UACM

Universidad Autónoma
de la Ciudad de México

NADA HUMANO ME ES AJENO

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA DE SOFTWARE

Desarrollo de software a la medida bajo la gestión de proyectos

T E S I S

PARA OBTENER EL TÍTULO DE

LICENCIADO EN INGENIERÍA DE SOFTWARE

P R E S E N T A

IXCHEL GIUSEPPE MORENO NAVARRETE

D I R E C T O R

DR. JUAN CARLOS CRUZ ROMERO

Ciudad de México, junio de 2025.

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Agradecimientos

Quiero dedicar esta tesis, con todo mi cariño y gratitud, a mis padres, quienes han sido mi mayor inspiración y apoyo a lo largo de este camino.

A ustedes, mamá y papá, gracias por su amor incondicional, por creer en mí incluso en los momentos en que yo dudaba, y por enseñarme con su ejemplo el valor del esfuerzo, la honestidad y la perseverancia.

Su confianza, sus palabras de aliento, sus sacrificios silenciosos y su fe constante en mí me dieron la fuerza para seguir adelante. Esta meta no sería posible sin todo lo que han hecho por mí, y cada logro que obtenga será siempre también suyo.

Gracias por estar siempre a mi lado. Esta tesis es para ustedes.

ÍNDICE.....

¡Error! Marcador no definido.

Capítulo 1 Introducción

Antecedentes

Los sistemas de punto de venta (POS) han sido una herramienta esencial para las empresas durante décadas. Inicialmente, estos sistemas consistían en simples registros manuales que permitían a los negocios registrar transacciones de manera básica. Con el tiempo, y especialmente desde la aparición de la tecnología electrónica en los años 70, los terminales POS comenzaron a incluir funciones avanzadas como el escaneo de códigos de barras y el procesamiento de tarjetas de crédito. Esto revolucionó la forma en que se gestionaban las ventas y los inventarios, permitiendo una mayor eficiencia y precisión (Stevenson, 2015).

En la actualidad, los sistemas POS han seguido evolucionando, incorporando tecnologías modernas como la computación en la nube y la integración con dispositivos móviles. Esta evolución ha permitido una mayor flexibilidad y funcionalidad, adaptándose a las necesidades específicas de diferentes industrias, incluyendo las farmacias (Chopra & Meindl, 2015). Las farmacias, en particular, tienen requisitos operativos únicos que incluyen la gestión de inventarios de medicamentos, la integración con sistemas de prescripción médica y el cumplimiento de estrictas regulaciones sanitarias (HealthIT.gov, n.d.). Un sistema POS a medida puede abordar estas necesidades específicas, ofreciendo beneficios significativos como una mayor eficiencia operativa, mejor control de inventarios, reducción de errores y un servicio al cliente mejorado.

Además, existen diversos tipos de sistemas de punto de venta disponibles actualmente, cada uno con sus propias ventajas y desventajas (Whitman & Mattord, 2018):

Punto de Venta Tradicional (Basado en Hardware):

- **Ventajas**: Mayor control sobre el hardware y software utilizados funciona offline en caso de pérdida de conexión a Internet, puede adaptarse específicamente a las necesidades de la farmacia.
- **Desventajas**: Mayor costo inicial debido a la inversión en hardware y software, actualizaciones y mantenimiento pueden ser más laboriosos, menos flexibilidad en términos de movilidad y acceso remoto.

Punto de Venta en la Nube (Cloud-Based):

- **Ventajas**: Acceso en cualquier lugar con conexión a Internet, actualizaciones automáticas de software, menor inversión inicial en hardware.
- **Desventajas**: Dependencia de una conexión a Internet estable, posibles preocupaciones de seguridad y privacidad de datos en la nube, costos recurrentes de suscripción.

Punto de Venta Móvil:

- **Ventajas**: Movilidad, puede utilizarse en dispositivos móviles como tabletas y teléfonos, permite atender a los clientes directamente en el piso de ventas, actualizaciones más fáciles y rápidas.
- **Desventajas**: Mayor susceptibilidad a robos o pérdidas de dispositivos móviles, puede requerir dispositivos adicionales como lectores de tarjetas.

POS con Integración de Pago Móvil:

- **Ventajas**: Permite a los clientes pagar con sus dispositivos móviles o billeteras electrónicas, agiliza el proceso de pago y reduce la necesidad de manejar efectivo, atrae a clientes que prefieren opciones de pago digitales.
- **Desventajas**: Requiere asegurarse de que el sistema de pago móvil esté bien integrado y sea seguro, puede haber tarifas adicionales asociadas con los pagos móviles.

A continuación, se presenta una tabla comparativa de algunos software POS disponibles en el mercado, que servirá como referencia para destacar las diferencias clave con el sistema POS a desarrollar:

Software POS	Plataforma	Funcionalidades principales	Ventajas destacadas	Desventajas
Square POS	Móvil / Web	Inventario, ventas, clientes, pagos móviles	Fácil de usar, integración con hardware móvil	Requiere conexión constante a Internet
Vend	Basado en la nube	Gestión de inventario, fidelización, reportes	Interfaz moderna, buen soporte técnico	Costos mensuales recurrentes
Lightspeed	Escritorio / Nube	Gestión de inventario, CRM, análisis de datos	Ideal para retail especializado	Curva de aprendizaje inicial
Pharmacy POS	Escritorio	Enfocado en farmacias, gestión de medicamentos y recetas	Cumple regulaciones sanitarias	Interfaz limitada, no siempre compatible
Sistema propuesto	Multiplataforma	Inventario, ventas, recetas, seguridad, movilidad	Adaptado a farmacias, interfaz intuitiva	En desarrollo, requiere pruebas y ajustes

Esta comparación permite destacar las fortalezas del sistema POS a desarrollar, el cual se centra en la personalización, la integración con sistemas médicos y el cumplimiento normativo.

El desarrollo de software a la medida juega un papel crucial en la creación de sistemas POS específicos para farmacias. Este tipo de desarrollo permite la creación de aplicaciones que satisfacen las necesidades únicas de un cliente, proporcionando soluciones personalizadas que no se pueden obtener con software comercial estándar. La gestión de proyectos es esencial en este contexto, ya que asegura que el desarrollo del software se realice de manera eficiente y efectiva, cumpliendo con todos los requisitos y objetivos establecidos (Sommerville, 2016).

Justificación

La implementación de un sistema de punto de venta a la medida es fundamental para satisfacer las necesidades específicas de las farmacias. Aunque existen soluciones comerciales de software POS, estas no siempre pueden adaptarse completamente a las particularidades de una farmacia, lo que puede resultar en limitaciones operativas y en una menor eficiencia (Kumar & Reinartz, 2018). La personalización de un sistema POS permite integrar funcionalidades específicas como la gestión detallada de inventarios de medicamentos, la integración con sistemas de prescripción electrónica y el cumplimiento de las regulaciones sanitarias (Whitman & Mattord, 2018).

Los sistemas de punto de venta tradicionales, aunque robustos, pueden presentar desventajas en términos de costos iniciales elevados y menor flexibilidad. Por otro lado, los sistemas en la nube ofrecen una solución más flexible y accesible, aunque dependen de una conexión a Internet estable. Los sistemas móviles, mientras tanto, proporcionan movilidad y flexibilidad, pero pueden ser más vulnerables a problemas de seguridad (Fling, 2009). La integración de pagos móviles es una característica cada vez más demandada que mejora la experiencia del cliente, pero también requiere consideraciones adicionales de seguridad (HealthIT.gov, n.d.).

La necesidad de un sistema POS a la medida en una farmacia se justifica por varios factores (Sommerville, 2016):

- **Personalización y Adaptabilidad**: Un sistema desarrollado específicamente para una farmacia puede adaptarse mejor a sus operaciones diarias y requerimientos únicos, mejorando la eficiencia y reduciendo errores.
- **Seguridad y Privacidad**: La gestión de datos sensibles, como la información de prescripciones y datos personales de los clientes, requiere altos estándares de seguridad. Un sistema personalizado puede garantizar que se implementen las medidas de seguridad adecuadas.
- **Optimización de Inventarios**: La gestión eficiente de inventarios es crucial en una farmacia para evitar desabastecimientos y caducidades. Un sistema a medida puede ofrecer soluciones específicas para la gestión de inventarios de medicamentos.
- **Cumplimiento Regulatorio**: Las farmacias deben cumplir con regulaciones específicas. Un sistema POS personalizado puede integrarse con sistemas de prescripción electrónica y asegurar el cumplimiento de todas las normativas.
- **Mejora del Servicio al Cliente**: Un sistema POS eficiente puede mejorar significativamente el servicio al cliente, proporcionando un proceso de venta más rápido y preciso, así como opciones de pago convenientes.
- **Estimación de costos del desarrollo del sistema de punto de venta:**

Fase / Capítulo	Horas Hombre	Rol Principal	Costo Estimado (MXN)
I. Inicio	320	Analista / Líder de Proyecto	$38,400
II. Planificación	320	Líder de Proyecto	$38,400
III. Ejecución	480	Dev Backend / Frontend	$72,000
IV. Control	480	QA / Project Manager	$57,600
V. Cierre	400	Todos los roles	$48,000
Total del Proyecto	**2,000 hrs**	-	**$254,400**

Esta tabla representa la estimación de recursos y costos para el desarrollo del sistema de punto de venta a la medida, alcanzando un total aproximado de **2,000 horas hombre** y un **costo estimado de $254,400 MXN**. Este monto cubre todas las fases del desarrollo del proyecto, desde el análisis inicial hasta el cierre del mismo, involucrando profesionales clave como analistas, líderes de proyecto, desarrolladores backend/frontend, y personal de aseguramiento de calidad (QA).

Comparado con los costos de soluciones comerciales existentes en el mercado, el desarrollo propio representa una alternativa más económica y estratégica. Por ejemplo:

- **Sistemas en renta:** pueden costar entre **$800 y $2,000 MXN mensuales por sucursal**, lo que representa entre **$9,600 y $24,000 MXN anuales**, sin incluir módulos extra o licencias adicionales.
- **Sistemas de compra:** suelen tener un costo inicial entre **$20,000 y $80,000 MXN**, pero también requieren pagos recurrentes por soporte, actualizaciones o personalizaciones, lo cual incrementa su costo real.

Además del factor económico, el sistema desarrollado a la medida garantiza una solución **totalmente alineada con las necesidades del cliente**, permitiendo funcionalidades específicas como:

- Gestión y alerta de productos por caducar.
- Reportes avanzados por semana y mes.
- Aplicación móvil con notificaciones y pedidos.
- Integración con Excel para importación/exportación.
- Control de stock y sugerencias inteligentes de compra.

Esto ofrece **una mayor eficiencia operativa**, evita gastos recurrentes y proporciona una plataforma tecnológica adaptada completamente al entorno de la farmacia. En consecuencia, no solo resulta más económico a largo plazo, sino que también es **más funcional, escalable y personalizable**.

