Desarrollo de un API Rest para conectar la plataforma de servicios retail (Whatoko retail) con sistemas de proveedores dropshiping

Autor

Kevin Alonso Luna Bustos

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

Doctor en Ciencias de la Computación

Universidad Industrial de Santander

Facultad de Ingenierías Físicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Ingeniería de Sistemas

Bucaramanga

2023

Agradecimientos

Quiero expresar mi más sincero agradecimiento a las personas que me han apoyado y motivado durante mi proceso de aprendizaje en la universidad.

En primer lugar, a mi padre Alonso Luna por su apoyo incondicional y motivación constante, que me ha llevado a alcanzar mis metas académicas. También quiero agradecer a mi madre María Bustos por estar siempre a mi lado en los momentos buenos y malos, y ayudarme a superar los obstáculos que he encontrado en el camino.

No puedo dejar de mencionar a mis hermanos Erick y Johan, quienes han sido un soporte vital en mi vida y siempre me han hecho sentir que todo es posible, sin importar las adversidades.

Además, quiero agradecer a mis amigos de la universidad, Jesús Martínez, Camilo Bayona, Angie Fuentes y Cristian Rueda, por haber estado siempre a mi lado en los momentos inolvidables de la universidad y por haberme brindado su amistad sincera.

Quiero expresar mi agradecimiento a la empresa A&A Soluciones por brindarme la oportunidad de realizar mi práctica profesional, la cual me ha ayudado a crecer tanto como profesional como persona. También quiero agradecer a Juan Carlos Abaunza y a Francisco Javier González por acompañarme en mi formación y en completar mi etapa de aprendizaje.

Además, no puedo olvidar mencionar a la profesora Sonia Gamboa mi exdirectora de proyecto por aceptar dirigir esta iniciativa y por compartir su conocimiento para lograr su éxito y al profesor Gabriel Pedraza por haberme acompañado y apoyado en el proceso final.

Por último, deseo expresar mi gratitud a la Universidad Industrial de Santander por permitirme formarme profesionalmente, y por brindarme la oportunidad de alcanzar mis metas académicas.

Tabla de Contenido

Pág.

Lista de Figuras

Pág.

Glosario

API: Una API (Application Programming Interface) es un conjunto de reglas y protocolos que permiten a diferentes aplicaciones interactuar entre sí y compartir información de manera programática.

BPO: Business Process Outsourcing se refiere a la subcontratación de procesos de negocio no esenciales a terceros proveedores de servicios para reducir costos y mejorar la eficiencia. Los procesos de negocio comunes que se subcontratan en BPO incluyen la contabilidad, el procesamiento de datos, la gestión de recursos humanos y el soporte técnico.

Broker: es un intermediario que se encarga de conectar diferentes sistemas y aplicaciones, permitiendo que intercambien información y se comuniquen entre sí., se encarga de recibir la información de los publishers y distribuirla a los subscribers que se han suscrito a los canales correspondientes. Los publishers no necesitan conocer la identidad o el número de los subscribers, simplemente publican la información en el canal correspondiente y el Broker se encarga de enviarla a los suscriptores adecuados.

Controladores: son componentes de software que se encargan de procesar las solicitudes recibidas por el API y generar las respuestas correspondientes. es responsable de implementar una o varias operaciones que se pueden realizar a través del API, y está diseñado para recibir los parámetros de entrada que se necesitan para llevar a cabo estas operaciones.

Dropshipping: es un modelo de negocio en el que el minorista no mantiene inventario de los productos que vende, sino que los adquiere directamente del proveedor, quien se encarga de enviar los productos al cliente final en nombre del minorista.

Endpoints: Los endpoints son los puntos de acceso en un API a través de los cuales se pueden realizar diferentes operaciones y obtener información. Cada endpoint suele estar asociado a un recurso específico y puede ser accesible mediante una URL determinada.

ITO: Information Technology Outsourcing se refiere a la subcontratación de servicios de tecnología de la información a terceros proveedores de servicios. Los servicios de TI comunes que se subcontratan en ITO incluyen la gestión de redes, el desarrollo de software, la gestión de bases de datos y el soporte técnico.

KPO: Knowledge Process Outsourcing se refiere a la subcontratación de procesos de negocio basados en el conocimiento, como la investigación y el análisis, a proveedores de servicios especializados. El objetivo de KPO es aprovechar la experiencia y el conocimiento de los expertos en áreas específicas para mejorar la calidad de los procesos y la toma de decisiones.

ORM: Un ORM es una biblioteca o framework que mapea las tablas de una base de datos relacional a objetos en un lenguaje de programación orientado a objetos. El ORM crea una relación entre las tablas de la base de datos y las clases y objetos en la aplicación, lo que permite a los desarrolladores acceder a los datos utilizando métodos y atributos de objetos en lugar de escribir SQL directamente.

Publisher: es un componente que se encarga de enviar información a través de un canal determinado, sin saber quiénes son los destinatarios específicos. En lugar de eso, los subscribers pueden suscribirse a los canales específicos y recibir la información correspondiente.

Publish/Subscriber: El patrón Publisher-Subscriber (también conocido como Pub/Sub) es un patrón de diseño de software que se utiliza para crear sistemas distribuidos y escalables. En este patrón, los componentes se dividen en dos roles: los publishers y los subscribers.

Retail: se refiere al negocio de la venta de productos y servicios directamente al consumidor final. se utiliza para describir la parte del negocio que se enfoca en la venta minorista.

Repositorio: El patrón repositorio consiste en separar la lógica que recupera los datos y los asigna a un modelo de entidad de la lógica de negocios que actúa sobre el modelo, esto permite que la lógica de negocios sea independiente del tipo de dato que comprende la capa de origen de datos.

REST: (Representational State Transfer) es un estilo arquitectónico utilizado para diseñar servicios web que se basan en HTTP y siguen ciertos principios, como la separación de recursos y operaciones, la utilización de verbos HTTP para indicar las acciones que se realizan sobre los recursos, entre otros.

Rutas: Las rutas son las URLs que se utilizan para acceder a los diferentes endpoints en un API. Cada ruta está asociada a un endpoint específico y puede incluir parámetros que permiten personalizar la solicitud.

Subscriber: es un componente que se encarga de recibir y procesar la información enviada por los publishers. Los subscribers se suscriben a los canales correspondientes y reciben la información cuando esta es enviada por los publishers.

Token: es un tipo de credencial que se utiliza para verificar la identidad de un usuario o una aplicación en un sistema de software o una plataforma en línea. Este token se genera en el momento en que se realiza la autenticación exitosa y se utiliza para identificar y validar a los usuarios y aplicaciones que intentan acceder a recursos protegidos por el sistema.

UI: (User Interface) se refiere a la interfaz gráfica que permite a los usuarios interactuar con un producto o servicio. La UI incluye elementos como botones, menús, iconos y campos de entrada de datos, y se enfoca en crear una experiencia visualmente atractiva y fácil de usar para el usuario

UX: User Experience se refiere a la experiencia general que tiene un usuario al interactuar con un producto o servicio, como un sitio web, una aplicación móvil o un dispositivo electrónico. La UX se enfoca en crear productos que sean fáciles de usar, accesibles, atractivos y satisfagan las necesidades del usuario.

Resumen

Título: Desarrollo de un API REST para conectar la plataforma de servicios Whatoko retail con sistemas de proveedores dropshipping

Autor: Kevin Alonso Luna Bustos

Palabras Clave: API REST, e-commerce, dropshipping, microservicio, broker de mensajería

Descripción: El siguiente proyecto busca la integración de artículos de proveedores, con la plataforma de Whatoko retail definida por la empresa A&A Soluciones Tic, a través de los servicios web de los proveedores, integrándolos como un servicio más de la plataforma siguiendo un modelo de negocio dropshipping. Este documento tiene como fin exponer el proceso de investigación, análisis y desarrollo realizado durante la realización del proyecto.

El dropshipping hoy en día se usa como una herramienta que busca generar interés de los compradores a través de los productos ofrecidos en los ecommerce, con este fin los comerciantes venden productos de diferentes proveedores en sus plataformas delegando las funciones de logística, tales como el almacenamiento y envió de los productos, lo que resulta en una ventaja de reducción de costos. Los comerciantes realizan esta práctica como una forma de obtener comisiones y reconocimiento como plataforma e-commerce, para que esto se pueda lograr es necesario que haya una integración entre las dos partes, y Whatoko retail busca ofrecer está integración permitiendo visualizar y sincronizar los productos de los proveedores y hacer gestión directa para que se puedan integrar en diferentes e-commerce.

El proyecto se desarrolló con una arquitectura limpia y escalable basada en microservicios usando, Docker para el desarrollo y despliegue de las imágenes de manera local, SonarQube para hacer pruebas de calidad de código, y Postman para realizar las pruebas de funcionamiento del API. El desarrollo se hizo siguiendo los estándares del ciclo de vida de software con una metodología de desarrollo DevOps, con Typescript como lenguaje de programación y MySQL como gestor de base de datos, a continuación, en el documento se describirá la realización del proyecto siguiendo cada una de sus etapas.

---

* Trabajo de Grado

** Facultad de Ingeniería Físico Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira. Doctor en Ciencias de la Computación

Abstract

Title: Development of a REST API to connect the Whatoko retail service platform with dropshipping provider systems

Author: Kevin Alonso Luna Bustos

Keywords: REST API, e-commerce, dropshipping, microservice, messaging broker.

Description: The following project aims the integration of supplier items, with the Whatoko retail platform defined by the company A&A Soluciones Tic, through the suppliers' web services, integrating them as another platform service, following a dropshipping business model. The purpose of this document is to expose the process of research, analysis and development carried out during the realization of the project.

Dropshipping today is used as a tool that aims to generate interest from buyers through the products offered in ecommerce, for this purpose merchants sell products from different suppliers on their platforms, delegating logistics functions, such as storage. and shipping of products, resulting in a cost reduction advantage. Merchants carry out this practice to obtain commissions and recognize as an e-commerce platform, for this to be achieved it is necessary that there be an integration between the two parties, and Whatoko retail offers this, it seeks to offer this integration allowing to visualize and synchronize the suppliers' products and direct management so that they can be integrated into different electronic stores.

The project was developed with a clean and scalable architecture based on microservices using Docker for the development and use of images locally, SonarQube for code quality testing, and Postman for API performance testing. The development was done following the standards of the software life cycle with a DevOps development methodology, with Typescript as programming language and MySQL as database manager, then in the document the realization of the following project will be described in each one of its stages.

---

* Degree work

** Faculty of Physical Mechanical Engineering. School of Systems Engineering
Director: Gabriel Rodrigo Pedraza Ferreira. PhD in Computer Science

Introducción

La integración de tecnologías ha hecho que cada vez más empresas puedan crecer a partir de la tercerización de procesos a actores externos que se especializan en prestar un servicio que complementan estos procesos. Teniendo en cuenta esto, este proyecto se basa en la integración con proveedores de artículos para la venta en plataformas de comercio retail. Este proyecto entra a desarrollarse en un momento donde cada día esta industria se hace más grande, se dinamiza y las empresas buscan nuevas formas de llegar a los consumidores y perfilar mejor sus hábitos de compra, lo que ha llevado a que crezca la demanda de herramientas que sean capaces de expandir sus capacidades para mejorar este estilo de comercio, ideando varias y creativas formas de consolidar un mercado.

Este proyecto hace parte de un grupo de servicios en desarrollo por parte de la empresa desarrolladora A&Atic llamado Whatoko retail. El servicio que se estará exponiendo en este documento hace referencia a un API REST que busca integrar distintos servicios web de proveedores o mayoristas de productos. La idea principal de este proyecto se basa en la integración y comunicación directa de comerciante y proveedor para la venta y compra de productos, sincronizando los productos ofrecidos por estos proveedores en las plataformas retail de los clientes de Whatoko retail. Esto se logra implementando un modelo de negocio conocido como dropshipping el cual está planteado con la idea de poder vender productos de proveedores mayoristas, en donde estos proveedores se encargan de la gestión y envió de los productos a los clientes finales.

En ese sentido, este proyecto entra a desarrollarse en una década donde el comercio electrónico en América Latina está creciendo y Colombia no se queda atrás. Las personas cada vez

más prefieren realizar sus compras en línea con medios de pagos que resultan más fáciles, lo cual ha generado un amento en la confianza de los usuarios y el crecimiento del comercio retail electrónico en Colombia.

Este modelo se podría catalogar como un modelo de envió directo de pedidos, en el que no es necesario almacenar y mantener los productos ofrecidos en inventario, lo que puede resultar más llamativo para los comerciantes que buscan crecer y atraer más clientes en su comercio para aumentar las ventas.

En la actualidad son cada vez más las empresas que apuestan a la integración de nuevos servicios a través de APIs, empresas que consideran primordial el uso de estas tecnologías para sus procesos.

1. Planteamiento y justificación del problema

Debido al crecimiento que ha tenido el comercio electrónico en todo el mundo, beneficiado por los confinamientos de la pandemia, América Latina se ha posicionado por encima de otras regiones y, tal como lo afirma el informe "Cómo el área de pagos digitales y comercio electrónico está ganando terreno en América Latina", respaldado por datos de Americas Market Intelligence y el World Bank, presentado por Ebanx, su comercio electrónico minorista tuvo un crecimiento del 40% en el año 2021.

Este proyecto se desarrolla en una arquitectura basada en microservicios, la cual está pensada para que permita una integración simple y escalable para el desarrollo de software, además de permitir una fácil comunicación entre los diferentes servicios que conforman un proyecto, haciéndola así una arquitectura ideal para Whatoko retail, que está enfocado a la mejora de la experiencia y administración de los e-commerce y el retail a partir de la integración con distintos servicios.

Uno de estos servicios consiste en un sistema de ventas estilo dropshipping, utilizado hoy en día por pequeños y medianos comercios debido a su fácil implementación y a que permite la venta de artículos provenientes de empresas proveedoras sin la necesidad de preocuparse por almacenamiento y logística de los envíos. El sistema de dropshipping permite extraer productos de los proveedores y cargarlos en las plataformas de los e-commerce de los clientes de Whatoko retail, con la intención de delegar el almacenamiento y gestión de envió a los proveedores,

Para esto, se desarrolló un API REST en el que se implementó una interfaz para la integración de diversas empresas proveedoras a la plataforma de Whatoko retail a través de sus servicios web, en el que se sincronizan los modelos de productos y especificaciones del producto

con el modelo de la plataforma, esto permite integrar distintos proveedores para abastecer todas las necesidades de suministros de los clientes permitiéndoles cargar los datos de los productos en sus e-commerce con productos de futuros proveedores.

Este proyecto se da gracias a la oportunidad brindada por A&Atic soluciones, una empresa desarrolladora y exportadora de servicios KPO/BPO/ITO de personal remoto, entre los que se destacan los servicios de desarrollo de aplicaciones móviles, diseño de interfaces UX/UI, desarrollo Frontend y Backend. El desarrollo de la practica en A&Atic es una oportunidad de crecimiento como profesional que me permite adoptar y aprender nuevas tecnologías que se implementan en la empresa, además de poder mejorar mis habilidades de comunicación, responsabilidad, motivación, paciencia y trabajo en equipo, entre otras.

El desarrollo de la práctica se da bajo un trabajo colaborativo siguiendo el marco de trabajo Scrum a través de Zoho Sprints, la cual es una plataforma con herramientas de gestión de proyectos ágil que ayuda a los equipos a adoptar un enfoque iterativo y colaborativo para el trabajo. Para el control de versiones del proyecto se usó la plataforma de GitLab y el software de control de versiones Git siguiendo las fases de desarrollo y ciclo de vida del desarrollo de software, implementando la metodología de desarrollo DevOps.

El proyecto se desarrolló con la ayuda del líder técnico del proyecto que brindó asesoramiento y apoyo en la definición en cada fase del desarrollo, en un trabajo conjunto para el reforzamiento de los conocimientos. Un factor importante para el desarrollo de la practicas fue la capacidad de poder identificar y dar una solución integral al planteamiento del problema propuesto que implica una interacción con las distintas fases del desarrollo del software y uso de nuevas tecnologías.

2. Objetivos

2.1 Objetivo General

Desarrollar el servicio de dropshipping de Whatoko Retail que permita hacer conexión con diferentes proveedores con el fin de extraer información de artículos para la sincronización en los catálogos de los e-commerce de los clientes de la plataforma.

2.2 Objetivos Específicos

- Desarrollar un API REST en Node.js con TypeScript y Express que permita centralizar el proceso de integración de la información de los productos de diferentes proveedores conectados.
- Implementar un sistema transaccional con MySQL y Sequelize como ORM.
- Desarrollar un módulo de validación y autenticación de usuarios para el API, reforzando la seguridad y protección de los datos.
- Hacer pruebas de funcionamiento y error para validar los protocolos del API e implementar sus respectivas modificaciones.
- Generar la documentación requerida para uso del API, con sus respectivos canales de acceso y descripción de los protocolos.

3. Marco de referencia

3.1 Marco teórico

3.1.1 El comercio electrónico en la actualidad

El e-commerce, también conocido como e-business, se basa en la transacción de bienes y servicios por medio de comunicaciones electrónicas. Aunque las personas se han familiarizado con el comercio electrónico en las últimas décadas, en realidad ha sido todo un proceso de consolidación de varias tecnologías desde hace ya varios años. El origen y evolución del comercio electrónico ha tenido avances que fijaron el camino a lo que hoy existe, se podría decir que el comercio electrónico empezó a crecer a medida que se implementaban nuevos protocolos de seguridad en las transacciones en línea, además de la creación de las redes sociales y el uso masivo de los celulares cambiaron completamente el panorama permitiendo que aún mas empresas apostarán al crecimiento implementado nuevos sistemas que facilitaran el proceso de compra en las plataformas. En los últimos años hubo un acelerador en el comercio electrónico causado por el confinamiento, lo que permitió que muchas personas empezaran a realizar sus compras en línea.

3.1.2 División del comercio electrónico

El comercio electrónico se puede generalizar en dos modelos de negocios básicos, de empresa a empresa (B2B) y de empresa a consumidor (B2C). En B2B, las empresas realizan negocios con sus proveedores, distribuidores y otros socios a través de redes electrónicas. En B2C, las empresas venden productos y servicios a los consumidores. Aunque B2C es el más conocido por el público en general, B2B es la forma que realmente domina el comercio electrónico en términos de ingresos.

3.1.3 Modelo de negocio dropshipping

El interés de las empresas en el modelo de negocio de dropshipping B2B se debe en gran medida al hecho de que esta forma de gestionar el flujo ofrece ventajas logísticas, lo cual resulta en una gran reducción de gastos en almacenamiento y gestión de envíos de los productos. En la práctica, el dropshipping funciona como el envío directo, pero sin necesidad de un centro de distribución. Una empresa puede utilizar este proceso de venta para vender un producto sin tener que gestionar su propio inventario. Tan pronto como un cliente ordena un producto, la información se envía al proveedor, quien se encarga de enviar el artículo directamente. Por lo tanto, este método de suministro reduce las diversas etapas intermedias de la cadena de suministro para maximizar la eficiencia, minimizar los costos y plazos de entrega.

El dropshipping no debe confundirse con la pura venta online con proveedores especializados cuyo negocio se dedica al comercio electrónico. La idea del dropshipping B2B es proporcionar envíos directos de los puntos de almacenamiento a los puntos de venta para complementar su principal modelo de distribución tradicional.

3.1.4 Informe anual de API e integración – 2022

En su informe pudieron concluir – “Las API, la integración y los microservicios sustentan el viaje de transformación digital de la mayoría de las organizaciones empresariales y, por lo tanto, siguen siendo esenciales para sus operaciones y los productos que ofrecen.”

La importancia de las API en los comercios electrónicos cumple un papel fundamental en la escalabilidad, debido a que ayudan a los minoristas a entender mejor a sus clientes y, por tanto, ofrecerles mejores productos que se ajusten a los requerimientos del mercado. Esto se consigue gracias a los conocimientos que pueden aportar el uso de las API.

Figura 1

Informe anual de API e integración – 2022 | Software AG & Vanson Bouner

Nota. El gráfico muestra la importancia que tienen los servicios API para las organizaciones en los años 2021 y 2022. Tomado de EBANX (2021/2022). "Cómo los pagos digitales y el comercio electrónico están ganando terreno en América Latina". *Estudio más allá de las fronteras 2021/2022.*

La implementación de APIs como método de creación de servicios que ayuden a mejorar los procesos de logística y marketing en los comercios resulta indispensable para ganar una posición en el mercado. La creación de servicios basados en API para la gestión de comercios electrónicos es cada vez más común para integrar diferentes funcionalidades.

3.1.4.1 Comercio electrónico en América Latina.

El comercio electrónico en América Latina ha tenido un crecimiento que ha estado por encima que muchas otras regiones en el mundo, y se espera que continue con promedio del 31% anual hasta 2025, según estudios realizados por Americas Market Intelligence. Esto implica que el comercio en América Latina puede duplicarse de aquí a tres años, llegando a muchas más personas que tendrán acceso a compras en línea, productos digitales y viajes. Este crecimiento no

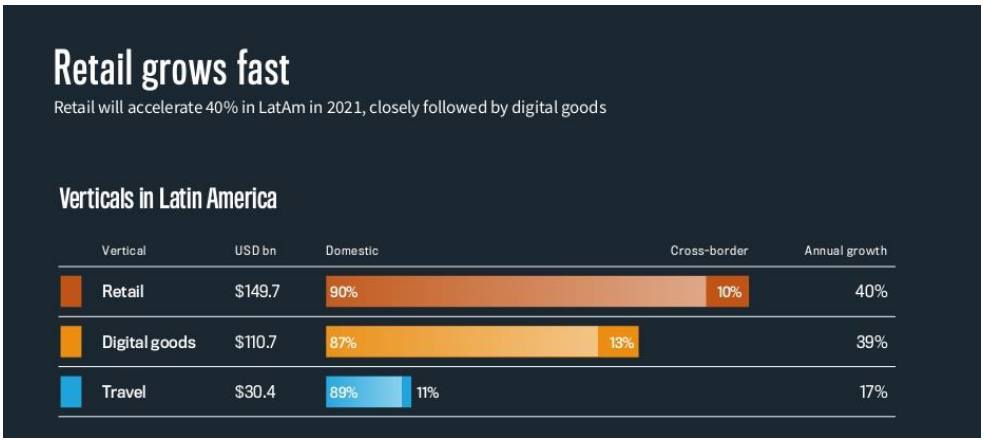
solo se aplica a lideres en el mercado como lo son Brasil, México y Argentina, sino además a países con menos influencia, donde se espera que las tasas de crecimiento lleguen hasta un 70% anual. El panorama del mercado digital en América latina resulta ser amplio en tres aspectos: el profundo impacto que ha tenido el comercio móvil en la región, la digitalización del comercio minorista y el potencial de un nuevo punto óptimo como lo es América Central.

3.1.4.2 Crecimiento del mercado minorista, productos digitales y viajes.

El campo de más crecimiento en América Latina en el año 2021 fue el comercio minorista llegando a una cifra del 40% seguido del comercio de los productos digitales.

Figura 2

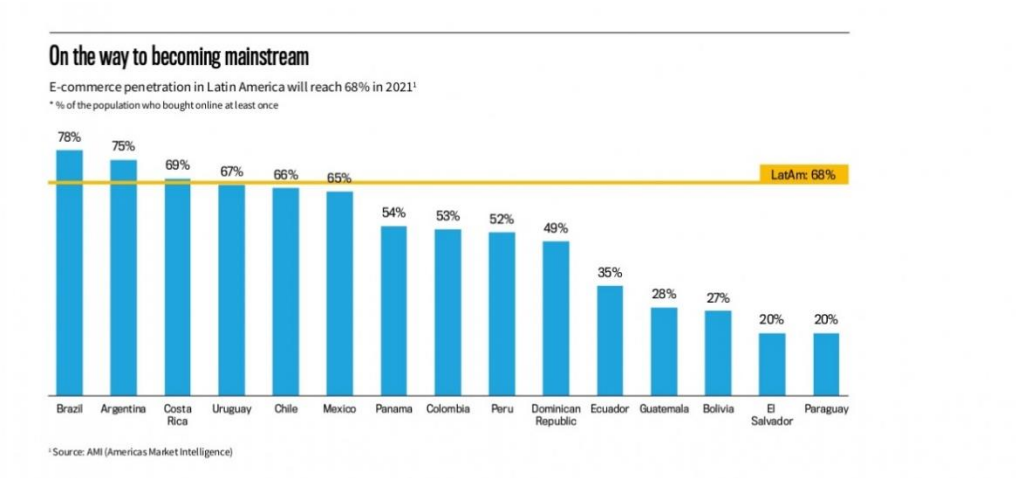
Mayor crecimiento económico en Latinoamérica



Nota: El campo de más crecimiento en América Latina en el año 2021 fue el comercio minorista llegando a una cifra del 40% seguido del comercio de los productos digitales. Gráfico: EBANX | Fuente: AMI (Americas Market Intelligence), World Bank | Considerando 15 países de América Latina

Figura 3

Porcentaje de compras en línea en 2021



Nota: Porcentaje de población en América Latina que ha comprado al menos una vez en línea alcanzo un 68% en 2021. Gráfico: EBANX | Fuente: AMI (Americas Market Intelligence), World Bank | Considerando 15 países de América Latina,

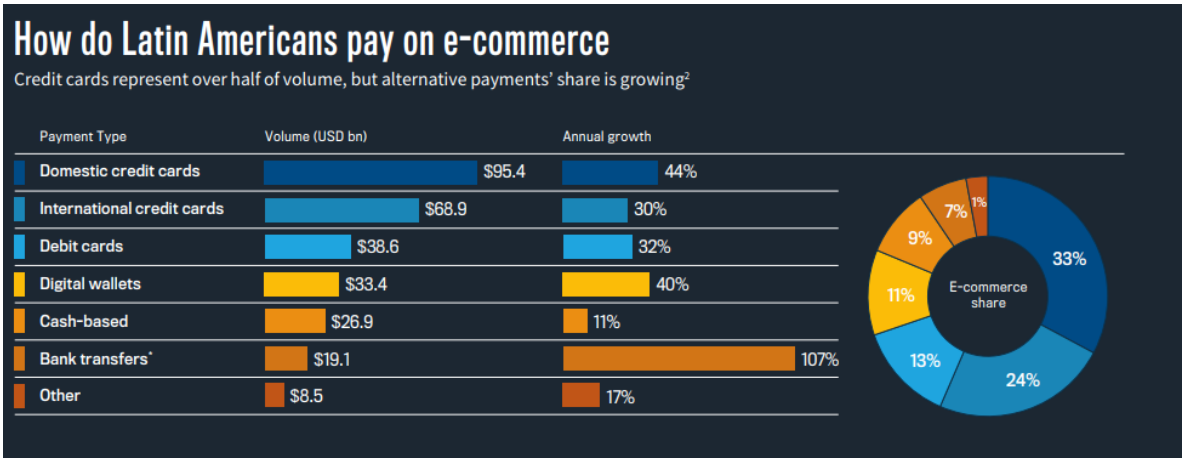
3.1.4.3 Métodos de pagos digitales.

El comercio digital en América Latina se está transformando gracias a nuevos métodos de pago que se están convirtiendo en los favoritos de muchas más personas para realizar sus compras, lo que ha permitido un crecimiento en el comercio electrónico trayendo nuevos consumidores.

Porcentaje del total de ventas en los comercios electrónicos en América Latina según su forma de pago. Las tarjetas de créditos representan más de la mitad del volumen de las ventas, pero siguen creciendo otras alternativas.

Figura 4

Cómo se paga en los comercios en línea en América Latina



Nota. Gráfico: EBANX | Fuente: AMI (Americas Market Intelligence), World Bank | Considerando 15 países de América Latina,

3.2 Marco metodológico

3.2.1 Arquitectura basada en microservicios

Una arquitectura de microservicios consta de una colección de servicios autónomos y especializados, cada uno de estos servicios es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado. ("Diseño de arquitectura de microservicios - Azure Architecture Center ...") Los microservicios resultan sencillos de implementar, escalar y administrar debido a su arquitectura descentralizada y libertad tecnológica, que permite a grupos de desarrolladores reducidos especializados hacer las labores de actualización de forma ágil.

Otros beneficios que se deben de mencionar son:

- La resistencia al cambio y a los errores, si hay algún error, las aplicaciones lo manejan degradando la funcionalidad sin bloquear toda la aplicación.

- Código reutilizable, un servicio escrito para una determinada función se puede usar como un componente básico para otras funcionalidades de otros servicios, la división del software en módulos pequeños y bien definidos ayuda a reutilizar el código acoplándolo y comunicando varios microservicios.
- Escalado flexible, los microservicios permiten que cada servicio se escale de forma independiente para satisfacer la necesidad de nuevas características demandadas.

