**Ficha del documento**

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Revisión** | **Autor** | **Modificación** |
|  |  |  |  |
|  |  |  |  |

Documento validado por las partes en fecha:

**Integrantes:**

|  |  |
| --- | --- |
| Simón Mena - Desarrollador Backend Juan Pablo Báez - Arquitecto de Software | **Rol Definido** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Evaluacion 1:**

**1. Introducción**

Este documento tiene como propósito definir los requerimientos funcionales y no funcionales del nuevo sistema de Perfulandia SPA. Está dirigido a desarrolladores, testers, clientes, y partes interesadas, y servirá como base para el desarrollo del sistema basado en microservicios..

**1.1. Propósito**

**El propósito de este informe es definir los requisitos del sistema y el plan para su desarrollo, dirigido a las partes interesadas y el equipo de desarrollo.**

**1.2. Ámbito del Sistema**

**El sistema a desarrollar, llamado "Perfulandia Digital", reemplazará al actual sistema monolítico de la empresa. Permitirá gestionar usuarios, ventas, inventario, logística, atención al cliente, reportes y más, mediante microservicios independientes. No incluirá servicios de atención presencial o gestión financiera interna.**

**1.3. Definiciones, Acrónimos y Abreviaturas**

**SPA: Sociedad por Acciones**

**ERS: Especificación de Requisitos de Software**

**API: Interfaz de Programación de Aplicaciones**

**JWT: JSON Web Token**

**OAuth2: Protocolo de Autenticación**

**REST: Estilo de arquitectura para servicios web**

**1.4. Referencias**

Presentación "Ingeniería de Software – Caso Perfulandia"

Material docente de DUOC UC

**1.5. Visión General del Documento**

Este documento presenta una descripción general del sistema, sus requerimientos funcionales y no funcionales, planificación, estructura del equipo, estimación de costos, y diagramas relacionados.

**2. Descripción General**

En esta sección se describen todos aquellos factores que afectan al producto y a sus requisitos. No se describen los requisitos, sino su contexto. Esto permitirá definir con detalle los requisitos en la sección 3, haciendo que sean más fáciles de entender.

Normalmente, esta sección consta de las siguientes subsecciones: Perspectiva del producto, funciones del producto, características de los usuarios, restricciones, factores que se asumen y futuros requisitos.

**2.1. Perspectiva del Producto**

El sistema formará parte del ecosistema digital de Perfulandia, conectado a servicios de facturación electrónica y seguimiento de pedidos. Estará compuesto por microservicios desplegados en la nube con base de datos centralizada en MySQL.

**2.2. Funciones del Producto**

Registro y autenticación de usuarios

Gestión de productos, stock y pedidos

Procesamiento de ventas y facturas

Gestión de logística y envíos

Portal web de clientes

Panel administrativo

Notificaciones por email

Generación de reportes

**2.3. Características de los Usuarios**

Clientes: Usuarios generales sin conocimientos técnicos

Empleados de ventas: Personal con conocimientos básicos de sistemas

Gerentes: Usuarios avanzados que gestionan sucursales

Logística: Encargados de actualizar estados de envío

Administrador del sistema: Técnicos encargados del mantenimiento

**2.4. Restricciones**

Cumplimiento con la Ley de Protección de Datos Personales

Comunicación vía API REST segura (HTTPS)

Desarrollo bajo buenas prácticas de seguridad

Uso de tecnologías como Spring Boot, MySQL, Postman y Docker

**2.5. Suposiciones y Dependencias**

Se supone que el sistema actual será accesible durante la fase de migración

Se asume que los usuarios contarán con dispositivos con conexión a internet

La infraestructura en la nube estará disponible y operativa

**2.6. Requisitos Futuros**

Mejorar la adaptabilidad en el software y mejor escabilidad

**3. Requisitos Específicos**

Esta sección contiene los requisitos del sistema con un nivel de detalle suficiente para permitir su diseño, implementación y prueba. Cada requisito describe comportamientos observables por parte de los usuarios o sistemas externos, y está redactado de forma clara, verificable y trazable. Se identifican mediante un código único (RF para Requisitos Funcionales y RNF para Requisitos No Funcionales).

