ÁREA ACADÉMICA DE TECNOLOGÍAS EMERGENTES INDUSTRIALES E INFORMÁTICAS

PROGRAMA ACADÉMICO DE LICENCIATURA EN INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN E INNOVACIÓN DIGITAL

TÉCNICO SUPERIOR UNIVERSITARIO EN DESARROLLO DE SOFTWARE MULTIPLATAFORMA

DESARROLLO DE UN MIDDLEWARE SAAS PARA LA GESTIÓN CENTRALIZADA DE MARKETPLACE

PARA LA EMPRESA

SERVICIOS EN DESARROLLOS INFORMÁTICOS DEL BAJÍO S.A. DE C.V.

PRESENTA

EVELYN MONTSERRAT MARTÍNEZ CARLOS

PARA OBTENER EL TÍTULO DE TÉCNICA SUPERIOR UNIVERSITARIA EN DESARROLLO DE SOFTWARE MULTIPLATAFORMA

ASESORA ACADÉMICA: ROXANA ZARICELL BAUTISTA LÓPEZ
ASESOR ORGANIZACIONAL: LEONARDO MARTÍNEZ OROZCO

GENERACIÓN: 2024 – 2026
LEÓN, GUANAJUATO. AGOSTO 2026

RESUMEN

Palabras clave: SaaS, gestión de pedidos, marketplaces, integración de APIs, comercio electrónico multicanal.

El presente informe expone el desarrollo de una plataforma SaaS para la gestión centralizada de pedidos en múltiples marketplaces, realizado en Servicios en Desarrollos Informáticos del Bajío (SDIB), León, Guanajuato. El proyecto tuvo como propósito centralizar la recepción, procesamiento y visualización de pedidos provenientes de distintos canales de venta digital mediante una interfaz unificada con control de acceso basado en roles. El sistema se desarrolló sobre el framework Laravel con base de datos MariaDB, integrando las APIs de Mercado Libre y TikTok Shop mediante webhooks y automatización con Cron Jobs, e incorporando el diseño técnico de las integraciones de Amazon y Shein para implementación futura. Como resultado, se obtuvo una plataforma funcional validada en entorno de producción, lista para su comercialización como servicio tecnológico escalable.

AGRADECIMIENTOS

Agradezco a Servicios en Desarrollos Informáticos del Bajío (SDIB) por brindarme la oportunidad de realizar mi estadía profesional y permitirme participar en el desarrollo de un proyecto que fortaleció mis conocimientos y experiencia en el área de desarrollo de software.

Expreso mi más sincero agradecimiento al Ing. Leonardo Martínez Orozco, asesor organizacional, por la orientación, el acompañamiento y los conocimientos compartidos durante el desarrollo del proyecto. Sus observaciones y recomendaciones fueron fundamentales para enriquecer mi formación profesional y contribuir al logro de los objetivos establecidos.

De igual manera, agradezco a mi asesora académica Roxana Bautista López por el seguimiento, apoyo y asesoría brindados durante la elaboración del presente Informe Final de Estadía. Sus observaciones y sugerencias fueron de gran importancia para mejorar la calidad y presentación de este trabajo.

Finalmente, agradezco a la Universidad Tecnológica de León por proporcionarme la formación académica y las herramientas necesarias para afrontar este reto profesional y concluir satisfactoriamente esta etapa de mi formación.

ÍNDICE

INTRODUCCIÓN

El presente informe describe el desarrollo de un sistema de tipo SaaS (Software as a Service) para la gestión centralizada de pedidos en múltiples marketplaces, realizado en Servicios en Desarrollos Informáticos del Bajío (SDIB). El proyecto se desarrolla en el Área de Desarrollo del Departamento de Sistemas como parte de una iniciativa orientada a ampliar el catálogo de soluciones tecnológicas de la organización.

El proyecto surge ante la necesidad de administrar de forma centralizada los pedidos generados en distintos marketplaces. La gestión independiente de cada canal de venta incrementa la complejidad operativa y dificulta el seguimiento de los pedidos, por lo que se plantea el desarrollo de una solución que optimice la administración multicanal.

Con este propósito, se desarrolla un sistema SaaS basado en Laravel para centralizar la gestión de pedidos mediante la integración funcional con Mercado Libre y TikTok Shop, además de generar las guías técnicas para la futura incorporación de Amazon y Shein dentro de una arquitectura escalable.

Para el desarrollo del proyecto se aplica una metodología organizada en etapas de análisis, diseño, implementación, integración, validación y documentación técnica, las cuales permiten estructurar el desarrollo del sistema y verificar el cumplimiento de los requerimientos establecidos.

Como resultado, se obtiene una plataforma SaaS funcional que centraliza la recepción, procesamiento y visualización de pedidos desde una interfaz unificada con control de acceso basado en roles. El informe integra los antecedentes, el marco teórico, la metodología, los resultados y las conclusiones del proyecto.

1. ANTECEDENTES

Servicios en Desarrollos Informáticos del Bajío (SDIB) es una microempresa dedicada al desarrollo de software y consultoría en tecnologías de la información, con sede en Calle Montreal #101, Interior 1, Colonia Villa Verde, León, Guanajuato, México, C.P. 37288. La empresa está constituida en la región del Bajío desde 2010 y se orienta a brindar soluciones tecnológicas a la medida para instituciones y empresas del sector privado.

La organización cuenta con un equipo de aproximadamente diez colaboradores, distribuidos en áreas funcionales entre las que se encuentra el Departamento de Sistemas, dentro del cual opera el Área de Desarrollo. Es en esta área donde se lleva a cabo el proyecto de estadía, cuyo propósito es diseñar y desarrollar una plataforma de tipo SaaS (Software as a Service) para la gestión centralizada de pedidos en múltiples marketplaces.

La misión de SDIB consiste en brindar soluciones tecnológicas a instituciones y empresas de acuerdo con sus necesidades particulares, haciendo más eficientes sus procesos. Su visión es consolidarse como referente en innovación tecnológica dentro de la zona Bajío, generando soluciones basadas en el equilibrio entre costo, beneficio y calidad. Entre los valores que rigen su operación se encuentran el respeto, la responsabilidad, el compromiso, la innovación y la integridad, entre otros (SDIB, s.f.).

En el contexto del comercio electrónico, el crecimiento sostenido de plataformas como Mercado Libre, Amazon, Shein y TikTok Shop ha generado una demanda creciente de herramientas que permitan a los vendedores administrar sus operaciones de manera integrada. Ante este escenario, SDIB identificó la oportunidad de ampliar su oferta de servicios mediante el desarrollo de una solución tecnológica propia, orientada a atender las necesidades de

clientes que operan de forma simultánea en varios canales de venta digital. La empresa no contaba previamente con un producto de este tipo en su catálogo, ni con proyectos anteriores en esta línea de desarrollo.

2. PROBLEMÁTICA

A través del análisis de las operaciones de comercio electrónico multicanal realizadas por clientes potenciales de SDIB, se detectó que un segmento considerable de vendedores digitales gestiona sus actividades de venta de forma simultánea en distintas plataformas, sin contar con una herramienta técnica que unifique dichas operaciones en un solo entorno. Esta fragmentación genera costos operativos elevados, duplicidad de procesos y errores en la logística de pedidos.

Desde el Área de Desarrollo de SDIB se identificó que la empresa no disponía de una solución tecnológica propia para ofrecer como servicio en este segmento de mercado, lo que representaba una limitación para captar clientes que demandan una gestión profesional y automatizada de sus ventas digitales. La ausencia de dicha solución implicaba, además, una pérdida de oportunidad de negocio frente a una necesidad de mercado en crecimiento.

La problemática fue delimitada internamente mediante el análisis de las necesidades del comercio electrónico en la región y la revisión de los requerimientos operativos de los clientes objetivo. Se determinó que la solución debía contar con una arquitectura escalable para la gestión centralizada de pedidos provenientes de distintos marketplaces, considerando plataformas como Mercado Libre, TikTok Shop, Amazon y Shein. No obstante, debido a que cada plataforma establece condiciones independientes de acceso, autenticación y uso de sus APIs, el proyecto se delimitó a la implementación funcional de las integraciones con Mercado Libre y TikTok Shop, mientras que para Amazon y Shein se dejó definida la estructura técnica, necesaria para su incorporación futura. De esta manera, se diseñó el sistema para centralizar la visualización y gestión del estatus de pedidos desde una sola interfaz con control de acceso por roles de usuario, conservando una arquitectura flexible que facilita incorporar o

descartar integraciones conforme a la viabilidad técnica y comercial de cada marketplace.

3. JUSTIFICACIÓN

El desarrollo de la plataforma SaaS de gestión de pedidos para marketplaces responde a una necesidad concreta identificada tanto en el mercado de comercio electrónico como en la oferta de servicios de SDIB. El modelo de software como servicio (SaaS) es una forma de entregar software a través de internet, donde el proveedor hospeda la aplicación y los clientes acceden mediante suscripción, sin necesidad de instalación local (Microsoft Azure, 2026). La implementación de esta solución permite a la empresa incorporar un producto tecnológico propio a su catálogo, ampliando su capacidad de captación de clientes en un segmento de alta demanda y con potencial de crecimiento en la región del Bajío.

Para los clientes que operan en múltiples plataformas de venta digital, la existencia de una herramienta centralizada puede contribuir a reducir los costos operativos asociados a la gestión fragmentada de pedidos, así como una disminución en los errores logísticos derivados del uso de sistemas desconectados. La plataforma proporciona visibilidad unificada sobre las operaciones de venta, lo que contribuye a una toma de decisiones más eficiente.

Desde el punto de vista técnico, el uso del framework Laravel y la validación funcional del sistema mediante pruebas de integración con entornos sandbox y verificación por consola garantizan la estabilidad, escalabilidad y confiabilidad del sistema entregado. Laravel es un framework de aplicaciones web con una sintaxis expresiva y elegante que proporciona características potentes como inyección de dependencias, una capa de abstracción de base de datos, colas, trabajos programados y herramientas de inspección del entorno de ejecución (Laravel, 2026a). La validación del sistema se llevó a cabo mediante dos estrategias complementarias: la verificación funcional de los procesos

internos a través de Laravel Tinker, consola interactiva nativa del framework que permite ejecutar y depurar lógica de negocio directamente sobre el entorno real de la aplicación; y las pruebas de integración con cuentas sandbox de Mercado Libre y TikTok Shop, mediante las cuales se simuló el ciclo de vida completo de los pedidos en condiciones equivalentes a producción. La arquitectura desacoplada adoptada facilita la incorporación futura de nuevas integraciones sin afectar el funcionamiento del servicio existente, lo que protege la inversión realizada y favorece la continuidad operativa de los clientes.

En conjunto, el proyecto fortalece la posición competitiva de SDIB en el mercado regional de tecnologías de la información, alineándose con su misión de hacer más eficientes los procesos de sus clientes y con su visión de ser referente en innovación dentro de la zona Bajío.

4. OBJETIVOS

4.1. OBJETIVO GENERAL

Desarrollar un sistema SaaS de gestión centralizada de pedidos para marketplaces en Servicios en Desarrollos Informáticos del Bajío (SDIB), mediante el framework Laravel, con integración funcional de las APIs de Mercado Libre y TikTok Shop, e investigación técnica de los mecanismos de integración de Amazon y Shein, como solución tecnológica escalable orientada a la administración multicanal de ventas digitales.

4.2. OBJETIVOS METODOLÓGICOS

4.2.1. Analizar los requerimientos funcionales y técnicos necesarios para el desarrollo del sistema de gestión multicanal de pedidos en SDIB.

4.2.2. Diseñar la arquitectura del sistema y la estructura de base de datos requeridas para la administración centralizada de pedidos.

4.2.3. Implementar los módulos de autenticación, administración de usuarios y control de permisos de la plataforma.

4.2.4. Desarrollar el dashboard centralizado para la visualización y administración del estatus de pedidos desde una interfaz unificada.

4.2.5. Integrar las APIs de Mercado Libre y TikTok Shop, e investigar los mecanismos de integración de Amazon y Shein.

4.2.6. Validar el funcionamiento de los procesos críticos del sistema mediante la ejecución de pruebas funcionales y de integración que evidencien el

correcto comportamiento de la lógica interna y de las integraciones implementadas.

4.2.7. Elaborar la documentación técnica del sistema, incluyendo el diccionario de datos, los manuales de instalación y el reporte de validación funcional.

4.3. ALCANCE

El proyecto comprende el diseño, desarrollo e integración de un sistema SaaS de gestión de pedidos para marketplaces, desarrollado sobre el framework Laravel con infraestructura en la nube (VPS/AWS), base de datos MariaDB y PHP 8.2+, dentro del área de desarrollo del departamento de sistemas de SDIB. AWS ofrece un amplio conjunto de servicios globales basados en la nube, que incluyen computación, almacenamiento, redes y más, los cuales ayudan a las organizaciones a moverse más rápido, reducir costos de TI y favorecer la escalabilidad (Amazon Web Services, 2026a). Su infraestructura se construye alrededor de Regiones y Zonas de Disponibilidad, donde cada Región contiene múltiples Zonas de Disponibilidad aisladas físicamente, lo que permite diseñar aplicaciones más tolerantes a fallos y escalables (Amazon Web Services, 2026b).

El sistema incluye un módulo de administración con gestión de roles y permisos, un dashboard centralizado para la visualización y gestión del estatus de pedidos, y la integración funcional de las APIs de Mercado Libre y TikTok Shop, limitada a la lectura y sincronización de pedidos. Como parte del diseño escalable de la plataforma, se realizó la investigación técnica de los mecanismos de integración de las APIs de Amazon y Shein, documentando los requisitos de acceso, autenticación y endpoints relevantes como base para su implementación futura. Se incluye además una validación funcional de los procesos críticos del

sistema mediante pruebas de integración con entornos sandbox y verificación por consola.

El proyecto no contempla la integración de marketplaces adicionales a los señalados, ni la gestión de inventarios, facturación, logística de envíos o atención a clientes. La implementación en entornos de producción para clientes finales de SDIB queda fuera del alcance de la estadía, ya que el resultado se entregará como sistema funcional validado, listo para su comercialización.

5. MARCO TEÓRICO

5.1. COMERCIO ELECTRÓNICO MULTICANAL

El comercio electrónico ha experimentado un crecimiento sostenido en los últimos años, impulsado por la proliferación de plataformas de venta en línea que permiten a los vendedores llegar a distintos segmentos de mercado de forma simultánea. La venta multicanal se define como la práctica de ofrecer productos a través de dos o más canales digitales de manera paralela, como pueden ser marketplaces, tiendas en línea propias o redes sociales con funcionalidades de comercio (Shopify en español, 2024). Este modelo amplía el alcance comercial, pero genera también la necesidad de administrar operaciones dispersas en distintos sistemas, lo que incrementa la complejidad operativa y el riesgo de errores en la gestión de pedidos. La centralización de estas operaciones mediante herramientas tecnológicas especializadas representa una respuesta directa a esta problemática.

En el contexto de este proyecto, este modelo multicanal genera la necesidad crítica de administrar operaciones dispersas en distintos sistemas, lo que incrementa la complejidad operativa y el riesgo de errores en la gestión de pedidos. Por lo tanto, la plataforma propuesta surge como una respuesta directa a esta problemática, orientada a centralizar y automatizar estas operaciones mediante una interfaz unificada que optimice el control de inventarios y ventas.

5.2. SOFTWARE AS A SERVICE (SaaS)

El modelo SaaS (Software as a Service) es una forma de distribución de software en la que las aplicaciones se alojan en la nube y se ponen a disposición de los usuarios a través de internet, sin necesidad de instalación local (Mell & Grance, 2011). Este modelo elimina la necesidad de gestionar infraestructura propia por parte del cliente, reduce los costos de implementación y permite la

actualización continua del sistema sin interrupciones en el servicio. Para la plataforma propuesta, el modelo SaaS resulta pertinente porque permite ofrecer el sistema de gestión de pedidos como un servicio comercializable y accesible para distintos clientes sin que cada uno requiera una instalación independiente, lo cual favorece la escalabilidad del producto.

Para la solución tecnológica en desarrollo, la adopción del modelo SaaS resulta idónea debido a que permite comercializar el sistema de gestión de pedidos como un servicio accesible y bajo demanda para múltiples clientes independientes. Esto optimiza la escalabilidad del producto, reduce las barreras de entrada económicas para los usuarios y facilita el mantenimiento centralizado del software sin requerir intervenciones locales.

5.3. FRAMEWORK LARAVEL Y ARQUITECTURA MVC

Laravel es un framework de desarrollo de aplicaciones web para PHP que se caracteriza por una sintaxis expresiva y elegante. De acuerdo con su documentación oficial, Laravel provee la base necesaria para que los desarrolladores se concentren en construir funcionalidades sin preocuparse por aspectos repetitivos del desarrollo (Laravel, 2026a). El framework implementa el patrón de arquitectura MVC (Modelo-Vista-Controlador), que separa la lógica de negocio, la presentación y el control de flujo de datos en componentes independientes. Esta separación facilita el mantenimiento del código, permite la reutilización de componentes y reduce el acoplamiento entre las distintas capas del sistema. Para la solución en desarrollo, Laravel fue seleccionado por su capacidad para manejar grandes volúmenes de datos, su ecosistema de herramientas integradas para autenticación, gestión de rutas y conexión con bases de datos, así como su facilidad para integrar APIs externas de forma segura y estructurada.

5.4. APIS REST E INTEGRACIÓN DE MARKETPLACES

Una API (Application Programming Interface) es un conjunto de definiciones y protocolos que permite la comunicación entre distintos sistemas de software. Las APIs de tipo REST (Representational State Transfer) utilizan el protocolo HTTP y operan sobre recursos identificados mediante URLs, empleando métodos estándar como GET, POST, PUT y DELETE para realizar operaciones sobre los datos. Este tipo de arquitectura es ampliamente adoptado en plataformas de comercio electrónico por su simplicidad, escalabilidad y compatibilidad con distintos lenguajes y entornos de desarrollo. Plataformas como Mercado Libre exponen sus funcionalidades a través de APIs REST que permiten acceder a recursos como pedidos, publicaciones, usuarios y envíos desde aplicaciones externas (MercadoLibre S.R.L., 2026). De acuerdo con la documentación técnica de Mercado Libre, su API REST sigue una estructura de endpoints basada en recursos y utiliza OAuth 2.0 como protocolo de autenticación para autorizar aplicaciones de terceros (MercadoLibre S.R.L., 2026). La integración de múltiples APIs dentro de una arquitectura desacoplada, como la planteada en este desarrollo, permite que cada marketplace opere como un servicio independiente, de modo que la incorporación futura de nuevas plataformas no afecte el funcionamiento de las ya existentes.

5.5. BASES DE DATOS RELACIONALES Y MARIADB

Una base de datos relacional es un sistema que organiza la información en tablas compuestas por filas y columnas, donde las relaciones entre tablas se establecen mediante campos comunes. Este modelo permite mantener la integridad de los datos y realizar consultas complejas mediante el lenguaje SQL (Structured Query Language). MariaDB es un sistema gestor de bases de datos relacionales de código abierto, desarrollado por los creadores originales de MySQL, reconocido por su velocidad, escalabilidad y robustez en una amplia

variedad de aplicaciones (MariaDB Documentation, 2025). Es compatible con múltiples motores de almacenamiento y plugins, y cuenta con soporte para funcionalidades avanzadas como búsqueda vectorial y compatibilidad con Oracle y MySQL. En el contexto del diseño de la arquitectura, MariaDB fue seleccionado como motor de base de datos por su compatibilidad nativa con Laravel, su rendimiento en entornos de producción y su disponibilidad en infraestructuras de nube como AWS.

5.6. INFRAESTRUCTURA EN LA NUBE

La computación en la nube hace referencia al suministro bajo demanda de recursos de TI, tales como capacidad de cómputo, almacenamiento, redes y bases de datos, a través de internet, bajo un modelo de precios de pago por uso que elimina la necesidad de adquirir, asegurar y mantener hardware físico de manera local (Amazon Web Services, 2026c). Amazon Web Services (AWS) es uno de los proveedores de infraestructura en la nube más utilizados a nivel global.

La infraestructura global de AWS está construida conceptualmente en torno a regiones y zonas de disponibilidad; una región se define como una ubicación física en el mundo que contiene múltiples zonas de disponibilidad, las cuales consisten en uno o más centros de datos discretos que cuentan con alimentación eléctrica, refrigeración y conectividad de red redundantes alojados en instalaciones independientes (Amazon Web Services, 2026d). Esta arquitectura garantiza niveles elevados de alta disponibilidad, tolerancia a fallos y aislamiento ante contingencias, lo cual resulta crítico para mitigar la interrupción del servicio en una plataforma SaaS orientada a la gestión continua y en tiempo real de pedidos.

Adicionalmente, un Servidor Privado Virtual (VPS, por sus siglas en inglés) constituye una alternativa de infraestructura que ofrece un entorno virtualizado

con recursos asignados y dedicados dentro de un servidor físico compartido, proporcionando un mayor control sobre la configuración del sistema y de red en comparación con el alojamiento compartido tradicional. El despliegue de la solución tecnológica se contempla sobre infraestructura VPS o AWS, dependiendo de las condiciones operativas, requerimientos de escalabilidad y políticas financieras requeridas por la organización.

5.7. VALIDACIÓN FUNCIONAL Y PRUEBAS DE INTEGRACIÓN

La validación del comportamiento correcto de un sistema de software puede llevarse a cabo mediante distintos niveles de prueba, desde la verificación aislada de unidades individuales de código hasta la comprobación del flujo completo de operaciones en un entorno que replica las condiciones reales de producción. Para la validación de la plataforma desarrollada se adoptaron dos estrategias complementarias: la verificación funcional por consola mediante Laravel Tinker y las pruebas de integración con entornos sandbox de los marketplaces conectados.

Laravel Tinker es una consola interactiva de tipo REPL (Read-Eval-Print Loop) integrada de forma nativa en el framework, que permite ejecutar código PHP arbitrario dentro del contexto completo de la aplicación, con acceso directo a todos sus modelos, servicios, clases y conexiones a base de datos (Tinkerwell, s. f.). A diferencia de las pruebas automatizadas, Tinker permite inspeccionar el comportamiento de componentes específicos de forma inmediata, observar los valores retornados por cada método y verificar el estado resultante de la base de datos tras cada operación. Esta característica lo convierte en una herramienta idónea para la validación funcional de operaciones CRUD, lógica de negocio y servicios de integración durante la fase de desarrollo.

Las pruebas de integración, por su parte, consisten en la verificación del comportamiento del sistema ante condiciones que replican el entorno real de operación, evaluando no solo componentes individuales sino la interacción completa entre módulos internos y servicios externos. En el contexto de la plataforma desarrollada, esta validación se realizó mediante cuentas de prueba proporcionadas por Mercado Libre y TikTok Shop, las cuales permiten simular el ciclo de vida completo de un pedido, desde su generación en la plataforma externa hasta su recepción, procesamiento y persistencia en el sistema, sin afectar datos de producción. Este nivel de prueba es especialmente relevante para validar la correcta operación de los flujos de webhooks, la autenticación OAuth 2.0 y la sincronización de información entre plataformas.

5.8. JSON WEB TOKEN (JWT)

JSON Web Token (JWT) es un estándar abierto definido en el RFC 7519 del Internet Engineering Task Force (IETF) que establece un método compacto y auto contenido para transmitir información de forma segura entre dos partes mediante un objeto JSON (Jones et al., 2015). Esta información puede verificarse y ser objeto de confianza porque está firmada digitalmente. Los JWT pueden firmarse utilizando un secreto mediante el algoritmo HMAC, o mediante un par de claves pública y privada usando RSA o ECDSA. En la arquitectura del sistema, JWT se utiliza como mecanismo de autenticación en el API Gateway, permitiendo verificar la identidad de los usuarios y servicios que realizan solicitudes a la plataforma sin necesidad de consultar la base de datos en cada petición, lo que contribuye a la eficiencia y escalabilidad de la autenticación en una estructura de servicios desacoplados.

Dentro de la arquitectura de micro servicios o servicios desacoplados de la plataforma, JWT se implementa formalmente como el mecanismo de autenticación en el API Gateway. Su aplicación permite verificar de manera

segura y eficiente la identidad de los usuarios y servicios en cada petición sin saturar la base de datos relacional, garantizando así un alto rendimiento y la escalabilidad del sistema de control de acceso.

