Guía de Autenticación Multi-Tenant

Resumen

El sistema implementa autenticación multi-tenant usando NextAuth.js v4 con aislamiento completo de datos por tenant. Cada tenant tiene sus propios usuarios y solo pueden autenticarse en su subdominio específico.

Arquitectura

Componentes Principales

1. **NextAuth.js Configuration** (`/app/api/auth/[...nextauth]/route.ts`)
   - CredentialsProvider para login con email/password
   - Prisma Adapter para gestión de sesiones
   - Validación de tenant en autorización
   - Contraseñas hasheadas con bcryptjs

2. **Session Types** (`/types/next-auth.d.ts`)
   - Extensión de tipos de NextAuth
   - Incluye `tenant_id` y `role` en sesión
   - Type-safety completo en toda la aplicación

3. **Auth Utilities** (`/lib/auth.ts`)
   - `getServerSession()` : Obtener sesión en server components
   - `requireAuth()` : Proteger rutas (redirect a login si no autenticado)
   - `requireRole()` : Verificar permisos por rol
   - `getTenantFromSession()` : Extraer tenant_id de sesión

4. **Tenant Provider** (`/components/providers/tenant-provider.tsx`)
   - Context de React para datos del tenant
   - Obtiene tenant desde el subdominio
   - Disponible en toda la aplicación

5. **Middleware** (`/middleware.ts`)
   - Detección de subdominios
   - Protección de rutas
   - Redirección automática a login/dashboard

Flujo de Autenticación

1. Registro de Usuario

```
Usuario → /register → API /api/auth/register → Base de datos
```

Proceso:
1. Usuario completa formulario de registro
2. Sistema obtiene tenant desde subdominio (header `x-tenant-subdomain`)

3. Valida que email no exista en el tenant
4. Hashea contraseña con bcryptjs
5. Primer usuario del tenant recibe rol `OWNER`, siguientes `EMPLOYEE`
6. Redirecciona a `/login`

2. Login de Usuario

```
Usuario → /login → NextAuth → Verificación → Sesión JWT
```

Proceso:
1. Usuario ingresa email/password
2. Sistema obtiene tenant desde subdominio
3. Busca usuario por email Y tenant_id
4. Verifica contraseña hasheada
5. Crea sesión JWT con `id`, `email`, `tenant_id`, `role`
6. Redirecciona a `/dashboard`

3. Verificación de Sesión

```
Request → Middleware → Verifica Token → Permite/Redirige
```

Rutas Públicas:
- `/login`
- `/register`
- `/api/auth/*`
- `/api/tenant/current`

Rutas Protegidas:
- Todo lo demás requiere autenticación
- Middleware redirige a `/login` si no hay token

Roles de Usuario

El sistema soporta 5 roles definidos en Prisma:

```prisma
enum UserRole {
  OWNER       // Dueño del tenant, acceso completo
  ADMIN       // Administrador, gestión completa
  MANAGER     // Gerente, operaciones del día a día
  EMPLOYEE    // Empleado, acceso limitado
  VIEWER      // Solo lectura
}
```

Uso de Roles

En Server Components:

```tsx
import { requireRole } from '@/lib/auth';

export default async function AdminPage() {
  // Solo OWNER y ADMIN pueden acceder
  await requireRole(['OWNER', 'ADMIN']);

  return <div>Admin Panel</div>;
}
```

Verificar Roles:

```tsx
import { hasRole, isAdminOrOwner } from '@/lib/auth';

const isOwner = await hasRole('OWNER');
const canManage = await isAdminOrOwner();
```

Aislamiento Multi-Tenant

Seguridad por Tenant

1. **Autenticación:**
   - Usuario debe pertenecer al tenant del subdominio
   - No puede autenticarse en otros tenants con sus credenciales

2. **Sesión:**
   - JWT incluye `tenant_id`
   - Cada request lleva el tenant del usuario

3. **Base de Datos:**
   - Prisma middleware inyecta automáticamente `tenant_id`
   - Queries filtradas por tenant
   - Imposible acceder a datos de otros tenants

API Endpoints

POST /api/auth/register

Registra nuevo usuario en el tenant.

