TRABAJO PRACTICO INTEGRADOR DE PROGRAMACION II

Alumno: Daniel Alberto jimenez Valderrama
DNI: 95899862
Comisión: 12

Grupo: 48 DispositivoIoT--ConfiguracionRed

Dominio (elegir 1 pareja A → B)

Rol: Arquitectura y Diseño

Se debe elegir entre una de estas opciones para resolver el TPI.

Seleccionado:

DispositivoIoT → ConfiguracionRed

1. Diagrama UML de clases con:
    - Atributos (tipo/visibilidad) y métodos (firma/visibilidad).
    - A → B (1..1) unidireccional explícita.
    - Paquetes y dependencias principales.

Diagrama UML:

Clase A (DispositivoIoT): Contiene un atributo de referencia a la Clase B.

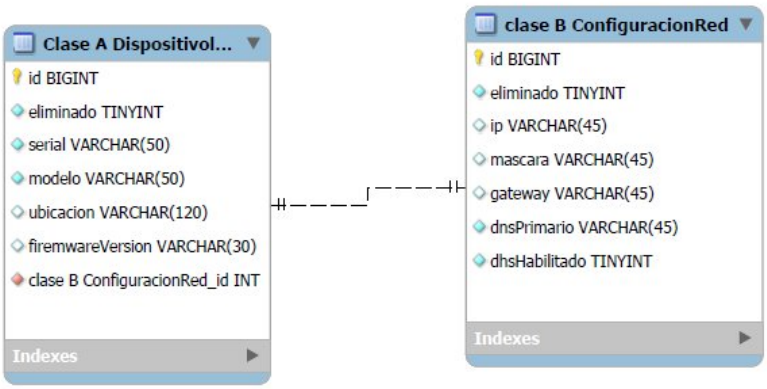
Atributos:

- id (Long - PK)
- eliminado (Boolean - Baja lógica)
- serial (String - NOT NULL, UNIQUE, máx. 50)
- modelo (String - NOT NULL, máx. 50)
- ubicación (String - máx. 120)
- firmwareVersion (String - máx. 30)

configuracionRed (ConfiguracionRed - Referencia 1-1 a B)

Clase B (ConfiguracionRed):

- id (Long - PK)
- eliminado (Boolean - Baja lógica)
- ip (String - máx. 45)
- máscara (String - máx. 45)
- gateway (String - máx. 45)
- dnsPrimario (String - máx. 45)
- dhcpHabilitado (Boolean - NOT NULL)

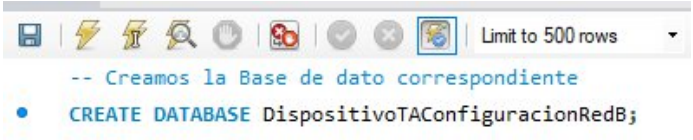


El diagrama ilustra la arquitectura de capas y la relación unidireccional 1→1.

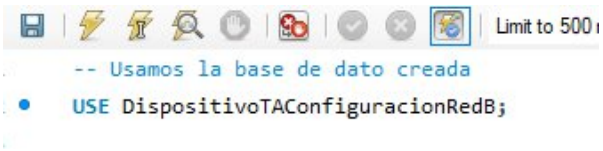
- **Relación:** DispositivoIoT tiene una asociación unidireccional 1..1 con ConfiguracionRed (la flecha va de A hacia B).
- **Capas:** Se muestran las dependencias: AppMenu llama a AService, que usa ADao y BDao, y todos dependen de las entities/

Diseño de la Base de Datos (MySQL):

Para garantizar la relación 1→1, aplicamos la clave foránea única en la tabla de B, quedando de la siguiente manera:



Usamos la Base de datos para crear las Tablas clase A y clase B:



- **Creamos las tablas Entidad A (DispositivoloT):**

SQL File 4* | SQL File 4* | SQL File 5* | SQL File 6*

Limit to 500 rows

```sql
-- Usamos la base de dato creada
USE DispositivoTAConfiguracionRedB;

-- Creamos las tablas de entidad A (DisposituvoloT)

CREATE TABLE DispositivoIoT (
id BIGINT PRIMARY KEY AUTO_INCREMENT,
serial VARCHAR(50) NOT NULL UNIQUE,
modelo VARCHAR(50) NOT NULL,
ubicacion VARCHAR(120),
firmware_version VARCHAR(30),
eliminado BOOLEAN NOT NULL DEFAULT FALSE
);
```