En resumen, el desarrollo de un sistema de punto de venta a la medida bajo una gestión de proyectos adecuada no solo satisface las necesidades operativas y regulatorias de una farmacia, sino que también mejora la eficiencia, la seguridad y el servicio al cliente. Esto resulta en una solución integral que contribuye al éxito y sostenibilidad del negocio.

Objetivo General

Desarrollar un sistema de punto de venta a la medida para farmacias, que integre la gestión de inventarios, procesamiento de ventas, gestión de recetas y prescripciones electrónicas, y funciones de seguridad y privacidad, asegurando un funcionamiento eficiente y conforme a las regulaciones sanitarias, a través de una interfaz amigable y adaptable a diferentes dispositivos.

Objetivos Específicos

- **Desarrollar un módulo de gestión de inventarios** que permita registrar y rastrear productos farmacéuticos y suministros, actualizar automáticamente el inventario después de cada venta o compra, y generar alertas cuando los niveles de inventario sean bajos.
- **Implementar un sistema de procesamiento de ventas** que registre transacciones de venta, calcule impuestos y descuentos aplicables, y genere facturas y recibos para los clientes.
- **Crear un módulo de gestión de recetas** que registre y gestione recetas médicas, verifique la autenticidad de las recetas y su validez, y genere etiquetas y advertencias para los medicamentos.
- **Integrar la funcionalidad de prescripciones electrónicas**, permitiendo la recepción y procesamiento de recetas electrónicas, y la verificación y registro automático de estas.
- **Desarrollar un sistema de registro de clientes** que mantenga un historial de compras y permita la implementación de programas de lealtad y recompensas.
- **Garantizar la seguridad y privacidad de los datos** mediante la implementación de medidas de seguridad conformes a las regulaciones de la industria farmacéutica.
- **Optimizar la usabilidad del sistema** a través de una interfaz de usuario intuitiva y fácil de usar para el personal de la farmacia, asegurando una experiencia de usuario fluida.
- **Asegurar la compatibilidad y disponibilidad del sistema**, garantizando su funcionamiento en diferentes dispositivos (computadoras, tabletas, dispositivos móviles) y navegadores web, y su alta disponibilidad durante las horas de operación de la farmacia.

- **Facilitar el mantenimiento y escalabilidad del sistema**, permitiendo actualizaciones y expansiones futuras sin interrupciones significativas.

Alcance

El sistema de punto de venta a desarrollar estará enfocado en satisfacer las necesidades específicas de una farmacia, abordando aspectos clave como la gestión de inventarios, procesamiento de ventas, gestión de recetas y prescripciones electrónicas, y la seguridad de datos. La interfaz del sistema será amigable y adaptable a diferentes dispositivos, asegurando una experiencia de usuario óptima.

Funciones incluidas en el sistema:

- **Gestión de inventarios:** Registro, rastreo y actualización automática de productos. Generación de alertas de medicamentos próximos a caducar o quedar sin stock.
- **Procesamiento de ventas:** Registro de transacciones, cálculo de impuestos y descuentos, generación de facturas y recibos.
- **Gestión de recetas:** Registro, verificación y gestión de recetas médicas.
- **Prescripciones electrónicas:** Integración y procesamiento de recetas electrónicas.
- **Registro de clientes:** Historial de compras y programas de lealtad.
- **Seguridad y privacidad:** Medidas de seguridad conformes a las regulaciones.
- **Interfaz de usuario:** Intuitiva y fácil de usar, compatible con diversos dispositivos.
- **Mantenimiento y escalabilidad:** Facilita actualizaciones y expansiones futuras.
- **Estadísticas:** Generación de estadísticas sobre medicamentos más vendidos y las fechas de mayor venta.
- **Servicios:** Conexión con dispositivos móviles para mostrar datos de ventas en una aplicación creada específicamente para este propósito.
- **Pedidos:** Creación de solicitudes de pedidos a proveedores.

Roles del sistema:

- **Administrador:** Permisos para habilitar, deshabilitar o eliminar usuarios, y gestionar configuraciones del sistema.

- **Empleado de farmacia:** Puede registrar ventas, gestionar inventarios y procesar recetas.
- **Cliente:** Puede acceder a su historial de compras y utilizar programas de lealtad.
- **Roles para desarrollar el punto de venta:** El desarrollo de un sistema de punto de venta a la medida requiere la participación de un equipo multidisciplinario. A continuación, se presentan los principales roles involucrados y su costo promedio por hora en pesos mexicanos, con base en el mercado nacional:

Rol	Descripción	Costo aproximado por hora (MXN)
Analista de requerimientos	Identifica necesidades del cliente y las traduce en requerimientos funcionales y técnicos.	$300 – $600
Arquitecto de software	Diseña la estructura del sistema, define tecnologías y asegura escalabilidad y seguridad.	$600 – $1,000
Desarrollador backend	Implementa la lógica de negocio, controladores, servicios y base de datos.	$350 – $700
Desarrollador frontend	Construye la interfaz gráfica y experiencia del usuario (HTML, CSS, JS, etc.).	$300 – $600
Diseñador UX/UI	Diseña interfaces amigables, centradas en la experiencia del usuario.	$300 – $500
Administrador de base de datos	Administra y optimiza la base de datos, respalda información y garantiza su integridad.	$350 – $700

Este sistema garantizará un funcionamiento eficiente y conforme a las regulaciones sanitarias, mejorando la operatividad y el servicio al cliente de la farmacia.

Capítulo 2 Marco Teórico

Introducción

En el desarrollo de software, comprender y aplicar principios fundamentales es esencial para crear sistemas eficientes, escalables y mantenibles. El libro de Ian Sommerville, *Software Engineering* (2011), proporciona una base sólida sobre las prácticas y metodologías en la ingeniería de software, abordando desde los fundamentos hasta los enfoques avanzados de desarrollo. Este capítulo se centra en la aplicación de principios clave en el desarrollo de sistemas, incluyendo patrones de diseño, arquitectura limpia, seguridad, y tecnologías web, con una atención especial a las herramientas y frameworks modernos.

Principios Fundamentales de Ingeniería de Software:

Sommerville (2011) destaca que la ingeniería de software es un enfoque disciplinado y estructurado para el desarrollo de software que busca mejorar la calidad y la eficiencia del proceso de desarrollo. Los principios fundamentales incluyen:

- **Enfoque Sistemático:** El desarrollo de software debe seguir un enfoque sistemático y organizado, que permita una planificación, diseño, implementación, y mantenimiento efectivos del sistema.
- **Desarrollo Iterativo e Incremental:** Las metodologías modernas de desarrollo, como el Proceso Unificado y Agile, abogan por ciclos iterativos e incrementales que permiten ajustar y mejorar el software continuamente basado en la retroalimentación y las necesidades cambiantes.
- **Gestión de Riesgos:** Identificar y mitigar riesgos desde las primeras fases del desarrollo es crucial para evitar problemas mayores en fases posteriores. Esto incluye riesgos técnicos, operacionales y de gestión.
- **Calidad y Mantenimiento:** La calidad del software es fundamental, no solo para cumplir con los requisitos funcionales, sino también para asegurar la robustez, seguridad y facilidad de mantenimiento del sistema a largo plazo.

Enfoque en el Desarrollo de Sistemas Web y Frameworks Modernos:

A medida que la tecnología evoluciona, las herramientas y tecnologías para el desarrollo de software también cambian. Sommerville (2011) discute cómo la evolución

de las tecnologías web y los frameworks modernos han transformado el desarrollo de sistemas, facilitando procesos como la integración continua, la gestión de dependencias, y el desarrollo ágil. Este capítulo examina estas tecnologías y cómo se aplican en el contexto del proyecto actual, utilizando herramientas y frameworks como Spring Boot, Maven, JSON, y técnicas de inyección de dependencias.

La comprensión de estos principios y tecnologías permite diseñar sistemas que no solo satisfacen los requisitos funcionales, sino que también son robustos, escalables y fáciles de mantener, contribuyendo a un desarrollo de software de alta calidad y eficiencia.

Principios del Proceso Unificado

El Proceso Unificado (PU) es una metodología de desarrollo de software que proporciona una estructura para la planificación, desarrollo, y gestión de proyectos. Basado en un enfoque iterativo e incremental, el PU se enfoca en la adaptación continua y la mejora del software a lo largo de su ciclo de vida. Sommerville (2011) detalla que el PU es crucial para garantizar la calidad y la efectividad del desarrollo de software al proporcionar un marco que guía la implementación de procesos y prácticas efectivas.

Fases Clave del Proceso Unificado

El Proceso Unificado se divide en varias fases clave que estructuran el ciclo de vida del desarrollo de software. Estas fases incluyen:

- **Inicio:** En esta fase se definen los objetivos del proyecto, se identifican los requisitos y se establece el alcance. Se realiza una evaluación preliminar de viabilidad y se planifica el desarrollo.
- **Elaboración:** Durante la elaboración, se desarrollan modelos detallados del sistema, se refina la arquitectura y se planifican las iteraciones del desarrollo. Es crucial para identificar riesgos y diseñar la base del sistema.
- **Construcción:** En esta fase, se implementa el sistema según los modelos y especificaciones desarrolladas. Se realiza la integración continua y se preparan versiones del software para pruebas y revisiones.
- **Transición:** La fase de transición implica la entrega del software al usuario final. Se realizan pruebas finales, se corrigen errores y se prepara el sistema para su despliegue y uso en producción.

Estas fases aseguran un enfoque estructurado y iterativo para el desarrollo, permitiendo ajustes continuos basados en la retroalimentación y cambios en los requisitos.

Fases de Trabajo y Disciplinas del Proceso Unificado

Cada fase del Proceso Unificado involucra varias disciplinas de trabajo que aseguran la cobertura integral del desarrollo:

- **Modelado de Negocio:** Define los procesos empresariales y cómo el sistema propuesto se alinea con los objetivos del negocio.
- **Análisis de Requisitos:** Identifica y documenta los requisitos del sistema desde la perspectiva del usuario y del negocio.
- **Diseño:** Desarrolla la arquitectura y el diseño detallado del sistema, asegurando que cumpla con los requisitos y estándares.
- **Implementación:** Codifica y construye el sistema basado en los diseños y modelos desarrollados.
- **Pruebas:** Verifica y valida el sistema para asegurar que cumple con los requisitos y funciona correctamente.
- **Despliegue:** Implementa el sistema en el entorno de producción y asegura que esté operativo para los usuarios finales.
- **Mantenimiento:** Realiza ajustes y mejoras continuas al sistema, corrigiendo errores y adaptándose a cambios en los requisitos.

Cada disciplina contribuye a un aspecto crucial del desarrollo, asegurando que el sistema sea funcional, eficiente y alineado con los objetivos del negocio.

