Especificación de Requisitos de Software (ERS)
Sistema "Descansos del Recuerdo SPA"
Versión FullStack con Microservicios

César Rojas Ramos
Lucas Moncada
Profesor: Viviana Poblete

26 de noviembre de 2025

Índice

1. Introducción

En esta sección se presenta una introducción general al documento de Especificación de Requisitos de Software (ERS) del sistema "Descansos del Recuerdo SPA" en su versión FullStack con arquitectura de microservicios. Se describen el propósito del documento, el ámbito del sistema, las definiciones relevantes, las referencias utilizadas y una visión global de la estructura del documento.
Este documento actualiza y amplía la ERS original, que consideraba un backend simulado mediante `json-server`. El sistema implementado actualmente es una solución completa basada en microservicios desarrollados con Spring Boot y una base de datos relacional MySQL, junto con un frontend moderno en React.

1.1. Propósito

El propósito de este documento es definir de manera clara, completa y verificable los requisitos del sistema de gestión de urnas funerarias "Descansos del Recuerdo SPA", en su versión FullStack con arquitectura de microservicios.
Este documento está dirigido a:

- Desarrolladores y mantenedores del sistema, para comprender el alcance funcional y técnico.
- Docentes y evaluadores, como referencia formal de los requisitos implementados y pendientes.
- Stakeholders del negocio (empresa funeraria, clientes internos), para validar que el sistema satisface las necesidades del negocio.

La ERS sirve como contrato entre las partes interesadas y el equipo de desarrollo, y como base para el diseño, implementación, pruebas y mantenimiento del sistema.

1.2. Ámbito del Sistema

El sistema "Descansos del Recuerdo SPA" es una aplicación web FullStack destinada a la gestión integral del catálogo de urnas funerarias, el proceso de compra y la administración interna de la empresa. A diferencia de la versión inicial basada en `json-server`, el sistema actual implementa:

- Un **frontend** desarrollado en React 18 con Vite y Bootstrap 5.
- Un **backend** real compuesto por múltiples microservicios Spring Boot.
- Una **base de datos** MySQL 8 con persistencia real de datos.
- **Autenticación y autorización** basada en JWT, con control de roles (cliente y administrador).

Funciones principales del sistema:

- Publicar y mantener un catálogo de urnas funerarias con información detallada (materiales, dimensiones, precio, stock, etc.).
- Permitir a clientes navegar, filtrar y buscar urnas, ver detalles y gestionar un carrito de compras.
- Permitir a clientes registrados realizar pedidos, gestionar sus direcciones y consultar su historial de órdenes.
- Proporcionar a administradores un panel para gestionar urnas (CRUD), inventario, pedidos, usuarios y comentarios.
- Gestionar información geográfica de regiones y comunas (Chile) para direcciones de envío.
- Mostrar un foro y reseñas asociadas a urnas a través de un microservicio social.

El sistema NO contempla en esta versión:

- Integración con pasarelas de pago reales (pago en línea).
- Integraciones con sistemas externos de logística o facturación.
- Funcionalidad de multi-tienda o multi-empresa.

1.3. Definiciones, Siglas y Abreviaturas

Catálogo Sección del sistema que muestra las urnas disponibles, con información como nombre, descripción, precio, stock y material.

Urna Producto funerario ofrecido por la empresa, con atributos como material, dimensiones, color, precio, peso y disponibilidad.

Pedido Solicitud formal de compra realizada por un cliente, asociada a uno o más ítems de urnas y a una dirección de envío.

Detalle de pedido Línea individual dentro de un pedido, que indica urna, cantidad y precio unitario.

Carrito Conjunto temporal de urnas seleccionadas por el cliente antes de confirmar un pedido.

Cliente Usuario del sistema que navega por el catálogo y realiza compras. Puede ser invitado (carrito local) o registrado (carrito persistido).

Administrador Usuario con permisos avanzados para gestionar productos, inventario, pedidos, usuarios y comentarios.

Frontend Parte visual del sistema, desarrollada en React, con la que interactúan cliente y administrador mediante un navegador web.

Backend Conjunto de microservicios Spring Boot que exponen APIs REST y gestionan la lógica de negocio y la persistencia.

