🎯 RBAC + Authentication Implementation Plan

Phase: Foundation Layer
Timeline: Week 1-2 of January 2026
Status: 🔴 CRITICAL - Must be perfect before V2 launch

📊 COMPLETE ROLE HIERARCHY

All Roles in AllWondrous:

typescript
```
export type UserRole =
  | 'super_admin'       // Platform administrator
  | 'solo_practitioner'  // Independent PT (NEW - was missing!)
  | 'studio_owner'       // Studio owner/operator
  | 'studio_manager'     // Studio operations manager
  | 'trainer'            // Studio employee trainer
  | 'receptionist'       // Front desk/admin
  | 'finance_manager'    // Financial operations
  | 'client';            // End user/client

export type StudioMode =
  | 'solo-pt'            // Solo practitioner (1 person business)
  | 'single-site'        // One location, multiple trainers

  | 'multi-site';        // Multiple locations, multiple trainers
```

🎭 COMPLETE PERMISSIONS MATRIX

Permission Categories:

typescript
```
export const PERMISSION_CATEGORIES = {
  // Client Management
  CLIENTS: {
    VIEW_OWN: 'clients:view:own',
    VIEW_ASSIGNED: 'clients:view:assigned',
    VIEW_STUDIO: 'clients:view:studio',
  VIEW_ALL: 'clients:view:all',
  CREATE: 'clients:create',
  EDIT_OWN: 'clients:edit:own',
  EDIT_ASSIGNED: 'clients:edit:assigned',
  EDIT_ALL: 'clients:edit:all',
  DELETE: 'clients:delete',
  EXPORT: 'clients:export',
  IMPORT: 'clients:import',
},

// Booking/Schedule Management
BOOKINGS: {
  VIEW_OWN: 'bookings:view:own',
  VIEW_ASSIGNED: 'bookings:view:assigned',
  VIEW_STUDIO: 'bookings:view:studio',
  VIEW_ALL: 'bookings:view:all',
  CREATE_OWN: 'bookings:create:own',
  CREATE_FOR_CLIENTS: 'bookings:create:clients',
  CREATE_ANY: 'bookings:create:any',
  EDIT_OWN: 'bookings:edit:own',
  EDIT_ASSIGNED: 'bookings:edit:assigned',
  EDIT_ALL: 'bookings:edit:all',
  CANCEL_OWN: 'bookings:cancel:own',
  CANCEL_ANY: 'bookings:cancel:any',
},

// Schedule/Availability
SCHEDULE: {
  VIEW_OWN: 'schedule:view:own',
  VIEW_TEAM: 'schedule:view:team',
  VIEW_ALL: 'schedule:view:all',
  MANAGE_OWN: 'schedule:manage:own',
  MANAGE_TEAM: 'schedule:manage:team',
  MANAGE_ALL: 'schedule:manage:all',
},

// Services/Sessions
SERVICES: {
  VIEW: 'services:view',
  CREATE: 'services:create',
  EDIT: 'services:edit',
  DELETE: 'services:delete',
  PRICING: 'services:pricing',
},
// Packages/Credits
PACKAGES: {
  VIEW: 'packages:view',
  CREATE: 'packages:create',
  EDIT: 'packages:edit',
  DELETE: 'packages:delete',
  SELL: 'packages:sell',
},

// Financial
FINANCE: {
  VIEW_OWN_EARNINGS: 'finance:view:own',
  VIEW_STUDIO_REVENUE: 'finance:view:studio',
  VIEW_ALL_FINANCIAL: 'finance:view:all',
  PROCESS_PAYMENTS: 'finance:payments:process',
  ISSUE_REFUNDS: 'finance:refunds:issue',
  MANAGE_PRICING: 'finance:pricing:manage',
  EXPORT_REPORTS: 'finance:reports:export',
},

// Team Management
TEAM: {
  VIEW: 'team:view',
  INVITE: 'team:invite',
  EDIT: 'team:edit',
  REMOVE: 'team:remove',
  ASSIGN_ROLES: 'team:roles:assign',
  MANAGE_PERMISSIONS: 'team:permissions:manage',
},

// Settings
SETTINGS: {
  VIEW_STUDIO: 'settings:view:studio',
  EDIT_STUDIO: 'settings:edit:studio',
  VIEW_BILLING: 'settings:view:billing',
  EDIT_BILLING: 'settings:edit:billing',
  MANAGE_INTEGRATIONS: 'settings:integrations:manage',
  MANAGE_FEATURES: 'settings:features:manage',
},

// Locations (Multi-site)
LOCATIONS: {
  VIEW: 'locations:view',
  CREATE: 'locations:create',
  EDIT: 'locations:edit',
  DELETE: 'locations:delete',
  ASSIGN_STAFF: 'locations:staff:assign',
},

// Trainer Aide (Premium Feature)
TRAINER_AIDE: {
  VIEW_TEMPLATES: 'trainer_aide:templates:view',
  CREATE_TEMPLATES: 'trainer_aide:templates:create',
  EDIT_TEMPLATES: 'trainer_aide:templates:edit',
  DELETE_TEMPLATES: 'trainer_aide:templates:delete',
  ASSIGN_TO_CLIENTS: 'trainer_aide:assign:clients',
  VIEW_PROGRESS: 'trainer_aide:progress:view',
  EXPORT_PROGRAMS: 'trainer_aide:programs:export',
},

// Reports/Analytics
REPORTS: {
  VIEW_OWN: 'reports:view:own',
  VIEW_STUDIO: 'reports:view:studio',
  VIEW_ALL: 'reports:view:all',
  EXPORT: 'reports:export',
  SCHEDULE: 'reports:schedule',
},

// Marketing/Communications
MARKETING: {
  VIEW_CAMPAIGNS: 'marketing:campaigns:view',
  CREATE_CAMPAIGNS: 'marketing:campaigns:create',
  SEND_EMAILS: 'marketing:emails:send',
  SEND_SMS: 'marketing:sms:send',
  MANAGE_AUTOMATIONS: 'marketing:automations:manage',
},

// Platform Admin (Super Admin Only)
PLATFORM: {
  VIEW_ALL_STUDIOS: 'platform:studios:view:all',
  EDIT_ANY_STUDIO: 'platform:studios:edit:any',
  LOGIN_AS_USER: 'platform:users:impersonate',
  MANAGE_FEATURES: 'platform:features:manage',
  VIEW_SYSTEM_LOGS: 'platform:logs:view',
  MANAGE_BILLING: 'platform:billing:manage',
},
```

