🔐 Sistema de Autenticación - App Bodegas

Este documento explica cómo funciona el sistema de autenticación implementado con **NextAuth.js**.

📋 Índice

Resumen

El sistema usa **NextAuth.js v4** con autenticación por email y contraseña (CredentialsProvider). Las contraseñas se hashean con **bcryptjs** y los usuarios se almacenan en PostgreSQL usando **Prisma**.

Tecnologías utilizadas:

- NextAuth.js v4.24.5
- bcryptjs (para hashear contraseñas)
- Prisma (para acceder a la base de datos)
- JWT (para manejar sesiones)

Roles y Permisos

👑 ADMIN

- Acceso completo a toda la aplicación
- Puede gestionar todas las bodegas
- Puede ver y modificar cualquier dato
- Acceso a `/admin/*`

📦 BODEGUERO

- Gestiona su bodega asignada
- Puede ver y editar productos de su bodega
- Puede gestionar pedidos y promociones
- Acceso a `/bodega/*` (solo su bodega)

👤 CLIENTE

- Puede ver productos de cualquier bodega
- Puede realizar pedidos

- Acceso limitado a información pública

Tabla de permisos por ruta:

Ruta	Admin	Bodeguero	Cliente	Público
`/login`	✅	✅	✅	✅
`/registro`	✅	✅	✅	✅
`/inicio`	✅	✅	✅	✅
`/bodegas`	✅	✅	✅	✅
`/admin/*`	✅	❌	❌	❌
`/bodega/*`	✅	✅*	❌	❌
`/pedidos`	✅	✅	✅	❌

*Solo su bodega asignada

Credenciales de Prueba

Después de ejecutar el seed, puedes usar estos usuarios:

Rol	Email	Contraseña
👑 Admin	admin@bodegas.com	password123
📦 Bodeguero	bodeguero@bodegas.com	password123
👤 Cliente	cliente@bodegas.com	password123

Cómo crear los usuarios de prueba:

```
# 1. Ejecutar la migración (si no lo has hecho)
npx prisma migrate dev --name agregar_autenticacion

# 2. Ejecutar el seed
npx tsx scripts/seed-usuarios.ts
```

Cómo Funciona

Flujo de Login:

1. Usuario ingresa email y contraseña en `/login`
2. NextAuth valida las credenciales contra la base de datos

3. Si son correctas, se crea un JWT con los datos del usuario
4. El JWT se guarda en una cookie segura
5. En cada request, el middleware verifica el JWT

Flujo de Registro:

1. Usuario llena el formulario en `/registro`
2. La API `/api/auth/registro` valida los datos
3. La contraseña se hashea con bcryptjs
4. Se crea el usuario en la base de datos (rol: CLIENTE)
5. Se inicia sesión automáticamente

Proteger Nuevas Rutas

Opción 1: Usando el Middleware (recomendado)

Edita `middleware.ts` para agregar rutas protegidas:

```
// Rutas que solo pueden acceder los ADMIN
const ADMIN_ROUTES = [
  '/admin',
  '/mi-nueva-ruta-admin', // ← Agregar aquí
];

// Rutas para BODEGUERO (y ADMIN)
const BODEGUERO_ROUTES = [
  '/bodega',
  '/bodegas',
  '/mi-nueva-ruta-bodega', // ← Agregar aquí
];
```

Opción 2: Usando el componente ProtectedRoute

Envuelve tu página con el componente:

```
// En tu page.tsx
import ProtectedRoute from '@/components/ProtectedRoute';

export default function MiPaginaProtegida() {
  return (
    <ProtectedRoute allowedRoles={['ADMIN', 'BODEGUERO']}>
      <div>Contenido solo para Admin y Bodeguero</div>
    </ProtectedRoute>
  );
}
```

Opción 3: En Server Components

```
import { getServerAuthSession } from '@/lib/auth';
import { redirect } from 'next/navigation';

export default async function MiPaginaServer() {
  const session = await getServerAuthSession();

  if (!session) {
    redirect('/login');
  }

  if (session.user.rol !== 'ADMIN') {
    redirect('/no-autorizado');
  }

  return <div>Solo Admin puede ver esto</div>;
}
```

Agregar Nuevos Usuarios

Vía API (para clientes):

Los usuarios se registran automáticamente como CLIENTE desde `/registro`.

Vía Prisma Studio (para Admin/Bodeguero):

```
# Abrir Prisma Studio
npx prisma studio
```

Luego navega a la tabla `Usuario` y crea un nuevo registro. **Importante:** debes hashear la contraseña antes de guardarla.

Vía Script:

Puedes crear un script similar a `seed-usuarios.ts`:

```
import { PrismaClient } from '@prisma/client';
import bcryptjs from 'bcryptjs';

const prisma = new PrismaClient();

async function crearUsuario() {
  const passwordHash = await bcryptjs.hash('miContraseña123', 10);

  await prisma.usuario.create({
    data: {
      email: 'nuevo@email.com',
      password: passwordHash,
      nombre: 'Nuevo Usuario',
      rol: 'BODEGUERO', // O 'ADMIN', 'CLIENTE'
      bodegaId: 'BOD_XXX', // Solo para BODEGUERO
    },
  });
}
```

Archivos Importantes

```
app-bodegas/
├── prisma/
│   └── schema.prisma          # Modelo Usuario y enum Rol
├── lib/
│   └── auth.ts                 # Funciones helper de auth
├── app/
│   ├── api/auth/
│   │   ├── [...nextauth]/route.ts  # Configuración NextAuth
│   │   └── registro/route.ts       # API de registro
│   ├── login/page.tsx          # Página de login
│   ├── registro/page.tsx       # Página de registro
│   └── no-autorizado/page.tsx  # Página de error 403
├── components/
│   ├── ProtectedRoute.tsx      # Componente de protección
│   └── SessionProvider.tsx     # Provider de NextAuth
├── middleware.ts               # Middleware de protección
└── scripts/
    └── seed-usuarios.ts        # Script para crear usuarios
```

Variables de Entorno

Asegúrate de tener estas variables en tu `.env`:

```
# Base de datos
DATABASE_URL="postgresql://..."

# NextAuth (IMPORTANTE: genera un secret único)
NEXTAUTH_SECRET="tu-secret-super-seguro-aqui"
NEXTAUTH_URL="http://localhost:3000"
```

Para generar un secret seguro:

```
openssl rand -base64 32
```

Troubleshooting

Error: “NEXTAUTH_SECRET is not set”

Agrega `NEXTAUTH_SECRET` a tu `.env`

Error: “Usuario no encontrado”

Verifica que ejecutaste el seed: `npx tsx scripts/seed-usuarios.ts`

Error: “Contraseña incorrecta”

La contraseña por defecto es `password123`

La sesión no persiste

Verifica que `NEXTAUTH_URL` coincide con tu URL de desarrollo

¡Listo! Ahora tienes un sistema de autenticación completo. 🎉