

PROYECTO TIENDA DE COLECCIONES Y PRENDAS

Desarrollo de Servicios Web 1 --- V.01.2025-III

ALUMNO :
Santamaria Olivos
Wilder Joseph

DOCENTE :
Pariona Yauricasa
Erick

CARRERA:
Desarrollo de
Sistemas de
Información

CICLO : IV

Contenido

1-Introducción del proyecto	4
1.1 Resumen del sistema desarrollado	4
1.2 Propósito del proyecto	4
1.3 Alcance del proyecto	4
1.4 Tecnologías usadas	5
1.5 Objetivos del proyecto	5
-Objetivo general.....	5
-Objetivos específicos.....	5
2-Modelo de Datos.....	6
2.1 Diagrama ER o Modelo Relacional (MySQL Workbench)	6
2.2 Definición de cada tabla	6
-Tabla Colecciones	6
-Tabla Prendas	7
3. Especificación Técnica del Desarrollo	7
3.2-Descripción de entidades JPA.....	9
ColeccionesEntity.java	9
PrendasEntity.java	11
3.3 Servicios Implementados	13
-Metodos CRUD.....	13
ColeccionesService:	13
• Optional<ColeccionesEntity> buscarPorId(Long id) : obtener una colección por su id. .. 13	13
• List<ColeccionesEntity> listado() : listar todas las colecciones. .. 13	13
• List<ColeccionesEntity> listadoActivos() : listar solo colecciones con estado = true. 13	13
• ColeccionesEntity guardar(ColeccionesEntity entidad) : crear nueva colección (valida nombre y año).	13
• ColeccionesEntity actualizar(ColeccionesEntity entidad) : actualizar colección existente (verifica existencia).	13
• void inactivar(Long id) : inactivar una colección (soft-delete).	13
PrendasService:	13
• List<PrendasEntity> listarTodas() : listar todas las prendas.	13
• Optional<PrendasEntity> obtenerPorId(Long id) : obtener prenda por id.	13
• List<PrendasEntity> listarPorColeccion(Long colecciónId) : listar prendas disponibles de una colección.	13
• List<PrendasEntity> buscarPorNombreOTalla(String termino) : búsqueda por nombre o talla (contiene, case-insensitive).	13

• List<PrendasEntity> filtrarPorEstado(String estado) : filtrar por valor de estado.	13
• PrendasEntity guardar(PrendasEntity prenda) : crear prenda (valida precio y asociación a colección).	13
• PrendasEntity actualizar(PrendasEntity prenda) : actualizar campos de una prenda existente.	13
• void inactivar(Long id) : cambiar estado a "agotada" (soft-delete).	13
-Lógica Usada	13
-Eliminación Lógica	14
3.4-Controladores	15
-Metodos	15
-Rutas	16
-Validaciones	16
4-Diseño de Prototipos para Interfaces (Thymelaf)	17
Inicio :	17
Listar prendas:	17
Registro de prenda :	18
Editar una prenda :	18
Nueva Colección :	19
Editar Colección :	19
Ver Detalle de la Colección :	20
Error 400:.....	20
Error 404:.....	20
Error 500 :	21
5-Manual de Usuario.....	21
5.1 ¿Cómo se ejecuta el proyecto?	21
5.2-Manual paso a paso	22
Cómo registrar.....	22
Cómo editar.....	23
Cómo eliminar (eliminación lógica).....	23
Cómo ver los hijos por padre (ver prendas de una colección)	24
6. Guía de Instalación o Configuración.....	24
6.1. Dependencias de Maven:.....	24
Archivo pom.xml (Dependencies):	25
Conexion H2 o MySQL:.....	25
H2 (Desarrollo):	25
Localmente (MySQL):	26

Importante:	27
SCRIPT DE LA BASE DE DATOS EN MYSQL:.....	27
7-Pruebas del Sistema	28
7.1-Pruebas Funcionales	28
8-README Técnico en GitHub:	29
9-Reflexión:	29

1-Introducción del proyecto

1.1 Resumen del sistema desarrollado

TiendaColecciones es una aplicación web desarrollada para la gestión integral de una tienda de ropa, permitiendo administrar colecciones de prendas y las prendas individuales asociadas a cada colección. El sistema incluye funcionalidades CRUD (Crear, Leer, Actualizar, Eliminar) para colecciones y prendas, con validaciones de negocio, manejo de excepciones personalizadas, páginas de error personalizadas (400, 404, 500) y búsqueda/filtrado en tiempo real. La aplicación está desplegada en producción utilizando Render.com con una base de datos PostgreSQL, y soporta desarrollo local con MySQL. La interfaz utiliza Thymeleaf con Bootstrap para una experiencia de usuario responsiva y moderna.

1.2 Propósito del proyecto

El propósito principal del sistema es proporcionar una herramienta eficiente y fácil de usar para propietarios de tiendas de ropa, permitiendo gestionar inventarios de colecciones y prendas de manera organizada. Facilita la administración de datos, mejora la experiencia del usuario con validaciones y retroalimentación visual, y asegura la integridad de los datos mediante excepciones y manejo de errores.

1.3 Alcance del proyecto

1. Gestión completa de colecciones: registro, listado, edición, eliminación y visualización de detalles.
2. Gestión de prendas: registro con asociación a colecciones, listado con filtros de búsqueda, edición, eliminación y cambio de estado (activo/inactivo).
3. Validaciones de negocio: reglas como precios positivos, nombres únicos, y manejo de excepciones para errores de validación.
4. Interfaz de usuario: páginas responsivas con Bootstrap y páginas de error personalizadas.
5. Persistencia de datos: soporte para MySQL en desarrollo y PostgreSQL en producción, con scripts SQL para inicialización.
6. Despliegue: aplicación con el uso de Docker, desplegada en Render.com con base de datos en la nube.

