**UNIVERSIDAD NACIONAL DE INGENIERÍA**

**FACULTAD DE INGENIERÍA INDUSTRIAL Y DE SISTEMAS**



**ESCUELA DE INGENIERÍA DE SISTEMAS**

**Curso:**

**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Proyecto:**

**Sistema de de control de pedidos en un taller de manufactura**

**Autores:**

Chicmana Arias, Piero Alejandro ([orcid.org/0009-0009-7963-6901](https://orcid.org/0009-0009-7963-6901))

Carlos Falcón, Enrique Armando ([orcid.org/0009-0003-6091-1332](https://orcid.org/0009-0003-6091-1332))

**Docente:**

Mag. Coronel Castillo, Eric Gustavo ([orcid.org/0000-0003-0494-5629](http://orcid.org/0000-0003-0494-5629))

**LIMA – PERÚ**

**2025**

# DEDICATORIA

A nuestro querido profesor,

Este proyecto, más allá de ser un simple conjunto de líneas de código, es el reflejo de todo lo que hemos aprendido bajo su guía. Cada decisión tomada, cada algoritmo implementado y cada validación diseñada ha sido posible gracias a la base sólida que nos ha proporcionado. En un mundo que avanza a la velocidad de la inteligencia artificial, donde la tecnología redefine constantemente lo posible, usted ha sido nuestro mentor, enseñándonos no solo el "cómo", sino también el "por qué" de la programación.  
“DIVIDE Y VENCERÁS”

# ÍNDICE GENERAL

Pág.

[**DEDICATORIA 3**](#_62hdyorlex4y)

[**ÍNDICE GENERAL 4**](#_nvute62x0glr)

[**RESUMEN 6**](#_noj72wjnlow2)

[**ABSTRACT 7**](#_gztgf4ddh5qj)

[**INTRODUCCIÓN 8**](#_qkclk9f0ounn)

[**OBJETIVOS 9**](#_fr09bydjg5di)

[OBJETIVO GENERAL 9](#_mpb9g2epbduc)

[OBJETIVOS ESPECÍFICOS 9](#_ubmp8h7ex6ql)

[**REQUERIMIENTOS 10**](#_85erlkxo41g1)

[REQUERIMIENTOS FUNCIONALES 10](#_ikhrmq5b5561)

[REQUERIMIENTOS NO FUNCIONALES 11](#_wbc6msbu2zu0)

[ALCANCES 11](#_7o8bd5lb83fq)

[LÍMITES 13](#_wa38pd2427m1)

[**ANÁLISIS DE REQUERIMIENTOS 15**](#_63766m9oc0m2)

[IDENTIFICCIÓN DE ACTORES 15](#_k1xe1j9vtle)

[CASOS DE USOS 15](#_7zi0bz709gux)

[**CU01 – MATRÍCULA 17**](#_e65k0nljhpr)

[ESPECIFICACIÓN DEL CASO DE USO 17](#_n650vzml65xn)

[CRITERIOS DE ACEPTACIÓN 20](#_36ruqav52n02)

[**CU02 – REGISTRAR PAGO DE CUOTA 23**](#_jfv4a6p107r2)

[ESPECIFICACIÓN DEL CASO DE USO 23](#_np78gg3nf0ke)

[CRITERIOS DE ACEPTACIÓN 26](#_wdgrtl5jtesr)

[**CU03 – CONSULTAS 29**](#_g0y892dc7xcn)

[ESPECIFICACIÓN DEL CASO DE USO 29](#_1jhbbsrwctsu)

[CRITERIOS DE ACEPTACIÓN 32](#_iyw6hr6up60w)

[**MODELO DE BASE DE DATOS 35**](#_64drk7ogh0uu)

[MODELO RELACIONAL 35](#_zc8imojitt9)

[**IMPLEMENTACIÓN DE LA BASE DE DATOS 40**](#_rrpsmxejxfvt)

