2025

FRESH&GO

DE LA TIENDA A TU MESA EN UN CLICK

FRESH&GO

MAR ALEJANDRA QUISPE ESCALANTE Y JESUS LACRUZ

Fecha: 1.09.2025 Versión: 1

Índice

Historia del Documento

Versión	Fecha	Razón del Cambio	Autor(es)
1.0	1/09/2025	Primer borrador	Mar Alejandra Quispe Escalante y Jesús Lacruz
1.1	6/10/2025	Cambio de la lógica de la arquitectura a una más sencilla	Mar Alejandra Quispe Escalante y Jesús Lacruz
1.2	11/05/2025	Cambio diagramas	Mar Alejandra Quispe Escalante
1.3	Proximamente	Primera revisión profe	Profe

Resume:

1 Introducción

1.1 Presentación de la empresa

*Nuestra propuesta se basa en una empresa ficticia llamada **Fresh and Go**, dedicada a la **logística de supermercados***

La empresa busca digitalizar sus operaciones mediante una plataforma que unifique los procesos de gestión de pedidos, control de flota y seguimiento de entregas, garantizando un servicio más rápido, fiable y trazable

Con Fresh and Go queremos representar una compañía que distribuye productos alimentarios desde los supermercados hasta los clientes finales, controlando los envíos y la flota de vehículos
*El objetivo principal es **unificar toda la información** (pedidos, vehículos, incidencias, usuarios, etc.) en una única plataforma sencilla, accesible y adaptable*

*A través de esta práctica, planteamos una **arquitectura inicial** que sirva como base para futuras ampliaciones, pero que en esta primera fase se mantenga simple y completamente realizable*

1.2 Eslogan y propuesta de valor

Eslogan: *"De la tienda a la mesa en un click"*

Propuesta de valor:
Nuestra idea es desarrollar una plataforma que ayude a gestionar pedidos y entregas de forma más organizada y transparente. No pretendemos construir un sistema complejo, sino una aplicación funcional que refleje el flujo de trabajo de una empresa logística moderna

Los puntos que más valor aportan a nuestra propuesta son:

- ***Eficiencia:*** *centralizar la gestión de pedidos, vehículos e incidencias en un mismo sistema.*
- ***Transparencia:*** *permitir la consulta del* ***estado actualizado*** *de los pedidos y las entregas desde una interfaz web.*
- ***Control:*** *simular lecturas de* ***ubicación y temperatura de los vehículos*** *para comprobar el estado de la flota y registrar incidencias cuando se detecten valores fuera de rango*

Creemos que este enfoque es realista y nos permite sentar una base sólida sobre la que podremos construir funcionalidades más avanzadas más adelante

2 Contexto del Proyecto

2.1 Actividad principal

Antes de definir la solución, analizamos cómo funcionan actualmente los sistemas de reparto en empresas reales del sector.
Observamos que los repartidores suelen utilizar aplicaciones que gestionan sus rutas y entregas, pero que no siempre están sincronizadas con los cambios de los pedidos o con la información del cliente.
A partir de esa base, planteamos una versión simplificada de ese modelo, centrada en integrar la gestión de pedidos, vehículos e incidencias.

En nuestro proyecto, hemos decidido centrarnos en una ***plataforma de gestión logística simple****, pensada para mejorar la coordinación entre los distintos actores que intervienen en el reparto: administradores, proveedores, conductores y clientes*

- ***Crear y consultar pedidos*** *desde una interfaz web (actualización limitada: estado y notas).*
- ***Asignar manualmente*** *un vehículo a un pedido (sin optimización de rutas).*
- ***Consultar el estado actual*** *de los pedidos (no "tiempo real", sino* ***último dato disponible****).*
- ***Registrar lecturas de flota simuladas*** *(ubicación y temperatura) obtenidas desde una* ***fuente simulada*** *(JSON/valores generados).*
- ***Crear incidencias automáticas simples*** *cuando una lectura simulada esté fuera de rango* ***(opción incluida solo si es trivial)***

