateProfile = async (updates: Partial<Profile>) => {
  const { data, error } = await supabase
    .from('profiles')
    .update(updates)
    .eq('id', user?.id)
    .select()
    .single();

  if (error) {
    toast({
      title: "Update failed",
      description: error.message,
      variant: "destructive",
    });
  } else {
    setProfile(data);
    toast({
      title: "Profile updated",
      description: "Your profile has been updated successfully.",
    });
  }

  return { data, error };
};
```

2. Avatar Upload

```
const uploadAvatar = async (file: File) => {
  const fileExt = file.name.split('.').pop();
  const fileName = `${user?.id}-${Date.now()}.${fileExt}`;
  const filePath = `avatars/${fileName}`;

  const { error: uploadError } = await supabase.storage
    .from('avatars')
    .upload(filePath, file);

  if (uploadError) {
    toast({
      title: "Upload failed",
      description: uploadError.message,
      variant: "destructive",
    });
    return { error: uploadError };
  }

  const { data: { publicUrl } } = supabase.storage
    .from('avatars')
    .getPublicUrl(filePath);
  await updateProfile({ avatar_url: publicUrl });

  return { publicUrl };
};
```

Security Features

1. Account Status Management

- **Active**: Normal user access
- **Inactive**: Account temporarily disabled
- **Banned**: Account permanently disabled

2. Session Management

- Automatic session refresh
- Secure token storage
- Session timeout handling

3. Password Security

- Supabase handles password hashing
- Email confirmation required
- Password reset functionality

Error Handling

1. Authentication Errors

```
const handleAuthError = (error: any) => {
  switch (error.message) {
    case 'Invalid login credentials':
      return 'Invalid email or password';
    case 'Email not confirmed':
      return 'Please check your email and confirm your account';
    case 'Too many requests':
      return 'Too many login attempts. Please try again later';
    default:
      return error.message;
  }
};
```

2. Network Error Handling

```
const handleNetworkError = (error: any) => {
  if (error.code === 'NETWORK_ERROR') {
    toast({
      title: "Connection Error",
      description: "Please check your internet connection and try again.",
      variant: "destructive",
    });
  }
};
```

Testing Authentication

1. Unit Tests

```
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import { AuthProvider } from '@/contexts/AuthContext';
import Login from '@/pages/Login';

test('login form submission', async () => {
  render(
    <AuthProvider>
      <Login />
    </AuthProvider>
  );

  fireEvent.change(screen.getByLabelText(/email/i), {
    target: { value: 'test@example.com' },
  });
  fireEvent.change(screen.getByLabelText(/password/i), {
    target: { value: 'password123' },
  });
  fireEvent.click(screen.getByRole('button', { name: /sign in/i }));

  await waitFor(() => {
    expect(screen.getByText(/signing in/i)).toBeInTheDocument();
  });
});
```

This authentication system provides a robust, secure foundation for user management in the Focus Hub platform.