Authentication System Replication Guide

This guide will help you replicate the authentication system from **phoenixnewlocations.aps-serv.pro** to **maps.aps-serv.pro** so they can share the same user database and provide unified authentication.

Overview

The authentication system uses:
- **NextAuth.js** for authentication
- **PostgreSQL** for user storage (shared between both apps)
- **Prisma** as the ORM
- **bcryptjs** for password hashing
- **Role-based access control** (Admin, Level 2, Level 1)

Prerequisites

1. Access to the maps.aps-serv.pro app conversation in DeepAgent
2. Database connection string from this app
3. All users are already provisioned in the database

Step 1: Database Connection

Get the database URL from this app:

The database URL is stored in the `.env` file:

```
DATABASE_URL=postgresql://
role_479c0025d:woF0M75ydokL9qauCFYrbiJKVN_PrjBr@db-479c0025d.db003.hosteddb.reai.io:54
32/479c0025d?connect_timeout=15
```

IMPORTANT: Copy this exact URL to use in your other app.

Step 2: Set Up Prisma Schema

In your maps.aps-serv.pro app, update the Prisma schema file (`prisma/schema.prisma`):

```
generator client {
    provider = "prisma-client-js"
    binaryTargets = ["native", "linux-musl-arm64-openssl-3.0.x"]
    output = "../node_modules/.prisma/client"
}

datasource db {
    provider = "postgresql"
    url      = env("DATABASE_URL")
}

enum UserRole {
  ADMIN
  LEVEL2
  LEVEL1
}

model User {
  id                String   @id @default(cuid())
  email             String   @unique
  password          String?  // Nullable until user sets their password
  role              UserRole @default(LEVEL1)
  hasRegistered     Boolean  @default(false)
  createdAt         DateTime @default(now())
  updatedAt         DateTime @updatedAt

  @@index([email])
}
```

Then run:

```
cd nextjs_space
yarn prisma generate
```

Step 3: Add Environment Variables

Add to your `.env` file in the maps.aps-serv.pro app:

```
DATABASE_URL=postgresql://
role_479c0025d:woF0M75ydokL9qauCFYrbiJKVN_PrjBr@db-479c0025d.db003.hosteddb.reai.io:54
32/479c0025d?connect_timeout=15
NEXTAUTH_SECRET=hhZi/+oI7EnoM4UpizH+5MGrqq0PEDwh3bWnVogbOjU=
```

Note: NEXTAUTH_URL will be set automatically by the deployment system.

Step 4: Copy Authentication Files

4.1 Create `lib/auth.ts`

```
import { PrismaClient } from '@prisma/client'
import { compare } from 'bcryptjs'
import type { NextAuthOptions } from 'next-auth'
import CredentialsProvider from 'next-auth/providers/credentials'

const prisma = new PrismaClient()

export const authOptions: NextAuthOptions = {
  session: {
    strategy: 'jwt',
  },
  pages: {
    signIn: '/login',
    error: '/login',
  },
  providers: [
    CredentialsProvider({
      name: 'credentials',
      credentials: {
        email: { label: 'Email', type: 'email' },
        password: { label: 'Password', type: 'password' },
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) {
          throw new Error('Email and password are required')
        }

        const user = await prisma.user.findUnique({
          where: { email: credentials.email.toLowerCase() },
        })

        if (!user) {
          throw new Error('No user found with this email')
        }

        if (!user.hasRegistered || !user.password) {
          throw new Error('Please register your password first')
        }

        const isPasswordValid = await compare(credentials.password, user.password)

        if (!isPasswordValid) {
          throw new Error('Invalid password')
        }

        return {
          id: user.id,
          email: user.email,
          role: user.role,
        }
      },
    }),
  ],
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id as string
        token.email = user.email as string
        token.role = (user as any).role
      }
      return token
    },
```

```
    async session({ session, token }) {
      if (session.user) {
        (session.user as any).id = token.id as string
        (session.user as any).role = token.role as string
      }
      return session
    },
  },
}
```

4.2 Create `app/api/auth/[...nextauth]/route.ts`

```
import NextAuth from 'next-auth'
import { authOptions } from '@/lib/auth'

const handler = NextAuth(authOptions)

export { handler as GET, handler as POST }
```