};

🎯 ROLE DEFINITIONS WITH FULL PERMISSIONS

1. Solo Practitioner (NEW - Critical Addition!)

typescript

```
export const SOLO_PRACTITIONER_ROLE = {
  role: 'solo_practitioner',
  displayName: 'Solo Practitioner',
  description: 'Independent personal trainer managing own clients',
  studioMode: 'solo-pt',

  permissions: [
    // Clients - Full control over own clients
    'clients:view:own',
    'clients:view:all',        // All = own clients (no team)
    'clients:create',
    'clients:edit:all',
    'clients:delete',
    'clients:export',
    'clients:import',

    // Bookings - Full control
    'bookings:view:own',
    'bookings:view:all',
    'bookings:create:own',
    'bookings:create:clients', // Book for their clients
    'bookings:edit:own',
    'bookings:edit:all',
    'bookings:cancel:own',
    'bookings:cancel:any',

    // Schedule - Full control
    'schedule:view:own',
    'schedule:view:all',
    'schedule:manage:own',
    'schedule:manage:all',

    // Services - Full control
    'services:view',
    'services:create',
'services:edit',
'services:delete',
'services:pricing',

// Packages - Full control
'packages:view',
'packages:create',
'packages:edit',
'packages:delete',
'packages:sell',

// Finance - Own earnings only
'finance:view:own',
'finance:view:studio',     // Studio = self
'finance:payments:process',
'finance:refunds:issue',
'finance:pricing:manage',
'finance:reports:export',

// Team - NO ACCESS (solo)
// No team permissions

// Settings - Full control
'settings:view:studio',
'settings:edit:studio',
'settings:view:billing',
'settings:edit:billing',
'settings:integrations:manage',
'settings:features:manage',

// Locations - NO ACCESS (solo, one location only)
// No location permissions

// Trainer Aide - FULL ACCESS (if purchased)
'trainer_aide:templates:view',
'trainer_aide:templates:create',
'trainer_aide:templates:edit',
'trainer_aide:templates:delete',
'trainer_aide:assign:clients',
'trainer_aide:progress:view',
'trainer_aide:programs:export',

// Reports - Own data
'reports:view:own',
    'reports:view:studio',     // Studio = self
    'reports:export',
    'reports:schedule',

    // Marketing - Full control
    'marketing:campaigns:view',
    'marketing:campaigns:create',
    'marketing:emails:send',
    'marketing:sms:send',
    'marketing:automations:manage',
  ],

  restrictions: {
    maxClients: null,          // Unlimited
    maxLocations: 1,           // Solo = 1 location
    maxStaff: 1,               // Just themselves
    canInviteTeam: false,
    canManageLocations: false,
    clientVisibility: 'all',   // See all = own clients
  },

  features: {
    teamManagement: false,
    multiLocation: false,
    advancedReporting: true,   // Can still have advanced reports
    trainerAide: 'optional',   // Can purchase add-on
    aiMagicInbox: 'optional',
    crossLocationCredits: false,
  },
};
```

