Documentación: Sistema de Autenticación

Proyecto: Auth System (Fullstack)
Autor: Erick Capilla
Fecha: 4 de Febrero, 2026

Introducción

Sistema integral de autenticación desarrollado como prueba técnica usando tecnologías como NestJS, ReactJs y MongoDB. La aplicación implementa un flujo de seguridad utilizando **JWT en Cookies HttpOnly**, validación de esquemas, y persistencia de sesiones con geolocalización.

Modelo de Datos (MongoDB)

El sistema utiliza **MongoDB** con una arquitectura de dos colecciones principales relacionadas. Se ha diseñado para ser eficiente en las lecturas y seguro en la gestión de credenciales.

Colección: Users

Almacena la identidad persistente de los usuarios.

Campo	Tipo	Descripción
name	String	Nombre completo del usuario.
email	String	Correo electrónico (Índice único).
password	String	Hash de la contraseña .

createdAt	Date	Fecha de registro.

Colección: Sessions

Almacena los registros de actividad y persistencia de sesión. Está vinculada a la colección de usuarios.

Campo	**Tipo**	**Descripción**
userId	ObjectId	Referencia al ID del usuario (Ref: User).
location	Object	Contiene country, city lat y lng capturados en el login.
lastSession	Date	Marca de tiempo para control de expiración.

Arquitectura de Seguridad (Backend)

Gestión de Sesiones mediante Cookies HttpOnly

A diferencia de las implementaciones tradicionales que almacenan JWT en `localStorage`, este sistema utiliza **Cookies HttpOnly y Secure**.

- **Beneficio:** El token no es accesible mediante scripts de JavaScript, lo que anula prácticamente cualquier intento de robo de sesión mediante ataques **Cross-Site Scripting (XSS)**.

Estrategia de Persistencia de Sesiones

Cada vez que un usuario se autentica, se genera un registro en la colección `sessions` de MongoDB.

- **Validación de Estado:** El sistema no solo confía en la validez criptográfica del JWT, sino que verifica en tiempo real si la sesión existe en la base de datos antes de permitir el acceso (Stateful JWT).

Mecanismo de Defensa: Rate Limiting

Para mitigar ataques de fuerza bruta, se implementó un `ThrottlerGuard`:

- **Lógica:** 3 intentos fallidos permitidos.
- **Penalización:** Bloqueo automático de la IP/Usuario por 5 minutos tras el tercer fallo.

Endpoints de la API

Método	Ruta	Descripción
POST	/api/auth/signup	Registra un nuevo usuario y hashea la contraseña.
POST	/api/auth/login	Valida credenciales, guarda ubicación y genera cookie. (Rate Limit)
GET	/api/auth/profile	Retorna datos del usuario si la cookie es válida.
POST	/api/auth/logout	Elimina la sesión en BD y limpia la cookie.

Estructura de Frontend (React)

Manejo de Estado con Context API

Un Context Provider que envuelve la aplicación y gestiona el estado de autenticación de forma persistente. Se centralizó la lógica en un `AuthContext`. Esto permite que:

- La aplicación sea reactiva: los componentes se re-renderizan automáticamente al cambiar el estado de autenticación.
- **Persistencia:** Al cargar la app, se dispara un efecto que consulta el endpoint `/auth/profile`. Si el navegador tiene la cookie válida, el usuario recupera su sesión automáticamente.

Validación de Esquemas con Zod

Toda la data de entrada se valida en el cliente mediante **Zod** antes de ser enviada al servidor. Esto mejora la experiencia de usuario (UX) al proporcionar feedback instantáneo sin necesidad de esperar una respuesta del servidor.

Protected Route

Componente de alto orden (HOC) que restringe el acceso a rutas privadas, redirigiendo al login si no existe una sesión válida.

Conclusión

El sistema no solo cumple con los requisitos funcionales, sino que escala a un nivel de producción al considerar la seguridad de las cookies, el límite de peticiones y el control de sesiones activas en el servidor.