Arquitectura de Autenticación - FreeBridge

📁 Estructura Modularizada

```
src/
├── utils/
│   └── sessionManager.js          # ☑ Funciones puras para localStorage
├── context/
│   └── AuthContext.jsx             # ☑ Estado global de autenticación
├── hooks/
│   └── useSessionTimeout.js        # ☑ Lógica de expiración de sesión
├── components/
│   ├── Navbar.jsx                  # ☑ UI únicamente (refactorizado)
│   └── loginForm.jsx               # ☑ Usa AuthContext
└── App.jsx                          # ☑ Configurado con AuthProvider
```

🔧 Componentes de la Arquitectura

1 **sessionManager.js** - Utilidades de Sesión

Ubicación: `src/utils/sessionManager.js`

Propósito: Funciones puras para manejar localStorage

Funciones principales:

- `getToken()` - Obtiene el token de autenticación
- `getUserRole()` - Obtiene el rol del usuario
- `setSessionData()` - Guarda datos de sesión completos
- `clearSession()` - Limpia todos los datos de sesión
- `getLastActivity()` - Obtiene timestamp de última actividad
- `setLastActivity()` - Actualiza timestamp de actividad
- `getTimeSinceLastActivity()` - Calcula tiempo desde última actividad

Ventajas:

- ☑ Sin dependencias de React
- ☑ Funciones puras y testeables
- ☑ Centraliza lógica de localStorage
- ☑ Fácil de mantener

---

2 **AuthContext.jsx** - Estado Global

Ubicación: `src/context/AuthContext.jsx`

Propósito: Context API para estado de autenticación global

Estado proporcionado:

```
{
  isAuthenticated,  // boolean: si hay sesión activa
  userRole,         // string: "Empresa" | "Freelancer" | null
  userId,           // string: ID del usuario
  userName,         // string: Nombre del usuario
  isLoading,        // boolean: cargando estado inicial
}
```

Métodos proporcionados:

```
{
  login(userData),        // Inicia sesión y guarda datos
  logout(redirect),       // Cierra sesión y opcionalmente redirige
  checkAuth(),            // Verifica estado de autenticación
  navigateToProfile(),    // Navega al dashboard según rol
}
```

Uso:

```
import { useAuth } from "../context/AuthContext";

function MiComponente() {
  const { isAuthenticated, logout, userRole } = useAuth();
  // ...
}
```

---

3 **useSessionTimeout.js** - Hook de Expiración

Ubicación: `src/hooks/useSessionTimeout.js`

Propósito: Hook personalizado que maneja expiración por inactividad

Configuración:

```
useSessionTimeout(30 * 60 * 1000); // 30 minutos
```

¿Qué hace?

- ☑ Detecta actividad del usuario (clicks, scroll, teclas)
- ☑ Actualiza timestamp con cada interacción
- ☑ Verifica expiración cada 1 minuto
- ☑ Cierra sesión automáticamente si pasa el tiempo límite

- ☑ Limpia event listeners al desmontar

Eventos detectados:

- `mousedown` - Clic del mouse
- `keydown` - Tecla presionada
- `scroll` - Desplazamiento
- `touchstart` - Toque en móvil
- `click` - Clic general
- `mousemove` - Movimiento del mouse

---

4 **App.jsx** - Configuración Principal

Ubicación: `src/App.jsx`

Estructura:

```
<BrowserRouter>
  <AuthProvider>
    {" "}
    {/* Proporciona contexto global */}
    <AppContent>
      {" "}
      {/* Usa el hook useSessionTimeout */}
      <AppRouter /> {/* Rutas de la aplicación */}
    </AppContent>
  </AuthProvider>
</BrowserRouter>
```

Por qué esta estructura:

- `AuthProvider` debe estar dentro de `BrowserRouter` para usar `useNavigate`
- `useSessionTimeout` debe estar dentro de `AuthProvider` para acceder al contexto
- `AppContent` es el componente intermedio que conecta todo

🔄 Flujo de Autenticación

Inicio de Sesión

```
1. Usuario completa formulario de login
   ↓
2. loginForm.jsx llama a loginUser() API
   ↓
3. loginForm usa login() del contexto
   ↓
4. AuthContext guarda datos vía sessionManager
   ↓
```

```
5. sessionManager escribe en localStorage
   ↓
6. Estado global se actualiza (isAuthenticated = true)
   ↓
7. Navbar re-renderiza mostrando botones de sesión activa
   ↓
8. useSessionTimeout inicia monitoreo de actividad
```

Durante la Sesión

```
Usuario interactúa con la app
   ↓
useSessionTimeout detecta evento (click, scroll, etc.)
   ↓
Actualiza lastActivity en localStorage
   ↓
Cada 1 minuto verifica tiempo transcurrido
   ↓
¿Pasaron más de 30 minutos? → NO → Continúa
                              ↓ SÍ
                         Cierra sesión automáticamente
```

Cierre de Sesión

```
Usuario hace click en "Cerrar Sesión"
   ↓
Navbar llama a logout(true)
   ↓
AuthContext ejecuta clearSession()
   ↓
sessionManager limpia localStorage
   ↓
Estado global se resetea
   ↓
Usuario es redirigido a "/"
```

☑ Ventajas de esta Arquitectura

Separación de Responsabilidades

- 📦 **Utils**: Solo funciones puras
- 🌐 **Context**: Solo estado global
- 🪝 **Hooks**: Solo lógica reutilizable
- 🎨 **Components**: Solo UI

Reutilización

```
// Cualquier componente puede usar autenticación
import { useAuth } from "../context/AuthContext";

function AnyComponent() {
  const { isAuthenticated, userRole } = useAuth();
  // ...
}
```

Testing

- ☑ sessionManager: Test unitarios de funciones puras
- ☑ AuthContext: Mock del provider
- ☑ useSessionTimeout: Test del hook aislado
- ☑ Components: Test con context mockeado

Mantenibilidad

- 🔧 Cambiar tiempo de expiración: Solo editar `App.jsx`
- 🔧 Agregar nuevo dato de sesión: Solo editar `sessionManager.js`
- 🔧 Modificar flujo de logout: Solo editar `AuthContext.jsx`

Escalabilidad

Fácil agregar:

- 🚀 Refresh tokens automáticos
- 🚀 Multi-sesión (múltiples pestañas)
- 🚀 Persistencia en base de datos
- 🚀 OAuth / Social login
- 🚀 Verificación 2FA

🧪 Cómo Probar

Cambiar tiempo de expiración (para testing)

En `App.jsx`:

```
useSessionTimeout(2 * 60 * 1000); // 2 minutos
```

Ver datos en consola

```
// En cualquier componente
import { getSessionData } from "../utils/sessionManager";

console.log(getSessionData());
```

Forzar cierre de sesión

```
const { logout } = useAuth();
logout(true); // Cierra y redirige
```

Migración de Componentes Antiguos

Si tienes componentes que usan localStorage directamente:

❌ Antes:

```
const token = localStorage.getItem("token");
const role = localStorage.getItem("userRole");
```

☑ Después:

```
import { useAuth } from "../context/AuthContext";

const { isAuthenticated, userRole } = useAuth();
```

Seguridad

- ☑ Sesiones expiran automáticamente
- ☑ Tokens no quedan activos indefinidamente
- ☑ Detección real de actividad del usuario
- ☑ Limpieza completa al cerrar sesión
- ⚠ **Pendiente**: Implementar tokens JWT con expiración del servidor

---

Recursos Adicionales

- React Context API
- Custom Hooks
- Event Listeners