Nota: las "lecturas IoT" serán ***simuladas****, no provienen de sensores reales; se actualizarán mediante peticiones REST o al recargar*

2.2 Sistemas de información existentes

En las empresas logísticas reales, los distintos sistemas (CRM, flota, sensores o aplicaciones móviles) suelen existir por separado. Sin embargo, rara vez están integrados entre sí, lo que obliga a los repartidores y a la oficina central a trabajar con herramientas distintas.
En nuestra simulación, representamos esa misma situación, pero simplificada, para poder trabajar sobre una arquitectura unificada

En la simulación de Fresh and Go consideramos varios sistemas que forman parte de la operación logística, aunque algunos serán implementados solo de manera básica o simulada

Sistema	*Función principal*	*Estado actual*
Frontend Web	*Interfaz donde los usuarios pueden crear pedidos, consultarlos y revisar incidencias.*	*En desarrollo; conectará con la API.*
API Unificada	*Gestiona la lógica del sistema, conecta con la base de datos y simula la integración con CRM e IoT.*	*En fase de diseño.*
BBDD Central (Pedidos, Vehículos, Incidencias, Usuarios)	*Almacena la información interna del sistema.*	*En construcción.*
CRM (simulado)	*Representa la gestión de clientes y proveedores.*	*No se implementa, solo se referencian sus datos por ID.*
IoT (simulado)	*Simula la obtención de lecturas de ubicación y temperatura.*	*Datos generados automáticamente o desde JSON.*

(De forma opcional, se contempla la figura del Administrador o Soporte técnico, que puede intervenir en el sistema para registrar manualmente información o resolver incidencias en caso de fallo)

2.3 Problemas de integración actuales

*Durante la investigación inicial del proyecto, uno de los integrantes del grupo consultó con personas que habían trabajado en empresas de mensajería y reparto (como GLS o SEUR). De esas conversaciones pudimos observar varios **problemas y carencias comunes** en los sistemas de trabajo actuales, que nos sirvieron de referencia para diseñar Fresh and Go.*

Los puntos más destacados fueron los siguientes:

- *En muchas empresas, los repartidores utilizan una **aplicación propia** que muestra el orden de entrega, direcciones y horarios, pero **no siempre está actualizada** o sincronizada con los cambios del cliente. Esto provoca que haya entregas canceladas en mitad de la ruta o direcciones incorrectas.*
- *La **comunicación entre la oficina central y los conductores** suele depender de llamadas o mensajes manuales. Aunque existen sistemas de aviso o alertas, **no hay una integración automática** con la gestión de incidencias o pedidos.*
- *Los vehículos cuentan con **GPS y sensores** (temperatura, velocidad o distancia), pero esos datos **no se aprovechan de forma centralizada**. La información se genera, pero no se guarda ni se comparte con otros módulos del sistema.*
- *Las **alertas de entrega o de sensores** no siempre quedan registradas como incidencias en una base de datos; muchas veces se gestionan al momento y luego se pierden. Esto dificulta el seguimiento y análisis de errores o retrasos.*
- *Los conductores pueden ver su propia ubicación y ruta, pero **los administradores o clientes no siempre tienen acceso claro** al estado actualizado del pedido o del vehículo.*
- *El sistema de control depende en gran medida de que los usuarios (conductores o personal de oficina) **mantengan la información al día de forma manual**. No existe un flujo automatizado entre la app de reparto, el CRM de clientes y la gestión de flota.*

*A partir de estos puntos, entendimos que el principal problema es la **falta de integración entre sistemas**:*

*Cada área (pedidos, flota, incidencias, clientes) maneja sus propios datos y herramientas. Por eso, con Fresh and Go buscamos diseñar una base sencilla que permita **unificar toda esa información** en una plataforma central y actualizada*

2.4 Objetivos de la integración

El propósito principal del proyecto es diseñar una arquitectura unificada que conecte los distintos módulos del sistema y mejore la visibilidad de la información logística.