[CREACIÓN DE LA BASE DE DATOS 40](#_7r6e1zhkpd7b)

[CREACIÓN DE LOS OBJETOS DE LA BASE DE DATOS 40](#_ajd3hss47j1)

[**CONCLUSIONES 81**](#_cyfyvzy3d6rg)

[**LECCIONES APRENDIDAS 83**](#_9m755c3743af)

[**BIBLIOGRAFÍA 84**](#_vn80n13a6erp)

[**ANEXOS 85**](#_bs8dvma026ng)

[Anexo 1: Titulo 86](#_bs8dvma026ng)

[Anexo 2: Titulo 87](#_bs8dvma026ng)

# RESUMEN

El presente informe describe el desarrollo del módulo backend del “Sistema de control de pedidos en un taller de manufactura” denominado Mecafab. Este módulo, implementado con Spring Boot y SQL Server, abarca las siguientes funcionalidades principales:

1. Registro de pedidos: Permite ingresar nuevas órdenes con datos del cliente, descripción del producto, fechas de solicitud y compromiso, generando un código único de pedido.
2. Asignación de pedidos: El jefe de taller puede asignar cada pedido a un operario específico, actualizando el estado a “En Producción” y registrando la fecha de modificación.
3. Actualización de estado: Se registran los cambios de estado intermedios en la tabla de historial, evitando asignaciones redundantes y asegurando trazabilidad.
4. Alertas de vencimiento: Mediante consultas a la base de datos, el sistema identifica pedidos próximos a su fecha comprometida (falta ≤ 2 días) y los filtra para notificar al jefe de taller.
5. Confirmación de entrega y comprobantes: Al marcar un pedido como entregado, se actualiza su estado a “Entregado”, se almacenan las fechas real y confirmada, y se genera un comprobante de entrega.
6. Validaciones y manejo de errores: Se aplican restricciones de integridad en la base de datos (CHECK), validaciones de DTOs con Jakarta Bean Validation y manejo centralizado de excepciones REST.

# ABSTRACT

This document presents the design and implementation of the backend module for **Mecafab**, an order control system developed for a manufacturing workshop. Built with Spring Boot and SQL Server, the backend provides a RESTful API that supports: (1) creation of new orders with unique codes and client data; (2) assignment of orders to operators, updating their status to “In Production”; (3) recording of all intermediate state changes in a history table to ensure full traceability; (4) automated identification of orders nearing their committed delivery date (≤ 2 days) for proactive alerts; (5) confirmation of delivery with real delivery timestamps and generation of delivery receipts; and (6) comprehensive data-validation and business-rule enforcement via Bean Validation and custom exceptions. The database schema enforces referential integrity and domain constraints, while the service layer orchestrates transactional workflows. Centralized exception handling guarantees consistent JSON error responses. This robust, layered architecture replaces manual procedures, enhancing planning accuracy, operational efficiency, and auditability.

# INTRODUCCIÓN

El presente informe describe el desarrollo del módulo **backend** del sistema de control de pedidos para el taller de manufactura “Mecafab”. En el contexto actual, la gestión manual de órdenes —realizada mediante formularios de papel y un pizarrón en el taller— ha demostrado ser ineficiente: se pierden registros, los retrasos no se detectan a tiempo y la visibilidad del flujo de trabajo es limitada.

Con el objetivo de digitalizar y optimizar este proceso, se eligió la arquitectura de microservicios basada en **Spring Boot** para exponer una API RESTful robusta, y **SQL Server** como sistema de gestión de la base de datos. Esta combinación permite separar claramente las capas de presentación, negocio y datos, facilitando la mantenibilidad, escalabilidad y seguridad de la aplicación.