- **Creamos la Entidad B (ConfiguracionRed):**

```sql
-- Usamos la base de dato creada
USE DispositivoTAConfiguracionRedB;

-- Creamos la tabla de entidad B (Configuracion Red)
CREATE TABLE ConfiguracionRed (
id BIGINT PRIMARY KEY AUTO_INCREMENT,
ip VARCHAR(45),
mascara VARCHAR(45),
gateway VARCHAR(45),
dns_primario VARCHAR(45),
dhcp_habilitado BOOLEAN NOT NULL,
eliminado BOOLEAN NOT NULL DEFAULT FALSE,
-- Clave Foránea Única: Asegura la relación 1:1.
-- Cada ConfiguracionRed pertenece a un y solo un DispositivoIoT.
dispositivo_id BIGINT NOT NULL UNIQUE,
FOREIGN KEY (dispositivo_id) REFERENCES DispositivoIoT(id)
ON DELETE CASCADE -- Elimina B si se elimina A físicamente
);
```

Script SQL de datos de prueba:

- **Insertamos un dispositivo a la entidad A**

```sql
USE DispositivoTAConfiguracionRedB;

-- Hacemos Scrip para datos de prueba (INSERT)

-- Insertar Dispositivo IoT (A)
INSERT INTO DispositivoIoT (serial, modelo, ubicacion, firmware_version)
VALUES ('S2024-001', 'SensorTemp-v2', 'Almacén 1', '2.1.0');

SELECT *
FROM DispositivoIoT
WHERE serial = 'S2024-001';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	serial	modelo	ubicacion	firmware_version	eliminado
1	S2024-001	SensorTemp-v2	Almacén 1	2.1.0	0
NULL	NULL	NULL	NULL	NULL	NULL

- **Insertamos una configuración de red (B):**

```sql
USE DispositivoTAConfiguracionRedB;

-- Hacemos Scrip para datos de prueba (INSERT)

-- Insertar Configuración de Red (B) para el Dispositivo (id=1)
INSERT INTO ConfiguracionRed (ip, mascara, gateway, dns_primario, dhcp_habilitado, dispositivo_id)
VALUES ('192.168.1.10', '255.255.255.0', '192.168.1.1', '8.8.8.8', FALSE, 1);

SELECT *
FROM ConfiguracionRed
WHERE ip = '192.168.1.10';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	ip	mascara	gateway	dns_primario	dhcp_habilitado	eliminado	dispositivo_id
1	192.168.1.10	255.255.255.0	192.168.1.1	8.8.8.8	0	0	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- **Insertamos otro dispositivo a la entidad A:**

