

**Acta de Constitución – InventPro**

# Indice de Contenidos

[**Indice de Contenidos 1**](#_heading=h.b7sgpp9mrcuj)

[**Información del proyecto 2**](#_heading=h.skiufnpp3t65)

[Datos 2](#_heading=h.r25o6atpe1mp)

[Patrocinadores 2](#_heading=h.hd6avyamat9x)

[Gerente de Proyecto 2](#_heading=h.jxvdey2ejz4)

[Lista de Interesados (stakeholders) 2](#_heading=h.n2th6dnewzv1)

[Cronograma de hitos principales 2](#_heading=h.3dpkimqjn07l)

[Presupuesto estimado 2](#_heading=h.qdqkj9sz2d17)

[**Descripción del proyecto 3**](#_heading=h.m209x3kdm5rf)

[Objetivos del Negocio 3](#_heading=h.l5rrazckn9rq)

[Justificación del proyecto – Contexto 3](#_heading=h.vpzf5ic56esf)

[Problema-Necesidad 3](#_heading=h.sl01m0x1citu)

[**Descripción del producto 3**](#_heading=h.q6gy5cqw0y08)

[Solución Propuesta 3](#_heading=h.48uxv98x70v5)

[Objetivos del proyecto 3](#_heading=h.sji7tkag782n)

[Objetivos de desarrollo 4](#_heading=h.oobphkvyeqqf)

[Entregables 4](#_heading=h.7x8x8xzhlezq)

[**Descripción del sistema 4**](#_heading=h.w80nmhr70eex)

[Requerimientos de alto nivel 4](#_heading=h.2dx0jlbb39s4)

[Premisas y restricciones 4](#_heading=h.9ckzmmxbhslt)

[Riesgos iniciales de alto nivel 5](#_heading=h.61po96dk1ktq)

[Especificaciones técnicas de las herramientas de desarrollo 5](#_heading=h.yf1n04pi0n6k)

[Tipo de Interfaz de Hardware 5](#_heading=h.em6v4dz902m9)

[Tipo de Interfaz de Software 5](#_heading=h.5nm4ai2rq4zx)

[Tipo de Interfaz de Usuario 5](#_heading=h.hhc4aed2dumv)

[**Requisitos de aprobación del proyecto 5**](#_heading=h.5g1eotqnxg94)

[**Aprobaciones y control de cambios 5**](#_heading=h.r3fe4gtmpss6)

# 

# 

# Información del proyecto

## Datos

|  | Empresa / Organización | TechPyME Solutions |
| --- | --- | --- |
| Nombre del Proyecto | InventPro. |
| Fecha de inicio/fin | 28/08/2025 |
| Cliente | PYMEs del sector retail/servicios |
| Patrocinador principal |  |
| Jefe de Proyecto | Javier Hermosilla – Líder Técnico / Backend Developer |

## 

## 

## Patrocinadores

| **Nombre** | **Cargo** | **Departamento / División** |
| --- | --- | --- |
| María Elena Rojas | Directora Ejecutiva | TechPyME Solutions / Dirección |
| Fernando Silva | Gerente de Innovación y Proyectos | TechPyME Solutions / Innovación |
| Catalina Muñoz | Jefa de Finanzas | TechPyME Solutions / Administración |
| Lorena Barrientos | Representante Legal | TechPyME Solutions / Asesoría Legal |
| Patricio Ortega | Mentor Técnico / Sponsor Técnico | TechPyME Solutions / Desarrollo TI |

## Gerente de Proyecto

| **Nombre** | **Cargo** | **Departamento / División** |
| --- | --- | --- |
| Javier Hermosilla | Líder Técnico / Jefe de Proyecto | Área de Desarrollo |

## 

## 

## Lista de Interesados (stakeholders)

| **Nombre** | **Tipo** | **Cargo** | **Departamento / División** |
| --- | --- | --- | --- |
| Franco Borotto | Equipo de Proyecto | Desarrollador / Documentación | Área de Desarrollo |
| Claudio Soto | Equipo de Proyecto | Desarrollador / Soporte | Área de Desarrollo |
| Javier Hermosilla | Equipo de Proyecto / Jefe | Backend Developer | |  | | --- |  | Área de Desarrollo | | --- | |
| Roberto Sanhueza | Cliente | Gerente | Comercial XYZ Ltda. Rubro ferretería |

