

RetailSync

Une première application pour comparer
SOAP, REST, GraphQL et gRPC
dans un système e-commerce

Module d'Architecture Logicielle

Février 2026

Au programme

- | | |
|--|-------|
| ① Le Contexte – Pourquoi utiliser 4 APIs différentes ? | 2 min |
| ② L'Architecture – Comment tout se connecte | 3 min |
| ③ Plongée dans les 4 APIs – Démonstrations et avantages | 6 min |
| ④ Comparaison Pratique – Taille des messages et décisions | 5 min |
| ⑤ Conclusion – Ce qu'il faut retenir | 2 min |

SOAP

REST

GraphQL

gRPC

Le problème : 4 besoins très différents

Nous simulons une chaîne de magasins qui doit communiquer avec **quatre acteurs différents**. Chacun a ses propres contraintes :

SOAP Les Usines (Achats B2B)

Vieux systèmes informatiques (SAP) très stricts. On a besoin d'une garantie totale sur le format des commandes, sans erreur.

REST Les Boutiques (Partenaires)

Veulent gérer leur stock facilement avec n'importe quel langage de programmation. Il faut faire très simple.

GraphQL Le(la) Gérant(e) (Tableau de bord)

A besoin de voir toutes les infos (stock, commandes) sur un seul écran, sans faire 10 requêtes différentes qui ralentissent le site.

gRPC Les Robots (Dans l'entrepôt)

Envoient leur position en continu (10 fois par seconde). Il faut que le message soit microscopique pour ne pas bloquer le réseau.

Le code vient après

Pour chaque API de ce projet, nous avons suivi une règle stricte :

- ① **Définir le “Contrat”** – On écrit d'abord un fichier qui décrit comment l'API va marcher.
- ② **Coder le serveur** – Le code est créé à partir du contrat, jamais l'inverse.
- ③ **Tester avec Postman** – Pour vérifier que ça fonctionne bien !

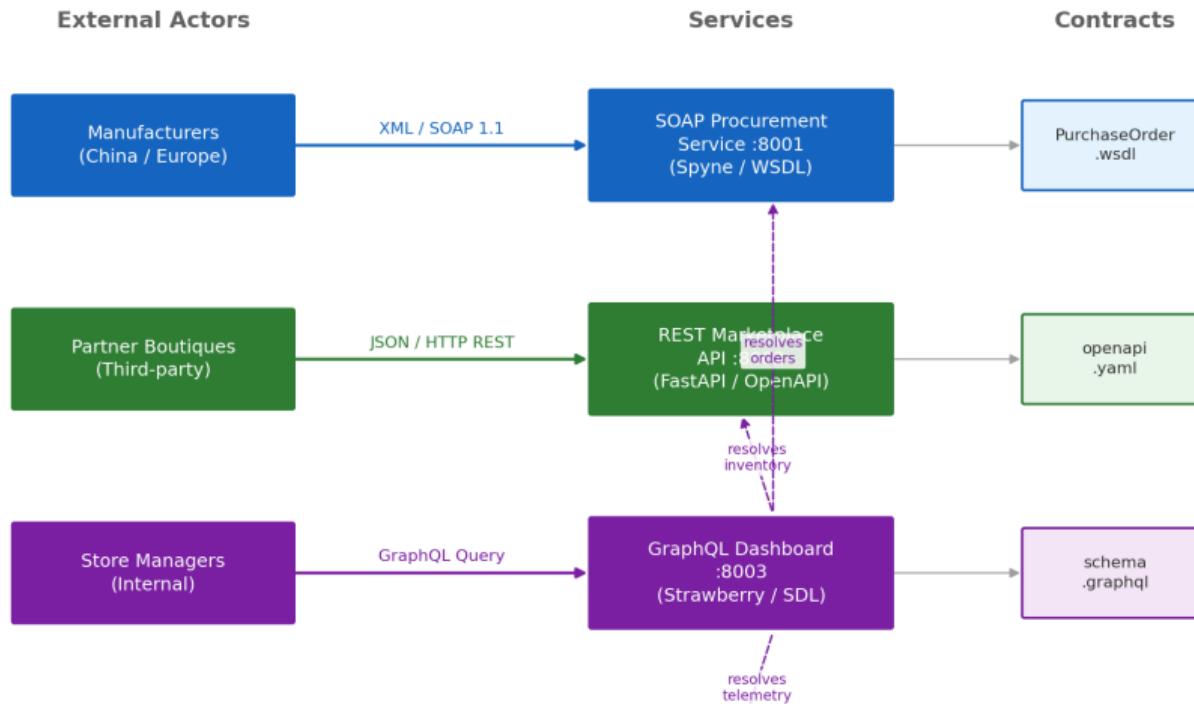
| Module | Le fichier Contrat |
|---------|--------------------|
| SOAP | PurchaseOrder.wsdl |
| REST | openapi.yaml |
| GraphQL | schema.graphql |
| gRPC | warehouse.proto |