Objetivos específicos:

1. *Desarrollar una API REST que gestione pedidos, vehículos, incidencias y usuarios.*
2. *Centralizar la información en una base de datos simple y bien estructurada.*
3. *Simular una integración IoT que proporcione lecturas básicas (ubicación y temperatura).*
4. *Permitir la consulta actualizada de pedidos y flota desde el Frontend.*
5. *Sentar las bases para futuras ampliaciones (ERP, optimización de rutas, pasarela de pagos, etc.)*

Objetivo general:

Unificar la gestión de pedidos y flota en una plataforma sencilla, coherente y fácil de ampliar en el futuro

3 Modelo de datos común

3.1 Identificación de entidades

Se identifican las siguientes entidades principales:

• Cliente: datos de identificación, dirección y pedidos asociados.

• Proveedor: información de empresas que abastecen de productos.

• Pedido: detalle de productos, estado, cliente y proveedor asociado.

• Vehículo: datos de matrícula, capacidad y sensores IoT.

• Ruta: trayecto planificado y estado en tiempo real.

• Incidencia: registro de fallos, retrasos o incidencias detectadas.

3.2 Estructura de datos iniciales (JSON Schema)

Ejemplo de estructura de datos inicial en formato JSON Schema:

```
{
  "pedido": {
    "id": "string",
    "cliente": {
      "id": "string",
      "nombre": "string",
      "direccion": "string"
    },
    "proveedor": {
      "id": "string",
      "nombre": "string"
    },
    "productos": [
      {
        "id": "string",
        "nombre": "string",
        "cantidad": "number"
```

```
      }
    ],
    "vehiculo": {
      "id": "string",
      "matricula": "string",
      "gps": "string",
      "temperatura": "number"
    },
    "estado": "pendiente|en_transito|entregado"
  }
}
```


4. Arquitectura inicial

En este apartado describimos la **arquitectura técnica inicial** que hemos diseñado para el proyecto *Fresh and Go*.
Nuestro objetivo no es construir un sistema complejo, sino **definir una estructura clara y funcional** que sirva de base para integrar los distintos módulos del sistema logístico (pedidos, flota, CRM e IoT).

Para ello, hemos utilizado el modelo **C4**, que nos permite representar la arquitectura desde distintos niveles de detalle:

1. Nivel 1 – Contexto general del sistema
2. Nivel 2 – Contenedores principales
3. Nivel 3 – Componentes internos de la API
4. Nivel 4 – Lógica de integración (decisiones 0/1)

4.1 Identificación de actores

Antes de definir la estructura técnica, identificamos los **actores principales** que interactúan con la plataforma:

Actor	*Descripción*	*Interacción principal*
Administrador	*Supervisa el funcionamiento general del sistema.*	*Gestiona usuarios, pedidos e incidencias.*
Supermercados / Proveedores	*Representan los puntos de origen de los pedidos.*	*Crean y actualizan pedidos en el sistema.*
Conductores / Repartidores	*Encargados de realizar las entregas.*	*Consultan rutas y notifican estados de entrega.*
Clientes finales	*Reciben los pedidos en destino.*	*Consultan el estado de su pedido o reciben notificaciones.*

Además de los actores humanos, el sistema se conecta con dos servicios externos simulados:

- CRM: para obtener datos de clientes y proveedores
- IoT: para recibir lecturas de ubicación y temperatura de los vehículos

4.2 Componentes principales

En el sistema identificamos cuatro contenedores principales, que representan las partes fundamentales de la arquitectura:

Contenedor	*Tipo*	*Responsabilidad principal*
Frontend Web / Postman	*Interfaz de usuario*	*Permite a los distintos actores crear pedidos, consultar estados e incidencias.*
API Unificada	*Backend REST (núcleo del sistema)*	*Orquesta los datos, gestiona la lógica de negocio, y conecta CRM, IoT y BBDD.*
BBDD Central (Pedidos/Flota)	*Persistencia de datos*	*Almacena y gestiona pedidos, vehículos, incidencias y usuarios.*
CRM / IoT simulados	*Servicios externos*	*Simulan datos de clientes, proveedores, ubicación y temperatura.*

