Plataforma de autogestión para automatizar las reglas de negocio del sector logístico

Andrea Ortiz Giraldo

Informe sobre la práctica profesional para optar por el título de: I**ngeniero de Sistemas de la Universidad de Antioquia**

Orientadores:
Danny Alexandro Múnera Ramírez - Doctor en Informática (Interno)
Martín Elías Quintero - Ingeniero de Sistemas (Externo)

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín
2023

Dedicatoria

A mi Padre, que ya no se encuentra en este plano terrenal, pero su apoyo fue fundamental para lograr este paso tan importante y sé que estará orgulloso.

Agradecimientos

A mi madre por no desfallecer nunca en estos años, por apoyarme incondicionalmente, y levantarme cuando incluso yo no podía. A mi familia, pareja y amigos, por confiar siempre en mis capacidades y recordármelo todos los días. Sin ellos nada habría sido posible.
También a todos los docentes de la Universidad de Antioquia por la formación que aportaron siempre en mi proceso académico y profesional.

Tabla de contenido

LISTA DE FIGURAS

RESUMEN

La empresa Guane Enterprises SAS, siempre está en busca de mejorar todos los desarrollos que tienen para las empresas en sus dos énfasis, sector energético y sector logístico. En este caso, enfocados en el sector logístico el objetivo es crear una plataforma de autogestión para automatizar las reglas del negocio, esto, teniendo en cuenta siempre llevar a cabo los requisitos funcionales y no funcionales del cliente. Para llevar a cabo los objetivos mencionados, se construyó la plataforma con base a una arquitectura de microservicios, para poder separar todas las funcionalidades y permitir la facilidad en la realización de pruebas. Se crearon además tres componentes fundamentales para el proyecto, la base de datos para el guardado de la información, el handler para exponer todas las funcionalidades y el motor de reglas de negocio que es el encargado de construirlas y procesarlas. Para los resultados se hicieron pruebas con librerías de Python que permitían corroborar que todo funcionara de manera correcta y acogiera todos los casos de uso.

Finalmente, se logró que el motor de reglas de negocio en conjunto con los demás elementos mejore el rendimiento de la organización para la cual fue construido. Es fácil de mantener y proporciona mayor flexibilidad a la hora de hacerlo escalable.

Palabras clave **— Sector logístico, transporte, microservicios, motor de reglas de negocio, base de datos, cotización, viaje*, transportista.***

ABSTRACT

The company Guane Enterprises SAS is always looking to improve all the developments they have for companies in its two emphases, energy sector and logistics sector. In this case, focused on the logistics sector, the objective is to create a self-management platform to automate the business rules, always taking into account the functional and non-functional requirements of the client. To carry out the mentioned objectives, the platform was built based on a microservices architecture, in order to separate all the functionalities and allow easy testing. Three fundamental components were also created for the project: the database to store the information, the handler to expose all the functionalities and the business rules engine, which is in charge of building and processing them. For the results, tests were made with Python libraries that allowed corroborating that everything worked correctly and that all the use cases were accepted.

Finally, the business rules engine, together with the other elements, improved the performance of the organization for which it was built. It is easy to maintain and provides greater flexibility in making it scalable.

Keywords **— Logistics sector, transport, microservices, business rules engine, database, quote, haul, carrier.**

I. INTRODUCCIÓN

Guane Enterprises SAS es una compañía que ha tenido un crecimiento significativo en los últimos años y que ofrece soluciones a problemas de la industria automatizando procesos por medio del desarrollo de sistemas informáticos que aplican herramientas de ciencia de datos. Esta empresa se desenvuelve en una serie de sectores económicos principales, en este caso, se hará enfoque en el sector logístico.

Entre los clientes más importantes de Guane se encuentra una empresa que funciona como un *Bróker,* es decir, que se encarga de recibir las peticiones de clientes para cotizaciones de envíos y darles una respuesta. Tercerizan la operación de todos los servicios logísticos correspondientes al transporte de mercancías sin tener vehículos, bodegas o conductores. Al encargarse de la gestión del transporte de carga tienen que realizar una serie de pasos para cumplir a cabalidad el proceso, estos van desde la cotización de los servicios, buscar capacidad de carga para el envío de la mercancía que el cliente necesite, encontrar los carriers o transportistas que pueden hacer la entrega y coordinar la recogida de la mercancía, hacer el seguimiento y tránsito de esta, además de validar los costos de servicio, es decir que aquello que se le haya cobrado al cliente sea lo correcto. También es importante resaltar que existen los TMS *(Transport Management System)* que se encargan de la conexión con todos los transportistas que finalmente realizan las entregas, en este caso se tiene una conexión con Revenova.

Con el fin de gestionar todas las cotizaciones y reglas del sector logístico se creó el asistente para la empresa. Este asistente se encarga desde el proceso de la ingesta de correos que traen las cotizaciones que los clientes desean hacer, hasta la terminación de todo el proceso logístico cuando el paquete ha sido entregado. La empresa realiza una conexión con los transportistas y cada uno tiene unas reglas de negocio específicas para el transporte de carga, pueden ser reglas propias de la empresa, de la ciudad en que se encuentren, del tipo de carga que se desee transportar o de condiciones especiales. Estas reglas influyen en el precio final de la cotización, por ende, cada vez que los transportistas crean una nueva regla de negocio o actualizan alguna ya creada, se debe realizar un proceso grande a nivel de código y acoplamiento para integrar todos los cambios necesarios para que el asistente no presente errores y funcione de la manera correcta.

En este informe se presenta el diseño y desarrollo de una plataforma de autogestión de reglas de negocio llamada Panel de administración, permite a la empresa interactuar de una forma más completa y cómoda con las reglas de negocio de los transportistas asociados a la logística de la empresa.

Para todas las funcionalidades del proyecto fue necesario usar una arquitectura de microservicios para todas las funcionalidades, una base de datos para guardar toda la información de reglas de negocio y un motor de reglas de negocio, que permita la creación de las mismas.

El resto de este informe incluye en la sección 2 los objetivos del proyecto, tanto principales como específicos. En la sección 3 se presenta el marco teórico con todos los conceptos necesarios para comprender totalmente todas las fases del desarrollo del proyecto. En la sección 4 se presenta todo sobre el análisis y diseño de la aplicación y los componentes fundamentales de la misma, con su respectiva profundización. En la sección 5 se presenta un pequeño resumen de la realización de pruebas que se hicieron para comprobar que se cumplieran todos los casos de uso requeridos por el cliente. En la sección 6 se presentan las conclusiones y por último la bibliografía.

II. OBJETIVOS

A. Objetivo general

Diseñar e implementar una plataforma de autogestión para automatizar las “reglas del negocio” del sector logístico.

B. Objetivos específicos

- Extraer y estructurar los requisitos funcionales y no funcionales del cliente de acuerdo con las necesidades.

- Construir un diseño e infraestructura basada en microservicios para el módulo de reglas de negocio.

- Desarrollar e implementar interfaces web para las reglas de negocio de los transportistas del sector logístico.

III. MARCO TEÓRICO

En este capítulo se presenta la revisión de algunos elementos teóricos necesarios para la comprensión de este trabajo. Primero se presenta una introducción del asistente, que es el anteriormente utilizado por la empresa. Esto servirá como punto de partida para entender cómo se maneja la información actualmente y los actores involucrados. También, se hace énfasis en las arquitecturas monolíticas y de microservicios. Es importante mencionar también que el uso de contenedores Docker y Kubernetes es un tema fundamental para la comprensión de este proyecto.

A. Versión Actual de la plataforma Asistente

Una de las empresas con la que trabaja Guane actualmente, funciona como *Broker,* es decir como un intermediario entre un cargador y un proveedor de servicios de carga [1]. Se encarga de recibir las peticiones de clientes para cotizaciones de envíos y darles una respuesta. Tercerizan la operación de todos los servicios logísticos correspondientes al transporte de mercancías sin tener vehículos, bodegas o conductores.

El proceso se realiza desde que se recibe un correo electrónico del cliente con la cotización del envío que desea realizar hasta la entrega de la mercancía.

Para que todo el proceso se lleve a cabo de manera correcta y tenga toda la trazabilidad desde el recibido del correo hasta la entrega de la mercancía, Guane creó un asistente digital para la empresa. Este se encarga de gestionar el proceso desde de la ingesta de correos que traen las cotizaciones de todos los clientes que desean enviar los paquetes, hasta la terminación de todo el proceso logístico cuando el paquete ha sido entregado.

