



Informe Proyecto de Grado

“Implementación para administración de errores de la empresa IKEA”

Estudiante: Fernando Zamorano Jadue

Carrera: Ingeniería Civil Informática

Asignatura: Pasantía Full-Time

Empresa: IKEA

Supervisor: Ramón Labbe

Profesor guía: Víctor Nivia

Tabla de contenido

1. Resumen	3
1.1. Resumen Ejecutivo	3
1.2. Abstract	4
2. Introducción	5
3. Objetivos	10
3.1. Objetivo General	10
3.2. Objetivos Específicos	10
4. Estado del arte	11
5. Soluciones propuestas	13
6. Evaluación económica	16
7. Metodología	19
8. Medidas de desempeño	20
9. Desarrollo e Implementación del proyecto	21
9.1. Desarrollo	21
9.2. Implementación	35
10. Resultados cualitativos y cuantitativos	36
11. Conclusiones y discusión	39
12. Referencias	40
13. Anexos	41

1. Resumen

1.1 Resumen Ejecutivo

En el presente documento se llevó a cabo una investigación sobre el funcionamiento logístico de la empresa IKEA, relacionado al área de Tecnologías de la Información y bases de datos. En este contexto, se realizó un análisis respecto a la base de datos RPC (Retail Product Catalogue) utilizada por la empresa.

Los productos de IKEA que son operados en Chile, se actualizan en la base de datos de MBVC (Master Version Business Center), desde donde determinados elementos son ingresados a RPC mediante una tarea automatizada denominada "Product Aggregator." Sin embargo, esta operación automatizada se ve afectada por errores en la base de datos de MVBC, al ingresar productos con datos erróneos debido a causas humanas, lo que implica que el área de IT debe ingresar productos forzosamente en RPC. Esta situación genera complicaciones en las operaciones tanto logísticas como económicas de la empresa, ya sea retrasos en importaciones, multas, despachos, gasto de horas hombre IT, ventas, entre otros.

La solución propuesta se materializa en una interfaz de software diseñada para el ingreso automatizado de productos en RPC. Esta interfaz aborda la raíz del problema, minimizando los errores asociados con la carga manual y optimizando el proceso de transferencia al RPC.

La implementación de esta solución ha demostrado resultados positivos durante su prueba, reduciendo los errores al ingresar productos registrados en 45 días, desde 84.956 a 16.991. Además, las horas de IT destinadas a la inyección de productos semanales, fue reducida de 8 horas a 2 horas.

1.2 Abstract

This document presents an investigation into the logistical operations of the company IKEA, specifically focusing on the realm of Information Technologies and databases. The study centers on the RPC database (Retail Product Catalogue) employed by the company.

In the case of IKEA in Chile, products are updated in the MBVC database (Master Version Business Center), and certain elements are then transferred to RPC through an automated task known as "Product Aggregator." However, this automated operation is susceptible to errors originating from the MVBC database, as products with incorrect data are sometimes entered due to human errors. This compels the IT department to manually input products into RPC, leading to complications in both logistical and economic operations, such as import delays, fines, dispatch issues, IT man-hours, and sales, among others.

The proposed solution materializes in the form of a software interface designed for the automated input of products into RPC. This interface addresses the root of the problem, minimizing errors associated with manual entry and optimizing the transfer process to RPC.

The implementation of this solution has yielded positive results during testing, reducing errors in product entry from 84,956 to 16,991. Additionally, the IT hours dedicated to weekly product injections have been reduced from 8 hours to 2 hours.

2. Introducción

La empresa IKEA corresponde a una compañía multinacional de origen sueco fundada hace más de 80 años, con presencia en más de 28 países, siendo parte de la industria de la producción y manufactura de artículos y muebles de hogar bajo el concepto de ventas minoristas. El modelo de negocio y propuesta de valor asociado a esta empresa se basa en una estrategia de liderazgo de costos. Este tipo de estrategia tiene como objetivo encontrar costos de producción más bajos que los de la competencia, de modo que esto permita rebajar los precios de sus productos y aumentar las ventas. De esta manera, la misión planteada por la empresa se basa en otorgar gran variedad de productos para el hogar y decoración, de diseños únicos y funcionales, a precios que permitan la accesibilidad a la mayoría de la población. Asimismo, la visión de IKEA consiste en "Crear un mejor día a día para la mayoría de las personas", donde se señala como objetivo principal lograr un cambio positivo a nivel mundial, desde la obtención de recursos para producir los artículos hasta lo que causa la adquisición del producto en las personas, priorizando la sostenibilidad y transparencia en cada etapa.

Actualmente, la empresa decidió abrir su primera tienda en Chile el año 2022, en el nuevo centro comercial Open Kennedy, ubicado en la comuna de Las Condes. Tras la popularidad de esta, donde el día de la inauguración 7500 personas llegaron a visitar la primera sede de IKEA, a fines del mismo año se decide abrir una segunda sucursal en la comuna de Cerrillos. Es así como, debido al éxito de IKEA en Chile, se ha generado un proceso de expansión en el continente por parte de la compañía, siendo sus próximos objetivos Perú y Colombia, donde el desarrollo de sucursales en este último ya se encuentra en proceso.

Con 456 tiendas en 62 mercados y 231 mil empleados a nivel mundial, entre el 1 de septiembre y el 31 de agosto de 2022, IKEA logró 44,6 billones de euros en ventas, lo que resultó en un alza de 6,44% comparado con el año 2021. Así, debido a estas condiciones de crecimiento, los coworkers o trabajadores de la empresa han debido enfrentar las consecuencias que conlleva realizar una rápida propagación, tales como desarrollar herramientas de software en un corto periodo de tiempo, complejidades en las implementaciones logísticas debido a una mayor demanda, falta de recursos humanos y económicos, entre otros. Esto ha provocado el descuido de ciertos procesos y

sistemas, puesto que se llevan a cabo tareas que requieren máxima prontitud, como podrían ser los procesos de software de ventas, seguimientos de productos y reembolsos.

Uno de los procesos importantes que resulta afectado por esta situación parte de la base de que todos los productos de la empresa IKEA se encuentran dentro de un catálogo denominado Catálogo Inter IKEA. De dicho catálogo, se escogen determinados productos en función del país en el que se quieran comercializar dichos artículos. La identidad que se encarga de este proceso es la MVBC (Master Version Business Center), la cual genera el catálogo de IKEA para Chile/Colombia, enriqueciendo los productos de este según los requerimientos de cada país, para posteriormente añadirlos dentro de su base de datos. Esta identidad envía determinados productos de su base de datos hacia la base de datos del RPC (Retail Product Catalogue), mediante una tarea programada denominada “Product Aggregator”. Dicha tarea automatizada, cada 24 horas, envía peticiones de ingreso de productos preestablecidos en MVBC hacia el RPC. En la base de datos de este último, solo se aceptan productos almacenables en bodegas, puesto que a esta identidad es a la que se accede para realizar todas las operaciones logísticas de productos, tales como importaciones, ventas, desplazamiento de productos, revisión de órdenes de compra, almacenaje en bodegas, entre otros.

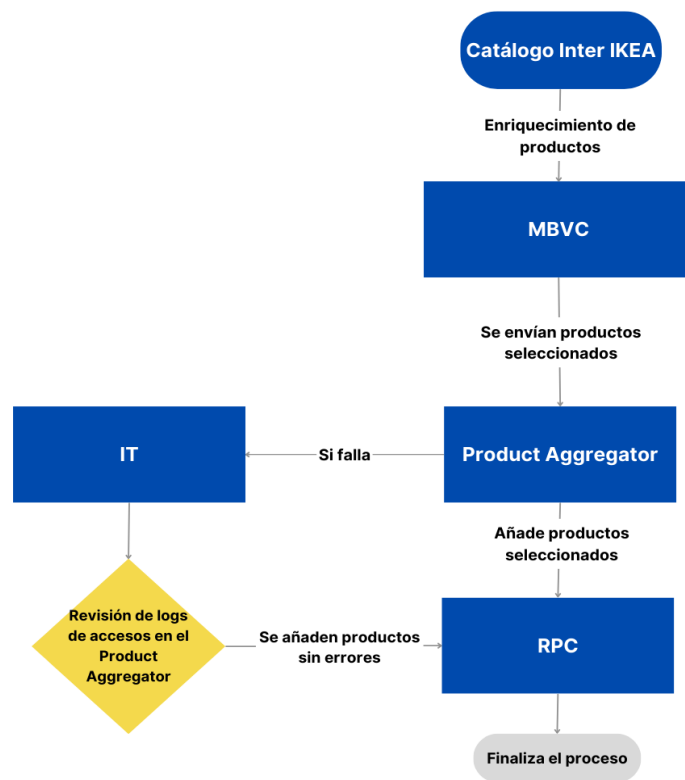


Figura 1. Diagrama del proceso completo del ingreso de productos a la base de datos RPC.

El problema que se genera de todo este proceso es que, cuando las personas que operan en MVBC ingresan productos con errores a sus bases de datos, el Product Aggregator no puede añadirlos de forma correcta en el RPC, ya sea por campos vacíos que son requeridos o por campos erróneos. Esto último es detectado tardíamente en el momento en que se intenta acceder a los productos en la base de datos del RPC y estos no existen. La ausencia de productos en esta base de datos, y su detección tardía, genera que dichos elementos no puedan ser sometidos a operaciones de logística, lo cual trae como consecuencia las siguientes situaciones:

- **Retención de importaciones en aduana:** Cuando llegan productos de diversos países mediante las importaciones, estos deben estar presentes dentro de las bases de datos del RPC, puesto que, en caso de no estar, dichos productos no pueden ser retirados de aduana. Esta situación genera la multa Demurrage, la cual corresponde a los cargos que se aplican a los clientes que retienen sus contenedores por más tiempo que el determinado, implicando un gasto de 100 a 150 dólares diarios por cada contenedor retenido en aduana.

- Imposibilidad de sufragar a proveedores: Que un producto no se encuentre en la base de datos, es sinónimo de que dicho producto no existe. Dado esto, en el caso de que se compren productos a proveedores y estos no hayan sido añadidos correctamente en la base de datos del RPC, se imposibilita la opción de remunerar a los proveedores los productos acordados, afectando claramente a las operaciones de negocio, congelando el mercado.
- Imposibilidad de realizar operaciones logísticas: Tanto el almacenaje de productos como su desplazamiento, no pueden ser llevados a cabo si su existencia no se reconoce dentro de las bases de datos. Por ejemplo, si se despachan productos a una bodega, estos no pueden ser aceptados hasta que hayan sido ingresados correctamente en RPC, lo cual puede llegar a generar horas de espera, y de la mano, retrasos en órdenes de compra y retrasos en ventas de productos. De esta manera, ambos factores se ven influenciados por la presencia o ausencia de los productos dentro de RPC. Asimismo, en promedio, se pueden llegar a retrasar hasta un día en realizar las inyecciones de productos. Por ello, cabe destacar que cada hora cuenta al momento de llevar a cabo estas acciones, puesto que las operaciones de mercado se congelan temporalmente.

Se han detectado productos que han tardado entre 3 y 4 semanas en ser añadidos correctamente en RPC, lo que pondría en riesgo el cumplimiento de la Ley de Pago a 30 días (Ley N. 21.131). Esta normativa sanciona a empresas que no han pagado a sus proveedores en 30 días desde la fecha de recepción de la factura, siendo equivalente al interés corriente para operaciones no reajustables en moneda nacional de más de 90 días. Además de esto, se generan intereses por mora de parte de los proveedores, en donde el comprador en mora debe pagar una comisión fija del 1% del saldo insoluto adeudado, es decir, el saldo restante que aún se debe pagar.

Actualmente, para poder combatir dicha problemática, los coworkers o trabajadores de IKEA que están intentando acceder al RPC, envían una lista con los elementos faltantes en esta base de datos al área de IT, quienes tienen el trabajo de revisar los logs de acceso del Product Aggregator, descargar dichos logs en un archivo “.json” y convertirlos manualmente a un Excel, en donde finalmente se identifican los errores que se generaron a la hora de intentar ingresar dichos productos. Posterior a ello, el área de IT realiza inyecciones de código con los productos faltantes y sin errores en el RPC, viéndose afectada la capacidad y rendimiento de IT por tener que satisfacer

esta necesidad. Todo el proceso anterior es consecuencia de la rápida propagación de la empresa y, por ende, del desarrollo precipitado de un software, lo que lleva a situaciones como la descrita previamente.

3. Objetivos

3.1. Objetivo General:

Concretar operaciones de logística sin contratiempos mediante la implementación de un software operativo dentro de IOMC, el cual corresponde a la interfaz desde la cual los trabajadores de IKEA realizan operaciones de negocio y logística, para comienzos de diciembre de 2023, de forma que permita a los asociados que accedan al RPC realizar inyecciones de productos, teniendo la capacidad de visualizar si se pudo actualizar un producto, si es que hubo algún error y cuál fue el error.

3.2. Objetivos Específicos:

1. Desarrollar un frontend en el cual los usuarios puedan verificar el estado de los productos, realizar inyecciones de productos y comprobar si hubo algún error.
2. Desarrollar un backend que obtenga los logs de acceso del Product Aggregator, permitiendo realizar inyecciones de productos en el RPC, así como revisar los productos presentes en este, lo cual es controlado por el usuario mediante el frontend.
3. Incrustar el software desarrollado en IOMC, para que corresponda a una funcionalidad más de dicho sitio y los empleados accedan desde dicha interfaz.
4. Establecer el software desarrollado en el uso cotidiano del área de IT para determinar si este tiene un real impacto en el trabajo diario de los colaboradores.