Comunicación:
Los componentes se comunican mediante **peticiones REST/JSON**, y la API centraliza toda la lógica de negocio. La BBDD se conecta con la API a través de SQL y los servicios externos (CRM e IoT) lo hacen mediante peticiones HTTP simuladas

4.3 Definición de flujos de información

El flujo principal de información se puede resumir de la siguiente manera:

1. Un actor (por ejemplo, un proveedor) realiza una petición a la API desde el Frontend (crear o consultar pedido).
2. La API Unificada recibe la solicitud y consulta los sistemas externos:
    - CRM (para validar clientes o proveedores).
    - IoT (para obtener el último estado de los vehículos).
3. Si las respuestas son válidas, la API guarda los datos consolidados en la BBDD Central.
4. Si algún sistema no responde (por ejemplo, CRM no disponible), se registra una incidencia y se notifica al administrador.
5. Finalmente, la API devuelve al usuario una respuesta en formato JSON con la información actualizada

Ejemplo práctico de flujo:

- El administrador o proveedor crea un pedido desde el Frontend.
- La API valida los datos con el CRM.
- Consulta el estado del vehículo en el IoT.
- Guarda el pedido y la información de la flota en la BBDD.
- Si hay un error (por ejemplo, lectura IoT no válida), genera una incidencia

4.4 Diagrama y descripción

A continuación se muestran los diagramas que representan los distintos niveles de la arquitectura del sistema

Nivel 1: Diagrama de Contexto

En este primer nivel quisimos representar **el entorno global en el que se mueve el sistema**. El objetivo no era solo mostrar quién interactúa con la plataforma, sino dejar claro **dónde empieza y dónde termina su responsabilidad**. Situamos al sistema *Fresh and Go* en el centro porque actúa como eje entre los distintos actores humanos (administrador, proveedores, conductores, clientes) y los sistemas externos (CRM e IoT).Este nivel nos ayudó a **definir los límites de nuestra solución**, entendiendo qué procesos pertenecen a la empresa (gestión de pedidos y entregas) y cuáles dependen de fuentes externas (datos de clientes o lecturas IoT). A partir de este análisis inicial pudimos establecer las conexiones necesarias entre los módulos, asegurando una comunicación clara y bien delimitada