5.9. SISTEMA DE COLAS DE TRABAJO Y AUTOMATIZACIÓN CON CRON JOBS

El procesamiento de pedidos provenientes de marketplaces externos implica operaciones de duración variable que, si se ejecutaran de forma síncrona durante la atención de una solicitud HTTP entrante, incrementarían el tiempo de respuesta del servidor y elevarían el riesgo de pérdida de datos ante intermitencias de red. Para mitigar esta problemática, Laravel incorpora de forma nativa un sistema de colas de trabajo (queue system) que permite diferir la ejecución de tareas intensivas a un proceso independiente del ciclo de vida de la petición web (Laravel, 2026b).

El sistema de colas de Laravel opera mediante el concepto de jobs: clases PHP encapsuladas que contienen la lógica de una tarea diferible y que son serializadas y persistidas en un medio de almacenamiento configurable denominado driver. Para entornos de producción sin infraestructura de mensajería dedicada, Laravel provee el database queue driver, que utiliza una tabla relacional (jobs) dentro de la base de datos existente como mecanismo de persistencia de la cola. Este enfoque elimina dependencias externas, garantiza la durabilidad de los mensajes ante reinicios del servidor y facilita el despliegue en infraestructuras de alojamiento compartido o VPS con acceso limitado a servicios adicionales.

El procesamiento de los jobs encolados se lleva a cabo mediante el comando php artisan queue: work, que instancia un trabajador (worker) encargado de consultar la tabla de cola, extraer el siguiente trabajo pendiente y

ejecutarlo. La variante --stop-when-empty instruye al trabajador para que finalice su ejecución una vez que la cola quede vacía, lo que resulta apropiado en escenarios donde el proceso es invocado de forma periódica en lugar de mantenerse activo de forma continua.

La automatización de dicha invocación periódica se delega a los Cron Jobs, una funcionalidad del sistema operativo Unix/Linux que permite programar la ejecución recurrente de comandos en intervalos definidos. En entornos de alojamiento web administrados, esta funcionalidad se expone al usuario a través del panel de control cPanel, que provee una interfaz gráfica para configurar las expresiones de temporización (cron expressions) sin requerir acceso directo al sistema operativo subyacente. Mediante esta integración, es posible establecer que el procesador de cola sea invocado automáticamente cada cierto intervalo, por ejemplo, cada minuto, garantizando la atención oportuna de los jobs pendientes sin intervención manual (Laravel, 2026c).

En el contexto de la plataforma desarrollada, el sistema de colas con database driver y Cron Jobs de cPanel cumple dos funciones operativas críticas: el procesamiento asíncrono de los webhooks entrantes de Mercado Libre y TikTok Shop mediante el job ProcessWebhookJob, ejecutado cada 60 segundos; y la renovación automatizada de los tokens de acceso OAuth 2.0 de cada plataforma mediante el comando RefreshPlatformTokens, programado a intervalos diferenciados según los tiempos de expiración establecidos por cada proveedor.

5.10. TAILWIND CSS

Tailwind CSS es un framework de CSS basado en el enfoque utility-first, diseñado para el desarrollo rápido de interfaces web modernas (Tailwind Labs Inc., 2026). A diferencia de los frameworks tradicionales, Tailwind proporciona un

conjunto de clases utilitarias predefinidas que pueden combinarse directamente en el marcado HTML para construir cualquier diseño sin necesidad de escribir hojas de estilo personalizadas. Este enfoque reduce el cambio de contexto entre archivos de HTML y CSS durante el desarrollo, y permite mantener la coherencia visual a través de un sistema de diseño configurable. En el desarrollo de la interfaz, Tailwind CSS se utiliza en conjunto con Laravel para construir los componentes visuales del dashboard centralizado, facilitando la creación de una experiencia intuitiva, responsiva y con una estructura de información clara para los usuarios de la plataforma.

5.11. WEBHOOKS Y COMUNICACIÓN BASADA EN EVENTOS

En los sistemas de integración con plataformas externas, existen dos estrategias fundamentales para la recepción de información: la consulta periódica (polling), en la que el sistema cliente interroga activamente al servidor externo a intervalos regulares; y la notificación basada en eventos, en la que es el servidor externo quien informa proactivamente al sistema receptor cuando ocurre un evento relevante. Esta segunda estrategia se instrumenta mediante los denominados webhooks: mecanismos de callback HTTP mediante los cuales una plataforma externa envía una solicitud POST al endpoint configurado por el sistema receptor en el momento exacto en que se produce un evento, como la creación o actualización de un pedido (Fowler, 2017).

La adopción de webhooks en la arquitectura de integración con marketplaces ofrece ventajas operativas determinantes: elimina la latencia inherente al polling periódico, reduce el consumo innecesario de cuotas de API y garantiza que cada evento sea notificado de forma inmediata sin que el sistema deba mantener un proceso de consulta activo de manera continua. Plataformas como Mercado Libre y TikTok Shop exponen este mecanismo como su canal principal para la notificación de eventos de pedido, requiriendo que el sistema

receptor registre previamente una URL pública accesible desde internet (MercadoLibre S.R.L., 2026; TikTok Shop Partner Center, 2025).

Dado que los webhooks son iniciados externamente de forma asíncrona y pueden llegar en ráfagas de alta concurrencia, su procesamiento no puede ejecutarse de forma síncrona dentro del ciclo de vida de la petición HTTP. La práctica estándar consiste en acusar recibo inmediato con un código HTTP 200 y delegar el procesamiento real a un sistema de colas en segundo plano, desacoplando así la disponibilidad del endpoint receptor de la duración variable de las operaciones de integración.

5.12. PROTOCOLO DE AUTORIZACIÓN OAUTH 2.0

OAuth 2.0 es un protocolo de autorización estándar de la industria que permite a aplicaciones de terceros obtener acceso limitado a los recursos de un servicio en nombre de un usuario, sin que dicho usuario deba compartir sus credenciales directamente con la aplicación solicitante. El protocolo define un flujo de delegación de autorización en el que el propietario del recurso concede un permiso explícito, como resultado del cual la aplicación recibe un access token de vigencia limitada y un refresh token de mayor duración destinado a la renovación automatizada de las credenciales (MercadoLibre S.R.L., 2026).

El flujo estándar de OAuth 2.0 para integraciones con marketplaces implica tres etapas diferenciadas: en primer lugar, la redirección del usuario al servidor de autorización de la plataforma externa para que otorgue el consentimiento; en segundo lugar, la recepción de un código de autorización temporal que el sistema intercambia por el par de tokens mediante una solicitud al servidor de recursos; y en tercer lugar, el uso del access token en los encabezados HTTP de cada petición subsecuente a la API, acompañado de la lógica de renovación automática cuando dicho token expira.

En la plataforma desarrollada, el protocolo OAuth 2.0 constituye el mecanismo de autenticación para las integraciones con Mercado Libre y TikTok Shop. Dado que cada plataforma establece tiempos de expiración distintos para sus tokens de acceso, el sistema implementa una rutina de renovación automatizada programada mediante Cron Jobs, garantizando la disponibilidad continua de credenciales válidas sin intervención manual del operador.

5.13. PATRONES DE DISEÑO: FACTORY E INTERFACES

Los patrones de diseño son soluciones reutilizables y documentadas a problemas recurrentes en el diseño de software orientado a objetos. Su adopción dentro de una arquitectura permite estandarizar la forma en que los componentes del sistema se relacionan entre sí, favoreciendo la extensibilidad, la mantenibilidad y la reducción del acoplamiento entre módulos.

El patrón Factory (fábrica) es un patrón de creación cuyo propósito es delegar la responsabilidad de instanciar objetos a una clase centralizada, en lugar de hacerlo directamente en el punto de uso. Mediante este patrón, el código que necesita un objeto de determinado tipo no necesita conocer cuál implementación concreta está siendo creada, sino que solicita a la fábrica el objeto correspondiente según un criterio de selección en tiempo de ejecución. Esto resulta especialmente valioso en sistemas que deben gestionar múltiples variantes de un mismo concepto, en este caso, múltiples marketplaces, ya que la incorporación de una nueva variante solo requiere registrarla en la fábrica sin modificar el código que la consume.

Las Interfaces son contratos estructurales que definen un conjunto de métodos que cualquier clase que los implemente debe proveer obligatoriamente, sin dictar la forma en que dichos métodos operan internamente. Su función en una arquitectura desacoplada es establecer un límite claro entre lo que el sistema

central espera de una integración y los detalles particulares de cada plataforma. Al programar contra interfaces en lugar de implementaciones concretas, el núcleo del sistema permanece estable ante la incorporación o sustitución de componentes externos, ya que cualquier nueva integración que respete el contrato definido será automáticamente compatible con la lógica central sin requerir cambios en ella.

En la plataforma desarrollada, la combinación de ambos patrones, junto con el patrón Strategy implícito en la selección dinámica del servicio de procesamiento, constituye el fundamento de la extensibilidad de la arquitectura de integración con marketplaces, permitiendo que la habilitación de nuevas plataformas como Amazon o Shein se realice mediante la adición de nuevas implementaciones de las interfaces existentes y su registro en las fábricas correspondientes, sin necesidad de modificar el núcleo del sistema.

6. METODOLOGÍA

6.1. ANÁLISIS DE REQUERIMIENTOS FUNCIONALES Y TÉCNICOS

El análisis de los requerimientos necesarios para el desarrollo del sistema de gestión multicanal de pedidos se ejecutó mediante la revisión exhaustiva de las necesidades operativas de la organización. A partir de este proceso, se identificaron y clasificaron los componentes funcionales, técnicos y de arquitectura indispensables para garantizar la consistencia, seguridad y escalabilidad en la administración de ventas procedentes de múltiples plataformas de comercio electrónico.

6.1.1. Requerimientos Funcionales

- **Sincronización en Tiempo Real / Near Real-Time:** Se requiere la captura automática de pedidos provenientes de los distintos canales de venta de forma automatizada. Para plataformas como Mercado Libre y TikTok Shop, el proceso se gestionará mediante webhooks, permitiendo la transferencia inmediata de datos tras la ocurrencia de eventos en los servidores de origen. Para entornos basados en consultas asíncronas periódicas o arquitecturas de gran escala como Shein y la API de socios comerciales de Amazon, se estableció una estrategia de consulta periódica (polling) eficiente basada en sus especificaciones técnicas de integración (Amazon Selling Partner API, 2024).
- **Mapeo de Estados Unificado:** Se determinó la necesidad de diseñar un mecanismo de homologación arquitectónico que traduzca de forma transparente los estados nativos de cada marketplace (tales como paid, shipped o delivered) a un flujo

interno, centralizado y estandarizado dentro del catálogo de estatus propio de la organización.

- **Control de Acceso Basado en Roles (RBAC):** Se definió un esquema de seguridad perimetral interna mediante la asignación de roles diferenciados a nivel de software. Este modelo restringe y audita las facultades dentro de la plataforma, delimitando funciones específicas para los perfiles de Administradores y Operadores de Almacén bajo el principio de privilegios mínimos (Lindemulder, s. f.).

6.1.2. Requerimientos Técnicos

- **Arquitectura Dirigida por Eventos (EDA):** Con el propósito de mitigar la pérdida de información ante ráfagas altas de peticiones simultáneas, se especificó el uso de una arquitectura orientada a eventos. Esta se implementó mediante el componente de colas nativo del framework Laravel, utilizando el controlador de persistencia en base de datos (database queue driver). Las solicitudes entrantes se encolarán de manera asíncrona; de este modo, si la API de un marketplace externo experimenta intermitencias o el servidor local sufre cargas elevadas, los payloads de los pedidos permanecen persistidos de forma segura en espera de ser procesados secuencialmente.
- **Estrategia de Idempotencia:** Para evitar la duplicidad de registros en la base de datos ante fallas de red, latencia HTTP o reintentos duplicados involuntarios de un mismo webhook, se definió un algoritmo de procesamiento único y restrictivo basado estrictamente en la validación del identificador único del pedido en el origen (order_id) (Google Cloud, s. f.).
- **Seguridad de los Datos:** Se determinó la obligatoriedad de proteger las credenciales críticas de conexión de las tiendas

enlazadas. Por consiguiente, se estableció el uso del estándar de cifrado avanzado AES-256 para el almacenamiento seguro dentro de la base de datos de los tokens de acceso y actualización (access_token y refresh_token), evitando la exposición de información sensible ante accesos no autorizados al repositorio de datos (National Institute of Standards and Technology, 2001).

6.2. DISEÑO DE LA ARQUITECTURA DEL SISTEMA Y ESTRUCTURA DE BASE DE DATOS

Con base en los requerimientos identificados, se diseñó la arquitectura del sistema bajo un enfoque desacoplado orientado a la separación de responsabilidades, de manera que la modificación o caída eventual de una integración no afecte el funcionamiento del resto de la plataforma.

Como se ilustra en la Imagen 1, la arquitectura definida establece un punto de entrada central mediante el API Gateway, componente responsable del control de acceso, la autenticación mediante JWT y el direccionamiento de solicitudes hacia los módulos internos del sistema. A partir de dicho punto, el procesamiento de pedidos se delega al componente central denominado Order Processor, encargado de consumir la cola de mensajería, procesar, validar y persistir los pedidos en la base de datos. Las integraciones con marketplaces, por su parte, se gestionaron mediante componentes independientes especializados por canal (Integration Workers), cada uno responsable de comunicarse con la API externa de su marketplace correspondiente y transformar la carga útil (payload) recibida al formato estándar de la organización.

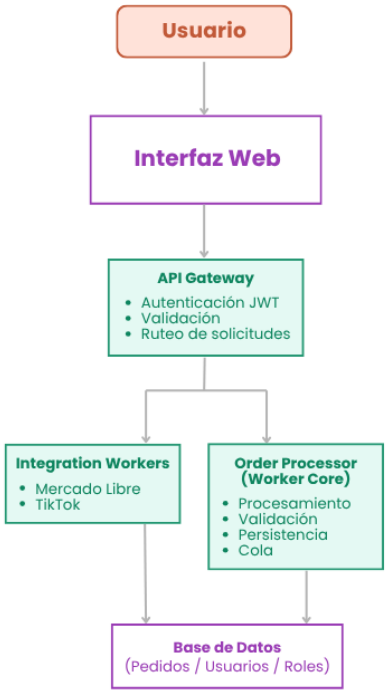

Imagen 1. Arquitectura general del sistema

Para la estructura de datos, se diseñó el modelo relacional de la base de datos considerando las entidades fundamentales para soportar la operación multicanal del sistema, garantizando la consistencia e integridad referencial de la información. El diagrama relacional de la base de datos se omite del presente informe debido a restricciones de formato derivadas de las dimensiones del esquema, cuya inclusión fragmentada comprometería la legibilidad de las relaciones entre entidades. Las relaciones entre tablas pueden consultarse a través de los índices de llaves foráneas documentados en la columna "Otros" del diccionario de datos, el cual se presenta en el Anexo 1.

6.3. IMPLEMENTACIÓN DE MÓDULOS DE AUTENTICACIÓN, ADMINISTRACIÓN DE USUARIOS Y CONTROL DE PERMISOS

La implementación del módulo de autenticación se llevó a cabo considerando el flujo de validación ilustrado en la Imagen 2. El proceso inicia con la captura de credenciales por parte del usuario; posteriormente, el módulo valida la información mediante los mecanismos de autenticación provistos por Laravel, lo que asegura la gestión controlada de las sesiones. Una vez validado el acceso, el sistema consulta los permisos asociados al rol correspondiente para habilitar únicamente las funcionalidades autorizadas, garantizando que únicamente el personal autorizado acceda a la información de la plataforma.

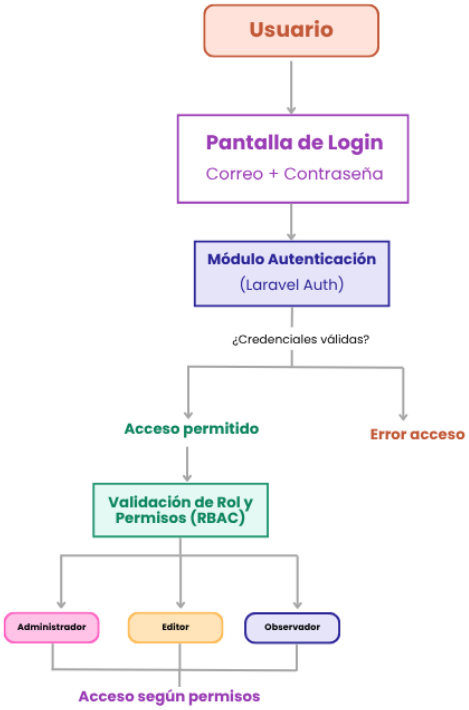

Imagen 2. Flujo de autenticación y control de permisos

Asimismo, se desarrolló un módulo de administración de usuarios que faculta el registro, consulta, modificación y desactivación de cuentas dentro de la

plataforma, de acuerdo con las necesidades operativas del equipo técnico. Adicionalmente, se configuró un esquema de control de acceso basado en roles y permisos (RBAC), mediante el cual se restringen o habilitan funcionalidades específicas de acuerdo con las responsabilidades asignadas a cada perfil de usuario. Los roles definidos corresponden a los perfiles de Administrador y Operador, disponiendo cada uno de accesos diferenciados a los componentes del sistema.

6.4. DESARROLLO DEL DASHBOARD CENTRALIZADO

El desarrollo de un dashboard centralizado tuvo como propósito proporcionar una visualización consolidada de la información más relevante del sistema. Para su construcción se utilizaron componentes visuales desarrollados con Laravel y Tailwind CSS, permitiendo la presentación de indicadores y datos operativos mediante una interfaz intuitiva y accesible desde una pantalla única en la Imagen 3 se encuentra el Wireframe del dashboard.

Durante el diseño de la interfaz se priorizó la experiencia del usuario mediante la organización de la información en secciones claramente identificables, facilitando el monitoreo continuo y el seguimiento ágil de los procesos administrados por la plataforma, el cual se puede apreciar el flujo de navegación del sistema en la Imagen 4.

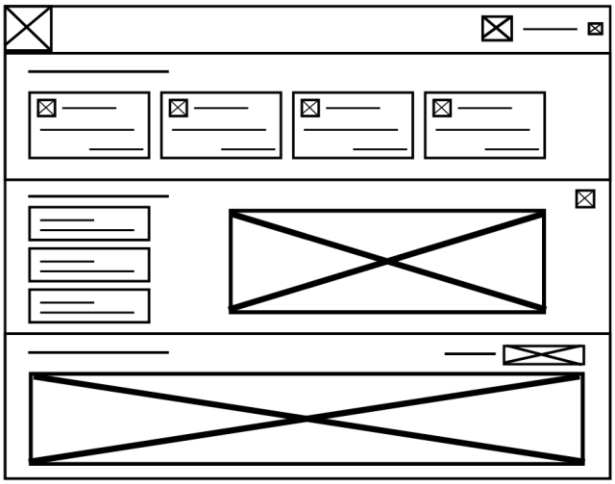

Imagen 3. Wireframe del Dashboard

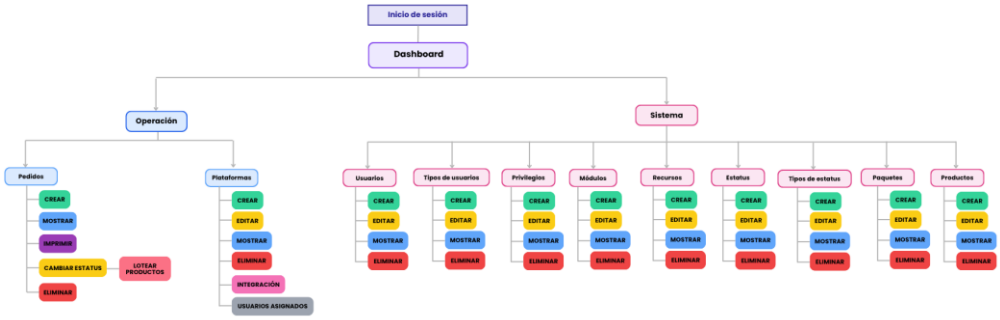

Imagen 4. Flujo de navegación del sistema

6.5. INTEGRACIÓN DE LAS API'S PARA LA SINCRONIZACIÓN DE INFORMACIÓN DE PEDIDOS

Para dar cumplimiento al objetivo específico orientado a integrar las APIs de Mercado Libre, Amazon, Shein y TikTok, se desarrolló un componente de conectividad unificado. Si bien la arquitectura base y los flujos operativos se validaron inicialmente mediante la integración con Mercado Libre, el diseño fue concebido de forma genérica para permitir la incorporación homogénea de los esquemas de comunicación asíncrona (webhooks) y por consultas periódicas (polling) requeridos por las demás plataformas comerciales.

6.5.1. Análisis previo y requisitos de conexión

Previo a la fase de codificación, se realizó un análisis riguroso de la documentación técnica oficial de los portales para desarrolladores de cada marketplace. Este diagnóstico permitió identificar las siguientes variables operativas críticas:

- Los métodos de autenticación y los flujos de autorización OAuth 2.0, así como los métodos getAccessToken () y refreshToken () que cualquier servicio de plataforma debe implementar.
- Los alcances y permisos necesarios para la lectura y seguimiento del ciclo de vida de los pedidos.
- La disponibilidad de eventos en tiempo real mediante webhooks HTTP POST.
- La estructura y el esquema de datos (payload) de los objetos de pedido devueltos por cada API.
- Las cuotas de consumo, límites de peticiones (rate limits) y políticas de renovación de credenciales de cada proveedor.

A partir de este análisis, se determinó la inconveniencia de construir conexiones rígidas o acopladas, optando por el diseño de una arquitectura genérica y escalable capaz de absorber las variaciones en los mecanismos de autenticación, firma y estructura de datos de cada plataforma sin afectar el núcleo del sistema. La Tabla 1 resume los esquemas de comunicación identificados para cada plataforma.

Tabla 1. Métodos de comunicación por marketplace

Marketplace	Autenticación	Notificación de eventos	Formato de datos
Mercado Libre	OAuth 2.0	Webhooks (HTTP POST)	JSON
TikTok Shop	OAuth 2.0	Webhooks (HTTP POST)	JSON
Amazon SP-API	OAuth 2.0 + LWA	Polling / Notificaciones	JSON
Shein	API Key	Polling	JSON

6.5.2. Diseño de la arquitectura de integración

Se diseñó e implementó una arquitectura desacoplada organizada en cuatro capas operativas. La Tabla 2 describe los componentes principales que integran esta arquitectura.

Tabla 2. Componentes de la arquitectura de integración

Componente	Función en la arquitectura de integración
WebhookController	Punto de entrada HTTP. Recibe las notificaciones POST externas, invoca la WebhookFactory para identificar la plataforma de origen y delega inmediatamente el procesamiento.
WebhookFactory	Identifica en tiempo de ejecución qué plataforma originó el evento entrante y dirige el flujo al validador y servicio correspondiente.
MercadoLibreSignatureValidator	Intercepta la petición e inspecciona los encabezados criptográficos para verificar que la notificación proviene de servidores legítimos de Mercado Libre. Si la firma no coincide, bloquea el proceso.
ProcessWebhookJob	Job despachado a la cola de Laravel (tabla jobs). Desacopla el procesamiento del pedido de la conexión HTTP, permitiendo devolver un código 200 OK de forma inmediata al marketplace.
MercadoLibreWebhookService	Extrae los datos del payload (ID del pedido, cliente, SKU, cantidades), realiza las llamadas de enriquecimiento a la API y mapea el resultado al modelo unificado del sistema.
MercadoLibreApiService	Cliente HTTP base que centraliza las peticiones a la API de Mercado Libre, inyectando automáticamente los encabezados de autenticación mediante el token activo.
TikTokSignatureValidator	Valida firma HMAC-SHA256 del encabezado x-tts-timestamp
TiktokWebhookService	Extrae los datos del payload (ID del pedido, cliente, SKU, cantidades), realiza las llamadas de enriquecimiento a la API y mapea el resultado al modelo unificado del sistema.
TiktokApiService	Cliente HTTP base que centraliza las peticiones a la API de Mercado Libre, inyectando automáticamente los encabezados de autenticación mediante el token activo.

La capa de autenticación y conexión base (Services\Platforms) centraliza los mecanismos para obtener, usar y renovar las credenciales OAuth 2.0. AuthServiceInterface define el contrato que cualquier servicio de plataforma debe implementar, con los métodos getAccessToken() y refreshToken(). La AuthFactory aplica el patrón Factory: en lugar de decidir en cada controlador a qué plataforma conectarse, el controlador le solicita a la Factory la instancia correcta pasándole el nombre del canal. MercadoLibreAuthService y TiktokAuthService son las implementaciones concretas que contienen la lógica, los endpoints y las firmas requeridas por cada API. Los clientes HTTP base MercadoLibreApiService y TiktokApiService centralizan todas las peticiones GET, POST y PUT hacia los servidores externos, inyectando automáticamente los encabezados de autenticación con el token activo.

La capa de recepción de pedidos y seguridad (Services\Webhooks) se apoya en una subcapa de validación (Services\Validators) que intercepta cada notificación entrante antes de cualquier procesamiento. MercadoLibreSignatureValidator y TikTokSignatureValidator inspeccionan los encabezados criptográficos del request utilizando las llaves secretas de la aplicación; si la firma no coincide, el proceso es bloqueado sin encolar el evento.

Este enfoque garantiza la mantenibilidad del sistema: la modificación de los esquemas de serialización o autenticación de una API específica queda confinada a su respectivo servicio, aislando la lógica de negocio principal de la organización. La Imagen 5 ilustra la organización de las capas de la arquitectura.

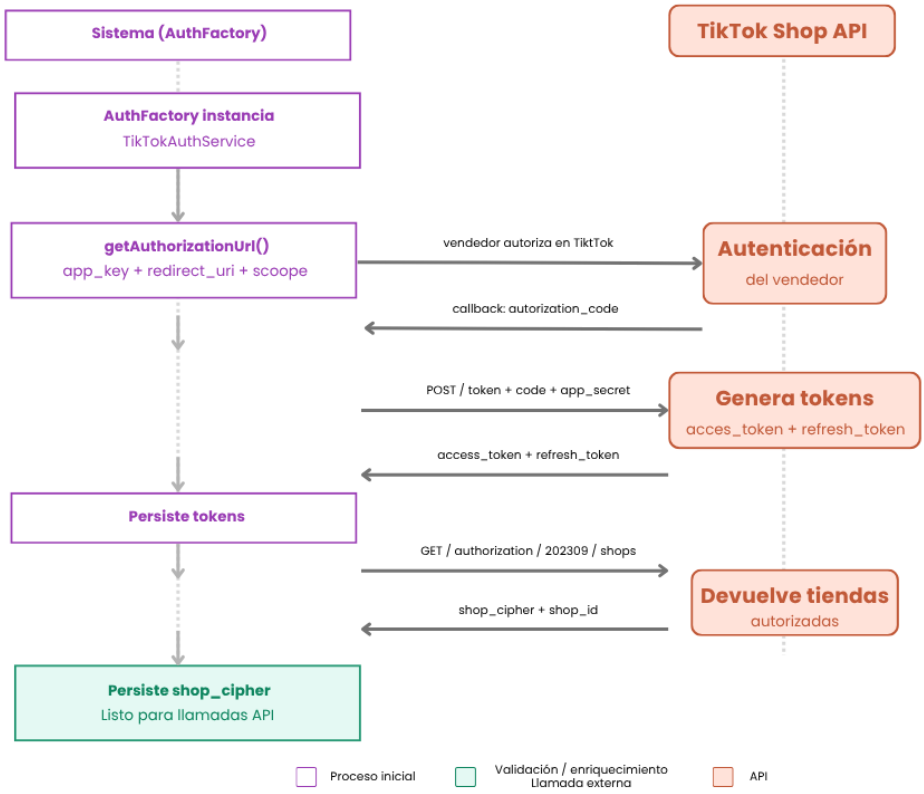

Imagen 5. Organización de las capas de la arquitectura.

6.5.3. Mecanismos de recepción y procesamiento de eventos

La captura de información opera principalmente a través de webhooks para las plataformas que soportan notificaciones reactivas (Mercado Libre y TikTok Shop). El procesamiento se divide en dos fases desacopladas: la recepción síncrona, controlada por el servidor web, y el procesamiento asíncrono, controlado por el Cron Job de cPanel. Esta separación garantiza que la conexión con el marketplace se libere de forma inmediata sin bloquear el servidor. El flujo completo sigue la secuencia descrita a continuación e ilustrada en la Imagen 6:

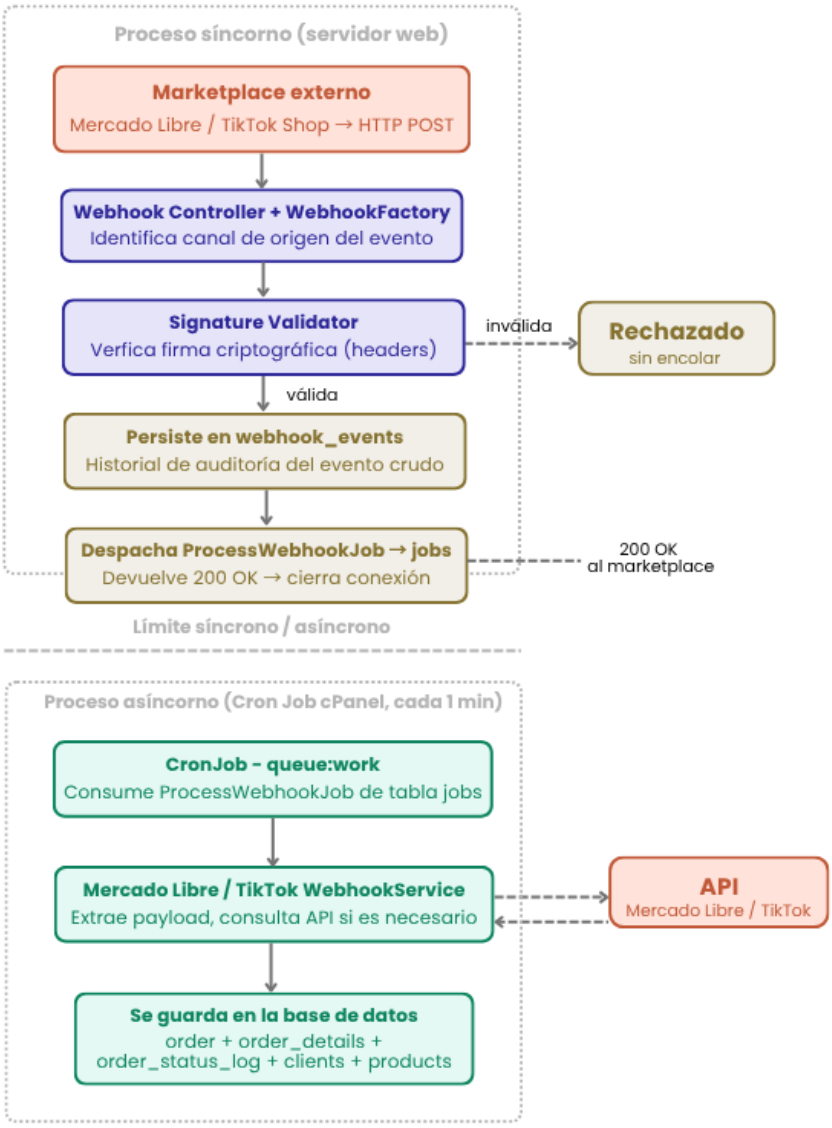

Imagen 6. Flujo de procesamiento de eventos mediante webhooks

a) Recepción e identificación: El WebhookController recibe la notificación HTTP POST externa. La WebhookFactory identifica en tiempo de ejecución el canal de origen del evento y dirige el flujo al validador y servicio correspondiente.

b) Validación de seguridad: El MercadoLibreSignatureValidator o TikTokSignatureValidator, según corresponda, intercepta la petición e inspecciona los encabezados criptográficos con la clave secreta de la

aplicación. Si la firma no coincide, el proceso se bloquea sin encolar el evento.

c) Registro y encolamiento: Una vez validada la autenticidad, el controlador persiste el evento crudo en la tabla webhook_events para historial de auditoría. Acto seguido, despacha un ProcessWebhookJob a la cola de trabajos de Laravel (tabla jobs) y devuelve de inmediato un código HTTP 200 OK al marketplace, cerrando la conexión entrante.

d) Procesamiento asíncrono por Cron Job: El Cron Job de cPanel, configurado para ejecutarse cada minuto mediante la bandera --stop-when-empty, despierta el procesador de colas de Laravel, el cual toma el ProcessWebhookJob acumulado e invoca el servicio especializado de la plataforma correspondiente (MercadoLibreWebhookService o TiktokWebhookService).

e) Enriquecimiento del payload: Los webhooks notifican únicamente el identificador del pedido y el estado actualizado. El servicio ejecuta una petición secundaria a la API REST del canal para recuperar el objeto de pedido completo, incluyendo datos del comprador, ítems y envío. Si se requiere consultar la API, el servicio obtiene el token activo mediante el AuthService correspondiente.

f) Estandarización y persistencia: El servicio mapea las propiedades nativas al modelo unificado del sistema e inserta los registros en las tablas orders (pedido general), order_details (artículos comprados) y order_status_logs (historial de estados), mapeando con los catálogos clients y products.

6.5.4. Automatización e infraestructura de procesamiento (Cron Jobs en cPanel)

Dado que el entorno de despliegue utiliza cPanel y no se dispone de un gestor de procesos continuo como Supervisor, la ejecución automatizada del sistema se delegó al Administrador de Tareas Cron de cPanel. Esta capa de automatización cubre dos frentes críticos, ilustrados en la Imagen 7:

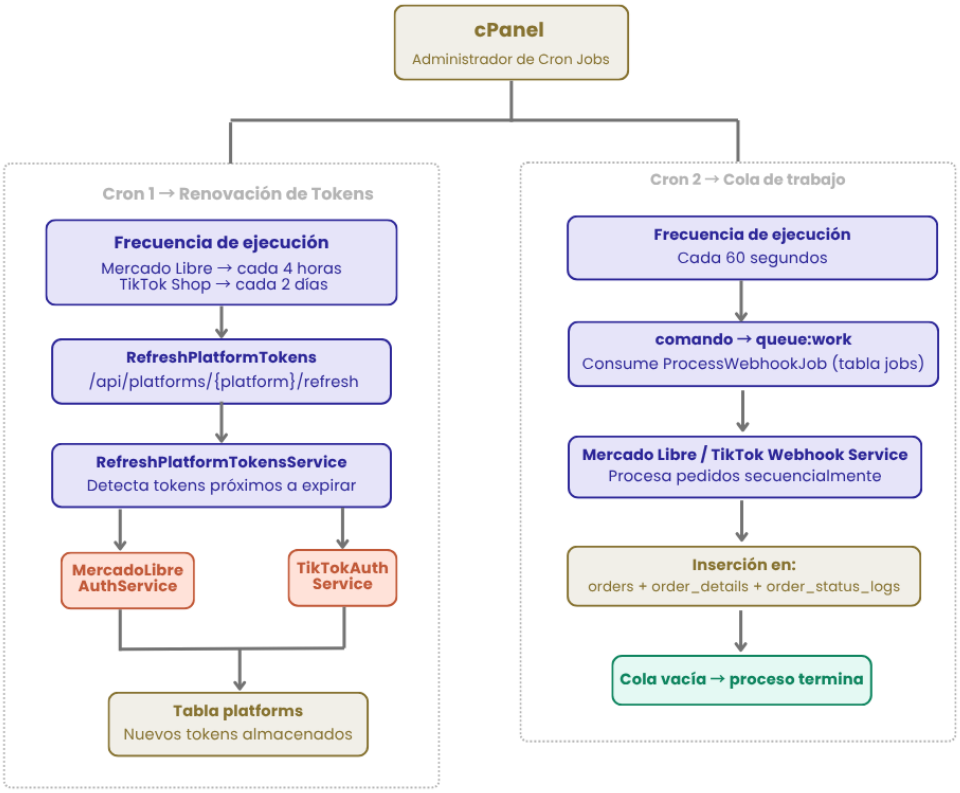

Imagen 7. Uso de Crons en cPanel.

- Renovación automática de tokens de acceso. Los access tokens provistos por Mercado Libre y TikTok Shop expiran tras un período determinado. Para evitar la interrupción de las conexiones de los usuarios, se programó un Cron Job el cual ejecuta la ruta /api/platforms/{platform}/refresh, el cual utiliza

RefreshPlatformTokensService para identificar qué tokens están próximos a expirar y ejecuta la renovación mediante MercadoLibreAuthService o TiktokAuthService según corresponda. Los nuevos tokens se almacenan en la tabla plataforms. El Cron Job ejecuta este comando cada 4 horas para Mercado Libre y cada 2 días para TikTok Shop, conforme a las políticas de expiración de cada plataforma.

- Procesamiento de la cola de trabajos. Cuando un pedido ingresa por webhook, el WebhookController despacha un ProcessWebhookJob a la cola de Laravel (tabla jobs) sin procesar la orden de forma inmediata. Para procesar esta lista de tareas pendientes, se programó un Cron Job que ejecuta el procesador de colas cada 60 segundos con la bandera --stop-when-empty. Esto permite que el proceso despierte, consuma todos los jobs acumulados de forma secuencial y se cierre limpiamente al vaciarse la cola, sin consumir memoria de manera continua.

6.5.5. Integración con la API de Mercado Libre

Para la integración con Mercado Libre se implementó MercadoLibreAuthService, que cumple el contrato AuthServiceInterface definiendo los métodos getAccessToken() y refreshToken(), así como el método getAuthorizationUrl(), el cual construye la URL de autorización con los parámetros response_type, client_id y redirect_uri requeridos por la plataforma (MercadoLibre S.R.L., 2026). La AuthFactory resuelve de forma polimórfica la instancia de este servicio al detectar el canal, sin estructuras condicionales en el controlador.

Una vez completado el flujo OAuth 2.0, el sistema almacena el access_token y el refresh_token. Se identificó que Mercado Libre requiere

persistir el seller_id en el modelo de plataforma, dado que este identificador es requerido en múltiples endpoints de la API para consultas posteriores.

La validación de autenticidad de los webhooks entrantes se delega a MercadoLibreSignatureValidator, el cual inspecciona los encabezados criptográficos de cada notificación utilizando la clave secreta de la aplicación. Si la firma no coincide, el proceso es bloqueado antes de registrar el evento.

El procesamiento del pedido es ejecutado por MercadoLibreWebhookService, el cual extrae del payload el identificador del pedido, cliente, SKU y cantidades, realiza la petición de enriquecimiento a la API mediante MercadoLibreApiService e inserta los registros normalizados en las tablas del sistema. Durante las pruebas se identificaron los siguientes desafíos específicos de esta plataforma:

- Obtención del SKU en la respuesta de la API: Mercado Libre no expone el SKU directamente en el objeto de respuesta principal. El campo seller_custom_field, inicialmente candidato para este dato se recuperaba vacío. Tras analizar la estructura completa de la respuesta, se localizó el SKU dentro del arreglo attributes, en el elemento con identificador SELLER_SKU, campo value_name. Una vez localizado este atributo, fue posible completar la relación automática con los productos del catálogo interno.
- Incompletitud de los payloads de webhook: Las notificaciones de Mercado Libre incluyen únicamente el identificador del recurso modificado. El servicio ejecuta una petición secundaria al endpoint de detalle del pedido para recuperar el objeto completo antes de procesar la información.

6.5.6. Integración con la API de TikTok Shop

La integración con TikTok Shop introdujo requisitos de autenticación y comunicación diferenciados respecto a Mercado Libre (TikTok Shop Partner Center, 2025). TiktokAuthService implementa AuthServiceInterface y define getAuthorizationUrl() construyendo la URL de autorización con los parámetros app_key, redirect_uri y scope requeridos por la plataforma. Una vez completado el flujo OAuth 2.0, el sistema ejecuta una consulta al endpoint GET /authorization/202309/shops para recuperar y persistir el shop_cipher, parámetro requerido en todas las solicitudes posteriores a la API.

Todas las peticiones a la API de TikTok Shop deben incluir una firma HMAC-SHA256 construida concatenando el timestamp, el app_key, el shop_cipher y los parámetros de la solicitud ordenados alfabéticamente. Esta lógica se centralizó en TiktokApiService, reutilizable por todos los endpoints.

La validación de webhooks entrantes se delega a TikTokSignatureValidator, que reconstruye la firma HMAC-SHA256 a partir del encabezado x-tts-timestamp y la clave secreta de la aplicación. El procesamiento del pedido es ejecutado por TiktokWebhookService, que solicita el detalle del pedido al endpoint POST /order/202309/orders/search con los parámetros order_id_list y page_size antes de persistir los registros.

Durante la implementación se identificaron y resolvieron los siguientes desafíos específicos:

- Pérdida del parámetro state en el flujo OAuth: Durante pruebas locales se detectó pérdida del parámetro state en el callback de OAuth 2.0, con el error oauth state no encontrado. El problema derivó de diferencias en la configuración de sesiones entre el

entorno de desarrollo y el servidor. La solución consistió en validar la persistencia del parámetro dentro del dominio configurado.

- Paginación obligatoria: TikTok Shop exige el parámetro page_size en todas las consultas, incluso en búsquedas de un único registro, generando el error PageSize is a required field. Se implementó una capa de abstracción en TiktokApiService que incluye automáticamente los parámetros de paginación con valores predeterminados en cada solicitud.
- Sincronización de zonas horarias: El mecanismo de firma requiere que el timestamp corresponda al momento exacto de la petición en UTC con tolerancia máxima de ±300 segundos. Desfases en la configuración horaria del servidor generaron respuestas de firma inválida de forma intermitente, lo que se resolvió forzando la generación del timestamp en UTC dentro de la función de firma centralizada.

6.5.7. Investigación para la integración con la API de Amazon

Debido a los alcances temporales del proyecto y a los requisitos regulatorios del entorno de producción de Amazon, el desarrollo técnico para esta plataforma se acotó a una fase de investigación y diseño de arquitectura preliminar. El análisis se estructuró en torno a tres ejes técnicos derivados del estudio de la documentación oficial de la Amazon Selling Partner API (SP-API) (Amazon Selling Partner API, 2024).

Protocolo de autenticación y obtención de credenciales. Amazon SP-API requiere, a diferencia de Mercado Libre y TikTok Shop, la configuración simultánea de dos entornos: Amazon Seller Central y AWS (Amazon Web Services). El flujo de credenciales implica el registro de un perfil de desarrollador público en Seller Central, la creación de un usuario IAM con

accesos programáticos en AWS para obtener el Access Key ID y el Secret Access Key, la definición de una política IAM que permita la acción sts:AssumeRole, y la vinculación del ARN del Rol IAM resultante con la aplicación registrada en Seller Central. Este proceso otorga las credenciales LWA (Login with Amazon): Client ID y Client Secret. Para incorporar esta plataforma a la arquitectura existente, se deberá crear AmazonAuthService, que implemente AuthServiceInterface y gestione el intercambio del código de autorización por el refresh_token, así como la renovación horaria del access_token, cuya vigencia es de 60 minutos.

Mecanismo de captura de eventos de pedidos. Se identificó una diferencia arquitectónica fundamental respecto a Mercado Libre y TikTok Shop: Amazon SP-API no envía notificaciones directamente a una URL HTTPS pública del servidor, sino que las enruta a través de Amazon SQS (Simple Queue Service) o Amazon EventBridge. La estrategia de integración determinada consiste en configurar una cola SQS en AWS para recibir eventos de tipo ORDER_CHANGE, implementar un comando de Laravel o un proceso puente que consulte periódicamente dicha cola, y al extraer el payload, dirigirlo a la WebhookFactory existente para su procesamiento mediante AmazonWebhookService, el cual despachará el ProcessWebhookJob para registrar los pedidos en orders y order_details. La validación de autenticidad de los mensajes se realizará mediante AmazonSignatureValidator, que verificará el hash MD5/SHA256 del payload proveniente del SDK de AWS, en lugar de validar un encabezado HTTP expuesto a internet como ocurre en las demás plataformas.

Mapeo de datos e identificadores únicos. Se analizó la estructura JSON devuelta por el endpoint GET /products/pricing/v0/items/{Asin}/offers, identificando los campos ASIN (identificador único de Amazon), SellerSKU y

ListPrice como los datos que AmazonProductSyncService, implementando ProductSyncInterface, deberá mapear hacia la tabla plataform_products. Para los pedidos, el evento ORDER_CHANGE incluye los campos AmazonOrderId, OrderStatus, FulfillmentType (AFN para envíos gestionados por Amazon, MFN para envíos directos del vendedor) y el resumen financiero necesario para construir el registro unificado del sistema.

Las especificaciones completas, la guía paso a paso para la obtención de credenciales y los modelos de payloads estructurados se documentan en el Anexo 2.

6.5.8. Investigación para la integración con la API de Shein

De manera análoga a la sección anterior, la incorporación de Shein se abordó desde una perspectiva de viabilidad tecnológica e investigación documental, con el objetivo de definir la ruta de implementación sin comprometer los tiempos de entrega del núcleo del sistema. El análisis tomó como base la documentación oficial de SHEIN Open Platform (SHEIN Developer Platform, s. f.).

Acoplamiento con la arquitectura de contratos existente. Se verificó que el flujo de autorización OAuth 2.0 de Shein es compatible con los contratos ya implementados en el sistema. SheinAuthService deberá implementar AuthServiceInterface, gestionando un flujo de tres etapas: generación de la URL de autorización con appId y redirectUrl codificada en Base64, recepción del tempToken en el callback y su intercambio mediante el endpoint /open-api/auth/get-by-token para obtener el OpenKeyId y el SecretKey. A diferencia de las demás plataformas, Shein no utiliza Bearer Tokens estáticos; en su lugar, exige que cada petición HTTP incluya los encabezados x-lt-openKeyId, x-lt-timestamp y x-lt-signature. Para registrar la plataforma en el sistema, se

añadirá el caso 'shein' en AuthFactory y WebhookFactory, y se registrará un nuevo registro en la tabla plataforms con el slug correspondiente.

Algoritmo de firma por petición. Se investigó el mecanismo criptográfico requerido por Shein para autenticar cada solicitud, que deberá implementarse en SheinSignatureValidator y en SheinApiService. El algoritmo consta de cinco pasos: concatenación del OpenKeyId, el Timestamp y el Path del endpoint separados por & para formar el valor VALUE; generación de una cadena aleatoria de cinco caracteres (RandomKey) que se anexa al SecretKey para formar la clave KEY; cifrado HMAC-SHA256 de VALUE con KEY y conversión del resultado binario a cadena hexadecimal; codificación Base64 de dicha cadena; y concatenación del RandomKey como prefijo del resultado Base64 para obtener el valor final del encabezado x-lt-signature. Este mecanismo de firma dinámica de un solo uso mitiga los ataques de repetición al invalidar cualquier firma capturada previamente.

Estructura de webhooks y sincronización de productos. Se analizó el esquema de notificaciones HTTP POST de Shein, que sigue el modelo JIT/Parent-Child Orders: el payload incluye el event (ORDER_CREATE), el openKeyId del vendedor, el orderNo, el orderStatus y la lista de paquetes con sus goodsIds y código de almacén. Este payload será procesado por SheinWebhookService para despacharlo al ProcessWebhookJob existente. Para la sincronización de productos, SheinProductSyncService, implementando ProductSyncInterface, consumirá los endpoints /open-api/product/ o /open-api/goods/, extrayendo los campos spuCode, sellerSku y price del nivel de SKU para mapearlos hacia plataform_products mediante SavePlatformProductsService.

La documentación completa de esta investigación, que abarca desde el registro en el portal de desarrolladores de SHEIN Open Platform hasta los esquemas JSON de productos y pedidos, se encuentra en el Anexo 3.

6.5.9. Sincronización de productos con plataformas externas

Como complemento al módulo de recepción de pedidos, se desarrolló una funcionalidad orientada a sincronizar los productos registrados en los marketplaces externos con el catálogo interno del sistema. El propósito consistió en mantener una correspondencia entre cada producto del catálogo maestro y su identificador externo en cada plataforma, habilitando las operaciones de integración relacionadas con pedidos e inventario.

Durante el análisis funcional se identificó que el sistema ya contaba con un catálogo maestro de productos administrado internamente, por lo que se redefinió la estrategia de sincronización para recuperar exclusivamente los campos mínimos necesarios para establecer la relación entre plataformas: el identificador externo del producto (external_product_id), el SKU externo (external_sku) y el precio registrado en el marketplace (price). Esta información se almacena en la tabla plataform_products, que actúa como tabla intermedia entre el producto interno y su representación en cada marketplace, permitiendo que un mismo producto del catálogo exista simultáneamente en distintas plataformas con identificadores independientes. La Tabla 3 describe la estructura de esta tabla y la Imagen 8 ilustra el modelo conceptual de relación.

Tabla 3. Estructura de la tabla plataformplataform_products

Campo	Tipo de dato	Descripción
id	INT (PK)	Identificador interno del registro de sincronización.
product_id	INT (FK)	Referencia al producto en el catálogo maestro del sistema.
platform_id	INT (FK)	Referencia a la plataforma marketplace vinculada.
external_product_id	VARCHAR	Identificador del producto en el marketplace externo.
external_sku	VARCHAR	SKU del producto registrado en la plataforma externa.
price	DECIMAL	Precio del producto registrado en la plataforma.

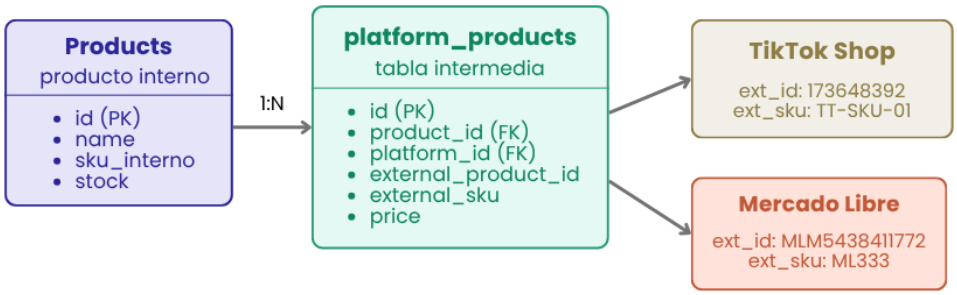

Imagen 8. Modelo conceptual de relación entre el catálogo interno y las plataformas externas

El diagrama muestra cómo un mismo producto interno puede existir simultáneamente en distintas plataformas mediante identificadores independientes gestionados por la tabla plataform_products.

La estructura permite que un mismo producto del catálogo interno exista simultáneamente en distintas plataformas conservando identificadores independientes, lo que evita la duplicación de registros y centraliza la administración del catálogo.

Para evitar el acoplamiento de la lógica de cada plataforma al controlador principal, se implementó una capa de servicios basada en contratos. ProductSyncInterface define el contrato de sincronización; MercadoLibreProductSyncService y TikTokProductSyncService lo implementan, traduciendo el modelo local Product al formato requerido por cada API y viceversa. ProductSyncService actúa como orquestador general, y SavePlatformProductsService persiste la relación en la tabla plataform_products una vez que la API confirma la vinculación. La Imagen 9 ilustra la arquitectura de servicios resultante.

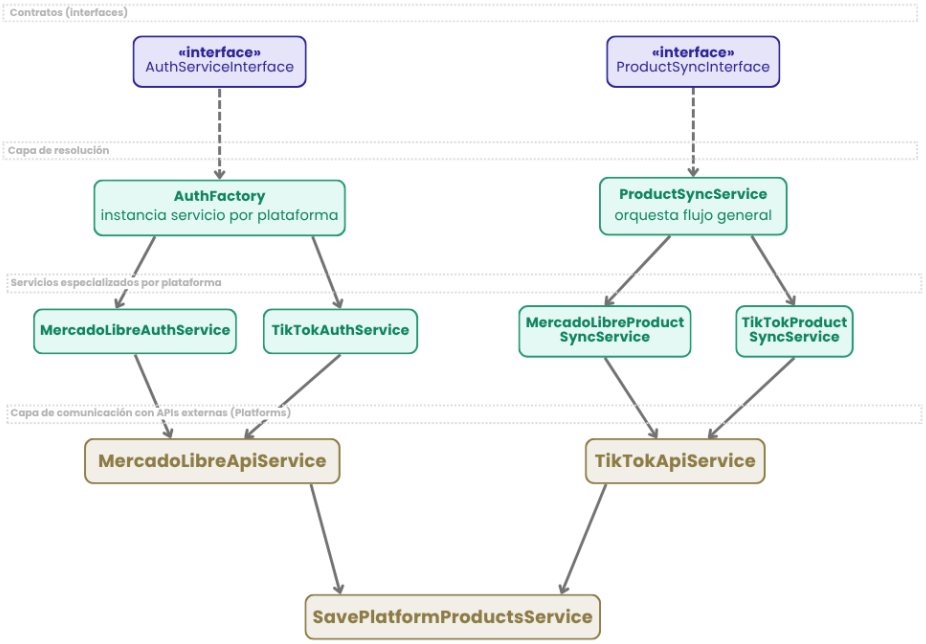

Imagen 9. Arquitectura de servicios basada en contratos para la sincronización de productos

Para Mercado Libre, el proceso consistió en dos llamadas secuenciales: primero se obtuvo el listado de identificadores del vendedor mediante GET /users/{seller_id}/items/search y posteriormente se consultó el detalle de cada producto. Para TikTok Shop se utilizó el endpoint POST /product/202309/products/search, siguiendo el flujo de validación de cuenta, obtención de la tienda autorizada, recuperación del shop_cipher, firma de la petición y consulta de productos. Los errores específicos encontrados durante la implementación con ambas plataformas fueron documentados en las secciones 6.5.5 y 6.5.6 respectivamente.

Se identificó que algunos productos podían no coincidir automáticamente debido a diferencias entre los SKU internos y externos. Para gestionar este escenario se desarrolló un mecanismo de vinculación manual: el sistema verifica si existe un producto interno con SKU coincidente; en caso afirmativo ejecuta la sincronización automática almacenando el registro con

estado synced; en caso negativo el producto queda en estado pending y se habilita la vinculación manual mediante una interfaz que permite visualizar productos sincronizados, detectar pendientes, buscar en el catálogo interno, establecer la relación manualmente y editar el SKU cuando sea necesario. La Imagen 10 ilustra este flujo de decisión.

Imagen 10. Flujo de decisión para la sincronización y vinculación manual de productos

6.5.10. Desafíos técnicos y estrategias de mitigación

Durante la etapa de despliegue y pruebas de las integraciones se identificaron y solventaron los siguientes desafíos técnicos:

Configuración de credenciales y restricciones de seguridad: El proceso requirió la configuración de las aplicaciones en las consolas de desarrolladores de cada marketplace, definiendo las URLs de redirección para los flujos OAuth 2.0, la verificación de endpoints mediante firmas

criptográficas y la gestión automática de la renovación de credenciales mediante los tokens access_token y refresh_token.

Mitigación de eventos duplicados y concurrencia: Las pruebas evidenciaron que variaciones consecutivas en el estado de una venta pueden generar ráfagas de notificaciones casi simultáneas. Este escenario se resolvió mediante la implementación de un control estricto de idempotencia basado en el campo order_id dentro del ProcessWebhookJob, descartando transacciones redundantes antes de impactar la base de datos.

Incompletitud en los datos de las notificaciones: Se resolvió mediante el enriquecimiento del payload en la fase asíncrona: dado que los webhooks frecuentemente notifican únicamente un identificador de cambio, el servicio ejecuta llamadas complementarias a los endpoints de detalle de pedido de manera controlada, evitando saturar las cuotas de peticiones permitidas por cada plataforma.

Saturación por carga transaccional: El desacoplamiento mediante la cola de Laravel previno la denegación de servicio ante picos de venta. El procesamiento asíncrono activado por el Cron Job permitió absorber el flujo de notificaciones entrantes a alta velocidad, procesándolas de manera regulada conforme a la capacidad operativa de la base de datos interna.

6.6. VALIDACIÓN DE LOS PROCESOS CRÍTICOS DEL SISTEMA

Para validar la estabilidad y el correcto funcionamiento de los procesos críticos del sistema, se diseñó una estrategia de validación funcional dirigida a los módulos que conforman el núcleo operativo de la plataforma: administración de usuarios y permisos, catálogo (productos, paquetes, plataformas), operación de pedidos (orders, order_details, order_status_log, order_details_lots) e

integración con los marketplaces Mercado Libre y TikTok Shop. La estrategia se describe en el diagrama de la Imagen 11.

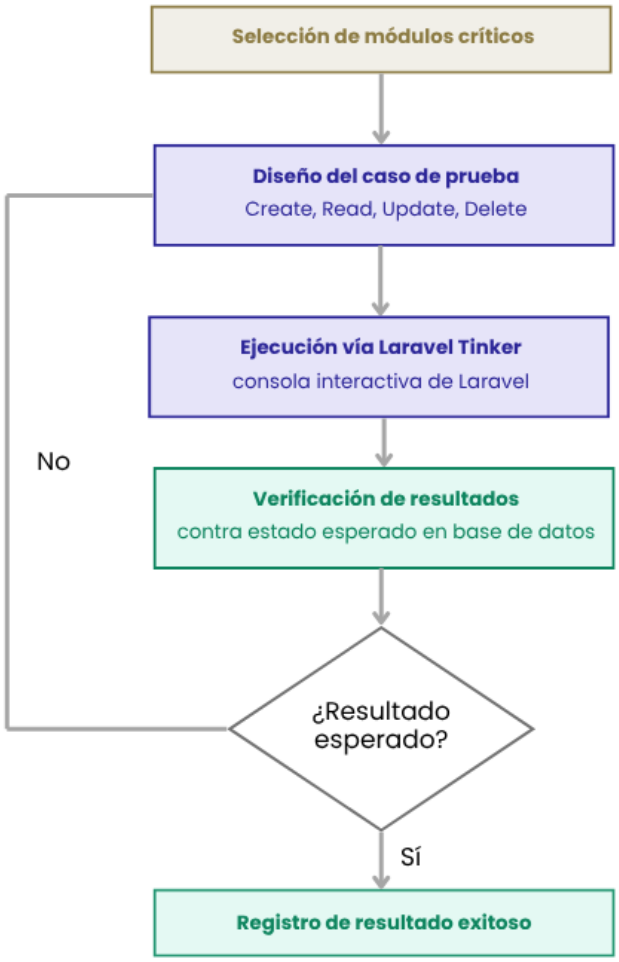

Imagen 11. Flujo metodológico de pruebas unitarias por módulo.

Las pruebas se ejecutaron sobre cada modelo Eloquent del sistema siguiendo el esquema CRUD (Create, Read, Update, Delete), con el objetivo de verificar que las operaciones de persistencia, consulta, actualización y eliminación lógica (soft delete) se comportaran de acuerdo con las reglas de negocio definidas para cada entidad. En una primera etapa, las pruebas se realizaron de forma interactiva mediante la herramienta Tinker de Laravel, lo que

permitió instanciar modelos, asignar atributos, ejecutar los métodos save() y delete(), y consultar los resultados directamente contra la base de datos, validando tanto los valores devueltos como el estado final de los registros.

Adicionalmente, se incluyeron pruebas sobre procesos compuestos considerados críticos para la operación del negocio: la generación de documentos PDF de pedidos, la sincronización de pedidos y productos provenientes de Mercado Libre y TikTok Shop, la recepción y almacenamiento de webhooks de notificación, y la asignación de usuarios a plataformas. Estos procesos involucran múltiples capas de la aplicación (modelos, servicios de integración, colas de procesamiento y generación de reportes), por lo que su validación permitió confirmar la correcta interacción entre los distintos componentes de la arquitectura.

6.7. ELABORACIÓN DE LA DOCUMENTACIÓN TÉCNICA, MANUALES DE INSTALACIÓN Y PRUEBAS

Para garantizar la mantenibilidad, escalabilidad y correcta transferencia tecnológica de la plataforma SaaS, se llevó a cabo un proceso estructurado de documentación técnica dividido en tres ejes fundamentales:

- Modelado y Estructuración de la Base de Datos: A partir de las migraciones desarrolladas nativamente en el framework Laravel, se procedió a realizar la ingeniería inversa y el mapeo relacional del sistema. Se utilizó una herramienta de modelado de datos para generar el diagrama relacional, asegurando la correspondencia exacta de las llaves foráneas y las tablas pivote. Debido a las dimensiones del esquema, el diagrama no se adjunta en el informe, ya que su inclusión fragmentada comprometería la legibilidad de las relaciones entre entidades. Posteriormente, se elaboró el diccionario de datos detallando el nombre de cada campo, tipo de dato,

restricciones de nulidad y descripciones funcionales, el cual se presenta en el Anexo 1.

- Diseño de Manuales de Usuario e Instalación (Guías de Navegación Visual): Con el objetivo de agilizar y estandarizar la creación de manuales procedimentales, se integró la herramienta de inteligencia artificial y captura automatizada Tango AI. Mediante esta plataforma, se registraron de forma interactiva y paso a paso los flujos operativos esenciales del entorno. Los manuales se segmentaron lógicamente en dos flujos críticos:
    - Módulos de Plataformas y Pedidos (Platforms & Orders): Documentación visual del ciclo de vida de una orden, la conexión mediante credenciales de plataformas externas (Mercado Libre y TikTok Shop) y la asignación multiusuario.
    - Administración y Configuración del Sistema (Módulos del Sistema): Guías detalladas de los catálogos base indispensables para la operación (gestión de productos, empaques, tipos de estatus, recursos, privilegios, etc).
- Generación del Reporte de Pruebas Unitarias y Funcionales: Se diseñó una estrategia de validación integral utilizando la consola interactiva php artisan tinker de Laravel. Se definieron casos de prueba específicos para comprobar de manera secuencial los flujos CRUD (Crear, Leer, Actualizar, Eliminar) de cada uno de los 9 módulos centrales del sistema. Las ejecuciones se validaron directamente sobre la persistencia en la base de datos, registrando las respuestas lógicas del ORM Eloquent ante escenarios de éxito y manejo de restricciones de integridad.

7. RESULTADOS

7.1. REQUERIMIENTOS FUNCIONALES Y TÉCNICOS DEL SISTEMA DE GESTIÓN MULTICANAL DE PEDIDOS

Se obtuvo un documento de requerimientos que consolidó 3 requerimientos funcionales y 3 requerimientos técnicos necesarios para el desarrollo de la plataforma SaaS de gestión multicanal de pedidos en SDIB.

Entre los requerimientos funcionales identificados se encuentran: la sincronización de pedidos provenientes de Mercado Libre, Amazon, Shein y TikTok Shop; la visualización centralizada del estatus de cada pedido; la administración de usuarios con distintos niveles de acceso; y la generación de notificaciones ante cambios en el estado de un pedido, los cuales se encuentran en la Imagen 12.

Requerimientos Funcionales		
ID	**Nombre**	**Descripción**
RF-01	Sincronización en Tiempo Real / Near Real-Time	Captura automática de pedidos mediante Webhooks (Mercado Libre, TikTok) o Polling eficiente (Amazon SP-API, Shein).
RF-02	Mapeo de Estados Unificado	Homologar los estados nativos de cada marketplace a un flujo interno de SDIB (ej: Catalogo de estatus).
RF-03	RBAC (Control de Acceso Basado en Roles)	Roles diferenciados para Administradores y Operadores de Almacén.

Imagen 12. Requerimientos funcionales

En cuanto a los requerimientos técnicos, se determinó el uso del framework Laravel como base del desarrollo, la integración mediante las APIs oficiales de cada marketplace, la implementación de autenticación basada en tokens JWT, y el uso de MySQL como sistema gestor de base de datos en el entorno de desarrollo, compatible con MariaDB, motor utilizado en el entorno de

producción sobre infraestructura cPanel, los cuales se encuentran en la Imagen 13.

Requerimientos Técnicos		
ID	**Nombre**	**Descripción**
RT-01	Arquitectura Dirigida por Eventos (EDA)	Uso del componente de colas nativo de Laravel para encolar los pedidos entrantes. Si la API de un marketplace se cae o satura, el sistema no pierde datos.
RT-02	Estrategia de Idempotencia	Procesamiento único de pedidos basado en el marketplace_order_id para evitar duplicidad en la base de datos si un Webhook se dispara dos veces.
RT-03	Seguridad	Cifrado AES-256 para los tokens de acceso de las APIs de las tiendas (access_token, refresh_token).

Imagen 13. Requerimientos técnicos

El análisis permitió delimitar el alcance funcional de la plataforma y establecer las condiciones técnicas que guiaron las etapas subsecuentes del desarrollo.

7.2. ARQUITECTURA DEL SISTEMA Y ESTRUCTURA DE BASE DE DATOS

Se obtuvo el diseño de la arquitectura del sistema y la estructura de base de datos requeridas para la administración centralizada de pedidos en la plataforma.

La arquitectura definida siguió el patrón MVC (Modelo-Vista-Controlador) propio del framework Laravel, complementado con una capa de servicios encargada de gestionar la comunicación con las APIs externas de cada marketplace. Se estableció la separación de responsabilidades entre los módulos de autenticación, gestión de pedidos, administración de usuarios y control de permisos, lo que facilitó el desarrollo modular e independiente de cada componente.

La estructura de base de datos resultante quedó conformada por 29 tablas, entre las que se incluyen: platforms, products, plataform_products, orders, order_details, webhook_events, users, statuses. El modelo relacional diseñado permitió vincular cada pedido con su marketplace de origen, su estatus actualizado y el usuario responsable de su seguimiento. Las especificaciones detalladas de los campos, tipos de datos, llaves primarias y foráneas de estas entidades se presentan en el Diccionario de Datos ubicado en el Anexo 1.

7.3. MÓDULOS DE AUTENTICACIÓN, ADMINISTRACIÓN DE USUARIOS Y CONTROL DE PERMISOS

Se implementaron los módulos de autenticación, administración de usuarios y control de permisos de la plataforma, los cuales quedaron integrados de forma funcional en el entorno de desarrollo.

El módulo de autenticación se desarrolló mediante el estándar JWT (JSON Web Tokens), lo que permitió gestionar sesiones sin estado (stateless) y asegurar el acceso a los endpoints protegidos de la aplicación. Se configuró la expiración y renovación de tokens conforme a los requerimientos de seguridad definidos en la etapa de análisis.

El módulo de administración de usuarios permitió registrar, editar, activar y desactivar cuentas dentro de la plataforma. El control de permisos se implementó bajo el modelo RBAC (Role-Based Access Control), mediante el cual se definieron tres roles: Administrador, Observador y Editor. Cada rol quedó asociado a un conjunto específico de permisos que determina las acciones disponibles dentro del sistema. Con ello se logró restringir el acceso a funciones sensibles según el perfil de cada usuario.

En la Imagen 14 se muestra la interfaz del módulo de administración de usuarios, y en la Imagen 15 se presenta la pantalla de asignación de roles y permisos.

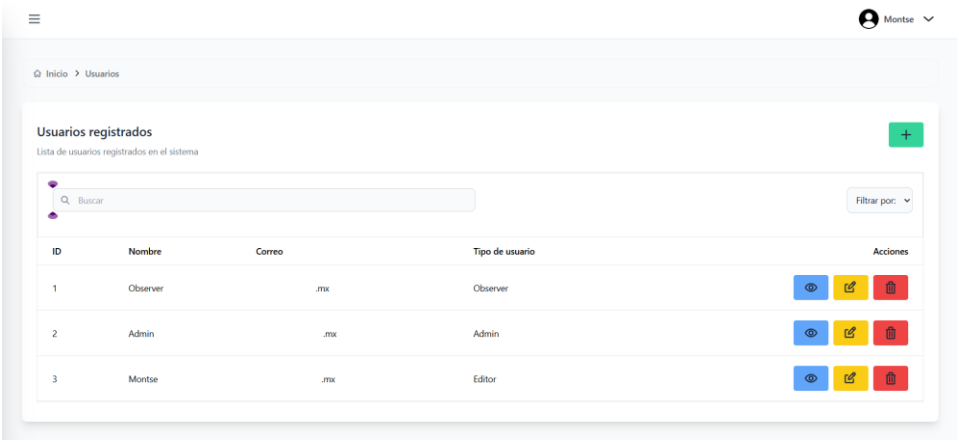

Imagen 14. Interfaz del módulo de administración de usuarios

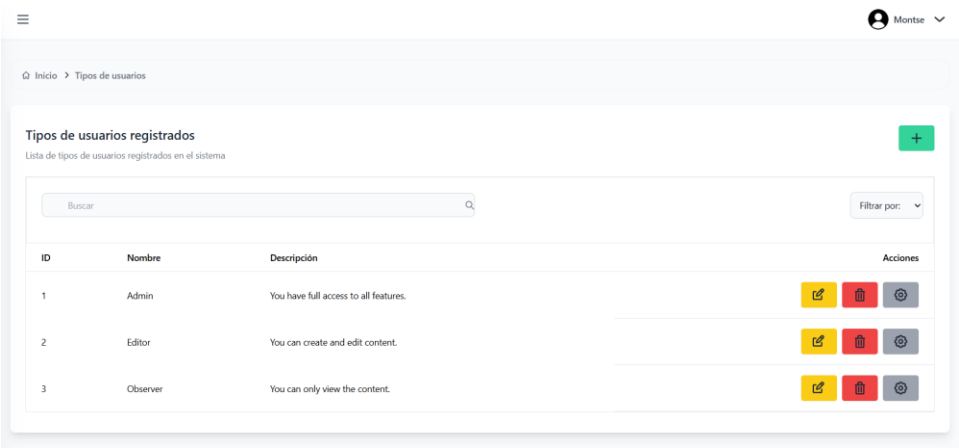

Imagen 15. Interfaz de la asignación de roles y permisos

7.4. DASHBOARD CENTRALIZADO PARA LA VISUALIZACIÓN Y ADMINISTRACIÓN DE PEDIDOS

Se desarrolló el dashboard centralizado que permite la visualización y administración del estatus de pedidos provenientes de los dos marketplaces integrados, Mercado Libre y TikTok Shop, desde una interfaz unificada.

El dashboard obtenido presentó los pedidos en una tabla interactiva con capacidad de filtrado por marketplace de origen, estatus del pedido, rango de fechas y término de búsqueda. Los estatus manejados por la plataforma incluyeron: nuevo, loteo en progreso, loteo pendiente de finalizar, loteo finalizado, enviado, entregado, cancelado, en proceso de devolución y devuelto. Cada registro mostró información resumida del pedido, incluyendo identificador, marketplace, fecha de creación, cliente y estatus actual, con acceso al detalle completo mediante selección del registro.

Adicionalmente, se integraron indicadores de resumen en la parte superior del dashboard que mostraron el total de pedidos por estatus en tiempo real, lo que facilitó la identificación del estado general de las operaciones de SDIB sin necesidad de consultar cada marketplace de forma independiente.

En las Imágenes 16, 17 y 18 se presentan la vista general del dashboard con pedidos de múltiples canales cargados simultáneamente.

