

MEMORIA TÉCNICA DEL PROYECTO: SISTEMA DE GESTIÓN Y VENTA DE CALZADO (eSneakerPro)

Código del Proyecto: DAM2-PROY-202X-XX

Versión del Documento: 1.0

Fecha: 01/02/2026

1. Introducción y Objetivo del Proyecto

El presente documento constituye la memoria técnica del proyecto eSneakerPro, una aplicación de escritorio desarrollada para el módulo de Desarrollo de Interfaces (DI) o Entornos de Desarrollo (ED) de 2º de Desarrollo de Aplicaciones Multiplataforma (DAM). El objetivo principal es implementar un sistema integral de gestión (CRUD) para un comercio electrónico especializado en calzado, separando claramente las funcionalidades de administración del catálogo y usuarios, de las operaciones de compra realizadas por los clientes finales.

2. Análisis de Requisitos

2.1. Requisitos Funcionales (RF)

- RF-01: Gestión de Perfiles. El sistema debe soportar dos perfiles de usuario diferenciados: Administrador y Cliente.
- RF-02: Autenticación y Autorización. Los usuarios deben autenticarse mediante credenciales (usuario/contraseña). El sistema debe redirigir a la interfaz correspondiente según su perfil.
- RF-03: CRUD Completo de Zapatillas (Administrador). Los usuarios con perfil *Administrador* deben poder:
 - Crear nuevos registros de zapatillas (modelo, marca, talla, precio, stock, imagen).
 - Leer/Consultar la lista completa de productos.
 - Actualizar la información de cualquier producto existente.
 - Eliminar productos del catálogo.
- RF-04: Gestión de Usuarios (Administrador). Los administradores deben poder Leer la lista de usuarios registrados, Actualizar sus datos y Eliminar cuentas. (La creación de usuarios se realizará mediante registro público).

- RF-05: Registro de Clientes. Un usuario anónimo debe poder registrarse como *Cliente* proporcionando datos básicos.
- RF-06: Catálogo y Compra (Cliente). Los usuarios con perfil *Cliente* deben poder:
 - Navegar por un catálogo de productos disponible.
 - Añadir/eliminar productos a un carrito de compra temporal.
 - Simular un proceso de checkout, asociando un método de pago (tarjeta) y generando un pedido (*Order*).
- RF-07: Gestión de Métodos de Pago (Cliente). El cliente debe poder asociar y gestionar una o más tarjetas de crédito/débito (*Card*) a su perfil para realizar compras.

2.2. Requisitos No Funcionales (RNF)

- RNF-01: Arquitectura. Aplicación de escritorio con arquitectura cliente-servidor, donde el cliente es una aplicación Java y el servidor es un SGBD SQL.
- RNF-02: Usabilidad. La interfaz debe ser intuitiva, con tiempos de respuesta inferiores a 2 segundos para operaciones CRUD básicas y navegación fluida entre ventanas.
- RNF-03: Mantenibilidad. El código debe seguir convenciones de Java y estar estructurado en capas (presentación, lógica, persistencia) para facilitar su modificación y extensión.
- RNF-04: Persistencia. Todos los datos críticos (usuarios, productos, pedidos) deben persistir en una base de datos relacional, garantizando su integridad mediante transacciones.
- RNF-05: Seguridad Básica. Las contraseñas deben almacenarse de forma segura (hash). El acceso a funcionalidades de administración estará estrictamente controlado por el perfil de usuario.
- RNF-06: Compatibilidad. La aplicación debe ejecutarse en cualquier entorno con Java Runtime Environment (JRE) 8 o superior instalado.

3. Stack Tecnológico y Justificación

- Lenguaje de Programación: Java SE 11. Se eligió por su robustez, portabilidad, amplia adopción en el entorno empresarial y su idoneidad para el desarrollo de aplicaciones de escritorio complejas con acceso a bases de datos.
- Interfaz Gráfica (UI): JavaFX con FXML. Framework moderno y oficial para interfaces Java, que permite una clara separación entre el diseño (mediante

archivos `.fxml` creados con Scene Builder) y la lógica de negocio, facilitando el mantenimiento y la reutilización de componentes.

- Persistencia de Datos: Hibernate ORM 5.x. Framework de mapeo objeto-relacional (ORM) que abstrae las consultas SQL, permitiendo trabajar con objetos Java (`entities`) y mejorando la productividad, la portabilidad entre diferentes SGBD y la gestión de transacciones.
- Base de Datos: MySQL 8.0 (o MariaDB). SGBD relacional de código abierto, ampliamente documentado y con un equilibrio óptimo entre rendimiento y facilidad de uso para proyectos académicos y profesionales de escala media.
- Patrón de Arquitectura: Se sigue una variante del patrón Modelo-Vista-Controlador (MVC):
 - Modelo: Entidades de Hibernate (`Profile`, `User`, `Shoe`, etc.).
 - Vista: Archivos FXML y sus controladores asociados.
 - Controlador: Clases Java que gestionan la lógica de aplicación, la interacción con la UI y la capa de persistencia (a menudo a través de clases DAO o Repository).
- Control de Versiones: Git, con repositorio alojado en GitHub/GitLab.

