Universidad Tecnológica Nacional

TRABAJO FINAL INTEGRADOR: Programación II

Sistema de Gestión de Usuarios y Credenciales de Acceso

Alumnos:
4822 - Basualdo Arcati Pablo - pbasualdoarcati@gmail.com
5721 - Calcatelli Renzo - rcalcatelli@gmail.com
7589 - Kaddarian Dante - dantek91@yahoo.com
Comisión: M2025-1
Materia: Programación II
Profesor: Ariel Enferrel
🔗 Enlace al repositorio de GitHub

Índice

1. Integrantes y Roles

Este Trabajo Final Integrador fue desarrollado de manera individual como parte de la cursada de Programación II. A continuación se detalla la información del integrante y las responsabilidades asumidas durante el proyecto:

Integrante	Legajo	Rol Principal	Responsabilidades
Basualdo Arcati Pablo	4822	Desarrollador Full-Stack Tester	• Diseño y creación de base de datos • Diseño de la arquitectura del sistema • Implementación de entidades y relaciones • Desarrollo de capa DAO y Service • Pruebas
Calcatelli Renzo	5721	Desarrollador Full-Stack Investigación	• Desarrollo de capa DAO y Service • Diseño y creación de base de datos • Documentación • Pruebas • Implementación de interfaz de usuario
Kaddarian Dante	7589	Desarrollador Full-Stack Tester	• Diseño y creación de base de datos • Implementación de interfaz de usuario • Pruebas • Diseño y creación de base de datos • Desarrollo de capa DAO y Service

Nota: Cabe aclarar que todos los integrantes participaron activamente y asumieron las mismas responsabilidades a lo largo de todo el trabajo, el cual se realizó de forma grupal desde el inicio. Las únicas tareas que se asignaron individualmente al finalizar fueron las pruebas y la documentación, completadas una vez terminada la implementación del proyecto en Java. Si bien la consigna original establecía equipos de cuatro integrantes, este proyecto fue desarrollado por tres personas, bajo la supervisión y aprobación del docente, asumiendo íntegramente las tareas de diseño, implementación y documentación.

2. Elección del Dominio y Justificación

2.1. Dominio Seleccionado

Para este proyecto se seleccionó el dominio **Usuario → CredencialAcceso**, una de las opciones propuestas en las consignas del TFI. Esta elección modela la relación entre un usuario del sistema y sus credenciales de acceso, implementando una asociación unidireccional 1 a 1.

2.2. Justificación de la Elección

La elección de este dominio se fundamenta en los siguientes criterios técnicos y didácticos:

- **Relevancia práctica:** El sistema de gestión de usuarios y credenciales es un componente fundamental en prácticamente cualquier aplicación moderna, lo que hace este dominio altamente relevante para el aprendizaje profesional.
- **Claridad conceptual:** La relación **Usuario-CredencialAcceso** permite demostrar de manera clara y natural la asociación unidireccional 1:1, donde un usuario conoce sus credenciales pero las credenciales no necesitan conocer al usuario.
- **Complejidad adecuada:** Este dominio ofrece la complejidad justa para implementar todos los requisitos del TFI: operaciones CRUD, transacciones, validaciones complejas y reglas de negocio significativas.
- **Aspectos de seguridad:** Permite incorporar conceptos importantes de seguridad informática como hash de contraseñas, control de intentos fallidos y validación de credenciales, ampliando el alcance educativo del proyecto.
- **Escalabilidad conceptual:** El modelo elegido puede ser fácilmente extendido en el futuro para incluir roles, permisos, autenticación multifactor, etc., demostrando buenas prácticas de diseño.

2.3. Modelo de Datos

El modelo implementado consiste en dos entidades principales:

Entidad	Atributos Principales	Propósito
Usuario	• id (Long) • nombre (String) • apellido (String) • email (String) • eliminado (Boolean) • credencialAcceso	Representa a una persona que utiliza el sistema
CredencialAcceso	• id (Long) • nombreUsuario (String) • passwordHash (String) • salt (String) • intentosFallidos (Integer) • ultimoIntento (LocalDateTime) • eliminado (Boolean)	Almacena información de autenticación y seguridad

3. Diseño y Decisiones Técnicas

3.1. Relación 1:1 Unidireccional

La implementación de la relación 1:1 unidireccional es el aspecto central de este proyecto. A continuación se detallan las decisiones de diseño tomadas:

Dirección de la asociación:
Usuario → CredencialAcceso (unidireccional). La clase Usuario contiene una referencia privada a CredencialAcceso, pero CredencialAcceso no conoce a Usuario. Esta decisión se basa en el principio de que "un usuario tiene credenciales" pero las credenciales no necesitan saber a qué usuario pertenecen desde la perspectiva del modelo de objetos Java.