En la Figura 1 se observa un diagrama resumido de la versión actual de la plataforma asistente usada por la empresa

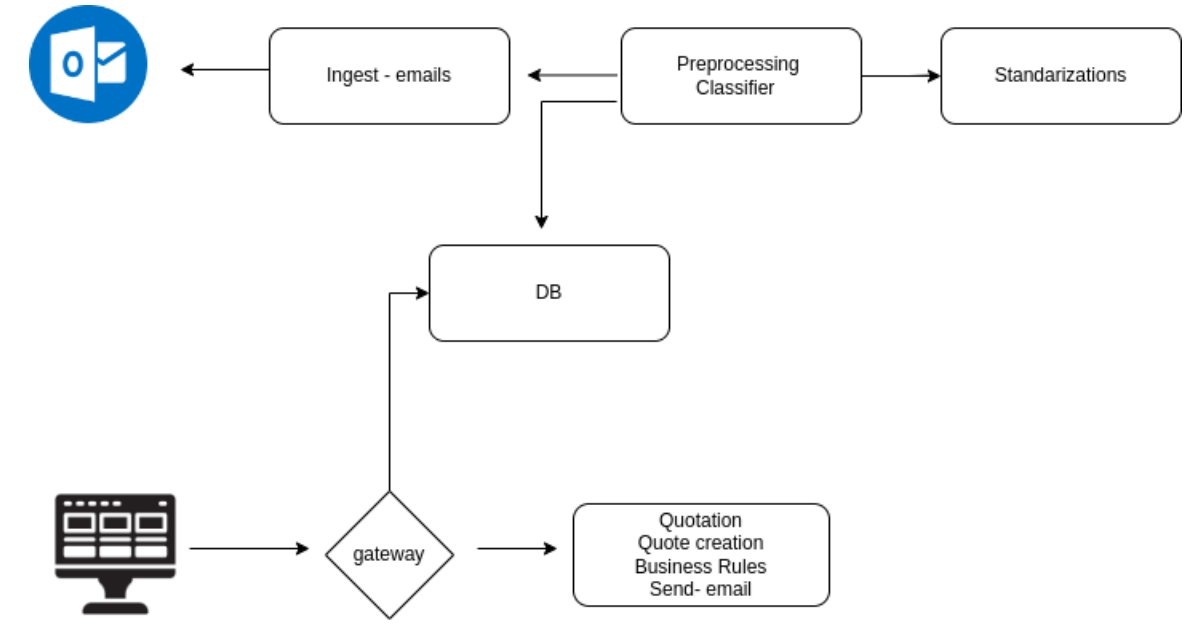

Figura 1: Diagrama versión actual plataforma usada por la empresa

Actualmente, el proceso de ingesta de correos se hace mediante Graph que es la API Rest de Microsoft Outlook [2]. Esta consulta la última hora de correos con base en los datos, los separa entre aquellos que son una cotización y los que no, esto se puede observar en la Figura 2 en la parte de *preprocessing - classifier*. Con los primeros se realiza un proceso cognitivo, con algoritmos que han sido creados e implementados por el equipo de ciencia de datos de la empresa Guane. Estos algoritmos se usan para realizar estandarizaciones, es decir, que todos los datos queden en el mismo formato o con las mismas unidades de medida o de peso. Esto con el fin de ser procesado con mayor facilidad.

En el Front-end del asistente se expone un formulario de cotización donde se autocompletan algunos campos con la información disponible, y con los datos ya estandarizados.

Cuando se tiene la cotización completa se consulta con Revenova, que es un *TMS (Transport Management System),* es decir, una herramienta de administración de transporte multimodal basada en la nube para agilizar la incorporación, el monitoreo, la facturación, la programación y más funciones relacionadas con el transportista [3]. Con esta herramienta se puede encontrar la información de todos los transportistas. Cuando Revenova devuelve toda la información de la cotización para el cliente, se realizan las tareas faltantes en el Back-end para devolver al cliente un correo con todos los precios y opciones disponibles. Toda esta parte del proceso que acaba de ser mencionada se observa en la Figura 2 en la parte de Gateway que realiza las peticiones.

Cuando el cliente elige las opciones más adecuadas, el asistente se encarga de la creación de la carga en Revenova y el seguimiento de esta. Cabe aclarar también que, en el flujo de trabajo del Back-end, se pasa por el proceso de las reglas de negocio, que son directivas que definen las actividades comerciales de una organización. Son importantes porque aclaran los objetivos de una organización y detallan cómo se llevarán a cabo los procesos.

Las reglas de negocio pueden ser informales, escritas o automatizadas [4]. En este caso se usan para verificar todas las tarifas de acuerdo con los transportistas, que cambian conforme a condiciones de cada uno, lugar de destino, o tipo de envío. Como se mencionó anteriormente, las reglas de negocio siempre son añadidas por los desarrolladores directamente en el código, pero se busca que puedan ser creadas y actualizadas por la empresa en una plataforma o panel de administración.

La plataforma actual está basada en microservicios por ende era importante para el panel de administración implementarlo de la misma manera porque además presenta beneficios muy importantes respecto a las arquitecturas monolíticas.

B. Arquitecturas monolíticas vs. basadas en microservicios.

En las aplicaciones con arquitecturas monolíticas todos los procesos están estrechamente asociados y se ejecutan como un solo servicio [5]. Por ende, hacer escalable la aplicación se vuelve cada vez más difícil y complejo, aumentando así también los errores y el impacto de estos.

Con una arquitectura de microservicios una aplicación se crea con componentes independientes que ejecutan cada proceso de la aplicación como un servicio [6]. Cada servicio entonces desempeña una sola función, y esto lo hace más fácil para actualizar, implementar y escalar el software para satisfacer todas las funciones de una aplicación.

Los beneficios de los microservicios son demasiados en comparación a una aplicación monolítica. Primero, la agilidad, esto beneficia el rendimiento de todo el escalado flexible, lo que permite que la aplicación pueda crecer y adecuarse a todas las necesidades de infraestructura. Otro punto importante es el código reutilizable, ya que el código se divide en pequeños fragmentos los equipos podrán usar funciones para diferentes propósitos y esto ahorra tiempo de desarrollo.

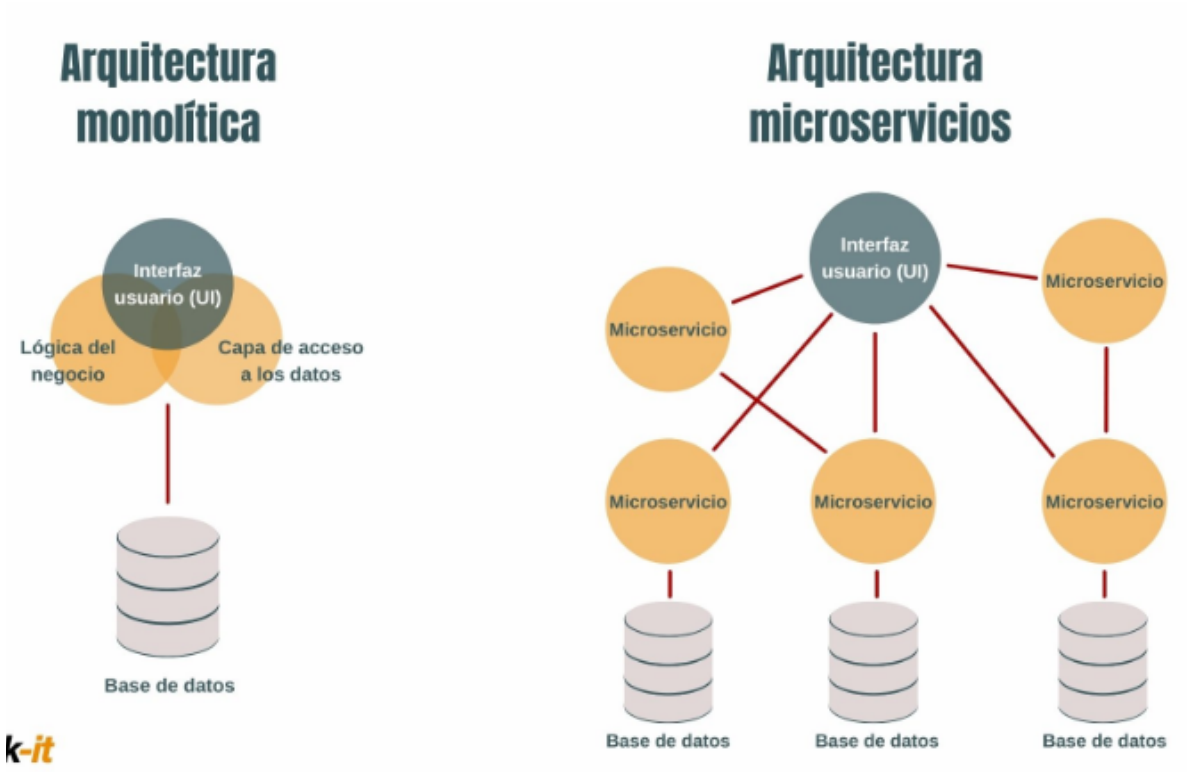

Figura 2: Arquitectura monolítica y Microservicios. Tomada de [7].

La Figura 2 presenta una comparación de la arquitectura monolítica y la de microservicios, donde se puede apreciar que la monolítica agrupa toda la funcionalidad y servicios en una base de código única, y aunque puede ser más fácil de desarrollar, si se desea aplicar un cambio en las funcionalidades se debe volver a lanzar la aplicación por completo, además de que encontrar problemas y fallos puede ser más complicado.

En la arquitectura de microservicios se desacoplan e independizan los componentes de la aplicación, por ende, todos funcionan como una aplicación en sí mismos, lo que hace que se puedan realizar despliegues y desarrollos de manera independiente.Es más seguro, pues si hay algún error no falla todo el sistema, además de que los desarrolladores pueden trabajar de manera simultánea y así reducir el tiempo de desarrollo, e incluso se facilitan los procesos de pruebas.

Una forma de implementar una aplicación basada en microservicios es a través de Docker, puesto que allí se pueden empaquetar las aplicaciones y sus dependencias.

C. Docker y Kubernetes

Docker es la tecnología de contenedores más popular para crear y compartir aplicaciones en contenedores, desde el escritorio hasta la nube [8]. Sin embargo, cuando el número de contenedores crece, la tarea de gestionarlos y administrarlos es compleja debido a la intrincada comunicación entre todos los servicios. Por esta razón, se hace necesario automatizar las tareas de orquestación y así lograr mantener todos los artefactos de software disponibles. Para lograr esta tarea, se utiliza software como Kubernetes que *"es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios"* [9].

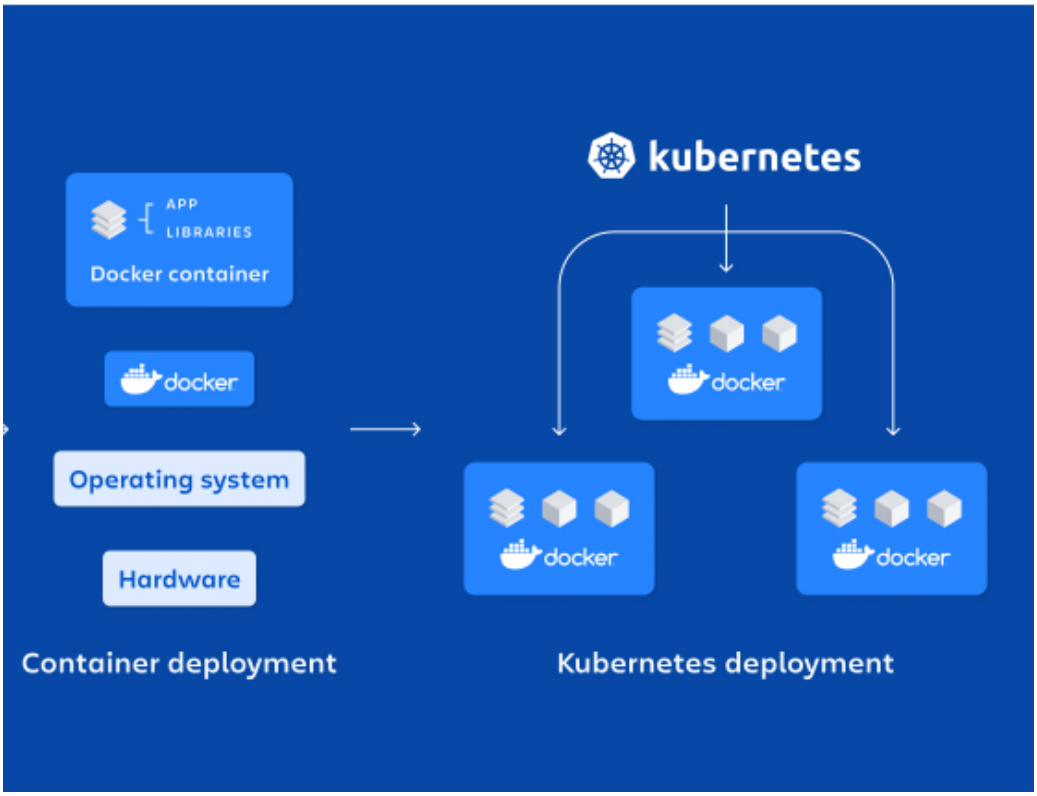

Figura 3: Uso de Docker y Kubernetes. Tomada de [10].

Kubernetes ofrece la capacidad de administrar con facilidad los contenedores y la infraestructura donde están desplegados. Esto lo hace por medio de artefactos software como los *configmaps* que permiten almacenar información no confidencial [11] .

En la Figura 3 se observa una imagen que representa lo que se explicó anteriormente y como a través de kubernetes se administran los contenedores e infraestructura donde se tienen las aplicaciones.

IV. ANÁLISIS Y DISEÑO

Para el desarrollo de la aplicación que se reporta en este informe, se realizó un proceso de análisis y diseño, para plasmar todos los requerimientos del cliente, después se realizó una construcción de la base de datos de acuerdo a las necesidades identificadas, de un handler para administrar las funcionalidades y de un motor de reglas de negocio, que es primordial para el funcionamiento del panel de administración.

A la hora de realizar el diseño se tuvieron en cuenta las necesidades del cliente. Analizando todos los requerimientos, tanto de Back-end como de Front-end necesario para cumplir con todas las funcionalidades. El deseo del cliente era tener una plataforma de autogestión de las reglas de negocio de los transportistas mencionados y explicados anteriormente, un Front-end dinámico donde se puedan agregar las reglas de forma fácil e interactiva y que este a su vez se conecte con el Back-end donde se verifiquen y gestionen las reglas

Se decidió primero realizar una base de datos para el guardado de las reglas de negocio y demás entidades necesarias para el flujo de trabajo, esta base de datos se llama PANEL-ADMIN-DB y será explicada más adelante. Se realizó también un motor de reglas de negocio, esta es la parte más importante y fundamental, puesto que allí se tienen las acciones correspondientes para todas las variables involucradas en cada regla, llamado BUSINESS-RULES-ENGINE que será explicado también más adelante. Finalmente, un Handler donde se manejan todas las acciones, este es consumido por el equipo de Front-end para la interfaz del panel de administración. Además el Handler se conecta con la base de datos para realizar cualquier acción CRUD (Create, Read, Update, Delete) y con el motor de reglas de negocio para exportarlas o chequearlas este es llamado en nuestro sistema PANEL-ADMIN-HANDLER.

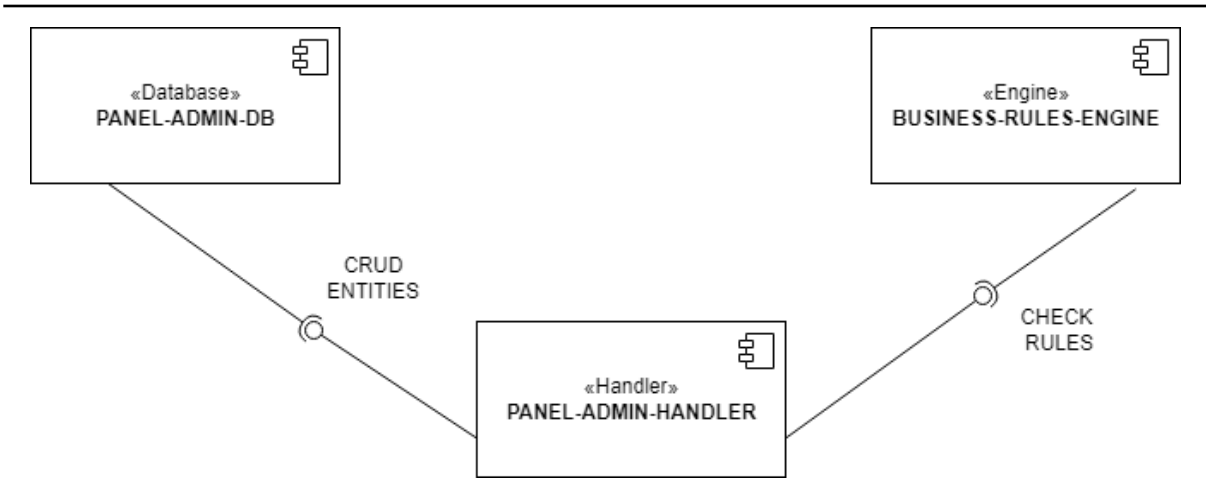

Figura 4. Diagrama de servicios principales de la aplicación

La Figura 4 presenta el estado actual de la aplicación, en este se tienen los 3 servicios de la aplicación y cómo se conectan entre ellos. La mayor parte del flujo se da en el handler (Panel-admin-handler) que se conecta con la base de datos (Panel-admin-db) para hacer el guardado, actualización o eliminación de los datos. Además de esto se conecta con el Engine (Business-rules-engine) que es donde se chequean todas las reglas de negocio, de acuerdo a las variables, condiciones y acciones.

A continuación se explica a detalle cada uno de los servicios del panel de administración mencionados anteriormente.

A. Base de datos - Panel Admin DB

Para el proyecto se usó una base de datos relacional, en este caso PostgreSQL, el cual es un sistema de código abierto de administración de bases de datos. Las consultas relacionales se basan en SQL, y posee además tipos de datos avanzados y permite ejecutar optimizaciones de rendimiento avanzadas [12]. Se usó Tortoise como ORM, que es un modelo de programación que

permite mapear las estructuras de una base de datos relacional (*SQL Server*, *Oracle*, MySQL, etc.), sobre una estructura lógica.

Tortoise es un ORM (*Object Relational Mapper)* fácil de usar inspirado en Django. Está grabado en su diseño que no está trabajando solo con tablas, trabaja con datos relacionales. Se usó también FastAPI como framework para construir la API de la base de datos en Python. Para la implementación de esta se tuvo en cuenta todas las entidades necesarias para el uso correcto de la aplicación.

Roles y entidades de la aplicación

Según las necesidades del cliente, para la aplicación se tienen 3 tipos de usuario, se diferencian de acuerdo a los permisos y cómo pueden interactuar tanto con las reglas como con las demás entidades. El GlobalAdmin, el CompanyAdmin y el DepartmentAdmin. Se explicarán los permisos de estos más adelante. Cabe aclarar y mencionar que para este proyecto se pueden tener varias compañías, por lo que fue diseñado para ser usado por varias empresas que tienen proyectos de desarrollo con Guane.

En la base de datos se tienen varias entidades que serán explicadas a continuación:

Accessorials: Son características específicas que pueden tener un envío, condiciones especiales de transporte o todo aquello que tenga un costo adicional. Pueden ser accedidos por todos los tipos de usuario, pero únicamente creados, modificados y eliminados por el usuario Global Admin.

Canned Responses: Son las respuestas que se generan a las cotizaciones que envían los usuarios.

Están en dos idiomas Inglés y Español. Estos pueden ser accedidos por cualquiera de los 3 tipos de usuarios, igual que creados y modificados, pero solamente eliminados por un tipo de usuario GlobalAdmin.

Carrier: Son los transportistas que hacen o realizan los envíos. Pueden ser accedidos u obtenidos por cualquier usuario que esté activo en el sistema, pero únicamente modificados, creados o eliminados por el tipo de usuario GlobalAdmin.

Company: Son las compañías que pueden interactuar con el sistema o las reglas de negocio, y solo pueden ser creadas, obtenidas, modificadas y eliminadas por el tipo de usuario GlobalAdmin.

Department: Son los departamentos que hay dentro de las compañías y que pueden interactuar con el sistema, solo pueden ser accedidos por los CompanyAdmin, pero creados, modificados o eliminados por el GlobalAdmin.

Perfil: Los perfiles están relacionados con los departamentos, compañías y reglas de negocio. Pueden solo ser accedidos, creados, modificados y eliminados por el tipo de usuario GlobalAdmin.

Usuarios: Son los tipos de usuarios ya mencionados anteriormente, los que interactúan con el sistema, el GlobalAdmin puede crear solo usuarios de tipo CompanyAdmin y DepartmentAdmin. El tipo de usuario CompanyAdmin solo CompanyAdmin y los DepartmentAdmin solo usuarios de su mismo tipo.

Reglas: Son las reglas de negocio que se van a crear, y que van a estar asociados a una compañía, un departamento y un perfil. Se tiene como verificación en este caso que el usuario que está logueado en el sistema dependiendo del tipo de usuario que sea pueda hacer la obtención, creación, modificación, y eliminación de las reglas de negocio.

Cabe destacar que también hay entidades intermedias, que son las relaciones de los *accessorials* con los perfiles, de los departamentos compañías y perfiles o de los carriers con perfiles.

Este diseño de la base de datos permite tener toda la información necesaria almacenada y va conectada a un Handler que será consumido por el equipo de Front-end y que a su vez está conectado al motor de reglas de negocio. Estos dos serán explicados más adelante.

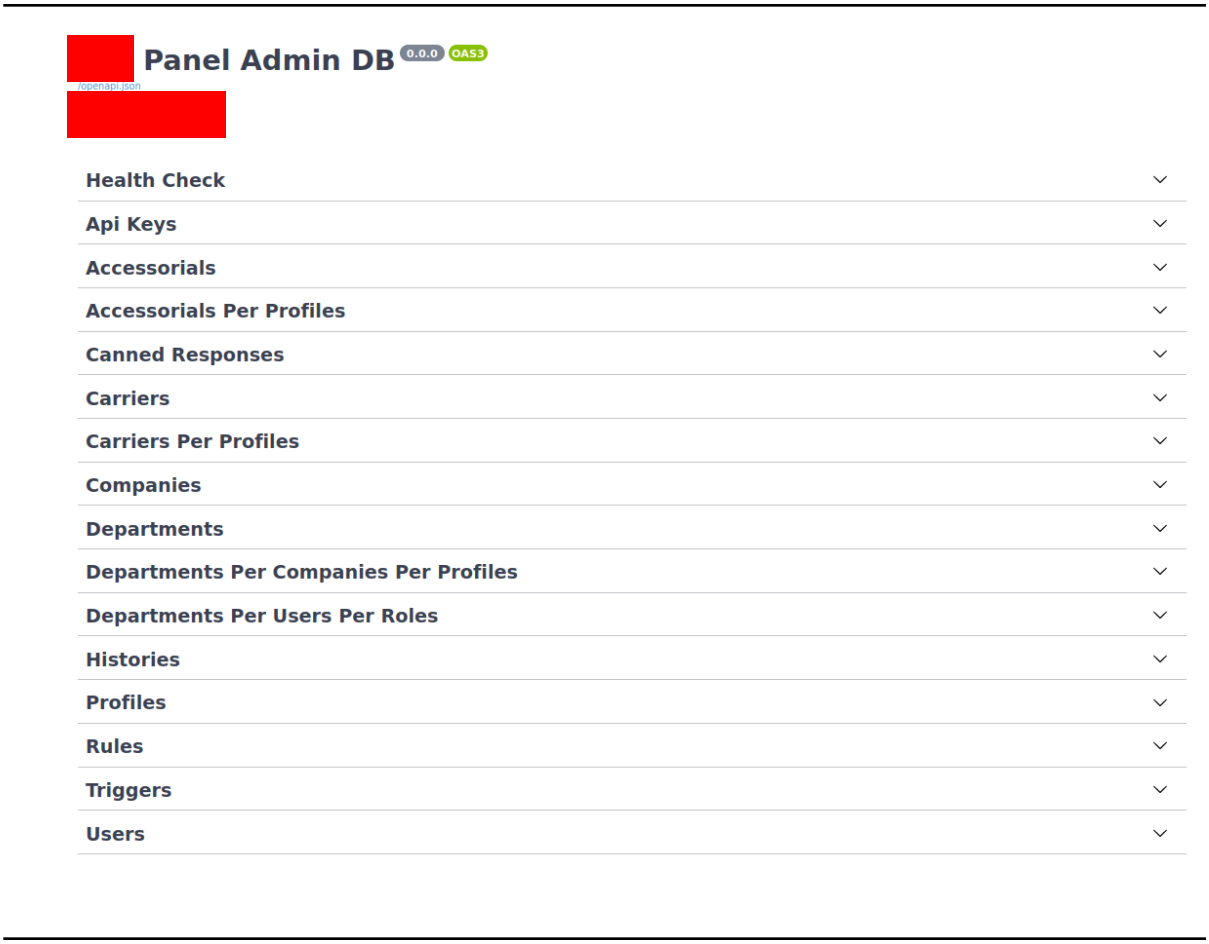

Figura 5. Vista de la API expuesta para la base de datos PANEL-ADMIN-DB

En la Figura 5 se presenta la vista de API de la base de datos Panel-admin-db donde se pueden apreciar todas las entidades ya explicadas anteriormente y que son las manejadas en la versión final de la aplicación.