Una arquitectura basada en microservicios igual tiene ciertos desafíos que se deben tener presente para su implementación, los que más destacan son:

- Falta de integridad de los datos, debido a que cada microservicio es responsable de la conservación de sus propios datos, por lo que la coherencia de los datos puede suponer un problema
- Latencia en la red, causado por la implementación de muchos servicios que tienen llamados internos lo que ralentiza la respuesta, si cada uno de estos servicios se encuentran en servidores diferentes.
- Fallos por control de versiones, es posible que varios servicios se actualicen en cualquier momento, por lo que, sin un cuidado de diseño estos podrían presentar problemas de compatibilidad con otros servicios.

3.2.1.1 Docker

Docker es una herramienta de virtualización de contenedores que permite empaquetar aplicaciones en un contenedor aislado que contiene todo lo necesario para que la aplicación se ejecute. El contenedor de Docker incluye la aplicación, sus dependencias y cualquier otra biblioteca o herramienta necesaria para ejecutarla.

En el contexto de DevOps, Docker se utiliza como una herramienta de integración y entrega continua (CI/CD) para simplificar el proceso de implementación de aplicaciones. Al empaquetar la aplicación en un contenedor, se puede garantizar que se ejecutará de manera confiable en cualquier entorno, desde el desarrollo hasta la producción.

Los equipos de DevOps pueden usar Docker para construir, probar y entregar aplicaciones de manera más eficiente y escalable. Al permitir que los equipos de desarrollo y operaciones trabajen en el mismo entorno, Docker ayuda a eliminar problemas de compatibilidad y reduce el tiempo de inactividad al simplificar el proceso de implementación.

3.2.1.2 Docker Compose.

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multi-contenedor en un entorno de desarrollo o producción. Se trata de una solución sencilla y eficiente para administrar múltiples contenedores Docker en un solo archivo de configuración.

Con Docker Compose, puedes especificar todos los contenedores, imágenes, volúmenes y red que necesitas para hacer funcionar tu aplicación en un solo archivo, y utilizar un comando simple para iniciar y detener la aplicación. Esto significa que puedes ahorrar tiempo y reducir errores al automatizar la configuración de la aplicación.

Además, Docker Compose también te permite escalar fácilmente tus contenedores en función de tus necesidades. Por ejemplo, puedes aumentar el número de contenedores en una aplicación que experimenta un aumento en el tráfico.

3.2.1.3 Cloud computing

En pocas palabras, la computación en la nube es la entrega de servicios informáticos, incluidos servidores, almacenamiento, bases de datos, redes, software, análisis e inteligencia, a través de Internet para ofrecer una innovación más rápida, recursos flexibles y economías de

escala. (“¿Qué se entiende por computación en la nube? - Tus Respuestas”) (“Que es la “Computación en La Nube” (Cloud Computing)”) Por lo general, solo se paga por los servicios en la nube que se usan, lo que lo ayuda a reducir sus costos operativos, ejecutar su infraestructura de manera más eficiente y escalar a medida que cambian las necesidades de su negocio.

3.2.3 API REST

La API es una interfaz a través de la cual los desarrolladores pueden interactúan más rápido con los datos, un API que está bien diseñada siempre es fácil de usar y facilita la vida del desarrollador. Uno de los tipos más populares de API para crear aplicaciones de microservicios se conoce como "API RESTFUL", "API REST" o “REST API”, el cual es un estándar popular entre los desarrolladores porque utiliza el protocolo HTTP, con los que la mayoría de los desarrolladores están familiarizados y les resulta fácil usar, a continuación, las características del API RESTFUL:

- Una API que utiliza el modelo REST (transferencia de estado representacional).
- Se basa en la codificación HTTP (Hypertext Transfer Protocol).
- Utiliza encriptación SSL (Secure Sockets Layer).
- Independiente del lenguaje, ya que puede usarlo para conectar aplicaciones y microservicios escritos en diferentes lenguajes de programación.
- Permiten crear una aplicación web con operaciones CRUD (crear, recuperar, actualizar, eliminar).

3.2.3.1 Transferencia de estado representacional

La transferencia de estado representacional (REST – Representational state transfer) es un estilo de arquitectura de software que describe una interfaz uniforme entre componentes desacoplados en un protocolo cliente/servidor, el termino se originó en el año 2000, por Roy Fielding en su tesis doctoral quien fue uno de los principales autores del protocolo HTTP.

En la actualidad el termino REST ha adoptado una definición más amplia a la de un principio, actualmente describe cualquier interfaz entre sistemas que implemente directamente el protocolo HTTP para realizar sus operaciones de obtención de datos y alteración de estos en formatos (JSON, XML, entre otros), bajo el seguimiento de algunos diseños fundamentales. Entre ellos se encuentra el protocolo cliente/servidor sin estado en el que cada petición HTTP contiene toda la información requerida para ser ejecutada; almacenamiento de cache en la memoria del cliente o del navegador, lo que mejora aún más la escalabilidad y el rendimiento; operaciones bien definidas que se aplican a todos los recursos de información a través de las peticiones HTTP; una sintaxis universal para identificar los recursos (URI); hipermedios para las transiciones de estado de la aplicación que típicamente se representa en formatos como HTML o XML.

3.2.3.2 Ventajas de REST.

Entre las principales ventajas que tiene el desarrollo de un API REST frente a otras alternativas como lo son SOAP, XML-RPC, etc. Es debido a sus principales fundamentos de diseño como se mencionaba, además de su escalabilidad dado por su fácil implementación. Por ejemplo, el no tener estado implica no tener que almacenar ninguna de las solicitudes anteriores lo que mejora el rendimiento. La implementación del API REST facilita el almacenamiento en caché, puede almacenar fácilmente en caché las solicitudes GET y POST; es fácil actualizar los datos en la base de datos en cualquier momento, dada la flexibilidad de responder a muchos clientes que solicitan diferentes tipos de datos.

3.2.3.3 Seguridad de las API.

Uno de los principales fundamentos en la implementación de un API REST es el uso del protocolo cliente/servidor, por ello siempre resulta indispensable gestionar y controlar los datos que fluyen entre ambas partes para su seguridad. Teniendo esto en cuenta, existen distintas

tecnologías para fortalecer la seguridad. Transport Layer Security (TLS), es un estándar que mantiene privada la conexión y verifica que los datos enviados estén cifrados y no se modifiquen, por otro lado, se pueden establecer reglas de limitación para proteger las API de ataques de denegación de servicio y picos de uso. Otros métodos de seguridad que se usan en conjunto son, los administradores de redes o analizadores de protocolos (packet sniffers) que monitorean y realizan pruebas de diagnóstico; la implementación de puertas de enlace de API también conocidas como API Gateway para autentificar, controlar y analizar el tráfico del API; y el estándar de autenticación basado en token Open Authorization (OAuth), en donde se permite que el servidor autentique la sesión en el protocolo HTTP.

3.2.3.4 Implementación de seguridad OAuth con Express.

Cuando se habla de OAuth es necesario saber que la autorización generalmente implica restringir ciertas funciones a los clientes. Estas funciones pueden ser métodos, páginas o endpoint del API REST. El middleware Express.js permite aplicar ciertas reglas sin problemas a todas las rutas, grupos de rutas, o rutas individuales, por ejemplo:

- Todas las rutas: app.get('*', auth)
- Grupos de rutas: app.get('/api/*', auth)
- Rutas individuales: app.get('/admin/users', auth)

En los ejemplos anteriores, auth() es una función con tres parámetros: req, res y next. Por ejemplo, en este middleware, puede llamar al servicio OAuth o consultar una base de datos para obtener el perfil de usuario para autorizarlo, verificar el JWT (JSON Web Tokens) o sesión web para autenticar al usuario.

3.2.4 Desarrollo ágil

("¿Qué es Agile? - Azure DevOps | Microsoft Learn") ("Desarrollo ágil de software - Metodologías 【2022】.") Ágil es un término que describe enfoques para el desarrollo de software que enfatizan la entrega incremental, la colaboración en equipo, la planificación y el aprendizaje continuos. El término Ágil fue acuñado en 2001 en el Manifiesto Ágil, en el que se propuso establecer principios para guiar un mejor enfoque del desarrollo de software. En esencia, el manifiesto declara cuatro declaraciones de valores que representan la base del movimiento Agile. Tal como está escrito, el manifiesto dice:

Hemos llegado a valorar:

- Individuos e interacciones sobre procesos y herramientas.
- Software de trabajo sobre documentación completa.
- Colaboración con el cliente sobre la negociación del contrato.
- Responde al cambio sobre el siguiente plan.

El manifiesto no implica que los elementos del lado derecho de estas declaraciones no sean importantes o necesarios. Más bien, los elementos de la izquierda son simplemente más valiosos.

3.2.5 Scrum

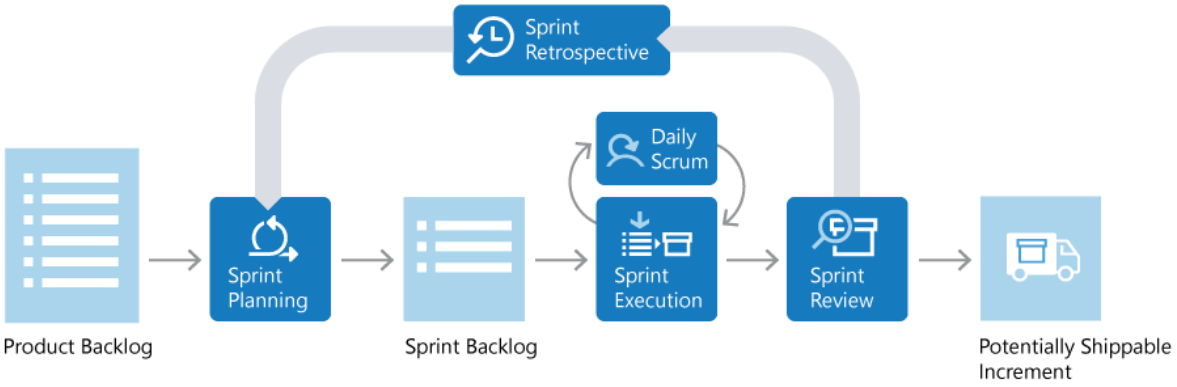
Scrum es un marco utilizado por los equipos para gestionar su trabajo. "Scrum implementa los principios de Agile como un conjunto concreto de artefactos, prácticas y roles." ("¿Qué es Scrum? - Azure DevOps | Microsoft Learn") ("¿Qué es Scrum? - Azure DevOps | Microsoft Learn")

El siguiente diagrama detalla el ciclo de vida iterativo de Scrum. ("¿Qué es Scrum? - Azure DevOps | Microsoft Learn") ("¿Qué es Scrum? - Azure DevOps | Microsoft Learn") Todo el ciclo

de vida se completa en períodos de tiempo fijos llamados sprints. Un sprint suele durar de 2 a 4 semanas.

Figura 5

Flujo de trabajo de SCRUM



Nota: Se muestra el flujo de trabajo del marco de trabajo SCRUM en el que puede ver la importancia de la realimentación de los procesos para mantener la calidad del proyecto ("¿Qué es Scrum? - Azure DevOps | Microsoft Learn") ("¿Qué es Scrum? - Azure DevOps | Microsoft Learn")

3.2.5.1 Roles Scrum. Scrum prescribe tres roles específicos:

Product owner: Responsable de lo que el equipo está construyendo y por qué lo están construyendo. El product owner es responsable de mantener el product backlog actualizado y en orden de prioridad.

Scrum master: Responsable de garantizar que el equipo siga el proceso Scrum, buscando cómo el equipo puede mejorar, al tiempo que resuelve los impedimentos y otros problemas de bloqueo que surgen durante el sprint.

Scrum team: Son los individuos que realmente construyen el producto. El equipo es quien determina la ingeniería del producto y la calidad que lo acompaña.

3.2.5.2 Artefactos y eventos

Product backlog: es una lista de elementos priorizada por valor que el equipo puede ofrecer. Los elementos en la parte superior del product backlog siempre deben estar listos para que el equipo los ejecute.

Sprint planning y sprint backlog: el sprint planning es una reunión donde el equipo de desarrollo de software planifica el trabajo a realizar durante un Sprint, que es un período corto de tiempo. Durante la reunión, se revisan y priorizan las funcionalidades a implementar y se dividen en tareas manejables. El resultado de la reunión es un plan detallado de trabajo y un conjunto de tareas para completar durante el Sprint.

El sprint backlog es una lista priorizada de tareas específicas que el equipo de desarrollo ha acordado completar durante el Sprint. El equipo se enfoca en completar estas tareas durante el Sprint para alcanzar los objetivos acordados en la reunión de sprint planning. La lista se actualiza diariamente para reflejar el progreso y asegurar la eficiencia en la finalización de las tareas.

Sprint execution y daily Scrum: Scrum no especifica cómo debe ejecutar el equipo, queda para que el equipo lo decida. El daily Scrum es una reunión diaria limitada a quince minutos. En donde cada miembro del equipo informa brevemente su progreso desde ayer, los planes para hoy y cualquier cosa que impida su progreso.

Task board: Enumera cada elemento del trabajo pendiente en el que está trabajando el equipo, desglosado en las tareas necesarias para completarlo. Las tareas se colocan en las columnas Por hacer, En curso y Terminado según su estado.

Sprint review: el equipo demuestra lo que ha logrado a las partes interesadas. Hacen una demostración del software y muestran su valor.

Sprint retrospective: el equipo se toma el tiempo para reflexionar sobre lo que salió bien y qué áreas necesitan mejorar. El resultado de la retrospectiva son acciones para el próximo sprint.

Increment: se denomina increment o potentially shippable increment, al resultado de un sprint que debe ser de calidad entregable, el cual debe cumplir con todos los criterios de calidad establecidos por el Scrum team y product owner.

3.2.5.3 Pilares y valores de Scrum

Transparencia

- Todas las partes involucradas en el proceso deben tener acceso a la información y ser capaces de comprender lo que están viendo.
- La claridad y accesibilidad de los aspectos importantes del proceso son fundamentales para la transparencia.

Inspección

- Los usuarios del método Scrum deben examinar regularmente sus artefactos y el progreso para detectar cualquier desviación indeseada.
- La inspección debe llevarse a cabo con la frecuencia adecuada para no interrumpir el flujo de trabajo.

Adaptación

- Cuando se detectan desviaciones que están fuera de los límites aceptables, es necesario ajustar el proceso.
- La adaptación debe llevarse a cabo lo antes posible para minimizar posibles desviaciones mayores en el futuro.

Para garantizar la puesta en marcha de los 3 pilares de Scrum y la confianza de todas las personas implicadas, el Equipo Scrum debe incorporar y practicar sus valores: compromiso, coraje, foco, apertura, respeto.

3.2.6 DevOps

DevOps combina desarrollo (Dev) y operaciones (Ops) para unir personas, procesos y tecnología en la planificación, desarrollo, testeo, lanzamiento y operaciones de aplicaciones. DevOps permite que roles anteriormente aislados como desarrollo, operaciones de TI, ingeniería de calidad y seguridad se coordinen y colaboren. ("Historia y evolución de la Ingeniería de Software. timeline.")

DevOps tiene un impacto en todo el ciclo de vida de una aplicación, desde su planificación y desarrollo hasta las etapas de prueba, lanzamiento y operaciones. Cada una de estas etapas está estrechamente interrelacionada con las demás, y no se limitan a un rol específico dentro de la organización.

Para adoptar una cultura DevOps, es necesario implementar prácticas específicas en cada etapa del ciclo de vida del software. Estas prácticas están diseñadas para automatizar, mejorar y acelerar el proceso de desarrollo y despliegue de software, desde la creación del código hasta la entrega al cliente final y la gestión de la infraestructura de producción.

- Desarrollo Ágil de Software
- Integración Continua (CI) y Entrega Continua (CD)
- Infraestructura como Código (IaC)
- Control de Versiones
- Gestión de la Configuración
- Monitorización Continua
- Microservicios

3.2.6.1 Unit testing con Jest y Supertest – TypeScript.

El uso de Jest y SuperTest para realizar Unit testing en TypeScript es una forma eficiente y efectiva de asegurarte de que tu código funciona de la manera esperada. Al escribir pruebaspuedes identificar errores y evitar que al escribir nuevo código el código ya existente se vea afectado, de esta manera siempre estamos seguros de que si los test pasan el nuevo código no ha sobre escrito o afectado el código ya existente.

3.2.6.2 Broker de mensajeria RabbitMQ con mqplib TypeScript.

El protocolo Publisher/Subscriber es un modelo de comunicación en el que un publisher (editor) envía mensajes a una queue (canal/cola), mientras que los subscribers (suscriptores) escuchan y reciben estos mensajes.

Implementar el protocolo Publisher/Subscriber con RabbitMQ y TypeScript es una forma eficiente de comunicar y coordinar múltiples microservicios. Al utilizar una plataforma de mensajería como RabbitMQ, puedes asegurarte de que los mensajes se entreguen de forma confiable y escalable.

3.2.6.3 Ciclo de vida del software.

Es importante tener en cuenta que cada fase del ciclo de vida del software es esencial y que el éxito del proyecto depende de una planificación cuidadosa y una ejecución efectiva en cada una de ellas.

- Planificación: En esta fase, se establecen los objetivos y requisitos del software, se definen los alcances del proyecto y se planifican los recursos necesarios.
- Análisis: En esta fase, se profundiza en los requisitos del software y se identifican los problemas técnicos que deben ser abordados.

- Diseño: En esta fase, se crea un plan detallado para el desarrollo del software, incluyendo la arquitectura, la interfaz de usuario y los detalles técnicos.
- Desarrollo: En esta fase, se lleva a cabo la codificación del software y se realiza la prueba unitaria.
- Pruebas: En esta fase, se lleva a cabo la verificación y validación del software para asegurarse de que cumpla con los requisitos y es funcional.
- Implementación: En esta fase, se instala el software en el entorno de producción y se prepara para su uso por parte de los usuarios.
- Mantenimiento: En esta fase, se realiza el soporte técnico y se corrigen errores y problemas que surjan después del lanzamiento del software.
- Descarte: En esta fase, se descartan los sistemas obsoletos y se reemplazan por soluciones más actualizadas.

3.2.6.4 CI/CD

CI/CD es un acrónimo que se refiere a las prácticas de Integración Continua (Continuous Integration) y Entrega Continua (Continuous Delivery) dentro del enfoque de DevOps.

La Integración Continua se refiere a una práctica de desarrollo de software en la que los desarrolladores integran su trabajo en un repositorio compartido varias veces al día. Cada integración se verifica automáticamente mediante la ejecución de pruebas automatizadas para detectar errores temprano en el ciclo de vida del desarrollo de software. Este enfoque de desarrollo acelera el tiempo de entrega y mejora la calidad del software.

La Entrega Continua se refiere a una práctica de desarrollo de software en la que el software se puede entregar a los usuarios o a producción de manera rápida y confiable en cualquier momento. La Entrega Continua se logra a través de la automatización del proceso de entrega y la

realización de pruebas exhaustivas para garantizar que el software sea de alta calidad y esté listo para su implementación.

En conjunto, la Integración Continua y la Entrega Continua forman una cadena de herramientas que se conoce como CI/CD, que se utiliza para crear y entregar software de alta calidad de manera rápida y confiable. Las prácticas de CI/CD se integran con el enfoque de DevOps para fomentar la colaboración y la comunicación entre los equipos de desarrollo, operaciones y seguridad, y mejorar la eficiencia y la calidad del software entregado.

3.2.6.4.1 Gitlab

GitLab es una plataforma de DevOps que proporciona una amplia variedad de herramientas y funcionalidades para el desarrollo, pruebas, implementación y monitoreo de aplicaciones. La plataforma incluye un conjunto de herramientas de CI/CD que permite la implementación de prácticas de Integración Continua y Entrega Continua.

GitLab ofrece un conjunto de herramientas para implementar prácticas de CI/CD en el proceso de desarrollo de software, que incluyen la definición de un pipeline de CI/CD, la configuración de runners, la automatización de pruebas y la entrega continua.

Para implementar CI/CD con GitLab, se pueden seguir los siguientes pasos generales:

- Crear un proyecto en GitLab y configurar un repositorio de código fuente.
- Crear un archivo gitlab-ci.yml en la raíz del repositorio para definir las etapas del pipeline de CI/CD.
- Configurar los runners de GitLab, que son agentes que ejecutan los trabajos de CI/CD. Los runners pueden ser compartidos o dedicados a un proyecto específico.

- Definir los trabajos de CI/CD en el archivo.gitlab-ci.yml. Cada trabajo debe ser definido por etapas y puede incluir tareas como compilación de código, pruebas unitarias, pruebas de integración, análisis estático de código, entre otras.
- Ejecutar el pipeline de CI/CD. Cada vez que se realiza un push al repositorio, GitLab inicia automáticamente la ejecución del pipeline, que sigue las etapas definidas en el archivo. gitlab-ci.yml.
- Configurar la Entrega Continua, que implica la automatización de la implementación del software en entornos de prueba y producción. Esto se puede lograr utilizando herramientas de orquestación de contenedores como Kubernetes o mediante la automatización de scripts de implementación.
- Monitorear y ajustar el pipeline de CI/CD según sea necesario para garantizar la calidad y la eficiencia del proceso de desarrollo y entrega de software.

4. Metodología

El desarrollo del proyecto en A&ATic está sujeto a una metodología de desarrollo de software DevOps siguiendo el marco de trabajo colaborativo de Scrum, de acuerdo con los parámetros organizacionales de la empresa. Además, se hizo uso de diferentes plataformas para gestionar y administrar el proyecto, haciendo el control de versiones con Git y la plataforma de GitLab para la integración continua, se utilizaron las plataformas de atlassian con confluence para la documentación de la planificación y diseño del proyecto a realizar, se usó Zoho Sprints para gestionar los sprints y tareas definidas en la estimación, se complementó el desarrollo con Skype que permitió una comunicación continua entre el equipo de desarrollo. La realización del proyecto se dio en dos etapas, las cuales corresponden a la definición del proyecto en la etapa uno, y el desarrollo del proyecto en la etapa dos.

En la etapa uno se definió el tema del proyecto se hizo un proceso de investigación acerca de los temas de comercio en línea, comercio retail, integración de plataformas, importancia de las API o servicios web en la actualidad, pudiendo entender la mejor el proyecto de integración de plataformas, que buscar generar un servicio más eficiente para diferentes comerciantes. Esta primera etapa fue una etapa en la que se buscó sumergirse en el problema del proyecto entendiendo mejor el estado del arte de las integraciones, ya que estos proyectos que son cada vez más comunes entre las organizaciones, de esta manera se logró un mejor entendimiento de lo que se quería y como plantear una solución integra a la problemática que se me había asignado, permitiendo generar soluciones que fueran acorde a lo que se me había pedido. Gracias a esta investigación, se pudo entender de manera más clara el alcance del proyecto y las necesidades de los comerciantes en cuanto a la eficiencia que podría generar la integración de diversos proveedores con los

comerciantes, prestándoles un servicio que incorporará diversos procesos organizacionales para la gestión de las plataformas ecommerce.

En la etapa dos se empezó a realizar el proyecto con lo aprendido en la etapa uno, continuando con el aprendizaje y empezando el desarrollo del proyecto. Esta etapa fue un proceso de aprendizaje que permitió poner a prueba los conocimientos adquiridos en la universidad y crecer profesionalmente en el área del desarrollo del software, siguiendo una de las prácticas más eficientes del desarrollo como lo es la metodología DevOps, junto con un marco de trabajo colaborativo como lo es Scrum, a continuación, se describirán cada una de las fases que se realizaron para lograr los objetivos de desarrollar el trabajo de grado.

En la primera fase del desarrollo del proyecto corresponde a la planificación y diseño del proyecto, en donde se empezó con la definición de los requerimientos la cual fue una fase de realimentación y creatividad para plantear una solución temprana que sirviera como molde a la definición final que se conseguiría en la estimación de las tareas a realizar, en esta fase se definieron las historias de usuario a partir de los requerimientos, exponiendo las funcionalidades a desarrollar, por otro lado se concretaron las tecnologías que se iban a usar en el desarrollo del proyecto la cuales fueron Typescript como extensión de tipado para Javascript como lenguaje de desarrollo en nodejs, las dependencias para el proyecto, entre las más importantes estaba Express un marco de aplicación web de Node.js mínimo y flexible que proporciona un conjunto sólido de funciones para aplicaciones web, sequelize como ORM (Object Relational Mapping) para la gestión de la base de datos con MySQL, Jsonwebtoken para la creación de tokens de autenticación y seguridad del api, Cors para el intercambio de recursos cruzados, Axios un cliente HTTP igual que Fetch, MySQL para la persistencia de los datos del API, RabbitMQ para la comunicación de los microservicios con el protocolo Publish/Subscriber, Gitlab para la integración continua y el

control de versiones con Git, y Docker siguiendo la metodología DevOps. Además, en esta primera fase se definió la arquitectura de solución del proyecto donde se estructuro y dividió la lógica de negocio del API en una serie de microservicios separados que se encargan de gestionar funcionalidades concretas relacionadas.

En la segunda fase se configuro el entorno de desarrollo, se prepara el entorno del proyecto, instalando las herramientas necesarias, como Docker y Docker-compose para la creación y ejecución de los contenedores de los microservicios, despliegue del contenedor de SonarQube para las pruebas de calidad del código, Nodejs como motor de ejecución del lenguaje de programación Javascript con Typescript, el IDE para el desarrollo del proyecto Visual Studio Code, se estableció el flujo de trabajo con Git, se configuro la conexión al repositorio con el protocolo de seguridad SSH, se siguió un modelo de control de versiones siguiendo las prácticas de Git Flow, en la creación y gestión de las ramas para la realización de las tareas definidas en la estimación del proyecto.

En la tercera fase se definieron las practicas a seguir para el desarrollo del proyecto siguiendo el marco de trabajo scrum, como los reportes diarios para la comunicación con el equipo, el cual consistía en un reporte escrito al iniciar la jordana y finalizar la jornada de trabajo, en donde se describía las tareas a realizar en el día y las tareas realizadas durante el día, describiendo cuales fueron los impedimentos que dificultaron el desarrollo de las tareas, se usó Skype para la comunicación del equipo, para poder solucionar las dudas que surgieran en el desarrollo de las tareas, y poder gestionar las dificultades mucho más rápido.

En la cuarta fase se inició con la inicialización de los microservicios definidos en la planificación y diseño del proyecto, los cuales conforman la arquitectura de solución planteada, utilizando las tecnologías definidas, se crean los diferentes microservicios que conforman la

arquitectura de la solución, utilizando Node.js y TypeScript. Cada microservicio es implementado como un contenedor de Docker, lo que facilita su gestión y despliegue. Esto permitió una mayor visibilidad en el proceso de desarrollo, dado a que el proyecto se ejecutaba en contenedores de Docker a través de Docker-compose lo que permitía que el proyecto fuera fácil de analizar, permitiendo una mayor calidad del software, en este punto se configuro la base del proyecto, para la ejecución de los microservicios en un solo entorno de ejecución con Docker-compose.

En la quinta fase se define la estructura del proyecto para el desarrollo del API separando la lógica de conexión de la base de datos (dominio), de la aplicación (casos de uso) para que el API resulte fácil de escalar a futuro, esto permitió que nuevas funcionalidades de lógica de negocio sean fáciles de incorporar en el código, permitiendo que se encarguen de gestionar las respuestas del API, Además se dio autonomía a los repositorios para realizar las consultas a la base de datos, se definieron las colas para los Publisher y Subscriber los cuales tienen como propósito comunicarse con los demás microservicios definidos para el proyecto global de la empresa, de esta manera permite poder integrar funcionalidades entre microservicios, razón por la cual se hace validación de los parámetros recibidos en los servicios del proyecto desarrollado, permitiendo que las consultas realizadas a través de este protocolo, generen errores sin tratar, haciendo más robusta el API. Por otro lado se configuro un módulo de consultas a los servicios web de los proveedores con Axios, el cual proporciona un conjunto de herramientas para realizar peticiones HTTP, lo que permitió configurar una clase de peticiones estandarizando los parámetros y cabeceras para conectarse a los servicios de los proveedores, para poder instanciar objetos de conexión a estos servicios proporcionado la ruta base del servicio web y el token de autorización al proveedor, de esta manera se puede gestionar varias conexiones a proveedores al mismo tiempo, haciendo que la integración de nuevos proveedores sea fácil de hacer, lo que se necesita es estandarizar el modelo

que llega del proveedor al modelo definido en el proyecto. Por otro lado, se creó un módulo de validación que recibiera un token de autenticación para realizar las peticiones del api, se documentó el código siguiendo el estándar de documentación JSDoc, que con el uso de compodoc una herramienta de automatización de la documentación, permite generar la documentación del código en base a la documentación existente en el mismo código, facilitando la creación de una interfaz para los desarrolladores que buscan escalar o usar el API.