Idée à retenir

En séparant la règle (le contrat) du code, l'application est beaucoup plus facile à modifier et tester.

Vue d'ensemble du Projet

RetailSync — System Architecture

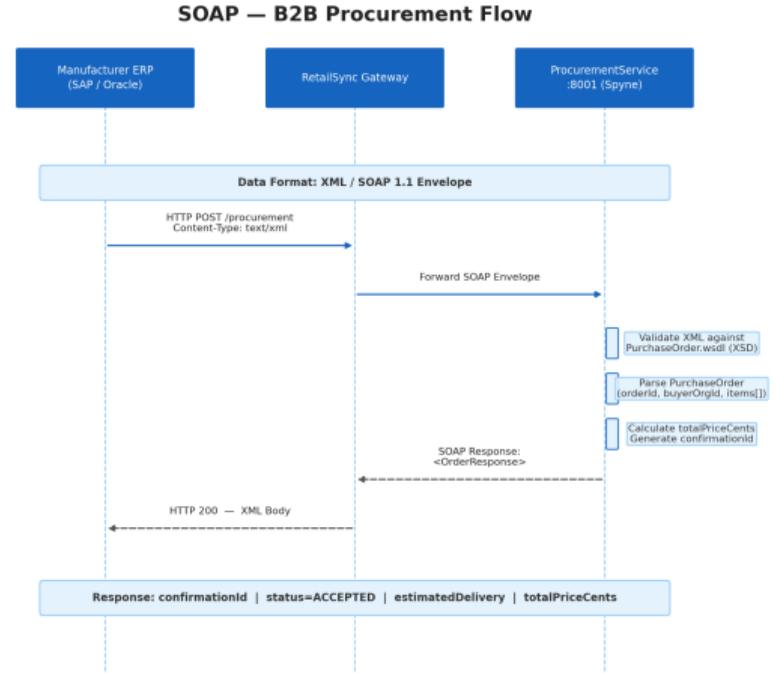


Format : Lourd mais ultra sécurisé (XML)

Pourquoi on l'a choisi :

- **Validation parfaite :** Impossible d'oublier un champ grâce au fichier WSDL qui vérifie tout.
- **Format d'entreprise :** C'est ce que les gros logiciels SAP/Oracle attendent.

- | | |
|-----------------------|-----------------------|
| + Erreurs impossibles | - XML dur à lire |
| + Très sécurisé | - Outils un peu vieux |



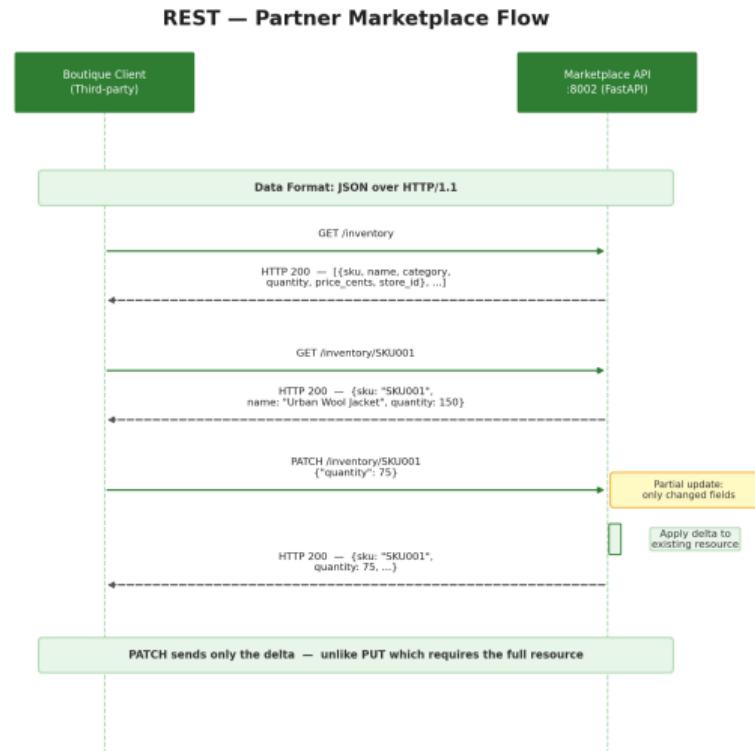
Format : Le grand classique du net (JSON)

