PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

INGENIERÍA DE SISTEMAS DE INFORMACIÓN

TRABAJO DE TITULACIÓN

Desarrollo de una aplicación web de Facturación a nivel prototipo.

Caso de estudio: Brother Corp

AUTOR:

Bolivar Gabriel Corella Rosales

QUITO DM, ENERO DE 2025

AGRADECIMIENTO

Quisiera agradecer primeramente a mi familia, por siempre estar conmigo y darme todas las facilidades para seguir a delante con la carrera que escogí, por siempre apoyarme y ayudarme durante todo este proceso, a mi mamá por darme la fuerza y el carácter para siempre ser un señor, ante todo, a mi papá por guiarme y ser un ejemplo para mí.

De igual manera a mis abuelitos por siempre apoyarme y ser un pilar importante durante todo el proceso, gracias por el apoyo incondicional.

A mis amigos que estuvieron junto a mí, gracias por compartir esta etapa de mi vida también a los amigos me guiaron desde el cielo.

Por último, a todos los docentes y equipo de la facultad de ingeniera por brindarme su ayuda para poder cumplir la meta y graduarme como un profesional.

Abstract:

This research project focused on the design and implementation of a web-based electronic billing system prototype for Brother Corp, an Ecuadorian company specialized in business management solutions. The study addressed the limitations of the organization's legacy desktop software by developing a modern, accessible platform that complied with the technical requirements established by Ecuador's (Servicio de Rentas Internas) (SRI).

The development process employed Extreme Programming (XP) methodology, which proved particularly effective for this type of project. This agile approach enabled continuous adaptation to requirements through short development cycles, constant testing, and frequent client feedback. The research team implemented key XP practices including test-driven development, continuous integration, and regular refactoring, which contributed significantly to the system's reliability and maintainability.

For the technical implementation, the project team selected a three-tier architecture that separated presentation, business logic, and data management layers. The backend was developed using C# with ASP.NET framework, while SQL Server served as the database management system. The frontend interface was developed with Flutter, ensuring cross platform compatibility and responsive design.

The resulting prototype successfully incorporated all core billing functionalities, including client management, product catalog administration, and electronic invoice generation with proper XML formatting as mandated by SRI regulations. The system also featured user access controls, data validation mechanisms, and basic reporting capabilities. Performance testing demonstrated that the application could handle typical operational loads with response times under three seconds for most transactions.

The research concluded that the transition from legacy systems to modern web applications can yield significant operational benefits for small and medium enterprises. These include improved accessibility, easier maintenance, and better compliance with regulatory requirements. The study also highlighted the importance of iterative development and continuous client involvement in software projects.

Índice

Abstract: I

Índice de Ilustraciones IV

Capítulo I: Introducción 1

1.1 Tema 1

1.2 Justificación 2

1.3 Planteamiento del problema 3

1.4 Objetivos 4

1.4.1 Objetivo General 4

1.4.2 Objetivos Específicos 4

1.5 Alcance 4

Capítulo II: Fundamentación Teórica 6

2.1 Marco teórico y conceptual 6

2.1.1 Estudio del arte del proceso de facturación 6

2.1.2 Desarrollo de conceptos teóricos fundamentales para el proyecto 7

2.1.3 Definiciones clave relacionadas con el desarrollo del sistema 9

2.2 Marco técnico 10

2.2.1 Metodología Ágil de desarrollo (Extreme Programming (XP) vs Scrum) 11

2.2.2 Lenguaje de programación y Framework 18

2.2.3 IDE de desarrollo 25

2.2.4 Base de datos 26

Capítulo III: Requerimientos para el desarrollo del Sistema de Facturación Electrónica 30

3.1 Casos de Uso 30

3.1.1 Caso de uso F1: Gestión Cliente 32

3.1.2 Caso de uso F2: Gestión Producto 39

3.1.3 Caso de uso F3: Gestión Empresa 44

3.1.4 Caso de uso F4: Gestión Detalle 49

3.1.5 Caso de uso F5: Gestión Cabecera 55

3.1.6 Caso de uso F6: Gestión Local 60

3.1.7 Caso de uso F7: Gestión Comprobante 65

3.2 Requerimientos Funcionales y no funcionales 70

3.2.1 Requerimientos Funcionales 71

3.2.2 Requerimientos No Funcionales 73

3.3 Modelo de la Base de Datos 76

3.3.1 UML 76

3.3.2 Modelo Físico PowerDesigner 77

3.4 Implementación completa de XP 78
3.4.1 Prácticas Fundamentales de XP Aplicadas en el Proyecto 78
3.4.2 Bitácora Semanal del Desarrollo (12 semanas) 85
Capítulo IV: Diseño e Implementación del sistema de Facturación. 98
Capítulo V: Conclusiones y Recomendaciones 109
5.1 CONCLUSIONES 109
5.2 RECOMENDACIONES 110
VI: Referencias bibliográficas 111
VII: Anexos: 113
7.1 Acta de Reuniones 113
7.2 Acta de Anexos 114
7.3 Acta de Aceptación 116

Índice de Ilustraciones

ILUSTRACIÓN 1 CASO DE USO GENERAL 31
ILUSTRACIÓN 2 CASO DE USO CLIENTE 32
ILUSTRACIÓN 3 CASO DE USO PRODUCTOS 39
ILUSTRACIÓN 4 CASO DE USO EMPRESA 44
ILUSTRACIÓN 5 CASO DE USO DETALLES 49
ILUSTRACIÓN 6 CASO DE USO CABECERA 55
ILUSTRACIÓN 7 CASO DE USO LOCAL 60
ILUSTRACIÓN 8 CASO DE USO COMPROBANTE 65
ILUSTRACIÓN 9 UML DE LA BDD 76
ILUSTRACIÓN 10 MODELO LOGICO DE LA BDD 77
ILUSTRACIÓN 11 END POINTS DEL BACKEND 98
ILUSTRACIÓN 12 RUTAS Y PARÁMETROS DE ENTRADA CABECERA Y CLIENTE 99
ILUSTRACIÓN 13 RUTAS Y PARÁMETROS DE ENTRADA DETALLE Y EMPRESA 99
ILUSTRACIÓN 14 RUTAS Y PARÁMETROS DE ENTRADA LOCAL Y PRODUCTO 100
ILUSTRACIÓN 15 RUTAS Y PARÁMETROS DE ENTRADA TIPOCOMPROBANTE Y USUARIO 100
ILUSTRACIÓN 16 LOGIN 101
ILUSTRACIÓN 17 CREACIÓN DE FACTURA 101
ILUSTRACIÓN 18 PANTALLA PRINCIPAL 102
ILUSTRACIÓN 19 MENÚ TIPO HAMBURGUESA 102
ILUSTRACIÓN 20 LISTADO DE CLIENTES 103
ILUSTRACIÓN 21 CREACIÓN DEL CLIENTE 103
ILUSTRACIÓN 22 LISTADO DE PRODUCTOS 104
ILUSTRACIÓN 23 CREACIÓN DE PRODUCTO 104
ILUSTRACIÓN 24 LISTADO DE FACTURAS 105
ILUSTRACIÓN 25 DATOS DE LA EMPRESA 105
ILUSTRACIÓN 26 CREACIÓN DE EMPRESA 106
ILUSTRACIÓN 27 LISTADO DE LOCALES 106
ILUSTRACIÓN 28 CREACIÓN DE LOCAL 107
ILUSTRACIÓN 29 LISTADO DE USUARIOS 107
ILUSTRACIÓN 30 CREACIÓN DE USUARIOS 108

Capítulo I: Introducción

En la actualidad, la transformación digital se ha convertido en un factor clave para el crecimiento y la sostenibilidad de las empresas. La necesidad de contar con herramientas tecnológicas eficientes, accesibles y seguras ha impulsado a muchas organizaciones a migrar sus sistemas tradicionales hacia plataformas web. En este contexto, la facturación electrónica es de suma importancia, no solo como una exigencia legal impuesta por el Servicio de Rentas Internas (SRI) en Ecuador, sino también como una oportunidad para mejorar los procesos en contabilidad, reducir errores y facilitar el control financiero.

Brother Corp, empresa ecuatoriana con más de dos décadas de experiencia en el desarrollo de soluciones ERP de escritorio, enfrenta hoy el desafío de actualizar su oferta tecnológica para responder a las nuevas demandas del mercado. Las limitaciones de su sistema actual entre ellas, la falta de acceso remoto, la dependencia del hardware del cliente y las dificultades para integrar nuevas funcionalidades han generado la necesidad de desarrollar una solución moderna, escalable y multiplataforma.

Este proyecto de titulación tiene como objetivo el diseño y desarrollo de un prototipo de aplicación web para la gestión de facturación electrónica, orientado a mejorar la eficiencia operativa de Brother Corp y preparar el camino hacia la migración completa de su sistema ERP a la web. En los apartados siguientes se abordará el contexto del problema, la justificación de la propuesta, los objetivos planteados, el alcance del proyecto y las consideraciones técnicas fundamentales.

1.1 Tema

Desarrollo de una aplicación web de Facturación a nivel prototipo. Caso de estudio: Brother Corp.

1.2 Justificación

La digitalización de los procesos empresariales es hoy en día fundamental para optimizar la eficiencia operativa y la competitividad en el mercado ecuatoriano. Dentro de estos procesos, la facturación juega un papel clave, ya que está directamente relacionada con el cumplimiento de obligaciones tributarias, la gestión administrativa y la trazabilidad de las transacciones comerciales.

La facturación electrónica es un método alternativo para emitir comprobantes de venta, retención y documentos complementarios, cumpliendo con las normas legales y reglamentarias establecidas por el SRI. Este sistema asegura la autenticidad del origen y la integridad del contenido de cada comprobante, gracias a la inclusión de la firma electrónica del emisor (SRI, 2025).

Brother Corp es una empresa ecuatoriana con más de 25 años de trayectoria en el desarrollo e implementación de soluciones ERP de escritorio, dirigidas a pequeñas y medianas empresas. La organización cuenta con una estructura operativa que incluye un equipo de desarrollo, soporte técnico y atención al cliente. Sus soluciones han sido implementadas en entidades como Kriollo por Ikaro, PagueYa, Soportec, GAD Parroquia Picaihua, Incoprov entre otras.

Actualmente, la empresa trabaja con un sistema de escritorio que presenta diversas limitaciones tecnológicas: falta de acceso remoto, dificultades en la actualización del software en equipos cliente, y restricciones en términos de escalabilidad y seguridad. Estas limitaciones han afectado su competitividad en un mercado que demanda cada vez más soluciones web, accesibles desde múltiples dispositivos y con tiempos de respuesta ágiles.

Diversos clientes de Brother Corp han manifestado su interés en contar con soluciones basadas en la web, que ofrezcan mayor flexibilidad, acceso multiplataforma y cumplimiento automático con los requisitos del Servicio de Rentas Internas (SRI), incluyendo la facturación electrónica. Ante este panorama, se propone el desarrollo de un prototipo de aplicación web para la gestión de facturación electrónica, como primer paso hacia la modernización de sus sistemas actuales.

Se empleó una arquitectura de tres niveles fundamentada en Cliente-Servidor, lo que facilitó la segmentación del sistema en tres niveles claramente delimitados: la capa de presentación, la capa de lógica empresarial y la capa de acceso a datos. Esta separación simplificó la conservación del sistema, dado que cada capa tuvo la posibilidad de cambiar de manera autónoma sin impactar directamente en las demás. Además, se incrementó la escalabilidad y el

reaprovechamiento del código, en particular en la lógica empresarial, que pudo ser utilizada en diversas interfaces. Además, se mejoró el desempeño global al impedir que los clientes ingresaran directamente a la base de datos, concentrando el procesamiento en el servidor de aplicaciones.

1.3 Planteamiento del problema

La evolución tecnológica y la normativa fiscal ecuatoriana han impulsado a las empresas a modernizar sus sistemas de gestión, con el fin de mejorar la eficiencia operativa, cumplir con los requisitos legales y ofrecer mejores servicios a sus clientes. Entre estos avances, la facturación electrónica se ha consolidado como una herramienta imprescindible.

Brother Corp, especializada en soluciones ERP de escritorio, enfrenta actualmente varios desafíos:

- Dificultades para ofrecer actualizaciones automáticas a sus clientes.
- Lentitud en la interacción entre interfaces.
- Problemas de compatibilidad con sistemas operativos modernos.
- Limitaciones de acceso remoto, que impiden una gestión flexible y en tiempo real.

Además, la ausencia de una solución de facturación electrónica basada en la web ha generado desventajas competitivas, como la pérdida de contratos con empresas que buscan soluciones modernas, accesibles desde dispositivos móviles y con integración directa al SRI.

Si bien la empresa cuenta con un equipo técnico calificado, su infraestructura actual requiere ser actualizada para responder a estas nuevas demandas del mercado. Por esta razón, se plantea el desarrollo de un prototipo de aplicación web de facturación electrónica, accesible desde cualquier navegador, con diseño responsivo y arquitectura escalable, que permita sentar las bases para la migración completa del ERP a una plataforma web.

Este prototipo utilizó el gestor de base de datos SQL Server, permitiendo un control centralizado y seguro de la información. De esta forma, se busca no solo cumplir con la normativa vigente del SRI, sino también mejorar la oferta de productos y servicios de Brother Corp frente a sus clientes actuales y potenciales.

1.4 Objetivos

1.4.1 Objetivo General

Desarrollar una aplicación web de facturación a nivel prototipo. Caso de estudio: Brother Corp

1.4.2 Objetivos Específicos

1 Analizar la situación actual de Brother Corp y levantar los requisitos funcionales y no funcionales del módulo de facturación electrónica.

2 Seleccionar una metodología de desarrollo ágil adecuada para la gestión eficiente del proyecto.

3 Diseñar la arquitectura de la aplicación, incluyendo la interfaz de usuario y el modelo de base de datos, basándose en los requerimientos levantados.

4 Implementar un prototipo funcional del módulo de facturación electrónica, con capacidad de emitir facturas en formato XML conforme a los lineamientos del SRI.

5 Validar el correcto funcionamiento del prototipo mediante pruebas de funcionalidad y usabilidad.

6 Documentar todo el proceso de desarrollo, resultados obtenidos y recomendaciones para futuras etapas de implementación y escalado del sistema.

1.5 Alcance

El presente proyecto culminó con el desarrollo de un prototipo funcional de aplicación web para la gestión de facturación electrónica en la empresa Brother Corp. Este prototipo fue desarrollado en un entorno de pruebas y tuvo como objetivo validar la viabilidad técnica y funcional del sistema en un entorno web. El prototipo incluyó:

- Levantamiento y análisis de los requisitos funcionales y técnicos del proceso de facturación.
- Diseño de la interfaz de usuario con enfoque responsivo para su uso en computadoras, tablets y dispositivos móviles.
- Diseño y modelado de la base de datos, alojado en SQL Server.

- Desarrollo de funciones CRUD (crear, leer, actualizar, eliminar) para productos, clientes y empleados.
- Generación de archivos XML de facturas, conformes al esquema requerido por el SRI, listos para su validación.
- Pruebas internas de funcionalidad, seguridad básica y usabilidad del prototipo.
- Documentación completa del proceso de desarrollo y sugerencias para la integración futura con el SRI.

El alcance del proyecto comprendió únicamente el desarrollo de un prototipo funcional en un entorno de pruebas, sin llegar a implementarse en un ambiente productivo. Se enfocó en validar los aspectos técnicos y operativos fundamentales del sistema, excluyendo características más avanzadas como la firma electrónica, la integración directa con servicios del SRI. A pesar de estas limitaciones, el prototipo proporcionó una base sólida sobre la cual se podrá construir una versión final lista para su despliegue en producción.

Capítulo II: Fundamentación Teórica

El presente capítulo expone los fundamentos teóricos, conceptuales y técnicos que respaldan el desarrollo del prototipo de facturación electrónica para la empresa Brother Corp. Se abordan los principales conceptos relacionados con los sistemas de información, la facturación electrónica en el contexto ecuatoriano y las tecnologías empleadas en el diseño e implementación de soluciones web modernas. A través del análisis del estado del arte y la revisión bibliográfica, se justifica la elección de herramientas, metodologías y marcos tecnológicos que han guiado la construcción del sistema, considerando tanto los requisitos normativos del Servicio de Rentas Internas (SRI) como las necesidades operativas y de modernización de la empresa.

Además, se compararon enfoques de desarrollo ágil como Scrum y Extreme Programming (XP), evaluando su idoneidad para proyectos de rápida iteración y validación continua como el que aquí se presenta. Finalmente, se detalla el marco técnico adoptado, que incluye el lenguaje de programación, frameworks de desarrollo frontend y backend, gestores de base de datos y entornos de desarrollo, todo ello enfocado en garantizar la viabilidad, escalabilidad y seguridad de la solución. Este capítulo proporciona así la base teórica y técnica necesaria para comprender las decisiones adoptadas durante el proceso de implementación del prototipo.

2.1 Marco teórico y conceptual

2.1.1 Estudio del arte del proceso de facturación

La facturación electrónica es una herramienta esencial en la parte contable como procesos contables y tributarios de las empresas modernas. En Ecuador, el Servicio de Rentas Internas ha establecido que es obligatorio emitir facturas electrónicas, especialmente aquellos que superen ingresos anuales de $20.000 o que no estén registrados bajo el régimen RIMPE o como negocios populares.

Este sistema de facturación tuvo como objetivo principal garantizar la autenticidad e integridad de los comprobantes de venta, por otro lado, también se puede reducir el uso de papel y simplificar los procesos fiscales. El SRI define la facturación electrónica como un mecanismo digital que permite emitir, recibir y validar comprobantes tributarios en formato XML, asegurando su legalidad mediante la firma electrónica del emisor (SRI, 2025)

El proceso de facturación implica la generación de documentos tributarios digitales con validez legal, como el formato estructurado como el **XML (Extensible Markup Language)**. Este lenguaje nos permite el intercambio de datos de forma estructurada, entendible tanto por humanos como por sistemas automatizados. (Amazon, 2024) describe XML como "un estándar flexible para la transferencia de datos entre plataformas distintas, permitiendo estructurar la información con jerarquías claras".

Según (Pérez, 2022), en un caso de estudio de la implementación de facturación electrónica en pymes ecuatorianas, se concluye que el uso de plataformas web reduce tanto los costos de infraestructura, facilita el proceso con el sistema, mejora el acceso remoto y la seguridad de la información, aspectos fundamentales en entornos empresariales dinámicos.

En este contexto, Brother Corp se encontraba en un proceso de modernización tecnológica que implico el paso de su sistema de escritorio a una solución Web. La meta de esta transición no solo fue cumplir con las normativas del SRI, sino también cumplir con las nuevas demandas del mercado, potenciando la accesibilidad, escalabilidad y seguridad del sistema.

En este proyecto, la facturación electrónica incluyó la creación del archivo XML que se ajustaron a las normas del SRI. A pesar de que el prototipo no contempló el envío directo al sistema de validación del SRI, sí estuvo listo para producir archivos capaces de ser incorporados en una fase posterior

2.1.2 Desarrollo de conceptos teóricos fundamentales para el proyecto

La base teórica del presente proyecto se fundamenta en los siguientes conceptos:

- **Sistemas de Información (SI):** Según (Kenneth C. Laudon, 2016), Un sistema de información es un grupo de elementos interconectados que recopilan, procesan, almacenan y distribuyen datos con el objetivo de respaldar la tomade dedici ones y el control en una entidad.
  Este proyecto es un componente de un sistema de datos enfocado en optimizar la efici encia en el proceso de facturación de Brother Corp.

- **Facturación Electrónica:** El (SRI, 2025) afirma que la facturación electrónica es un sistema jurídico de expedición de recibos digitales que sustituye al papel, garantizando su autenticidad a través de firmas digitales. Su puesta en marcha incrementa la claridad fiscal, disminuye los gastos operacionales y facilita un mayor seguimiento de las transacciones.
- **Metodología Ágil (Scrum):** (César Rodríguez, 2015), Fomentan un método iterativo e incremental en la creación de software, enfocado en la cooperación continua con el cliente, la aportación constante de valor y la habilidad para ajustarse a las variaciones. Scrum, una de las técnicas ágiles más conocidas, organiza la labor en sprints, con roles establecidos y entregables parciales que facilitan la verificación del progreso del proyecto en tiempo real.
- **Metodología Ágil (XP):** Extreme Programming (XP) es un enfoque ágil orientado a incrementar la calidad del software y la capacidad de reacción frente a las variaciones en las necesidades. Se creó en los años 90 por Kent Beck y se fundamenta en métodos como el desarrollo dirigido por pruebas (TDD), la programación en equipo, la integración constante y ciclos de desarrollo muy breves. De acuerdo con (Beck, 2004), XP aspira a proporcionar software operativo en etapas muy reducidas, con una comunicación intensa entre el equipo y el usuario, promoviendo de esta manera la flexibilidad y la calidad.
- **API RESTful:** La arquitectura API RESTful se fundamenta en un modelo cliente-servidor donde el cliente se relaciona con el servidor mediante llamadas HTTP organizadas de acuerdo con los principios REST (Transferencia de Estado Representativa). Cada recurso, ya sean facturas, clientes o productos, se presenta a través de una URI específica y se gestiona utilizando métodos HTTP convencionales (GET para lectura, POST para creación, PUT/PATCH para actualización y DELETE para su eliminación). Una API (Interfaz de Programación de Aplicaciones) es un grupo de especificaciones y protocolos que facilitan la interacción entre distintos sistemas de software. En el desarrollo web, las API RESTful son ampliamente utilizadas debido a su simplicidad y eficiencia. Según (Pautasso, 2009), REST es un estilo arquitectónico que permite a los sistemas interactuar a través de HTTP.

  El uso de APIs en este proyecto permitio desacoplar la lógica de negocio del cliente web, facilitando la escalabilidad, la reutilización del código y la integración futura con otras aplicaciones, como sistemas contables externos o validadores del SRI.

- **Desarrollo Web Responsivo:** Es un enfoque de diseño que permite que una aplicación web se adapte automáticamente al tamaño y tipo de dispositivo del usuario. Fue definido ampliamente por (Marcotte, 2014) y es esencial para garantizar la accesibilidad multiplataforma de una solución web moderna.
- **Seguridad de aplicación web:** Todo sistema que maneje datos sensibles, como información de clientes o transacciones, debe considerar aspectos de seguridad desde el diseño. Esto incluye autenticación, autorización, cifrado de datos, protección contra inyecciones SQL, y control de sesiones. Según (Springett, 2024), los ataques más comunes incluyen inyección de código, fallos de autenticación y exposición de datos sensibles.
- **Modelos de Datos relacional:** SQL Server es un sistema de gestión de bases de datos relacional (RDBMS) desarrollado por Microsoft, que organiza la información en tablas y permite relaciones entre ellas. Es ampliamente utilizado en entornos empresariales debido a su robustez, integración con otros servicios de Microsoft y su capacidad para manejar grandes volúmenes de datos.

2.1.3 Definiciones clave relacionadas con el desarrollo del sistema

A continuación, se definen los conceptos clave para el desarrollo del proyecto:

- **Prototipo**: Versión inicial y funcional de un sistema que permite evaluar su rendimiento, recoger retroalimentación y realizar ajustes antes de la implementación definitiva.
- **Requisitos Funcionales**: Son funciones específicas que el sistema debe cumplir, como registrar usuarios, generar facturas o exportar datos en formato XML.
- **Requisitos No Funcionales**: Atributos de calidad del sistema, como seguridad, usabilidad, disponibilidad, rendimiento y escalabilidad.
- **Interfaz de Usuario (UI)**: Medio gráfico a través del cual el usuario interactúa con la aplicación. Una buena UI debe ser clara, accesible y responsiva.
- **Diseño Web Responsivo**: Técnica que permite que una aplicación web se adapte automáticamente a distintos tamaños de pantalla y dispositivos. Es esencial para brindar una experiencia de usuario consistente.

- **Base de Datos Relacional**: Modelo de almacenamiento de información estructurada en tablas interrelacionadas. SQL Server será utilizado como sistema gestor, permitiendo consultas eficientes y administración centralizada.
- **API RESTful**: Interfaz que permite que aplicaciones se comuniquen entre sí usando el protocolo HTTP. Son fundamentales para separar el frontend del backend, facilitando la escalabilidad e interoperabilidad.
- **Metodología Scrum**: Enfoque ágil de desarrollo de software basado en entregas incrementales y colaboración continua entre equipos y clientes.
- **Metodología XP (Extreme Programming)**: Variante ágil centrada en buenas prácticas de desarrollo como TDD, programación en pareja, integración continua y ciclos cortos de entrega.
- **Seguridad en Aplicaciones Web**: Conjunto de prácticas y medidas que protegen los sistemas contra ataques como inyecciones SQL, robo de credenciales o acceso no autorizado.

2.2 Marco técnico

El marco técnico incluyó todos los componentes requeridos que facilitó el diseño, desarrollo e implementación del prototipo de una aplicación web para la facturación electrónica de Brother Corp. Este segmento es crucial, pues estableció los recursos tecnológicos, metodológicos e infraestructurales que garantizaron la viabilidad y el correcto desempeño del sistema sugerido.

En cualquier proyecto de software, disponer de un marco técnico definido facilita determinar con exactitud qué herramientas se emplearán: lenguajes de programación, bases de datos, arquitecturas, metodologías y ambientes laborales. En esta situación, se eligió una estructura de 3 capas fundamentada en servicios RESTful, empleando una base de datos SQL Server.

Además, se decidió utilizar una metodología ágil de desarrollo que posibilitó reaccionar rápidamente ante las variaciones en las necesidades, promover la cooperación constante con el cliente y ayudó entregas funcionales en lapsos de tiempo breves. En este contexto, se realizó un estudio comparativo entre las metodologías Scrum y Extreme Programming (XP), eligiendo la más apropiada para este proyecto.

A continuación, se describen los elementos técnicos escogidos para la creación de la solución, que incluyen lenguajes de programación, marcos de referencia, herramientas de control de versiones, diseño de arquitectura, seguridad y ambiente de implementación.

2.2.1 Metodología Ágil de desarrollo (Extreme Programming (XP) vs Scrum)

La selección de una metodología de desarrollo de software es una decisión crítica en todo proyecto tecnológico. En el caso del desarrollo de la aplicación web de facturación electrónica, se requirió una metodología que permita adaptarse con agilidad a cambios en los requerimientos, ofrecer entregables funcionales frecuentes y mantener una comunicación constante entre el equipo de desarrollo y los usuarios finales.

Las metodologías ágiles ofrecen un marco de trabajo ideal para este tipo de proyectos. Entre las más reconocidas se encuentran **Scrum** y **Extreme Programming (XP)**.

2.2.1.1 Scrum

Scrum es un marco ágil de gestión de proyectos inspirado en la formación de rugby SCRUM (César Rodríguez, 2015), que organiza el trabajo en iteraciones cortas llamadas sprints y define roles (Product Owner, Scrum Master y equipo de desarrollo), eventos (Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective) y artefactos (Product Backlog, Sprint Backlog, Incremento) para fomentar la autoorganización, la transparencia y la mejora continua; aunque nació en el desarrollo de software, su ciclo de inspección, adaptación y entrega incremental lo hace valioso para cualquier tipo de proyecto colaborativo. (Drumond, 2025)

Scrum al ser una metodología ágil facilita el desarrollo en este caso del prototipo Web mediante sprints de dos semanas, lo que facilita planificar entregas periódicas y demostrar avances concretos a los stakeholders con demos; esto construye confianza y garantiza que los requisitos de facturación electrónica se validen tempranamente. Las ceremonias Sprint Planning, Daily Stand-up, Sprint Review y Retrospective ofrecen un marco estructurado para identificar bloqueos, ajustar prioridades y promover la mejora continua, ideal cuando se requiere un compromiso claro de entregables y visibilidad compartida del progreso.

2.2.1.1.1 Implementación de Scrum

- Definición del Product Backlog

Se recopilaron y prioriza todas las funcionalidades del módulo de facturación electrónica (p. ej., emisión de factura, generación de XML, gestión de clientes y productos) en una lista ordenada por valor para el negocio.

- Sprints

Se divide el trabajo en iteraciones de dos semanas. Al inicio de cada sprint, se seleccionan del Product Backlog las historias de usuario comprometidas para esa iteración y se estiman en puntos.