En la sexta fase se realizaron las pruebas de calidad con sonarqube revisando los code smell y bugs que presenta el código desarrollando. De esta manera se validó que el proyecto no contuviese errores de codificación, se realizaron pruebas de funcionamiento con postman validando las respuesta de api, en cada uno de los endpoints desarrollados, de esta forma se verifico que todo estuviese funcionando correctamente y si hubiesen fallos poder arreglarlos en el momento, se terminó realizando las pruebas unitarias a cada una de las funcionalidades desarrolladas en los servicios de API, verificando el correcto funcionamiento con jest un framework de prueba que ayuda a automatizar las pruebas unitarias, de esta manera se mockearon las consultas a la base de datos haciendo que se hicieran pruebas unitarias de cada una de las partes sin depender de la conexión a la base de datos.

En la práctica hubo una serie de retos que fueron necesarios superar para poder realizar el proyecto estos retos hicieron que pudiera desarrollar mis habilidades en la gestión de proyectos, el primer reto que se me presento fue entender el problema que se me había asignado, esta parte resulta fundamental para poder definir una bases sólidas en las que el proyecto se pueda apoyar y crecer sin tener que reiniciar el proceso desde cero, la complejidad de este punto resulto en lo general que resultaba la problemática, dado que el campo es un área muy extensa que crece a pasos agigantados, por otro lado se pudo superar focalizando las necesidades en un grupo reducido de

funcionalidades que debía lograr el API, el planteamiento de la solución resulto difícil de definir al principio, por lo que el alcance del proyecto aún seguía borroso en lo que se buscaba, pero se fue logrando aclarar estas funcionalidades cada vez que se iteraba en las tareas a desarrollar, la escalabilidad del proyecto fue un factor reductor en la eficiencia del desarrollo debido a que si el código desarrollado continuaba de una manera poco legible, más adelante el proyecto no se podría escalar con facilidad, por lo que se siguieron consejos para hacer un código más limpio y escalable, lo que funciono y genero alivio en este punto, la documentación del código fue un problema que se me presento debido a la poca documentación realizada desde el comienzo del proyecto por lo que el proyecto contaba con una documentación dentro del código casi nula, este reto fue un proceso arduo me genero retrasos en el desarrollo de tareas posteriores, pero no fue un impedimento para lograr el objetivo final, siendo más eficiente en las tareas propuestas, por último la validación del proyecto más que un reto fue un proceso necesario pero arduo de realizar debido a la complejidad inicial que planteaba realizar pruebas funcionales con postman y pruebas unitarias con jest, igualmente más que un reto fue un proceso arduo necesario para poder realizar el proyecto de manera funcional.

4.1 Planificación

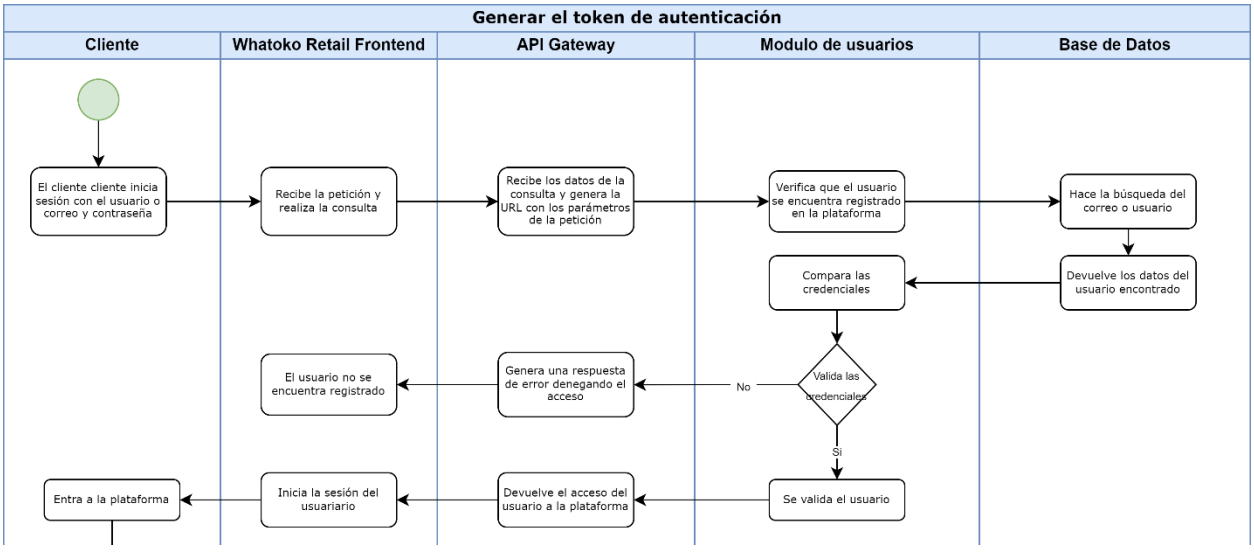
El sistema de dropshipping es una solución para los dueños de e-commerce que buscan satisfacer las necesidades de sus clientes sin tener que preocuparse por la gestión del inventario y el almacenamiento. Este modelo se basa en la integración de múltiples proveedores que ofrecen una amplia gama de productos para satisfacer las necesidades de los clientes. La idea principal detrás de este servicio es crear un entorno donde varios e-commerce puedan ser administrados como uno solo.

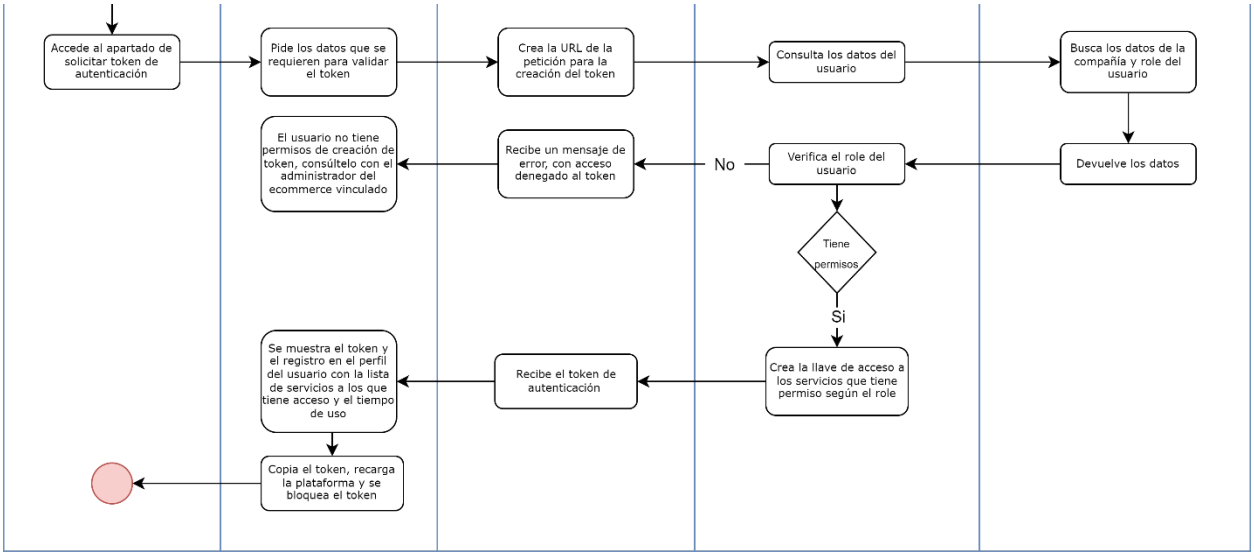
Para lograr esto, se ha desarrollado una plataforma que permite a los clientes registrados generar accesos a sus empleados con diferentes permisos de acceso a los servicios dependiendo de la sucursal de la compañia. Este enfoque permite a los clientes tener un mayor control sobre su negocio, al mismo tiempo que les permite centrarse en el marketing y la venta de productos. Además, para que el sistema de dropshipping sea eficaz, es necesario desarrollar dos servicios clave: un servicio de integración de e-commerce y un servicio de integración de proveedores. Estos servicios serán proporcionados a través de API REST y se encargarán de estandarizar y relacionar las peticiones para cargar productos de los proveedores y mostrarlos en los e-commerce de los clientes. Esto implica la automatización de los procesos de carga de productos, la actualización del stock y la creación de reglas de precios, entre otros. Al estandarizar el proceso de integración, se garantiza que los productos se carguen de manera uniforme y consistente en todos los e-commerce.

4.1.1 Diagramas de flujo de procesos de Whatoko Retail

Figura 6

Diagrama de flujo de generación del token de autenticación

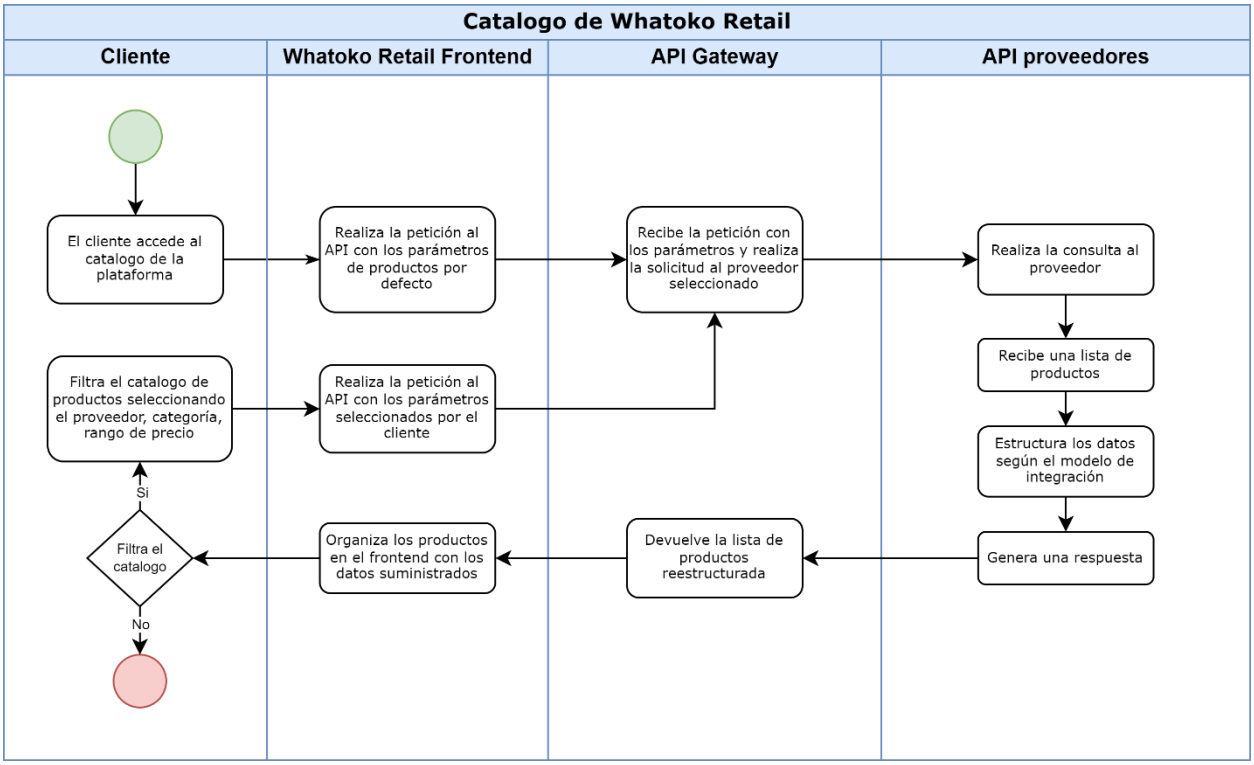




Nota: En este diagrama de flujo se recrea el proceso de creación de la llave de acceso a los servicios de la plataforma de Whatoko reatail, el token generado se crea en el módulo de usuario, se podrá acceder al token a través de una conexión a la base de datos que administra los usuarios en la plataforma, de esta manera se validara el acceso al servicio de dropshipping el cual solicitara primero las credenciales del usuario antes de realizar cualquier petición de cargue al módulo de dropshipping. Este módulo también tiene como fin poder gestionar los permisos a los diferentes servicios que se desarrollan en la plataforma, de esta manera cada compañía podrá gestionar los accesos a los diferentes servicios asociados a una sucursal dentro de la plataforma y cada usuario tendrá acceso exclusivamente a las sucursales y servicios que tenga un token de autenticación.

Figura 7

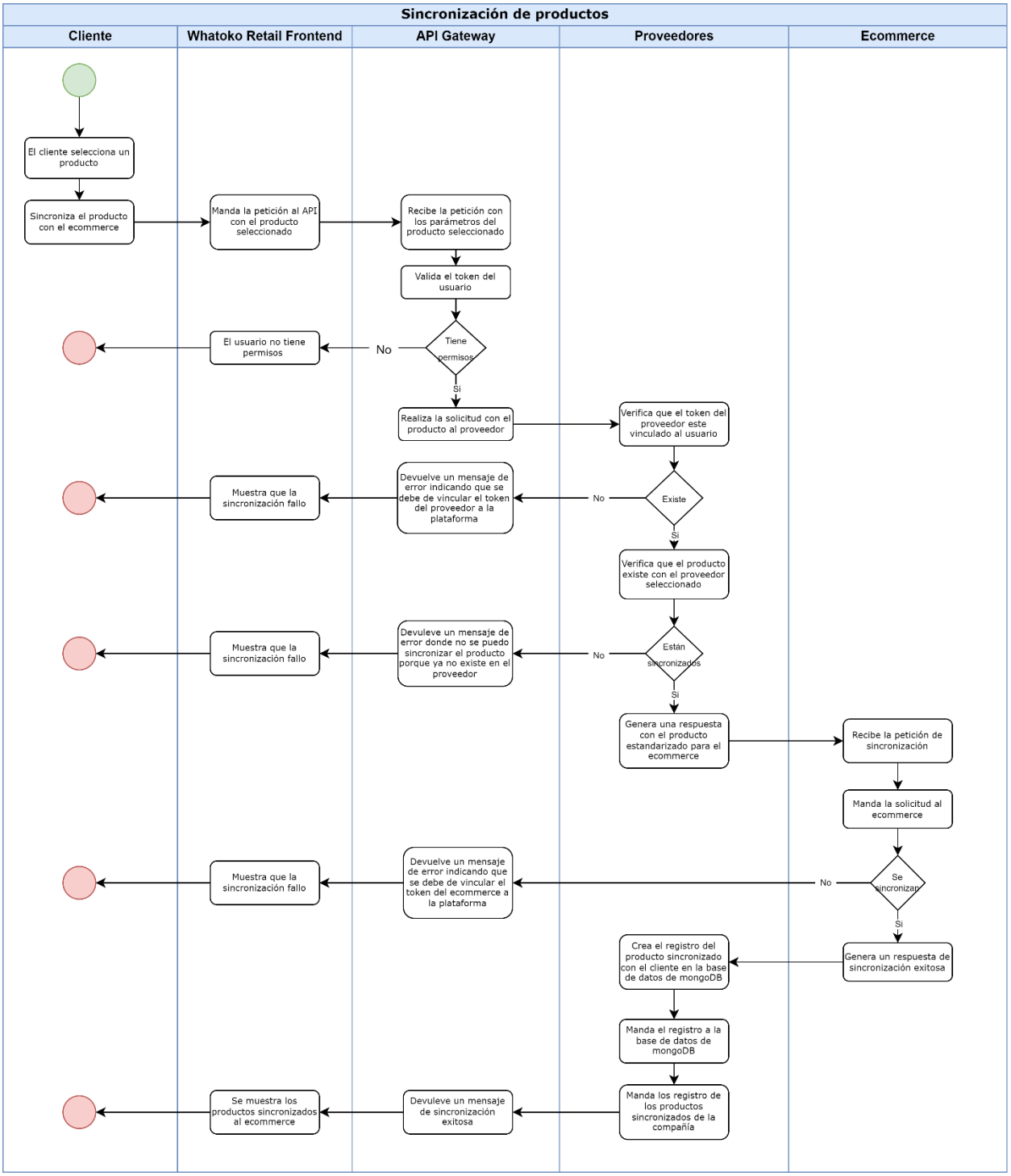
Diagrama de flujo para la consulta del catálogo de productos



Nota: En este diagrama se describe el proceso que se realiza para traer los productos de los proveedores a la plataforma y que se puedan visualizar por parte del cliente, por otro lado, el cliente podrá aplicar ciertos filtros con los que se espera que el cliente seleccione mejor los productos que necesita para su e-commerce, para filtrar los productos se permitirá filtrar por el proveedor, la categoría, un rango de precio.

Figura 8

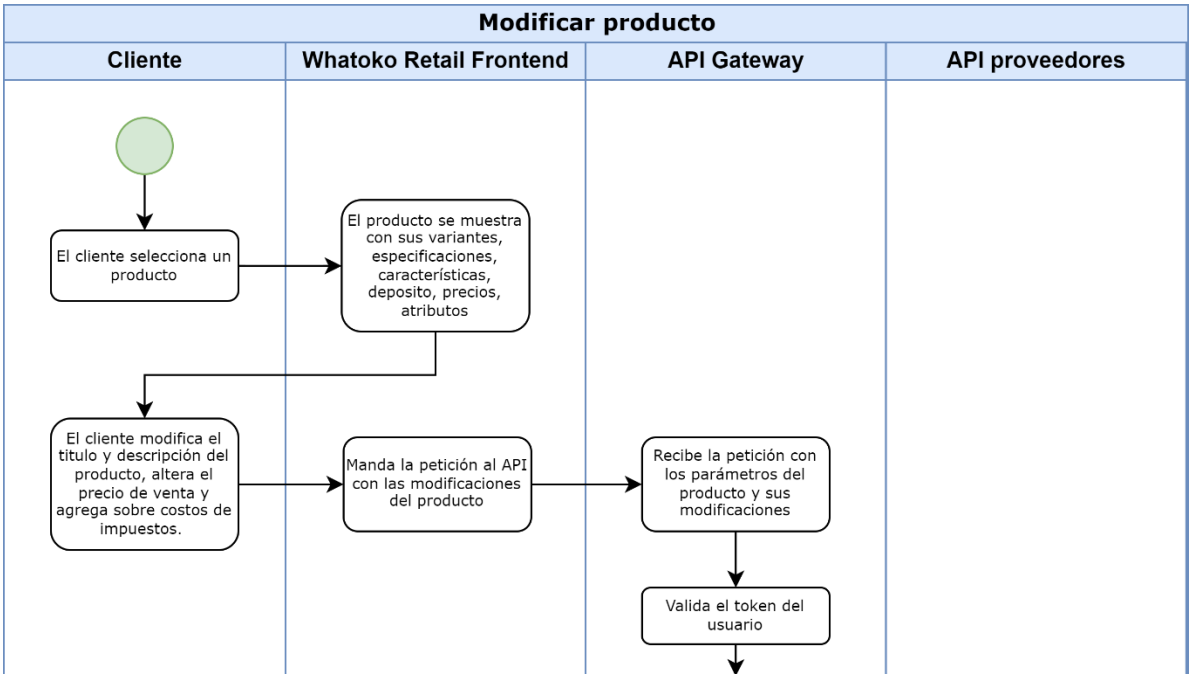
Diagrama de flujo de sincronización de productos al e-commerce

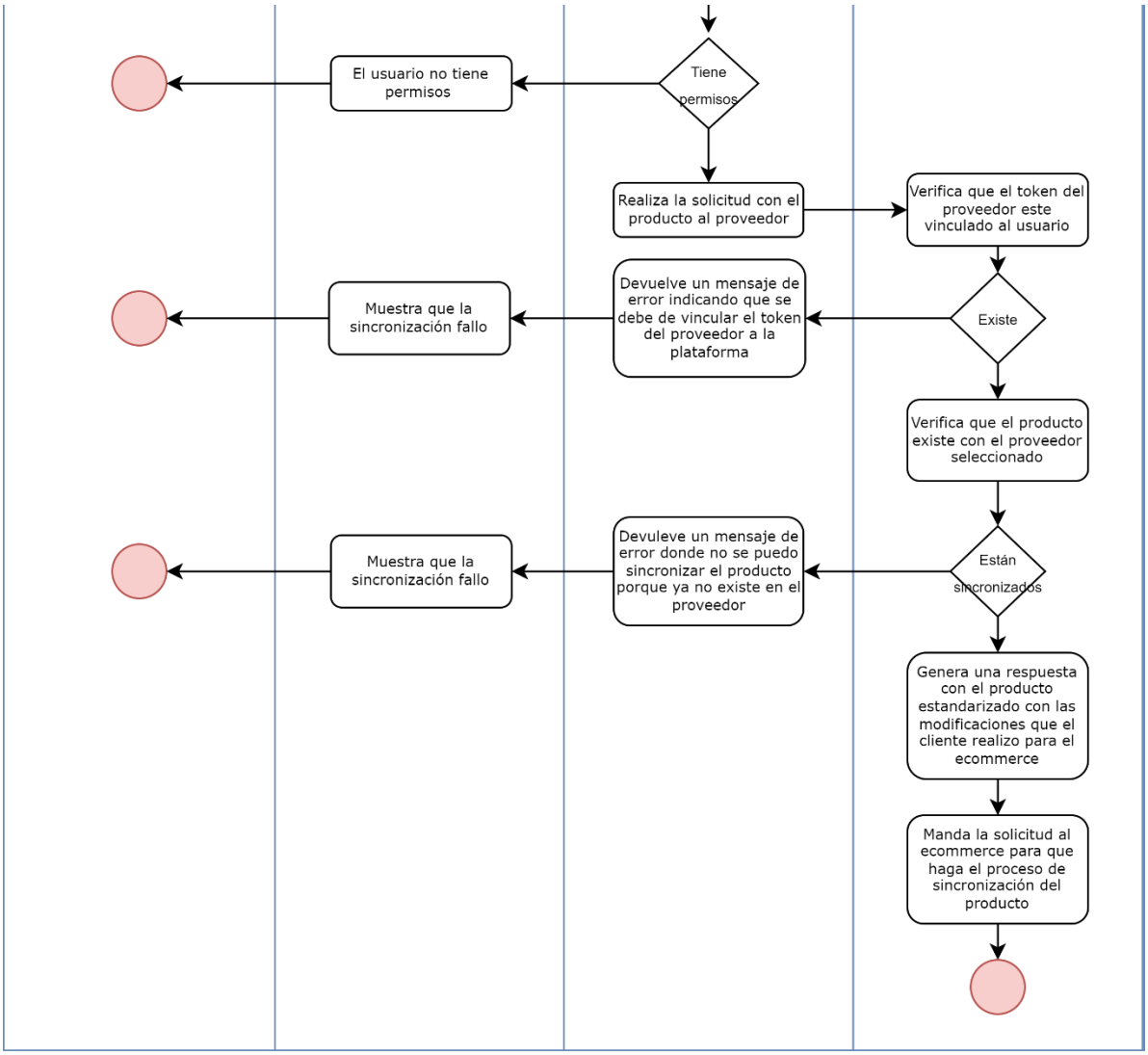


Nota: El proceso de sincronización de un producto o una lista de productos se lleva a cabo validando varios procesos, primero debemos validar que el usuario que intenta hacer la sincronización de los productos tenga los permisos en la plataforma de otro modo no se le permitirá hacer ninguna inclusión de productos al e-commerce vinculado con la cuenta, después de validar las credenciales, ser preguntara al proveedor si aún tiene el producto en stock y disponible, si el caso que hay una respuesta positiva, se hará la validación si el usuario tiene permisos de realizar peticiones al proveedor de esta manera el proceso de vinculación queda registrado al cliente y no a la plataforma de Whatoko, permitiendo que los proveedores sepan sobre sus clientes, después de validar los permisos con el proveedor se mandará la petición de sincronización al módulo de integración de e-commerce para que valide si el e-commerce ya se encuentra integrado con la plataforma y se pueda realizar el proceso de subida de los datos al e-commerce.

Figura 9

Diagrama de flujo de modificación de productos





Nota: En este diagrama se muestra como el cliente tiene el control sobre los productos que va a sincronizar con la plataforma de esta manera permite que el cliente decida cuáles serán sus márgenes de ganancia con el sistema de dropshipping, además de agregarles su toque personal al producto que podría ayudarle a destacar los entre los muchos otros que ofrece en su e-commerce, ya habiendo modificado los parámetros de los productos y sus variaciones se hará el proceso de validación de las credenciales y si todo resulta correcto, se continuara con el proceso de sincronización de productos en el e-commerce del cliente.

4.1.1.1 Conexión SSH repositorio GitLab

SSH (Secure Shell) es un protocolo de red utilizado para acceder de forma segura a dispositivos remotos a través de una conexión cifrada. En términos simples, SSH es un protocolo que te permite conectarse y controlar un dispositivo remoto, como un servidor o una computadora, de forma segura a través de Internet. Esto se logra mediante la creación de una conexión cifrada entre el dispositivo local y el remoto, lo que garantiza que la información transmitida esté protegida contra posibles interceptaciones o manipulaciones por parte de terceros.

La aplicación de SSH en un repositorio de GitLab permite una conexión segura y autenticada a través del protocolo SSH. Al utilizar SSH, se evita la necesidad de ingresar una contraseña cada vez que se accede al repositorio, lo que facilita el trabajo en equipo y la automatización de procesos.

4.2 Análisis

4.2.1 Modelo de base de datos Whatoko Retail comercios y proveedores

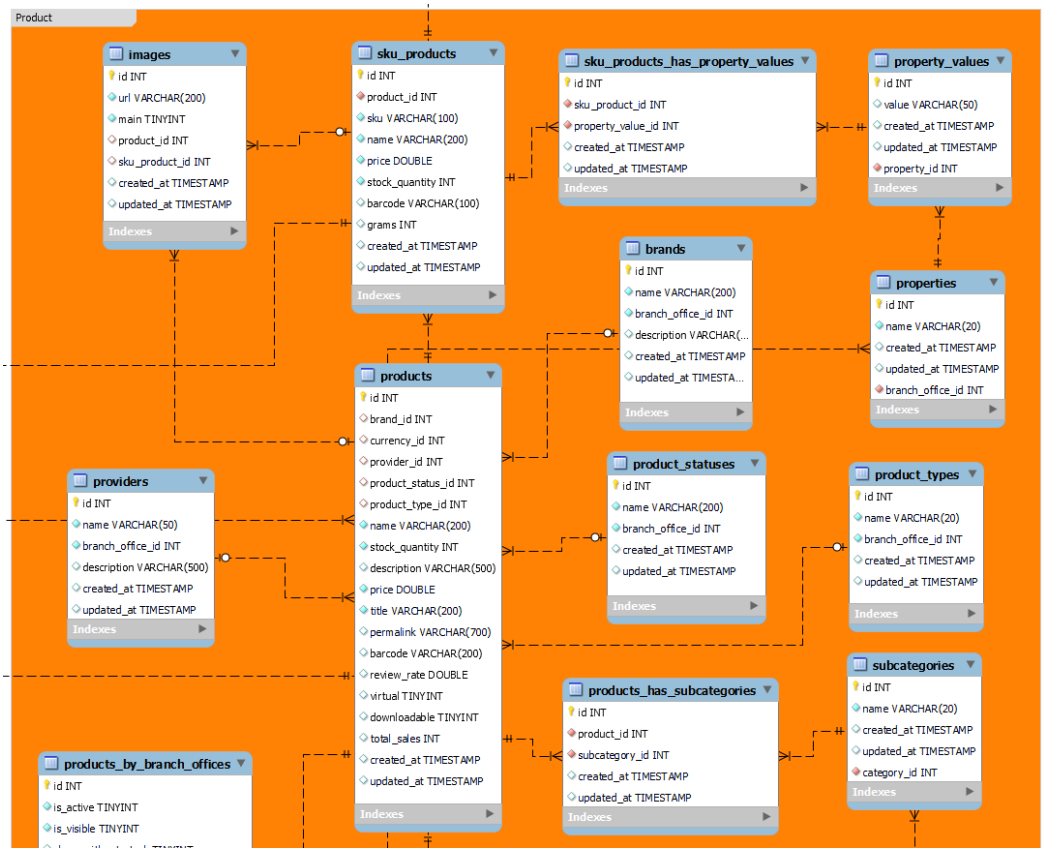
En este apartado se explica las razones de la formulación de la base de datos que se modelo, se describirá con que objetivos se definieron los datos, los métodos y formas de recolección de estos datos. Se requería crear un modelo de base de datos para la gestión y administración de e-commerce en Whatoko retail con el objetivo que los clientes puedan integrar los datos de los comercios como WooCommerce, Shopify, Vtex y Mercado Libre, y los diferentes proveedores con el fin de centralizar y facilitar los procesos de administración y gestión de los productos.