7. Funcionalidades adicionales: búsqueda en tiempo real, navegación entre colecciones y prendas, y manejo de errores globales.

1.4 Tecnologías usadas

- Backend: Java 17, Spring Boot 3.x (con Spring MVC, Spring Data JPA, Spring Validation), Maven para gestión de dependencias y build.
- Frontend: Thymeleaf como motor de plantillas, Bootstrap 5.3 para estilos responsivos, Bootstrap Icons para iconografía, y JavaScript nativo para funcionalidades dinámicas (ej. búsqueda en tiempo real).
- Base de datos: MySQL 8.0 para desarrollo local, PostgreSQL para producción (en Render) y con Hibernate como ORM.
- Herramientas adicionales: Lombok en Java, Jakarta Validation para validaciones, Docker para la ejecución de la aplicación sin complicaciones, y Git para control de versiones.
- Despliegue: Render.com como plataforma de hosting, con Docker para optimización de compilación sobre Maven.

1.5 Objetivos del proyecto

-Objetivo general

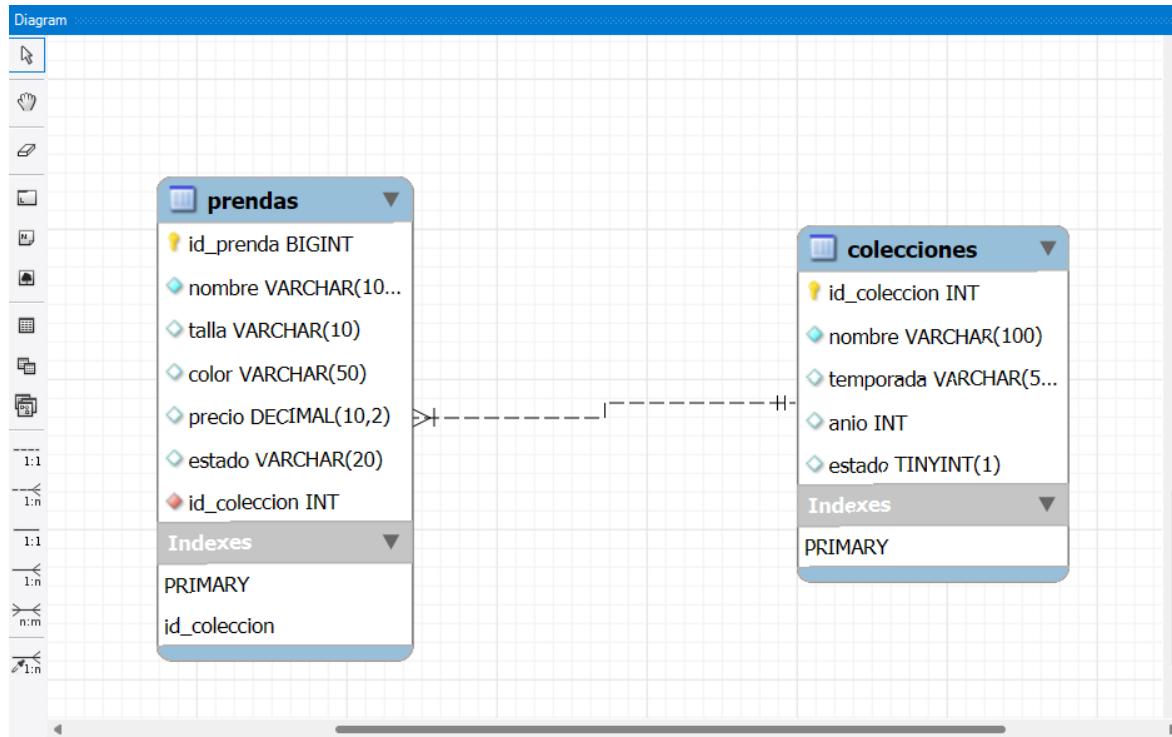
Desarrollar y desplegar una aplicación web completa para la gestión de inventarios en una tienda de ropa, permitiendo a los usuarios administrar colecciones y prendas de manera eficiente, con una interfaz intuitiva, validaciones robustas y despliegue en la nube, con el fin de demostrar competencias en desarrollo full-stack y DevOps.

-Objetivos específicos

- I. Implementar funcionalidades CRUD completas para colecciones y prendas, incluyendo validaciones de negocio, manejo de excepciones personalizadas para asegurar la integridad de los datos y una experiencia de usuario fluida
- II. Diseñar y personalizar páginas de error (400, 404, 500) con un estilo consistente al layout principal, utilizando Thymeleaf y Bootstrap, para mejorar la usabilidad y profesionalismo de la aplicación.
- III. Containerizar la aplicación con Docker y desplegarla en Render.com con una base de datos PostgreSQL en producción, garantizando escalabilidad, portabilidad y disponibilidad continua, mientras se mantiene compatibilidad con desarrollo local usando MySQL.

2-Modelo de Datos

2.1 Diagrama ER o Modelo Relacional (MySQL Workbench)



2.2 Definición de cada tabla

-Tabla Colecciones

Campos	Tipo de dato	Restricciones (NOT NULL, UNIQUE, FK)	Estado (A/I)
id_coleccion	INT	PRIMARY KEY, AUTO_INCREMENT, NOT NULL	-
nombre	VARCHAR(100)	NOT NULL, UNIQUE	-
temporada	VARCHAR(50)	NULLABLE	-
anio	INT	NOT NULL	-
estado	TINYINT(1)	NOT NULL, DEFAULT 1	1 = A (Activo), 0 = I (Inactivo)