## 

## Cronograma de hitos principales

| **Hito** | **Fecha tope** |
| --- | --- |
| H0 – Inicio / Acta de Constitución | 28/08/2025 |
| H1 – Arquitectura y Modelo de Datos | 12/09/2025 |
| H2 – Núcleo Backend | 30/09/2025 |
| H3 – Inventario Manual + Reportes | 15/10/2025 |
| H4 – Seguridad + Pruebas integración | 31/10/2025 |
| H5 – MVP en Demo | 30/11/2025 |

## 

## Presupuesto estimado

| **Esfuerzo en HH (MVP):** ~400 HH   * Backend/API: 160 HH * QA y pruebas: 60 HH * Documentación: 40 HH * Gestión: 40 HH * Reserva de riesgos: 100 HH   **Costo HH (estimación en CLP/USD):**   * Supongamos un valor de referencia de **CLP $10.000 / hora** (≈ USD 12). * Total estimado: **400 HH x CLP $10.000 = CLP $4.000.000 (≈ USD 4.800)**.   **Infraestructura (demo/nube):**   * PostgreSQL gestionado o VM: **USD 25–60/mes** (≈ CLP $23.000–55.000). * Monitoreo opcional: **USD 20/mes** (≈ CLP $18.000).   **Total estimado MVP (3 meses):**   * Esfuerzo: ≈ **USD 4.800 (CLP $4 millones)** * Infraestructura: ≈ **USD 150 (CLP $140.000)** * **Gran total:** ≈ **USD 5.000 (CLP $4,1 millones)** |
| --- |

# 

# 

# Descripción del proyecto

## Objetivos del Negocio

| El proyecto busca implementar una plataforma integral de gestión de inventarios y órdenes de compra orientada a PYMEs, con el fin de centralizar la administración de productos, categorías, proveedores, clientes, órdenes, inventario manual, reportes y ajustes de stock. La solución estará enfocada en entregar a las pequeñas y medianas empresas una herramienta accesible, segura y escalable, que les permita reducir errores en el control de inventario, optimizar procesos de compra/venta y disponer de información confiable en tiempo real para la toma de decisiones. |
| --- |

## 

## 

## 

## Justificación del proyecto – Contexto

| Actualmente, muchas PYMEs enfrentan problemas al manejar sus inventarios y compras utilizando métodos manuales (planillas, registros físicos o software poco especializado). Esto genera ineficiencias, errores humanos y pérdidas económicas.  De hecho, según el **Ministerio de Economía de Chile (Informe Anual 2023)**, más del 37 % de las quiebras registradas corresponden a PYMEs, lo que evidencia el impacto que tienen estas deficiencias en su sostenibilidad y competitividad.  El proyecto surge como respuesta a la necesidad de contar con una herramienta digital, segura y centralizada que permita a las PYMEs profesionalizar su gestión, reduciendo riesgos de pérdidas y aumentando su competitividad frente a empresas más grandes. |
| --- |

## Problema-Necesidad

| * Falta de herramientas accesibles y adaptadas a las necesidades de las PYMEs. * Errores frecuentes en el registro manual de inventario. * Ausencia de control automatizado sobre órdenes de compra y movimientos de stock. * Dificultad para generar reportes e indicadores confiables en tiempo real. |
| --- |

# Descripción del producto

## Solución Propuesta

| Se desarrollará una plataforma web modular orientada a la gestión de inventario y órdenes de compra para PYMEs. El sistema contemplará autenticación segura de usuarios, administración centralizada de productos, proveedores y clientes, y la generación de reportes dinámicos que faciliten la toma de decisiones.  La solución se implementará utilizando un backend en Node.js con Express, empleando Sequelize como ORM y PostgreSQL como motor de base de datos relacional. Esta arquitectura asegura integridad de datos, escalabilidad y facilidad de mantenimiento.  El frontend administrativo se considera en una fase siguiente, y estará diseñado para consumir los servicios de la API, permitiendo la interacción visual con los módulos principales. |
| --- |

## Objetivos del proyecto

| **Objetivo** | **Indicador de éxito** |
| --- | --- |
| **Alcance** | |
| Entregar una primera versión de la API (MVP) con los módulos principales funcionando. | Todos los endpoints definidos en Swagger estarán implementados y funcionando correctamente. Además, las pruebas de rendimiento deben mostrar que la mayoría de las consultas (95%) responden en menos de 300 milisegundos en el ambiente de demo. |

