
Aprendizaje-Express-ISS-02

Infraestructura de Base de Datos y ORM Sequelize

Configuración multi-motor (.env), módulo db.ts y reserva de seeders para la Fase I (Business)

ISS-02 Establece el Canal de Datos Aislado de la Lógica de Negocio

Propósito del ISS-02

- Instalar los drivers de Sequelize para 4 motores de BD relacionales.
- Declarar el archivo .env con la variable selectora DB_ENGINE.
- Crear el módulo src/database/db.ts para instanciar Sequelize y probar conexión.
- Reservar la ubicación src/database/seeders/ (sin lógica aún).
- Asegurar la compilación estricta de tipos con npx tsc --noEmit.

Integración en la Arquitectura

- Viene de ISS-01: reutiliza la clase App y la estructura de carpetas src/.
- Habilita ISS-03: entrega la instancia 'sequelize' lista para definir el modelo Client.
- Sin datos todavía: ISS-02 solo monta la tubería; las tablas se crean al llegar modelos.
- Sincronización automática: el método dbConnection() ejecuta sync({ force, alter }).

Bloque 1: Paquetes y Drivers para Soporte Multi-Motor en Tiempo de Ejecución

Sequelize 6

ORM base para mapeo objeto-relacional

 **sequelize@^6.37.8**

MySQL / MariaDB

Driver nativo recomendado para el laboratorio

 **mysql2@^3.24.4**

PostgreSQL

Driver Postgres + parser hstore

 **pg@^8.23.0 + pg-hstore**

SQL Server / Oracle

Drivers nativos para enterprise BD

 **tedious / oracledb**

Anatomía del Archivo .env y la Variable DB_ENGINE



Contenido del .env

```
PORT=4000
# Variable selectora de motor
DB_ENGINE=mysql

# Bloque MySQL
MYSQL_HOST=localhost
MYSQL_USER=admin
MYSQL_PASSWORD=MiNiCo57**
MYSQL_NAME=tecnogua
MYSQL_PORT=3306

# Bloque Postgres / MSSQL / Oracle...
```



Reglas del Manejo de Entorno

✓ Aislamiento de Secretos

Las credenciales residen fuera del repositorio en .env. Se recomienda versionar solo un .env.example.

✓ Cambio Transparente

Cambiar de MySQL a PostgreSQL exige unicamente modificar DB_ENGINE=postgres en el .env sin alterar código fuente.

✓ Multi-bloque Preconfigurado

Declarar las credenciales de todos los motores deja la infraestructura lista para migraciones inmediatas.

Bloque 2: Estructura del Módulo src/database/db.ts

Mapeo y Selección en db.ts

1. Interface DatabaseConfig: Define el contrato con dialect, host, username, password, database y port.
2. Diccionario dbConfigurations: Objeto Record<string, DatabaseConfig> que lee variables process.env con fallbacks seguros.
3. Selección selectedEngine: Toma process.env.DB_ENGINE || 'mysql' y valida que exista en el mapa dbConfigurations.
4. Instanciación Sequelize: Crea new Sequelize(...) configurando pool (max:5, min:0, acquire:30000, idle:10000) y logging.
5. Validación de Errores: Si DB_ENGINE apunta a un motor no registrado, lanza Error('Motor de base de datos no soportado').

Las Tres Exportaciones Centrales de src/database/db.ts

`sequelize`

Instancia ORM Singleton

Objeto principal de Sequelize con el pool de conexiones configurado. Utilizado por los modelos y el método `sync()`.

`getDatabaseInfo()`

Metadatos sin Secretos

Función que retorna `{ engine, config, connectionString }`. Permite auditar la cadena de conexión sin exponer credenciales.

`testConnection()`

Diagnóstico de Autenticación

Función asíncrona que ejecuta `sequelize.authenticate()`. Retorna booleano (`true/false`) sin propagar excepciones.

Bloque 3: Reserva de la Ubicación src/database/seeder/

Estado en ISS-02

La carpeta src/database/seeder/ se crea mediante 'mkdir -p' y queda totalmente reservada y VACÍA (opcionalmente con .gitkeep).