- Revisión del Sprint (Sprint Review)

Al final de cada sprint se demuestra ante los stakeholders el incremento funcional entregado (p. ej., módulo de emisión de factura PDF), se recopila feedback y se ajustan prioridades en el Product Backlog.

- Retrospectiva del Sprint (Sprint Retrospective)

Tras la revisión, se reflexiona sobre el sprint completado, identificando qué salió bien y qué puede mejorarse (estimaciones, procesos, herramientas) para optimizar el siguiente sprint.

2.2.1.1.2 Beneficios de Scrum

Entrega regular y predecible: Con sprints fijos, los stakeholders reciben demostraciones cada dos semanas, lo que facilita la planificación de despliegues y la gestión de expectativas.

Visibilidad y transparencia: El Product Backlog muestra en todo momento el progreso real y las tareas pendientes.

Adaptación continua: El feedback temprano en cada Sprint Review permite corregir rumbo rápidamente antes de avanzar demasiado en desarrollos que puedan no ajustarse a necesidades reales.

Mejora constante: Las retrospectivas fomentan la optimización del proceso de trabajo y la calidad del código sprint tras sprint.

2.2.1.2 Extreme Programming (XP)

Extreme Programming (XP) es un marco ágil de desarrollo de software que pone el foco en prácticas de ingeniería intensivas para garantizar alta calidad y rapidez de entrega. XP prescribe valores como comunicación constante, simplicidad del diseño y feedback continuo, y adopta prácticas clave como Test-Driven Development (TDD), donde antes de escribir código se generan pruebas automatizadas; Pair Programming, en el que dos desarrolladores colaboran en la misma tarea para reducir errores; e Integración Continua (CI), que asegura que cada cambio pase por un servidor de builds y pruebas antes de fusionarse

XP permite un flujo de trabajo muy dinámico mediante iteraciones cortas (a menudo de una semana o menos) y small releases, de modo que el cliente on-site prueba y valida funcionalidades casi a diario. Al no requerir ceremonias extensas, minimiza el overhead y maximiza el tiempo dedicado a la codificación y las pruebas. Gracias a TDD y CI, cada funcionalidad entregada llega con cobertura de pruebas, mientras que el feedback inmediato del cliente y el refactoring constante mantienen el código sencillo y alineado con los cambios de requisitos.

2.2.1.2.1 Implementación de XP

La implantación de XP en el proyecto de prototipo de facturación electrónica sigue un ciclo iterativo e incremental, guiado por prácticas de ingeniería muy concretas y feedback continuo:

Formación del equipo y roles

- **Cliente on-site:** representante de negocio (ej. área contable de Brother Corp) disponible para validar historias de usuario y aceptar entregables al instante.
- **Desarrolladores:** implementan, prueban y refactorizan el código.
- **Coach XP:** guía al equipo en la correcta adopción de prácticas (planning, TDD, pairing) y vela por la comunicación y valores de XP.

Planificación del producto y releases

- **Exploración (Product Life Cycle):** el cliente aporta historias de usuario, describiendo las funcionalidades deseadas (creación de factura, generación de XML, etc.).

- **Compromiso (Release Planning):** todo el equipo revisa el progreso, ajusta requisitos, estima historias y establece la fecha del siguiente release, comprometiéndose con un conjunto de funcionalidades.

Planificación de iteraciones (Iterations)

- Ciclos de **1 semana** o menos.
- En la sesión de iteración, se discuten en detalle las historias seleccionadas, se crean listas de tareas y se adapta el plan según el feedback y la capacidad del equipo.

Desarrollo de tareas y episodes

- Cada desarrollador toma tareas de la iteración y las convierte en "development episodes": pequeñas unidades de trabajo que incluyen test, código y refactorizado. Se asegura que cada historia tenga sus tests de aceptación automatizados antes de considerar la tarea completa.

Prácticas de ingeniería

- **Test-Driven Development (TDD):** antes de escribir código se crea un test que falla; luego se implementa la mínima lógica para hacerlo pasar y, finalmente, se refactoriza para mantener un diseño simple.
- **Integración Continua (CI):** cada cambio se integra y ejecuta automáticamente contra la suite de tests, garantizando que el sistema siga construyéndose sin errores.
- **Refactorización continua:** tras pasar los tests, se limpia el código para eliminar duplicaciones, mejorar nombres y simplificar estructuras.
- **Propiedad colectiva del código y diseño simple:** cualquier parte del código puede ser modificada por cualquiera, siempre manteniendo la metáfora del dominio y evitando complejidad innecesaria.

Small Releases y feedback constante

- Al terminar cada iteración se entrega un build desplegable (small release) al cliente on-site para validación. El feedback inmediato guía la repriorización de historias y los ajustes del siguiente ciclo.

Ajuste y mejora continua

- Las retrospectivas ligeras tras cada release/iteración permiten identificar bloqueos y optimizar procesos. El flujo de feedback bidireccional asegura que el prototipo evolucione alineado con las necesidades reales de facturación electrónica.

2.2.1.2.2 Beneficios de XP

1. **Alta calidad del software desde el inicio**
    - La práctica de *Test-Driven Development* (TDD) asegura que cada línea de código esté respaldada por pruebas automatizadas antes de pasar a producción, reduciendo defectos y retrabajo (Beck, 2004).
2. **Feedback continuo y ajuste rápido**
    - Con *small releases* al final de iteraciones muy cortas (1 semana o menos), el cliente on-site valida funcionalidades casi a diario, lo que permite priorizar historias de usuario de forma inmediata (Beck, 2004)
3. **Colaboración técnica intensiva**
    - *Pair Programming* fomenta la difusión de conocimiento, la revisión de código en tiempo real y la reducción de errores, al tener dos desarrolladores compartiendo “driver” y “navigator” (Beck, 2004).
4. **Integración continua y refactorización constante**
    - Cada cambio se integra automáticamente en el repositorio principal y se ejecuta contra la suite de tests, garantizando que el sistema permanezca siempre construible y evitando “deudas técnicas” acumuladas (Beck, 2004).
5. **Simplicidad y propiedad colectiva**
    - El código se mantiene lo más sencillo posible y cualquier miembro del equipo puede modificarlo, facilitando la adaptación a nuevos requerimientos sin crear puntos críticos de dependencia (Beck, 2004).

2.2.1.3 Conclusión de metodología Ágil

Tras comparar en profundidad Scrum y Extreme Programming (XP), se evidencio que XP ofrece ventajas decisivas para el desarrollo del prototipo de facturación electrónica de Brother Corp. La naturaleza crítica de componentes como la generación de XML conforme al esquema del SRI, la robustez de la API REST y la integración con SQL Server exige un enfoque que priorice la calidad técnica, la rapidez de adaptación y la colaboración estrecha entre el equipo de desarrollo y los expertos de negocio.

En concreto, XP aporta:

1. **Control riguroso de la calidad**
    - Con *Test-Driven Development (TDD)*, cada nueva funcionalidad nace acompañada de pruebas automatizadas que garantizan su correcto comportamiento, Esto reduce drásticamente el riesgo de fallos en producción y facilita la detección temprana de errores, alineándose con el objetivo de entregar un prototipo robusto y confiable.
2. **Retroalimentación continua y rápida validación**
    - Las *small releases* diarias y la presencia de un **cliente on-site** permiten verificar al instante el cumplimiento de requisitos, acelerando ciclos de ajuste y minimizando el desvío respecto a las necesidades reales de facturación electrónica.
3. **Reducción de deuda técnica**
    - La combinación de *Integración Continua* e *Iteraciones cortas* impulso la refactorización constante, manteniendo el código limpio, modular y fácil de escalar en etapas posteriores, lo que resulta esencial para la futura migración completa del ERP de escritorio a la web.
4. **Enfoque pragmático y bajo overhead**
    - Al prescindir de ceremonias extensas, XP maximizo el tiempo de desarrollo efectivo, concentrando la gestión ágil en reuniones breves de planificación y retrospectiva que garantizan alineación sin sacrificar velocidad.

Además, XP se alineó de forma natural con los objetivos de este proyecto de titulación: entregar un prototipo plenamente funcional en plazos acotados, validar de inmediato las características más críticas y construir una base de código que sirva como cimiento para el sistema final.

Su énfasis en prácticas de ingeniería rigurosas y su capacidad para incorporar cambios de forma fluida lo convirtieron en la opción óptima frente a metodologías más orientadas a la gestión de proyectos, como Scrum.

Por todo lo anterior, se adoptó XP como la metodología de desarrollo para este prototipo de facturación electrónica Web, sentando así las bases para una solución web escalable, segura y conforme a la regulación del SRI, y asegurando al mismo tiempo que el equipo de Brother Corp cuente con un proceso de trabajo sostenible y de alta calidad para etapas futuras.

Aspecto	**Scrum**	**XP**
Enfoque principal	Gestión y planificación de proyectos en sprints de 2–4 semanas.	Prácticas de ingeniería que garantizan calidad y rapidez.
Ritmo de entregas	Cada sprint (2 semanas).	Iteraciones cortas (1 semana o menos) y *small releases* diarias.
Ceremonias / Overhead	Varias reuniones estructuradas (Planning, Daily, Review, Retrospective).	Mínimo overhead: planificación ligera y retrospectivas breves.
Garantía de calidad	Retroalimentación basada en demostraciones al final del sprint.	Cobertura de pruebas previa al desarrollo (TDD).
Adaptabilidad al cambio	Adaptación al inicio de cada sprint.	Cambios incorporados en cada iteración y refactorización.
Colaboración	Roles definidos y reuniones formales.	Trabajo en pareja y comunicación constante.
Ideal para…	Proyectos con necesidad de visibilidad ante múltiples stakeholders.	Equipos pequeños, prototipos técnicos, cambios frecuentes.

2.2.2 Lenguaje de programación y Framework

2.2.2.1 Lenguajes de programación para el BackEnd

2.2.2.1.1 C#

C# es un lenguaje de propósito general, moderno y orientado a objetos, desarrollado por Microsoft para la plataforma .NET. Diseñado para ser simple, seguro y expresivo, C# combina características de lenguajes como C++ y Java con un sistema de tipos estático que previene errores en tiempo de compilación. Además, C# ofrece un extenso conjunto de bibliotecas de clases (BCL) para acceso a datos, manipulación de XML, concurrencia y criptografía, lo que lo convierte en la opción natural para construir la lógica de negocio de un sistema de facturación electrónica que deba interactuar de forma segura y eficiente con SQL Server (Skeet, 2019).

Tipado y detección temprana de errores

C# emplea un sistema de tipos estático, lo que significa que muchas inconsistencias en el uso de variables y llamadas a métodos se detectan en tiempo de compilación, reduciendo drásticamente la posibilidad de errores en producción y mejorando la fiabilidad de componentes críticos como la generación de XML fiscal (ecma, 2023).

Rendimiento y eficiencia

C#, ejecutado sobre el Common Language Runtime (CLR), aprovecha la compilación Just-In-Time (JIT) y optimizaciones de código nativo, así como un recolector de basura de alta frecuencia y una gestión de hilos optimizada, lo que se traduce en un comportamiento más predecible y menor latencia en operaciones transaccionales de alta carga (Microsoft, CLR via C#, 2012)

Ecosistema y bibliotecas empresariales

C#/.NET dispone de un entorno unificado con ADO.NET, Entity Framework, System.Xml y librerías criptográficas empresariales, todas optimizadas para interactuar con SQL Server, lo que simplifica la serialización de datos, el acceso transaccional y el cifrado de información (Skeet, 2019).

Herramientas y soporte de desarrollo

Visual Studio 2022, optimizado para C#, ofrece depuración avanzada, análisis estático de código, perfiles de CPU y memoria e infraestructura de hot reload, acelerando el ciclo de desarrollo y reduciendo errores críticos (Microsoft, Notas de la versión de Visual Studio 2022, 2022)

Escalabilidad y despliegue

C#/.NET Framework 4.8.1 está plenamente soportado en Windows Server con IIS y contenedores Windows, e incorpora soluciones de alta disponibilidad como Always On Availability Groups, garantizando SLAs corporativos y recuperación ante desastres (Microsoft, Documentación de Microsoft SQL, 2025).

2.2.2.1.2 Python

Python es un lenguaje de programación de alto nivel, interpretado y de tipado dinámico, desarrollado por la Python Software Foundation con un fuerte énfasis en la legibilidad y la simplicidad de su sintaxis (Python, 2025). Su filosofía de "baterías incluidas" proporciona una biblioteca estándar muy completa para manipulación de datos, redes y XML, lo que acelera el prototipado de aplicaciones (Lutz, 2009). Gracias a su curva de aprendizaje suave y su comunidad, Python resulta ideal para pruebas rápidas de concepto, aunque en entornos .NET/SQL Server suele requerir capas adicionales de interoperabilidad.

Tipado y detección temprana de errores

Python, al ser un lenguaje de tipado dinámico, ofrece mayor flexibilidad y velocidad de prototipado, pero incurre en un riesgo mayor de errores de tipo en tiempo de ejecución, que deben mitigarse mediante pruebas exhaustivas o con el uso de type hints (Python, 2025).

Rendimiento y eficiencia

Python, basado en un intérprete como CPython, incurre en un overhead mayor debido a la interpretación línea a línea y al Global Interpreter Lock (GIL), que limita el paralelismo real en hilos y puede impactar el rendimiento bajo alta concurrencia (Python, 2025).

Ecosistema y bibliotecas empresariales

Python cuenta con bibliotecas de terceros como SQLAlchemy, lxml y cryptography, pero su integración con entornos Windows o con Entity Framework no es directa, lo que añade complejidad al configurar pipelines de despliegue y depuración en infraestructuras basadas en .NET (Lutz, 2009).

Herramientas y soporte de desarrollo

Python IDEs como PyCharm o las extensiones de Python en VS Code facilitan la codificación rápida y cuentan con depuradores integrados, pero carecen de una suite de diagnóstico y profiling tan madura e integrada como la de Visual Studio para aplicaciones de misión crítica (Python, 2025).

Escalabilidad y despliegue

Python se despliega de manera nativa en múltiples plataformas (Linux, macOS, Windows) y en contenedores Docker, pero para alcanzar niveles de resiliencia y monitoreo equivalentes requiere motores web adicionales (uWSGI, Gunicorn) y configuraciones de clustering más complejas (Python, 2025).

2.2.2.1.3 Justificación de elección de C#

- **Seguridad y robustez en tiempo de compilación**
  El tipado estático de C# y la detección anticipada de errores mitigan riesgos en módulos sensibles como la generación de facturas XML.
- **Rendimiento y concurrencia**
  La arquitectura CLR con JIT y gestión eficiente de hilos garantiza tiempos de respuesta adecuados bajo cargas transaccionales intensivas.
- **Integración nativa con la plataforma .NET y SQL Server**
  El uso de ADO.NET, Entity Framework y herramientas como PowerDesigner en un ecosistema unificado facilita el modelado de datos, el acceso a la base y el despliegue continuo.
- **Herramientas de desarrollo sólidas**
  Visual Studio 2022 ofrece un entorno completo de depuración, profiling y análisis de código estático, clave para proyectos de alta criticidad como un sistema de facturación electrónica.
- **Soporte corporativo y escalabilidad**
  La compatibilidad y las garantías de Microsoft en actualizaciones, seguridad y alta disponibilidad hacen de C# y .NET la apuesta más segura para el prototipo y las etapas productivas futuras.

Por estos motivos, C# se consolidó como el lenguaje de backend óptimo para la aplicación de facturación electrónica de Brother Corp, al superar a Python en aspectos clave como la seguridad, el rendimiento y la alineación con la infraestructura tecnológica existente. Su madurez, junto con la disponibilidad de herramientas de desarrollo avanzadas en Visual Studio y el amplio respaldo de la comunidad, reforzaron la elección de C# como pilar fundamental del backend de la aplicación.

2.2.2.2 Lenguajes de programación FrontEnd

2.2.2.2.1 Flutter

Flutter es un kit de desarrollo de interfaz de usuario de código abierto creado por Google, que permite construir aplicaciones nativas para móviles, web y escritorio a partir de una única base de código en Dart (Flutter, 2024). A diferencia de las aproximaciones híbridas que dependen de componentes web embebidos, Flutter compila directamente a código nativo y utiliza su propio motor de gráficos para renderizar widgets, garantizando un rendimiento cercano al de aplicaciones construidas de forma nativa. Su arquitectura reactiva simplifica la gestión de estado y la actualización de la interfaz, mientras que el "hot reload" acelera drásticamente el ciclo de desarrollo al permitir ver cambios en tiempo real sin perder el estado de la aplicación. Para el prototipo de facturación electrónica, Flutter ofrece la agilidad necesaria para diseñar vistas responsivas que se adapten a distintos tamaños de pantalla, facilita la integración con APIs RESTful y asegura una experiencia de usuario consistente en Windows, macOS, Linux, Android y iOS.

Tipado y detección temprana de errores

Flutter (Dart) utiliza un sistema de tipos estático con comprobaciones en tiempo de compilación, lo que permite detectar incompatibilidades de tipos y errores de API antes de ejecutar la aplicación, aumentando la confiabilidad del código (Dart, 2024).

Rendimiento y eficiencia

Flutter compila directamente a código nativo en cada plataforma (móvil, escritorio y web) usando su propio motor gráfico Skia, garantizando un rendimiento cercano al de las aplicaciones nativas sin depender de un motor JavaScript (Flutter, 2024).

Ecosistema y bibliotecas

Flutter dispone de pub.dev, un repositorio centralizado de paquetes que cubre desde widgets UI específicos hasta integración con servicios nativos, todo gestionado en Dart sin necesidad de JavaScript intermedio (Flutter, 2024)

Herramientas y soporte de desarrollo

Flutter ofrece Flutter DevTools y el comando flutter CLI, integrados en VS Code o Android Studio, con "hot reload" instantáneo, inspección de widget tree y profiling de rendimiento nativo (Flutter, 2024).

Multiplataforma y alcance

Flutter permite compilar la misma base de código Dart para iOS, Android, web, Windows, macOS y Linux, ofreciendo una experiencia de usuario homogénea y reduciendo costos de mantenimiento (Flutter, 2024).

2.2.2.2.2 Angular

Angular es una plataforma de desarrollo de aplicaciones web de código abierto diseñada por Google, basada en TypeScript y que permite construir aplicaciones de una sola página (SPA) con arquitecturas modulares y escalables (Angular, 2025). Emplea un sistema de componentes, inyección de dependencias y un compilador ahead-of-time (AOT) que optimiza el código para producción, además de un enrutador integrado y servicios para manejo de formularios, validación y comunicación HTTP. La estrecha integración con TypeScript ofrece tipado estático, detectando errores en tiempo de compilación y facilitando el refactorizado de grandes bases de código. Angular dispone de un robusto ecosistema de librerías oficiales como Angular Material para componentes UI para programación reactiva, lo que lo convierte en una opción muy popular para proyectos empresariales que requieran mantenimiento a largo plazo y arquitecturas basadas en el navegador.

Tipado y detección temprana de errores

Angular (TypeScript) también se basa en tipado estático gracias a TypeScript, ofreciendo detección de errores en compilación y autocompletado en IDE, pero al compilar a JavaScript introduce una capa adicional que puede enmascarar discrepancias entre el código fuente y el código de producción (Angular, 2025).

Rendimiento y eficiencia

Angular ejecuta en el navegador sobre JavaScript/DOM, por lo que las actualizaciones de UI dependen del motor JavaScript y del reconciliador de cambios de Angular; aunque el compilador AOT y el change detection optimizado mejoran la velocidad, sigue existiendo overhead al manipular el DOM (Angular, 2025).

Ecosistema y bibliotecas

Angular cuenta con un ecosistema amplio de módulos NPM, incluidas librerías oficiales como Angular Material y RxJS, pero requiere administrar dependencias de JavaScript/TypeScript y sus correspondientes tipos (@types) para garantizar la compatibilidad (Angular, 2025).

Ecosistema y bibliotecas

Angular cuenta con un ecosistema amplio de módulos NPM, incluidas librerías oficiales como Angular Material y RxJS, pero requiere administrar dependencias de JavaScript/TypeScript y sus correspondientes tipos (@types) para garantizar la compatibilidad (Angular, 2025).

Herramientas y soporte de desarrollo

Angular utiliza Angular CLI junto con herramientas como ng serve, ng build y extensiones de VS Code para TypeScript; incluye scaffolding, linting y testing pero su "hot reload" (Hot Module Replacement) depende de Webpack y no siempre conserva el estado de la aplicación (Angular, 2025).

Multiplataforma y alcance

Angular está diseñado para aplicaciones web (SPA); aunque puede integrarse con soluciones como Electron para escritorio o NativeScript para móvil, cada plataforma suele requerir configuraciones y código adicionales (Angular, 2025).

2.2.2.2.3 Justificación de elección de Flutter

Para el desarrollo del prototipo de facturación electrónica, Flutter se seleccionó como framework frontend por varias razones clave:

Único lenguaje Dart: A diferencia de Angular, que depende de JavaScript/TypeScript, Flutter utiliza exclusivamente Dart, reduciendo la complejidad de tener que gestionar dos lenguajes.

Compilación nativa: Flutter compila directamente a código nativo en todas las plataformas soportadas (móvil, web y escritorio), lo que asegura un rendimiento muy cercano al de las apps nativas sin depender de un motor JavaScript.

Motor de renderizado propio: Al renderizar widgets mediante su propio motor gráfico Skia, Flutter ofrece animaciones fluidas y un control preciso de la UI, sin variaciones entre navegadores o versiones.

Hot reload y productividad: Su sistema de "hot reload" permite iterar rápidamente sobre la UI y la lógica de la aplicación, acelerando el desarrollo y la validación de diseños responsivos.

Multiplataforma real: Con un solo código Dart se cubren Windows, macOS, Linux, Android y iOS, ideal para una solución que debe ser accesible desde cualquier dispositivo.

2.2.2.3 ASP.NET(.NET Framework 4.8.1)

El .NET Framework 4.8.1 es la última versión "madura" de la plataforma de ejecución de Microsoft, optimizada para entornos Windows Server y equipos de escritorio. Ofrece compatibilidad garantizada con bibliotecas, parches de seguridad continuos y un sólido modelo de despliegue en entornos corporativos (Microsoft, Documentación de .NET Framework, s.f.).Con soporte integrado para Entity Framework, ADO.NET y el Entity Data Model, facilita el mapeo objeto-relacional y la gestión de transacciones, elementos críticos al interactuar con una base de datos SQL Server que contendrá información sensible de facturación. La biblioteca System.Web y el motor de ASP.NET proporcionan mecanismos de gestión de sesiones, autenticación y autorización, así como pipelines de solicitud altamente configurables que son imprescindibles para asegurar la integridad de los datos.

2.2.2.4 ASP.NET Web API

ASP.NET Web API es un framework ligero para la creación de servicios HTTP que expongan recursos como endpoints RESTful basados en controladores y atributos de enrutamiento. Diseñado para integrarse de forma nativa con la Web API permite serializar y deserializar datos en formatos como JSON o XML, aplicar filtros de acción y delegar la seguridad a middleware configurable (Microsoft, ASP.NET Core, 2024). En el contexto de la facturación electrónica, esta capacidad de exponer los procesos de emisión de facturas, generación de XML y gestión de catálogos de productos y clientes como servicios RESTful facilita el desacoplo entre frontend y backend, incrementa la reutilización y simplifica futuras integraciones con otros sistemas contables o validadores externos.

2.2.2.5 ADO.NET Entity Data Model

El **ADO.NET Entity Data Model** (EDM) es una capa de mapeo objeto relacional incluida en la plataforma .NET que permite representar las tablas, vistas y relaciones de una base de datos como clases y asociaciones en C#. Mediante un archivo EDMX, Entity Framework genera un modelo conceptual, un esquema de almacenamiento y las reglas de mapeo entre ambos, de modo que el desarrollador puede trabajar con objetos .NET fuertemente tipados en lugar de escribir sentencias SQL manualmente (Microsoft, Introducción a Entity Framework, s.f.). Además, su soporte para LINQ to Entities facilita la escritura de consultas expresivas y seguras en tiempo de compilación, mientras que las capacidades de "Database-First" y "Code-First" ofrecen flexibilidad a la hora de diseñar o evolucionar el esquema de datos (Microsoft, Migraciones de Code First, s.f.).

2.2.3 IDE de desarrollo

2.2.3.1 IDE de desarrollo para Backend: Visual Studio 2022

Visual Studio 2022 es el Entorno de Desarrollo Integrado (IDE) de referencia para la plataforma .NET, diseñado para maximizar la productividad y la calidad del código en proyectos empresariales. Su soporte nativo para C# y ASP.NET (.NET Framework 4.8.1) incluye IntelliSense inteligente, refactorizaciones de código basadas en el compilador Roslyn, depuración avanzada con puntos de interrupción condicionales y "Hot Reload" para ver cambios en tiempo real sin reiniciar la aplicación (Microsoft, Notas de la versión de Visual Studio 2022, 2022). Además, Visual Studio integra diagnósticos de rendimiento, profilers de CPU y memoria, y herramientas de análisis estático que permiten detectar vulnerabilidades y "code smells" antes de llegar a producción. La conexión directa con SQL Server en el Explorador de servidores y el diseñador de modelos EDMX acelera la creación y mantenimiento del Entity Data Model. Finalmente, su integración con Git y Azure DevOps facilita la gestión de ramas, revisiones de código (pull requests) y despliegues automáticos, garantizando trazabilidad y control de versiones en cada artefacto del proyecto.

2.2.3.2 Entorno de desarrollo Frontend: Visual Studio Code

Visual Studio Code es un editor de código ligero, multiplataforma y extensible, ideal para el desarrollo con Flutter y Dart. Esta combina arranque instantáneo con un ecosistema de extensiones específico—como el plugin "Flutter" de Dart-Code—que provee autocompletado inteligente, "hot reload" de la interfaz y depuración en dispositivos reales o emuladores (Microsoft, Flutter, 2018). Su terminal integrado y su barra de estado ofrecen acceso rápido a

comandos de CLI (por ejemplo, flutter run o git) sin cambiar de ventana. VS Code cuenta también con integraciones para análisis de código estático (Dart Analysis), formateo automático y snippets personalizados, lo cual simplifica la creación de interfaces responsivas y la gestión de estados en la aplicación web. Gracias a su flexibilidad, el equipo puede instalar sólo las extensiones necesarias y ajustar el entorno a su flujo de trabajo, manteniendo un IDE ágil y de bajo consumo de recursos.

2.2.4 Base de datos

Para el diseño del modelo de datos de la aplicación de facturación electrónica se utilizó **SAP PowerDesigner**, una herramienta que permitió abarcar todo el ciclo de modelado desde lo conceptual y lo lógico hasta la generación física del esquema. Entre sus funcionalidades destaca la generación automática de código DDL para diversos motores (Oracle, SQL Server, PostgreSQL, MySQL, entre otros), lo que facilito la creación, modificación o eliminación de estructuras en la base de datos conforme a los requisitos de almacenamiento (powerdesigner, 2024).

A continuación, dos ventajas clave del uso de PowerDesigner:

- **Enfoque Model-Driven Architecture (MDA) que alinea negocio y TI**
  PowerDesigner implementa la metodología MDA, partiendo de un modelo conceptual de alto nivel (entidades, relaciones, reglas de negocio) y derivando automáticamente los modelos lógico y físico. Esto asegura que los requerimientos de facturación —como las estructuras de factura, clientes y productos— se traduzcan fielmente en un esquema DDL consistente, reduciendo errores manuales y acelerando la puesta en marcha del prototipo.
- **Automatización de generación de DDL e ingeniería inversa**
  La herramienta genera el SQL necesario para crear tablas, índices, restricciones y demás objetos. Además, su capacidad de ingeniería inversa permite modelar bases de datos existentes, lo que agiliza migraciones o integraciones con sistemas legados de Brother Corp y minimiza el riesgo de inconsistencias.

2.2.4.1 Comparación entre SQL Server y MySql

Licencia y coste

Microsoft SQL Server está disponible en varias ediciones, desde la gratuita **Express** hasta las de pago **Standard** y **Enterprise**, cuya licencia se basa en el número de núcleos de CPU o dispositivos de acceso. Esta estructura puede implicar un coste significativo cuando se busca alta concurrencia y escalabilidad empresarial, pero a cambio ofrece garantías de soporte oficial de Microsoft, actualizaciones de seguridad y un largo ciclo de vida, fundamentales para entornos críticos como los de facturación electrónica (Microsoft, Documentación de Microsoft SQL, 2025).

MySQL Community Edition es de código abierto bajo GPL, lo que elimina el coste de licencias, pero sus características empresariales avanzadas están disponibles en la edición Enterprise, de suscripción paga, o mediante plugins de terceros (Oracle, 2025)

Rendimiento y escalabilidad

SQL Server integra motores de almacenamiento como el tradicional **rowstore** y el **columnstore** optimizado para análisis de grandes volúmenes de datos; además, su tecnología **In-Memory OLTP** permite ejecutar transacciones de alta velocidad con latencias mínimas, ideal para picos de emisión masiva de facturas (Microsoft, Documentación de Microsoft SQL, 2025).

MySQL, con su motor predeterminado InnoDB, ofrece buen rendimiento en operaciones CRUD y lectura intensiva, pero carece de un componente nativo de OLTP en memoria tan evolucionado, por lo que en escenarios de alta concurrencia puede requerir ajustes manuales de configuración y particionamiento para mantener tiempos de respuesta bajos (Oracle, 2025).

Seguridad y cumplimiento

Dentro de lo que es seguridad, SQL Server proporciona cifrado **TDE** (Transparent Data Encryption) para proteger datos en reposo, así como **Always Encrypted** para resguardar datos sensibles con claves que nunca abandonan el cliente, y un completo framework de auditorías a

nivel de fila y columna, que es esencial para cumplir normativas tributarias y de protección de datos (Microsoft, Documentación de Microsoft SQL, 2025).

MySQL dispone de cifrado en reposo e **SSL/TLS** para datos en tránsito, y mecanismos de autenticación pluggable, pero sus funcionalidades de auditoría y control de acceso granular suelen requerir la edición Enterprise o extensiones de terceros, lo que añade complejidad al cumplimiento de estándares de seguridad estrictos (Oracle, 2025).

Ecosistema y herramientas de desarrollo

SQL Server se integra de forma nativa con Visual Studio y PowerDesigner, lo que permite a los desarrolladores modelar, generar código DDL y depurar procedimientos almacenados desde el mismo IDE, y aprovechar características de Entity Framework directamente para mapeo objeto-relacional (Skeet, 2019).

MySQL ofrece MySQL Workbench para diseño y administración, así como conectores en múltiples lenguajes (PHP, Java, Python), pero su experiencia dentro del entorno .NET es menos fluida, requiriendo a menudo componentes de terceros o configuraciones adicionales para aprovechar al máximo el ORM de Entity Framework y las herramientas de Microsoft.

Gestión, monitoreo y soporte

En cuanto a administración y seguimiento, SQL Server proporciona SQL Server Management Studio (SSMS) y Azure Data Studio. La asistencia de Microsoft abarca una documentación detallada y los SLAs de la empresa.

MySQL ofrece MySQL Workbench y MySQL Enterprise Monitor, sin embargo, las habilidades de diagnóstico en tiempo real y las sugerencias de optimización tienden a ser menos sofisticadas sin un plan de asistencia Enterprise (Oracle, 2025).

2.2.4.2 Justificación de la elección de SQL Server

Pese a que MySQL se destaca por su coste cero en la edición Community y su versatilidad, SQL Server cumple de manera más efectiva con los requisitos del prototipo de facturación electrónica de Brother Corp por las siguientes razones:

1. **Integración directa con.NET y PowerDesigner:** Permite el modelado y la ejecución directa desde el IDE y la herramienta MDA, disminuyendo la fricción en el proceso laboral.
2. **Características avanzadas de seguridad:** La encriptación clara de los datos y las auditorías locales garantizan la salvaguarda de información delicada, esencial para la información tributaria.
3. **Mejora del desempeño en transacciones:** Los motores in-memory y el indexado adaptable facilitan la gestión de picos de transacciones durante cierres contables o validaciones en masa de facturas.
4. **Soporte corporativo y alta disponibilidad**: La disponibilidad de ediciones Enterprise con SLA de Microsoft garantiza continuidad operativa y recuperación rápida ante fallos.

Por estas ventajas, **SQL Server** fue el motor seleccionado para desarrollar la base de datos del sistema, ofreciendo un entorno robusto, seguro y escalable para la aplicación web de facturación electrónica.

Capítulo III: Requerimientos para el desarrollo del Sistema de Facturación Electrónica

3.1 Casos de Uso

En este capítulo se ilustran las situaciones de uso que establecen las relaciones fundamentales entre los participantes y el prototipo de la aplicación web para facturación electrónica de Brother Corp. Cada caso de uso detalla el curso de sucesos desde la acción inicial del actor hasta el desenlace previsto y registra tanto las rutas principales como las opciones y las excepciones. Este grado de minuciosidad:

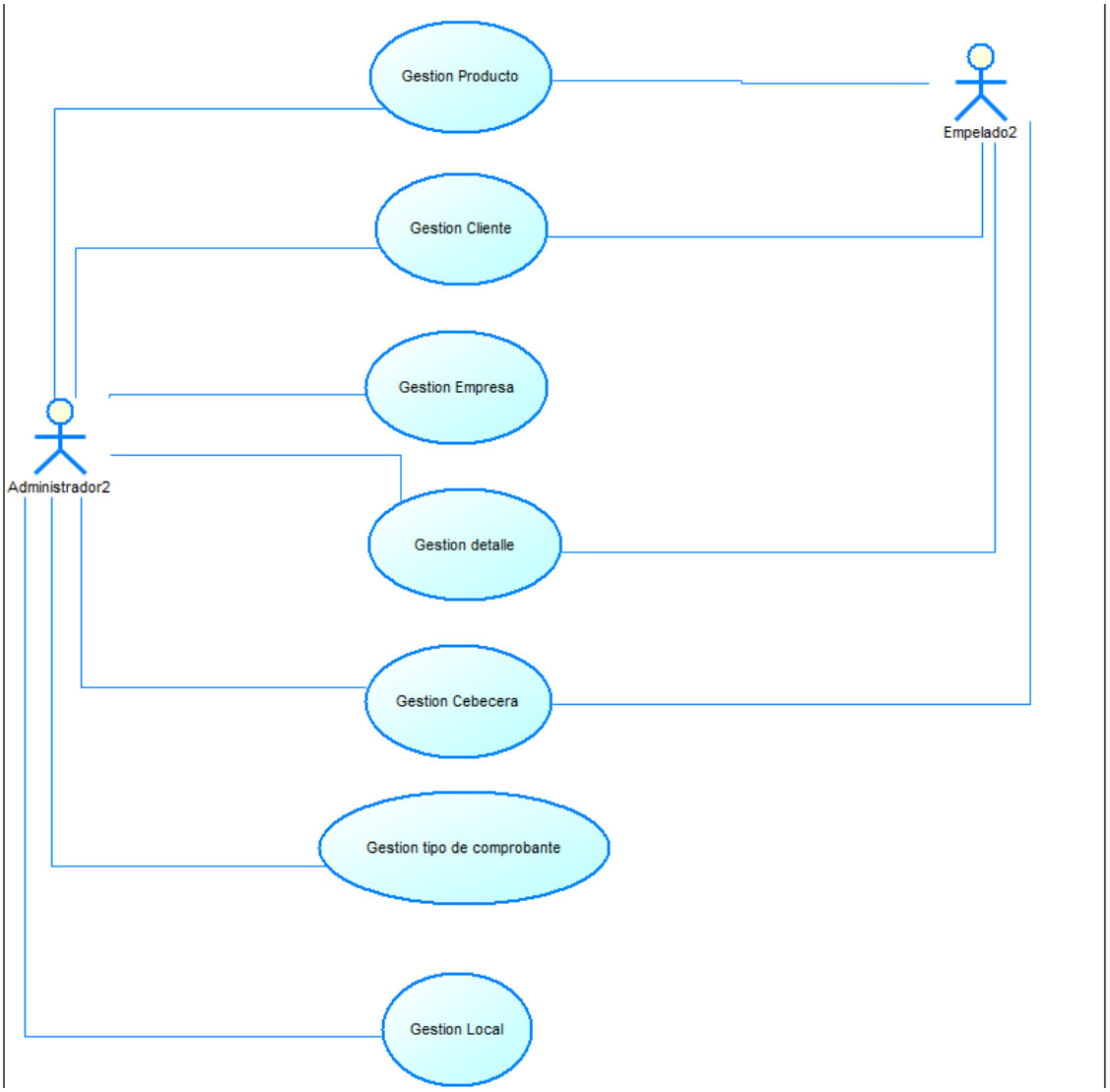
- Garantiza que los requerimientos funcionales sean recolectados en su totalidad antes de comenzar el diseño técnico.
- De forma precisa identifica los participantes y sus obligaciones dentro del sistema.
- Fundamenta los esquemas de secuencia y el modelo físico de información de la base de datos.

A continuación, se presenta una primera aproximación a los actores y casos de uso más relevantes para el prototipo:

Actores Principales

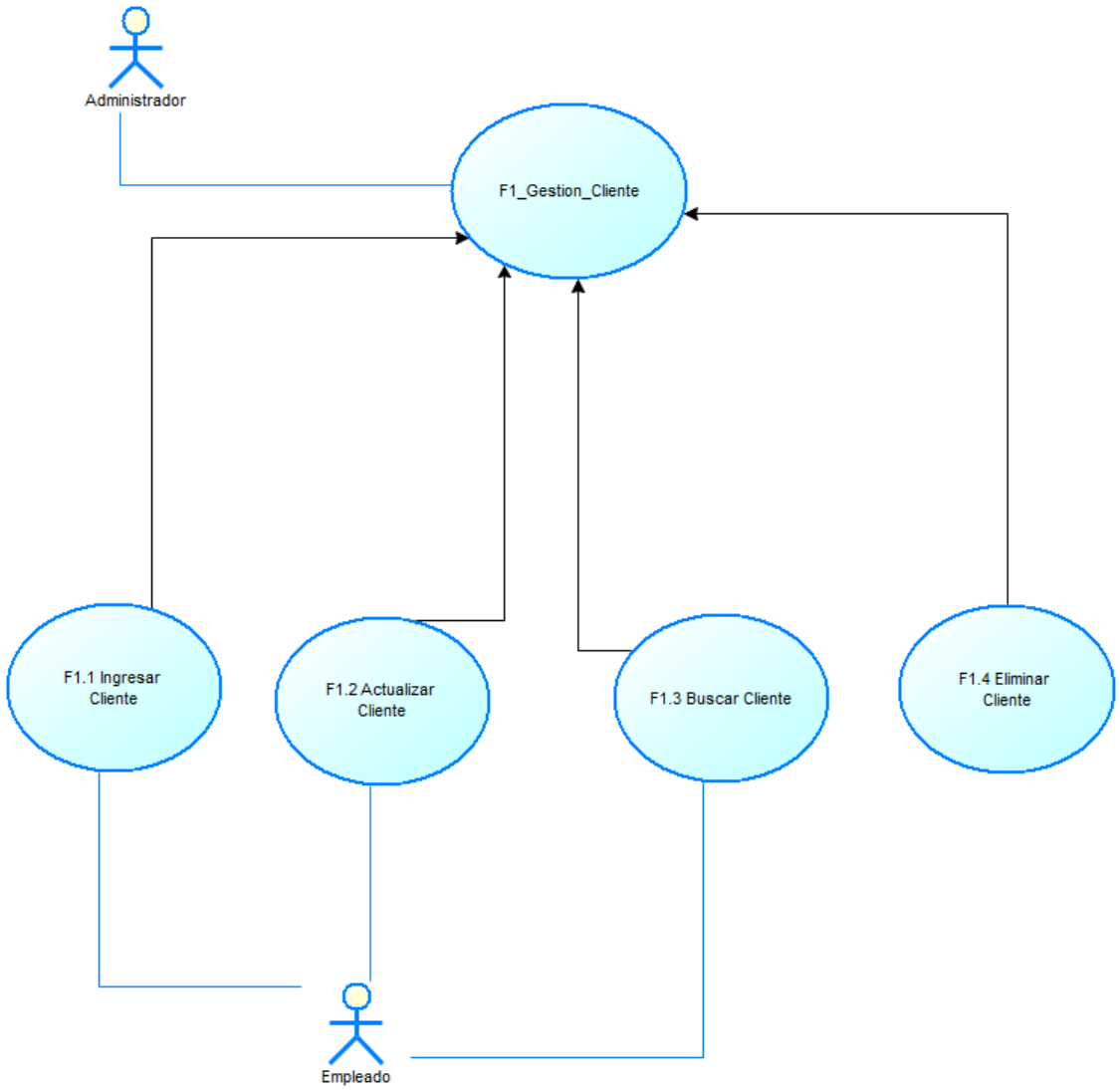
- Administrador: Usuario con permisos para configurar parámetros del sistema, gestionar usuarios y revisar auditorías.
- Empleado de Ventas: Usuario encargado de crear, emitir y consultar facturas, así como de gestionar catálogos de clientes y productos.

Ilustración 1 Caso de Uso General



3.1.1 Caso de uso F1: Gestión Cliente

Ilustración 2 Caso de Uso Cliente



F1.1: Ingresar Cliente

Descripción:

Permite al actor registrar un nuevo cliente en el sistema introduciendo sus datos personales y fiscales, que quedan almacenados en la base de datos.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **"Ingresar Cliente"** en el menú de Gestión de Clientes.
2. El sistema despliega un formulario en blanco con campos:
    - Razón social
    - Tipo de identificación
    - Identificación
    - Dirección
    - Teléfono
    - Correo electrónico
3. El actor ingresa la **razón social** del cliente.
4. El actor ingresa el **Tipo de Identificación**.
5. El actor ingresa la **Identificación**.
6. El actor ingresa la **Dirección**.
7. (Opcional) El actor ingresa **teléfono** y **correo electrónico**.
8. El actor presiona **Guardar**.
9. El sistema valida que la cédula/RUC no exista ya en la base de datos (E1) y que sea un número de identificación válida(E2).
10. El sistema inserta un nuevo registro en la tabla Cliente.

11. El sistema muestra mensaje:

“Cliente ingresado correctamente.”

Flujos Alternativos:

- **A1. Cédula/RUC ya registrado (paso 9):**

    1. El sistema muestra:

“El cliente con cédula/RUC X ya existe. ¿Desea actualizar sus datos?”

    2. El actor elige “Sí” → redirige al flujo F1.2 (Actualizar Cliente).
    3. O bien “No” → vuelve al formulario con los datos ingresados para editar o cancelar.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Fallo de conexión a la base de datos.	Verificar el estado del servidor y volver a intentar.
E2	Validación de formato de cédula/RUC.	Mostrar mensaje de error y solicitar corrección.

F1.2: Actualizar Cliente

Descripción:

Permite editar los datos de un cliente existente para corregir o actualizar información personal o de contacto.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **"Actualizar Cliente"** en el menú de Gestión de Clientes.
2. El sistema muestra una lista de clientes con opción de buscar/filtrar.
3. El actor ubica el cliente deseado y presiona **Editar**.
4. El sistema carga el formulario con los datos actuales del cliente.
5. El actor modifica los campos necesarios (dirección, teléfono, correo, etc.) (E1).
6. El actor presiona **Guardar Cambios**.
7. El sistema actualiza el registro en la base de datos(E2).
8. El sistema muestra mensaje:

"Datos del cliente actualizados correctamente."

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Error de validación de datos.	Indicar campo erróneo y permitir corrección.
E2	Error al actualizar en la base de datos.	Revisar integridad referencial y volver a intentar.

F1.3: Buscar Cliente

Descripción:

Permite al actor localizar clientes mediante criterios de búsqueda y visualizar sus datos.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor ingresa criterios de búsqueda en un campo (Razón Social, identificación).
2. El actor presiona **Buscar** (E2).
3. El sistema ejecuta la consulta SELECT con los filtros.
4. El sistema muestra una tabla de resultados con los datos del cliente (E1).
5. El actor puede seleccionar un registro para ver detalles completos.

Flujos Alternativos:

- **A1. Sin resultados:**

    1. Si no hay coincidencias, el sistema muestra:

"No se encontraron clientes con los criterios ingresados."

    2. El actor puede ajustar criterios y volver a buscar.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Timeout de la consulta.	Optimizar índice en la base de datos.
E2	Falla de conexión.	Reconectar y reintentar búsqueda.

F1.4: Eliminar Cliente

Descripción:

Permite al actor dar de baja un cliente existente, liberando cualquier referencia si no existen facturas asociadas.

Actores:

- Administrador

Flujo Principal:

1. El actor selecciona **"Eliminar Cliente"** en el menú.
2. El sistema muestra lista de clientes y el actor ubica el cliente a eliminar.
3. El actor presiona el botón **Eliminar** junto al registro.
4. El sistema muestra diálogo de confirmación:

"¿Está seguro de eliminar al cliente X? Esta acción no se puede deshacer."

5. El actor confirma la eliminación.
6. El sistema verifica que no existan facturas asociadas (E1).
7. El sistema marca como inactivo el registro en la base de datos.
8. El sistema muestra mensaje:

"Cliente eliminado correctamente."

Flujos Alternativos:

- **A1. Facturas asociadas (paso 6):**
    1. Si hay facturas vinculadas, el sistema muestra:

"El cliente X tiene facturas asociadas. ¿Desea desactivarlo en su lugar?"

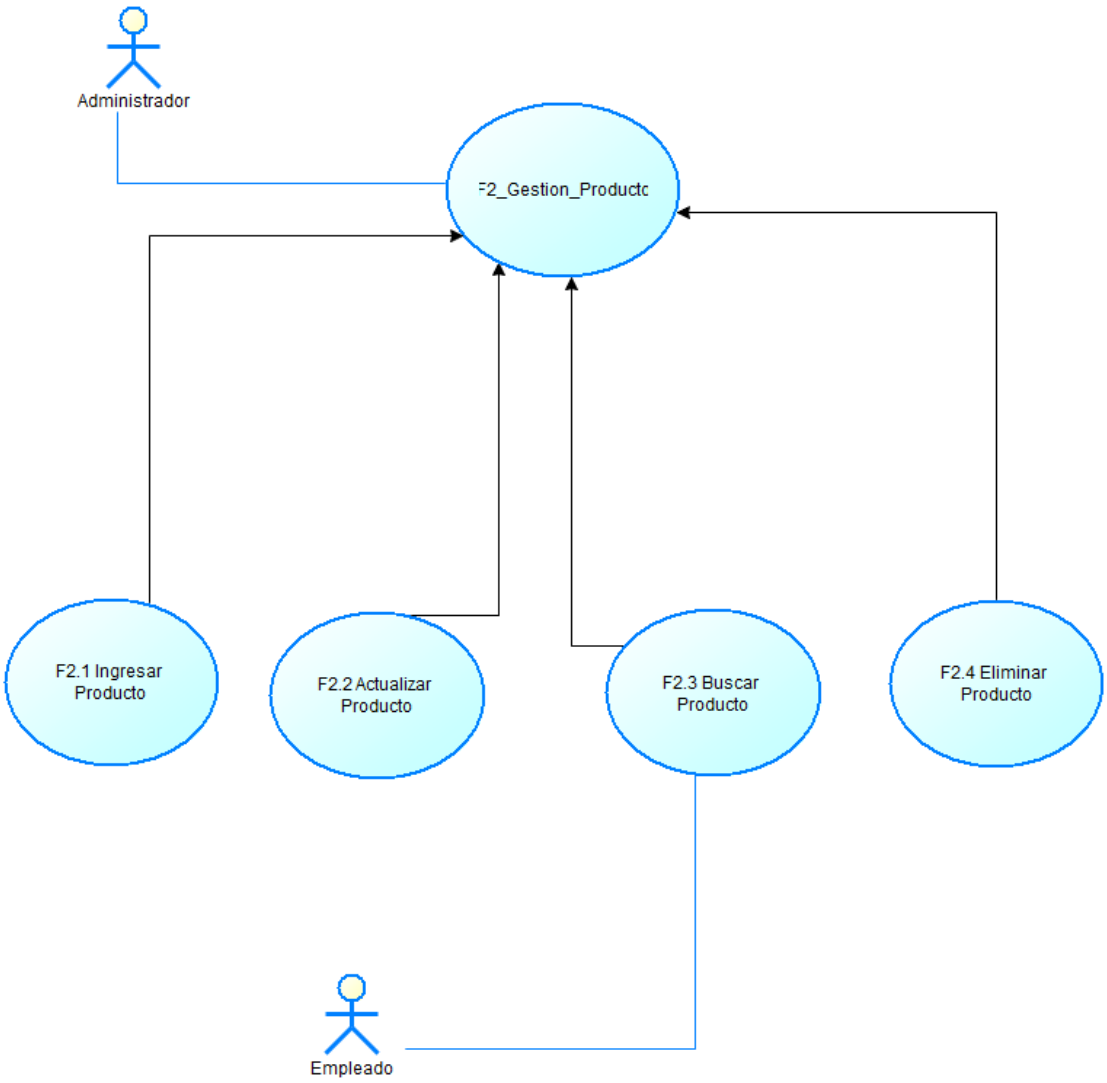
    2. El actor elige "Desactivar" → el sistema marca el cliente como inactivo.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Restricción de integridad referencial en facturas.	Mostrar opción de “desactivar” y registrar auditoría.

3.1.2 Caso de uso F2: Gestión Producto

Ilustración 3 Caso de Uso Productos



F2.1: Ingresar Producto

Descripción:

Registrar un nuevo producto capturando su información básica del producto y almacenarla en la BD.

Actores: Administrador.

Flujo principal:

1. El actor selecciona **“Ingresar Producto”**.
2. El sistema despliega un formulario con campos:
    - Código Interno
    - Código Externo
    - Descripción
    - Precio unitario
    - Cantidad en stock
    - IVA
3. El actor completa los campos obligatorios.
4. El actor pulsa **Guardar**.
5. El sistema valida que el código no exista (E1).
6. El sistema inserta el registro en la tabla Producto.
7. El sistema confirma: “Producto ingresado correctamente.”

Flujo alternativo A1 (código duplicado):

- Si en el paso 5 detecta que el código ya existe, muestra:

“El producto con código X ya existe. ¿Desea actualizarlo?”

- El actor elige “Sí” → redirige a F2.2. O “No” → vuelve al formulario.

Excepciones:

Excepción	**Causa**	**Solución**
E1	Conexión BD caída o timeout.	Reconectar y reintentar.
E2	Formato inválido en precio o stock.	Mostrar error y solicitar corrección.

F2.2: Actualizar Producto

Descripción:

Modificar datos de un producto existente para reflejar cambios de precio, stock o descripción.

Actores: Administrador.

Flujo principal:

1. El actor elige **"Actualizar Producto"**.
2. El sistema muestra lista o buscador de productos.
3. El actor selecciona un producto y pulsa **Editar**.
4. El sistema carga el formulario con datos actuales.
5. El actor modifica campos deseados.
6. El actor pulsa **Guardar Cambios**.
7. El sistema valida código único si fue editado (E1).
8. El sistema actualiza el registro.
9. Mensaje: "Datos del producto actualizados correctamente."

Flujo alternativo A1 (código duplicado):

- Igual al de F2.1, redirige o corrige.

Excepciones:

Excepción	Causa	Solución
E1	Error de integridad referencial.	Revisar dependencias y volver a intentar.
E2	Fallo al escribir en la base.	Revisar permisos y estado de la BD.

F2.3: Buscar Producto

Descripción:

Localizar productos mediante filtros de código o nombre y visualizar sus datos.

Actores: Administrador, Empleado.

Flujo principal:

1. El actor ingresa criterio de búsqueda.
2. Pulsa **Buscar**.
3. El sistema ejecuta SELECT con filtro.
4. Muestra tabla de resultados con columnas básicas.
5. El actor puede seleccionar un producto para ver detalles.

Flujo alternativo A1 (sin resultados):

- Muestra “No se encontraron productos.” y permite refinar búsqueda.

Excepciones:

Excepción	**Causa**	**Solución**
E1	Timeout de consulta.	Optimizar índices y reintentar.

F2.4: Eliminar Producto

Descripción:

Dar de baja un producto, siempre que no haya transacciones asociadas (ventas o facturas).

Actores: Administrador.

Precondiciones: El producto no debe estar vinculado a facturas.

Flujo principal:

1. El actor selecciona **"Eliminar Producto"**.
2. El sistema lista los productos.
3. El actor pulsa **Eliminar** junto al producto deseado.
4. El sistema pide confirmación:

"¿Confirmar eliminación del producto X?"

5. El actor confirma.
6. El sistema verifica referencias (E1).
7. El sistema marca como inactivo.
8. El sistema Muestra "Producto eliminado correctamente."

Flujo alternativo A1 (referencias existentes):

- Muestra:

"No puede eliminarse. Existen facturas asociadas."

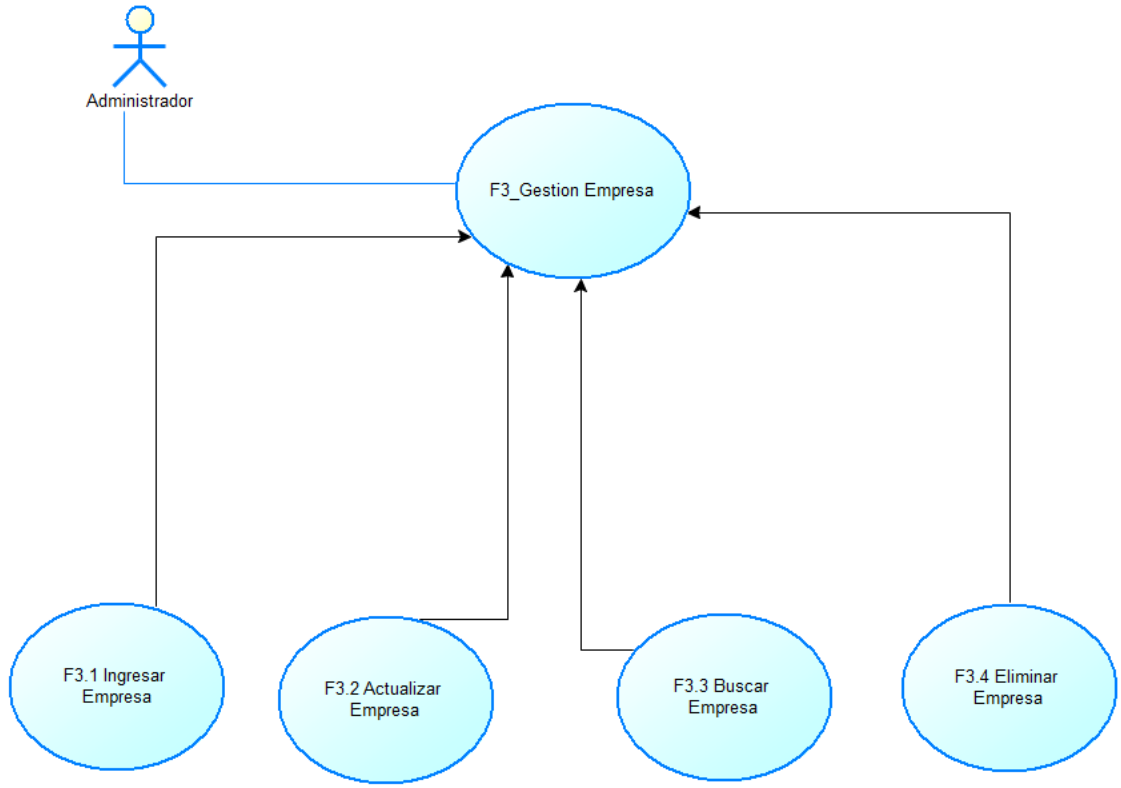
- Ofrece la opción de "Desactivar" en lugar de eliminar.

Excepciones:

Excepción	Causa	Solución
E1	Restricción FK por facturas existentes.	Sugerir desactivación y registrar acción.

3.1.3 Caso de uso F3: Gestión Empresa

Ilustración 4 Caso de Uso Empresa



F3.1: Ingresar Empresa

Descripción:

Registrar una nueva entidad empresarial (matriz o sucursal) capturando sus datos legales y de contacto.

Actor:

- Administrador

Flujo Principal:

1. El Administrador selecciona **"Ingresar Empresa"** en el menú de Gestión de Empresa.
2. El sistema muestra un formulario con campos obligatorios.
3. El Administrador completa cada campo.
4. Presiona **Guardar**.

5. El sistema valida que el RUC no exista (E1).
6. Se inserta el nuevo registro en la tabla Empresa.
7. El sistema muestra:

“Empresa registrada correctamente.”

Flujo Alternativo A1 (RUC duplicado):

- Si el RUC ya está registrado, el sistema muestra:

“El RUC X ya existe. ¿Desea actualizar sus datos?”

- El Administrador puede elegir “Sí” para ir al flujo F3.2 o “No” para corregir el formulario.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Conexión interrumpida con la BD.	Verificar servidor y reconectar.
E2	Formato de RUC inválido.	Mostrar mensaje y solicitar corrección.

F3.2: Actualizar Empresa

Descripción:

Modificar los datos de una empresa existente para mantener la información al día.

Actor:

- Administrador

Flujo Principal:

1. El Administrador selecciona **"Actualizar Empresa"**.
2. El sistema muestra la lista de empresas con opción de buscar/filtrar.
3. El Administrador selecciona una empresa y pulsa **Editar**.
4. El sistema carga el formulario con los datos actuales.
5. El Administrador realiza las modificaciones necesarias (dirección, teléfono, etc.).
6. Presiona **Guardar Cambios**.
7. El sistema valida que el RUC nuevo no duplique otro registro (E1).
8. Se actualiza el registro en la base de datos.
9. El sistema muestra:

"Datos de la empresa actualizados correctamente."

Flujo Alternativo A1 (RUC duplicado):

- Si el RUC ingresado ya corresponde a otra empresa, se muestra mensaje de error y se solicita corrección.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Error de validación de datos.	Indicar campo erróneo y permitir corrección.
E2	Error al actualizar en la base de datos.	Revisar integridad referencial y reintentar.

F3.3: Buscar Empresa

Descripción:

Permite al Administrador ubicar una o varias empresas usando criterios de búsqueda.

Actor:

- Administrador

Flujo Principal:

1. El Administrador introduce criterios de búsqueda (razón social, RUC).
2. Pulsa **Buscar**.
3. El sistema ejecuta consulta SELECT con filtros aplicados.
4. Muestra tabla de resultados con columnas: Razón social, RUC, Estado.
5. El Administrador puede seleccionar un registro para ver detalles completos.

Flujo Alternativo A1 (sin resultados):

- Si no hay coincidencias, el sistema muestra:

"No se encontraron empresas con los criterios ingresados."

- Permite refinar criterios de búsqueda.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Timeout de consulta.	Revisar índices y optimizar consulta.
E2	Falla de conexión.	Reconectar y reintentar búsqueda.

F3.4: Eliminar Empresa

Descripción:

Dar de baja una empresa del catálogo, siempre que no existan registros de facturación asociados.

Actor:

- Administrador

Flujo Principal:

1. El Administrador selecciona **"Eliminar Empresa"**.
2. El sistema presenta la lista de empresas.
3. El Administrador pulsa **Eliminar** junto a la empresa deseada.
4. El sistema muestra confirmación:

"¿Está seguro de eliminar la empresa X? Esta acción es irreversible."

5. El Administrador confirma.
6. El sistema verifica que no existan facturas vinculadas (E1).
7. El sistema marca como inactiva la empresa.
8. Se muestra mensaje:

"Empresa eliminada correctamente."

Flujo Alternativo A1 (facturas asociadas):

- Si existen facturas, el sistema ofrece:

"No es posible eliminar. Tiene facturas asociadas. ¿Desea inactivar la empresa?"

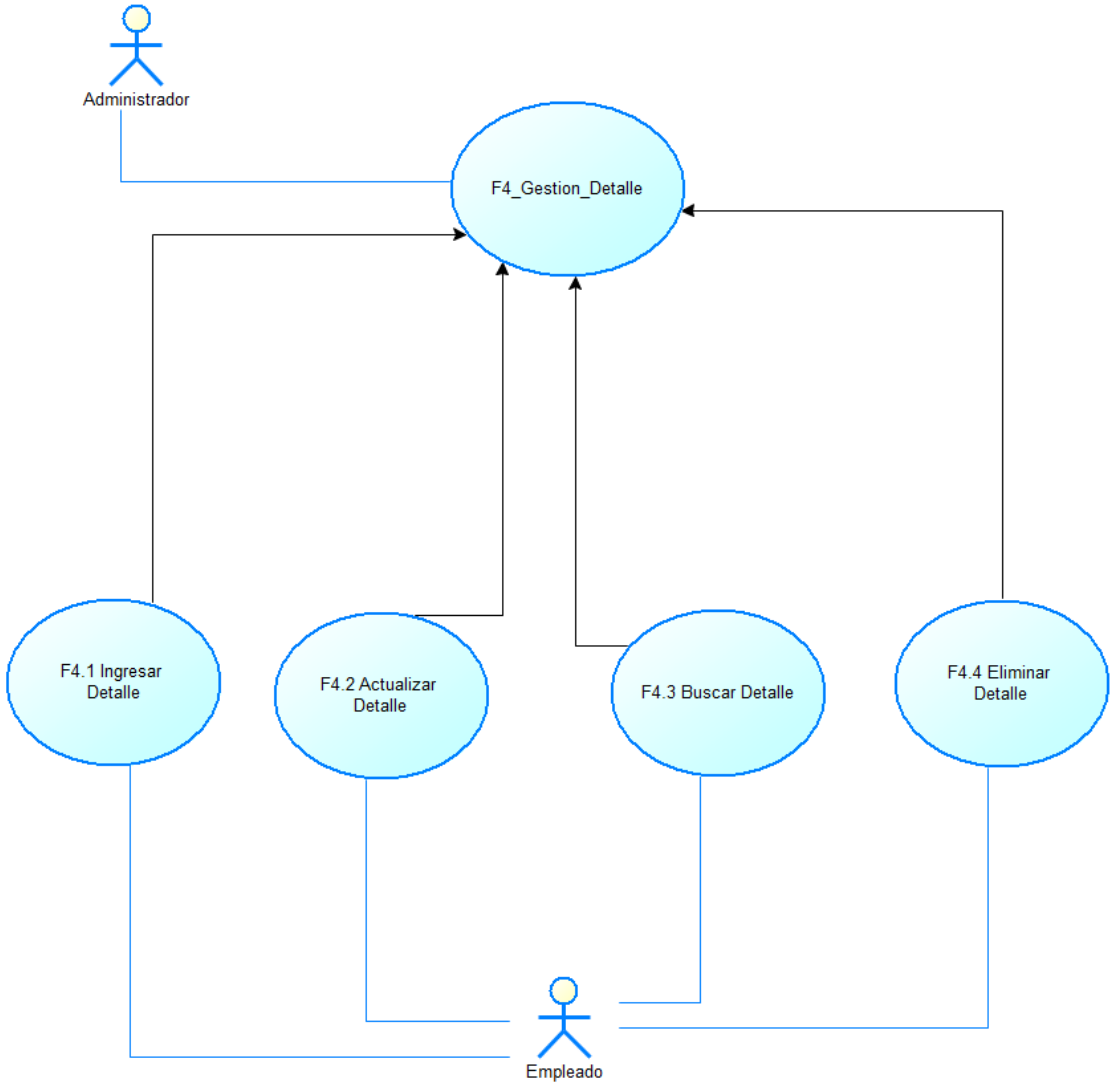
- El Administrador puede aceptar y el sistema marca la empresa como inactiva.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Restricción de integridad referencial por facturas.	Ofrecer inactivación y registrar auditoría.

3.1.4 Caso de uso F4: Gestión Detalle

Ilustración 5 Caso de Uso Detalles



F4.1: Ingresar Detalle

Descripción:

Añadir una nueva línea de detalle a una factura existente, especificando el producto, cantidad, precio unitario e impuestos correspondientes.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **“Ingresar Detalle”** en el módulo de Detalle de Factura.
2. El sistema pide al actor después de generar la cabecera de la factura a la cual agregar el detalle.
3. El actor selecciona la factura destino.
4. El sistema despliega un formulario con los campos que tiene el detalle.
5. El actor selecciona el producto y la **cantidad**.
6. El actor presiona **Agregar Línea**.
7. El sistema valida:
    - Que la cantidad no supere el stock disponible (E1).
    - Que los datos estén completos (E2).
8. El sistema inserta un nuevo registro en la tabla Detalle, vinculándolo a la cabecera.
9. El sistema muestra:

“Línea de detalle agregada correctamente.”

10. Se actualiza el total de la factura.

Flujos Alternativos:

- **A1. Producto sin stock suficiente (paso 8):**
    1. El sistema muestra:

“No hay suficiente stock para el producto seleccionado.”

    2. El actor puede reducir la cantidad o cancelar la operación.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Stock insuficiente en inventario.	Mostrar advertencia y solicitar nueva cantidad.
E2	Conexión caída o timeout en BD.	Verificar conexión y reintentar.

F4.2: Actualizar Detalle

Descripción:

Modificar los datos de una línea existente en la factura (p. ej., cantidad, precio unitario o impuestos).

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **"Actualizar Detalle"**.
2. El sistema muestra la lista de facturas; el actor elige una.
3. Se despliega la tabla de líneas de detalle de esa factura.
4. El actor pulsa **Editar** junto a la línea deseada.
5. El sistema carga un formulario con los valores actuales.
6. El actor modifica **cantidad**, **precio** o **impuesto**.
7. El actor presiona **Guardar Cambios**.
8. El sistema valida nuevamente stock y formatos (E1, E2).
9. El sistema actualiza el registro en Detalle.
10. El sistema muestra:

"Detalle de factura actualizado correctamente."

11. Totales de factura recalculados.

Flujos Alternativos:

- **A1. Stock insuficiente (paso 8):** Similar al A1 de F4.1.
- **A2. Datos incompletos:** Se resaltan campos faltantes y solicita corrección.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Validación de stock fallida.	Ajustar cantidad o notificar al inventario.
E2	Error en actualización BD.	Revisar integridad referencial y reintentar.

F4.3: Buscar Detalle

Descripción:

Localizar líneas de detalle de factura usando filtros (número de factura, producto, rango de fechas).

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor elige **"Buscar Detalle"**.
2. El sistema muestra campos de filtro:
    - Número de factura
    - Código o nombre de producto
    - Fecha de emisión de factura (rango)
3. El actor ingresa criterios y pulsa **Buscar**.
4. El sistema ejecuta SELECT con los filtros.
5. Muestra tabla.
6. El actor puede seleccionar una línea para ver más detalles.

Flujos Alternativos:

- **A1. Sin resultados:**

"No se encontraron líneas con esos criterios."
Permite refinar filtros.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Consulta muy pesada (timeout).	Optimizar índices y volver a buscar.

F4.4: Eliminar Detalle

Descripción:

Borrar una línea de detalle de una factura, recalculando posteriormente los totales.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **"Eliminar Detalle"**.
2. El sistema pide seleccionar la factura y muestra sus líneas.
3. El actor pulsa **Eliminar** junto a la línea deseada.
4. El sistema muestra diálogo de confirmación:

"¿Eliminar línea de producto X de la factura Y?"

5. El actor confirma.
6. El sistema elimina el registro en Detalle.
7. Se recalculan subtotales e impuestos de la factura.
8. El sistema muestra:

"Línea de detalle eliminada correctamente."

Flujos Alternativos:

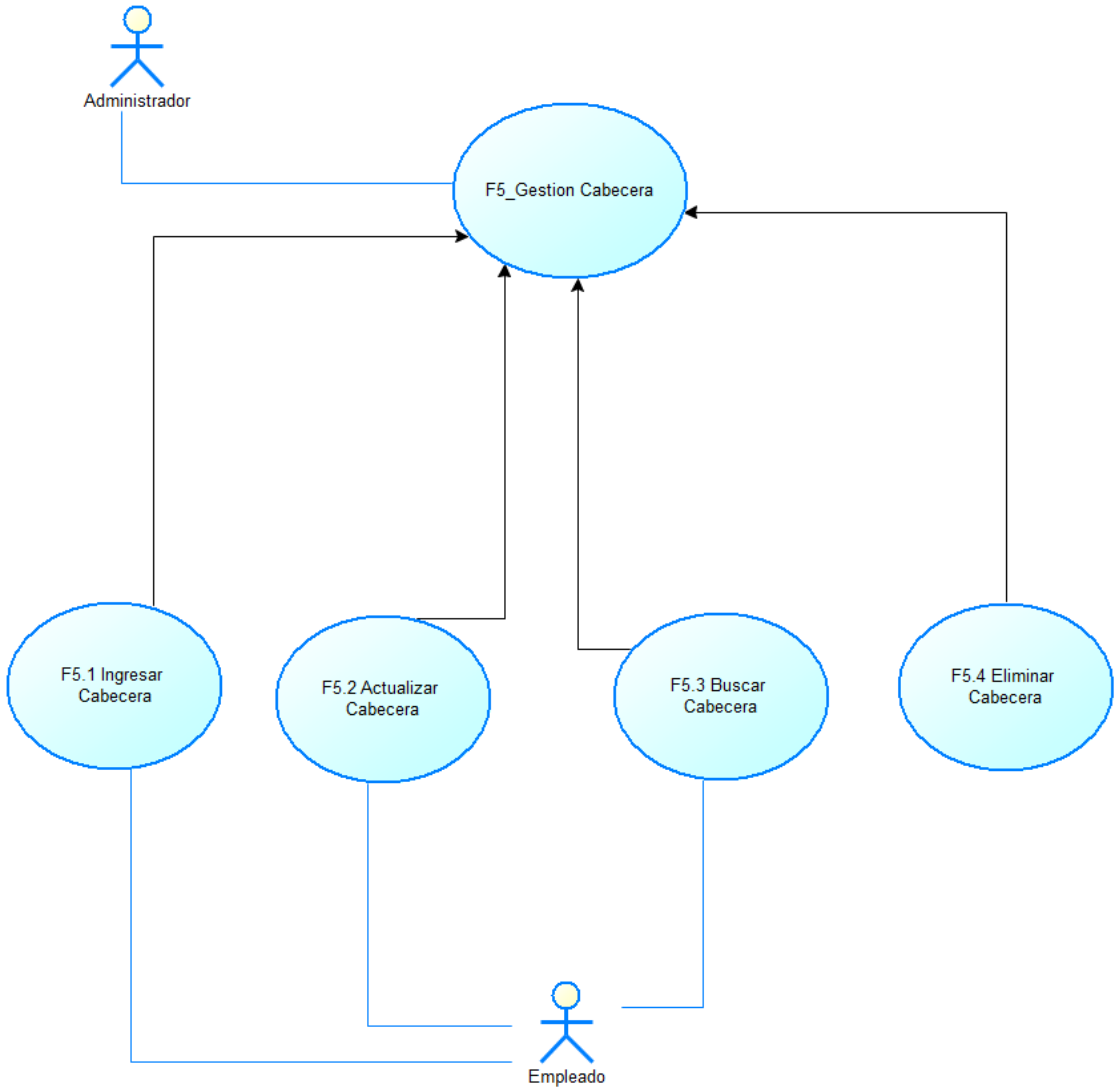
- **A1. Dependencias de stock u otros procesos:**
  Si la eliminación afecta a otras tablas, mostrar advertencia y requerir autorización especial.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Error de integridad referencial en DB.	Verificar referencias y ajustar lógica.

3.1.5 Caso de uso F5: Gestión Cabecera

Ilustración 6 Caso de Uso Cabecera



F5.1: Ingresar Cabecera

Descripción:

Registrar la cabecera de una nueva factura, asociándose a un cliente y a la empresa emisora, e iniciando el proceso de facturación.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **“Ingresar Cabecera”** en el módulo de Cabecera de Factura.
2. El sistema despliega un formulario con campos obligatorios.
3. El actor completa o verifica los campos.
4. El actor presiona **Guardar**.
5. El sistema valida que el número de factura sea único (E1) y que el cliente y empresa existan.
6. El sistema inserta un nuevo registro en la tabla Cabecera.
7. El sistema muestra:

“Cabecera de factura ingresada correctamente.”

Flujo Alternativo A1 (número duplicado):

- Si el número de factura ya existe, el sistema muestra:

“El número de factura X ya está registrado. Genere otro número.”

- El actor modifica el número o acepta la generación automática.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Fallo en la validación de unicidad del número.	Verificar la regla de generación y reintentar.
E2	La conexión con la base de datos caía.	Reconectar al servidor y volver a intentar.

F5.2: Actualizar Cabecera

Descripción:

Modificar los datos generales de una factura existente antes de cerrarla (p. ej., cambiar forma de pago).

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **"Actualizar Cabecera"**.
2. El sistema muestra una lista o buscador de facturas abiertas.
3. El actor elige la factura y pulsa **Editar**.
4. El sistema carga el formulario con los datos actuales.
5. El actor modifica los campos permitidos (forma de pago, fecha, empresa).
6. El actor presiona **Guardar Cambios**.
7. El sistema valida la unicidad si el número fue editado (E1).
8. El sistema actualiza el registro en Cabecera.
9. El sistema muestra:

"Cabecera de factura actualizada correctamente."

Flujos Alternativos:

- **A1. Número duplicado (paso 7):** Igual que en F5.1.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Error de integridad al actualizar la factura.	Verificar estado de la transacción y reintentar.
E2	Conexión interrumpida.	Reconectar y reintentar.

F5.3: Buscar Cabecera

Descripción:

Localizar cabeceras de factura mediante filtros (número, cliente, fecha, empresa) y visualizar datos generales.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor elige **"Buscar Cabecera"**.
2. El sistema muestra campos de filtro.
3. El actor ingresa criterios y pulsa **Buscar**.
4. El sistema ejecuta SELECT con los filtros.
5. Se muestra tabla con columnas.
6. El actor puede seleccionar una cabecera para ver o editar.

Flujo Alternativo A1 (sin resultados):

- Muestra:

"No se encontraron facturas con esos criterios."

- Permite refinar búsqueda.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Consulta pesada (timeout).	Optimizar índices; reintentar.

F5.4: Eliminar Cabecera

Descripción:

Eliminar o inactivar la cabecera de una factura que no contenga detalles o que esté abierta, liberando el número para uso futuro.

Actores:

- Administrador

Flujo Principal:

1. El actor selecciona **"Eliminar Cabecera"**.
2. El sistema presenta lista de cabeceras.
3. El actor pulsa **Eliminar** junto a la factura deseada.
4. El sistema muestra confirmación:

"¿Eliminar cabecera de la factura X? Se eliminarán los datos generales."

5. El actor confirma.
6. El sistema verifica que no haya líneas de detalle (E1).
7. El sistema marca como inactiva la cabecera.
8. El sistema muestra:

"Cabecera eliminada correctamente."

Flujos Alternativos:

- **A1. Detalles asociados (paso 6):**

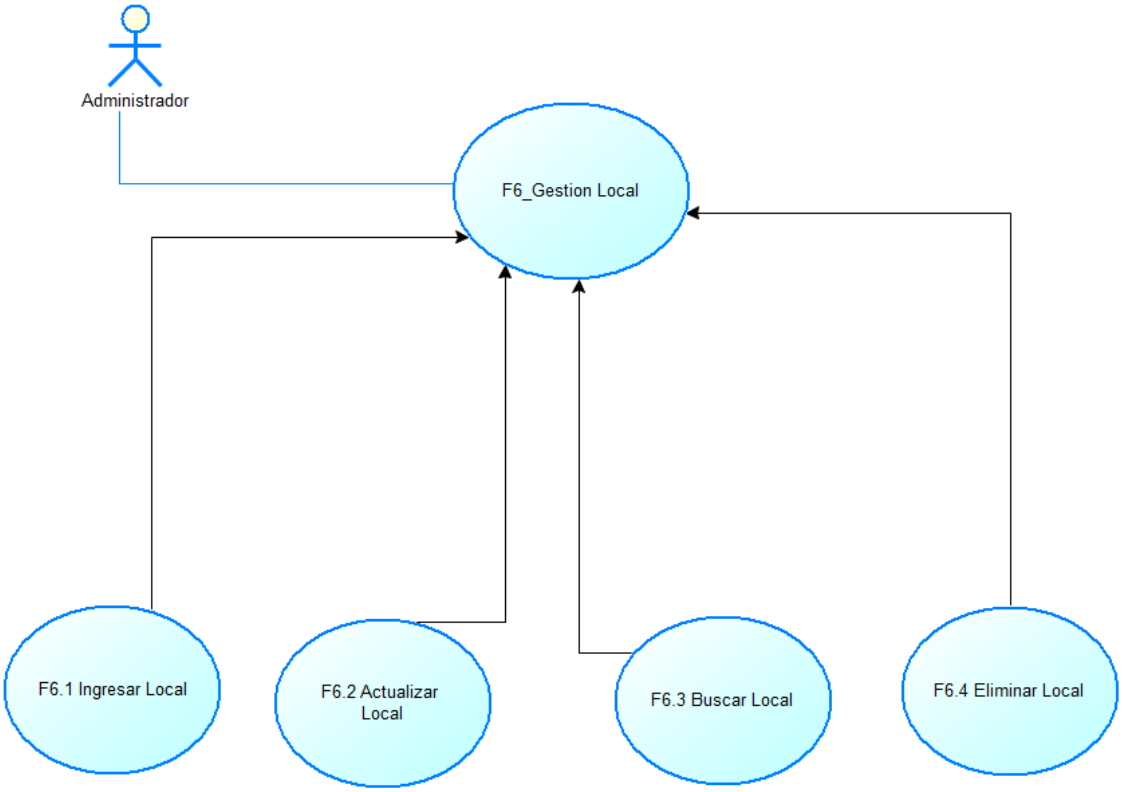
"No se puede eliminar. La factura tiene líneas de detalle. ¿Desea anularla en su lugar?"

Excepciones:

Excepción	Causa	Solución propuesta
E1	Integridad referencial bloqueada por detalles.	Ofrecer anulación y registrar auditoría.

3.1.6 Caso de uso F6: Gestión Local

Ilustración 7 Caso de Uso Local



F6.1: Ingresar Local

Descripción:

Registrar un nuevo local (sucursal) vinculándolo a una empresa existente, definiendo su dirección, código de establecimiento y estado.

Actor:

- Administrador

Flujo Principal:

1. El Administrador selecciona **"Ingresar Local"** en el menú de Gestión de Local.
2. El sistema muestra un formulario con campos obligatorios.
3. El Administrador completa los campos.
4. Presiona **Guardar**.

5. El sistema valida que:
    - El EmpresaId exista en la tabla Empresa.
    - No exista ya un local con el mismo EmpresaId y CodigoEstablecimiento.
6. El sistema inserta un nuevo registro en Local.
7. El sistema muestra:

"Local registrado correctamente (ID: X)."

Flujo Alternativo A1 – Código duplicado:

- Si ya hay un registro con la misma empresa y código de establecimiento, el sistema muestra:

"Ya existe un local para esta empresa con el código de establecimiento X."

- Ofrece al Administrador corregir el código o cancelar la operación.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Conexión interrumpida con la base de datos.	Verificar servidor y reintentar.
E2	Error referencial (EmpresaId inexistente).	Seleccionar una empresa válida.

F6.2: Actualizar Local

Descripción:

Modificar los datos de un local existente para actualizar dirección, código o estado.

Actor:

- Administrador

Flujo Principal:

1. El Administrador selecciona **"Actualizar Local"**.
2. El sistema muestra la lista de locales con búsqueda/filtro por empresa, código o estado.
3. El Administrador elige un local y pulsa **Editar**.
4. El sistema carga el formulario con los datos actuales.
5. El Administrador modifica los campos requeridos.
6. Presiona **Guardar Cambios**.
7. El sistema valida (EmpresaId, CodigoEstablecimiento) si el código cambió.
8. El sistema actualiza el registro en Local.
9. Mensaje:

"Local actualizado correctamente."

Flujo Alternativo A1 – Código duplicado:

- Igual a A1 de F6.1, solicita corrección antes de guardar.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Error de validación de datos.	Mostrar campos erróneos y permitir corrección.
E2	Fallo al actualizar la tabla.	Verificar integridad referencial y reintentar.

F6.3: Buscar Local

Descripción:

Localizar locales mediante filtros de empresa, código de establecimiento o estado y visualizar sus atributos.

Actor:

- Administrador

Flujo Principal:

1. El Administrador selecciona **"Buscar Local"**.
2. El sistema ofrece campos de filtro:
    - Empresa (dropdown)
    - Código de establecimiento (texto)
3. El Administrador ingresa criterios y pulsa **Buscar**.
4. El sistema ejecuta SELECT con los filtros aplicados.
5. Muestra una tabla con columnas: LocalId, EmpresaId, Direccion, CodigoEstablecimiento, Estado.
6. El Administrador puede seleccionar un registro para ver detalles o editar.

Flujo Alternativo A1 – Sin resultados:

- El sistema muestra:

"No se encontraron locales con esos criterios."

- Permite refinar los filtros.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Timeout de la consulta.	Optimizar índices y reintentar.

F6.4: Eliminar Local

Descripción:

Eliminar o inactivar un local, siempre que no existan facturas o dependencias asociadas.

Actor:

- Administrador

Flujo Principal:

1. El Administrador selecciona **"Eliminar Local"**.
2. El sistema lista los locales y el Administrador pulsa **Eliminar** junto al deseado.
3. El sistema muestra diálogo de confirmación:

"¿Eliminar el local código X de la empresa Y?"

4. El Administrador confirma.
5. El sistema verifica integridad referencial (no haya facturas) – paso E1.
6. El sistema marca como inactivo (Estado = 0).
7. Mensaje:

"Local eliminado correctamente."

Flujo Alternativo A1 – Dependencias existentes:

- Si existen facturas asociadas, el sistema muestra:

"No se puede eliminar. Hay facturas relacionadas. ¿Desea inactivar el local?"

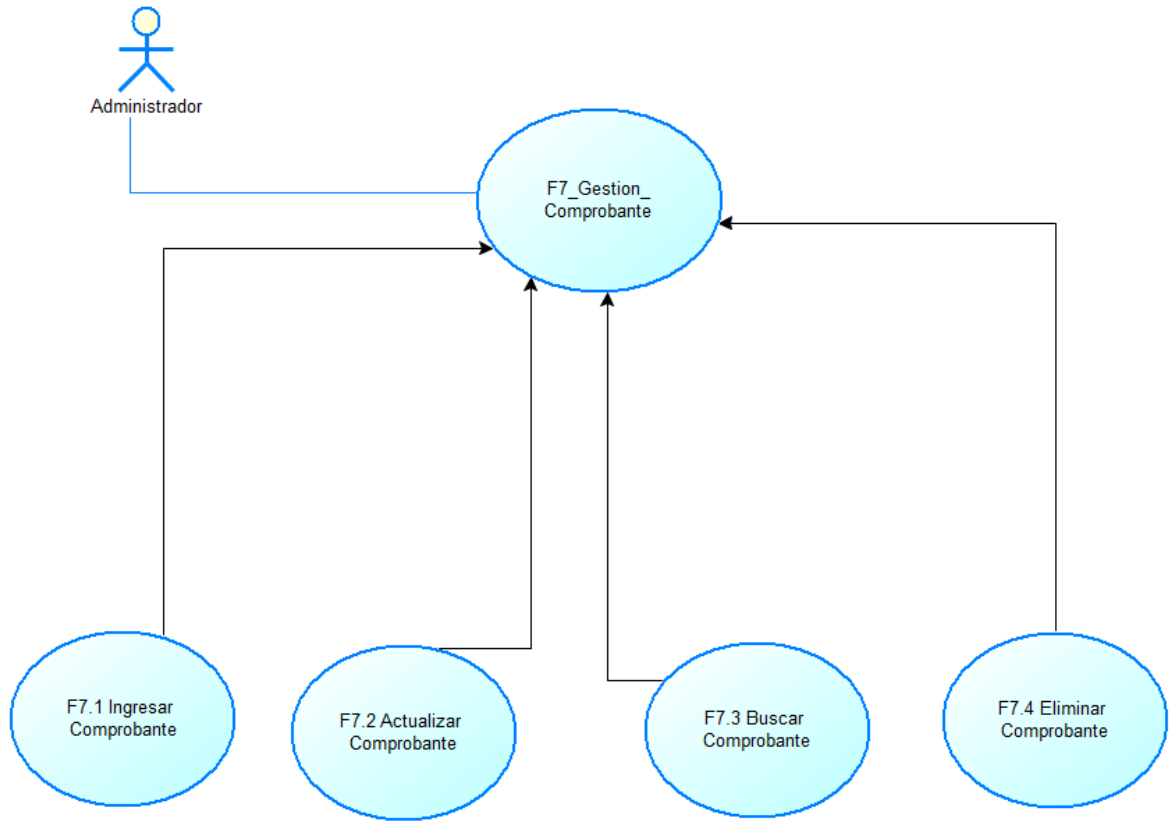
- Permite al Administrador optar por inactivación.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Restricción de integridad en facturas.	Ofrecer inactivación y registrar auditoría.

3.1.7 Caso de uso F7: Gestión Comprobante

Ilustración 8 Caso de Uso Comprobante



F7.1: Ingresar Tipo de Comprobante

Descripción:

Registrar un nuevo tipo de comprobante asociándolo a un local y punto de emisión, definiendo su serie, descripción y estado.

Actores:

- Administrador

Flujo Principal:

1. El actor selecciona **"Ingresar Tipo de Comprobante"** en el menú de configuración.
2. El sistema muestra un formulario con los campos.
3. El actor completa todos los campos obligatorios.

4. Presiona **Guardar**.
5. El sistema valida la unicidad de la combinación (E1).
6. El sistema inserta un nuevo registro en TipoComprobante.
7. El sistema muestra:

"Tipo de comprobante registrado correctamente."

Flujo Alternativo A1 – Duplicidad detectada:

- Si la combinación clave ya existe, el sistema muestra:

"Este tipo de comprobante ya existe para el local y punto de emisión seleccionados."

- Permite corregir los datos o cancelar la operación.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Restricción de unicidad violada.	Revisar datos y ajustar antes de guardar.
E2	Conexión caída a la base de datos.	Verificar servidor y reintentar.

F7.2: Actualizar Tipo de Comprobante

Descripción:

Modificar los parámetros de un tipo de comprobante existente (serie, descripción, estado o reasignar a otro local/punto).

Actores:

- Administrador

Flujo Principal:

1. El actor selecciona **"Actualizar Tipo de Comprobante"**.
2. El sistema presenta un buscador o lista de tipos de comprobantes.
3. El actor elige un registro y pulsa **Editar**.
4. El sistema carga el formulario con los datos actuales.
5. El actor modifica los campos necesarios.
6. Presiona **Guardar Cambios**.
7. El sistema valida unicidad si se cambió alguna clave compuesta (E1).
8. El sistema actualiza el registro en la tabla TipoComprobante.
9. El sistema muestra:

"Tipo de comprobante actualizado correctamente."

Flujo Alternativo A1 – Violación de unicidad:

- Igual que en F7.1, solicita corrección.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Error de validación de datos.	Mostrar campos erróneos para corrección.
E2	Error de actualización en la base de datos.	Revisar integridad referencial y reintentar.

F7.3: Buscar Tipo de Comprobante

Descripción:

Localizar tipos de comprobantes mediante filtros (tipo, local, punto de emisión, serie o estado) y visualizar sus atributos.

Actores:

- Administrador
- Empleado

Flujo Principal:

1. El actor selecciona **"Buscar Tipo de Comprobante"**.
2. El sistema ofrece filtros:
3. El actor ingresa criterios y pulsa **Buscar**.
4. El sistema ejecuta SELECT con los filtros aplicados.
5. Muestra una tabla con las columnas.
6. El actor puede seleccionar un registro para ver detalles o editar.

Flujo Alternativo A1 – Sin resultados:

- El sistema muestra:

"No se encontraron tipos de comprobante con esos criterios."

- Permite ajustar los filtros.

Excepciones:

Excepción	**Causa**	**Solución propuesta**
E1	Consulta muy pesada (timeout).	Optimizar índices y volver a buscar.

F7.4: Eliminar Tipo de Comprobante

Descripción:

Eliminar o inactivar un tipo de comprobante, siempre que no haya facturas emitidas asociadas a él.

Actores:

- Administrador

Flujo Principal:

1. El actor selecciona **"Eliminar Tipo de Comprobante"**.
2. El sistema muestra la lista de tipos de comprobante.
3. El actor pulsa **Eliminar** junto al registro deseado.
4. El sistema muestra confirmación:

"¿Eliminar tipo de comprobante X (serie Y)?"

5. El actor confirma.
6. El sistema verifica integridad referencial (E1).
7. El sistema marca como inactivo (Estado = 0).
8. El sistema muestra:

"Tipo de comprobante eliminado correctamente."

Flujo Alternativo A1 – Referencias existentes:

- Si hay facturas que usan este comprobante, el sistema muestra:

"No se puede eliminar. Existen facturas asociadas. ¿Desea eliminar?"

- El actor puede optar por eliminar.

Excepciones:

Excepción	Causa	Solución propuesta
E1	Restricción de integridad referencial por facturas.	Ofrecer inactivación y registrar auditoría.

3.2 Requerimientos Funcionales y no funcionales

Durante el desarrollo del sistema de facturación electrónica, la correcta definición de los requerimientos constituyó una de las etapas más críticas del proyecto, ya que permitió identificar con claridad las funcionalidades que el sistema debía ofrecer, así como las condiciones bajo las cuales debía operar. En el marco de la ingeniería de software, los requerimientos se clasificaron en dos tipos principales: funcionales y no funcionales.

Los requerimientos funcionales describieron las acciones específicas que el sistema debía realizar, es decir, las funciones que permitieron a los usuarios cumplir sus objetivos dentro de la plataforma. En el caso del presente proyecto, estos requerimientos abarcaron desde el registro y autenticación de usuarios, hasta la gestión integral de la facturación electrónica, clientes, productos, empresas y comprobantes, incluyendo la generación de archivos XML conforme a los lineamientos técnicos establecidos por el Servicio de Rentas Internas (SRI) del Ecuador.

Por su parte, los requerimientos no funcionales se refirieron a las cualidades del sistema, tales como su rendimiento, seguridad, usabilidad, portabilidad y mantenibilidad. Estos aspectos no definieron directamente lo que el sistema hacía, sino cómo debía comportarse para asegurar confiabilidad y eficiencia. En este proyecto, se consideraron aspectos clave como la arquitectura en tres capas, el diseño responsivo, la compatibilidad tecnológica con .NET y SQL Server, la escalabilidad para futuras extensiones, y el cumplimiento de normativas legales.

La identificación de estos requerimientos se llevó a cabo mediante el análisis de los procesos internos actuales de Brother Corp, el estudio de los estándares técnicos del SRI, y la aplicación de buenas prácticas en ingeniería de software. El cumplimiento de dichos requerimientos aseguró que el sistema no solo fuera funcional, sino también robusto, seguro y preparado para evolucionar conforme a las necesidades futuras de la empresa.

A continuación, se presentan los requerimientos funcionales y no funcionales del sistema desarrollado, organizados en tablas estructuradas para facilitar su análisis, validación y trazabilidad.

3.2.1 Requerimientos Funcionales

Código	Nombre	Descripción
RF01	Autenticación y control de acceso	El sistema debía permitir a los usuarios iniciar sesión con credenciales válidas y restringir funcionalidades según los roles (administrador, Usuario).
RF02	Gestión de clientes	Registrar, consultar, actualizar, buscar y eliminar/inactivar clientes con validación de identificación única (RUC o cédula).
RF03	Gestión de productos	Permitir el manejo completo del catálogo de productos: ingresar, editar, consultar, buscar y eliminar/inactivar, validando códigos únicos.
RF04	Gestión de empresas	Registrar y mantener datos legales y de contacto de empresas (razón social, RUC, etc.), con verificación de duplicados.
RF05	Gestión de locales	Registrar, modificar, consultar e inactivar locales (sucursales) por empresa y código de establecimiento.
RF06	Gestión de tipos de comprobante	Asociar tipos de comprobante a locales y puntos de emisión,

		permitiendo su edición y desactivación.
RF07	Generación de cabecera de factura	Crear la cabecera de la factura incluyendo cliente, fecha, empresa, forma de pago, y número único.
RF08	Gestión de detalle de factura	Añadir, editar o eliminar líneas de productos con validación de stock, precios, impuestos y vinculación a la cabecera correspondiente.
RF09	Emisión de factura completa	Finalizar la factura calculando totales, IVA, descuentos, y generando un XML conforme al esquema del SRI.
RF10	Búsqueda avanzada	Realizar búsquedas filtradas de clientes, productos, facturas, locales y comprobantes mediante múltiples criterios.
RF11	Anulación e inactivación lógica	Si un registro tenía dependencias, debía ser marcado como inactivo en lugar de ser eliminado físicamente.
RF12	Registro de errores y excepciones	Capturar eventos de error en la operación del sistema y mostrar mensajes adecuados.
RF13	Validación de datos obligatorios	Todos los formularios debían validar campos

		obligatorios y formatos válidos.
RF14	Generar el XML	Generar el XML compatible con el SRI.
RF15	Auditoría de acciones críticas	Registrar en una bitácora los eventos importantes como eliminación/inactivación de datos, login, y creación de facturas.

3.2.2 Requerimientos No Funcionales

Código	Nombre	Descripción
RNF01	Arquitectura en tres capas	El sistema debía estar construido en capas (presentación, lógica,

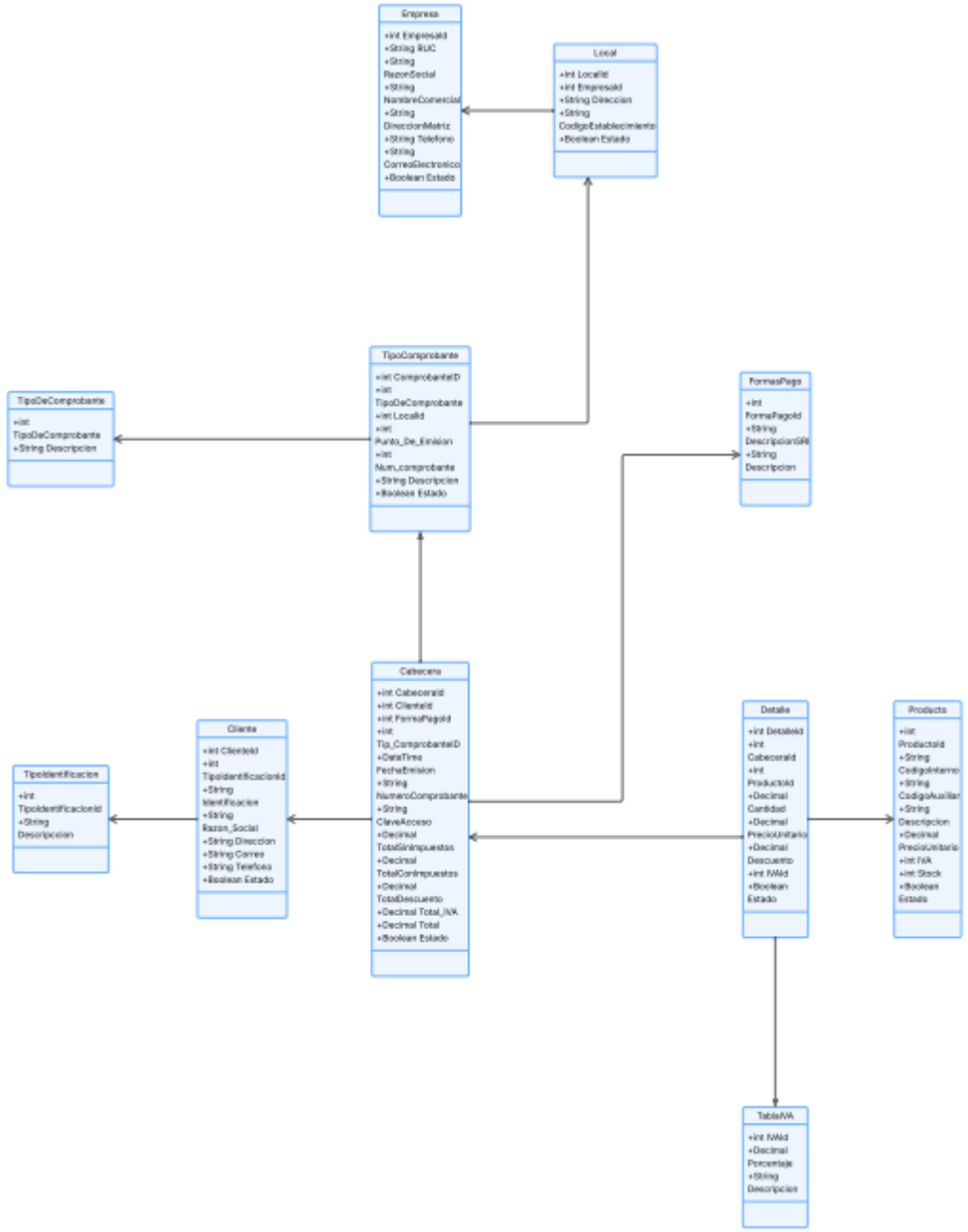
		datos) para mantener modularidad y escalabilidad.
RNF02	Interfaz responsiva	Debía ser accesible desde distintos dispositivos, adaptándose automáticamente al tamaño de pantalla.
RNF03	Desempeño	Las operaciones críticas debían ejecutarse en menos de 5 segundos.
RNF04	Seguridad de acceso	Toda comunicación con el servidor debía estar cifrada y las contraseñas encriptadas.
RNF05	Protección contra ataques comunes	El sistema debía estar protegido contra inyecciones SQL.
RNF06	Trazabilidad	Las acciones sensibles debían quedar registradas con usuario, fecha y descripción.
RNF07	Compatibilidad tecnológica	Compatible con SQL Server 2019+, .NET Framework 4.8.1 y navegadores modernos.
RNF08	Mantenibilidad del código	El sistema debía seguir buenas prácticas de codificación y contar con pruebas automatizadas.
RNF09	Escalabilidad funcional	La estructura del prototipo debía permitir agregar

		funcionalidades nuevas sin rediseños.
RNF10	Respaldo de base de datos	El sistema debía permitir respaldos periódicos para prevenir la pérdida de información.
RNF11	Cumplimiento normativo del SRI	Toda factura debía seguir los requerimientos técnicos establecidos por el SRI.
RNF12	Tolerancia a errores y fallos	El sistema debía manejar fallos con mensajes claros, evitando pérdida de datos.
RNF13	Independencia del cliente	El sistema debía funcionar desde navegador web, sin instalaciones locales.
RNF14	Actualización y despliegue sencillo	Las nuevas versiones debían desplegarse en el servidor sin afectar a los usuarios activos.

3.3 Modelo de la Base de Datos

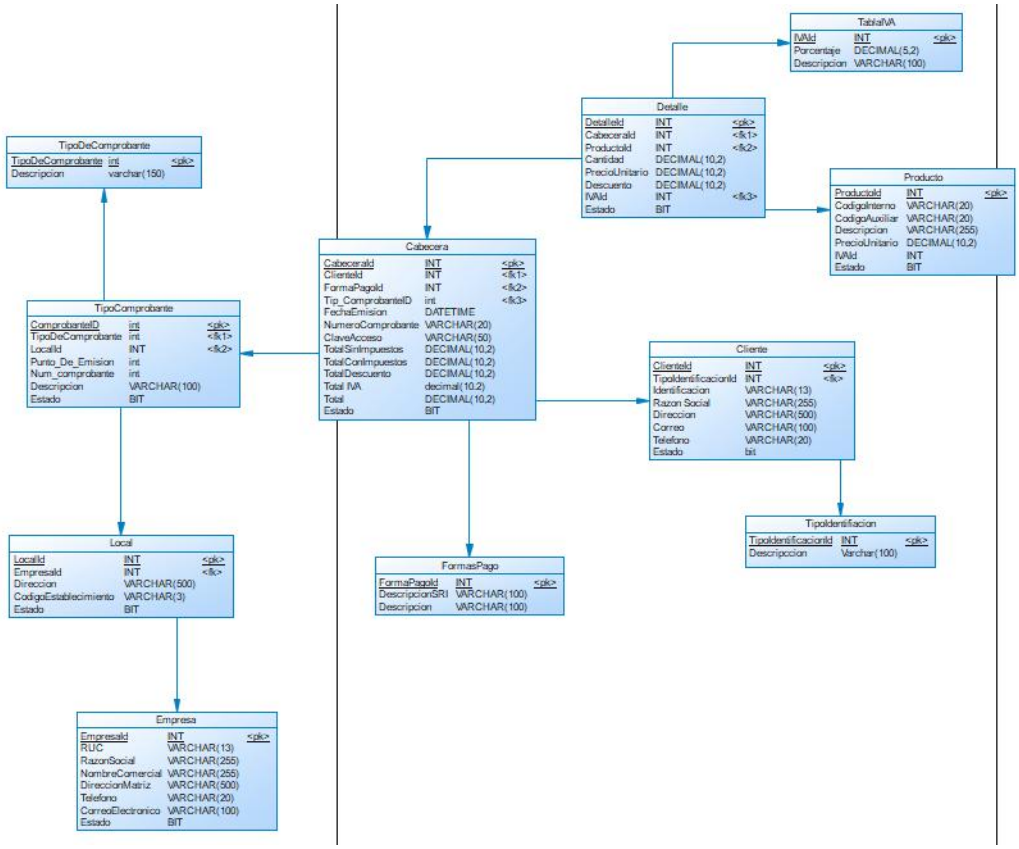
3.3.1 UML

Ilustración 9 UML de la BDD



3.3.2 Modelo Físico PowerDesigner

Ilustración 10 Modelo logico de la BDD



3.4 Implementación completa de XP

El proyecto consistió en desarrollar un prototipo de sistema web de facturación electrónica para la empresa (*Brother Corp*) en un plazo de 12 semanas, llevado a cabo por un solo desarrollador. Se implementó una arquitectura de tres capas (presentación, lógica de negocio y acceso a datos) utilizando SQL Server para la base de datos, un API RESTful en C# con ASP.NET para la lógica de negocio, y una interfaz móvil/web en Flutter como capa de presentación. El sistema abarcó funcionalidades clave como la gestión de clientes, productos, empresa, emisión de comprobantes y generación de facturas electrónicas en formato XML conforme a las normas del SRI en Ecuador.

Dada la naturaleza dinámica del desarrollo y la necesidad de adaptarse rápidamente a cambios en requerimientos y obtener software de alta calidad, se eligió Extreme Programming (XP) como metodología ágil. XP enfatiza valores de comunicación, simplicidad, feedback continuo, coraje y respeto, apoyándose en 12 prácticas fundamentales que garantizan rapidez y calidad en la entrega. A pesar de tratarse de un desarrollo individual, se adaptaron todas las prácticas de XP para simular un entorno disciplinado similar a un equipo. A continuación, se describen las 12 prácticas fundamentales de XP aplicadas en el proyecto, con ejemplos concretos de *dónde, cuándo, por qué y cómo* se emplearon. Posteriormente, se presenta una bitácora semana a semana de las actividades del desarrollo, destacando en cada fase las prácticas XP utilizadas.

3.4.1 Prácticas Fundamentales de XP Aplicadas en el Proyecto

- **1. El Juego de Planificación (Planning Game)**

En XP, el Juego de Planificación es la práctica de planificar de forma iterativa con la participación cercana del cliente. En este proyecto, al inicio de la semana 1 se realizó una planificación inicial donde el desarrollador definió junto con un representante de negocio los objetivos del primer ciclo. Se escribieron historias de usuario para las funcionalidades principales. Estas historias se priorizaron según el valor para el negocio y complejidad, definiendo así el alcance de las primeras iteraciones. El resultado de esta planificación fueron objetivos semanales claros, plazos estimados y pasos a seguir por iteración. Cada nueva semana iniciaba con una breve sesión de planificación donde se ajustaban las prioridades en base al feedback de la semana anterior. Esta adaptación constante permitió guiar el trabajo de desarrollo de forma realista y enfocado en máximo valor. Por ejemplo, tras completar la base

de datos en la iteración 1, se reordenó el backlog para abordar en la iteración 2 la API de emisión de facturas.

- **2. Pequeñas Entregas (*Small Releases*)**

La práctica de Pequeñas Entregas implicó dividir el proyecto en incrementos funcionales frecuentes. En lugar de esperar al final para mostrar resultados, se realizaron entregas parciales al término de cada iteración. En este proyecto, se planificaron lanzamientos quincenales (cada 2 semanas) coincidiendo con fases técnicas: al finalizar la semana 4 se entregó un primer incremento que incluía la base de datos y un prototipo inicial del API; al finalizar la semana 8, un segundo incremento con la API completa y algunos endpoints consumidos desde un cliente de prueba; y al finalizar la semana 12, el producto integrado con la interfaz Flutter. Estas entregas, aunque fueran prototipos, se desplegaron en un entorno de pruebas para que el cliente pudiera probar la funcionalidad real. Las entregas pequeñas y frecuentes permitieron obtener retroalimentación temprana y constante durante todo el desarrollo. Por ejemplo, tras la primera entrega se descubrió, gracias al feedback, que faltaba incluir el campo "dirección del cliente" en la base de datos para cumplir requisitos legales, lo que se corrigió inmediatamente en la siguiente iteración. Asimismo, las entregas incrementales facilitaron detectar errores de forma rápida y supervisar el desempeño del sistema.

- **3. Metáfora del Sistema (*System Metaphor*)**

La Metáfora en XP busca establecer un lenguaje común y una visión simple del diseño del sistema. Aunque el equipo era de una sola persona, se definió una metáfora para mantener coherencia conceptual entre el dominio de negocio y la implementación técnica. En este caso, la metáfora acordada fue concebir el sistema como un "taller de facturación digital", donde: la base de datos es el almacén de información (clientes, productos, facturas), la API es el encargado que procesa pedidos (solicitudes CRUD, generación de factura), y la interfaz Flutter es el mostrador por donde el usuario interactúa para realizar las transacciones. Esta metáfora ayudó a usar terminología consistente en el código. Según XP, una buena metáfora del sistema facilita que todos los involucrados tengan un entendimiento compartido del diseño, con nombres coherentes y estructura comprensible. En la práctica, la metáfora guio decisiones de diseño, de forma tal que el código reflejara claramente el proceso de emisión de una factura, tal como ocurriría en un taller de facturación tradicional.

- **4. Diseño Simple (*Simple Design*)**

Siguiendo los valores de XP, se adoptó un diseño lo más simple posible en cada etapa del desarrollo. La regla fue: diseñar "lo más simple que pueda funcionar" para cumplir las historias actuales, evitando complejidad innecesaria. En la fase de base de datos (semanas 1–2), esto significó crear tablas y relaciones mínimas para soportar las funcionalidades básicas (clientes, productos, empresa, factura cabecera y detalle), sin añadir campos o entidades que no fueran requeridas en las historias de usuario presentes. Por ejemplo, no se incluyó inicialmente una tabla de usuarios/roles porque la facturación electrónica mínima no lo necesitaba en esta etapa. El diseño simple también se aplicó al escribir código: durante la implementación del API REST (semanas 3–7), se escribieron métodos claros y pequeños, evitando duplicación de lógica y favoreciendo la legibilidad. De acuerdo con XP, el mejor diseño es aquel más simple que cumple con los requisitos actuales, pasa todos los tests, no duplica código y utiliza el menor número de clases y métodos posible. Cada vez que emergía una complejidad accidental, se refactorizó dividiendo en funciones más simples. Esta filosofía de diseño incremental garantizó que el sistema pudiera adaptarse fácilmente a cambios en requisitos, cada nueva característica se integró ajustando el diseño sólo en lo necesario y evolucionando la arquitectura gradualmente, en lugar de pre-diseñar extensiones que quizá nunca se usarían aplicando el principio YAGNI: You Ain't Gonna Need It, y también la principio KISS: Keep it simple, stupid. Gracias a este diseño sencillo e incremental, no fue necesario rehacer grandes módulos el código se mantuvo limpio y comprensible para futuras ampliaciones.

- **5. Refactorización (*Refactoring*)**

La Refactorización continua fue una práctica central durante todo el desarrollo. En XP, refactorizar significa mejorar el diseño interno del código sin cambiar su funcionalidad externa, eliminando redundancias y aumentando la claridad. En el proyecto, se refactorizó de manera sistemática al final de cada pequeña entrega y cuando las pruebas pasaban. Por ejemplo, durante la semana 5, tras implementar la lógica de cálculo de impuestos en la API, se detectó código duplicado en las funciones de cálculo del IVA y descuento; en lugar de dejarlo así, se creó una única función reutilizable para el cálculo de totales, simplificando el código. Del mismo modo, en la semana 9, tras integrar la interfaz Flutter con el API, se observó que ciertos nombres de variables y métodos en el código Dart no eran consistentes con los de la API, lo que podía causar confusión; se procedió a renombrarlos y reorganizar la estructura de archivos en Flutter para reflejar mejor la arquitectura en capas. Cada refactorización se hizo asegurándose primero de tener pruebas que validaran el comportamiento y luego verificando

que todos los tests continuarán pasando después de los cambios. Esta disciplina garantizó que el código permaneciera de alta calidad y fácil de mantener. La refactorización constante eliminó funciones innecesarias y duplicadas, mejorando la coherencia del código y manteniendo el diseño lo más simple posible.

- **6. Desarrollo Guiado por Pruebas (TDD)**

El proyecto adoptó Desarrollo Guiado por Pruebas (Test-Driven Development). Con TDD, antes de implementar una funcionalidad, se escribía primero una prueba unitaria que fallaba (porque la funcionalidad aún no existía), luego se escribía el código mínimo necesario para pasar la prueba, y finalmente se refactoriza el código asegurando que las pruebas siguieran pasando. En la práctica, durante la implementación del API en C#, se desarrolló un conjunto de pruebas para la lógica de negocio crítica (por ejemplo, cálculo de impuestos, generación de número de factura, validación de montos). En la semana 4, antes de codificar la función que calcula el total de una factura con IVA y descuento, se creó una prueba que instanciaba una factura con items de prueba y verificaba que el total calculado coincidiera con el valor esperado según las reglas tributarias. Solo luego de definir esta prueba (que inicialmente falló), se implementó la lógica de cálculo hasta lograr que la prueba pasará, y se refactorizó para eliminar duplicaciones. Esto garantizó que cada nueva funcionalidad viniera respaldada por pruebas desde su nacimiento, reduciendo drásticamente la probabilidad de fallos futuros. Además de pruebas, se incluyeron pruebas de integración básicas: por ejemplo, en la semana 7, tras exponer el endpoint de creación de factura, se escribió un test que llamaba al API con datos de factura y luego consultaba la base de datos para verificar que el registro se guardó correctamente. Esta cultura de pruebas exhaustivas y automatizadas permitió al desarrollador refactorizar con confianza y asegura que el código cumpla con los requisitos del cliente. Cabe destacar que TDD también sirvió como documentación viva del comportamiento del sistema, ya que las pruebas describían en lenguaje técnico las especificaciones.

- **7. Integración Continua (*Continuous Integration*)**

Aun siendo un solo desarrollador, se aplicó la práctica de Integración Continua (CI) para que todas las partes del sistema funcionaran siempre de forma conjunta y se detectarán rápidamente incompatibilidades. Durante el desarrollo local se colaboró mediante pruebas interactivas con Swagger UI, que permitió validar en tiempo real cada endpoint del API. Cada vez que se implementaba o modificaba un controlador en C#, se lanzaba la aplicación en modo depuración y se abría Swagger para probar el Crud de las tablas. De igual forma, el proyecto Flutter tenía su propio flujo de CI para asegurarse de que build de la app no se rompía con nuevos cambios.

Según XP, los equipos no esperan al final de la iteración para integrar, integran constantemente, a menudo varias veces al día. Siguiendo ese principio, en este proyecto nunca pasaron más de 1–2 días sin una build funcional del sistema completo. Por ejemplo, cuando en la semana 8 se desarrolló la primera pantalla de Flutter que consumía el API de productos, el desarrollador hizo pequeños commits tras cada avance significativo (listar productos, añadir producto, etc.), verificando en cada integración que la app seguía funcionando correctamente con la API. La CI ayudó a identificar muy pronto problemas de integración entre componentes: en una ocasión, un test de CI falló indicando que un cambio en el esquema de la base de datos (campo agregado a la tabla Factura) no se reflejó en el modelo de datos del API; este fallo se detectó el mismo día y se corrigió antes de continuar, evitando defectos acumulados. Así, la integración continua actuó como una red de seguridad, garantizando que el sistema permaneciera siempre construible y que los nuevos cambios no rompieran funcionalidades existentes. Esta práctica, combinada con las pruebas, previno sorpresas al final de proyecto y permitió entregar iteraciones estables al cliente on-site con confianza.

- **8. Programación en Pareja (*Pair Programming*)**

La Programación en Pareja usualmente implica que dos programadores trabajan juntos en un mismo computador, alternando los roles de "conductor" (quien escribe el código) y "navegante" (quien revisa y sugiere). En un contexto de desarrollador único, esta práctica se tuvo que adaptar creativamente. Para emular los beneficios, en este caso se recurrió a revisiones de código frecuentes y a colaboraciones puntuales: por ejemplo, durante la semana 6, al enfrentar la compleja generación del XML de factura, se invitó a un colega desarrollador a revisar el código en una sesión breve de pair programming informal. En esa sesión, el colega actuó como segundo par de ojos, revisando la lógica y aportando sugerencias para mejorar la legibilidad y corregir un posible error en el formato XML.

- **9. Propiedad Colectiva del Código (*Collective Code Ownership*)**

La práctica de Propiedad Colectiva del Código establece que el código le pertenece a todo el equipo, y cualquiera puede modificar cualquier parte cuando sea necesario. En el caso de ser una persona, se trabajó con un único repositorio accesible y con documentación interna suficiente para que cualquier desarrollador externo pudiera entender y modificar el sistema. En la práctica, se dio libertad para refactorizar y mejorar cualquier componente del código en cualquier momento. Por ejemplo, en la semana 9, al refinar la interfaz de Flutter, se detectó que era más eficiente agregar un DTO para acelerar las consultas de los detalles, aunque la API ya estaba terminada semanas atrás. Este enfoque evitó tocar la lógica de la API y previno la

duplicación de código. Además, se siguieron estándares colectivos más altos de calidad: por ejemplo, antes de dar una historia por terminada, se validó que el código de esa parte cumplía con las normas de formato y estaba suficientemente cubierto, asumiendo la responsabilidad total del código como lo haría un equipo completo.

10. Estándares de Codificación (*Coding Standards*)

Desde el inicio se establecieron estándares de codificación unificados para todo el proyecto, tal como XP recomienda. Se definieron convenciones claras en cuanto a nombres de variables, formato de código, organización de proyectos y uso de comentarios en ciertas partes de código, de modo que pareciera escrito por una sola persona. En C#, se siguieron las guías de estilo de .NET. En Flutter/Dart, se adoptaron las recomendaciones oficiales de estilo de Dart. Gracias a esto, el código mantuvo una consistencia uniforme, facilitando su lectura y refactorización en cualquier momento. Por ejemplo, en la semana 5 se decidió un estándar para los nombres de endpoints REST y se renombraron dos controladores cuyo nombre no coincidía exactamente con este estándar. Esto evitó confusiones a la hora de consumir el API desde Flutter y hizo más sencilla la depuración. Se puede concluir que, los estándares de codificación actuaron como una guía para que todo el código fuente mantuviera alta legibilidad y uniformidad.

11. Cliente On-Site (*On-Site Customer*)

XP promueve tener un cliente on-site trabajando cotidianamente con el equipo de desarrollo, de modo que las dudas se aclaren al instante y las prioridades se ajusten con feedback inmediato. En un proyecto individual, no se disponía físicamente de un cliente todo el tiempo, pero se simuló esta práctica manteniendo un contacto constante con el usuario referente del sistema. En este caso, se actuó como interlocutor con un contador de confianza de Brother Corp. Cada vez que surgía una decisión de negocio o duda de requisitos, se consultaba rápidamente con este "cliente" vía reuniones breves o mensajes. Por ejemplo, en la semana 3, al diseñar el esquema de la factura en la base de datos, se verificó con el cliente qué campos eran obligatorios según la normativa del SRI (como el número de autorización, clave de acceso, etc.), evitando suposiciones erróneas. Asimismo, al concluir cada iteración semanal, se realizaba una demostración al y se recogía su feedback. Un caso concreto ocurrió en la semana 8: tras mostrar el formulario de emisión de factura en la app Flutter al cliente, este sugirió incluir la visualización del IVA. Dicha sugerencia se incorporó de inmediato en la siguiente iteración, ajustando la historia de usuario correspondiente. Aunque el cliente no estuvo "sentado al lado" físicamente todos los días, sí estuvo disponible continuamente para

responder preguntas. Esta colaboración estrecha garantizó que el producto final se alineara con las necesidades reales: ningún requerimiento crítico pasó desapercibido, y se pudo aceptar o corregir funcionalidades casi en tiempo real, tal como XP lo preconiza.

- **12. Ritmo de Trabajo Sostenible (*Sustainable Pace / 40-Hour Week*)**

Finalmente, se respetó el principio XP de mantener un ritmo de desarrollo sostenible, evitando sobrecargar con jornadas excesivas. En concreto, se planificó el trabajo para ajustarse a unas 40 horas por semana, distribuidas en 5 días laborables, reservando descansos los fines de semana. En el proyecto, esto se tradujo en no recurrir a horas extra de manera habitual; si bien hubo momentos puntuales de alta intensidad (por ejemplo, al acercarse una entrega importante al final de la semana 12), se compensaron reduciendo la carga al inicio de la semana siguiente, evitando el agotamiento. Gracias a esto, se minimizaron errores por cansancio y se mantuvo la moral alta durante las 12 semanas. Al final de cada iteración, se realizaba una pequeña retrospectiva personal para evaluar la productividad y ajustar el ritmo si era necesario, priorizando siempre la calidad sobre la prisa.

Habiendo detallado cómo se aplicaron las doce prácticas fundamentales de XP en el proyecto, a continuación, se presenta la bitácora de desarrollo semana a semana. En esta bitácora se registran las actividades realizadas en cada una de las 12 semanas del proyecto, indicando cómo y qué prácticas XP destacaron en cada fase técnica (base de datos, API, interfaz) y en cada iteración temporal. Esto ilustra de forma cronológica cómo un único desarrollador llevó adelante un proceso XP disciplinado, simulando el comportamiento de un equipo ágil completo.

3.4.2 Bitácora Semanal del Desarrollo (12 semanas)

Semana 1: Planificación Inicial y Diseño de la Base de Datos

- **Planificación y alcance:** Se llevó a cabo la planificación general del proyecto. Se reunió con el cliente representante para definir el alcance inicial. Se identificaron las principales historias de usuario: gestión de clientes, productos, empresa, emisión de facturas electrónicas (XML) y manejo de comprobantes. Cada historia se priorizó y se estimó cuáles podrían completarse en la primera iteración. Se estableció como meta de la semana 1 tener el diseño de la base de datos listo y revisado.
- **Metáfora y modelo de dominio:** Durante esta semana se definió la metáfora del sistema "taller de facturación digital", para alinear terminología. Se delineó el modelo entidad relación de las tablas propuestas para Clientes, Productos, Empresa, Cabecera, Detalle, etc. Este modelo se diseñó en PowerDesigner respetando los nombres del dominio, facilitando la comunicación con el cliente.
- **Diseño simple de la base de datos:** Aplicando la práctica de Simple Design, el modelo incluía solo los campos necesarios para cumplir los requisitos actuales. Esto mantuvo el esquema sencillo y entendible.
- **Feedback inmediato del cliente:** A mitad de la semana, se realizó una revisión del modelo con el cliente on-site. El cliente validó que el diseño contemplaba la información necesaria para las facturas electrónicas según SRI (confirmando por ejemplo que incluimos el campo de "clave de acceso" del SRI). Esta colaboración temprana evitó malentendidos.
- **Planificación de iteración:** Al finalizar la semana 1, se ajustó el plan de la siguiente iteración (semana 2) en base al progreso: el modelo E/R estaba terminado y aprobado, por lo que en la semana 2 se procedería a implementar la base de datos física en SQL Server y se lleno con algunos datos de prueba.

Semana 2: Implementación de la Base de Datos y Pruebas Iniciales

- **Construcción de la base de datos:** Siguiendo el diseño aprobado, se creó la base de datos en SQL Server. Se escribieron scripts DDL para generar las tablas Cliente, Producto, Empresa, etc, con sus claves primarias y foráneas.
- **Pequeña entrega interna:** A mediados de semana 2, se generó una primera versión funcional de la base de datos, que se consideró como un small release interno. Se cargaron datos de ejemplo y una breve demo de consultas SQL. Esta pequeña entrega permitió obtener feedback: el cliente verificó que los datos obligatorios legales estuvieran presentes. Por ejemplo, notó que en Empresa faltaba el campo RUC (número de identificación tributaria) esta omisión se corrigió de inmediato agregando dicho campo, demostrando la ventaja de entregar pronto y corregir con el feedback.
- **Pruebas de aceptación del cliente (manuales):** El cliente on-site definió algunas pruebas de aceptación sencillas sobre la base de datos. Por otro lado se revisó los atributos de las tablas y si estas eran útiles para la empresa.
- **Preparación para lógica de negocio:** Hacia el final de la semana 2, con la base de datos estable, se comenzó a planificar la capa de negocio. Se creó en Visual Studio 2022 un proyecto de biblioteca de clases C# para Acceso a Datos utilizando Entity Framework para mapear las tablas a entidades. Este es un paso de diseño incremental: se comenzó a implementar sólo lo necesario para conectar con la base de datos.
- **Sostenibilidad del ritmo:** Cabe recalcar que hasta ahora se mantuvo un ritmo constante, se cumplió el horario planeado sin necesidad de trabajo nocturno ni de fin de semana, lo cual es coherente con la práctica XP de ritmo sostenible. Se hizo retrospectiva de las dos primeras semanas: los objetivos se habían cumplido en tiempo, así que la planificación para la siguiente iteración (API) fue realista basándose en la velocidad observada.

Semana 3: Inicio del Desarrollo del API REST (Capa de Negocio)

- **Planificación de la API (Juego de Planificación):** Al inicio de la semana 3 se realizó una sesión de planning enfocada en la capa de lógica de negocio. Se seleccionaron historias de usuario relacionadas con el API: por ejemplo, Registrar nuevo cliente vía API POST Consultar lista de productos GET. Emitir una factura POST /facturas genera XML. El cliente priorizó primero las operaciones CRUD básicas de clientes y productos, dejando la emisión completa de factura para después, dado que dependía de tener clientes y productos listos.

- **Configuración del proyecto y CI:** Se creó el proyecto ASP.NET (Framework) Web API en Visual Studio 2022, estructurado en n capas. Se integró desde el inicio con la base de datos creada a través de Entity Framework, generando los DbContext y modelos a partir del esquema que ya se tenía en SQL Server.

- **TDD en acción (primera prueba unitaria):** Antes de implementar el primer endpoint real, se aplicó Test-Driven Development. Se escribió una prueba unitaria para la lógica de creación de un cliente: en un escenario de prueba, se probaría un método CrearCliente (nombre, identificación) y luego se verificaría que la base de datos contenga ese cliente. Obviamente, ese método no existía, así que la prueba no compilaba; se creó el método con implementación mínima tirando una excepción o retornando null para luego hacer que fallara el test, y a partir de allí se implementó lo necesario para que pasara. Al final se tenía no solo el endpoint POST /clientes funcionando, sino también una prueba automatizada que validaba su comportamiento correcto.

- **Entrega incrementa y feedback:** Al terminar la semana 3, ya estaban operativos dos endpoints (clientes y productos). Se desplegó el API en el IIS de prueba local de la empresa para que el cliente on-site pudiera probarlos mediante herramientas como el Swagger. Esto constituyó una entrega parcial del API. El cliente probó agregar un cliente y obtener la lista de productos, dando su visto bueno y comentarios: por ejemplo, sugirió que el campo “identificación” del cliente debería validar el formato de cédula/RUC. Este feedback se tomó nota para implementarlo la siguiente semana se añadió la validación en la lógica de negocio. Nuevamente, la entrega temprana descubrió un requisito no explicitado inicialmente, que pudo acomodarse fácilmente gracias a que el diseño era simple.

Semana 4: Desarrollo de Funcionalidades CRUD y Pruebas Unitarias

- **Expansión del API CRUD:** En la semana 4 se continuó con las demás operaciones CRUD. Se implementaron los endpoints de obtención, actualización y eliminación para clientes y productos. Cada endpoint se construyó aplicando TDD en la medida de lo posible: primero escribiendo pruebas unitarias que simulaban las operaciones y verificaban los resultados, por ejemplo, intentar eliminar un producto y luego comprobar que ya no existe en la base.

- **Integración continua varias veces al día:** La práctica de Continuous Integration se hizo evidente con múltiples integraciones diarias: tras completar cada endpoint o corrección importante. A mitad de semana hubo un fallo de integración: tras añadir un atributo en una tabla en la base de datos para un campo "Teléfono", una prueba falló porque la lógica de validación no contemplaba teléfono nulo. La CI detectó el fallo inmediatamente; se arregló antes de continuar.

- **Cliente on-site disponible:** Hubo interacción con el cliente durante la semana para ajustar requisitos en tiempo real. Por ejemplo, al implementar la actualización de producto PUT, surgió la duda de si permitir cambiar el código único de producto. En lugar de asumir, se consultó al cliente on site de inmediato, el cliente indicó que se necesitaba 2 tipos de códigos externo e interno. Con esa respuesta, se ajustó la lógica.

- **Preparación para la funcionalidad de facturación:** Hacia el final de la semana 4, con CRUD básicos listos, se preparó para la funcionalidad principal de facturación electrónica. Se volvió a revisar la normativa del SRI para asegurarse de los datos requeridos en el comprobante electrónico. Se creó la estructura de clases Cabecera y Detalle en el modelo de la API, y se delineó un plan de pruebas.

- **Pequeña entrega y retrospectiva:** Al terminar la semana 4, se tenía un API casi completo para mantenimiento de entidades básicas. Se hizo una demo integrando todo lo logrado hasta ahora: se mostró al cliente cómo mediante Swagger se podían crear, listar, actualizar y eliminar clientes y productos, y cómo esto afectaba los datos reales en la base. El cliente quedó satisfecho con el progreso y aportó feedback menor. Se realizó una retrospectiva interna la velocidad de entrega estaba siendo buena, todas las historias planificadas se habían completado a tiempo, y las pruebas automatizadas daban confianza para seguir añadiendo funcionalidades más complejas en la siguiente fase.

Semana 5: Implementación de Lógica de Negocio de Facturación (XML)

- **Empezando emisión de facturas:** La semana 5 estuvo centrada en la historia de usuario más importante: “Como usuario, quiero emitir facturas electrónicas y obtener el XML válido para el SRI”. Se planificó dividir esta funcionalidad en sub-tareas: creación de factura (en la base de datos) con sus detalles, cálculo de impuestos y totales, generación del archivo XML con el formato SRI.

- **TDD para cálculo de totales:** Se aplicó TDD rigurosamente al implementar la lógica de negocio tributaria. Primero se escribieron pruebas unitarias para casos de facturas: por ejemplo, una prueba verificó que, para una factura con 2 productos, el subtotal, el IVA y el total se calcularán correctamente según las reglas aplicando IVA solo a productos que llevan IVA.

- **Generación de XML conforme al SRI:** Una vez que la creación de factura en el sistema estaba lista y testeada, se abordó la generación del archivo XML de la factura. Para validar esta parte, se crearon pruebas de integración tras emitir una factura, luego se validó que el XML resultante contuviera ciertos nodos obligatorios (ej. RUC del emisor, número de autorización, detalles de productos).

- **Colaboración (par) en XML:** Generar el XML SRI correctamente es complejo. Se buscó apoyo, se tuvo que contactar con un colega con experiencia en facturación electrónica para revisar juntos el formato XML. En una sesión de una hora, equivalente a pair programming puntual, se revisó que las primeras partes al generar el XML estén correctas.

- **Entrega al cliente para pruebas:** Al finalizar la semana 5, el sistema podía emitir una factura, Se desplegó y se le entregó al cliente on-site un XML con las primeras partes generadas para que lo revisara. Descubrió un pequeño detalle – la razón social de la empresa emisora debía ir en mayúsculas según la norma, y nuestro sistema no lo estaba forzando. Este feedback fue recibido inmediatamente.

Semana 6: Finalización de la API y Publicación en IIS

- **Compleción de la API**

Se concluyó el desarrollo de todos los endpoints CRUD de todas las tablas necesarias. La lógica de negocio y las validaciones quedaron completas, incluyendo:

- Validación de existencia de cliente y productos antes de crear facturas.
- Cálculo de totales y subtotales con IVA y descuentos.
- Secuenciación automática y única de números de factura.

- **Seguridad y autenticación**

Para proteger los datos de acceso, las credenciales de usuario se almacenan cifradas en la base de datos. Se implementó un mecanismo de token JWT, donde:

- El usuario realiza login (POST /api/auth/login) enviando usuario y contraseña.
- La API valida credenciales y devuelve un token JWT firmado.
- Todos los endpoints protegidos requieren el token en la cabecera Authorization: Bearer token.

- **Pruebas con Swagger UI**

Se desplegó Swagger UI en el propio proyecto, permitiendo:

- Ejecutar manualmente cada endpoint (GET, POST, PUT, DELETE).
- Verificar esquemas de petición y respuesta en tiempo real.
- Validar flujos completos (crear cliente, producto, emitir factura).

- **Despliegue en IIS para Red Interna**

La API se publicó en Windows con IIS, quedando accesible en la red interna de la empresa:

- Los endpoints están listos para ser consumidos por la app Flutter y otras herramientas internas.

- **Release 2**

Al culminar la semana, se entregó formalmente la versión final de la API:

- Todos los servicios están documentados en Swagger.
- El cliente on-site validó los flujos críticos y solicitó sólo mejoras menores (p.ej., filtros por nombre).
- Con esto, la capa de negocio y datos quedó completa y lista para integrarse definitivamente con la interfaz de usuario.

Semana 7: Diseño de la Interfaz en Flutter y Planificación de la UI

- **Planificación de la capa de presentación:** Con el backend listo, la semana 7 se enfocó en la interfaz de usuario. En la planificación inicial de esta iteración, el cliente on-site aportó su perspectiva sobre el flujo que un usuario final es decir un administrador esperaría. Se priorizó primero tener pantallas para las entidades básicas como clientes, productos y luego la de facturación.

- **Arranque del proyecto Flutter:** Se configuró el entorno Flutter/Dart en Visual Studio Code. Se creó un nuevo proyecto Flutter multiplataforma. Aplicando diseño simple, inicialmente se generaron pantallas muy básicas con textos y sin estilos complejos, enfocándose en la estructura necesaria para mostrar datos. La idea era iterar rápidamente en funcionalidad antes de invertir en pulir la estética.

- **Metáfora en la UI:** Gracias a la metáfora del "taller de facturación", la estructura de la app se diseñó coherentemente una pantalla principal interactiva con opciones rápidas como administrar Clientes, Productos, Ver Facturas, análoga a secciones en un sistema contable. Cada sección tenía su pantalla de listado. Esta consistencia hizo que incluso un primer prototipo "en borrador" fuera entendido fácilmente por el cliente.

- **Consumo del API (integración frontend-backend):** Se implementó la integración con el API REST. Siguiendo XP de integración continua, desde la primera pantalla se comenzó a consumir el endpoint correspondiente. Para probarlo, se utilizó el API desplegado en local; se habilitó CORS en el backend para permitir peticiones desde la app en desarrollo. Se escribió una pequeña prueba manual: al abrir la pantalla de clientes en la app, debía traer los clientes reales de la base. La primera ejecución mostró un error, se corrigió inmediatamente.

- **Primera demo de UI al cliente:** Al finalizar la semana 7, se tenía una versión inicial de la app con la lista de clientes y productos funcional y una pantalla de formulario de nueva factura. Se preparó una demo interna donde el cliente pudo navegar en la app en la web. Aunque la interfaz no tenía diseño, el cliente pudo navegar, ver los clientes cargados y productos, comprobando la conectividad con el backend. El cliente comentó que la experiencia era intuitiva, pero solicitó agregar una búsqueda en la lista de productos debido a que podría haber muchos. Esta retroalimentación alimentó las tareas de la siguiente semana. Se refleja así la entrega temprana y retroalimentación continua, ahora en el contexto de la interfaz de usuario.

- **Semana 8: Desarrollo de Funcionalidades de UI y Feedback de Usabilidad**
- **Implementación de formularios y validaciones en UI:** En la semana 8, el foco estuvo en completar la funcionalidad de la interfaz. Se implementó la pantalla para crear un nuevo cliente desde la app y enviarlo al API (POST /clientes). Asimismo, se añadió la funcionalidad de búsqueda filtrada en las listas. Se aprovechó el diseño simple del backend: para implementar la búsqueda de productos, se hizo una consulta al API con query parameter, y dado que ya se había agregado el endpoint de búsqueda la semana anterior, la integración fue directa. En la app, se añadieron validaciones de campos (por ejemplo, obligatorio ingresar nombre y RUC al crear cliente, con mensajes de error claros en caso contrario).
- **Pruebas de UI (widget testing):** Aunque en menor medida que las pruebas backend, se escribieron algunas pruebas de widget en Flutter para asegurar comportamientos críticos. Por ejemplo, una prueba verificó que al ingresar datos válidos en el formulario de nuevo cliente y simular el tap en "Guardar", se llamaba al método que invocaba el API y que la lista de clientes se actualizaba mostrando el nuevo cliente. Este tipo de prueba ayudó a prevenir regresiones en la interfaz cuando se hicieron cambios de última hora en la lógica de estado.
- **Mejora de experiencia de usuario:** El cliente on-site fue involucrado intensamente esta semana para pulir la usabilidad. Se le entregó una actualización de la aplicación casi a diario aplicando small releases muy frecuentes, incluso diarios en esta fase de afinamiento. Con su feedback, se realizaron varios refinamientos: por ejemplo, se reorganizaron campos en el formulario de factura para que primero se seleccione el cliente y luego se añadan productos; se añadió un diálogo de confirmación antes de emitir la factura; también se incorporó la visualización del resultado de la emisión (mostrando el número de factura generado y permitiendo compartir el XML por correo).

Semana 9: Flujo Completo de Facturación en la App y Pruebas Integradas

- **Integración UI–API**
    - Pantalla “Emitir Factura” en Flutter consume POST /api/facturas.
    - Selección de cliente y productos, indicador de carga durante la generación de XML.
- **Confirmación de éxito**
    - Diálogo “Factura #X emitida” al recibir respuesta del API.
    - Notificación de que el XML quedó almacenado en el servidor.
- **Pruebas end-to-end manuales**
    - Flujo: crear cliente → crear producto → emitir factura → validar en BD.
    - Corrección inmediata del refresco de la lista de facturas en la app.
    - Cliente on-site validó el proceso como prueba de aceptación.
- **Ajustes por feedback**
    - Añadido resumen de montos y datos de la factura tras emisión.
- **Trazabilidad**
    - Nueva tabla Logs para registrar fecha e ID de cada factura.
    - API ajustado para insertar en Logs; pantalla oculta en la app para revisar registros.

Semana 10: Pruebas Finales, Optimización y Refactorización

- **Control de Calidad (QA)**
    - Pruebas con clientes y productos; emisión de facturas.
    - Validación de reglas de negocio.
- **Optimización de rendimiento**
    - Índice en campo FechaEmision de facturas.
    - Respuestas para listas de clientes y productos en la API.
    - Facil de manejar al tener un numero grande de productos.
- **Refactorización final**
    - Limpieza de código muerto y renombrado de métodos para mayor claridad.
    - Reorganización de módulos en Flutter; eliminación de print de depuración.
- **Pruebas de "quiebre"**
    - Cliente intentó casos extremos (nombres muy largos); se corrigieron fallos reportados.

Semana 11: Preparación de Entrega Final y Revisión de Requisitos

- **Verificación de cumplimiento de requisitos:** En la penúltima semana, se verificó exhaustivamente cada requerimiento funcional y no funcional del proyecto contra el sistema implementado. Se utilizó la lista inicial de requerimientos para validar uno por uno. Todos los requerimientos funcionales (gestión de clientes, productos, empresa, emisión de comprobantes, generación de XML conforme SRI) fueron demostrados en el sistema. También se comprobó que se cumplían requerimientos no funcionales relevantes, como usabilidad básica de la UI, rendimiento aceptable (el sistema respondía en menos de 3 segundos a peticiones normales), y seguridad mínima.
- **Ensayo de entrega y capacitación:** Se organizó una sesión de pre-entrega donde el desarrollador hizo una demostración integral del sistema como si fuera el día de entrega oficial. El cliente on-site participó junto con otros stakeholders de la empresa (por ejemplo, un gerente de TI de Brother Corp invitado para ver el prototipo). Durante la demo, se mostró todo el flujo: desde crear un cliente hasta la obtención del XML de factura y cómo ese XML podría subirse al portal del SRI. La respuesta fue muy positiva.
- **Retrospectiva de proyecto (equipo):** Internamente, se realizó una retrospectiva personal de todo el ciclo XP aplicado. Se identificó qué prácticas funcionaron mejor: las pruebas automatizadas (TDD) fueron cruciales para la confianza en cambios, la comunicación constante con el cliente evitó cambios grandes, las integraciones continuas detectaron errores temprano. También reconoció desafíos: la falta de un par programador constante fue suplida con revisiones, pero quizá un segundo desarrollador habría acelerado algunas tareas. En general, la retrospectiva concluyó que adoptar XP, incluso solo, fue altamente beneficioso para la calidad y cumplimiento del proyecto, validando la decisión tomada en la planificación inicial.

Semana 12: Entrega Final, Documentación y Cierre del Proyecto

- **Entrega final del prototipo:** La semana 12 marcó la entrega final del proyecto. Se desplegó la última versión estable de la aplicación en un entorno controlado. El backend se publicó en un servidor de prueba IIS y la app Flutter se entregó. El cliente on-site firmó el acta de recepción conforme del prototipo, señal de que todas las funcionalidades estaban implementadas según lo esperado. Esta entrega final ocurrió en la fecha comprometida originalmente, demostrando la efectividad de la planificación adaptativa de XP (no hubo retrasos en plazos gracias a los ajustes iterativos).

- **Documentación exhaustiva:** Se concluyó la documentación escrita del proyecto, incluyendo este apartado metodológico que refleja cómo se aplicó XP. Gracias a la propiedad colectiva y estándares de código, mucho de esto ya estaba en parte documentado en el código mismo; aun así, se generaron diagramas UML finales.

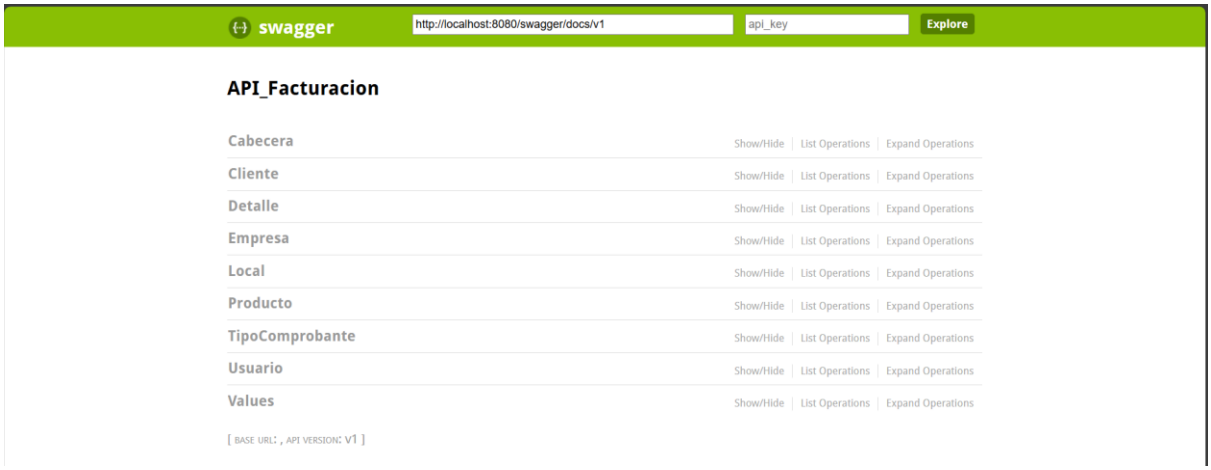
- **Cierre y reflexión:** Finalmente, se llevó a cabo una reunión de cierre con el cliente, en la que se presentó formalmente el resultado. El cliente destacó que el desarrollo fue muy transparente y colaborativo, notando que prácticamente fue parte del proceso mediante las iteraciones y feedback, lo cual coincide con el objetivo de XP de involucrar al cliente en todo momento. Desde el punto de vista académico, se reflexionó que aplicar XP en un proyecto unipersonal requirió disciplina extra, pero se pudo simular con éxito un entorno de equipo ágil. Esta experiencia confirmó que las prácticas de XP adaptadas apropiadamente pueden beneficiar incluso a desarrollos de una sola persona, produciendo software de alta calidad en plazos. En conclusión, el proyecto de 12 semanas finalizó exitosamente, demostrando cómo Extreme Programming, con sus 12 prácticas fundamentales, guio el proceso para lograr un prototipo completo de sistema de facturación electrónica web cumpliendo con los lineamientos del SRI y las expectativas del cliente, todo ello con un desarrollo ágil, disciplinado y documentado profesionalmente.

Capítulo IV: Diseño e Implementación del sistema de Facturación.

En este capítulo se presenta una visión práctica del prototipo de facturación electrónica desarrollado, ilustrada mediante capturas de pantalla tanto de la interfaz de usuario en Flutter como de la documentación de los endpoints expuestos en Swagger UI. Por un lado, las imágenes de la aplicación móvil/web muestran el flujo de trabajo completo: desde la gestión de clientes y productos hasta la emisión de facturas electrónicas y la visualización de resultados. Por otro lado, las capturas de Swagger describen de forma gráfica y estructurada cada uno de los servicios RESTful disponibles con sus rutas, parámetros de entrada, modelos de datos y códigos de respuesta, permitiendo entender con precisión cómo interactúa la capa de presentación con la lógica de negocio.

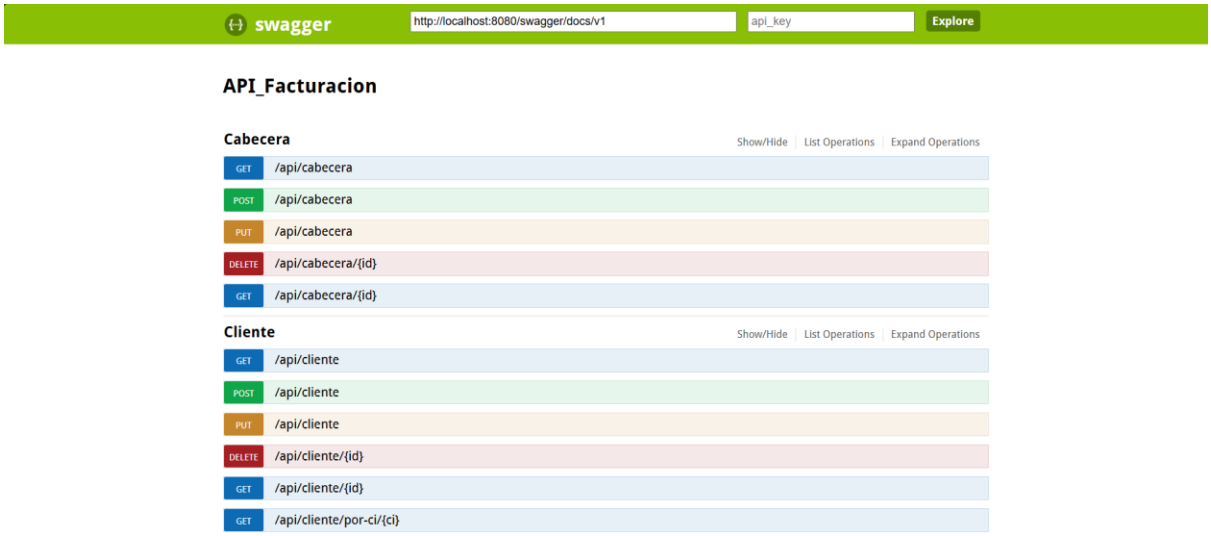
A través de esta combinación de evidencias visuales, se pretende demostrar no solo la usabilidad y el diseño responsivo del front-end, sino también la correcta implementación de la API: sus operaciones CRUD, la generación del XML de facturación y los mecanismos de validación.

Ilustración 11 End Points del Backend



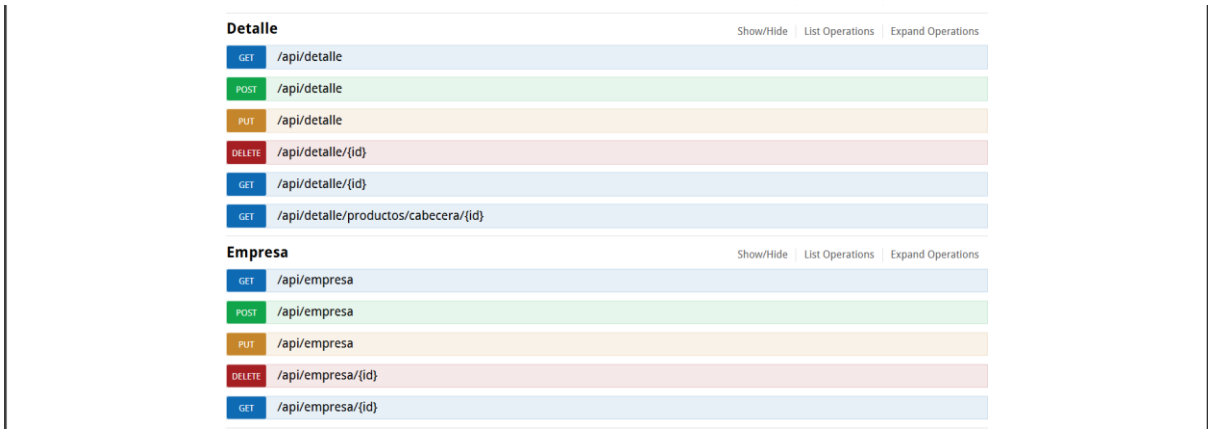
Nota: Visualización del Swagger generado desde el backend.

Ilustración 12 Rutas y Parámetros de Entrada Cabecera y Cliente



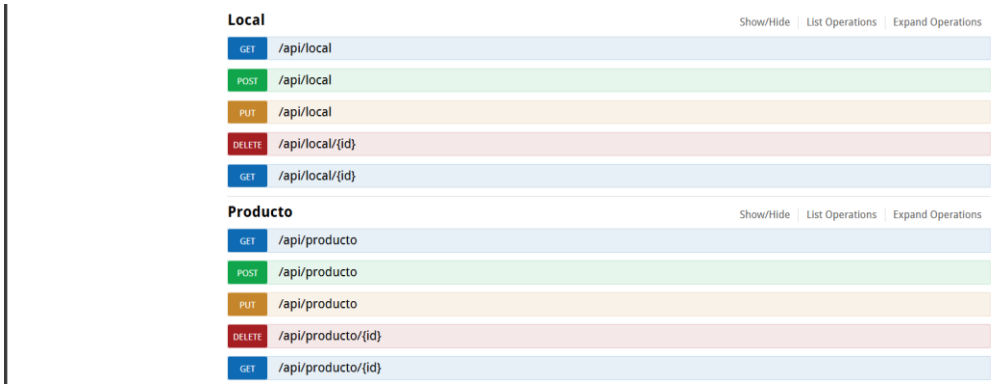
Nota: Visualización de los endpoints que tiene la Cabecera y Cliente con sus respectivos parámetros y llamadas.

Ilustración 13 Rutas y Parámetros de Entrada Detalle Y Empresa



Nota: Visualización de los endpoints que tiene el Detalle y Empresa con sus respectivos parámetros y llamadas.

Ilustración 14 Rutas y Parámetros de Entrada Local y Producto

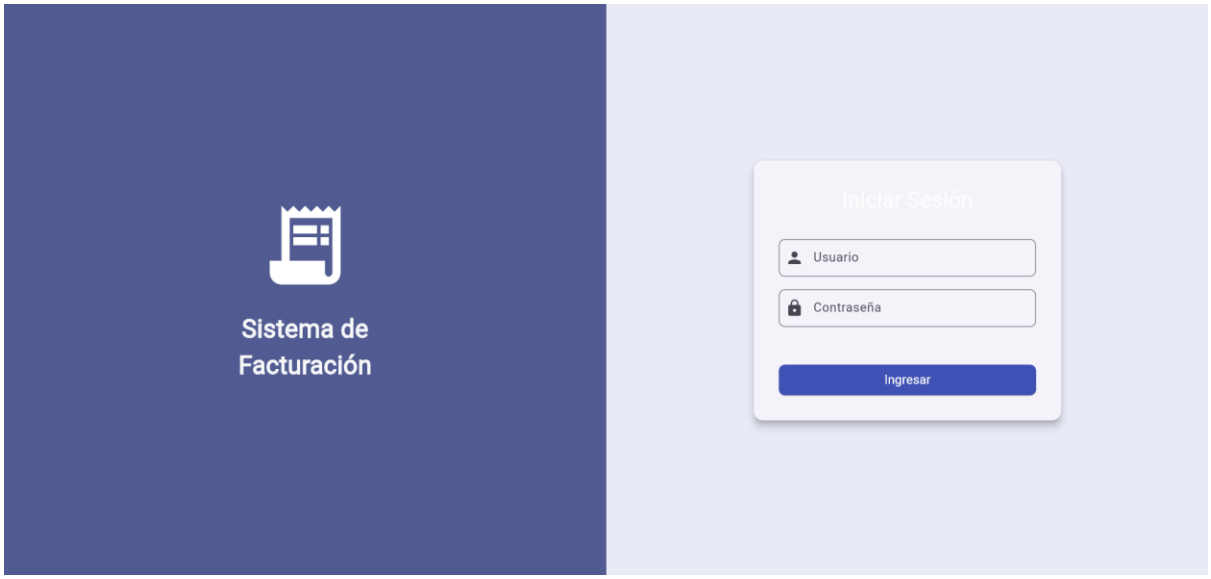


Nota: Visualización de los endpoints que tiene la Local y Producto con sus respectivos parámetros y llamadas.

Ilustración 15 Rutas y Parámetros de Entrada TipoComprobante y Usuario

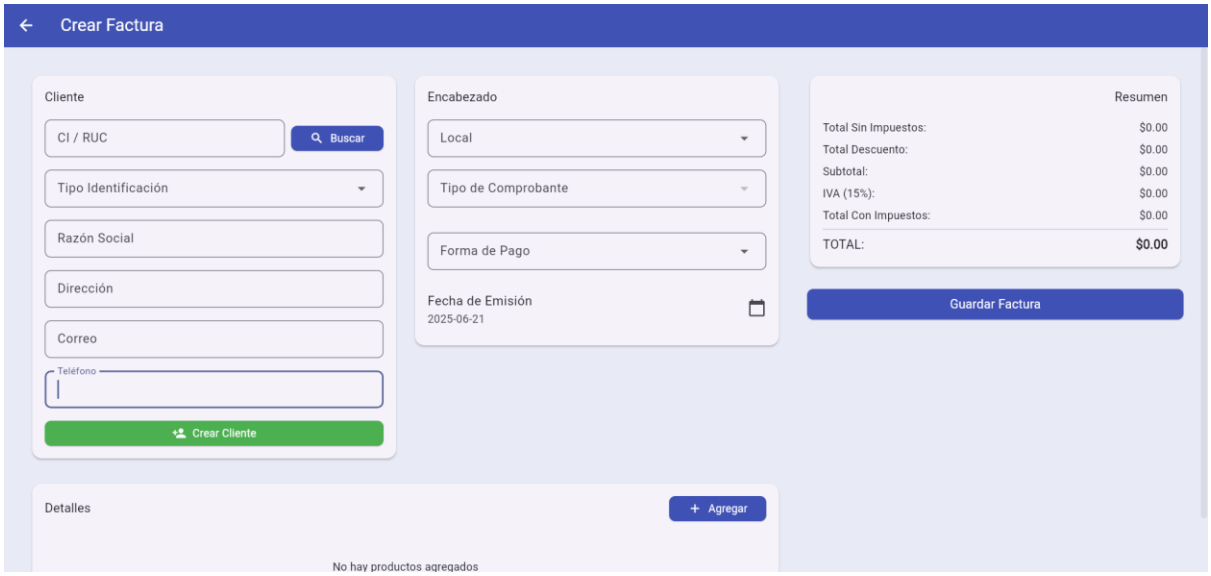
Nota: Visualización de los endpoints que tiene la Tipo Comprobante y Usuario con sus respectivos parámetros y llamadas.

Ilustración 16 Login



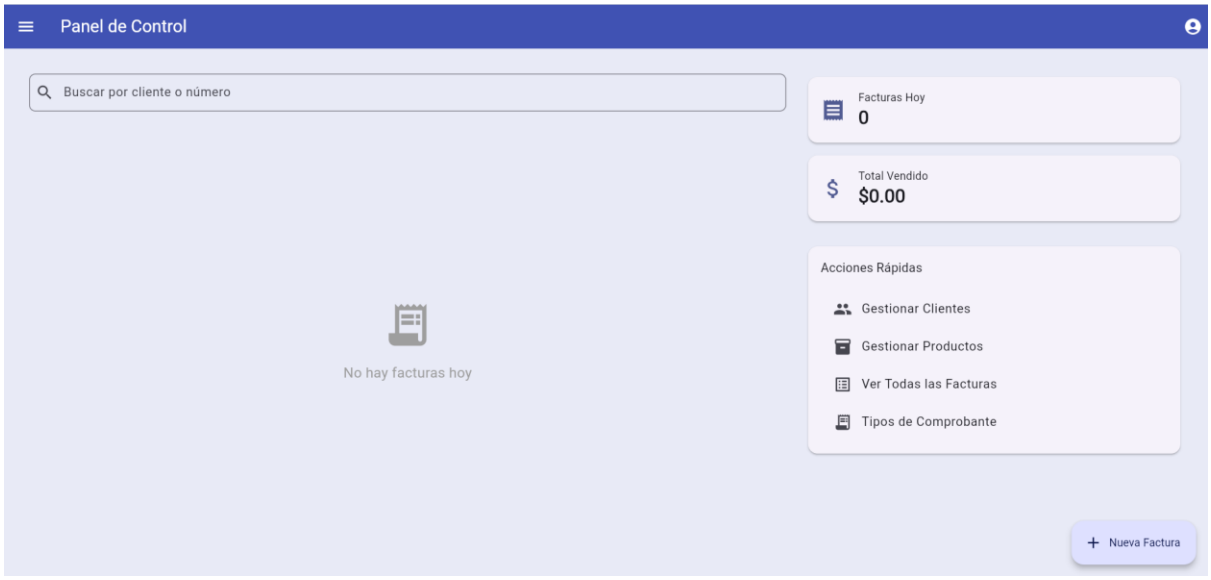
Nota: Pantalla de ingreso, dependiendo a la empresa se le crea un usuario administrador que puede agregar mas usuarios una vez que ingrese.

Ilustración 17 Creación de Factura



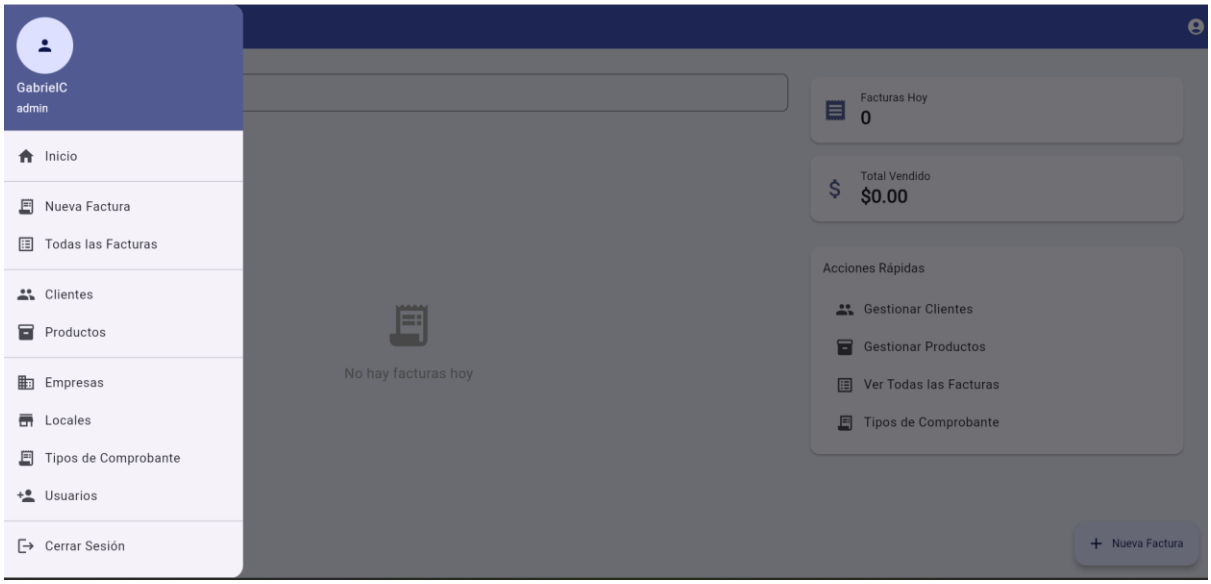
Nota: Creación de factura donde se busca por Cedula o RUC del cliente, en caso de no existir se puede crear el cliente, dentro del encabezado se escoge el local, tipo de comprobante Forma de pago y por ultimo la fecha, los detalles se escoge dependiendo los productos y stock que exista y por ultimo tenemos el resumen donde calcula todos los totales dentro de la factura.

Ilustración 18 Pantalla Principal



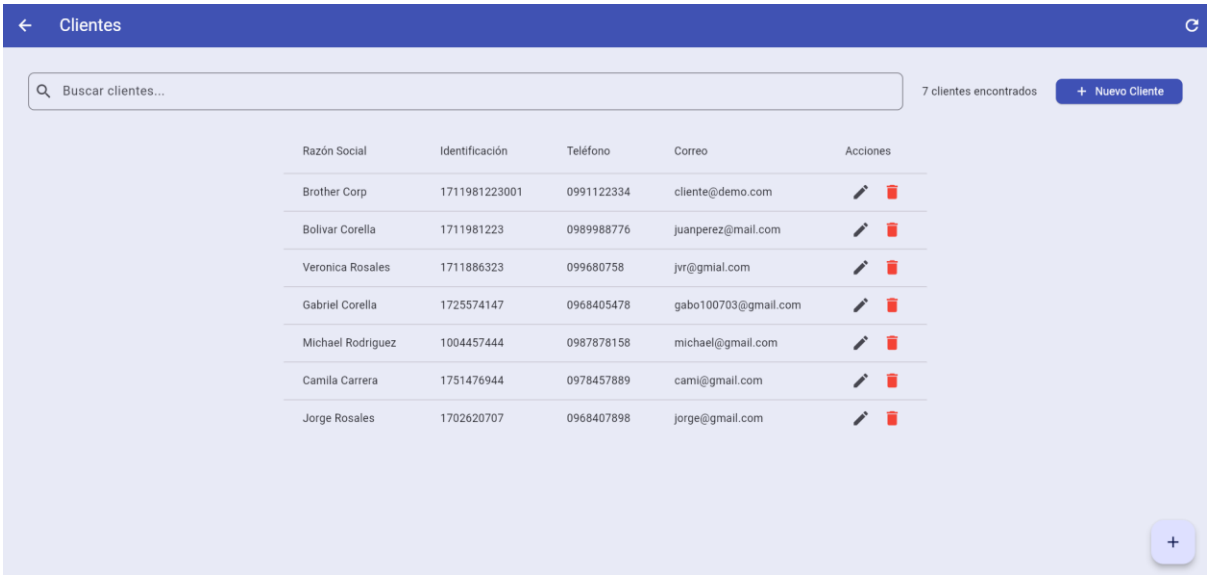
Nota: Pantalla principal donde muestra las facturas creadas del día, total de ventas y accesos rápidos hacia otras pantallas.

Ilustración 19 Menú Tipo Hamburguesa



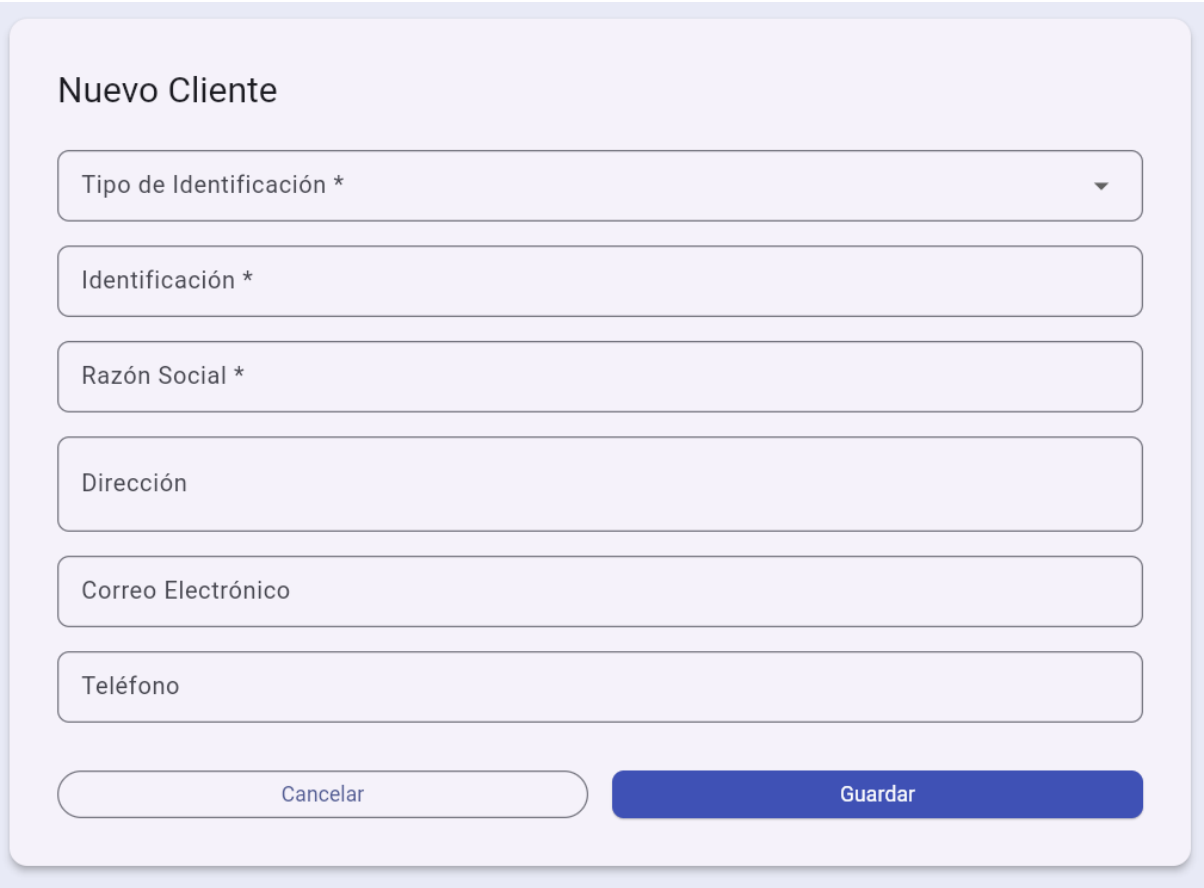
Nota: Menú lateral donde que se muestra todas las opciones que tiene el sistema.

Ilustración 20 Listado de Clientes



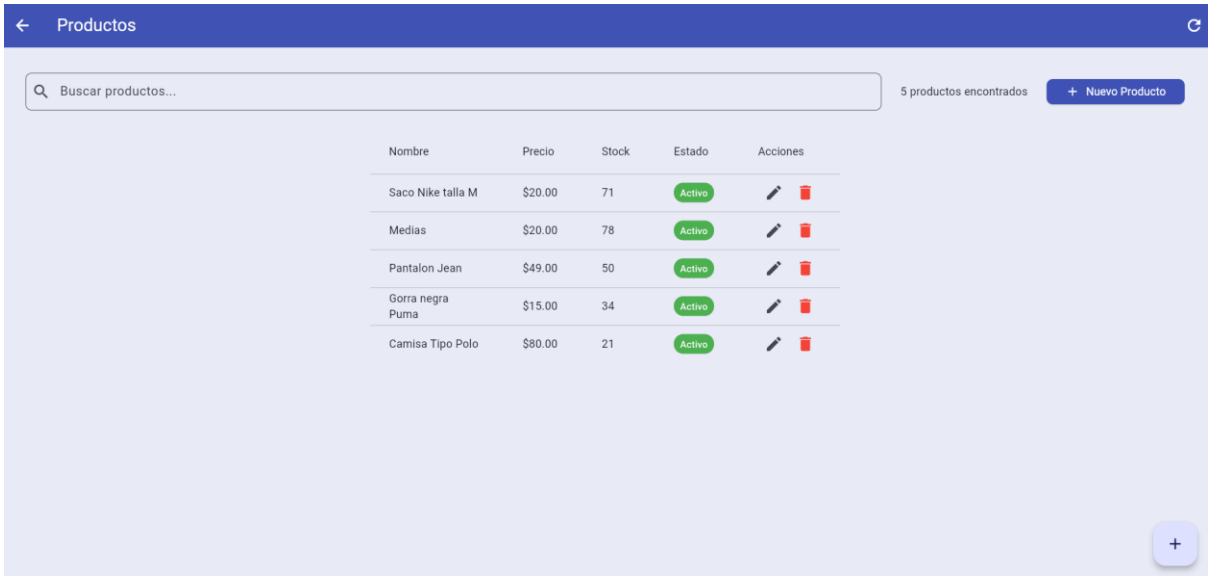
Nota: Listado de clientes, donde se puede filtrar por identificación o razón social, por otro lado, tenemos acciones como eliminar, actualizar o crear nuevo cliente.

Ilustración 21 Creación del Cliente



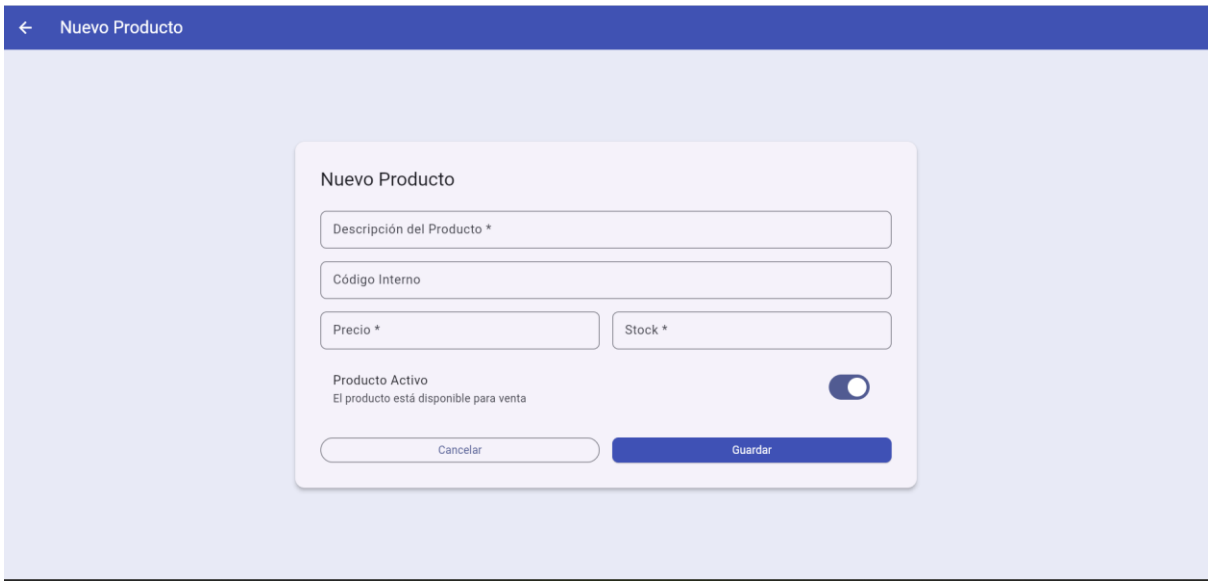
Nota: Pantalla de creación de cliente con sus respectivos campos.

Ilustración 22 Listado de Productos



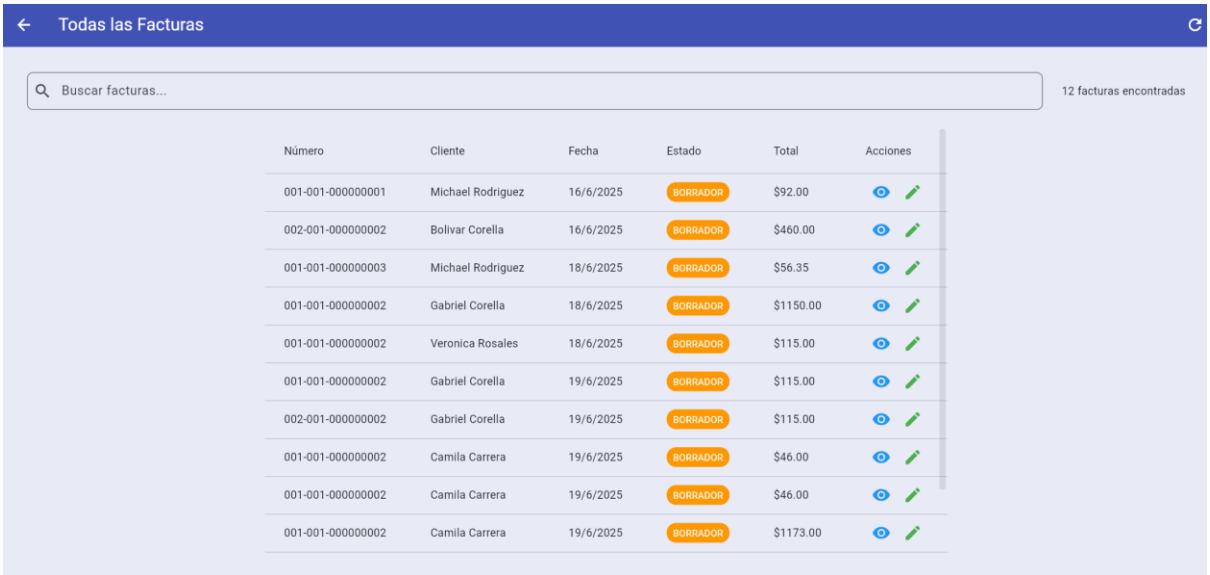
Nota: Listado de Productos, donde se puede filtrar por nombre, por otro lado, tenemos acciones como eliminar, actualizar o crear nuevo producto.

Ilustración 23 Creación de Producto



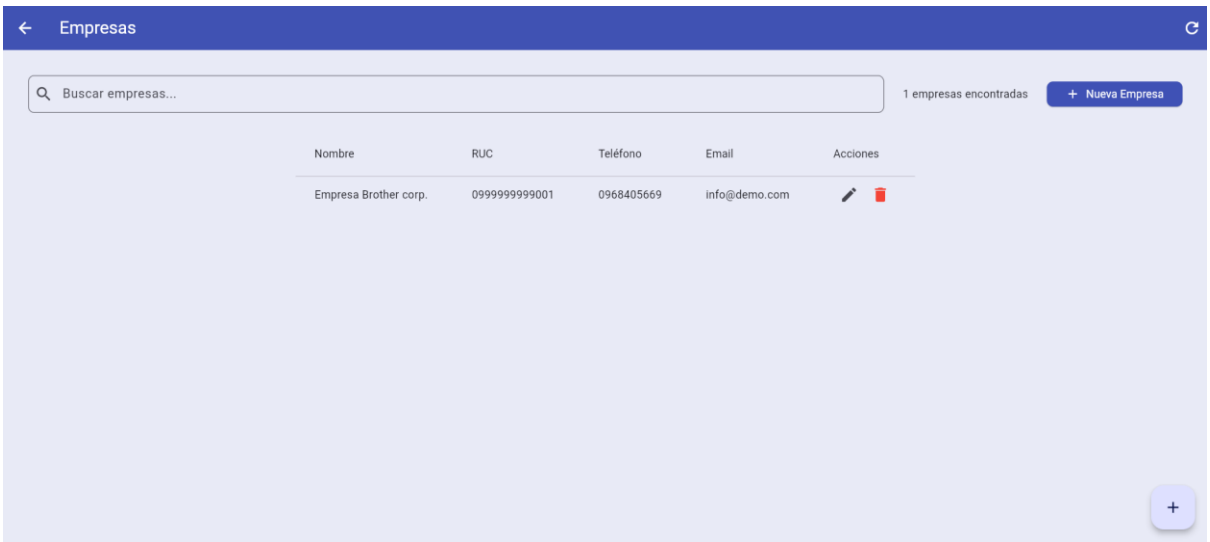
Nota: Pantalla de creación de producto con sus respectivos campos.

Ilustración 24 Listado de Facturas



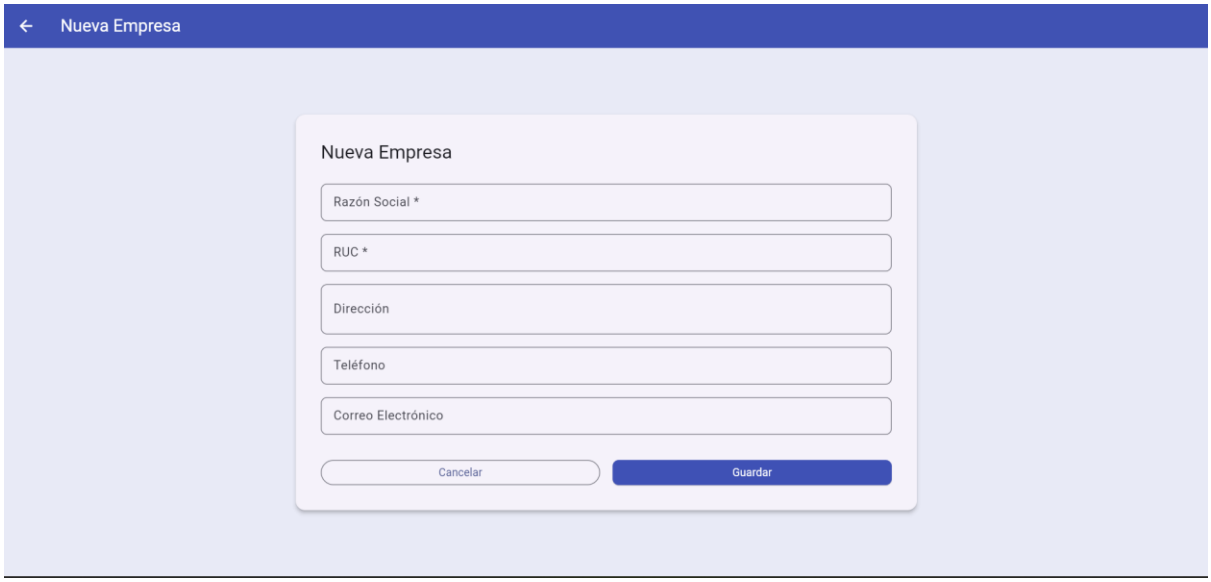
Nota: Listado de Facturas, donde se puede filtrar por cliente y número, por otro lado, tenemos acciones como eliminar y actualizar factura.

Ilustración 25 Datos de la Empresa



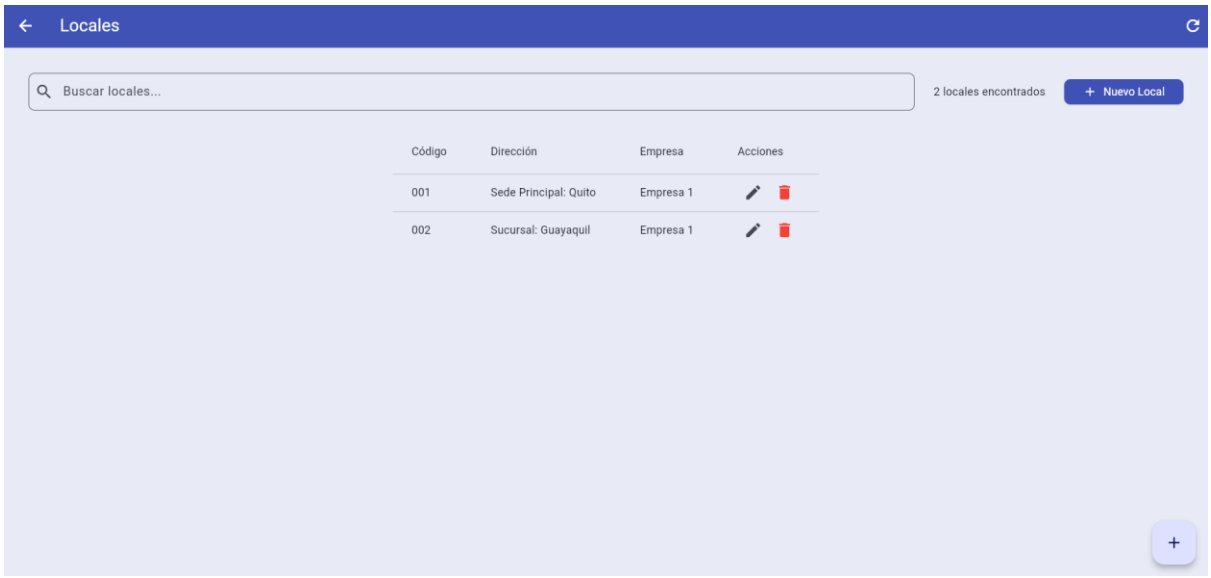
Nota: Información de al empresa, donde se puede actualizar, eliminar o crear una nueva empresa de ser necesario.

Ilustración 26 Creación de Empresa



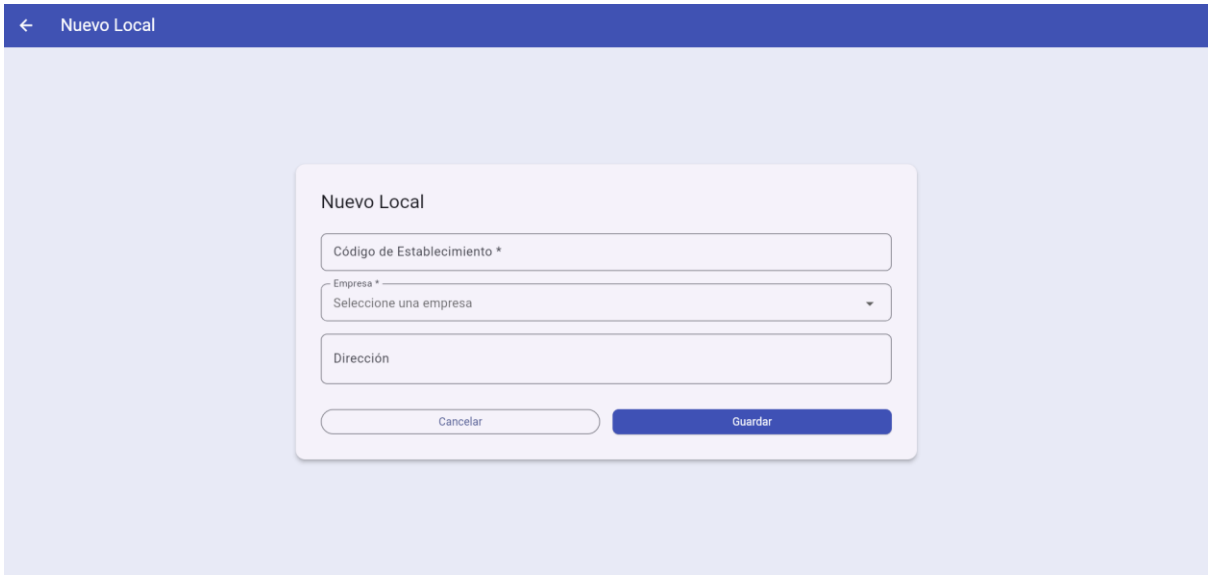
Nota: Pantalla de creación de Empresa con sus respectivos campos.

Ilustración 27 Listado de Locales



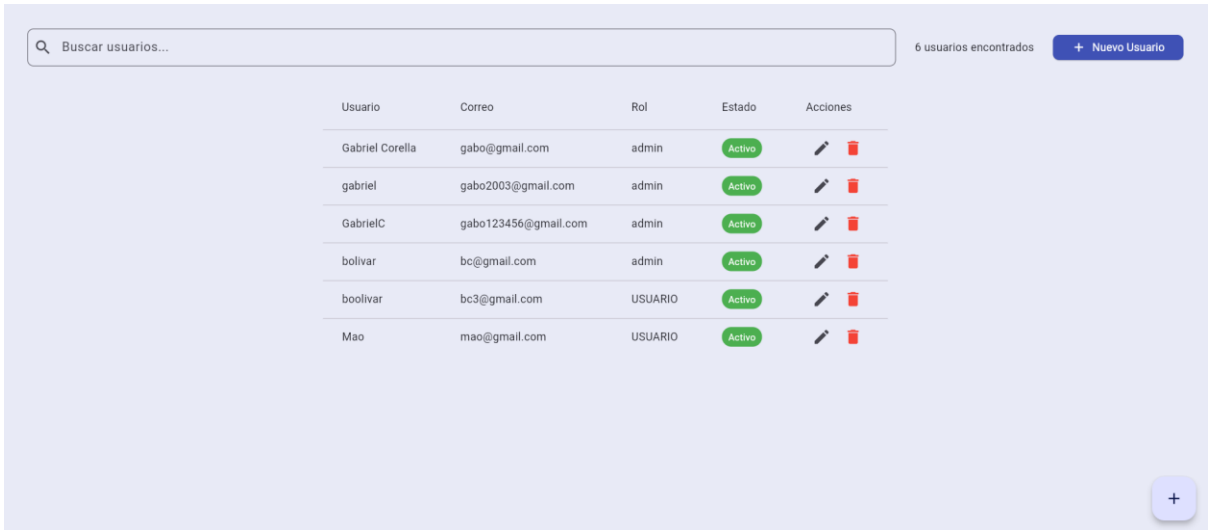
Nota: Listado de locales que pueden ser modificados, eliminados o de ser necesario crear un nuevo local.

Ilustración 28 Creación de Local



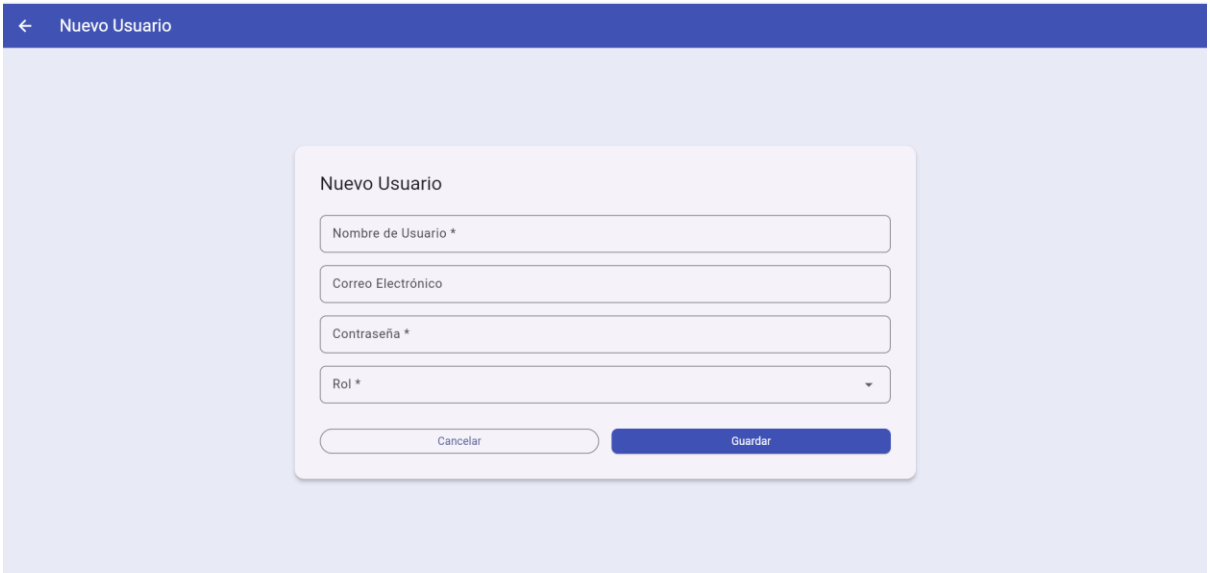
Nota: Pantalla de creación de Local con sus respectivos campos.

Ilustración 29 Listado de Usuarios



Nota: Listado de usuarios de la empresa con sus respectivos roles.

Ilustración 30 Creación de Usuarios



Nota: Pantalla de creación de Usuario con sus respectivos campos.

Capítulo V: Conclusiones y Recomendaciones

5.1 CONCLUSIONES

1. La selección y aplicación de XP como metodología de desarrollo demostró ser altamente efectiva para el desarrollo del prototipo de facturación electrónica. Las prácticas de Test-Driven Development (TDD) e integración continua garantizaron la calidad del código desde las primeras iteraciones, mientras que las entregas incrementales permitieron validar continuamente los requisitos del SRI. La reducción del overhead de XP frente a Scrum maximizó el tiempo de desarrollo efectivo, aspecto crucial en un proyecto de titulación.

2. Con el siguiente proyecto se demostró exitosamente la viabilidad técnica de la migración de un sistema ERP de escritorio hacia plataformas web mediante una arquitectura de tres capas basada en servicios RESTful. La integración entre Flutter como frontend multiplataforma, ASP.NET Web API como backend y SQL Server como motor de base de datos, proporcionó una base sólida y escalable. Esta combinación tecnológica no solo cumple con los requisitos actuales de facturación electrónica, sino que establece una base sólida para futuras expansiones del sistema.

3. La implementación del prototipo evidenció que el cumplimiento de los estándares XML del SRI valida su viabilidad técnica y operativa para Brother Corp. La capacidad del sistema para generar archivos XML conformes al esquema tributario del Ecuador posiciona a Brother Corp como una empresa apta para capitalizar oportunidades de mercado que sus competidores con sistemas legacy no pueden aprovechar, teniendo en cuenta las herramientas modernas que podemos encontrar el día de hoy.

4. La selección del ecosistema Microsoft se toma como una decisión acertada para el contexto actual de la empresa Brother Corp. La integración nativa entre componentes, herramientas de desarrollo modernas y el soporte corporativo a largo plazo proporcionaron un entorno de desarrollo eficiente y una plataforma de producción confiable. Esta coherencia tecnológica redujo significativamente la curva de aprendizaje del equipo y minimizó riesgos de integración.

5.2 RECOMENDACIONES

1. Se propuso, como paso fundamental, desarrollar e incorporar un sistema del módulo de facturación casi completo. Hasta ahora, el prototipo ha sido capaz de generar XML válidos. Aunque la arquitectura de tres capas es adecuada para el prototipo, la migración completa del ERP requerirá mayor flexibilidad. Los microservicios permitirán actualizaciones independientes, mejor tolerancia a fallos y distribución de carga más eficiente durante picos de facturación masiva.
2. Aunque el prototipo fue validado en un ambiente controlado, su comportamiento bajo condiciones reales de uso aún no se ha verificado. Por ello, se recomendó realizar una prueba de carga con un grupo reducido de clientes, lo que permitiría medir la estabilidad y la experiencia de usuario en escenarios de negocio.
3. Se propone implementar un framework de seguridad robusto que incluya autenticación multi-factor, autorización basada en roles, cifrado de datos sensibles tanto en tránsito como en reposo, y auditoría completa de transacciones tributarias. Por otro lado, se puede considerar la integración con Azure Active Directory para gestión centralizada de identidades.
4. Dado que los datos de facturación contienen información valiosa para la toma de decisiones, se planteó para un próximo desarrollo la implementación de dashboards interactivos que muestren indicadores clave como; ventas filtradas por cliente o periodo, productos con mayor volumen de facturación y tiempos promedio de emisión de comprobantes. Este módulo de inteligencia de negocio también incluiría alertas tempranas, que notifiquen al equipo responsable acerca de caídas en los volúmenes de facturación o de errores recurrentes en la emisión, además se contempló la posibilidad de exportar reportes en formatos ejecutivos, tales como PDF o Excel.

VI: Referencias bibliográficas

Amazon. (2024). *¿Qué es XML?* Obtenido de AWS: https://aws.amazon.com/es/what-is/xml/

Angular. (2025). *Introduction to the Angular docs*. Obtenido de Angular: https://v17.angular.io/docs

Beck, K. (2004). Extreme Programming Explained. En K. Beck, *Extreme Programming Explained*. Boston: Addison-Wesley.

César Rodríguez, R. D. (2015). ¿Por qué implementar Scrum? *DIALNET*, 125-144.

Dart. (2024). *Introduction to Dart*. Obtenido de Dart: https://dart.dev/language

Drumond, C. (2025). *Qué es scrum y cómo empezar*. Obtenido de Atlassian: https://www.atlassian.com/es/agile/scrum

ecma. (2023). *C# Language Specification*.

Flutter. (2024). *Build beautiful native apps in record time*. Obtenido de Flutter: https://flutter-website-staging.firebaseapp.com/

Kenneth C. Laudon, J. L. (2016). *Sistemas de información gerencial*. Cuidad de Mexico: Pearson.

Lutz, M. (2009). *Learning Python*. O'REILLY.

Marcotte, E. (2014). *Responsive web Desing* .

Microsoft. (2012). *CLR via C#*. : Online Training Solutions, Inc.

Microsoft. (18 de 04 de 2018). *Flutter*. Obtenido de Dart Code: https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter

Microsoft. (2022). *Notas de la versión de Visual Studio 2022*. Obtenido de Microsoft: https://learn.microsoft.com/es-mx/visualstudio/releases/2022/release-notes

Microsoft. (2024). *ASP.NET Core*. Obtenido de Build RESTful APIs with ASP.NET Web API: https://learn.microsoft.com/en-us/aspnet/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api

Microsoft. (2025). *Documentación de Microsoft SQL*. Obtenido de Microsoft: https://learn.microsoft.com/es-mx/sql/?view=sql-server-ver16

Microsoft. (s.f.). *Documentación de .NET Framework*. Obtenido de Microsoft: https://learn.microsoft.com/es-mx/dotnet/framework/

Microsoft. (s.f.). *Introducción a Entity Framework*. Obtenido de .NET: https://learn.microsoft.com/es-mx/dotnet/framework/data/adonet/ef/overview

Microsoft. (s.f.). *Migraciones de Code First*. Obtenido de .NET: https://learn.microsoft.com/es-es/ef/ef6/modeling/code-first/migrations/

Oracle. (2025). *MySQL*. Obtenido de Oracle: https://dev.mysql.com/doc/

Pautasso, C. (2009). Composing RESTful services with JOpera. Faculty of Informatics, University of Lugano, Switzerlan: Pautasso.

Pérez, J. G. (2022). Implementación de sistemas de facturación electrónica en pymes ecuatorianas: Beneficios y desafíos. *Revista de Tecnología y Negocios*, 45-58. Obtenido de Revista de Tecnología y Negocios.

powerdesigner. (2024). *PowerDesigner Main Features*. Obtenido de PowerDesigner: https://www.powerdesigner.biz/EN/powerdesigner/powerdesigner-features.html?

Python. (2025). *The Python Language Reference*. Obtenido de Python: https://docs.python.org/3/reference/#the-python-language-reference

Radigan, D. (2025). *Kanban*. Obtenido de Atlassian: https://www.atlassian.com/es/agile/kanban

Skeet, J. (2019). *C# In depth*. NW: MANNING.

Springett, S. (2024). *OWASP Top Ten*. Obtenido de OWASP : https://owasp.org/www-project-top-ten/

SRI. (2025). *SRI*. Obtenido de Servicio de Rentas Internas: https://www.sri.gob.ec/web/intersri/home

VII: Anexos:

7.1 Acta de Reuniones

Acta de Reuniones Sistema de Facturación

Fecha	Desarrollador	Cliente	Detalles
4/04/2025			Planificación Inicial y Diseño de la Base de Datos
11/04/2025			Implementación de la Base de Datos y Pruebas Iniciales
18/04/2025			Inicio del Desarrollo del API REST (Capa de Negocio)
25/04/2025			Desarrollo de Funcionalidades CRUD y Pruebas Unitarias
2/05/2025			Implementación de Lógica de Negocio de Facturación (XML)
9/05/2025			Finalización de la API y Publicación en IIS
16/05/2025			Diseño de la Interfaz en Flutter y Planificación de la UI
23/05/2025			Desarrollo de Funcionalidades de UI y Feedback de Usabilidad
30/05/2025			Flujo Completo de Facturación en la App y Pruebas Integradas
6/06/2025			Pruebas Finales, Optimización y Refactorización
13/06/2025			Preparación de Entrega Final y Revisión de Requisitos
20/06/2025			Entrega Final, Documentación y Cierre del Proyecto

7.2 Acta de Anexos

ACTA DE ANEXOS A LAS ACTAS APROBATORIAS DE DOCUMENTOS

1. Asistentes

- Sr. Bolívar Corella – Gerente General

2. Orden del Día

1. Lectura y anexión de las actas aprobatorias de los siguientes documentos de tesis: Casos de Uso, Especificación de Requerimientos, Modelo de Base de Datos.
2. Aprobación por parte del Gerente General.
3. Firma de la presente acta.

3. Desarrollo

3.1. Lectura de Actas Aprobatorias

El Sr. Gabriel Corella procede a dar lectura resumida de las actas aprobatorias correspondientes a cada uno de los documentos listados a continuación, las cuales se incorporan como anexos al presente:

- Anexo I: Acta de Aprobación de Casos de Uso
- Anexo II: Acta de Aprobación de Especificación de Requerimientos
- Anexo III: Acta de Aprobación del Modelo de Base de Datos

3.2. Verificación de Integridad

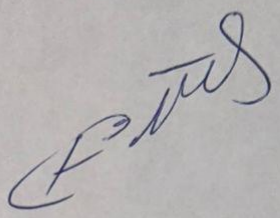
Se confirma que cada acta contiene fecha, firma de los participantes y el detalle de los entregables revisados.

4. Aprobación

El Sr. Bolívar Corella, en su calidad de Gerente General, declara:
"He revisado y acepto formalmente los documentos:
1. Casos de Uso (Anexo I)
2. Especificación de Requerimientos (Anexo II)
3. Modelo de Base de Datos (Anexo III)
correspondientes al proyecto de tesis, los cuales quedan aprobados en todas sus partes."

Quito, Ecuador 10 de mayo de 2025

5. Firma

Nombre	Cargo	Firma
Bolívar Corella	Gerente General	

Anexos

- Anexo I: Acta de Aprobación de Casos de Uso
- Anexo II: Acta de Aprobación de Especificación de Requerimientos
- Anexo III: Acta de Aprobación del Modelo de Base de Datos

7.3 Acta de Aceptación

Acta de Aceptación del Sistema

Proyecto: Sistema de Facturación Electrónica prototipo
Cliente: Brother Corp
Desarrollador: Gabriel Corella
Fecha: 22/06/2025

En la ciudad de Quito, a los 22 días del mes de junio de 2025, se reúnen las partes interesadas para proceder a la revisión final y aceptación del Sistema de Facturación Electrónica desarrollado como prototipo para Brother Corp.

Se acuerda que el sistema cumple con los requerimientos funcionales y no funcionales establecidos en el documento de especificaciones y validado mediante pruebas de aceptación realizadas entre las semanas 9 y 11 del proyecto. A continuación, se detallan los puntos evaluados:

- Gestión de clientes: creación, edición, consulta y desactivación.
- Gestión de productos: creación, edición, consulta e inactivación.
- Configuración de datos de la empresa y sucursales.
- Emisión de facturas electrónicas: captura de cabecera y detalle, generación de XML conforme al SRI.
- Autenticación mediante token JWT y almacenamiento seguro de credenciales.
- API RESTful consumible desde la red interna, desplegada en IIS.
- Interfaz Flutter: flujo completo de facturación y usabilidad.
- Mecanismos de trazabilidad: tabla de logs de facturación.
- Pruebas finales de rendimiento.

Conforme a lo anterior, el cliente acepta formalmente el prototipo del sistema y autoriza su uso en el entorno de pruebas interno.

Nombre y Firma	Rol / Cargo
Gerente General de Brother Corp	Cliente Brother Corp
Gabriel Corella	Desarrollador Jr.