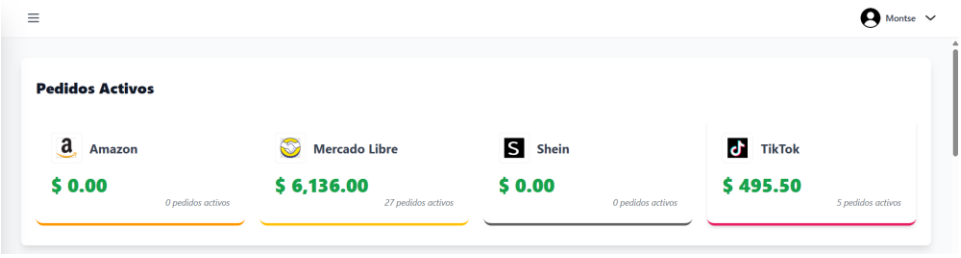

Imagen 16. Dashboard del sistema, sección Pedidos Activos

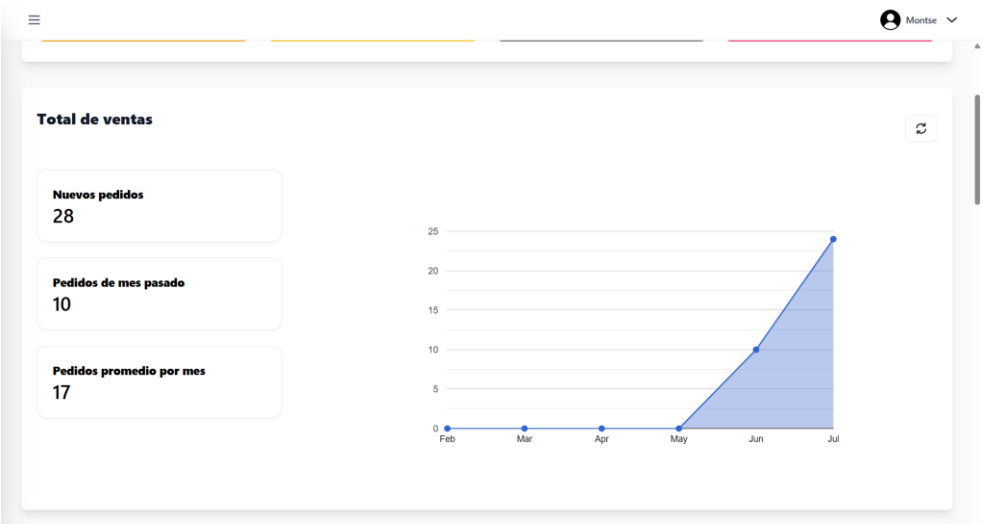

Imagen 17. Dashboard del sistema, sección Total de ventas

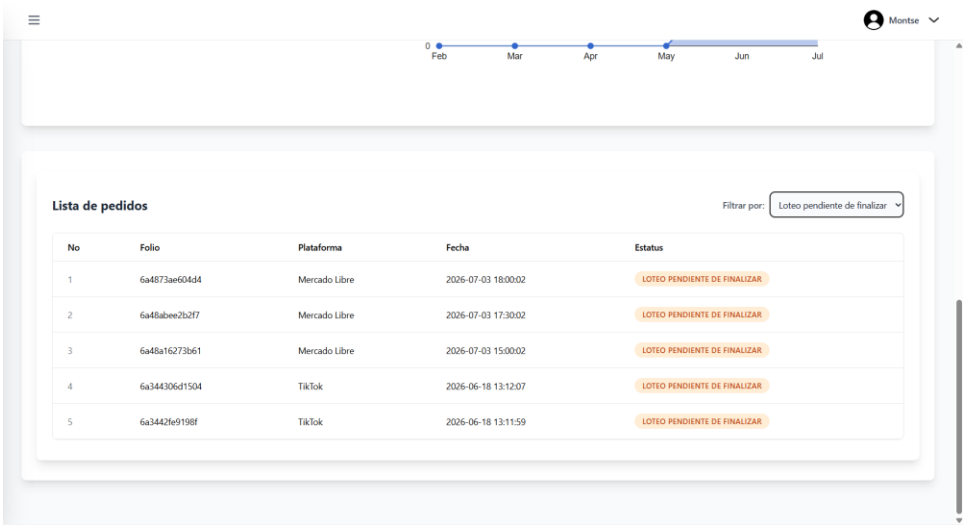

Imagen 18. Dashboard del sistema, sección Lista de pedidos filtrados por el estatus "Loteo pendiente de finalizar"

7.5. INTEGRACIÓN DE LAS API'S DE MERCADO LIBRE Y TIKTOK SHOP PARA LA SINCRONIZACIÓN DE INFORMACIÓN DE PEDIDOS

Los resultados obtenidos durante la implementación y pruebas del módulo de integración confirmaron el funcionamiento correcto de la arquitectura desacoplada descrita en la sección 6.5. Las validaciones se ejecutaron sobre el entorno de producción desplegado en cPanel, cubriendo los flujos de recepción de pedidos por webhook, la sincronización de productos y la operación continua de los Cron Jobs en ambas plataformas integradas.

7.5.1. Integración con Mercado Libre

Se procesaron exitosamente más de 20 pedidos recibidos mediante webhook. Para cada evento, el sistema completó la secuencia de validación de firma, registro en webhook_events, encolamiento del ProcessWebhookJob y persistencia en orders, order_details y order_status_logs. Durante las pruebas se identificó que los servidores de Mercado Libre reenvían la notificación cuando el servidor tarda en confirmar la recepción, generando más de cinco intentos de inserción duplicada. El mecanismo de idempotencia implementado mediante la verificación del event_id en webhook_events bloqueó la totalidad de estos intentos, sin que se registrara ningún pedido duplicado en la base de datos. Los pedidos que se obtuvieron se muestran en la Imagen 19 y en la Imagen 20 se muestra el detalle de un pedido.

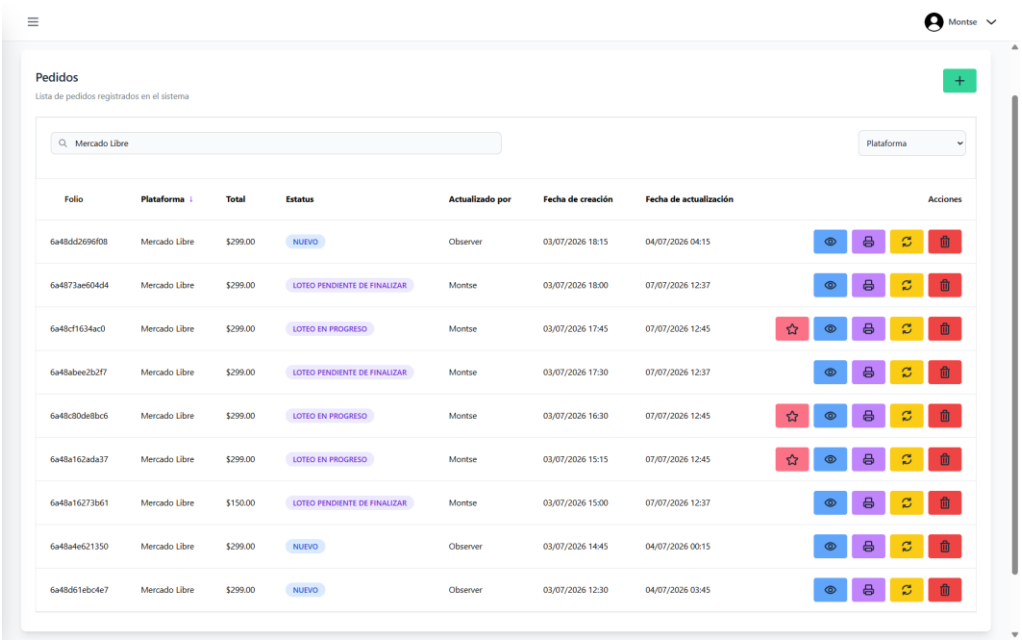

Imagen 19. Pedidos de Mercado Libre

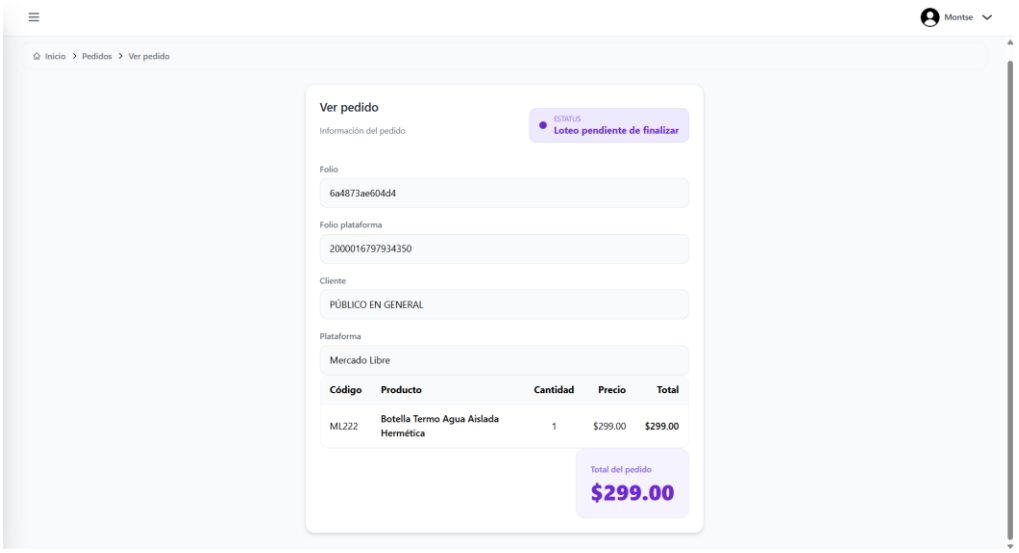

Imagen 20. Detalle de uno de los pedidos de Mercado Libre

En cuanto a la sincronización de productos, se recuperaron los cinco productos disponibles en la cuenta vinculada. Cuatro de ellos alcanzaron estado synced de forma automática, logrando una tasa de sincronización del 80 %. El producto restante quedó en estado pending por discrepancia entre

el SKU externo y el del catálogo interno, resolviéndose mediante el mecanismo de vinculación manual. En la Imagen 21 se muestra la integración completada de Mercado Libre, y la sincronización de los productos se muestra en la Imagen 22.

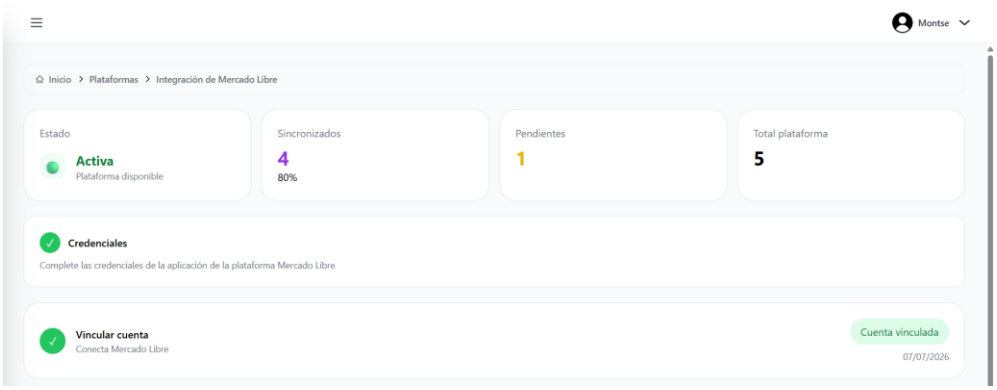

Imagen 21. Integración completa de Mercado Libre

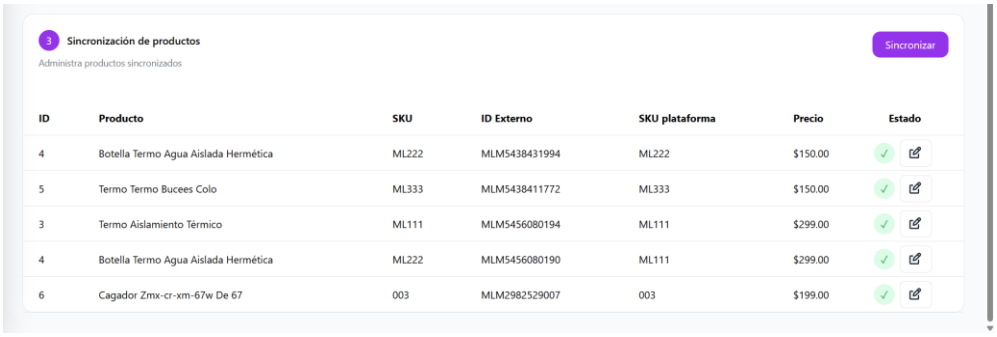

Imagen 22. Sincronización de los productos de Mercado Libre

7.5.2. Integración con TikTok Shop

Se procesaron más de 10 pedidos durante el período de pruebas, los cuales se muestran algunos en la Imagen 23 y en la Imagen 24 se muestra el detalle de uno de los pedidos.

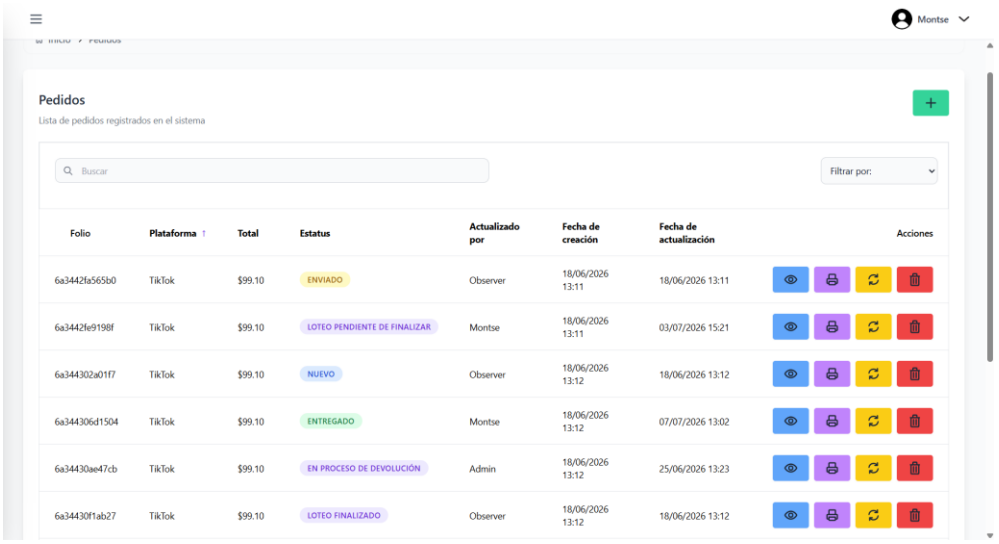

Imagen 23. Pedidos de TikTok Shop

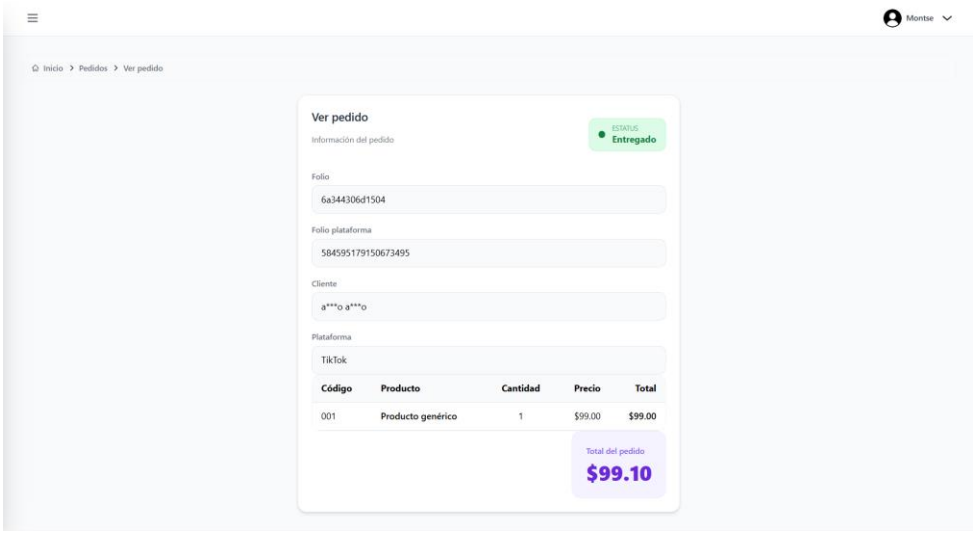

Imagen 24. Detalle de uno de los pedidos de TikTok Shop

El sistema completó correctamente la validación HMAC-SHA256 mediante el encabezado x-tts-timestamp, el registro en webhook_events y la persistencia en las tablas de negocio. En la sincronización de productos se recuperó y vinculó automáticamente el único producto disponible en la tienda

conectada (1/1), sin requerir vinculación manual. Se muestra la integración completa y sincronización en la Imagen 25.

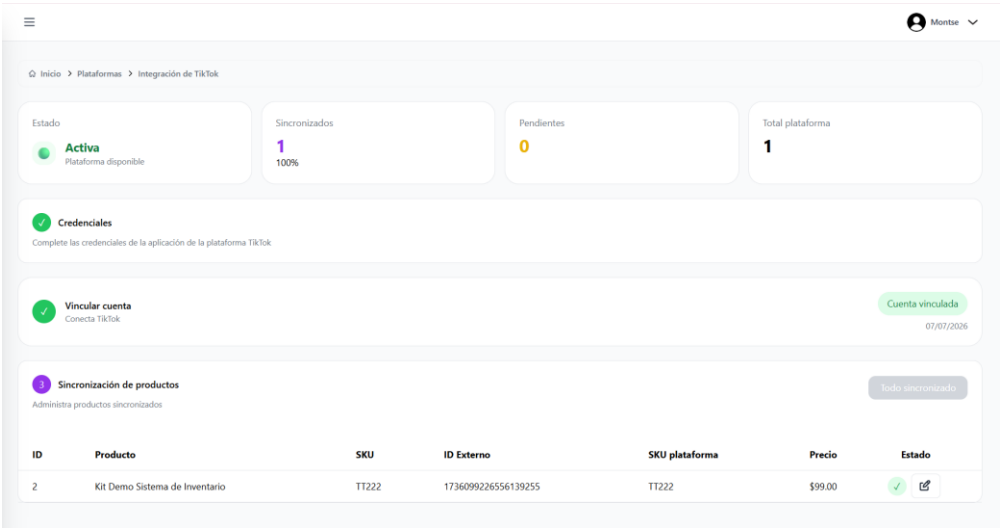

Imagen 25. Integración y sincronización completa de TikTok Shop

7.5.3. INVESTIGACIÓN Y DISEÑO DE INTERFACES PARA AMAZON Y SHEIN

Como resultado del objetivo específico orientado a la investigación de los mecanismos de integración de Amazon Selling Partner API (SP-API) y SHEIN Open Platform, se desarrollaron dos guías técnicas y arquitectónicas (incorporadas formalmente en el Anexo 2 y Anexo 3 de este documento). Estos entregables no representan únicamente revisiones teóricas, sino que constituyen el diseño técnico definitivo y el mapeo lógico necesario para la futura expansión del sistema.

A continuación, se detallan los resultados y hallazgos técnicos clave obtenidos de este proceso de investigación:

a) Especificación del Modelo de Integración para Amazon SP-API: A través del análisis detallado de la infraestructura de Amazon, se

determinó el flujo de autenticación basado en AWS IAM (Identity and Access Management) y Login with Amazon (LWA). El resultado de esta investigación técnica permitió estructurar:

- Mapeo de Datos de Listados de Productos: Se identificó y documentó la conversión lógica del parámetro SellerSKU de Amazon hacia la columna sku de la tabla relacional interna products, así como el almacenamiento del identificador ASIN dentro de la tabla intermedia plataform_products.
- Diseño del Flujo Asíncrono de Pedidos: Se especificó el consumo de eventos estructurados bajo el formato de notificación ORDER_CHANGE de Amazon mediante una arquitectura orientada a eventos, determinando la distinción operativa entre pedidos gestionados por el vendedor (MFN) y por Amazon (AFN) para el correcto control de inventarios.

b) Especificación del Modelo de Integración para SHEIN Open Platform: La investigación de la plataforma de SHEIN derivó en el diseño del componente de seguridad y el esquema de mensajería asíncrona para la recepción de órdenes:

- Algoritmo de Validación de Firmas: Se documentó detalladamente la lógica matemática y criptográfica requerida por SHEIN para la firma de peticiones HTTP en cada request, mapeando la estructura que deberá incluir el componente local SheinSignatureValidator para la prevención de ataques de suplantación de identidad.
- Homologación de Webhooks (Parent-Child / JIT Orders): Se estructuraron conceptualmente los esquemas JSON para el procesamiento automático de notificaciones de órdenes individuales o múltiples, listos para ser procesados bajo la

arquitectura de tareas en segundo plano (Queue Jobs) del sistema central.

c) Validación de Extensibilidad de la Arquitectura Central (Patrones de Diseño): El resultado más significativo de esta investigación fue la validación práctica de la viabilidad y escalabilidad de la arquitectura actual del software. Se demostró que la implementación previa de los patrones de diseño Factory e Interfaces para Mercado Libre y TikTok Shop dejó los canales de conexión completamente preparados para los nuevos marketplaces.

Como se detalla en la documentación técnica, la futura habilitación en código de Amazon y SHEIN no requerirá reestructurar la base de datos ni modificar el núcleo del SaaS; bastará con heredar las interfaces existentes (AuthServiceInterface y ProductSyncInterface) y registrar los nuevos casos en las fábricas lógicas (AuthFactory y WebhookFactory), reduciendo el tiempo de desarrollo futuro en un estimado del 80%.

7.5.4. Automatización e infraestructura (Cron Jobs)

Los Cron Jobs operaron sin interrupciones durante el período documentado. La configuración del Cron Job que utiliza RefreshPlatformTokens mantuvo vigentes las credenciales de ambas plataformas sin registrar ninguna interrupción de conexión por token expirado, en la Imagen 26 se muestra la configuración en cPanel. El procesador de cola ejecutado cada 60 segundos consumió la totalidad de los ProcessWebhookJob acumulados en cada ciclo.

Imagen 26. Crons configurados en cPanel

7.6. VALIDACIÓN DE LOS PROCESOS CRÍTICOS DEL SISTEMA

Como resultado de la ejecución de la estrategia de validación descrita en el apartado de metodología correspondiente, se confirmó el correcto funcionamiento de los procesos críticos del sistema. En el Anexo 4, se resume las pruebas realizadas por módulo, indicando la operación ejecutada, la acción realizada y el resultado obtenido.

Se confirmó que las operaciones CRUD de los módulos de administración (usuarios, tipos de usuario, módulos, recursos, privilegios, tipos de estatus y estatus) se ejecutaron sin errores, manteniendo la integridad referencial entre las entidades relacionadas (por ejemplo, entre recursos y módulos, o entre privilegios y recursos). Asimismo, se validó el funcionamiento de los módulos de catálogo (paquetes y productos) y del módulo operativo de pedidos, donde se comprobó la correcta creación, actualización del estatus, registro en bitácora (order_status_log) y manejo de lotes (order_details_lots).

En cuanto a los procesos de integración, se verificó que la sincronización de pedidos y productos de Mercado Libre y TikTok Shop se realizó de forma correcta, preservando la relación entre los identificadores externos (external_product_id, external_sku) y los registros del catálogo local. De igual forma, se validó la recepción y el procesamiento de webhooks de notificación de ambas plataformas, confirmando que los eventos (event_id, topic) y su contenido (payload) se almacenaron correctamente para su posterior procesamiento.

Finalmente, se validó la generación del documento PDF de pedido, comprobando que el reporte generado incluyera correctamente los datos de la plataforma de origen, el cliente, la fecha, el total y el detalle de productos del pedido correspondiente.

Los resultados obtenidos permiten concluir que los procesos críticos del sistema operan de manera estable y conforme a lo esperado, cumpliendo con el objetivo planteado de validar el funcionamiento de los módulos críticos mediante verificación funcional por consola y pruebas de integración con entornos sandbox.

7.7. DOCUMENTACIÓN TÉCNICA

Como resultado del proceso metodológico, se consolidó el compendio técnico indispensable para el despliegue, auditoría y auditoría futura de la plataforma.

7.7.1 Diagrama y Diccionario de Datos del SaaS

Se obtuvo un esquema relacional normalizado que mapea la arquitectura del sistema. El diccionario de datos describe exhaustivamente las tablas operativas (como orders, order_details), de infraestructura de sincronización (webhook_events, integration_logs) y de seguridad (users, privileges). Este documento proporciona una guía clara de cómo interactúa el inventario local de productos con los identificadores únicos multicanal de los marketplaces conectados.

7.7.2. Manuales Interactivos de Instalación y Configuración del Sistema

A través del uso de Tango AI, se generaron manuales interactivos de alta fidelidad visual que facilitan la adopción del sistema por parte de

administradores técnicos y usuarios operativos. Estos manuales se encuentran desplegados en la nube a través de los siguientes módulos accesibles:

- Guía Operativa de Plataformas e Integración de Pedidos: Concentra los pasos secuenciales para conectar las APIs externas, validar flujos asíncronos de webhooks y el procesamiento automatizado en lotes de órdenes. Enlace: https://app.tango.us/app/workflow/Platforms-and-Orders-e88a35330aab4338813d74b45aa29eb3
- Guía de Configuración y Catálogos de Control Interno: Documenta la parametrización inicial requerida por la plataforma en el módulo de Sistema, guiando paso a paso en el llenado de módulos esenciales como la creación de privilegios, estatus base del negocio y usuarios.
  Enlace 1: https://app.tango.us/app/workflow/M-dulos-del-Sistema-55da2771c95542d8a9f21298a5a508fa
  Enlace 2: https://app.tango.us/app/workflow/M-dulos-del-Sistema-2d4a0fa9134b49b9b2e320aa6e657d2a

7.7.3. Reporte y Bitácora de Pruebas Unitarias

El resultado de las pruebas demostró la robustez de la base de datos y la correcta lógica interna del backend en Laravel. Se generó un reporte detallado donde se adjuntan las capturas del comportamiento del sistema en entornos controlados independientes mediante Artisan Tinker, logrando un 100% de éxito en las operaciones CRUD simuladas.

CONCLUSIONES

A partir del desarrollo del proyecto de estadía realizado en Servicios en Desarrollos Informáticos del Bajío (SDIB), se presentan las conclusiones derivadas de los resultados obtenidos, en correspondencia con cada uno de los objetivos metodológicos planteados, el alcance definido y el objetivo general del informe.