---

2. Studio Owner

typescript

```
export const STUDIO_OWNER_ROLE = {
  role: 'studio_owner',
  displayName: 'Studio Owner',
  description: 'Owner/operator of fitness studio',
  studioMode: 'single-site' | 'multi-site',

  permissions: [
    // Clients - Full access
    'clients:view:all',
'clients:create',
'clients:edit:all',
'clients:delete',
'clients:export',
'clients:import',

// Bookings - Full access
'bookings:view:all',
'bookings:create:any',
'bookings:edit:all',
'bookings:cancel:any',

// Schedule - Full access
'schedule:view:all',
'schedule:manage:all',

// Services - Full access
'services:view',
'services:create',
'services:edit',
'services:delete',
'services:pricing',

// Packages - Full access
'packages:view',
'packages:create',
'packages:edit',
'packages:delete',
'packages:sell',

// Finance - Full access
'finance:view:all',
'finance:payments:process',
'finance:refunds:issue',
'finance:pricing:manage',
'finance:reports:export',

// Team - Full access
'team:view',
'team:invite',
'team:edit',
'team:remove',
'team:roles:assign',
'team:permissions:manage',
    // Settings - Full access
    'settings:view:studio',
    'settings:edit:studio',
    'settings:view:billing',
    'settings:edit:billing',
    'settings:integrations:manage',
    'settings:features:manage',

    // Locations - Full access (if multi-site)
    'locations:view',
    'locations:create',
    'locations:edit',
    'locations:delete',
    'locations:staff:assign',

    // Trainer Aide - Full access
    'trainer_aide:templates:view',
    'trainer_aide:templates:create',
    'trainer_aide:templates:edit',
    'trainer_aide:templates:delete',
    'trainer_aide:assign:clients',
    'trainer_aide:progress:view',
    'trainer_aide:programs:export',

    // Reports - Full access
    'reports:view:all',
    'reports:export',
    'reports:schedule',

    // Marketing - Full access
    'marketing:campaigns:view',
    'marketing:campaigns:create',
    'marketing:emails:send',
    'marketing:sms:send',
    'marketing:automations:manage',
  ],

  restrictions: {
    maxClients: null,
    maxLocations: null,        // Unlimited for multi-site
    maxStaff: null,
    canInviteTeam: true,
    canManageLocations: true,
    clientVisibility: 'all',
  },

  features: {
    teamManagement: true,
    multiLocation: true,       // Available for upgrade
    advancedReporting: true,
    trainerAide: 'optional',
    aiMagicInbox: 'optional',
    crossLocationCredits: true,
  },
};
```

---

3. Studio Manager

typescript

```
export const STUDIO_MANAGER_ROLE = {
  role: 'studio_manager',
  displayName: 'Studio Manager',
  description: 'Operations manager for studio',

  permissions: [
    // Clients - Full access
    'clients:view:studio',
    'clients:create',
    'clients:edit:all',
    'clients:export',

    // Bookings - Full access
    'bookings:view:studio',
    'bookings:create:clients',
    'bookings:edit:all',
    'bookings:cancel:any',

    // Schedule - Team access
    'schedule:view:team',
    'schedule:manage:team',

    // Services - View + edit
    'services:view',
    'services:edit',

    // Packages - Can sell
    'packages:view',
    'packages:sell',

    // Finance - View studio revenue
    'finance:view:studio',
    'finance:payments:process',
    'finance:refunds:issue',
    'finance:reports:export',

    // Team - View only
    'team:view',

    // Settings - View only
    'settings:view:studio',

    // Locations - View + assign
    'locations:view',
    'locations:staff:assign',

    // Trainer Aide - View templates
    'trainer_aide:templates:view',
    'trainer_aide:assign:clients',
    'trainer_aide:progress:view',

    // Reports - Studio level
    'reports:view:studio',
    'reports:export',

    // Marketing - Can execute
    'marketing:campaigns:view',
    'marketing:emails:send',
    'marketing:sms:send',
  ],

  restrictions: {
    canInviteTeam: false,
    canManageLocations: false,
    clientVisibility: 'studio',
  },
};
```

---

4. Trainer (Studio Employee)

```typescript
export const TRAINER_ROLE = {
  role: 'trainer',
  displayName: 'Trainer',
  description: 'Studio employee trainer',

  permissions: [
    // Clients - Assigned only (configurable)
    'clients:view:assigned',   // Can be upgraded to 'studio' by owner
    'clients:edit:assigned',

    // Bookings - Own + assigned
    'bookings:view:assigned',
    'bookings:create:clients', // For their clients
    'bookings:edit:assigned',
    'bookings:cancel:own',

    // Schedule - Own only
    'schedule:view:own',
    'schedule:manage:own',

    // Services - View only
    'services:view',

    // Packages - View only
    'packages:view',

    // Finance - Own earnings
    'finance:view:own',

    // Trainer Aide - Full access (for their clients)
    'trainer_aide:templates:view',
    'trainer_aide:templates:create',
    'trainer_aide:templates:edit',
    'trainer_aide:assign:clients',  // Assigned clients only
    'trainer_aide:progress:view',

    // Reports - Own only
    'reports:view:own',
  ],

  restrictions: {
    clientVisibility: 'assigned', // Configurable by owner
    canInviteTeam: false,
    canManageLocations: false,
  },

  configurableBy: 'studio_owner',
  configurableSettings: {
    clientVisibility: ['assigned', 'studio'],
    canManageBookings: true,       // Owner can toggle
    canViewAllClients: false,      // Owner can enable
  },
};
```

---

5. Receptionist

typescript
```
export const RECEPTIONIST_ROLE = {
  role: 'receptionist',
  displayName: 'Receptionist',
  description: 'Front desk / admin staff',

  permissions: [
    // Clients - Studio level
    'clients:view:studio',
    'clients:create',
    'clients:edit:all',

    // Bookings - Full access
    'bookings:view:studio',
    'bookings:create:any',
    'bookings:edit:all',
    'bookings:cancel:any',

    // Schedule - View all
    'schedule:view:all',

    // Services - View
    'services:view',

    // Packages - Sell
    'packages:view',
    'packages:sell',

    // Finance - Process payments
    'finance:payments:process',
  ],

  restrictions: {
    clientVisibility: 'studio',
    canInviteTeam: false,
    canManageLocations: false,
  },
};
```