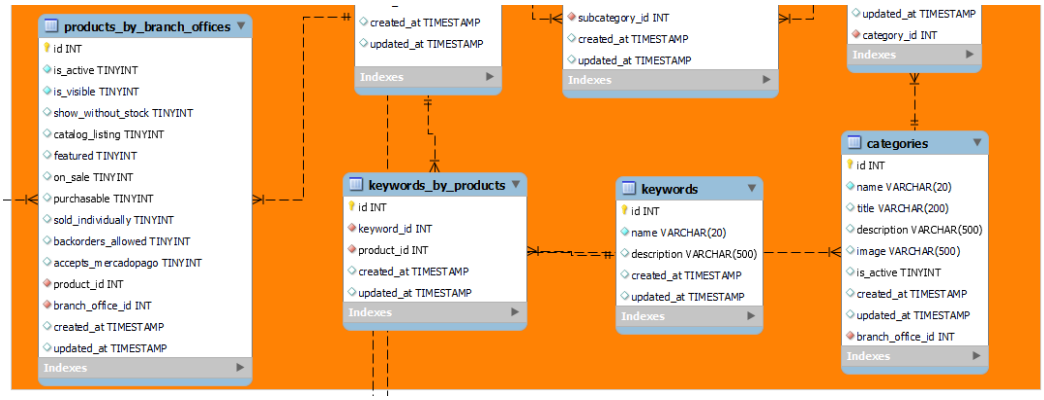
Primero se investigaron los datos de cada uno de los e-commerce antes mencionados, a través de la documentación de las APIs que disponen, en una tabla de excel recolectando los datos más importantes y representativos para la gestión de los productos, con el fin de homologar los datos presentes en cada uno. De esta manera se logra que a la hora de recopilar la información de

los comercios de los clientes se pueda gestionar de manera centralizada sin que se vea afectado por incorporar datos que no están presentes en otros e-commerce, la base de datos se trató de mantener con los datos suficientes para que se pueda realizar las modificaciones y gestión de los productos, después de definir los datos que se comparten en la plataforma e-commerce de los clientes, se hizo una homologación de los proveedores de intcomex y Aliexpress, de igual manera se homologo cuáles eran los datos que compartían entre ellos y los que se acoplaban a los ya definidos en los e-commerce, de esta manera se definieron los datos más importantes que teníamos que mantener para la integración de estas plataformas.

Figura 10

Módulo de producto y variantes junto con sus tablas complementarias





Nota: El módulo de base de datos expuesto corresponde a la parte que se encarga de gestionar los productos de los clientes provenientes de los e-commerce como de los proveedores de productos, esta modulo estandarizo los modelos de productos expuestos es los diferentes servicios REST de las plataformas e-commerce y los proveedores de Aliexpress e Intcomex.

Para recolectar los datos de productos del proveedor Intcomex se vio una limitación en los datos debido a que el API solo proporciona unos pocos, por este motivo se deberá hacer una validación con el cliente de la plataforma para que diligencie los datos faltantes y puedan integrarlos en los e-commerce. Por otra parte, el API de Aliexpress presenta una serie de datos de productos que se ven focalizados en la tabla SKU, centrando los valores en esta tabla.

Los datos que corresponden a la configuración que tiene el producto con la sucursal (branch_office) presenta una serie de validaciones en la tabla que asocia ambos modelos que ayuda a definir la configuración del producto que se va a exportar a los e-commerces.

SKU es una tabla que corresponde a los productos únicos entre todas las variaciones posibles que pueda tener el producto principal, es decir, cuando un comerciante cuenta con un producto que presenta propiedades de tipo, color, tamaño, material, talla, entre otras, cada una de estas propiedades puede generar una cantidad de valores posibles para el mismo producto, por lo que resulta necesario hacerle una diferenciación, esto lo logran los e-commerce y en general las

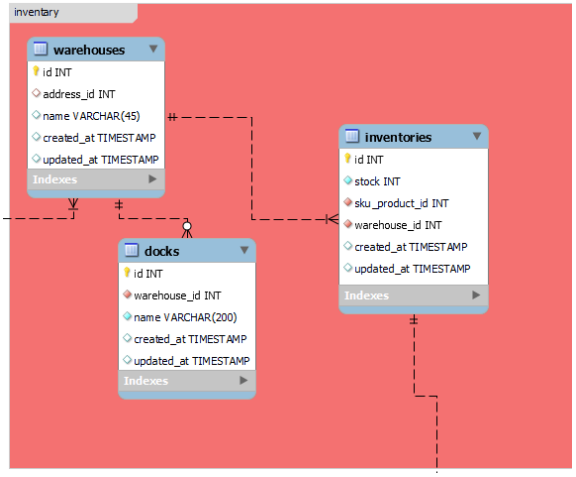
plataforma creando una tabla que lleve la unicidad de los productos dependiendo de estas propiedades.

Para la especificación de los productos, en general todas las plataformas implementan un modelo basado en propiedades y valores para los productos, haciendo que estos productos a pesar de tratarse del mismo se diferencien entre sí, para lograr especificar un producto como una camisa, color azul, talla XL, es necesario tomar estas tablas como importantes para el modelo que se formuló, de esta manera conservamos este principio que es mayoría.

En todos los e-commerce y proveedores se presenta la clasificación por categoría, pero había excepción con las subcategorías, no todos integran esta característica, pero aun teniendo esto en cuenta, se generó un modelo que presentara la opción más amplia para cubrir los posibles casos, por esta razón se implementó en el modelo las categorías y subcategorías de los productos.

Figura 11

Módulo de gestión de inventarios

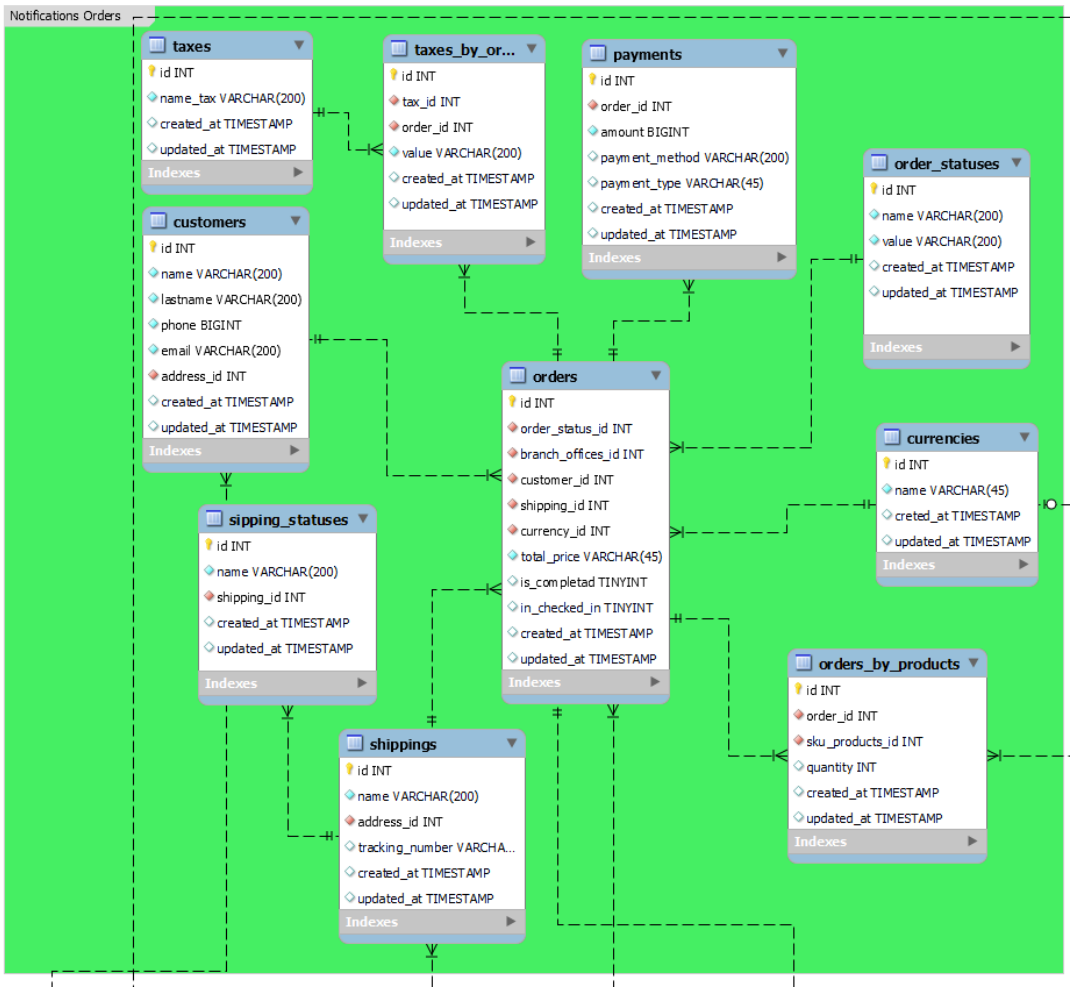


Nota: Esta tabla de inventario es una M:N cumple la labor de relacionar una variación de producto (SKU) con su respectivo deposito (warehouse), el objetivo es tener control de los

productos disponibles. Esta tabla permite identificar el lugar donde se almacenan los productos y si es el caso identificar la ubicación respectiva.

Figura 12

Módulo de órdenes de envío



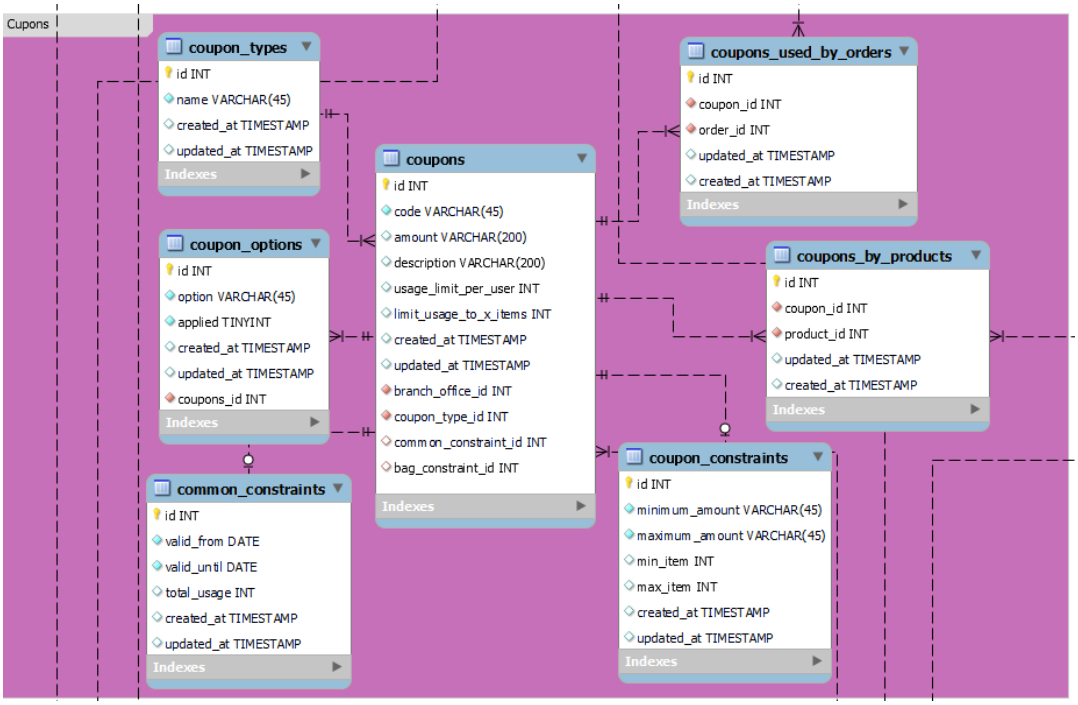
Nota: Las ordenes son una de las tablas más importantes en los e-commerce debido que estas llevan la información del comercio de los productos, de esta manera resulta necesario hacer la recepción de los datos y envíos desde la plataforma, el fin de esta tabla es poder tener reportes de las ventas, junto con todos los datos asociados a la venta.

Debido a que se manejan varios tax en la en una orden y todos son muy diversos, se decidió que los tax se añadieran a los órdenes por medio de una tabla M:N en donde la tabla contiene el valor que tiene cada tax, mientras en la tabla taxes se almacena el tipo de tax.

Las tablas almacenan la información del estado, destino, información del comprador, pagos, impuestos, etc. Principalmente se tiene el tipo de envío y la dirección a donde se dirige.

Figura 13

Módulo de gestión de cupones y descuentos



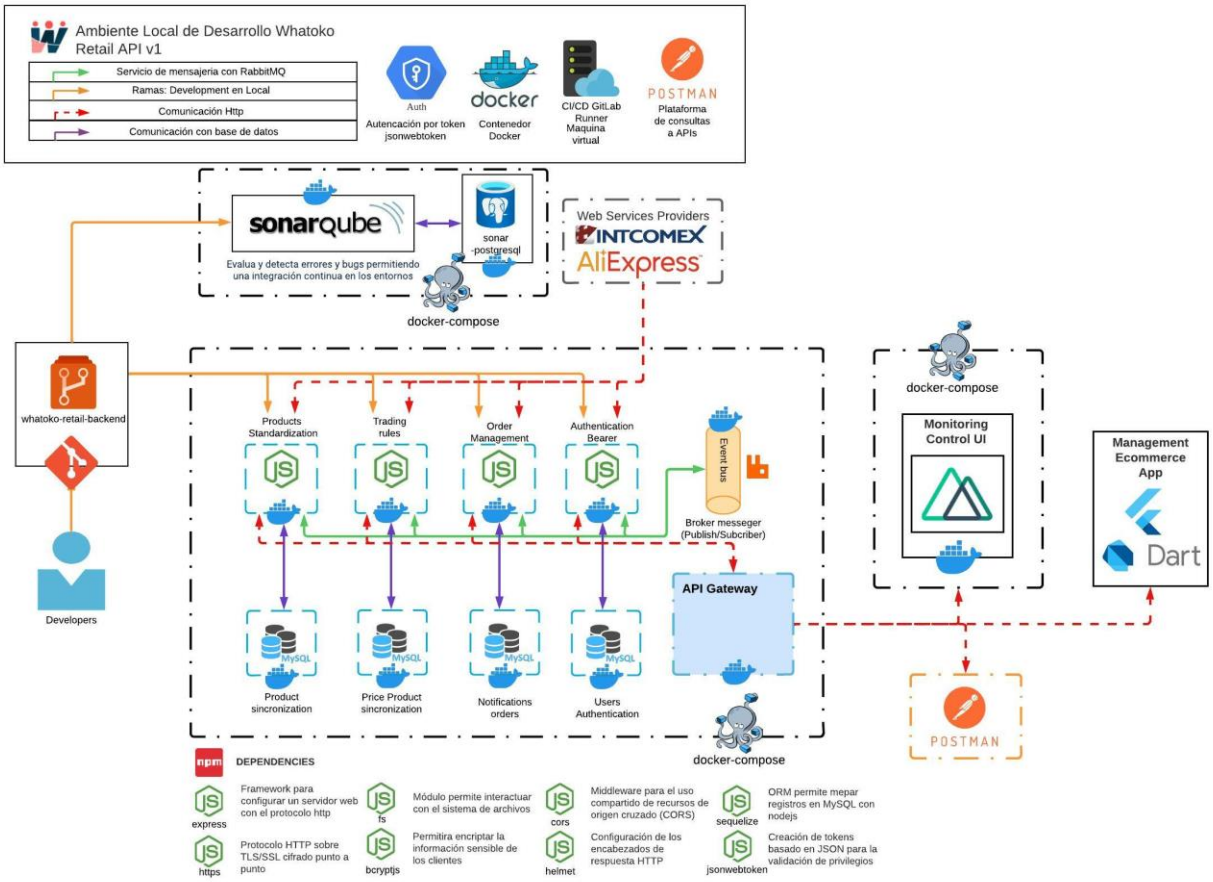
Nota: Esta tabla contiene los datos correspondientes a los cupones, estos cupones se usan en las órdenes y aplican para varios productos por lo que los cupones están relacionados con las órdenes y con los productos.

4.3 Diseño

En este punto se diagrama la arquitectura del proyecto, definiendo los microservicios a desarrollar, dado la complejidad de la sincronización de los datos en la base de datos, mas adelante, se tratará el microservicio de productos

Figura 14

Diagrama de arquitectura entorno local

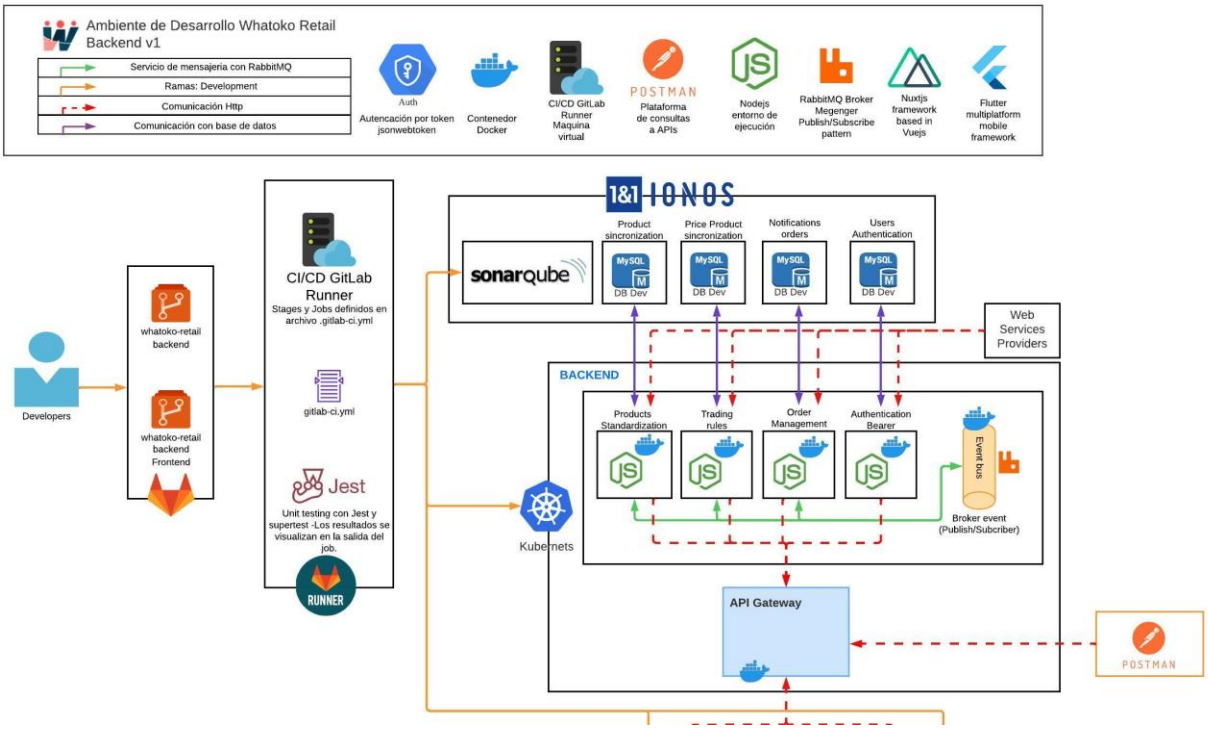


Note: Los microservicios definidos para dividir las funcionalidades del API fueron: el microservicio de estandarización de los productos de los proveedores, en el que se homologaran los datos suministrados en los modelos requeridos para generar las respuestas del API, por otro

lado se creara un microservicio de precios o reglas comerciales, el cual tiene como objetivo redefinir los precios de los productos según las reglas definidas en el microservicio, las cuales serán costos de envió, porcentajes de ganancias entre otros factores, también poder hacer consultas periódicamente para validar el precio real de los productos. El microservicio gestor de ordenes cubrirá la parte de generar pedidos y envíos de los productos los cuales se tendrán que hacer en función al servicio que tenga definido el proveedor validando los datos suministrados, en este microservicio se tendrán que crear funcionalidades para cada proveedor los datos suministrados así por el cliente así como la respuesta suministrada por el proveedor serán almacenadas en MySQL, por último se contará con un microservicio de autenticación que valide las credenciales y roles de los usuarios almacenadas en el Core de Whatoko retail.

Figura 15

Diagrama de arquitectura entorno de desarrollo



Nota: El diagrama de arquitectura en un entorno de desarrollo, testeo y UAT, cuenta con un bus de eventos que se usará para mantener la lógica de la base de datos conectada a los servicios, se usará Jest, con Supertest para realizar los testeos en lo pipe runners de GitLab, se tendrá a Kubernet como orquestador de los servicios definidos en el backend, se estará validando las practicas usadas en el código y los posibles bugs con SonarQube, para realizar entregas continuas en el backend.

4.3.1 Definición de las tecnologías a usar en el proyecto

4.3.1.1 Dependencias

Bcryptjs: Es una utilidad de criptografica hash que transforma cualquier dato entrante en una serie de caracteres de lonGitud fija.

Cors: Intercambio de recursos de origen cruzado (CORS) es un mecanismo basado en el encabezado HTTP que permite que un servidor indique cualquier origen (dominio, esquema o puerto) que no sea el suyo propio desde el cual un navegador debería permitir la carga de recursos.4.4 Codificación

Helmet: es un paquete de Node.js que ayuda a proteger su servidor de algunas vulnerabilidades web conocidas al configurar los encabezados de respuesta HTTP de manera adecuada. Viene con una colección de varias funciones de middleware que configuran los encabezados de seguridad que se devuelven desde la aplicación Express.

Sequelize: es un ORM moderno de TypeScript y Node.js para Oracle, Postgres, MySQL, MariaDB, SQLite y SQL Server, y más. Con un sólido soporte de transacciones, relaciones, carga ansiosa y diferida, replicación de lectura y más.

Jsonwebtoken: JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un

objeto JSON. Los JWT se pueden firmar usando un secreto (con el algoritmo HMAC) o un par de claves pública/privada usando RSA o ECDSA.

Express: Express es un marco de aplicación web de Node.js mínimo y flexible que proporciona un conjunto sólido de funciones para aplicaciones web y móviles.

4.3.1.2 Dependencias de unit testing

Jest: es un framework de prueba que ayuda a automatizar las pruebas unitarias.

4.3.1.3 Herramientas complementarias

Postman: Plataforma de API para diseñar, construir, probar e iterar las API.

Git: Software de control de versiones

GitLab: Servicio para el control de versiones y DevOps, CI/CD

SonarQube: es un software para evaluar el código fuente y ayudar a detectar los code smells antes de cualquier merge request.

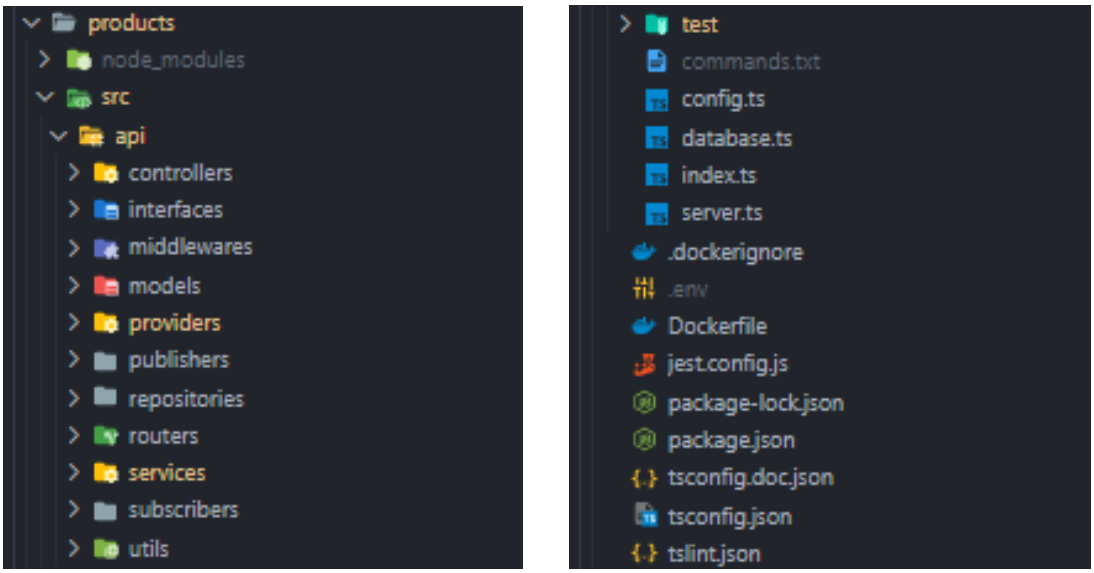
Docker: proporciona una manera estándar de ejecutar código. Docker es un OS para contenedores, similar a una mv-virtualizada, los contenedores virtualizan el sistema operativo de un servidor.

4.4 Alistamiento del proyecto

Estructura del proyecto: Se genero una estructura que escalable siguiendo el patrón de diseño MVC, con una separación de la lógica de conexión a la base de datos y funcionalidades del API, siguiendo una estructura como se plantea a continuación

Figura 16

Estructura del microservicio products-service



Nota: la estructura del proyecto mostrada en las imágenes sigue un patrón de diseño modelo, vista, controlador, con una capa de datos, la cual define las peticiones a realizar en la base de datos con el ORM de sequelize, separando la lógica del dominio de la lógica de la aplicación la cual se define en un módulo de servicios los cuales se usarán para mantener conectados los demás microservicios, con el patrón de publish-subscriber, y el bróker de mensajería RabbitMQ, como se ve en los directorios de publishers y subscribers los cuales contienen la lógica de conexión a las queues y exchanges del bróker.

- Instalar las dependencias: Las dependencias que se usaran en el proyecto son: express, https, fs, bcryptjs, helmet, cors, sequelize, jsonwebtoken, jest, supertest, axios, dotenv, morngan, TypeScript, para este último se debe de configurar los archivos tsconfig, tslint, y gitignore.

- Crear el dockerfile: Se debe de generar el docker file para ejecutar el microservicio que se va a estar desarrollando
- Dockerizar proyecto local: Una vez definido el o los servicios a desplegar se tiene que crear el docker compose con todos los servicios a desplegar, estos incluyen la base de datos y gestores de bases de datos si son necesarios.
- Definir las variables de entorno: Se tiene que definir las variables de entorno del proyecto, tanto las locales como las de desarrollo, se debe de crear los archivos de las variables con la indicación del entorno a usar, con su comentario para cada variable.
- Conexión al repositorio: Una vez configurado la estructura del proyecto se hará el primer commit al repositorio, con la configuración inicial.
- Desplegar el SonarQube localmente: Se desplegará el SonarQube con el repositorio ya configurado para empezar con las validaciones de los bugs o errores
- Definir la documentación del proyecto: Se debe definir donde se desplegará la documentación del proyecto, por lo que los métodos pueden cambiar, se tiene la opcion de usar tsdoc con typedoc para desplegar con GitLab o usar directamente readthedocs
- Configurar el servidor con express y Typescript, headers, cors, methods, middlewares, routes.
- Conexión a la base de datos: Se debe de realizar la conexión a la base de datos con sequelize. y validar la conexión
- Validar conexión a la base de datos: Se debe de definir una prueba unitaria para validar la conexión a la base de datos, para eso, se hará una consulta a una de las tablas y validar el contenido

- Configurar cors: Se tiene que definir las propiedades del cors para el entorno de desarrollo y local
- Definir los headers: Se definirán los encabezados con hermet el cual tiene una preconfiguración de seguridad que ayuda a proteger de vulnerabilidades de los encabezados en un servidor express
- Configurar Morgan: Usar morgan en un entorno local para poder visualizar a las peticiones que se ejecutan en el servidor
- Condicionar las variables de entorno: Para ejecutar como es debido los entornos se hará un filtro de las variables de entorno que se deben de ejecutar para tener estandarizado las variables de esta manera se filtraran según el entorno que se ejecute

4.5. Desarrollo

En este apartado se estará explicando el desarrollo del proyecto, mostrando, como se desarrolla la lógica del servicio de productos que se encarga de la gestión de los productos en la plataforma de Whatoko retail.

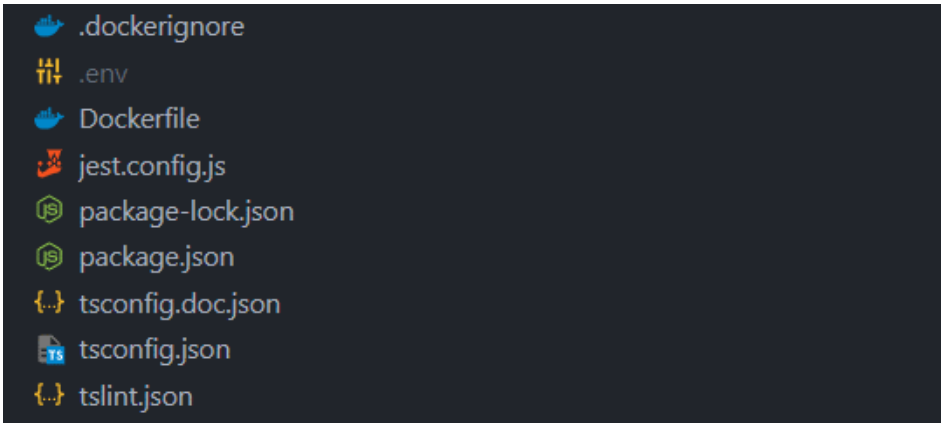
4.5.1 Configuración inicial de API REST node y express junto con TypeScript

Para la realización del proyecto se configuro la base del proyecto API con node y TypeScript, en el que es necesario que sigamos unos pasos para poder iniciar un proyecto y posteriormente dockerizarlo. Para iniciar un API REST con Node.js y Express usando TypeScript usando la herramienta de línea de comandos “tsc”, la cual debe ser instalada de manera global previamente con npx install -g tsc, al momento de ejecuta “tsc” se crea el archivo de configuración tsconfig.json para que el proyecto pueda validar los tipos de datos entre otra validaciones y opciones que nos ofrece TypeScript, se configuran las dependencias usando los módulos que validan el funcionamiento con TypeScript @types.