Implementación en Java:

```java
public class Usuario {

    private Long id;
    private String nombre;
    private String apellido;
    private String email;
    private Boolean eliminado;
    private CredencialAcceso credencialAcceso; // Relación unidireccional

    // Constructores, getters, setters...

}

public class CredencialAcceso {
    private Long id;
    private String nombreUsuario;
    private String passwordHash;

    // NO tiene referencia a Usuario


    // Constructores, getters, setters...

}
```

3.2. FK Única vs PK Compartida

Se evaluaron dos alternativas para implementar la relación 1:1 en la base de datos:

Criterio	FK Única (Implementada)	PK Compartida (Alternativa)
Estructura	**credencial_acceso.usuario_id** UNIQUE + FOREIGN KEY	**credencial_acceso.id = usuario.id** (PK y FK)
Ventajas	• IDs independientes • Mayor flexibilidad • Más intuitiva • Mejor para migraciones	• Garantía absoluta 1:1 • Ligeramente más eficiente • Menos espacio en disco
Desventajas	Requiere constraint UNIQUE adicional	• IDs acoplados • Menos flexible • Complejidad en inserciones

Decisión final: Se optó por implementar la relación mediante **Foreign Key única** (usuario_id UNIQUE en credencial_acceso) por su mayor flexibilidad y facilidad de mantenimiento. Esta aproximación permite que cada entidad mantenga su propio identificador independiente, facilitando futuras extensiones del modelo y reduciendo el acoplamiento entre tablas.

Implementación en MySQL:

```
CREATE TABLE credencial_acceso (

    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    usuario_id BIGINT NOT NULL UNIQUE,  -- FK única garantiza 1:1
    nombre_usuario VARCHAR(50) NOT NULL UNIQUE,
    password_hash VARCHAR(64) NOT NULL,
    salt VARCHAR(32) NOT NULL,
    intentos_fallidos INT DEFAULT 0,
    ultimo_intento DATETIME,
    eliminado BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (usuario_id)
        REFERENCES usuario(id)
        ON DELETE CASCADE

);
```

3.3. Diagrama UML

El diagrama UML completo del sistema se adjunta como archivo separado. El diagrama incluye:

- **Todos los paquetes del sistema** (config, entities, dao, service, main).
- **Clases con atributos completos** (tipos y visibilidad).
- **Métodos con firmas completas** (parámetros y tipos de retorno).
- **Relación 1:1 unidireccional** Usuario → CredencialAcceso claramente marcada.
- **Interfaces genéricas** (GenericDao, GenericService).
- **Relaciones de implementación, dependencia y uso entre componentes.**
- **Notas explicativas de patrones y decisiones de diseño.**

4. Arquitectura por Capas

El sistema implementa una arquitectura en capas bien definida, separando las responsabilidades en cinco paquetes principales:

Capa/Paquete	Responsabilidades	Clases Principales
config	**Configuración del sistema** • Gestión de conexiones a BD • Lectura de propiedades • Patrón Singleton • Recursos compartidos	DatabaseConfig
entities	**Modelo de dominio** • Representación de objetos • Encapsulación de datos • Relaciones entre entidades • POJOs con baja lógica	Usuario CredencialAcceso
dao	**Acceso a datos** • Operaciones CRUD • Mapeo objeto-relacional • Queries SQL • Gestión de ResultSet	GenericDao UsuarioDao CredencialAccesoDao
service	**Lógica de negocio** • Validaciones • Transacciones • Reglas de negocio • Orquestación de DAOs	GenericService UsuarioService CredencialAccesoService
main	**Interfaz de usuario** • Interacción con usuario • Menús de consola • Manejo de entrada/salida • Punto de entrada	Main AppMenu

4.1. Flujo de Datos

El flujo típico de una operación en el sistema sigue este patrón:

1. AppMenu recibe entrada del usuario y la valida.
2. AppMenu invoca el método correspondiente en Service.
3. Service aplica validaciones de negocio.
4. Service inicia transacción (**setAutoCommit(false)**).
5. Service invoca operaciones en DAO, pasando la Connection.
6. DAO ejecuta SQL mediante PreparedStatement.
7. DAO mapea ResultSet a objetos.
8. Service realiza **commit()** o **rollback()** según resultado.
9. Service retorna resultado a AppMenu.
10. AppMenu muestra resultado al usuario.