La sección de Backend implementa las siguientes funcionalidades clave:

1. **Registro de pedidos**: generación de un identificador único y almacenamiento de datos del cliente, descripción del producto, fechas de solicitud y entrega comprometida.
2. **Asignación y cambio de estado**: asignación de operarios, registro de cada transición de estado en un histórico y validación para evitar cambios redundantes.
3. **Alertas de vencimiento**: consulta de pedidos con fecha comprometida próxima (≤ 2 días) para notificaciones proactivas.
4. **Confirmación de entrega**: registro de la fecha real y generación de comprobantes de entrega, con validaciones de integridad de datos.
5. **Manejo de errores**: validaciones a nivel de DTO y base de datos, excepciones de negocio y control centralizado de errores REST.

Además, se detallan los requerimientos funcionales y no funcionales, la estructura de la base de datos (modelo relacional y diccionario de datos), los criterios de aceptación para cada caso de uso y los límites del sistema. Este enfoque integral garantiza una solución que mejora la planificación, reduce errores manuales y ofrece trazabilidad completa del ciclo de vida de cada pedido.

# OBJETIVOS

## OBJETIVO GENERAL

Desarrollar e implementar el módulo backend del sistema de control de pedidos **Mecafab**, basado en Spring Boot y SQL Server, que permita gestionar de forma digital todo el ciclo de vida de las órdenes —desde su creación hasta la entrega— garantizando integridad, trazabilidad y alertas proactivas.

## OBJETIVOS ESPECÍFICOS

* **Diseñar el modelo de datos** relacional que incluya tablas para clientes, pedidos, estados, historial de cambios y comprobantes de entrega, estableciendo claves primarias, foráneas y restricciones de integridad.
* **Implementar la creación de pedidos** mediante un endpoint REST que genere códigos únicos y almacene datos de cliente, descripción, fechas de solicitud y compromiso.
* **Desarrollar la asignación y actualización de estado** de pedidos, registrando cada transición en una tabla de historial para asegurar trazabilidad completa.
* **Configurar consultas de alerta** que identifiquen automáticamente los pedidos con fecha comprometida próxima (≤ 2 días) para notificaciones al jefe de taller.
* **Implementar la confirmación de entrega**, actualizando el estado a “Entregado”, registrando la fecha real y generando el comprobante de entrega de forma transaccional.
* **Aplicar validaciones y manejo de errores**, utilizando Jakarta Bean Validation en los DTOs, validaciones de base de datos (CHECK) y un controlador global de excepciones REST para respuestas JSON consistentes.
* **Documentar la API** y definir los criterios de aceptación para cada caso de uso, facilitando su mantenimiento y futura evolución.

# REQUERIMIENTOS

## REQUERIMIENTOS FUNCIONALES

**RF1 - Registrar Pedidos**

* **Actor principal**: Jefe de taller (usuario autenticado).
* **Descripción**: El sistema permite crear un nuevo pedido, generando automáticamente un código único con formato PED-YYYYMM-XXX. Se capturan: datos del cliente, descripción del producto, tipo de trabajo, fecha de solicitud y fecha de entrega comprometida.
* **Entradas**:
  + Identificador del cliente (ID).
  + Descripción del producto.
  + Tipo de trabajo.
  + Fecha de entrega comprometida.
* **Salida**:
  + Confirmación de creación con código de pedido.
* **Precondición**:
  + El usuario debe estar autenticado y tener rol “Jefe”.
  + El cliente debe existir en la base de datos.
* **Postcondición**:
  + Se inserta un registro en Pedido con estado\_id = 1 (Registrado).
  + Se registra la fecha de creación (fecha\_registro) y el historial inicial en HistorialEstadoPedido.

**RF2 – Asignar Pedidos**

* **Actor principal**: Jefe de taller.
* **Descripción**: Permite asignar un operario a un pedido existente, cambiando su estado a “En Producción” y registrando la fecha de modificación.
* **Entradas**:
  + Código o ID del pedido.
  + ID del operario a asignar.
* **Salida**:
  + Confirmación de la asignación y del nuevo estado.
* **Precondición**:
  + El pedido debe existir y estar en estado “Registrado”.
  + El operario debe existir y estar activo.
* **Postcondición**:
  + Se actualizan operario\_asignado\_id, estado\_id = 2 (En Producción) y fecha\_modificacion.
  + Se añade un registro en HistorialEstadoPedido con la transición.