-Tabla Prendas

Campos	Tipo de dato	Restricciones (NOT NULL, UNIQUE, FK)	Estado (A/I)
id_prenda	BIGINT	PRIMARY KEY, AUTO_INCREMENT, NOT NULL	-
nombre	VARCHAR(100)	NOT NULL	-
talla	VARCHAR(50)	NULLABLE	-
color	VARCHAR(50)	NOT NULL	-
precio	DECIMAL(10,2)	NOT NULL , CHECK (precio >= 0)	1 = A (Activo), 0 = I (Inactivo)
estado	VARCHAR(20)	NOT NULL, DEFAULT 'disponible'	'disponible' → A (Activo)
id_coleccion	INT	NOT NULL , FOREING KEY	-

3. Especificación Técnica del Desarrollo

Paquete base: com.santamaria.tienda.Continua3

Controller:

com.santamaria.tienda.Continua3.controller

- InicioController.java
- ColeccionController.java
- PrendasController.java

Service :

com.santamaria.tienda.Continua3.service

- ColeccionesService.java
- PrendasService.java

Service.impl (implementaciones):

com.santamaria.tienda.Continua3.service.impl

- ColeccionesServiceImpl.java
- PrendasServiceImpl.java

Repository:

com.santamaria.tienda.Continua3.repository

- ColeccionesRepository.java
- PrendasRepository.java

Entity:

com.santamaria.tienda.Continua3.entity

- ColeccionesEntity.java
- PrendasEntity.java

DTO:

com.santamaria.tienda.Continua3.dto

- ColeccionesDto.java
- PrendasDto.java

Mapper :

com.santamaria.tienda.Continua3.mapper

- ColeccionesMapper.java — ColeccionesMapper.java
- PrendasMapper.java — PrendasMapper.java

Expcion:

com.santamaria.tienda.Continua3.exception

- ColeccionNoEncontradaException.java
- DatosInvalidosException.java
- PrecioInvalidoException.java
- PrendaSinColecciónException.java

Config:

com.santamaria.tienda.Continua3.config

- ConfigException.java — ConfigException.java

Util:

com.santamaria.tienda.Continua3.util

- ConstantesApp.java — ConstantesApp.java

Plantillas (Thymeleaf):

templates

- layout.html — layout.html
- index.html — index.html

colecciones/

listado.html, registro.html, lista.html, form.html, detalle.html

prendas/

registro.html, lista.html, form.html

error/

400.html, 404.html, 500.html

Clase principal:

- Continua3Application.java

3.2-Descripción de entidades JPA

ColeccionesEntity.java

```
@Entity
@Table(name = "colecciones")
@Getters
@Setters
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ColeccionesEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_coleccion")
    private Long idColeccion;

    @Column(nullable = false, length = 100)
    private String nombre;

    @Column(length = 50)
    private String temporada;

    @Column(nullable = false)
    private Integer anio;

    @Column(nullable = false)
    @Builder.Default
    private Boolean estado = true;

    @OneToMany(mappedBy = "colección")
    private List<PrendasEntity> prendas;
}
```

Anotaciones de clase:

- `@Entity` : marca la clase como entidad JPA.
- `@Table(name = "colecciones")` : mapea la entidad a la tabla colecciones.
- Lombok: `@Getter, @Setter, @AllArgsConstructorConstructor, @NoArgsConstructorConstructor, @Builder`.

Campos (campo — tipo — mapping/columna — anotaciones actuales — descripción / restricciones / valor por defecto):

- ❖ `idColeccion` — `Long` — `@Id, @GeneratedValue(strategy = GenerationType.IDENTITY), @Column(name = "id_coleccion")`
- ❖ PK, generado por la BD (`AUTO_INCREMENT`). Mapea a `BIGINT` en la BD.
- ❖ `nombre` — `String` — `@Column(nullable = false, length = 100)`
- ❖ Nombre de la colección. Obligatorio (NOT NULL), longitud máxima 100.
- ❖ `temporada` — `String` — `@Column(length = 50)`
- ❖ Campo opcional para temporada (p. ej. "Verano").
- ❖ `anio` — `Integer` — `@Column(nullable = false)`
- ❖ Año de la colección. Obligatorio.
- ❖ `estado` — `Boolean` — `@Column(nullable = false), @Builder.Default private Boolean estado = true`
- ❖ Flag Activo/Inactivo. Se guarda como valor booleano en la entidad; en BD se espera `TINYINT(1)` (1 = true, 0 = false). Valor por defecto true.
- ❖ `prendas` — `List<PrendasEntity>` — `@OneToMany(mappedBy = "colección")`
- ❖ Relación 1:N con `PrendasEntity`. `mappedBy = "colección"` indica que el propietario de la FK es `PrendasEntity`.

Relaciones:

- `@OneToMany(mappedBy = "colección")` (JPA implementaciones). Es el lado no propietario; la FK está en `prendas.id_colección`

PrendasEntity.java

```
@Entity
@Table(name = "prendas")
@Builder
public class PrendasEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_prenda")
    private Long idPrenda;

    @Column(nullable = false, length = 100)
    private String nombre;

    @Column(length = 10)
    private String tallas;

    @Column(length = 50)
    private String color;

    @Column(nullable = false, precision = 10, scale = 2)
    private BigDecimal precio;

    @Column(nullable = false, length = 20)
    @Builder.Default
    private String estado = "disponible";

    @ManyToOne(optional = false)
    @JoinColumn(name = "id_coleccion", nullable = false)
    private ColeccionesEntity colección;
}
```

Anotaciones de clase:

- `@Entity`
- `@Table(name = "prendas")`
- Lombok: `@Getter, @Setter, @AllArgsConstructorConstructor, @NoArgsConstructorConstructor, @Builder.`

Campos (campo — tipo Java — mapping/columna — anotaciones actuales — descripción / restricciones / valor por defecto):