Limit to 500 rows

```sql
USE DispositivoTAConfiguracionRedB;

-- Hacemos Scrip para datos de prueba (INSERT)

-- Insertar otro Dispositivo IoT (A)
INSERT INTO DispositivoIoT (serial, modelo, ubicacion, firmware_version)
VALUES ('S2024-002', 'Camara-v1', 'Oficina Principal', '1.5.3');

SELECT *
FROM DispositivoIoT
WHERE serial = 'S2024-002';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	serial	modelo	ubicacion	firmware_version	eliminado
2	S2024-002	Camara-v1	Oficina Principal	1.5.3	0
NULL	NULL	NULL	NULL	NULL	NULL

Conclusión:

Podemos decir que la fase de diseño ha concluido exitosamente con la definición formal del modelo, donde se establecieron las entidades de dominio con sus atributos y restricciones. Se creó el **script** que utiliza una clave foránea única en la tabla ConfiguracionRed para garantizar la relación 1:1, un punto clave de la arquitectura. Este trabajo sienta las **bases sólidas y validadas** para que las siguientes fases del proyecto puedan proceder con el desarrollo de servicios y la capa de acceso a datos.

ALUMNO: IGNACIO MATEO DENUNCIATO

ROL ESPECIALISTA EN PERSISTENCIA (DAO)

En mi sección del trabajo desempeñé el rol de Especialista en Persistencia, cuya responsabilidad principal es implementar toda la capa de acceso a datos (DAO) y asegurar que los distintos módulos del sistema puedan guardar, modificar, consultar y eliminar información en la base de datos MySQL de manera segura y estructurada.

Mi trabajo se centró especialmente en la persistencia de dos entidades fundamentales del sistema IoT:

1. ConfiguraciónRed

2. DispositivoIot

Para ambas entidades desarrollé:

● La clase de entidad (mapeo del objeto Java → tabla SQL).

● Los DAO genéricos y las interfaces concretas.

● Las implementaciones JdbcDAO, que realizan las operaciones SQL reales.

● La integración con la base de datos a través de DatabaseConnection.

● Un servicio (DispositivoIotService) que utiliza los DAO para realizar transacciones completas.

● Un Test (TestNacho) para demostrar y validar el funcionamiento.

FUNCIONAMIENTO DE MI MÓDULO (Persistencia)

Mi módulo es el encargado de:

✔ 1. Conectar con la base MySQL

A través de la clase:

DatabaseConnection.getConnection()

que establece la conexión usando el driver oficial de MySQL.

✔ 2. Crear, leer, actualizar y eliminar datos (CRUD)

Esto se implementa mediante el patrón DAO (Data Access Object).

Desarrollé:

● ConfiguracionRedDao + JdbcConfiguracionRedDao

● DispositivoIotDao + JdbcDispositivoIotDao

Cada uno incluye métodos como:

● crear()

● leer()

● leerTodos()

● actualizar()

● eliminar()

● y búsquedas especiales (buscarPorIp, buscarPorSerial)

Estos DAO permiten que el resto del sistema trabaje con objetos Java sin preocuparse por el SQL.

✔ 3. Mantener integridad entre Dispositivo y Configuración

Un DispositivoIot depende de una ConfiguraciónRed.

Implementé un servicio:

DispositivoIotService

que permite crear un dispositivo y su configuración asociada en una sola transacción:

● Se inserta primero la configuración.

● Luego se asocia al dispositivo.

● Finalmente se inserta el dispositivo.

Si algo falla:

se hace rollback, garantizando que no queden datos inconsistentes.

Esto asegura atomicidad en la operación.

✔ 4. Test de funcionamiento

El archivo TestNacho.java ejecuta todo el flujo:

1. Crear una ConfiguraciónRed.

2. Crear un DispositivoIot asociado.

3. Guardarlos dentro de una transacción.

4. Confirmar en consola la creación exitosa.

Esto sirve como demostración funcional de la persistencia.

CÓMO SE INTEGRA MI PARTE CON EL RESTO DEL SISTEMA

Mi módulo actúa como la base fundamental sobre la cual los otros integrantes construyen.

En particular:

✔ 1. Capa de Servicios (compañero responsable)

Mi módulo entrega métodos (DAO + Service) que permiten:

● Consultar dispositivos.

● Asociar configuraciones.

● Actualizar firmware.

● Marcar dispositivos como eliminados.

Los servicios dependen directamente del código que yo desarrollé.

✓ 2. Capa de Presentación o Controladores

Otro compañero puede usar mi Service para:

● Mostrar la lista de dispositivos.

● Agregar nuevos dispositivos al sistema.

● Buscar por IP o Serial.

● Actualizar configuraciones.

Todo esto se logra gracias a los métodos que implementé.

✓ 3. Base de Datos (otro integrante)

Aunque la base fue creada por otro compañero, mi módulo es quien realmente la usa, enviando:

● INSERT

● SELECT

● UPDATE

● DELETE

Esto une directamente el código Java con MySQL.

✓ 4. Documentación y pruebas

Mi test integrado permite que el equipo:

● Verifique que la base funciona,

● Que los datos se guardan correctamente,

● Y que la transacción completa se ejecuta sin errores.

RESUMEN:

Mi rol fue desarrollar el módulo de Persistencia del sistema. Implementé las entidades, las interfaces DAO y sus implementaciones JDBC para las tablas configuracion_red y dispositivo_iot. También desarrollé un servicio transaccional que permite crear dispositivos junto con su configuración de red de forma atómica. Finalmente realicé un test de integración (TestNacho) que demuestra el correcto

guardado de los datos en MySQL. Mi módulo se integra con las capas de servicio, presentación y con el

modelo de base de datos, permitiendo que todo el sistema pueda consultar y persistir información de

manera consistente

LÓGICA DE NEGOCIO

Intégrate: Luis Enrique Denegri
Rol: Especialista en Lógica de Negocio
Responsabilidad: Implementación de Service Layer

Mi parte del trabajo

Como responsable de la capa de servicio, implementé el núcleo de la lógica de negocio del sistema, ubicándome entre la interfaz de usuario (AppMenu) y la capa de acceso a datos (DAO).

Archivos Implementados

service/

- **GenericService.java - Interfaz genérica con operaciones CRUD**
- **DispositivoIotService.java - Servicio principal con transacciones**
- **ConfiguracionRedService.java - Servicio de ConfiguracionRed**

exceptions/

- **ServiceException.java - Excepción general**
- **ValidationException.java - Excepción de validación**

Responsabilidades Principales

1. Orquestación de Transacciones

Implementé transacciones atómicas que garantizan que al crear un DispositivoIoT, su ConfiguracionRed asociada también se cree correctamente, o ninguna de las dos se persista.

Flujo implementado:

conn.setAutoCommit(false) // Inicio de transacción

crear ConfiguracionRed // Crear entidad B
crear DispositivoIoT // Crear entidad A
conn.commit() // Confirmar si todo OK
conn.rollback() // Revertir si hay error

2. Validaciones de Negocio

Implementé validaciones exhaustivas:

Serial: obligatorio, único, formato S2024-XXX

Modelo: obligatorio, máximo 50 caracteres

IPs: formato IPv4 válido cuando DHCP está deshabilitado

Firmware: formato vX.Y.Z (ej: v2.1.0)

Relación 1→1: cada DispositivoIoT debe tener exactamente una ConfiguracionRed

3. Búsqueda por Campo Relevante

Implementé el método buscarPorSerial(String serial) como requisito del TP, permitiendo localizar dispositivos por su identificador único.

4. Manejo de Excepciones

Creé una jerarquía de excepciones que proporciona mensajes claros:

ServiceException: errores generales del sistema

ValidationException: errores en validaciones de datos

REGLAS DE NEGOCIO IMPLEMENTADAS

1. Unicidad de Serial: No pueden existir dos dispositivos con el mismo serial

2. Relación 1→1 Obligatoria: Todo dispositivo debe tener configuración de red

3. Configuración Manual vs DHCP: Si DHCP está deshabilitado, IP/máscara/gateway son obligatorios

4. Baja Lógica en Cascada: Al eliminar un dispositivo, su configuración también se marca como eliminada

INTEGRACIÓN CON OTRAS CAPAS

Con capa DAO (inferior):

Los servicios usan interfaces DAO, no implementaciones concretas

Los DAOs reciben Connection externa para participar en transacciones

Requisito crítico comunicado a compañero de DAO

Con capa de Presentación (superior):

AppMenu instancia los servicios con inyección de dependencias

Manejo de excepciones para mostrar errores al usuario

Documentación detallada proporcionada al compañero de UI

FLUJO DE UNA OPERACIÓN COMPLETA

Usuario (AppMenu)
↓
DispositivoIotService
↓ validar datos
↓ abrir transacción
↓
ConfiguracionRedDao.crear(config, conn)
↓ INSERT en ConfiguracionRed (BD)
↓
DispositivoIotDao.crear(dispositivo, conn)
↓ INSERT en DispositivoIoT (BD)
↓
conn.commit()
↓
Retorna dispositivo creado

MÉTODOS PRINCIPALES IMPLEMENTADOS

DispositivoIotService:

```
DispositivoIoT insertar(DispositivoIoT dispositivo)
DispositivoIoT actualizar(DispositivoIoT dispositivo)
void eliminar(Long id)
DispositivoIoT getById(Long id)
List<DispositivoIoT> getAll()
DispositivoIoT buscarPorSerial(String serial)  // Requerido por TP
```

EJEMPLO DE VALIDACIÓN IMPLEMENTADA

```
// Validar formato de serial con expresión regular
private boolean validarFormatoSerial(String serial) {
   String serialPattern = "^S\\d{4}-\\d{3}$";
   return serial.matches(serialPattern);
   // Ejemplos válidos: S2024-001, S2025-999
}

// Validar unicidad
if (existeSerialEnBD(dispositivo.getSerial())) {
   throw new ValidationException(
```

```
        "Ya existe un DispositivoIoT con el serial: " + serial
    );
}

// Validar relación 1→1
if (dispositivo.getConfiguracionRed() == null) {
    throw new ValidationException(
        "DispositivoIoT debe tener una ConfiguracionRed asociada"
    );
}
```

COORDINACION Y DESARROLLO

Alumno: Nicolas Copertino

Comision 8

Rol lider de proyecto y documentacion

1. Introducción

La clase AppMenu representa la interfaz de usuario mediante consola para la gestión de la entidad DispositivoIoT y su relación con ConfiguraciónRed, ambos almacenados en una base de datos MySQL.

Su función principal es permitir al usuario realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar lógicamente) y ejecutar una búsqueda por un campo relevante, en este caso el serial del dispositivo.

El menú utiliza un servicio (DispositivoIotService), el cual a su vez se conecta a la base de datos utilizando JDBC a través del DAO correspondiente.

2. Funcionalidades del menú

La clase implementa un menú interactivo con seis opciones principales:

1) Crear DispositivoIoT (transaccional)

Permite registrar un nuevo dispositivo en la base de datos.

El usuario ingresa manualmente:

● Serial (único)

● Modelo

● Ubicación

● Versión de firmware

Antes de persistir los datos, el sistema valida:

● Campos de texto no vacíos

● Violación de unicidad → evita seriales duplicados

Si ocurre un intento de duplicado, se muestra el mensaje:

"✕Error: Serial ya existe (violación de unicidad)."

2) Leer DispositivoIoT por ID

El usuario ingresa un ID.

El sistema convierte el valor y lo consulta en la base de datos mediante

service.getById(id).

Manejo de errores:

● Si el usuario ingresa texto → " Debe ingresar un número válido."

● Si el ID no existe → "✕No existe un dispositivo con ese ID."

3) Listar todos los DispositivosIoT

Muestra todos los registros existentes, incluyendo los que tienen baja

lógica (eliminado = true).

Si la tabla está vacía:

" La base de datos está vacía."

4) Actualizar DispositivoIoT

El usuario ingresa el ID del dispositivo y el sistema valida que exista.

Permite modificar:

● Modelo

● Ubicación

● Versión de firmware

El sistema muestra el valor actual y permite dejar el campo vacío para no modificarlo.

Si el ID no existe → mensaje de error.

5) Eliminar DispositivoIoT (baja lógica)

La eliminación no borra el registro físicamente.

Si el ID no existe o ya estaba eliminado:

"✕No existe el dispositivo o ya estaba eliminado."

6) Buscar DispositivoIoT por Serial (campo relevante)

El usuario ingresa un serial.

El sistema consulta mediante:

service.getBySerial(serial)

Si no se encuentra coincidencia:

"✕No existe un dispositivo con ese serial."

Esta funcionalidad cumple con el requisito de búsqueda por campo único o relevante definido en el modelo.

3. Manejo robusto de errores

El programa incluye un enfoque sólido para evitar fallos comunes:

✔Validación de opciones del menú

Usa Integer.parseInt dentro de un try/catch.

✔Validación de IDs numéricos

Evita fallos por texto ingresado en campos numéricos.

✔Manejo de excepciones SQL

Captura errores provenientes del DAO/Service como:

● Violación de unicidad (serial duplicado)

● Registros inexistentes

● Problemas de conexión

✔Inputs vacíos en actualización

Permite mantener valores actuales al dejar el campo vacío.

4. Integración con la base de datos

Este menú trabaja junto con:

● DispositivoIotService

● DAO JDBC

● MySQL con las tablas:

o DispositivoIoT

o ConfiguracionRed (relación 1:1)

Todas las operaciones CRUD del menú se traducen en:

● INSERT

● SELECT

● UPDATE

● UPDATE eliminado = true (baja lógica)

Esto garantiza que el programa esté totalmente conectado y operando

sobre la base de datos del usuario.