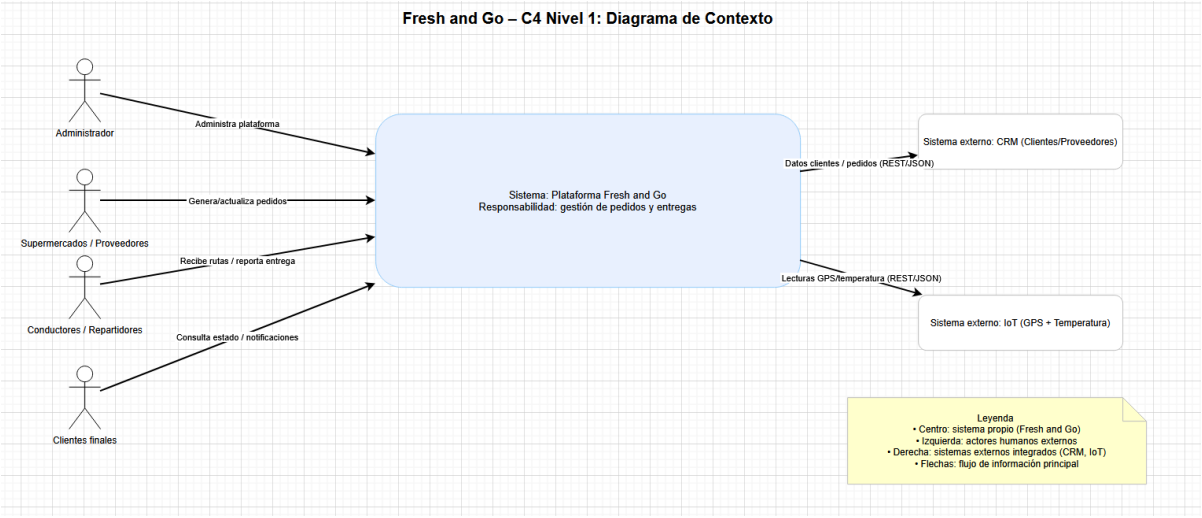

Ilustración 1: Representa la vista general del sistema Fresh and Go, mostrando su relación con los actores humanos (administrador, proveedores, conductores y clientes) y los sistemas externos (CRM e IoT) que participan en el flujo de información

Nivel 2: Diagrama de Contenedores

El diagrama de contenedores nos permitió **estructurar el sistema internamente**, identificando qué partes lo componen y cómo se comunican entre sí. Dividimos la solución en tres capas principales: interfaz (Frontend), lógica (API) y datos (BBDD). Con este enfoque conseguimos **separar responsabilidades**, lo que facilita futuras ampliaciones sin afectar al resto del sistema.

Aquí se decidió que toda la comunicación pase por la **API Unificada**, de modo que ni el Frontend ni los servicios externos accedan directamente a la base de datos. Esta decisión aporta **seguridad, trazabilidad y control**, además de permitir el registro de incidencias o errores centralizados.
También hace posible que la API gestione integraciones externas (CRM e IoT) sin modificar la estructura del resto del sistema.

Este nivel refleja, por tanto, la **arquitectura lógica** de la solución y su alineación con los objetivos de integración planteados en el proyecto

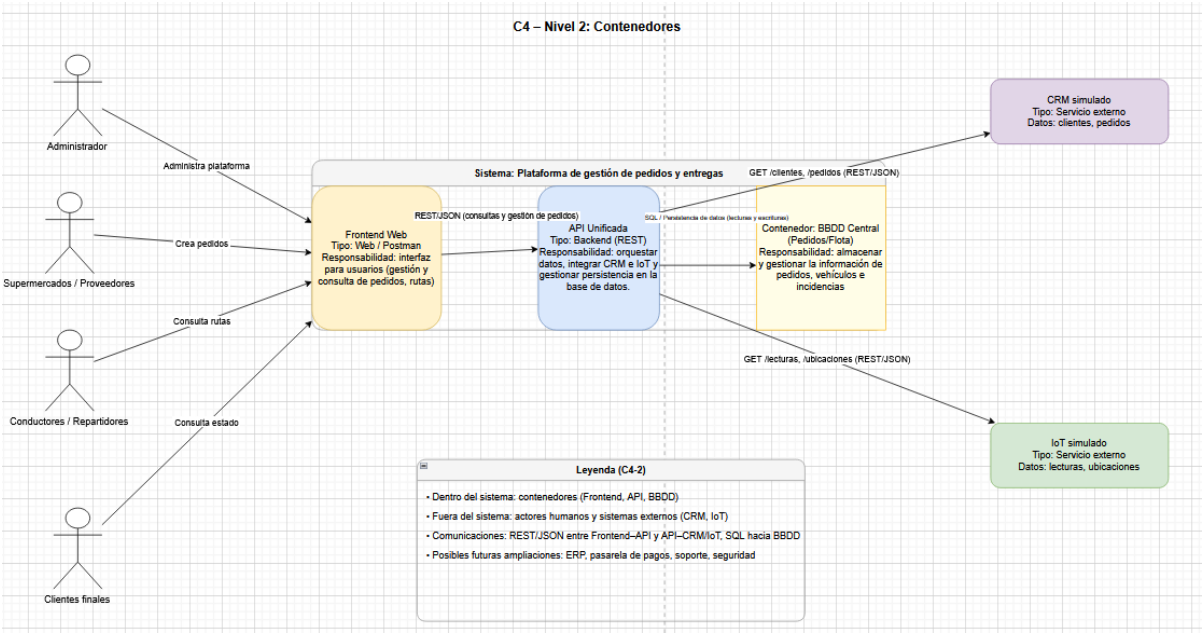

ILUSTRACIÓN 2: MUESTRA LA ESTRUCTURA GENERAL DEL SISTEMA COMPUESTA POR LOS CONTENEDORES PRINCIPALES: FRONTEND, API UNIFICADA, BBDD CENTRAL Y SERVICIOS EXTERNOS. REFLEJA LAS COMUNICACIONES REST/JSON Y LAS CONSULTAS SQL ENTRE LOS DISTINTOS MÓDULOS

Nivel 3: Diagrama de Componentes

En este nivel nuestro propósito fue mostrar **cómo se gestionan las peticiones**, **dónde se aplican las reglas de negocio** y **cómo se controlan los accesos a los datos**.

Diseñamos la API siguiendo una arquitectura en capas, con módulos bien diferenciados (controladores, servicios, repositorios y adaptadores). Esta estructura modular nos permite modificar o ampliar partes concretas (por ejemplo, la lógica de negocio o la conexión con la base de datos) sin afectar al resto.

Una decisión importante fue incluir **adaptadores para los servicios externos** (CRM e IoT), de forma que la API pueda simular integraciones reales sin depender de un entorno en la nube.

Gracias a esto, el sistema es fácilmente ampliable a futuro: bastaría reemplazar los adaptadores simulados por conexiones reales si la empresa lo necesitara.

En resumen, este nivel demuestra que la API está pensada para **crecer de forma controlada**, manteniendo simplicidad y claridad en las interacciones internas.

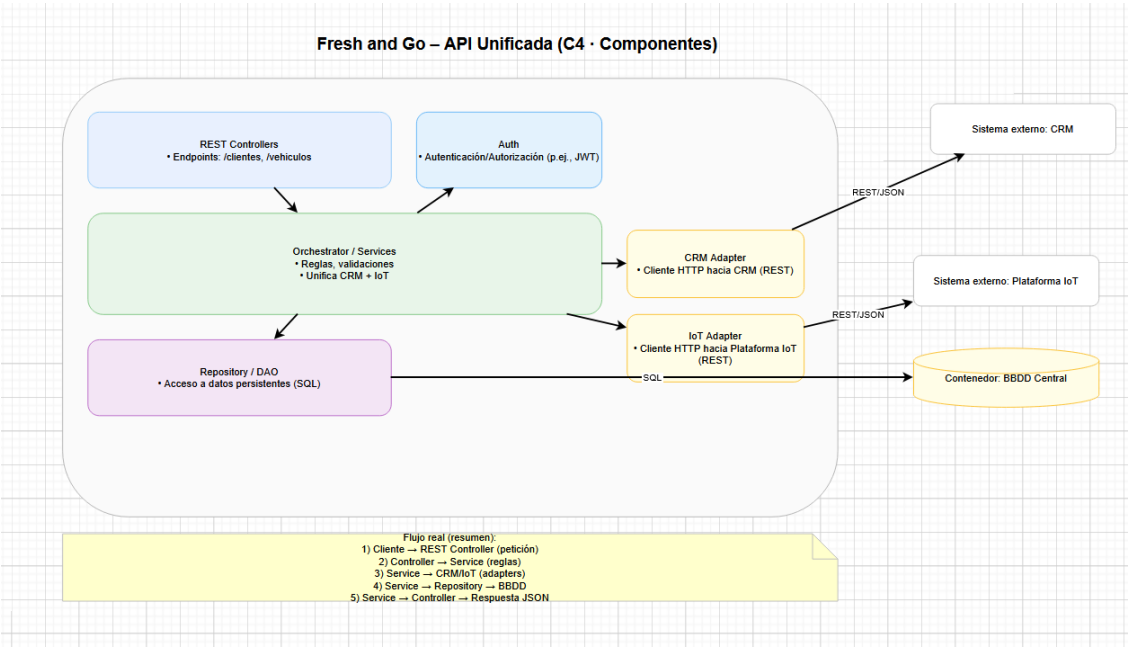

Ilustración 3: Detalla la composición interna de la API Unificada, destacando los componentes responsables de la gestión de peticiones, validaciones, lógica de negocio y acceso a la base de datos, además de los adaptadores para CRM e IoT