Para la realización del API fue necesario dockerizar el proyecto con Docker, por lo que primero se configuro la imagen del proyecto la cual contiene todos los comandos de ejecución y configuración inicial del contenedor. Estas imágenes se van a poder ejecutar desde la línea de comando siguiendo los comandos de Docker, pero la opción para el desarrollo se basa en la configuración de ejecución de una serie de servicios necesarios para el correcto funcionamiento del todos lo microservicios que se vayan a ejecutar en el desarrollo. Los servicios necesarios para el desarrollo del proyecto fueron el contenedor del proyecto que se desarrolló, un contenedor con un servidor de MySQL una interfaz para la administración de la base de datos PhpMyAdmin, y el bróker de mensajería RabbitMQ.

Figura 17

Configuración inicial de un proyecto de node con TypeScript dockerizado



Note: En esta imagen se evidencia los archivos de configuración para el desarrollo de un proyecto node con TypeScript y docker, los archivos package.json contienen toda la infomación del proyecto node y las dependencias que se usan, los archivos de configuracion de TypeScript y de jest, y por ultimo la dockerizacion del proyecto

Figura 18

Configuracion inicial del proyecto API REST con express

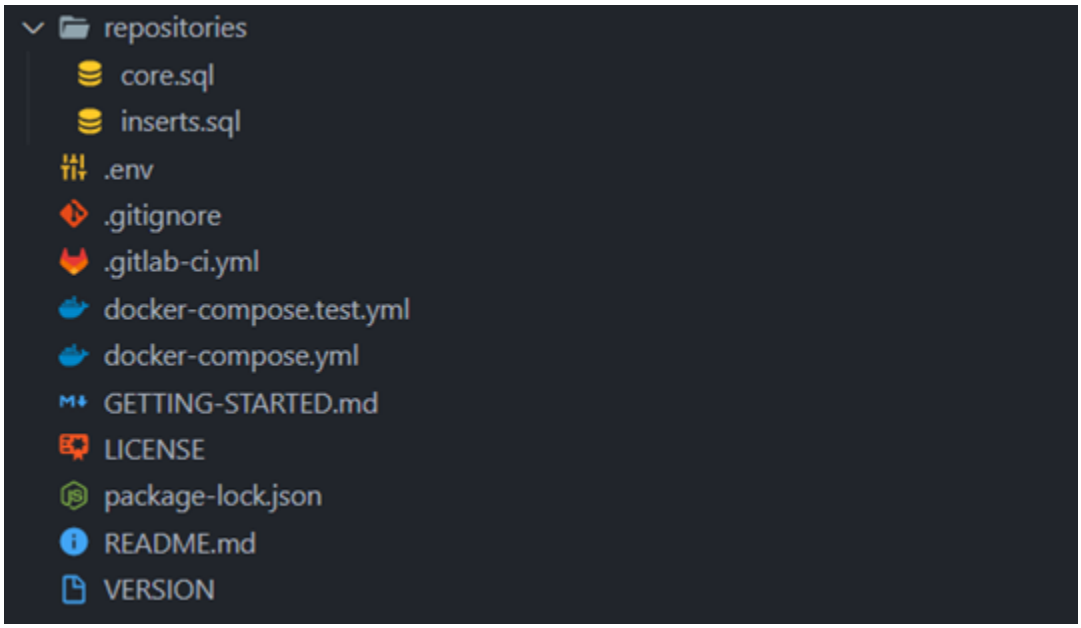
El inicio del proyecto contra de la configuración de la conexión a la base de datos con el ORM de sequelize, un archivo de configuración que se encarga de recibir las varibles de entorno y estructuralas en un objeto json, el cual se configura dependiendo del entorno en el que se ejecute el proyecto, esta variables de entorno son las creadenciales de conexión a la base de datos en MySQL, al broker de RabbitMQ, la clave de cifrado del token de autenticación, puerto de ejecución y configuracion DDoS.

En el archivo server se configuran las cabeceras y cors del servicidor de express, se asignan los middleware para hacer la respuestas del servicidor mas entendible, se usa hetmet como middleware de seguridad configurando las cabeceras y desactivando configuraciones predeterminadas del servidor de express.

El archivo init se encarga de inicializar las configuraciones previamente definidas y ejecutar el servidor de express.

Figura 19

Archivos de ejecución de los microservicios



Nota: Como se muestra en la imagen se tienen los archivos de ejecución de los contenedores del proyecto en el Docker Compose, el cual contiene toda la lógica de ejecución de esto contenedores, asociados a una misma red, vinculados si es necesario y corriendo al mismo tiempo, en el directorio repositorio se tienen los recursos para la creación e inserción de datos de prueba en la base de datos que se usa en los microservicios. Se define la ejecución para la integración continua con el repositorio montado en GitLab.

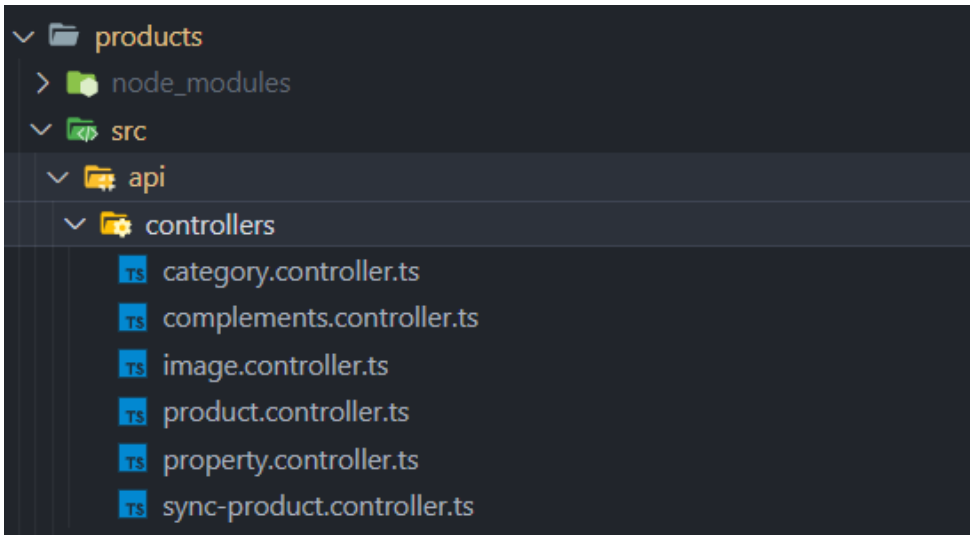
4.5.2 Desarrollo del microservicio de productos

El microservicio de productos se encarga de administrar los productos vinculados a una sucursal, este servicio busca sincronizar productos asociados a e-commerces y proveedores y almacenarlos en la base de datos definida anteriormente, este servicio está compuesto por una serie de endpoints

que se encargan de gestionar y administrar los datos asociados a una sucursal. Este microservicio hace parte del conjunto de microservicios para el proyecto de dropshipping.

Figura 20

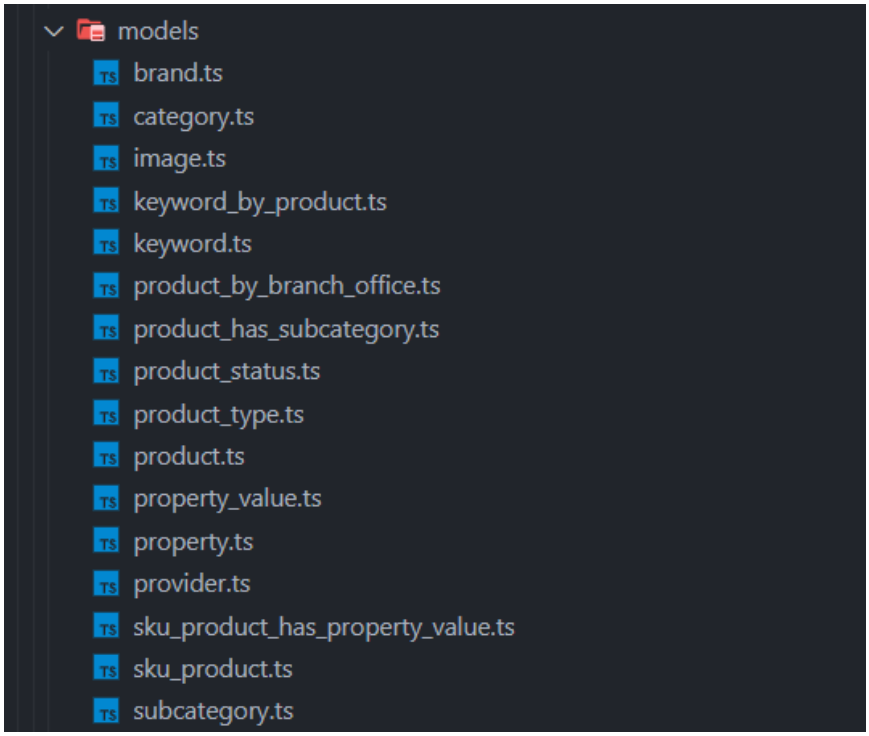
Controladores del microservicio de productos



Nota: Los controladores están definidos según los servicios del API definidos, esto quiere decir que existen seis servicios para la gestión del API que acopla la lógica de la base de datos en estas funciones

Figura 21

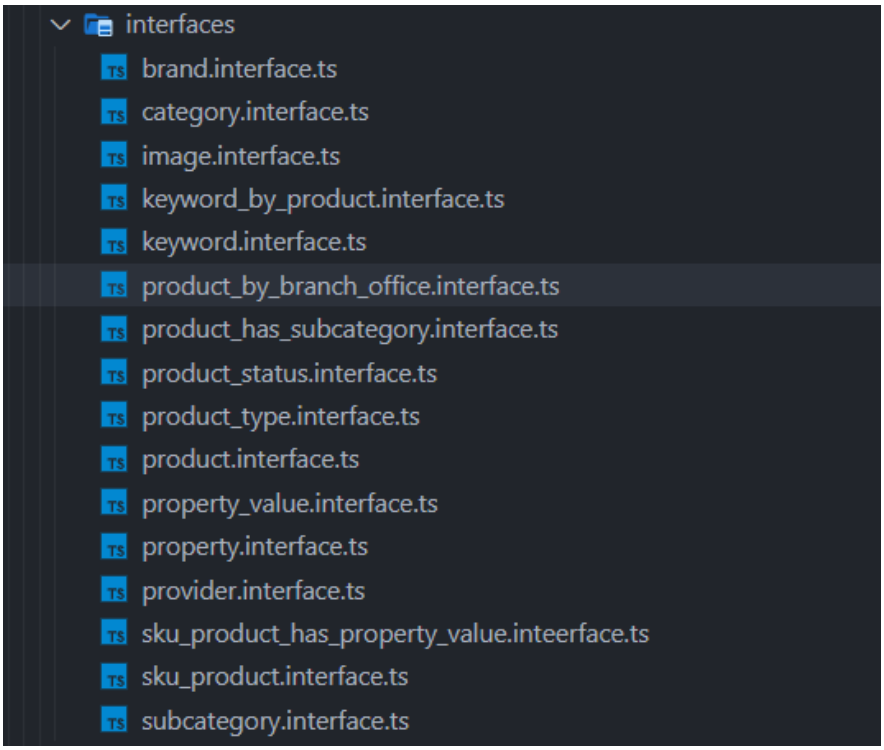
Modelos de Sequelize-TypeScript referentes a la base de datos



Nota: En la imagen se puede ver la definición de los archivos correspondiente a los modelos que se encuentran en la base de datos, estos modelos se conectan a la base de datos que se crea en el contenedor de mysql y podemos matipular las consultas y datos con el ORM de sequelize, por otro lado, el proyecto está desarrollado en TypeScript por lo que el módulo a usar para la definición de los modelos cambia, y el uso de anotaciones esta presenta en la librería sequelize-TypeScript

Figura 22

Interfaces de los modelos de la base de datos



Nota: Estructura de las interfaces de los modelos definidos con sequelize.

Las interfaces en los modelos Sequelize se utilizan para definir una estructura de tipos personalizada que se utiliza en la definición del modelo. Una interfaz en Sequelize es una forma de especificar los tipos de datos que se utilizarán en las propiedades del modelo.

Las interfaces permiten definir y reutilizar tipos de datos personalizados para las propiedades del modelo, lo que hace que sea más fácil y eficiente definir modelos complejos. Las interfaces también pueden ser utilizadas para asegurar que los modelos se ajusten a un conjunto de requisitos predefinidos y para garantizar que los modelos sean consistentes y estén libres de errores.

Además, las interfaces también pueden ser utilizadas para definir relaciones entre modelos, lo que facilita el acceso a los datos relacionados en la base de datos. En resumen, las interfaces en los modelos Sequelize se utilizan para: definir una estructura de tipos personalizada para las propiedades del modelo, asegurar que los modelos se ajusten a un conjunto de requisitos predefinidos, garantizar la consistencia y la calidad de los datos en el modelo, definir relaciones entre modelos y facilitar el acceso a los datos relacionados en la base de datos.

4.5.2.1 Middleware

Un middleware de autenticación es una función que se ejecuta antes de que se maneje una petición HTTP y que verifica si el usuario que hace la petición está autenticado o no. Si el usuario está autenticado, se permite que la petición siga adelante y se maneje, de lo contrario, se devuelve una respuesta de error.

Los middlewares de autenticación se utilizan comúnmente en aplicaciones web que requieren que los usuarios inicien sesión para acceder a ciertas rutas o recursos. Estos middlewares son útiles para proteger las rutas de la aplicación y asegurarse de que sólo los usuarios autorizados puedan acceder a ellas.

En términos generales, un middleware de autenticación puede ser implementado de la siguiente manera:

Verificar si el usuario está autenticado; El primer paso es verificar si el usuario está autenticado o no. Esto se puede hacer de varias maneras, por ejemplo, verificando si existe una sesión válida o si se proporciona un token de autenticación válido en la cabecera de la petición.

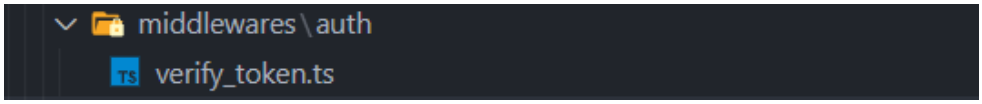
Si el usuario no está autenticado, redirigir a la página de inicio de sesión; Si el usuario no está autenticado, se debe redirigir a la página de inicio de sesión o mostrar un mensaje de error. En una aplicación web, esto se puede hacer mediante una redirección a la ruta de inicio de sesión.

Si el usuario está autenticado, continuar con la petición; Si el usuario está autenticado, se permite que la petición siga adelante y se maneje. En una aplicación web, esto puede implicar permitir que el usuario acceda a una ruta protegida o mostrarle un recurso Restringido.

En resumen, un middleware de autenticación se utiliza para proteger las rutas de una aplicación web y asegurarse de que sólo los usuarios autorizados puedan acceder a ellas. Los middlewares de autenticación son útiles en aplicaciones web que requieren que los usuarios inicien sesión para acceder a ciertas rutas o recursos.

Figura 23

Middleware de autenticación de token de sucursal



Nota: Middlware de autenticación con Jsonwebtoken, se encarga de validar las creadenciales asociadas a una sucursal, si la sucursal no se encuentra registrada, lanzara un error de autenticación.

4.5.2.2 Proveedor intcomex

Intcomex es un distribuidor principal de valor agregado de productos de TI enfocado únicamente en servir a América Latina y el Caribe. Distribuye equipos informáticos, componentes, periféricos, software, sistemas informáticos, accesorios, productos de red y electrónica de consumo digital a más de 40.000 clientes locales en más de 45 países. Ofrece compras de una sola fuente a sus clientes al proporcionar una selección en stock de más de 5700 productos de más de 220 proveedores, incluidos muchos de los principales fabricantes de productos de TI del mundo.

Atiende los mercados de productos de TI de América Latina y el Caribe utilizando un modelo de distribución dual. Opera tanto como un agregador mayorista con sede en Miami como un distribuidor en el país.

4.5.2.3 Intcomex web services

Los Servicios Web de Intcomex, o IWS para abreviar, es un conjunto de servicios web que permiten a los clientes interactuar con Intcomex para realizar operaciones como obtener información sobre productos o realizar pedidos mediante programación.

IWS utiliza la transferencia de estado representacional (REST) para exponer su funcionalidad. Los clientes utilizan los servicios web a través de solicitudes HTTP utilizando una URL que especifica el método de operación y los parámetros necesarios para obtener la información deseada.

La respuesta HTTP de estas solicitudes se devuelve mediante JSON (de forma predeterminada) o respuestas con formato XML que contienen los resultados de la llamada solicitada.

IWS se puede integrar en la tienda web, la aplicación móvil o cualquier otro punto de venta de un cliente para proporcionar una forma para que los usuarios finales realicen pedidos directamente a Intcomex.

4.5.2.4 Consumir APIs de terceros a través de Axios

Axios es una biblioteca de JavaScript para realizar solicitudes HTTP desde el navegador o desde Node.js. Permite enviar y recibir datos a través de los protocolos HTTP y HTTPS de manera sencilla y eficiente.

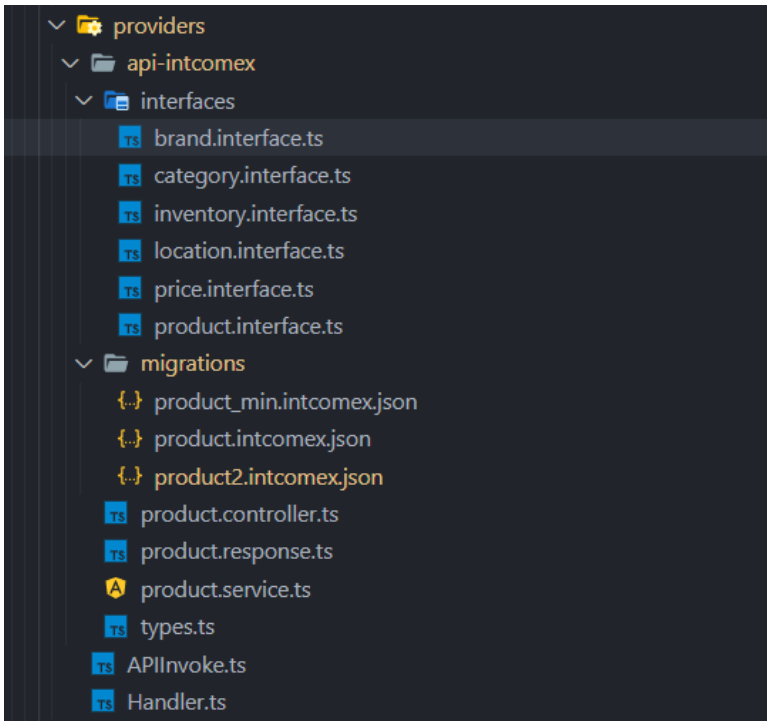
Las ventajas de Axios con respecto a las bibliotecas estándar HTTP nativas de node.js para llamadas API son las siguientes:

- Intercepción de solicitudes y respuestas
- Manejo de errores simplificado
- Protección contra XSRF
- Soporte para el progreso de carga
- Tiempo de espera de respuesta
- La posibilidad de cancelar solicitudes
- Soporte para navegadores modernos y antiguos
- Transformación automática de datos JSON

Axios es muy popular en el desarrollo de aplicaciones web y móviles, ya que simplifica la interacción con APIs y servicios web, y es compatible tanto con navegadores web como con Node.js.

Figura 24

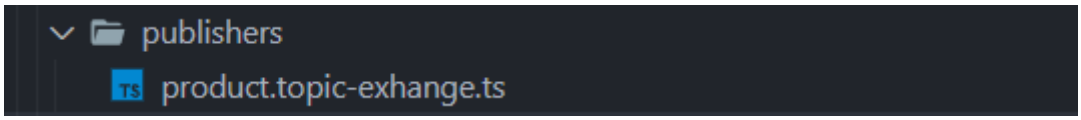
Proveedores de productos asociados a Whatoko retail



Nota: En la imagen se muestra cómo se estructuro la lógica para consumir el API de intcomex y estandarizar los productos, lo que

Figura 25

Publisher cola de mensajería product topic exchange



Nota: Un "topic exchange" es un tipo de "exchange" (intercambio) en RabbitMQ, que permite la publicación de mensajes utilizando un patrón de enrutamiento basado en "topics" (temas), como agregar productos a las ordenes generadas en los ecomerces o agregar cupones a ciertos productos antes de ser vinculados con los e-commerce. De esta manera siempre los productos van a estar sincronizados con los demás microservicios de la aplicación.

Cuando se publica un mensaje en un "topic exchange", el mensaje se envía a todas las colas que han sido vinculadas a ese exchange con un "binding key" (clave de enlace) que coincide con el patrón de enrutamiento del mensaje. El patrón de enrutamiento se especifica como una cadena de caracteres separados por puntos, por ejemplo: "foo.bar.baz".

Los "binding keys" de las colas pueden contener caracteres especiales para indicar comodines en el patrón de enrutamiento. Los dos tipos de comodines que se pueden utilizar son:

* (asterisco): Se puede utilizar en un binding key para reemplazar exactamente un solo término en el patrón de enrutamiento. Por ejemplo, la clave "foo.*.baz" coincidirá con los patrones "foo.bar.baz" y "foo.qux.baz", pero no con "foo.bar.qux.baz".

(almohadilla): Se puede utilizar en un binding key para reemplazar cero o más términos en el patrón de enrutamiento. Por ejemplo, la clave "foo.#" coincidirá con cualquier patrón que comience con "foo.", como "foo.bar.baz", "foo.qux.baz", "foo.bar.qux.baz", y así sucesivamente.

El uso de un "topic exchange" es útil para implementar patrones de suscripción flexibles en RabbitMQ, ya que los consumidores pueden suscribirse a los mensajes que les interesan utilizando "binding keys" específicos. Por ejemplo, un consumidor que se suscribe a la clave "foo.*.baz" solo recibirá los mensajes que coincidan con ese patrón de enrutamiento.

4.5.2.5 Repositorios en sequelize TypeScript

Definir repositorios en una aplicación que utiliza Sequelize es una técnica común para encapsular la lógica de acceso a la base de datos y mantenerla separada de la lógica de negocio de la aplicación. En lugar de trabajar directamente con los modelos de Sequelize en todas partes de la aplicación, se puede crear una capa adicional de abstracción que actúa como intermediario entre los modelos y el resto de la aplicación.

Hay varias ventajas de definir repositorios con Sequelize, entre ellas:

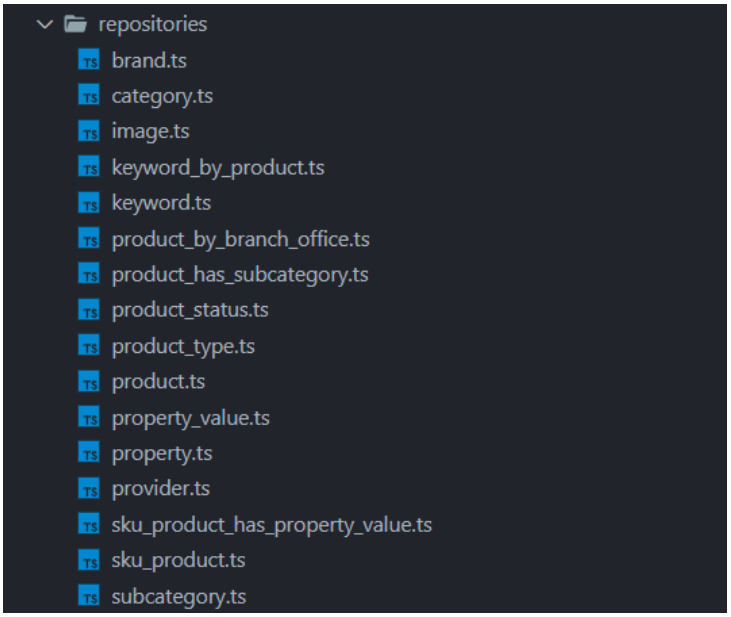
- Abstracción: los repositorios proporcionan una capa de abstracción sobre los modelos de Sequelize, lo que significa que el resto de la aplicación no necesita conocer los detalles de cómo se implementa la base de datos. Esto hace que la aplicación sea más modular y fácil de mantener.
- Flexibilidad: los repositorios permiten definir y controlar cómo se realizan las consultas a la base de datos. Esto es particularmente útil cuando se trabaja con modelos complejos o se necesita realizar operaciones de agregación y filtrado de datos.
- Reutilización: los repositorios pueden ser reutilizados en diferentes partes de la aplicación, lo que reduce la duplicación de código y facilita el mantenimiento.

- Testing: los repositorios son fáciles de probar, ya que se pueden crear pruebas unitarias para cada método del repositorio sin tener que preocuparse por el estado de la base de datos.

En resumen, definir repositorios con Sequelize permite una mayor abstracción y flexibilidad en el acceso a la base de datos, lo que hace que la aplicación sea más modular, reutilizable y fácil de mantener. Además, los repositorios son fáciles de probar y permiten un mayor control sobre las consultas que se realizan a la base de datos.

Figura 26

Repositorios de los modelos creados con Sequelize



Nota: Los repositorios se encargan de crear los métodos necesarios con la lógica para traer los datos de la base de datos de manera ordenada, y filtrando las consultas generales. En este caso lo repositorios son la conexión directa a la base de datos que trae la información necesaria siempre con los métodos ya definidos

4.5.2.6 Routas de express

En una aplicación web que utiliza Express como framework, las rutas son los puntos de entrada a la API que permiten a los clientes enviar solicitudes HTTP y recibir respuestas. Las rutas se definen mediante métodos de Express que corresponden a los verbos HTTP, como GET, POST, PUT y DELETE, entre otros.

Para crear una ruta en Express, se debe usar el objeto Router de Express para definir una instancia de enrutador que se pueda utilizar para declarar las rutas. A continuación, se muestra un ejemplo básico de cómo manejar una ruta GET "/categories":

```
import { Auth } from "../middlewares/auth/verify_token";
import { Router } from "express";
import { CategoryController } from "../controllers/category.controller";

const routes = Router();
routes.get("/categories", Auth.verifyToken, CategoryController.getCategories);
export default routes;
```

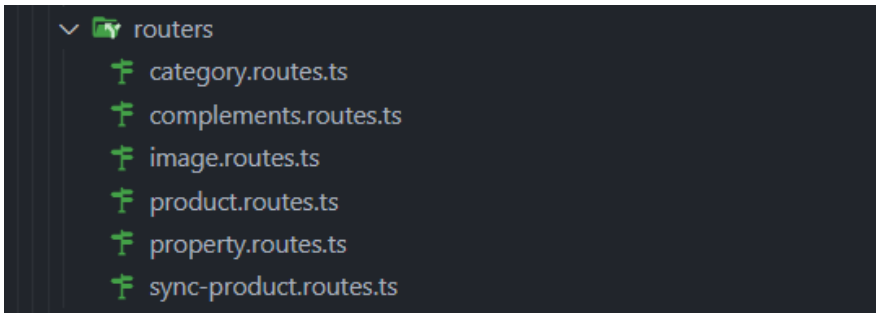
La ruta se registra en la aplicación utilizando el método use de Express, que se utiliza para montar middleware y enrutadores. En este caso, se monta el enrutador en la ruta "/categories", lo que significa que todas las solicitudes que lleguen a la raíz de la aplicación serán manejadas por el enrutador.

```
 /**
   * Método encargado de inicializar las rutas
   */
  public routes(): void {
    this.app.use(`${this.routePrefix}`, RoutesProduct);
    this.app.use(`${this.routePrefix}`, RoutesCategory);
    this.app.use(`${this.routePrefix}`, RoutesSyncProducts);
    this.app.use(`${this.routePrefix}`, RoutesProperty);
    this.app.use(`${this.routePrefix}`, RoutesComplement);
    this.app.use(`${this.routePrefix}`, RoutesImage);
  }
```

Este es solo un ejemplo básico, y las rutas pueden ser mucho más complejas dependiendo de las necesidades de la aplicación. Sin embargo, el proceso básico para definir una ruta en Express es el mismo: crear una instancia de enrutador, definir la ruta utilizando un método HTTP y una función de devolución de llamada, y registrar el enrutador en la aplicación utilizando el método use.