- ❖ `idPrenda` — `Long` — `@Id, @GeneratedValue(strategy = GenerationType.IDENTITY), @Column(name = "id_prenda")`
PK (BIGINT), generado por BD.
- ❖ `nombre` — `String` — `@Column(nullable = false, length = 100)`
Nombre de la prenda. Obligatorio.
- ❖ `talla` — `String` — `@Column(length = 10)`
Talla (opcional).
- ❖ `color` — `String` — `@Column(length = 50)`
Color (opcional).
- ❖ `precio` — `BigDecimal` — `@Column(nullable = false, precision = 10, scale = 2)`
Precio de la prenda. Obligatorio. Mapea a DECIMAL(10,2).
- ❖ `estado` — `String` — `@Column(nullable = false, length = 20), @Builder.Default private String estado = "disponible"`
Estado textual (ej. "disponible", "agotado"). Obligatorio. Default "disponible".
- ❖ `colección` — `ColeccionesEntity` — `@ManyToOne(optional = false), @JoinColumn(name = "id_colección", nullable = false)`
Relación N:1 con ColeccionesEntity. Esta entidad es la propietaria de la FK `id_colección`.

Relaciones y fetch:

- `@ManyToOne(optional = false)` por defecto es EAGER en JPA; si prefieres evitar cargas innecesarias, usar `@ManyToOne(fetch = FetchType.LAZY, optional = false)`

3.3 Servicios Implementados

-Metodos CRUD

ColeccionesService:

- **Optional<ColeccionesEntity> buscarPorId(Long id)** : obtener una colección por su id.
- **List<ColeccionesEntity> listado()** : listar todas las colecciones.
- **List<ColeccionesEntity> listadoActivos()** : listar solo colecciones con estado = true.
- **ColeccionesEntity guardar(ColeccionesEntity entidad)** : crear nueva colección (valida nombre y año).
- **ColeccionesEntity actualizar(ColeccionesEntity entidad)** : actualizar colección existente (verifica existencia).
- **void inactivar(Long id)** : inactivar una colección (soft-delete).

PrendasService:

- **List<PrendasEntity> listarTodas()** : listar todas las prendas.
- **Optional<PrendasEntity> obtenerPorId(Long id)** : obtener prenda por id.
- **List<PrendasEntity> listarPorColeccion(Long colecciónId)** : listar prendas disponibles de una colección.
- **List<PrendasEntity> buscarPorNombreOTalla(String termino)** : búsqueda por nombre o talla (contiene, case-insensitive).
- **List<PrendasEntity> filtrarPorEstado(String estado)** : filtrar por valor de estado.
- **PrendasEntity guardar(PrendasEntity prenda)** : crear prenda (valida precio y asociación a colección).
- **PrendasEntity actualizar(PrendasEntity prenda)** : actualizar campos de una prenda existente.
- **void inactivar(Long id)** : cambiar estado a "agotada" (soft-delete).

-Lógica Usada

Validaciones comunes y reglas aplicadas antes de persistir:

- Comprobación de nulidad de la entidad (rechaza peticiones vacías).
- ColeccionesService.guardar: valida nombre no vacío; valida año en rango razonable (ej. 1900–2100). Si estado es nulo para nuevos registros se inicializa a true.
- PrendasService.guardar: valida precio > 0 (lanza PrecioInvalidoException si falla); valida que exista y tenga id la colección asociada (lanza PrendaSinColeccionException); si estado es nulo se asigna ConstantesApp.ESTADO_DISPONIBLE.

Consultas/repo usadas en lógica:

- Repositorios usan métodos derivados de Spring Data JPA (findById, findAll, findByEstado, findByNombreContainingIgnoreCaseOrTallaContainingIgnoreCase, findByColeccion_IdColeccionAndEstado, etc.) para filtrar y obtener sólo registros relevantes.

Patrón de actualización:

- actualizar(...) carga la entidad existente, actualiza campos permitidos y persiste de nuevo (evita sobrescribir campos no controlados).

Manejo de errores:

- Reglas de validación específicas lanzan excepciones de dominio (DatosInvalidosException, PrecioInvalidoException, PrendaSinColeccionException) para ser capturadas y traducidas a respuestas/ toasts en la capa web.

-Eliminación Lógica

Cómo se implementa:

- No se eliminan filas físicamente; se cambia un campo de estado para marcar el registro como inactivo.
- Colecciones: campo estado (Boolean) → true = Activa, false = Inactiva. Método inactivar(id) pone estado = false.
- Prendas: campo estado (String) → valores en ConstantesApp ("disponible", "agotada"). Método inactivar(id) setea estado = ConstantesApp.ESTADO_AGOTADA.

3.4-Controladores

La aplicación tiene controladores MVC que sirven las vistas Thymeleaf y manejan las operaciones del usuario. Los controladores principales son: InicioController, ColeccionController y PrendasController.

-Metodos

InicioController:

- `Home(Model modelo)` — carga la página principal (index) con metadatos de la app.

ColeccionController:

- `listarColecciones(Model modelo)` — lista colecciones.
- `formularioNuevaColeccion(Model modelo)` — prepara formulario con ColeccionesDto vacío.
- `editarColeccion(Long id, Model modelo, RedirectAttributes mensajes)` — carga colección como DTO para editar.
- `verDetalleColeccion(Long id, Model modelo, RedirectAttributes mensajes)` — muestra detalle de colección.
- `guardarColeccion(@Valid @ModelAttribute("colección") ColeccionesDto colecciónDto, BindingResult erroresValidacion, Model modelo)` — procesa creación/edición usando DTO y mapper.
- `inactivarColeccion(Long id)` — inactiva colección.
- Manejo de errores HTTP: `manejarErrorArgumento()` (400), `manejarErrorServidor()` (500).

