Admin Login Flow - Technical Documentation

Overview

This document provides a comprehensive technical overview of the admin login and role-based access control system implemented in CatchBarrels.

Architecture

1. Role-Based Access Control (RBAC)

The system uses a role-based approach with three primary roles:

- `player` (default): Regular athletes using the app
- `coach` : Coaches with access to admin features
- `admin` : Full system administrators with all permissions

2. Database Schema

```
model User {
  id       String  @id @default(uuid())
  email    String?
  password String?

  // Role & Access Control
  role     String  @default("player") // "player", "coach", "admin"
  isCoach  Boolean @default(false)    // Quick check for coach access

  // ... other fields
}
```

3. NextAuth Configuration

Two separate authentication providers:

Admin Credentials Provider

```
CredentialsProvider({
  id: 'admin-credentials',
  name: 'Admin Credentials',
  // Only accepts users with role === 'admin' or 'coach'
})
```

Regular Credentials Provider

```
CredentialsProvider({
  id: 'credentials',
  name: 'Credentials',
  // Accepts all users (for testing/future use)
})
```

Whop OAuth Provider

```
{
  id: 'whop',
  name: 'Whop',
  // For athlete login via Whop subscription
}
```

Authentication Flows

Flow 1: Admin Login

```
1. User visits /auth/login
2. Clicks "Admin" tab
3. Enters admin email + password
4. Client calls signIn('admin-credentials', credentials)
5. Server validates:
   - Email exists
   - Password matches
   - role === 'admin' OR role === 'coach'
6. JWT token created with role info
7. Session includes { user: { role, isCoach } }
8. Client redirects to /admin
```

Flow 2: Athlete Login (Whop)

```
1. User visits /auth/login
2. Clicks "Sign in with Whop"
3. OAuth flow to Whop
4. Whop validates subscription
5. Callback with user data
6. Server syncs membership tier
7. JWT token created with membershipTier
8. Client redirects to /dashboard
```

Flow 3: Test User Login

```
1. User clicks "Test User" quick login button
2. Client calls signIn('credentials', test_credentials)
3. Server validates credentials
4. JWT token created
5. Client redirects to /dashboard
```

Authorization

Route Protection

Server-Side (Page Component)

```
// app/admin/page.tsx
export default async function AdminDashboardPage() {
  const session = await getServerSession(authOptions);

  if (!session) {
    redirect('/auth/login?callbackUrl=/admin');
  }

  const userRole = session.user?.role || 'player';
  const hasAdminAccess = userRole === 'admin' || userRole === 'coach';

  if (!hasAdminAccess) {
    redirect('/dashboard?error=unauthorized');
  }

  // Render admin content
}
```

Middleware (Global)

```
// middleware.ts
export async function middleware(request: NextRequest) {
  const token = await getToken({ req: request });

  // Check for admin routes
  if (pathname.startsWith('/admin')) {
    const userRole = token?.role || 'player';
    const hasAdminAccess = userRole === 'admin' || userRole === 'coach';

    if (!hasAdminAccess) {
      return NextResponse.redirect('/dashboard?error=unauthorized');
    }
  }

  // ... other checks
}
```

Session Management

JWT Token Structure

```
interface JWT {
  id: string;              // User ID
  email: string;           // User email
  username: string;        // Username
  role: string;            // "player" | "coach" | "admin"
  isCoach: boolean;        // Quick coach check
  membershipTier?: string; // For Whop users
  membershipStatus?: string;
  whopUserId?: string;     // For Whop users
}
```

Session Object (Client)

```
const { data: session } = useSession();

session?.user {
  id: string;
  email: string;
  name: string;
  role: string;       // Access user role
  isCoach: boolean;   // Quick check
}
```

Client-Side Components

Login Page

File: `app/auth/login/login-client.tsx`

Features:
- Tabbed interface (Athlete | Admin)
- Separate forms with different providers
- Role-based redirect logic
- Quick login buttons for testing

State Management:

```
const [loginMode, setLoginMode] = useState<'athlete' | 'admin'>('athlete');
```

Submit Handler:

```
const provider = loginMode === 'admin'
  ? 'admin-credentials'
  : 'credentials';

const defaultCallback = loginMode === 'admin'
  ? '/admin'
  : '/dashboard';
```

Security Considerations

1. Password Hashing

```
import bcrypt from 'bcryptjs';

const hashedPassword = await bcrypt.hash(password, 10);
const isValid = await bcrypt.compare(inputPassword, hashedPassword);
```

2. Role Validation

Always validate role on:
- **Server-side** page components
- **API routes**
- **Middleware**

Never trust client-side role checks alone!

3. Session Expiry

JWT tokens expire based on NextAuth configuration. Users must re-login after expiry.

4. HTTPS Only

All authentication must happen over HTTPS in production.

