
FASE II — AUTH CON RBAC

ISS-13: Middlewares de Acceso y las Tres Modalidades en Rutas

Materialización de las tres modalidades (OPEN, JWT, JWT + RBAC) mediante middlewares componibles (authenticate y authorize) aplicados a las rutas de negocio sin alterar controladores ni servicios.

Propósito del ISS-13 e Integración Arquitectónica

Objetivo del Módulo

- **Capa Interceptora Transversal:** Insertar la barrera de seguridad entre la petición HTTP entrante y las rutas/controladores de la aplicación.
- **Consumo de la Matriz RBAC:** La matriz de permisos construida en ISS-12 pasa de ser un conjunto inerte de datos a decidir cada petición en tiempo real.
- **Cero Alteración de Negocio:** Los controladores, servicios y repositorios de la Fase I no se modifican en absoluto; la seguridad es declarativa en la capa de enrutado.
- **Efecto Inmediato por Datos:** Al no almacenar roles/permisos en el JWT, modificar un permiso en la BD surte efecto en la siguiente petición sin reemitir tokens.

Flujo de Petición Protegida

1. HTTP Request

Petición entrante con encabezado Authorization: Bearer <token>

2. authenticate

Valida JWT HS256, usuario 'active' en BD y adjunta req.auth (401 si falla)

3. authorize

Normaliza URL (method, path) y busca concesión activa en RBAC (403 si falla)

4. Controller / Handler

Ejecuta la lógica de negocio de la Fase I y retorna 200/201

Las Tres Modalidades de Acceso por Ruta

1. Modalidad OPEN

Middleware: Ninguno

Exigencia: Petición libre sin credenciales de identidad previas.

Propósito: Puntos de entrada al sistema o documentación pública.

Ejemplos:

- POST /api/sesion/login
- POST /api/sesion/refresh
- POST /api/sesion/logout
- GET /api/docs (OpenAPI UI)
- GET /api/docs.json

2. Modalidad JWT

Middleware: authenticate

Exigencia: Access token JWT válido y usuario con status 'active' en BD.

Código de Error: HTTP 401 Unauthorized (sin token o caducado).

Propósito: Operaciones que requieren conocer la identidad sin exigir concesión RBAC.

Ejemplos:

- GET /api/sesion/perfil
- GET /api/permisos
- GET /api/sesiones
- DELETE /api/sesiones

3. JWT + RBAC

Middleware: authenticate, authorize

Exigencia: Identidad válida Y concesión activa sobre la tupla (method, path).

Códigos de Error: 401 (sin token) / 403 (sin concesión - Deny by default).

Propósito: Proteger todo el CRUD de negocio y la administración del sistema.

Ejemplos:

- /api/clientes...
- /api/productos...
- /api/ventas...
- /api/usuarios...

Anatomía de authenticate.middleware.ts (Autenticación)

Secuencia de Autenticación

- 1. Extracción del Header:** Parsea 'Authorization: Bearer <token>' mediante RFC 6750. Si falta o está mal formado → 401.
- 2. Verificación Criptográfica:** Firma HS256 con JWT_SECRET, emisor (TOKEN_ISSUER) y audiencia (TOKEN_AUDIENCE).
- 3. Revalidación en Base de Datos:** Consulta ``UsersRepository.findById(sub)`` para asegurar que el usuario existe y mantiene ``status === 'active'``.
- 4. Inyección de Contexto:** Escribe en ``req.auth = { id, username, email, tokenId }`` para consumo de middlewares y controladores subsiguientes.

Revalidación Activa en BD (Stateless vs State)

¿Por qué consultar la BD si el JWT ya está firmado?

Un JWT es autocontenido y sigue siendo válido criptográficamente aunque la cuenta de usuario haya sido desactivada o eliminada.

La revalidación del estado del usuario (`status === 'active'`) en cada petición garantiza que la desactivación de un usuario invalide sus tokens de acceso INMEDIATAMENTE (401), sin esperar a que el token expire.

Código de Error Unificado: 401 Unauthorized

Cualquier fallo en la cadena de autenticación (falta de token, firma inválida, expiración o usuario inactivo) retorna homogéneamente un HTTP 401.