Figura 27

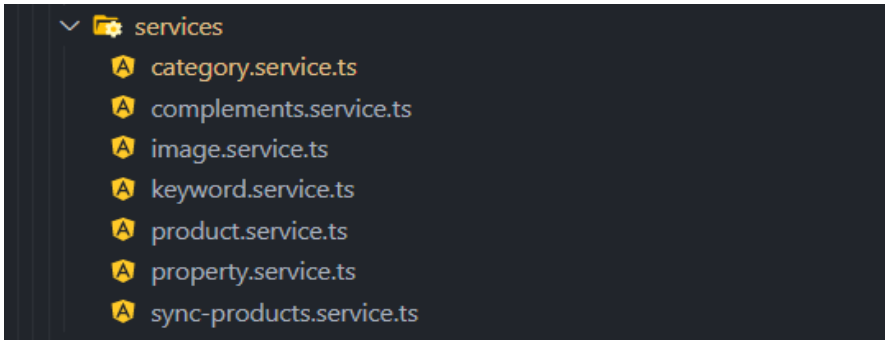
Rutas de conexión al API



Nota: Se definen las rutas de conexión al api, que permite la conexión con los diferentes métodos definidos en los servicios del API

Figura 28

Servicios de gestión y administración de productos



Nota: Los servicios que se ven en la imagen se encarga de la gestión de los productos asociados desde un e-commerce como los productos sincronizados desde un proveedor

El servicio de categoría se encarga de agregar, consultar, actualizar y eliminar registros y relaciones de los modelos categoría y subcategoría, con producto, siempre procurando que las consultas estén bien enfocadas hacia una sucursal.

Los servicios complementarios hacen referencia a modelos asociados a productos que ayudan a complementar el producto en sí, como lo puede ser la marca de un producto, el proveedor, el estado y el tipo de producto, estos modelos se encargan de filtrar los productos y agruparlos en grupo más grandes que ayudan a filtrar los productos asociados a una sucursal.

Los servicios de imagen se encargan de agregar y eliminar imanes a productos ya vinculados a una sucursal

El servicio de keyword agrega y elimina palabras claves asociadas a un producto o a una sucursal, se encarga de recibir las palabras claves de los proveedores que sirva para poder filtrar los productos según la clasificación de proveedor.

El servicio de propiedades corresponde a la gestión de los productos unitarios o variantes de un producto definido por una propiedad, como puede ser el color, la talla, el material o cualquier otra propiedad que afecte el producto, de esta manera se controlan todas las variantes asociadas a un producto y se pueden configurar de manera independiente

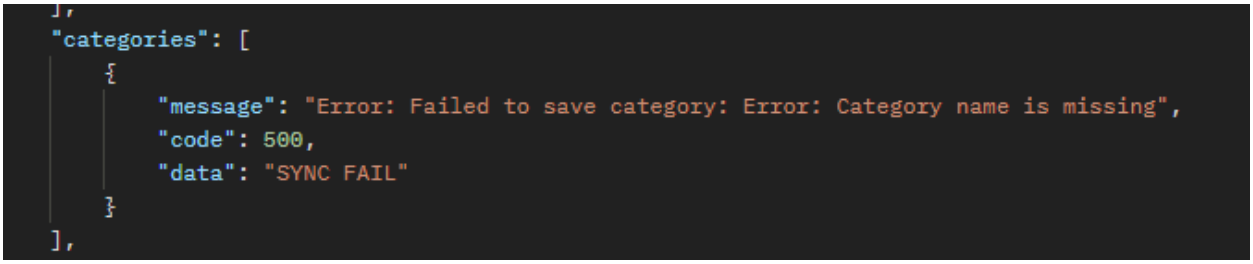
El servicio de productos precisamente se encarga de la gestión de los productos y sus variantes, agregando nuevas configuraciones en las propiedades o imágenes asociadas a la variante.

El servicio de sincronización, este servicio se podría catalogar como el más importante, ya que permite la sincronización de los productos consultados en el API de intcomex y sincroniza los

datos con los modelos de Whatoko Retail. Este proceso se hace de manera independiente y si llega a suceder algún error el error se trata de manera independiente en el mimo modelo, sin perjudicar toda la sincronización del producto.

Figura 29

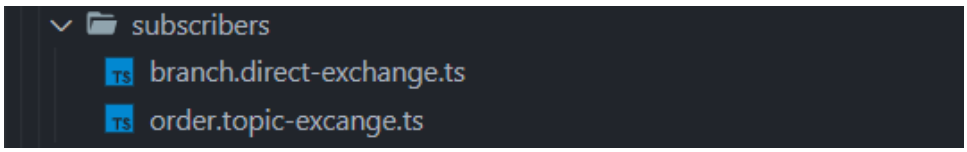
Gestión de errores en la sincronización de productos



Nota: Como se ve en la imagen si algún apartado de la sincronización llega a salir mal, el API se encarga de devolver un mensaje de error en el atributo que no se pudo sincronizar, con el mensaje de error por el cual fallo la sincronización.

Figura 30

Subscriber Branch derect Exchange y Order topic exchange

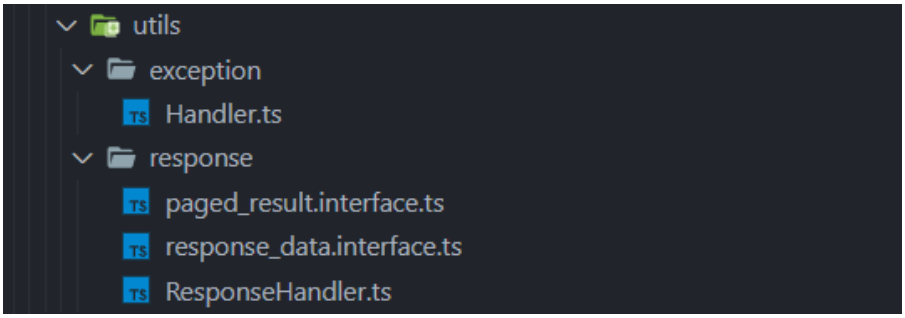


Un "Direct Exchange" en RabbitMQ es un tipo de intercambio que enruta mensajes a colas basadas en la coincidencia exacta de la clave de enrutamiento. En un "Subscriber Branch Direct Exchange", los suscriptores se unen a una cola específica en el intercambio directo y reciben los mensajes enviados a esa cola.

Por otro lado, un "Topic Exchange" en RabbitMQ es un tipo de intercambio que enruta mensajes a colas basadas en patrones de clave de enrutamiento que utilizan caracteres comodines (* y #). En un "Order Topic Exchange", los mensajes se envían a una cola específica basada en el patrón de clave de enrutamiento que coincida con el patrón de la clave de enrutamiento del mensaje.

Figura 31

Utils manejo de exceptiones y respuestas del API



En el manejo de las excepciones se tiene un método warningHandler que se utiliza para manejar rutas no encontradas en la aplicación. Toma tres parámetros: log, que es el objeto de error o excepción que se ha producido, data, que es cualquier otro dato relevante que se deba incluir en el mensaje de error, y saved, que indica si se debe guardar la excepción en un archivo de registro. El método devuelve un objeto JSON que contiene el mensaje de error y cualquier otro dato relevante.

la clase ResponseHandler, que se utiliza para enviar respuestas HTTP estandarizadas desde una API de Express. La clase tiene dos métodos estáticos: response y paginateResult.

El método response se utiliza para enviar una respuesta HTTP a una solicitud realizada a la API. Toma dos parámetros: res, que es el objeto de respuesta de Express, y data, que es un objeto que contiene la información que se enviará como respuesta. El método establece el código de

estado de la respuesta según el valor de data.statusCode, y luego envía un objeto JSON que contiene los siguientes campos:

- success: un booleano que indica si la solicitud se ha completado con éxito.

error: un objeto que contiene información detallada sobre cualquier error que se haya producido.

- data: los datos que se envían como respuesta, que pueden ser una matriz paginada si data.limit y data.offset están definidos y data.data es un array.
- message: un mensaje de texto opcional que describe el resultado de la solicitud.
- limit y offset: los parámetros de paginación, si se han utilizado.

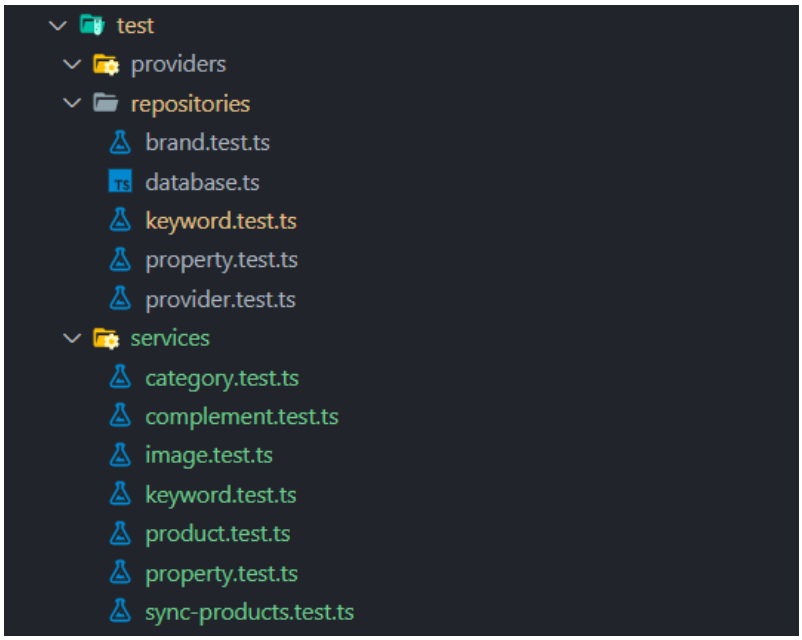
El método paginateResult se utiliza para paginar una matriz de datos. Toma tres parámetros: limit, que es el número máximo de elementos por página, offset, que es el índice del primer elemento de la página actual, y data, que es la matriz de datos que se va a paginar. El método devuelve un objeto que contiene la página actual, el número total de páginas, el número total de elementos y la matriz de datos paginados.

4.5.2.7 Realizar pruebas unitarias con Jest

En este directorio se definieron las pruebas con Jest para los servicios del API, Jest es un framework de pruebas que se utiliza comúnmente para realizar pruebas de unidad en aplicaciones Node.js. Con Jest, podemos escribir pruebas que aseguren que las funciones y módulos funcionen como se espera.

Figura 32

Directorio de pruebas a las funcionalidades del API



Nota: En la imagen se puede ver como se realizan test a los servicios previamente mencionados y algunos repositorios, donde se valida el funcionamiento mockeando las respuestas de la base de datos.

Para comenzar a realizar pruebas en Node.js con Jest, podemos seguir los siguientes pasos:

- Instalar Jest: podemos instalar Jest utilizando npm en nuestra terminal con el siguiente comando: $npm install jest --save-dev
- Configurar Jest y personalizar su comportamiento, podemos crear un archivo jest.config.js en la raíz de nuestro proyecto. Este archivo debe ser un módulo de JavaScript que exporte un objeto con las opciones de configuración de Jest.

Ejemplo de configuración:

```
module.exports = {
  roots: ['<rootDir>/tests'],
  testMatch: ['**/*.test.js'],
  moduleFileExtensions: ['js', 'json'],
  verbose: true,
};
```

- roots: especifica la ruta de las carpetas que contienen los archivos de prueba.
- testMatch: especifica el patrón de los archivos de prueba que Jest buscará. Por defecto, Jest busca todos los archivos que coinciden con el patrón **/__tests__/**/*.js?(x) y **/?(*.)+(spec|test).js?(x).
- moduleFileExtensions: especifica las extensiones de archivo que Jest considerará al buscar módulos. Por defecto, Jest busca módulos con las extensiones .js, .json y .node.
- verbose: especifica si Jest debe imprimir información adicional durante la ejecución de las pruebas.
- Ejecutar el comando jest junto con los flags necesarios para la ejecución de jest, un ejemplo podría ser el siguiente:

  $ jest --watchAll --verbose --config jest.config.js

Los mocks son herramientas muy útiles para simular el comportamiento de objetos y funciones en los tests unitarios. Con Jest, podemos crear mocks para cualquier objeto o función que queramos utilizar en nuestros tests. Los mocks nos permiten:

- Aislar el código que estamos probando, de manera que podamos centrarnos en una parte específica del código y evitar tener que depender de otros módulos o dependencias.
- Probar situaciones difíciles o inesperadas, como errores de red o errores en bases de datos, sin tener que depender de un ambiente real.

- Hacer pruebas más rápidas y eficientes, ya que no tenemos que realizar operaciones costosas, como operaciones de lectura y escritura en bases de datos, en cada prueba.

Al utilizar mocks, podemos simular el comportamiento de objetos y funciones de tal manera que podamos probar nuestro código sin depender de ambientes reales o sin afectar a otros módulos o dependencias.

Para utilizar mocks en Jest, podemos utilizar los métodos jest.fn() y jest.spyOn(), que nos permiten crear funciones y objetos simulados. También podemos utilizar los métodos mockImplementation() y mockReturnValue() para establecer el comportamiento deseado para nuestros mocks.

Es importante tener en cuenta que, aunque los mocks son muy útiles, también pueden introducir problemas en nuestros tests si no se utilizan correctamente. Por ejemplo, si establecemos un mock para una función o objeto que es utilizado por otros tests, podríamos afectar negativamente la calidad de estos tests. Además, si no restauramos los mocks después de cada prueba, podríamos interferir con otros tests que utilicen los mismos objetos o funciones.

Para evitar estos problemas, es importante seguir algunas buenas prácticas al utilizar mocks:

- Utilizar mocks sólo cuando sea necesario, y asegurarse de que los mocks estén aislados del resto del código.
- Restaurar los mocks después de cada test, utilizando los métodos mockrestore() o mockClear() para evitar interferencias con otros tests.
- Probar los mocks en situaciones reales para asegurarnos de que están simulando correctamente el comportamiento deseado.

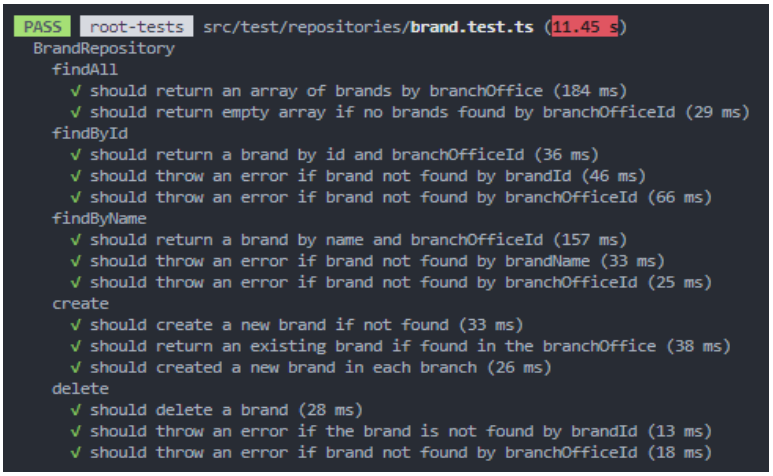
- Utilizar los mocks sólo para simular comportamientos específicos, y no para cubrir todos los casos posibles.

En resumen, los mocks son una herramienta muy útil para hacer tests unitarios más eficientes y efectivos. Pero es importante utilizarlos con cuidado y seguir las buenas prácticas para evitar problemas en nuestros tests.

Veremos cómo se presentan los datos de ejecución de las pruebas con jest cuando los test cumplen su función y cuando algo está fallando en la configuración del test o el código al cual se le hizo validación falla por haber alterado su estructura

Figura 33

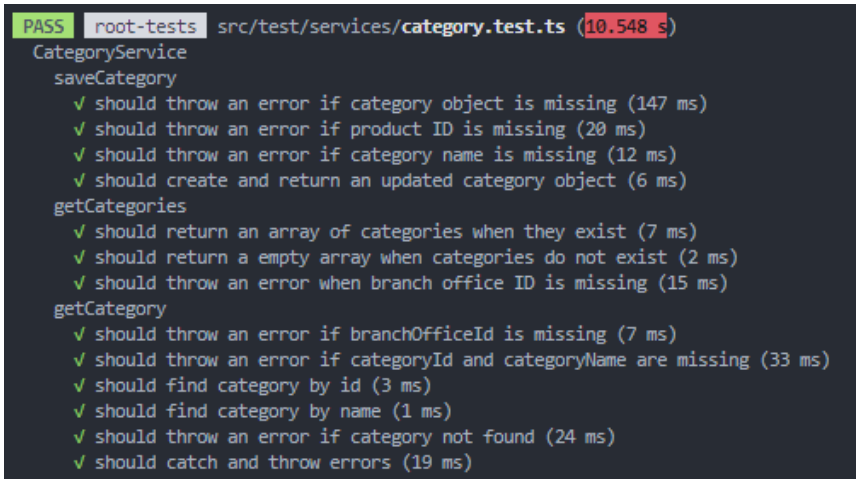
Pruebas de validación a los métodos del repositorio Brand



Nota: Como se puede observar en el resultado del test, el test cumplio con los test definidos, los cuales se encargan de validar el funcionamiento de procesos de inserción en la base de dastos y correcto funcionamiento de la logica definida para la administracion de estos datos.

Figura 34

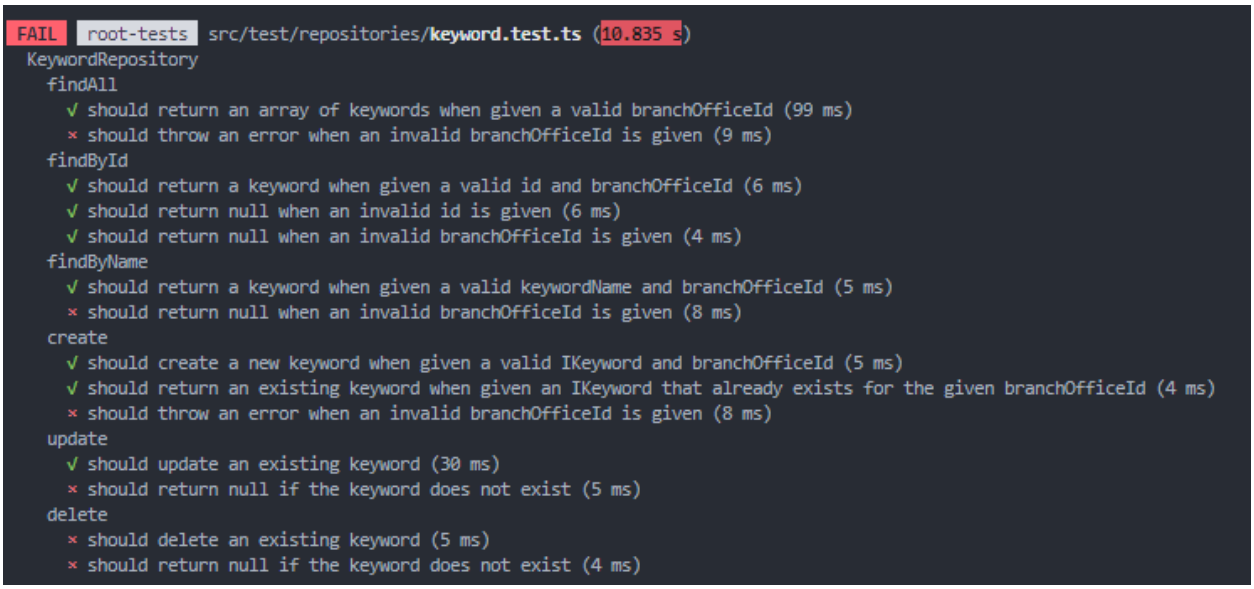
Pruebas de validación al servicio de categoría



Nota: En este test se valida la inserción en la base de datos, la obtención de las categorias asociadas a una sucursal y la consulta de una categoría por su nombre o id igualmente asociada a una sucursal.

Figura 35

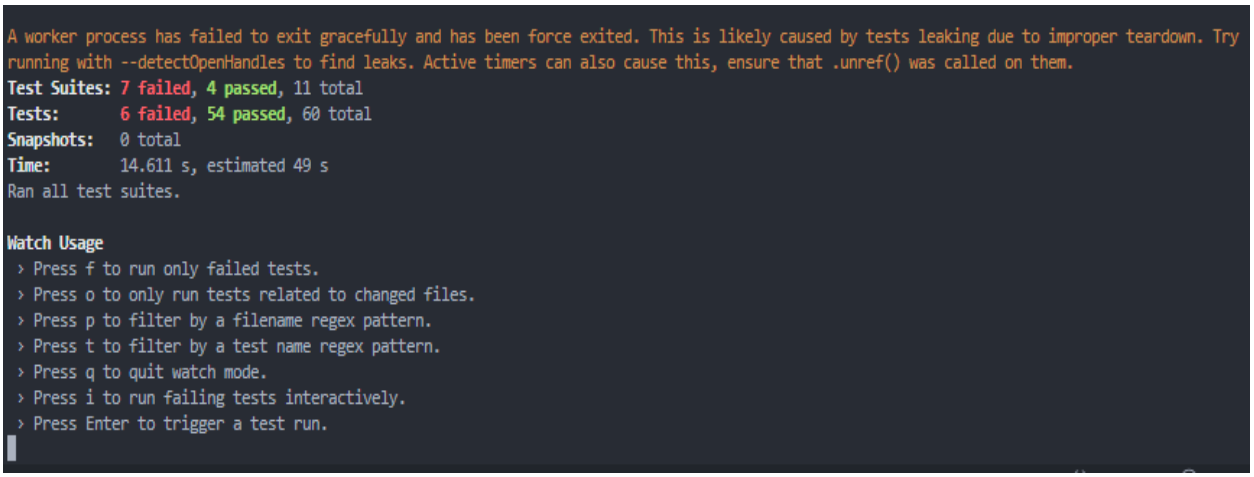
Pruebas de validación a los metodos del repositorio Keyword



Nota: En estas pruebas se validan los métodos del repositorio keyword, en esta imagen se ve como hay pruebas que se encuentran fallando, por lo que su funcionamiento pudo ser alterado debido a modificaciones o porque las pruebas se encuentran mal definidos, la respuesta a este caso es que las pruebas se encuentran incompletas, no se han terminado de definir por lo que es necesario entrar a revisar cuales con las causas que el test este fallando

Figura 36

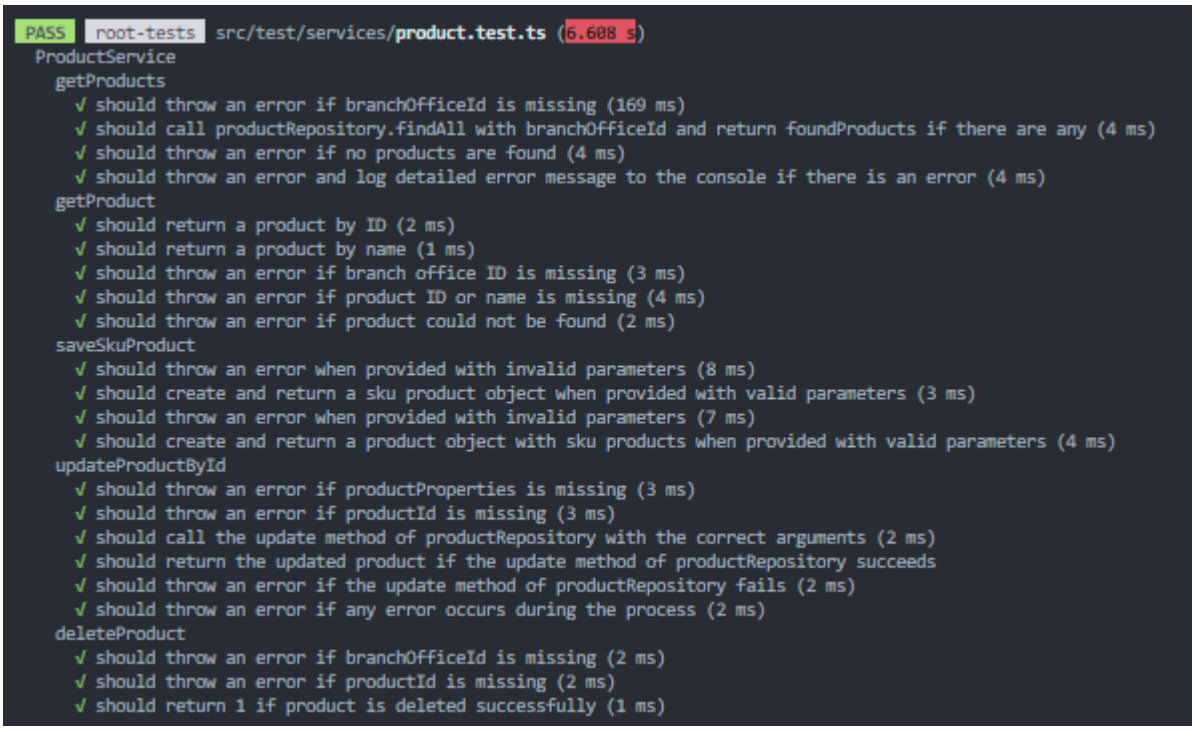
Resultado de la ejecución de las pruebas unitarias con Jest



Nota: En este punto podemos detallar como fue el comportamiento general de las pruebas, en este caso, tenemos los resultados que podemos interpretar de la siguiente manera, las pruebas suites o suites de pruebas, correspondes a los módulos o conjuntos de pruebas de un apartado de código o bloque de nuestro proyecto, como puede ser una funcionalidad. En el caso de los tests o pruebas, son aquellas pruebas que se aplican a la lógica para validar el correcto funcionamiento de esta funcionalidad, en la que se validan los resultados y errores lanzados por excepciones.

Figura 37

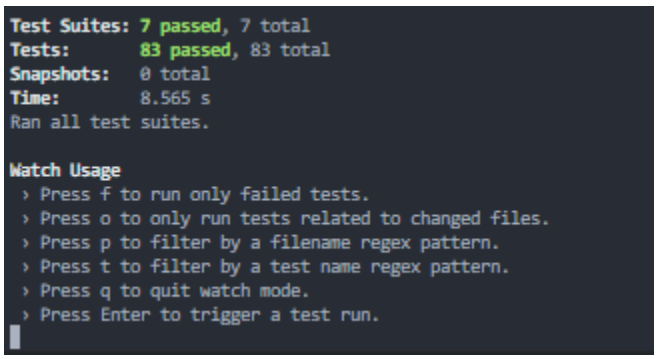
Resultado de validación de los métodos de la clase ProductService



Nota: En la figura anterior se muestra el resultado de las pruebas unitarias implementadas para la clase de ProductService, la cual se encarga de la gestión y administración de los productos, crear, leer, actualizar y eliminar productos que están asociados a una sucursal de una compañía.

Figura 38

Resultado de la ejecución de las pruebas unitarias corregidas



Nota: En esta figura se muestra el resultado de las pruebas ya terminadas y corregidas, habiendo terminado las pruebas para los demás servicios junto con los repositorios, en el que se validaron la correcta implementación de los métodos en cada una de las clases.

4.5.2.8 Documentación del código

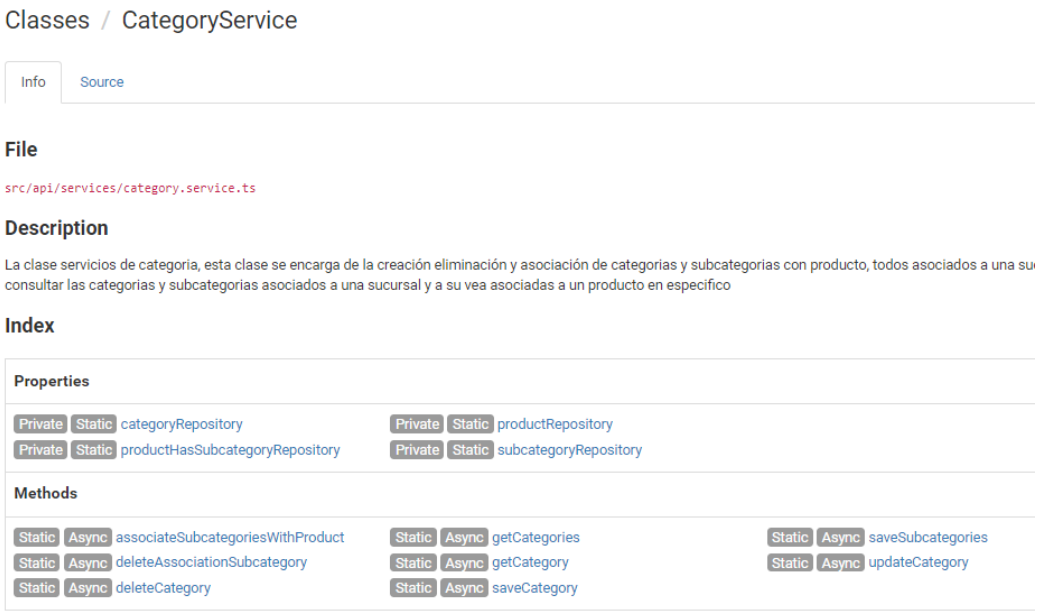
Los comentarios se definieron con un formato JSDoc el cual sirve documentar el código JavaScript/TypeScript mediante la adición de comentarios especiales en el código fuente. Estos comentarios se escriben en un formato especial, similar al formato utilizado por las etiquetas HTML, y contienen información sobre el propósito, la sintaxis y los parámetros de las funciones, métodos y variables del código.