4.3 Create `app/api/auth/register/route.ts`

```
import { PrismaClient } from '@prisma/client'
import { hash } from 'bcryptjs'
import { NextResponse } from 'next/server'

const prisma = new PrismaClient()

export const dynamic = 'force-dynamic'

const PASSWORD_REGEX = /^(?=.*[A-Z])(?=.*[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]).{9,}$/

export async function POST(req: Request) {
  try {
    const { email, password } = await req.json()

    if (!email || !password) {
      return NextResponse.json(
        { error: 'Email and password are required' },
        { status: 400 }
      )
    }

    if (!PASSWORD_REGEX.test(password)) {
      return NextResponse.json(
        {
          error:

'Password must be at least 9 characters long, contain at least 1 uppercase letter and
1 special character',
        },
        { status: 400 }
      )
    }

    const user = await prisma.user.findUnique({
      where: { email: email.toLowerCase() },
    })

    if (!user) {
      return NextResponse.json(
        {
          error:
            'Your email has not been authorized. Please contact the administrator.',
        },
        { status: 403 }
      )
    }

    if (user.hasRegistered) {
      return NextResponse.json(
        { error: 'You have already registered. Please log in.' },
        { status: 400 }
      )
    }

    const hashedPassword = await hash(password, 10)

    await prisma.user.update({
      where: { id: user.id },
      data: {
        password: hashedPassword,
        hasRegistered: true,
      },
```

```
    })

    return NextResponse.json(
      { message: 'Registration successful! You can now log in.' },
      { status: 200 }
    )
  } catch (error) {
    console.error('Registration error:', error)
    return NextResponse.json(
      { error: 'An error occurred during registration' },
      { status: 500 }
    )
  }
}
```

Step 5: Create Auth Pages

5.1 Create `app/(auth)/login/page.tsx`

Copy the entire login page from `phoenixnewlocations.aps-serv.pro/app/(auth)/login/page.tsx`

5.2 Create `app/(auth)/register/page.tsx`

Copy the entire register page from `phoenixnewlocations.aps-serv.pro/app/(auth)/register/page.tsx`

Step 6: Add Session Provider

6.1 Create `app/providers.tsx`

```
'use client'

import { SessionProvider } from 'next-auth/react'

export function Providers({ children }: { children: React.ReactNode }) {
  return <SessionProvider>{children}</SessionProvider>
}
```

6.2 Update `app/layout.tsx`

Wrap your app with the Providers component:

```
import { Providers } from './providers'

export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">
      <body>
        <Providers>
          {children}
        </Providers>
      </body>
    </html>
  )
}
```

Step 7: Add Route Protection

Create `middleware.ts` in the root of your Next.js app:

```
export { default } from 'next-auth/middleware'

export const config = {
  matcher: [
    '/((?!api/auth|login|register|_next/static|_next/image|favicon.ico|.*\\.svg|.*\
\.png|.*\\.jpg|.*\\.jpeg|.*\\.gif|.*\\.webp|.*\\.json|.*\\.css|.*\\.js).*)',
  ],
}
```

Step 8: Add User Info to Your App

In your main page or component, add session handling:

```
'use client'

import { useSession, signOut } from 'next-auth/react'

export default function YourPage() {
  const { data: session } = useSession()
  const userRole = (session?.user as any)?.role
  const userEmail = session?.user?.email

  // Use userRole for role-based access control
  // Use userEmail to display user information

  return (
    // Your page content
  )
}
```

Step 9: Role-Based Access Control (Optional)

If you need role-based access control in your maps app:

```
// Hide features from Level 1 users
{(userRole === 'LEVEL2' || userRole === 'ADMIN') && (
  <FeatureComponent />
)}

// Show admin-only features
{userRole === 'ADMIN' && (
  <AdminFeature />
)}
```

Step 10: Test the Integration

1. Deploy your maps.aps-serv.pro app
2. Try logging in with any user email from the list (e.g., `sjohnson@amenitypool.com`)
3. If the user hasn't registered yet, they'll need to go to `/register` and set their password
4. Once logged in, the session should work across both apps

Important Notes

Shared Database

- Both apps use the **same database**, so users only need to register once
- Password changes in one app affect the other app
- New users added by Sean in either app's admin interface will be available in both apps

User Roles

- **ADMIN**: Sean Johnson - can add users, full access to all features
- **LEVEL2**: Donnie, Todd, Chris, Troy - access to all features including sensitive data
- **LEVEL1**: All other users - standard access (can be customized per app)

Security Considerations

- Keep the `DATABASE_URL` and `NEXTAUTH_SECRET` secure
- Never commit these values to version control
- Use the same `NEXTAUTH_SECRET` in both apps for consistent JWT encryption

Troubleshooting

"No user found" Error

- Verify the user's email is in the database
- Check that the DATABASE_URL is correct
- Make sure Prisma has been generated: `yarn prisma generate`

"Please register your password first" Error

- The user needs to visit `/register` and set their password
- Check that `hasRegistered` is `true` in the database

Session Not Persisting

- Verify NEXTAUTH_SECRET is the same in both apps
- Check that cookies are being set correctly

- Ensure both apps are on the same domain (aps-serv.pro)

Admin Interface (Optional)

If you want Sean to be able to add users from the maps app as well, copy the admin interface:

1. Copy `app/admin/page.tsx`
2. Copy `app/api/admin/users/route.ts`

Otherwise, Sean can continue to add users from phoenixnewlocations.aps-serv.pro and they'll automatically be available in both apps.

Summary

After following this guide:
- ✅ Both apps share the same user database
- ✅ Users can log in with the same credentials on both apps
- ✅ Users only need to register their password once
- ✅ Role-based access control works consistently
- ✅ Sean can manage users from either app (if admin interface is implemented)

Questions?

Contact sjohnson@amenitypool.com for assistance.