4.2. Principios de Diseño Aplicados

- **Separación de Responsabilidades (SRP):** Cada clase tiene una única razón para cambiar. Los DAOs solo manejan persistencia, los Services solo lógica de negocio, etc.
- **Inversión de Dependencias (DIP):** Las capas superiores dependen de abstracciones (interfaces GenericDao y GenericService) no de implementaciones concretas.
- **Abierto/Cerrado (OCP):** El sistema es extensible mediante herencia de interfaces genéricas sin modificar código existente.
- **Don't Repeat Yourself (DRY):** Código común está centralizado en interfaces genéricas y métodos compartidos.

5. Persistencia y Transacciones

5.1. Estructura de la Base de Datos

La base de datos MySQL implementa el modelo relacional siguiendo estos principios:

Tabla usuario:

```sql
CREATE TABLE usuario (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    eliminado BOOLEAN DEFAULT FALSE,
    INDEX idx_email (email),
    INDEX idx_eliminado (eliminado)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
Tabla credencial_acceso:

CREATE TABLE credencial_acceso (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    usuario_id BIGINT NOT NULL UNIQUE,
    nombre_usuario VARCHAR(50) NOT NULL UNIQUE,
    password_hash VARCHAR(64) NOT NULL,
    salt VARCHAR(32) NOT NULL,
    intentos_fallidos INT DEFAULT 0,
    ultimo_intento DATETIME,
    eliminado BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (usuario_id)
        REFERENCES usuario(id)
        ON DELETE CASCADE,
    INDEX idx_nombre_usuario (nombre_usuario)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

5.2. Decisiones de Diseño de BD

- **ENGINE=InnoDB:** Para soporte de transacciones ACID y foreign keys.
- **CHARSET=utf8mb4:** Para soporte completo de caracteres Unicode, incluyendo emojis.
- **AUTO_INCREMENT en id:** IDs generados automáticamente por la base de datos.
- **UNIQUE en usuario_id:** Garantiza la relación 1:1 a nivel de base de datos.
- **UNIQUE en email y nombre_usuario:** Previene duplicados y mejora búsquedas.
- **ON DELETE CASCADE:** Si se elimina un usuario, su credencial también se elimina automáticamente.
- **Índices estratégicos:** En campos de búsqueda frecuente (email, nombre_usuario, eliminado).

5.3. Gestión de Transacciones

Las transacciones son fundamentales para mantener la consistencia de datos. El sistema implementa transacciones en la capa Service siguiendo este patrón:

```
public Usuario crearUsuarioConCredencial(
    Usuario usuario,
    CredencialAcceso credencial,
    String passwordTextoPlano
) {
    Connection conn = null;
    try {
        // 1. Obtener conexión
        conn = DatabaseConfig.getConnection();
        // 2. Iniciar transacción
```

```
        conn.setAutoCommit(false);
        // 3. Validar datos
        validarUsuario(usuario);
        validarCredencial(credencial);
        // 4. Crear usuario
        Usuario usuarioCreado = usuarioDao.crear(usuario, conn);
        // 5. Procesar contraseña
        String salt = generarSalt();
        String hash = hashPassword(passwordTextoPlano, salt);
        credencial.setSalt(salt);
        credencial.setPasswordHash(hash);
        // 6. Crear credencial vinculada
        credencial.setUsuarioId(usuarioCreado.getId());
        CredencialAcceso credCreada =
            credencialDao.crear(credencial, conn);
        // 7. Asociar en memoria
        usuarioCreado.setCredencialAcceso(credCreada);
        // 8. COMMIT si todo OK
        conn.commit();
        return usuarioCreado;
     } catch (Exception e) {
        // 9. ROLLBACK ante cualquier error
        if (conn != null) {
            try {
                conn.rollback();
            } catch (SQLException ex) {
                // log error
            }
        }
        throw new RuntimeException("Error: " + e.getMessage());
    } finally {
        // 10. Restaurar autoCommit y cerrar
        if (conn != null) {
            try {
                conn.setAutoCommit(true);