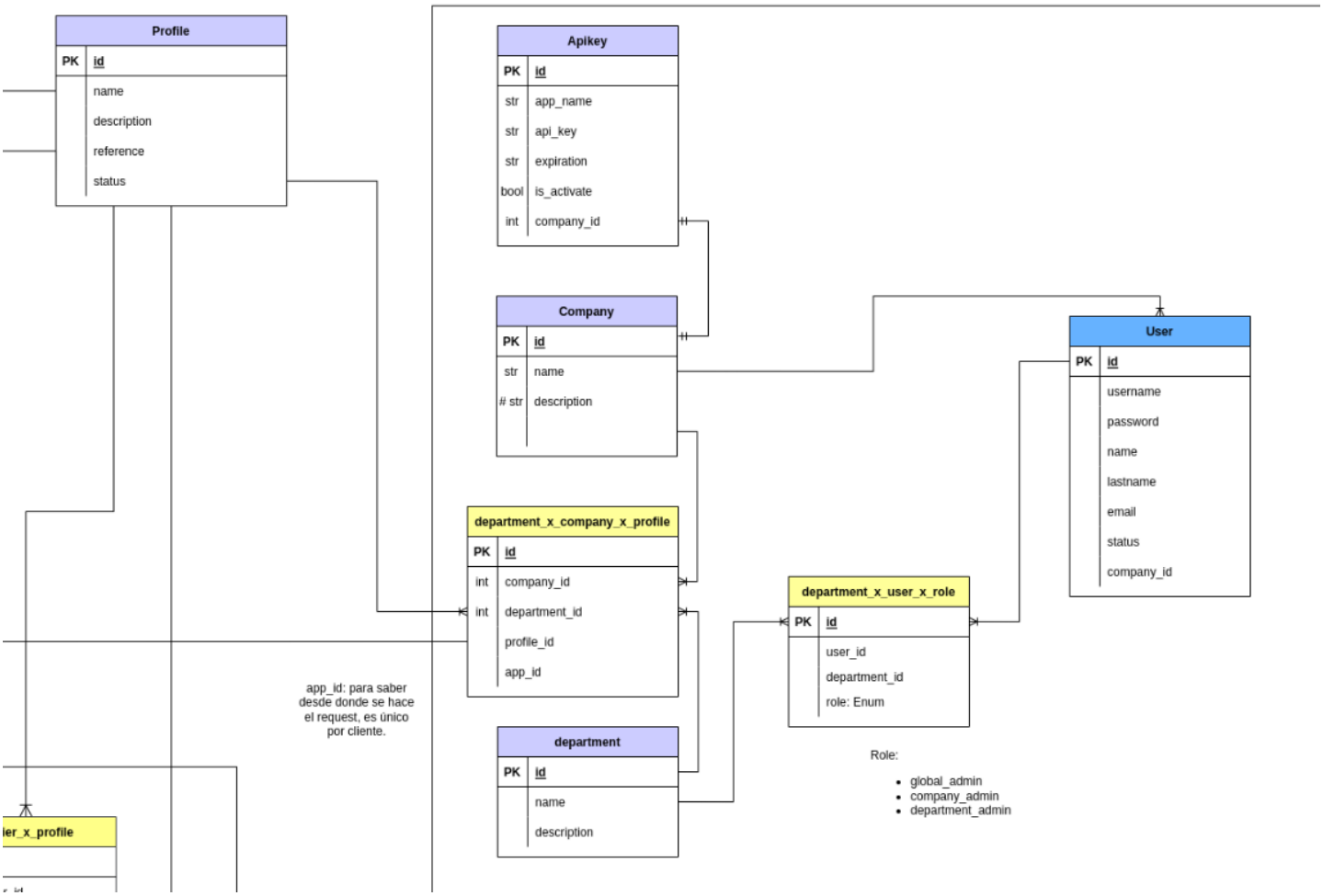

Figura 6. Fragmento del diagrama entidad relación.

En la Figura 6 se presenta un fragmento del diagrama entidad relación donde se presentan las relación entre *Company, Department, User y Perfil*.

B. Motor de reglas de negocio- Business Rules Engine

Un motor de reglas de negocio es un programa de software que ejecuta procesos de decisión basados en una lógica empresarial predefinida [13]
Para el desarrollo de un motor de reglas de negocio, en este contexto, se deben tener en cuenta varios aspectos importantes, uno de ellos son el tipo de condiciones que se verifican para realizar alguna acción, en este caso, son aspectos y variables que tienen que ver con la logística del transporte terrestre. También las acciones a realizar y sobre cuáles entidades se efectúan.

El motor comprende tres puntos o componentes importantes: las variables, las condiciones y las acciones. Además de un adicional que son las fórmulas.
Las reglas se crean estableciendo condiciones de manera que cuando se calcula una variable que desencadena la condición, se realiza alguna acción. De esta manera es importante, definir de manera correcta las variables, acciones y condiciones.

Variables

Las variables son aquellos valores importantes en el sistema, en este caso, son valores que interactúan dentro de un *Haul* en particular. Allí hay muchos valores que cambian dependiendo de cualquier condición.
Para definir una variable se debe tener en cuenta primero de qué tipo será.
Se manejan 5 tipos diferentes:

- Numérico
- String
- Booleano
- Select_multiple

El tipo de dato numérico abarca cualquier tamaño de número
El tipo de dato String permite cualquier tipo de cadena de carácteres de tipo texto.
El tipo de dato Booleano permite solo los valores predeterminados True o False.

El tipo de dato Select Multiple es para manejar listas o varias opciones de datos.

Fue importante definir el tipo de las variables ya que con esta información se sabe que operadores se pueden usar para verificar las condiciones.

Al definir las variables también se tiene en cuenta la categoría a la que pertenecen, es decir, puede ser que una variable tenga relación con el nombre de un *Carrier*, o con el nombre de un *Accessorial*, o directamente con el paquete que se vaya a enviar, por ende, al definirla también se debe definir la categoría a la que pertenece.

Adicional a esto, se incluye un campo extra, llamado SourceType. Este especifica si la variable viene de base de datos, de alguna API externo o es ingresada por el usuario.

Otro campo que se encuentra al realizar esta definición es el trigger. Este hace referencia a la parte del proceso donde se usa la variable, muchas veces se usan a la hora de hacer una cotización o a veces solo para enviar un correo, se especifican con el fin de que el Front.-end las separe y para que la elección del usuario sea más dinámica.
En la Figura 7 se encuentra la implementación de la definición de una variable, está codificada en el lenguaje Python. Tiene todos los campos mencionados en esta sección.

```python
@select_multiple_rule_variable(
    category="Carrier",
    label="Name",
    source=SourceTypes.DATABASE,
    options_type=OptionsTypes.STRING,
    triggers=[Trigger.BUSINESS_RULES, Trigger.SEND_EMAIL],
)
def carrier_names(self) -> list[str]:
    return [carrier.carrier for carrier in (self.haul.carriers or [])]
```

Figura 7: Definición de una variable de tipo Select Multiple.

Condiciones y Operadores

Los operadores deben ser usados para la estructura del motor de reglas de negocio y definir si alguna variable cumple cierta condición para realizar una acción. Los operadores se crean dependiendo del tipo de dato de la variable, así, hay unos especiales para el tipo numérico, para el string, boolean o select multiple.

Como ejemplo se mencionan los siguientes: *equal_to, contains, contains in list, greather than. less than, does not contain,* etc. Cada operador tiene una función diferente, puede ser comparar si dos valores son iguales, si un valor está contenido en una lista, si un valor es menor o mayor a otro, si está dentro de dos rangos de valores, o cualquier funcionalidad específica para verificar las condiciones. También, cuando se construyen las reglas, se definen qué condiciones se deben cumplir. Se usa la palabra ANY para que se cumpla alguna y la palabra ALL para que se cumplan todas.

En la Figura 5 se presenta la implementación de un operador, está codificado en el lenguaje Python.

```python
@type_operator(FIELD_NUMERIC)
def greater_than_or_equal_to(self, other_numeric):
    return self.greater_than(other_numeric) or self.equal_to(other_numeric)
```

Figura 8. Ejemplo de definición de un operador

Acciones

Las acciones están disponibles cuando se activa o cumpla una condición que se verifica como se mencionó anteriormente. Estas se escriben siempre de acuerdo a las necesidades del cliente y a las funcionalidades que la empresa necesite. Para determinar o definir una acción, se tiene inicialmente la categoría que funciona igual que en las variables, si es de un *carrier,* de un *accessorial,* de un mensaje, etc

También se tienen los parámetros, que en este caso, son las variables sobre las cuales se aplicará la acción y finalmente todo el procedimiento que se aplicará sobre ella, por ejemplo sumar algún valor o concatenarlo, e incluso bloquear alguna variable por su nombre. Es importante aclarar que

también en este motor de reglas de negocio se añadió algo llamado Fórmulas. Muchas de las variables que componen un *Haul* o viaje se calculan por fórmulas matemáticas que están casi siempre relacionadas directamente con el peso, medidas del artículo a enviar o cualquier otro factor que se decida. Debido a esto, se tuvo que crear también la manera de chequear que las fórmulas que añada el usuario desde el Front-end estén bien escritas y tengan lógica. Muchas de las acciones creadas manejan cambios en las variables mediante fórmulas.

Para que el motor de reglas de negocio pueda interpretar las reglas, se tiene que construir un archivo JSON que las contenga. La estructura de ese archivo debe contener las condiciones, nombres de la variables, operadores y valores a evaluar, además de las acciones a realizar si se cumplen las condiciones.

Creación del JSON con las reglas de negocio.

Finalmente, las reglas son un archivo de tipo JSON que es interpretado por el motor de reglas ya construido.
El archivo JSON se ve de la siguiente manera:

Figura 9. Ejemplo de JSON con las reglas de negocio a ser procesadas por el motor

En la Figura 9 se puede observar un ejemplo de archivo tipo JSON con las reglas de negocio a ser procesadas por el motor. En la parte de arriba están las condiciones. Hay un ALL, lo que quiere decir que se deben cumplir todas las condiciones como se mencionó anteriormente. Las condiciones en este caso son:

- Que el nombre del *carrier* esté en la lista que hay en *Values*
- Que el nombre del *accessorial* este en la lista de *Values*

Las acciones en este caso serían incrementar el valor del precio del envío para esos *carriers* en específico.