UML

El Lenguaje de Modelado Unificado (UML) es una herramienta fundamental en el Proceso Unificado para la visualización, especificación, construcción y documentación de sistemas. UML proporciona un conjunto de diagramas que facilitan la comprensión y comunicación del diseño del sistema:

- **Diagramas de Casos de Uso:** Representan las interacciones entre los usuarios y el sistema, definiendo las funcionalidades del sistema desde la perspectiva del usuario.
- **Diagramas de Clases:** Muestran la estructura estática del sistema, incluyendo las clases, atributos y relaciones.
- **Diagramas de Secuencia:** Representan el flujo de mensajes entre los objetos del sistema a lo largo del tiempo.
- **Diagramas de Actividad:** Muestran el flujo de trabajo y las actividades dentro del sistema.

UML facilita una comunicación clara entre los miembros del equipo y proporciona una documentación integral del sistema, lo cual es crucial para el desarrollo efectivo y la gestión del proyecto.

Arquitectura Limpia

Crear un sistema mantenible, escalable y fácil de probar no es una tarea sencilla. La arquitectura de software es una herramienta fundamental que proporciona una base sólida para que un programa funcione de manera efectiva y confiable. Sin embargo, una buena arquitectura por sí sola no garantiza el correcto funcionamiento del sistema. Es necesario también especificar las mejores prácticas para el desarrollo de los componentes del sistema.

Atributos No Funcionales:

La arquitectura de software aborda los atributos no funcionales que definen las características de calidad del sistema. Estos atributos no son las funcionalidades específicas del sistema, sino restricciones sobre su funcionamiento, como disponibilidad y robustez.

Arquitectura Limpia:

Las arquitecturas limpias permiten separar el sistema en diferentes capas, donde cada capa solo tiene acceso a capas de nivel inferior. Esta separación clara de responsabilidades se realiza a través de interfaces, evitando problemas comunes en arquitecturas no limpias, como el acoplamiento elevado entre componentes y la dificultad para realizar cambios.

Ventajas de una Arquitectura Limpia:

- **Independencia del Framework:** Reduce la complejidad de realizar cambios o actualizaciones al no depender de bibliotecas de funciones específicas.
- **Independencia de la Interfaz de Usuario (UI):** Permite realizar cambios en la UI sin afectar el resto del sistema.
- **Independencia de la Base de Datos:** Desacopla el modelo de dominio de la base de datos, facilitando cambios en la base de datos.
- **Independencia de Dependencias Externas:** Las reglas de negocio no dependen de interfaces externas.
- **Comprobabilidad:** Permite probar las reglas de negocio sin necesidad de la interfaz de usuario, la base de datos, o el servidor web.

Seguridad en el Desarrollo

La seguridad es fundamental en el desarrollo de sistemas informáticos para proteger la confidencialidad, integridad y disponibilidad del sistema. La seguridad informática se centra en preservar estos atributos y abordar vulnerabilidades comunes. Candel (2020) señala que los principios de seguridad para el desarrollo de software incluyen:

- **Minimizar el Área de Superficie de Ataque:** Reducir los servicios expuestos y deshabilitar módulos no utilizados para limitar posibles vulnerabilidades.
- **Seguridad Predeterminada:** Asegurar que las configuraciones de bibliotecas sean seguras antes de liberar el software.
- **Privilegios Mínimos:** Limitar los privilegios de los usuarios a solo aquellos necesarios para su rol.
- **Validación de Datos de Entrada:** Limpiar y validar los datos de entrada para prevenir inyecciones de código y otras amenazas.
- **Defensa en Profundidad:** Implementar múltiples capas de defensa para proteger el sistema contra intrusiones.
- **Control Seguro de Errores:** Proporcionar mensajes de error mínimos para evitar revelar información sensible.
- **Separación de Funciones:** Aplicar principios de diseño como SOLID para minimizar el impacto de errores y vulnerabilidades.

La seguridad del sistema no debe basarse únicamente en la confidencialidad de su implementación. Es fundamental aplicar las mejores prácticas de seguridad desde el inicio del desarrollo.

Sistemas Web

Los sistemas web han evolucionado significativamente con el tiempo, adoptando diferentes enfoques y tecnologías para mejorar la interacción y la funcionalidad. Según Oriols y Gutiérrez (2018), los sistemas web se pueden clasificar en tres tipos:

- **Cliente y Servidor Estáticos:** En este tipo, el servidor solo envía páginas estáticas al cliente, sin realizar cambios dinámicos. El cliente solo muestra estas páginas, sin capacidad de modificar el contenido.
- **Cliente Estático y Servidor Dinámico:** Aquí, el cliente es estático y no realiza cambios, mientras que el servidor es responsable de modificar los datos. Cada solicitud del cliente resulta en una página actualizada enviada por el servidor.
- **Cliente/Servidor Dinámicos:** Tanto el cliente como el servidor pueden realizar cambios en las páginas. El cliente, a menudo usando JavaScript, puede modificar la página sin necesidad de una nueva solicitud al servidor.

La mayoría de los sistemas web modernos emplean el enfoque de Cliente/Servidor Dinámicos, que se puede implementar siguiendo dos patrones de diseño:

- **Multi-Page Applications (MPA):** En este enfoque, el servidor envía páginas completas, lo que puede causar problemas de lentitud en aplicaciones grandes. Para mitigar esto, se puede usar AJAX para permitir que el cliente tenga capacidad dinámica.
- **Single Page Applications (SPA):** En este enfoque, el cliente es responsable de modificar la página, mientras que el servidor solo envía la información necesaria para actualizar la página. Este enfoque minimiza la lentitud de navegación y mejora la experiencia del usuario.

Herramientas de desarrollo

Maven y JSON

- **Maven:** Maven es una herramienta de gestión y automatización de proyectos para Java. Facilita la construcción, gestión de dependencias y el mantenimiento de proyectos de software. Maven utiliza un archivo de configuración (pom.xml) para definir las dependencias del proyecto, las versiones y las configuraciones necesarias para compilar y construir el software. Esto permite una gestión coherente y eficiente de los recursos y facilita la integración con otras herramientas y sistemas de construcción.
- **JSON:** JavaScript Object Notation (JSON) es un formato ligero para el intercambio de datos que es fácil de leer y escribir para los humanos, y fácil de parsear y generar para las máquinas. JSON se utiliza comúnmente para transmitir datos entre un servidor y una aplicación web en formato de texto. Su estructura basada en texto hace que sea ideal para aplicaciones web modernas, ya que facilita la comunicación entre el cliente y el servidor.

Spring Framework Boot

Spring Boot:

Es una extensión del framework Spring que simplifica la configuración y el desarrollo de aplicaciones Java. Proporciona una serie de características y mejoras que permiten un desarrollo más rápido y sencillo de aplicaciones basadas en Spring. Entre sus características destacan:

- **Configuración Automática:** Spring Boot ofrece una configuración automática para muchas de las configuraciones

comunes necesarias en las aplicaciones, reduciendo la necesidad de configuraciones manuales.

- **Aplicaciones Autónomas:** Permite crear aplicaciones autónomas con servidores integrados, eliminando la necesidad de desplegar aplicaciones en servidores externos.
- **Dependencias Simplificadas:** Facilita la gestión de dependencias mediante la utilización de starters, que agrupan y gestionan dependencias comunes.

Inyección de Dependencias:

La inyección de dependencias es un patrón de diseño clave en Spring Framework que promueve la separación de responsabilidades y la modularidad. Permite a los desarrolladores inyectar dependencias en una clase en lugar de crear las instancias dentro de la clase misma. Esto facilita el mantenimiento, las pruebas y la escalabilidad del código. Spring Boot utiliza este patrón para manejar la creación y gestión de los beans, proporcionando un entorno controlado y eficiente para el desarrollo de aplicaciones.

Capítulo 3 Requisitos

Visión general del proyecto

El proyecto tiene como objetivo desarrollar un sistema de punto de venta (POS) personalizado para una farmacia, que permita la gestión eficiente de ventas, inventario, usuarios, generación de facturas, y conexión con una aplicación móvil. Este sistema no solo optimizará los procesos operativos diarios, sino que también facilitará la toma de decisiones estratégicas mediante reportes detallados y estadísticas.

Se utilizarán tecnologías como **Spring Boot 4** para el backend, **MySQL 8** como base de datos, **Java 17.0.1** como lenguaje de programación, **CSS Bootstrap** para la interfaz de usuario, y **JavaScript** para la interacción del lado del cliente. La gestión del proyecto se realizará en **Windows 10**, con **Git** como sistema de control de versiones y **Gradle** como herramienta de automatización. Además, se emplearán **JUnit** para pruebas unitarias y **Thymeleaf** para la generación de vistas.

El sistema integrará módulos para gestionar promociones, descuentos automáticos para productos próximos a caducar, notificaciones sobre productos faltantes o por caducar, y permitirá la generación de reportes para apoyar la toma de decisiones estratégicas (Jacobson, Booch, & Rumbaugh, 2000).

Propósito del proyecto

El propósito de este proyecto es proporcionar una herramienta robusta y eficiente que optimice la gestión de ventas e inventarios de una farmacia, permitiendo a los usuarios gestionar productos, usuarios, finanzas, y recibir notificaciones mediante una aplicación móvil. Además, facilitará la toma de decisiones mediante reportes detallados sobre ventas y productos más vendidos (Sommerville, 2011).

Objetivos

- Desarrollar un sistema de punto de venta que permita la venta de productos, gestión de inventario y generación de facturas de manera eficiente.
- Implementar un control de acceso basado en roles para gestionar la seguridad del sistema.
- Integrar una aplicación móvil para recibir notificaciones y realizar pedidos, así como gestionar productos más vendidos y productos por caducar.
- Facilitar la importación y exportación de productos mediante archivos Excel.
- Desarrollar un sistema de reportes y estadísticas que ayude en la toma de decisiones estratégicas (Sommerville, 2011).
- Implementar un sistema de promociones automáticas para productos cercanos a su fecha de caducidad.

Aspectos deseados

El sistema debe ser fácil de usar y con una interfaz intuitiva. Además, debe ser:

- **Escalable**: Capaz de manejar un número creciente de usuarios y productos sin pérdida de rendimiento.
- **Integrable**: Debe poder conectarse con aplicaciones móviles y otros sistemas futuros.
- **Seguro**: Debe cumplir con los principios de seguridad mencionados por Jacobson, Booch y Rumbaugh (2000) y tener un sistema de autenticación robusto.

Factores críticos de éxito

- **Comunicación continua** con el cliente para garantizar que el desarrollo esté alineado con las necesidades del negocio.
- **Cumplimiento de plazos** mediante metodologías ágiles como Scrum (Schwaber & Sutherland, 2017).
- **Capacitación del personal** para la correcta adopción del sistema.
- **Seguridad y estabilidad del sistema**, minimizando riesgos de manejo de datos.
- **Sincronización con la aplicación móvil** para gestionar pedidos y notificaciones en tiempo real.
- **Corte de caja eficiente** para cerrar operaciones diarias sin problemas.

