

APRENDIZAJE-EXPRESS — ISS-10

Feature Users

(Identidades, Hashes y Permisos Efectivos)

Fase II: Auth con RBAC | CRUD por capas con bcrypt (12 rondas), DTOs sin password y consulta de matriz

Propósito del ISS-10: Identidades de Usuario y Contrato de Seguridad

Objetivos de la Capa de Identidad

- **Gestión de Identidades:** CRUD completo sobre la tabla `users` mediante patrón en 4 capas (Repository, Service, Controller, Routes).
- **Invariante del Hash:** La contraseña NUNCA entra ni sale en claro de la aplicación ni de las respuestas HTTP.
- **Bcrypt en Hooks:** El hash se calcula en los hooks `beforeCreate` y `beforeUpdate` del modelo `User` con coste de 12 rondas.
- **Rutas JWT + RBAC:** Declaración formal de las 9 rutas protegidas con middlewares `authenticate` y `authorize`.

Puntos Clave de Arquitectura

- **Doble Red de Protección:** Repository excluye `password` por `WITHOUT\_PASSWORD` y el mapper `toUserResponse` lo remueve explícitamente.
- **Control de Duplicados (409):** `assertUnique` valida `username` y `email` en el Service respondiendo HTTP 409 sin reventar la BD con 500.
- **Operaciones No-CRUD:** Soporta Cambio de Contraseña (`PATCH /:id/password`) y Consulta de Permisos Efectivos (`GET /:id/permisos`).
- **Desacoplamiento RBAC:** `getEffectivePermissions` delega en `ResourceRolesService` (ISS-12), manteniendo separado Users de la matriz.

Estructura por Capas y Contrato DTO del Feature Users

1. DTOs & Mappers

- CreateUserDto
- UpdateUserDto / Patch
- ChangePasswordDto
- UserResponseDto
- Mapper ``toUserResponse`` (Remueve password)

2. Repository

- UsersRepository
- Excluye ``password`` por defecto
- ``findByIdWithPassword``
- ``findByIdentifierWithPassword``
- ``findConflicts`` (username/email)

3. Service

- UsersService
- Valida unicidad (``assertUnique``)
- Maneja ``changePassword``
- Consulta ``getEffectivePermissions``
- Aplica ``findOrFail`` (404 si inactive)

4. Controller & Routes

- UsersController (BaseController)
- Manejo de errores con ``this.run``
- UsersRoutes (9 endpoints)
- Modalidad JWT + RBAC
- ``authenticate`` + ``authorize``

Seguridad del Hash de Contraseña y la Doble Red de Protección

BCRYPT 12 Rondas en Hooks del Modelo

- 1. Coste 12 Rondas:** Compromiso óptimo entre consumo de CPU en el servidor y resistencia contra ataques de fuerza bruta.
- 2. Hooks ``beforeCreate`` / ``beforeUpdate``:** Intercepta cualquier creación o edición en el modelo ``User`` para calcular el hash automáticamente.
- 3. Normalización ``beforeValidate``:** ``username`` y ``email`` se convierten a minúsculas y sin espacios antes de guardar.
- 4. Hash Indescifrable:** Bcrypt incluye un salt aleatorio en el propio string de 60 caracteres. ``comparePassword`` reevalúa en memoria.

Doble Red Anti-Filtración de Contraseñas

- ✓ **Red 1: Repository ``WITHOUT_PASSWORD``:** Las consultas estándar (``findAllActive``, ``findById``) aplican ``attributes: { exclude: ['password'] }` a nivel SQL.
- ✓ **Red 2: Mapper ``toUserResponse``:** En la transformación a DTO, se extrae `{ password, ...safe } = user.toJSON()`` asegurando que la clave nunca viaje en el JSON HTTP.
- ✓ **Lecturas Explicitas ``WithPassword``:** Solo dos métodos del repository leen el hash: ``findByIdWithPassword`` (cambio de clave) y ``findByIdentifierWithPassword`` (login).

Operaciones Especiales: Cambio de Contraseña y Permisos Efectivos

Cambio de Contraseña (`PATCH /:id/password`)

- No es un `PATCH` genérico: tiene su propio DTO `ChangePasswordDto` y endpoint dedicado.
- Exige la contraseña actual (`current\_password`) y la nueva (`new\_password`).
- Obtiene la credencial almacenada mediante `findByldWithPassword(id)`.
- Compara la contraseña actual en memoria con `comparePassword(plain, hash)`.
- Si la contraseña actual no coincide, responde HTTP 400 Bad Request.
- Al asignar `user.password = new\_password`, el hook `beforeUpdate` la vuelve a hashear.

Permisos Efectivos (`GET /:id/permisos`)

- Permite auditar los recursos reales que un usuario puede ejecutar según sus roles.
- Recorre el grafo completo RBAC mediante `ResourceRolesService.findEffectiveForUser(id)`.
- Exige que los 4 eslabones estén en estado `active`: User → RoleUser → Role → ResourceRole → Resource.
- Retorna la lista de pares `(method, path)` concedidos (ej. `GET /api/clientes`).
- Misma lógica interna utilizada por el middleware `authorize` en el ISS-13.

Seeder Canónico Idempotente y Especificación Swagger OpenAPI

Seeder de Usuarios Canónicos (`users.seeder.ts`)

- Usuario `admin` (ADMIN (58 recursos)):

Clare: `Admin123!` | Acceso total al sistema

- Usuario `seller` (SELLER (7 recursos)):

Clare: `Seller123!` | Operación de catálogo y ventas

- Idempotencia & Reconciliación: Usa `findOrCreate` por `username` y reactiva el usuario si estaba inactivo.
- Extras con Faker: Si `count > 2`, genera usuarios aleatorios sin rol para verificar que reciben 403.

Documentación OpenAPI (`users.swagger.ts`)

- Documenta los 9 endpoints del feature bajo la etiqueta 'Usuarios'.
- Aplica `security: [{ bearerAuth: [] } ]` en todas las operaciones (modalidad JWT + RBAC).
- Reutiliza respuestas de error estándar: 401 Unauthorized, 403 Forbidden, 404 Not Found, 409 Conflict.
- Define los esquemas JSON: `User`, `UserCreate`, `UserUpdate`, `UserPatch`, `ChangePassword`.
- Indica explícitamente que la respuesta del servidor NUNCA incluye el campo `password`.

Criterios de Cierre y Validación GATE (DoD del ISS-10)

✅ Checklist de Aceptación y Definición de Hecho (DoD)

- ✅ **1. Estructura DTO y Mappers:** Seis DTOs creados en `dto/`. `UserResponseDto` y `toUserResponse` excluyen la contraseña.
- ✅ **2. Repositorio Seguro:** Filtro `WITHOUT\_PASSWORD` en consultas de API. Lectura de hash exclusiva en metodos `WithPassword`.
- ✅ **3. Unicidad y Diagnóstico (409):** Creación/edición con `username` o `email` duplicado responde 409 Conflict explícito sin reventar la BD.
- ✅ **4. Cambio de Contraseña Protegida:** `PATCH /:id/password` exige `current\_password`, la compara en memoria y rehasea con bcrypt.
- ✅ **5. Seeder Canónico Idempotente:** Ejecución de `npm run db:seed` genera/reconcilia `admin` y `seller` sin duplicar filas.
- ✅ **6. Verificación de Compilación y Arranque:** `npx tsc --noEmit` responde sin errores y `npm run dev` arranca el servidor correctamente.