No contiene scripts *.seeder.ts ni orquestador Runner en esta fase.

Evolución Posterior (ISS-04)

Separar infraestructura de sembrado evita acoplamientos:

- En ISS-04 se integrará @faker-js/faker.
- Se creará counts.ts para controlar la cantidad de registros falsos.
- Se creará index.ts (SeedersRunner) para ejecutar la siembra idempotente.

Integración del Módulo db.ts en la Clase App (src/config/index.ts)

Método dbConnection() en la Clase App

1. Lectura de Metadatos: Llama a `getDatabaseInfo()` para imprimir el motor activo en consola.
2. Comprobación de Conexión: Ejecuta `'await testConnection()'`. Si falla, aborta con `process.exit(1)`.
3. Sincronización ORM: Ejecuta `'await sequelize.sync({ force, alter: !force })'`.
4. Manejo de Claves Foráneas: En MySQL deshabilita temporalmente `FOREIGN_KEY_CHECKS` durante el sync.
5. Control por Entorno: La variable `DB_SYNC_FORCE=true` recrea tablas desestimando estructuras previas.

Regla de Oro: Orden de Arranque Estricto en listen()

```
1° await this.dbConnection()    —>    2° await  
    this.app.listen(port)
```

Primero la Infraestructura de Datos, después la Atención de Peticiones HTTP.

? ¿Por qué no abrir el puerto antes de terminar el sync()?

Si se abre el puerto HTTP antes de terminar el 'sync({ alter: true })', las sentencias DDL (ALTER TABLE, DROP/ADD FOREIGN KEY) compiten con las peticiones entrantes concurrentes. Esto provoca deadlocks y errores intermitentes de FK en la base de datos.

Comandos de Verificación y Diagnóstico del ISS-02



Comandos de Verificación

- **Compilación de Tipos:**

```
npx tsc --noEmit
```

- **Comprobar Archivos Creados:**

```
test -f src/database/db.ts && test -f .env &&  
test -d src/database/seeder
```

- **Prueba de Arranque y Sync:**

```
npm run dev
```

Diagnóstico de Fallos Habituales

- ✗ **ECONNREFUSED**

Motor de BD no iniciado o puerto incorrecto en .env.

- ✗ **Access denied for user**

Credenciales (MYSQL_USER/MYSQL_PASSWORD) erróneas.

- ✗ **Motor no soportado**

Valor en DB_ENGINE no existe en mapa dbConfigurations.

Matriz de Criterios GATE de Cierre (DoD del ISS-02)

Checklist de Aceptación Consolidado

- [x] 3.1 Paquetes Sequelize y drivers (mysql2, pg, pg-hstore, tedious, oracledb) instalados. — Dependencias listas para multi-motor.
- [x] 3.1 Archivo .env existente con PORT, DB_ENGINE y bloques de credenciales. — Variables de entorno configuradas.
- [x] 3.2 Archivo src/database/db.ts creado y exportando sequelize, getDatabaseInfo y testConnection. — Singleton de infraestructura listo.
- [x] 3.3 Carpeta src/database/seeder/ existente y reservada (sin lógica aún). — Ubicación de seeders fijada.
- [x] Compilador TypeScript en verde con 'npx tsc --noEmit' (0 errores de tipos). — Verificación estricta de compilación.
- [x] Servidor 'npm run dev' inicia e intenta conectar a la base de datos sin crash. — Prueba de ejecución en caliente.

¿Qué habilita el Cierre del ISS-02?

ISS-03 — Feature Client (CRUD por capas: A...E)

Al disponer del canal de conexión en db.ts:

1. Nace la primera cadena vertical completa: HTTP → Controller → Service → Repository → Model → Sequelize → BD.
2. Se crea la primera tabla real 'clients' en la base de datos mediante el sync automatizado.
3. Se integran los hooks de modelo para el hashéo de contraseñas con bcryptjs.