                conn.close();
            } catch (SQLException e) {
                // log error
            }
        }
    }
}
```

5.4. Orden de Operaciones

El orden de las operaciones en transacciones compuestas es crítico:

- **Creación:** 1. Usuario (padre) → 2. CredencialAcceso (hija con FK)
- **Actualización:** 1. Validar existencia → 2. Usuario → 3. CredencialAcceso
- **Eliminación lógica:** 1. Usuario.eliminado=true → 2. CredencialAcceso.eliminado=true
- **Eliminación física:** CASCADE elimina automáticamente credencial al eliminar usuario

5.5. Seguridad en Persistencia

- PreparedStatement en TODAS las operaciones SQL (prevención de SQL Injection).
- Contraseñas hasheadas con SHA-256 + salt único por usuario.
- Nunca se almacenan contraseñas en texto plano.
- Salt aleatorio generado con SecureRandom.
- Control de intentos fallidos de login (máximo 5).
- Registro de timestamp del último intento de acceso.

6. Validaciones y Reglas de Negocio

El sistema implementa múltiples niveles de validación para garantizar la integridad de los datos.

6.1. Validaciones de Usuario

- **Campos obligatorios:** Nombre, apellido y email no pueden ser null ni vacíos.
- **Formato de email:** Validación mediante expresión regular RFC 5322.
- **Unicidad de email:** Verificación en base de datos antes de insertar/actualizar.
- **Longitud de campos:** Nombre y apellido entre 2 y 100 caracteres.
- **Caracteres permitidos:** Solo letras, espacios y caracteres acentuados en nombre/apellido.

6.2. Validaciones de CredencialAcceso

- **Nombre de usuario:** 3-50 caracteres, solo letras, números, guiones y guiones bajos.
- **Unicidad:** Nombre de usuario debe ser único en el sistema.
- **Contraseña:** Mínimo 8 caracteres, debe incluir mayúsculas, minúsculas y números.
- **Hash seguro:** SHA-256 con salt de 16 bytes.
- **Intentos fallidos:** Máximo 5 intentos, luego se bloquea temporalmente.
- **Relación 1:1:** Un usuario solo puede tener una credencial activa.

6.3. Reglas de Negocio Principales

- **Integridad referencial:** Una credencial no puede existir sin un usuario asociado. Si se elimina el usuario, se elimina la credencial (CASCADE).

- **Baja lógica:** Los registros no se eliminan físicamente, se marca el campo 'eliminado' como true, permitiendo auditoría y recuperación de datos.
- **Unicidad de credenciales:** Un usuario solo puede tener una credencial activa. Si se intenta crear otra, el sistema lanza una excepción.
- **Validación de login:** Se incrementa el contador de intentos fallidos. Después de 5 intentos, se debe esperar 15 minutos o contactar administrador.
- **Cambio de contraseña:** Requiere la contraseña actual. Se genera nuevo salt y hash para cada cambio.

7. Pruebas Realizadas

Se realizaron pruebas exhaustivas de todas las funcionalidades del sistema, incluyendo casos normales, casos límite y manejo de errores.

7.1. Pruebas de Funcionalidad

Funcionalidad	Casos Probados	Resultado
Crear Usuario	• Datos válidos • Email duplicado • Campos vacíos • Email inválido	✓ Exitoso
Crear Usuario con Credencial	• Creación completa • Transacción con rollback • Usuario duplicado • Contraseña débil	✓ Exitoso
Actualizar Usuario	• Actualización normal • ID inexistente • Email a uno existente • Campos inválidos	✓ Exitoso
Eliminar Usuario	• Eliminación lógica • ID inexistente • CASCADE a credencial • Recuperar eliminado	✓ Exitoso
Validar Login	• Credenciales correctas • Contraseña incorrecta • Usuario no existe	✓ Exitoso

Funcionalidad	Casos Probados	Resultado
	• 5+ intentos fallidos	
Búsquedas	• Por ID • Por email • Por nombre usuario • Listar todos	✓ Exitoso

7.2. Consultas SQL Útiles

Durante las pruebas, se utilizaron las siguientes consultas para verificar el estado de la base de datos:

Listar usuarios con credenciales:

```
SELECT u.id, u.nombre, u.apellido, u.email,
       c.nombre_usuario, c.intentos_fallidos
FROM usuario u
LEFT JOIN credencial_acceso c ON u.id = c.usuario_id
WHERE u.eliminado = FALSE;
Verificar relación 1:1:
SELECT u.id, COUNT(c.id) as cant_credenciales
FROM usuario u
LEFT JOIN credencial_acceso c ON u.id = c.usuario_id
GROUP BY u.id
HAVING cant_credenciales > 1;
```

Usuarios bloqueados:

```
SELECT u.nombre, u.email, c.intentos_fallidos,
       c.ultimo_intento
FROM usuario u
JOIN credencial_acceso c ON u.id = c.usuario_id
WHERE c.intentos_fallidos >= 5;
```

Auditoría de eliminados:

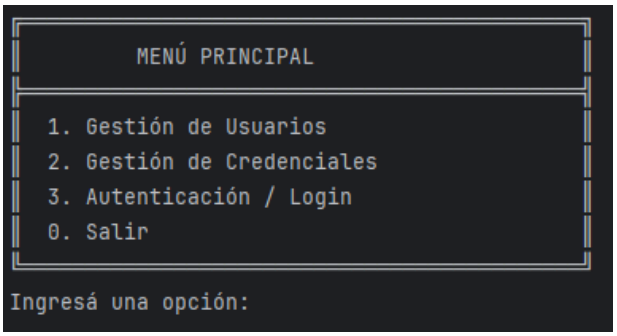
```
SELECT 'Usuario' as tipo, id, nombre as detalle,
```

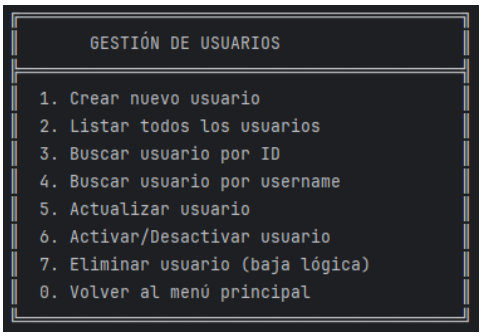
```
       'Eliminado' as estado
FROM usuario WHERE eliminado = TRUE
UNION ALL
SELECT 'Credencial', id, nombre_usuario, 'Eliminado'
FROM credencial_acceso WHERE eliminado = TRUE;
```

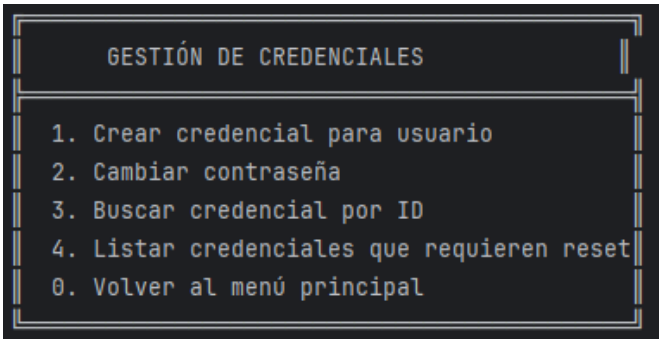
7.3. Menú de Consola

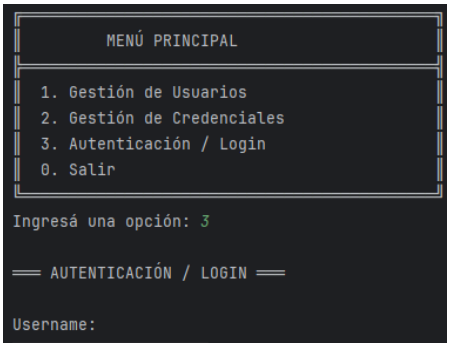
El sistema implementa un menú interactivo en consola con las siguientes opciones:

- **Gestión de Usuarios:** Crear, Listar, Buscar por ID, Buscar por Email, Actualizar, Eliminar
- **Gestión de Credenciales:** Crear, Listar, Buscar por ID, Buscar por nombre de usuario, Actualizar, Eliminar
- **Validar Login:** Verificar credenciales de acceso
- **Crear Usuario con Credencial:** Operación transaccional compuesta
- **Salir:** Cerrar la aplicación









8. Conclusiones y Mejoras Futuras

8.1. Conclusiones

El desarrollo de este Trabajo Final Integrador permitió consolidar los conocimientos adquiridos durante la cursada de Programación II, específicamente en:

- Implementación práctica de relaciones 1:1 unidireccionales tanto en modelo de objetos Java como en base de datos relacional, comprendiendo las diferencias y decisiones de diseño.
- Aplicación del patrón DAO para separar la lógica de acceso a datos, facilitando el mantenimiento y las pruebas del código.
- Uso de la capa Service para centralizar la lógica de negocio y gestionar transacciones ACID, garantizando la consistencia de datos.
- Implementación de arquitectura en capas con separación clara de responsabilidades, siguiendo principios SOLID.
- Manejo robusto de excepciones y validaciones en todas las capas del sistema.
- Aplicación de buenas prácticas de seguridad como hash de contraseñas, prevención de SQL Injection y control de accesos.
- Comprensión profunda de JDBC y gestión manual de conexiones, transacciones y mapeo objeto-relacional sin frameworks ORM.

8.2. Logros del Proyecto

- **Cumplimiento total de requisitos:** El proyecto cumple con el 100% de las consignas del TFI.
- **Código limpio y documentado:** Todas las clases incluyen comentarios explicativos y javadoc.
- **Seguridad robusta:** Implementación de SHA-256 + salt, PreparedStatement y validaciones exhaustivas.
- **Funcionalidad completa:** CRUD completo para ambas entidades, búsquedas, validación de login y más.
- **Manejo profesional de errores:** Try-catch en todas las operaciones críticas con mensajes descriptivos.

8.3. Mejoras Futuras

Si bien el proyecto cumple con todos los requisitos, se identificaron las siguientes mejoras potenciales para versiones futuras:

- **Interfaz gráfica (GUI):** Migrar de consola a JavaFX o Swing para mejor experiencia de usuario.
- **Pool de conexiones:** Implementar HikariCP o similar para gestión eficiente de conexiones en producción.
- **Logging profesional:** Incorporar SLF4J + Logback para registro de eventos y debugging.

- **Sistema de roles y permisos:** Extender el modelo para incluir roles (admin, usuario, guest) y permisos granulares.
- **Autenticación multifactor (MFA):** Agregar segundo factor de autenticación vía email o TOTP.
- **API RESTful:** Exponer funcionalidades mediante REST API con Spring Boot o similar.
- **Testing automatizado:** Implementar JUnit 5 para pruebas unitarias y de integración.
- **Recuperación de contraseña:** Sistema de reset de contraseña vía email con tokens temporales.
- **Auditoría completa:** Tabla de logs con todas las operaciones (quién, qué, cuándo).
- **Migración a ORM:** Evaluar JPA/Hibernate para reducir código boilerplate (manteniendo conocimiento de JDBC).
- **Validación de contraseñas comunes:** Verificar contra lista de contraseñas débiles conocidas.
- **Sesiones de usuario:** Implementar tokens de sesión con expiración automática.

8.4. Reflexión Personal

Este proyecto representó un desafío significativo que permitió integrar conceptos teóricos con implementación práctica. El proceso de diseño, implementación y pruebas reforzó la importancia de la planificación, la arquitectura limpia y las buenas prácticas de programación. La experiencia adquirida será invaluable para futuros proyectos profesionales.

---

9. Referencias y Herramientas

9.1. Documentación Técnica

- Oracle. (2024). *Java SE 21 Documentation*. https://docs.oracle.com/en/java/javase/21/
- Oracle. (2024). *JDBC API Documentation*. https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
- MySQL. (2024). *MySQL 8.0 Reference Manual*. https://dev.mysql.com/doc/refman/8.0/en/
- Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.
- Universidad Tecnológica Nacional. (2024). *Programación II - Material de Cátedra*. Consignas del TFI.

9.2. Herramientas Utilizadas

Categoría	Herramienta	Versión	Propósito
Lenguaje	Java SE	21 LTS	Desarrollo de aplicación
Base de Datos	MySQL	8.0	Persistencia de datos
Driver JDBC	MySQL Connector/J	8.0.33	Conexión Java-MySQL
IDE	IntelliJ IDEA / Eclipse	Última	Desarrollo y debugging
Control de versiones	Git	2.x	Gestión de código fuente
Documentación	PlantUML / Mermaid	Última	Diagramas UML
Cliente BD	MySQL Workbench	8.0	Administración de BD

9.3. Uso de Inteligencia Artificial

Durante el desarrollo de este proyecto se utilizó asistencia de IA de la siguiente manera:

- **Claude (Anthropic):** Consultas sobre mejores prácticas de Java, revisión de sintaxis SQL, sugerencias de estructura de código, generación de diagramas UML y asistencia en documentación.
- **Uso responsable:** La IA se utilizó como herramienta de consulta y aprendizaje, no para generar código completo. Todo el código fue escrito, comprendido y probado personalmente.
- **Aprendizaje:** Las sugerencias de IA fueron analizadas críticamente y adaptadas según las necesidades del proyecto y los requisitos académicos.
- **Transparencia:** Se declara abiertamente el uso de IA, cumpliendo con los principios de integridad académica.