Restricciones

- El sistema debe ser compatible con **Windows 10** y seguir los lineamientos de **Spring Boot 4**.
- La base de datos central será **MySQL 8**.
- El sistema debe generar reportes en tiempo real sin afectar el rendimiento.
- El desarrollo se realizará con **Java 17.0.1**, **CSS Bootstrap**, y **JavaScript**.

Lugares de interés

El sistema se implementará en una farmacia y se utilizará para:

- **Caja registradora**: Lugar donde se realizarán las ventas y la interacción directa con el cliente.
- **Almacén**: Gestión del inventario con alertas para productos próximos a caducar.
- **Área administrativa**: Gestión de usuarios, permisos y reportes.
- **Móvil**: Proveer acceso desde dispositivos móviles para gestionar productos, pedidos y recibir notificaciones.

Actores

- **Administrador**: Usuario encargado de gestionar el inventario, ventas, usuarios, y reportes.

- **Cajero/Vendedor**: Usuario que procesará ventas y gestionará el corte de caja.
- **Cliente móvil**: Usuario de la aplicación que recibirá notificaciones y realizará pedidos.
- **Proveedor**: Encargado de abastecer productos y cargarlos al sistema mediante archivos Excel.

Casos de uso

Los casos de uso del sistema de punto de venta se describen a continuación, y se presentan de forma gráfica en la Imagen 3.2: Diagrama de casos de uso del sistema de punto de venta.

Caso de uso 1: Gestión de inventario

- **Actor principal**: Administrador
- **Descripción**: Permite añadir, modificar o eliminar productos del inventario.
- **Flujo alternativo**: Si los datos son incorrectos, el sistema mostrará un error.

Caso de uso 2: Realización de ventas

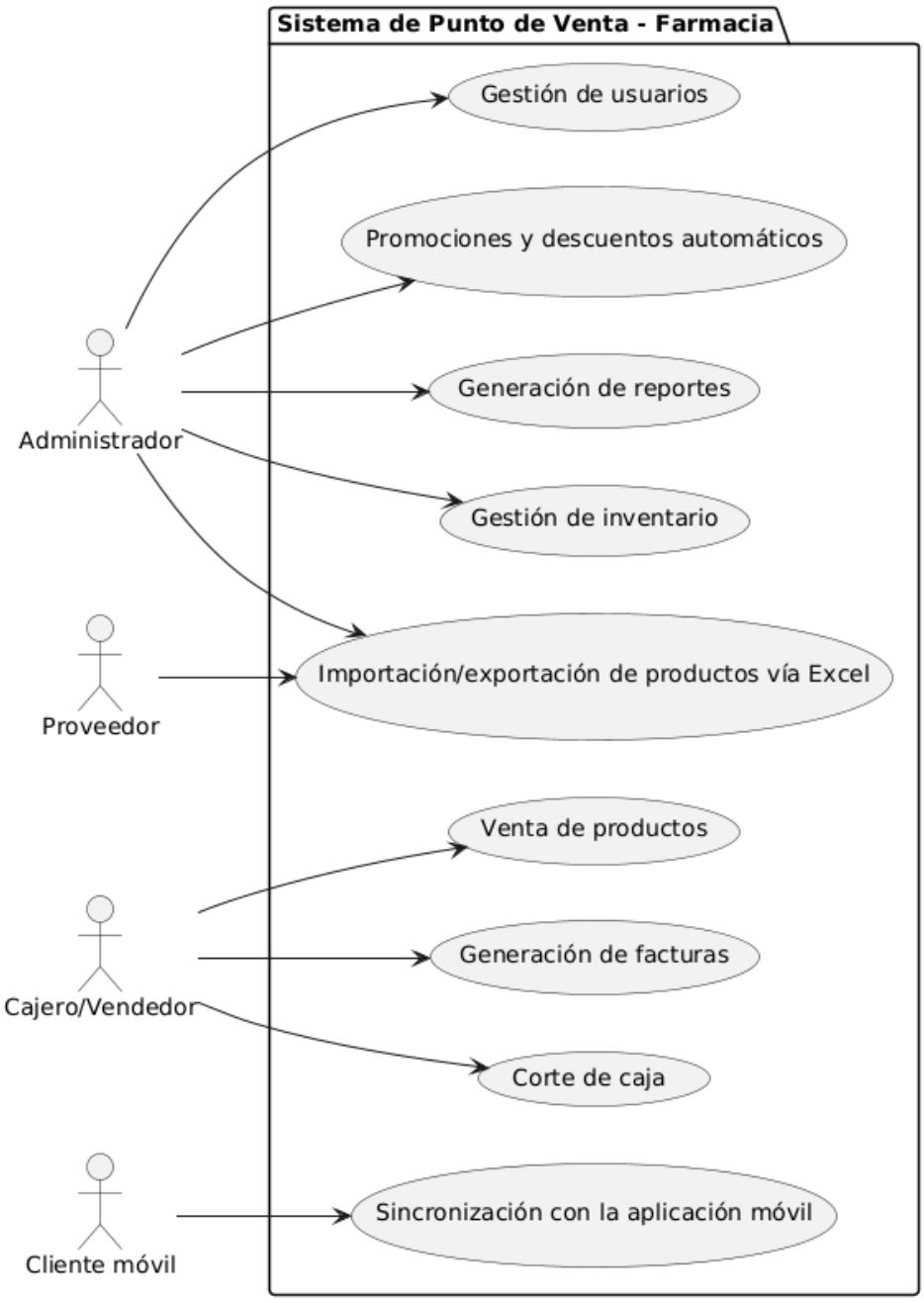
- **Actor principal**: Cajero/Vendedor
- **Descripción**: Permite registrar ventas y actualizar el inventario.
- **Flujo alternativo**: Si no hay stock, se muestra un mensaje de advertencia.

Caso de uso 3: Generación de reportes

- **Actor principal**: Administrador
- **Descripción**: Permite generar reportes de ventas semanales y mensuales.

Caso de uso 4: Sincronización con aplicación móvil

- **Actor principal**: Cliente móvil
- **Descripción**: El cliente recibe notificaciones sobre stock o productos próximos a caducar y puede realizar pedidos.



(Imagen 3.2: Diagrama de casos de uso del sistema de punto de venta)

Capítulo 4 Análisis

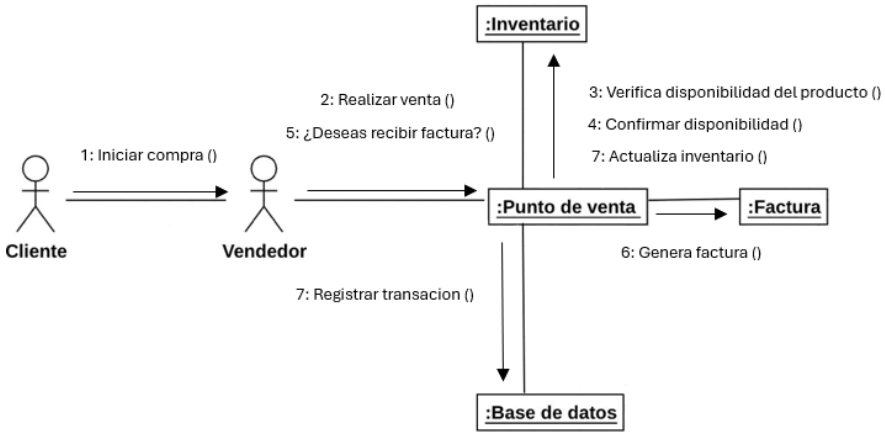
En este capítulo se describirá el proceso de análisis de los requisitos recopilados para el desarrollo del sistema de punto de venta (POS) para una farmacia. El objetivo es identificar las funcionalidades clave y la interacción entre los componentes del sistema. Se presentarán diagramas de colaboración, secuencia y clases para ilustrar cómo se gestionan las interacciones entre los actores y los objetos principales del sistema.

Diagrama de colaboración

El **Diagrama de Colaboración** que se muestra en la **Imagen 4.1: Diagrama de Colaboración para "Gestionar venta de productos"**, ilustra la interacción entre los objetos principales del sistema en este caso de uso. En este escenario, un cliente realiza una compra en la farmacia, y el sistema debe procesar la venta, actualizar el inventario y generar una factura.

El flujo principal de la colaboración comienza cuando el Cajero envía un mensaje al objeto POS para registrar una venta. El objeto POS verifica la disponibilidad del producto en el Inventario. Si el producto está disponible, el objeto Inventario actualiza la cantidad de stock y envía un mensaje de confirmación al POS. El sistema entonces genera una Factura, que se envía al cliente, y el POS almacena la transacción en la base de datos.

Este proceso ilustra cómo los diferentes componentes colaboran para completar una transacción de venta de manera eficiente.



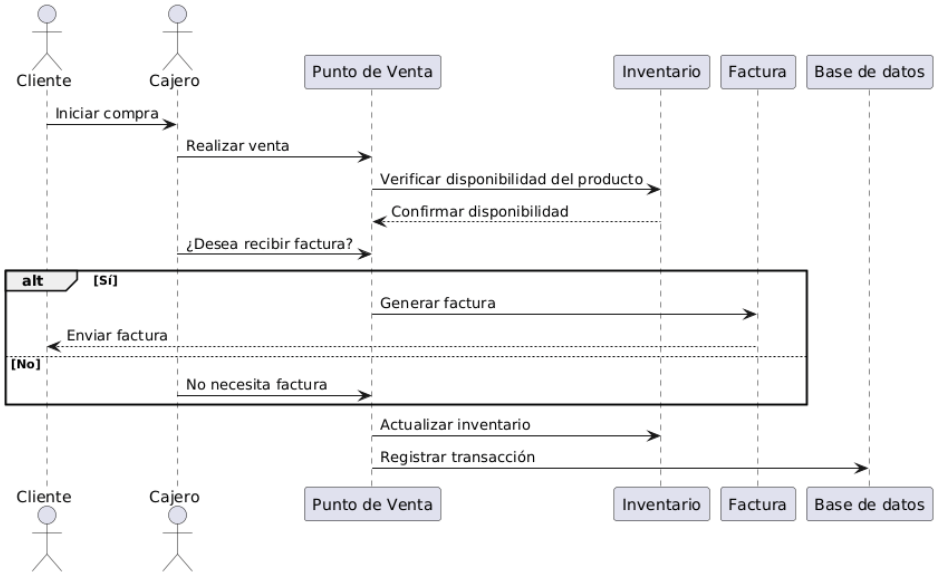
(Imagen 4.1: Diagrama de Colaboración para "Gestionar venta de productos")

Diagrama de secuencia

El **Diagrama de Secuencia** que se presenta en la **Imagen 4.2: Diagrama de Secuencia para "Gestionar venta de productos"**, refleja el mismo caso de uso "Gestionar venta de productos", detallando el orden específico en el que ocurren las interacciones entre los objetos. Esto permite visualizar los pasos en una secuencia cronológica.