La documentación JSDoc se utiliza para generar automáticamente documentación legible a partir del código fuente, lo que facilita la comprensión del código y su reutilización. Además, los editores de código modernos y los entornos de desarrollo integrado (IDE) pueden utilizar la documentación JSDoc para proporcionar información contextual, sugerencias y autocompletado de código.

Las etiquetas JSDoc más comunes incluyen @param, que describe los parámetros de una función, @returns, que describe el valor de retorno de una función, y @example, que proporciona ejemplos de uso de la función. Las etiquetas JSDoc también se pueden utilizar para describir las propiedades y los métodos de un objeto, así como para documentar los módulos y las clases en un proyecto.

Figura 39

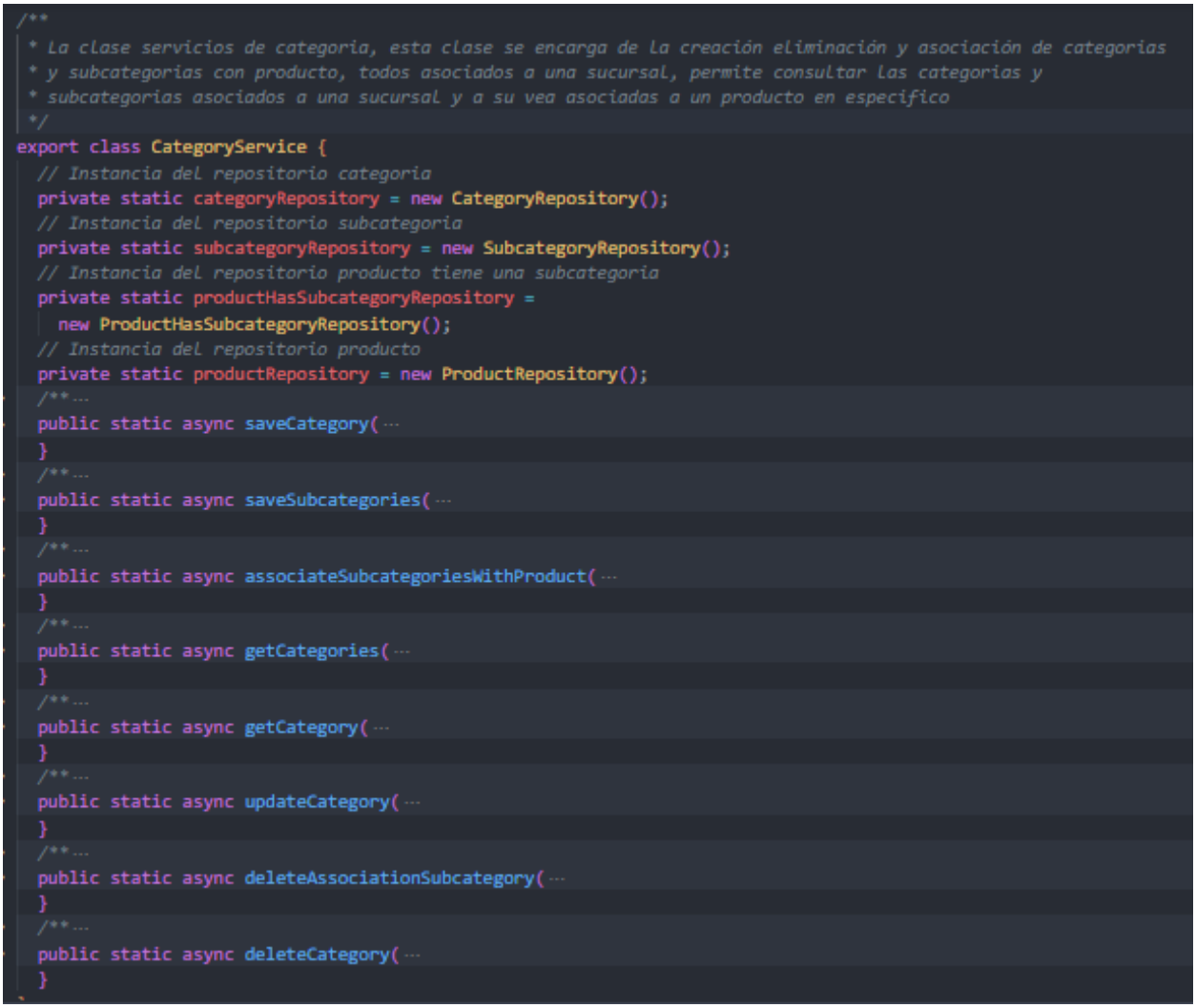
Documentación auto generada de la clase CategoryService



Nota: Como se puede ver en la imagen la documentación en el código fuente se expresa en una vista HTML después de ejecutar el comando de compodoc en el proyecto, el cual proporciona un proyecto con la documentación del código fuente.

Figura 40

Documentación del código fuente para de la clase CategoryService

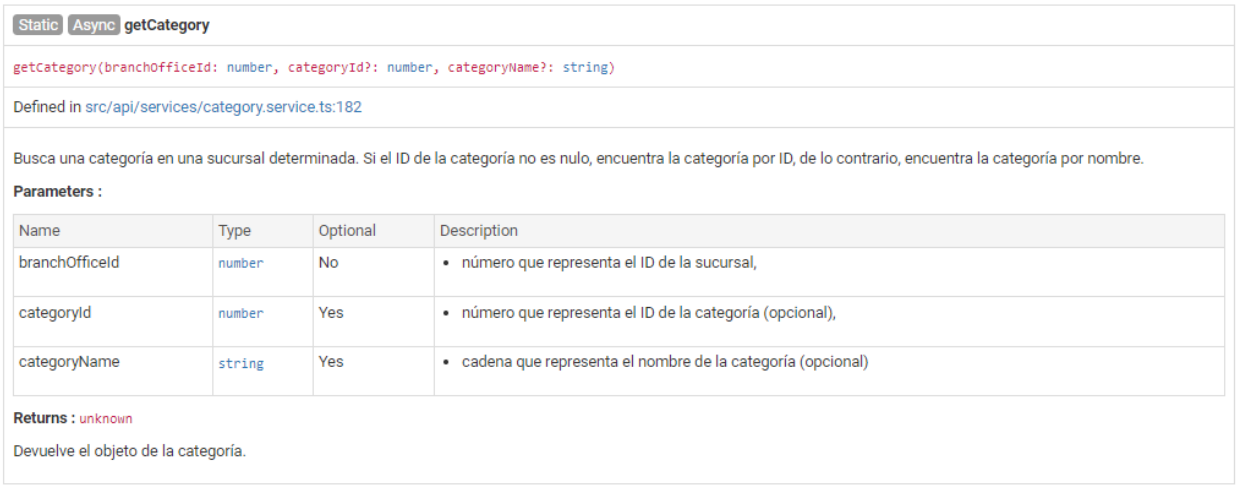


```
/**
 * La clase servicios de categoria, esta clase se encarga de la creación eliminación y asociación de categorias
 * y subcategorias con producto, todos asociados a una sucursal, permite consultar las categorias y
 * subcategorias asociados a una sucursal y a su vea asociadas a un producto en especifico
 */
export class CategoryService {
  // Instancia del repositorio categoria
  private static categoryRepository = new CategoryRepository();
  // Instancia del repositorio subcategoria
  private static subcategoryRepository = new SubcategoryRepository();
  // Instancia del repositorio producto tiene una subcategoria
  private static productHasSubcategoryRepository =
    new ProductHasSubcategoryRepository();
  // Instancia del repositorio producto
  private static productRepository = new ProductRepository();
  /**...
  public static async saveCategory(...
  }
  /**...
  public static async saveSubcategories(...
  }
  /**...
  public static async associateSubcategoriesWithProduct(...
  }
  /**...
  public static async getCategories(...
  }
  /**...
  public static async getCategory(...
  }
  /**...
  public static async updateCategory(...
  }
  /**...
  public static async deleteAssociationSubcategory(...
  }
  /**...
  public static async deleteCategory(...
  }
```

Nota: En la imagen se muestra los mismos métodos que se exponen en la documentación generada con compodoc, dado a que se siguió el formato de documentación

Figura 41

Documentación generada del método getCategory de la clase CategoryService



Static Async **getCategory**

```
getCategory(branchOfficeId: number, categoryId?: number, categoryName?: string)
```

Defined in src/api/services/category.service.ts:182

Busca una categoría en una sucursal determinada. Si el ID de la categoría no es nulo, encuentra la categoría por ID, de lo contrario, encuentra la categoría por nombre.

Parameters :

Name	Type	Optional	Description
branchOfficeId	number	No	• número que representa el ID de la sucursal,
categoryId	number	Yes	• número que representa el ID de la categoría (opcional),
categoryName	string	Yes	• cadena que representa el nombre de la categoría (opcional)

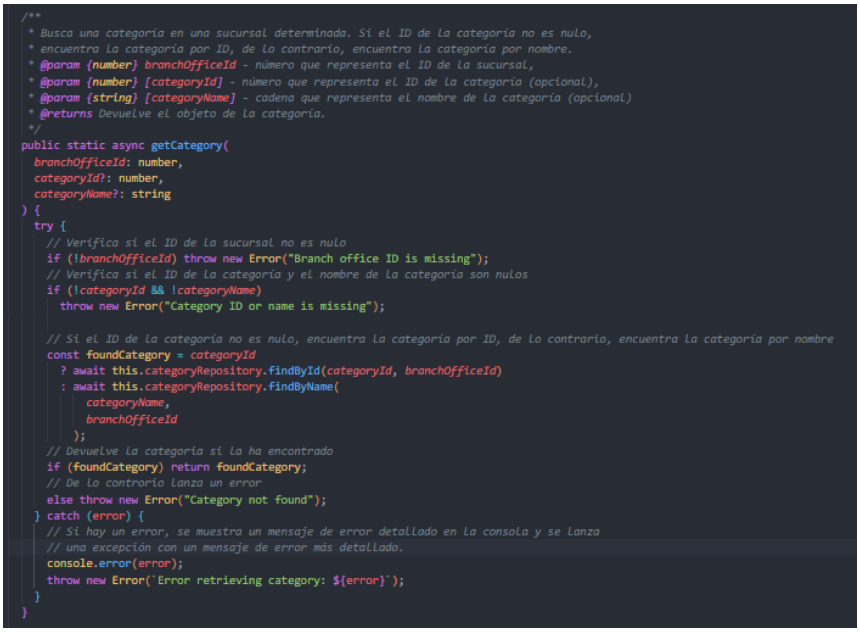
Returns : unknown

Devuelve el objeto de la categoría.

Nota: La documentación generada con comopdoc siguiendo el formado de JSDoc, se muestran lo comentarios ingresados en la documentación del código fuente.

Figura 42

Documentación del método getCategory de la clase CategoryService



```
/**
 * Busca una categoría en una sucursal determinada. Si el ID de la categoría no es nulo,
 * encuentra la categoría por ID, de lo contrario, encuentra la categoría por nombre.
 * @param {number} branchOfficeId - número que representa el ID de la sucursal,
 * @param {number} [categoryId] - número que representa el ID de la categoría (opcional),
 * @param {string} [categoryName] - cadena que representa el nombre de la categoría (opcional)
 * @returns Devuelve el objeto de la categoría.
 */
public static async getCategory(
  branchOfficeId: number,
  categoryId?: number,
  categoryName?: string
) {
  try {
    // Verifica si el ID de la sucursal no es nulo
    if (!branchOfficeId) throw new Error("Branch office ID is missing");
    // Verifica si el ID de la categoría y el nombre de la categoría son nulos
    if (!categoryId && !categoryName)
      throw new Error("Category ID or name is missing");

    // Si el ID de la categoría no es nulo, encuentra la categoría por ID, de lo contrario, encuentra la categoría por nombre
    const foundCategory = categoryId
      ? await this.categoryRepository.findById(categoryId, branchOfficeId)
      : await this.categoryRepository.findByName(
          categoryName,
          branchOfficeId
        );
    // Devuelve la categoría si la ha encontrado
    if (foundCategory) return foundCategory;
    // De lo contrario lanza un error
    else throw new Error("Category not found");
  } catch (error) {
    // Si hay un error, se muestra un mensaje de error detallado en la consola y se lanza
    // una excepción con un mensaje de error más detallado.
    console.error(error);
    throw new Error(`Error retrieving category: ${error}`);
  }
}
```

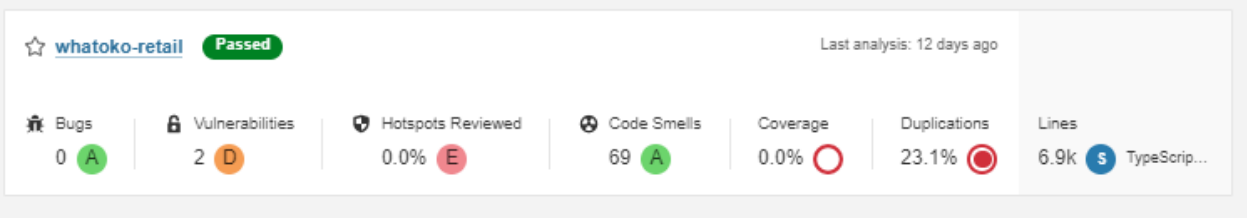
Nota: En la imagen se muestra lo comentarios con el formato de JSDoc y la comentación incode del método, en donde se usan las anotaciones de @param y @returns para describir el comportamiento del método.

4.5.2.9 SonarQube local: verificar la calidad del código

SonarQube es una plataforma de análisis estático de código abierto que se utiliza para evaluar la calidad del código fuente. Su objetivo principal es identificar y corregir problemas de calidad de código, tales como bugs, vulnerabilidades de seguridad, duplicación de código, complejidad, y otros.

Figura 43

Resultados del proyecto calidad del código



Nota: Como se muestra en la imagen el resultado del proyecto en cuanto a bugs fue aceptable, por lo que no encontró ningún bug en el proyecto, por otro lado, existen unas vulnerabilidades de otro microservicio llamado versión, el cual es externo al microservicio de productos

SonarQube ofrece una amplia variedad de herramientas para evaluar la calidad del código, incluyendo análisis de código estático, pruebas unitarias, medición de la complejidad del código, detección de duplicación de código, y otras. Además, también proporciona una amplia gama de métricas y estadísticas para ayudar a los desarrolladores a evaluar la calidad del código y tomar decisiones informadas sobre cómo mejorar su código.

SonarQube es una herramienta esencial para cualquier equipo de desarrollo de software que desee mejorar la calidad de su código y garantizar que su software sea seguro, escalable y fácil de mantener.

4.5.4 Respuestas del API

Las respuestas del API se formularon para la gestión de los productos conectados desde los e-commerce así como los productos sincronizados desde los proveedores, de esta manera se podrá gestionar ambos flujos en una misma base de datos homologada.

4.5.4.1 Servicio de Categorías

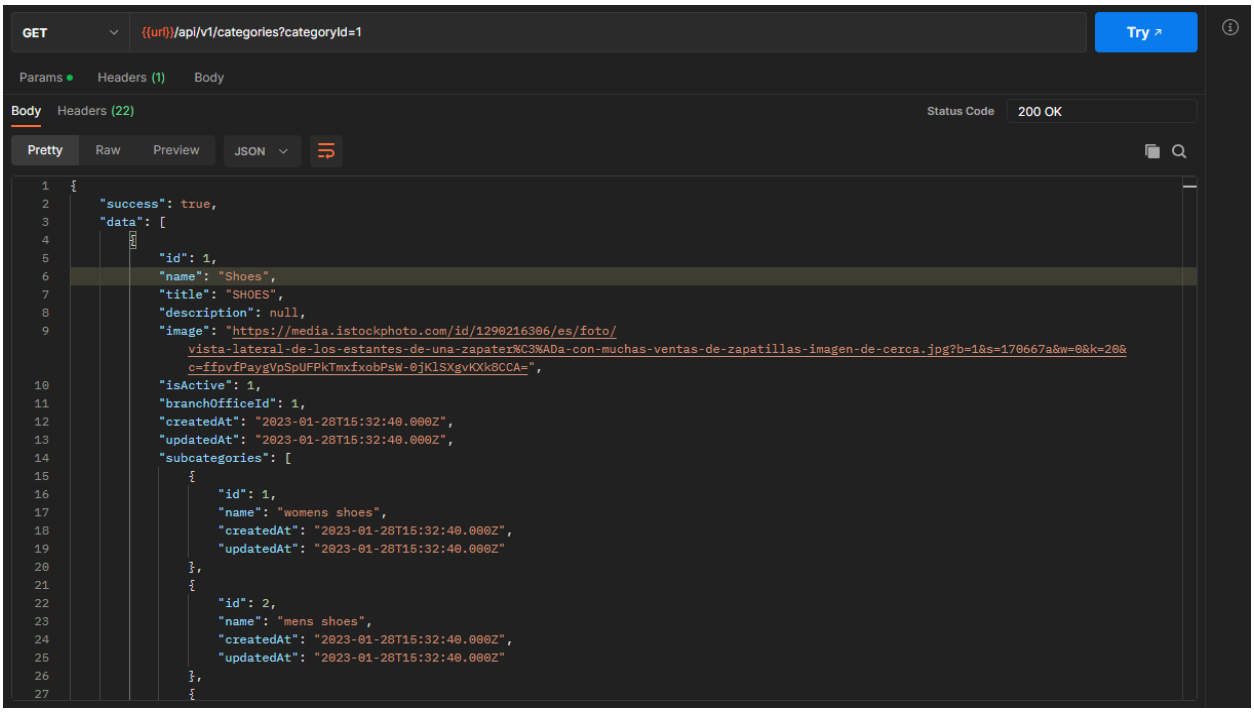
Este servicio se encarga de entregar información a nivel de clasificación de los productos. Cada respuesta depende de la sucursal en la cual esté registrado el usuario autenticado.

4.5.4.1.1 Obtener Lista de Categorías

Este endpoint se encarga de entregar la lista de categorías y subcategorías para una sucursal. En la siguiente captura se puede ver un ejemplo de la respuesta.

Figura 44

Petición al Endpoint para listar las categorías de una sucursal



Nota: En la imagen se puede observar una respuesta correcta por parte del servidor, la cual nos retorna una lista de objetos con el resultado de la petición, los datos como una lista de objetos traídos de la base de datos y un mensaje asociado a la respuesta.

4.5.4.2 Servicio de Consultas Complementarias

Este servicio se encarga de gestionar los modelos complementarios relacionados a productos, como la marca, el tipo y estado del producto y el proveedor, este servicio cuenta con una serie de endpoints de consulta, eliminación y agregación de los registros.

4.5.4.2.1 Obtener tipos de productos

Esta consulta nos retorna qué tipos de productos existen en el servicio. Por ejemplo, productos perecederos, físicos, descargables, etc. En la siguiente captura se puede ver una respuesta.

Figura 45

Petición al Endpoint para retornar los tipos de productos

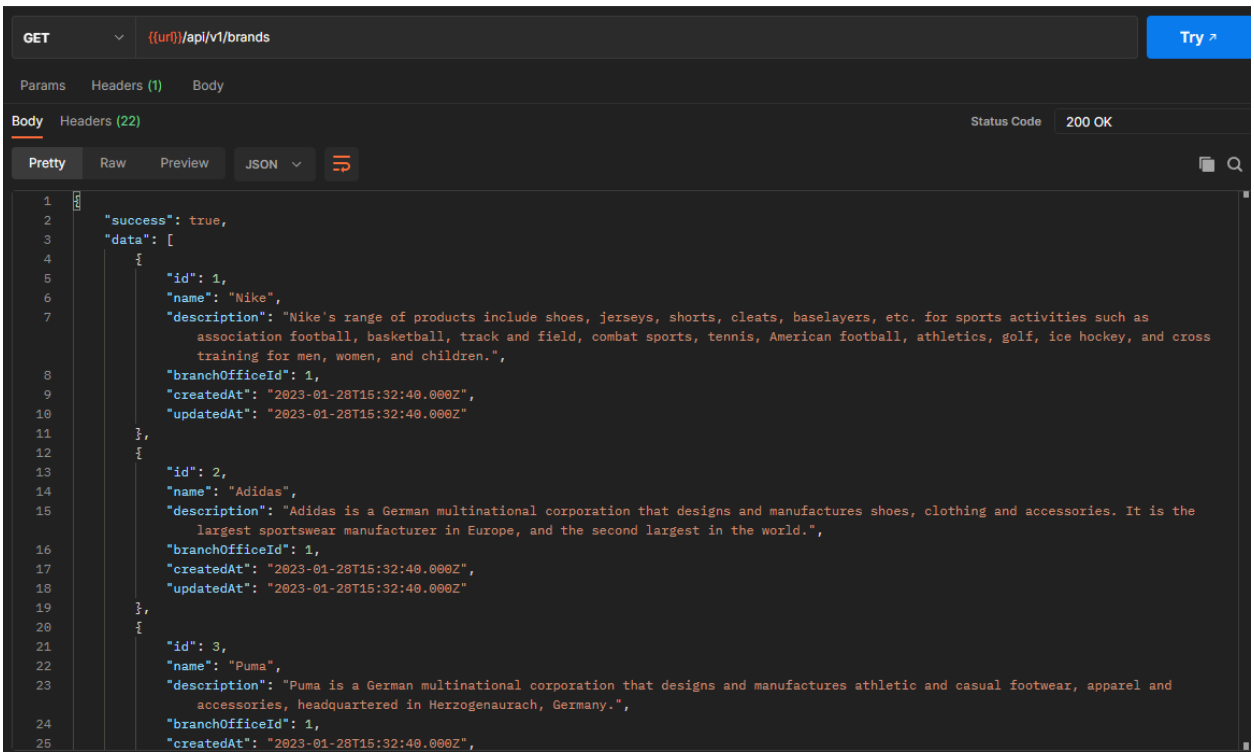
Nota: Para este caso, vemos que retorna un objeto compuesto de la siguiente manera: success indica el resultado de la consulta; data retorna una lista de objetos tipo de producto con sus propiedades; message un texto que nos indica un mensaje describiendo el resultado.

4.5.4.2.2 Obtener Marcas de productos

Esta consulta nos retorna todas las marcas de productos en una sucursal. En la siguiente captura podemos ver un ejemplo de ello.

Figura 46

Petición al Endpoint para listar las marcas de una sucursal



Nota: Este endpoint retorna un objeto con el resultado de la transacción, la lista de datos traídos de la base de datos y un mensaje que describe el resultado de la consulta.

4.5.4.3 Servicio de Productos

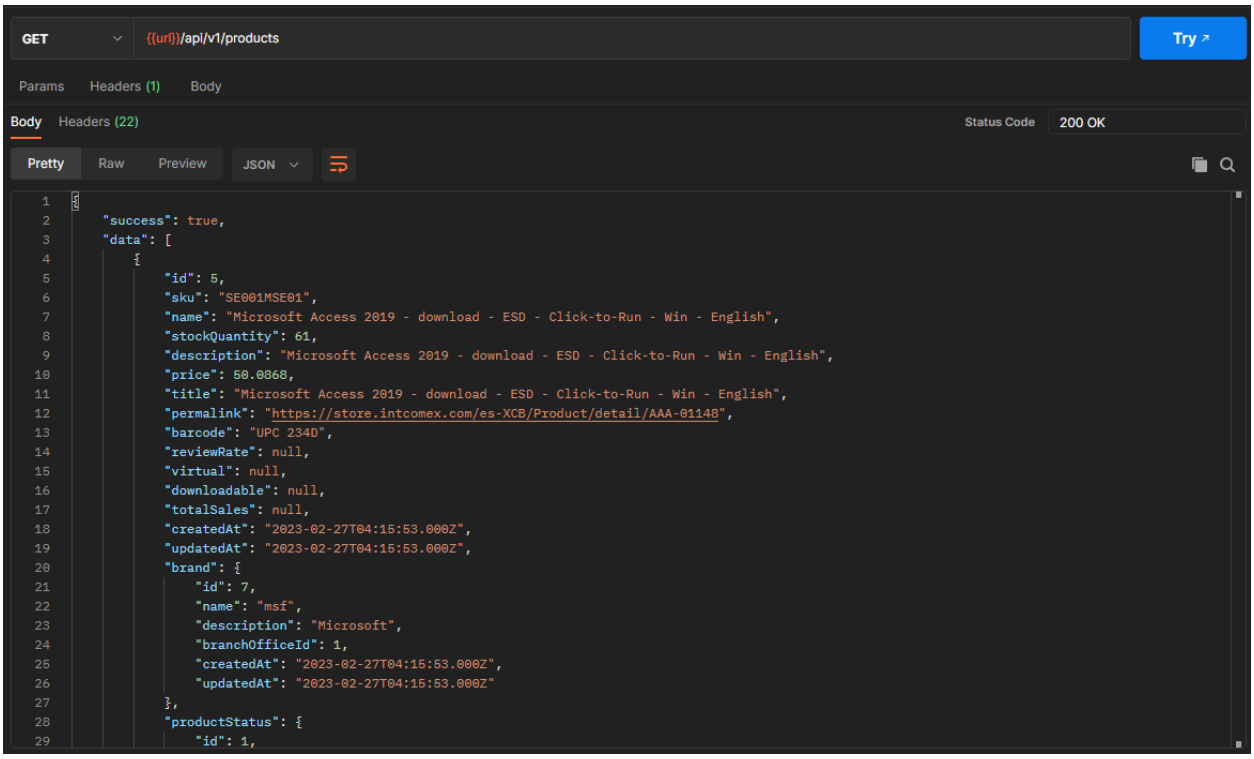
Este servicio entrega información de los productos de una sucursal con el cual el usuario se encuentre registrado.

4.5.4.3.1 Obtener lista de productos de una Sucursal

Esta consulta la lista de productos con su precio, nombre, marca, estado, tipo de producto, etc. En la siguiente captura podemos ver el resultado.

Figura 47

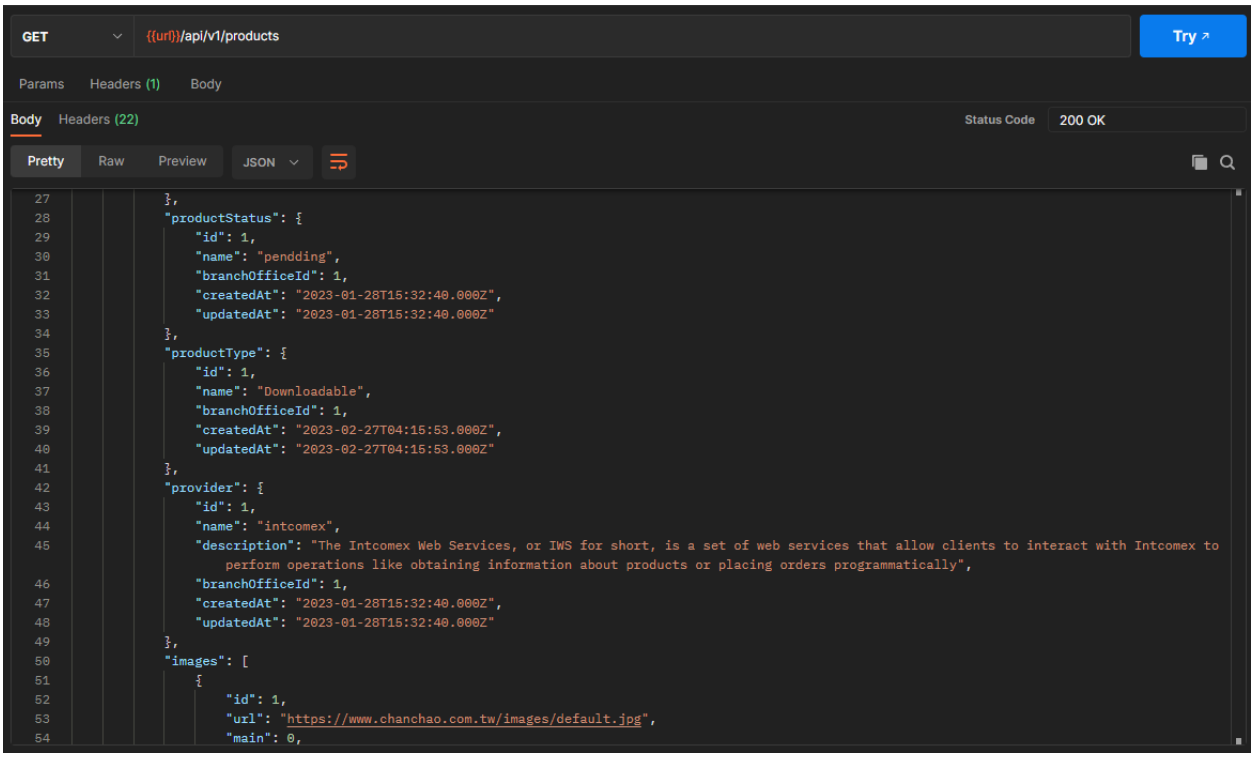
Petición al Endpoint de lista productos



Nota: En el cuerpo de la respuesta se puede observar una lista de objetos producto con todas sus propiedades. Allí se puede observar nombre, precio, identificador, fecha de creación, además de otros datos como la marca, el estado del producto y demás.