# 

| **Objetivo** | **Indicador de éxito** |
| --- | --- |
| **Calidad** | |
| Asegurar que la aplicación sea confiable y segura desde el inicio. | No se deben detectar vulnerabilidades graves o críticas en las pruebas de seguridad, y cada entrada de datos pasará por validaciones automáticas con Zod para evitar errores o abusos. |
|  |  |

| **Objetivo** | **Indicador de éxito** |
| --- | --- |
| **Cronograma** | |
| Cumplir con las fases planificadas del proyecto (H0 a H5) sin grandes atrasos. | Cada hito se entregará dentro de un margen de una semana respecto a la fecha acordada, dejando registro en actas de avance. |
|  |  |
|  |  |
| **Tiempos de Desarrollo** |  |
| Organizar el trabajo de forma ordenada y predecible. | Se trabajará en sprints de dos semanas, con reuniones de revisión y minutas de seguimiento. Esto estará sincronizado con la Carta Gantt y la EDT para asegurar que el avance real coincida con lo planificado. |

| **Objetivo** | **Indicador de éxito** | |
| --- | --- | --- |
| **Costos** | | |
| Mantener el esfuerzo de desarrollo dentro del presupuesto y las horas estimadas. | | La diferencia entre lo planificado y lo realmente invertido en horas de trabajo no debe superar el 15%. |

## 

## Objetivos de desarrollo

| * **Diseño modular y desacoplado:** Implementar la aplicación bajo un esquema por capas claramente diferenciadas, con una hoja de ruta hacia arquitecturas más avanzadas como la hexagonal. * **Consistencia en operaciones críticas:** Asegurar que las transacciones de inventario y órdenes se ejecuten bajo principios **ACID**, garantizando integridad y confiabilidad en los datos. * **Pruebas progresivas:** Desarrollar pruebas unitarias e integrar pruebas funcionales de manera creciente, priorizando la cobertura de las áreas críticas del sistema. * **Documentación continua:** Mantener documentación viva y actualizada (Swagger, ERS, Casos de Uso, Mockups) como soporte a la trazabilidad y la gestión del conocimiento. |
| --- |

## 

## Entregables

| * **E1. Acta de Constitución**  Documento que formaliza el inicio del proyecto, su alcance, objetivos y responsabilidades. * **E2. ERS – Especificación de Requisitos Software**  Documento formal de requisitos funcionales y no funcionales (formato IEEE 830). * **E3. Casos de Uso (alto nivel y extendidos)**  Descripción de los escenarios de interacción entre usuarios y sistema. * **E4. Mockups / Prototipado IU**  Prototipos visuales de las pantallas principales del sistema. * **E5. Matriz EDT (WBS) y Planilla de Requerimientos**  Estructura de desglose de trabajo y trazabilidad de requisitos. * **E6. Carta Gantt**  Cronograma visual con las fases, tareas e hitos principales. * **E7. Swagger API (OpenAPI)**  Documentación navegable de los endpoints de la API. * **E8. Modelo de datos (DER y migraciones)**  Diagrama entidad–relación y scripts de base de datos. * **E9. Minutas y Documentos de avance de sprint**  Evidencias de seguimiento, acuerdos y control de cambios. * **E10. Informe Técnico del MVP**  Documento final con resultados, conclusiones y lecciones aprendidas. |
| --- |

# Descripción del sistema

## Requerimientos de alto nivel

| **Gestión de usuarios y roles**   * El sistema debe permitir autenticación segura de usuarios. * Roles principales: Administrador, Bodeguero y Vendedor. * Funciones y permisos diferenciados por rol.   **Gestión de inventario**   * Registro, actualización y consulta de productos. * Control de stock en tiempo real con alertas automáticas (stock mínimo/crítico). * Ajustes de inventario con trazabilidad (historial de cambios y responsables).   **Gestión de órdenes de compra y ventas**   * Creación, seguimiento y cierre de órdenes. * Impacto automático en el inventario al procesar una orden. * Generación de comprobantes (PDF / XLS) de órdenes.   **Gestión de proveedores y clientes**   * Registro y consulta de datos de proveedores. * Registro y consulta de datos de clientes. * Asociación de órdenes a proveedores/clientes.   **Reportes dinámicos**   * Reportes de inventario, ventas y compras. * Exportación a PDF y Excel. * Visualización de métricas en panel administrativo.   **Aplicación secundaria (móvil de soporte)**   * Notificaciones automáticas para administradores y encargados de bodega. * Soporte para alertas críticas (ej. stock bajo, órdenes pendientes). * Funcionalidad básica de consulta de inventario.   **Seguridad y sesiones**   * Autenticación con JWT y manejo seguro de contraseñas. * Expiración de sesión con reautenticación. * Acceso restringido según permisos.   **Accesibilidad y usabilidad**   * Interfaz web responsiva (desktop, tablet, móvil). * Navegación optimizada para tareas rápidas en bodega. * Cumplimiento básico de estándares WCAG.   **Infraestructura y base tecnológica**   * **Backend**: desarrollado en Node.js con Express y Sequelize, desplegado en la nube mediante Railway. * **Base de datos**: centralizada en PostgreSQL gestionado en Railway. * **Frontend**: desarrollado en React con Tailwind CSS, desplegado en Vercel como aplicación web responsiva y PWA para bodeguero. * **Arquitectura**: escalable bajo un modelo en capas, con opción de evolución hacia microservicios en futuras versiones. |
| --- |