---

6. Finance Manager

typescript

```
export const FINANCE_MANAGER_ROLE = {
  role: 'finance_manager',
  displayName: 'Finance Manager',
  description: 'Financial operations manager',

  permissions: [
    // Clients - View for financial purposes
    'clients:view:studio',

    // Finance - Full access
    'finance:view:all',
    'finance:payments:process',
    'finance:refunds:issue',
    'finance:pricing:manage',
    'finance:reports:export',

    // Packages - Pricing
    'packages:view',
    'packages:edit',          // For pricing updates

    // Reports - Financial
    'reports:view:all',
    'reports:export',
    'reports:schedule',

    // Settings - Billing
    'settings:view:billing',
    'settings:edit:billing',
  ],

  restrictions: {
    clientVisibility: 'studio',
    canInviteTeam: false,
  },
};
```

---

7. Client

typescript
```
export const CLIENT_ROLE = {
  role: 'client',
  displayName: 'Client',
  description: 'End user / fitness client',

  permissions: [
    // Own data only
    'clients:view:own',
    'clients:edit:own',

    // Own bookings
    'bookings:view:own',
    'bookings:create:own',     // If self-booking enabled
    'bookings:cancel:own',

    // View services
    'services:view',
    'packages:view',

    // Own schedule
    'schedule:view:own',

    // Own finances
    'finance:view:own',

    // Own progress (Trainer Aide)
    'trainer_aide:progress:view',
  ],

  restrictions: {
    clientVisibility: 'own',
    selfBookingAllowed: true,  // Configurable by studio
  },
};
```

---

8. Super Admin

typescript
```
export const SUPER_ADMIN_ROLE = {
  role: 'super_admin',
  displayName: 'Super Admin',
  description: 'Platform administrator',

  permissions: [
    // ALL permissions from all roles, plus:
    'platform:studios:view:all',
    'platform:studios:edit:any',
    'platform:users:impersonate',
    'platform:features:manage',
    'platform:logs:view',
    'platform:billing:manage',
  ],

  restrictions: {
    // No restrictions
  },
};
```

🗄️ DATABASE SCHEMA

1. Add Role Column to Profiles

sql
```
-- Add role system to profiles
ALTER TABLE profiles
    ADD COLUMN IF NOT EXISTS role TEXT DEFAULT 'client',
    ADD COLUMN IF NOT EXISTS is_super_admin BOOLEAN DEFAULT false,
    ADD COLUMN IF NOT EXISTS role_permissions JSONB DEFAULT '[]'::jsonb,
    ADD COLUMN IF NOT EXISTS custom_permissions JSONB DEFAULT '[]'::jsonb;

-- Constraint: Valid roles
ALTER TABLE profiles
    ADD CONSTRAINT valid_role CHECK (
        role IN (
            'super_admin',
            'solo_practitioner',
            'studio_owner',
            'studio_manager',
            'trainer',
            'receptionist',
            'finance_manager',
            'client'
        )
    );

-- Index for performance

CREATE INDEX idx_profiles_role ON profiles(role);
```

---

2. Add Role to Instructors

sql

```
-- Add to instructors table
ALTER TABLE instructors
    ADD COLUMN IF NOT EXISTS role TEXT DEFAULT 'trainer',
    ADD COLUMN IF NOT EXISTS can_manage_bookings BOOLEAN DEFAULT true,
    ADD COLUMN IF NOT EXISTS can_view_all_clients BOOLEAN DEFAULT false,
    ADD COLUMN IF NOT EXISTS client_visibility TEXT DEFAULT 'assigned',
    ADD COLUMN IF NOT EXISTS accessible_locations UUID[] DEFAULT ARRAY[]::UUID[],
    ADD COLUMN IF NOT EXISTS permissions_override JSONB DEFAULT '{}'::jsonb;

-- Constraint
ALTER TABLE instructors
    ADD CONSTRAINT valid_instructor_role CHECK (
        role IN (
            'solo_practitioner',
            'studio_owner',
            'studio_manager',
            'trainer'
        )
    );

-- Client visibility options
ALTER TABLE instructors
    ADD CONSTRAINT valid_client_visibility CHECK (
        client_visibility IN ('own', 'assigned', 'studio', 'all')
    );
```

---

3. Create Role Permissions Table (Optional)

sql
```sql
-- For custom role configurations
CREATE TABLE IF NOT EXISTS role_permissions (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   studio_id UUID REFERENCES bs_studios(id),
   role TEXT NOT NULL,
   permissions TEXT[] NOT NULL,
   restrictions JSONB DEFAULT '{}'::jsonb,
   created_at TIMESTAMPTZ DEFAULT NOW(),
   updated_at TIMESTAMPTZ DEFAULT NOW(),
   UNIQUE(studio_id, role)
);

-- Studios can customize permissions per role
-- Example: Studio A wants trainers to see all clients

-- Example: Studio B wants trainers to see assigned only
```

