

FASE II — SEGURIDAD Y CONTROL DE ACCESO (RBAC)

ISS-09: Base de Seguridad Compartida y Modelos Auth

Infraestructura criptográfica (bcrypt + JWT HS256), utilidades compartidas en src/shared/auth/ y arquitectura relacional de las 6 tablas RBAC.

Fase II: Transición hacia un Modelo de Seguridad Transversal y Robusto

Evolución de la Arquitectura (Fase I vs. Fase II)

- **Fase I (Business):** Rutas públicas abiertas ('SIN AUTH') enfocadas en lógica comercial de negocio.
- **Fase II (Auth + RBAC):** Capa transversal de seguridad, identidades, roles, recursos y sesiones revocables.
- **Aislamiento Arquitectónico:** Las utilidades de seguridad se construyen en `src/shared/auth/` antes de proteger las rutas de negocio.
- **Compatibilidad con Fase I:** Ninguna tabla o ruta existente en la Fase I se altera durante el desarrollo del ISS-09.

Entregables Principales del ISS-09

- ✓ **Primitivas Criptográficas:** Hasheo bcrypt con 12 rondas de salting y hashes SHA-256 para tokens.
- ✓ **Gestión de JWT (HS256):** Firma y verificación de Access Tokens leyendo variables de entorno (`.env`).
- ✓ **Modelado de Datos RBAC:** Creación e integración de las 6 tablas fundamentales de autorización.
- ✓ **Asociaciones Centralizadas:** Configuración de relaciones M:N en `rbac.associations.ts` mediante `import side-effect`.

Primitivas Criptográficas en src/shared/auth/

1. Hasheo & Claves (password.ts)

- **bcrypt (12 Rondas):** Hasheo asíncrono con factor de costo 12 para credenciales de usuario.
- **Función hashPassword():** Recibe clave en texto plano y genera el hash seguro.
- **Función verifyPassword():** Compara clave plana contra hash almacenado de forma segura.
- **Hash SHA-256:** Genera resúmenes deterministas para Refresh Tokens opacos.

2. Tokens JWT HS256 (jwt.ts)

- **Firma HS256:** Genera Access Tokens usando JWT_SECRET y expiración configurable (JWT_EXPIRES_IN).
- **Contrato AuthPayload:** Define el payload mínimo con user_id, username, role_id y email.
- **Función generateToken():** Instancia y firma tokens firmados simétricamente.
- **Función verifyToken():** Valida firma, vigencia y decodifica el payload.

3. Matching de Rutas (resource-match.ts)

- **Coincidencia Dinámica:** Compara URLs solicitadas con rutas parametrizadas en BD.
- **Expresiones Regulares:** Convierte patrones como /api/clientes/:id a Regex.
- **Normalización:** Elimina query strings y trailing slashes en la comparación.
- **Base para Middleware:** Alimenta el middleware authorize para el matcheo RBAC.

El Modelo Relacional RBAC: Las 6 Tablas de Autenticación

Tabla 1: User (users)

Identidades de usuario con username, email, password_hash y status ('active'/'inactive').

Tabla 2: Role (roles)

Catálogo de roles del sistema (ej. ADMIN, SELLER) con nombre único y descripción.

Tabla 3: Resource (resources)

Catálogo de endpoints protegidos por par único UQ(method, path).

Tabla 4: RoleUser (role_users)

Tabla pivote M:N que asigna roles a usuarios con marca temporal y estado.

Tabla 5: ResourceRole (resource_roles)

Tabla pivote M:N que concede permisos de recursos a roles (Matriz RBAC).

Tabla 6: RefreshToken (refresh_tokens)

Sesiones de usuario guardando únicamente el hash SHA-256 (token_hash) y family_id.