**RF3 – Actualizar Estado del Pedido**

* **Actor principal**: Operario / Empleado.
* **Descripción**: Permite cambiar el estado de un pedido a cualquiera de los intermedios (“Listo para Entregar”, “En Espera de Material”, etc.), siempre que el nuevo estado difiere del actual, y registra cada cambio para trazabilidad.
* **Entradas**:
  + Código o ID del pedido.
  + Nuevo estado (ID de EstadoPedido).
* **Salida**:
  + Confirmación del cambio aplicado.
* **Precondición**:
  + El pedido debe existir.
  + El nuevo estado debe tener un orden\_logico superior al actual y no ser igual.
* **Postcondición**:
  + Se actualiza estado\_id, fecha\_modificacion.
  + Se inserta un registro en HistorialEstadoPedido.

**RF4 – Alertas de Vencimientos o Retrasos**

* **Actor principal**: Sistema (visualización por Jefe de taller).
* **Descripción**: El sistema identifica y lista los pedidos cuyo plazo de entrega comprometida vence en ≤ 2 días o ya está vencido, para permitir acciones preventivas.
* **Entradas**:
  + Parámetro de anticipación (por defecto, 2 días).
* **Salida**:
  + Listado de pedidos próximos a vencer o vencidos, con sus fechas y responsables.
* **Precondición**:
  + Existen pedidos en la base de datos con estado distinto de “Entregado” o “Cancelado”.
* **Postcondición**:
  + Se muestra un reporte/filter endpoint consumible por el frontend o consola.

**RF5 – Registrar Entrega Final y Generar Comprobante**

* **Actor principal:** Jefe de taller / Supervisor.
* **Descripción**: Permite confirmar la entrega final de un pedido. Al hacerlo, se registran las fechas de entrega real y confirmada, se marca el pedido como entregado y se genera un comprobante (guardado en ComprobanteEntrega).
* **Entradas**:
  + Código o ID del pedido.
* **Salida**:
  + Confirmación de entrega y contenido del comprobante.
* **Precondición**:
  + El pedido debe estar en estado “Listo para Entregar” y no estar ya marcado como entregado.
* **Postcondición**:
  + Se actualizan es\_entregado = 1, fecha\_entrega\_real, fecha\_entrega\_confirmada.
  + Se inserta un registro en ComprobanteEntrega.

## REQUERIMIENTOS NO FUNCIONALES

**Usabilidad**

El sistema debe contar con una **interfaz amigable y accesible** para usuarios que no son expertos en informática, como operarios o jefes de taller. Las pantallas deben ser simples e intuitivas: listas de pedidos con botones visibles para cambiar estados y formularios claros para registrar nuevos pedidos, reduciendo al mínimo la posibilidad de error humano.

**Eficiencia**

Dado que el taller maneja un volumen moderado de pedidos (decenas simultáneamente), el sistema debe **responder sin demoras perceptibles**. Las consultas, filtros y actualizaciones deben ejecutarse en tiempo real para apoyar la toma de decisiones del jefe de taller.

**Confiabilidad**

La aplicación debe garantizar la **consistencia y persistencia de los datos**, incluso ante cierres inesperados. Los pedidos no deben perderse ni corromperse. Se deben implementar **copias de respaldo automáticas** —por ejemplo, cada noche— para asegurar la recuperación ante fallos graves.

**Seguridad**

El sistema debe contar con **mecanismos de autenticación** y control de roles. Solo los supervisores podrán realizar acciones críticas como eliminar pedidos o marcar entregas finales, mientras que los operarios solo podrán modificar estados intermedios. Esto previene accesos no autorizados y mantiene la integridad de la información.