---

4. Create Permission Audit Log

sql
```sql
-- Track permission changes
CREATE TABLE IF NOT EXISTS permission_audit_log (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   studio_id UUID REFERENCES bs_studios(id),
   changed_by UUID REFERENCES profiles(id),
   target_user_id UUID REFERENCES profiles(id),
   action TEXT NOT NULL,            -- 'granted', 'revoked', 'role_changed'
   permission TEXT,
   old_value TEXT,
   new_value TEXT,
   reason TEXT,
   created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_permission_audit_studio ON permission_audit_log(studio_id);

CREATE INDEX idx_permission_audit_user ON permission_audit_log(target_user_id);
```

🔐 AUTHENTICATION IMPLEMENTATION

Supported Auth Methods:

1. **Email/Password** (Manual)
2. **Google OAuth**
3. **Magic Link** (Email only, passwordless)
4. **Apple Sign In** (Future)

Supabase Auth Configuration:

typescript
```
// lib/supabase-auth.ts

import { createClientComponentClient } from '@supabase/auth-helpers-nextjs';

export const supabase = createClientComponentClient();

// Sign up with email/password
export async function signUpWithEmail(
  email: string,
  password: string,
  metadata: {
    firstName: string;
    lastName: string;
    phone?: string;
    role: UserRole;
    studioMode?: StudioMode;
  }
) {
  const { data, error } = await supabase.auth.signUp({
    email,
    password,
    options: {
      data: {
        first_name: metadata.firstName,
        last_name: metadata.lastName,
        phone: metadata.phone,
        role: metadata.role,
        studio_mode: metadata.studioMode,
      },
      emailRedirectTo: `${window.location.origin}/auth/callback`,
    },
  });
  if (error) throw error;

  // Create profile record
  if (data.user) {
    await supabase.from('profiles').insert({
      id: data.user.id,
      email: data.user.email,
      first_name: metadata.firstName,
      last_name: metadata.lastName,
      phone: metadata.phone,
      role: metadata.role,
      v2_active: true,
      primary_platform: 'v2',
    });
  }

  return data;
}

// Sign in with Google
export async function signInWithGoogle() {
  const { data, error } = await supabase.auth.signInWithOAuth({
    provider: 'google',
    options: {
      redirectTo: `${window.location.origin}/auth/callback`,
      queryParams: {
        access_type: 'offline',
        prompt: 'consent',
      },
    },
  });

  if (error) throw error;
  return data;
}

// Sign in with email/password
export async function signInWithEmail(email: string, password: string) {
  const { data, error } = await supabase.auth.signInWithPassword({
    email,
    password,
  });

  if (error) throw error;
```

```typescript
  return data;
}

// Magic link (passwordless)
export async function signInWithMagicLink(email: string) {
  const { data, error } = await supabase.auth.signInWithOtp({
    email,
    options: {
      emailRedirectTo: `${window.location.origin}/auth/callback`,
    },
  });

  if (error) throw error;
  return data;
}

// Sign out
export async function signOut() {
  const { error } = await supabase.auth.signOut();
  if (error) throw error;
}
```

Auth Callback Handler:

typescript
```typescript
// app/auth/callback/route.ts

import { createRouteHandlerClient } from '@supabase/auth-helpers-nextjs';
import { cookies } from 'next/headers';
import { NextResponse } from 'next/server';

export async function GET(request: Request) {
  const requestUrl = new URL(request.url);
  const code = requestUrl.searchParams.get('code');

  if (code) {
    const supabase = createRouteHandlerClient({ cookies });
    await supabase.auth.exchangeCodeForSession(code);

    // Get user
    const { data: { user } } = await supabase.auth.getUser();

    if (user) {
      // Check if profile exists
      const { data: profile } = await supabase
        .from('profiles')
        .select('*')
        .eq('id', user.id)
        .single();

      if (!profile) {
        // Create profile from OAuth data
        await supabase.from('profiles').insert({
          id: user.id,
          email: user.email,
          first_name: user.user_metadata.full_name?.split(' ')[0],
          last_name: user.user_metadata.full_name?.split(' ').slice(1).join(' '),
          avatar_url: user.user_metadata.avatar_url,
          role: 'client', // Default role
          v2_active: true,
          primary_platform: 'v2',
        });

        // Redirect to onboarding
        return NextResponse.redirect(`${requestUrl.origin}/onboarding`);
      }

      // Redirect to dashboard
      const dashboardUrl = profile.role === 'client'
        ? '/client/dashboard'
        : '/studio/dashboard';

      return NextResponse.redirect(`${requestUrl.origin}${dashboardUrl}`);
    }
  }

  // Fallback redirect
  return NextResponse.redirect(`${requestUrl.origin}/`);
}
```

