Módulo de Autenticación mediante Tokens JWT

Propósito General

Este módulo implementa el sistema completo de generación y verificación de JSON Web Tokens (JWT) para la aplicación. Proporciona los mecanismos necesarios para crear tokens de acceso seguros que autentican a los usuarios en el sistema, así como para validar estos tokens en solicitudes subsiguientes. La implementación utiliza el algoritmo HS256 (HMAC con SHA-256) que ofrece un balance óptimo entre seguridad y rendimiento.

Arquitectura de Seguridad

El servicio se basa en la biblioteca jose, una implementación moderna y robusta de los estándares JWT que sigue las mejores prácticas de seguridad actuales. A diferencia de implementaciones más antiguas, jose proporciona una API más segura por defecto y mejor manejo de claves criptográficas.

Configuración de Clave Secreta

El componente fundamental del sistema es la clave secreta utilizada para firmar y verificar los tokens. Esta clave se deriva de la variable de entorno JWT_SECRET mediante un proceso de encoding a bytes utilizando TextEncoder. Este enfoque garantiza que la clave esté en el formato adecuado para las operaciones criptográficas.

La seguridad del sistema completo depende críticamente de la fortaleza de JWT_SECRET. La documentación incluye una recomendación explícita para generar una clave segura utilizando el módulo crypto de Node.js, sugiriendo un mínimo de 24 caracteres pero recomendando 64 bytes (128 caracteres hexadecimales) para máxima seguridad.

Generación de Tokens de Acceso

La función signAccessToken es responsable de crear nuevos tokens JWT. Esta función acepta un objeto payload que debe contener como mínimo las siguientes propiedades: sub (subject - identificador único del usuario), email (dirección de correo electrónico), name (nombre del usuario), y role (rol del usuario en el sistema).

El proceso de generación del token sigue un flujo estructurado: primero se establece el algoritmo de firma como HS256 en el header protegido, luego se marca el tiempo de emisión del token, se define el tiempo de expiración basado en la variable de entorno JWT_EXPIRES_IN, y finalmente se firma el token utilizando la clave secreta.

El uso de setIssuedAt automáticamente incluye la timestamp de creación (claim "iat"), mientras que setExpirationTime establece el claim "exp" basado en la configuración de

la aplicación. Esto proporciona un control granular sobre la validez temporal de los tokens.

Verificación de Tokens de Acceso

La función verifyAccessToken implementa el proceso inverso, tomando un token stringificado y verificando su autenticidad y validez. Esta función realiza múltiples validaciones automáticamente: verifica la firma criptográfica, valida que el token no haya expirado, y asegura que la estructura del token sea correcta.

Cuando la verificación es exitosa, la función retorna el payload decodificado del token, que incluye toda la información del usuario originalmente embebida durante la generación más los claims estándar de JWT (iat, exp). Si la verificación falla por cualquier motivo (token expirado, firma inválida, formato incorrecto), la función lanza una excepción que debe ser manejada apropiadamente por el middleware de autenticación.

Gestión de Tiempos de Expiración

El sistema utiliza la variable de entorno JWT_EXPIRES_IN para controlar la validez temporal de los tokens. Los valores típicos siguen el formato de string de tiempo de JavaScript (ej: "15m" para 15 minutos, "1h" para una hora, "7d" para 7 días). Esta configuración permite ajustar fácilmente la política de seguridad según los requisitos específicos de la aplicación sin modificar el código.

Consideraciones de Seguridad

La implementación utiliza HMAC-SHA256, un algoritmo simétrico que es eficiente y seguro cuando la clave secreta se mantiene confidencial. Para aplicaciones distribuidas o microservicios, podría considerarse el uso de algoritmos asimétricos como RS256, pero HS256 es completamente adecuado para la mayoría de aplicaciones monolíticas.

El servicio no incluye mecanismos de revocación de tokens, lo que significa que los tokens son válidos hasta su expiración. Para requisitos de seguridad más estrictos, podría implementarse una lista negra de tokens revocados.

Integración con el Flujo de Autenticación

Este servicio se integra directamente con el middleware de autenticación y los controladores de auth. Típicamente, después de que un usuario se autentica exitosamente con un proveedor externo como Google, el sistema utiliza signAccessToken para generar un token JWT propio que se envía al cliente. Este token luego se incluye en el header Authorization de las solicitudes subsiguientes y es verificado por verifyAccessToken en el middleware.

Mejores Prácticas de Implementación

La documentación enfatiza la importancia de generar una clave secreta fuerte utilizando herramientas criptográficas apropiadas. La recomendación de usar crypto.randomBytes(64) genera 64 bytes de datos aleatorios, resultando en una clave de 128 caracteres hexadecimales que es virtualmente imposible de adivinar mediante ataques de fuerza bruta.

Para entornos de producción, se recomienda almacenar JWT_SECRET en variables de entorno seguras y nunca comitearla en repositorios de código. Además, debería rotarse periódicamente según la política de seguridad de la organización.

Manejo de Errores

El servicio propaga excepciones que deben ser capturadas y manejadas por las funciones caller. Los errores típicos incluyen tokens expirados, firmas inválidas, y tokens malformados. Cada uno de estos casos debe resultar en una respuesta HTTP apropiada (generalmente 401 Unauthorized) cuando ocurren durante la verificación en el middleware.

Esta implementación proporciona una base sólida y segura para la autenticación basada en tokens, siguiendo estándares industry y buenas prácticas de seguridad mientras mantiene una API simple y fácil de usar para los desarrolladores de la aplicación.