Figura 48

Continuación de la petición al endpoint de listar productos.



Nota: En esta imagen se observa información restado de la consulta, donde se puede ver el estado del producto, el tipo del producto, el proveedor o vendedor del producto y la información de una imagen del producto.

4.5.4.4 Servicio de Imágenes

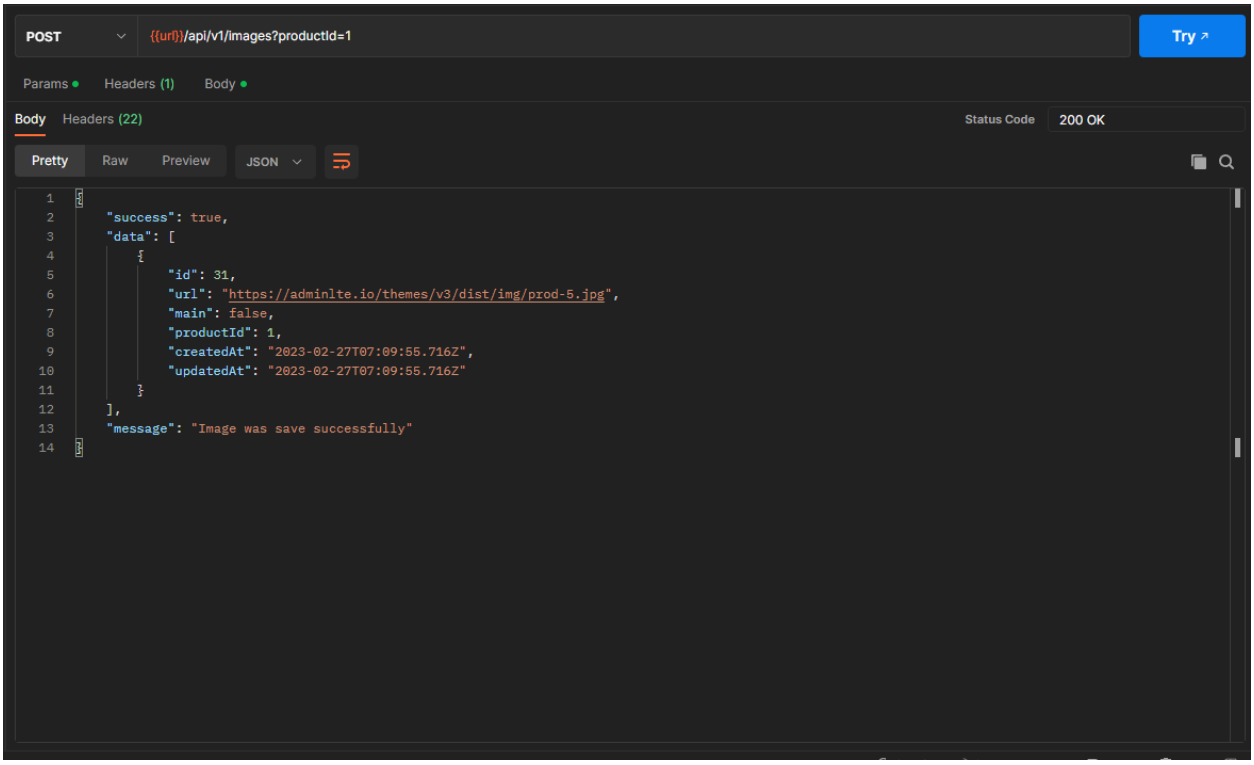
Este servicio se encarga de entregar imágenes de los productos. Cada respuesta depende de la sucursal en la cual esté registrado el usuario autenticado.

4.5.4.4.1 Obtener Imagen

Este endpoint se encarga de entregar una imagen según el identificador. En la siguiente captura se puede ver un ejemplo de la respuesta.

Figura 49

Petición al endpoint de obtener una imagen de un producto



Nota: Esta consulta retorna una imagen según el identificador de la base de datos. Se pueden ver campos como la url de la imagen, la llave foránea con la cual está asociada la imagen y su fecha de creación.

4.5.4.5 Servicio de Propiedades

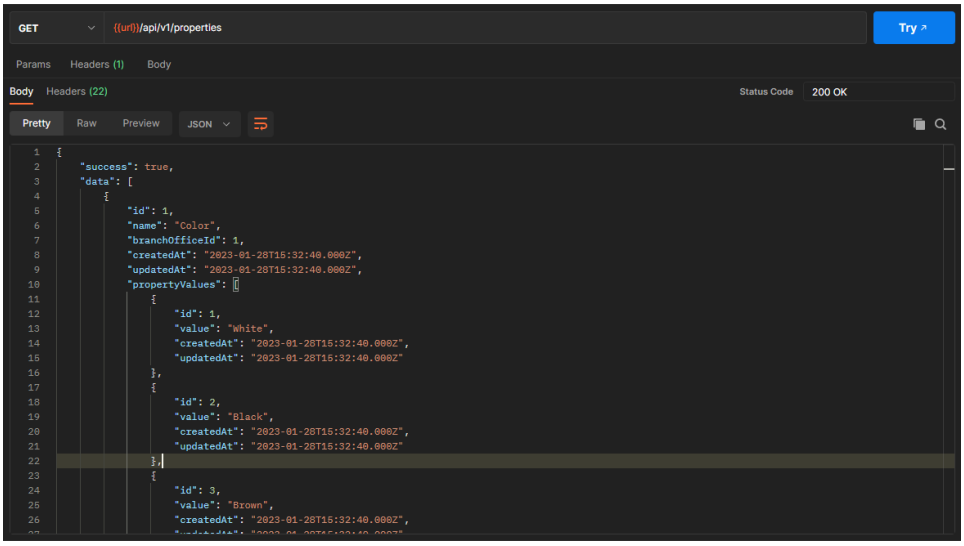
Este servicio se encarga de entregar imágenes de los productos. Cada respuesta depende de la sucursal en la cual esté registrado el usuario autenticado.

4.5.4.5.1 Obtener lista de propiedades para un producto

Este endpoint se encarga de entregar la lista de propiedades para asignar a un producto cuando este se crea en una sucursal. En la siguiente captura se puede ver un ejemplo de la respuesta.

Figura 50

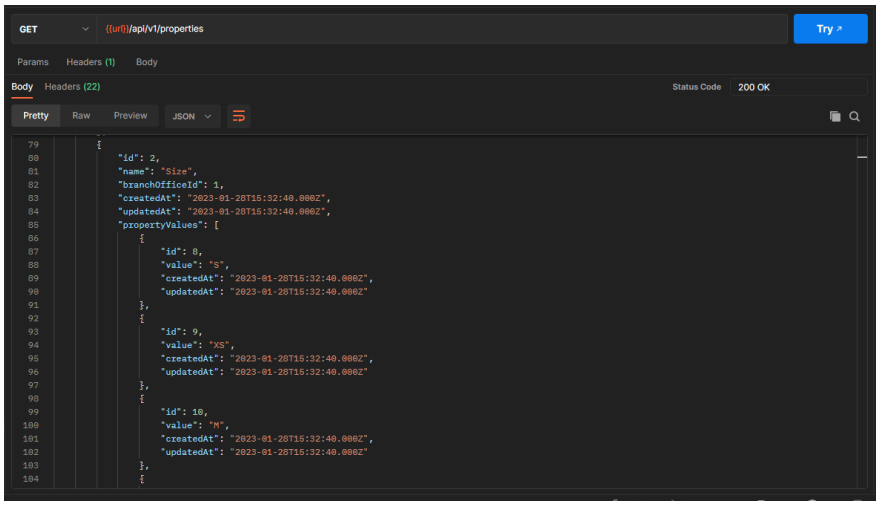
Petición al Endpoint de obtener la lista de Propiedades para un producto



Nota: En esta imagen se observa la respuesta con una lista de objetos que contienen distintos tipos de propiedades. Cada Propiedad está compuesta por un dominio de valores posibles. Por ejemplo, para el caso del color, se tiene la lista de colores que pueden existir en un producto.

Figura 51

Continuación de la lista de Propiedades



Nota: En la imagen se muestra una propiedad llamada talla, esta propiedad se aplica para productos que sean de tipo ropa.

4.5.4.5.2 Agregar una propiedad a un producto

Este endpoint se encarga agregar una propiedad con su información en una sucursal. En la siguiente captura se puede ver un ejemplo de la respuesta.

4.5.4.6 Servicio de Sincronización de Productos

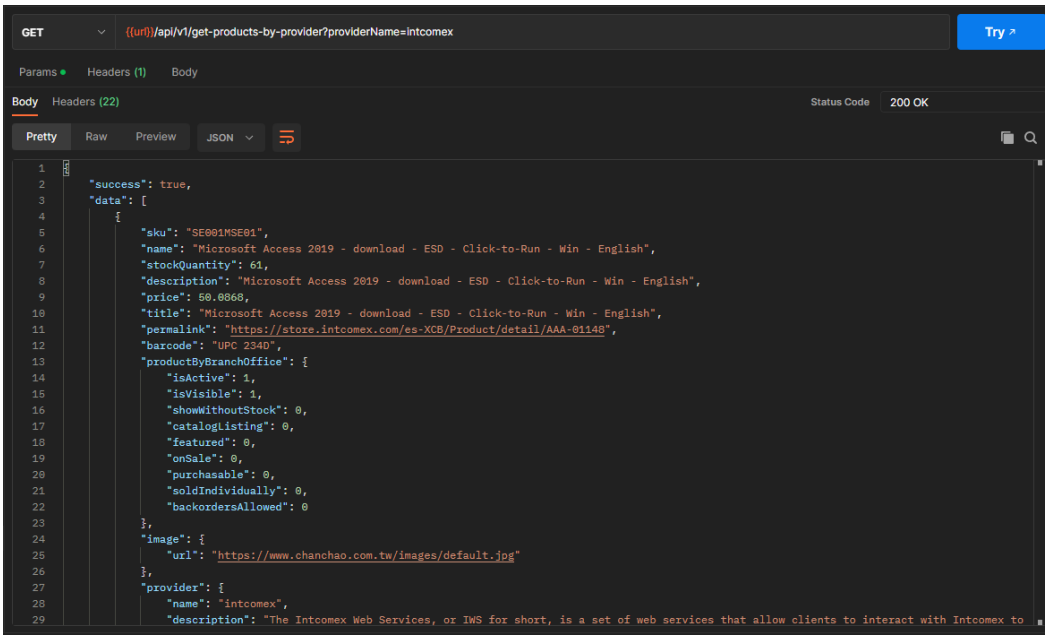
Este servicio se encarga de sincronizar los productos de los proveedores registrados en la plataforma. Cada respuesta depende de la sucursal en la cual esté registrado el usuario autenticado.

4.5.4.6.1 Obtener Productos de un Proveedor.

Este endpoint se encarga de obtener los productos de un proveedor, este se especifica según el identificador de este en la petición. En la siguiente captura se puede ver un ejemplo de respuesta.

Figura 52

Petición al Endpoint de obtener productos de un proveedor en específico

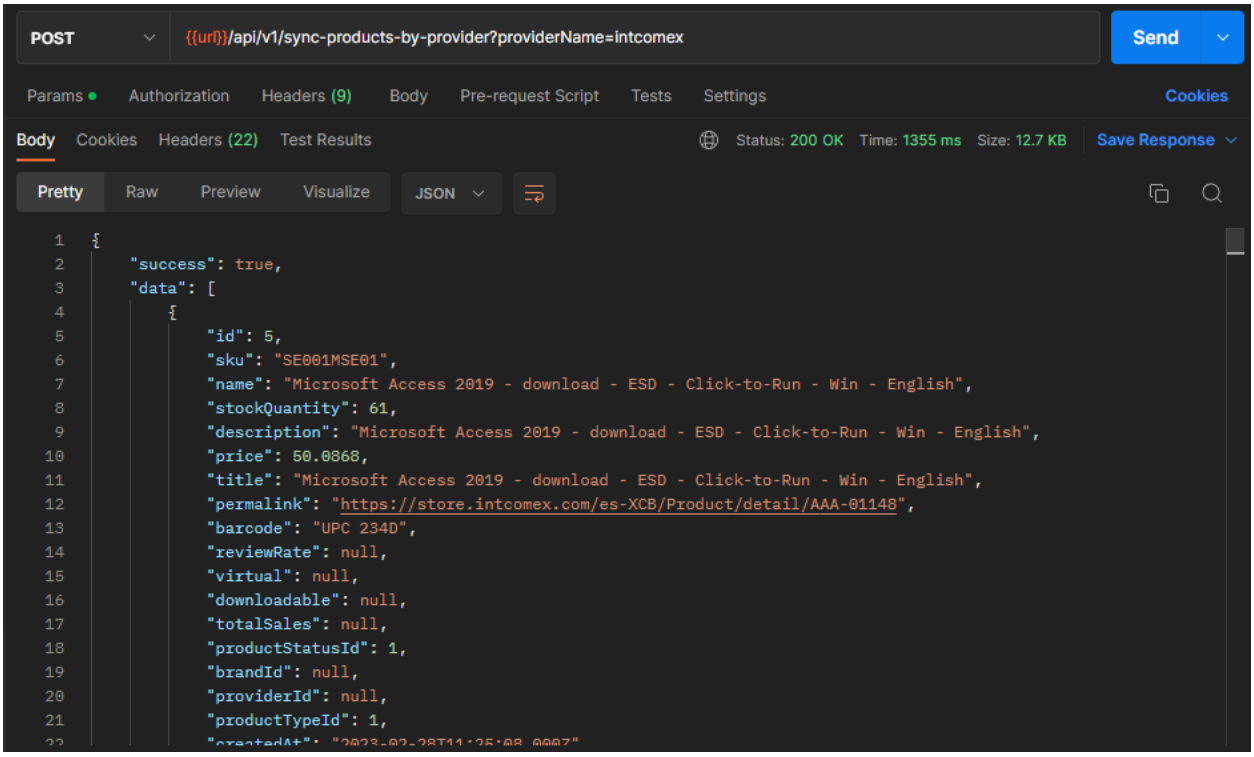


4.5.4.6.1 Sincronizar Productos de un Proveedor.

Este endpoint se encarga de sincronizar los productos de un proveedor, este se especifica según el identificador de este en la petición. En la siguiente captura se puede ver un ejemplo de respuesta.

Figura 53

Petición al Endpoint de sincronizar productos de un proveedor en específico



Nota: En esta respuesta se manda como parámetro query el identificador del proveedor con el cual se quiere obtener sus productos.

4.5.5 Creación de las vistas para el funcionamiento del api con react y TypeScript

Para crear un nuevo proyecto con React y TypeScript, se puede utilizar la herramienta de línea de comandos create-react-app con la opción --template TypeScript.

A continuación, se detallan los pasos necesarios para crear el proyecto:

- Instalar create-react-app globalmente mediante el siguiente comando en la terminal:

  $ npm install -g create-react-app

- Crear un nuevo proyecto React con TypeScript mediante el siguiente comando en la terminal:

```
$ npx create-react-app my-app --template TypeScript
```

En este ejemplo, my-app es el nombre del nuevo proyecto.

- Esperar a que se instalen todas las dependencias y se cree la estructura de archivos.
- Una vez que se haya creado el proyecto, se pueden ejecutar los siguientes comandos para probarlo en un servidor de desarrollo:

```
$ cd my-app
$ npm start
```

Esto abrirá la aplicación en un navegador web en la dirección http://localhost:3000 y se refrescará automáticamente cada vez que se haga un cambio en los archivos del proyecto.

4.5.5.1 Definir las rutas en proyecto react con TypeScript

Para definir las rutas en un proyecto React con TypeScript, se puede utilizar una librería de manejo de rutas como react-router-dom.

Para empezar, se debe instalar la librería react-router-dom mediante el comando:

```
$ npm install react-router-dom
```

Luego, se puede definir un archivo de rutas, donde se especifican las diferentes rutas que la aplicación va a manejar.

4.5.5.2 Instalación de Template

Para el desarrollo de las vistas se instaló y configuro un Template opensource conocido como AdminLTE el cual proporciona unos componentes en HTML y se hacen la conversión con la extensión de vscode llamada HTML to JSX, de esta manera se obtiene los componentes ya listos para ser expuestos como componentes de react.

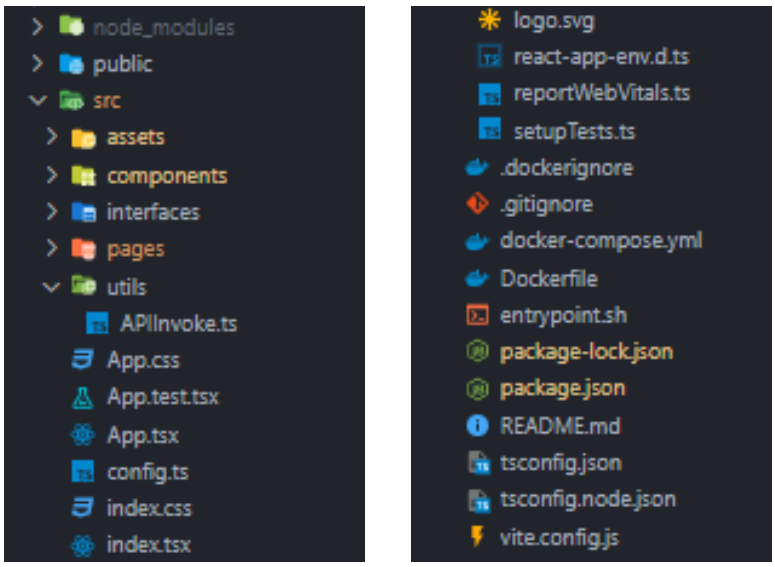
Para lograr configurar el Template AdminLTE, primero debemos descargar el proyecto base, ya sea por la página oficial o través de npm con el comando ‘npm install admin-lte@^3.2 –

save', una vez descargado el proyecto debemos pasar los directorios de dist y plugings al proyecto public de react, estos directorios contienen toda la información relacionada con el funcionamiento de las clases que se usan para definir los diseños

Un paso más para que el proyecto quede bien configurado es pasar los recursos de link en la cabecera del proyecto, y los scripts al final del body, de esta manera se logra configurar correctamente el template en el proyecto.

Figura 54

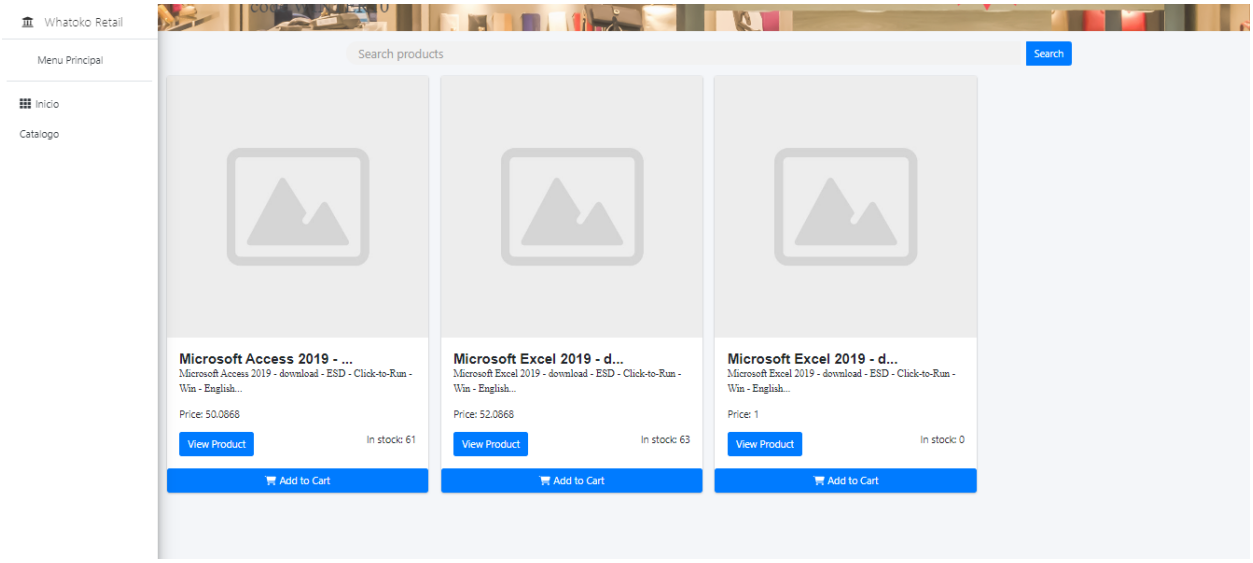
Estructura del proyecto frontend se desarrolló de la siguiente manera



Nota: Proyecto react-ts para el desarrollo del servicio de gestion de productos, en estas imágenes se muestran cómo se estructuro el proyecto frontend, en donde desarrollan las vistas de las funcionalidades definidas en el API de productos que fue desarrollado, el proyecto cuenta con los archivos de validación de typescript, donde se puede definir cuáles son las practicas a tener en cuenta para el desarrollo del proyecto, además se dockerizo el proyecto frontend.

Figura 55

Vistas de productos cargados desde el API desarrollada



Nota: Vista simulada de los productos que llegan del API desarrollada, los productos mostrados corresponden a una lista de productos mockeados, simulando los datos que se reciben del web service de Intcomex. Esta vista busca mostrar los productos en un catálogo, donde el cliente que quiera sincronizar los productos a su negocio del proveedor consultado, lo pueda hacer y llevarlos a las diferentes ecommerce vinculados en la plataforma de whatoko-retail.

5. Conclusiones

Este apartado se presenta los aspectos más relevantes en el desarrollo de la práctica y el proyecto realizado en A&A-Tic. Primero, se recapitulan los objetivos y cómo se lograron cumplir. El objetivo principal fue desarrollar un API REST en Node.js con TypeScript y Express que permita centralizar el proceso de integración de la información de los productos de diferentes proveedores conectados. Se logró crear un API REST que gestiona los productos sincronizados de proveedores y los productos de e-commerce de los clientes vinculados a la plataforma de Whatoko retail. El API se encarga de gestionar los datos relacionados a un producto a través de consultas, incluyendo la creación, obtención, actualización y eliminación de registros de los modelos definidos en la estandarización de datos en la base de datos. Para lograr esto, se homologaron los datos de los e-commerces Wocommerce, Shopify, Vtex y Mercado libre, junto con la plataforma de proveedores como Intcomex y Aliexpress.

El segundo objetivo fue implementar un sistema transaccional con MySQL y Sequelize como ORM. La conexión a la base de datos se hizo con Sequelize, lo que permitió olvidarse de la configuración de las clases DTO y DAO a la base de datos y definir de manera más sencilla las consultas. Aunque esto resulta más sencillo a nivel de gestión y desarrollo del software, resta puntos en los tiempos de respuesta que puede llegar a manejar las consultas muy complejas debido a una configuración previa ya definida que no se puede alterar fácilmente.

Resumiendo, se logró realizar un API REST para centralizar el proceso de integración de información de productos para diferentes proveedores y se implementó un sistema transaccional con MySQL y Sequelize como ORM.

Este proyecto me permitió aprender sobre la homologación de datos, la gestión de bases de datos y la importancia de la eficiencia en los tiempos de respuesta en el desarrollo de software.

Otro objetivo era desarrollar un módulo de validación y autenticación de usuarios para el API, para reforzar la seguridad y protección de los datos.

En el proyecto se desarrolló el módulo de autenticación de un token generado con jsonwebtoken el cual contiene la información de la sucursal que realiza las peticiones, en caso de que el token no contenga información valida de la sucursal será invalido y no se podrá acceder a los endpoints del API, de esta manera se logra proteger los datos almacenados. Otra gestión que se desarrolló para la API, fue el definir las consultas de los datos únicamente hacia la sucursal definida en el token, es decir los datos de las demás sucursales no se podrán consultar por las demás sucursales y no habrá datos sobre puestos, cada sucursal manejara una independencia de los datos que se almacenan y se asocian a esta sucursal, se aplicaron middleware de validación de a conexión, como las cabecearas y cors para así poder gestionar la conexión de otros servicios al api y tener una mayor seguridad en las consultas, y por último, en cada consulta los datos requeridos son validados y gestionados con mensajes de error.

Continuando con los objetivos, otro era hacer pruebas de funcionamiento y error para validar los protocolos del API, e implementar sus respectivas modificaciones.

Las pruebas de funcionamiento con postman desarrolladas en el apartado anterior ayudaron a validar el correcto funcionamiento de los endpoints del API, igualmente se desarrollaron otro tipo de validaciones, conocidas como pruebas unitarias las cuales consisten en validar pequeñas funcionalidades asociadas aun un bloque de código, como puede ser una función. Estas pruebas se desarrollaron con jest, el cual presenta un stack de métodos para la simulación del comportamiento del código, sin entrar a validar ninguna consulta en la base de datos, eso evita problemas en fallos de la conexión, en la autenticación, y latencias del servidor, permitiendo que los datos sean probados sin depender de ningún factor externo a la prueba.

Por último, se tiene que generar la documentación requerida para uso del API, con sus respectivos canales de acceso y descripción de los protocolos.

La documentación del api se generó con compodoc el cual se encarga de automatizar su creación a través de los comentarios del código fuente que siguen el formato JSDoc, esta práctica ahorra mucho tiempo en el desarrollo de una interfaz gráfica que permita validar y ver estos comentarios de manera visual y a la mano para los diferentes desarrolladores, que buscan usar, refactorizar y mantener el código generado,

Habiendo cumplido con los objetivos del proyecto, agradezco la oportunidad de desarrollar este proyecto que me permitió entender lo importantes que tiene los sistemas de integración. Estos sistemas facilitan y ayudan a gestionar mejor los procesos de negocio de las organizaciones, permitiendo mejorar la productividad y eficiencia teniendo como referencia este proyecto, el cual se basa en la integración de varias plataformas ERM e-commerce y la sincronización de productos de proveedores en estos e-commerce , tiene como propósito el mejorar el rendimiento de muchos de estos e-commerce que podrían verse beneficiados por la venta de productos de terceros que se encargan de toda esta gestión logística de los productos.

Los sistemas de integración novedosos ayudan a mejorar la gestión de las organizaciones y está claro que apostar a este tipo de negocios de alguna manera beneficia los clientes que quieran mejorar y delegarles estas funcionalidades a terceros.

Referencias Bibliográficas

Bigelow, S. J. (2021, 02 05). *Guide to building an enterprise API strategy*. Retrieved from TechTarget: https://www.techtarget.com/searchapparchitecture/Guide-to-building-an-enterprise-API-strategy

Bigelow, S. J. (2022, 11). *What is public cloud? Everything you need to know*. Retrieved from TechTarget: https://www.techtarget.com/searchcloudcomputing/definition/public-cloud

Bojinov, V. (2018, 04). *RESTful Web API Design with Node.js 10: Learn to create robust RESTful web services with Node.js, MongoDB, and Express.js, 3rd Edition*. Retrieved from GitHub: https://github.com/PacktPublishing/RESTful-Web-API-Design-with-Node.js-10-Third-Edition

Bourne, V. (2021). *Annual APIs and Integration Report 2021*. Retrieved from Software AG: https://www.softwareag.com/content/dam/softwareag/global/marketing-material/en/ebook/webmethods/vanson-bourne-report-2021.pdf.sagdownload.inline.1621547053450.pdf

Castiblanco, J. (2022, 04 06). *Cómo los pagos digitales y el comercio electrónico están ganando terreno en América Latina*. Retrieved from imk.global: https://imk.global/como-los-pagos-digitales-y-el-comercio-electronico-estan-ganando-terreno-en-america-latina/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Retrieved from UCI: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Gillis, A. S. (2022). *REST API (RESTful API)*. Retrieved from TechTarget: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API

Microsoft Learn. (2022). *Estilo de arquitectura de microservicios*. Retrieved from Microsoft Learn: https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

Microsoft Learn. (2023, 02 15). *¿Qué es DevOps?* Retrieved from Microsoft Learn: https://learn.microsoft.com/es-es/DevOps/what-is-DevOps

Microsoft Learn. (2023, 01 18). *Diseño de API web RESTful*. Retrieved from Microsoft Learn: https://learn.microsoft.com/es-es/azure/architecture/best-practices/api-design