🛡️ PERMISSION CHECKING SYSTEM

Core Permission Functions:

typescript

```
// lib/permissions.ts

import { PERMISSION_CATEGORIES } from './roles';
import type { UserRole } from './types';

// Role-based permission mapping
const ROLE_PERMISSIONS: Record<UserRole, string[]> = {
  super_admin: ['*'], // All permissions

  solo_practitioner: [
    ...Object.values(PERMISSION_CATEGORIES.CLIENTS),
    ...Object.values(PERMISSION_CATEGORIES.BOOKINGS),
    ...Object.values(PERMISSION_CATEGORIES.SCHEDULE),
    ...Object.values(PERMISSION_CATEGORIES.SERVICES),
    ...Object.values(PERMISSION_CATEGORIES.PACKAGES),
    'finance:view:own',
    'finance:view:studio',
    'finance:payments:process',
    'finance:refunds:issue',
    'finance:pricing:manage',
    'finance:reports:export',
    ...Object.values(PERMISSION_CATEGORIES.SETTINGS),
    ...Object.values(PERMISSION_CATEGORIES.TRAINER_AIDE),
    'reports:view:own',
    'reports:view:studio',
    'reports:export',
    'reports:schedule',
    ...Object.values(PERMISSION_CATEGORIES.MARKETING),
  ],

  studio_owner: ['*'], // All permissions except platform

  studio_manager: [
    'clients:view:studio',
    'clients:create',
    'clients:edit:all',
    'clients:export',
    'bookings:view:studio',
    'bookings:create:clients',
    'bookings:edit:all',
    'bookings:cancel:any',
    'schedule:view:team',
    'schedule:manage:team',
    'services:view',
    'services:edit',
    'packages:view',
    'packages:sell',
    'finance:view:studio',
    'finance:payments:process',
    'finance:refunds:issue',
    'finance:reports:export',
    'team:view',
    'settings:view:studio',
    'locations:view',
    'locations:staff:assign',
    'trainer_aide:templates:view',
    'trainer_aide:assign:clients',
    'trainer_aide:progress:view',
    'reports:view:studio',
    'reports:export',
    'marketing:campaigns:view',
    'marketing:emails:send',
    'marketing:sms:send',
  ],

  trainer: [
    'clients:view:assigned',
    'clients:edit:assigned',
    'bookings:view:assigned',
    'bookings:create:clients',
    'bookings:edit:assigned',
    'bookings:cancel:own',
    'schedule:view:own',
    'schedule:manage:own',
    'services:view',
    'packages:view',
    'finance:view:own',
    'trainer_aide:templates:view',
    'trainer_aide:templates:create',
    'trainer_aide:templates:edit',
    'trainer_aide:assign:clients',
    'trainer_aide:progress:view',
    'reports:view:own',
  ],

  receptionist: [
    'clients:view:studio',
    'clients:create',
    'clients:edit:all',
    'bookings:view:studio',
    'bookings:create:any',
    'bookings:edit:all',
    'bookings:cancel:any',
    'schedule:view:all',
    'services:view',
    'packages:view',
    'packages:sell',
    'finance:payments:process',
  ],

  finance_manager: [
    'clients:view:studio',
    ...Object.values(PERMISSION_CATEGORIES.FINANCE),
    'packages:view',
    'packages:edit',
    'reports:view:all',
    'reports:export',
    'reports:schedule',
    'settings:view:billing',
    'settings:edit:billing',
  ],

  client: [
    'clients:view:own',
    'clients:edit:own',
    'bookings:view:own',
    'bookings:create:own',
    'bookings:cancel:own',
    'services:view',
    'packages:view',
    'schedule:view:own',
    'finance:view:own',
    'trainer_aide:progress:view',
  ],
};

// Check if user has permission
export function hasPermission(
  userRole: UserRole,
  permission: string,
  customPermissions?: string[]
): boolean {
  // Super admin has all permissions
  if (userRole === 'super_admin') return true;

  // Check custom permissions first (if studio customized)
  if (customPermissions?.includes(permission)) return true;

  // Check role-based permissions
  const rolePermissions = ROLE_PERMISSIONS[userRole] || [];

  // Wildcard check
  if (rolePermissions.includes('*')) return true;

  // Exact match
  if (rolePermissions.includes(permission)) return true;

  // Category wildcard (e.g., 'clients:*')
  const [category] = permission.split(':');
  if (rolePermissions.includes(`${category}:*`)) return true;

  return false;
}

// Check if user has ANY of the permissions
export function hasAnyPermission(
  userRole: UserRole,
  permissions: string[],
  customPermissions?: string[]
): boolean {
  return permissions.some(p => hasPermission(userRole, p, customPermissions));
}

// Check if user has ALL permissions
export function hasAllPermissions(
  userRole: UserRole,
  permissions: string[],
  customPermissions?: string[]
): boolean {
  return permissions.every(p => hasPermission(userRole, p, customPermissions));
}

// Get all permissions for role
export function getRolePermissions(role: UserRole): string[] {
  return ROLE_PERMISSIONS[role] || [];

}
```

React Hooks for Permissions:

typescript

```
// hooks/use-permissions.ts

import { useUser } from './use-user';
import { hasPermission, hasAnyPermission, hasAllPermissions } from '@/lib/permissions';

export function usePermissions() {
  const { user, profile } = useUser();

  const can = (permission: string) => {
    if (!profile) return false;
    return hasPermission(
      profile.role,
      permission,
      profile.custom_permissions
    );
  };

  const canAny = (permissions: string[]) => {
    if (!profile) return false;
    return hasAnyPermission(
      profile.role,
      permissions,
      profile.custom_permissions
    );
  };

  const canAll = (permissions: string[]) => {
    if (!profile) return false;
    return hasAllPermissions(
      profile.role,
      permissions,
      profile.custom_permissions
    );
  };

  const is = (role: UserRole | UserRole[]) => {
    if (!profile) return false;
    const roles = Array.isArray(role) ? role : [role];
    return roles.includes(profile.role);
  };
  return {
    can,
    canAny,
    canAll,
    is,
    role: profile?.role,
    isSuperAdmin: profile?.is_super_admin,
    isSoloPractitioner: profile?.role === 'solo_practitioner',
    isStudioOwner: profile?.role === 'studio_owner',
    isTrainer: profile?.role === 'trainer',
    isClient: profile?.role === 'client',
  };
}
```