Pourquoi on l'a choisi :

- **Universel** : Tout le monde sait faire une requête GET ou PATCH.
- **Très facile à mettre en place** : Aucun fichier complexe à générer. Pas besoin d'outils compliqués.

+ Hyper simple
+ Rapide à coder

- Pas de validation forte
- Vieux design par requêtes séparées



Format : Flexible (JSON)

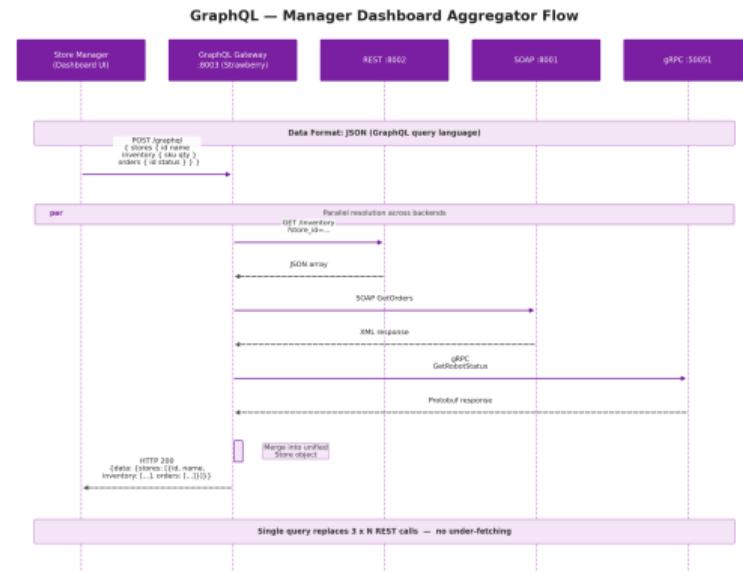
Pourquoi on l'a choisi :

- **Requête unique** : Parfait pour une interface web qui veut charger plein d'informations à la fois.
- **À la carte** : On ne télécharge que les informations qu'on veut afficher sur l'écran.

+ Une seule requête
HTTP

+ Économise internet

- Nouveau concept à apprendre
- Difficile de mettre en cache



Format : Binaire ultra léger (Protobuf)

Pourquoi on l'a choisi :

- **Microscopique :** Un message fait 40 octets (contre 250 en REST). On gagne 6x de vitesse!
- **Temps réel :** Le serveur et le robot se parlent en continu sans refaire de nouvelle connexion (streaming bidirectionnel).