A continuación, se detallan los principios aplicados en esta especificación:

Corrección: Cada requisito responde a una necesidad real detectada en el análisis del sistema actual.

No ambigüedad: Los requisitos se expresan de forma precisa, con lenguaje claro y sin interpretaciones múltiples.

Completitud: Se han incluido tanto funcionalidades principales como casos alternativos (errores, validaciones, etc.).

Consistencia: No hay requisitos que se contradigan entre sí.

Clasificación: Los requisitos se clasifican por tipo (funcionales / no funcionales) y se enumeran.

Verificabilidad: Todos los requisitos permiten ser evaluados mediante pruebas.

Modificabilidad: La estructura modular permite actualizar o modificar requisitos fácilmente.

Trazabilidad: Cada requisito está numerado y documentado, lo que permite asociarlos con funcionalidades específicas del sistema.

**3.1 Requisitos comunes de las interfaces**

**En esta sección se describen las interfaces necesarias para la interacción entre los usuarios, el sistema, otros softwares, hardware involucrado y medios de comunicación. Estas interfaces son esenciales para garantizar una correcta integración, experiencia de usuario y funcionalidad del sistema.**

**3.1.1 Interfaces de usuario**

**El sistema contará con dos interfaces principales:**

**Portal Web de Clientes:**

**Acceso mediante navegador web.**

**Registro, inicio de sesión y recuperación de contraseña.**

**Visualización de catálogo de productos, carrito de compras, historial de pedidos y estado de envíos.**

**Formulario de reseñas, reclamos y devoluciones.**

**Compatible con dispositivos móviles (diseño responsivo).**

**Panel Administrativo:**

**Accesible para usuarios con roles: Administrador, Gerente, Logística y Empleado de Ventas.**

**Gestión de usuarios, productos, pedidos, reportes e inventario.**

**Notificaciones internas y control de roles de acceso.**

**Gráficos de rendimiento y reportes descargables (PDF, CSV).**

**3.1.2 Interfaces de hardware**

**El sistema se alojará en un entorno de servidor en la nube.**

**Se requiere acceso a internet desde dispositivos cliente (PC, notebook, tablet o smartphone).**

**No se requiere hardware especializado adicional para operar el sistema desde el lado del cliente.**

**En el servidor se utilizarán contenedores Docker que se ejecutan en un entorno compatible con Linux.**

**3.1.3 Interfaces de software**

**El sistema se integrará con los siguientes productos o servicios de software:**

**MySQL: Base de datos relacional utilizada por todos los microservicios para el almacenamiento persistente de información (usuarios, productos, pedidos, etc.).**

**API de Facturación Electrónica (SII): Para generar boletas/facturas en cumplimiento con la normativa chilena.**

**SMTP/Email Service (ej. SendGrid o Gmail API): Para el envío de notificaciones automáticas de estado de pedido, recuperación de contraseña y confirmación de registro.**

**Spring Boot Framework: Utilizado para el desarrollo de los microservicios REST.**

**Postman: Herramienta para pruebas de las APIs desarrolladas.**

**3.1.4 Interfaces de comunicación**

**API REST: Todos los microservicios se comunican entre sí mediante servicios RESTful utilizando protocolo HTTP/HTTPS.**

**API Gateway: Punto de entrada para todas las solicitudes externas. Redirige a los microservicios correspondientes y maneja la autenticación.**

**OAuth2 + JWT: Sistema de autenticación y autorización mediante tokens firmados que contienen datos del usuario y su rol.**

**Formato de datos: JSON será el formato estándar para la entrada y salida de datos entre los servicios y la interfaz de usuario.**