PrendasController:

- `listarPrendas(Model modelo, Long colecciónId, String estado, String búsqueda)` — lista con filtros opcionales.
- `registroPrenda(Model modelo, Long colecciónId)` — prepara formulario con PrendasDto vacío.
- `editarPrenda(Long id, Model modelo, RedirectAttributes redirectAttributes)` — convierte entidad a DTO para edición.
- `guardar(@Valid @ModelAttribute("prenda") PrendasDto prendaDto, BindingResult erroresValidacion, Model modelo, RedirectAttributes mensajesRedirección)` — procesa usando DTO, mapper y validaciones robustas.

- `cambiarEstado(Long id, String estado, RedirectAttributes mensajes)` — cambia estado de prenda.
- Manejo de errores HTTP: `manejarErrorArgumento()` (400), `manejarErrorServidor()` (500).

-Rutas

- GET / → InicioController.home (vista index).
- GET /web/coleccion y GET /web/coleccion/listado → listado de colecciones.
- GET /web/coleccion/registroColeccion → formulario nueva colección.
- GET /web/coleccion/editar/{id} → editar colección.
- GET /web/coleccion/detalle/{id} → detalle colección.
- POST /web/coleccion/guardar → guardar colección.
- POST /web/coleccion/inactivar/{id} → inactivar colección.
- GET /web/prendas y GET /web/prendas/listar_prendas → listado de prendas (con ?colecciónId=..&estado=..&busqueda=..).
- GET /web/prendas/registroPrenda → formulario nueva prenda (opcional ?colecciónId=..).
- GET /web/prendas/editar/{id} → editar prenda.
- POST /web/prendas/guardar → guardar prenda.
- POST /web/prendas/cambiar-estado/{id} → cambiar estado de prenda.

-Validaciones

En DTOs (@Valid en controladores):

- PrendasDto: @NotBlank nombre, @NotNull @DecimalMin precio, @NotNull idColección
- ColeccionesDto: @NotBlank nombre, @NotNull @Min @Max año, estado boolean

Manejo de Errores:

- Errores de validación → retorna al formulario con mensajes
- Errores de negocio → RedirectAttributes con mensajes de error
- Errores HTTP → páginas de error personalizadas (400, 404, 500)

Mappers implementados:

- PrendasMapper: toEntity(), toDto(), updateEntity()
- ColeccionesMapper: toEntity(), toDto(), updateEntity()

4-Diseño de Prototipos para Interfaces (Thymelaf)

Inicio :

The dashboard page features a sidebar on the left with a dark blue background and white icons for 'Gestión de Prendas', 'Inicio', 'Prendas', and 'Colecciones'. The main area has a light gray background with a central header 'Dashboard'. In the center is a large blue shopping bag icon with a white checkmark inside. Below it is the text 'Sistema completo para gestionar tu inventario de ropa y colecciones'. At the bottom are two buttons: a red one labeled 'Ver Prendas' and a blue one labeled 'Nueva Prenda'. A small info box in the bottom right corner displays 'Información del Sistema' with details: Wilder Santamaría Olivos, Ciclo V, IDAT.

Listar prendas:

The 'Lista de Prendas' page follows the same dark sidebar design. The main content area is titled 'Lista de Prendas'. It includes a red 'Nueva Prenda' button at the top right. Below are three filter dropdowns: 'Filtrar por colección' (set to 'Todas las colecciones'), 'Estado' (set to 'Todos los estados'), and 'Buscar en tiempo real' (with an input field 'Nombre o talla...'). A blue 'Filtrar' button is located to the right of the filters. A table below shows a single item: #1, Camisa con rayas negras, Colección de Verano, S, Blanco Humo, \$/ 90.00, Disponible. To the right of the table are edit and delete icons.

Registro de prenda :

Gestión de Prendas

← Volver a Lista

Datos de la Prenda

Nombre *

Ingrese el nombre de la prenda

Colección *

Seleccione una colección

Talla

Seleccione talla

Color

Ej: Azul, Rojo, Negro

Precio *

S/ 0.00

Ingrese el precio en soles (mínimo S/ 0.01)

Estado

Disponible

Estado Inicial de la prenda en el inventario

Guardar Cancelar

Editar una prenda :

Gestión de Prendas

← Volver a Lista

Datos de la Prenda

Nombre *

Camisa con rayas negras

Colección *

Colección de Verano (2025)

Talla

S

Color

Blanco Humo

Precio *

S/ 90.00

Ingrese el precio en soles (mínimo S/ 0.01)

Estado

Disponible

Estado inicial de la prenda en el inventario

Actualizar Cancelar Ver Colección

Nueva Colección :

Complete el formulario con los datos de la colección

← Volver

Datos de la Colección

Nombre *
Ej: Colección Verano 2024
Ingrese un nombre descriptivo y único para la colección

Temporada
Seleccione una temporada
Temporada a la que pertenece esta colección

Año *
2024
Año de lanzamiento de la colección (1900 - 2100)

Colección activa
Las colecciones activas están disponibles para asociar prendas

Cancelar Guardar

Editar Colección :

Complete el formulario con los datos de la colección

←

Datos de la Colección

Nombre *
Colección de Verano
Ingrese un nombre descriptivo y único para la colección

Temporada
Primavera
Temporada a la que pertenece esta colección

Año *
2025
Año de lanzamiento de la colección (1900 - 2100)