**Mantenibilidad**

El sistema debe ser fácil de modificar y escalar. Agregar nuevos estados de pedido o ajustar el flujo de trabajo no debe requerir reescribir el sistema desde cero. Se recomienda una arquitectura modular y una buena documentación del código para facilitar futuras mejoras.

## LÍMITES

**Alcance del módulo backend**

1. **Sin capa de presentación (frontend)**: El backend expone únicamente los endpoints REST para gestión de pedidos; no incluye interfaces de usuario web o móvil ni componentes UI, delegando esa responsabilidad a un cliente externo.
2. **Sin gestión de inventario y costos**: No contempla inventario de materiales, cálculo de costos de producción o control de stock; el alcance se limita al ciclo de vida de pedidos.
3. **Sin notificaciones automáticas**: No envía correos electrónicos, SMS ni notificaciones push de forma nativa; solo provee endpoints para consultar alertas de vencimiento que deben integrarse con servicios externos.
4. **Historial de auditoría limitado**: El único mecanismo de trazabilidad es la tabla HistorialEstadoPedido; no existe un registro de auditoría extendido para otras operaciones.
5. **Instancia única de base de datos**: Opera con una sola base de datos SQL Server sin soporte integrado de replicación, particionamiento o multi-tenant.

**Restricciones**

* **Uso de JdbcTemplate**: Se implementa con Spring Boot y JdbcTemplate, descartando frameworks ORM (JPA/Hibernate).
* **Almacenamiento de comprobantes**: Los comprobantes de entrega se guardan como texto codificado en la columna contenido de ComprobanteEntrega, sin integración con almacenamiento externo.
* **Dependencia de SQL Server**: La lógica SQL (funciones GETDATE(), DATEADD()) está pensada para SQL Server, por lo que no es compatible con otros motores sin ajustes.
* **Despliegue stateless**: Diseñado para ejecutarse como servicio stateless en contenedores (Docker/Kubernetes), sin depender de archivos locales.

# ANÁLISIS DE REQUERIMIENTOS

## IDENTIFICCIÓN DE ACTORES

| **CÓDIGO** | **NOMBRE** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ACT01** | Jefe de Taller | Usuario con rol “Jefe”. Registra pedidos, asigna operarios, confirma entregas y genera alertas. |
| **ACT02** | Operario | Usuario con rol “Operario”. Actualiza estados intermedios de los pedidos. |
| **ACT03** | Sistema | Actor “automático” que ejecuta la lógica de alertas de vencimiento y expone el endpoint correspondiente. |

## CASOS DE USOS

| **CÓDIGO** | **ACTOR** | **NOMBRE** | **DESCRIPCIÓN** |
| --- | --- | --- | --- |
| **CU01** | Jefe de Taller | Registrar Pedido | Crea un nuevo pedido con código único y datos de cliente, describe el producto y fechas de solicitud/entrega. |
| **CU02** | Jefe de Taller | Asignar Operario a Pedido | Asigna un operario a un pedido existente, cambia estado a “En Producción” y registra la fecha de modificación. |
| **CU03** | Operario | Actualizar Estado de Pedido | Marca un pedido con cualquiera de los estados intermedios (En Proceso, En Espera, Listo para Entregar, etc.) |
| **CU04** | Sistema | Generar Alertas de Vencimiento | Identifica pedidos con fecha comprometida a ≤ 2 días o vencidos y los expone en un endpoint de alertas. |
| **CU05** | Jefe de Taller | Confirmar Entrega y Generar Boleta | Cambia de estado a “Entregado”, registra fecha real y genera comprobante de entrega. |
| **CU06** | Jefe de Taller | Consultar Pedidos | Filtra y lista pedidos por estado, operario, cliente o rango de fechas (endpoint REST consumible). |