4. Estado del arte

La última década se ha caracterizado por la gran cantidad de avances tecnológicos que se presentan continuamente, en donde el alto consumo de dichas tecnologías ha demandado a las empresas mejorar la eficiencia de sus procesos logísticos, la gestión de inventario y su registro en las bases de datos, siendo estos pilares fundamentales para compañías de diversas áreas. Surge así la necesidad de mantener un control claro y actualizado de los productos presentes, buscando reducir el margen de error al realizar operaciones de logística, incentivando así a emplear nuevas tecnologías. Se ha identificado que el uso efectivo de herramientas tecnológicas, especialmente en el ámbito del e-commerce, genera un impacto significativo en las ventas. De esta forma, se destaca la importancia de integrar de manera efectiva las tecnologías de la información y comunicación en los procesos logísticos y de inventario, consolidándose como elementos clave para el éxito y la competitividad empresarial en la era actual (Medina, 2018).

Los beneficios de tener un inventario correctamente registrado, pueden reflejarse claramente en el caso de la empresa Sodimac. De esta manera, se detectó que el quiebre de stock es causado en un 40.38% por parte de los proveedores, generando pérdidas económicas tanto para ellos como para los consumidores. En consecuencia, se desarrolló un modelo de revisión periódica mejorada aplicable a la logística de la empresa Sodimac. Usada esta metodología, se logró comprobar que los productos elegidos que producen quiebres de stock pueden ser manejados bajo un sistema de revisión periódica, contando con un stock de seguridad y obteniendo mayores beneficios para la empresa. Como resultado, se obtuvo una disminución en los costos de inventario y los costos por ventas perdidas para los productos seleccionados, obteniéndose una disminución anual del 61% respecto al modelo anterior (Santos y Alarcón, 2015).

Es necesario tener en cuenta que, en la actualidad, las operaciones logísticas están fuertemente vinculadas al área de la informática. En una empresa de logística, se detectó que existía la necesidad de agilizar los procesos logísticos en las bodegas, buscando llevar procesos manuales al campo de la automatización informática. Respecto a este proceso, se detectó un lento manejo de los productos de las bodegas al transcribirlos en forma digital a un sistema de bases de datos. La solución que se empleó fue el desarrollo de un prototipo emisor/receptor de datos vía radiofrecuencia para el control e ingreso de productos a bodega, utilizando un microcontrolador y

un lector de códigos de barras en los cuales se recopile información e ingrese la mercadería de manera remota, permitiendo optimizar el proceso. Como resultado se desarrolló una base de datos junto con una interfaz amigable para el usuario, ayudando al control de productos e ingreso de estos mismos (Pozo y Macancela, 2011).

Además de los beneficios anteriormente mencionados respecto a las bases de datos, hoy en día existen herramientas de inteligencia de negocios que permiten obtener ventajas competitivas en las empresas de retail. A modo de ejemplo, una empresa de electrodomésticos, perteneciente a un mercado altamente competitivo, implementó la inteligencia de negocios con el objetivo de transformar la información de las bases de datos en conocimiento, para obtener una ventaja competitiva en el mundo del retail. Como resultado, se detectaron ciertas condiciones que favorecen la venta de productos. Así, existe una relación sólida entre la venta de colchones y sus diversos accesorios, con un porcentaje de confianza superior al 95%. Además, la compra de artículos de mayor costo tiende a impulsar la venta de sus respectivos accesorios o complementos, siempre que el precio de estos últimos sea menor o igual al 25% del artículo principal (Báez *et al.*, 2018).

5. Soluciones propuestas

En base a la investigación realizada previamente respecto a la manera en la que se han solucionado problemáticas similares en otros contextos, surgen 3 potenciales ideas de solución para el presente proyecto:

1. Desarrollar un sistema de revisión periódica de los productos añadidos a RPC por parte de la tarea Product Aggregator. Así, resulta trascendental llevar a cabo un control diario de las inyecciones de determinados productos, pues se reduciría el riesgo de que las operaciones logísticas de IKEA se vean afectadas. Para esto, se debería desarrollar un software que se encargue de realizar un reporte con los problemas generados al intentar agregar un producto, el cual debe evidenciar la falla de forma amena para el usuario.
2. Desarrollar un software que permita el ingreso de productos a la base de datos de RPC mediante lectores QR. Para esto, sería necesario invertir en el hardware solicitado, además de vincular estas herramientas al software desarrollado. La finalidad de esta solución es que se expanda a todos los sectores en donde se lleve a cabo el manejo de productos. De esta forma, si un producto no es encontrado dentro de RPC en el momento en que se esté llevando a cabo determinada operación, el funcionario podrá ingresar inmediatamente los productos a RPC.
3. Desarrollar una interfaz amigable para los funcionarios, sobre la cual estos mismos puedan ingresar uno o más productos, y también revisar sus estados. De esta forma, ya no sería necesario la participación de IT dentro del proceso de las inyecciones, y, asimismo, se podrían realizar controles sobre los productos para verificar si es que fueron añadidos correctamente, y en caso de que no, tener la capacidad de visualizar el error, para posteriormente modificarlos y agregarlos correctamente.

La principal ventaja de la primera solución consiste en la detección temprana de problemas. Sin embargo, queda descartada por los costos asociados que puede implicar mantener operativa una tarea diaria de esta categoría.

Respecto a la segunda, si bien presenta ventajas de precisión, velocidad y eficiencia en el ingreso de productos, se descarta puesto que requeriría invertir tanto en hardware como en el entrenamiento para los funcionarios. Además, esto último puede generar resistencia al cambio por parte de los usuarios.

De esta forma, la solución seleccionada corresponde a la tercera, la cual se basa en el desarrollo e implementación de un software amigable, el cual permitirá a los trabajadores de IKEA mediante la interfaz de IOMC, realizar inyecciones, actualizaciones y revisiones de productos en RPC. Las razones por la cual se escoge esta solución son diversas: presenta bajos costos al utilizarse solo cuando se necesita ingresar productos, y no periódicamente, en comparación con la primera; se implementaría como una herramienta adicional en el software que ya utilizan los funcionarios, evitando que se genere resistencia al cambio por parte de estos; no requiere de inversión en hardware adicional, en comparación con la segunda solución; le otorga autonomía a los funcionarios, sin requerir al área de IT; permite la detección inmediata de los errores y la actualización inmediata de los productos.

Para cumplir con este cometido, se desarrollará un frontend que permitirá a los usuarios ver el estado de los productos, ingresarlos al sistema y, además, verificar si es que hubo algún error al realizar la acción de ingreso. El principal software que se utilizará para el desarrollo de frontend corresponde a Next.js, un software construido sobre React.js. Sus principales ventajas, y el motivo por el cual se seleccionó para este proyecto, son su rápido y directo renderizado respecto al servidor sobre el sitio web, además de su capacidad para implementar un server side, donde es posible desarrollar peticiones de APIs de manera sencilla. Los lenguajes de programación que emplea este framework son javascript y typescript.

Por otra parte, también se debe de llevar a cabo el desarrollo de un backend que permita establecer una comunicación efectiva entre el frontend y los sistemas requeridos, tales como Product

Aggregator y RPC. Respecto al primero, los endpoints deben de ser capaces de recopilar y gestionar los registros de acceso, recopilando eventos y errores. Sobre el segundo, debe ser capaz de llevar a cabo las solicitudes de inyección de productos provenientes del frontend, verificando el estado de la solicitud, si es que se llevó a cabo, o si se presentaron errores al enviar la solicitud. Los endpoints salientes del frontend se desarrollarán a través del framework de Next.js, puesto que esta herramienta tiene la capacidad de poner a disposición los endpoints desde el mismo frontend. Por otra parte, para el desarrollo del backend se utilizará el framework de Spring Boot con el lenguaje de programación java.

Finalmente, es necesario incrustar el software desarrollado dentro de la interfaz de IOMC, para que corresponda a una funcionalidad más de dicho sitio web, y los operarios puedan así acceder a dicha herramienta a través de esta interfaz. Para esto, se llevarán a cabo adaptaciones del software desarrollado, para posteriormente agregarlo a IOMC, el cual opera mediante el framework Angular.

Para la implementación de la solución, se utilizará una metodología ágil con enfoque custom. En otras palabras, las prácticas empleadas durante el desarrollo de la solución se irán definiendo sobre la marcha. Esta práctica se emplea con la finalidad de sacar un MVP lo más rápido posible, obtener feedback por parte de los usuarios y llevar a cabo mejoras.

Otro factor importante a tener en cuenta respecto a las herramientas y frameworks que se utilizarán, es que algunas de estas son frameworks que son impartidos por la empresa, tal como es el caso de Angular. Por ende, el resto de las herramientas que se usan están preseleccionadas bajo este criterio.

6. Evaluación Económica

Para la evaluación económica del proyecto, se tomó en cuenta lo siguiente:

- Se seleccionó una tasa de descuento = 9,5%, correspondiente a la tasa de interés según el Banco Central de Chile, la cual fue actualizada el 06 de septiembre del presente año (Alonso, 2023).
- Respecto a los costos, solo se tomaron en cuenta el saldo de un practicante mensual, equivalente a \$220.000 pesos chilenos, y el sueldo mensual de un programador senior promedio en Chile, equivalente a \$1.400.000 pesos chilenos aproximadamente. Por otro lado, las licencias, frameworks y herramientas adicionales, no fueron tomadas en cuenta para este análisis, puesto que vienen incluidas dentro del presupuesto de la empresa y no se requiere un gasto adicional para este proyecto.
- Se estima que el estudio tendrá una duración de 5 meses, puesto que es la fecha establecida para el proyecto.

Períodos		Agosto	Septiembre	Octubre	Noviembre	Diciembre
Costos fijos mensuales		\$-1.620.000	\$-1.620.000	\$-1.620.000	\$-1.620.000	\$-1.620.000
VAN	\$-6.220.328,23					
Tasa descuento	9,5%					
Inversión total	\$ -8.100.000					

Tabla 1. Evaluación económica del proyecto.

En cuanto a la evaluación realizada, en la Tabla 1 es posible evidenciar que llevar a cabo este proyecto requeriría de una inversión inicial de \$8.100.000 pesos chilenos, obteniéndose un VAN negativo con un valor de \$6.220.328 pesos chilenos. Ahora bien, no es posible medir el impacto real de este proyecto desde esta perspectiva, puesto que, al solo existir costos, el VAN siempre resultará negativo.

Ahora bien, para dar una idea del impacto que genera el proyecto, se calculó el valor promedio por cada contenedor importado, el cual equivale a \$18.000.000 pesos chilenos. Esta estimación se realizó considerando un contenedor con un tamaño de 60 metros cúbicos, el cual puede ser importado de diversos continentes, como Europa, Asia, y América del Norte. Cada container puede

contener 30 pallets, y cada pallet puede contener 20 unidades en promedio, en donde cada unidad se valoriza en \$20.000 pesos chilenos en promedio. De esta forma, se obtiene la cifra total por cada contenedor, previamente mencionada. Considerando que hasta a la fecha, existen 9.824 productos en Chile que aún no son añadidos correctamente a RPC, se calcula una pérdida de \$294.720.000 pesos chilenos en productos que no pueden ser utilizados.

De esta forma, si añadimos el estudio anterior a la evaluación económica, se obtiene lo siguiente:

Períodos		Agosto	Septiembre	Octubre	Noviembre	Diciembre
Costos fijos mensuales		\$-1.620.000	\$-1.620.000	\$-1.620.000	\$-1.620.000	\$-1.620.000
Inversión total	\$ -8.100.000					
Tasa de descuento	9,5%					
Valor total productos sin ingresar	\$ 294.720.000					
VAN	\$288.499.672					

Tabla 2. Evaluación económica actualizada del proyecto.

En la Tabla 2, es posible notar como mejora considerablemente el VAN al considerar la suma total de todos los productos que no han podido ser ingresados correctamente en Chile.

Adicionalmente, se realizó una evaluación de riesgos, analizando cuales son los riesgos de realizar el proyecto, y los riesgos de no realizarlo:

- Riesgos de realizar el proyecto: Posible inversión adicional en caso de que el proyecto no se complete en la fecha estimada, o la pérdida de la inversión y horas hombre de los programadores en caso de no poder completarse.
- Riesgos de no realizar el proyecto: Dependencia manual de la inyección de productos, gastos de horas hombre en IT, posibles multas en aduana, retrasos en remunerar a proveedores, inhabilitación de operaciones logísticas de productos, ralentización en procesos de órdenes de compra y de venta, reducción de la capacidad de trabajo del área IT, inexistencias de productos en RPC sobre los cuales se aplican operaciones, entre otras situaciones derivadas de las mencionadas.

El riesgo de no implementar el proyecto implica continuar realizando de manera manual una tarea que necesariamente debería estar automatizada, especialmente considerando todas las operaciones que dependen de esta.

7. Metodología

El proyecto surge como consecuencia de una profunda investigación respecto a la manera en que funcionaban las operaciones de logística de IKEA asociadas al área de IT. De esta forma, a través de charlas y entrevistas, fue posible detectar la necesidad de automatizar el proceso de ingreso de productos para dejar de realizarlo de forma manual. En base a que el proyecto se fue estructurando tras la detección del problema, y buscando un desarrollo rápido, se optó por una metodología ágil, empleando tickets de Jira para ir añadiendo componentes y funcionalidades según las necesidades presentadas. Además, esta metodología es óptima para producir un MVP lo antes posible, y modificarlo mientras se va testeando a la vez. Se hará uso de una carta Gantt para organizar y estructurar el paso a paso del proyecto.