- El Cajero inicia la venta enviando el mensaje "realizarVenta" al POS.
- El POS solicita al Inventario la disponibilidad del producto.
- El Inventario verifica el stock y responde con un mensaje confirmando la disponibilidad.
- El POS genera una Factura y actualiza el registro de ventas.
- El Cliente recibe la factura, y el sistema almacena la transacción.

Este diagrama permite una mejor comprensión del flujo lógico y temporal en la gestión de ventas dentro del sistema.



(Imagen 4.2: Diagrama de Secuencia para "Gestionar venta de productos")

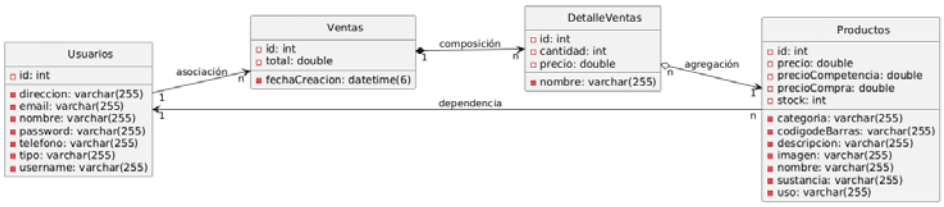
Diagrama de clases

El **Diagrama de Clases** que se muestra en la **Imagen 4.3: Diagrama de Clases del sistema de punto de venta**, proporciona una representación estructural del sistema de punto de venta orientado a objetos. Las clases clave identificadas en el análisis incluyen:

- **POS**: La clase principal que coordina las ventas y la interacción con otros objetos.
- **Producto**: Clase que representa los artículos en el inventario, con atributos como nombre, precio y cantidad en stock.
- **Inventario**: Clase que gestiona el stock de productos, incluyendo métodos para agregar y eliminar productos.
- **Factura**: Clase que se encarga de la creación y almacenamiento de facturas generadas en cada venta.

- **Usuario**: Clase que representa a los diferentes tipos de usuarios del sistema (administrador, cajero).

Este diagrama también ilustra las relaciones de herencia y agregación entre las clases. Por ejemplo, la clase **Usuario** tiene diferentes subclases para los roles de administrador y cajero, mientras que la clase **Factura** está asociada con **Producto** para registrar los detalles de cada venta.



(Imagen 4.3: Diagrama de Clases del sistema de punto de venta)

Análisis de requisitos

Los requisitos identificados en el capítulo anterior se han modelado en estos diagramas para asegurar que todas las funcionalidades del sistema de punto de venta se reflejan de manera precisa. El análisis incluye la interacción con la aplicación móvil, permitiendo a los usuarios recibir notificaciones sobre productos próximos a caducar, gestionar el inventario y realizar pedidos.

Los diagramas también reflejan cómo el sistema manejará la importación y exportación de datos a través de archivos Excel, la generación de reportes semanales y

mensuales, y la implementación de promociones automáticas para productos que están cerca de caducar.

Estos diagramas permiten visualizar y documentar cómo las diferentes partes del sistema colaboran y actúan en conjunto para garantizar que los procesos de venta y gestión de inventario en la farmacia funcionen de manera eficiente.

Capítulo 5 Diseño

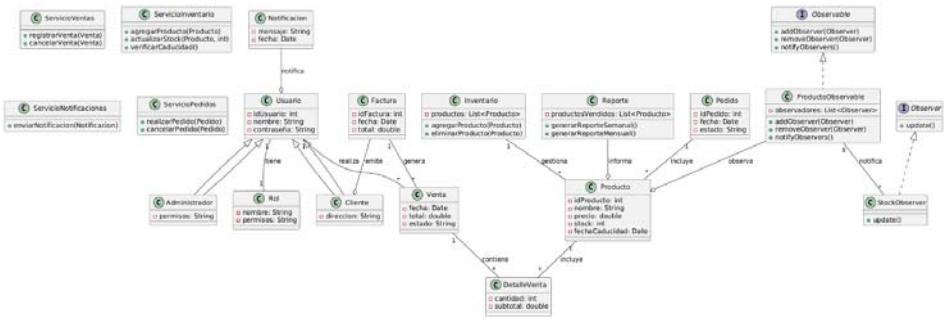
En este capítulo, se detallarán las tareas realizadas para crear el diseño del sistema de punto de venta para la farmacia. Siguiendo el Proceso Unificado, se utilizaron diagramas UML para representar de manera gráfica las decisiones de diseño tomadas para las diversas funcionalidades del sistema.

Diagrama de clases

En el **Diagrama de Clases** que se presenta en la **Imagen 5.1: Diagrama de Clases del sistema de ventas e inventario**, se pueden observar las relaciones entre las clases que conforman el sistema. Un aspecto clave de este diseño es el uso predominante de la composición y agregación en las relaciones entre las clases. Como se vio en el capítulo de análisis, las clases que representaban a los usuarios presentaban relaciones de herencia, mientras que en este diagrama se ha optado por utilizar composición para algunas de las clases, favoreciendo así la modularidad y flexibilidad del sistema, siguiendo el principio de diseño "Favor the composition of objects over class inheritance".

Además, se ha aplicado el patrón de diseño **Observer**, donde ciertos componentes del sistema reaccionan automáticamente a los cambios en el estado de otros objetos, lo que permite una actualización fluida y sincronizada de los datos en tiempo real, fundamental para gestionar el inventario, la facturación y las notificaciones de productos próximos a caducar o faltantes de stock.

El diagrama de clases refleja también la relación entre los productos, facturas y usuarios, facilitando la correcta organización y la implementación del sistema de ventas y control de inventario.

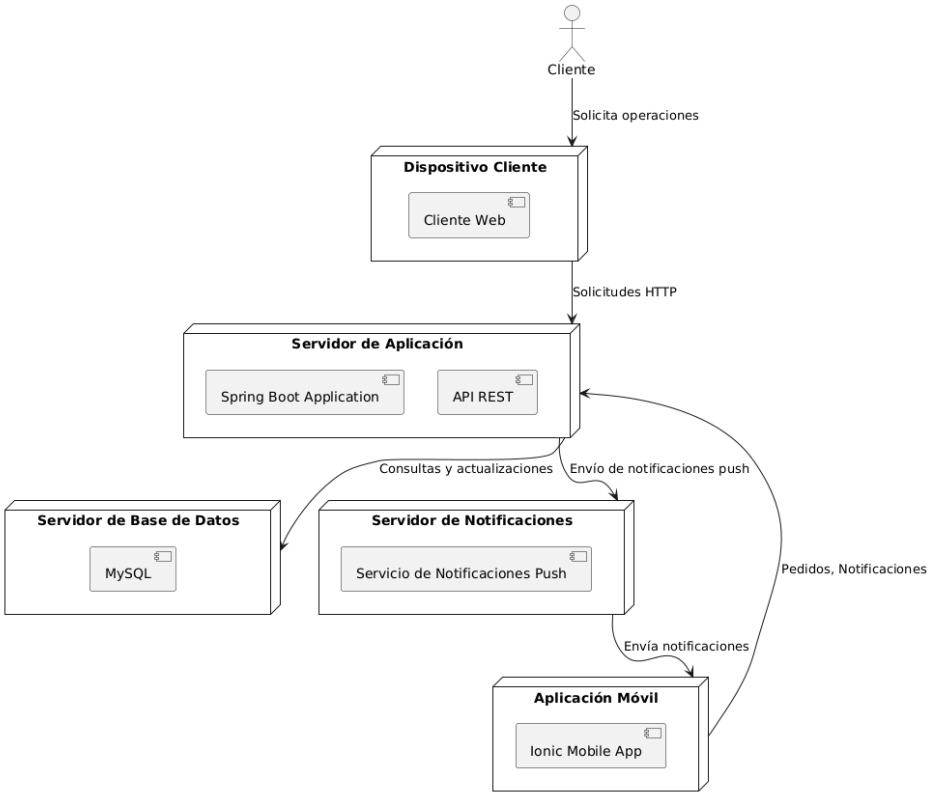


(Imagen 5.1: Diagrama de Clases del sistema de ventas e inventario)

Diagrama de despliegue

Para optimizar el rendimiento del sistema y asegurar su capacidad de escalabilidad, se ha diseñado una **Arquitectura de Despliegue** representada en la **Imagen 5.2: Diagrama de Despliegue del sistema de punto de venta**, basada en la separación de componentes. El sistema está previsto para ser instalado en dos servidores separados: uno para la aplicación web que incluye la gestión del punto de venta y otro dedicado a la base de datos. Esta configuración permite un manejo eficiente de las operaciones comerciales, como el procesamiento de ventas, facturación y la actualización de inventario en tiempo real.

El sistema también está diseñado para soportar el protocolo HTTPS, garantizando así la seguridad en las comunicaciones. Los empleados podrán acceder al sistema desde cualquier dispositivo conectado a la red, como computadoras, tablets o teléfonos móviles, permitiendo una gestión centralizada y segura.

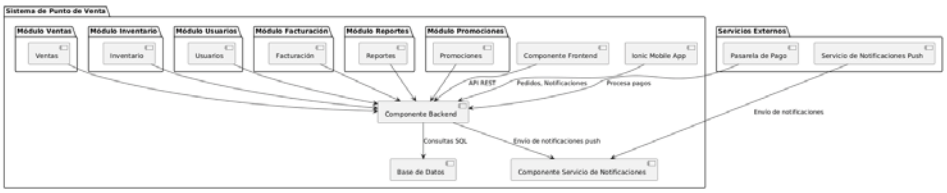


(Imagen 5.2: Diagrama de Despliegue del sistema de punto de venta)

Diagrama de componentes

El **sistema de punto de venta** está conformado por múltiples componentes, los cuales interactúan entre sí a través de interfaces bien definidas. En la **Imagen 5.3: Diagrama de Componentes del sistema de punto de venta**, se visualiza la relación entre componentes como el módulo de ventas, gestión de inventario, usuarios y facturación.

Por ejemplo, el componente que maneja el inventario interactúa directamente con la base de datos para actualizar las existencias en tiempo real y generar alertas sobre productos que están próximos a agotarse o caducar. Además, los servicios de facturación están diseñados para integrarse con sistemas externos de generación de facturas electrónicas.



(**Imagen 5.3: Diagrama de Componentes del sistema de punto de venta)**

Arquitectura

El sistema sigue una arquitectura de **tres capas**, que incluye la capa de presentación (frontend), la capa de lógica de negocio (backend) y la capa de persistencia (base de datos). En la capa de presentación, se utiliza **Thymeleaf** para renderizar las vistas del sistema de ventas, mientras que en la capa de lógica de negocio, **Spring Boot** se encarga de procesar las transacciones y coordinar las operaciones entre las distintas partes del sistema.