El *endpoint* más importante del motor de reglas de negocio es el *Checker*. Este recibe un *haul* completo con toda la información y las reglas de negocio que se desean aplicar y devuelve la información del *Haul* habiendo aplicado todas las reglas necesarias. Este es el *endpoint* que consume el Panel-admin-handler como se muestra en la Figura 4.

C. Handler - Panel Admin Handler

El último servicio de la aplicación es el Handler. Este es el servicio donde se encuentran todas las funcionalidades y que es consumido por el equipo de Front-end para las vistas y lo que se expone al usuario.

En el *handler* se tiene el servicio de login donde pueden ingresar los 3 tipos de usuarios:

- GlobalAdmin
- Company Admin
- Department Admin

De acuerdo con estos roles tienen los permisos de interactuar con las diferentes entidades. Para cualquier operación CRUD se conecta directamente con la base de datos.

Para el consumo del servicio del engine o motor de reglas tienen 4 endpoints diferentes. Uno de GET y 3 de POST.

Los 3 endpoints de POST son los más importantes, el primero es el *check rules* que interactúa directamente con el *checker* del motor, ahí se le hace el envío del *Haul* o viaje con las reglas a aplicar y retorna el objeto modificado de acuerdo cin ellas.

El segundo endpoint es el *check–fórmula.* Como también ya se había mencionado, se verifica que las fórmulas matemáticas que agregue el usuario desde la vista de la aplicación deben estar correctamente escritas y formuladas.

El tercer endpoint es el *Check fórmula in rule* verifica si la fofmula se aplica correctamente dentro de las reglas.

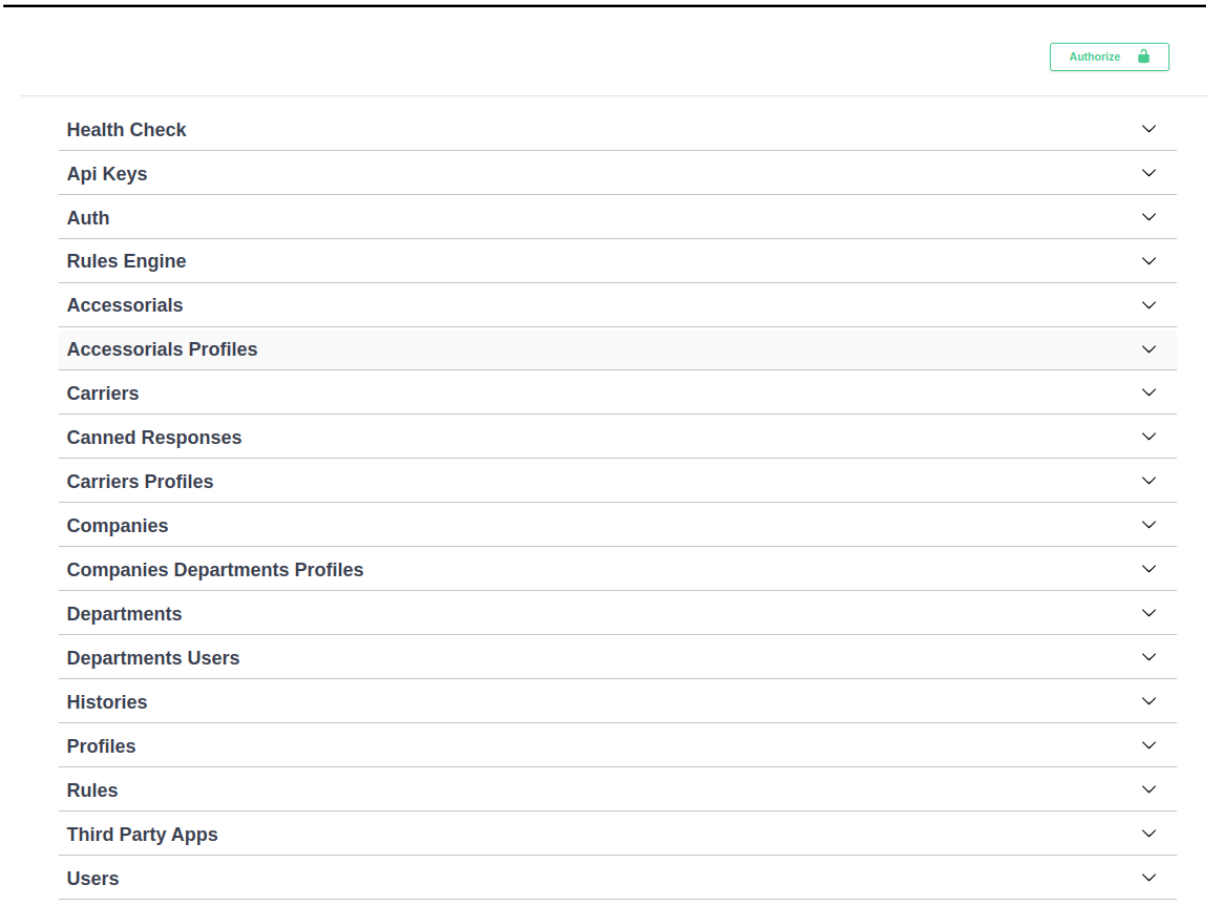

Figura 10: VIsta de la API del Handler

En la figura 10 se muestra la vista de la Api del *Handler,* allí se presentan todos los endpoints de las entidades. Tanto las que tienen conexión con el motor de reglas de negocio como con la base de datos. Estos endpoints son los que consumen los del equipo de Front-end.

D. Funcionalidades adicionales

Durante el proceso de desarrollo de la aplicación surgieron funcionalidades adicionales que deseaban ser incluidas por el cliente.

Primero, se creó un historial por cada método Post, Patch o Delete que el usuario realiza y la información es guardada en una tabla llamada History en la base de datos.

Se usó el MIddleware de FastAPI para lograr obtener los datos necesarios para el guardado como el tipo de usuario que está accediendo al sistema, el *body* de la solicitud,y los *path_params y query_paramas. "Un middleware es una función que se hace con cada solicitud antes de que sea*

procesada por cualquier operación de ruta específica y con cada respuesta antes de devolverla" [14]. Después de obtener los datos se usa una *Background Task* que es una librería de FastAPI para el guardado en la base de datos y que no interrumpa el flujo normal de la aplicación. "*Es útil para las operaciones que deben realizarse después de una solicitud, pero que el cliente realmente no tiene que esperar a que se complete la operación antes de recibir la respuesta" [15].*

Otra funcionalidad especial que se realizó fue el reporte de fallas o errores de la aplicación a un canal de Slack, la cual "*es una aplicación de mensajería para empresas que conecta a las personas con la información que necesitan." [16]*
Lo que se hace es que al recibir cualquier error o excepción en el funcionamiento envíe un reporte a un canal en la aplicación de mensajería que contiene la información del fallo, el momento en que ocurrió y todos los datos importantes para encontrar el lugar exacto donde se presenta la excepción. Esto se hace con una función que crea el mensaje y consume el servicio expuesto por Slack.

También se tiene otra funcionalidad que era importante para la empresa que es guardar el historial del motor de reglas de negocio, es decir, después de pasarle el Haul y las reglas a revisar, guarda los datos de todas las condiciones que verificó y las acciones que fueron aplicadas.
Se puede guardar un historial completo en Inglés y Español, uno completo solo en Inglés, uno completo solo en Español, un historial resumido en inglés y un historial resumido en Español.

V. REALIZACIÓN DE PRUEBAS

Las pruebas son importantes e indispensables para que el software quede de calidad y menos propenso a errores.

Python tiene un framework de prueba llamado PyTest, es un framework con muchas funcionalidades y que permite hacer desde pruebas pequeñas a pruebas de gran escala.

Pytest además viene con Fixtures, lo cual es útil para el manejo de recursos. Los Fixture **"***Proporcionan una línea de base fija para que las pruebas se ejecuten de manera confiable y produzcan resultados consistentes y repetibles. La inicialización puede configurar servicios, estado u otros entornos operativos" [17].*

Otra librería importante que se usó para las pruebas es MOCK. Los mocks permiten reemplazar partes de la aplicación que se quieren testear con objetos simulados. Esto es importante porque algunas veces se tienen servicios que son cruciales en la aplicación pero que al interactuar con ellos se pueden tener efectos secundarios no deseados al ejecutar prueba, por ende, al mockear los servicios se minimizan estos errores.

Se realizaron pruebas unitarias y de integración a los tres servicios de la aplicación. Se verifica que todas las operaciones CRUD se hagan de forma correcta y que la conexión que se da entre los diferentes servicios no falle o presente excepciones.
Cada vez que se añade una nueva entidad al código se añaden sus respectivas pruebas para evitar errores a la hora de desplegar y no dañar otras funcionalidades.
Para las pruebas se utilizó la librería de Pytest y Mock que fueron explicadas anteriormente.

A. Pruebas a la base de datos

En la base de datos se realizaron pruebas unitarias a todos los endpoints. Se verifica que realice las operaciones CRUD de manera correcta, que no haya conflicto con claves primarias, secundarias, repetición de datos, o mal tipado.

Primero se construyen los datos de prueba para cada una de las entidades, luego se hacen las peticiones al servicio con los datos y se verifica que entreguen una respuesta correcta.

```python
@pytest.mark.parametrize(
    "input, expected",
    [
        (tests_constants.PREFIX_USER, 200),
    ],
)
def test_get_by_id_complete_info(test_app: TestClient, input: str, expected: int):
    """
    Get users complete info, filter id
    """
    response = test_app.get(f"{tests_constants.SERVICE_URL}{input}/1/complete-info")
    assert response.status_code == expected, response.text
    assert response.json()["id"] == 1
```

Figura 11. Ejemplo de prueba en la base de datos.

En la Figura 11 se muestra un ejemplo de prueba realizado en la base de datos, en este en específico se prueba el traer los usuarios por un Id.

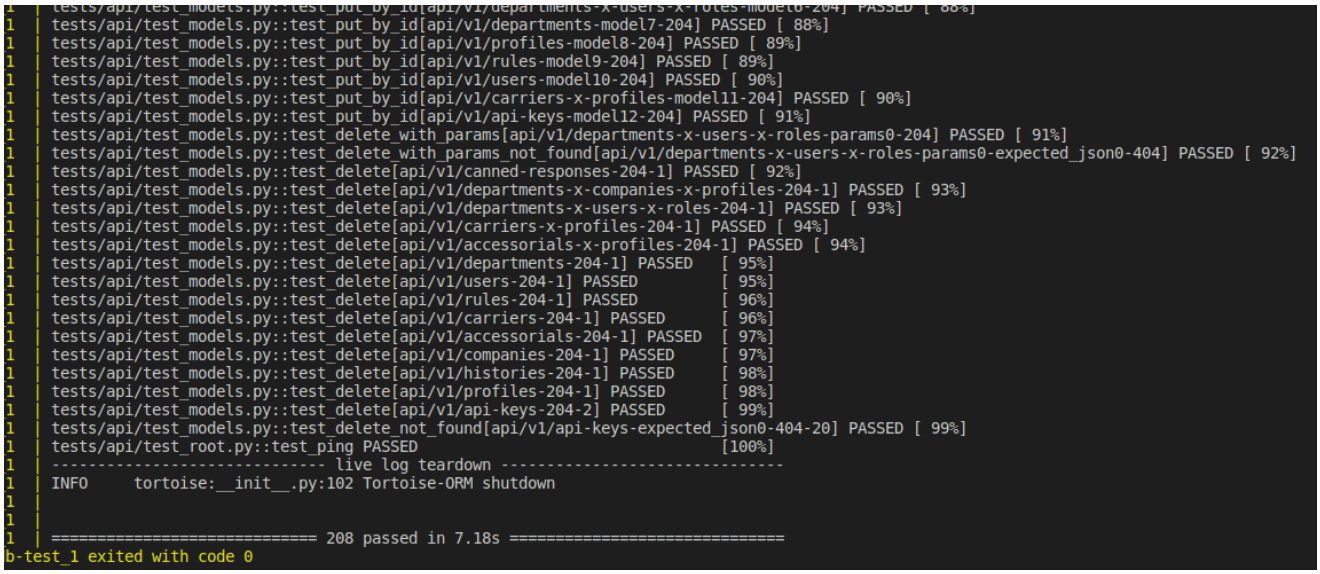

Figura 12. 208 Pruebas de la base de datos.

En la Figura 12 se pueden observar los resultados de las 208 pruebas que se realizan a todas las entidades y operaciones CRUD de la base de datos.

B. Pruebas al motor de reglas de negocio

Las pruebas realizadas al motor de reglas de negocio tenían que comprobar que los operadores, variables, condiciones y acciones estuvieran escritas de forma correcta. Por ende, se realizaron tanto pruebas unitarias como de integración.

Las pruebas de integración se encargaron de verificar que todos los demás componentes en conjunto permitieran que el motor de reglas de negocio las procese adecuadamente.
Se creó un archivo llamado *pass_data.py* que contenía toda la información que deberían devolver las pruebas de ser correcto el funcionamiento y un JSON que contiene información del Haul y las reglas a aplicar, este se ingresa al inicio de las pruebas.

```
Eduardo Velasquez Velez, 3 months ago | 2 authors (Daniel Santa and others)
class StringOperatorTests(TestCase):
    def test_operator_decorator(self):
        self.assertTrue(StringType("foo").equal_to.is_operator)

    def test_string_equal_to(self):
        self.assertTrue(StringType("foo").equal_to("foo"))
        self.assertFalse(StringType("foo").equal_to("Foo"))

    def test_string_equal_to_case_insensitive(self):
        self.assertTrue(StringType("foo").equal_to_case_insensitive("FOo"))
        self.assertTrue(StringType("foo").equal_to_case_insensitive("foo"))
        self.assertFalse(StringType("foo").equal_to_case_insensitive("blah"))

    def test_string_starts_with(self):
        self.assertTrue(StringType("hello").starts_with("he"))
        self.assertFalse(StringType("hello").starts_with("hey"))
        self.assertFalse(StringType("hello").starts_with("He"))
```

Figura 13. Ejemplo de pruebas unitarias para los operadores de tipo String.

En la Figura 13 se muestra un ejemplo de pruebas unitarias para los operadores de tipo *String*. Aquí se prueba que todos los que se crearon estén funcionando correctamente.

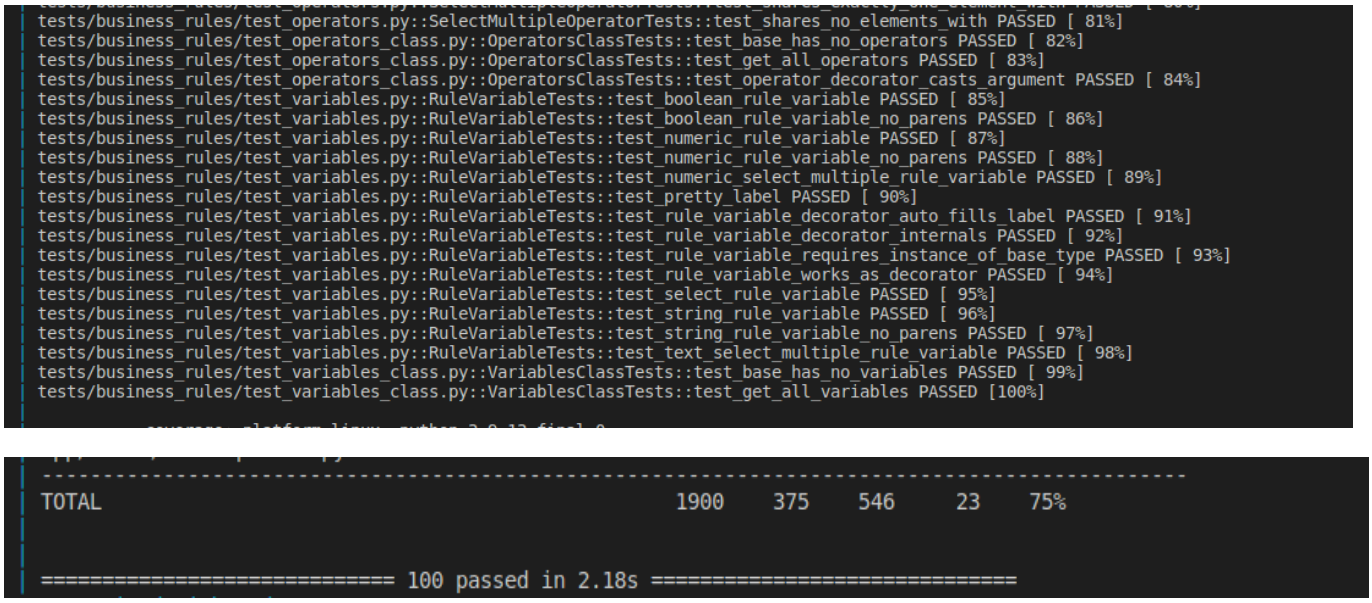

Figura 14. 100 pruebas del motor de reglas de negocio.

En la Figura 14 observan los resultados de las 100 pruebas que se realizan en el motor de reglas de negocio, tanto pruebas unitarias para cada entidad como pruebas de integración que verifican el flujo correcto de la aplicación.

C. Pruebas al Handler

En el handler se realizaron pruebas unitarias a todos los endpoints. Se verifica que realice las operaciones de manera correcta tanto las de conexión a la base de datos como las de conexión con el motor de reglas de negocio.

Primero se construyen los datos de prueba para cada una de las entidades, luego se hacen las peticiones al servicio con los datos y se verifica que se entregue una respuesta correcta.

Es importante aclarar que las pruebas realizadas en el *handler* se hicieron usando la librería *Mock,* que se explicó anteriormente.

```python
@pytest.mark.asyncio
async def test_get_all_carrier(test_app: TestClient, monkeypatch, fastapi_dep):
    async def mock_get_all_carrier_test(skip=0, limit=1, payload={}):
        return {"count": 1, "skip": skip, "limit": limit, "data": []}

    monkeypatch.setattr(
        carrier_service, "get_all_with_count", mock_get_all_carrier_test
    )

    with fastapi_dep(app).override({deps.get_current_active_user: user_fake}):
        response = test_app.get(url=BASE_URL, params={"skip": 0, "limit": 1})
        assert response.status_code == 200, response.text
        assert response.json() == {"count": 1, "skip": 0, "limit": 1, "data": []}
```

Figura 15. Ejemplo de prueba unitaria para el servicio de obtener todos los carrier

En la Figura 15 se muestra un ejemplo de test realizado en el *handler,* en este en específico se prueba el traer todos los *carrier.*

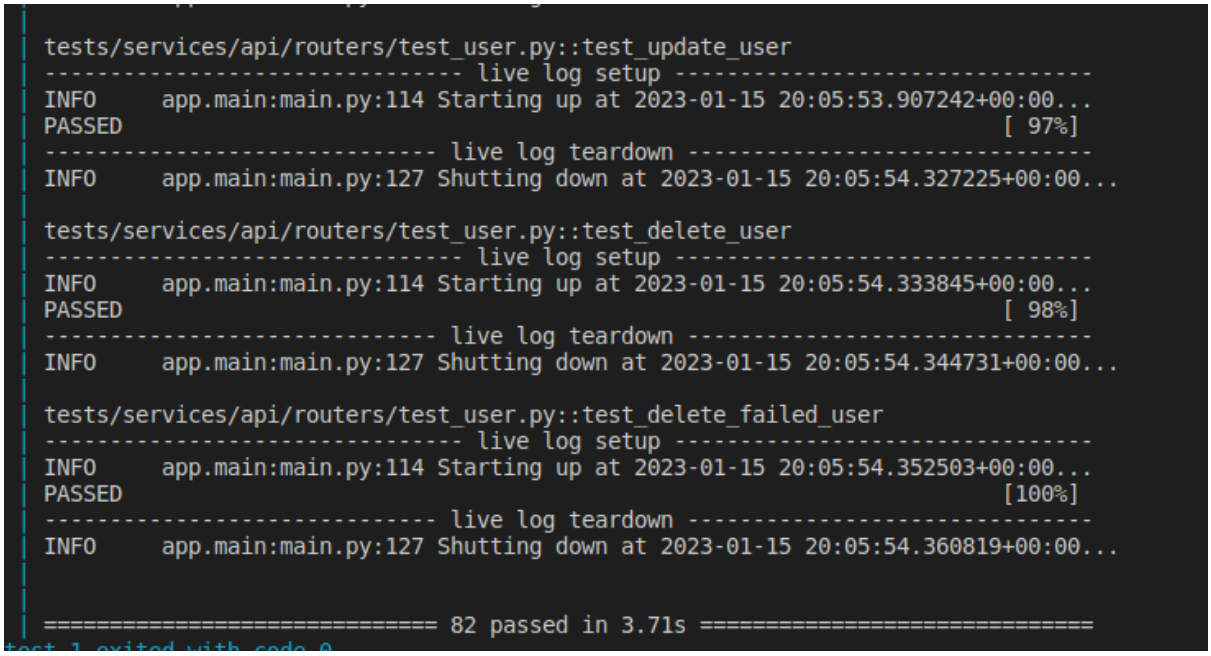

Figura 16. 82 pruebas del handler.

En la Figura 16 se observan los resultados de las 82 pruebas que se realizan en el *handler,* tanto pruebas unitarias para cada entidad como pruebas de integración que verifican el flujo correcto de la aplicación y la conexión con base de datos y el motor de reglas de negocio.

VI. CONCLUSIONES

En este informe se presentó el desarrollo de la aplicación panel de administración con la cual se mejora el añadido de las reglas de negocio de los carriers asociados a una compañía, para hacerlo más rápido y menos propenso a errores.

Se desarrolló un motor de reglas de negocio que mejora el rendimiento de la organización, puesto que, se puede hacer actualizaciones de reglas o creación de nuevas sin intervenir directamente en el código. Por ende, es más fácil de mantener y proporciona mayor flexibilidad a la hora de enfrentar nuevos retos, para la realización se utilizó el lenguaje de programación Python, el uso de todas las librerías de este lenguaje y el conocimiento de los frameworks permitió llevar a cabo y desarrollar todas las funcionalidades requeridas por el cliente, tanto las planteadas inicialmente como aquellas que surgieron después. Además el uso de Docker facilitó la modularidad de la aplicación, puesto que la orquestación de los contenedores permitió crear, actualizar y reparar todos los servicios sin necesidad de modificar todos los módulos. Además de que la implementación se dio de forma más rápida.

No solo fue importante el uso correcto de las tecnologías, sino que además la implementación y escritura de documentación y comunicación constante con todos los miembros del equipo permitió que las tareas se llevarán a cabo sin inconvenientes y acoplando todos los componentes de manera eficaz permitiendo que el flujo se realice como debería. Finalmente, un buen proceso de análisis de los requisitos, posterior diseño e implementación con las tecnologías y la buena comunicación permitieron la funcionalidad completa y adicionales.

VII. REFERENCIAS

[1] «What Does a Freight Brokerage Do?» https://www.atsinc.com/blog/what-does-a-freight-brokerage-do (accedido 13 de octubre de 2022).

[2] «abheek-das. (s/f). *Use the Outlook mail REST API*» Microsoft.com.https://learn.microsoft.com/en-us/graph/api/resources/mail-api-overview?view=graph-rest-1.0 (accedido 6 de Diciembre de 2022)

[3] «Revenova TMS», *Capterra*. https://www.capterra.co/software/144582/revenova-tms (accedido 13 de octubre de 2022).

[4] «10 ejemplos de reglas de negocios en la automatización de procesos». https://www.processmaker.com/es/blog/10-examples-of-business-rules-and-logic/ (accedido 13 de octubre de 2022).

[5] «¿Qué son los microservicios? | AWS». https://aws.amazon.com/es/microservices/ (accedido 13 de octubre de 2022).

[6] «¿Qué son los microservicios? | AWS». https://aws.amazon.com/es/microservices/ (accedido 13 de octubre de 2022).

[7] Click-IT, «Arquitectura monolítica vs arquitectura de microservicios: ¿cuál debo elegir? | Click-IT | Servicios tecnológicos y de consultoría». https://click-it.es/arquitectura-monolitica-vs-arquitectura-de-microservicios-cual-debo-elegir/ (accedido 5 de diciembre de 2022).

[8] «Contenedores de Docker | ¿Qué es Docker? | AWS». https://aws.amazon.com/es/docker/ (accedido 13 de octubre de 2022).

[9] «¿Qué es Kubernetes? | Kubernetes». https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/ (accedido 13 de octubre de 2022).

[10] «Atlassian. (s/f). *Comparación entre Kubernetes y Docker*. Atlassian.». https://www.atlassian.com/es/microservices/microservices-architecture/kubernetes-vs-docker (accedido 6 de Diciembre de 2022)

[11] «Pod | Google Kubernetes Engine (GKE) | Google Cloud». https://cloud.google.com/kubernetes-engine/docs/concepts/pod (accedido 13 de octubre de 2022).

[12] S. Borges, «¿Qué es PostgreSQL? - Para qué sirve, Características e Instalación», *Infranetworking*, 19 de noviembre de 2019. https://blog.infranetworking.com/servidor-postgresql/ (accedido 5 de diciembre de 2022).

[13] «Trabajar con un motor de reglas de negocio | Elegir un BRE por sus beneficios». https://www.processmaker.com/es/blog/working-with-business-rules-engines/ (accedido 5 de diciembre de 2022).

[14] «Middleware - FastAPI». https://fastapi.tiangolo.com/tutorial/middleware/ (accedido 5 de diciembre de 2022).

[15] «Background Tasks - FastAPI». https://fastapi.tiangolo.com/tutorial/background-tasks/ (accedido 5 de diciembre de 2022).

[16] «¿Qué es Slack? | Slack». https://slack.com/intl/es-co/help/articles/115004071768-%C2%BFQu%C3%A9-es-Slack- (accedido 5 de diciembre de 2022).

[17] «pytest fixtures: explicit, modular, scalable — pytest documentation». https://docs.pytest.org/en/6.2.x/fixture.html (accedido 5 de diciembre de 2022).