## 

## 

## Premisas y restricciones

| * El desarrollo se realizará con tecnologías web estándar: Node.js (ES Modules) en backend, Express, Sequelize como ORM, PostgreSQL como base de datos y React + Tailwind en frontend. * La aplicación secundaria será un mini-frontend móvil (PWA o React Native según decisión) orientado a notificaciones y mantenciones rápidas; no se desarrollará una aplicación de escritorio en esta fase. * El proyecto se ejecuta en un entorno académico/piloto; no se prevé despliegue productivo ni SLA de disponibilidad empresarial en esta fase. * No se contemplan integraciones con sistemas ERP, pasarelas de pago o sistemas contables externos en la versión inicial (quedan como mejoras futuras). * La base de datos será PostgreSQL; inicialmente se trabajará con instancias locales o de laboratorio (no con bases de datos productivas). * Para pruebas de integración y UAT se dispondrá de un entorno de staging separado; los datos de producción no se usarán en pruebas. * Se asume acceso a internet para sincronización, notificaciones y acceso a repositorios; la funcionalidad offline será limitada (caching y operaciones degradadas en PWA), sin soporte completo offline para escrituras críticas. * El mini-frontend móvil podrá usar capacidades offline limitadas (IndexedDB / cache) para consultas rápidas; las operaciones que modifiquen stock deberán sincronizarse con el servidor y serán tratadas como operaciones críticas. * Los entregables priorizados son un MVP (flujos críticos: login, CRUD productos, crear orden, ajuste de stock, notificaciones y reportes exportables); funcionalidades avanzadas quedan fuera del alcance inicial. * Se asumirá disponibilidad parcial del equipo (3 integrantes) para desarrollo, pruebas y documentación; tareas cronológicas y horas serán estimadas en el plan de trabajo. * El proyecto queda restringido por tiempo y recursos: no se implementarán integraciones hardware complejas (impresoras fiscales, balanzas industriales) en la fase MVP. * Las dependencias y librerías deberán fijarse a versiones compatibles; cambios drásticos de versión durante desarrollo pueden generar re-planificación. * Documentación mínima requerida: README de instalación, variables .env, scripts de base de datos y documentación Swagger de la API; la ausencia de documentación completa limitará aceptación. * Pruebas: se exige cobertura mínima en lógica crítica (tests unitarios e integración); pruebas E2E son recomendadas pero pueden quedar como alcance opcional según tiempo. * Seguridad en el transporte: las conexiones desde y hacia entornos remotos deben usar TLS/SSL en despliegues reales; para desarrollo local esto podrá omitirse temporalmente. * Soporte de navegadores: se garantizará compatibilidad con navegadores modernos (Chrome, Edge, Firefox); navegadores legacy no serán prioridad. * Requisitos de hardware mínimo para desarrollo/pruebas: máquina con CPU moderna, 4GB+ RAM, Node.js y PostgreSQL instalados; para pruebas móviles se utilizarán emuladores o dispositivos reales con capacidades razonables. * Responsabilidad de backups: durante el desarrollo el equipo mantendrá copias de seguridad periódicas; no se garantiza recuperación ante pérdida en entornos del cliente sin acuerdos previos. * Restricción legal y de privacidad: todo uso de datos reales deberá cumplir la normativa aplicable; por defecto se usarán datos de prueba o anonimización para demos y UAT. |
| --- |