Para la capa de persistencia, se ha optado por **MySQL**, una base de datos relacional que permite almacenar de manera eficiente el inventario, ventas, usuarios y otros datos clave del sistema. La arquitectura sigue el patrón **"Clean Architecture"**, que asegura una clara separación de responsabilidades, facilitando la escalabilidad y mantenibilidad del sistema.

Este enfoque modular garantiza que el sistema sea flexible y permita futuras expansiones, como la incorporación de nuevas funcionalidades o la integración con sistemas externos para generar reportes avanzados o análisis de ventas.

Selección de tecnologías

La selección de tecnologías para el desarrollo de este sistema de punto de venta se realizó en función de los objetivos y requisitos específicos del cliente, priorizando la eficiencia, escalabilidad y facilidad de uso.

Para el backend, se eligió **Spring Boot** debido a su robustez y su capacidad para gestionar transacciones complejas de manera eficiente, además de ofrecer un marco altamente escalable y adaptable. En el front-end, se optó por **Thymeleaf** y **CSS Bootstrap** , lo que permite desarrollar una interfaz de usuario responsiva, intuitiva y accesible desde cualquier dispositivo.

Como gestor de bases de datos, seleccionó **MySQL** gracias a su rendimiento, confiabilidad y capacidad para manejar grandes volúmenes de datos. Su integración con **Spring Data JPA** simplifica la gestión de consultas y actualizaciones en la base de datos, optimizando el flujo de desarrollo.

Asimismo, se incorporó **JUnit** para garantizar la calidad del software mediante pruebas unitarias efectivas, asegurando que cada componente funcione correctamente. Finalmente, **Maven** fue adoptada como herramienta de construcción, lo que facilita la gestión de dependencias, el control del ciclo de vida del proyecto y la producción de código limpio y eficiente.

Esta combinación de tecnologías proporciona una base sólida para un sistema de punto de venta funcional, escalable y fácilmente mantenible.

Capítulo 6 Implantación

En este capítulo se detallará la implantación del sistema de punto de venta para una farmacia, abarcando las capas que corresponden tanto al backend como al frontend. Además, se abordará la estructura del proyecto, la configuración de Spring Boot y las herramientas utilizadas para la implementación del sistema.

Desarrollo Back-end

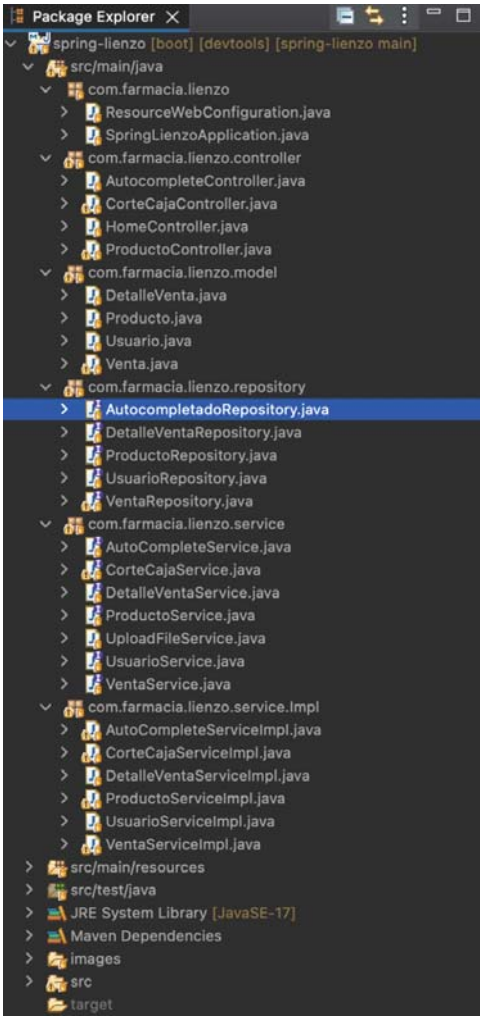
Para el desarrollo de la lógica de negocio del sistema, se ha empleado el entorno de desarrollo integrado (IDE) **Spring Tools Suite (STS)**. La elección de este IDE se basó en su integración nativa con proyectos Spring Boot y su capacidad para manejar de manera eficiente la configuración de proyectos basados en **Maven**. STS simplifica el proceso de desarrollo al ofrecer herramientas avanzadas para la depuración y la gestión de dependencias.

Estructura del proyecto

En la Imagen 6.1.1 se puede observar la estructura del proyecto, donde los paquetes se organizan según una arquitectura modular que sigue los principios de la arquitectura de capas y limpia, dividiéndose en varias categorías principales: controladores, servicios, repositorios, configuración, ventas, usuarios y productos. Esta organización facilita la identificación rápida de clases y componentes relacionados con aspectos específicos del sistema, lo que mejora la mantenibilidad y escalabilidad del sistema a largo plazo.

Dentro de los paquetes de venta e inventario, se sigue una arquitectura modular basada en los principios de arquitectura limpia. Esta estructura modular permite claridad y eficiencia en el diseño del sistema de punto de venta. El paquete de venta contiene los servicios implementados, como la gestión de transacciones, mientras que el paquete de inventario contiene el modelo y las interfaces tanto para la capa de servicios como para la infraestructura. Esta última capa incluye los servicios RESTful y las interacciones con la base de datos.

La estructura de este proyecto está organizada para soportar escalabilidad y mantenimiento, agrupando las clases según su funcionalidad dentro de la arquitectura limpia.

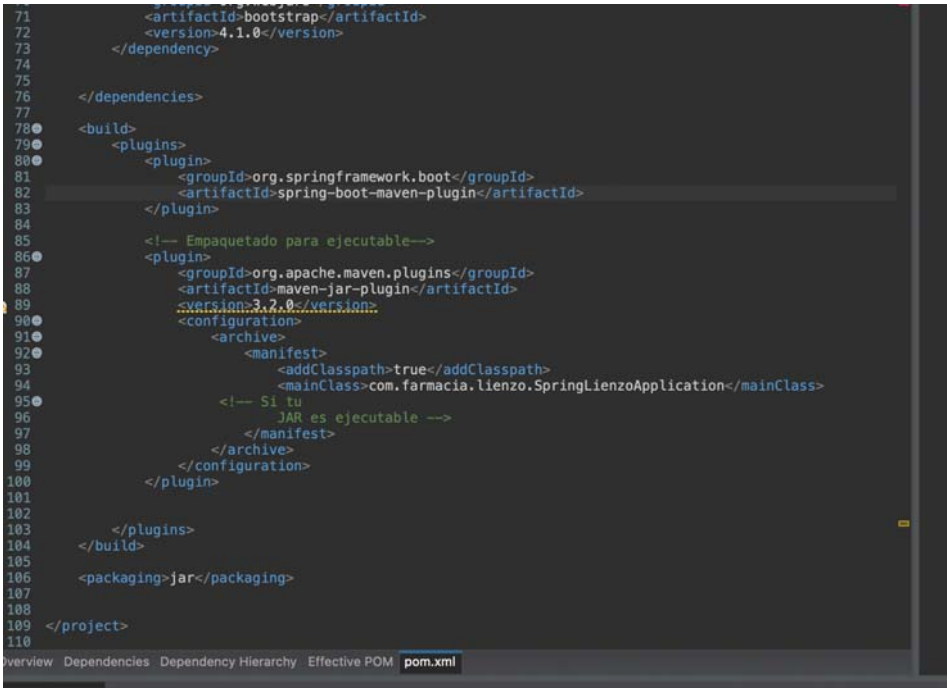


(Imagen 6.1.1: Estructura del proyecto)

Configuración del proyecto

La configuración principal se encuentra en el archivo pom.xml, que define las dependencias y las configuraciones de compilación y ejecución del proyecto. En la Imagen 6.1.2: Sección de dependencias y plugins del archivo pom.xml, se observan las dependencias y plugins más importantes:

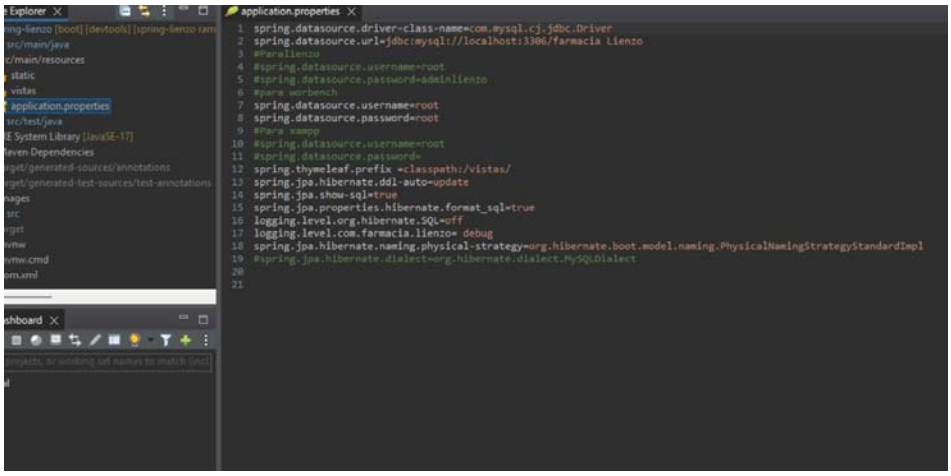
- org.springframework.boot: Proporciona las herramientas necesarias para crear, empaquetar y ejecutar el proyecto con Spring Boot.
- org.springframework.boot: Proporciona soporte para la integración con JPA para el manejo de la persistencia de datos.
- org.springframework.boot: Permite añadir funcionalidades de seguridad como autenticación y autorización.
- org.springframework.boot: Facilita el desarrollo de controladores RESTful para la interacción con la capa de presentación.
- org.projectlombok: Simplifica el desarrollo eliminando la necesidad de escribir código repetitivo para getters, setters y constructores.
- org.springframework.security: Para realizar pruebas de autenticación y seguridad.
- io.jsonwebtoken: Para la gestión de autenticación basada en tokens JWT.
- org.springframework.boot: Proporciona soporte para la integración de Spring Data y JPA (Java Persistence API).
- org.springframework.boot: Aporta funcionalidades de seguridad como autenticación y protección contra ataques comunes.
- org.springframework.boot: Soporta la integración de Thymeleaf para el desarrollo del front-end.
- org.springframework.boot: Facilita el desarrollo de aplicaciones web usando Spring MVC y API RESTful.
- org.projectlombok: Simplifica la creación de clases Java mediante anotaciones.