## 

## 

# CU01 - CU02 – Registrar y Asignar Pedido

## ESPECIFICACIÓN DEL CASO DE USO

| **Código** | **CU01** | |
| --- | --- | --- |
| **Nombre** | Registrar y asignar un pedido | |
| **Descripción** | Este caso de uso engloba la creación de un nuevo pedido con generación de código único y su asignación a un operario, cambiando el estado inicial a “En Producción” y registrando la trazabilidad en el historial de estados. | |
| **Actores** | **Jefe de taller** | |
| **Precondición** | Usuario autenticado con rol “Jefe”.  Cliente y operario previamente registrados en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | El jefe de taller completa el formulario con: ID de cliente, descripción de producto, tipo de trabajo y fecha de entrega comprometida. |
| **2** | El sistema valida la existencia del cliente; si no existe, lanza ClienteNoEncontradoException. |
| **3** | El sistema genera automáticamente el código del pedido con formato PED-YYYYMM-XXX. |
| **4** | El sistema inserta el registro en Pedido con estado\_id = 1 (Registrado). |
| **5** | El sistema registra un evento en HistorialEstadoPedido indicando la transición a “Registrado”. |
| **6** | El jefe de taller selecciona el operario responsable. |
| **7** | El sistema valida la existencia del operario; si no existe, lanza OperarioNoEncontradoException. |
| **8** | El sistema actualiza operario\_asignado\_id, cambia estado\_id a 2 (“En Producción”), actualiza la fecha de modificación y registra la transición en HistorialEstadoPedido. |
| **Postcondición** | El pedido queda registrado y asignado con estado “En Producción” y su historial de estados actualizado. | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | ClienteNoEncontradoException: “Cliente no encontrado”. |
| **2** | Error de base de datos al insertar el pedido. |
| **3** | Error al insertar evento en historial; operación abortada. |
| **4** | OperarioNoEncontradoException: “Operario no encontrado”. |
| **5** | EstadoYaAsignadoException: si ya estaba en “En Producción”. |
| **Comentarios** | Criterios de aceptación aplicables a este caso de uso: Validaciones Previas, Registro de Pedido y Asignación de Pedido a Empleado. | |

## CRITERIOS DE ACEPTACIÓN

#### 1. Validación de Datos de Entrada

**Descripción:**

El sistema debe verificar que todos los campos obligatorios del DTO PedidoRequestDTO y de AsignacionPedidoDTO estén presentes y cumplan sus restricciones de Bean Validation.

**Criterios:**

* **descripcionProducto y tipoTrabajo** no pueden estar vacíos ni nulos (@NotBlank).
* **fechaSolicitud** debe ser hoy o fecha pasada (@FutureOrPresent).
* **fechaEntregaComprometida** debe ser fecha futura (@Future).
* **estadoId**, **clienteId**, **usuarioCreadorId**, **operarioAsignadoId** (en asignación) no pueden ser nulos (@NotNull).
* En caso de fallo de validación, el endpoint devuelve **400 Bad Request** con un JSON de errores limpió (“field: mensaje”) manejado por GlobalExceptionHandler.

#### 2. Validación de Cliente

**Descripción:**

Antes de persistir el pedido, el sistema confirma que el **clienteId** existe en la tabla Cliente.

**Criterios:**

* Si no existe, lanzar ClienteNoEncontradoException.
* El cliente inexistente produce **404 Not Found** con mensaje: “Cliente no encontrado con ID: X.”

#### 3. Registro de Pedido

**Descripción:**

El sistema genera un código único, inserta el pedido en la base de datos y registra su estado inicial.

**Criterios:**

* El código sigue el patrón PED-YYYYMM-XXX.
* Se inserta un registro en Pedido con:
  + estado\_id = 1 (“Registrado”)
  + fecha\_registro = GETDATE()
  + es\_entregado = 0
* El endpoint retorna **200 OK** (o **201 Created** si así se prefiere) con mensaje: “Pedido registrado correctamente.”