## 

## 

## 

## 

## 

## 

## 

## 

## 

## Riesgos iniciales de alto nivel

| ***Fallas en la sincronización entre backend y frontend por modelos mal definidos***  **Impacto:** datos inconsistentes, errores en UI y retracto en pruebas de integración.  **Mitigación:** definir contratos de API (Swagger/OpenAPI) y realizar pruebas de integración tempranas.  ***Retrasos por conflictos en dependencias o actualizaciones de librerías***  **Impacto:** bloqueo de tareas, tiempo extra en resolver incompatibilidades.  **Mitigación:** fijar versiones estables en package.json, crear un entorno de staging y revisar actualizaciones en ventanas planificadas.  ***Vulnerabilidades de seguridad (JWT mal gestionado, inyección SQL, XSS, etc.)***  **Impacto:** brechas de datos, pérdida de confianza y penalizaciones académicas/profesionales.  **Mitigación:** aplicar buenas prácticas de seguridad desde el diseño, auditorías básicas de seguridad y validaciones server/client.  ***Subestimación del tiempo requerido para pruebas e integración continua***  **Impacto:** entrega incompleta o con baja calidad; re-trabajo inevitable.  **Mitigación:** reservar un bloque de sprints para testing, automatizar pruebas unitarias e integración y planificar UAT.  ***Alcance incontrolado (scope creep)***  **Impacto:** aumento de tiempo y esfuerzo, desviación de objetivos del MVP.  **Mitigación:** definir y aprobar un backlog priorizado; aplicar control de cambios para peticiones adicionales.  ***Pérdida o corrupción de datos por errores en migraciones o scripts de BD***  **Impacto:** pérdida de trabajo y necesidad de restauraciones; riesgo en demos.  **Mitigación:** usar migraciones versionadas, respaldos regulares y ensayos de restauración en entorno de pruebas.  ***Baja disponibilidad o compromiso de los stakeholders.***  **Impacto:**retrasos en validaciones, ambigüedad en requisitos y demoras en aprobación.  **Mitigación:** calendarizar reuniones de revisión periódicas y definir puntos de decisión concretos.  **Requerimientos incompletos o mal definidos**  **Impacto:** desarrollo de funcionalidades que no satisfacen la necesidad real.  **Mitigación:** sesiones tempranas de refinamiento con prototipos y validación frecuente con el cliente.  ***Problemas de rendimiento y escalabilidad no detectados en pruebas iniciales***  **Impacto:** respuesta lenta en consultas/reportes, mala experiencia en demos.  **Mitigación:** pruebas de carga básicas en etapas intermedias y optimización de consultas críticas (índices, paginación).  ***Falta de documentación y transferencia de conocimiento***  **Impacto:** dificultad para mantener o continuar el proyecto; dependencia de individuos.  **Mitigación:** documentar decisiones arquitectónicas, endpoints principales y pasos de despliegue; registrar sesiones de handover.  ***Resistencia a la adopción por parte de usuarios finales (cambio de procesos)***  **Impacto:** el sistema no es usado o se usa parcialmente, disminuyendo el valor entregado.  **Mitigación:** diseñar workflows intuitivos, preparar material de capacitación y ejecutar pruebas de usuario tempranas.  ***Problemas de cumplimiento legal o de privacidad al usar datos reales***  **Impacto:** sanciones, rechazo de la entrega o necesidad de eliminar datos.  **Mitigación:** usar datos de prueba o anonimización en demos; validar requisitos legales antes de usar información real. |
| --- |

## Especificaciones técnicas de las herramientas de desarrollo

| **Backend:**   * **Lenguaje y entorno:** Node.js (ES Modules). * **Framework principal:** Express.js para la construcción de la API REST. * **ORM y base de datos:** Sequelize como ORM y **PostgreSQL** como motor de base de datos relacional. * **Gestión de seguridad:** JWT para autenticación, bcrypt para encriptación de contraseñas, y middlewares de validación (Zod).   **Frontend (planificado para fases posteriores):**   * **Framework:** React.js (con herramientas de inicialización como *Create React App* o equivalentes). * **Estilos y diseño responsivo:** Tailwind CSS como framework principal; posibilidad de uso complementario de Bootstrap en casos puntuales. * **Compatibilidad:** Navegadores modernos (Chrome, Edge, Firefox).   **Control de versiones:**   * **Sistema de control:** Git. * **Repositorio y colaboración:** GitHub para gestión de ramas, issues y control de integraciones.   **Pruebas:**   * **Backend:** Jest para pruebas unitarias y de integración, apoyado en **Supertest** para pruebas de endpoints. * **Frontend (fase posterior):** Librerías de pruebas de componentes como React Testing Library   **Documentación:**   * **APIs:** Swagger (OpenAPI) para la documentación técnica y navegable de los endpoints. * **Guías y manuales:** Documentos en formato **Markdown** para instalación, configuración y guías de uso/despliegue. * **Control de versiones de documentación:** Integrada en el mismo repositorio del proyecto para asegurar consistencia. |
| --- |

