Trabajo Práctico Integrador - Programación 2

Sistema de Gestión de Dispositivos IoT

Universidad Tecnológica Nacional

Tecnicatura Universitaria en Programación

Integrantes: - Tiseira, Gustavo Hernán - Rol: Líder Técnico - Vergara, David Emanuel - Rol: Desarrollador - López, Mauricio Eduardo - Rol: Desarrollador

Fecha de Entrega: Noviembre 2025

Repositorio: https://github.com/brujoh88/tpi-prog2-iot

---

1. INTRODUCCIÓN

1.1 Descripción del Dominio

El proyecto implementa un sistema de gestión de dispositivos IoT (Internet of Things) con configuración de red asociada, modelando una relación 1:1 unidireccional entre DispositivoIoT y ConfiguracionRed.

Se eligió este dominio por su **relevancia práctica** en entornos industriales y por permitir **reutilizar el modelo de datos del TFI de Base de Datos I**, lo que permitió enfocarse en la calidad del código Java y la correcta implementación de transacciones.

1.2 Objetivos

Objetivo General: Desarrollar una aplicación Java que implemente persistencia mediante JDBC, utilizando el patrón DAO y Service Layer, con manejo robusto de transacciones.

Objetivos Específicos: - Modelar relación 1:1 unidireccional con integridad referencial - Implementar CRUD completo con PreparedStatement - Gestionar transacciones con commit/rollback explícito - Aplicar arquitectura por capas bien definidas - Implementar baja lógica y validaciones en capa de servicios

---

2. DISEÑO

2.1 Modelo de Datos

Relación **1:1 unidireccional** de DispositivoIoT hacia ConfiguracionRed:

```
DispositivoIoT (1)     > (1) ConfiguracionRed
```

DispositivoIoT: id, eliminado, serial (UNIQUE), modelo, ubicacion, firmwareVersion, configuracionRed

ConfiguracionRed: id, eliminado, ip (UNIQUE), mascara, gateway, dnsPrimario, dhcpHabilitado

2.2 Decisiones de Diseño

Relación 1:1 - FK Única Se implementó mediante **Foreign Key única** en ConfiguracionRed:

```
ALTER TABLE ConfiguracionRed
ADD COLUMN dispositivo_id BIGINT NOT NULL UNIQUE,
ADD CONSTRAINT fk_configuracion_dispositivo
    FOREIGN KEY (dispositivo_id) REFERENCES DispositivoIoT(id)
    ON DELETE CASCADE;
```

Justificación: Más simple que PK compartida, el constraint UNIQUE garantiza la relación 1:1, y ON DELETE CASCADE mantiene integridad referencial.

Unidireccionalidad DispositivoIoT conoce su ConfiguracionRed, pero no viceversa. Esto **evita referencias circulares**, previene StackOverflow en toString(), y define responsabilidades claras.

Baja Lógica Campo `eliminado` en ambas tablas permite **preservar datos históricos** para auditoría y reportes, sin romper integridad referencial.

---

3. ARQUITECTURA

3.1 Estructura de Capas

```
CAPA MAIN (UI)                    ← AppMenu, Main

CAPA SERVICE (Negocio)            ← Services + Transacciones

CAPA DAO (Persistencia)           ← DAOs + PreparedStatement

CAPA CONFIG                       ← DatabaseConnection

    ↓ JDBC ↓

  MySQL
```

3.2 Responsabilidades por Capa

Config: Provee conexiones centralizadas mediante DatabaseConnection (Singleton), lee credenciales de `config.properties`.

Entities: POJOs con constructores, getters/setters, y toString() sin recursión.

DAO: Acceso exclusivo a BD mediante PreparedStatement. Recibe Connection externa (no crea ni cierra). Interface genérica `GenericDao<T>` con implementaciones concretas.

Service: Orquesta transacciones (commit/rollback), aplica validaciones de negocio, y garantiza reglas como la relación 1:1. Crea y cierra conexiones.

Main: Interfaz de consola con Scanner, manejo de excepciones, y validación de entradas. Solo llama a Service, nunca a DAO directamente.

---

4. PERSISTENCIA Y TRANSACCIONES

4.1 Patrón de Transacciones

Patrón utilizado en todas las operaciones complejas:

```
public void insertarDispositivoConConfiguracion(
    DispositivoIoT dispositivo,
    ConfiguracionRed configuracion) throws Exception {

    Connection conn = null;
    try {
        // 1. Obtener conexión
        conn = DatabaseConnection.getConnection();

        // 2. Iniciar transacción
        conn.setAutoCommit(false);

        // 3. Validar datos
        validarDispositivo(dispositivo);
        validarConfiguracion(configuracion);

        // 4. Crear DispositivoIoT primero para obtener su ID
        dispositivoIoTDao.crear(dispositivo, conn);

        // 5. Setear el dispositivo_id en ConfiguracionRed
        configuracion.setDispositivoId(dispositivo.getId());

        // 6. Crear ConfiguracionRed con el dispositivo_id válido
        configuracionRedDao.crear(configuracion, conn);
```

```
            // 7. Asociar configuración al dispositivo en memoria
            dispositivo.setConfiguracionRed(configuracion);

            // 8. Commit si todo OK
            conn.commit();

        } catch (Exception e) {
            // 9. Rollback en caso de error
            if (conn != null) {
                try {
                    conn.rollback();
                } catch (SQLException ex) {
                    ex.printStackTrace();
                }
            }
            throw e;

        } finally {
            // 9. Restaurar autocommit y cerrar
            if (conn != null) {
                try {
                    conn.setAutoCommit(true);
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

4.2 Garantías ACID

Atomicidad: Operaciones múltiples se ejecutan como unidad, cualquier fallo revierte todo.

Consistencia: Validaciones previas + constraints de BD (FK, UNIQUE, CHECK) aseguran integridad.

Aislamiento: READ_COMMITTED (default MySQL), cada transacción opera independientemente.

Durabilidad: `conn.commit()` persiste en disco, InnoDB garantiza durabilidad con WAL.

4.3 Manejo de Errores SQL

Código	Error	Acción
1062	Duplicate entry	`DuplicateEntityException` con mensaje específico
1452	FK constraint fails	`EntityNotFoundException`
3819	Check constraint violated	`ValidationException` (formato IP inválido)

4.4 PreparedStatement

Todas las consultas usan PreparedStatement para **prevenir SQL injection**, mejorar rendimiento (cache de plan), y mantener claridad separando SQL de datos.

---

5. VALIDACIONES Y REGLAS DE NEGOCIO

5.1 Validaciones por Entidad

DispositivoIoT: - Serial: no nulo, no vacío, único en BD - Modelo, ubicación: no nulos, no vacíos

ConfiguracionRed: - IP: formato IPv4 válido, única en BD - Máscara, gateway, DNS: formato IPv4 válido - DHCP: si está deshabilitado, IP es requerida

5.2 Regla de Relación 1:1

Nivel BD: `dispositivo_id BIGINT NOT NULL UNIQUE`

Nivel Service: Verificar antes de asociar que ni el dispositivo ni la configuración estén ya relacionados.

---

6. PRUEBAS

6.1 Casos de Prueba

CRUD Exitoso: Crear, leer, listar, actualizar, eliminar lógico - todos funcionan correctamente recuperando también la relación 1:1.

Validaciones: Serial duplicado, IP duplicada, IP inválida, campos vacíos - todas las excepciones lanzan mensajes claros.

Transacciones: - Commit exitoso: ambas entidades persisten - Rollback por serial duplicado: ninguna persiste - Rollback por IP duplicada: ninguna persiste

Evidencia de Rollback:

```
-- Antes: SELECT COUNT(*) FROM ConfiguracionRed;  → 10
-- Intento con serial duplicado (crea config primero)
-- Después: SELECT COUNT(*) FROM ConfiguracionRed;  → 10 (sin cambios)
```

---

7. CONCLUSIONES

7.1 Aprendizajes Principales

1. **Arquitectura por capas:** Separación de responsabilidades facilitó desarrollo colaborativo
2. **Transacciones JDBC:** Control manual más complejo pero con mayor control que JPA
3. **PreparedStatement:** Esencial para seguridad y rendimiento
4. **Relación 1:1 con FK única:** Más simple que alternativas
5. **Baja lógica:** Valiosa para auditoría, requiere disciplina en consultas

7.2 Dificultades Encontradas

Orden de creación: Determinar qué crear primero en transacciones. Solución: crear ConfiguracionRed primero, luego asociar y crear DispositivoIoT.

Recursión en toString(): Solución: DispositivoIoT.toString() solo muestra ID de configuración, no el objeto completo.

7.3 Mejoras Futuras

1. **Pool de conexiones** (HikariCP)
2. **Testing automatizado** (JUnit 5)
3. **Logging estructurado** (SLF4J + Logback)
4. **Interfaz gráfica** (JavaFX)

7.4 Trabajo en Equipo

División clara: Gustavo (arquitectura, transacciones), David (entities, DAOs), Mauricio (services, menú). Git con branches facilitó colaboración sin conflictos.

---

8. REFERENCIAS

8.1 Documentación Técnica

1. **Java SE 21 Documentation** - https://docs.oracle.com/en/java/javase/21/
2. **MySQL 8.0 Reference Manual** - https://dev.mysql.com/doc/refman/8.0/en/
3. **JDBC Tutorial (Oracle)** - https://docs.oracle.com/javase/tutorial/jdbc/

8.2 Patrones de Diseño

4. **Fowler, Martin.** "Patterns of Enterprise Application Architecture" - Patrones DAO, Service Layer
5. **Gamma et al.** "Design Patterns" - Patrón Singleton

8.3 Herramientas Utilizadas

6. **Apache NetBeans IDE** - Entorno de desarrollo
7. **MySQL Workbench 8.0** - Gestión de base de datos
8. **Git + GitHub** - Control de versiones
9. **PlantText** (https://www.planttext.com/) - Diagramas UML

8.4 Integración con TFI de Base de Datos I

11. **Video TFI Base de Datos I** - https://www.youtube.com/watch?v=PwBVHe8esg&t=635s

8.5 Uso de Inteligencia Artificial

Durante el desarrollo se utilizó **Claude Code** (Anthropic) como asistencia en: - Generación de estructura inicial de clases DAO - Sugerencias de manejo de excepciones SQL - Revisión de patrones de transacciones - Generación de documentación

Todo el código fue revisado, comprendido y adaptado por el equipo. Las decisiones de diseño y lógica de negocio fueron tomadas por los integrantes.

Herramienta: Claude 4.5 Sonnet (Claude Code CLI) **Fecha:** Noviembre 2025 **Alcance:** Asistencia en código, no desarrollo autónomo

---

ANEXOS

A. Configuración del Entorno

Archivo: config.properties

```
db.url=jdbc:mysql://localhost:3306/iot
db.user=root
db.password=[PASSWORD]
db.driver=com.mysql.cj.jdbc.Driver
```

B. Estructura del Repositorio

```
proyecto-tpi-prog2/
  README.md
  docs/
      INFORME_TECNICO.md
```

```
    GUION_VIDEO.md
    diagrama-uml.png
sql/
    schema.sql
    data.sql
src/
    config/
    entities/
    dao/
    service/
    exceptions/
    util/
    main/
lib/
    mysql-connector-j-8.0.33.jar
.gitignore
```

Nota: Proyecto desarrollado con Apache NetBeans IDE

Fin del Informe

Fecha de elaboración: Noviembre 2025 Versión: 1.0