# CU03 – Actualizar Estado de Pedido

## ESPECIFICACIÓN DEL CASO DE USO

| **Código** | **CU03** | |
| --- | --- | --- |
| **Nombre** | **Actualizar Estado de Pedido** | |
| **Descripción** | Permite a un operario cambiar el estado de un pedido a cualquiera de los intermedios (por ejemplo, “En proceso”, “En espera de material”, “Listo para Entregar”), siempre que el nuevo estado sea distinto al actual y su orden lógico sea superior, garantizando trazabilidad y evitando retrocesos. | |
| **Actores** | **Operario** | |
| **Precondición** | * Usuario autenticado con rol “Operario”. * El pedido con ID existe en la base de datos. * El estado actual del pedido no es final. | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | El operario envía PUT /api/pedidos/cambiar-estado con CambioEstadoDTO (pedidoId, nuevoEstadoId). |
| **2** | El sistema valida que el pedido exista; si no, lanza PedidoNoEncontradoException. |
| **3** | Verifica que nuevoEstadoId ≠ estado actual; si coincide, EstadoYaAsignadoException. |
| **4** | Comprueba que el ordenLogico del nuevo estado sea mayor; si no, ReglaNegocioException. |
| **5** | Actualiza estado\_id y fecha\_modificacion en Pedido. |
| **6** | Inserta un registro en HistorialEstadoPedido. |
| **7** | Retorna 200 OK: “Estado actualizado correctamente.” |
| **Postcondición** | Pedido con el nuevo estado y su historial actualizado. | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | PedidoNoEncontradoException (404): “No se encontró el pedido con ID X”. |
| **2** | EstadoYaAsignadoException (400): “El estado ya está aplicado al pedido”. |
| **3** | ReglaNegocioException (400): “El nuevo estado no es válido según el flujo”. |
| **4** | Error de base de datos (500). |
| **Comentarios** | * — * — | |

## 

## CRITERIOS DE ACEPTACIÓN

#### 1. Validación de Datos de Entrada

**Descripción:**

El sistema debe verificar que el DTO CambioEstadoDTO contenga todos los campos obligatorios (pedidoId, nuevoEstadoId, usuarioResponsableId) y que cumplan las anotaciones de Bean Validation.

**Criterios:**

* pedidoId, nuevoEstadoId y usuarioResponsableId no pueden ser nulos (@NotNull).
* Si falla la validación, el endpoint responde **400 Bad Request** con un JSON de errores:

{ "timestamp": "...", "codigo": "VALIDATION\_ERROR", "mensaje": "campo: mensaje" }

#### 2. Validación de Pedido Existente

**Descripción:**

Antes de procesar el cambio de estado, el sistema debe confirmar que existe un registro en la tabla Pedido con el pedidoId recibido.

**Criterios:**

* Si no existe, lanzar PedidoNoEncontradoException.
* La respuesta será **404 Not Found** con cuerpo:

{ "timestamp": "...", "codigo": "NOT\_FOUND", "mensaje": "No se encontró el pedido con ID: X." }

#### 3. Validación de Estado Nuevo

**Descripción:**

El sistema debe comprobar que el nuevoEstadoId recibido es distinto del estado actual y que su ordenLogico es superior, evitando retrocesos o reasignaciones idénticas.

**Criterios:**

* Si nuevoEstadoId == estadoActual, lanzar EstadoYaAsignadoException → **400 Bad Request** con:

“El estado que se intenta asignar ya está aplicado al pedido.”

* Si ordenLogico(nuevo) <= ordenLogico(actual), lanzar ReglaNegocioException → **400 Bad Request** con:

“El nuevo estado no es válido según el flujo lógico.”

#### 4. Validación de Estado Nuevo

**Descripción:**

Una vez validados, el sistema actualiza el registro Pedido estableciendo el nuevo estado\_id y la columna fecha\_modificacion.