Asociaciones Centralizadas e Importación Side-Effect

Configuración del Grafo RBAC en Sequelize

- **Centralización M:N:** El archivo `rbac.associations.ts` declara todas las relaciones pertenece-a y tiene-muchos entre los 6 modelos.
- **User <-> Role:** Relación N:M a través de `RoleUser` (`user.belongsToMany(Role, { through: RoleUser })`).
- **Role <-> Resource:** Relación N:M a través de `ResourceRole` (`role.belongsToMany(Resource, { through: ResourceRole })`).
- **User -> RefreshToken:** Relación 1:N donde un usuario posee múltiples sesiones/refresh tokens.

Patrón: Import Side-Effect

- ⚡ **¿Qué es?:** Es la importación de un módulo únicamente para ejecutar su código de inicialización sin capturar exportaciones.
- ⚡ **Sintaxis en `src/config/index.ts`:** `import '../features/auth/rbac.associations';`
- ⚡ **Momento de Ejecución:** Se ejecuta al instanciar la clase `App`, asegurando que el motor de Sequelize conozca las claves foráneas antes de sincronizar o consultar.
- ⚡ **Resultado:** Garantiza consultas de inclusión (`include: [Role, Resource]`) limpias y tipadas.

Coincidencia Dinámica de Rutas HTTP (resource-match.ts)

El Desafío de los Parámetros Dinámicos en la Matriz RBAC

En la base de datos, las rutas se registran con marcadores de parámetros (ej. GET /api/clientes/:id). Sin embargo, las peticiones HTTP entrantes traen valores reales (ej. GET /api/clientes/25). La utilidad resourceMatch resuelve esta discrepancia mediante Expresiones Regulares.

Reglas de Normalización de Rutas

1. Limpieza de Query Strings (?search=val -> /path).
2. Remoción de Trailing Slashes (/api/clientes/ -> /api/clientes).
3. Conversión de :param a patrón Regex (^/api/clientes/[^/]+\$).
4. Comparación Insensible a Mayúsculas/Minúsculas.

Ejemplos de Matching Validado

- Request: GET /api/clientes/15
Recurso BD: GET /api/clientes/:id --> MATCH OK
- Request: PUT /api/productos/99?fast=true
Recurso BD: PUT /api/productos/:id --> MATCH OK
- Request: POST /api/ventas
Recurso BD: POST /api/ventas --> MATCH OK

Verificación Unitario-Funcional y Sincronización

1. Unicidad de Hashes bcrypt (password.ts)

Verificar que dos llamadas a `hashPassword('123456')` generen hashes con salting distinto pero ambos retornen `true` en `verifyPassword()`.

2. Firmas y Vencimiento JWT (jwt.ts)

Generar un token firmado con `generateToken()` y comprobar que `verifyToken()` retorne el `AuthPayload` exacto (`user_id`, `username`, `role_id`, `email`).

3. Sincronización de Tablas en BD

Comprobar con `sequelize.sync()` la creación limpia de las 6 tablas relacionales (`users`, `roles`, `resources`, `role_users`, `resource_roles`, `refresh_tokens`).

Criterios de Cierre GATE — ISS-09 (DoD)

Definition of Done (DoD) — Checklist Obligatorio

- [x] **Compilación TypeScript Cero Errores:** Ejecución exitosa de `npx tsc --noEmit`.
- [x] **Módulo `src/shared/auth/` Completo:** Existencia de `password.ts`, `jwt.ts`, `resource-match.ts` y `auth-user.ts`.
- [x] **Modelos RBAC Creados:** Modelos Sequelize creados para User, Role, Resource, RoleUser, ResourceRole y RefreshToken.
- [x] **Asociaciones Sincronizadas:** Archivo `rbac.associations.ts` importado en `src/config/index.ts` sin conflictos.
- [x] **Base de Datos Actualizada:** 6 nuevas tablas creadas en el motor de BD sin afectar las tablas de negocio.

Siguiente Hito: ISS-10

Con la base de seguridad y las 6 tablas relacionales listas, el ISS-10 construirá el CRUD de Identidades de Usuario (Users) con hasheo de claves, aislamiento de credenciales con `toUserResponse` y cambio de contraseñas.

 **HABILITADO: ISS-10 (Users)**