Component-Level Protection:

typescript

```
// components/protected.tsx

import { usePermissions } from '@/hooks/use-permissions';
import { Alert } from '@/components/ui/alert';

interface ProtectedProps {
  permission?: string;
  permissions?: string[];
  requireAll?: boolean;
  role?: UserRole | UserRole[];
  fallback?: React.ReactNode;
  children: React.ReactNode;
}

export function Protected({
  permission,
  permissions,
  requireAll = false,
  role,
  fallback,
  children,
}: ProtectedProps) {
  const perms = usePermissions();

  // Role check
  if (role && !perms.is(role)) {
    return fallback || <AccessDenied />;
  }

  // Permission check
  if (permission && !perms.can(permission)) {
    return fallback || <AccessDenied />;
  }

  // Multiple permissions
  if (permissions) {
    const hasAccess = requireAll
      ? perms.canAll(permissions)
      : perms.canAny(permissions);

    if (!hasAccess) {
      return fallback || <AccessDenied />;
    }
  }

  return <>{children}</>;
}

function AccessDenied() {
  return (
    <Alert variant="destructive">
      <AlertTitle>Access Denied</AlertTitle>
      <AlertDescription>
        You don't have permission to view this content.
      </AlertDescription>
    </Alert>
  );
}

// Usage examples:
// <Protected permission="clients:create">
//   <CreateClientButton />
// </Protected>

// <Protected role="solo_practitioner">
//   <SoloPractitionerDashboard />
// </Protected>

// <Protected permissions={['finance:view:studio', 'reports:export']} requireAll>
//   <FinancialReports />
// </Protected>
```

Route Protection (Middleware):

typescript

```typescript
// middleware.ts

import { createMiddlewareClient } from '@supabase/auth-helpers-nextjs';
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

// Route protection rules
const ROUTE_PERMISSIONS: Record<string, {
  roles?: UserRole[];
  permissions?: string[];
}> = {
  '/studio/team': {
    permissions: ['team:view'],
  },
  '/studio/settings/billing': {
    permissions: ['settings:edit:billing'],
  },
  '/studio/locations': {
    permissions: ['locations:view'],
  },
  '/super-admin': {
    roles: ['super_admin'],
  },
  '/trainer-aide': {
    permissions: ['trainer_aide:templates:view'],
  },
};

export async function middleware(req: NextRequest) {
  const res = NextResponse.next();
  const supabase = createMiddlewareClient({ req, res });

  // Get session
  const { data: { session } } = await supabase.auth.getSession();

  if (!session) {
    // Redirect to login
    return NextResponse.redirect(new URL('/login', req.url));
  }

  // Get user profile
  const { data: profile } = await supabase
    .from('profiles')
    .select('role, is_super_admin, custom_permissions')
    .eq('id', session.user.id)
    .single();

  if (!profile) {
    return NextResponse.redirect(new URL('/onboarding', req.url));
  }

  // Check route permissions
  const pathname = req.nextUrl.pathname;
  const routeRule = Object.keys(ROUTE_PERMISSIONS).find(route =>
    pathname.startsWith(route)
  );

  if (routeRule) {
    const rule = ROUTE_PERMISSIONS[routeRule];

    // Check role
    if (rule.roles && !rule.roles.includes(profile.role)) {
      return NextResponse.redirect(new URL('/unauthorized', req.url));
    }

    // Check permissions
    if (rule.permissions) {
      const hasAccess = rule.permissions.some(permission =>
        hasPermission(profile.role, permission, profile.custom_permissions)
      );

      if (!hasAccess) {
        return NextResponse.redirect(new URL('/unauthorized', req.url));
      }
    }
  }

  return res;
}

export const config = {
  matcher: [
    '/studio/:path*',
    '/super-admin/:path*',
    '/trainer-aide/:path*',
  ],
};
```

📋 IMPLEMENTATION CHECKLIST

Week 1: Database + Backend (Days 1-7)

Day 1-2: Database Schema

- Add `role` column to `profiles`
- Add `is_super_admin` to `profiles`
- Add `platform_version` columns (from earlier plan)
- Add role columns to `instructors`
- Create `role_permissions` table
- Create `permission_audit_log` table
- Add constraints and indexes
- Test on staging first!