(Imagen 6.1.2: Sección de dependencias y plugins)

Configuración de Spring Boot

En el **archivo application.properties**, que se muestra en la **Imagen 6.1.3: Configuraciones del archivo application.properties**, se definen las propiedades del proyecto. Este archivo permite especificar el entorno de ejecución, las rutas de los servicios y otras configuraciones esenciales. Por ejemplo, se establece el perfil de desarrollo o producción con la propiedad spring.profiles.active. También se definen las propiedades de conexión a la base de datos y la configuración de JWT (JSON Web Token) para la seguridad.



(**Imagen 6.1.3: Configuraciones del archivo application.properties)**

Pruebas unitarias con JUnit 5:

Además, el proyecto incluye JUnit 5 como dependencia clave para la realización de pruebas unitarias. JUnit 5 permite probar de forma aislada las funcionalidades de los distintos módulos del sistema, como los controladores REST, los servicios de negocio y las operaciones de persistencia. A través de esta herramienta, se realizan pruebas para validar que los métodos del sistema se comporten como se espera bajo diferentes condiciones, garantizando así la estabilidad y confiabilidad del sistema.

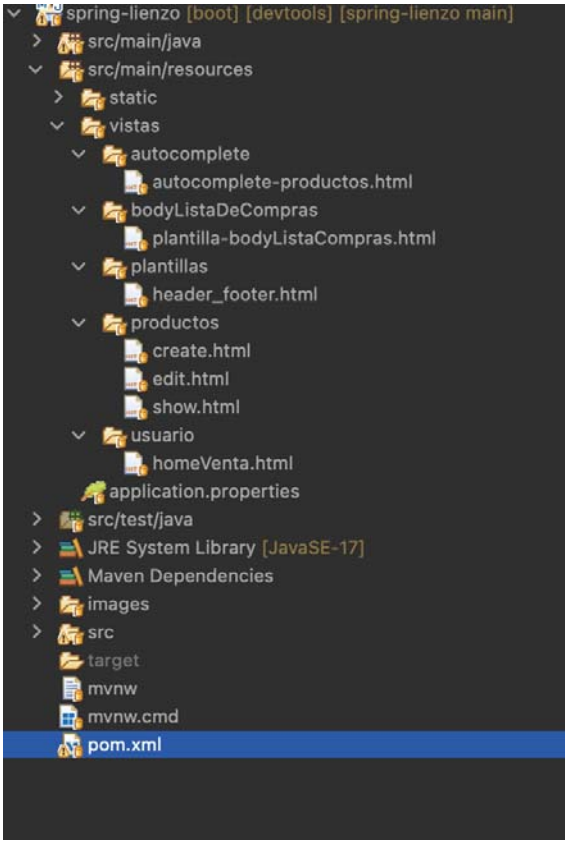
Las pruebas también integran Mockito para la simulación de dependencias y comportamientos en los componentes, facilitando la creación de entornos controlados para probar situaciones específicas del sistema sin necesidad de una base de datos real o conexiones externas.

Desarrollo front-end

Para el desarrollo del front-end se ha optado por el uso de **Thymeleaf**, un motor de plantillas para Java que se integra perfectamente con Spring Boot. Thymeleaf permite crear vistas dinámicas utilizando un enfoque basado en plantillas HTML, lo que facilita la creación de interfaces de usuario para el sistema de punto de venta.

La **estructura de los paquetes en el front-end** se divide en componentes organizados según la arquitectura de capas, siguiendo los principios de **MVC (Modelo-Vista-Controlador)**, como se muestra en la **Imagen 6.4: Estructura de paquetes del front-end**. El paquete principal de vistas contiene las plantillas HTML generadas por Thymeleaf, mientras que en los controladores se gestionan las solicitudes HTTP y la lógica de presentación.

Thymeleaf es ideal para la construcción de aplicaciones web basadas en servidor, ya que permite una integración sencilla con los modelos del back-end, garantizando que los datos presentados a los usuarios estén sincronizados con la lógica de negocio implementada en las capas de servicio.



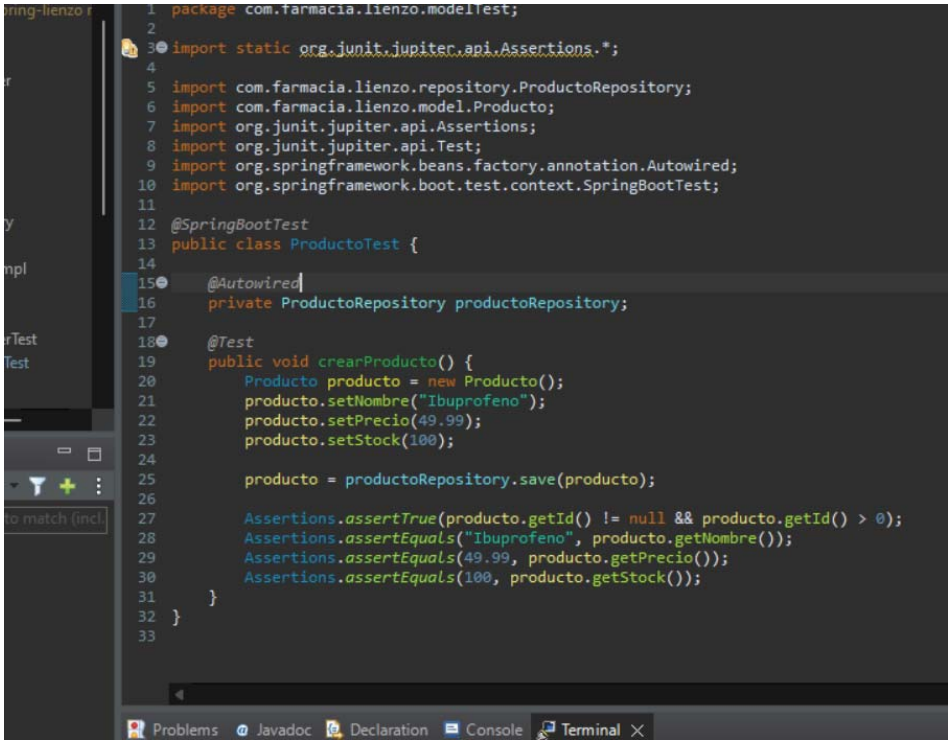
(**Imagen 6.4: Estructura de paquetes del front-end)**

Capítulo 7 Pruebas

Las pruebas son fundamentales en el desarrollo de software para asegurar la calidad y prevenir posibles errores. Para el sistema de punto de venta desarrollado para la farmacia, se llevaron a cabo pruebas utilizando JUnit 5 para pruebas unitarias y Spring Framework MockMvc para pruebas de integración.

Pruebas Unitarias

Las pruebas unitarias se centran en verificar la lógica interna de los objetos y sus métodos, asegurando que cada componente individual funcione correctamente. En la Figura 7.1 se muestra un ejemplo de una prueba unitaria que valida la creación de un producto en el sistema de inventario. Este ejemplo comprueba que el producto, una vez guardado, tiene un identificador asignado y que sus atributos esenciales, como el nombre, el precio y la cantidad, cumplen con los requisitos establecidos. Se utiliza assertTrue para confirmar que el identificador sea mayor que cero, lo cual valida que el producto se ha almacenado correctamente.

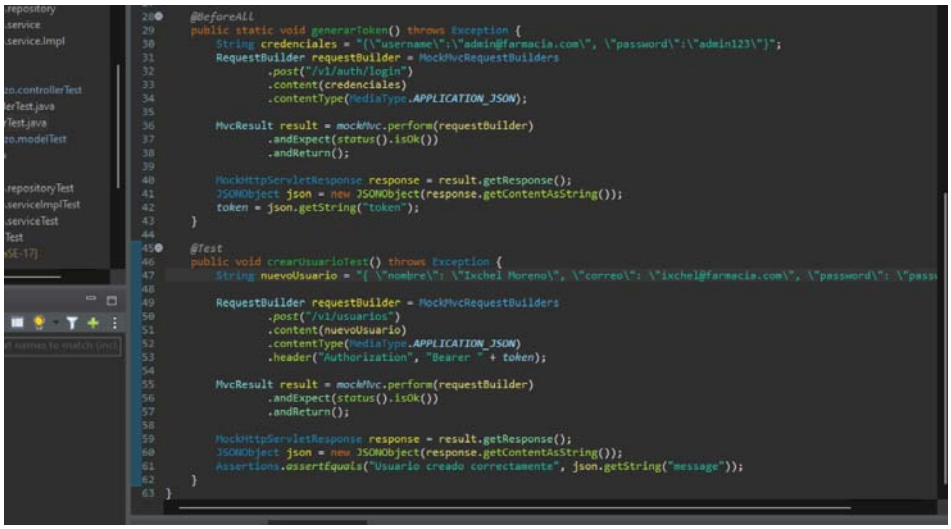


(Imagen 7.1: Prueba unitaria para la creación de un producto)

Pruebas de Integración

Las pruebas de integración se utilizan para verificar que los diferentes componentes del sistema funcionen correctamente en conjunto. En este sistema, se realizaron pruebas de integración en los controladores que exponen los servicios REST para el consumo de la API. En la Imagen 7.2, se muestra una prueba de integración que valida el funcionamiento del controlador de UsuarioController, el cual permite la creación y autenticación de usuarios en el sistema de punto de venta.

Esta prueba se inicia configurando un token válido mediante una solicitud POST de autenticación, necesaria para acceder a los servicios protegidos. Posteriormente, se envía una solicitud POST al controlador UsuarioController para crear un nuevo usuario en el sistema, comprobando que la respuesta sea correcta y que el usuario se haya creado con éxito en la base de datos.



(Imagen 7.2: Prueba de integración para la creación de un usuario)

Capítulo 8 Estimación de Costos y Diagrama de Gantt

Este capítulo presenta una estimación detallada del costo del proyecto de desarrollo de un sistema de punto de venta para una farmacia, considerando el tiempo de trabajo, los roles involucrados y los salarios promedio. Además, se presenta un cronograma visual a través de un diagrama de Gantt.