Colección activa
Las colecciones activas están disponibles para asociar prendas

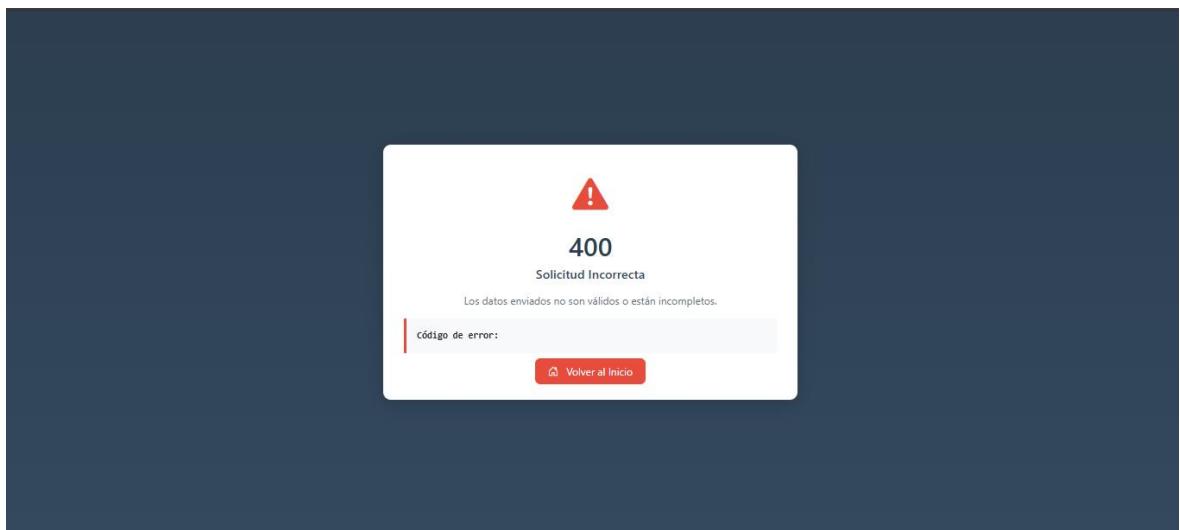
Cancelar Actualizar

Ver Detalle de la Colección :

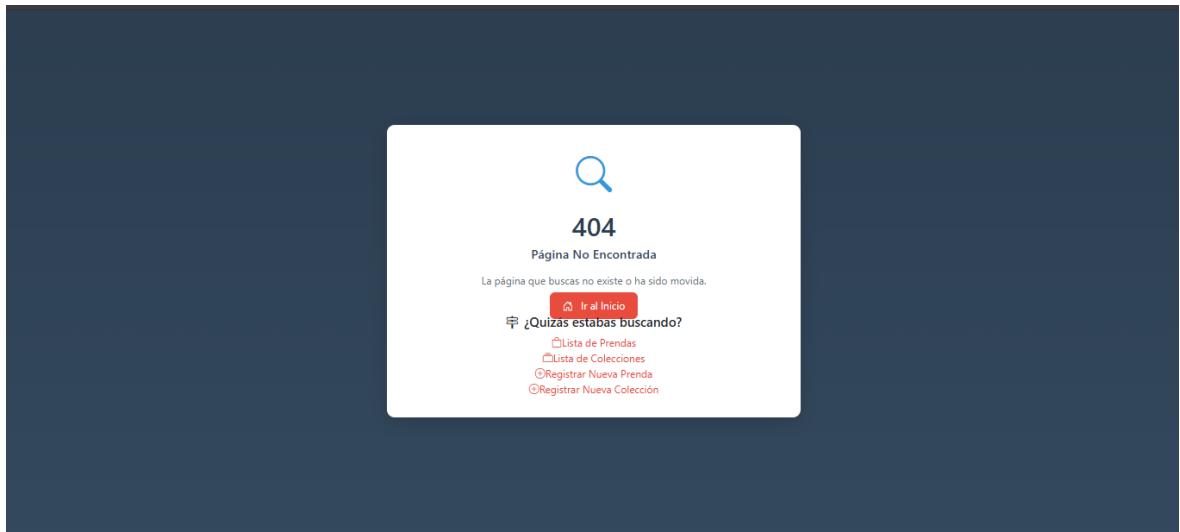
The screenshot shows a web application interface for managing collections. On the left is a sidebar with icons for Home, Items, and Collections. The main area has a title 'Detalle de Colección'. It contains two main sections: 'Información de la Colección' and 'Estadísticas'. The 'Información de la Colección' section displays details: ID (1), Nombre (Colección de Verano), Temporada (Primavera), Año (2025), and Estado (Activa). The 'Estadísticas' section shows 1 Total Prendas and 1 Disponibles. Below these are sections for 'Prendas de esta Colección' and a table listing items like 'Camisa con rayas negras'.

ID	NOMBRE	TALLA	COLOR	PRECIO	ESTADO	ACCIONES
1	Camisa con rayas negras	S	Blanco Humo	S/ 90.00	Disponible	<button>Editar</button> <button>Agotar</button>

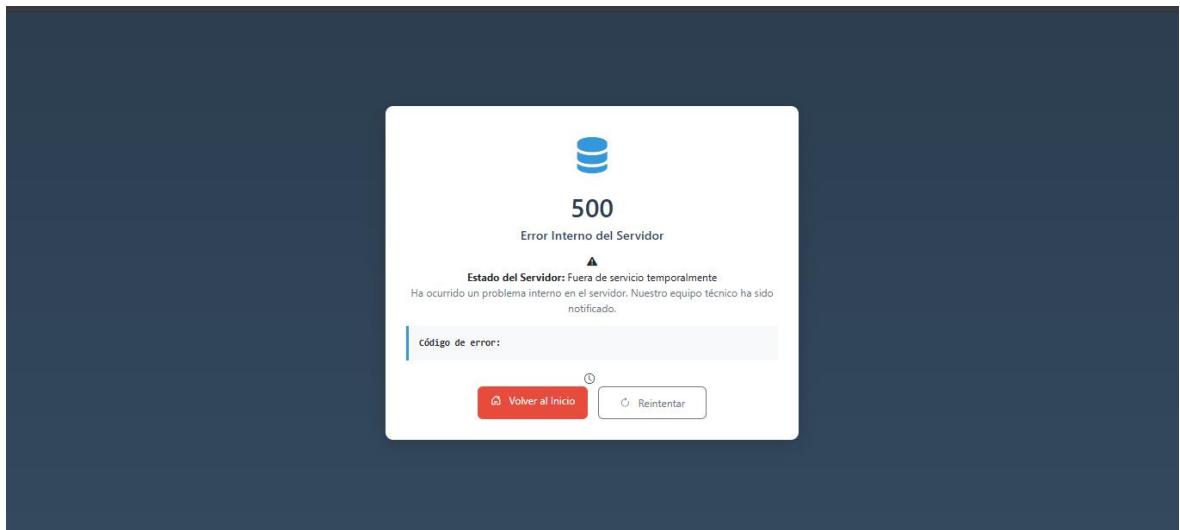
Error 400:



Error 404:



Error 500 :



5-Manual de Usuario

5.1 ¿Cómo se ejecuta el proyecto?