**3.2 Requisitos funcionales**

**1.El sistema debe permitir crear, editar y eliminar cuentas de usuarios.**

**2.Los empleados deben poder registrar ventas y generar facturas.**

**3.Los gerentes deben gestionar el inventario y pedidos por sucursal.**

**4.Los clientes deben poder registrarse, iniciar sesión y comprar en línea.**

**5.El sistema debe permitir aplicar descuentos y cupones en compras.**

**6.Debe enviarse notificación por email con el estado de los pedidos.**

**7.El sistema debe procesar devoluciones y reclamos.**

**8.Logística debe poder actualizar el estado de los envíos.**

**9.El sistema debe generar reportes de ventas e inventario.**

**10.Se debe permitir dejar reseñas y calificaciones de productos.**

**3.3 Requisitos no funcionales**

**1.El sistema debe estar disponible al menos el 99% del tiempo.**

**2.El sistema debe estar desarrollado bajo buenas prácticas de seguridad.**

**3.Debe ser escalable para soportar nuevas sucursales o módulos.**

**4.La base de datos debe estar protegida y respaldada periódicamente.**

**5.El sistema debe cumplir con la Ley de Protección de Datos Personales.**

**6.Los servicios deben comunicarse vía API REST segura (HTTPS).**

**(es una interfaz de programación de aplicaciones (API) que se ajusta a los principios de diseño del estilo arquitectónico de transferencia de estado representacional (REST).**

**7.El sistema debe ser mantenible con código modular y documentado.**

**8.La interfaz de cliente debe ser intuitiva y responsiva.**

**3.3.1 Requisitos de rendimiento**

**El sistema debe procesar al menos 50 transacciones por segundo en condiciones normales y escalar hasta 200 tps en campañas masivas.**

**El 95% de las solicitudes deben responder en menos de 1 segundo, y el 100% en un máximo de 3 segundos.**

**Debe soportar un mínimo de 300 usuarios simultáneos sin pérdida de rendimiento.**

**La base de datos debe manejar 100 consultas concurrentes sin errores.**

**La sincronización entre microservicios no debe superar los 500 ms de desfase.**

**3.3.2 Seguridad**

**El sistema debe implementar OAuth2 con JWT para autenticación y control de acceso basado en roles.**

**Toda comunicación entre cliente y servidor debe estar protegida mediante HTTPS (TLS 1.2 o superior).**

**Las contraseñas y datos sensibles se almacenarán cifrados usando AES-256.**

**Se deberán registrar los accesos y acciones relevantes en logs de auditoría.**

**Solo los módulos autorizados podrán comunicarse entre sí mediante validación de tokens.**

**Se aplicarán mecanismos de verificación de integridad para evitar manipulación de datos críticos.**

**3.3.3 Fiabilidad**

**El sistema debe garantizar una tolerancia a fallos mediante la replicación de datos en clúster MySQL.**

**En caso de caída de un microservicio, los demás deben continuar operando sin interrupciones.**

**El tiempo medio entre fallos (MTBF) debe ser de al menos 30 días.**

**El sistema debe contar con mecanismos de autorreintento para operaciones críticas fallidas.**

**Todos los errores deben ser registrados y notificados automáticamente al equipo de soporte técnico.**

**3.3.4 Disponibilidad**

**El sistema debe estar disponible al menos el 99% del tiempo mensual, excluyendo mantenimientos programados.**

**Los servicios críticos (login, ventas, pagos) deben contar con monitorización en tiempo real.**

**Se deben realizar copias de seguridad automáticas diarias de la base de datos.**

**En caso de fallo, el sistema debe recuperarse en un tiempo máximo de 15 minutos (RTO).**

**El tiempo máximo de pérdida aceptable de datos (RPO) es de 5 minutos.**

**3.3.5 Mantenibilidad**

**El sistema debe ser mantenible con código modular y documentado. La arquitectura de microservicios con bajo acoplamiento y alta cohesión facilita el mantenimiento y la evolución del sistema.**

**3.3.6 Portabilidad**

**Dada la arquitectura de microservicios y el uso de tecnologías como Docker, el sistema de Perfumelandia SPA tendrá una alta portabilidad. Esto significa que:**

**Independencia de la plataforma: Los microservicios empaquetados en contenedores Docker pueden ejecutarse en una variedad de entornos, ya sean entornos de desarrollo locales, servidores en la nube (como AWS, Azure, Google Cloud) o infraestructura local, con mínimas modificaciones. Esto se evidencia en el diagrama de despliegue que muestra el uso de un clúster ECS en AWS (Región Santiago y Concepción) para los contenedores Docker.**

**Facilidad de migración entre proveedores de nube: La estandarización a través de contenedores facilita el cambio de un proveedor de servicios en la nube a otro, o incluso a una infraestructura híbrida, sin reescribir gran parte del código o la configuración.**