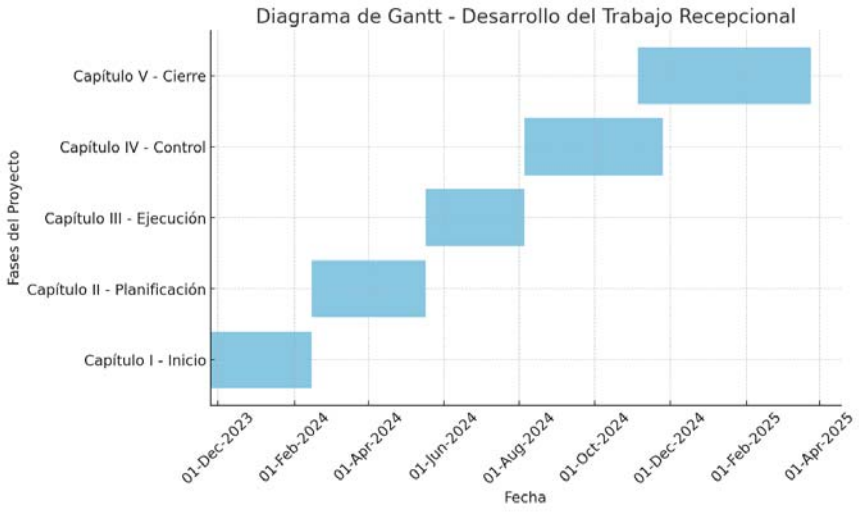
Estimación de Costos por Fase

Fase	Duración	Rol Principal	Horas estimadas	Costo estimado (MXN)	Costo estimado (USD)
Inicio	2 meses	Analista / Líder de Proyecto	320	$38,400	$2,240
Planificación	2 meses	Líder de Proyecto	320	$38,400	$2,240
Ejecución	3 meses	Dev Backend/Frontend	480	$72,000	$4,200
Control	3 meses	QA / Project Manager	480	$57,600	$3,360
Cierre	2 meses	Todos (mínimo esfuerzo)	320	$48,000	$2,800

El total estimado del proyecto es de aproximadamente $254,400 MXN o $14,840 USD. Este costo incluye todas las etapas clave del proyecto desde la planificación hasta la entrega.

Diagrama de Gantt

En la Imagen 8.2 se presenta el diagrama de Gantt, el cual representa la duración y el orden de ejecución de cada una de las fases del proyecto, desde diciembre de 2023 hasta abril de 2025.



(Imagen 8.2: Diagrama de Gantt)

Este análisis de costos y cronograma permite visualizar de forma clara el esfuerzo requerido para llevar a cabo el desarrollo del sistema POS a la medida. La planificación detallada en fases asegura una gestión eficiente del tiempo y los recursos humanos, fundamentales para el éxito del proyecto.

Capítulo 9 Conclusiones

El desarrollo de un sistema de software a medida, como el punto de venta para una farmacia, es un proceso multifacético que requiere un enfoque estructurado y riguroso de ingeniería de software. En este proyecto, se ha adoptado una metodología orientada a proyectos y basada en el Proceso Unificado para guiar el análisis, diseño e implementación del sistema. A través de la aplicación de buenas prácticas y la combinación de habilidades teóricas y prácticas, el sistema desarrollado no solo satisface las necesidades y expectativas del cliente, sino que también mantiene una alta calidad y seguridad en sus procesos.

La ingeniería de software ha sido crucial en cada fase, desde la recolección y análisis de requisitos hasta el diseño y desarrollo de una arquitectura robusta. La implementación en Spring Boot y Maven, junto con herramientas como JUnit para las pruebas unitarias y Spring Framework MockMvc para las pruebas de integración, ha

permitido una integración eficaz de todas las funcionalidades del sistema, asegurando su desempeño óptimo y su correcta interacción con la aplicación móvil asociada.

En resumen, este proyecto reafirma la importancia de contar con un equipo multidisciplinario, donde cada miembro aporta conocimientos y habilidades específicas que complementan el desarrollo del software. La metodología adoptada y las herramientas utilizadas han sido fundamentales para construir un sistema que no solo cumple con las especificaciones iniciales, sino que también permite una futura escalabilidad y mantenimiento del mismo.

9.1 Mejoras futuras el sistema ha sido desarrollado considerando su alcance inicial y los requisitos establecidos por el cliente, logrando una implementación exitosa y funcional. No obstante, se encuentra diseñado para facilitar futuras actualizaciones y escalabilidad. Esto significa que, en caso de que el cliente requiera mejoras, ampliaciones de funcionalidad o integraciones adicionales, estas podrán realizarse sin inconvenientes.

Entre las posibles mejoras futuras se incluirán recomendaciones específicas sobre escalabilidad, optimización de rendimiento y nuevas funcionalidades que puedan adaptarse a las necesidades cambiantes del negocio. De esta manera, el sistema está preparado para evolucionar junto con las demandas del cliente, asegurando su vigencia y efectividad a largo plazo.

10 Cronograma

25 de noviembre 2023 – 15 de febrero 2024	15 de febrero 2023 – 17 de mayo 2024.	17 de mayo 2024 – 5 de agosto 2024.	5 de agosto 2024 – 25 de noviembre 2024.	5 de noviembre 2024 – 25 de marzo 2025.
Capítulo I	Capítulo II	Capítulo III	Capítulo IV	Capítulo V
Inicio	Planificación	Ejecución	Control	Cierre
1 Contextualización del proyecto 2 Objetivos y alcance 3 Justificación de la elección del enfoque de gestión de proyectos 4 Visión del proyecto	1 Desglose del trabajo y asignación de recursos 2 Planificación temporal y elaboración de cronogramas 3 Asignación de Recursos	1 Análisis 1.1 Diagrama de objetos 1.2 Diagrama de secuencia de los casos más relevantes 1.3 Diagrama de clases 2 Diseño 2.1 Determinar el diseño del sistema 2.2 Aplicar principio de diseño	1 Monitoreo del avance del proyecto 2 Gestión de riesgos 3 Ajustes en el plan según sea necesario	1 Entrega del producto 2 Evaluación del proyecto 3 Lecciones aprendidas

5 Identificación de requisitos y definición del proyecto 6 Análisis de viabilidad 7 Gestión de Riesgos 8 Glosario	4 Calidad del Software 5 Gestión de Cambios 6 Presupuesto	2.3 Diagrama de capas 2.4 Diseño de modelo ER 3 Programación 4 Pruebas 4.1 Diseño de casos de prueba unitarias, de integración, de aceptación. 4.2 Ejecución y posible corrección de Pruebas unitarias 4.3 Ejecución y posible corrección de Pruebas de integración 4.4 Ejecución y posible corrección de Pruebas de integración 5 Implementación 5.1 Configuración de Spring Boot 5.2 Capa de Datos y Persistencia 5.3 Capa de servicios 5.4 Capa de controlador y vistas 5.5 Cambios de las posibles vistas 5.6 Gestión de riesgos 6 Mantenimiento		

11 Glosario

1. **API (Application Programming Interface):** Conjunto de definiciones y protocolos que permite la comunicación entre diferentes aplicaciones de software.
2. **Backend:** Parte del sistema que gestiona la lógica del negocio y la interacción con la base de datos.
3. **Base de Datos Relacional:** Sistema de almacenamiento estructurado basado en tablas que permite gestionar grandes volúmenes de información.
4. **Cloud Computing:** Modelo de computación que permite el acceso a recursos de hardware y software de manera remota.
5. **Framework:** Conjunto de herramientas y bibliotecas diseñadas para facilitar el desarrollo de software.

6. **JSON (JavaScript Object Notation):** Formato ligero de intercambio de datos basado en texto.
7. **MVC (Modelo-Vista-Controlador):** Patrón de arquitectura de software que separa la lógica del negocio, la interfaz de usuario y el control de flujo.
8. **POS (Point of Sale):** Sistema de software y hardware utilizado para gestionar las ventas y transacciones en un comercio.
9. **Spring Boot:** Framework de desarrollo en Java que simplifica la creación de aplicaciones empresariales.
10. **Thymeleaf:** Motor de plantillas para Java que permite integrar lógica de backend con HTML dinámico.
11. **SOLID:** Acrónimo que representa cinco principios de diseño de software orientado a objetos destinados a mejorar la calidad, mantenibilidad y escalabilidad del código. Estos principios son:

    S (Single Responsibility Principle): Principio de responsabilidad única.

    O (Open/Closed Principle): Principio de abierto/cerrado.

    L (Liskov Substitution Principle): Principio de sustitución de Liskov.

    I (Interface Segregation Principle): Principio de segregación de interfaces.

    D (Dependency Inversion Principle): Principio de inversión de dependencias.

    Estos principios ayudan a diseñar sistemas modulares y de bajo acoplamiento, facilitando la gestión de errores y la incorporación de cambios.

12 Referencias

- Candel, J. M. O. (2020). *Desarrollo seguro en ingeniería de software*. Marcombo.
- Candel, J. (2020). *Fundamentos de la seguridad informática*. Editorial Tecnológica.
- Chopra, S., & Meindl, P. (2015). *Supply chain management: Strategy, planning, and operation* (6th ed.). Pearson.
- Escalona, M. J., & Koch, N. (2002). *Ingeniería de requisitos en aplicaciones para la web–un estudio comparativo*. Universidad de Sevilla.
- Fling, B. (2009). *Mobile design and development*. O'Reilly Media.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2003). *Patrones de diseño*. Pearson.
- Gutiérrez Faraoni, F. J. (2015). *Desarrollo de una aplicación web con Spring Framework para un gestor de un recetario*.

- Hernández, L. M. A., Romero, V. A. P., González, S. A. S., & Rodríguez, J. A. V. (2021). *Arquitectura REST para el desarrollo de aplicaciones web empresariales*. Revista Electrónica sobre Tecnología, Educación y Sociedad, 8(15).
- HealthIT.gov. (n.d.). *What is an electronic health record (EHR)?* Recuperado de: https://www.healthit.gov/faq/what-electronic-health-record-ehr
- Jacobson, I., Booch, G., & Rumbaugh, J. (2000). *El proceso unificado de desarrollo de software*. Pearson Educación.
- Jacobson, I., Booch, G., & Rumbaugh, J. (2000). *The Unified Software Development Process*. Addison-Wesley.
- Kumar, V., & Reinartz, W. (2018). *Customer relationship management: Concept, strategy, and tools* (3rd ed.). Springer.
- Martin, R. C. (2018). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson.
- Oriols, M. B., & Gutierrez, J. A. G. (2018). *El gran libro de Angular*. Alfaomega.
- Oriols, A., & Gutiérrez, J. A. G. (2018). *El gran libro de Angular*. Editorial Digital.
- Pressman, R. S. (2002). *Ingeniería de software: Un enfoque práctico*. McGraw-Hill.
- Schwaber, K., & Sutherland, J. (2017). *The Scrum Guide*. Scrum.org.
- Sommerville, I. (2011). *Ingeniería de software*. Pearson.
- Sommerville, I. (2011). *Software Engineering* (9th ed.). Addison-Wesley.
- Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.
- Stevenson, W. J. (2015). *Operations management* (12th ed.). McGraw-Hill Education.
- Whitman, M. E., & Mattord, H. J. (2018). *Principles of information security* (6th ed.). Cengage Learning.

Sitios Web

- *Gradle vs Maven Comparison*. Recuperado el 1 de julio de 2023, de https://gradle.org/maven-vs-gradle/
- *Spring Framework Overview*. Recuperado el 1 de julio de 2023, de https://docs.spring.io/spring-framework/reference/overview.html
- *Angular Documentation*. Recuperado el 1 de julio de 2023, de https://docs.angular.lat/