**Criterios:**

* Se ejecuta el UPDATE Pedido SET estado\_id = ?, fecha\_modificacion = GETDATE() WHERE id = ? y afecta exactamente 1 fila.

“El estado que se intenta asignar ya está aplicado al pedido.”

* El endpoint devuelve **200 OK** con mensaje:

“Estado actualizado correctamente.”

#### 5. Validación de Estado Nuevo

**Descripción:**

Cada cambio de estado debe quedar trazado en la tabla HistorialEstadoPedido.

**Criterios:**

* Insertar un nuevo registro con los campos pedido\_id, estado\_anterior\_id, estado\_nuevo\_id, usuario\_id y fecha\_cambio = GETDATE().
* La operación de historial debe ser parte de la misma transacción; si falla, se revierte todo el cambio de estado.

#### 6. Validación de Estado Nuevo

**Descripción:**

El backend debe capturar y formatear todas las excepciones para no exponer detalles internos.

**Criterios:**

* Respuestas JSON con campos { timestamp, codigo, mensaje }.
* No exponer stack trace ni excepciones de Java.
* Ejemplos de mensaje:
  + “No se encontró el pedido con ID: X.”
  + “El estado que se intenta asignar ya está aplicado al pedido.”
  + “El nuevo estado no es válido según el flujo lógico.”
  + “Error interno del servidor” (para excepciones genéricas).

# CU04 – Generar Alertas de Vencimiento

## ESPECIFICACIÓN DEL CASO DE USO

# CU05 – Confirmar Entrega y Generar Boleta

## ESPECIFICACIÓN DEL CASO DE USO

# CU06 – Consultar Pedidos

## ESPECIFICACIÓN DEL CASO DE USO

# MODELO DE BASE DE DATOS

## MODELO RELACIONAL

(IMAGEN DE LA DB COMPLETA)

# IMPLEMENTACIÓN DE LA BASE DE DATOS

## CREACIÓN DE LA BASE DE DATOS

# CONCLUSIONES

**Evaluación General del Proyecto**:

* El proyecto logró establecer una base sólida para la gestión escolar a nivel primario, permitiendo optimizar procesos clave como la matrícula, pagos y asignación de horarios.
* La implementación con Spring Boot y SQL Server garantiza escalabilidad y mantenimiento futuro.

**Limitaciones y Futuro Desarrollo**:

* Aunque se ha cumplido con los objetivos principales, el sistema presenta áreas de mejora, como la inclusión de funciones para el registro directo de estudiantes y el manejo avanzado de reportes.
* Se recomienda priorizar la integración de un frontend y la automatización de procesos más complejos en futuras iteraciones.

**Impacto Tecnológico**:

* Este sistema puede servir como base para expandirse a otras instituciones educativas, integrando niveles superiores y funcionalidades adicionales.

**Importancia de la Validación de Datos**:

* La implementación de validaciones y controles asegura la integridad de los datos, lo que es esencial en entornos educativos.

# LECCIONES APRENDIDAS

**Importancia de los Requerimientos Iniciales**:

* La claridad en la recopilación y análisis de requerimientos es fundamental para el éxito del proyecto.

**Impacto de la Documentación**:

* La documentación técnica detallada ayuda a garantizar la mantenibilidad del sistema y facilita futuras mejoras.

**Gestión de Alcances**:

* Definir un alcance realista desde el inicio evita sobrecarga de trabajo y garantiza entregables de calidad.

**Validaciones y Seguridad**:

* Incorporar validaciones a nivel de base de datos y backend refuerza la seguridad y coherencia del sistema.

**Aprendizaje Colaborativo**:

* El proyecto destaca la importancia del trabajo en equipo y la distribución efectiva de tareas.

# BIBLIOGRAFÍA

Microsoft. (2024). *SQL Server Management Studio* (versión 20) [Herramienta de gestión de bases de datos]. Microsoft. <https://learn.microsoft.com/en-us/sql/ssms/>