Anatomía de authorize.middleware.ts & Deny by Default

Resolución del Permiso

- 1. Verificación de Identidad:** Confirma que ``req.auth`` exista (garantizado si ``authenticate`` está delante; si falta → 401).
- 2. Normalización de la Petición:** Llama a ``normalizePath(req.originalUrl)`` para limpiar query strings, hashes y barras duplicadas.
- 3. Consulta de la Cadena RBAC:** Ejecuta ``ResourceRolesRepository.findEffectiveForUser(userId)`` recorriendo el grafo de 4 eslabones.
- 4. Evaluador de Coincidencia:** Usa ``isOperationGranted(permissions, method, path)`` comparando la petición contra los patrones del catálogo (ej. ``/api/clientes/:id``).

Postura de Seguridad: Deny by Default

Regla Binaria y Pesimista

Si no existe una concesión activa en ``resource_roles`` que cubra exactamente el método HTTP y el patrón de la ruta solicitada, la petición es rechazada de inmediato con HTTP 403 Forbidden.

- No existen comodines (*).
- No existen permisos implícitos.
- Una ruta nueva nace 100% cerrada hasta concederse explícitamente.

Distinción Crucial: 401 vs 403

- 401 Unauthorized: No sé quién eres (fallo de identidad).
- 403 Forbidden: Sé quién eres, pero no tienes concesión sobre este recurso.

Parche de las Rutas de Negocio (Protección Fase I)

Protección de las 5 Features de Negocio de la Fase I

Las 5 familias de rutas de negocio desarrolladas en la Fase I pasan de estar abiertas (OPEN) a estar estrictamente protegidas bajo la modalidad 3 (JWT + RBAC). Para lograrlo, únicamente se inyectan los middlewares `authenticate, authorize` antes del controlador en cada archivo `.routes.ts`.

Las 5 Features Parcheadas y sus Rutas:

1. **clients.routes.ts:** /api/clientes... (GET, POST, PUT, PATCH, DELETE) → Modalidad JWT + RBAC
2. **product-types.routes.ts:** /api/tipos-producto... (GET, POST, PUT, PATCH, DELETE) → Modalidad JWT + RBAC
3. **products.routes.ts:** /api/productos... (GET, POST, PUT, PATCH, DELETE) → Modalidad JWT + RBAC
4. **sales.routes.ts:** /api/ventas... (GET, POST, PUT, PATCH, DELETE) → Modalidad JWT + RBAC
5. **product-sales.routes.ts:** /api/detalle-ventas... (GET, POST, PUT, PATCH, DELETE) → Modalidad JWT + RBAC

Matriz de Decisión y Verificación HTTP (curl)

Matriz de Comportamiento HTTP

- GET /api/clientes (sin token) → HTTP 401 Unauthorized
- GET /api/clientes (token seller) → HTTP 200 OK (seller tiene concesión)
- POST /api/clientes (token seller) → HTTP 403 Forbidden (seller no tiene concesión)
- POST /api/clientes (token admin) → HTTP 201 Created (admin tiene los 58 permisos)
- Cualquier ruta (token alterado/exp) → HTTP 401 Unauthorized

Comprobación Eficiente de la Seguridad

Independencia de Controladores:

Si un controlador o servicio tuviera que verificar roles o permisos manualmente, el patrón estaría roto.

Aislamiento de Responsabilidades:

`authenticate` solo se preocupa de la identidad; `authorize` solo se preocupa de la matriz RBAC.

Transparencia para el Cliente:

La respuesta de error indica claramente si se requiere autenticar (401) o solicitar concesión de permisos (403).

Crterios de Cierre GATE (DoD del ISS-13)

Checklist de Aceptación Obligatorio:

[x] Criterio 18.1: ``authenticate.middleware.ts`` valida token Bearer, firma HS256, issuer/audience, expiración y status active del usuario.

[x] Criterio 18.2: ``authorize.middleware.ts`` resuelve (method, path) con ``resource-match.ts`` y aplica Deny by Default (403 sin concesión).

[x] Criterio 18.3: Barrel ``src/features/auth/access/index.ts`` reexporta ambos middlewares de acceso.

[x] Criterio 18.4: Las 5 features de negocio (clients, product-types, products, sales, product-sales) aplican ``authenticate, authorize``.

[x] Criterio 18.5: Documentadas las tres modalidades de acceso y sus códigos de respuesta HTTP en la guía y en Swagger.

[x] Criterio 18.6: Verificación funcional: sin token → 401; seller creando cliente → 403; admin creando cliente → 201.

[x] Verificación de Tipos: ``npx tsc --noEmit`` ejecuta sin errores de compilación y ``npm run dev`` levanta el servidor sin fallos.

 **HABILITA: ISS-14 (Feature RefreshTokens: Sesiones renovables, opacas y revocables).**