Nivel 4: Diagrama de Lógica de Integración (con decisiones 0/1)

Este último nivel muestra **cómo responde la API ante situaciones reales de error o falta de datos**, algo que ocurre con frecuencia en los entornos logísticos. Durante el diseño quisimos reflejar de forma visual **las decisiones críticas del sistema** cuando uno de los servicios externos (CRM o IoT) no responde o devuelve información no válida. La representación mediante decisiones binarias (0/1) nos permite simplificar el flujo y dejar claro qué ocurre en cada caso:
si el sistema externo responde correctamente (1), el proceso continúa;
si no lo hace (0), se genera automáticamente una **incidencia**, que queda registrada en la base de datos.

Este nivel no busca mostrar procesos técnicos complejos, sino **demostrar la capacidad de la API para gestionar errores y mantener la integridad del sistema**. De esta forma, se asegura que el usuario siempre reciba una respuesta, incluso cuando parte de la información proviene de servicios no disponible

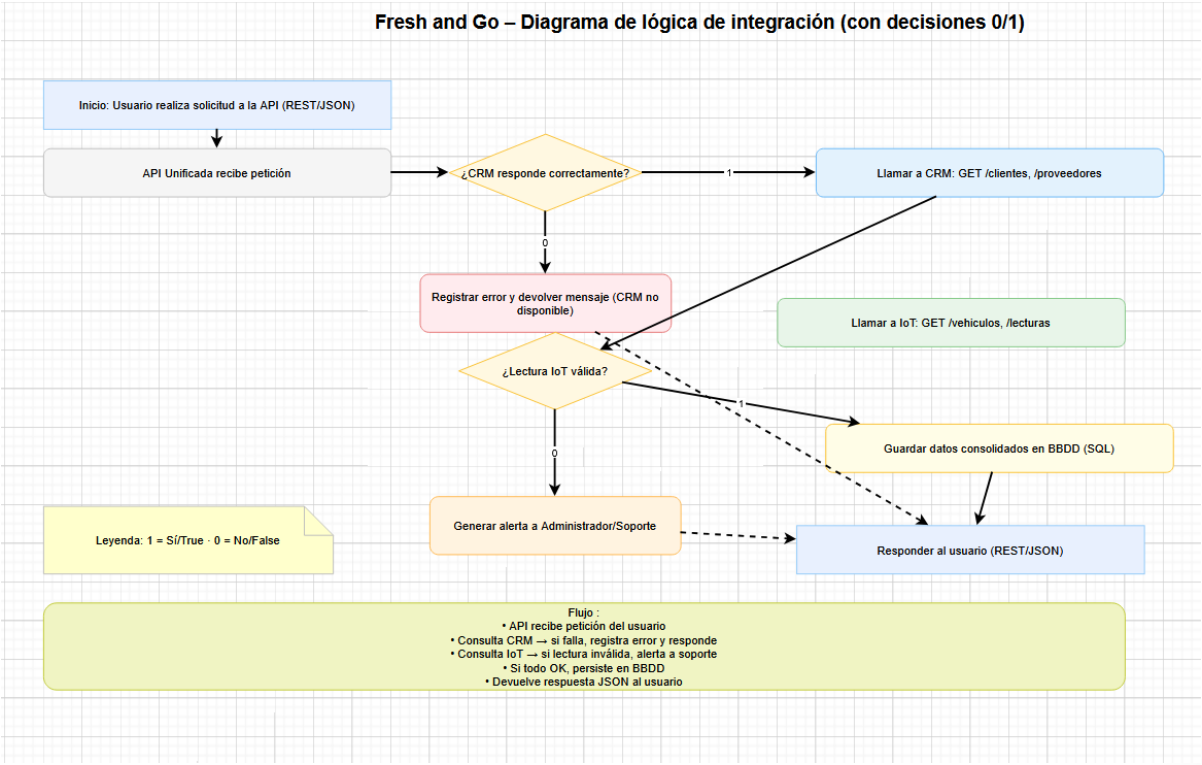

ILUSTRACIÓN 4: DESCRIBE EL FLUJO LÓGICO DE LA API AL COMUNICARSE CON CRM E IOT. MUESTRA LAS DECISIONES BINARAS (0/1) QUE DETERMINAN EL REGISTRO DE INCIDENCIAS O LA ACTUALIZACIÓN DE DATOS EN LA BASE DE DATOS SEGÚN LA RESPUESTA RECIBIDA

El contexto nos ayudó a definir límites, los contenedores a distribuir funciones, los componentes a estructurar la API y la lógica a prever errores

5. Conclusiones y evolución futura

5.1. Beneficios esperados

Con el desarrollo de este proyecto hemos podido sentar las bases para una plataforma logística unificada, que centraliza la información de pedidos, vehículos, usuarios e incidencias.
Aunque se trata de una versión inicial y simplificada, el diseño nos permite entender cómo una empresa del sector puede digitalizar sus operaciones y mejorar la comunicación entre los distintos actores.

Entre los principales beneficios que esperamos destacar se encuentran:

- Integración y centralización de la información: todos los módulos (pedidos, flota e incidencias) quedan conectados a través de la API Unificada, evitando duplicidades y mejorando la trazabilidad de los datos.