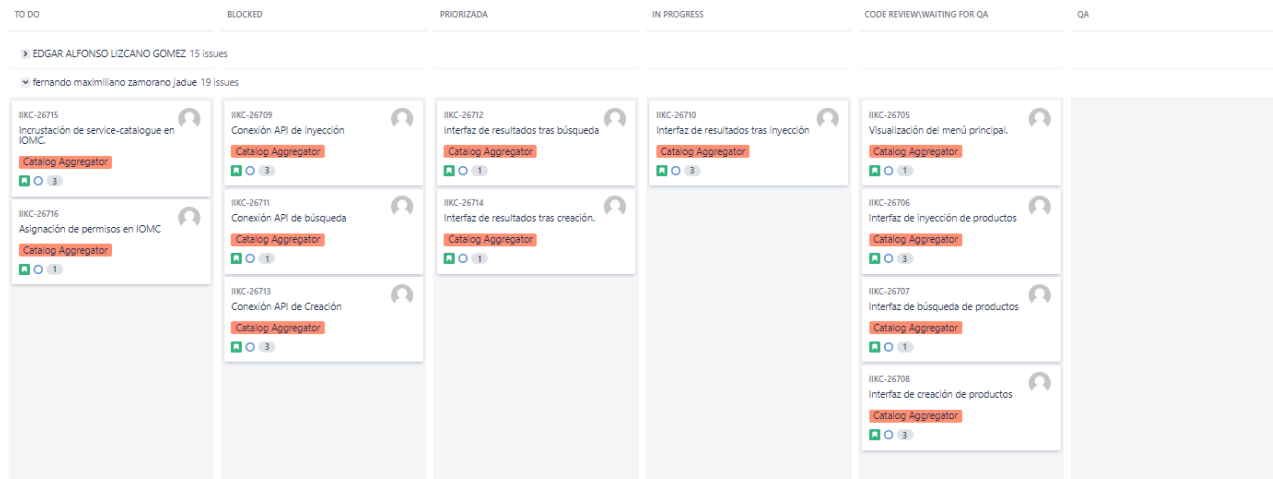


Figura 2. Proyecto estructurado en base a metodología ágil.

8. Medidas de desempeño

El éxito del proyecto se basará en las siguientes medidas de desempeño:

1. **Cantidad de errores al ingresar los productos:** Desde la fecha 01/10/23 hasta el 14/11/23, se detectaron 84.956 logs de acceso con errores, es decir, intentos fallidos de ingresar productos. Se busca que, tras la implementación del proyecto, esta cantidad se reduzca al menos en un 80%.
2. **Cantidad de productos sin ingresar:** Aproximadamente en IKEA Chile, hay 9824 productos vigentes que no han sido ingresados exitosamente en la base de datos de RPC, es decir, que contienen errores. Se busca que, tras la implementación, esta cifra no aumente considerablemente, y a lo largo de un semestre, disminuya considerablemente.
3. **Tasa de Éxito de Inyecciones:** Calcula el porcentaje de productos ingresados con éxito sin requerir la atención de IT.
4. **Capacidad de IT:** Se registrará la cantidad de horas hombre destinadas por parte de IT a la inyección de productos tras la implementación de la herramienta. Actualmente, el 20% del área de IT realiza tareas relacionadas con el ingreso de productos, destinando 8 horas semanales por persona a este suceso.

9. Desarrollo del proyecto basado en la metodología, implementación.

9.1 Desarrollo:

Para empezar, es importante destacar que, por petición de la empresa, se añadió al desarrollo del proyecto el ingreso de productos a la base de datos “Service-Catalog”. En resumen, los productos de la empresa IKEA se dividen en 2 categorías: los que pueden ser almacenados en bodega y los que no. Los productos almacenables se registran en la base de datos de RPC, mientras que los productos no almacenables, como cubiertas de cocina, comida, recetas, servicios y otros, se almacenan en la base de datos “Service-Catalog”. De esta forma, el backend hacia el cual apuntan todas las APIs y que realiza todas las verificaciones internas, se denomina “Catalog-Aggregator”. Así, la herramienta denominada “Product-Aggregator” pasó a ser parte de “Catalog-Aggregator”, el cual tiene la capacidad de ingresar productos tanto en RPC como en “Service-Catalog”. La elaboración de la arquitectura de la solución se evidencia en las Figuras 3 y 4.

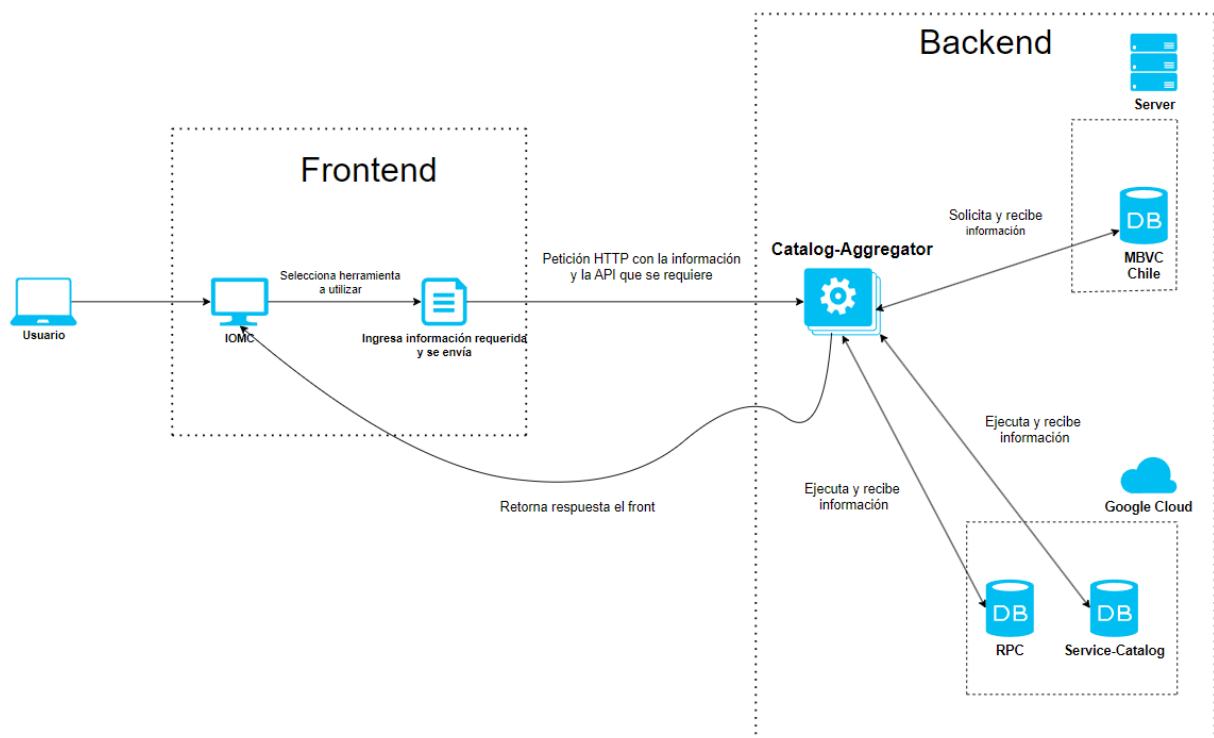


Figura 3. Diagrama de solución.

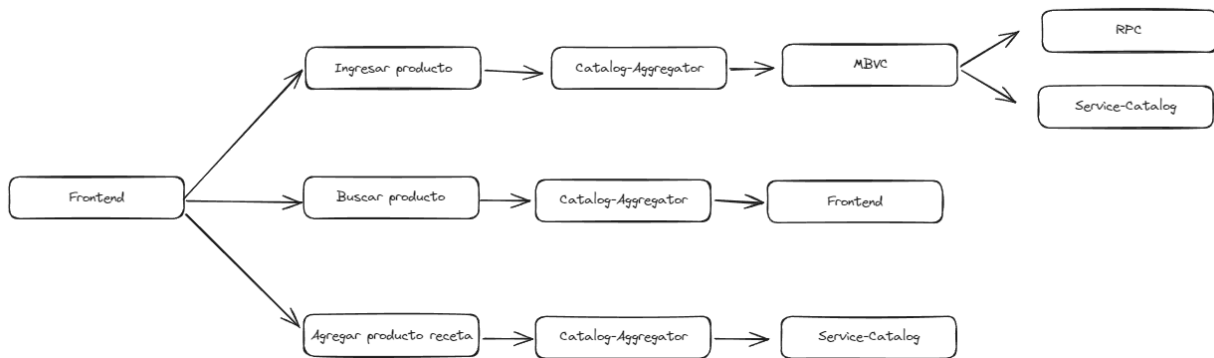


Figura 4. Diagrama de flujo.

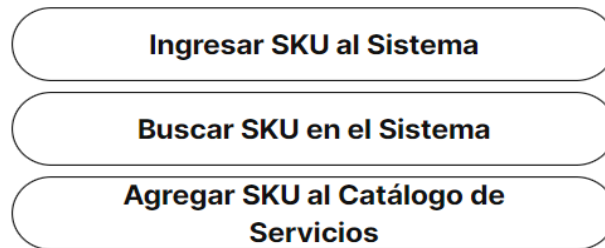
Para cumplir con el propósito, se desarrolló un frontend que permite a los usuarios visualizar el estado de los productos, ingresar nuevos productos y verificar la presencia de errores durante estas acciones. El principal framework utilizado para el desarrollo del frontend es Next.js, construido sobre React.js. Se eligió Next.js debido a su rápido rendimiento de renderizado en el servidor web y su capacidad para implementar fácilmente solicitudes de API en el lado del servidor. Los lenguajes de programación empleados en este framework son JavaScript y TypeScript. Adicionalmente, se emplearon las herramientas de Tailwind CSS y CSS, siendo la primera empleada para implementar los estilos de IKEA utilizados en IOMC, y la segunda para la estructuración de los modales desarrollados en Next Js.

En paralelo, se llevó a cabo el desarrollo de un backend para facilitar la comunicación efectiva entre el frontend y los sistemas necesarios, como el Catalog-Aggregator y RPC. En relación con el primero, los endpoints recopilan y gestionan registros de acceso, eventos y errores. En cuanto al segundo, el backend maneja las solicitudes de inyección de productos desde el frontend, verificando el estado de la solicitud y manejando posibles errores. Los endpoints del backend se implementarán en el frontend en el lado del servidor. Para el desarrollo del backend, se está utilizando el framework Spring Boot con el lenguaje de programación Java.

Para desplegar el código, se utiliza Ubuntu para poder emplear Linux, y mediante este, ejecutar NodeJs y NpmJs (Node Package Manager), y así desplegar el código en un localhost. Además, se emplea Git para poder trabajar con repositorios de Gitlab.

A continuación, se muestra el funcionamiento del frontend:

Seleccione una acción

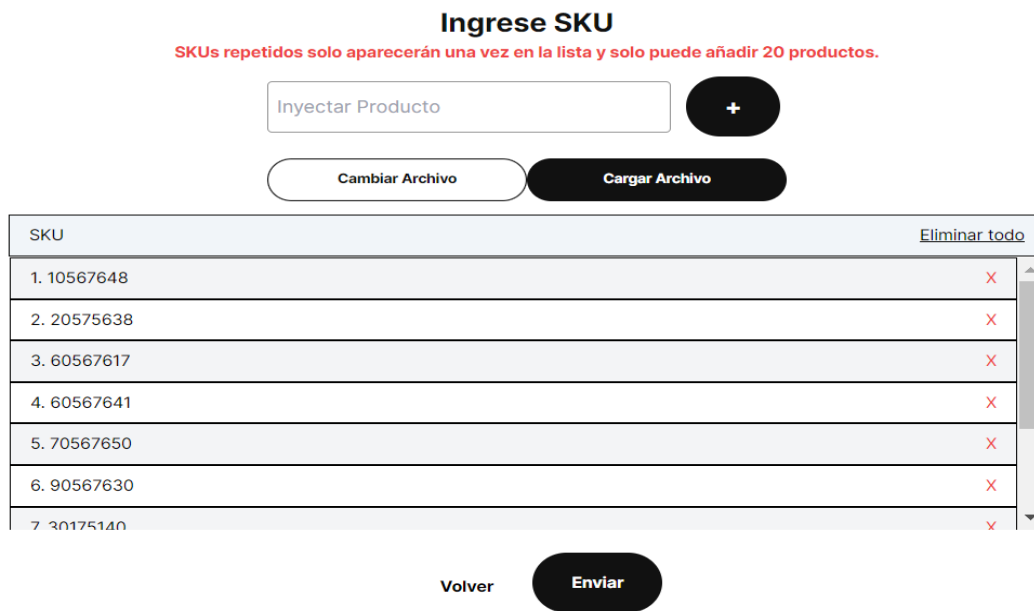


Three buttons are displayed vertically, each with a rounded rectangular shape and a thin border. The first button contains the text "Ingresar SKU al Sistema". The second button contains the text "Buscar SKU en el Sistema". The third button contains the text "Agregar SKU al Catálogo de Servicios".

Figura 5. Interfaz inicial.

Inicialmente se encuentra el menú principal, desde donde el usuario tiene las opciones de “Ingresar SKU al Sistema”, “Buscar SKU en el Sistema” y “Agregar SKU al Catálogo de Servicios”.

1. Cuando se seleccione la primera opción, se tendrá acceso a la siguiente interfaz:



The interface is titled "Ingreso SKU". Below the title is a red warning message: "SKUs repetidos solo aparecerán una vez en la lista y solo puede añadir 20 productos." Below this is a form with a text input labeled "Inyectar Producto" and a black button with a white "+" sign. Below the form are two buttons: "Cambiar Archivo" and "Cargar Archivo". Below these buttons is a table with a header row "SKU" and a button "Eliminar todo" in the top right corner. The table contains seven rows of product data, each with a red "X" in the rightmost column. Below the table are two buttons: "Volver" and "Enviar".

SKU	
1. 10567648	X
2. 20575638	X
3. 60567617	X
4. 60567641	X
5. 70567650	X
6. 90567630	X
7. 30175140	X

Figura 6. Interfaz de ingreso de productos.

Desde esta pantalla, es posible añadir productos de forma manual o mediante un archivo Excel, puesto que es el formato con el que se suelen manejar estos elementos. Además, es posible subir más de un archivo, en donde, una vez cargados, se pueden añadir productos hasta cumplir con las limitaciones. Respecto a estas últimas, se implementaron 2 restricciones para prevenir el envío de datos erróneos al backend. La primera, es que solo se pueden añadir 20 elementos en total, puesto que, si se excede este valor, las solicitudes enviadas a los endpoints se tornan muy pesadas, ralentizando el proceso. La segunda, es que los SKUs que ya se ingresaron previamente a la lista, no se volverán a agregar, ya que enviar valores duplicados también afecta el comportamiento del backend. Continuando con las opciones, el usuario tiene la posibilidad de eliminar un SKU en particular, o eliminarlos todos si así lo desea.