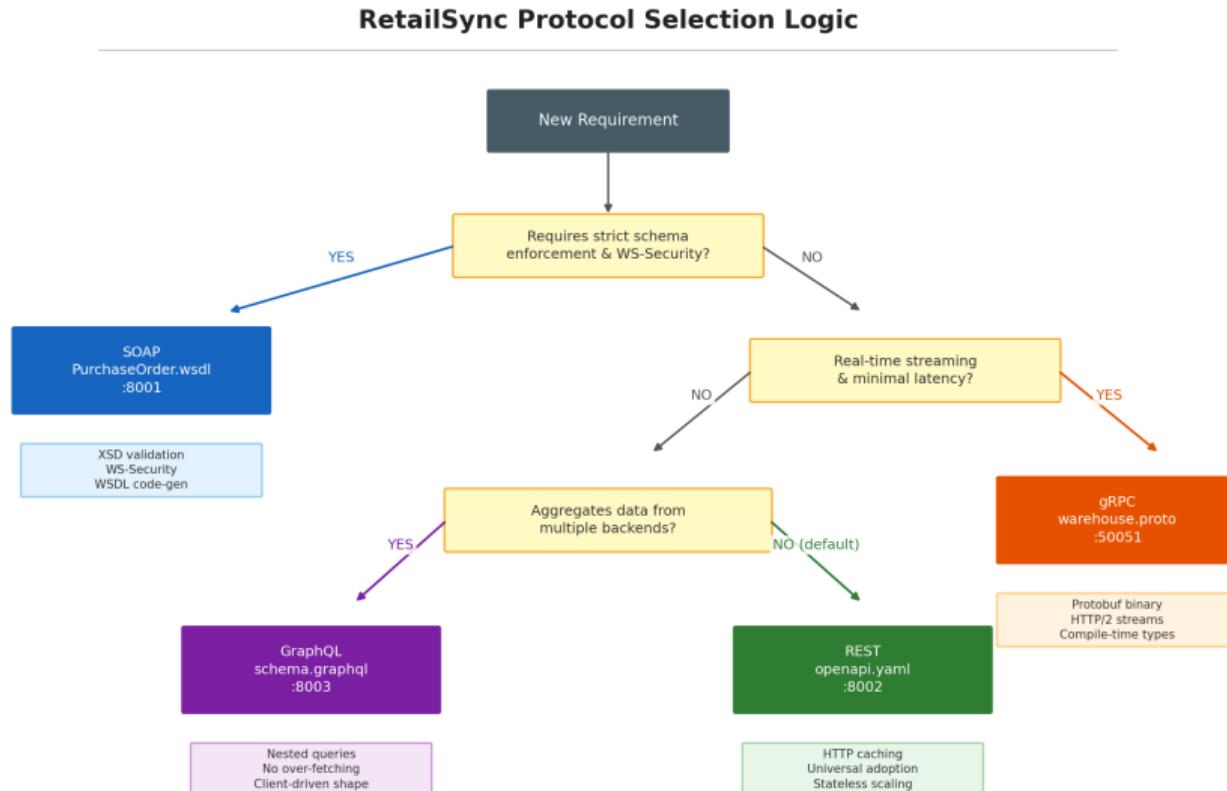
+ Ultra rapide / léger
+ Streaming réel

- Illisible sans outil
- Difficile pour le web

gRPC — Warehouse Robot Bi-directional Streaming



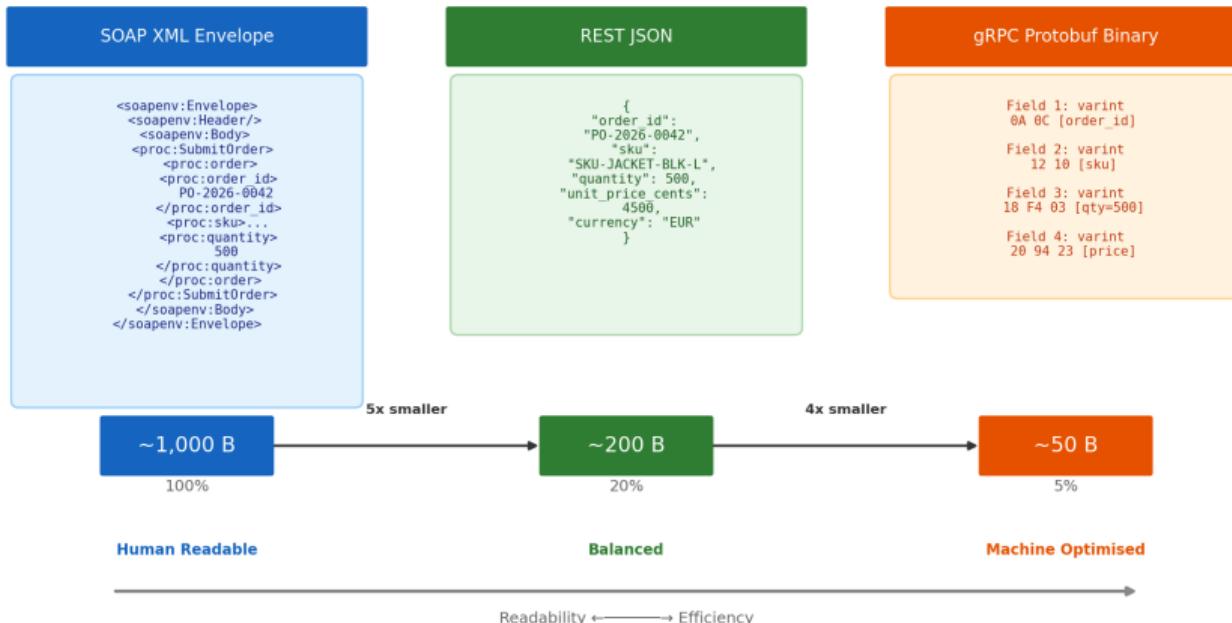
Pourquoi telle ou telle API ? (Arbre de décision)



