

ISS-07: Feature Product (Productos + Clave Foránea)

Capa de Datos & FK

Modelo Product vinculado mediante 'product_type_id' a ProductType con declaración explícita de asociaciones Sequelize.

Validación Cruzada

ProductsService inyecta ProductTypesRepository para verificar existencia (404) y estado activo (400) antes de guardar.

Cableado & Swagger

Exposición de /api/productos (SIN AUTH), seeder enlazado con tipos de producto y documentación OpenAPI 3.0.3.

Propósito del ISS-07 e Integración de Relaciones Relacionales

Objetivos Clave del Módulo

- Construir el CRUD completo del feature 'products' bajo el prefijo /api/productos.
- Introducir la primera Clave Foránea (FK) del sistema: 'product_type_id' referenciando a 'product_types'.
- Implementar la validación entre repositorios/servicios sin generar acoplamiento circular.
- Garantizar que no se pueda registrar un producto con un tipo de producto inactivo o inexistente.
- Declarar las asociaciones bidireccionales en 'products.associations.ts'.

Esquema Relacional Product ↔ ProductType

1. Tabla 'product_types'

Contiene los registros padre (ej: 'Electrónica', 'Bebidas') con estado 'active'/'inactive'.

2. Tabla 'products'

Contiene los registros hijo con la columna FK 'product_type_id' apuntando a 'product_types.id'.

3. Relación Sequelize

Product.belongsTo(ProductType) y
ProductType.hasMany(Product).

4. Carga Mapeada

Soporte de 'include: [{ model: ProductTypeModel }] ' en el repositorio para retornar la entidad anidada.

Capa de Datos: Modelo Product y DTOs de Contrato

Campos del Modelo Sequelize ('products')

- **id:** INTEGER, PRIMARY KEY, AUTO_INCREMENT
- **product_type_id:** INTEGER, FK (references product_types.id)
- **name:** STRING(100), NOT NULL
- **code:** STRING(50), UNIQUE, NOT NULL
- **description:** TEXT, NULLABLE
- **price:** DECIMAL(10,2), NOT NULL
- **stock:** INTEGER, DEFAULT 0
- **status:** ENUM('active','inactive'), DEFAULT 'active'
- **created_at / updated_at:** DATE, TIMESTAMPS AUTOMÁTICOS

DTOs de Entrada y Salida

CreateProductDto

Exige product_type_id, name, code, price y stock inicial.

UpdateProductDto

Reemplazo completo PUT de las propiedades editables del producto.

PatchProductDto

Actualización parcial PATCH (ej. modificar únicamente precio o stock).

ProductResponseDto

Estructura de salida JSON enmascarando campos internos e incluyendo la entidad anidada productType si se requiere.

Regla de Negocio: Inyección de ProductTypesRepository

Paso 1: Petición HTTP

POST /api/productos
Payload con
product_type_id: 5

Paso 2: ProductsService

Injecta
ProductTypesRepository.f
indById(5)

Paso 3: Verificación Estado

¿Existe y status ==
'active'?

Paso 4: Resultado

Si NO existe -> 404
AppError
Si Inactivo -> 400
AppError
Si OK -> Persiste en BD

Asociaciones Sequelize: products.associations.ts

Patrón de Import Side-Effect y Declaración de Relaciones

- **Archivo dedicado:** 'src/features/business/products/products.associations.ts' encapsula las asociaciones relacionales del modelo Product.
- **Declaración belongsTo:** ProductModel.belongsTo(ProductTypeModel, { foreignKey: 'product_type_id', as: 'productType' });
- **Declaración hasMany:** ProductTypeModel.hasMany(ProductModel, { foreignKey: 'product_type_id', as: 'products' });
- **Import Side-Effect:** Se importa en 'src/config/index.ts' como 'import \"../features/business/products/products.associations\";' para registrar las relaciones en Sequelize antes de sincronizar la BD.
- **Ventaja de Diseño:** Evita dependencias circulares directas entre archivos de modelo y centraliza la configuración ORM.

Seeder con Faker & Documentación Swagger

Seeder de Productos (products.seeder.ts)

- Dependencia previa: Consulta los 'product_types' existentes en la BD.
- Manejo de vacíos: Si no existen tipos de productos, ejecuta primero 'productTypesSeeder()'.
- Generación Faker: Genera códigos únicos, nombres comerciales, precios y stock aleatorios.
- Idempotencia: Trunca o limpia la tabla 'products' antes de sembrar.

Especificación Swagger (products.swagger.ts)

- Etiqueta OpenAPI: Tag 'Productos (SIN AUTH)'.
- Rutas Documentadas: /api/productos y /api/productos/{id} (GET, POST, PUT, PATCH, DELETE).
- Esquemas OpenAPI: Definición de ProductSchema y ProductInputSchema.
- Registry Central: Registrado en 'src/swagger/index.ts' e inspeccionable en /api/docs.

Cableado e Integración Global en el Backend

1. Rutas (routes/index.ts)

```
router.use('/productos', productRoutes);
```

2. Asociaciones (config/index.ts)

```
import  
'../features/business/products/products.associations';
```

3. Seeders (seeders/index.ts)

Invocación de productsSeeder(counts.products) dentro del runner.

4. Swagger (swagger/index.ts)

Consolidación de productSwagger en el registro OpenAPI global.

Crterios de Cierre GATE & Definición de Terminado (DoD)

Checklist de Verificación Obligatoria (ISS-07)

- [✓] **Compilador Estricto:** `npx tsc --noEmit` ejecuta con 0 errores de tipos en TypeScript.
- [✓] **Base de Datos Sincronizada:** Tabla 'products' creada en la BD con la restricción FK hacia 'product_types'.
- [✓] **Validación Cruzada Operativa:** Intento de crear producto con `product_type_id` inactivo o inexistente retorna 400 o 404 respectivamente.
- [✓] **Pruebas REST HTTP:** Ejecución exitosa de peticiones en 'products.http' (CRUD completo).
- [✓] **Seeder & Swagger Probados:** '`npm run db:seed`' siembra productos con FKs válidas y Swagger UI expone /api/productos en modo SIN AUTH.