Es importante destacar que, para el proceso de inyección de productos, se necesita el funcionamiento de 3 endpoints. El primero corresponde a un método POST denominado “execute”. El segundo también es un método POST, denominado “multiget”. Finalmente, el tercero corresponde a un método GET, denominado “GET service”.

Cuando el usuario seleccione “Enviar”, se mandará la lista de elementos mediante un array hacia el servidor. Una vez recibido el arreglo, se ejecutará el endpoint de “execute”, el cual comienza el proceso de la inyección, y de forma inmediata, responderá con el código 201, informando que ya se ha enviado una solicitud para ingresar los productos. Así, Catalog-Aggregator es capaz de detectar si un producto corresponde a RPC o Service-Catalog, por lo que el lugar al que se ingrese el elemento depende de las características de cada SKU. Mientras esto se realiza, el usuario no podrá manipular la lista de los productos, es decir, no podrá añadir ni eliminar elementos (Véase el anexo, figuras 28, 29 y 30).

Ingrese SKU

SKUs repetidos solo aparecerán una vez en la lista y solo puede añadir 20 productos.

+

Seleccionar archivo

Ninguno archivo selec.

SKU	Eliminar todo
1. 00223878	X
2. 00295530	X
3. 00257463	X
4. 789000003	X
5. 00017133	X
6. 00028508	X
7.	X

Volver

•

Figura 7. Visualización de “execute” en ejecución.

Mientras se espera la respuesta, el botón de enviar pasará a ser un botón de carga, y una vez que se tenga respuesta, se abrirá el siguiente modal:

X

¡Sus productos se han ingresado correctamente!

Si desea visualizar su estado, porfavor espere 30 segundos.

•

Figura 8. Modal de espera mientras se ingresan los productos.

En este punto, el endpoint de “execute” tiene un comportamiento asíncrono. La creación de una solicitud para el ingreso de productos es inmediata, por lo que la API responde de forma inmediata con el código 201. Sin embargo, el proceso de ingreso de productos en RPC y Service-Catalog funciona de forma asíncrona, puesto que el algoritmo debe de revisar en MBVC si los productos existen o no, y posteriormente, añadirlos donde corresponda. Se calculó que 20 productos tardan 30 segundos aproximadamente en ser ingresados. Por esto, el modal que aparece tras enviar los elementos cuenta con un contador de 30 segundos antes de permitir al usuario revisar el estado de los productos. Transcurrido este periodo, se ejecutarán los endpoints de búsqueda, “multiget” y “GET service”. El primero, obtiene el registro de las inyecciones en RPC. El segundo, obtiene el registro de las inyecciones en Service-Catalog. Ambos endpoints operan mediante Catalog-Aggregator para llevar a cabo sus funciones (Véase el anexo, figuras 32, 33 y 34). Una vez realizado todo lo anterior, el usuario podrá ver la siguiente interfaz:



Figura 9. Modal mediante el cual se visualizan los resultados del multiget.

Al presionar el botón de “Revisar”, se podrán ver los resultados de la inyección de la siguiente forma:

Resultados			
Id	SKU	Ubicación	Estado
1	789000003	Service-Catalog	Q
2	00017133	RPC	Q
3	87654321	Service-Catalog	Q
4	00028508	RPC	Q
5	12345678	-	!
6	12312312	-	!
7	00030262	RPC	Q
8	11223344	-	!
9	00081534	RPC	Q
10	00089163	RPC	Q
11	87878799	-	!

Volver

Descargar
Resultados

Figura 10. Interfaz de resultados.

Así, el usuario puede verificar el estado de los productos tras la inyección presionando el botón del triángulo rojo, el cual abre un modal con 3 casos posibles:

X

Número SKU: 00223878

ApiInternalServerErrorException()

Figura 11. Modal con errores de RPC.

X

Número SKU: 789000003

Producto encontrado

Figura 12. Modal encontrado en service.

X

Número SKU: 00257463

El producto no se encontró, es probable que no se encuentre
ingresado en MBVC

Figura 13. Modal con producto no encontrado.

En el caso de los productos encontrados en RPC, es necesario tener presente que se encuentran en el siguiente formato:

```
[
  {
    "classificationId": "string",
    "classificationName": "string",
    "commerce": "string",
    "country": "string",
    "createdAt": "string",
    "dateDiscontinuedProduct": "string",
    "displayName": "string",
    "displayNameInLocalLanguage": "string",
    "errors": [
      "string"
    ],
    "hscode": "string",
    "id": "string",
    "itemComplements": [ ]
  }
]
```

Figura 14. Formato de respuesta de los productos en RPC.

De esta respuesta, se seleccionarán los elementos de “number”, correspondiendo este al número de SKU, y “errors”, correspondiendo a los errores que puede presentar un SKU. Para el manejo de errores, se pueden presentar 3 casos:

1. El producto no contiene errores, es decir, “errors” tiene el valor de NULL. Esto implica que el producto fue ingresado correctamente.
2. El producto contiene un error, es decir, “errors” contiene uno o más elementos tipo string. Esto implica que el producto no pudo ser añadido de forma correcta, y el usuario puede revisar el porqué.
3. No se retorna nada, lo que implica que el producto no existe en RPC.

Finalmente, tras encontrar los resultados, el usuario puede descargar el reporte en un archivo Excel, el cual, en base al ejemplo anterior, contiene la siguiente información:

	A	B	C	D
1	Id	SKU	Ubicación	Estado
2		1 789000003	Service-Catalog	Producto encontrado en Servicios
3		2 00017133	RPC	Producto ingresado correctamente en RPC
4		3 87654321	Service-Catalog	Producto encontrado en Servicios
5		4 00028508	RPC	Producto ingresado correctamente en RPC
6		5 12345678	-	No encontrado
7		6 12312312	-	No encontrado
8		7 00030262	RPC	Producto ingresado correctamente en RPC
9		8 11223344	-	No encontrado
10		9 00081534	RPC	Producto ingresado correctamente en RPC
11		10 00089163	RPC	Producto ingresado correctamente en RPC
12		11 87878799	-	No encontrado
13		12 00257646	RPC	ApiInternalServerErrorException()
14		13 77777777	-	No encontrado
15		14 88888888	-	No encontrado
16		15 00130167	RPC	Producto ingresado correctamente en RPC
17		16 00267706	RPC	ApiInternalServerErrorException()
18		17 00091415	RPC	Producto ingresado correctamente en RPC
19		18 00069768	RPC	Producto ingresado correctamente en RPC
20		19 00132798	RPC	Producto ingresado correctamente en RPC
21				

Figura 15. Excel generado a partir de los resultados.

2. Cuando se ejecute la opción de “Buscar un SKU en el Sistema”, se mostrará en pantalla la siguiente interfaz:

Buscar SKUs

SKUs repetidos solo aparecerán una vez en la lista y solo puede ingresar hasta 100 SKUs

+

Cambiar Archivo

Cargar Archivo

SKU	Eliminar todo
24. 60580548	X
25. 80580547	X
26. 40580554	X
27. 00580546	X
28. 70580538	X
29. 60580567	X

Volver

Buscar

Figura 16. Interfaz de búsqueda.


Esta interfaz es exactamente igual a la de inyección, con la diferencia de que permite hasta 100 elementos. Al presionar el botón de “Buscar”, se enviarán los datos al servidor y se ejecutarán los mismos endpoints que se utilizaban para buscar los resultados tras una inyección, “multiget” y “GET service” (Véase anexo, figura 31 y 32). Una vez ejecutados, se muestra el siguiente modal:



Figura 17. Formulario de creación de productos.

El cual, al presionar el botón de “revisar”, redirige a resultados, como se mostró con anterioridad (Figura 10).

3.- Se ejecuta la opción de “Agregar un SKU al Catálogo de Servicios”, mostrándose la siguiente interfaz (Véase anexo, figuras 35, 36 y 37):



Ingresa los valores

Nº

Search Description

Description 2

Product Area No.

Nombre botón categoría POS

Unit Price Including VAT

Enviar

Volver

The image shows a web form titled "Ingresa los valores" (Enter the values). It contains six text input fields, each with a label to its left: "Nº", "Search Description", "Description 2", "Product Area No.", "Nombre botón categoría POS", and "Unit Price Including VAT". Below the input fields is a large, light gray button labeled "Enviar" (Send). At the bottom center of the form, there is a small link labeled "Volver" (Return).

Figura 18. Formulario de creación de productos.

Se empleó la librería react-hook-form para crear el formulario presentado en la interfaz anterior, debido a su facilidad para limitar los inputs del formulario según se requiera. Desde esta interfaz, el usuario podrá crear productos para el catálogo de servicios. La diferencia entre ingresar un producto, y crear un producto, se basa en la composición de cada uno. En el área de “food”, existen productos unitarios, como, por ejemplo, arroz y pollo asado. Sin embargo, estos productos no suelen venderse de forma unitaria, sino que se venden como productos receta, en formato de arroz con pollo asado. En base a esta necesidad, se habilita esta opción en la interfaz.

Continuando con el funcionamiento, para que el usuario pueda crear un producto receta, debe de completar todos los campos necesarios, para así poder habilitar el botón de “Enviar”. Una vez hecho esto, se recibe el formulario desde el servidor.

En consecuencia, se ejecuta el endpoint POST “service”, con un método post, el cual responde con un estado 201 en caso de ser llevado a cabo correctamente. Una vez que se recibe este estado, se muestra el siguiente modal:

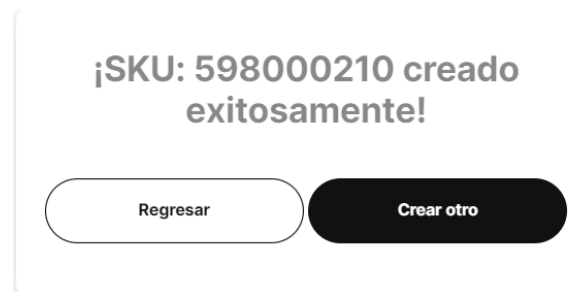


Figura 19. Respuesta exitosa al crear un producto.

De esta forma, es posible regresar al menú principal, o crear otro producto, el cual redirige al formulario en blanco.

Adicionalmente, en la interfaz de resultados, los productos que corresponden al tipo receta traen la información del producto creado, como se puede ver a continuación en la Figura 20:

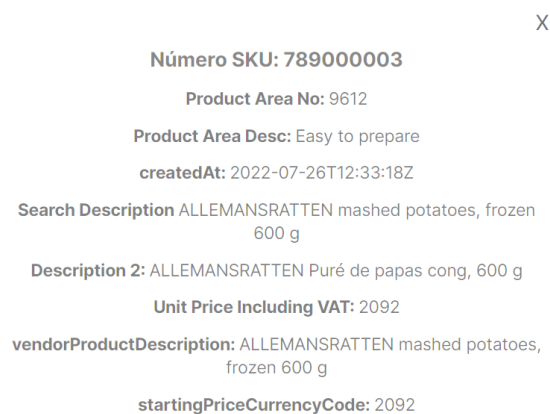


Figura 20. Información de un producto receta.

Finalmente, es posible notar el registro de los logs de acceso en el proyecto Catalog-Aggregator ubicado en Google Cloud Platform (GCP), el cual tiene todas las peticiones enviadas desde el frontend mediante las APIs.

Set up an automated pipeline for this workload SET UP DISMISS

OVERVIEWDETAILSOBSERVABILITYREVISION HISTORYEVENTSLOGSAPP ERRORSYAML

Suggested filters (0)

Container logs Severity: Default Filter Search all fields and values

SEVERITY	TIMESTAMP	SUMMARY
>	2023-12-01 12:56:04.760 CLST	Sku list [00500533, 00500934, 00500534, 00500571, 00500544, 00500529, 00500572, 40500568, 00500570, 10500536, 00500565, 00500558, 10500937, 00500551, 20500569, 30500540, 10500555, 10500541, ...
>	2023-12-01 13:07:16.148 CLST	Sku list [00500533, 00500934, 00500534, 00500571, 00500544, 00500529, 00500572, 40500568, 00500570, 10500536, 00500565, 00500558, 10500937, 00500551, 20500569, 30500540, 10500555, 10500541, ...
>	2023-12-01 13:17:47.890 CLST	Sku list [12312312] not found
>	2023-12-01 13:18:37.292 CLST	Sku list [12312312] not found
>	2023-12-01 13:18:46.515 CLST	Sku list [12312312] not found
>	2023-12-01 13:21:03.494 CLST	Sku list [12312312] not found
>	2023-12-01 13:21:13.264 CLST	Sku list [12312312] not found
>	2023-12-01 13:23:22.955 CLST	Sku list [12312312] not found
>	2023-12-01 13:24:20.664 CLST	Sku list [12312312] not found
>	2023-12-01 13:25:22.280 CLST	Sku list [12312312] not found
>	2023-12-01 13:31:26.752 CLST	Searching item info by sku 789000003
>	2023-12-01 13:31:26.989 CLST	Searching item info by sku 00257463
>	2023-12-01 13:31:26.995 CLST	Sku 00257463 not found
>	2023-12-01 13:31:27.026 CLST	Searching item info by sku 77777777
>	2023-12-01 13:31:27.833 CLST	Sku 77777777 not found
>	2023-12-01 13:31:27.063 CLST	Searching item info by sku 88888888
>	2023-12-01 13:31:27.069 CLST	Sku 88888888 not found
>	2023-12-01 16:00:01.799 CLST	Item service saved successfully ItemService{id=1151, number=12345678, vendorDescription=test, displayName=test, displayNameLocal=null, startingPrice=5000.0, currencyCode=CLP, createdAt=2023-...

No newer entries found matching current filter.

Figura 21. Logs de acceso en GCP Catalog-Aggregator.

9.2 Implementación

Para la correcta integración del software, es necesario añadirlo en la interfaz de IOMC para que funcione como una característica adicional del sitio web.



Figura 22. Interfaz de IOMC con Catalog-Aggregator implementado.

Como se puede ver, el botón presente en la navbar, permite a los usuarios acceder a la interfaz de Catalog-Aggregator. Este componente está desarrollado en el framework de Angular, el cual es empleado en IOMC. Para realizar la adaptación del frontend de NextJs con el de Angular de IOMC, se insertará mediante un `<iframe>`, por lo que una vez que se acceda a Catalog-Aggregator, será posible ver la página HTML de NextJs incrustada en la página HTML de IOMC. Para llevar esto a cabo, se desplegará la interfaz desarrollada en una URL determinada, para luego importarla en el iframe.

10. Resultados cualitativos y cuantitativos.

- Resultados cualitativos:

La interfaz del software y su funcionamiento ha tenido buena aceptación por parte del equipo de IKEA. Mediante entrevistas, se les enseñó un demo a las personas que en un futuro tendrán acceso a la herramienta, teniendo esta buena aceptación por parte de los clientes. Se obtuvieron diversas opiniones sobre oportunidades de mejora, sin embargo, todas estas se pueden llevar a cabo trabajando sobre la misma herramienta, sin necesidad de cambiarla y siempre buscando la mayor comodidad para el usuario.

- Resultados cuantitativos:

Antes de presentar los resultados, se debe tener en cuenta que la herramienta durante el mes de diciembre, solo fue testeada en QA (quality assurance), y no en producción. Esto se debió a que la empresa durante este mes pasó por diversas situaciones, como rotación de personal, capacitaciones y adicionalmente, llevó a cabo una migración de todos sus proyectos en GCP (Google Cloud Platform). Por ende, la integración de la herramienta en producción se pospuso hasta fines de diciembre.

En base a las medidas de desempeño, se realizaron las siguientes estimaciones para mediados de febrero:



Figura 23. Gráfico de cantidad de errores registrados al ingresar productos.

En la figura 23, es posible notar la estimación que se espera para la cantidad de errores generados al intentar ingresar productos a mediados de febrero, reduciéndose en un 80%. Lo anterior se realiza en base a que, actualmente, cuando se ingresa un producto, se presentan de 5 a 6 errores en promedio al intentar ingresarlo. Con la nueva herramienta, se espera que no se presenten más de 1 a 2 errores, reduciendo la tasa de 84.956 logs de acceso erróneos en 1/5, obteniéndose así 16.991 logs. Durante el mes de diciembre, en el ambiente de QA (quality assurance), ha sido posible ingresar productos correctamente en su segunda iteración, generando un solo log de acceso con errores por producto y de esta forma, respaldando lo planteado.



Figura 24. Gráfico con la cantidad de horas antes y después de la implementación del proyecto.

En la figura 24, se aprecia la cantidad de horas semanales destinadas por cada integrante de IT a la inyección de productos, comparando el antes y después de la implementación. Esta aproximación está estructurada sobre simulaciones realizadas en QA durante el mes de diciembre, en donde se realizó el proceso completo de la inyección de productos, registrando tanto el personal como las horas destinadas por parte del equipo de IT para completar el proceso, obteniéndose una reducción de 8 a 2 horas semanales.

Finalmente, se espera que la cifra de 9.824 productos que aún no se logran ingresar correctamente, no aumente durante los meses de enero y febrero. Esta estimación se realiza en base a que, con la implementación de la nueva herramienta, y al obtener una menor cantidad de errores en los logs de acceso, no se debería extender el ingreso de productos, disminuyendo la brecha de productos sin ingresar. Ahora bien, es necesario tener en cuenta que depende de los usuarios llevar a cabo el procedimiento anterior, por lo que esta estimación puede verse afectada según el comportamiento de los funcionarios.

11. Conclusiones y discusión.

Como se puede apreciar en los resultados, la implementación del proyecto genera diversos beneficios para la empresa, reduciendo riesgos, horas de trabajo y abriendo la posibilidad de recuperar los bienes, tanto económicos como de inventario, de los 9.824 productos de Chile que aún no han sido ingresados correctamente. Además, el software tuvo una rápida aceptación por parte de los usuarios, facilitando su implementación. Actualmente, el software se está empleando en QA, lo cual demuestra el interés de la empresa en comenzar a utilizarlo en un futuro cercano en el área de producción.

Asimismo, existe aún gran oportunidad de mejora, puesto que el software fue desarrollado como un MVP, existiendo muchos procesos que se pueden automatizar o facilitar aún más para el usuario. Es trascendental continuar realizando mediciones respecto a las medidas de desempeño, al menos durante 6 meses más, para poder medir su funcionamiento a largo plazo. También se sugiere que, en un futuro, se implemente el software para países como Colombia.

Por otro lado, se sugiere implementar en QA todas las características que existen en producción, con la finalidad de poder realizar una simulación y testeo de mayor calidad. Lo anterior se plantea considerando que los productos que existen en QA no son los mismos que existen en producción. Por esto, existen pocos productos con los cuales realizar pruebas, a diferencia de la gran cantidad y variedad de productos que existen en producción. Además, también se le recomienda a la empresa destinar más personas al proyecto, con la finalidad de mejorar el código. Por ejemplo, realizar un refactor para estructurar adecuadamente el código; aplicar medidas de seguridad, tales como prohibir ciertos caracteres en los campos de entrada; restringir permisos a usuarios, limitar el uso de la herramienta, manejar errores y tener un mayor feedback a futuro por parte de los usuarios.

12. Referencias

1. Alonso, C. (05 de septiembre, 2023). Banco Central deja la tasa de interés en 9,5% y se consolida visión de que a fines de año TPM se ubicará entre 7,75% y 8%. *Diario La Tercera*. Recuperado de: <https://www.latercera.com/pulso/noticia/banco-central-se-apega-al-mercado-y-baja-la-tasa-de-interes-en-75-puntos-base/KA4G4FZBUFAOTFTH6DS6KRE5ZI/#:~:text=Y%20el%20ente%20rector%20cu mpli%C3%B3,nivel%20desde%20julio%20de%202022>.
2. Báez, J., Paredes, C., Sosa, G. y García, M. (2018). *Descubriendo reglas de asociación en bases de datos del sector retail usando R*. XXIV Congreso Argentino de Ciencias de la Computación, Universidad Nacional de Asunción, San Lorenzo, Paraguay. Recuperado de: http://sedici.unlp.edu.ar/bitstream/handle/10915/73220/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y
3. Pozo, D. y Macancela, A. (2011). *Diseño y desarrollo de un prototipo emisor / transmisor de datos vía radiofrecuencia para control e ingreso de productos de bodega, utilizando microcontroladores, lector de códigos de barra y base de datos*. [Tesis para título de Ingeniería en Sistemas, Universidad Politécnica Salesiana]. Recuperado de: <https://dspace.ups.edu.ec/bitstream/123456789/1484/7/UPS%20-%20ST000880.pdf>
4. Santos, F. y Alarcón, R. (2015). *Propuesta de Mejora del Modelo de Inventario, para Aumentar el Nivel de Servicio de Productos que Presentan Quiebres de Stock en Sodimac S.A, por Incumplimiento en las Condiciones de Entrega de los Proveedores*. [Tesis para título de Ingeniería Civil Industrial, Universidad de Valparaíso]. Recuperado de: <http://dspace.opengeek.cl/bitstream/handle/uvsc/733/Santos%20Yunis%2c%20Francisco%20-%20Alarco%cc%81n%20Rivera%2c%20Richard.pdf?sequence=1&isAllowed=y>
5. Medina, S. (2018). *ESTUDIO DE LAS ESTRATEGIAS DE E-COMMERCE DESDE EL CASO DE LAS EMPRESAS MULTINACIONALES COMO EL GRUPO ÉXITO, FALABELLA, DAFITI, AMAZON, WALMART Y ZARA*. [Tesis de grado]. Universidad Agustiniana, Bogotá. Recuperado de <https://repositorio.uniagustiniana.edu.co/handle/123456789/362>
6. Ley 21131 de 2019. Pago a 30 días. [Ministerio de economía, fomento y turismo]. 3 de enero de 2019. Recuperado de: <https://www.bcn.cl/leychile/navegar?idNorma=1127890>

13. Anexos

13.1. Multa de Demurrage: 100 a 150 dólares por contenedor diarios.

<https://www.empresaoceano.cl/desde-cuando-el-demurrage-se-volvio-una-constante-en-el-transporte#:~:text=Cabe%20se%20B1alar%20que%20estas%20multas,de%20internaci%C3%B3n%20evitar%20estos%20cobros>

13.2. Figuras

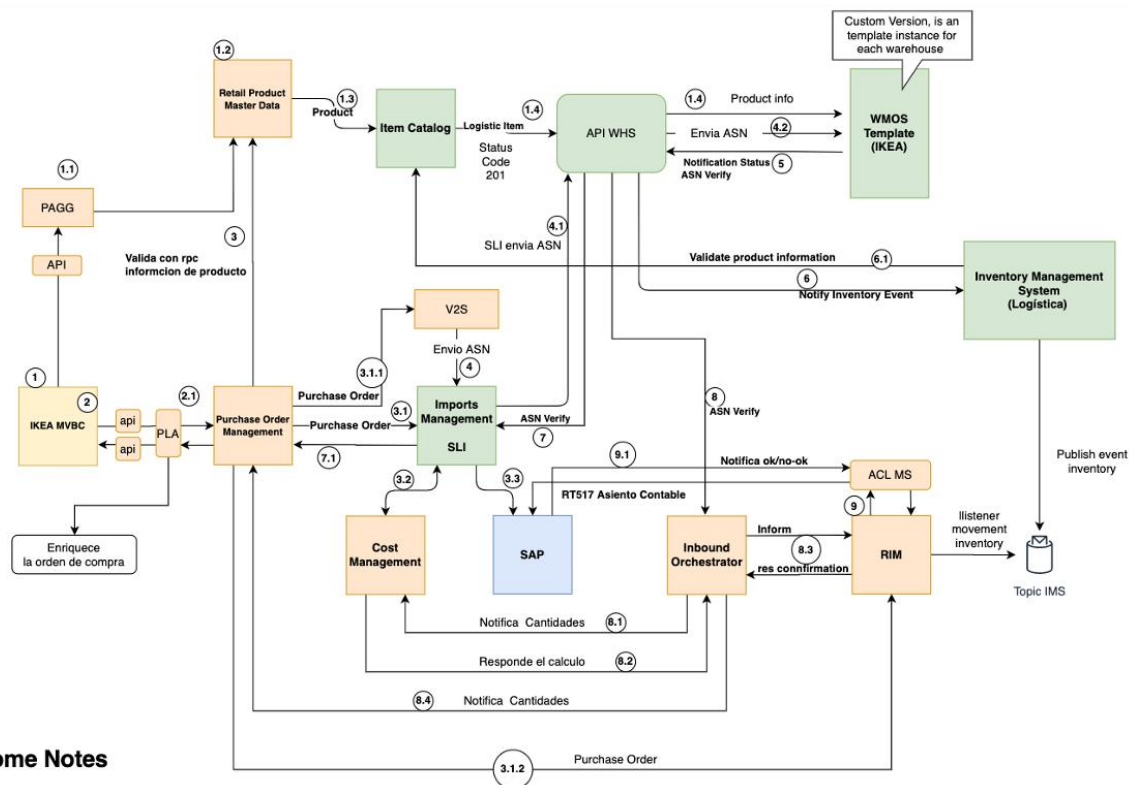


Figura 25. Esquema del funcionamiento logístico del proceso descrito.

13.3. Código server-side

Token de verificación:

```
//token
const apiUrl = 'https://api-qa.ikso-tech.com/catalog-aggregator/oauth2/token';
const fetchData = async () => {
  const formData = new FormData();
  formData.append('client_id', process.env.CLIENT_ID ?? '');
  formData.append('client_secret', process.env.CLIENT_SECRET ?? '');
  formData.append('grant_type', process.env.GRANT_TYPE ?? '');

  try {
    const response = await fetch(apiUrl, {
      method: 'POST',
      body: formData,
    });
    if (response.ok) {
      const data = await response.json();
      return data;
    }
  } catch (error) {}
  console.error('Error:', error);
  return null;
};

const responseData = await fetchData();
```

Figura 26. Método de obtención del token de seguridad.

Hook:

```
import { useEffect, useReducer } from 'react';
const userReducer = (state: { userEmail: string | null, country: string | null }, action: any) => {
  switch (action.type) {
    case 'SET_USER_EMAIL':
      return { ...state, userEmail: action.payload };
    case 'SET_COUNTRY':
      return { ...state, country: action.payload };

    default:
      return state;
  }
}

const useUserInformation = () => {
  const [userInformation, dispatch] = useReducer(userReducer, { userEmail: null, country: null });

  useEffect(() => {
    const handleMessage = (event: { data: { userEmail: string, country: string } }) => {
      if (event.data.userEmail) {
        sessionStorage.setItem('userEmail', event.data.userEmail);
        sessionStorage.setItem('country', event.data.country);
        dispatch({ type: 'SET_USER_EMAIL', payload: event.data.userEmail });
        dispatch({ type: 'SET_COUNTRY', payload: event.data.country });
      }
    };

    const userEmail = sessionStorage.getItem('userEmail');
    const country = sessionStorage.getItem('country');

    if (userEmail && country) {
      handleMessage({ data: { userEmail, country } });
    }
    window.addEventListener('message', handleMessage);
    return () => {
      window.removeEventListener('message', handleMessage);
    };
  }, []);

  return [userInformation, dispatch] as const;
};

export default useUserInformation;
```

Figura 27. Obtención del correo y país del usuario.

Inyección:

```
const postSku = async (tasksArray: any) => {
  setAbleInput(false);
  setIsLoading(true);
  const token = await SkuService.getTokenService();
  const result = await SkuService.postSku(tasksArray, token, userInfo);
  if(result === 201){
    setTemporalResponse(true);
    setIsLoading(false);

    await new Promise(resolve => setTimeout(resolve, 30000))
    setmodalCounter(true);
    const secondToken = await SkuService.getTokenService();
    const searchResults = await SkuService.searchSku(tasksArray, secondToken, userInfo);
    setSearchResult(searchResults);
    setAbleInput(true);
  }
}
```

Figura 28. Llamado a postSku y envío del arreglo.

```
postSku: async (formData: any, token: any, userInfo: any) => {
  const apiRoute = '/api/injection';
  const body = {formData, token, userInfo}
  const data = await fetch(apiRoute, {
    method: "POST",
    body: JSON.stringify(body)
  });
  return await data.json();
}}
```

Figura 29. Llamado a la ruta de inyección en lado del servidor.

```

import { NextRequest, NextResponse } from "next/server";
export async function POST(req: NextRequest) {

  const firstresponseJson = await req.json();
  const token = firstresponseJson.token.access_token;
  const formData = firstresponseJson.formData;
  const userInfo = firstresponseJson.userInfo;
  try{
    const API_BASE_PATH= process.env.API_BASE_PATH;
    let url = `${API_BASE_PATH}/catalog-aggregator/v1/execute/single`;
    const headers = {
      'x-user': userInfo.userEmail,
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json',
    }
    const body = {
      "classification": [
        "0",
        "1",
        "2",
        "3",
        "4",
        "5",
        "6",
        "7",
        "8",
        "9"
      ],
      "country": userInfo.country,
      "commerce": "IKS",
      "filterType": "SINGLE",
      "sku": formData,

      "serviceClassificationIds": [
        "96",
        "97"
      ],
      "serviceFullClassificationIds": [
        "1853",
        "0741",
        "0742"
      ]
    }

    const response = await fetch(url, {method: 'POST', body: JSON.stringify(body), headers: headers})
    if (response.status===201){
      return NextResponse.json(201);
    }

    if (!response.ok){
      const data = await response.json();
      return NextResponse.json({ 'first error': data?.detail }, { status: response.status, statusText: data?.detail });
    }
  }
}

```

Figura 30. Envío de información a la API de inyección y manejo de la respuesta.

Buscar:

```
export default function Home({tasksArray,setTasksArray,setShowComponent, setSearchResult, userInfo}:any) {
  const [showLoadingModal, setShowLoadingModal] = useState(false);
  const [ableCheckButton, setAbleCheckButton] = useState(false);
  const [warningMessage, setWarningMessage] = useState(false);
  const searchSku = async (tasksArray: any) => {
    const token = await SkuService.getTokenService();
    setShowLoadingModal(true);
    const result = await SkuService.searchSku(tasksArray, token, userInfo);
    setSearchResult(result);
    setAbleCheckButton(true);
  }
}
```

Figura 31. Llamado a searchSku y envío del arreglo.

```
24 searchSku: async (formData: [], token:any, userInfo: any) => {
25   const apiRouteRPC = '/api/searchingRPC';
26   const body = {"sku": formData, token: token, userInfo};
27   const dataFromRPC = await fetch(apiRouteRPC, {
28     method: "POST",
29     body: JSON.stringify(body),
30   });
31   });
32
33   const rpcToJson = await dataFromRPC.json();
34   const skuInRpc = rpcToJson.map((item: any) => {
35     {
36       number: item.number,
37       errors: item.errors,
38       zone: 'RPC'
39     }
40   })
41
42   const skuNotInRpc: number[] = [];
43   formData.forEach(item => {
44     if (!(skuInRpc.map((item: any) => item.number)).includes(item)) {
45       skuNotInRpc.push(item);
46     }
47   });
48
49   const skuInService = [];
50   const skuNotFound = [];
51   if(skuNotInRpc.length !== 0){
52     for(let i = 0; i < skuNotInRpc.length; i++){
53       const apiRouteService = `/api/searchingService?sku=${skuNotInRpc[i]}&token=${token.access_token}&userCountry=${userInfo.country}`;
54       const dataFromService = await fetch(apiRouteService, {
55         method: "GET",
56       });
57       const serviceResponse = await dataFromService.json();
58       if(serviceResponse.error === undefined){
59         skuInService.push({
60           number: skuNotInRpc[i],
61           object: serviceResponse,
62           zone: 'Service',
63         });
64       }
65       }else{
66         skuNotFound.push(skuNotInRpc[i]);
67       }
68     }
69   });
70
71   const data = {
72     RPC: skuInRpc,
73     Service: skuInService,
74     Missing: skuNotFound
75   }
76   return data;
77 }
```

Figura 32. Llamado a las rutas de búsqueda en el servidor.

```

//searchingRPC
const firstresponseJson = await req.json();
const API_BASE_PATH= process.env.API_BASE_PATH;

let url = `${API_BASE_PATH}/catalog-aggregator/v1/multiget`;
const headers = {
  'x-country': userInfo.country,
  'x-commerce': 'IKS',
  'Content-Type': 'application/json',
  'Authorization': `Bearer ${responseData.access_token}`
};

try{
const response = await fetch(url, {method: 'POST', body: JSON.stringify(firstresponseJson), headers: headers})
if (!response.ok){
  return NextResponse.json([]);
}

const data = await response.json();
return NextResponse.json(data);
} catch (e: any) {
return NextResponse.json({ 'error': e.message });
}
}

```

Figura 33. Envío de información a la API de búsqueda de RPC y manejo de la respuesta.

```

const responseData = await fetchData();
const [userInfo] = useUserInformation();

const API_BASE_PATH= process.env.API_BASE_PATH;
let url = `${API_BASE_PATH}/catalog-aggregator/v1/service/${request.nextUrl.searchParams.get('sku')}`;

const headers = {
  'x-country': userInfo.country ,
  'Authorization': `Bearer ${responseData.access_token}`
};

const res = await fetch(url, { method: "GET", headers });
try {
  if (!res.ok) {
    const errorData = await res.json();
    throw new Error(`Error ${res.status}: ${res.statusText}. ${JSON.stringify(errorData)}`);
  }

  const data = await res.json();
  return NextResponse.json(data);
} catch (e: any) {
  return NextResponse.json({ 'error': e.message });
}
}

```

Figura 34. Envío de información a la API de búsqueda de Servicios y manejo de la respuesta.

Crear:

```
export default function Home({userInfo}: any) {
  const { register, handleSubmit, formState, reset } = useForm();
  const [showModal, setShowModal] = useState(false);
  const [skuNumber, setSkuNumber] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const createSku = async (formData: any) => {
    formData.startingPriceValue = parseInt(formData.startingPriceValue, 10);
    setIsLoading(true);
    const token = await SkuService.getTokenService();

    const result = await SkuService.createSku(formData, token, userInfo);
    if(result===201){
      setShowModal(true);
    }
    setIsLoading(false);
  };

  const onSubmit = (data:any) => {
    setSkuNumber(data.number)
    createSku(data);
  };

  const resetForm = () => {
    setShowModal(false);
    reset();
  };
}
```

Figura 35. Llamado a createSku y envío del formulario.


```

createSku: async (formData: any, token: any, userInfo: any) => {
  const apiRoute = '/api/creation';
  const body = {formData, token, userInfo}
  const data = await fetch(apiRoute, {
    method: "POST",
    body: JSON.stringify(body),
  });
  return await data.json();
},

```

Figura 36. Llamado a la ruta de creación en lado del servidor.

```

import { NextRequest, NextResponse } from "next/server";
export async function POST(req: NextRequest) {

  const firstresponseJson = await req.json();
  const token = firstresponseJson.token.access_token;
  const formData = firstresponseJson.formData;
  const userInfo = firstresponseJson.userInfo;
  const currencyCode = userInfo.country === 'CL' ? 'CLP' : userInfo.country === 'CO' ? 'COP' : '';
  try{
    const API_BASE_PATH= process.env.API_BASE_PATH;
    let url = `${API_BASE_PATH}/catalog-aggregator/v1/service`;
    const headers = {
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json',
    }

    const body = {
      "number": formData.number,
      "displayName": formData.displayName,
      "displayNameInLocallanguage": formData.displayNameInLocallanguage,
      "classificationId": formData.classificationId,
      "classificationName": formData.classificationName,
      "startingPriceValue": formData.startingPriceValue,
      "currencyCode": currencyCode,
      "country": userInfo.country,
      "commerce": 'IKS'
    }

    const response = await fetch(url, {method: 'POST', body: JSON.stringify(body), headers: headers})
    if (response.status===201){
      return NextResponse.json(201);
    }

    if (!response.ok){
      const data = await response.json();
      return NextResponse.json({ error: data?.detail }, { status: response.status, statusText: data?.detail });
    }
    const data = await response.json();

    return NextResponse.json(data);
  } catch (e: any) {
  }
}

```

Figura 37. Envío de información a la API de creación y manejo de la respuesta.

13.4. Código client-side

```
1 import React, { useState } from "react";
2 import Modal from '../../app/injectSku/Modals/Modal';
3 import * as XLSX from 'xlsx';
4
5 export default function Home({ tasksArray, showComponent, setShowComponent, searchResult }: any) {
6   const [showModal, setShowModal] = useState(false);
7   const [skuNumber, setSkuNumber] = useState('');
8   const [message, setMessage] = useState('');
9
10  const exportToExcel = () => {
11    const headers = ['Id', 'SKU', 'Ubicación', 'Estado'];
12    const data = [headers, ...tasksArray.map((task: string, index: number) => {
13      const rpcMatch = searchResult.RPC.find((rpcItem: any) => rpcItem.number === task);
14      const serviceMatch = searchResult.Service.find((serviceItem: any) => serviceItem.number === task);
15      const location = rpcMatch ? 'RPC' : (serviceMatch ? 'Service-Catalog' : '-');
16      const message = rpcMatch
17        ? rpcMatch.errors !== null
18        ? rpcMatch.errors
19        : ['Producto ingresado correctamente en RPC']
20      : serviceMatch
21        ? ['Producto encontrado en Servicios']
22        : ['No encontrado'];
23
24      return [index + 1, task, location, message ? message.join(', ') : ''];
25    })];
26
27    const ws = XLSX.utils.aoa_to_sheet(data);
28    const wb = XLSX.utils.book_new();
29    XLSX.utils.book_append_sheet(wb, ws, 'Resultados');
30    XLSX.writeFile(wb, 'resultados.xlsx');
31  };
32 }
```

Figura 38. Armado y descarga del Excel en resultados.

```

3
4 return (
5   <main className="flex flex-col justify-center items-center bg-white w-full h-screen">
6     {!showModal && <h1 className="text-2xl font-bold mb-4">Resultados</h1>}
7     {!showModal && (
8       <div className="flex w-2/4 h-3/4">
9         <div className="w-full p-4">
10           <div className="flex bg-gray-200 p-2 border border-black">
11             <div className="w-1/4 text-center items-center">Id</div>
12             <div className="w-1/4 text-center items-center">SKU</div>
13             <div className="w-1/4 text-center items-center">Ubicación</div>
14             <div className="w-1/4 text-center items-center">Estado</div>
15           </div>
16           <div className="overflow-y-auto border border-black h-4/5">
17             {tasksArray.map((task: string, index: number) => {
18               const rpcMatch = searchResult.RPC.find((rpcItem: any) => rpcItem.number === task);
19               const serviceMatch = searchResult.Service.find((serviceItem: any) => serviceItem.number === task);
20               const location = rpcMatch ? 'RPC' : (serviceMatch ? 'Service-Catalog' : '-');
21               const message = rpcMatch
22                 ? rpcMatch.errors !== null
23                 ? rpcMatch.errors
24                 : ['Producto ingresado correctamente en RPC']
25               : serviceMatch
26               ? [
27                 <span key="classificationId">
28                   <span className="font-bold">Product Area No: </span>
29                   {`${serviceMatch.object.classificationId}`}
30                 </span>,
31                 <span key="classificationName">
32                   <span className="font-bold">Product Area Desc: </span>
33                   {`${serviceMatch.object.classificationName}`}
34                 </span>,
35                 <span key="createdAt">
36                   <span className="font-bold">createdAt: </span>
37                   {`${serviceMatch.object.createdAt}`}
38                 </span>,
39                 <span key="displayName">
40                   <span className="font-bold">Search Description </span>
41                   {`${serviceMatch.object.displayName}`}
42                 </span>,
43                 <span key="displayNameInLocalLanguage">
44                   <span className="font-bold">Description 2: </span>
45                   {`${serviceMatch.object.displayNameInLocalLanguage}`}
46                 </span>,
47                 <span key="startingPriceValue">
48                   <span className="font-bold">Unit Price Including VAT: </span>
49                   {`${serviceMatch.object.startingPriceValue}`}
50                 </span>,
51                 <span key="vendorProductDescription">
52                   <span className="font-bold">vendorProductDescription: </span>
53                   {`${serviceMatch.object.vendorProductDescription}`}
54                 </span>,
55                 <span key="startingPriceCurrencyCode">
56                   <span className="font-bold">startingPriceCurrencyCode: </span>
57                   {`${serviceMatch.object.startingPriceValue}`}
58                 </span>,
59               ]
60               : null;
61             })}
62

```

Figura 39. Interfaz de resultados.

```

return (
  <div
    key={index}
    className={flex ${{
      index % 2 === 0 ? "bg-gray-200" : "bg-gray-100"
    }} p-2 text-center justify-center items-center`}
  >
    <div className="w-1/4">{index + 1}</div>
    <div className="w-1/4">{task}</div>
    <div className="w-1/4">{location}</div>
    <div className="w-1/4">
      <button
        onClick={() => {
          setSkunumber(task);
          setMessage(message);
          setShowModal(true);
        }}
        type="button"
        className="btn btn--small leading-icon btn--danger"
      >
        {{{rpcMatch && rpcMatch.errors===null}}}{message===null}} && <svg className="svg-icon btn__icon w-3 h-3">
          <path fill="#FF0000" d="M11.0836 14.1337h2v-6h-2v6zm1 1.5c-.6904 0-1.25-.5597-1.25 1.25 0 .6904.5596 1.25 1.25 1.25.6903 0 1.25-.5596 1.25-1.25 0-.6903-.5597-1.25-1.25-1.25z"/></path>
          <path fill="#FF0000" fillRule="evenodd" clipRule="evenodd" d="m1.1152 20.418 10-19h1.7699l10 19-.8849 1.4657zm2.5413-.5343h16.6873l12.0002 4.0388 3.6565 19.8837z"/></path>
        </svg>
      </button>
      {{{rpcMatch && rpcMatch.errors===null}}}{serviceMatch}} && <svg focusable="false" viewBox="0 0 24 24" width="24" height="24" className="svg-icon" aria-hidden="true">
        <path fill="black" fillRu
      </button>
    </div>
  </div>
);
}}
</div>
</div>
}
div>
{!showComponent && !showModal && (
  <div className="flex justify-end mb-5">
    <button onClick={() => setShowComponent(true)} type="button" className="w-1/2 btn btn--terciary text-lg mt-3"><span className="btn__inner"><span className="btn__label">Volver</span></span></button>
    <button onClick={exportToExcel} type="button" className="mt-3 w-1/2 btn btn--emphasised text-lg"><span className="btn__inner"><span className="btn__label">Descargar Resultados</span></span></button>
  </div>
)
}
/ div>
div>
{showModal && (
  <Modal setShowModal={setShowModal} skunumber={skunumber} message={message} />
)}
</div>
</div>
</div>

```

Figura 40. Interfaz de resultados.

```

7 components / services / responses / modals / injectionModal / 40 ExecutionResponse / Home
1 'use client'
2 import './executionResponse'
3
4
5 export default function Home({setTemporalResponse, setShowComponent, setmodalCounter, modalCounter}: any) {
6
7
8   return (
9     <div className="fixed inset-0 flex items-center justify-center z-50">
10       <div className="modalBackground fixed inset-0 bg-gray-800 opacity-50"></div>
11       <div className="modalContainer bg-white w-96 p-6 rounded-lg shadow-lg">
12         <div className="titleCloseBtn flex justify-end">
13           <button
14             className="text-gray-500 hover:text-gray-700 focus:outline-none"
15             onClick={() => {
16               setTemporalResponse(false);
17               setmodalCounter(false);
18             }}
19           >
20             X
21           </button>
22         </div>
23         <div className="title mb-4">
24           <h1 className="text-2xl font-bold">¡Sus productos se han ingresado correctamente!</h1>
25         </div>
26         <div className="title mb-4">
27           <h1 className="text-2xl font-bold">Si desea visualizar su estado, porfavor espere 30 segundos.</h1>
28         </div>
29
30         <div className="body mb-6">
31           <button onClick={() => {{setShowComponent(false); setTemporalResponse(false); setmodalCounter(false)}}}
32             type="button"
33             className={`w-1/2 btn ${!modalCounter ? 'btn btn--loading btn--primary' : 'btn btn--primary'} text-lg`}
34             disabled={!modalCounter}
35           >
36             <span className="btn__inner">
37               <span className="btn__label">{!modalCounter ? 'Revisar' : 'Revisar'}</span>
38               {!modalCounter && <span className="btn__loader"></span>}
39             </span>
40           </button>
41         </div>
42       </div>
43     </div>
44   );
45 }
46
47

```

Figura 41. Interfaz del modal de inyección.

```

1 'use client'
2 import './searchingResponse.css';
3
4
5 export default function Home({setShowLoadingModal, setShowComponent, setAbleCheckButton, ableCheckButton}: any) {
6   return (
7     <div className="fixed inset-0 flex items-center justify-center z-50">
8       <div className="modalBackground fixed inset-0 bg-gray-800 opacity-50"></div>
9       <div className="modalContainer bg-white w-96 p-6 rounded-lg shadow-lg">
10         <div className="titleCloseBtn flex justify-end">
11           <button
12             className="text-gray-500 hover:text-gray-700 focus:outline-none"
13             onClick={() => {
14               setShowLoadingModal(false);
15               setAbleCheckButton(false);
16             }}
17           >
18             X
19           </button>
20         </div>
21         <div className="title mb-4">
22           <h1 className="text-2xl font-bold">Estamos buscando sus productos</h1>
23         </div>
24         <div className="title mb-4">
25           <h1 className="text-2xl font-bold">Por favor, espere un momento</h1>
26         </div>
27
28
29         <div className="body mb-6">
30           <button onClick={() => {{setShowComponent(false); setShowLoadingModal(false); setAbleCheckButton(false)}}}
31             type="button"
32             className={`w-1/2 btn ${!ableCheckButton ? 'btn btn--loading btn--primary' : 'btn btn--primary'} text-lg`}
33             disabled={!ableCheckButton}
34           >
35             <span className="btn_inner">
36               <span className="btn_label">{!ableCheckButton ? 'Revisar' : 'Revisar'}</span>
37               {!ableCheckButton && <span className="btn_loader"></span>}
38             </span>
39           </button>
40         </div>
41       </div>
42     </div>
43   );
44 }
45

```

Figura 42. Interfaz del modal de búsqueda.

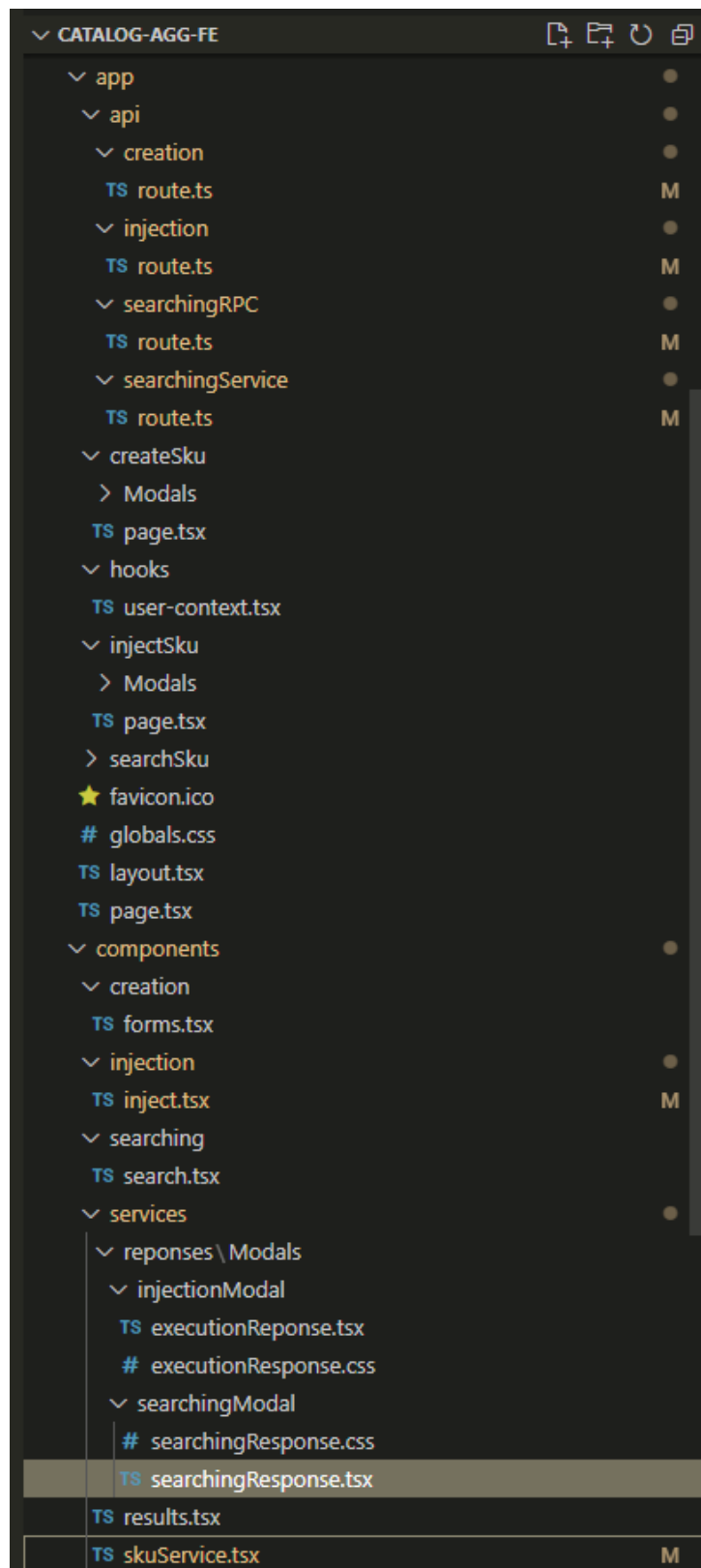


Figura 43. Estructuración del código.

```

const selectedFile = (e: React.ChangeEvent<HTMLInputElement>) => {
  const selectedFile = e.target.files?.[0];
  if (selectedFile) {
    setFile(selectedFile);
  }
};

const confirmedForm = (e: React.FormEvent) => {
  e.preventDefault();
  if (file) {
    const reader = new FileReader();
    reader.onload = (e) => {
      const fileData = e.target?.result as ArrayBuffer;
      const workbook = XLSX.read(fileData, { type: 'binary' });
      const firstSheetName = workbook.SheetNames[0];
      const jsonData: any[] = XLSX.utils.sheet_to_json(workbook.Sheets[firstSheetName]);
      const skus: any[] = jsonData.map(item => item.SKU);

      setTasksArray((prevTasksArray: any) => {
        let updatedTasksArray = [...prevTasksArray];
        if (ableInput) {
          for (let i = 0; i < skus.length; i++) {
            const skuToAdd = skus[i].toString();
            if (!updatedTasksArray.includes(skuToAdd) && updatedTasksArray.length < 20 && skuToAdd.length > 7) {
              updatedTasksArray.push(skuToAdd);
            }
          }
        }

        return updatedTasksArray;
      });
    };
    reader.readAsArrayBuffer(file);
  }
};

```

Figura 44. Subida de archivo para inyección.


```

const [file, setFile] = useState<File | null>(null);
const [task, setTask] = useState<string>('');
const inputChange = (e: any) => {
  setTask(e.target.value);
};

const inputSubmit = (e: any) => {
  e.preventDefault();
  if (task.trim() && !tasksArray.includes(task) && tasksArray.length<20 && ableInput && task.length>7) {
    setTasksArray([...tasksArray, task]);
    setTask('');
  }
  else{
    setTask('');
  }
};

const handleDelete = (index: any) => {
  setTasksArray(tasksArray.filter((_:any, i:any) => i !== index));
};

return (
  <main className="flex flex-col justify-center items-center bg-white min-h-screen w-1/2">
    {!temporalReponse && <div className="flex flex-col justify-center items-center bg-white w-full">
      <h1 className="text text--heading-m">Ingresa SKU</h1>
      <p className="text-red-500 font-bold mb-3">SKUs repetidos solo aparecerán una vez en la lista y solo puede añadir 20 productos.</p>
      <div className="flex flex-col w-1/2 items-center">
        <form onSubmit={inputSubmit} className="w-full flex">
          <div className="w-4/5">
            <div className="input-field label-wrapper--text-input mt-1"><div className="input-field_wrapper"><input type="text" value={task} onChange={inputChange} /></div>
            </div>
            <div className="w-1/5 ml-3">
              <button type="submit" className="btn btn--primary mb-5"><span className="btn_inner"><span className="btn_label text-xl">+</span></span></button>
            </div>
          </form>
        </div>
        <div className="w-full">
          <form className="w-full items-center text text-center" onSubmit={confirmedForm}>
            {!file && <input type="file" name="file" onChange={selectedFile} />}
            {file &&

```

Figura 45. Subida de archivos e interfaz de inyección.

```

ingResponse.tsx  TS inject.tsx  TS page.tsx ...app  TS page.tsx ...injectSku
ponents > injection > TS inject.tsx > Home > [0] postSku
{!file && <input type="file" name="file" onChange={selectedFile} />}
{file &&
  <div className="w-full flex"> <button className="w-1/2 btn btn--secondary btn--small btn--fluid" onClick={() => setFile(null)}><span className="btn__inner">
  <button className="w-1/2 btn btn--primary btn--small btn--fluid" type="submit"><span className="btn__inner"><span className="btn__label">Cargar Archivo
  </form>
</div>
</div>

{tasksArray.length !== 0 && <div className="w-full border border-white mt-3">
  {tasksArray.length !== 0 && (
    <div className="border border-black">
      <div className="flex">
        <div className="w-1/2 bg-slate-100 py-2 px-4 text-left">SKU</div>
        <div className="w-1/2 bg-slate-100 py-2 px-4 text-right">
          <button disabled={!tableInput} className="underline" onClick={() => setTasksArray([])}>
            Eliminar todo
          </button>
        </div>
      </div>
    </div>
  )}
  <div className="w-full h-64 border border-white overflow-y-auto">
    {tasksArray.map((task: any, index: any) => (
      <div
        key={index}
        className={`flex ${
          index % 2 === 0 ? 'bg-gray-100' : 'bg-white'
        } border border-black`}
      >
        <div className="w-1/2 py-2 px-4">
          {index + 1}. {task}
        </div>
        <div className="w-1/2 py-2 px-4 text-right">
          <button
            className="text-red-500 hover:text-red-700 focus:outline-none"
            disabled={!tableInput}
            onClick={() => handleDelete(index)}
          >
            X
          </button>
        </div>
      </div>
    ))}
  </div>
</div>

```

Figura 46. Interfaz de inyección.

```

3
4
5 {!temporalReponse && <div className="flex justify-end mt-5">
6   <a href="/" type="button" className="w-1/2 btn btn--terciary text-lg mt-1 ml-2"><span className="btn__inner"><span className="btn__label">Volver</span></span></a>
7
8 <button onClick={() => {{postSku(tasksArray)}}}
9   type="button"
10  className={`w-1/2 btn ${isLoading ? 'btn btn--loading btn--primary btn--fluid' : 'btn btn--primary'} text-lg`
11  disabled={isLoading || tasksArray.length === 0}
12  >
13    <span className="btn__inner">
14      <span className="btn__label">{isLoading ? 'Enviar' : 'Enviar'}</span>
15      {isLoading && <span className="btn__loader"></span>}
16    </span>
17  </button>
18
19 </div>
20 </div>
21
22 {temporalReponse && <ExecutionResponse setTemporalResponse={setTemporalResponse} setShowComponent={setShowComponent} setmodalCounter={setmodalCounter} modalC
23
24 </main>
25
26
27

```

Figura 47. Interfaz de inyección.

```

const selectedFile = (e: React.ChangeEvent<HTMLInputElement>) => {
  const selectedFile = e.target.files?.[0];
  if (selectedFile) {
    setFile(selectedFile);
  }
};

const confirmedForm = (e: React.FormEvent) => {
  e.preventDefault();
  if (file) {
    const reader = new FileReader();
    reader.onload = (e) => {
      const fileData = e.target?.result as ArrayBuffer;
      const workbook = XLSX.read(fileData, { type: 'binary' });
      const firstSheetName = workbook.SheetNames[0];
      const jsonData: any[] = XLSX.utils.sheet_to_json(workbook.Sheets[firstSheetName]);
      const skus: string[] = jsonData.map(item => item.SKU);

      setTasksArray((prevTasksArray: any) => {
        let updatedTasksArray = [...prevTasksArray];

        for (let i = 0; i < skus.length; i++) {
          const skuToAdd = skus[i].toString();
          if (!updatedTasksArray.includes(skuToAdd) && skuToAdd.length > 7 && updatedTasksArray.length < 100) {
            updatedTasksArray.push(skuToAdd);
          }
        }

        return updatedTasksArray;
      });
    };
    reader.readAsArrayBuffer(file);
  }
};

```

Figura 48. Subida de archivo para búsqueda.

```

const [file, setFile] = useState<File | null>(null);
const [task, setTask] = useState<string>('');
const inputChange = (e: any) => {
  setTask(e.target.value);
};

const inputSubmit = (e: any) => {
  e.preventDefault();
  if (task.trim() && !tasksArray.includes(task) && task.length>7 && tasksArray.length < 100) {
    setTasksArray([...tasksArray, task]);
    setTask('');
  }
  else{
    setTask('');
  }
};

const handleDelete = (index: any) => {
  setTasksArray(tasksArray.filter((_:any, i:any) => i !== index));
};

return (
  <main className="flex flex-col justify-center items-center bg-white min-h-screen w-1/2">
    {!showLoadingModal && <div className="flex flex-col justify-center items-center bg-white w-full">
      <h1 className="text text--heading-m">Buscar SKUs</h1>
      <p className="text-red-500 font-bold mb-3">SKUs repetidos solo aparecerán una vez en la lista y solo puede ingresar hasta 100 SKUs</p>
      <div className="flex flex-col w-1/2 items-center">
        <form onSubmit={inputSubmit} className="w-full flex">
          <div className="w-4/5">
            <div className="input-field label-wrapper--text-input mt-1"><div className="input-field__wrapper"><input type="text" value={task} onChange={inputChange} /></div>
            <div className="w-1/5 ml-3">
              <button type="submit" className="btn btn--primary mb-5"><span className="btn__inner"><span className="btn__label text-xl">+</span></span></button>
            </div>
          </div>
          <div className="w-full">
            <form className="w-full items-center text text-center" onSubmit={confirmedForm}>
              {!file && <input type="file" name="file" onChange={selectedFile} />}
              {file &&
                <div className="w-full flex"> <button className="w-1/2 btn btn--secondary btn--small btn--fluid" onClick={() => setFile(null)}><span className="btn__inner"><span className="btn__label">Cargar Archivo</span></span></button>
                <button className="w-1/2 btn btn--primary btn--small btn--fluid" type="submit"><span className="btn__inner"><span className="btn__label">Cargar Archivo</span></span></button>
              </div>
            </form>
          </div>
        </div>
      </div>
    )
  )

```

Figura 49. Interfaz de búsqueda.

```

{({tasksArray.length !== 0} && <div className="w-full border border-white mt-3">
  {tasksArray.length !== 0 && (
    <div className="border border-black">
      <div className="flex">
        <div className="w-1/2 bg-slate-100 py-2 px-4 text-left">SKU</div>
        <div className="w-1/2 bg-slate-100 py-2 px-4 text-right">
          <button className="underline" onClick={() => setTasksArray([])}>
            Eliminar todo
          </button>
        </div>
      </div>
    </div>
  )}
</div>
<div className="w-full h-64 border border-white overflow-y-auto">
  {tasksArray.map((task: any, index: any) => (
    <div
      key={index}
      className={`flex ${
        index % 2 === 0 ? 'bg-gray-100' : 'bg-white'
      } border border-black`}
    >
      <div className="w-1/2 py-2 px-4">
        {index + 1}. {task}
      </div>
      <div className="w-1/2 py-2 px-4 text-right">
        <button
          className="text-red-500 hover:text-red-700 focus:outline-none"
          onClick={() => handleDelete(index)}
        >
          X
        </button>
      </div>
    </div>
  ))}
</div>
</div>

{!showLoadingModal && <div className="flex justify-end mt-5">
  <a href="/" type="button" className="w-1/2 btn btn--terciary text-lg mt-1 ml-2"><span className="btn__inner"><span className="btn__label">Volver</span></span></a>

  <button onClick={() => {{searchSku(tasksArray)}}}
    type="button"
    className="w-1/2 btn 'btn btn--primary btn--fluid' : 'btn btn--primary'" text-lg"
    disabled={tasksArray.length === 0}
  >
    <span className="btn__inner">
      <span className="btn__label">Buscar</span>
      <span className="btn__loader"></span>
    </span>
  </button>
</div>
</div>

</main>
);
}

```

Figura 50. Interfaz de búsqueda.

```

3
4
5 {!showLoadingModal && <div className="flex justify-end mt-5">
6   <a href="/" type="button" className="w-1/2 btn btn--terciary text-lg mt-1 ml-2"><span className="btn__inner"><span className="btn__label">Volver</span></span></a>
7
8   <button onClick={() => {{searchSku(tasksArray)}}}
9     type="button"
10    className="w-1/2 btn 'btn btn--primary btn--fluid' : 'btn btn--primary'" text-lg"
11    disabled={tasksArray.length === 0}
12  >
13    <span className="btn__inner">
14      <span className="btn__label">Buscar</span>
15      <span className="btn__loader"></span>
16    </span>
17  </button>
18
19 </div>
20 </div>
21
22 {showLoadingModal && <SearchingResponse setShowLoadingModal={setShowLoadingModal} setShowComponent={setShowComponent} setAbleCheckButton={setAbleCheckButton}
23
24 </main>
25 );
26 }

```

Figura 51. Interfaz de búsqueda.

```

export default function Home() {
  const { register, handleSubmit, formState, reset } = useForm();
  const [showModal, setShowModal] = useState(false);
  const [skuNumber, setSkuNumber] = useState('');
  const [isLoading, setIsLoading] = useState(false);

  const createSku = async (formData: any) => {
    formData.startingPriceValue = parseInt(formData.startingPriceValue, 10);
    setIsLoading(true);
    const result = await SkuService.createSku(formData);
    if(result===201){
      setShowModal(true);
    }
    setIsLoading(false);
  };

  const onSubmit = (data:any) => {
    setSkuNumber(data.number)
    createSku(data);
  };

  const resetForm = () => {
    setShowModal(false);
    reset();
  };
}

```

Figura 52. Manejo del formulario de creación.

```

return (
  <main className="flex flex-col justify-center items-center bg-white h-1/2 w-2/3 ">
    {showModal && <div className="w-4/5 flex flex-col justify-center items-center rounded p-8 h-1/2 border border-black">
      <h2 className="text-2xl font-bold mb-6">Ingresa los valores</h2>
      <form className="w-full items-center" onSubmit={handleSubmit(onSubmit)}>
        <div className="form-field">
          <div className="input-field input-field--prefixed input-field--suffixed label-wrapper label-wrapper--text-input">
            <label className="text text--heading-xl">Nº</label> <div className="input-field__wrapper">
              <input {...register("number", { required: true })}className="w-full px-3 py-2 border rounded-md focus:outline-none focus:border-blue-500"/></div>
            </div>
          </div>

          <div className="form-field">
            <div className="input-field input-field--prefixed input-field--suffixed label-wrapper label-wrapper--text-input">
              <label className="text text--heading-xl">Search Description</label> <div className="input-field__wrapper">
                <input {...register("displayName", { required: true })}className="w-full px-3 py-2 border rounded-md focus:outline-none focus:border-blue-500"/></div>
              </div>
            </div>

            <div className="form-field">
              <div className="input-field input-field--prefixed input-field--suffixed label-wrapper label-wrapper--text-input">
                <label className="text text--heading-xl">Description 2</label> <div className="input-field__wrapper">
                  <input {...register("displayNameInLocalLanguage", { required: true })}className="w-full px-3 py-2 border rounded-md focus:outline-none focus:border-b">
                </div>
              </div>
            </div>

            <div className="form-field">
              <div className="input-field input-field--prefixed input-field--suffixed label-wrapper label-wrapper--text-input">
                <label className="text text--heading-xl">Product Area No.</label> <div className="input-field__wrapper">
                  <input {...register("clasificationId", { required: true })}className="w-full px-3 py-2 border rounded-md focus:outline-none focus:border-blue-500"/></div>
                </div>
              </div>

            <div className="form-field">
              <div className="input-field input-field--prefixed input-field--suffixed label-wrapper label-wrapper--text-input">
                <label className="text text--heading-xl">Nombre botón categoría POS</label> <div className="input-field__wrapper">
                  <input {...register("classificationName", { required: true })}className="w-full px-3 py-2 border rounded-md focus:outline-none focus:border-blue-500"/>
                </div>
              </div>

            <div className="form-field">
              <div className="input-field input-field--prefixed input-field--suffixed label-wrapper label-wrapper--text-input">
                <label className="text text--heading-xl">Unit Price Including VAT</label> <div className="input-field__wrapper">
                  <input {...register("startingPriceValue", { required: true })}className="w-full px-3 py-2 border rounded-md focus:outline-none focus:border-blue-500"/>
                </div>
              </div>
            </div>
          </form>
        </div>
      </div>
    </main>
  );
}

```

Figura 53. Interfaz de creación.

```

<div className="flex justify-center">
  <button
    disabled={!formState.isValid || isLoading}
    type="submit"
    value="Submit"
    className={`w-1/3 btn ${isLoading ? 'btn btn--loading btn--primary' : 'btn btn--primary'} text-lg`}
  >
    <span className="btn_inner">
      <span className="btn_label">{isLoading ? 'Enviar' : 'Enviar'}</span>
      {isLoading && <span className="btn_loader"></span>}
    </span>
  </button>
</div>
</form>
</div>
<div>
  {showModal && <Modal setShowModal={setShowModal} skuNumber={skuNumber} resetForm={resetForm}></Modal>}
</div>

</main>
);
}

```

Figura 54. Interfaz de creación.

```
use client
import React from "react";
import './Modal.css'

export default function Home({ setShowModal, skuNumber, resetForm }: any) {

  const cleanForm = () => {
    setShowModal(false);
    resetForm();
  }

  return (
    <div className="fixed inset-0 flex items-center justify-center z-50">
      <div className="modalBackground fixed inset-0 bg-gray-800 opacity-50"></div>
      <div className="modalContainer bg-white w-96 p-6 rounded-lg shadow-lg">
        <div className="title mt-1 mb-5">
          <h1 className="text-3xl font-bold">¡SKU: {skuNumber} creado exitosamente!</h1>
        </div>

        <div className="buttons flex justify-between mt-5">

          <a href="/" type="button" className="btn btn--secondary btn--fluid text-s"><span className="btn_inner"><span className="btn_label">Regresar</span></span></a>

          <button
            onClick={cleanForm}
            type="button"
            className="ml-1 btn btn--primary btn--fluid"
          >
            <span className="btn_inner">
              <span className="btn_label">Crear otro</span>
            </span>
          </button>

        </div>
      </div>
    </div>
  );
}
```

Figura 55. Modal de creación.

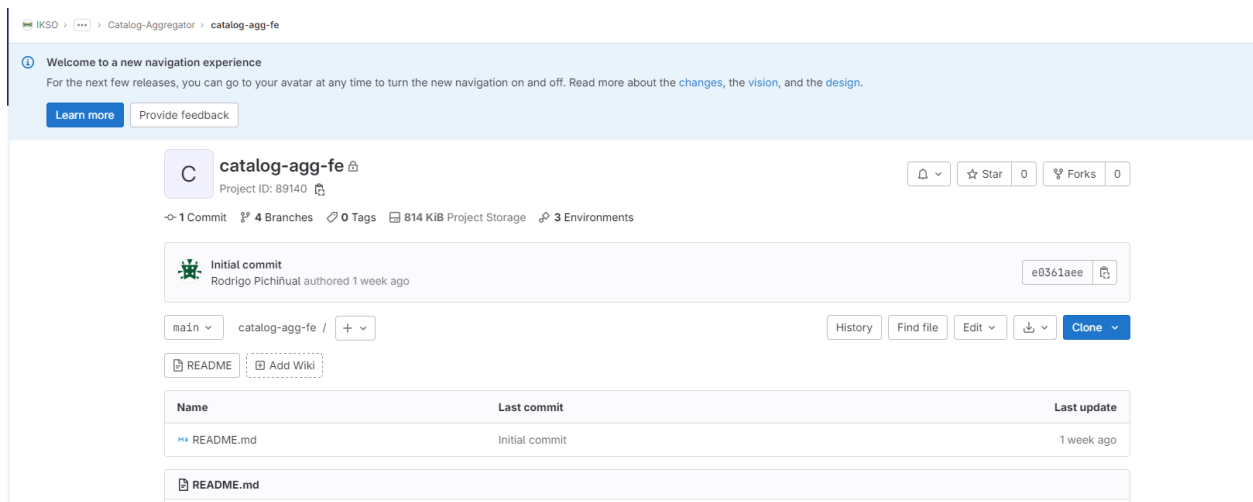


Figura 56. Repositorio frontend.