- **Versión de Java:** Java 17 (JDK 17) — usar una distribución compatible (Adoptium, Oracle, Amazon Corretto, etc.).
- **Comando de ejecución (desarrollo, PowerShell):**
 - Ejecutar directamente con Maven: mvn spring-boot:run
- **Compilar y ejecutar el JAR (recomendado para producción/local run):**

- mvn clean package -DskipTests
- java -jar target/*.jar

Ruta inicial del sistema (URLs de entrada):

- Página raíz: [http://localhost:8080/](http://localhost:8080)
- Páginas principales del proyecto:
 - Listado de prendas: <http://localhost:8080/web/prendas/listar>
 - Formulario [registro/edición](http://localhost:8080/web/prendas/registroPrenda)
prenda: <http://localhost:8080/web/prendas/registroPrenda> (o /web/prendas/editar/{id})
 - Listado/gestión de colecciones: <http://localhost:8080/web/colecciones/listar> (o páginas relacionadas en /web/colecciones/*)

Importante:

- Asegúrate de que la base de datos esté en ejecución y que las propiedades en src/main/resources/application.properties (URL, usuario, contraseña) estén correctas antes de ejecutar.
- Si ejecutas desde un IDE (IntelliJ/Eclipse/VS Code), configura el SDK/JDK a Java 17 y ejecuta la clase principal com.idat.tienda.Continua3.Continua3Application.
- Si el JAR no aparece en target/ después de mvn package, revisa la salida de Maven por errores de compilación.

5.2-Manual paso a paso

Cómo registrar

Un elemento padre (Colección):

- ❖ Ir a la lista de colecciones: <http://localhost:8080/web/colecciones/listar>
- ❖ Hacer clic en el botón o enlace "Nueva Colección" (o similar).

- ❖ Completar el formulario con los campos obligatorios (por ejemplo: nombre, descripción).
- ❖ Pulsar Guardar. Verás la nueva colección en la lista si la operación fue exitosa.
- ❖ Un elemento hijo (Prenda) asociado a una Colección:

- ❖ Ir a la lista de prendas: <http://localhost:8080/web/prendas/listar>
- ❖ Hacer clic en "Registrar prenda" o "Nueva prenda".
- ❖ En el formulario de prenda, seleccionar la Colección padre desde el control correspondiente (campo Colección o idColeccion).
- ❖ Rellenar los demás campos (nombre, talla, precio, etc.).
- ❖ Pulsar Guardar. La prenda quedará asociada a la colección seleccionada.

Cómo editar

Editar una Colección:

- ❖ En <http://localhost:8080/web/colecciones/listar>, localizar la colección a editar.
- ❖ Hacer clic en el enlace o botón Editar de esa fila (normalmente apunta a </web/colecciones/editar/{id}>).
- ❖ Modificar los campos necesarios en el formulario.
- ❖ Pulsar Guardar para aplicar los cambios.

Editar una Prenda:

- ❖ En <http://localhost:8080/web/prendas/listar>, localizar la prenda y hacer clic en Editar (ruta típica: </web/prendas/editar/{id}>).
- ❖ Cambiar los campos (si quieres cambiar la colección, seleccionar otra en el campo correspondiente).
- ❖ Pulsar Guardar.

Cómo eliminar (eliminación lógica)

El sistema usa eliminación lógica (no borra la fila de la BD; cambia su estado a inactivo).

- ❖ En la lista correspondiente ([/web/colecciones/listar](#) o [/web/prendas/listar](#)) localiza la fila del elemento.
- ❖ Pulsar el botón Inactivar / Eliminar (según la interfaz); este botón normalmente llama a una acción que cambia estado = false o estado = "INACTIVO" para ese id.
- ❖ Confirmar la acción si aparece un diálogo.
- ❖ El elemento ya no aparecerá en listados activos; se conserva en la BD para auditoría/recuperación.

Cómo ver los hijos por padre (ver prendas de una colección)

Desde la lista de colecciones:

- ❖ Ir a <http://localhost:8080/web/colecciones/listar>.
- ❖ Localizar la colección deseada y hacer clic en el enlace Ver prendas o Listado prendas (si la plantilla lo incluye). Esto suele llevar a la lista de prendas filtrada por esa colección.

Desde la lista de prendas:

- ❖ Ir a <http://localhost:8080/web/prendas/listar>.
- ❖ Si la UI ofrece filtros, seleccionar la Colección para ver sólo las prendas de esa colección.