**Flexibilidad en el entorno de desarrollo: Los desarrolladores pueden replicar fácilmente el entorno de producción en sus máquinas locales utilizando Docker, lo que agiliza el desarrollo y las pruebas, y reduce los problemas de "funciona en mi máquina".**

**Compatibilidad con diferentes sistemas operativos: Los contenedores Docker proporcionan un entorno aislado que incluye todas las dependencias necesarias, haciendo que las aplicaciones sean agnósticas al sistema operativo subyacente del host.**

**3.4 Otros Requisitos**

**Se utilizará OAuth2 para la autorización, basado en tokens de acceso que controlan permisos y tiempo de duración. JWT se utilizará como formato de token, conteniendo información del usuario firmada digitalmente.**

**4. Propuesta de Planificación**

**4.1 Descripción general acerca de la Planificación**

**La planificación se basa en un ciclo de vida de software con fases de recolección de requisitos, análisis, diseño, desarrollo, pruebas, implementación y mantenimiento. Se utilizará una metodología ágil para la implementación iterativa.**

**4.1.2 Definición del Equipo de Trabajo**

**Simon mena:Desarrollador Backend reación y mantenimiento de la parte del software que funciona en el servidor, la cual no es visible para el usuario final.**

**Juanpablo baez: Arquitecto de software diseña la estructura y los componentes de un sistema de software, garantizando su funcionalidad, escalabilidad, y seguridad**

**4.1.3 Definición de Actividades principales del Proyecto**

**Etapa 1: Análisis y Planificación (Semanas 1-3): Levantamiento de requerimientos funcionales y no funcionales, entrevistas simuladas con perfiles de usuario, revisión del sistema monolítico actual e identificación de cuellos de botella, y definición de objetivos del nuevo sistema.**

**Entregables: Documento de requerimientos del sistema, Informe de análisis del sistema actual.**

**Etapa 2: Diseño de la Nueva Arquitectura (Semanas 4-6): Diseño de la arquitectura de microservicios con vistas 4+1 (lógica, desarrollo, procesos, física y escenarios), elaboración de diagramas de casos de uso, diagramas de clases, diagrama de despliegue en la nube, y definición de servicios (autenticación, inventario, pedidos, clientes).**

**Entregables: Diseño arquitectónico detallado, especificaciones de cada microservicio.**

**Etapa 3: Planificación de la Migración y Gestión del Proyecto (Semanas 6-7): Desarrollo del plan de migración del sistema monolítico a microservicios, identificación de riesgos y medidas de mitigación, configuración del tablero colaborativo (Trello, Miro), y asignación de tareas individuales y tiempos estimados.**

**Entregables: Plan de migración con cronograma, capturas o enlaces del trabajo colaborativo.**

**Etapa 4: Implementación de Prototipo (Semanas 8-10): Desarrollo de los primeros microservicios (usuarios, inventario, ventas) utilizando herramientas como Spring Boot, Docker, MySQL, y despliegue inicial en entorno de pruebas.**

**Entregables: Código en GitHub, manual técnico y de instalación, documentación del uso de frameworks y herramientas.**

**Etapa 5: Evaluación Ética y Validación (Semana 11): Análisis de implicaciones éticas (privacidad de datos, impacto en el empleo, responsabilidad del desarrollador) y pruebas de funcionamiento básico.**

**Entregables: Informe de evaluación ética, resultados de pruebas iniciales.**

**Etapa 6: Presentación Final y Defensa del Proyecto (Semana 12): Preparación y ensayo de la presentación, subida final del repositorio y envío del enlace, exposición oral.**

**Entregables: Repositorio actualizado y documentado, presentación clara y estructurada.**

**4.1.4 Diagrama EDT**

[Insertar la Estructura EDT en formato diagrama consolidada que resolviste con tu equipo]

**4.1.5 Carta Gantt**



**Esta carta Gantt representa la planificación de un proyecto dividido en cinco etapas: Análisis, Diseño, Desarrollo, Pruebas y Mantenimiento, distribuidas entre el 7 y el 18 de abril de 2025.**