Day 3-4: Authentication Setup

- Configure Supabase Auth providers:
  - Email/Password
  - Google OAuth
  - Magic Link
- Create auth callback handler
- Implement signup functions
- Implement login functions
- Test all auth flows

Day 5-7: Permission System

- Create `lib/permissions.ts`
- Define all permission constants
- Create `hasPermission()` function
- Create `usePermissions()` hook
- Create `<Protected>` component
- Add middleware route protection

- Test permission checks

Week 2: Frontend Integration (Days 8-14)

Day 8-9: Auth UI

- Login page (email + Google)
- Signup page (email + Google)
- Password reset flow
- Email verification
- Magic link flow

Day 10-11: Role-Based UI

- Solo Practitioner dashboard
- Studio Owner dashboard
- Trainer dashboard
- Client dashboard
- Conditional navigation menus
- Permission-based component visibility

Day 12-13: Team Management

- Invite team members
- Assign roles
- Configure permissions
- Remove team members
- Audit log viewer

Day 14: Testing + Polish

- End-to-end role tests
- Permission boundary tests
- Auth flow tests
- Mobile testing
- Bug fixes

🎯 ROLE ASSIGNMENT FLOW

Solo Practitioner Signup:

typescript

```
// app/(v2)/signup/solo-pt/page.tsx

async function handleSoloPTSignup(data: SignupData) {
  // 1. Create auth user
  const { data: authData } = await signUpWithEmail(
    data.email,
    data.password,
    {
      firstName: data.firstName,
      lastName: data.lastName,
      phone: data.phone,
      role: 'solo_practitioner',
      studioMode: 'solo-pt',
    }
  );

  // 2. Create studio (solo PT is their own studio)
  const { data: studio } = await supabase
    .from('bs_studios')
    .insert({
      name: `${data.firstName} ${data.lastName} PT`,
      owner_id: authData.user.id,
      platform_version: 'v2',
      studio_mode: 'solo-pt',
    })
    .select()
    .single();

  // 3. Create instructor record
  await supabase.from('instructors').insert({
    user_id: authData.user.id,
    first_name: data.firstName,
    last_name: data.lastName,
    email: data.email,
    mobile: data.phone,
    store_name: studio.id,
    role: 'solo_practitioner',
    platform_version: 'v2',
  });

  // 4. Redirect to dashboard
  router.push(`/studio/${studio.id}/dashboard`);
}
```

Studio Owner Inviting Trainer:

typescript
```
// app/(v2)/studio/[id]/team/invite/page.tsx

async function inviteTrainer(data: InviteData) {
  // 1. Send invitation email
  const { data: invitation } = await supabase
    .from('team_invitations')
    .insert({
      studio_id: studioId,
      email: data.email,
      role: data.role, // 'trainer', 'manager', etc.
      invited_by: session.user.id,
      permissions: data.customPermissions, // Optional
    })
    .select()
    .single();

  // 2. Send email via Elastic Email
  await sendInvitationEmail({
    to: data.email,
    inviterName: profile.first_name,
    studioName: studio.name,
    role: data.role,
    inviteLink: `${siteUrl}/accept-invite/${invitation.id}`,
  });
}

// When trainer accepts:
async function acceptInvitation(invitationId: string) {
  // 1. Get invitation
  const { data: invitation } = await supabase
    .from('team_invitations')
    .select('*')
    .eq('id', invitationId)
    .single();

  // 2. Create auth user (if new) or link existing
  // 3. Create instructor record
  await supabase.from('instructors').insert({
    user_id: session.user.id,
    studio_id: invitation.studio_id,
    role: invitation.role,
    permissions_override: invitation.permissions,
  });

  // 4. Update profile role
  await supabase
    .from('profiles')
    .update({ role: invitation.role })
    .eq('id', session.user.id);
}
```

🧪 TESTING SCENARIOS

Solo Practitioner Tests:

typescript
```typescript
describe('Solo Practitioner Permissions', () => {
  it('should see all own clients', async () => {
    const clients = await getClients(soloPTUser);
    expect(clients).toHaveLength(soloPT.totalClients);
  });

  it('should create services', async () => {
    const result = await createService(soloPTUser, serviceData);
    expect(result.success).toBe(true);
  });

  it('should NOT see team management', async () => {
    const canViewTeam = hasPermission(soloPTUser.role, 'team:view');
    expect(canViewTeam).toBe(false);
  });

  it('should access Trainer Aide if purchased', async () => {
    const canUseTrainerAide = hasPermission(
      soloPTUser.role,
      'trainer_aide:templates:create'
    );
    expect(canUseTrainerAide).toBe(true);
  });

  it('should manage own billing', async () => {
    const canManageBilling = hasPermission(
      soloPTUser.role,
      'settings:edit:billing'
    );
    expect(canManageBilling).toBe(true);
  });
});
```

📚 SUMMARY

What We're Building:

1. **8 Distinct Roles** (including Solo Practitioner!)
2. **60+ Granular Permissions** across 10 categories
3. **3 Auth Methods** (Email, Google, Magic Link)
4. **Studio-Level Customization** (owners can adjust trainer permissions)
5. **Audit Trail** (track all permission changes)
6. **Component + Route Protection** (frontend + backend security)

Solo Practitioner = First-Class Citizen:

- ✅ **Same power as Studio Owner** (for their business)
- ✅ **Full Trainer Aide access** (if purchased)
- ✅ **No team management** (not needed, just 1 person)
- ✅ **No multi-location** (solo = 1 location)
- ✅ **Full financial control**
- ✅ **Full client management**
- ✅ **Perfect for independent PTs!**

Implementation Priority:

Week 1: Database + Backend (Foundation)
Week 2: Frontend + Testing (User-facing)