




Estrategia de Seguridad - Sistema de Inspecciones Vehiculares

Resumen

Este documento describe la estrategia de seguridad implementada en la aplicación de inspecciones vehiculares.




Niveles de Seguridad

1. Autenticación (Firebase Auth)

-  Todos los usuarios deben estar autenticados para acceder a la aplicación
-  Contraseñas seguras gestionadas por Firebase
-  Tokens JWT para cada sesión

2. Reglas de Firestore

Las reglas actuales verifican:

-  Usuario autenticado para todas las operaciones de lectura
-  Usuario autenticado para operaciones de escritura
-  Propietario del recurso para actualizaciones de inspecciones

IMPORTANTE: Las reglas de Firestore NO verifican roles usando `get()` para evitar problemas de performance y timing. En su lugar:

- La verificación de roles se hace en el código de la aplicación
- Los componentes y páginas verifican el rol del usuario
- Los Route Guards protegen las rutas según el rol

3. Route Guards (Protección de Rutas)

Ubicación: `components/auth/route-guard.tsx`

Protege las rutas según el rol:

- **Admin:** Puede acceder a todas las rutas `/admin/*`
- **Conductor:** Solo puede acceder a rutas `/conductor/*`

4. Verificación en Componentes

Cada componente crítico verifica el rol del usuario mediante:

```
const { user } = useAuth();

if (user?.role !== 'administrador') {
  // Redirigir o mostrar error
}
```

Advertencias Importantes

¿Por qué no usar `get()` en las reglas de Firestore?

Las reglas con `get()` causan:

- ❌ Problemas de timing en la propagación de tokens
- ❌ Errores “Missing or insufficient permissions”
- ❌ Performance degradada por múltiples consultas

Solución adoptada:

- ✅ Verificación de autenticación en Firestore rules
- ✅ Verificación de roles en el código de la aplicación
- ✅ Route Guards para proteger rutas

¿Es seguro esto?

Sí, porque:

1. Solo usuarios autenticados pueden acceder a los datos
2. Los Route Guards previenen acceso no autorizado a rutas
3. Los componentes verifican roles antes de mostrar acciones críticas
4. Firebase Auth gestiona la autenticación de forma segura

¿Qué pasa si un usuario malicioso intenta manipular los datos?

- ✅ **Lectura:** Solo usuarios autenticados pueden leer datos
- ✅ **Escritura:** Solo usuarios autenticados pueden escribir
- ✅ **Inspecciones:** Solo el conductor que creó la inspección puede modificarla
- ✅ **Usuarios:** Solo el propio usuario puede actualizar su documento

Mejoras Futuras (Opcional)

Si en el futuro necesitas mayor seguridad, puedes:

1. Implementar Custom Claims:

- Almacenar el rol en Firebase Auth custom claims
- Usar `request.auth.token.role` en Firestore rules
- Requiere Firebase Admin SDK y configuración adicional

2. API Routes con verificación:

- Crear endpoints en `/api/*` que verifiquen roles
- Proxy de todas las operaciones críticas

3. Middleware de Next.js:

- Verificar roles en el servidor antes de renderizar
- Requiere Next.js 12+ con middleware

✅ Recomendaciones

Para mantener la seguridad:

1. ✅ No compartas credenciales de admin
2. ✅ Usa contraseñas seguras
3. ✅ Revisa los logs de Firebase regularmente

4. ☒ Mantén actualizadas las dependencias
5. ☒ No expongas las variables de entorno

Soporte

Si tienes dudas sobre la seguridad, consulta:

- [Documentación de Firebase Security Rules](https://firebase.google.com/docs/firestore/security/rules-structure) (<https://firebase.google.com/docs/firestore/security/rules-structure>)
- [Firebase Auth Best Practices](https://firebase.google.com/docs/auth/web/auth-best-practices) (<https://firebase.google.com/docs/auth/web/auth-best-practices>)