API REST Conjunto de endpoints HTTP que permiten el intercambio de datos en formato JSON entre frontend y backend.

JWT *JSON Web Token*, estándar para autenticación y autorización basada en tokens firmados.

1.4. Referencias

- Estándar IEEE 830 para Especificación de Requisitos de Software.
- ERS original del sistema "Descanzos del Recuerdo SPA" (versión con backend simulado mediante `json-server`).
- Repositorio del sistema FullStack con microservicios (código fuente del frontend y backend).
- Documentación Swagger expuesta por cada microservicio.

1.5. Visión General del Documento

Este documento está estructurado de acuerdo al estándar IEEE 830:

- La Sección 1 presenta la introducción, alcance, definiciones y referencias.
- La Sección 2 describe el sistema desde un punto de vista general: arquitectura, funciones, usuarios, restricciones y suposiciones.
- La Sección 3 detalla los requisitos específicos, tanto funcionales como no funcionales, incluyendo interfaces externas, casos de uso por rol y restricciones de calidad.

El objetivo es que el documento sirva como guía para el desarrollo, validación y mantenimiento del sistema FullStack de "Descansos del Recuerdo SPA".

2. Descripción General

En esta sección se describen los factores que afectan al sistema y su contexto, sin entrar aún en el detalle exhaustivo de cada requisito funcional.

2.1. Perspectiva del Producto

El sistema "Descansos del Recuerdo SPA" se concibe como una solución web de comercio electrónico especializada en urnas funerarias, implementada con una arquitectura de microservicios.

Arquitectura general

- **Frontend**: Aplicación SPA (Single Page Application) en React 18, creada con Vite, que utiliza Bootstrap 5 para el diseño responsivo y Context API para la gestión de estado global (por ejemplo, el carrito).

- **Backend**: Conjunto de microservicios Spring Boot (versión 3.x), cada uno con responsabilidad bien definida:
  - Microservicio de autenticación y usuarios.
  - Microservicio de catálogo de urnas y catálogos de soporte (materiales, colores, modelos).
  - Microservicio de inventario.
  - Microservicio de pedidos y carrito.
  - Microservicio de ubicaciones (regiones y comunas).
  - Microservicio social (comentarios de foro y reseñas de urnas).
- **Base de datos**: MySQL 8 con esquema normalizado para usuarios, urnas, inventario, pedidos, detalles, comentarios, direcciones y catálogos de soporte.

El producto se integra como una solución cliente-servidor accesible mediante navegador web moderno, usando comunicación HTTP/HTTPS y datos en formato JSON.

2.2. Funciones del Producto

A nivel macro, el sistema ofrece las siguientes funcionalidades:

- **Catálogo público de urnas**:
  - Listado paginado de urnas con imagen, nombre, precio y disponibilidad.
  - Filtros por nombre, código interno, material y rango de precio.
  - Visualización de detalles de cada urna (descripción, dimensiones, peso, material, color, modelo, galería de imágenes).
- **Carrito de compras**:
  - Agregar urnas al carrito desde la vista de catálogo o desde el detalle.
  - Ver un carrito flotante con contador de ítems.
  - Modificar cantidades y eliminar productos.
  - Persistencia del carrito en `localStorage` (invitado) y en el backend (usuario autenticado).
- **Gestión de pedidos**:
  - Proceso de checkout con selección o registro de dirección de envío.
  - Creación de pedidos con estado inicial (por ejemplo, "Pendiente").
  - Disminución automática del stock al confirmar pedido.
  - Consulta de pedidos por parte del cliente y del administrador.
- **Gestión de usuarios y autenticación**:
  - Registro de nuevos usuarios (rol cliente).
  - Inicio de sesión con correo y contraseña.
  - Recuperación de contraseña mediante flujo de restablecimiento.
  - Sesiones seguras mediante tokens JWT.
- **Panel administrativo**:
  - CRUD completo de urnas, incluidas múltiples imágenes por producto.
  - Gestión de inventario (ajustes, aumentos y disminuciones de stock con motivos y registro de usuario responsable).
  - Gestión de pedidos (cambio de estado, eliminación de pedidos cancelados).
  - Gestión de usuarios (listado, edición, cambio de estado y eliminación).
  - Dashboard con métricas generales del sistema.

- **Ubicaciones y direcciones**:
  - Consulta de regiones y comunas de Chile.
  - Gestión de direcciones por usuario.
- **Foro y reseñas**:
  - Listado de comentarios de foro.
  - Reseñas asociadas a urnas específicas.
  - Creación de nuevos comentarios o reseñas por usuarios.

2.3. Características de los Usuarios

Cliente invitado

- Persona que accede al sitio sin autenticación.
- Puede navegar por el catálogo, ver detalles de urnas y gestionar un carrito temporal (persistido en el navegador).
- No puede confirmar pedidos hasta registrarse e iniciar sesión.

Cliente registrado

- Usuario con cuenta en el sistema (rol "Cliente").
- Además de lo anterior, puede:
  - Confirmar pedidos y revisar su historial.
  - Gestionar direcciones de envío.
  - Acceder a funcionalidades sociales (comentarios y reseñas), según las reglas definidas.

Administrador

- Usuario con rol "Administrador".
- Responsable de gestionar el catálogo de urnas, inventario, pedidos y usuarios.
- Requiere conocimientos básicos de herramientas de gestión y uso de plataformas web.

2.4. Restricciones

- El sistema debe estar desarrollado con las tecnologías definidas:
  - Frontend: React, Vite, Bootstrap.
  - Backend: Spring Boot, Java, APIs REST.
  - Base de datos: MySQL 8.
- El sistema debe ser accesible desde navegadores modernos en dispositivos de escritorio y móviles (diseño responsivo).
- La solución se enfoca exclusivamente en la venta de urnas funerarias; no se contemplan otros tipos de productos.
- La autenticación y autorización deben implementarse mediante JWT, con separación clara de roles.
- Todos los microservicios deben estar debidamente documentados a través de Swagger u otra herramienta similar.
- Las credenciales sensibles (por ejemplo, de base de datos) deben gestionarse mediante variables de entorno en los entornos de despliegue.

2.5. Suposiciones y Dependencias

- Se asume la disponibilidad de un servidor de aplicaciones para desplegar los microservicios Spring Boot y una instancia de MySQL accesible.
- Se asume que los usuarios cuentan con acceso a Internet y un navegador moderno.
- Se asume que la empresa dispondrá de personal con rol de administrador para mantener actualizado el catálogo y el inventario.
- El sistema depende de:
  - El correcto funcionamiento de los microservicios y su comunicación vía HTTP.
  - La disponibilidad de la base de datos MySQL.
  - La configuración adecuada de CORS para permitir el acceso desde el frontend.

3. Requisitos Específicos

En esta sección se detallan los requisitos específicos del sistema, incluyendo interfaces externas, requisitos funcionales y no funcionales.

3.1. Interfaces Externas

3.1.1. Interfaces de Usuario

El sistema cuenta con las siguientes interfaces de usuario principales:

Pantallas públicas (cliente)

- **Página de catálogo**:
  - Ruta: `/` o `/catalogo`.
  - Muestra un grid responsivo de tarjetas de urnas, con imagen principal, nombre, precio y estado de stock.
  - Ofrece filtros por nombre/código, material y rango de precio.
  - Incluye paginación (por ejemplo, 12 productos por página).
  - Muestra un botón flotante de carrito siempre visible con contador de ítems.
- **Modal de detalle de urna**:
  - Se abre al hacer clic en una tarjeta del catálogo.
  - Presenta descripción detallada, dimensiones, peso, material, color, modelo y galería de imágenes.
  - Incluye botón para agregar al carrito, mostrando el stock disponible.
- **Carrito de compras**:
  - Componente flotante con badge de cantidad de ítems.
  - Modal con listado de productos, cantidades editables y cálculo automático del total.
  - Botón para proceder al checkout.
- **Página "Nosotros"**:
  - Ruta: `/nosotros`.
  - Muestra información institucional de la empresa.
- **Pantallas de autenticación**:
  - Rutas: `/login`, `/registro`, y eventualmente pantallas de recuperación de contraseña.
  - Formularios con validación de campos obligatorios y formato de correo.

Panel administrativo

- **Layout de administración**:
  - Ruta base: `/admin`.
  - Barra lateral o menú con acceso a sección de urnas, inventario, pedidos, usuarios y dashboard.
  - Todo el contenido protegido mediante rutas que requieren rol de administrador.
- **Gestión de urnas**:
  - Ruta: `/admin/urnas`.
  - Tabla paginada con búsqueda y filtros.
  - Botones de acción para crear, editar y eliminar urnas.
  - Modal con formulario completo de urna e imágenes.
- **Gestión de inventario**:
  - Ruta: `/admin/inventario`.
  - Tabla con stock actual, cantidad mínima y máxima.
  - Formularios para ajustar, aumentar o disminuir stock, indicando motivo.
- **Gestión de pedidos**:
  - Ruta: `/admin/pedidos`.
  - Tabla con pedidos, datos del cliente, dirección y estado actual.
  - Controles para cambiar estado y eliminar pedidos.
- **Gestión de usuarios**:
  - Ruta: `/admin/usuarios`.
  - Listado de usuarios con búsqueda y filtros.
  - Formulario para editar datos, rol y estado (activo/inactivo).

3.1.2. Interfaces de Hardware

- Servidor de aplicaciones capaz de ejecutar los microservicios Spring Boot y el servidor de base de datos MySQL.
- Estaciones de trabajo de los usuarios con navegador moderno (Chrome, Edge, Firefox, etc.) y conexión a Internet.

3.1.3. Interfaces de Software

A nivel de software, el sistema expone e invoca varias APIs REST agrupadas por microservicio. A continuación se resumen las principales:

Microservicio de autenticación y usuarios

- Autenticación:
  - `POST /api/auth/login`: Iniciar sesión (devuelve token JWT).
  - `POST /api/auth/register`: Registrar usuario.
  - `GET /api/auth/me`: Obtener perfil del usuario autenticado.
  - `POST /api/auth/forgot-password`: Solicitar restablecimiento de contraseña.
  - `POST /api/auth/reset-password`: Confirmar restablecimiento de contraseña.
- Gestión de usuarios:
  - `GET /api/usuarios`: Listar usuarios.
  - `GET /api/usuarios/{id}`: Obtener un usuario por su identificador.
  - `PUT /api/usuarios/{id}`: Actualizar datos de un usuario.
  - `DELETE /api/usuarios/{id}`: Eliminar usuario.
- Direcciones:
  - `GET /api/direcciones/usuario/{usuarioId}`: Listar direcciones de un usuario.
  - `POST /api/direcciones`: Crear nueva dirección.
  - `PUT /api/direcciones/{id}`: Actualizar dirección.

Microservicio de catálogo

- Urnas:
  - `GET /api/urnas`: Listar urnas (con posibles filtros en la query).
  - `GET /api/urnas/{id}`: Obtener urna específica.
  - `POST /api/urnas`: Crear nueva urna.
  - `PATCH /api/urnas/{id}`: Actualizar urna.
  - `DELETE /api/urnas/{id}`: Eliminar urna.
- Catálogos de soporte:
  - `GET /api/materiales`, `/api/colores`, `/api/modelos`: Listar materiales, colores y modelos.
- Imágenes:
  - `POST /api/imagenes/{urnaId}`: Subir imagen de urna (`multipart/form-data`).

Microservicio de inventario

- `GET /api/inventario`: Listar inventario completo.
- `GET /api/inventario/urna/{urnaId}`: Obtener inventario de una urna.
- `PATCH /api/inventario/ajustar/{urnaId}`: Ajustar stock.
- `PATCH /api/inventario/aumentar/{urnaId}`: Aumentar stock.
- `PATCH /api/inventario/disminuir/{urnaId}`: Disminuir stock.

Microservicio de pedidos y carrito

- Pedidos:
  - `POST /api/pedidos`: Crear nuevo pedido.
  - `GET /api/pedidos`: Listar todos los pedidos.
  - `GET /api/pedidos/usuario/{usuarioId}`: Listar pedidos de un usuario.
  - `PUT /api/pedidos/{id}/estado/{estadoId}`: Cambiar estado del pedido.
  - `DELETE /api/pedidos/{id}`: Eliminar pedido.
- Carrito:
  - `GET /api/carrito/{usuarioId}`: Obtener carrito persistido de un usuario.
  - `POST /api/carrito/{usuarioId}/items`: Agregar ítem al carrito.
  - `DELETE /api/carrito/{usuarioId}/items/{urnaId}`: Eliminar ítem.
  - `DELETE /api/carrito/{usuarioId}`: Vaciar carrito.

Microservicio de ubicaciones

- `GET /api/regiones`: Listar regiones.
- `GET /api/comunas/region/{regionId}`: Listar comunas de una región.

Microservicio social

- `GET /api/comentarios/foro`: Listar comentarios del foro.
- `GET /api/comentarios/urna/{urnaId}`: Listar reseñas de una urna.
- `POST /api/comentarios`: Crear comentario o reseña.

3.2. Requisitos Funcionales

A continuación se detallan los requisitos funcionales, agrupados por rol principal.

3.2.1. Requisitos funcionales para el rol Cliente

RF-CLI-1 El sistema debe permitir a cualquier usuario (invitado o autenticado) visualizar el catálogo de urnas en una vista de tarjetas con paginación.

RF-CLI-2 El sistema debe permitir filtrar el catálogo por nombre o código interno, material y rango de precio.

RF-CLI-3 El sistema debe mostrar, para cada urna, el stock disponible en tiempo real, combinando la información de catálogo e inventario.

RF-CLI-4 El sistema debe permitir al usuario visualizar un detalle ampliado de una urna, con descripción completa, dimensiones, peso, material, color, modelo y galería de imágenes.

RF-CLI-5 El sistema debe permitir agregar una o más urnas al carrito desde la vista de catálogo y desde la vista de detalle.

RF-CLI-6 El sistema debe mantener un carrito de compras para usuarios invitados mediante almacenamiento en el navegador.

RF-CLI-7 El sistema debe mantener un carrito de compras persistente para usuarios autenticados utilizando el microservicio de carrito.

RF-CLI-8 El sistema debe permitir al usuario consultar el contenido del carrito en un modal, modificar cantidades y eliminar productos.

RF-CLI-9 El sistema debe recalcular automáticamente el total del carrito al modificar cantidades o eliminar productos.

RF-CLI-10 El sistema debe permitir al cliente registrado iniciar el proceso de checkout desde el carrito.

RF-CLI-11 El sistema debe requerir que el cliente esté autenticado para confirmar un pedido.

RF-CLI-12 El sistema debe permitir al cliente seleccionar una dirección de envío existente o registrar una nueva durante el checkout.

RF-CLI-13 El sistema debe generar un pedido con estado inicial (por ejemplo, “Pendiente”) al completar correctamente el checkout.

RF-CLI-14 El sistema debe disminuir el stock de inventario correspondiente a cada urna incluida en un pedido confirmado.

RF-CLI-15 El sistema debe permitir al cliente visualizar el historial de sus pedidos, incluyendo fecha, estado y total.

RF-CLI-16 El sistema debe permitir a un usuario registrarse proporcionando nombre, correo electrónico y contraseña, con validaciones adecuadas.

RF-CLI-17 El sistema debe permitir a un usuario autenticarse mediante correo y contraseña y recibir un token JWT en caso de éxito.

RF-CLI-18 El sistema debe permitir iniciar un flujo de recuperación de contraseña mediante el envío de correo electrónico.

RF-CLI-19 El sistema debe permitir a los clientes gestionar sus direcciones (crear, editar y listar).

RF-CLI-20 El sistema debe permitir a los clientes visualizar comentarios del foro y reseñas asociadas a urnas.

RF-CLI-21 El sistema debe permitir a los clientes autenticados crear nuevos comentarios o reseñas, según las políticas definidas.

3.2.2. Requisitos funcionales para el rol Administrador

RF-ADM-1 El sistema debe permitir a los administradores acceder a un panel de administración protegido.

RF-ADM-2 El sistema debe permitir listar todas las urnas en una tabla paginada con filtros y búsqueda.

RF-ADM-3 El sistema debe permitir crear nuevas urnas mediante un formulario que incluya nombre, descripciones, precio, dimensiones, peso, material, color, modelo, disponibilidad y estado.

RF-ADM-4 El sistema debe permitir editar urnas existentes, actualizando cualquier campo relevante.

RF-ADM-5 El sistema debe permitir eliminar urnas, previa confirmación.

RF-ADM-6 El sistema debe permitir subir una o más imágenes asociadas a una urna.

RF-ADM-7 El sistema debe permitir visualizar y gestionar el inventario de cada urna (stock actual, mínimo, máximo, estado y ubicación física).

RF-ADM-8 El sistema debe permitir ajustar manualmente el stock de una urna, registrando motivo y usuario responsable.

RF-ADM-9 El sistema debe permitir aumentar o disminuir stock de forma controlada, registrando los movimientos.

RF-ADM-10 El sistema debe permitir listar todos los pedidos realizados por los clientes.

RF-ADM-11 El sistema debe permitir visualizar el detalle de un pedido (cliente, dirección, ítems, total, estado).

RF-ADM-12 El sistema debe permitir cambiar el estado de los pedidos (por ejemplo: Pendiente, Preparado, Despachado, Entregado, Cancelado).

RF-ADM-13 El sistema debe permitir eliminar pedidos cancelados o con errores, previa confirmación.

RF-ADM-14 El sistema debe permitir listar todos los usuarios registrados con sus datos básicos y rol.

RF-ADM-15 El sistema debe permitir editar la información de usuarios (nombre, correo, rol, estado).

RF-ADM-16 El sistema debe permitir activar o desactivar usuarios mediante un campo de estado.

RF-ADM-17 El sistema debe permitir eliminar usuarios, previa confirmación y según las restricciones de negocio.

RF-ADM-18 El sistema debe permitir revisar y moderar comentarios o reseñas, según las políticas definidas (por ejemplo, ocultar comentarios inapropiados).

3.2.3. Requisitos funcionales comunes

RF-COM-1 El sistema debe validar en el frontend los campos obligatorios en los formularios (login, registro, gestión de urnas, etc.).

RF-COM-2 El sistema debe validar en el backend la integridad de los datos recibidos (tipos, formatos, rangos).

RF-COM-3 El sistema debe utilizar tokens JWT para autenticar y autorizar las operaciones de los usuarios.

RF-COM-4 El sistema debe proteger las rutas del frontend que requieren autenticación, redirigiendo a la pantalla de login en caso de no contar con sesión válida.

RF-COM-5 El sistema debe restringir el acceso a las funciones administrativas únicamente a usuarios con rol administrador.

3.3. Requisitos No Funcionales

3.3.1. Requisitos de Rendimiento

- El sistema debe soportar la navegación por el catálogo con respuesta percibida como ágil por el usuario (tiempos de respuesta típicos inferiores a 2–3 segundos en operaciones normales).
- Debe implementarse paginación en las vistas de listado para evitar cargas excesivas de datos en una sola petición.
- Debe existir una caché en memoria para listas estáticas (como materiales, colores y modelos) con un tiempo de vida configurable (por ejemplo, 5 minutos).
- Cada petición HTTP del frontend al backend debe tener un *timeout* configurado (por ejemplo, 15 segundos) para evitar bloqueos indefinidos.

3.3.2. Seguridad

- Las credenciales de los usuarios (contraseñas) deben almacenarse en la base de datos en forma encriptada (por ejemplo, con BCrypt).
- El sistema debe utilizar JWT para gestionar sesiones de usuario en el frontend y autenticación en el backend.
- Las rutas protegidas del frontend deben validar la presencia y validez del token JWT antes de mostrar contenido sensible.
- Debe configurarse CORS para limitar el acceso a los microservicios únicamente desde dominios autorizados.
- Deben implementarse validaciones y sanitización de entradas para reducir el riesgo de ataques de inyección o XSS.

3.3.3. Fiabilidad

- Debe existir un manejo centralizado de errores HTTP en el cliente, mostrando mensajes comprensibles para el usuario.
- Las operaciones críticas (como crear pedidos, ajustar stock, eliminar registros) deben requerir confirmación del usuario.
- Deben manejarse adecuadamente los errores de red (desconexiones, tiempo de espera agotado), informando al usuario y permitiendo reintentos.
- El sistema debe garantizar la consistencia entre pedidos e inventario, asegurando que la disminución de stock se realice de forma transaccional cuando se crea un pedido.

3.3.4. Disponibilidad

- La arquitectura basada en microservicios debe permitir desplegar cada servicio de forma independiente, facilitando mantenimiento y escalabilidad.
- Deben estar disponibles scripts de inicialización de la base de datos y documentación de despliegue para reproducir el entorno en distintos servidores.
- Se recomienda implementar mecanismos de supervisión (por ejemplo, *health checks*) para verificar el estado de los microservicios.

3.3.5. Mantenibilidad

- La solución debe mantener una separación clara de responsabilidades entre frontend, microservicios y capa de datos.
- El código debe organizarse en módulos y componentes reutilizables (por ejemplo, componentes de React bien separados por función).
- La documentación del código (comentarios y descripciones en Swagger) debe mantenerse actualizada.

3.3.6. Pruebas y cobertura (frontend)

El frontend del sistema "Descansos del Recuerdo SPA" cuenta con pruebas unitarias automatizadas, ejecutadas mediante *Vitest* y *@testing-library/react*, orientadas a garantizar el correcto funcionamiento de la lógica de validación y de componentes de interfaz críticos. Actualmente se dispone de:

- **Pruebas de validadores de negocio**
  - Archivo de pruebas: `src/utils/validators.test.js`.
  - Módulo cubierto: `src/utils/validators.js`.
  - Funciones probadas, entre otras:
    - `isValidEmail`: valida correos electrónicos válidos e inválidos.
    - `isValidPrice`: rechaza precios negativos o cero y acepta precios positivos.
    - `isValidStock`: valida que el stock sea numérico y mayor o igual a cero.
    - `isValidPassword`: exige un mínimo de caracteres para la contraseña.
    - `isPositiveNumber`: verifica valores numéricos positivos.
    - `isValidName`, `isValidAddress`, `isValidPhone`: validan formato de nombre, longitud mínima de dirección y formato de teléfono chileno.
    - `isRequired`: verifica que un campo obligatorio no sea nulo ni vacío.
    - `validateProductForm`: retorna un objeto de errores cuando el formulario de producto es inválido y un objeto vacío cuando es válido.

- **Pruebas de componente de interfaz de usuario**
  - Archivo de pruebas: `src/components/common/AlertBadge.test.jsx`.
  - Componente cubierto: `src/components/common/AlertBadge.jsx`.
  - Casos probados:
    - Cuando el stock es `0`, el componente muestra el texto *"Agotado"* y aplica la clase CSS correspondiente al estado agotado.
    - Cuando el stock es menor al umbral de stock bajo (por ejemplo, `3`), muestra una advertencia de stock bajo (incluyendo la cantidad) y la clase de estilo de alerta.
    - Cuando el stock es suficiente (por ejemplo, `10`), muestra el texto de stock normal ("Stock: 10") y la clase de estilo de éxito.
- **Cobertura actual de pruebas (evidencia)**
  - La ejecución de `npm run test - -coverage` (o comando equivalente) reporta:
    - 2 archivos de pruebas ejecutados.
    - 14 pruebas en total, todas con resultado satisfactorio (*passed*).
    - 100 % de cobertura de sentencias, ramas, funciones y líneas para los módulos `src/utils/validators.js` y `src/components/common/AlertBadge.jsx`.
- **Requisito de mantenimiento de pruebas**
  - A medida que se incorporen nuevas reglas de validación o componentes críticos al frontend, se deben extender las pruebas unitarias para mantener una cobertura alta sobre la lógica de negocio y la interfaz asociada al flujo de compra (catálogo, carrito y gestión de formularios).

3.3.7. Portabilidad

- El frontend basado en React debe poder desplegarse en cualquier servidor capaz de servir archivos estáticos o en plataformas de hospedaje modernas.
- Los microservicios Spring Boot deben poder ejecutarse en entornos compatibles con Java, como servidores on-premise o servicios en la nube.
- El uso de MySQL como base de datos facilita la migración entre distintas plataformas que soporten este motor.

3.4. Otros Requisitos

- La interfaz gráfica debe mantener una estética acorde al rubro funerario: sobria, respetuosa y clara.
- La aplicación debe seguir buenas prácticas de accesibilidad siempre que sea posible (contraste adecuado, tamaño de fuente legible, mensajes de error claros).
- Se recomienda disponer de un entorno de pruebas para validar nuevas funcionalidades antes de su paso a producción.