Admin Seeding

Seed Script

File: `scripts/seed-admin.ts`

What it does:
1. Reads `ADMIN_EMAIL` and `ADMIN_PASSWORD` from env
2. Checks if user exists
3. Creates new user OR updates existing user
4. Sets role to `admin`
5. Sets `isCoach` to `true`
6. Sets `membershipTier` to `elite`
7. Marks profile as complete

Usage:

```
yarn tsx scripts/seed-admin.ts
```

Environment Variables

Required for Authentication

```
# NextAuth
NEXTAUTH_SECRET="your-secret-key-here"
NEXTAUTH_URL="https://catchbarrels.app"

# Whop OAuth
WHOP_CLIENT_ID="your-whop-client-id"
WHOP_CLIENT_SECRET="your-whop-client-secret"

# Admin Seeding
ADMIN_EMAIL="admin@catchbarrels.app"
ADMIN_PASSWORD="your-secure-password"

# Database
DATABASE_URL="postgresql://..."
```

Testing

Manual Testing Checklist

- [ ] **Admin Login**
- [ ] Navigate to /auth/login
- [ ] Click “Admin” tab
- [ ] Enter admin credentials
- [ ] Verify redirect to /admin
- [ ] Verify admin dashboard loads
- [ ] **Athlete Login**
- [ ] Navigate to /auth/login
- [ ] Use “Test User” quick login
- [ ] Verify redirect to /dashboard
- [ ] Verify no access to /admin
- [ ] **Whop OAuth**
- [ ] Click “Sign in with Whop”
- [ ] Complete Whop OAuth flow
- [ ] Verify redirect to /dashboard
- [ ] Verify membership sync
- [ ] **Unauthorized Access**
- [ ] Login as athlete
- [ ] Try to access /admin directly
- [ ] Verify redirect to /dashboard with error
- [ ] **Role Persistence**

- [ ] Login as admin
- [ ] Refresh page
- [ ] Verify still on /admin
- [ ] Logout and verify session cleared

Troubleshooting

“Invalid admin credentials or insufficient permissions”

Causes:
- User role is not `admin` or `coach`
- Password incorrect
- User doesn’t exist

Solution:
- Run seed script again
- Verify env vars are set
- Check database for user role

Admin redirected to /dashboard

Causes:
- Session not updated with new role
- Middleware not recognizing role
- JWT token cached with old role

Solution:
- Clear browser cookies
- Logout and login again
- Check middleware logs
- Verify role in database

Quick login buttons not working

Causes:
- Test users not seeded
- Wrong provider being used

Solution:
- Check if users exist in database
- Verify handleQuickLogin uses correct provider

API Reference

Sign In

```
import { signIn } from 'next-auth/react';

// Admin login
await signIn('admin-credentials', {
  email: 'admin@example.com',
  password: 'password',
  redirect: false,
  callbackUrl: '/admin'
});

// Regular login
await signIn('credentials', {
  username: 'user@example.com',
  password: 'password',
  redirect: false,
  callbackUrl: '/dashboard'
});

// Whop OAuth
await signIn('whop', {
  callbackUrl: '/dashboard'
});
```

Get Session

```
// Client-side
import { useSession } from 'next-auth/react';
const { data: session } = useSession();

// Server-side
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth-options';
const session = await getServerSession(authOptions);
```

Check Role

```
// Client-side
const userRole = session?.user?.role || 'player';
const isAdmin = userRole === 'admin' || userRole === 'coach';

// Server-side
const userRole = session.user?.role || 'player';
const hasAdminAccess = userRole === 'admin' || userRole === 'coach';
```

File Structure

```
barrels_pwa/nextjs_space/
├── app/
│   ├── admin/                      # Admin-only routes
│   │   ├── page.tsx                # Admin dashboard (protected)
│   │   ├── admin-dashboard-client.tsx
│   │   └── .../
│   ├── auth/
│   │   └── login/
│   │       ├── page.tsx            # Login page wrapper
│   │       └── login-client.tsx    # Login UI with tabs
│   └── .../
├── lib/
│   ├── auth-options.ts             # NextAuth configuration
│   ├── admin/                      # Admin utilities
│   └── .../
├── middleware.ts                   # Global auth middleware
├── scripts/
│   └── seed-admin.ts               # Admin user seeding
├── types/
│   └── next-auth.d.ts              # NextAuth type extensions
└── docs/
    ├── ADMIN_SETUP.md              # This file
    └── ADMIN_LOGIN_FLOW.md         # Admin flow documentation
```

Future Enhancements

1. **Multi-Factor Authentication (MFA)** for admin accounts
2. **Audit Logging** for admin actions
3. **Role-based permissions** (granular permissions within admin)
4. **Admin user management UI** (create/edit/delete admins)
5. **Session timeout** warnings
6. **IP whitelisting** for admin access

Summary

✅ **Complete Features**:
- Separate admin credentials provider
- Role-based route protection
- Middleware-level authorization
- Admin seeding script
- Tabbed login UI
- Environment-based configuration
- Comprehensive documentation

🔒 **Security**:
- bcrypt password hashing
- JWT-based sessions
- Server-side role validation

- Middleware protection
- Secure environment variables

📚 **Documentation**:
- Setup guide (ADMIN_SETUP.md)
- Technical flow (this file)
- Code comments throughout
- Type safety with TypeScript

Last Updated: November 26, 2024
Version: 1.0
Status: Production Ready