Comparatif de taille des messages

Payload Overhead Comparison — Order Object

Same business data (SKU, quantity, price) encoded in three wire formats



GraphQL, le "Chef d'Orchestre"

GraphQL: The Omnichannel Unified Interface

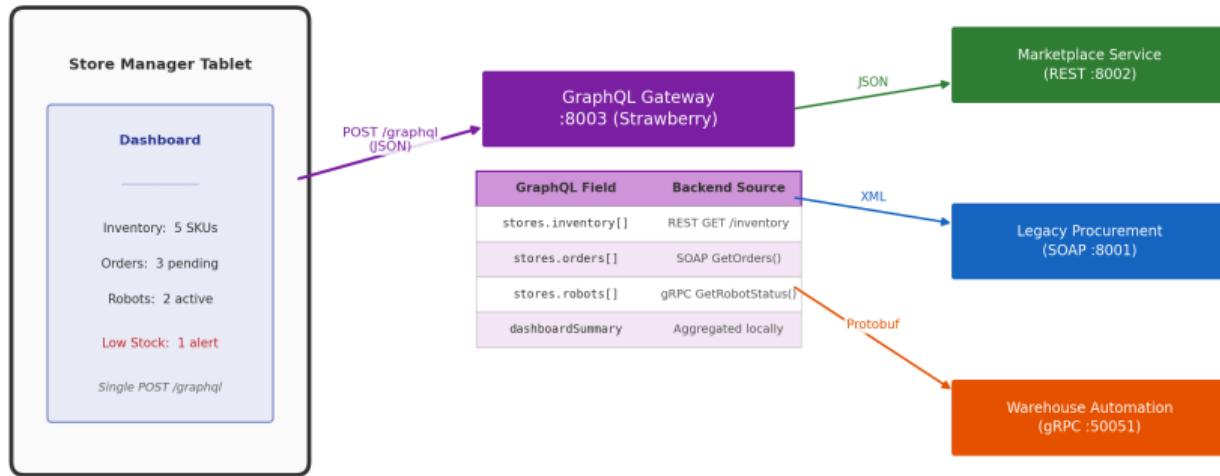


Tableau de Synthèse

| Critères | SOAP | REST | GraphQL | gRPC |
|------------------|----------------------|------------------|-----------------------|-----------------------|
| Format | XML | JSON | JSON | Protobuf |
| Connexion | HTTP/1.1 | HTTP/1.1 | HTTP/1.1 | HTTP/2 |
| Sécurité format | Très stricte (WSDL) | Moyenne | Flexible | Stricte (.proto) |
| Vitesse/Taille | Très lent (Lourd) | Basique | Economique | Ultra rapide (Léger) |
| Streaming Direct | Non | Non | Partiel | Oui (Dans les 2 sens) |
| Le Meilleur Pour | Tâches Critiques/B2B | Public et Simple | Sites Web (Affichage) | Robots et Temps Réel |

Il n'y a pas d'API parfaite. Il n'y a que le **bon type d'API pour le bon besoin.**

Que faut-il retenir de ce POC ?

- ➊ **Il n'y a pas de gagnant final.** Chaque API fait son travail parfaitement en fonction du besoin. L'avenir est de savoir les utiliser au bon endroit.
- ➋ **REST est notre valeur sûre.** C'est le choix facile et intelligent pour 80% des projets classiques.
- ➌ **Le contrat avant le code.** Créer des règles (fichiers wsdl, proto, openapi) rend l'équipe de développement beaucoup plus autonome et réduit les erreurs.
- ➍ **GraphQL n'efface pas REST.** Il se met par-dessus, pour rendre l'interface web plus simple à utiliser sans tout casser derrière.
- ➎ **La vitesse gRPC est indispensable.** Pour des machines comme les robots travaillant 10 fois par seconde, SOAP ferait s'écrouler le réseau wifi de l'entrepôt, gRPC le laisse respirer.

Merci pour votre attention !

Avez-vous des questions ?

SOAP

:8001

REST

:8002

GraphQL

:8003

gRPC

:50051

github.com/AyaMor/omnichain-retail-mesh