**-Azul (Análisis): Revisión del sistema actual y levantamiento de requerimientos, realizado los días 7 y 8 de abril.**

**-Verde (Diseño): Diseño de arquitectura y diagramas del sistema, desarrollado los días 9 y 10 de abril.**

**-Naranja (Desarrollo): Implementación de servicios principales, que abarca del 11 al 14 de abril.**

**-Morado (Pruebas): Pruebas funcionales e integración, realizadas el 15 y 16 de abril.**

**-Rojo (Mantenimiento): Documentación y ajustes finales, programados para el 17 y 18 de abril.**

**Cada color indica una etapa específica, mostrando claramente las fechas asignadas a cada tarea.**

**4.1.6 Resumen Costos del Desarrollo del Proyecto**

**El costo total estimado del proyecto es de $20.000.000 CLP (por ejemplo, 200 horas a $100.000 CLP/hora). Existe un costo adicional opcional de $350.000 CLP, lo que elevaría el total a $20.350.000 CLP.**

**Evaluacion 2:**

**1. Aplicar técnicas de modelamiento de software para representar la arquitectura y diseño del sistema usando estándares de UML, promoviendo una base sólida para el desarrollo y la integración.**

**Esto se refleja en la sección "Modelo de Vistas 4+1 aplicado al proyecto", donde se especifica el uso de UML para cada una de las vistas (Lógica, Desarrollo, Procesos, Física y Escenarios). Se mencionan explícitamente los diagramas UML a utilizar, como:**

**Diagrama de Caso de Uso**

**Diagrama de Clases**

**Diagrama de Paquetes**

**Diagrama de Componentes**

**Diagrama de Secuencia**

**Diagrama de Actividad**

**Diagrama de Despliegue**

**El objetivo de promover una base sólida para el desarrollo e integración se infiere de la descripción detallada de cada vista y de la forma en que cada diagrama contribuye a la comprensión del sistema.**

**2. Diseñar la arquitectura del software considerando los patrones de diseño 4+1. Los diagramas serán definidos por el docente.**

**El informe ya incluye una sección dedicada al "Modelo de Vistas 4+1 aplicado al proyecto".**

**Se detalla cada una de las 5 vistas (Lógica, Desarrollo, Procesos, Física y Escenarios), y para cada una se mencionan los diagramas UML específicos que deben ser confeccionados, tal como se solicitó anteriormente y que corresponderían a los "diagramas definidos por el docente".**

**Diagrama caso de uso completo.**

**Diagrama de secuencia; presentar dos ejemplos a partir del proyecto.**

**Diagrama de comunicación; presentar dos ejemplos a partir del proyecto.**

**Diagrama de clases; completa.**

**Diagrama de actividad; presentar dos ejemplos a partir del proyecto.**

**Diagrama de despliegue; presentar dos ejemplos a partir del proyecto.**

**3. Define estándares de calidad en el diseño de software, de acuerdo con los fundamentos de la ingeniería de software, contribuyendo a la calidad del software.**

**Esta parte está cubierta en la sección "Estándares de calidad aplicados en el diseño de software".**

**Se mencionan explícitamente los siguientes estándares y cómo contribuyen a la calidad del software:**

**Fiabilidad (garantizada por la replicación de datos en MySQL Cluster).**

**Mantenibilidad (facilitada por la arquitectura de microservicios).**

**Eficiencia (optimizada por el balanceo de carga automático).**

**Seguridad (a través de encriptación AES-256 y OAuth2).**

**Diagramas en plataformas como StarUML, Diagrams.net, Visio o Bizagi.**

**-Vista Lógica:**

**Muestra la funcionalidad del sistema.**

**Diagramas: clases, casos de uso.**

**-Vista de Desarrollo:**

**Organización interna del código.**

**Diagramas: componentes, paquetes.**



**-Vista de Procesos:**

**Comportamiento en tiempo de ejecución (flujos, concurrencia).**

**Diagramas: secuencia, actividad.**

**-Vista Física:**

**Cómo se despliega el sistema en la infraestructura.**

**Diagramas: despliegue, nodos.**

**-Vista de Escenarios (+1):**

**Casos de uso críticos para validar la arquitectura.**

**Ejemplo: alto tráfico en ventas.**

**Propuesta de Patrones de diseño que identifiquen como adecuados adoptar para este proyecto, debidamente justificados. (Mínimo dos patrones)**