- Eficiencia operativa: la automatización de tareas básicas, como la asignación de pedidos o el registro de incidencias, reduce errores humanos y agiliza el trabajo diario.
- Transparencia en la gestión: la plataforma permite consultar el estado actualizado de pedidos y vehículos, ofreciendo una visión más clara para administradores y conductores.
- Escalabilidad: la arquitectura modular facilita añadir nuevas funcionalidades sin modificar la estructura existente.
- Base para una futura integración real: aunque actualmente los sistemas externos (CRM e IoT) son simulados, la API ya está preparada para conectarse con fuentes reales de datos en el futuro

En conjunto, el proyecto proporciona una solución práctica y flexible, que puede evolucionar hacia un entorno de trabajo más automatizado y conectado

5.2. Posibles ampliaciones

Durante el desarrollo del sistema identificamos varias **mejoras y ampliaciones** que podrían implementarse en futuras versiones del proyecto.
No forman parte del alcance actual, pero se han tenido en cuenta en el diseño para facilitar su incorporación posterior.

Algunas de las más relevantes son:

- **Optimización de rutas:** integrar un módulo que calcule automáticamente las rutas más eficientes según ubicación, tipo de entrega y horario.
- **Integración real con servicios IoT:** conectar sensores de temperatura y geolocalización reales para registrar lecturas en tiempo real.
- **Conexión con un ERP o sistema de inventario:** permitir que el sistema logístico se sincronice con los datos de stock de los supermercados o centros de distribución.
- **Sistema de notificaciones al cliente:** enviar alertas automáticas (por correo o SMS) sobre el estado de los pedidos.
- **Panel analítico (dashboard):** incluir métricas de rendimiento, entregas por día, incidencias y niveles de temperatura promedio.
- **Despliegue en entorno cloud:** alojar la aplicación en la nube para facilitar el acceso remoto y la escalabilidad

Estas ampliaciones no solo mejorarían la funcionalidad del sistema, sino que también lo acercarían a una solución profesional completa, manteniendo la misma arquitectura y filosofía de diseño planteadas en esta versión inicial

FRESH&GO

DE LA TIENDA A TU MESA EN UN CLICK