Acceso directo (si la aplicación soporta parámetros):

- ❖ Usar una URL con parámetro de colección, por ejemplo:
<http://localhost:8080/web/prendas/listar?colecciónId=1>

6. Guía de Instalación o Configuración

6.1. Dependencias de Maven:

- **Core web:** spring-boot-starter-web
- **Plantillas Thymeleaf:** spring-boot-starter-thymeleaf
- **JPA / Data:** spring-boot-starter-data-jpa
- **Validación (si aún no está):** spring-boot-starter-validation
- **H2 (para desarrollo/in-memory):** com.h2database:h2
- **MySQL driver (para producción/local con MySQL):** mysql:mysql-connector-java (o com.mysql:mysql-connector-j según repositorio)

- **Lombok (opcional, si el proyecto lo usa):** org.projectlombok:lombok
- **Spring Boot DevTools (opcional, desarrollo):** org.springframework.boot:spring-boot-devtools

Archivo pom.xml (Dependencies):

- org.springframework.boot:spring-boot-starter-data-jpa (versión gestionada por el parent Spring Boot)
- org.springframework.boot:spring-boot-starter-thymeleaf (versión gestionada por el parent Spring Boot)
- org.springframework.boot:spring-boot-starter-validation (versión gestionada por el parent Spring Boot)
- org.springframework.boot:spring-boot-starter-web (versión gestionada por el parent Spring Boot)
- org.springframework.boot:spring-boot-devtools (versión gestionada por el parent Spring Boot) — scope: runtime, optional: true
- com.mysql:mysql-connector-j (versión no especificada en el POM — runtime)
- org.postgresql:postgresql (versión no especificada en el POM — runtime) — (para despliegue en Render)
- org.projectlombok:lombok:1.18.42
- org.springframework.boot:spring-boot-starter-test (versión gestionada por el parent Spring Boot) — scope: test

Conexion H2 o MySQL:

- Archivo por defecto: application.properties
- Recomendación: usar perfiles (application-dev.properties, application-prod.properties) para separar H2 (dev) y MySQL (prod)

H2 (Desarrollo):

```
# Puerto de la app (opcional)

server.port=8080

# DataSource H2

spring.datasource.url=jdbc:h2:mem:tienda;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.driver-class-name=org.h2.Driver
```

```
spring.datasource.username=sa  
spring.datasource.password=  
  
# JPA / Hibernate  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format_sql=true  
  
# H2 console (útil en desarrollo)  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console
```

Localmente (MySQL):

```
# Puerto de la app  
server.port=8080  
  
# DataSource MySQL  
spring.datasource.url=jdbc:mysql://localhost:3306/tienda_db?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC  
spring.datasource.username=root  
spring.datasource.password=tu_password  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
  
# JPA / Hibernate  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=false
```

```
spring.jpa.properties.hibernate.format_sql=false  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

```
# Pool (opcional config para HikariCP)  
spring.datasource.hikari.maximum-pool-size=10
```

Importante:

- Ajusta `spring.jpa.hibernate.ddl-auto` con cuidado:
 - `update` es conveniente en desarrollo.
 - En producción, prefiera `validate` o manejar migraciones con Flyway/Liquibase.
- Asegúrate de que el conector MySQL esté en `pom.xml` y que MySQL esté accesible (host, puerto, credenciales).
- Para H2 recuerda usar `http://localhost:8080/h2-console` y el JDBC URL debe coincidir con `spring.datasource.url`.

SCRIPT DE LA BASE DE DATOS EN MYSQL:

```
CREATE DATABASE IF NOT EXISTS tienda_ropa;  
USE tienda_ropa;  
  
-- Tabla: colecciones  
CREATE TABLE IF NOT EXISTS colecciones (  
    id_coleccion INT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL,  
    temporada VARCHAR(50),  
    anio INT NOT NULL,
```

```

        estado TINYINT(1) NOT NULL DEFAULT 1
    );

-- Tabla: prendas

CREATE TABLE IF NOT EXISTS prendas (
    id_prenda BIGINT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    talla VARCHAR(10),
    color VARCHAR(50),
    precio DECIMAL(10,2) NOT NULL,
    estado VARCHAR(20) NOT NULL DEFAULT 'disponible',
    id_coleccion INT NOT NULL,
    FOREIGN KEY (id_coleccion) REFERENCES colecciones(id_coleccion)
);

```

-- Fin del script

7-Pruebas del Sistema

7.1-Pruebas Funcionales

Caso	Acción	Resultado Esperado	Resultado Obtenido	Estado
1	Registrar prenda y asociarla a una colección	La prenda se guarda y aparece en el listado de prendas de la colección	OK	Correcto
2	Editar prenda existente	Los cambios se guardan y se reflejan en la vista/listado	OK	Correcto
3	Eliminar (lógico) prenda	El estado de la prenda cambia a inactivo y deja de mostrarse en listados activos	OK	Correcto

4	Ver hijos por padre (prendas por colección)	Al seleccionar una colección se muestran solo sus prendas activas	OK	Correcto
5	Acceso a formulario de registro de colección	El formulario carga sin errores	OK	Correcto
6	Registrar una colección	Colección se registra correctamente en cada campo	OK	Correcto
7	Cambiar estado de colección	Se actualiza el estado a inactivo	OK	Correcto
8	Editar una colección	Se puede editar la colección correctamente	OK	Correcto

8-README Técnico en GitHub:

- URL del repositorio en GitHub:
<https://github.com/WilderSantamaria18/TiendaColecciones>
- Ruta del README en el repositorio (GitHub):
<https://github.com/WilderSantamaria18/TiendaColecciones/blob/main/README.md>

9-Reflexión:

1. ¿Qué aprendí en este proyecto?

Aprendí a organizar la aplicación en capas: Controller → Service → Repository. Usé DTOs y mappers para separar la vista de la lógica y la persistencia. Mejoré en depurar plantillas Thymeleaf y corregir bindings de formularios. Aprendí el flujo práctico: compilar, probar y versionar cambios con Git.

2. ¿Qué dificultades tuve y cómo las resolví?

Tuve errores 500 por expresiones Thymeleaf incompatibles; quité/ajusté esas expresiones. Los formularios esperaban Entities y los adapté para usar DTOs en las vistas.

3. ¿Qué mejoraría si tuviera más tiempo?

Añadiría tests automáticos (unitarios e integración) para cubrir la lógica crítica. Implementaría un @ControllerAdvice global para centralizar el manejo de errores.