## Tipo de Interfaz de Hardware

| * **API REST sobre HTTP:**  El sistema ofrece un “motor” central al que se conectan todas las funciones. Este motor responde en **formato JSON**, lo que hace sencillo intercambiar datos con distintas aplicaciones. En esta primera fase funcionará sobre **HTTP**, suficiente para pruebas y desarrollo. Cuando el sistema pase a producción, se usará **HTTPS** para asegurar las comunicaciones. * **Panel web (React, futuro):**  Se desarrollará un **panel administrativo** con React y Tailwind, que permitirá a los usuarios trabajar desde el navegador. Allí podrán gestionar inventarios, órdenes, reportes y usuarios de forma visual y sencilla. * **Base de datos (PostgreSQL):**  Toda la información —productos, clientes, proveedores, órdenes e inventario— se guardará en **PostgreSQL**, una base de datos robusta que asegura que los datos siempre estén correctos y consistentes. * **Contenedores Docker:**  Tanto el backend como la base de datos podrán “empaquetarse” en **contenedores Docker**, lo que hace que sea más fácil instalar el sistema, moverlo entre servidores y replicarlo en ambientes de pruebas o producción. * **Servicios de correo (a futuro):**  Más adelante, el sistema podrá conectarse con servicios de correo electrónico para enviar notificaciones automáticas, como confirmaciones de órdenes, alertas de stock bajo o reportes periódicos. |
| --- |

## Tipo de Interfaz de Software

## 

| * **API REST sobre HTTPS:**  Toda la lógica del sistema se expone a través de una API segura. Esto significa que cualquier cliente autorizado puede comunicarse con el servidor mediante peticiones estándar (GET, POST, PUT, DELETE) y recibir respuestas en formato JSON. El uso de HTTPS asegura que los datos viajen cifrados. * **Cliente web (React en fase posterior):**  Está previsto un frontend administrativo desarrollado en React + Tailwind que se conectará directamente a la API. Desde allí, los usuarios podrán manejar productos, órdenes, inventarios y reportes de forma visual e intuitiva. * **Base de datos PostgreSQL:**  El sistema guarda y organiza la información en PostgreSQL, una base de datos relacional robusta que asegura integridad referencial, control transaccional (por ejemplo, en el descuento automático de stock) y consultas rápidas. * **Contenedores Docker:**  Todo el backend y la base de datos pueden empaquetarse en contenedores Docker, lo que facilita la instalación, despliegue y replicación en distintos ambientes (desarrollo, pruebas, producción) sin diferencias en la configuración. * **Integración futura con servicios de correo:**  El plan es habilitar notificaciones automáticas vía email (ejemplo: confirmación de órdenes, alertas de stock bajo, informes programados). Esto se logrará conectando la API con servicios externos de correo (SMTP propio o proveedores como SendGrid). |
| --- |

## 

## Tipo de Interfaz de Usuario

| Se implementará un panel web con React como framework principal y Tailwind CSS para los estilos y diseño responsivo. La aplicación contemplará roles diferenciados (Administrador, Bodeguero, Vendedor) con navegación por módulos y accesos restringidos según permisos.  Se priorizará una experiencia de usuario clara, con tablas de productos y órdenes que incluyan búsqueda, filtros y paginación, así como formularios validados de manera automática.  El frontend será probado con herramientas de testing y se diseñará bajo principios de accesibilidad básica y compatibilidad con navegadores modernos. |
| --- |

# Requisitos de aprobación del proyecto

| * **H0 – Inició:** Aprobación del Acta de Constitución y validación del plan inicial. * **H1 – Arquitectura y Modelo de Datos:** Validación del modelo de datos y aprobación del Swagger base como contrato de API. * **H2 a H4 – QA de módulos núcleo:** Aprobación tras superar las pruebas de calidad en los módulos principales (Usuarios, Productos, Órdenes, Inventario, Reportes). * **H5 – MVP en Demo:** Aprobación final con la entrega de la versión mínima viable en ambiente de demostración, acompañada de documentación técnica y minutas de cierre. |
| --- |

# Aprobaciones y control de cambios

| Versión | Nombre | Rol | Fecha | Firma |
| --- | --- | --- | --- | --- |
| 1.0 | Javier Hermosilla | Gerente de proyecto/ Líder técnico | 28/08/2025 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |