Desarrollo de un sistema de facturación electrónica como servicio (SaaS) con arquitectura multitenant

Lara Caicedo, Nicolás Andrés

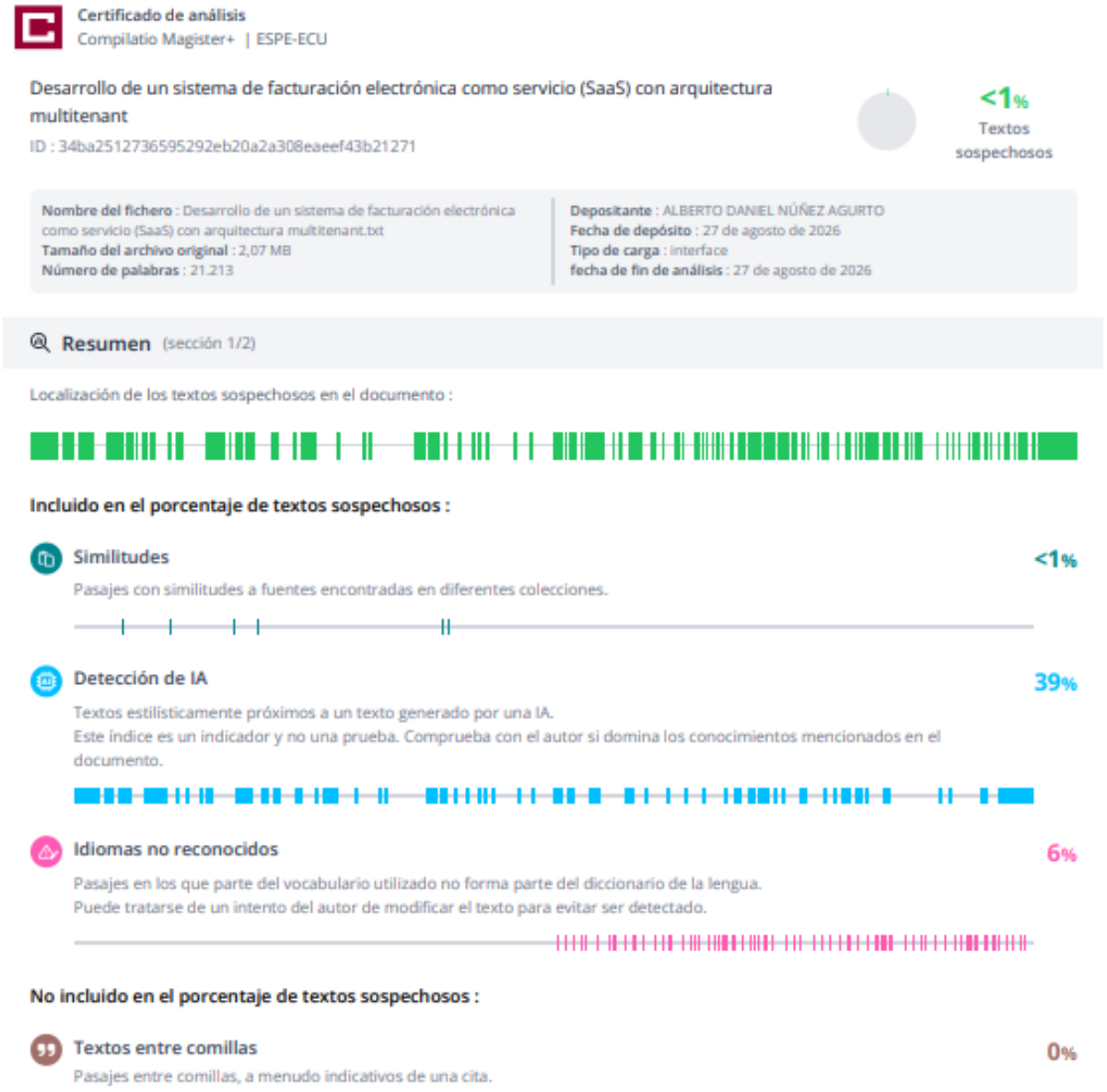
Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Trabajo de integración curricular, previo a la obtención del título de Ingeniero en Tecnologías de Información

Ph.D. Mgtr. Ing. Núñez Agurto, Alberto Daniel.

27 de julio de 2026



Certificado de análisis
Compilatio Magister+ | ESPE-ECU

Desarrollo de un sistema de facturación electrónica como servicio (SaaS) con arquitectura multitenant

ID : 34ba2512736595292eb20a2a308eaeef43b21271

<1% Textos sospechosos

Nombre del fichero : Desarrollo de un sistema de facturación electrónica como servicio (SaaS) con arquitectura multitenant.txt
Tamaño del archivo original : 2,07 MB
Número de palabras : 21.213

Depositante : ALBERTO DANIEL NÚÑEZ AGURTO
Fecha de depósito : 27 de agosto de 2026
Tipo de carga : interface
fecha de fin de análisis : 27 de agosto de 2026

Resumen (sección 1/2)

Localización de los textos sospechosos en el documento :

Incluido en el porcentaje de textos sospechosos :

Similitudes <1%
Pasajes con similitudes a fuentes encontradas en diferentes colecciones.

Detección de IA 39%
Textos estilísticamente próximos a un texto generado por una IA.
Este índice es un indicador y no una prueba. Comprueba con el autor si domina los conocimientos mencionados en el documento.

Idiomas no reconocidos 6%
Pasajes en los que parte del vocabulario utilizado no forma parte del diccionario de la lengua.
Puede tratarse de un intento del autor de modificar el texto para evitar ser detectado.

No incluido en el porcentaje de textos sospechosos :

Textos entre comillas 0%
Pasajes entre comillas, a menudo indicativos de una cita.

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
ALBERTO DANIEL NUNEZ AGURTO

Ph.D. Mgtr. Ing. Núñez Agurto, Alberto Daniel.

Director

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Certificación

Certifico que el trabajo de integración curricular: "**Desarrollo de un sistema de facturación electrónica como servicio (SaaS) con arquitectura multitenant**" fue realizado por el señor **Lara Caicedo, Nicolás Andrés,** el mismo que cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE; habiéndose efectuado una revisión integral de forma y de fondo conforme a los lineamientos y formatos institucionales vigentes. Además, fue revisado y analizada en su totalidad por la herramienta de prevención y/o verificación de similitud de contenidos; razón por la cual me permito acreditar y autorizar para que se lo sustente públicamente.

Santo Domingo de los Tsáchilas, 27 de julio de 2026

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
ALBERTO DANIEL
NUÑEZ AGURTO

Ph.D. Mgtr. Ing. Núñez Agurto, Alberto Daniel.
Director
C.C.: 1716572548

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Responsabilidad de Autoría

Yo, **Lara Caicedo, Nicolás Andrés**, con cédula de ciudadanía n°2350710287 declaro que el contenido, ideas y criterios del trabajo de integración curricular**: Desarrollo de un sistema de facturación electrónica como servicio (SaaS) con arquitectura multitenant** es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos, y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Santo Domingo de los Tsáchilas, 27 de julio de 2026

Lara Caicedo, Nicolás Andrés

C.C.: 2350710287

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Autorización de Publicación

Yo **Lara Caicedo, Nicolás Andrés**, con cédula de ciudadanía n°2350710287, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de integración curricular: **Título: Desarrollo de un sistema de facturación electrónica como servicio (SaaS) con arquitectura multitenant** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Santo Domingo de los Tsáchilas, 27 de julio de 2026

Lara Caicedo, Nicolás Andrés

C.C.: 2350710287

Dedicatoria

A mi madre, Dorie Caicedo, por ser el pilar sobre el que se sostiene todo lo que soy. Por cada sacrificio silencioso, cada desvelo y cada palabra de aliento que me sostuvo cuando el camino se hizo cuesta arriba. Este logro es tan tuyo como mío.

A mi padre, Mauricio Lara, por enseñarme con el ejemplo el valor del trabajo constante y por confiar en mis decisiones incluso cuando el destino no era claro.

A mi hermano, Santiago Lara, por acompañarme en cada etapa, por ser cómplice, apoyo y razón para seguir adelante.

A mi abuelito, Jorge Caicedo, por sus enseñanzas, su cariño y por recordarme siempre de dónde vengo.

A mis amigos Josué, Jair, Raúl y Brandon, por hacer de estos años universitarios un tiempo que recordaré con alegría.

Agradecimiento

Quiero expresar mi más sincero agradecimiento a Dios, por darme la salud, la fortaleza y la perseverancia necesarias para culminar esta etapa de mi formación profesional. A mi familia, Dorie Caicedo, Mauricio Lara, Santiago Lara y Jorge Caicedo, por su apoyo incondicional a lo largo de toda mi carrera. Su respaldo constante, tanto material como emocional, hizo posible que hoy alcance esta meta.

A la Universidad de las Fuerzas Armadas ESPE y a sus docentes, quienes a lo largo de estos años compartieron sus conocimientos y experiencia, formándome no solo como profesional sino también como persona. De manera especial, agradezco a mi tutor de tesis, por su guía, paciencia y valiosas observaciones durante el desarrollo de este trabajo, cuyo aporte fue fundamental para llevarlo a buen término. A mis amigos Josué, Jair, Raúl y Brandon, por su compañía, su ayuda desinteresada y por los momentos compartidos dentro y fuera de las aulas.

Índice de contenidos

Índice de tablas

Índice de figuras

Resumen

En Ecuador, la facturación electrónica es una obligación tributaria regulada por el SRI que exige comprobantes XML firmados bajo el estándar XAdES-BES y transmitidos mediante SOAP. Las soluciones disponibles para las PYMES presentan limitaciones críticas: los sistemas locales impiden actualizaciones normativas centralizadas y las plataformas en la nube carecen de aislamiento de datos, generando riesgos de fuga de información y costos de infraestructura ociosa. Ante este problema, se desarrolló una plataforma SaaS multitenant utilizando Next.js, Node.js con TypeScript y Express, PostgreSQL, Sequelize ORM y Redis. El aislamiento de datos entre empresas se implementó mediante el identificador empresa_id propagado automáticamente en cada consulta a través de middlewares JWT, garantizando que cada contribuyente acceda únicamente a su propia información dentro de una infraestructura compartida. La arquitectura incorporó Redis como capa de caché para reducir la latencia, mientras que el módulo de emisión integró el firmado digital XAdES-BES y la comunicación con los servicios web SOAP del SRI. El sistema fue construido con Scrum en seis sprints, totalizando 661 horas, cubriendo los 25 requerimientos funcionales y los siete tipos de comprobantes exigidos por el SRI. Su funcionamiento se comprobó mediante pruebas de caja negra, pruebas de estrés con Locust y una auditoría bajo OWASP Top 10, que implementó control de acceso con roles, cifrado de credenciales, prevención de inyección SQL y trazabilidad de auditoría. Los resultados evidenciaron el cumplimiento de la totalidad de los casos de prueba y un comportamiento estable bajo carga, demostrando que la solución es viable, segura y escalable para la gestión tributaria de múltiples contribuyentes en Ecuador.

Palabras clave: Facturación electrónica, SaaS, arquitectura multitenant, XAdES-BES, SRI

Abstract

In Ecuador, electronic invoicing is a tax obligation regulated by the SRI (Internal Revenue Service), which requires XML invoices signed under the XAdES-BES standard and transmitted via SOAP. Available solutions for SMEs have critical limitations: local systems prevent centralized regulatory updates, and cloud platforms lack data isolation, generating risks of data leaks and idle infrastructure costs. To address this problem, a multi-tenant SaaS platform was developed using Next.js, Node.js with TypeScript and Express, PostgreSQL, Sequelize ORM, and Redis. Data isolation between companies was implemented using the company_id identifier, automatically propagated in each query via JWT middleware, ensuring that each taxpayer accesses only their own information within a shared infrastructure. The architecture incorporated Redis as a caching layer to reduce latency, while the issuance module integrated XAdES-BES digital signing and communication with the SRI's SOAP web services. The system was built using Scrum in six sprints, totaling 661 hours, covering the 25 functional requirements and the seven types of invoices required by the SRI. Its operation was verified through black-box testing, stress testing with Locust, and an audit under the OWASP Top 10 standard, which implemented role-based access control, credential encryption, SQL injection prevention, and audit trails. The results showed compliance with all defined test cases and stable behavior under load, demonstrating that the solution is viable, secure, and scalable for tax management of multiple taxpayers in Ecuador.

Keywords: Electronic invoicing, SaaS, multitenant architecture, XAdES-BES, SRI

CAPÍTULO I

Introducción

En el Ecuador, la facturación electrónica constituye un mecanismo tributario de carácter obligatorio regulado por el Servicio de Rentas Internas (SRI) [1]. A través de las resoluciones NAC-DGERCGC13-00236 y NAC-DGERCGC14-00157 [2], [3], el SRI estableció el cronograma de obligatoriedad para la emisión de comprobantes electrónicos tanto en el sector público como en el privado, abarcando documentos como facturas, notas de crédito, notas de débito, comprobantes de retención, guías de remisión, liquidaciones de compra y notas de venta bajo el Régimen Impositivo Simplificado Ecuatoriano (RISE) [4]. En conjunto, este marco normativo define las condiciones técnicas y legales que todo sistema de facturación electrónica debe satisfacer para operar de manera válida en el país.

Esta normativa exige que cada comprobante sea generado en formato XML conforme a los esquemas definidos por el SRI [5], firmado digitalmente mediante el estándar XAdES-BES con un certificado P12 reconocido por una entidad de certificación autorizada, y transmitido a los servicios web del SRI para su recepción y autorización. Estos requisitos evidencian que la emisión electrónica no constituye un proceso trivial, sino una secuencia técnica rigurosa que condiciona el diseño de cualquier plataforma que pretenda automatizarla.

A pesar de la existencia de este marco legal, muchas pequeñas y medianas empresas (PYMEs) del país enfrentan dificultades para acceder a soluciones tecnológicas integrales debido a los altos costos de adquisición, licencias e infraestructura que demandan los sistemas tradicionales [6]. Considerando que la facturación electrónica representa un paso fundamental hacia la modernización de la gestión financiera [7], el presente trabajo de titulación propone el desarrollo de un sistema de facturación electrónica bajo el modelo de Software como Servicio (SaaS), sustentado en una arquitectura multitenant**.** Este enfoque permite a múltiples empresas (tenants) compartir de forma eficiente y segura la misma infraestructura y lógica de negocio, reduciendo drásticamente los costos operativos para los contribuyentes.

Planteamiento del problema

El proceso de emisión de un comprobante electrónico en Ecuador implica una cadena de pasos técnicos de considerable complejidad que las empresas deben ejecutar de forma correcta y oportuna para cumplir con sus obligaciones tributarias. Cada documento debe ser construido en un archivo estructurado XML siguiendo esquemas específicos según su tipo, firmado digitalmente

mediante el estándar criptográfico XAdES-BES con un certificado PKCS#12, enviado al servicio web de recepción del SRI mediante el protocolo SOAP, y posteriormente consultado mediante hilos síncronos o asíncronos para obtener su estado de autorización [6]. Esta complejidad técnica representa una barrera significativa para las empresas que carecen de herramientas especializadas, lo que justifica la necesidad de soluciones que automaticen y simplifiquen dicho proceso.

Esta complejidad se multiplica al considerar que la normativa vigente contempla diversos tipos de comprobantes (facturas, notas de crédito, notas de débito, retenciones y guías de remisión), cada uno con sus propias reglas de negocio y validaciones, sumado a la constante evolución y actualización de los esquemas técnicos dictados por la autoridad tributaria.

Desde la perspectiva de la ingeniería de software, las soluciones actuales presentan deficiencias arquitectónicas importantes. Los sistemas tradicionales de escritorio o distribuidos dificultan el mantenimiento y la escalabilidad, ya que cualquier cambio normativo del SRI obliga a actualizar cada instalación por separado, elevando los costos y el riesgo de incumplimiento legal. A esto se suma que, al migrar hacia la nube, muchas plataformas de bajo costo carecen de arquitecturas multitenancy, lo que puede derivar en fugas de información financiera entre organizaciones o en bloqueos del sistema ante alta concurrencia.

Asimismo, la persistencia de datos tradicional carece de mecanismos de optimización de rendimiento. El flujo constante de consultas hacia catálogos maestros y el procesamiento síncrono de documentos XML e interfaces web pesadas elevan drásticamente la latencia de respuesta, superando los umbrales tolerables para la operación comercial en tiempo real. A esto se suma la vulnerabilidad perimetral de las API convencionales, las cuales carecen de tipado estricto y abstracciones seguras para mitigar ataques por inyección o accesos no autorizados a nivel de organización.

Las empresas que no cuentan con un sistema unificado para este proceso recurren habitualmente a soluciones fragmentadas: una gestión manual e ineficiente desde el portal estatal inviable a mediano y alto volumen de transacciones o software propietario con costos de instalación, licencias e infraestructura elevados que resultan prohibitivos para el sector de las PYMEs [8]. A nivel de gestión empresarial, la ausencia de una plataforma integrada con los módulos de clientes, proveedores, productos y control de secuenciales obliga a los administradores a mantener datos duplicados en distintas herramientas, perdiendo la trazabilidad de los documentos

emitidos [9]. Esta fragmentación operativa refleja la carencia de una herramienta centralizada capaz de integrar la gestión tributaria y comercial en un solo entorno de trabajo.

Finalmente, la falta de funcionalidades automatizadas para procesos críticos como la ejecución de tareas programadas para cobros recurrentes, el procesamiento automático de firmas digitales y la notificación automatizada de los archivos RIDE y XML degrada la agilidad operativa de los negocios. Por lo tanto, el problema radica en la ausencia de una plataforma tecnológica centralizada, multitenant y de alto rendimiento que resuelva la complejidad del ciclo tributario de extremo a extremo de forma aislada, segura y accesible, eliminando la alta latencia en consultas y garantizando la adaptabilidad normativa inmediata para múltiples organizaciones concurrentes.

Formulación del problema o pregunta de investigación

¿Cómo diseñar y desarrollar un sistema web de facturación electrónica bajo el modelo SaaS y con arquitectura multitenant, utilizando un frontend en Next.js y un backend en Node.js, TypeScript, Express, PostgreSQL y Sequelize, que centralice, automatice y aísle de forma segura la gestión y emisión de comprobantes tributarios electrónicos (facturas, notas de crédito, notas de débito, retenciones, guías de remisión, liquidaciones de compra y notas de venta) integrados a los servicios SOAP del SRI, en conformidad con la normativa vigente en Ecuador?

Objetivos

Objetivo general:

Desarrollar un sistema de facturación electrónica basado en el modelo SaaS, utilizando una arquitectura multitenant, que permita a múltiples empresas gestionar sus procesos de facturación de forma segura, escalable y centralizada, cumpliendo con los requisitos funcionales y normativos vigentes.

Objetivos específicos:

- Analizar y definir los requisitos funcionales, técnicos y normativos necesarios para el desarrollo de un sistema de facturación electrónica bajo un modelo SaaS multitenant.
- Diseñar la arquitectura del sistema de facturación electrónica bajo el modelo SaaS con enfoque multitenant, garantizando el aislamiento lógico de datos y la gestión eficiente de múltiples empresas.
- Implementar los módulos principales del sistema de facturación electrónica, incorporando mecanismos de seguridad y control por tenant.

- Validar el funcionamiento, desempeño y seguridad del sistema mediante pruebas funcionales, de rendimiento y de mitigación de vulnerabilidades.

Justificación

Justificación Tecnológica

Desde la perspectiva de la ingeniería de software, este proyecto se justifica por la necesidad de implementar arquitecturas altamente eficientes y escalables mediante el modelo SaaS con enfoque multitenant. La adopción de una estrategia de aislamiento de datos a nivel de base de datos compartida mediante el identificador único empresa_id, administrado a través del ORM Sequelize y PostgreSQL, representa una solución óptima para mitigar los problemas de concurrencia y optimizar el uso de recursos de hardware en el servidor.

Asimismo, la integración del stack tecnológico compuesto por Next.js en el frontend y Node.js con TypeScript en el backend representa una propuesta de diseño moderna e innovadora. Esta aproximación introduce un modelo arquitectónico desacoplado y unificado bajo un mismo lenguaje de programación (Full-Stack TypeScript), lo que optimiza la eficiencia del desarrollo, disminuye la latencia transaccional y eleva los estándares de mantenibilidad frente a las soluciones tradicionales del mercado, capaz de soportar de manera segura y centralizada procesos críticos como el firmado digital XAdES-BES y el consumo de los servicios web SOAP del SRI.

Justificación Práctica

En el ámbito operativo, los sistemas tradicionales de facturación electrónica en el Ecuador suelen demandar infraestructura dedicada o instalaciones locales independientes para cada negocio, lo que eleva drásticamente los costos de mantenimiento técnico y licenciamiento. La plataforma propuesta resuelve este problema de raíz al permitir que múltiples empresas coexistan de forma aislada en una única infraestructura. Esto transforma radicalmente la mantenibilidad del software: ante cualquier modificación técnica o actualización normativa dispuesta por la autoridad tributaria, el sistema se actualiza centralizadamente en el servidor. Con esto, se garantiza que todos los tenants

operen bajo la ley de manera inmediata y transparente, eliminando los despliegues individuales y mitigando el riesgo de desfase legal o sanciones tributarias.

Justificación Social y Económica

A nivel socioeconómico, este trabajo de titulación proporciona una alternativa tecnológica accesible y de bajo costo operativo para las PYMEs y contribuyentes naturales del país, quienes comúnmente enfrentan barreras económicas para adquirir sistemas ERP o software propietario costoso. Al unificar en una sola herramienta la gestión de clientes, proveedores, productos y la emisión automatizada de los siete comprobantes electrónicos obligatorios (facturas, notas de crédito, notas de débito, retenciones, guías de remisión, liquidaciones de compra y notas de venta), este proyecto se convierte en un motor de transformación digital, agilidad comercial y formalización tributaria para el ecosistema empresarial ecuatoriano.

Alcance del proyecto

El presente proyecto comprende el diseño, desarrollo y despliegue de una plataforma web tipo SaaS orientada a automatizar la facturación electrónica para múltiples empresas, cumpliendo con las normativas del SRI.

El sistema contará con un Frontend web moderno y responsivo, y un Backend con API REST capaz de gestionar de forma independiente la información de cada organización, todo dentro de una base de datos compartida pero correctamente aislada por empresa.

En cuanto a sus funcionalidades principales, la plataforma permitirá procesar firmas electrónicas (PKCS#12), generar y enviar comprobantes XML al SRI tanto en ambiente de pruebas como en producción, emitir automáticamente el RIDE en PDF, y notificar los documentos a los clientes por correo electrónico.

Cada empresa podrá administrar de forma autónoma sus catálogos de clientes, productos, impuestos y puntos de emisión. Además, el sistema incorporará caché en memoria para mejorar el rendimiento, tareas programadas para facturación recurrente, y un dashboard con indicadores de venta.

En materia de seguridad, la plataforma se desarrollará bajo los estándares del OWASP Top 10. El proyecto se dará por concluido una vez realizadas las pruebas de integración, pruebas de carga, auditoría de código y entrega de la documentación técnica y manuales de usuario, garantizando un producto listo para producción.

CAPÍTULO II

Estado del arte

El desarrollo de plataformas de facturación electrónica y la adopción de arquitecturas orientadas a servicios en la nube han sido objeto de constante investigación en el ámbito de la ingeniería de software en el Ecuador. A continuación, se analizan los principales trabajos previos que abordan soluciones tecnológicas adaptadas a las normativas del SRI y arquitecturas eficientes para la persistencia y aislamiento de datos.

En el ámbito nacional, múltiples investigadores han explorado el desarrollo de software para automatizar el ciclo tributario digital exigido por el SRI. En la Escuela Politécnica Nacional, Muñoz Burgos [6] desarrolló un sistema web dedicado a la facturación electrónica implementando procesos para la generación de estructuras XML y el firmado digital bajo el estándar XAdES-BES. No obstante, dicho estudio se enfocó en un modelo arquitectónico tradicional y monolítico diseñado para una estructura empresarial centralizada, lo que limita su capacidad de escalar hacia entornos globales compartidos por múltiples empresas independientes y descentralizadas.

Por otra parte, ante la necesidad de enlazar la facturación con la infraestructura en la nube, en la Universidad Nacional de Chimborazo, Ávalos Trujillo [7] evaluó el despliegue de sistemas fiscales sobre plataformas de servicios integrados (Azure) utilizando metodologías de desarrollo rápido. Su investigación demostró que la computación en la nube optimiza el rendimiento y el control financiero corporativo, pero dejó abierta la necesidad de evaluar arquitecturas capaces de consolidar el almacenamiento de múltiples corporaciones bajo esquemas relacionales más económicos para el sector de las PYMEs.

Asimismo, en la Universidad Politécnica Salesiana, Coral Navarro y Ñacato Ganchala [8] investigaron el desarrollo de un software de facturación enfocado para microempresas en el Ecuador a través de una arquitectura basada en microservicios alojados en la nube de Amazon Web Services (AWS). Si bien los autores lograron demostrar los beneficios financieros de la computación en la nube para reducir costos de hardware en pequeñas organizaciones, la segmentación y el aislamiento de datos se plantearon mediante servicios de computación distribuidos de manera independiente, lo que incrementa la complejidad del despliegue y eleva los costos operativos mensuales de infraestructura al requerir instancias de procesamiento individuales para cada organización.

En una línea de investigación orientada a la unificación de datos de gestión y fiscalidad, Pincay Zavala [9] implementó en la Universidad Estatal del Sur de Manabí un sistema web comercial acoplado con facturación electrónica. Esta investigación demostró que la consolidación de módulos de clientes, proveedores y productos reduce notablemente la redundancia de datos y los errores operativos humanos en la contabilidad. Sin embargo, el diseño del software y el modelado de su base de datos se estructuraron bajo un enfoque de inquilino único, lo que imposibilita que la plataforma sea comercializada de manera directa bajo un modelo de distribución masiva sin requerir múltiples clonaciones del código fuente y de los esquemas de bases de datos.

Como contraparte en el entorno comercial y tecnológico actual del país, existen soluciones consolidadas en el mercado privado bajo el modelo SaaS, como es el caso de la plataforma Amelia, desarrollada por la empresa BeGroup [10]. Este sistema destaca por ofrecer un entorno web integrado para la gestión de comprobantes electrónicos, optimizando la experiencia de usuario y centralizando operaciones comerciales en la nube.

Sin embargo, al tratarse de un software propietario cerrado, los detalles específicos de su arquitectura de software o sus mecanismos internos de aislamiento de datos para la coexistencia de múltiples tenants no son de acceso público para la comunidad académica. Esto genera la necesidad de desarrollar proyectos de investigación independientes que documenten, validen y pongan a prueba arquitecturas multitenant eficientes utilizando stacks modernos y accesibles como Next.js, Node.js y Sequelize.

El diseño de plataformas SaaS exige estrategias sólidas para gestionar y proteger grandes volúmenes de información. Torres Erazo [11], en la Universidad de las Fuerzas Armadas ESPE, realizó un estudio comparativo entre arquitecturas monolíticas y de microservicios aplicadas a la facturación electrónica en Ecuador, evaluadas bajo la norma ISO 25010. Sus resultados indican que los microservicios superan a los sistemas monolíticos en mantenibilidad, escalabilidad y autonomía de despliegue, al permitir que cada servicio se actualice de forma independiente sin afectar al resto del sistema. No obstante, el estudio no aborda los mecanismos de aislamiento de datos necesarios cuando múltiples empresas comparten simultáneamente la misma infraestructura en un entorno multi-tenant.

Finalmente, el reto crítico del aislamiento de datos en sistemas compartidos fue abordado desde la ingeniería de software aplicada. De acuerdo con los patrones de arquitectura de bases de

datos multitenant documentados por Bytebase [12], la persistencia de datos en escenarios SaaS escalables puede resolverse eficientemente mediante tres esquemas de aislamiento: bases de datos independientes por inquilino (database-per-tenant), un esquema dedicado por inquilino (schema-per-tenant), o una base de datos única con un identificador de inquilino compartido (tenant_id). De estos esquemas, el modelo de identificador compartido se perfila como el más adecuado para una solución SaaS orientada a múltiples pequeñas y medianas empresas, al equilibrar el aislamiento lógico de los datos con un uso eficiente de los recursos de infraestructura.

Complementariamente, la guía de arquitectura SaaS de WorkOS [13] demuestra que el uso de un identificador común complementado con capas lógicas del ORM reduce drásticamente el consumo de recursos en el servidor y simplifica el mantenimiento del sistema al ejecutar migraciones centralizadas de manera inmediata para toda la red de usuarios concurrentes, aunque transfiere al desarrollador la responsabilidad total del aislamiento, dado que cualquier omisión del tenant_id en una consulta representa un riesgo directo de fuga de datos entre tenants. Por ello, el control riguroso del identificador de inquilino se convierte en una decisión de diseño crítica, ya que de su correcta aplicación depende la integridad y la confidencialidad de la información de cada empresa.

Marco teórico

En esta sección presenta las bases conceptuales, arquitectónicas, tecnológicas y normativas sobre las que se construye el desarrollo de plataformas orientadas a servicios lógicos distribuidos. A lo largo de este marco teórico se exploran los paradigmas de la computación en la nube, las formas de aislar datos en entornos relacionales compartidos, los mecanismos criptográficos de la firma digital y los protocolos de comunicación que exige la administración tributaria ecuatoriana, todo ello con el propósito de sentar las bases científicas para diseñar soluciones de software que sean, a la vez, escalables y seguras.

Computación en la Nube y Modelo Software como Servicio (SaaS)

La computación en la nube (Cloud Computing) se define como un modelo tecnológico que permite el acceso omnipresente, adaptado y bajo demanda a un conjunto compartido de recursos de computación configurables (redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser aprovisionados y liberados rápidamente con un mínimo esfuerzo de gestión o interacción con el proveedor del servicio [14]. Este paradigma arquitectónico se fundamenta en la

abstracción física del hardware mediante capas de virtualización, optimizando la asignación dinámica de recursos.

Para entender mejor cómo se organiza y gobierna este tipo de soluciones, es importante conocer cómo la industria tecnológica estructura la computación en la nube. En términos generales, esta se clasifica en dos grandes categorías: los modelos de despliegue, que definen dónde y cómo se aloja la infraestructura, y los modelos de servicio, que determinan qué le ofrece el proveedor al usuario final.

Modelos de Despliegue

Los modelos de despliegue determinan la propiedad, el entorno de localización de la infraestructura física y los límites de accesibilidad de los recursos computacionales:

- **Nube Pública:** La infraestructura física y los recursos lógicos son propiedad de un proveedor de servicios externo (como AWS, Google Cloud o Microsoft Azure), quien los administra y distribuye dinámicamente entre múltiples usuarios independientes (tenants) a través de internet. Es la base operativa idónea para sistemas comerciales de alta disponibilidad y bajo coste de entrada.
- **Nube Privada:** Los recursos informáticos se utilizan de forma exclusiva por una sola organización. Ofrece un control estricto sobre la soberanía de los datos y el endurecimiento perimetral, pero exige costos elevados de mantenimiento, licenciamiento y actualización física.
- **Nube Híbrida:** Integra infraestructuras públicas y privadas mediante protocolos de interoperabilidad. Permite que los datos y aplicaciones críticas se ejecuten en entornos privados controlados, mientras que las operaciones masivas o el almacenamiento a largo plazo aprovechan la escalabilidad de la nube pública.

Modelos de Servicio

Los modelos de servicio determinan cómo se distribuyen las responsabilidades entre el equipo de desarrollo y el proveedor de nube. En esencia, definen hasta qué punto el desarrollador tiene control sobre el software, la infraestructura y los recursos tecnológicos, y a partir de qué nivel el proveedor se encarga de gestionar todo lo demás.

- **Infraestructura como Servicio (IaaS):** Representa la capa base o de hardware virtualizado. El proveedor suministra capacidades fundamentales de procesamiento, redes y almacenamiento virtual. El usuario tiene el control del sistema operativo, el

middleware y las aplicaciones, pero se desentiende de la gestión física de los centros de datos.

- **Plataforma como Servicio (PaaS):** Es la capa intermedia. El proveedor abstrae el sistema operativo e implementa un entorno de ejecución completo (motores de bases de datos, *runtimes* de lenguajes y herramientas de compilación). El desarrollador se enfoca estrictamente en la construcción de la aplicación, omitiendo la administración de parches informáticos o aprovisionamiento de servidores.
- **Software como Servicio (SaaS):** Dentro de los modelos de servicio establecidos en la computación en la nube, SaaS representa la capa de abstracción más alta, donde las aplicaciones son alojadas por un proveedor de servicios y puestas a disposición de los usuarios finales a través de una red, comúnmente la internet, utilizando interfaces accesibles como los navegadores web [15]. Este nivel de abstracción es el que permite ofrecer aplicaciones complejas como un servicio accesible desde el navegador, sin que el usuario final deba administrar la infraestructura tecnológica subyacente.

En este modelo, el cliente no gestiona ni controla la infraestructura subyacente de la nube (servidores, sistemas operativos, almacenamiento o redes), ni tampoco las características funcionales específicas de la aplicación, y se exceptúan las configuraciones particulares del entorno de usuario [15]. Desde la perspectiva de la ingeniería de software, el modelo SaaS reduce los costos de adquisición, mantenimiento y despliegue, permitiendo la entrega continua de actualizaciones de manera centralizada y transparente para toda la red de consumidores. Esta naturaleza lo convierte en la solución óptima para dotar a múltiples organizaciones de capacidades transaccionales fiscales sin transferirles la complejidad operativa de la infraestructura.

Arquitectura Multi-Tenant

El modelo SaaS exige una arquitectura Multi-Tenant, en la que una única instancia de la aplicación atiende simultáneamente a múltiples tenants [16]. A diferencia del modelo Single-Tenant, donde cada tenant dispone de su propia instancia dedicada, el enfoque multi-tenant optimiza el uso de recursos de cómputo y simplifica las tareas de mantenimiento y actualización de la plataforma. Uno de los principales desafíos en un sistema multi-tenant es garantizar que los datos de cada tenant permanezcan aislados y protegidos, evitando accesos no autorizados entre cuentas. Para abordar esto, se reconocen tres esquemas fundamentales de persistencia en entornos relacionales compartidos:

- **Base de Datos Independiente (Database-per-tenant):** En este modelo, cada tenant cuenta con su propia base de datos completamente aislada, lo que ofrece el mayor nivel de seguridad y simplifica tareas como los respaldos individuales. Sin embargo, esta independencia tiene un costo: la infraestructura se encarece y la complejidad de mantenimiento crece proporcionalmente a medida que se incorporan nuevos tenants.
- **Esquema por Inquilino (Schema-per-tenant):** En este modelo, los tenants comparten la misma base de datos, pero cada uno dispone de su propio esquema lógico, manteniendo sus datos organizados e independientes. Esto optimiza el uso de recursos frente a una base de datos completamente separada por empresa, aunque las migraciones representan un desafío moderado al deberse aplicar de forma consistente sobre cada esquema existente.
- **Base de Datos Compartida (Shared Database / Discriminator Column):** En este enfoque, todos los tenants comparten las mismas tablas dentro de una única base de datos. El aislamiento se logra mediante una columna indexada, comúnmente llamada empresa_id o tenant_id, que filtra los registros de cada organización en cada consulta. Si bien esto reduce costos operativos y optimiza el uso de recursos del servidor, la responsabilidad del aislamiento recae en la lógica de la aplicación o en mecanismos como la seguridad a nivel de filas (Row-Level Security) que ofrecen los motores relacionales modernos.

La elección del motor de base de datos es una decisión fundamental en una arquitectura Multi-Tenant. En la ingeniería de software, este debate se centra en dos paradigmas: las bases de datos NoSQL (como MongoDB), de esquema flexible, y las bases de datos relacionales - RDBMS (como PostgreSQL), estructuradas en tablas con restricciones de integridad referencial.

Para el presente sistema se seleccionó PostgreSQL, debido a la solidez de sus propiedades ACID y a que aplica las restricciones de integridad directamente en el motor mediante llaves primarias, foráneas e índices compuestos. Esto garantiza que el identificador empresa_id, encargado de separar lógicamente los datos de cada organización, sea siempre obligatorio e inmutable, eliminando cualquier riesgo de filtración de información entre empresas.

Marco Regulatorio de la Facturación Electrónica en Ecuador

El SRI, como organismo técnico y autónomo encargado de la gestión tributaria en el Ecuador, establece las directrices técnicas y legales para la transición obligatoria hacia el esquema

de emisión de comprobantes electrónicos. Este modelo sustituye a los documentos preimpresos físicos por archivos estructurados basados en estándares digitales, garantizando la validez legal, la inalterabilidad de la información y la fiscalización en tiempo real de las transacciones comerciales [17]. De esta manera, el organismo tributario no solo moderniza la gestión fiscal, sino que traslada al contribuyente la responsabilidad de contar con herramientas tecnológicas capaces de cumplir con dichos estándares.

Los comprobantes electrónicos reconocidos por la legislación vigente responden a finalidades tributarias concretas y deben seguir una estructura uniforme que permita su procesamiento posterior de forma ordenada y coherente. A continuación, en la Tabla *1* se describen los principales tipos de comprobantes electrónicos reconocidos por la normativa tributaria ecuatoriana, junto con su función específica dentro del proceso de facturación.

Tabla 1

Tipos de Comprobantes Electrónicos y su Función Tributaria

Comprobante	Descripción y función tributaria
Factura	Comprobante principal de venta de bienes o servicios (código SRI: 01).
Nota de Crédito	Anula parcial o totalmente una factura emitida. Se usa para devoluciones o descuentos.
Nota de Débito	Cobra cargos adicionales sobre una transacción previamente facturada.
Comprobante de Retención	Emitido por el agente retentor al realizar una compra, indicando IVA o Impuesto a la Renta retenido.
Guía de Remisión	Ampara el traslado de mercadería entre establecimientos o hacia el cliente.
Liquidación de Compra	Se emite para adquisiciones a personas naturales sin RUC.
Proforma	Cotización o presupuesto previo a la venta. Sin efecto tributario.

Para garantizar que cada documento fiscal emitido en el país pueda identificarse de manera única e inequívoca, el SRI establece como requisito obligatorio la generación de una clave de acceso numérica de 49 dígitos. Esta clave no es un simple código: condensa información clave tanto de la transacción como del emisor, lo que permite que los sistemas gubernamentales la indexen y validen de forma automatizada sin ambigüedad alguna. Como se muestra en la Tabla *2*,

esta clave se compone de varios campos que codifican información específica del comprobante y su emisor.

Tabla 2

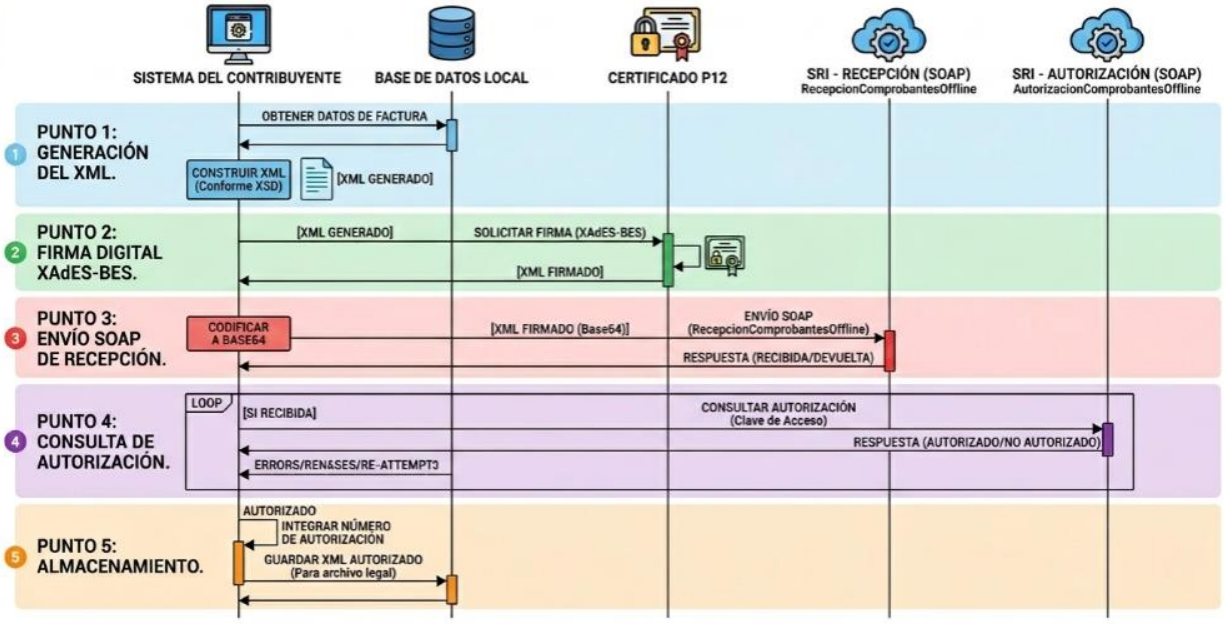
Estructura de la Clave de Acceso de 49 Dígitos

Campo	Descripción
Fecha (8 dígitos)	Fecha de emisión en formato ddmmaaaa
Tipo de comprobante (2 dígitos)	Código del tipo de documento según catálogo SRI
RUC del emisor (13 dígitos)	Número de registro único de contribuyentes
Ambiente (1 dígito)	1 = Pruebas / Certificación, 2 = Producción
Serie (6 dígitos)	Establecimiento (3 dígitos) + Punto de emisión (3 dígitos)
Secuencial (9 dígitos)	Número correlativo del comprobante
Código numérico (8 dígitos)	Generado aleatoriamente por el sistema emisor
Tipo de emisión (1 dígito)	1 = Normal (emisión en línea)
Dígito verificador (1 dígito)	Calculado mediante el Algoritmo Módulo 11

El proceso de facturación electrónica en su modalidad offline, actualmente establecida como el estándar obligatorio, sigue una secuencia de fases bien definidas en las que intervienen el emisor, los algoritmos de firmado digital y los servidores de la administración pública. Como se muestra en la Figura 1, este flujo comprende cinco puntos principales: generación del XML, firma digital XAdES-BES, envío SOAP de recepción, consulta de autorización y almacenamiento del comprobante autorizado.

Figura 1

Diagrama de Flujo del Ciclo de Emisión Electrónica del SRI.



Arquitectura de Aplicaciones Web Modernas y Capa de Presentación

El desarrollo de plataformas empresariales modernas se apoya en arquitecturas desacopladas, donde la interfaz de usuario (Frontend) opera de forma independiente a la lógica de negocio (Backend), comunicándose a través de APIs RESTful sobre HTTPS [18]. Esta separación de responsabilidades favorece la escalabilidad y el mantenimiento de las aplicaciones, al permitir que la capa de presentación y la lógica de negocio evolucionen de forma independiente.

En la capa de presentación, frameworks como Next.js extienden las capacidades de React al combinar renderizado del lado del servidor (SSR) y generación estática (SSG) según las necesidades de cada página. Con SSR, el servidor procesa la petición, consulta las APIs y entrega al navegador un HTML ya construido, lo que mejora los tiempos de carga, reduce el procesamiento en el dispositivo del usuario y mantiene la lógica sensible fuera del alcance del cliente.

Las arquitecturas de aplicaciones web modernas han evolucionado hacia modelos desacoplados basados en el paradigma Cliente-Servidor. En este enfoque, la interfaz de usuario y la lógica de negocio operan de forma independiente, comunicándose exclusivamente mediante el intercambio de datos estructurados (en formato JSON) a través de servicios web seguros.

- **Capa de Presentación (Frontend):** La capa de presentación es el entorno visual con el que interactúa el contribuyente. Para su desarrollo se eligió Next.js, un framework de código abierto basado en React, que incorpora renderizado del lado del servidor (SSR): el servidor pre-renderiza cada página y la entrega lista al navegador, lo que se traduce en una interfaz rápida y responsiva, adecuada para la gestión masiva de comprobantes. La capa visual se desarrolla en TypeScript, cuyo tipado estricto permite detectar errores en tiempo de compilación y garantiza la estabilidad de los formularios contables.
- **Capa de Servicios (Backend):** El backend se construye sobre Node.js, cuya arquitectura asíncrona y modelo de I/O sin bloqueo permite procesar miles de solicitudes XML concurrentes con un consumo mínimo de memoria, algo esencial para la transmisión masiva de comprobantes al SRI. Sobre esta base, Express organiza el enrutamiento y expone la API RESTful, gestionando mediante middlewares la validación de identidad del tenant y la comunicación con el frontend.

  TypeScript complementa esta capa añadiendo tipado estático al flujo de negocio del servidor. Esto resulta especialmente valioso en facturación electrónica, ya que permite definir tipos estrictos para los esquemas del SRI etiquetas XML, estructuras de impuestos, formatos de fechas fiscales de modo que cualquier inconsistencia en un comprobante se detecta en tiempo de compilación, antes de llegar a producción.

Control de Acceso y Seguridad de la Información

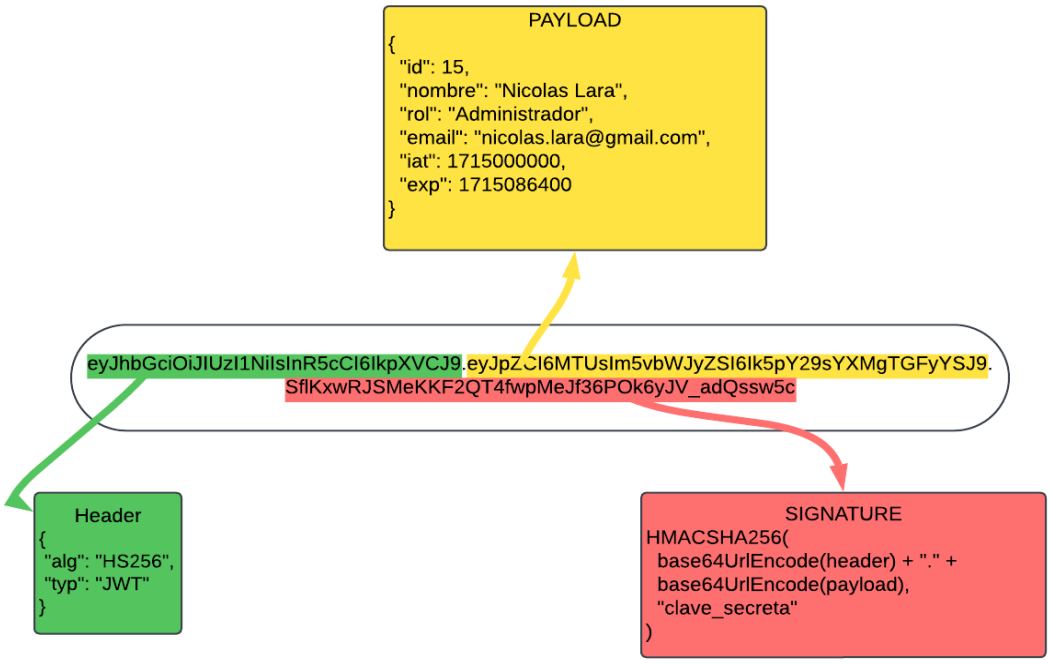
Proteger la confidencialidad, integridad y disponibilidad de la información en entornos web requiere aplicar controles de seguridad sólidos, tanto en el perímetro como en las capas lógicas del sistema. Uno de los enfoques más consolidados para lograrlo es el Control de Acceso Basado en Roles (RBAC, Role-Based Access Control), que organiza los permisos del sistema en torno a roles organizacionales predefinidos en lugar de asignarlos directamente a cada usuario [21]. De esta forma, cada persona accede únicamente a las funciones que su rol le permite, lo que reduce de manera natural el riesgo de escalada de privilegios y evita que información transaccional sensible quede expuesta a quienes no deberían tenerla.

Para la gestión de identidades y la autorización en arquitecturas distribuidas y desacopladas, se emplea el estándar industrial JSON Web Tokens (JWT, RFC 7519). Un JWT es una estructura compacta y autocontenida que permite transmitir información de forma segura entre

las partes como un objeto JSON, cuya veracidad es verificada criptográficamente mediante una firma digital basada en un algoritmo de hash secreto compartido (HMAC) o un par de claves pública/privada (RSA) [19]. Como se muestra en la Figura 2, esta estructura se compone de tres partes separadas por puntos: Header, Payload y Signature, codificadas en Base64Url y concatenadas para formar el token final.

Figura 2

Estructura Conceptual del JSON Web Token

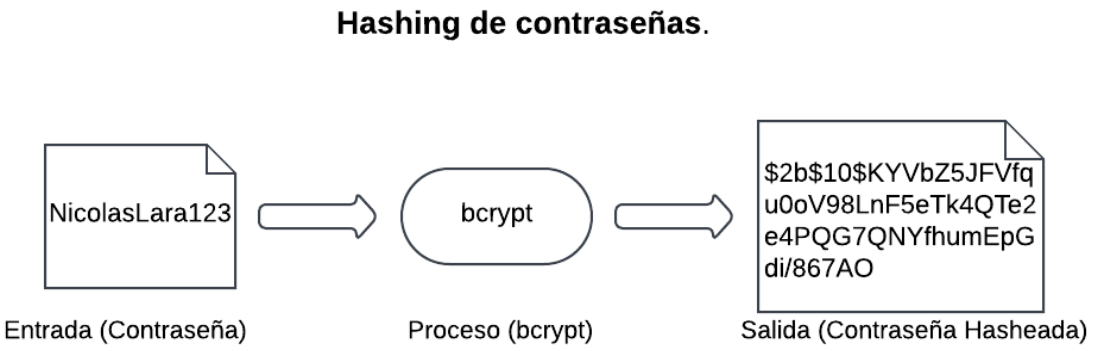


En la capa de persistencia de credenciales, el almacenamiento de contraseñas prohíbe el uso de texto plano debido a vulnerabilidades de fuga de información. Se requiere la aplicación de algoritmos de hash criptográficos unidireccionales de alta resistencia, como Bcrypt, los cuales incorporan intrínsecamente un valor aleatorio denominado Salt (sal) [20]. El proceso de hashing transforma la cadena original en un compendio alfanumérico irreversible, donde la inclusión del Salt previene ataques basados en tablas de arcoíris (rainbow tables) y ataques de fuerza bruta por diccionario, asegurando que contraseñas idénticas generen hashes completamente diferentes. Como se muestra en la Figura 3, este proceso toma la contraseña en texto plano como entrada, la procesa mediante el algoritmo

bcrypt y produce como salida un hash que incorpora tanto el salt como el costo computacional utilizado.

Figura 3

Flujo Conceptual de Hashing de Contraseñas



Adicionalmente, la protección contra ataques dirigidos a las capas de transporte e hipertexto (como Cross-Site Scripting - XSS, Clickjacking y vulnerabilidades de inyección) se refuerza mediante la configuración estricta de cabeceras HTTP de seguridad. Herramientas como Helmet operan como middlewares en entornos de servidor, inyectando de forma automatizada directivas restrictivas en las respuestas enviadas al navegador del cliente.

Como se muestra en la Tabla *3*, estas cabeceras cumplen funciones defensivas específicas frente a distintos vectores de ataque.

Tabla 3

Cabeceras HTTP de Seguridad y su Función Defensiva

Cabecera HTTP	Protección que ofrece
Content-Security-Policy	Previene ataques de Cross-Site Scripting (XSS) restringiendo las fuentes de contenido permitidas.
X-Frame-Options	Previene ataques de Clickjacking impidiendo que la página sea cargada dentro de un iframe.
X-Content-Type-Options	Previene el MIME sniffing, forzando al navegador a respetar el tipo de contenido declarado.

Strict-Transport-Security (HSTS)	Fuerza el uso de HTTPS en todas las comunicaciones futuras con el servidor.
X-XSS-Protection	Activa el filtro de XSS integrado en navegadores más antiguos.
Referrer-Policy	Controla qué información de la URL de origen se envía en peticiones a terceros.

Firma Digital y Protocolos de Comunicación Gubernamental

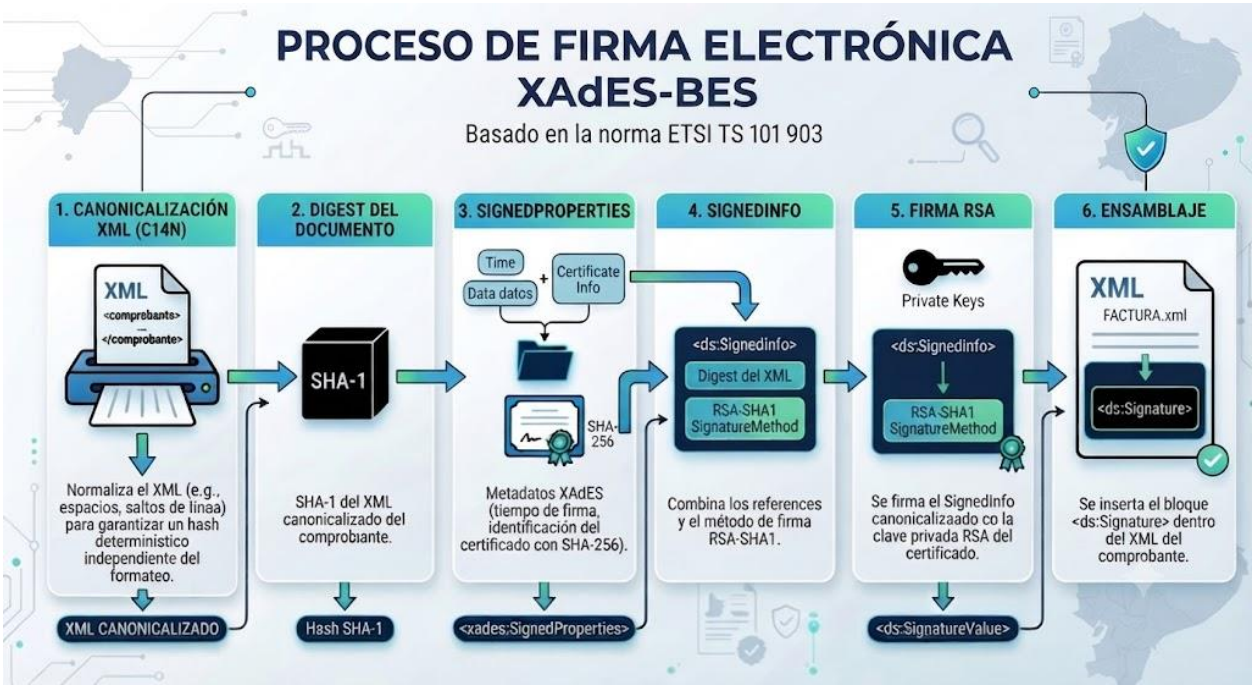
La validez jurídica de un comprobante electrónico en el Ecuador está supeditada a la incorporación de una firma digital avanzada, fundamentada en la criptografía asimétrica. Este mecanismo utiliza un par de claves criptográficas acopladas matemáticamente: una clave privada, resguardada exclusivamente por el emisor para realizar el cifrado del compendio de datos, y una clave pública, distribuida en un certificado digital para que los receptores y el SRI verifiquen la autenticidad e integridad del documento original [21]. Los certificados digitales emitidos por las entidades de certificación acreditadas en el país se encapsulan bajo el estándar PKCS#12 (archivos con extensión .p12), el cual define una sintaxis segura para el almacenamiento conjunto de llaves privadas, certificados públicos y cadenas de confianza protegidas por contraseña.

Para el ámbito fiscal, la estructura del firmado digital debe cumplir estrictamente con el estándar XAdES-BES (XML Advanced Electronic Signatures - Basic Electronic Profile), definido por la norma técnica ETSI TS 101 903 [22]. Este perfil inyecta dentro del documento estructurado XML etiquetas específicas que contienen el valor de la firma, los datos del certificado del firmante y las propiedades de las propiedades firmadas (signed properties), impidiendo cualquier modificación posterior del contenido bajo pena de invalidar la firma criptográfica.

Como se muestra en la Figura 4, este proceso comprende seis etapas: canonicalización del XML, generación del digest, construcción de las SignedProperties, ensamblaje del SignedInfo, firma RSA y ensamblaje final del bloque de firma dentro del documento.

Figura 4

Diagrama de Bloques del Proceso de Firmado Digital Bajo el Estándar XAdES-BES



Una vez firmado el documento XML, la transmisión de la información hacia la plataforma del SRI se ejecuta mediante el consumo de servicios web basados en el Protocolo de Acceso a Objetos Simples (SOAP). SOAP es un protocolo estándar de mensajería ligera que utiliza XML como formato de codificación para el intercambio de información estructurada en entornos descentralizados [23]. La interacción con el SRI es síncrona-asíncrona y requiere la invocación secuencial de endpoints específicos expuestos por la autoridad tributaria. Como se muestra en la Tabla *4*, estos servicios cumplen funciones diferenciadas dentro del ciclo de recepción y autorización de comprobantes.

Tabla 4

Servicios Web SOAP Oficiales del SRI

Servicio SOAP	Función
RecepcionComprobantesOffline	Recibe el XML firmado codificado en Base64. Responde RECIBIDA (aceptado para

	procesamiento) o DEVUELTA (errores estructurales o de validación).
AutorizacionComprobantesOffline	Consulta el estado de autorización por clave de acceso. Responde AUTORIZADO (con número de autorización), EN PROCESAMIENTO o NO AUTORIZADO.

Dado que el servicio de autorización gubernamental opera de forma asíncrona, los sistemas de integración deben estar preparados para manejar esperas e interrupciones sin perder la consistencia de la operación. Cuando el servidor devuelve un estado como EN PROCESAMIENTO, no basta con ignorar la respuesta o reintentar de inmediato: las buenas prácticas de ingeniería de software indican que se deben implementar algoritmos de reintento automatizado con tiempos de espera progresivos ya sea de forma lineal o mediante retroceso exponencial (backoff) de modo que el sistema aguarde con inteligencia antes de volver a consultar, evitando así saturar los servidores del Estado y garantizando que la transacción fiscal quede debidamente registrada y autorizada.

Automatización y Procesamiento Asíncrono en Segundo Plano

En plataformas SaaS de alta demanda, las tareas de procesamiento intensivo o periódico no deben ejecutarse dentro del ciclo de petición del usuario, ya que esto afecta directamente la experiencia en la interfaz. Para resolverlo, se recurre a tareas en segundo plano (background tasks) gestionadas mediante planificadores que interpretan expresiones cron el mismo estándar de los sistemas Unix y que en entornos Node.js se implementan fácilmente con librerías como node-cron [30]. Este enfoque permite automatizar procesos recurrentes como la emisión periódica de comprobantes o la conciliación masiva de datos fiscales, ejecutándose en intervalos calendarizados, sin intervención humana y sin afectar el rendimiento de la aplicación.

Cuando los flujos de automatización implican el envío de archivos a través de HTTP, es necesario manejar peticiones codificadas bajo el estándar multipart/form-data. Para ello, middlewares como Multer interceptan el flujo binario entrante y se encargan de asignarlo en memoria o escribirlo de forma controlada en disco [31]. En arquitecturas cloud, los archivos sensibles como los certificados .p12 se convierten a cadenas Base64, lo que transforma su contenido binario en texto legible por los motores relacionales. Esto permite almacenarlos

directamente en la base de datos de forma cifrada, eliminando la dependencia del almacenamiento local del servidor y facilitando la escalabilidad horizontal de los servicios.

Para cerrar el ciclo transaccional, el sistema genera automáticamente documentos PDF a partir de las estructuras XML autorizadas, utilizando motores como PDFKit que construyen el archivo directamente en memoria, sin necesidad de instanciar navegadores virtuales y con un menor costo computacional [32]. Una vez generado el documento, su distribución al cliente final se realiza de forma asíncrona mediante agentes de correo como Nodemailer, que operan sobre SMTP con cifrado TLS/SSL y permiten adjuntar de forma simultánea tanto el XML como el PDF dentro de plantillas de correo dinámicas en HTML.

Optimización de Rendimiento y Capas de Caché

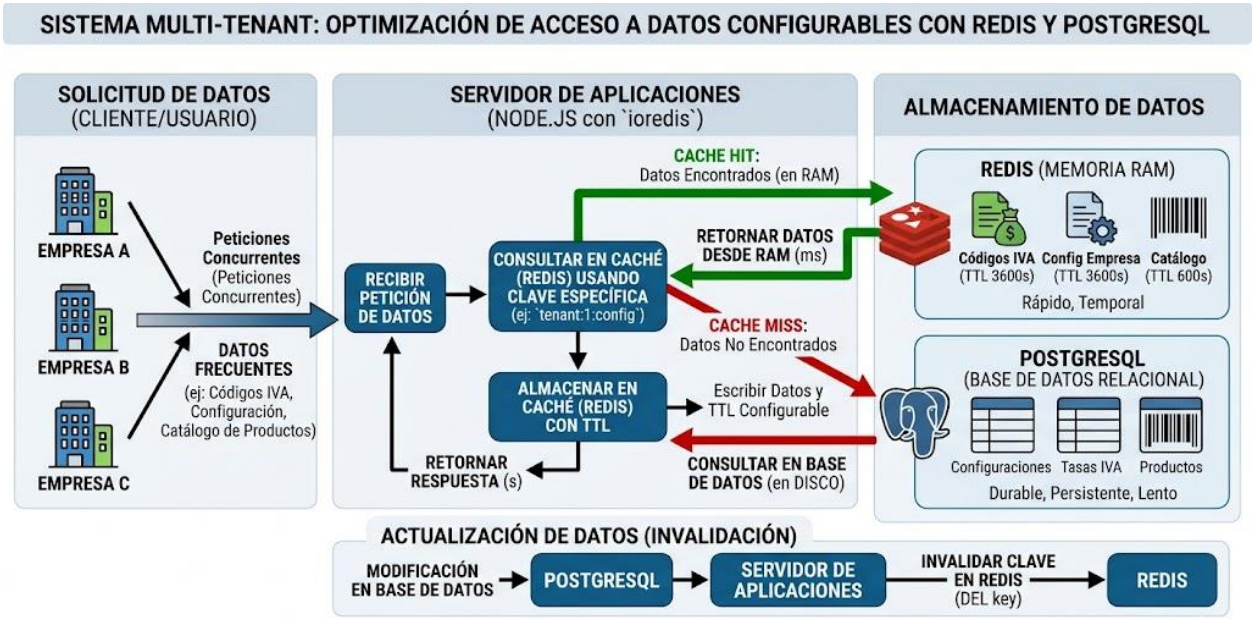
En entornos transaccionales compartidos por múltiples tenants concurrentes, el acceso repetitivo a bases de datos relacionales basadas en almacenamiento en disco genera cuellos de botella físicos e incrementa la latencia de respuesta del sistema. Para mitigar este impacto, la ingeniería de datos promueve la incorporación de capas de caché intermedias basadas en almacenes de estructuras de datos en memoria NoSQL de tipo clave-valor de alto rendimiento, como Redis [24]. Al mantener los datos de configuración recurrentes, tokens de sesión activos y catálogos estáticos directamente en la memoria volátil RAM, se reducen los tiempos de lectura a escalas de submilisegundos.

El acoplamiento técnico entre la base de datos relacional persistente y la memoria en caché se rige por la aplicación del patrón estructural Cache-Aside (o carga diferida) [25]. Bajo esta lógica de diseño, cuando la aplicación backend requiere un registro, consulta en primera instancia a la capa de caché en memoria; si el dato se encuentra presente, ocurre un acierto de caché (Cache Hit) y la información se retorna de inmediato. En caso contrario, ante una ausencia de caché (Cache Miss), el sistema ejecuta la consulta SQL pesada sobre el motor relacional persistente, almacena una copia del resultado en el motor en memoria con un tiempo de vida definido (TTL, Time-To-Live) para garantizar su posterior frescura, y finalmente entrega la respuesta al cliente.

Como se muestra en la Figura 5, este patrón se aplica en un entorno multi-tenant, donde Redis actúa como capa de caché de acceso rápido y PostgreSQL como almacenamiento persistente, incluyendo además el mecanismo de invalidación de claves ante actualizaciones en la base de datos.

Figura 5

Diagrama de Flujo Operativo del Patrón de Diseño Cache-Aside



Auditoría de Seguridad Automatizada y Análisis Dinámico de Aplicaciones

La fase final en el diseño e implementación segura de una plataforma transaccional distribuida consiste en la verificación y validación proactiva de los controles lógicos integrados en el código fuente. El Análisis Dinámico de Seguridad (DAST, por sus siglas en inglés) representa una metodología de pruebas en la cual se evalúa una aplicación web en tiempo de ejecución, simulando vectores de ataque externos sin poseer conocimiento previo de la estructura interna del software o del sistema gestor de bases de datos [26]. Para la ejecución sistemática de este procedimiento bajo estándares globales, la industria de la ciberseguridad adopta OWASP Zed Attack Proxy (ZAP), una infraestructura de código abierto orientada a la detección automatizada de vulnerabilidades en aplicaciones web y arquitecturas de APIs RESTful.

Al operar como un proxy inverso interceptor entre la capa de presentación (Frontend) y el servidor de aplicaciones (Backend), OWASP ZAP ejecuta escaneos pasivos y ataques activos controlados [27]. Este proceso permite identificar fallos de seguridad críticos categorizados dentro del índice de referencia OWASP Top 10, tales como configuraciones incorrectas del servidor, directrices CORS mal estructuradas o debilidades en la gestión de sesiones concurrentes, generando reportes

de auditoría técnicos indispensables para el endurecimiento definitivo (hardening) de la plataforma SaaS antes de su transición hacia entornos de producción. Como se muestra en la Tabla *5*, este análisis dinámico se ejecuta a través de seis fases operativas, cada una con un componente lógico y un objetivo de seguridad específico.

Tabla 5

Fases Operativas y Componentes Lógicos del Análisis Dinámico de Ciberseguridad con OWASP ZAP

Fase y Componente	Descripción Técnica	Objetivo de Seguridad
1. Reconocimiento	Intercepción y captura del tráfico HTTP/HTTPS entre el cliente y el servidor backend.	Establecer la línea base del comportamiento transaccional legítimo.
2. Exploración	Mapeo automatizado y recursivo de las rutas, formularios y endpoints de la API.	Determinar la superficie de ataque expuesta en la plataforma.
3. Análisis Pasivo	Inspección de las peticiones y respuestas en tránsito sin modificar los datos originales.	Detectar de forma preliminar fallos de configuración y ausencia de cabeceras.
4. Análisis Activo	Inyección controlada de vectores de ataque (*payloads*) dirigidos a las funciones del backend.	Evaluar la vulnerabilidad del sistema ante los riesgos del *OWASP Top 10*.
5. Clasificación	Catalogación automatizada de las anomalías descubiertas bajo el estándar métrico CVSS.	Clasificar y priorizar los riesgos según su severidad (Alta, Media, Baja).
6. Reporte	Consolidación de las evidencias y hallazgos en un documento técnico estructurado.	Proveer la guía necesaria para la remediación de vulnerabilidades (*hardening*).

Metodología de Desarrollo de Software

El desarrollo de software requiere metodologías capaces de adaptarse a cambios en los requisitos y de proporcionar entregas continuas de valor. Por ello, los enfoques ágiles han ganado

relevancia frente a los modelos tradicionales, al favorecer la flexibilidad, la colaboración y la mejora continua durante el ciclo de desarrollo. Scrum es un marco de trabajo ágil ampliamente utilizado en el desarrollo de software por su enfoque iterativo e incremental, el cual facilita la gestión de proyectos complejos y la entrega continua de valor.

Su aplicación se fundamenta en tres pilares: la transparencia, que garantiza la visibilidad del proceso; la inspección, que permite revisar periódicamente el progreso; y la adaptación, que posibilita realizar ajustes oportunos en función de los resultados obtenidos [28]. Estos principios convierten a Scrum en un marco especialmente pertinente para proyectos de desarrollo de software en los que los requisitos evolucionan y se requiere entregar valor de forma progresiva.

Roles del equipo Scrum

Scrum define tres roles principales dentro del equipo de desarrollo:

- **Product Owner:** Es responsable de maximizar el valor del producto resultante del trabajo del equipo. Gestiona y prioriza el Product Backlog, definiendo qué funcionalidades deben desarrollarse y en qué orden, según el valor que aportan al negocio o usuario final.
- **Scrum Master:** Actúa como facilitador del proceso. Se encarga de que el equipo comprenda y aplique correctamente los principios y eventos de Scrum, removiendo obstáculos que impidan el avance del trabajo.
- **Development Team:** Es el grupo de profesionales que ejecuta el trabajo técnico necesario para convertir los elementos del Backlog en incrementos de software funcionales.

Artefactos de Scrum

Scrum se apoya en tres artefactos principales que dan trazabilidad al trabajo:

- **Product Backlog:** Es una lista ordenada y dinámica de todo lo que se conoce que es necesario en el producto. Constituye la única fuente de requisitos del proyecto y está en constante evolución.
- **Sprint Backlog:** Es el subconjunto de elementos del Product Backlog seleccionados para ser desarrollados durante un Sprint específico, junto con el plan para entregarlos.
- **Incremento:** Es la suma de todos los elementos del Product Backlog completados durante un Sprint, integrados con los incrementos de Sprints anteriores. Debe

encontrarse en un estado utilizable, independientemente de si el Product Owner decide publicarlo de inmediato.

Fases y eventos del ciclo Scrum

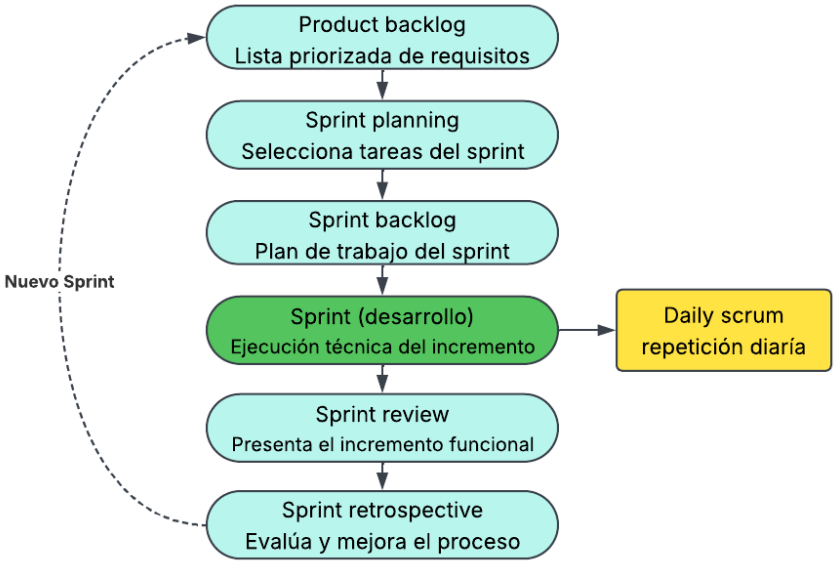
El trabajo en Scrum se organiza en bloques de tiempo fijos llamados Sprints, con una duración típica de dos a cuatro semanas, dentro de los cuales se desarrollan los siguientes eventos de forma secuencial:

- **Sprint Planning:** Se definen el objetivo del Sprint, las tareas a desarrollar y los elementos seleccionados del Product Backlog que conformarán el Sprint Backlog.
- **Sprint Backlog**: Conjunto de actividades planificadas que guía el trabajo diario del equipo durante la ejecución del Sprint.
- **Daily Scrum:** Reunión diaria breve destinada a revisar el progreso, identificar impedimentos y ajustar el plan de trabajo.
- **Sprint:** Período de desarrollo en el que el equipo implementa las funcionalidades comprometidas para generar un incremento del producto.
- **Sprint Review:** Revisión del incremento desarrollado y recopilación de retroalimentación para orientar los siguientes pasos del proyecto.
- **Sprint Retrospective:** Evaluación interna del Sprint para identificar oportunidades de mejora en procesos, herramientas y colaboración del equipo.

Como se muestra en la Figura 6, estas fases se ejecutan de forma cíclica e iterativa, repitiéndose Sprint tras Sprint hasta alcanzar el objetivo final del producto.

Figura 6

Diagrama de Ciclo de la Metodología SCRUM



Enfoque de Investigación: Design Science Research (DSR)

Aunque Scrum organizó el proceso de desarrollo del software, fue necesario adoptar un enfoque de investigación que sustentara la creación de la plataforma como una contribución académica. En este sentido, se empleó DSR, metodología orientada al diseño, construcción y evaluación de artefactos tecnológicos destinados a resolver problemas reales [29]. Este enfoque resulta adecuado para el presente trabajo, ya que permite estructurar el proceso desde la identificación de la necesidad de una solución de facturación electrónica multi-tenant para PYMES ecuatorianas hasta la validación de un sistema funcional y evaluado.

El proceso DSR comprende seis fases principales:

- Identificación del problema y motivación: definición de la necesidad y justificación de la solución propuesta.
- Definición de objetivos: establecimiento de los resultados que debe alcanzar el artefacto.

- Diseño y desarrollo: construcción de la arquitectura y funcionalidades del sistema.
- Demostración: aplicación del artefacto en escenarios representativos del problema.
- Evaluación: verificación del cumplimiento de los objetivos mediante pruebas y métricas.
- Comunicación: difusión de los resultados y aportes obtenidos a través del presente documento de investigación.

Mapeo de las fases DSR al desarrollo del proyecto

Dado que el artefacto de este proyecto ya fue desarrollado siguiendo la metodología ágil Scrum, las fases de DSR no representan un proceso adicional o paralelo, sino un marco interpretativo que organiza y justifica retrospectivamente el trabajo ya documentado en los siguientes capítulos, como se muestra en la Tabla 6.

Tabla 6

Mapeo de Fases DSR a la Estructura del Proyecto

Fase DSR	Correspondencia en el proyecto
Identificación del problema	Planteamiento del problema y justificación (Capítulo 1)
Definición de objetivos	Objetivo general y objetivos específicos (Capítulo 1)
Diseño y desarrollo	Arquitectura del sistema, desarrollo iterativo mediante Scrum (Sprints 1-6), implementación de módulos funcionales (Capítulos 2-4)
Demostración	Ejecución de casos de uso funcionales del sistema desplegado (Capítulo 4)
Evaluación	Pruebas de seguridad con OWASP ZAP, verificación de cumplimiento de Requerimientos No Funcionales (Capítulo 4)
Comunicación	El presente documento de tesis

CAPÍTULO III

Metodología

Para gobernar el desarrollo de la plataforma SaaS multi-tenant, se implementa el marco ágil Scrum, seleccionado sobre modelos predictivos tradicionales por su capacidad de gestionar entornos complejos y cambiantes, como el ecosistema fiscal ecuatoriano y las arquitecturas en la nube. Su enfoque iterativo mediante Sprints permite validar incrementalmente los componentes críticos del sistema aislamiento transaccional, criptografía de firmas y servicios SOAP del SRI, reduciendo el riesgo de fallas en producción y facilitando la adaptación ante cambios normativos. En las siguientes secciones se detalla el proceso de la metodología Scrum aplicado al desarrollo del presente proyecto.

Definición de Roles del Equipo

La estructura del proyecto sigue los roles oficiales de Scrum, distribuidos según las competencias de cada participante:

- **Product Owner (Ing. Daniel Núñez):** Responsable de definir y priorizar el Product Backlog, garantizando que los flujos transaccionales cumplan con los estándares de validación del SRI.
- **Scrum Master (Nicolás Lara):** Facilita la correcta aplicación de las ceremonias Scrum, remueve impedimentos y optimiza el flujo de trabajo del equipo.
- **Development Team (Nicolás Lara):** Como desarrollador Full-Stack, es responsable de la implementación técnica completa: persistencia relacional en PostgreSQL con Sequelize ORM, interfaces reactivas en Next.js, endpoints en Node.js/Express, caché con Redis y auditorías de seguridad bajo OWASP Top 10.

A continuación, se presenta la Tabla *7* que resume la distribución de roles y responsabilidades dentro del equipo de desarrollo:

Tabla 7

Estructura y Responsabilidades de los Roles SCRUM

Rol	Responsable	Responsabilidades
Product Owner	Ing. Daniel Núñez	Priorización del Product Backlog, validación de requerimientos de negocio y aprobación del cumplimiento normativo con el SRI.

Scrum Master	Nicolás Lara	Garantizar el cumplimiento del marco ágil, eliminar impedimentos del equipo y facilitar los eventos de cada Sprint.
Development Team	Nicolás Lara	Diseño y desarrollo Full-Stack del sistema, incluyendo base de datos (PostgreSQL/Sequelize), interfaces (Next.js), API (Node.js/Express), caché (Redis), firma digital XAdES-BES y pruebas de seguridad (OWASP ZAP).

Product Backlog del Sistema

El Product Backlog representa la fuente principal de requisitos del sistema, estructurado como una lista dinámica de necesidades descompuestas en Historias de Usuario (US) y priorizadas según su valor técnico, dependencias arquitectónicas y criticidad fiscal ante el SRI.

Este inventario abarca tanto requisitos funcionales como no funcionales, permitiendo vincular cada necesidad del inquilino con los componentes de backend y frontend del sistema.

En la Tabla *8* se detallan las Historias de Usuario definidas para el desarrollo de la plataforma:

Tabla 8

Inventario del Product Backlog General del Sistema

ID	Historia de Usuario	Prioridad	Módulo
US-01	Como usuario, quiero autenticarme con credenciales seguras y JWT	Alta	Autenticación
US-02	Como administrador, quiero gestionar usuarios y roles del sistema	Alta	Usuarios / Roles
US-03	Como administrador, quiero registrar y configurar los datos de la empresa (RUC, razón, social, régimen tributario, ambiente SRI)	Alta	Empresa
US-04	Como administrador, quiero cargar y gestionar la firma electrónica (.p12) para la emisión de comprobantes	Alta	Firma Electronica

ID	Historia de Usuario	Prioridad	Módulo
US-05	Como administrador, quiero configurar establecimientos y puntos de emisión con sus secuenciales	Alta	Establecimientos / puntos de emisión
US-06	Como administrador, quiero configurar los códigos de IVA aplicables a los documentos	Alta	Códigos IVA
US-07	Como usuario, quiero gestionar el catálogo de clientes y proveedores	Alta	Clientes / Proveedores
US-08	Como usuario, quiero gestionar el catálogo de productos con grupos e impuestos	Alta	Productos
US-09	Como usuario, quiero emitir facturas electrónicas con firma XAdES y autorización SRI	Alta	Facturas
US-10	Como usuario, quiero emitir notas de crédito electrónicas	Alta	Notas de crédito
US-11	Como usuario, quiero emitir notas de débito electrónicas	Alta	Notas de débito
US-12	Como usuario, quiero emitir guías de remisión electrónicas	Media	Guías de Remisión
US-13	Como usuario, quiero emitir liquidaciones de compra electrónicas	Media	Liquidaciones de Compra
US-14	Como usuario, quiero emitir comprobantes de retención electrónicos	Media	Retenciones
US-15	Como usuario, quiero emitir notas de venta (RISE) electrónicas	Media	Notas de Venta
US-16	Como usuario, quiero emitir proformas y generar su PDF	Media	Proformas
US-17	Como usuario, quiero generar y enviar por correo el PDF de cada comprobante autorizado	Media	Reportes / PDF
US-18	Como usuario, quiero configurar y automatizar la facturación recurrente	Baja	Recurrentes

ID	Historia de Usuario	Prioridad	Módulo
US-19	Como administrador, quiero visualizar el dashboard con KPIs y estadísticas de facturación	Baja	Dashboard
US-20	Como desarrollador, quiero implementar Sequelize ORM para gestionar las interacciones con la base de datos de forma segura, estructurada y mantenible	Alta	ORM / Base de Datos
US-21	Como administrador del sistema, quiero que el sistema cumpla con las medidas de seguridad OWASP Top 10 para proteger los datos sensibles de la empresa y sus comprobantes electrónicos	Alta	Seguridad

Requerimientos Funcionales (RF)

Los Requerimientos Funcionales determinan rigurosamente las operaciones conductuales, algoritmos de cálculo, transformaciones estructurales de datos y flujos lógicos que la plataforma debe ejecutar de manera mandataria ante estímulos directos del emisor o del ecosistema transaccional interconectado. Aislados formalmente en una matriz de especificación técnica, estos requerimientos modelan las reglas de negocio impositivas dispuestas por la ficha técnica del SRI, incorporando procesos críticos como la inyección del nodo criptográfico XAdES-BES mediante certificados PKCS#12 y la interoperabilidad asíncrona mediante sockets basados en el protocolo SOAP.

Como se muestra en la Tabla *9*, estos requerimientos se encuentran organizados por módulo funcional, prioridad de implementación e identificador único de trazabilidad.

Tabla 9

Especificación de Requerimientos Funcionales

ID	Requisito Funcional	Prioridad	Módulo
RF-01	Autenticación mediante correo, contraseña y JWT	Alta	Autenticación
RF-02	Recuperación y restablecimiento de contraseña por correo	Alta	Autenticación

ID	Requisito Funcional	Prioridad	Módulo
RF-03	Gestión de usuarios	Alta	Usuarios
RF-04	Gestión y asignación de roles	Alta	Roles
RF-05	Configuración de datos de la empresa y ambiente SRI	Alta	Empresa
RF-06	Carga y validación de firma electrónica (.p12)	Alta	Firma Electrónica
RF-07	Activar y desactivar firma electrónica	Alta	Firma Electrónica
RF-08	Gestión de establecimientos	Alta	Establecimientos
RF-09	Configuración de puntos de emisión y secuenciales	Alta	Puntos de Emisión
RF-10	Configuración de códigos IVA	Alta	Códigos IVA
RF-11	Gestión de clientes	Alta	Clientes
RF-12	Gestión de proveedores	Alta	Proveedores
RF-13	Gestión de productos e impuestos	Alta	Productos
RF-14	Emisión de facturas electrónicas y autorización SRI	Alta	Facturas
RF-15	Emisión de notas de crédito electrónicas	Alta	Notas de Crédito
RF-16	Emisión de notas de débito electrónicas	Alta	Notas de Débito
RF-17	Emisión de guías de remisión electrónicas	Media	Guías de Remisión
RF-18	Emisión de liquidaciones de compra electrónicas	Media	Liquidaciones
RF-19	Emisión de comprobantes de retención electrónicos	Media	Retenciones
RF-20	Emisión de notas de venta RISE	Media	Notas de Venta

ID	Requisito Funcional	Prioridad	Módulo
RF-21	Gestión de proformas con PDF	Media	Proformas
RF-22	Generación de PDF con código de barras	Media	Reportes
RF-23	Envío de comprobantes por correo electrónico	Media	Correo
RF-24	Facturación recurrente automatizada	Baja	Recurrentes
RF-25	Dashboard con KPIs de facturación	Baja	Dashboard

Requerimientos No Funcionales (RNF) y Atributos de Calidad

Los Requerimientos No Funcionales establecen las restricciones técnicas, estándares de seguridad y criterios de desempeño que determinan la robustez del sistema. Actúan como marco de control de calidad para las tecnologías del stack (TypeScript, PostgreSQL, Redis, OWASP Top 10), garantizando de forma medible la mitigación de vulnerabilidades, la integridad de los esquemas XML y una latencia de respuesta en la API dentro de los umbrales aceptables para operaciones comerciales distribuidas. Como se muestra en la Tabla *10*, estos requerimientos se encuentran clasificados por categoría de calidad e indicador medible de cumplimiento.

Tabla 10

Matriz de Requerimientos No Funcionales y Atributos de Calidad

ID	Requisito No Funcional	Categoría	Indicador
RNF-01	Cifrado de contraseñas con bcrypt	Seguridad	bcrypt ≥ 10 rounds
RNF-02	Autenticación con tokens JWT	Seguridad	Expiración configurable
RNF-03	Protección de rutas con autenticación y roles	Seguridad	Acceso denegado sin token
RNF-04	Implementación de medidas OWASP Top 10	Seguridad	OWASP aplicado

ID	Requisito No Funcional	Categoría	Indicador
RNF-05	Almacenamiento JWT en cookies httpOnly	Seguridad	Cookie httpOnly habilitada
RNF-06	Configuración de cabeceras HTTP seguras	Seguridad	Helmet.js configurado
RNF-07	Tiempo máximo de respuesta de la API ≤ 3s	Rendimiento	Respuesta ≤ 3 segundos
RNF-08	Implementación de caché con Redis	Rendimiento	Redis activo
RNF-09	Disponibilidad del sistema del 99%	Disponibilidad	Uptime ≥ 99%
RNF-10	Interfaz responsiva para escritorio y tablet	Usabilidad	Compatible desde 768px
RNF-11	Arquitectura multi-tenant por empresa	Escalabilidad	Aislamiento por empresa
RNF-12	Uso de Sequelize ORM y consultas seguras	Mantenibilidad	Sequelize implementado
RNF-13	Registro de logs de comunicación con SRI	Trazabilidad	Logs almacenados
RNF-14	Generación XML conforme al esquema SRI	Cumplimiento	XML válido SRI
RNF-15	Firma digital con estándar XAdES	Cumplimiento	Firma XAdES válida

Diseño de la Arquitectura del Sistema

Para dar solución a las problemáticas identificadas en la línea base, se diseñó una arquitectura de software moderna, desacoplada y orientada a servicios en la nube. El pilar fundamental de la propuesta radica en la eficiencia del paradigma multi-inquilino (Multitenancy), el cual permite optimizar el uso de recursos computacionales compartidos, garantizando la alta disponibilidad, la seguridad en el aislamiento de datos y una latencia mínima en las peticiones concurrentes.

Topología del Sistema y Despliegue en la Nube

El sistema se construyó sobre una arquitectura distribuida, alojada en un servidor privado virtual (VPS) con Linux (Ubuntu Server). Esto permite reducir costos de licencias, aprovechar mejor el hardware y tener control total sobre la seguridad y la red.

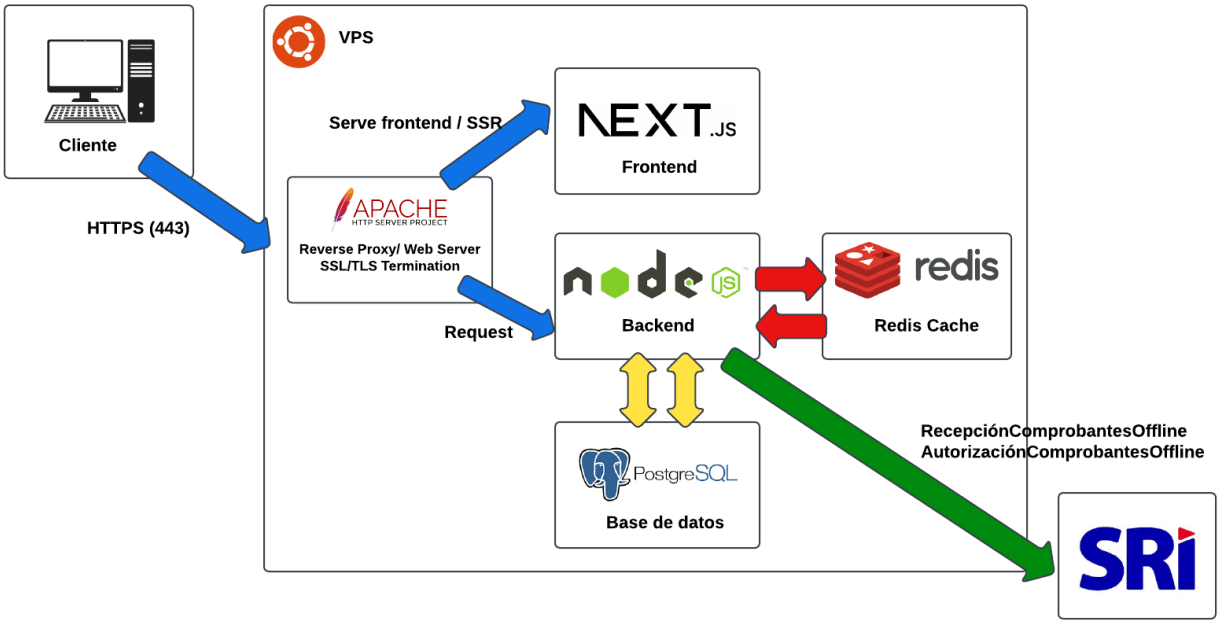
La estructura organiza las funciones del sistema en capas independientes que se comunican de forma segura mediante HTTPS, lo que facilita el mantenimiento y sostiene el modelo SaaS. A continuación, se describe el rol de cada componente:

- **Frontend (Next.js):** Es la interfaz visual con la que interactúa el usuario. Diseñada para adaptarse a cualquier dispositivo, se comunica con el backend mediante peticiones asíncronas sin almacenar datos sensibles ni lógica de negocio.
- **Backend (Node.js + Express + TypeScript):** Es el núcleo del sistema. Gestiona los flujos financieros, valida la seguridad, controla el acceso por empresa mediante tokens JWT y se comunica con los servicios del SRI.
- **Redis (Caché):** Almacena temporalmente en memoria datos que se consultan frecuentemente y cambian poco, como tarifas de IVA o configuraciones del SRI, reduciendo los tiempos de respuesta y evitando consultas repetitivas a la base de datos.
- **PostgreSQL (Base de datos):** Es el motor principal de almacenamiento. Garantiza la integridad de los datos, maneja múltiples empresas de forma simultánea y está optimizado para soportar una alta carga de transacciones concurrentes.

En la Figura 7 se muestra esta arquitectura, que aloja de forma integrada el frontend en Next.js, el backend en Node.js, la caché en Redis y la base de datos en PostgreSQL, además de gestionar la comunicación saliente hacia los servicios SOAP del SRI para la recepción y autorización de comprobantes.

Figura 7

Diagrama de Arquitectura Distribuida y Despliegue en VPS



Modelo de Aislamiento Lógico de Datos (*Multitenancy Logic*)

Uno de los retos más importantes en una plataforma SaaS es garantizar que la información de cada empresa esté completamente separada de las demás. Para este proyecto se optó por una base de datos compartida, donde todos los clientes coexisten en la misma estructura, pero sus datos están aislados de forma lógica. Esto reduce costos y simplifica el mantenimiento.

Para evitar cualquier fuga de información entre empresas, se aplicaron tres medidas clave:

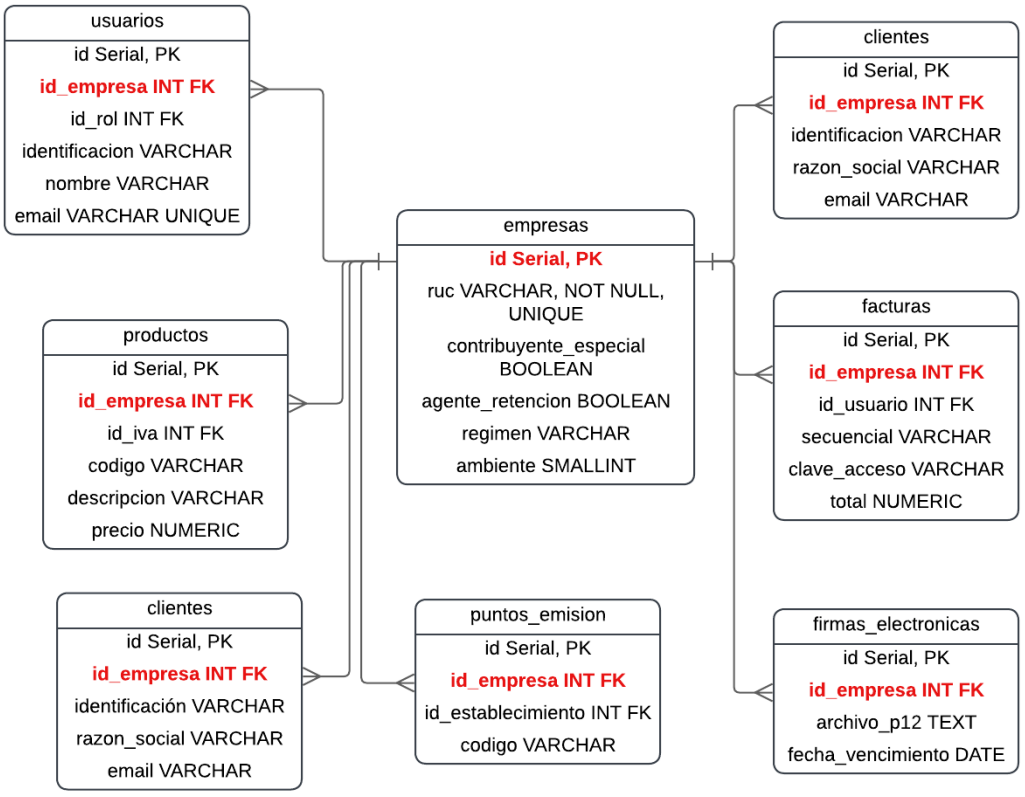
- **Identificador por empresa (empresa_id):** Todas las tablas con datos sensibles o transaccionales incluyen una columna que identifica a qué empresa pertenece cada registro, asegurando una separación clara desde la base de datos.
- **Filtrado automático mediante ORM:** A través de Sequelize, cada petición al servidor pasa por un middleware que extrae el identificador de la empresa desde el token JWT del usuario y lo inyecta automáticamente en cada consulta, de modo que cada usuario solo accede a sus propios datos.

- **Protección ante manipulación:** Aunque un usuario intente alterar los parámetros de una petición, el backend ignora esos cambios y restringe la búsqueda únicamente a los registros de su empresa, manteniendo el entorno de cada cliente completamente aislado.

En la Figura 8 se muestra el diagrama entidad-relación simplificado con este enfoque multitenant, donde la tabla empresas actúa como nodo central y cada tabla relacionada incorpora la columna id_empresa como llave foránea de aislamiento.

Figura 8

Diagrama Entidad-Relación con Enfoque Multitenant (Simplificado)



Lógica del Negocio Core y Ciclo de Vida Tributario

Para garantizar la integridad de los datos, el cumplimiento normativo del SRI y el aislamiento entre empresas, el módulo de facturación se diseñó bajo el modelo de una Máquina de Estados Finitos (FSM). Esta arquitectura asegura que cada comprobante siga un flujo ordenado y controlado, donde cada cambio de estado es predecible, auditable e irreversible una vez finalizado.

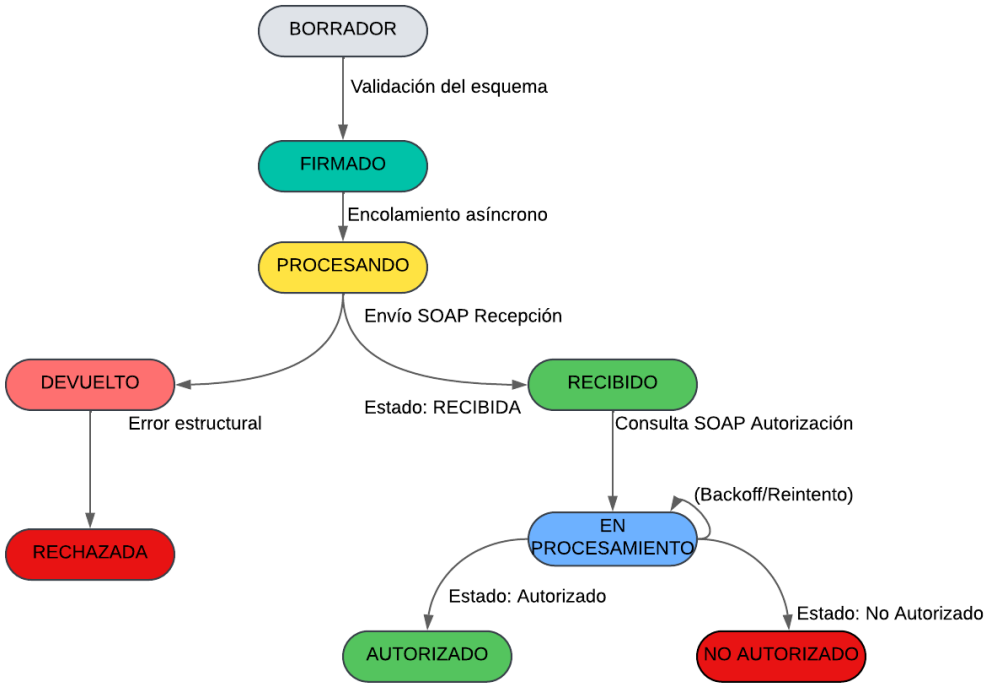
Modelo de Estados del Comprobante Electrónico

El ciclo de vida del comprobante se gobierna localmente en el motor de la API (Node.js/Express) mediante mutaciones persistidas en la base de datos relacional (PostgreSQL) a través del ORM Sequelize.

A continuación, en la Figura 9 se presenta la abstracción visual del comportamiento dinámico del sistema:

Figura 9

Diagrama de Máquina de Estados del Ciclo de Vida del Comprobante Electrónico



Matriz de Transición de Estados del Ciclo de Vida Tributario

Como se detalla en la Tabla 11 y la Figura 9, se implementó una máquina de estados finitos que gobierna el ciclo de vida de cada comprobante electrónico, desde su creación hasta su resolución final ante el SRI, garantizando la consistencia e integridad de los documentos en el entorno multitenant.

Tabla 11

Matriz de Transición de Estados del Ciclo de Vida del Comprobante Electrónico

Estado Inicial	Estado Destino	Evento / Desencadenante	Impacto en el Sistema e Infraestructura
-	BORRADOR	El usuario llena el formulario y guarda el documento en el sistema.	El registro se almacena en PostgreSQL en estado inicial, sin XML generado.
BORRADOR	FIRMADO	Se ejecuta el componente KeyAccessProvider y el firmado criptográfico XAdES-BES.	Se genera la clave de acceso de 49 dígitos, se construye el XML y se firma digitalmente con el certificado `.p12`.
FIRMADO	RECIBIDO	Respuesta exitosa del servicio web de recepción del SRI (validarComprobante).	El SRI acepta el documento y se registra la comunicación en `log_sri`.
FIRMADO	DEVUELTO	El SRI devuelve errores de estructura o validación en la recepción del documento.	El documento es rechazado y queda disponible para su corrección y reenvío.
DEVUELTO	RECHAZADA	El usuario interactúa con la interfaz web, corrige los campos erróneos y guarda los cambios.	Se limpian las trazas de firma previas y el registro queda listo para iniciar un nuevo proceso de compilación XML.
RECIBIDO	EN_PROCESAMIENTO	El servicio de autorización del SRI devuelve el estado **EN PROCESAMIENTO**.	Se registra el intento en `log_sri` y el sistema programa un reintento automático con backoff exponencial.

EN_PROCESAMIENTO	EN_PROCESAMIENTO	El scheduler consulta nuevamente al SRI y este responde que el documento sigue en procesamiento.	Se incrementa el contador de reintentos en Redis y, si supera el límite establecido, se genera una alerta para revisión manual.
RECIBIDO / EN_PROCESAMIENTO	AUTORIZADO	Respuesta definitiva del servicio web del SRI con etiqueta <estado>AUTORIZADO.	El ciclo finaliza exitosamente: se extrae el XML autorizado, se genera el RIDE en PDF, se almacena de forma segura y se envía por correo SMTP.
RECIBIDO / EN_PROCESAMIENTO	NO_AUTORIZADO	Respuesta definitiva del servicio web del SRI con etiqueta <estado>NO AUTORIZADO.	El SRI rechaza el comprobante de forma permanente, se almacenan los motivos en la base de datos y se notifica al emisor.

Generación e Implementación de la Clave de Acceso de 49 Dígitos

La Clave de Acceso es el identificador único universal exigido por la legislación tributaria ecuatoriana para consolidar la validez de un comprobante electrónico. El sistema implementa un motor algorítmico en Node.js que construye dinámicamente esta cadena de 49 caracteres numéricos a partir de los datos transaccionales del *tenant*.

Estructura de la Clave de Acceso

La composición se divide estrictamente en 9 campos posicionales fijos:

Clave = Fecha (8) + Tipo (2) + RUC (13) + Ambiente (1) + Serie (6) + Secuencial (9) + CódigoNumérico (8) + TipoEmisión (1) + DV (1)

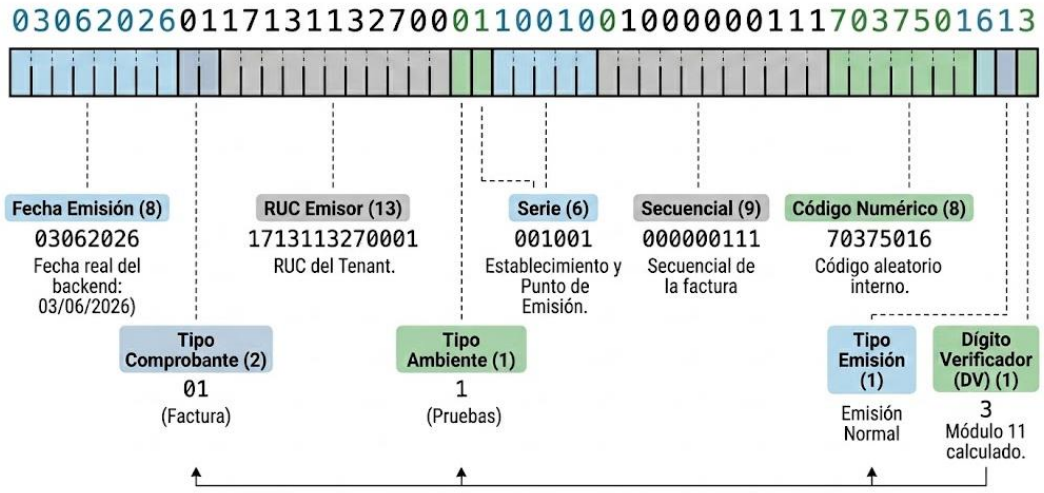
A modo de evidencia de ejecución real del backend, se presenta un caso de estudio real para una factura emitida en el entorno de desarrollo:

- **Ejemplo real generado:**
  0306202601171311327000110010010000001117037501613

En la Figura 10 se muestra la descomposición posicional de esta clave de 49 dígitos, evidenciando cómo cada segmento corresponde a un campo específico.

Figura 10

Descomposición Posicional y Campos de la Clave de Acceso Real de 49 Dígitos



Algoritmo Módulo 11 para el Dígito Verificador (DV)

El cuadragésimo noveno dígito se calcula aplicando el Algoritmo de la Modificación del Módulo 11, cuyo factor de ponderación es un ciclo aritmético invertido del 2 al 7. La lógica matemática implementada sigue las siguientes fases:

1. Multiplicación de cada uno de los 48 dígitos de la clave base por su factor multiplicador posicional de derecha a izquierda.
2. Sumatoria total de los productos obtenidos (S).
3. Extracción del residuo de la división para 11 (R = S (mod11)).
4. Cálculo del dígito base: DV = 11 - R.
5. Evaluación de excepciones normadas por el SRI: si DV = 11 → DV = 0: si DV = 10 → DV = 1.

A continuación, en la Figura 11 se presenta la implementación lógica de la función Generar Clave de Acceso, desarrollada en TypeScript dentro del backend del sistema.

Figura 11

Función para Generar Clave de Acceso

```
export function generarClaveAcceso(
  fechaEmision: string,       // YYYY-MM-DD
  ruc: string,
  ambiente: number,
  codEstablecimiento: string,
  codPuntoEmision: string,
  secuencial: string          // 9 dígitos
): string {
  const [yyyy, mm, dd] = fechaEmision.split('-');
  const fecha = `${dd}${mm}${yyyy}`;
  const codigoNumerico = String(Math.floor(Math.random() * 100000000)).padStart(8, '0');
  const cuerpo = `${fecha}01${ruc}${ambiente}${codEstablecimiento}${codPuntoEmision}${secuencial}${codigoNumerico}1`;
  return `${cuerpo}${modulo11(cuerpo)}`;
}
```

Ciclo de Vida del Desarrollo e Implementación Tecnológica (Sprints)

El ciclo de vida del desarrollo del artefacto tecnológico se estructuró bajo un cronograma riguroso regido por las fases iterativas e incrementales del marco ágil Scrum. La ejecución material de la plataforma requirió un horizonte temporal absoluto de 18 semanas, equivalentes a 90 días de desarrollo, distribuidos simétricamente en seis (6) Sprints con una duración fija de tres (3) semanas por cada ciclo operativo.

El desarrollo completo de los 25 requerimientos funcionales del sistema SaaS multi-tenant, incluyendo diseño, codificación, integración de seguridad y pruebas End-to-End, representa un esfuerzo total estimado de 661 horas de ingeniería. Esta carga se distribuyó estratégicamente entre las capas de persistencia, backend y frontend, asegurando un flujo continuo de entregables validados.

En la Tabla *12* se presenta la planificación temporal, la distribución del esfuerzo y el alcance definido para cada ciclo del proyecto:

Tabla 12

Planificación y Distribución de Sprints

Sprint	Fechas	Horas	Backend	Frontend

Sprint 1	01 abril – 21 abril	**95 h**	Diseño de BD PostgreSQL, entorno TypeScript y autenticación JWT.	Configuración Next.js y formularios de acceso de usuarios.
Sprint 2	22 abril – 12 mayo	**105 h**	Módulos empresariales e integración de firma digital XAdES-BES.	Formularios empresariales y carga segura de firma electrónica (.p12).
Sprint 3	13 mayo – 02 junio	**110 h**	Integración Sequelize, migraciones y caché con Redis.	Gestión responsiva de clientes, proveedores y productos.
Sprint 4	03 junio – 23 junio	**111 h**	Generación XML, integración SOAP SRI, PDF y notificaciones SMTP.	Emisión de facturas y panel de estados tributarios.
Sprint 5	24 junio – 14 julio	**137 h**	Comprobantes complementarios y seguridad OWASP Top 10.	Interfaces para emisión de comprobantes electrónicos.
Sprint 6	15 julio – 04 agosto	**103 h**	Cron jobs, facturación recurrente y endpoints analíticos.	Dashboard KPI, pruebas E2E y manuales del sistema.
	Total	**661 h**		

Sprint 1: Fundación del Ecosistema y Autenticación de Usuarios

- **Periodo:** 01 de abril – 21 de abril 2026.
- **Descripción:** Este primer ciclo se centró en el aprovisionamiento de la infraestructura local y en la nube, estableciendo TypeScript como sistema de tipado estático global. Se diseñó la base de datos relacional en PostgreSQL, se implementó la autenticación sin estado mediante JWT firmados en el backend, con contraseñas protegidas por el algoritmo bcrypt, y se desarrollaron en Next.js los formularios de acceso y recuperación de cuentas.

Como se muestra en la Tabla *13*, el Sprint 1 detalla la planificación y ejecución de las tareas correspondientes, incluyendo su estado de avance y la comparación entre las horas estimadas y las horas reales empleadas.

Tabla 13

Matriz de Planificación y Ejecución - Sprint 1

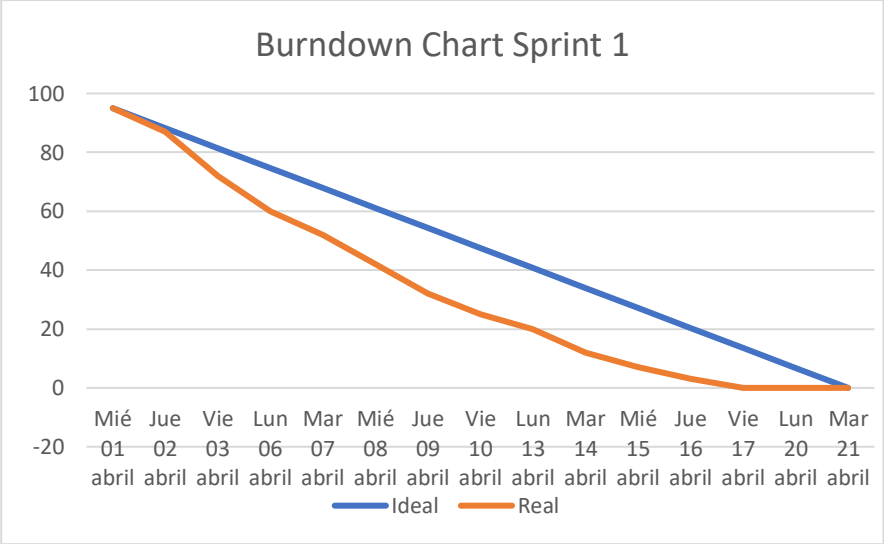
N°	Capa	Tarea	Historia	Estado	H. Estimadas	H. Reales
1	Backend	Configuración del entorno de desarrollo (Node.js, Next.js, TypeScript, PostgreSQL, Redis)	-	completado	8	8
2	Backend	Diseño y modelado de base de datos	-	completado	15	15
3	Backend	Módulo de autenticación con JWT (login, registro)	US-01	completado	12	12
4	Backend	Recuperación y restablecimiento de contraseñas	US-01	completado	8	8
5	Backend	CRUD de usuarios	US-02	completado	10	10
6	Backend	Gestión de roles y permisos	US-02	completado	10	10
7	Backend	Middleware de autenticación JWT	US-01	completado	7	7
8	Frontend	Configuración del proyecto Next.js (estructura, dependencias, rutas)	-	completado	5	5
9	Frontend	Página de login con validación de formulario	US-01	completado	8	8
10	Frontend	Página de registro de usuario	US-02	completado	5	5
11	Frontend	Página de recuperación de contraseña	US-01	completado	4	4
12	Frontend	Layout base y navegación principal	-	completado	3	3

TOTAL, SPRINT 1	95	95

La ejecución del Sprint 1 se desarrolló de manera exitosa. Como se observa en la Figura 12, se mantuvo un ritmo de trabajo superior al planificado durante las primeras semanas, lo que permitió adelantar avances y completar la totalidad de las 95 horas estimadas dentro del plazo establecido.

Figura 12

Curva de Avance Real vs. Ideal – Sprint 1



Sprint 2: Configuración Corporativa y Módulo de Firma Electrónica

- **Período:** 22 de abril – 12 de mayo 2026.
- **Descripción**: El segundo ciclo implementó el núcleo de parametrización del inquilino, incluyendo la gestión de empresas (RUC, regímenes tributarios, ambientes del SRI), sucursales, puntos de emisión y secuenciales. El hito principal fue el desarrollo del módulo criptográfico, encargado de procesar los certificados de firma electrónica PKCS#12 (.p12), almacenar las llaves bajo cifrado e inyectar la firma digital bajo el estándar XAdES-BES, garantizando la validez legal de los documentos emitidos.

Como se muestra en la Tabla *14*, esta matriz detalla la planificación y ejecución correspondiente al sprint, incluyendo el estado de avance y la comparación entre las horas estimadas y las horas reales empleadas.

Tabla 14

Matriz de Planificación y Ejecución - Sprint 2

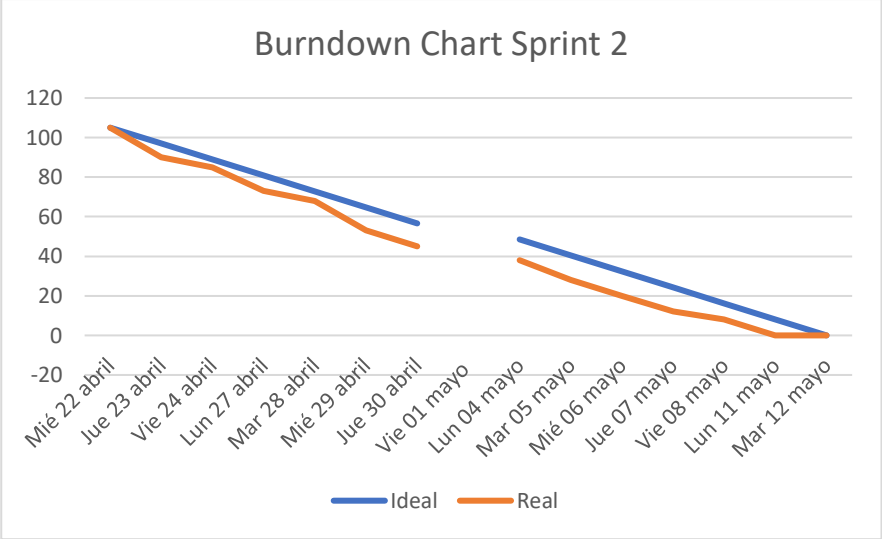
N°	Capa	Tarea	Historia	Estado	H. Estimadas	H. Reales
1	Backend	Módulo de empresa (RUC, razón social, nombre comercial, régimen, contribuyente especial, RIMPE)	US-03	completado	15	15
2	Backend	Configuración del ambiente SRI (pruebas / producción)	US-03	completado	5	5
3	Backend	Módulo de firma electrónica: carga de archivo .p12, contraseña cifrada, fecha de expiración	US-04	completado	12	12
4	Backend	Activación / desactivación de firma electrónica	US-04	completado	5	5
5	Backend	Implementación del algoritmo de firma XAdES (xml-crypto, node-forge)	US-04	completado	15	15
6	Backend	Módulo de establecimientos (CRUD)	US-05	completado	8	8
7	Backend	Módulo de puntos de emisión y secuenciales por tipo de documento	US-05	completado	8	8
8	Backend	Módulo de códigos IVA configurables por empresa	US-05	completado	7	7
9	Frontend	Formulario de configuración de empresa	US-03	completado	10	10
10	Frontend	UI de carga y gestión de firma electrónica	US-04	completado	8	8
11	Frontend	UI de establecimientos y puntos de emisión	US-05	completado	8	8
12	Frontend	UI de configuración de códigos IVA	US-06	completado	4	4
		TOTAL, SPRINT 2			105	105

El Sprint 2 se completó exitosamente entre el 22 de abril y el 12 de mayo de 2026, cumpliendo las 105 horas estimadas. Como se observa en la Figura 13, la interrupción

visible en el gráfico corresponde a un feriado, tras el cual se retomó el trabajo manteniendo un ritmo superior al planificado hasta cerrar el sprint sin inconvenientes.

Figura 13

Curva de Avance Real vs. Ideal – Sprint 2



Sprint 3: Gestión de Catálogos y Abstracción de Persistencia con ORM

- **Período:** 13 de mayo – 02 de junio 2026.
- **Descripción:** Este ciclo se orientó a la optimización arquitectónica y construcción de los catálogos operativos del sistema. Se integró Sequelize ORM para abstraer las consultas hacia PostgreSQL, aplicando validaciones de integridad referencial a nivel de modelo. Adicionalmente, para mitigar cuellos de botella en accesos concurrentes a datos de lectura frecuente como productos e impuestos, se implementó una capa de caché en memoria con Redis, reduciendo significativamente la latencia en el backend.

Como se muestra en la Tabla *15*, esta matriz detalla las tareas ejecutadas durante el sprint, su estado de avance y la comparación entre las horas estimadas y las horas reales invertidas.

Tabla 15

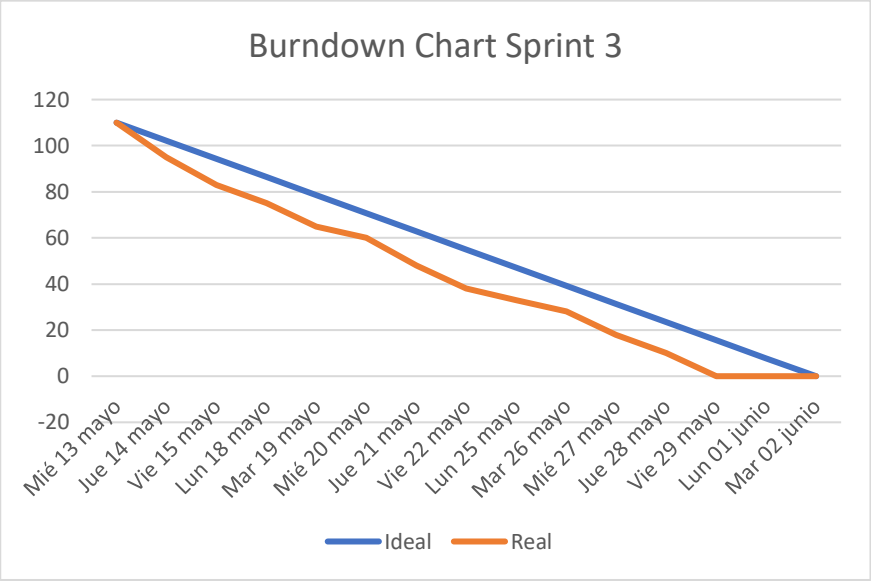
Matriz de Planificación y Tareas - Sprint 3

N°	Capa	Tarea	Historia	Estado	H. Estimadas	H. Reales
1	Backend	Módulo de clientes (CRUD, activación/desactivación, tipos de identificación)	US-07	completado	15	15
2	Backend	Módulo de proveedores (CRUD completo)	US-07	completado	12	12
3	Backend	Módulo de grupos de productos	US-08	completado	8	8
4	Backend	Módulo de productos (CRUD, IVA, ICE, IRBPNR)	US-08	completado	10	10
5	Backend	Instalación y configuración de Sequelize con PostgreSQL	US-20	completado	5	5
6	Backend	Definición de modelos Sequelize (clientes, proveedores, grupos, productos)	US-20	completado	12	12
7	Backend	Migración de consultas raw SQL a Sequelize en los módulos del sprint	US-20	completado	10	10
8	Backend	Validaciones a nivel de modelo con Sequelize	US-20	completado	5	5
9	Backend	Integración de caché Redis para consultas frecuentes	-	completado	5	5
10	Frontend	UI de gestión de clientes (listado, crear, editar, eliminar)	US-07	completado	10	10
11	Frontend	UI de gestión de proveedores	US-07	completado	8	8
12	Frontend	UI de catálogo de productos y grupos	US-08	completado	10	10
		TOTAL, SPRINT 3			110	110

El Sprint 3 se completó exitosamente entre el 13 de mayo y el 2 de junio de 2026, cumpliendo las 110 horas estimadas. Como se observa en la Figura 14, el ritmo de avance real fue consistentemente superior al planificado a lo largo de todo el sprint, lo que permitió cerrar todas las tareas sin contratiempos.

Figura 14

Curva de Avance Real vs. Ideal – Sprint 3



Sprint 4: Núcleo Core de Facturación Electrónica e Interoperabilidad SRI

- **Período:** 03 de junio – 23 de junio 2026.
- **Descripción:** Este ciclo representó la fase de mayor complejidad técnica y regulatoria, consolidando el flujo principal de emisión de facturas. Se desarrolló el motor de serialización de estructuras XML bajo los esquemas del SRI, el cliente asíncrono SOAP para la comunicación con los servicios gubernamentales de recepción y autorización, y una máquina de estados para el control del ciclo de vida de cada documento. Complementariamente, se integraron los módulos de generación del PDF con código de barras y el envío automático por correo mediante SMTP.

Como se muestra en la Tabla *16*, esta matriz detalla las tareas ejecutadas durante el sprint, su estado de avance y la comparación entre las horas estimadas y las horas reales invertidas.

Tabla 16

Matriz de Planificación y Tareas - Sprint 4

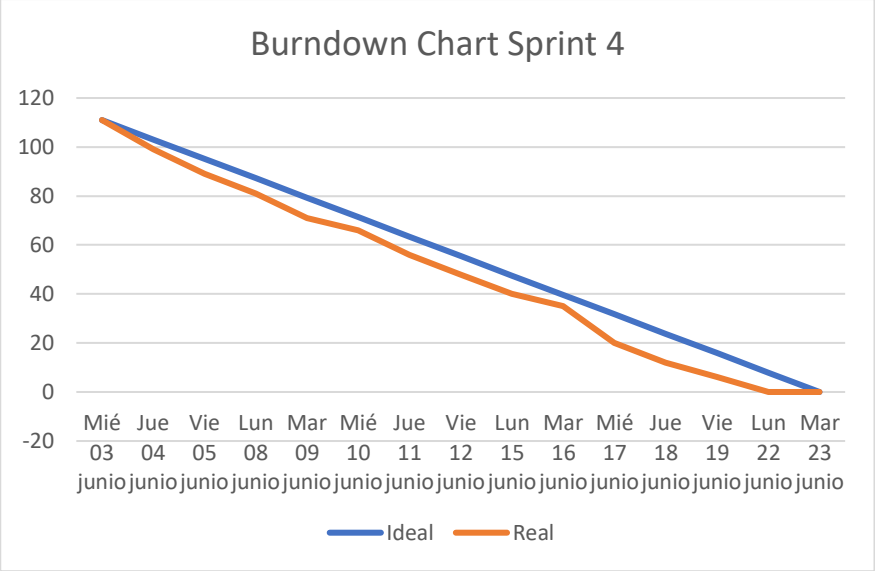
N°	Capa	Tarea	Historia	Estado	H. Estimadas	H. Reales
1	Backend	Generación de XML de factura según esquema SRI	US-09	completado	12	12
2	Backend	Firma digital de XML con XAdES y envío al SRI (SOAP)	US-09	completado	10	10
3	Backend	Consulta de autorización y manejo de estados (borrador → emitido → autorizado)	US-09	completado	8	8
4	Backend	Generación de PDF de factura con código de barras	US-17	completado	10	10
5	Backend	Envío de factura por correo electrónico (SMTP)	US-17	completado	5	5
6	Backend	Generación de XML y PDF de nota de venta RISE	US-15	completado	10	10
7	Backend	Módulo de proformas con generación de PDF	US-16	completado	8	8
8	Backend	Cliente SOAP SRI con reintentos y manejo de errores	US-09	completado	8	8
9	Backend	Log de comunicación SRI (log_sri)	-	completado	5	5
10	Frontend	Formulario de creación y emisión de facturas	US-09	completado	15	15
11	Frontend	Listado y detalle de facturas con estado de autorización	US-09	completado	8	8
12	Frontend	UI de nota de venta RISE	US-15	completado	6	6
13	Frontend	UI de proformas	US-16	completado	6	6
		TOTAL SPRINT 4			111	111

El Sprint 4 se completó exitosamente entre el 3 y el 23 de junio de 2026, cumpliendo las 111 horas estimadas. Como se observa en la Figura 15 en este sprint el avance real siguió de cerca

la línea ideal, mostrando un ritmo más uniforme y constante en comparación con sprints anteriores, cerrando todas las tareas dentro del plazo previsto.

Figura 15

Curva de Avance Real vs. Ideal – Sprint 4



Sprint 5: Comprobantes Complementarios y Mitigación de Vulnerabilidades OWASP

- **Período:** 24 de junio – 14 de julio 2026.
- **Descripción:** Este ciclo amplió las capacidades tributarias del sistema con el desarrollo de los comprobantes complementarios obligatorios: Notas de Crédito, Notas de Débito, Guías de Remisión, Liquidaciones de Compra y Comprobantes de Retención, reutilizando los módulos de firmado XAdES-BES y transmisión SOAP. En paralelo, se ejecutó la auditoría de seguridad bajo el estándar OWASP Top 10, implementando contramedidas en Express como cabeceras seguras, sanitización de entradas contra inyecciones SQL y políticas CORS restrictivas.

Como se muestra en la Tabla 17, esta matriz detalla las tareas ejecutadas durante el sprint, su estado de avance y la comparación entre las horas estimadas y las horas reales invertidas.

Tabla 17

Matriz de Planificación y Tareas - Sprint 5

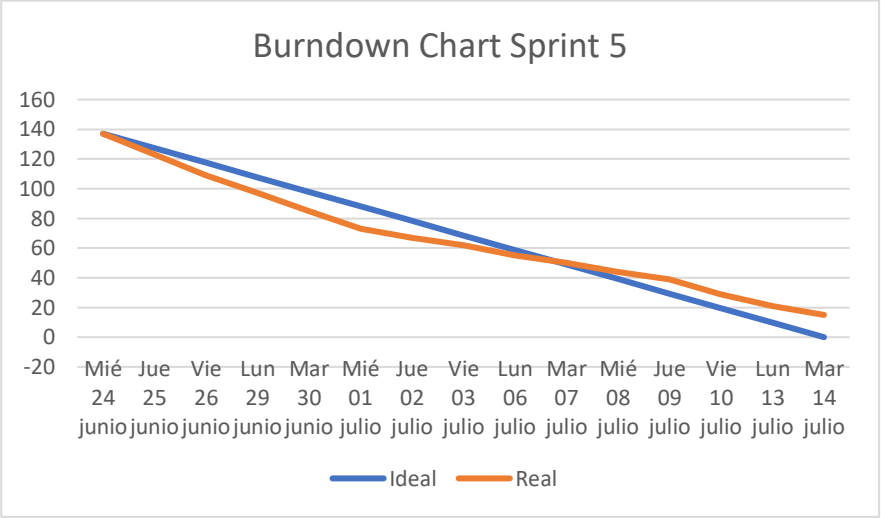
N°	Capa	Tarea	Historia	Estado	H. Estimadas	H. Reales
1	Backend	XML, firma y autorización de nota de crédito + PDF + correo	US-10, US-17	completado	14	14
2	Backend	XML, firma y autorización de nota de débito + PDF + correo	US-11, US-17	completado	14	14
3	Backend	XML, firma y autorización de guía de remisión + PDF	US-12, US-17	completado	12	12
4	Backend	XML, firma y autorización de liquidación de compra + PDF + correo	US-13, US-17	completado	12	12
5	Backend	XML, firma y autorización de retención + PDF	US-14, US-17	completado	12	12
6	Backend	A01 — Control de acceso: revisión y refuerzo de middleware de roles y permisos	US-21	completado	6	6
7	Backend	A02 — Fallos criptográficos: revisión de bcrypt, JWT y cifrado de firma .p12	US-21	completado	5	5
8	Backend	A03 — Inyección: sanitización de inputs y consultas parametrizadas con Sequelize	US-21	completado	7	7
9	Backend	A05 — Configuración insegura: Helmet.js, CORS estricto y variables de entorno	US-21	completado	5	5
10	Backend	A07 — Fallos de autenticación: rate limiting y bloqueo por intentos fallidos	US-21	completado	6	6
11	Backend	A09 — Registro y monitoreo: auditoría de	US-21	completado	5	5

		accesos y operaciones críticas				
12	Frontend	UI de nota de crédito y nota de débito	US-10, US-11	completado	10	10
13	Frontend	UI de guía de remisión y liquidación de compra	US-12, US-13	completado	8	8
14	Frontend	UI de retenciones	US-14	completado	6	6
15	Frontend	A01 — Middleware Next.js para protección de rutas privadas	US-21	completado	4	4
16	Frontend	A03 — Sanitización de inputs y evitar XSS	US-21	completado	4	4
17	Frontend	A05 — Security headers en next.config.js (CSP, X-Frame-Options)	US-21	completado	3	3
18	Frontend	A07 — JWT en httpOnly cookies, no en localStorage	US-21	completado	4	4
		TOTAL SPRINT 5			137	137

El Sprint 5 se completó exitosamente entre el 24 de junio y el 14 de julio de 2026, cumpliendo las 137 horas estimadas. Como se observa en la Figura 16, el avance real fue superior al planificado durante la mayor parte del sprint, manteniéndose por encima de la línea ideal y permitiendo cerrar todas las tareas dentro del plazo establecido.

Figura 16

Curva de Avance Real vs. Ideal – Sprint 5



Sprint 6: Automatización, Analítica y Cierre del Ciclo de Software

- **Período:** 15 de julio – 04 de agosto 2026.
- **Descripción:** El último ciclo consolidó las funcionalidades operativas y la estabilización del sistema. Se implementaron Cron Jobs para el procesamiento automatizado de comprobantes recurrentes y endpoints analíticos para alimentar un dashboard con indicadores clave de negocio (KPIs de facturación, productos frecuentes y tendencias de ingresos). El sprint concluyó con las pruebas End-to-End, depuración del sistema y la elaboración de los manuales técnicos y de usuario de la plataforma.

Como se muestra en la Tabla 18, esta matriz detalla las tareas ejecutadas durante el sprint, su estado de avance y la comparación entre las horas estimadas y las horas reales invertidas.

Tabla 18

Matriz de Planificación y Tareas - Sprint 6

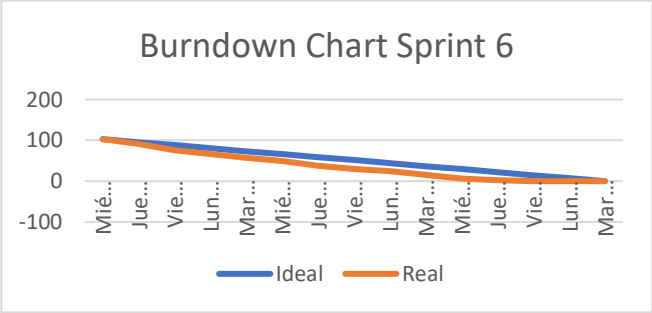
N°	Capa	Tarea	Historia	Estado	H. Estimadas	H. Reales
1	Backend	Módulo de facturación recurrente (configuración + cron automático)	US-18	completado	12	12

2	Backend	Dashboard con KPIs: resumen, totales del mes, comparativa, top productos y clientes frecuentes	US-19	completado	15	15
3	Backend	Pruebas integrales de todos los módulos	-	completado	10	10
4	Backend	Corrección de bugs detectados en pruebas	-	completado	8	8
5	Frontend	UI de facturación recurrente	US-18	completado	8	8
6	Frontend	Dashboard con gráficos y KPIs	US-19	completado	12	12
7	Frontend	Pruebas de interfaz y flujo completo del sistema	-	completado	8	8
8	Frontend	Correcciones finales de UI	-	completado	5	5
9	Ambos	Documentación técnica del sistema	-	completado	10	10
10	Ambos	Manual de usuario	-	completado	8	8
11	Ambos	Capítulo de metodología SCRUM	-	completado	5	5
12	Ambos	Entrega final del proyecto	-	completado	2	2
		TOTAL SPRINT 6			103	103

El Sprint 6 se completó exitosamente entre el 15 de julio y el 4 de agosto de 2026, cumpliendo las 103 horas estimadas. Al ser el sprint final, concentró las pruebas integrales, correcciones y la entrega del proyecto, lo que se refleja en la Figura 17 con un avance real notablemente más acelerado que el planificado, finalizando todas las tareas con anticipación.

Figura 17

Curva de Avance Real vs. Ideal – Sprint 6



Técnicas para el Aseguramiento de la Calidad y Validación del Sistema

Para garantizar el correcto funcionamiento de la plataforma, se definió una estrategia de calidad enfocada en dos frentes: verificar que la lógica del sistema cumpla con las normativas del SRI, y asegurar que la plataforma sea robusta frente a posibles ataques en entornos de alta concurrencia.

Pruebas Funcionales y de Aceptación (Caja Negra)

La validación del sistema se realizó mediante pruebas de caja negra, evaluando exclusivamente las entradas, salidas y comportamientos observables del software, sin considerar su estructura interna. Este método permitió contrastar cada incremento desarrollado contra los 25 requisitos funcionales definidos en el Product Backlog.

El proceso de pruebas funcionales se dividió en tres fases metodológicas:

- **Pruebas Unitarias:** Se validaron individualmente los formularios en Next.js y los endpoints en Express, verificando restricciones de entrada como la validación del dígito verificador del RUC o cédula antes de interactuar con la base de datos.
- **Pruebas de Integración:** Se verificó la interoperabilidad entre módulos, evaluando flujos donde intervienen múltiples componentes como Sequelize ORM, la caché en Redis y la transmisión asíncrona del XML hacia los servidores del SRI mediante SOAP.
- **Pruebas End-to-End (E2E):** Se simularon escenarios reales de uso para los perfiles de Administrador y Operador, validando ciclos completos como el aprovisionamiento de un nuevo inquilino, la carga de su firma .p12, la generación de comprobantes y la recepción de la autorización del SRI junto con el PDF por correo SMTP.

Validación de Seguridad Perimetral (OWASP Top 10)

Dado que la plataforma gestiona información financiera, transaccional y llaves criptográficas privadas de múltiples empresas, la seguridad del sistema se rigió bajo el estándar internacional OWASP Top 10. La validación consistió en auditorías de vulnerabilidades e implementación de contramedidas en el código fuente para cubrir los siguientes vectores de ataque:

- **A01:2021 Control de Acceso Deficiente:** Se implementaron middlewares de control de acceso basado en roles, verificando que cada petición incluya obligatoriamente un token JWT válido. Además, se validó que un usuario de un inquilino no pueda manipular parámetros en las consultas para acceder a registros de otra empresa, garantizando el aislamiento lógico de los datos.
- **A02:2021 Fallos Criptográficos:** Se verificó que las contraseñas de los usuarios sean procesadas con bcrypt aplicando un mínimo de 10 rondas de hash. Asimismo, se validó que las contraseñas de los certificados .p12 se almacenen en PostgreSQL bajo cifrado simétrico en reposo, impidiendo su exposición en texto plano.
- **A03:2021 Inyección:** Se validó que todas las interacciones con PostgreSQL se realicen mediante consultas parametrizadas a través de Sequelize ORM, prohibiendo el uso de concatenación directa de cadenas de texto introducidas por el usuario para prevenir inyecciones SQL.
- **A05:2021 Configuración Insegura:** Se implementó Helmet.js para inyectar cabeceras HTTP críticas como X-Content-Type-Options, X-Frame-Options y Content-Security-Policy. Adicionalmente, se configuraron políticas CORS restrictivas permitiendo peticiones únicamente desde dominios de confianza autorizados.
- **A07:2021 Fallos de Autenticación:** Se implementó Rate Limiting en el endpoint /login, bloqueando automáticamente las direcciones IP o cuentas que registren un número inusual de intentos fallidos consecutivos, previniendo ataques de fuerza bruta.
- **A09:2021 Fallos en el Registro y Supervisión:** Se implementó un sistema centralizado de auditoría mediante las tablas log_sri y logs de auditoría, registrando con marcas de tiempo cada intento de acceso, modificación de privilegios, error de comunicación con el SRI y evento crítico del sistema, garantizando la trazabilidad forense ante cualquier anomalía.

CAPÍTULO IV

El desarrollo de la presente plataforma web tipo SaaS multitenant surge como respuesta a las limitaciones técnicas, operativas y económicas que enfrentan las PYMEs en el Ecuador para cumplir con el esquema transaccional obligatorio dictado por el SRI. Para establecer la línea base del proyecto, se evaluó el comportamiento del entorno pre-implementación bajo dos ejes fundamentales: la eficiencia operativa del usuario final y el costo computacional de la infraestructura tecnológica tradicional.

Evaluación Operativa del Proceso de Facturación Manual

Muchas microempresas y personas naturales que no cuentan con un sistema propio usan el portal gratuito "SRI en Línea" para emitir sus facturas electrónicas. Aunque cumple con lo que exige la ley, la herramienta es bastante básica: no automatiza tareas, no guarda un historial completo y no se conecta fácilmente con otros sistemas, lo que hace que facturar sea un proceso lento, repetitivo y con alto riesgo de cometer errores. Al analizar cómo se manejaba la contabilidad antes de implementar el sistema, se encontró que emitir una sola factura de forma manual obliga al usuario a seguir una serie de pasos en orden, sin saltarse ninguno:

- **Autenticación Síncrona:** Acceso repetitivo al portal estatal mediante credenciales individuales por cada sesión de facturación.
- **Digitación e Inserción de Datos del Cliente:** Introducción manual y carácter por carácter del número de RUC o cédula, nombres completos, dirección y correo electrónico del receptor, al no contar con un módulo indexado de clientes frecuentes.
- **Codificación e Ingreso de Ítems:** Búsqueda y transcripción manual de los códigos de productos, descripción de bienes o servicios, cantidades y precios unitarios en cada transacción.
- **Desglose y Cálculo de Impuestos:** Configuración manual de las tarifas de Impuesto al Valor Agregado (IVA) vigentes (v.g., 0% o 15%) y retenciones aplicables, incrementando el riesgo de discrepancias contables por fallos en el cálculo aritmético externo.
- **Firmado Criptográfico y Transmisión SOAP:** Carga del archivo de firma electrónica .p12 de forma local y envío manual a través del navegador web, proceso que se ve

gravemente afectado durante los periodos de alta concurrencia y saturación de los servidores del SRI.

Este proceso tan fragmentado genera cuellos de botella importantes en negocios que manejan un volumen medio o alto de transacciones. Además, la información solo se podía guardar descargando los archivos XML y PDF uno por uno, lo que impedía consolidar reportes de ventas, hacer seguimiento para auditorías y tomar decisiones financieras en el momento que se necesitaban.

Diagnóstico de Infraestructura y Concurrencia de Recursos

Desde el punto de vista del desarrollo de software, la mayoría de soluciones comerciales que existen para reducir el trabajo manual funcionan sobre arquitecturas monolíticas o de un solo cliente por servidor. Esto obliga a cada empresa a operar y mantener su propio servidor web y base de datos por separado, con la infraestructura completamente independiente.

Para una PYME ecuatoriana, mantener infraestructura dedicada en la nube como instancias en AWS o Azure implica reservar recursos de cómputo que permanecen en su mayoría subutilizados. Por eso, la Tabla 19 presenta la base técnica que justifica optar por un modelo compartido, pero con los datos de cada empresa aislados de forma segura mediante un identificador único (empresa_id).

Tabla 19

Capacidad Dedicada y Subutilización de la Infraestructura (Single-tenant)

Componente del Servidor	Capacidad Dedicada (Mínima)	Porcentaje de Ocio de CPU / RAM
Servidor Web / API	2 vCPU / 4 GB RAM	85% (Solo se usa en horario comercial)
Servidor de Base de Datos	1 vCPU / 2 GB RAM (PostgreSQL)	90% (Inactiva entre transacciones)
Almacenamiento y SSL	20 GB SSD + Certificado dedicado	0%

Balance Global por Empresa	**Infraestructura Single-Tenant Aislada**	**Eficiencia: < 15%**

El análisis muestra que el 85% del tiempo los servidores dedicados permanecen ociosos, lo que significa que cada empresa reserva recursos de cómputo que casi no utiliza. Por ello, el proyecto propone migrar a una infraestructura SaaS Multitenant, donde múltiples empresas comparten una misma plataforma, optimizando el aprovechamiento de los recursos y elevando la eficiencia operativa sin sacrificar disponibilidad ni rendimiento.

Desarrollo e Implementación del Sistema (Ciclo de Sprints)

Para construir la plataforma se utilizó la metodología ágil Scrum, organizando el desarrollo en ciclos llamados Sprints. Cada uno incluyó las etapas de planificación, diseño de interfaces, programación (backend y frontend) y pruebas. A continuación, se describe el avance cronológico del proyecto, enfocado en los módulos principales del sistema SaaS Multitenant y la facturación electrónica.

Fundación del Sistema, Control de Acceso Colectivo y Gestión de Identidades

El primer sprint tuvo una duración de 3 semanas, del 1 al 21 de abril de 2026, con 95 horas planificadas. Su objetivo fue configurar el entorno base del proyecto, diseñar el modelo relacional de datos e implementar el sistema de autenticación de tres factores junto con el control de accesos basado en roles (RBAC).

Entorno Visual de la Capa de Presentación (Frontend)

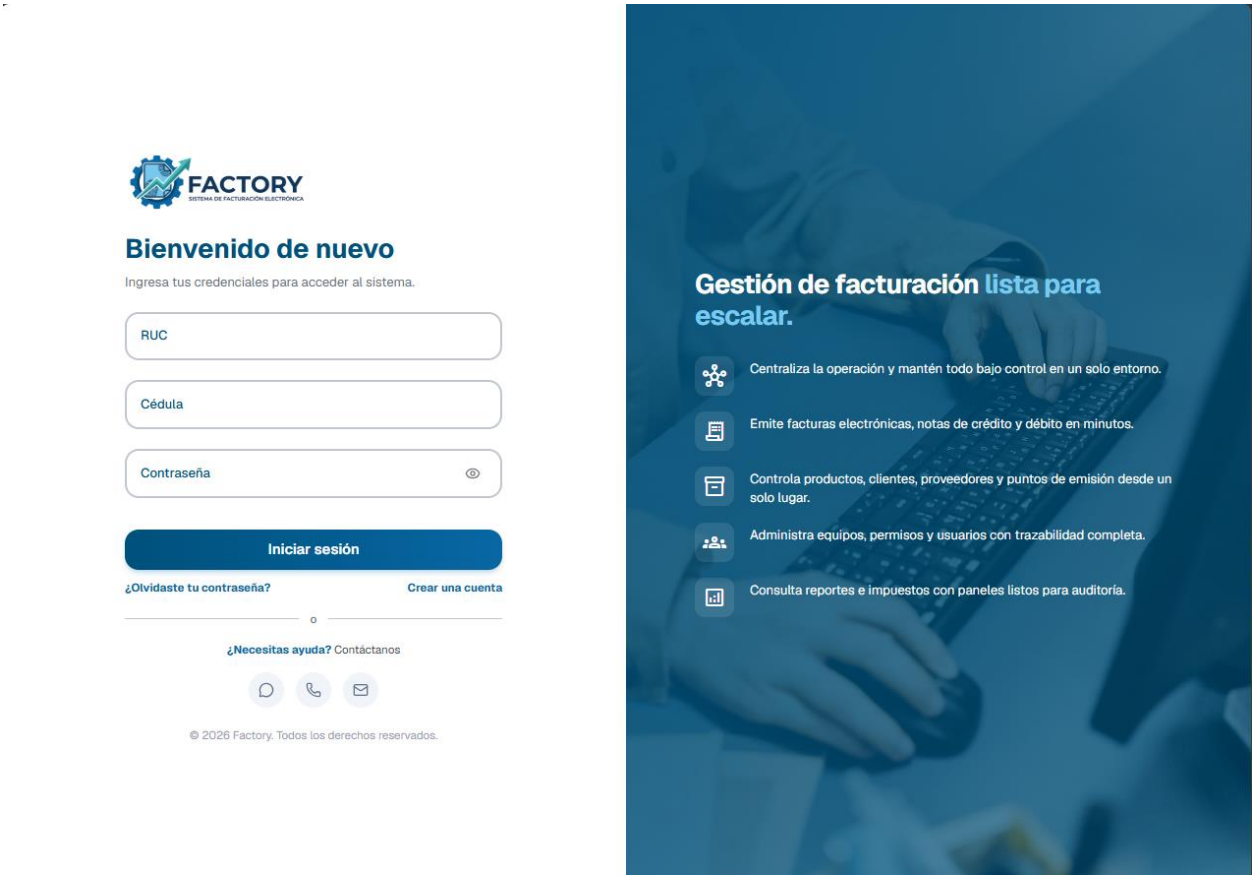
En esta subcapa se estructuró el proyecto en Next.js, implementando un diseño modular enfocado en la captura limpia de credenciales y la seguridad perimetral del lado del cliente.

- **Módulo de Autenticación (Login) (US-01):** Interfaz de acceso que solicita explícitamente tres parámetros de seguridad: el RUC de la empresa, la Cédula del usuario y la Contraseña. Este diseño asegura que el intento de autenticación se ejecute directamente sobre el contexto de un *tenant* específico, evitando colisiones de credenciales entre diferentes organizaciones que utilicen la plataforma.
- **Interfaces de Registro y Recuperación de Credenciales (US-01, US-02):** Vistas responsivas provistas de validaciones dinámicas para flujos de registro y el envío automatizado de tokens para el restablecimiento de contraseñas.

Como se muestra en la Figura 18, la pantalla de inicio solicita tres datos: RUC, cédula y contraseña. Al hacer clic en "Ingresar", el sistema envía esa información de forma segura a la API. Si las credenciales son correctas, el servidor devuelve un token JWT que contiene el identificador de la empresa, el usuario y su rol, lo que permite que la interfaz muestre u oculte automáticamente las opciones del menú según los permisos de cada colaborador.

Figura 18

Interfaz de Control de Acceso y Autenticación por Empresa



Parametrización del Inquilino, Gestión de Emisión y Núcleo Criptográfico XAdES

El segundo sprint tuvo una duración de 3 semanas, del 22 de abril al 12 de mayo de 2026, con 105 horas planificadas. Su objetivo fue desarrollar los módulos de configuración de la empresa, carga de certificados digitales, registro seguro de sucursales y puntos de emisión, e implementar el núcleo criptográfico para el firmado electrónico XAdES-BES.

Entorno Visual de la Capa de Presentación (Frontend)

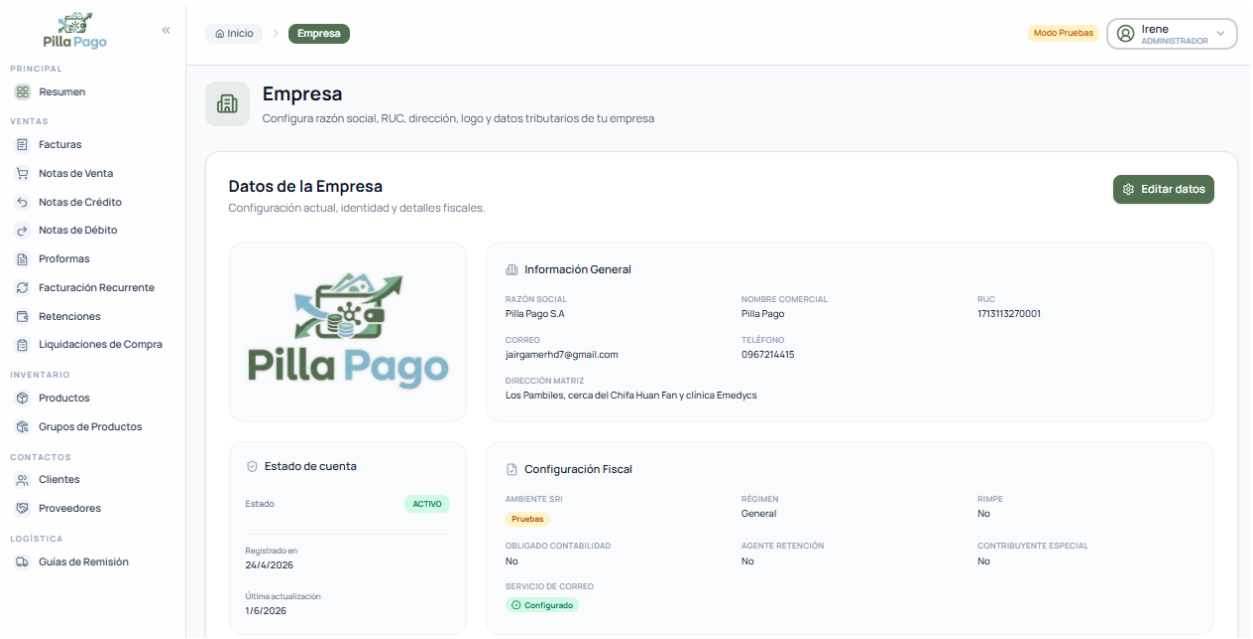
El enfoque de este ciclo se centró en las interfaces de administración tributaria y configuraciones del entorno corporativo.

- **Panel de Configuración de la Empresa y Firma (US-03, US-04):** Formulario exclusivo para el administrador enfocado en gestionar los datos fiscales de la empresa (Régimen, RIMPE, Contribuyente Especial) y el canal de carga del archivo de firma electrónica .p12.
- **UI de Establecimientos y Puntos de Emisión (US-05):** Tablas interactivas donde el inquilino añade y asigna los códigos de sucursales (ej. 001) y puntos de impresión (ej. 002) necesarios para la secuencia numérica fiscal.

Como se observa en la Figura 19, la pantalla muestra los campos necesarios para declarar correctamente la información de la empresa ante el SRI, como la razón social, dirección, tipo de contribuyente y demás parámetros tributarios requeridos para la generación de comprobantes electrónicos válidos.

Figura 19

Interfaz de Parametrización del Tenant



Persistencia con Sequelize, Catálogos Base y Optimización con Redis

El tercer sprint tuvo una duración de 3 semanas, del 13 de mayo al 2 de junio de 2026, con 110 horas planificadas. Su objetivo fue implementar Sequelize ORM como capa de persistencia,

migrar las consultas directas, modelar las relaciones de los catálogos de clientes, proveedores y productos, e integrar Redis para optimizar las consultas de alta demanda.

Entorno Visual de la Capa de Presentación (Frontend)

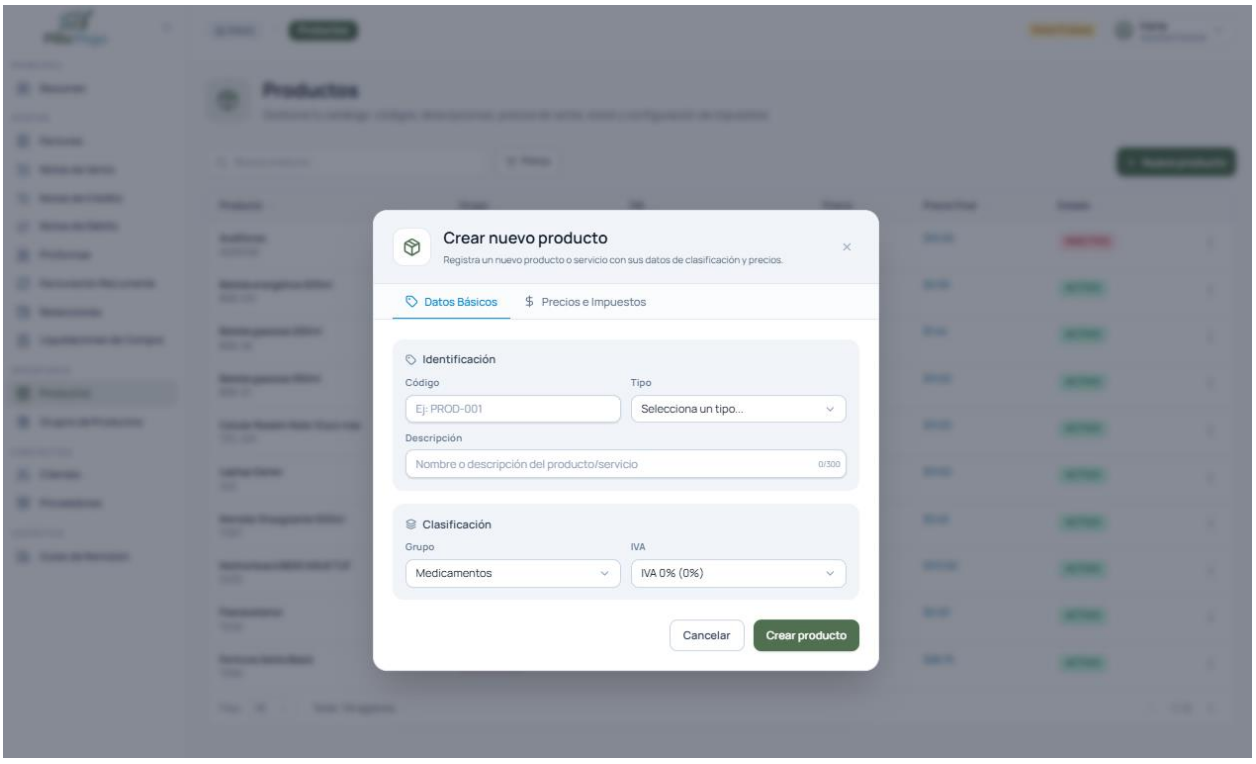
Con este sprint, el sistema quedó equipado con las herramientas necesarias para gestionar y almacenar la información transaccional de cada negocio.

- **Módulo de Administración de Clientes y Proveedores (US-07):** Interfaz fluida con tablas parametrizadas que permiten búsquedas inteligentes por número de identificación comercial o nombres, aislando de forma estricta los registros de terceros entre empresas.
- **Módulo de Catálogo Maestro de Productos y Categorías (US-08):** Formulario estructurado para mapear el inventario de bienes o servicios de los inquilinos, enlazándolos con selectores fijos para los desgloses de impuestos (IVA, ICE, IRBPNR).

Como se observa en la Figura 20, las vistas del catálogo cargan automáticamente las tasas impositivas configuradas en el sprint anterior. Al guardar un producto, la información se actualiza en tiempo real y se refleja de inmediato en los módulos de facturación que dependen de estos datos.

Figura 20

UI de Administración y Mapeo del Inventario Corporativo



Generación XML, Integración SOAP con el SRI, RIDE y Notificaciones SMTP

El cuarto sprint tuvo una duración de 3 semanas, del 3 al 23 de junio de 2026, con 111 horas planificadas. Su objetivo fue construir el motor de generación de documentos XML bajo el esquema oficial del SRI, integrar la comunicación SOAP para el envío y autorización de comprobantes electrónicos (facturas, notas de venta RISE y proformas), diseñar la representación impresa del documento electrónico (RIDE) en formato PDF e implementar el sistema de envío de correos SMTP.

Entorno Visual de la Capa de Presentación (Frontend)

El desarrollo en la capa del cliente Next.js se orientó a la manipulación dinámica de transacciones comerciales complejas y al monitoreo del ciclo de vida de los comprobantes electrónicos (US-09, US-15, US-16).

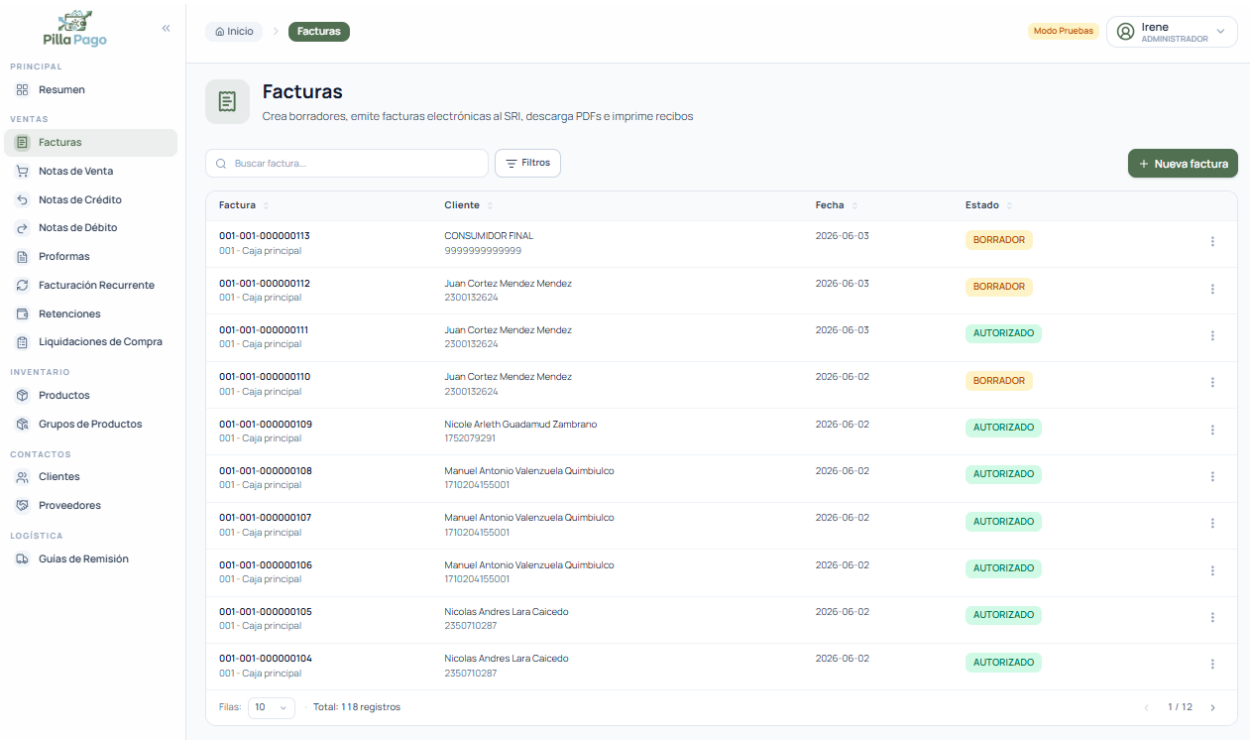
- **Formulario de emisión de facturas:** Interfaz dinámica que calcula en tiempo real las bases imponibles, desglose de IVA, ICE y retenciones. Incluye búsqueda rápida de clientes y productos del catálogo desarrollado en el sprint anterior.

- **Consola de seguimiento tributario:** Panel centralizado donde el usuario visualiza el estado de cada comprobante en tiempo real (Borrador → Firmado → Enviado → Recibido → Autorizado / Rechazado), con opción de descargar el XML y el PDF (RIDE) de forma inmediata.
- **Notas de venta y proformas:** Interfaces simplificadas adaptadas a las reglas del régimen RIMPE para contribuyentes de negocios populares, así como para la emisión de ofertas comerciales previas a la facturación.

Como se observa en la Figura 21, la consola de emisión oculta toda la complejidad técnica del backend al usuario. Al presionar "Emitir", el sistema bloquea el formulario para evitar cambios accidentales mientras se procesa la solicitud. Si el SRI devuelve algún error, la interfaz lo interpreta y muestra un mensaje claro al usuario indicando exactamente qué campo debe corregir.

Figura 21

Panel de Emisión de Comprobantes y Monitor de Estados SOAP SRI



Comprobantes Complementarios y Seguridad OWASP Top 10

El quinto sprint tuvo una duración de 3 semanas, del 24 de junio al 14 de julio de 2026, con 137 horas planificadas. Su objetivo fue desarrollar los motores de generación, firmado y envío

de los comprobantes electrónicos complementarios (notas de crédito, notas de débito, guías de remisión, liquidaciones de compra y comprobantes de retención), además de realizar una auditoría de seguridad basada en el estándar OWASP Top 10 para fortalecer el entorno multitenant.

Entorno Visual de la Capa de Presentación (Frontend)

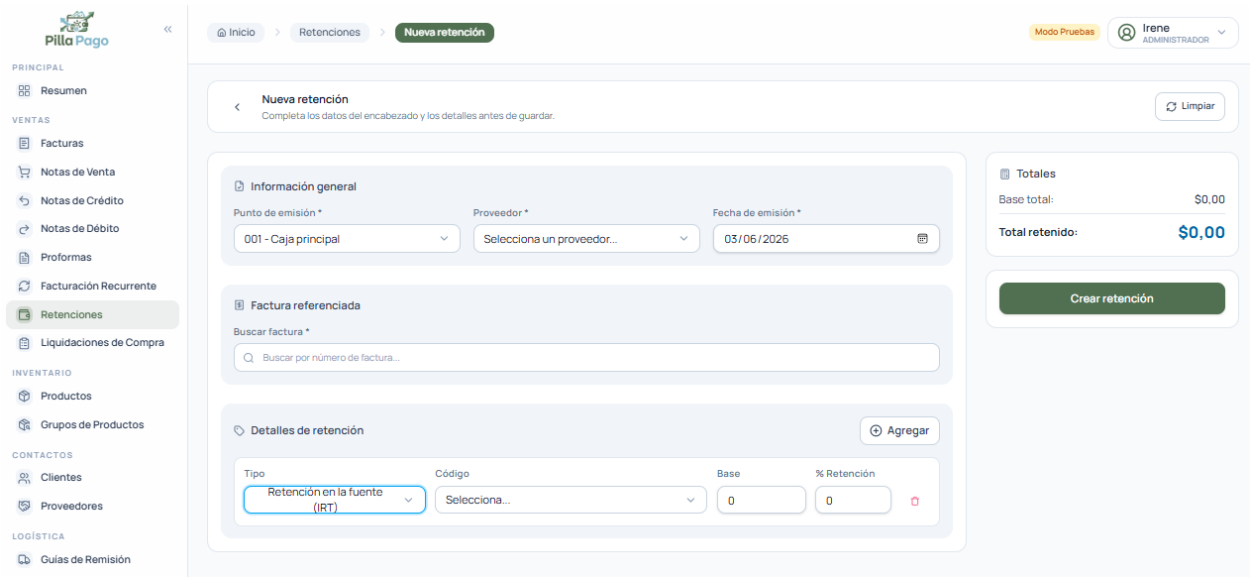
El frontend absorbió los flujos especializados de la contabilidad tributaria, reutilizando componentes atómicos de diseño e integrando capas perimetrales de seguridad a nivel del cliente (US-10, US-11, US-12, US-13, US-14, US-21).

- **Retenciones y documentos de ajuste:** Formularios dinámicos que adaptan sus campos según el tipo de comprobante seleccionado. El módulo de retenciones mapea automáticamente los casilleros fiscales a partir del catálogo base, agilizando el proceso de registro.
- **Middleware de seguridad y protección de rutas:** Se implementó un middleware en Next.js que controla el acceso a cada sección de la plataforma según el rol del usuario, bloqueando cualquier intento de acceso no autorizado. Además, se configuraron cabeceras de seguridad Content Security Policy (CSP) para reforzar la protección del entorno multitenant.

Como se observa en la Figura 22, la interfaz muestra únicamente las opciones que corresponden al rol de cada usuario. Los datos ingresados pasan por una función de limpieza que neutraliza caracteres especiales para prevenir ataques XSS. Además, los tokens de sesión se almacenan en cookies con la bandera httpOnly, impidiendo que puedan ser leídos o manipulados desde el navegador.

Figura 22

Interfaz de Retenciones Electrónicas



Automatización, Facturación Recurrente, Dashboard de KPIs y Cierre Técnico

El sexto sprint tuvo una duración de 3 semanas, del 15 de julio al 4 de agosto de 2026, con 103 horas planificadas. Su objetivo fue implementar tareas programadas (Cron Jobs) para automatizar la facturación recurrente, construir las consultas necesarias para mostrar métricas financieras en tiempo real en el dashboard, ejecutar las pruebas extremo a extremo (E2E) y compilar la documentación técnica final del proyecto.

Entorno Visual de la Capa de Presentación (Frontend)

Este sprint añadió al sistema de capacidades analíticas avanzadas, consolidando todos los datos transaccionales generados en los sprints anteriores.

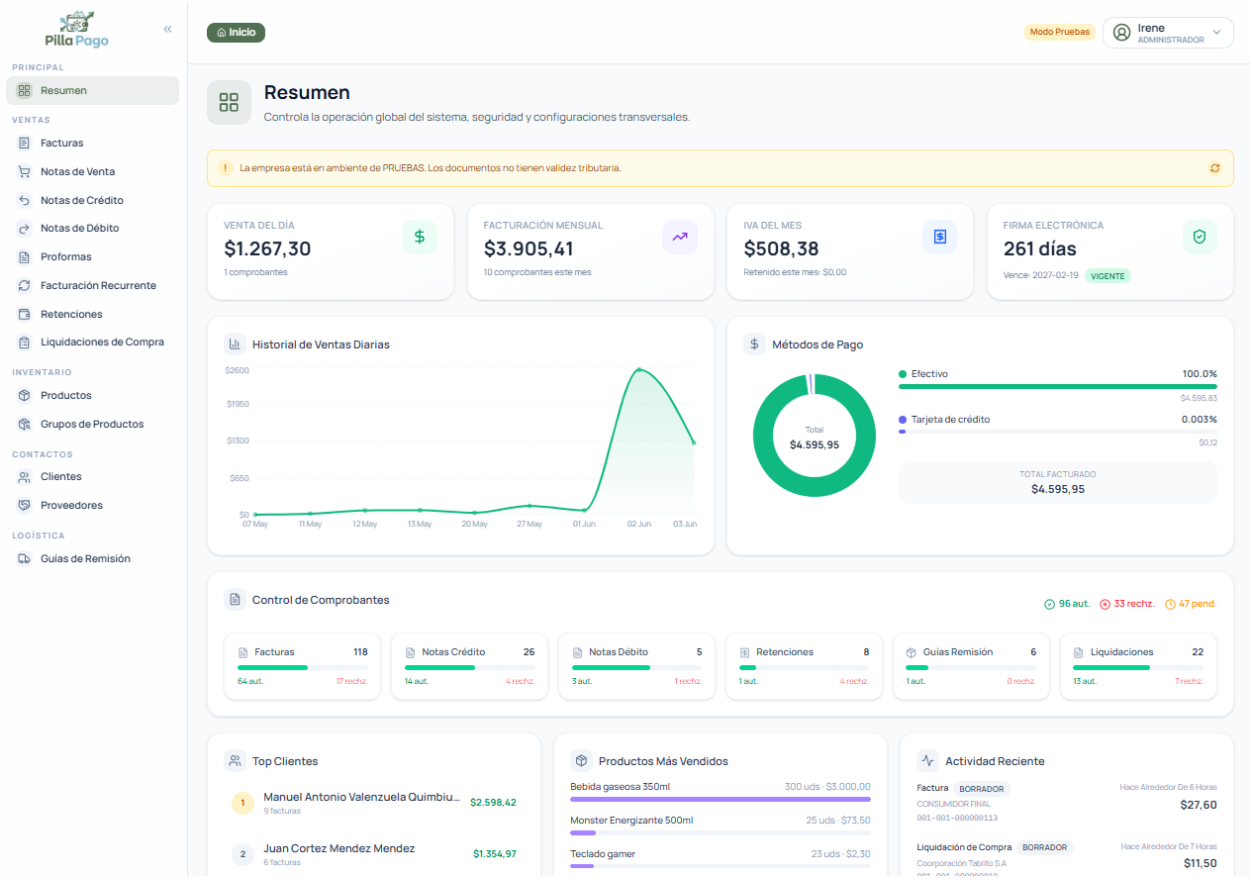
- **Dashboard de KPIs:** Interfaz gráfica que muestra el comportamiento comercial de cada empresa mediante gráficos interactivos, incluyendo la facturación mensual, comparativas interanuales, los 10 productos más vendidos y alertas automáticas sobre certificados de firma electrónica próximos a vencer.

- **Facturación recurrente:** Interfaz donde el usuario configura contratos de servicios fijos, asociando un cliente, productos, frecuencia de cobro (mensual, trimestral o anual) y la fecha en que el sistema ejecutará automáticamente la emisión del comprobante.

Como se observa en la Figura 23, el dashboard transforma los datos operativos en información estratégica útil para la toma de decisiones. La interfaz realiza consultas asíncronas optimizadas hacia el backend, logrando un renderizado de gráficos rápido y una experiencia fluida para el usuario.

Figura 23

Dashboard Ejecutivo de Control Transaccional Multi-tenant con Métricas de Rendimiento



Abstracción y Especialización por Tipo de Comprobante

Si bien el núcleo criptográfico y la comunicación SOAP son compartidos y reutilizados entre todos los comprobantes, el sistema aplica el patrón de diseño Factory para aislar las reglas

y validaciones específicas de cada tipo de documento. La Tabla 20 detalla las especificaciones aplicadas por el sistema para cada clase documental.

Tabla 20

Particularidades Lógicas de Software por Tipo de Comprobante

Comprobante Electrónico	Código SRI	Campo Crítico / Regla Lógica en Software	Patrón / Comportamiento Específico
Factura	1	Mapeo obligatorio de datos del adquirente, desglose de impuestos (IVA 0%, 15%, etc.) y formas de pago financieras.	Valida de forma estricta tipos de cédula, RUC, pasaporte o la etiqueta genérica de "Consumidor Final".
Nota de Crédito	4	Campos requeridos obligatorios: <FechaEmisión>, <factura> y <motivo>.	Modifica la consistencia contable vinculando el documento con el UUID/ID de la factura original en la base de datos.
Nota de Débito	5	Requiere nexos estrictos con el comprobante base modificado para el cobro de intereses o gastos administrativos.	Aplica el cálculo incremental sobre los saldos originales del módulo de cuentas por cobrar.
Guía de Remisión	6	Campos críticos: <rucTransportista>, <razonSocialTransportista>, <placa>, <fechaIniTransporte> y <destinatario>.	No posee valor monetario global. El backend suprime las validaciones financieras y activa la verificación de fechas lógicas de ruta.

Comprobante de Retención	7	Especificar el código de retención en la fuente (ej. 312, 343) o IVA.	Requiere la validación del campo (Formato MM/AAAA) asociado al periodo contable activo de la declaración.
Liquidación de Compra	3	Emisión exclusiva del agente de retención hacia proveedores que por su nivel cultural no poseen RUC.	El sistema autogenera el nodo de retención interno simulando un doble flujo transaccional.

Validación Funcional mediante Pruebas de Caja Negra

Para validar que la plataforma cumple con las reglas de negocio, los lineamientos técnicos y los requisitos del SRI, se diseñó un plan de pruebas de Caja Negra de forma manual. Este enfoque permitió evaluar el sistema desde la perspectiva del usuario final, verificando la usabilidad de la interfaz, los controles de formularios, el manejo de errores y el comportamiento ante flujos de navegación reales. Para el diseño de los escenarios de prueba se aplicaron de forma analítica dos técnicas formales de ingeniería de software:

- **Pruebas Funcionales:** Permitieron verificar de manera sistemática que el comportamiento operativo y las características de la plataforma cumplan estrictamente con los requerimientos funcionales especificados (tales como la correcta emisión de documentos, la firma electrónica en formato XAdES-BES y el flujo de autorización ante el SRI), validando que las respuestas del sistema ante las entradas de los usuarios sean las correctas sin necesidad de evaluar el código interno.
- **Pruebas de Estrés:** Enfocadas en evaluar la estabilidad, la resiliencia y la capacidad de respuesta de la arquitectura multi-tenant al ser sometida a escenarios de alta concurrencia y carga masiva de transacciones simultáneas. Estas pruebas permitieron identificar el punto de ruptura del sistema y el comportamiento de los recursos bajo presión extrema, garantizando la alta disponibilidad del servicio de facturación en la nube.

Pruebas funcionales

A continuación, en la Tabla 21, se presenta la matriz consolidada que resume la ejecución y cobertura de las pruebas de caja negra automatizadas mediante la herramienta Locust. La cuantificación de los casos de prueba ejecutados corresponde a la verificación sistemática de escenarios críticos, tanto positivos como de manejo de errores, asociados a cada Requisito Funcional (RF) implementado en la plataforma. Esta matriz permite evidenciar el grado de cobertura funcional alcanzado y los resultados obtenidos durante el proceso de validación del sistema.

Tabla 21

Matriz Resumen de Ejecución y Cobertura de Pruebas de Caja Negra por Requisito Funcional

Método	Caso de prueba	Estado esperado	Requests	Fallos
POST	TC01 · Autenticación JWT — clave inválida	401	4	0
POST	TC01 · Autenticación JWT — credenciales válidas	200	4	0
POST	TC02 · Recuperación contraseña — código inválido	400	1	0
POST	TC02 · Recuperación contraseña — solicitud válida	200	1	0
POST	TC03 · Gestión usuarios — body vacío	400	9	0
GET	TC03 · Gestión usuarios — listar	200	9	0
GET	TC04 · Asignación de roles — listar usuarios con rol	200	10	0
POST	TC04 · Asignación de roles — rol inexistente	400	10	0
GET	TC05 · Empresa/Ambiente — listar ambientes	200	8	0
GET	TC05 · Empresa/Ambiente — obtener empresa	200	8	0

Método	Caso de prueba	Estado esperado	Requests	Fallos
GET	TC06 · Firma electrónica — listar	200	6	0
POST	TC06 · Firma electrónica — sin archivo .p12	400	6	0
PATCH	TC07 · Activar firma — firma inexistente	400	8	0
GET	TC07 · Activar firma — listar para verificar estado	200	8	0
POST	TC08 · Establecimientos — body vacío	400	7	0
GET	TC08 · Establecimientos — listar	200	7	0
GET	TC09 · Puntos/Secuenciales — listar puntos	200	6	0
GET	TC09 · Puntos/Secuenciales — tipos de documento	200	6	0
GET	TC10 · Códigos IVA — inexistente	404	7	0
GET	TC10 · Códigos IVA — listar	200	7	0
POST	TC11 · Clientes — body vacío	400	8	0
GET	TC11 · Clientes — listar	200	8	0
POST	TC12 · Proveedores — body vacío	400	10	0
GET	TC12 · Proveedores — listar	200	10	0
POST	TC13 · Productos — body vacío	400	12	0
GET	TC13 · Productos — listar	200	12	0

Método	Caso de prueba	Estado esperado	Requests	Fallos
POST	TC14 · Facturas — body vacío	400	5	0
GET	TC14 · Facturas — listar	200	5	0
POST	TC15 · Notas crédito — body vacío	400	8	0
GET	TC15 · Notas crédito — listar	200	8	0
POST	TC16 · Notas débito — body vacío	400	9	0
GET	TC16 · Notas débito — listar	200	9	0
POST	TC17 · Guías remisión — body vacío	400	7	0
GET	TC17 · Guías remisión — listar	200	7	0
POST	TC18 · Liquidaciones compra — body vacío	400	9	0
GET	TC18 · Liquidaciones compra — listar	200	9	0
POST	TC19 · Retenciones — body vacío	400	6	0
GET	TC19 · Retenciones — listar	200	6	0
POST	TC20 · Notas venta RISE — body vacío	400	9	0
GET	TC20 · Notas venta RISE — listar	200	9	0
GET	TC21 · Proformas — PDF inexistente	404	5	0
GET	TC21 · Proformas — listar	200	5	0

Método	Caso de prueba	Estado esperado	Requests	Fallos
GET	TC22 · PDF código de barras — factura inexistente	404	6	0
GET	TC22 · PDF código de barras — nota crédito inexistente	404	6	0
POST	TC23 · Envío correo — factura inexistente	404	7	0
POST	TC23 · Envío correo — nota crédito inexistente	404	7	0
POST	TC24 · Recurrentes — body vacío	400	8	0
GET	TC24 · Recurrentes — listar	200	8	0
GET	TC25 · Dashboard KPIs — obtener	200	10	0
POST	[SETUP] Login inicial	200	2	0
TOTAL	**Agregado**	—	**362**	**0**

Resumen de Resultados de las Pruebas Funcionales

Para validar el correcto funcionamiento del sistema desarrollado, se ejecutaron pruebas sobre los procesos e integraciones más relevantes. La Tabla *22* resume los casos de prueba aplicados, los resultados esperados, los resultados obtenidos y el estado de cumplimiento de cada uno de ellos.

Tabla 22

Resultados Obtenidos en la Ejecución de Pruebas

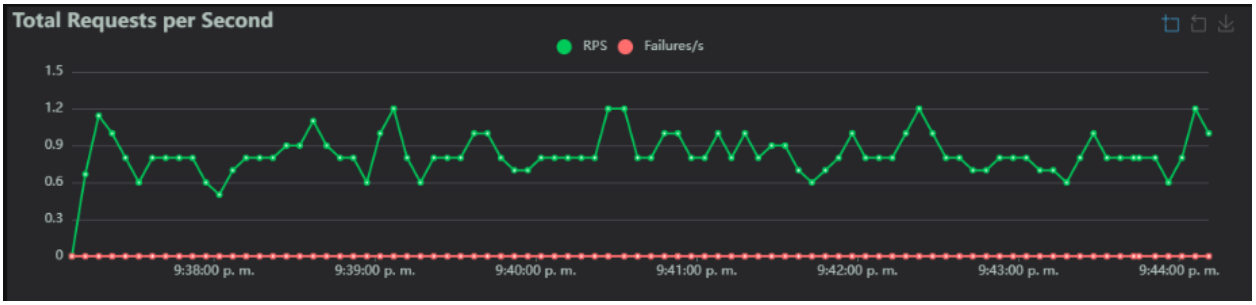
ID	Caso de Prueba	Requisitos	Resultado Esperado	Resultado Obtenido	Estado
CP-MAN-01	Control de Accesos y Autenticación	RF-01, RF-04	El sistema debe validar las credenciales, impedir accesos inválidos y	El sistema rechazó credenciales incorrectas y permitió el acceso correcto con	EXITOSO

			permitir el ingreso de usuarios autorizados.	redirección al dashboard.	
CP-MAN-02	Validación y Resguardo de Firma Electrónica	RF-06, RF-07	El sistema debe validar el formato y la contraseña de la firma electrónica antes de almacenarla.	El sistema rechazó archivos y contraseñas inválidas, y registró correctamente la firma válida.	EXITOSO
CP-MAN-03	Ciclo Core de Facturación y Conectividad SOAP SRI	RF-14, RF-22, RF-23	El sistema debe generar, firmar y enviar la factura al SRI obteniendo su autorización y generando el RIDE.	La factura fue procesada, autorizada por el SRI y el RIDE se generó correctamente.	EXITOSO

La Figura 24 muestra el comportamiento temporal de las pruebas automatizadas ejecutadas mediante la herramienta Locust. Se observa que la tasa de solicitudes por segundo (RPS) se mantuvo estable durante toda la ejecución, con valores cercanos a una solicitud por segundo, mientras que la tasa de fallos permaneció en cero. Estos resultados evidencian la correcta respuesta del sistema ante los escenarios evaluados y corroboran la ausencia de errores durante la validación funcional de los requisitos implementados.

Figura 24

Evolución de las Solicitudes por Segundo (RPS) y Tasa de Fallos Durante la Ejecución de las Pruebas Automatizadas con Locust.



Conclusiones del Proceso de Validación

La ejecución de los 25 escenarios de prueba de caja negra manual arrojó un 100% de efectividad, registrando cero fallos críticos en el entorno de producción local y de pruebas del SRI. Se comprobó que las interfaces desarrolladas actúan como una capa segura que impide al usuario ingresar datos que violen las directrices tributarias, garantizando la consistencia relacional de la base de datos PostgreSQL multi-tenant y la correcta comunicación transaccional orientada a servicios.

Pruebas de estrés

A continuación, en la Tabla 23, se presentan los resultados consolidados de las pruebas de estrés ejecutadas mediante la herramienta Locust. La evaluación consideró escenarios de lectura, escritura y picos de demanda asociados a los principales módulos del sistema, con el propósito de analizar su comportamiento bajo condiciones de alta concurrencia y carga sostenida. En total, se procesaron 6 463 solicitudes sin registrar fallos, alcanzando una tasa de éxito del 100 %. Los resultados obtenidos evidencian la capacidad del sistema para mantener la disponibilidad y la estabilidad operativa ante volúmenes elevados de peticiones y escenarios de uso intensivo.

Tabla 23

Resultados Consolidados de las Pruebas de Estrés Ejecutadas Mediante Locust.

Método	Escenario de prueba	Solicitudes ejecutadas	Fallos
POST	TC01 [LECTURA] Autenticación JWT	99	0
GET	TC03 [LECTURA] Listar usuarios	60	0
GET	TC04 [LECTURA] Usuarios con roles	61	0
GET	TC05 [LECTURA] Ambiente SRI	74	0
GET	TC05 [LECTURA] Datos empresa	118	0
GET	TC05 [PICO] Ráfaga empresa	138	0

Método	Escenario de prueba	Solicitudes ejecutadas	Fallos
GET	TC06 [LECTURA] Listar firmas electrónicas	74	0
GET	TC07 [LECTURA] Estado de firmas	70	0
GET	TC08 [LECTURA] Listar establecimientos	99	0
GET	TC09 [LECTURA] Listar puntos de emisión	118	0
GET	TC09 [LECTURA] Tipos de documento	66	0
GET	TC10 [LECTURA] Listar códigos IVA	102	0
GET	TC11 [ESCRITURA] Buscar cliente por identificación	80	0
POST	TC11 [ESCRITURA] Crear cliente	105	0
GET	TC11 [LECTURA] Listar clientes	193	0
GET	TC11 [PICO] Ráfaga clientes	347	0
GET	TC12 [LECTURA] Listar proveedores	142	0
GET	TC13 [ESCRITURA] Productos paginados	98	0
GET	TC13 [LECTURA] Listar productos	125	0
GET	TC14 [ESCRITURA] Facturas paginadas	115	0
POST	TC14 [ESCRITURA] Validación: factura inválida → 400	57	0

Método	Escenario de prueba	Solicitudes ejecutadas	Fallos
GET	TC14 [LECTURA] Listar facturas	227	0
GET	TC14 [PICO] Ráfaga facturas	494	0
POST	TC15 [ESCRITURA] Validación: nota crédito inválida → 400	42	0
GET	TC15 [LECTURA] Listar notas crédito	149	0
GET	TC15 [PICO] Ráfaga notas crédito	271	0
GET	TC16 [LECTURA] Listar notas débito	111	0
GET	TC17 [LECTURA] Listar guías de remisión	114	0
GET	TC17 [PICO] Ráfaga guías remisión	246	0
GET	TC18 [LECTURA] Listar liquidaciones de compra	104	0
GET	TC18 [PICO] Ráfaga liquidaciones	229	0
POST	TC19 [ESCRITURA] Validación: retención inválida → 400	54	0
GET	TC19 [LECTURA] Listar retenciones	120	0
GET	TC19 [PICO] Ráfaga retenciones	286	0
GET	TC20 [LECTURA] Listar notas de venta RISE	84	0
GET	TC20 [PICO] Ráfaga notas venta RISE	230	0

Método	Escenario de prueba	Solicitudes ejecutadas	Fallos
POST	TC21 [ESCRITURA] Validación: proforma inválida → 400	53	0
GET	TC21 [LECTURA] Listar proformas	108	0
GET	TC21 [PICO] Ráfaga proformas	207	0
GET	TC22 [LECTURA] Log SRI (documentos generados)	77	0
GET	TC24 [LECTURA] Listar recurrentes	64	0
GET	TC24 [PICO] Ráfaga recurrentes	135	0
GET	TC25 [ESCRITURA] Dashboard bajo carga	87	0
GET	TC25 [LECTURA] Dashboard KPIs	174	0
GET	TC25 [PICO] Ráfaga dashboard	455	0
POST	[SETUP] Login	1	0
TOTAL	**Aggregated**	**6463**	**0**

Auditoría de Ciberseguridad Defensiva Bajo el Estándar OWASP Top 10

Dado que la plataforma almacena datos financieros, certificados digitales y claves de acceso de múltiples empresas en una misma infraestructura, se realizó una auditoría de ciberseguridad defensiva basada en el estándar OWASP Top 10. Esta auditoría se ejecutó mediante revisión estática del código fuente y verificación de los controles arquitectónicos, con el objetivo de validar que las medidas de seguridad implementadas durante el desarrollo mitiguen eficazmente las vulnerabilidades más críticas y garanticen el aislamiento total de los datos de cada empresa.

Matriz de Mitigación y Cumplimiento de Riesgos OWASP

A continuación, en la Tabla *24* se presenta la matriz consolidada que evalúa los controles defensivos implementados en el núcleo de la plataforma (Backend en Node.js/Express y Frontend en Next.js), detallando el mecanismo técnico utilizado para verificar el cumplimiento de cada riesgo auditado:

Tabla 24

Controles de Seguridad Implementados y Validación Basada en OWASP Top 10 (2021)

Código OWASP	Categoría de Riesgo	Control Defensivo Implementado	Mecanismo de Verificación Técnica	Estado Final
A01:2021	Fallos en el Control de Acceso (Broken Access Control)	Implementación de aislamiento por tenant y control de roles mediante middleware.	Inspección de consultas y validación de accesos entre diferentes cuentas.	Mitigado
A02:2021	Fallos Criptográficos (Cryptographic Failures)	Cifrado de contraseñas con Bcrypt y protección de archivos de firma electrónica.	Auditoría de algoritmos criptográficos y almacenamiento seguro de credenciales.	Mitigado
A03:2021	Inyección (Injection)	Uso de consultas parametrizadas mediante ORM y validación de entradas.	Simulación de ataques de inyección SQL y análisis de puntos de entrada.	Mitigado
A05:2021	Configuración Insegura (Security Misconfiguration)	Implementación de Helmet.js, políticas CORS y gestión segura de variables de entorno.	Revisión de configuraciones del servidor y cabeceras HTTP.	Mitigado
A07:2021	Fallos en la Identificación y Autenticación (Identification and Authentication Failures)	Autenticación mediante JWT, cookies seguras y limitación de intentos de acceso.	Verificación de sesiones, almacenamiento seguro de tokens y pruebas de autenticación.	Mitigado

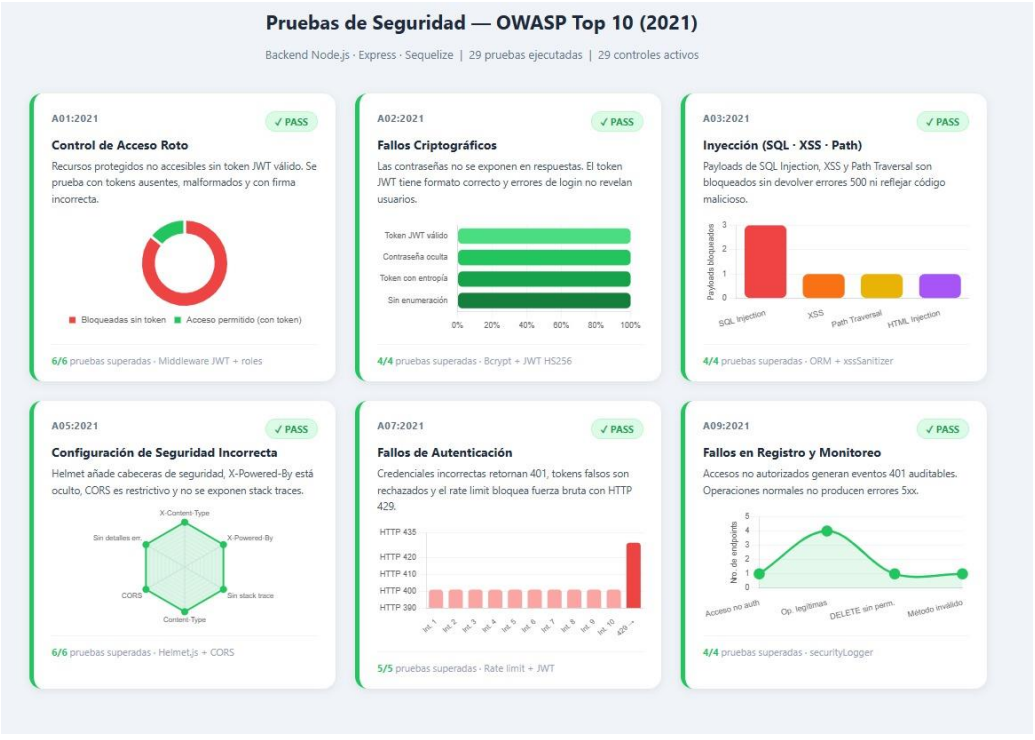
A09:2021	Fallos en el Registro y Monitoreo de Seguridad (Security Logging and Monitoring Failures)	Registro de eventos y auditoría de transacciones críticas del sistema.	Verificación de generación y almacenamiento de logs de auditoría.	Mitigado

Pruebas de seguridad

La Figura 25 muestra la ejecución de la batería de pruebas automatizadas de seguridad implementadas en Python para la validación de los controles asociados al estándar OWASP Top 10 (2021). Se observa que la totalidad de los casos definidos finalizaron con el estado PASSED, alcanzando un 100 % de éxito en la verificación de los mecanismos de protección implementados. Estos resultados confirman la efectividad de las medidas de seguridad adoptadas para mitigar los riesgos correspondientes a las categorías A01, A02, A03, A05, A07 y A09 del estándar OWASP.

Figura 25

Reporte consolidado de resultados de las pruebas de seguridad basadas en OWASP Top 10 (2021)



Análisis Detallado de Controles Críticos de Aislamiento Multi-tenant

Para demostrar la efectividad de la ciberseguridad defensiva en la capa lógica del sistema, se detallan las auditorías de código realizadas en los vectores de mayor impacto para el modelo de negocio:

Mitigación de A01:2021 - Control de Acceso y Secuestro de Datos Multi-tenant

El riesgo crítico en una arquitectura compartida es la vulnerabilidad IDOR: un usuario de la Empresa A podría alterar variables en la URL o en las peticiones HTTP para acceder a las facturas de la Empresa B, sin contar con autorización para ello.

- **Evidencia de Auditoría Defensiva:** El sistema no confía en los datos enviados desde el cliente. En su lugar, el tenant_id se extrae directamente del token JWT almacenado en una cookie httpOnly y se inyecta automáticamente en cada consulta de la base de datos, garantizando que ningún usuario pueda acceder a información fuera de su empresa.

Mitigación de A03:2021 - Prevención de Inyecciones de Código y SQL

Sin las debidas protecciones, un atacante podría manipular campos como descripciones de productos o nombres de clientes para inyectar comandos SQL maliciosos y acceder a la base de datos completa.

- **Evidencia de Auditoría Defensiva:** Se verificó que el sistema no utiliza consultas SQL construidas por concatenación de texto. Toda la persistencia se gestiona a través de Sequelize, que ejecuta consultas parametrizadas de forma automática, neutralizando cualquier intento de inyección SQL.

Mitigación de A07:2021 - Blindaje de Autenticación y Manejo de Sesión

El robo de credenciales o la interceptación de tokens de acceso comprometería la validez de las firmas digitales delegadas al sistema.

- **Evidencia de Auditoría Defensiva:** Se verificó que los tokens JWT no se almacenan en localStorage ni sessionStorage, ya que son vulnerables a ataques XSS. En su lugar, se utilizan cookies con la bandera httpOnly activada, impidiendo que scripts maliciosos puedan leer el token desde el navegador y evitando el robo de sesiones.

CAPÍTULO V

Conclusiones

- El análisis de la normativa del SRI permitió identificar 25 requerimientos funcionales y 15 no funcionales, distribuidos en 21 historias de usuario. La complejidad del ciclo tributario ecuatoriano hace inviable una solución local para las PYMES debido a la carga de mantenimiento que impone y a la subutilización de los recursos de una infraestructura dedicada, lo que justifica técnicamente la adopción del modelo SaaS multitenant.
- La arquitectura multitenant con PostgreSQL, Sequelize ORM y el identificador `empresa_id` garantizó el aislamiento lógico de datos entre empresas sin incrementar costos de infraestructura. Complementado con Apache como proxy inverso y Redis como caché, el sistema soporta múltiples empresas simultáneas bajo una única infraestructura, reduciendo significativamente los costos operativos.
- El sistema implementó los 25 requerimientos funcionales, cubriendo los siete tipos de comprobantes electrónicos del SRI con el estándar criptográfico XAdES-BES y la clave de acceso de 49 dígitos mediante el Algoritmo Módulo 11. Los seis sprints totalizaron 661 horas sin desviaciones, evidenciando la efectividad de la metodología Scrum en proyectos de alta complejidad técnica.
- La validación se realizó mediante pruebas de caja negra que verificaron los 25 requerimientos funcionales, complementadas con una auditoría OWASP Top 10 que confirmó seis controles críticos de seguridad, incluyendo control de acceso con JWT, cifrado de credenciales, prevención de inyección SQL y trazabilidad de auditoría, garantizando un entorno seguro para gestionar información financiera y certificados digitales de múltiples empresas. A partir de estos resultados, se proyecta como trabajo futuro la incorporación de inteligencia artificial en la plataforma, mediante un asistente conversacional o bot que oriente a los usuarios en la emisión de comprobantes y resuelva dudas frecuentes sobre el proceso tributario, automatizar tareas y detectar posibles inconsistencias en la facturación. De igual forma, se plantea la mejora continua del sistema mediante la optimización de sus módulos actuales, la incorporación de nuevos requerimientos que el SRI habilite, con el fin de mantener la plataforma actualizada y alineada con las necesidades de las empresas usuarias.

Recomendaciones

- **Implementar Row-Level Security (RLS) en PostgreSQL:** Esta medida añade una capa de protección independiente a la lógica de la aplicación, eliminando el riesgo de fuga de datos ante un eventual error de programación que omita el filtro por empresa_id. Al ser una función nativa de PostgreSQL, no requiere cambios en el stack tecnológico actual.
- **Sustituir el procesamiento asíncrono por una cola de mensajes dedicada:** Se recomienda migrar el procesamiento asíncrono actual hacia una cola de mensajes persistente con BullMQ sobre Redis, ya incluido en el stack del proyecto. Esto aportaría tres beneficios: persistencia de tareas ante reinicios del servidor, control granular de reintentos por tipo de error y visibilidad del estado de las colas desde el dashboard, fortaleciendo la trazabilidad de cada comprobante.
- **Comercialización como producto SaaS para PYMEs:** Los resultados del proyecto demuestran que la plataforma es viable para su comercialización bajo un modelo de suscripción mensual. Se recomienda definir precios escalonados según el volumen de comprobantes emitidos, incluyendo un nivel gratuito o de bajo costo para microempresas y personas naturales con RISE, convirtiéndola en una solución de impacto real para el ecosistema empresarial ecuatoriano.

Referencias:

[1] «Facturación Electrónica - intersri - Servicio de Rentas Internas». Accedido: 28 de mayo de 2026. [En línea]. Disponible en: https://www.sri.gob.ec/facturacion-electronica

[2] «Ecuador está avanzando hacia la vanguardia de la factura electrónica». Accedido: 28 de mayo de 2026. [En línea]. Disponible en: https://blog.groupseres.com/escalando-hacia-la-cima-de-la-factura-electronica-en-ecuador

[3] Siigo, «Obligados a facturar electrónicamente según el SRI», Siigo.com. Accedido: 28 de mayo de 2026. [En línea]. Disponible en: https://www.siigo.com/ec/blog/contabilidad-y-finanzas/obligados-a-facturar-electronicamente/

[4] «Facturación Electrónica en Ecuador: Guía Completa 2026», Factuplan. Accedido: 28 de mayo de 2026. [En línea]. Disponible en: https://factuplan.com.ec/blog/facturacion-electronica-ecuador-guia-completa

[5] «Factura electrónica y Guía de Remisión en Ecuador | EDICOM ES». Accedido: 28 de mayo de 2026. [En línea]. Disponible en: https://edicomgroup.com/es/factura-electronica/ecuador

[6] P. A. M. Burgos, R. P. R. Guevara, y M. G. P. Aguilar, «DESARROLLO DE UN SISTEMA WEB DE FACTURACIÓN ELECTRÓNICA».

[7] «Repositorio Digital UNACH: Software para facturación electrónica sobre plataforma Azure utilizando la metodología desarrollo rápido de aplicaciones (RAD)». Accedido: 21 de mayo de 2026. [En línea]. Disponible en: http://dspace.unach.edu.ec/handle/51000/12848

[8] M. R. C. Navarro, «DESARROLLO DE UN SOFTWARE DE FACTURACIÓN CON INTELIGENCIA ARTIFICIAL MEDIANTE EL USO DE MICROSERVICIOS EN LA NUBE DE AWS ENFOCADO PARA MICROEMPRESAS EN ECUADOR», 2024.

[9] É. L. Pincay Zavala, «SISTEMA WEB PARA LA GESTIÓN DE INFORMACIÓN EN COMERCIO CON FACTURACIÓN ELECTRÓNICA EN EL ALMACÉN SERVI CELL», bachelorThesis, Jipijapa-Unesum, 2023. Accedido: 21 de mayo de 2026. [En línea]. Disponible en: http://repositorio.unesum.edu.ec/handle/53000/4792

[10] «Begroup ec». Accedido: 21 de mayo de 2026. [En línea]. Disponible en: https://begroupec.com/

[11] «Estudio comparativo para la facturación electrónica entre ar». Accedido: 21 de mayo de 2026. [En línea]. Disponible en: https://redi.cedia.edu.ec/document/207716

[12] «Multi-Tenant Database Architecture Patterns Explained», Bytebase. Accedido: 21 de mayo de 2026. [En línea]. Disponible en: https://www.bytebase.com/blog/multi-tenant-database-architecture-patterns-explained/

[13] «The developer’s guide to SaaS multi-tenant architecture — WorkOS». Accedido: 21 de mayo de 2026. [En línea]. Disponible en: https://workos.com/blog/developers-guide-saas-multi-tenant-architecture

[14] P. Mell y T. Grance, «The NIST Definition of Cloud Computing».

[15] «Mastering Cloud Computing», ScienceDirect. Accedido: 22 de mayo de 2026. [En línea]. Disponible en: http://www.sciencedirect.com:5070/book/monograph/9780124114548/mastering-cloud-computing

[16] F. Chong, G. Carraro, y R. Wolter, «Multi-Tenant Data Architecture».

[17] «Comprobantes electrónicos - intersri - Servicio de Rentas Internas». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://www.sri.gob.ec/comprobantes-electronicos

[18] «Architectural Styles and the Design of Network-based Software Architectures». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm
[19] M. B. Jones, J. Bradley, y N. Sakimura, «RFC 7519: JSON Web Token (JWT)», IETF Datatracker. Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://datatracker.ietf.org/doc/html/rfc7519
[20] N. Provos y D. Mazières, «A Future-Adaptable Password Scheme».
[21] «Cryptography and Network Security: Principles and Practice (8th Edition) - eTextBookUniversity». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://etextbookuniversity.com/product/cryptography-and-network-security-principles-and-practice-8th-edition/
[22] European Telecommunications Standards Institute, *TS 101 903 - V1.4.2 - Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)*. 2010. Accedido: 22 de mayo de 2026. [En línea]. Disponible en: http://archive.org/details/etsi_ts_101_903_v01.04.02
[23] «SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://www.w3.org/TR/soap12-part1/
[24] «Redis in Action - Josiah Carlson», Manning Publications. Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://www.manning.com/books/redis-in-action
[25] claytonsiemens77, «Cache-Aside Pattern - Azure Architecture Center». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://learn.microsoft.com/en-us/azure/architecture/patterns/cache-aside
[26] «OWASP Top Ten Web Application Security Risks | OWASP Foundation». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://owasp.org/www-project-top-ten/
[27] «The ZAP Homepage», ZAP. Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://www.zaproxy.org/
[28] «Scrum Guide | Scrum Guides». Accedido: 26 de mayo de 2026. [En línea]. Disponible en: https://scrumguides.org/scrum-guide.html
[29] K. Peffers, T. Tuunanen, M. A. Rothenberger, y S. Chatterjee, «A Design Science Research Methodology for Information Systems Research», *Journal of Management Information Systems*, vol. 24, n.º 3, pp. 45-77, dic. 2007, doi: 10.2753/MIS0742-1222240302.
[30] «node-cron: Job scheduling for Node.js». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://www.npmjs.com/package/node-cron
[31] «Multer middleware». Express.js. Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://expressjs.com/en/resources/middleware/multer/
[32] «PDFKit: A JavaScript PDF generation library for Node and the browser». Accedido: 22 de mayo de 2026. [En línea]. Disponible en: https://pdfkit.org/