**. Patrón API Gateway**

**Descripción: El patrón API Gateway actúa como un punto de entrada único para todas las solicitudes de los clientes. En lugar de que los clientes interactúen directamente con los microservicios individuales, dirigen sus solicitudes a la API Gateway, que luego las enruta a los servicios apropiados. La API Gateway puede realizar funciones como autenticación, autorización, limitación de velocidad, almacenamiento en caché, transformación de protocolos y agregación de respuestas.**

**Justificación para Perfulandia:**

**Simplificación del Cliente: Los clientes (front-end de Perfulandia, aplicaciones móviles, etc.) solo necesitan conocer un único endpoint de API, simplificando la lógica del lado del cliente y reduciendo la complejidad de la comunicación con múltiples microservicios.**

**Seguridad Centralizada: La API Gateway puede centralizar la autenticación y autorización (utilizando OAuth2, como se menciona en los requisitos), aplicando políticas de seguridad antes de que las solicitudes lleguen a los microservicios internos. Esto reduce la superficie de ataque y la complejidad de seguridad en cada microservicio individual.**

**Reducción de Latencia y Consumo de Recursos: Al agregar respuestas de múltiples microservicios en una sola respuesta al cliente, se reduce la latencia de red y el número de solicitudes que el cliente debe realizar, optimizando el uso de recursos tanto del cliente como de la red.**

**Abstracción de Microservicios Internos: La API Gateway oculta la topología interna de la arquitectura de microservicios, permitiendo que los servicios internos evolucionen de forma independiente sin afectar a los clientes. Esto es crucial para la mantenibilidad y evolución futura de Perfulandia.**

**Gestión de Fallos y Resiliencia: Una API Gateway puede implementar mecanismos de tolerancia a fallos como Circuit Breaker o Retry, aislando a los clientes de fallos en servicios individuales y mejorando la resiliencia general del sistema.**

**2. Patrón Saga**

**Descripción: El patrón Saga es una forma de gestionar la consistencia de datos en sistemas distribuidos que requieren transacciones que abarcan múltiples servicios. Una Saga es una secuencia de transacciones locales, donde cada transacción local actualiza la base de datos de un servicio específico y publica un evento. Si una transacción local falla, la Saga ejecuta una serie de transacciones de compensación para deshacer los cambios realizados por las transacciones locales anteriores. Existen dos implementaciones principales:**

**Coreografía: Cada servicio participante publica eventos que son consumidos por otros servicios, que a su vez reaccionan a esos eventos y ejecutan sus propias transacciones locales. No hay un coordinador central.**

**Orquestación: Un orquestador central (un servicio de Saga) dirige el flujo de trabajo de la Saga, enviando comandos a los servicios participantes y reaccionando a los eventos de respuesta.**

**Justificación para Perfulandia:**

**Consistencia de Datos en Transacciones Distribuidas: El proyecto Perfulandia involucra operaciones críticas que afectan a múltiples microservicios, como la realización de pedidos que implican el servicio de Carrito, Pedidos, Pagos e Inventario. Por ejemplo, al procesar un pedido, es necesario descontar el stock del inventario, registrar el pedido y procesar el pago. Si alguna de estas operaciones falla, es crucial que se deshagan las acciones anteriores para mantener la consistencia de los datos. El patrón Saga es ideal para manejar este tipo de transacciones distribuidas.**

**Evitar Transacciones Distribuidas 2PC (Two-Phase Commit): En un entorno de microservicios, el uso de transacciones distribuidas tradicionales (2PC) es problemático debido a su impacto en la disponibilidad y la escalabilidad. Saga ofrece una alternativa que permite la consistencia eventual sin sacrificar la independencia de los servicios y la capacidad de escalado.**

**Flexibilidad y Desacoplamiento: Tanto en la coreografía como en la orquestación, el patrón Saga fomenta el desacoplamiento entre los servicios. Los servicios individuales solo necesitan preocuparse por su propia transacción local y la publicación/consumo de eventos, sin tener conocimiento directo de la lógica de otros servicios.**