4. Diseño del Sistema

4.1. Diagrama de Clases (Esquemático)

HAY QUE HACER EL DIAGRAMAAAAAAAAAAAAA!!!!!!

4.2. Diseño de la Interfaz y Experiencia de Usuario (UX/UI)

- Estructura: La aplicación se compone de ventanas (`Stage`) independientes para Login, Registro, Panel de Administración (con pestañas o menús para `Shoe` y `User`), Catálogo de Cliente y Carrito de Compra. La navegación es modal y lineal para evitar que el usuario se pierda.
- Paleta de Colores y Tipografía:
 - Colores Principales: Azul claro (`#4A90E2` o similar) sobre fondos blancos (`#FFFFFF`).
 - Justificación:
 - Azul: Color ampliamente asociado a la confianza, la seguridad y la profesionalidad, crucial para una aplicación que maneja datos

personales y transacciones. Su tono claro reduce la fatiga visual y transmite una sensación moderna y tecnológica.

- Blanco: Proporciona el máximo contraste, mejora la legibilidad del texto y ofrece un aspecto limpio y ordenado, esencial para presentar listados de productos y formularios de datos de manera clara.
 - Esta combinación cumple con directrices básicas de accesibilidad (WCAG), al asegurar un ratio de contraste suficiente para usuarios con deficiencias visuales leves.
- Componentes: Se utilizan controles estándar de JavaFX (`TableView` para listados CRUD, `TextField` con validación, `Button` con estilos consistentes, `ImageView` para fotografías de producto). Se prioriza la usabilidad mediante agrupación lógica de controles, mensajes de error contextuales y confirmaciones en operaciones destructivas (eliminar).

5. Implementación y Pruebas (Testing)

5.1. Estrategia de Pruebas

Dada la naturaleza académica del proyecto, se ha implementado un enfoque de pruebas manuales sistemáticas, estructuradas por módulos y perfiles de usuario. No se ha implementado un framework de testing automatizado (JUnit) en esta fase, pero se reconoce su necesidad en un entorno de producción.

5.2. Errores Comunes y Soluciones

1. Error: `LazyInitializationException` en Hibernate al acceder a colecciones (`items` en un `Order`) fuera del contexto de una sesión.
 - Causa: Intento de cargar datos asociados después de cerrar la sesión de Hibernate.
 - Solución: Utilizar `JOIN FETCH` en las consultas JPQL/HQL para cargar las asociaciones necesarias de forma anticipada (`eager`) dentro de la transacción. Ejemplo: `SELECT o FROM Order o JOIN FETCH o.items WHERE o.user.id = :userId`.
2. Error: Actualización de la `TableView` de JavaFX no refleja los cambios tras una operación CRUD en la base de datos.
 - Causa: La lista (`ObservableList`) que alimenta la tabla no se actualiza tras modificar la fuente de datos subyacente.

- Solución: Limpiar y volver a cargar la lista desde la base de datos después de cada operación de inserción, modificación o eliminación. Emitir eventos de actualización o utilizar `PropertyValueFactory` correctamente enlazado a las propiedades de las entidades.
- 3. Error: Problemas de concurrencia al editar y guardar una entidad desatendida (`detached entity`).
 - Causa: Se recupera una entidad en una sesión, se presenta en la UI, se modifica y se intenta guardar en una nueva sesión sin reasociarla (`merge`).
 - Solución: Utilizar el patrón DAO (Data Access Object) con un método `update()` que maneje la operación `merge()` de Hibernate dentro de una transacción nueva. Alternativamente, trabajar con DTOs (Data Transfer Objects) para la capa de presentación.
- 4. Error: Validación de formularios inconsistente (ej., email, números de tarjeta).
 - Causa: Validación solo en el lado cliente (UI) o falta de validación en la capa de negocio/persistencia.
 - Solución: Implementar validación en cascada:
 - Cliente: Uso de `TextField` con `TextFormatter` o listeners para formato básico.
 - Servidor: Anotaciones Bean Validation (`@NotNull`, `@Email`, `@Size`) en las entidades y validación en el servicio/controlador antes de persistir.

6. Conclusión y Trabajo Futuro

Se ha desarrollado exitosamente una aplicación de escritorio funcional que cumple con todos los requisitos funcionales y no funcionales establecidos. eSneakerPro demuestra un uso competente de tecnologías estándar del ecosistema Java (JavaFX, Hibernate, MySQL) y la aplicación de patrones de diseño y principios de arquitectura de software.

Como líneas de mejora y trabajo futuro se proponen:

1. Implementación de un sistema de logging (Log4j2) para facilitar la depuración en producción.
2. Desarrollo de una suite de pruebas automatizadas unitarias (JUnit) e integrales (TestFX para la UI).
3. Refactorización hacia una arquitectura más desacoplada, introduciendo una capa de servicios y el uso de inyección de dependencias (Spring Framework o CDI).

4. Migración a un modelo cliente-servidor con API REST (Spring Boot) y un frontend web (React/Angular) para ampliar la accesibilidad.