Se analizaron los requerimientos funcionales y técnicos necesarios para el desarrollo del sistema mediante la revisión de las necesidades operativas de la organización y el estudio de la documentación oficial de cada marketplace. Este proceso permitió consolidar tres requerimientos funcionales, sincronización de pedidos en tiempo real, mapeo de estados unificado y control de acceso basado en roles, y tres requerimientos técnicos centrados en la arquitectura orientada a eventos, la estrategia de idempotencia y la seguridad en el almacenamiento de credenciales, los cuales guiaron las etapas subsecuentes del desarrollo.

Se diseñó la arquitectura del sistema y la estructura de base de datos requeridas para la administración centralizada de pedidos bajo un enfoque desacoplado que separa las responsabilidades entre el API Gateway, los Integration Workers y el Order Processor. La estructura de base de datos resultante quedó conformada por 29 tablas que vinculan cada pedido con su marketplace de origen, su estatus actualizado y el usuario responsable de su seguimiento, garantizando la consistencia e integridad referencial de la información.

Se implementaron los módulos de autenticación, administración de usuarios y control de permisos mediante el estándar JWT para la gestión de sesiones sin estado y el modelo RBAC para la diferenciación de accesos. Se definieron tres perfiles de usuario, Administrador, Editor y Observador, cada uno

asociado a un conjunto específico de permisos que restringe las funcionalidades disponibles según las responsabilidades asignadas, logrando un esquema de seguridad perimetral interna acorde con los requerimientos definidos.

Se desarrolló el dashboard centralizado que permite la visualización y administración del estatus de pedidos provenientes de Mercado Libre y TikTok Shop desde una interfaz unificada. La plataforma presenta los pedidos en una tabla interactiva con filtrado por marketplace, estatus y rango de fechas, e integra indicadores de resumen en tiempo real que facilitan la identificación del estado general de las operaciones de SDIB sin necesidad de consultar cada canal de forma independiente.

Se integró de forma funcional la API de Mercado Libre, procesando más de 20 pedidos mediante webhook con un mecanismo de idempotencia que bloqueó la totalidad de los intentos de inserción duplicada, y alcanzando una tasa de sincronización de productos del 80 %. La integración con TikTok Shop procesó más de 10 pedidos con validación HMAC-SHA256 y una sincronización de productos del 100 %. Adicionalmente, se desarrollaron las guías técnicas y el diseño arquitectónico para la futura integración de Amazon SP-API y Shein Open Platform, validando que la arquitectura basada en interfaces y patrones Factory permite incorporar ambas plataformas sin modificar el núcleo del sistema.

Se validó el funcionamiento de los módulos críticos del sistema mediante la ejecución de casos de prueba CRUD sobre los nueve módulos centrales de la plataforma utilizando Laravel Tinker, obteniendo un resultado exitoso del 100 % en las operaciones de creación, consulta, actualización y eliminación lógica. Las pruebas de integración realizadas con cuentas sandbox de Mercado Libre y TikTok Shop confirmaron el correcto funcionamiento de los flujos de webhooks, la renovación automatizada de tokens y la sincronización de productos en condiciones equivalentes a producción.

Se elaboró la documentación técnica del sistema conformada por el diccionario de datos de las 29 tablas, los manuales interactivos de instalación y configuración generados mediante Tango AI, el reporte de validación funcional por módulo, y las guías técnicas de integración para Amazon y Shein incorporadas como Anexos 2 y 3. Este compendio proporciona la base necesaria para el despliegue, mantenimiento y expansión futura de la plataforma.

De manera general, el proyecto logró el objetivo planteado: se desarrolló una plataforma SaaS de gestión centralizada de pedidos para marketplaces, construida sobre el framework Laravel, que integra de forma funcional los canales de Mercado Libre y TikTok Shop y establece el diseño técnico para la incorporación de Amazon y Shein. Los resultados obtenidos corresponden con el alcance definido para la estadía y demuestran la viabilidad de la solución como producto tecnológico escalable orientado a la administración multicanal de ventas digitales en SDIB.

REFERENCIAS

Amazon Selling Partner API. (2024). *Onboarding as a Developer.* https://developer-docs.amazon/sp-api/docs/onboarding-overview

Amazon Web Services. (2026a). *AWS overview. General SAP Guides.* https://docs.aws.amazon.com/sap/latest/general/overview-aws.html

Amazon Web Services. (2026b). *Global infrastructure.* En Overview of Amazon Web Services. https://docs.aws.amazon.com/whitepapers/latest/aws-overview/global-infrastructure.html

Amazon Web Services. (2026c). *¿Qué es la computación en nube?* https://aws.amazon.com/es/what-is-cloud-computing/

Amazon Web Services. (2026d). *Infraestructura global de AWS.* https://aws.amazon.com/es/about-aws/global-infrastructure/

Fowler, M. (2017, 7 de febrero). *What do you mean by "Event-Driven"?* MartinFowler.com. https://martinfowler.com/articles/201701-event-driven.html

Google Cloud. (s. f.). *¿Qué es la idempotencia? Guía para la confiabilidad de las APIs.* https://cloud.google.com/discover/idempotency?hl=es-419

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT) (RFC 7519).* Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc7519

Laravel. (2026a). *Installation | Laravel 13.x - The clean stack for Artisans and agents.* https://laravel.com/docs/13.x#meet-laravel

Laravel. (2026b). *Queues | Laravel 13.x - The clean stack for Artisans and agents.* https://laravel.com/docs/13.x/queues

Laravel. (2026c). *Queues | Laravel 13.x - The clean stack for Artisans and agents.* https://laravel.com/docs/13.x/queues#creating-jobs

Lindemulder, G. (s. f.). *¿Qué es el control de acceso basado en roles (RBAC)? Think | IBM.* https://www.ibm.com/mx-es/think/topics/rbac

MariaDB Documentation. (2025). *About MariaDB.* https://mariadb.com/docs/general-resources/about/about-mariadb

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing (Special Publication 800-145).* National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

MercadoLibre S.R.L. (2026). *Documentation Mercado Libre.* API docs. https://global-selling.mercadolibre.com/devsite/en_us/design-considerations-global-selling

Microsoft Azure. (2026). *¿Qué es el software como servicio (SaaS)? Microsoft.* https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-saas

National Institute of Standards and Technology. (2001). *Advanced Encryption Standard (AES) (Federal Information Processing Standards Publication 197).* U.S. Department of Commerce. https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

SDIB. (s. f.). *Misión, visión y valores.* Servicios en Desarrollos Informáticos del Bajío. https://sdib.com.mx/pages/mision-vision.html

SHEIN Developer Platform. (s. f.). *Solution Introduction.* https://open.sheincorp.com/documents/system/efc12d67-1c00-452d-8907-30eecd93106a

Shopify en español. (2024, 18 de noviembre). *¿Qué es la venta multicanal? Beneficios clave + consejos para empezar.* Blog de Shopify. https://www.shopify.com/es/blog/venta-multicanal

Tailwind Labs Inc. (2026). *Styling with utility classes - Core concepts.* tailwindcss. https://tailwindcss.com/docs

TikTok Shop Partner Center. (2025). *Seller authorization guide.* https://partner.tiktokshop.com/docv2/page/seller-authorization-guide

Tinkerwell. (s. f.). *About Tinkerwell*. https://tinkerwell.app/docs/5/getting-started/about

GLOSARIO

AES-256 (Advanced Encryption Standard de 256 bits): Esquema de cifrado simétrico de nivel militar utilizado para proteger y encriptar datos altamente sensibles en la base de datos.

API (Application Programming Interface): Conjunto de definiciones y protocolos que permite que dos aplicaciones de software se comuniquen e intercambien datos entre sí.

Cron Job (Tarea Programada): Administrador de procesos que ejecuta comandos o tareas automáticamente en intervalos de tiempo predefinidos desde el servidor.

Firma de Autenticación (Signature Verification): Mecanismo criptográfico de seguridad en las cabeceras de los webhooks que permite verificar que la notificación proviene realmente del marketplace oficial y no de un atacante tratando de suplantar identidades.

Idempotencia: Propiedad de un algoritmo que garantiza que, aunque una misma operación o petición se ejecute múltiples veces, el resultado final no cambie ni genere registros duplicados.

JIT Orders (Just-In-Time Orders): Flujo logístico donde el pedido se procesa y surte de manera inmediata bajo demanda del cliente, requiriendo un mapeo JSON ágil.

JWT (JSON Web Token): Estándar abierto basado en JSON para la creación de tokens de acceso compactos y autónomos.

Laravel: Framework de desarrollo web en PHP estructurado bajo el patrón MVC.

MariaDB: Sistema de gestión de bases de datos relacionales utilizado para el almacenamiento persistente y estructurado de los pedidos, productos y usuarios.

Marketplace: Plataforma en línea que actúa como intermediaria entre compradores y múltiples vendedores independientes.

Middleware: Capa de software que actúa como puente o intermediario entre diferentes aplicaciones, sistemas o bases de datos.

Multicanal: Arquitectura de software en la que una sola instancia de la aplicación sirve a múltiples clientes, garantizando el aislamiento seguro de los datos de cada uno.

Payload: Conjunto de datos útiles transportados dentro de una petición HTTP o un webhook.

PHP Artisan Tinker: Entorno interactivo de consola de comandos nativo de Laravel que permite probar y validar flujos de código, modelos y bases de datos manualmente sin necesidad de una interfaz gráfica.

Polling: Técnica de consulta periódica donde el cliente realiza peticiones recurrentes a un servidor API cada cierto intervalo de tiempo para verificar si hay datos nuevos.

Queue Jobs (Colas de Procesamiento): Arquitectura que desplaza tareas pesadas de manera asíncrona a segundo plano, evitando que el servidor se sature o experimente pérdida de transacciones comerciales ante altas cargas de datos.

SaaS (Software as a Service / Software como Servicio): Modelo de distribución de software donde el soporte y los datos se alojan en servidores de un proveedor y los clientes acceden a ellos a través de Internet.

SKU (Stock Keeping Unit): Código identificador único (alfanumérico) asignado internamente a cada producto para el control de inventario.

SP-API (Selling Partner API): La infraestructura de API moderna y basada en REST de Amazon.

Tailwind CSS: Framework de diseño visual basado en clases de utilidad (*utility-first*) empleado para la construcción de una interfaz de usuario responsiva y fluida.

Token (Access Token / Refresh Token): Credenciales digitales temporales que permiten a un sistema realizar peticiones seguras y autorizadas en nombre de las tiendas enlazadas sin exponer sus contraseñas reales.

Webhook: Mecanismo de comunicación asíncrona mediante el cual un servidor externo envía una notificación HTTP automática a tu sistema en respuesta a un evento en tiempo real.

ANEXOS

Anexo 1. Diccionario de datos

El Anexo 1 fue generado en el entorno de desarrollo con MySQL, y la estructura es compatible con el entorno de producción MariaDB sin modificaciones al esquema.

SERVICIOS EN DESARROLLOS INFORMÁTICOS DEL BAJÍO

Diccionario de Datos

Sistema de Gestión de Pedidos

Última actualización: 03/07/2026
Versión: 2.0
Responsable: Montserrat Martínez

SDIB

Tabla de contenido

1. Introducción

Un diccionario de datos es una colección de metadatos como el nombre del objeto, el tipo de datos, el tamaño, la clasificación y las relaciones con otros activos de datos. Lo utilizan administradores de datos, analistas e ingenieros para comprender y creer en los activos de datos. Ayuda en la creación de datos auténticos, transparentes, consistentes y sólidos para toda la organización.

Para cada servidor, las definiciones de tablas (entidades/colecciones) se enumeran en orden. Las propiedades detalladas de los elementos de datos (tipo de datos, tamaño, nulalidad, opcionalidad, índices, claves externas, restricciones) están organizadas en formato tabular.

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2. Servidor: sdib.com.mx_3306 (MySQL)

Versión: 5.7.44-48

1 base de datos se enumera a continuación.

2.1. Base de datos: sdibinfo_market

Objetos de base de datos que se enumerarán: 29 Tablas

2.1.1. Tablas

2.1.1.1. Tabla: assigned_users

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	user_id	bigint(20)	✓	Índice: assigned_users_user_id_for...
3	plataform_id	bigint(20)	✓	Índice: assigned_users_plataform_...
4	created_by	bigint(20)		Índice: assigned_users_created_by...
5	updated_by	bigint(20)		Índice: assigned_users_updated_b...
6	deleted_at	timestamp		
7	created_at	timestamp		
8	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
assigned_users_user_id_foreign	NORMAL	BTREE	user_id
assigned_users_plataform_id_foreign	NORMAL	BTREE	plataform_id

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

assigned_users_created_by_foreign	NORMAL	BTREE	created_by
assigned_users_updated_by_foreign	NORMAL	BTREE	updated_by

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
assigned_users_created_by_foreign	created_by	sdibinfo_market.users	id	RESTRICT	RESTRICT
assigned_users_plataform_id_foreign	plataform_id	sdibinfo_market.plataforms	id	CASCADE	RESTRICT
assigned_users_updated_by_foreign	updated_by	sdibinfo_market.users	id	RESTRICT	RESTRICT
assigned_users_user_id_foreign	user_id	sdibinfo_market.users	id	CASCADE	RESTRICT

2.1.1.2. Tabla: cache

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	key 🔑	varchar(191)	✓	
2	value	mediumtext	✓	
3	expiration	int(11)	✓	
4	deleted_at	timestamp		

2.1.1.3. Tabla: cache_locks

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	key 🔑	varchar(191)	✓	
2	owner	varchar(191)	✓	
3	expiration	int(11)	✓	
4	deleted_at	timestamp		

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2.1.1.4. Tabla: clients

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	code	varchar(50)	✓	
3	name	varchar(255)	✓	
4	address	varchar(255)	✓	
5	neighborhood	varchar(255)	✓	
6	city	varchar(255)	✓	
7	state	varchar(255)	✓	
8	rfc	varchar(50)	✓	Índice: clients_rfc_unique
9	zip_code	varchar(20)	✓	
10	created_id	bigint(20)	✓	Índice: clients_created_id_foreign
11	updated_id	bigint(20)	✓	Índice: clients_updated_id_foreign
12	deleted_at	timestamp		
13	created_at	timestamp		
14	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
clients_rfc_unique	UNIQUE	BTREE	rfc
clients_created_id_foreign	NORMAL	BTREE	created_id
clients_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
clients_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
clients_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

4

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2.1.1.5. Tabla: failed_jobs

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	uuid	varchar(191)	✓	Índice: failed_jobs_uuid_unique
3	connection	text	✓	
4	queue	text	✓	
5	payload	longtext	✓	
6	exception	longtext	✓	
7	failed_at	timestamp	✓	Predeterminado: CURRENT_TIMESTAMP
8	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
failed_jobs_uuid_unique	UNIQUE	BTREE	uuid

2.1.1.6. Tabla: integration_logs

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	platform	varchar(191)	✓	
3	type	varchar(191)	✓	
4	event_id	varchar(191)		
5	payload	json		
6	headers	json		
7	response	json		
8	error	text		
9	created_at	timestamp		
10	updated_at	timestamp		

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2.1.1.7. Tabla: job_batches

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id	varchar(191)	✓	
2	name	varchar(191)	✓	
3	total_jobs	int(11)	✓	
4	pending_jobs	int(11)	✓	
5	failed_jobs	int(11)	✓	
6	failed_job_ids	longtext	✓	
7	options	mediumtext		
8	cancelled_at	int(11)		
9	created_at	int(11)	✓	
10	finished_at	int(11)		
11	deleted_at	timestamp		

2.1.1.8. Tabla: jobs

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id	bigint(20) Auto incremento	✓	
2	queue	varchar(191)	✓	Índice: jobs_queue_index
3	payload	longtext	✓	
4	attempts	tinyint(3)	✓	
5	reserved_at	int(10)		
6	available_at	int(10)	✓	
7	created_at	int(10)	✓	
8	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
jobs_queue_index	NORMAL	BTREE	queue

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2.1.1.9. Tabla: migrations

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	int(10) Auto incremento	✓	
2	migration	varchar(191)	✓	
3	batch	int(11)	✓	

2.1.1.10. Tabla: modules

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	
3	created_at	timestamp		
4	updated_at	timestamp		
5	deleted_at	timestamp		

2.1.1.11. Tabla: order_detail_lots

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	order_detail_id	bigint(20)	✓	Índice: order_detail_lots_order_de...
3	quantity	int(11)	✓	
4	created_id	bigint(20)	✓	Índice: order_detail_lots_created_i...
5	updated_id	bigint(20)	✓	Índice: order_detail_lots_updated_...
6	deleted_at	timestamp		

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

7	created_at	timestamp		
8	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
order_detail_lots_order_detail_id_unique	UNIQUE	BTREE	order_detail_id
order_detail_lots_created_id_foreign	NORMAL	BTREE	created_id
order_detail_lots_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
order_detail_lots_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
order_detail_lots_order_detail_id_foreign	order_detail_id	sdibinfo_market.order_details	id	CASCADE	RESTRICT
order_detail_lots_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.12. Tabla: order_details

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id	bigint(20) Auto incremento	✓	
2	order_id	bigint(20)	✓	Índice: order_details_order_id_fore...
3	product_id	bigint(20)	✓	Índice: order_details_product_id_f...
4	price	decimal(10, 2)	✓	

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

5	quantity	int(11)	✓	
6	total	decimal(10, 2)	✓	
7	lotted_quantity	int(11)	✓	Predeterminado: 0
8	created_id	bigint(20)	✓	Índice: order_details_created_id_fo...
9	updated_id	bigint(20)	✓	Índice: order_details_updated_id_f...
10	deleted_at	timestamp		
11	created_at	timestamp		
12	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
order_details_order_id_foreign	NORMAL	BTREE	order_id
order_details_product_id_foreign	NORMAL	BTREE	product_id
order_details_created_id_foreign	NORMAL	BTREE	created_id
order_details_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
order_details_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
order_details_order_id_foreign	order_id	sdibinfo_market.orders	id	CASCADE	RESTRICT
order_details_product_id_foreign	product_id	sdibinfo_market.products	id	CASCADE	RESTRICT
order_details_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2.1.1.13. Tabla: order_status_logs

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	order_id	bigint(20)	✓	Índice: order_status_logs_order_id...
3	status_id	bigint(20)	✓	Índice: order_status_logs_status_id...
4	created_id	bigint(20)	✓	Índice: order_status_logs_created_...
5	updated_id	bigint(20)	✓	Índice: order_status_logs_updated...
6	deleted_at	timestamp		
7	created_at	timestamp		
8	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
order_status_logs_order_id_foreign	NORMAL	BTREE	order_id
order_status_logs_status_id_foreign	NORMAL	BTREE	status_id
order_status_logs_created_id_foreign	NORMAL	BTREE	created_id
order_status_logs_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
order_status_logs_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

order_status_logs_order_id_foreign	order_id	sdibinfo_market.orders	id	CASCADE	RESTRICT
order_status_logs_status_id_foreign	status_id	sdibinfo_market.statuses	id	CASCADE	RESTRICT
order_status_logs_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.14. Tabla: orders

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id	bigint(20) Auto incremento	✓	
2	folio	varchar(191)	✓	Índice: orders_folio_unique
3	folio_platform	varchar(191)		
4	plataform_id	bigint(20)	✓	Índice: orders_plataform_id_foreig...
5	status_id	bigint(20)	✓	Índice: orders_status_id_foreign
6	client_id	bigint(20)		Índice: orders_client_id_foreign
7	total	decimal(10, 2)	✓	
8	lot_user_id	bigint(20)		Índice: orders_lot_user_id_foreign
9	created_id	bigint(20)	✓	Índice: orders_created_id_foreign
10	updated_id	bigint(20)	✓	Índice: orders_updated_id_foreign
11	deleted_at	timestamp		
12	created_at	timestamp		
13	updated_at	timestamp		

Índices

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

Nombre	Tipo	Método	Campos
orders_folio_unique	UNIQUE	BTREE	folio
orders_plataform_id_foreign	NORMAL	BTREE	plataform_id
orders_status_id_foreign	NORMAL	BTREE	status_id
orders_client_id_foreign	NORMAL	BTREE	client_id
orders_lot_user_id_foreign	NORMAL	BTREE	lot_user_id
orders_created_id_foreign	NORMAL	BTREE	created_id
orders_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
orders_client_id_foreign	client_id	sdibinfo_market.clients	id	CASCADE	RESTRICT
orders_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
orders_lot_user_id_foreign	lot_user_id	sdibinfo_market.users	id	SET NULL	RESTRICT
orders_plataform_id_foreign	plataform_id	sdibinfo_market.plataforms	id	CASCADE	RESTRICT
orders_status_id_foreign	status_id	sdibinfo_market.statuses	id	CASCADE	RESTRICT
orders_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.15. Tabla: packagings

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	
3	code	int(11)	✓	

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

4	code_sat	varchar(50)	✓	
5	created_id	bigint(20)	✓	Índice: packagings_created_id_fore...
6	updated_id	bigint(20)	✓	Índice: packagings_updated_id_for...
7	deleted_at	timestamp		
8	created_at	timestamp		
9	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
packagings_created_id_foreign	NORMAL	BTREE	created_id
packagings_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
packagings_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
packagings_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.16. Tabla: password_reset_tokens

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	email 🔑	varchar(191)	✓	
2	token	varchar(191)	✓	
3	created_at	timestamp		
4	deleted_at	timestamp		

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2.1.1.17. Tabla: personal_access_tokens

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	tokenable_type	varchar(191)	✓	Índice: personal_access_tokens_to...
3	tokenable_id	bigint(20)	✓	Índice: personal_access_tokens_to...
4	name	text	✓	
5	token	varchar(64)	✓	Índice: personal_access_tokens_to...
6	abilities	text		
7	last_used_at	timestamp		
8	expires_at	timestamp		Índice: personal_access_tokens_ex...
9	created_at	timestamp		
10	updated_at	timestamp		
11	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
personal_access_tokens_token_unique	UNIQUE	BTREE	token
personal_access_tokens_tokenable_type_tokenable_id_index	NORMAL	BTREE	tokenable_type, tokenable_id
personal_access_tokens_expires_at_index	NORMAL	BTREE	expires_at

2.1.1.18. Tabla: plataform_products

Campos

Pos.	Nombre	Tipo	No Nulo	Otros

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

1	id 🔑	bigint(20) Auto incremento	✓	
2	plataform_id	bigint(20)	✓	Índice: plataform_products_platafo...
3	product_id	bigint(20)		Índice: plataform_products_produc...
4	external_product_id	varchar(191)	✓	
5	external_sku	varchar(191)		
6	price	decimal(10, 2)		
7	created_id	bigint(20)	✓	Índice: plataform_products_create...
8	updated_id	bigint(20)	✓	Índice: plataform_products_update...
9	created_at	timestamp		
10	updated_at	timestamp		
11	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
plataform_products_plataform_id_foreign	NORMAL	BTREE	plataform_id
plataform_products_created_id_foreign	NORMAL	BTREE	created_id
plataform_products_updated_id_foreign	NORMAL	BTREE	updated_id
plataform_products_product_id_foreign	NORMAL	BTREE	product_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar

15

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

plataform_products_created_id_foreign	created_id	sdibinfo_market.users	id	RESTRICT	RESTRICT
plataform_products_plataform_id_foreign	plataform_id	sdibinfo_market.plataforms	id	CASCADE	RESTRICT
plataform_products_product_id_foreign	product_id	sdibinfo_market.products	id	SET NULL	RESTRICT
plataform_products_updated_id_foreign	updated_id	sdibinfo_market.users	id	RESTRICT	RESTRICT

2.1.1.19. Tabla: plataforms

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	
3	status_id	bigint(20)		Índice: plataforms_status_id_foreig...
4	slug	varchar(191)		
5	color	varchar(7)		Predeterminado: '#FFFFFF'
6	icon_path	varchar(191)		
7	active	tinyint(4)	✓	Predeterminado: 1
8	oauth_url	varchar(191)		
9	auth_url	varchar(191)		
10	refresh_url	varchar(191)		
11	client_id	varchar(191)		
12	service_id	varchar(191)		
13	client_secret	text		
14	redirect_uri	varchar(191)		

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

15	access_token	text		
16	refresh_token	text		
17	expires_at	timestamp		
18	deleted_at	timestamp		
19	created_id	bigint(20)	✓	Índice: plataforms_created_id_fore...
20	updated_id	bigint(20)	✓	Índice: plataforms_updated_id_for...
21	created_at	timestamp		
22	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
plataforms_created_id_foreign	NORMAL	BTREE	created_id
plataforms_updated_id_foreign	NORMAL	BTREE	updated_id
plataforms_status_id_foreign	NORMAL	BTREE	status_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
plataforms_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
plataforms_status_id_foreign	status_id	sdibinfo_market.statuses	id	SET NULL	RESTRICT
plataforms_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.20. Tabla: privileges

Campos

Pos.	Nombre	Tipo	No Nulo	Otros

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

1	id 🔑	bigint(20) Auto incremento	✓	
2	resource_id	bigint(20)	✓	Índice: privileges_resource_id_fore...
3	name	varchar(191)	✓	
4	created_at	timestamp		
5	updated_at	timestamp		
6	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
privileges_resource_id_foreign	NORMAL	BTREE	resource_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
privileges_resource_id_foreign	resource_id	sdibinfo_market.resources	id	CASCADE	RESTRICT

2.1.1.21. Tabla: products

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	
3	code	varchar(191)	✓	
4	code_sat	varchar(50)	✓	
5	packaging_id	bigint(20)	✓	Índice: products_packaging_id_for...
6	created_id	bigint(20)	✓	Índice: products_created_id_foreig...

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

7	updated_id	bigint(20)	✓	Índice: products_updated_id_forei...
8	deleted_at	timestamp		
9	created_at	timestamp		
10	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
products_packaging_id_foreign	NORMAL	BTREE	packaging_id
products_created_id_foreign	NORMAL	BTREE	created_id
products_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
products_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
products_packaging_id_foreign	packaging_id	sdibinfo_market.packagings	id	CASCADE	RESTRICT
products_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.22. Tabla: resources

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	module_id	bigint(20)	✓	Índice: resources_module_id_forei...
3	name	varchar(191)	✓	
4	created_at	timestamp		
5	updated_at	timestamp		
6	deleted_at	timestamp		

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

Índices

Nombre	Tipo	Método	Campos
resources_module_id_foreign	NORMAL	BTREE	module_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
resources_module_id_foreign	module_id	sdibinfo_market.modules	id	CASCADE	RESTRICT

2.1.1.23. Tabla: sessions

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	varchar(191)	✓	
2	user_id	bigint(20)		Índice: sessions_user_id_index
3	ip_address	varchar(45)		
4	user_agent	text		
5	payload	longtext	✓	
6	last_activity	int(11)	✓	Índice: sessions_last_activity_index
7	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
sessions_user_id_index	NORMAL	BTREE	user_id
sessions_last_activity_index	NORMAL	BTREE	last_activity

2.1.1.24. Tabla: status_types

Campos

Pos.	Nombre	Tipo	No Nulo	Otros

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

1	id	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	
3	deleted_at	timestamp		
4	created_id	bigint(20)	✓	Índice: status_types_created_id_fo...
5	updated_id	bigint(20)	✓	Índice: status_types_updated_id_fo...
6	created_at	timestamp		
7	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
status_types_created_id_foreign	NORMAL	BTREE	created_id
status_types_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
status_types_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
status_types_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.25. Tabla: statuses

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

3	order	int(11)		
4	status_type_id	bigint(20)	✓	Índice: statuses_status_type_id_fo...
5	created_id	bigint(20)	✓	Índice: statuses_created_id_foreign
6	updated_id	bigint(20)	✓	Índice: statuses_updated_id_foreig...
7	deleted_at	timestamp		
8	created_at	timestamp		
9	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
statuses_status_type_id_foreign	NORMAL	BTREE	status_type_id
statuses_created_id_foreign	NORMAL	BTREE	created_id
statuses_updated_id_foreign	NORMAL	BTREE	updated_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
statuses_created_id_foreign	created_id	sdibinfo_market.users	id	CASCADE	CASCADE
statuses_status_type_id_foreign	status_type_id	sdibinfo_market.status_types	id	CASCADE	RESTRICT
statuses_updated_id_foreign	updated_id	sdibinfo_market.users	id	CASCADE	CASCADE

2.1.1.26. Tabla: user_type_privilege

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	user_type_id 🔑	bigint(20)	✓	

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2	privilege_id	bigint(20)	✓	Índice: user_type_privilege_privileg...
3	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
user_type_privilege_privilege_id_foreign	NORMAL	BTREE	privilege_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
user_type_privilege_privilege_id_foreign	privilege_id	sdibinfo_market.privileges	id	CASCADE	RESTRICT
user_type_privilege_user_type_id_foreign	user_type_id	sdibinfo_market.user_types	id	CASCADE	RESTRICT

2.1.1.27. Tabla: user_types

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	Índice: user_types_name_unique
3	description	varchar(191)		
4	created_at	timestamp		
5	updated_at	timestamp		
6	deleted_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
user_types_name_unique	UNIQUE	BTREE	name

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

2.1.1.28. Tabla: users

Campos

Pos.	Nombre	Tipo	No Nulo	Otros
1	id 🔑	bigint(20) Auto incremento	✓	
2	name	varchar(191)	✓	
3	email	varchar(191)	✓	Índice: users_email_unique
4	email_verified_at	timestamp		
5	password	varchar(191)	✓	
6	remember_token	varchar(100)		
7	created_at	timestamp		
8	updated_at	timestamp		
9	deleted_at	timestamp		
10	two_factor_secret	varchar(191)		
11	two_factor_enabled	tinyint(1)	✓	Predeterminado: 0
12	user_type_id	bigint(20)		Índice: users_user_type_id_foreign

Índices

Nombre	Tipo	Método	Campos
users_email_unique	UNIQUE	BTREE	email
users_user_type_id_foreign	NORMAL	BTREE	user_type_id

Clave Foraneas

Nombre	Campos	Tabla Referida	Campos Referidos	En Eliminar	En actualizar
users_user_type_id_foreign	user_type_id	sdibinfo_market.user_types	id	RESTRICT	RESTRICT

2.1.1.29. Tabla: webhook_events

Campos

Pos.	Nombre	Tipo	No Nulo	Otros

Servidor: sdib.com.mx_3306 (MySQL) > Base de datos: sdibinfo_market

1	id 🔑	bigint(20) Auto incremento	✓	
2	platform	varchar(191)		
3	event_id	varchar(191)	✓	Índice: webhook_events_event_id_...
4	topic	varchar(191)		
5	payload	json	✓	
6	processed_at	timestamp		
7	deleted_at	timestamp		
8	created_at	timestamp		
9	updated_at	timestamp		

Índices

Nombre	Tipo	Método	Campos
webhook_events_event_id_unique	UNIQUE	BTREE	event_id

25

Anexo 2. Guía de integración con Amazon SP-API

Guía de integración con Amazon SP-API

En el presente documento, se explica como realizar la integración del marketplace Amazon.
Documentación oficial:

https://developer.amazon.com/docs/app-submission/manage-account-and-permissions.html

https://developer.amazon.com/docs/app-submission/manage-account-and-permissions.html

https://aws.amazon.com/es/sqs/getting-started/

Creación de la Aplicación en Amazon (Obtención de Credenciales)

Amazon utiliza el **Selling Partner API (SP-API)**. A diferencia de otras plataformas, requiere tanto una cuenta en **Amazon Seller Central** como en **AWS (Amazon Web Services)** por cuestiones de seguridad. Los pasos para obtener tus credenciales desde cero son:

1. **Registrar el Perfil de Desarrollador en Seller Central:**
   - Inicia sesión en la cuenta de Amazon Seller Central.
   - Ve al menú (hamburguesa) ➡️ **Apps y servicios** (Partner Network) ➡️ **Desarrollar apps** (Develop Apps).
   - Completa el **Developer Profile** seleccionando la opción de **Public Developer** (ya que se está construyendo un SaaS para que múltiples tiendas/usuarios se conecten).
   - Selecciona los **Roles** necesarios para el negocio (ej. *Product Listing* para productos y *Direct Consumer Shipping* u *Orders* para traerte los pedidos). *Nota: Los roles que acceden a información confidencial del cliente (PII) pasan por una revisión de seguridad más estricta por parte de Amazon.*
2. **Configuración de Seguridad en AWS (IAM Role):**
   - Crea una cuenta gratuita en **AWS**.
   - Ve al servicio **IAM (Identity and Access Management)** y crea un **Usuario IAM** con accesos programáticos. Obtén su *Access Key ID* y *Secret Access Key*.
   - Crea una **Política IAM** personalizada que permita la acción `sts:AssumeRole`.
   - Crea un **Rol IAM** (IAM Role) y añade una relación de confianza con el ARN de la entidad de Amazon SP-API (`sellingpartnerapi-na.amazon.com` o la de su región). Copia el **ARN del Rol**, lo necesitarás en el siguiente paso.
3. **Registrar la Aplicación en Seller Central:**
   - Regresa a Seller Central ➡️ **Desarrollar apps**.
   - Haz clic en **Add new app client** (Añadir nuevo cliente de aplicación).
   - Agrega un nombre a la aplicación, selecciona el tipo de API (**SP-API**) e introduce el **ARN del Rol IAM** que creaste en AWS.
   - Una vez creada, Amazon otorgará las credenciales de **LWA (Login with Amazon)**:

- **Client ID** (ID de cliente)
- **Client Secret** (Secreto de cliente)

Implementación en el proyecto

1. Capa de Autenticación y API Base (`app>Services>Platforms`)

Amazon utiliza OAuth2 mediante su servidor **LWA**. Se necesitará crear los servicios correspondientes y vincularlos al Factory:

- `AmazonAuthService.php` : Debe implementar `AuthServiceInterface` . Se encargará de intercambiar el código de autorización inicial por un `refresh_token` de Amazon, así como de renovar el `access_token` de manera horaria (el cual expira cada 60 minutos).

▼ **Validación de Firma de Autenticación (Amazon AWS / SP-API)**

En la arquitectura, se cuenta con un directorio dedicado a la seguridad llamado `app>Services>Validators`, donde se almacena `MercadoLibreSignatureValidator.php` y `TikTokSignatureValidator.php`. Para Amazon, se deberá añadir `AmazonSignatureValidator.php`.

Sin embargo, a diferencia de Mercado Libre o TikTok que envían un hash simple en los headers (`X-Signature`), Amazon SP-API utiliza la **firma AWS Signature Version 4 (SigV4)** o bien confía en la firma criptográfica nativa de sus servicios si se usa un destino como **Amazon SQS** para recibir los mensajes.

Como los eventos de Amazon se extraen de una cola SQS utilizando una conexión cifrada con las credenciales de AWS (Access Key y Secret Key), la "validación de firma" del webhook muta: en lugar de validar un HTTP Header expuesto a internet, lo que se hace es validar la autenticidad y verificar el hash MD5/SHA256 del payload del mensaje que viene de la SDK de AWS.

Si se desea validar manualmente una petición directa de Amazon o el payload firmado del webhook, el validador debe lucir así:

```php
namespace App\Services\Validators;

class AmazonSignatureValidator
{
    public function validate($payload, $signatureHeader): bool
    {
        // Amazon firma sus mensajes usando claves públicas rotativas (X
-Amz-Signature).
        // 1. Decodificar el JSON del mensaje de SQS.
        // 2. Verificar que el campo "Type" sea "Notification".
        // 3. Confirmar que el "SigningCertURL" provenga de un dominio o
ficial de Amazon (sns.amazonaws.com o amazon.com).
        // 4. Descargar el certificado público y verificar el 'Signatur
e' con openssl_verify().

        $certUrl = $payload['SigningCertURL'] ?? '';
        if (!str_ends_with(parse_url($certUrl, PHP_URL_HOST), 'amazonaw
s.com')) {
            return false;
        }

        // Simulación o verificación nativa de OpenSSL
        return true;
    }
}
```

- `AmazonApiService.php`: Contendrá las peticiones HTTP directas hacia los endpoints de Amazon (por ejemplo, `GET /orders/v0/orders` para traer la lista de pedidos).

▼ **Estructuras JSON**

Webhook de Amazon (Pedidos)

Cuando ocurre un evento en Amazon (por ejemplo, el evento `ORDER_CHANGE`), el JSON que llega a la cola SQS y que posteriormente procesará el `AmazonWebhookService` (para despacharlo a `ProcessWebhookJob`) contiene la información del cambio de estado del pedido.

Ejemplo real estandarizado de un webhook de Amazon SP-API

```
{
  "NotificationMetaData": {
    "NotificationType": "ORDER_CHANGE",
    "PayloadVersion": "1.0",
    "UniqueId": "9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d"
  },
  "Payload": {
    "OrderChangeNotification": {
      "SellerId": "A3NEXAMPLERETAIL",
      "AmazonOrderId": "123-1234567-1234567",
      "PurchaseDate": "2026-07-06T18:30:00Z",
      "LastUpdateDate": "2026-07-06T18:35:00Z",
      "OrderStatus": "Unshipped",
      "DestinationPostalCode": "44600",
      "FulfillmentType": "AFN",
      "Summary": {
        "OrderTotal": {
          "CurrencyCode": "MXN",
          "Amount": "450.00"
        },
        "NumberOfItemsShipped": 0,
        "NumberOfItemsUnshipped": 1
      }
    }
  }
}
```

Nota: `FulfillmentType: AFN` significa que es enviado por Amazon (FBA), mientras que `MFN` significa que lo envía el vendedor directamente (FBM).

Productos (Amazon SP-API)

Cuando el `AmazonProductSyncService` (que hereda el `ProductSyncInterface`) mande a llamar al endpoint `GET /products/pricing/v0/items/{Asin}/offers`, se obtiene un JSON enfocado en listados, SKUs y ASINs (el identificador único de Amazon).

Este es el JSON típico que consume el servicio para mapear a la tabla `plataform_products`:

```
{
  "payload": {
    "ASIN": "B07XEXAMPLE",
    "MarketplaceId": "A1AM78C64UM0Y8",
    "Summary": {
      "TotalSupplyQuantity": 45,
      "ListPrice": {
        "Amount": 299.99,
        "CurrencyCode": "MXN"
      }
    },
    "Offers": [
      {
        "SellerSKU": "PROD-AMZ-001",
        "Price": {
          "Amount": 299.99,
          "CurrencyCode": "MXN"
        },
        "AvailableQuantity": 15,
        "ItemCondition": "New"
      }
    ]
  }
}
```


- **Actualizar** `AuthFactory.php`: Añadir el caso de uso para Amazon

2. Capa de Sincronización de Productos (`app>Services>Integrations`)

Para mapear los productos de Amazon (con sus ASIN / SKU) hacia el inventario local:

- `AmazonProductSyncService.php`: Esta clase implementará `ProductSyncInterface`. Donde estará la lógica para consultar los listados de Amazon y estandarizar la información al formato que la base de datos entiende.

3. Capa de Recepción de Pedidos y Eventos (Webhooks)

Aquí hay un cambio importante respecto a Mercado Libre o TikTok. **Amazon SP-API no envía webhooks directamente a una URL HTTPS pública** del servidor. Amazon envía las notificaciones a través de **Amazon SQS (Simple Queue Service)** o **Amazon EventBridge**.

- **Cómo solucionarlo:**
  1. Configurar en AWS una cola SQS para recibir las notificaciones de nuevos pedidos de Amazon.
  2. Crear un comando en Laravel o un webhook puente que lea periódicamente esa cola SQS de Amazon.

3. Al extraer el payload del pedido de SQS, llamar a `WebhookFactory` pasándole el JSON recibido.
4. Crear `AmazonWebhookService.php` dentro de `app>Services>Webhooks` para procesar ese payload de manera asíncrona, el cual terminará despachando el proceso hacia el Job existente: `ProcessWebhookJob.php` para registrar los pedidos en las tablas `orders` y `order_details`.

4. Base de Datos y Modelos

- **Migraciones**: Deberás revisar tus tablas para soportar los identificadores únicos de Amazon (por ejemplo, el campo `amazon_order_id` en las órdenes o asegurar que las columnas de tokens soporten la estructura de Amazon). En tu tabla `plataforms`, registrarás un nuevo registro con el slug `'amazon'`.

Anexo 3. Guía de integración con SHEIN Open Platform

Guía de integración con SHEIN Open Platform

En el presente documento, se explica como realizar la integración del marketplace Shein.
Documentación oficial:

SHEIN Developer Platform
Provide technology-connected business solutions, such as product, order, return, procurement etc. Provide technical service support.
https://open.sheincorp.com/documents/system/dad0d1c7-be76-4b03-a735-4e23f012bdd9

SHEIN Developer Platform
Provide technology-connected business solutions, such as product, order, return, procurement etc. Provide technical service support.
https://open.sheincorp.com/documents/apidoc/detail/3001520

SHEIN Developer Platform
Provide technology-connected business solutions, such as product, order, return, procurement etc. Provide technical service support.
https://open.sheincorp.com/documents/msgdoc/detail/3000910

Obtener las credenciales

Shein cuenta con la **SHEIN Open Platform** para gestionar integraciones mediante API. Para crear la aplicación desde cero y conectar las tiendas de los clientes (vendedores), se siguen estos pasos:

- **Registro en el Portal de Desarrolladores:**
  - Entra a https://open.sheincorp.com/
  - Regístrarse utilizando un correo electrónico y completa el perfil técnico (como Partner/Developer de Software). Se deberá enviar la documentación empresarial que valide que se es un proveedor de software (SaaS).
- **Creación de la Aplicación (Developer App):**
  - Una vez aprobado el perfil, ir a la sección de **Console / App Management** y hacer clic en **Create New App**.
  - Configura el nombre de la aplicación y, lo más importante, se define la **Redirect URL (OAuth Callback)**. Esta apuntará al servidor de Laravel (se sugiere utilizar la url ya construida).
- **Obtención de las Claves del Desarrollador:**
  - Al guardar la aplicación, Shein asignará un **App ID (App Key)** y un **App Secret**.

Guía de integración con SHEIN Open Platform 1

- **Flujo de Autorización del Vendedor (OAuth2):**
  - El sistema deberá generar un enlace de autorización que incluirá el `appId` y la `redirectUrl` codificada en BASE64.
  - El cliente (vendedor) hará clic en este enlace desde el sistema, iniciará sesión en su **SHEIN Seller Backend**, aceptará los permisos y Shein lo devolverá al sistema con un token temporal (`tempToken`).
  - El backend consume el endpoint `/open-api/auth/get-by-token` usando las credenciales de la Aplicación para intercambiar ese token temporal por un **OpenKeyId** (identificador de la tienda) y un **SecretKey** (clave para firmar peticiones, que se deberá guardar de forma encriptada en la base de datos).

Conexión con la API

SHEIN no usa tokens portadores estáticos tradicionales (como `Bearer Token`) en cada endpoint. En su lugar, exige que cada petición HTTP contenga headers específicos de autenticación y una firma criptográfica dinámica de un solo uso para mitigar ataques de repetición.

1. Headers Obligatorios en cada Request:

- `x-lt-openKeyId`: El identificador de autorización obtenido del vendedor.
- `x-lt-timestamp`: Timestamp actual en milisegundos (ej. `1740709414000`).
- `x-lt-signature`: La firma calculada.

2. Algoritmo de Generación de la Firma
Para construir el validador o el generador de firmas dentro de los servicios (un nuevo `SheinSignatureValidator`), se deben seguir estos 5 pasos:

- **Paso A (Datos):** Concatenar el `OpenKeyId`, el `Timestamp` y el `Path` del endpoint separados por un ampersand `&`.
  $$\text{VALUE} = \text{OpenKeyId} + \text{"\&"} + \text{Timestamp} + \text{"\&"} + \text{Path}$$

Ejemplo: `B96C1541...&1740709414000&/open-api/order/purchase-order-info`

- **Paso B (Llave):** Generar un string aleatorio de 5 caracteres (`RandomKey`, ej: `test1`) y concatenarlo al final del `SecretKey`.
  $$\text{KEY} = \text{SecretKey} + \text{RandomKey}$$
- **Paso C (Cifrado):** Aplicar un cifrado **HMAC-SHA256** utilizando la llave (`KEY`) sobre los datos (`VALUE`), y convertir el resultado binario resultante en una cadena en formato hexadecimal (**Hex String**).
- **Paso D (Codificación):** Aplicar codificación **Base64** al string hexadecimal obtenido en el paso anterior.
- **Paso E (Firma Final):** Concatenar el `RandomKey` (los 5 caracteres aleatorios iniciales) justo antes de la cadena en Base64. Ese string final es el valor del header `x-lt-signature`.

Estructura JSON

1. Estructura de Productos (Comprehensive Product Query)

Cuando se consultan los listados de productos en SHEIN a nivel de SPU (Shop Unit) o SKU utilizando endpoints como `/open-api/product/` o `/open-api/goods/`, el JSON devuelto sigue un formato estandarizado que se mapeará en `SavePlatformProductsService.php`:


```
{
  "code": "0",
  "msg": "OK",
  "traceId": "shein-trace-8372491bca71",
  "info": [
    {
      "spuCode": "SPU_SHEIN_12345",
      "sellerSku": "MISHU-CAMISA-NEGRA-L",
      "productTitle": "Camisa Casual Manga Larga Elegante",
      "mainImage": "https://img.sheincorp.com/images/prod_123.jpg",
      "status": "LISTED",
      "categoryInfo": {
        "categoryId": 9012,
        "categoryName": "Blusas y Camisas"
      },
      "skus": [
        {
          "skuCode": "SKU_SHEIN_98765",
          "price": 19.99,
          "currency": "USD",
          "stock": 150,
          "attributes": [
            { "name": "Color", "value": "Negro" },
            { "name": "Size", "value": "L" }
          ]
        }
      ]
    }
  ]
}
```


2. Estructura de Webhooks (Eventos de Pedidos)

Cuando un cliente realice una compra, los servidores de SHEIN enviarán una notificación HTTP POST asíncrona al endpoint registrado en `WebhookController.php`. El payload típico para la creación de una orden o despacho logístico es estructurado bajo las especificaciones de entrega de paquetes individuales o múltiples (*JIT / Parent-Child Orders*):

```
{
  "event": "ORDER_CREATE",
  "timestamp": 1753760372427,
  "openKeyId": "B96C15416C9240DF96BAA0BC9B367C6D",
  "data": {
    "orderNo": "SH20260706-9912",
    "orderStatus": "AWAITING_DELIVERY",
    "packageInfoList": [
      {
        "packageNo": "PKG-8827110",
        "goodsIds": [1002391, 1002392],
        "quantity": 2,
        "warehouseCode": "WH-CN-01"
      }
    ],
    "shippingAddress": {
      "city": "Zapopan",
      "state": "Jalisco",
      "countryCode": "MX",
      "zipCode": "45010"
    }
  }
}
```

El controlador del webhook tomará este payload completo, validará su procedencia mediante la firma HTTP, guardará el evento en bruto utilizando el modelo `WebhookEvent.php`, y encolará de inmediato un `ProcessWebhookJob.php` para insertar la información en las tablas relacionales `orders` y `order_details` sin saturar la memoria del servidor.

Modificación de las Factorías (Factories existentes)

Para que el sistema reconozca a Shein de manera dinámica, se debe registrar el caso en los archivos de control:

- `AuthFactory.php`: Añadir el caso `'shein'` para que devuelva una instancia de `SheinAuthService`.
- `WebhookFactory.php`: Añadir el caso `'shein'` para direccionar el procesamiento al `SheinWebhookService`.
- En `ProductSyncInterface.php`: Crear la clase `SheinProductSyncService` heredando esta interfaz para homologar la descarga de inventario.
- **Base de Datos (`plataforms`)**: Insertar mediante un Seeder o migración un nuevo registro en la tabla `plataforms` con el slug `'shein'` y sus respectivas credenciales

Anexo 4. Reporte de pruebas unitarias

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Usuario	**CREATE**	Se instanció un nuevo usuario asignando nombre, correo electrónico, contraseña encriptada con bcrypt y tipo de usuario	`> $user = new App\Models\User();`
			`> $user->name = 'Usuario de Prueba';`
			`> $user->email = 'test_tinker@saas.com';`
			`> $user->password = bcrypt('password123');`
			`> $user->user_type_id = 4;`
			`> $user->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el usuario mediante el campo correo electrónico	`> $userConsultado = App\Models\User::where('email', 'test_tinker@saas.com')->first();`	Registro recuperado correctamente con los atributos esperados

	UPDATE	Se modificó el atributo nombre del usuario consultado	`> $userConsultado->name = 'Usuario de Prueba';`
			`> $userConsultado->save();`
			`=true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro del usuario mediante soft delet	`> $userConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(:	name varchar(191)	email varchar(191)	email_v time	password varchar(191)	re	created_at timestamp	updated_at timestamp	deleted_at timestamp
4	Usuario Tinker Modifi...	test_tinker@saas.com	(Null)	$2...		2026-06-29 11:20:20	2026-06-29 11:32:52	2026-06-29 11:32:52

Imagen 1. Pruebas de la sección de Usuario mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Tipos de usuarios	**CREATE**	Se creó un nuevo tipo de usuario con nombre y descripción	`$ut = new App\Models\UserType();`
			`$ut->name = 'test';`
			`$ut->description = 'test';`
			`$ut->save();`
				Registro creado exitosamente

	READ	Se consultó el tipo de usuario por nombre	`$utConsultado = App\Models\UserType::where('name', 'test')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre del tipo de usuario	`$utConsultado->name = 'Rol Actualizado';`
			`$utConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$utConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(…	name varchar(191)	description varchar(191)	created_at timestamp	updated_at timestamp	deleted_at timestamp
4	Rol Actualizado	test	2026-06-29 11:15:00	2026-06-29 11:38:31	2026-06-29 11:38:31

Imagen 2. Pruebas de la sección de Tipos de usuarios mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Módulos	**CREATE**	Se creó un nuevo módulo asignando un nombre	`$mod = new App\Models\Module();`
			`$mod->name = 'Módulo de Prueba';`
			`$mod->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el módulo por nombre	`$modConsultado = App\Models\Module::where('name', 'Módulo de Prueba')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre del módulo	`$modConsultado->name = 'Módulo Cambiado';`
			`$modConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$modConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(…	name varchar(191)	created_at timestamp	updated_at timestamp	deleted_at timestamp
12	Módulo Cambiado	2026-06-29 11:46:23	2026-06-29 11:47:16	2026-06-29 11:47:16

Imagen 3. Pruebas de la sección de Módulos mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Recursos	**CREATE**	Se creó un nuevo recurso asociado a un módulo mediante module_id	`$res = new App\Models\Resource();`
			`$res->name = 'Acción Crear';`
			`$res->module_id = 1;`
			`$res->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el recurso por nombre	`$resConsultado = App\Models\Resource::where('name', 'Acción Crear')->first();`	Registro recuperado correctamente
	UPDATE	Se modificó el atributo nombre del recurso	`$resConsultado->name = 'Acción Forzar';` `$resConsultado->save();`	Actualización guardada exitosamente
	DELETE	Se eliminó el registro mediante soft delete	`$resConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(2	module_id # bigint(20)	name ABC varchar(191)	created_at timestamp	updated_at timestamp	deleted_at timestamp
	1	Acción Forzar	2026-06-29 11:48:25	2026-06-29 11:51:38	2026-06-29 11:51:38

Imagen 4. Pruebas de la sección de Recursos mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Privilegios	**CREATE**	Se creó un nuevo privilegio asociado a un recurso mediante resource_id	`$priv = new App\Models\Privilege();`
			`$priv->name = 'Permiso Inicial';`
			`$priv->resource_id = 1;`
			`$priv->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el privilegio por nombre	`$privConsultado = App\Models\Privilege::where('name', 'Permiso Inicial')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre del privilegio	`$privConsultado->name = 'Permiso Modificado';`
			`$privConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$privConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(2	resource_id # bigint(20)	name varchar(191)	created_at timestamp	updated_at timestamp	deleted_at timestamp
2	1	Permiso Modificado	2026-06-29 11:53:31	2026-06-29 11:54:19	2026-06-29 11:54:19

Imagen 5. Pruebas de la sección de Privilegios mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Tipos de estatus	**CREATE**	Se creó un nuevo tipo de estatus con referencia a usuario creador y actualizador	`$st = new App\Models\StatusType();`
			`$st->name = 'Flujo Interno';`
			`$st->created_id = 2;`
			`$st->updated_id = 2;`
			`$st->save();`
				Registro creado exitosamente

	READ	Se consultó el tipo de estatus por nombre	`$stConsultado = App\Models\StatusType::where('name', 'Flujo Interno')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre del tipo de estatus	`$stConsultado->name = 'Flujo Externo';`
			`$stConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$stConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(2	name ABC varchar(191)	created_id # bigint(20)	updated_id # bigint(20)	created_at timestamp	updated_at timestamp	deleted_at timestamp
3	Flujo Externo	2	2	2026-06-29 11:57:30	2026-06-29 11:58:34	2026-06-29 11:58:34

Imagen 6. Pruebas de la sección de Tipos de estatus mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Estatus	**CREATE**	Se creó un nuevo estatus asociado a un status_type_id, con referencia a usuario creador y actualizador	`$est = new App\Models\Status();`
			`$est->name = 'Fase Beta';`
			`$est->status_type_id = 1;`
			`$est->created_id = 2;`
			`$est->updated_id = 2;`
			`$est->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el estatus por nombre	`$estConsultado = App\Models\Status::where('name', 'Fase Beta')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre del estatus	`$estConsultado->name = 'Fase Alfa';`
			`$estConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$estConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(:	name varchar(191)	order # int(11)	status_type_id # bigint(20)	created_id # bigint(20)	updated_id # bigint(20)	deleted_at timestamp	created_at timestamp	updated_at timestamp
15	Fase Alfa	(Null)	1	2	2	2026-06-29 12:00:55	2026-06-29 12:00:10	2026-06-29 12:00:55

Imagen 7. Pruebas de la sección de Estatus mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Paquetes	**CREATE**	Se creó un nuevo paquete con nombre, código, código SAT y referencia de usuario creador y actualizador	`$pack = new App\Models\Packaging();`
			`$pack->name = 'Caja de Madera Extra Grande';pack`
			`$pack->code = 2222;`
			`$pack->code_sat = H87;`
			`$pack->created_id = 2;`
			`$pack->updated_id = 2;`
			`$pack->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el paquete por nombre	`$packConsultado = App\Models\Packaging::where('name', 'Caja de Madera Extra Grande')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre del paquete	`$packConsultado->name = 'Caja de Madera Premium';`
			`$packConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$packConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(2	name varchar(191)	code # int(	code_sat varcha	created_id # bigint(20	updated_id # bigint(20)	created_at timestamp	updated_at timestamp	deleted_at timestamp
2	Caja de Madera Premium	[illegible]	[illegible]	2	2	2026-06-29 12:04:17	2026-06-29 12:05:49	2026-06-29 12:05:49

Imagen 8. Pruebas de la sección de Paquetes mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Productos	**CREATE**	Se creó un nuevo producto con nombre, código, código SAT y packaging_id asociado	`$prod = new App\Models\Product();`
			`$prod->name = 'Gis Volátil';`
			`$prod->code = 2222;`
			`$prod->code_sat = 101010101010;`
			`$prod->packaging_id = 1;`
			`$prod->created_id = 2;`
			`$prod->updated_id = 2;`
			`$prod->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el producto por nombre	`$prodConsultado = App\Models\Product::where('name', 'Gis Volátil')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre del producto	`$prodConsultado->name = 'Gis Volátil Pro';`
			`$prodConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$prodConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(:	name ABC varchar	code ABC var	code_sat ABC varchar(50)	packaging_id # bigint(20)	created_id # bigint(20)	updated_id # bigint(20)	created_at timestamp	updated_at timestamp	deleted_at timestamp
7	Gis Volá...			1	2	2	2026-06-29 12:08:23	2026-06-29 12:09:55	2026-06-29 12:09:55

Imagen 9. Pruebas de la sección de Productos mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Pedidos	**CREATE**	Se creó un nuevo pedido con folio, folio de plataforma, estatus, plataforma, cliente y total	`$or = new App\Models\Order();`
			`$or->folio = 'test123';`
			`$or->folio_platform = 'test';`
			`$or->status_id = 1;`
			`$or->plataform_id = 1;`
			`$or->client_id = 1;`
			`$or->total = 100;`
			`$or->created_id = 2;`
			`$or->updated_id = 2;`
			`$or->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el pedido por folio	`$orConsultado = App\Models\Order::where('folio', 'test123')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo status_id del pedido	`$orConsultado->status_id = 2;`
			`$orConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$orConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(;	folio varchar	folio_platform varchar(19	plataform_id # bigint(20)	status_id # bigint(	client_id # bigint	total # deci	lot_user_id # bigint(20	created_id # bigint(20	updated_id # bigint(20)	deleted_at timestamp	created_at timestamp	updated_at timestamp
8	test123	test	1	2	1	100.00	(Null)	2	2	2026-06-29 13:56:54	2026-06-29 13:33:42	2026-06-29 13:56:54

Imagen 10. Pruebas de la sección de Pedidos mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Detalle de Pedido	**CREATE**	Se creó un detalle de pedido asociando order_id, product_id, precio y cantidad	`$orDe = new App\Models\OrderDetail();`
			`$orDe->order_id = 8;`
			`$orDe->product_id = 1;`
			`$orDe->price = 100;`
			`$orDe->quantity = 2;`
			`$orDe->total = 200;`
			`$orDe->created_id = 2;`
			`$orDe->updated_id = 2;`
			`$orDe->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el detalle de pedido por lotted_quantity	`$orDeConsultado = App\Models\OrderDetail::where('lotted_quantity', 2)->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo lotted_quantity del detalle de pedido	`$orDeConsultado->lotted_quantity = 2;`
			`$orDeConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delet	`$orDeConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigi	order_id # bigint(	product_id # bigint(2(	price # decir	quantity # int(11	total # decima	lotted_quantity # int(11)	created_id # bigint(2(	updated_id # bigint(20	deleted_at timestamp	created_at timestamp	updated_at timestamp
7	8	1	100.00	2	200.00	2	2	3	2026-06-29 14:0...	2026-06-29 13:40:56	2026-06-29 13:40:56

Imagen 11. Pruebas de la sección de Detalle del pedido mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Bitácora de Estatus de Pedido	**CREATE**	Se registró un nuevo evento de cambio de estatus asociado a order_id	`$orSt = new App\Models\OrderStatusLog();`
			`$orSt->order_id = 8;`
			`$orSt->status_id = 2;`
			`$orSt->created_id = 2;`
			`$orSt->updated_id = 2;`
			`$orSt->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el registro por order_id	`$orStConsultado = App\Models\OrderStatusLog::where('order_id', 8)->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo status_id del registro	`$orStConsultado->status_id = 2;`
			`$orStConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$orStConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(:	order_id # bigint(2	status_id # bigint(2	created_id # bigint(2(	updated_id # bigint(20	deleted_at timestamp	created_at timestamp	updated_at timestamp
10	8	3	2	2	2026-06-29 14:02:36	2026-06-29 14:01:26	2026-06-29 14:01:29
9	8	2	2	2	2026-06-29 14:02:36	2026-06-29 13:47:20	2026-06-29 13:47:20

Imagen 12. Pruebas de la sección de Bitácora de estatus del pedido mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Lotes de Detalle de Pedido	**CREATE**	Se creó un lote asociado a order_detail_id con su cantidad correspondiente	`$lot = new App\Models\OrderDetailLot();`
			`$lot->order_detail_id = 7;`
			`$lot->quantity = 2;`
			`$lot->created_id = 2;`
			`$lot->updated_id = 2;`
			`$lot->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó el lote por order_detail_id	`$lotConsultado = App\Models\OrderDetailLot::where('order_detail_id', 7)->first();`	Registro recuperado correctamente

Resultado en la base de datos

id # bigint(2	order_detail_id # bigint(20)	quantity # int(11)	created_id # bigint(20)	updated_id # bigint(20)	deleted_at timestamp	created_at timestamp	updated_at timestamp
2	7	2	2	2	2026-06-29 14:03:35	2026-06-29 13:54:35	2026-06-29 13:54:35

Imagen 13. Pruebas de la sección de Lotes de detalle del pedido mediante Artisan Tinker

Módulo	Operación	Acción realizada	Resultado
Impresión de Pedido	**GENERACIÓN DE PDF**	Se generó el documento PDF del pedido a partir de un folio de TikTok, incluyendo plataforma, cliente, fecha, total y detalle de productos	Documento PDF generado y descargado correctamente
Resultado del sistema			

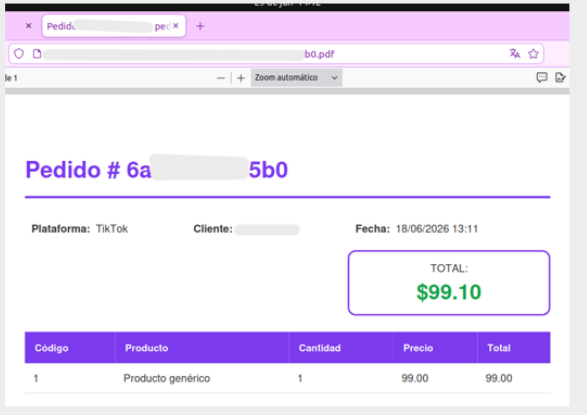

Imagen 14. Pruebas de la sección de Impresión del pedido del sistema

Módulo		Operación	Acción realizada	Resultado
Integración Mercado Libre	***Order***	**SINCRONIZACIÓN**	Se sincronizó un pedido proveniente de Mercado Libre con su folio de plataforma	Registro creado correctamente con plataform_id correspondiente

Resultado en la base de datos

id # bigint(:	folio varchar	folio_pl varc	plataform_id # bigint(20)	status_id # bigint(	client_id # bigint(20)	total # deci	lot_user_id # bigint(20	created_id # bigint(20	updated_id # bigint(20)	deleted_at timestamp	created_at timestamp	updated_at timestamp
10	6. .	200...	2	1	(Null)	150.00	(Null)	1	1	(Null)	2026-06-29 15:00:02	2026-06-29 15:00:03

Imagen 15. Pruebas de la Integración de Mercado Libre – Sincronización del pedido

Módulo		Operación	Acción realizada	Resultado
Integración Mercado Libre	***Order Detail***	**SINCRONIZACIÓN**	Se sincronizó el detalle del pedido de Mercado Libre	Registro creado correctamente

Resultado en la base de datos

id # bigi	order_id # bigint(	product_id # bigint(2(	price # decir	quantity # int(11	total # decima	lotted_quantity # int(11)	created_id # bigint(2(	updated_id # bigint(20	deleted_at timestamp	created_at timestamp	updated_at timestamp
8	10	4	150.00	1	150.00	0	1	1	(Null)	2026-06-29 15:00:02	2026-06-29 15:00:03

Imagen 16. Pruebas de la Integración de Mercado Libre – Sincronización del detalle del pedido

Módulo		Operación	Acción realizada	Resultado
Integración Mercado Libre	***Status Log***	**SINCRONIZACIÓN**	Se registró el evento de estatus del pedido de Mercado Libre	Registro creado correctamente

Resultado en la base de datos

id # bigint(2	order_id # bigint(2	status_id # bigint(2	created_id # bigint(20	updated_id # bigint(20	deleted_at timestamp	created_at timestamp	updated_at timestamp
15	10	1	1	1	(Null)	2026-06-29 15:00:03	2026-06-29 15:00:03

Imagen 17. Pruebas de la Integración de Mercado Libre – Sincronización del estado del pedido

Módulo		Operación	Acción realizada	Resultado
Integración Mercado Libre	***Webhook***	**RECEPCIÓN**	Se recibió y almacenó el webhook con evento, tópico y payload en formato JSON	Webhook procesado y registrado correctamente

Resultado en la base de datos

id # bigint(2	platform ABC varchar(191)	event_id ABC varchar(191)	topic ABC varchar(19	payload ABC json	processed_at timestamp	dele t	created_at timestamp	updated_at timestamp
36	Mercado Libre	06...	orders_v2	{ _ .	2026-06-29 15:00:02	(...	2026-06-29 15:00:02	2026-06-29 15:00:02

Imagen 18. Pruebas de la Recepción del webhook de Mercado Libre

Módulo		Operación	Acción realizada	Resultado
Integración Mercado Libre	***Productos sincronizados***	**SINCRONIZACIÓN**	Se sincronizaron productos de Mercado Libre vinculando external_product_id y external_sku con el catálogo local	Registros creados correctamente (4 productos sincronizados)

Resultado en la base de datos

id # bigint(:	plataform_id # bigint(20)	product_id # bigint(20)	external_product_id ABC varchar(191)	external_sku ABC varchar(191)	price # decimal(1	created_id # bigint(20)	updated_id # bigint(20)	created_at timestamp	updated_at timestamp
8	2	5	M	ML	150.00	2	2	2026-06-23 13:51:42	2026-06-29 14:44:45

id # bigint(:	name ABC varchar(191)	code ABC varchar	code_sat ABC varchar(50)	packaging_id # bigint(20)	created_id # bigint(20)	updated_id # bigint(20)	created_at timestamp	updated_at timestamp	deleted_at timestamp
	Cagador Zmx-cr-xm-67w De 67			1	2	2	2026-06-23 16:24:10	2026-06-23 16:24:10	(Null)
	Termo Termo Bucees Colo			1	2	2	2026-06-23 16:23:30	2026-06-23 16:23:30	(Null)
	Botella Termo Agua Aislada Hermética			1	2	2	2026-06-23 16:22:56	2026-06-23 16:22:56	(Null)
	Termo Aislamiento Térmico			1	2	2	2026-06-23 16:22:28	2026-06-23 16:22:28	(Null)

Imagen 19. Pruebas de la Integración de Mercado Libre – Sincronización de productos

Módulo		Operación	Acción realizada	Resultado
Integración TikTok Shop	***Order***	**SINCRONIZACIÓN**	Se sincronizó un pedido proveniente de TikTok Shop con su folio de plataforma	Registro creado correctamente con plataform_id correspondiente

Resultado en la base de datos

id # bigint(7)	folio varchar	folio_pl varc	plataform_id # bigint(20)	status_id # bigint(	client_id # bigint(20)	total # deci	lot_user_id # bigint(20	created_id # bigint(20	updated_id # bigint(20)	deleted_at timestamp	created_at timestamp	updated_at timestamp
7	6 ..	5. ...	4	4	3	99.10	(Null)	1	1	(Null)	2026-06-18 13:12:15	2026-06-18 13:12:15

Imagen 20. Pruebas de la Integración de TikTok Shop – Sincronización del pedido

Módulo		Operación	Acción realizada	Resultado
Integración TikTok Shop	***Order Detail***	**SINCRONIZACIÓN**	Se sincronizó el detalle del pedido de TikTok Shop	Registro creado correctamente

Resultado en la base de datos

id # bigi	order_ # bigint(	product_id # bigint(2(	price # decir	quantity # int(11	total # decima	lotted_quantity # int(11)	created_id # bigint(2(	updated_id # bigint(20	deleted_at timestamp	created_at timestamp	updated_at timestamp
6	7	1	99.00	1	99.00	0	1	1	(Null)	2026-06-18 13:12:17	2026-06-18 13:12:17

Imagen 21. Pruebas de la Integración de TikTok Shop – Sincronización del detalle de pedido

Módulo		Operación	Acción realizada	Resultado
Integración TikTok Shop	***Status Log***	**SINCRONIZACIÓN**	Se registró el evento de estatus del pedido de TikTok Shop	Registro creado correctamente

Resultado en la base de datos

id # bigint(…	order_id # bigint(2	status_id # bigint(2…	created_id # bigint(2…	updated_id # bigint(20	deleted_at timestamp	created_at timestamp	updated_at timestamp
6	7	4	1	1	(Null)	2026-06-18 13:12:18	2026-06-18 13:12:18

Imagen 22. Pruebas de la Integración de TikTok Shop – Sincronización del estado del pedido

Módulo		Operación	Acción realizada	Resultado
Integración TikTok Shop	***Webhook***	**RECEPCIÓN**	Se recibió y almacenó el webhook de notificación con evento, tópico y payload en formato JSON	Webhook procesado y registrado correctamente

Resultado en la base de datos

id # bigint(2	platform varchar(191)	event_id varchar(191)	topic varchar	payload json	processed_at timestamp	deleted_at timestam	created_at timestamp	updated_at timestamp
7	TikTok	7(1		"{ ...	(Null)	(Null)	2026-06-26 01:46:11	2026-06-26 01:46:11

Imagen 23. Pruebas de la Recepción del webhook de TikTok Shop

Módulo		Operación	Acción realizada	Resultado
Integración TikTok Shop	***Productos sincronizados***	**SINCRONIZACIÓN**	Se sincronizó un producto de TikTok Shop vinculando external_product_id y external_sku con el catálogo local	Registro creado correctamente

Resultado en la base de datos

id # bigint(;	plataform_id # bigint(20)	product_id # bigint(20)	external_product_id varchar(191)	external_sku varchar(191)	price # decimal(1	created_id # bigint(20)	updated_id # bigint(20)	created_at timestamp	updated_at timestamp
6	4	2	1 5	2	99.00	2	2	2026-06-23 11:34:05	2026-06-24 10:17:41

id # bigint(;	name varchar(191)	code varchar	code_sat varchar(50)	packaging_id # bigint(20)	created_id # bigint(20)	updated_id # bigint(20)	created_at timestamp	updated_at timestamp	deleted_at timestamp
2	Kit Demo Sistema de Inventario			1	2	2	2026-06-29 12:56:07	2026-06-29 12:56:10	(Null)

Imagen 24. Pruebas de la Integración de TikTok Shop – Sincronización del producto

Módulo		Operación	Acción realizada	Resultado
Integración	***Configuración Mercado Libre***	**VERIFICACIÓN**	Se verificaron los parámetros de configuración de la integración (URLs de autenticación, oauth y refresh)	Configuración almacenada y vigente, con expiración registrada

Resultado en la base de datos

slug varchar(191)	icon_path varchar(191)	oauth_url varchar(191)	auth_url varchar(191)	refresh_url varchar(1
mercadolibre	plataforms/NLx...	https:...	https:/...	(Null)

redirect_uri varchar(191)	client_id varchar(191)	service_id varchar(19	client_secret text	access_token text	refresh_token text	expires_at timestamp
https:/...	7 ...9	(Null)	e...	e...	...	2026-06-29 17:00:02

Imagen 25. Pruebas de la Configuración de Mercado libre

Módulo		Operación	Acción realizada	Resultado
Integración	***Configuración TikTok Shop***	**VERIFICACIÓN**	Se verificaron los parámetros de configuración de la integración	Configuración almacenada y vigente, con expiración registrada

Resultado en la base de datos

slug	icon_path	oauth_url	auth_url	refresh_url	client_id
tiktok	plataforms/7sB...	https:/,	https:/,	https:/,	€...lm

redirect_uri	client_id	service_id	client_secret	access_token	refresh_token	expires_at
https:/...	(...lm	7€...4	e...)...	e...i...	(...ZT...	2026-07-06 18:00:02

Imagen 26. Pruebas de la Configuración de TikTok Shop

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Plataformas	**CREATE**	Se creó una nueva plataforma con nombre y status_id, generándose de forma automática los atributos de configuración	`$plat = new App\Models\Plataform();`
			`$plat->name = 'Test Platforms';`
			`$plat->status_id = 11;`
			`$plat->created_id = 2;`
			`$plat->updated_id = 2;`
			`$plat->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó la plataforma por nombre	`$platConsultado = App\Models\Plataform::where('name', 'Test Platforms')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo nombre de la plataforma	`$platConsultado->name = 'Update Platforms ';`
			`$platConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$platConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # big	name varchar(191)	status_id # bigint(20)	created_id # bigint(20)	updated_id # bigint(20)	created_at timestamp	updated_at timestamp	deleted_at timestamp
7	Update Platforms	11	2	2	2026-06-29 12:35:39	2026-06-29 12:36:23	2026-06-29 12:36:23

Imagen 27. Pruebas de la sección de Plataformas mediante Artisan Tinker

Módulo	Operación	Acción realizada	Ejemplo	Resultado

Usuarios Asignados	**CREATE**	Se asignó un usuario a una plataforma mediante plataform_id y user_id	`$ua = new App\Models\AssignedUser();`
			`$ua->plataform_id = 1;`
			`$ua->user_id = 1;`
			`$ua->created_by = 2;`
			`$ua->updated_by = 2;`
			`$ua->save();`
			`= true`
				Registro creado exitosamente

	READ	Se consultó la asignación por user_id	`$uaConsultado = App\Models\AssignedUser::where('user_id', '1')->first();`	Registro recuperado correctamente

	UPDATE	Se modificó el atributo plataform_id de la asignación	`$uaConsultado->plataform_id = 3;`
			`$uaConsultado->save();`
			`= true`
				Actualización guardada exitosamente

	DELETE	Se eliminó el registro mediante soft delete	`$uaConsultado->delete();` `= true`	Eliminación lógica ejecutada correctamente

Resultado en la base de datos

id # bigint(2	user_id # bigint(20	plataform_id # bigint(20)	created_by # bigint(20)	updated_by # bigint(20)	deleted_at timestamp	created_at timestamp	updated_at timestamp
5	1	3	2	2	2026-06-29 13:14:51	2026-06-29 13:12:59	2026-06-29 13:14:51

Imagen 28. Pruebas de la sección de Usuarios asignados mediante Artisan Tinker