**Manejo de Fallos Complejo: El patrón Saga está diseñado específicamente para manejar fallos en transacciones distribuidas, lo cual es vital para un sistema como Perfulandia donde las fallas en el procesamiento de pedidos o pagos podrían tener graves consecuencias. Las transacciones de compensación aseguran que el sistema pueda recuperarse de estados inconsistentes.**

**EVALUACION 3:**

**1. Diseño del Prototipo Funcional**

**El prototipo representa un sistema de gestión de inventario para múltiples sedes, orientado a empresas en crecimiento como Perfulandia SPA, que requieren escalabilidad, seguridad y eficiencia en la administración de productos y pedidos.**

**Herramientas utilizadas:**

**Canva Mockup: Utilizado para crear interfaces simuladas con interactividad por hipervínculos, permitiendo validar la experiencia de usuario sin desarrollo de código.**

**Requisitos funcionales implementados:**

**Inicio de sesión con roles (Administrador, Operador, Sucursal)**

**Panel de control con estadísticas, alertas y pedidos activos**

**Gestión de productos por sede (alta, edición, consulta)**

**Gestión de pedidos por estado y sucursal**

**Administración de usuarios y permisos**

**Requisitos no funcionales considerados:**

**Seguridad básica: acceso diferenciado por roles**

**Usabilidad: interfaz clara y navegación intuitiva**

**Escalabilidad: estructura modular para crecimiento futuro**

**Relación con el Modelo 4+1:**

**Vista lógica: Interfaces clave del sistema (login, dashboard, formularios)**

**Vista de procesos: Flujo entre módulos simulados**

**Vista de desarrollo: Segmentación visual por funcionalidades**

**Vista física: Simulación de acceso distribuido por sedes**

**Vista de escenarios: Casos como “Crear pedido” o “Editar producto”**

**2. Plan de Pruebas (ISO 25000)**

**El plan de pruebas se basó en la normativa ISO/IEC 25000 SQuaRE (Software Product Quality Requirements and Evaluation), que contempla atributos como funcionalidad, fiabilidad, usabilidad, eficiencia y mantenibilidad.**

**Documentos entregados:**

**Planilla de casos de prueba: Incluye ID, caso, pasos, datos, resultados esperados y obtenidos.**

**Evidencias de pruebas: Capturas y descripciones de resultados según cada funcionalidad del prototipo.**

**Normativas aplicadas:**

**ISO/IEC 25010 (Modelo de calidad)**

**ISO/IEC 25040 (Proceso de evaluación)**

**3. Control de Cambios (Versionado del Prototipo)**

**Se documentaron todas las versiones del prototipo funcional con sus respectivas modificaciones y fechas.**

**Planilla de control de cambios:**

**Versión inicial (v1.0): Diseño de login y dashboard**

**Versión 1.1: Agregado módulo de productos**

**Versión 1.2: Implementación de gestión de pedidos**

**Versión 1.3: Inclusión de vista de estadísticas y mejoras de usabilidad**

**Calidad Del Software:**

**Para asegurar la calidad del prototipo funcional del sistema Perfulandia Digital, se utilizó como referencia el modelo ISO/IEC 25010, el cual define las características fundamentales que debe cumplir un producto de software de calidad. A continuación, se detalla cómo se abordaron estas características en el prototipo desarrollado:**

**Funcionalidad: Se implementaron funciones clave como login por rol, gestión de productos por sede, manejo de pedidos y visualización de estadísticas. Estas funcionalidades fueron diseñadas según los requisitos funcionales levantados previamente.**

**Usabilidad: Las interfaces fueron diseñadas con un enfoque claro e intuitivo, utilizando mockups con navegación simulada para facilitar la validación de la experiencia de usuario.**

**Seguridad: Se simula un acceso diferenciado por rol, con control de permisos y sesiones para usuarios del sistema.**

**Eficiencia: El flujo de navegación entre módulos está optimizado para representar un uso fluido y sin interrupciones.**

**Mantenibilidad: El diseño por componentes (pantallas independientes) permite modificar elementos sin afectar el resto del prototipo.**

**Portabilidad: El prototipo fue diseñado con estructura web responsiva, compatible con distintos dispositivos y navegadores.**

**Estas medidas garantizan una base sólida para la futura implementación del sistema real, considerando tanto la validación temprana con usuarios como el cumplimiento de estándares de calidad del software.**