Request:

```json
{
  "name": "Juan Pérez",
  "email": "juan@ejemplo.com",
  "password": "contraseña123"
}
```

Response:

```
{
  "message": "Usuario registrado exitosamente",
  "user": {
    "id": "...",
    "name": "Juan Pérez",
    "email": "juan@ejemplo.com",
    "role": "OWNER",
    "created_at": "..."
  }
}
```

POST /api/auth/signin

Maneja NextAuth (no llamar directamente).

POST /api/auth/signout

Maneja NextAuth (no llamar directamente).

GET /api/tenant/current

Obtiene información del tenant actual.

Response:

```
{
  "tenant": {
    "id": "...",
    "name": "Mi Empresa",
    "subdomain": "miempresa",
    "business_type": "SCOOTER_RENTAL",
    ...
  }
}
```

Componentes de UI

Login Page (`/app/login/page.tsx`)

- Formulario con validación Zod
- React Hook Form
- Manejo de errores
- Link a registro

Register Page (`/app/register/page.tsx`)

- Formulario con validación Zod
- Confirmación de contraseña
- React Hook Form
- Link a login

Navbar (`/components/layout/navbar.tsx`)

- Logo del tenant
- Nombre de usuario
- Badge de rol

- Botón logout
- Enlaces a secciones principales

Testing Local

1. Setup Base de Datos

```
# En /home/ubuntu/rental_management
npx prisma migrate dev
```

2. Crear Tenant de Prueba

```
// seed.ts o script de prueba
const tenant = await prisma.tenant.create({
  data: {
    name: "Empresa Demo",
    subdomain: "demo",
    business_type: "SCOOTER_RENTAL",
  }
});
```

3. Simular Subdominios

Opción A: Editar /etc/hosts (Linux/Mac)

```
127.0.0.1  demo.localhost
```

Opción B: Usar ngrok con subdominios

```
ngrok http 3000 --subdomain=demo
```

4. Probar Flujo Completo

1. Abrir `http://demo.localhost:3000` (o tu URL de ngrok)
2. Sistema redirige a `/login`
3. Click en “crear una cuenta nueva”
4. Completar formulario de registro
5. Login con credenciales
6. Ver dashboard

5. Verificar Aislamiento

1. Crear segundo tenant en base de datos
2. Agregar entrada en /etc/hosts: `127.0.0.1  empresa2.localhost`
3. Abrir `http://empresa2.localhost:3000`
4. Intentar login con credenciales del primer tenant
5. Debe fallar con “Credenciales inválidas”

Seguridad

Contraseñas

- Hasheadas con bcryptjs (12 rounds)
- Nunca almacenadas en texto plano
- Validación mínima: 8 caracteres

JWT Tokens

- Firmados con `NEXTAUTH_SECRET`
- Almacenados en cookies httpOnly
- Expiran según configuración de NextAuth

Variables de Entorno Requeridas

```
DATABASE_URL="postgresql://..."
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="secreto-generado-con-openssl"
```

Generar NEXTAUTH_SECRET:

```
openssl rand -base64 32
```

Troubleshooting

Error: “Tenant no encontrado”

- Verificar que existe un tenant con ese subdomain en la base de datos
- Verificar header `x-tenant-subdomain` en middleware

Error: “Credenciales inválidas”

- Usuario no existe en este tenant
- Contraseña incorrecta
- Verificar que el email coincide con el tenant correcto

Redirección infinita

- Verificar que rutas públicas estén en el array del middleware
- Verificar configuración de NEXTAUTH_URL

Token no persiste

- Verificar NEXTAUTH_SECRET configurado
- Verificar cookies habilitadas en navegador
- Verificar dominio de cookies en producción

Próximos Pasos

1. Implementar recuperación de contraseña
2. Agregar autenticación 2FA
3. Implementar invitaciones de usuario

4. Logs de auditoría de accesos
5. Configuración de permisos granulares

Referencias

- NextAuth.js Documentation (https://next-auth.js.org/)
- Prisma Multi-Tenant (https://www.prisma.io/docs/guides/database/multi-tenant)
- Next.js Middleware (https://nextjs.org/docs/app/building-your-application/routing/middleware)