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 :

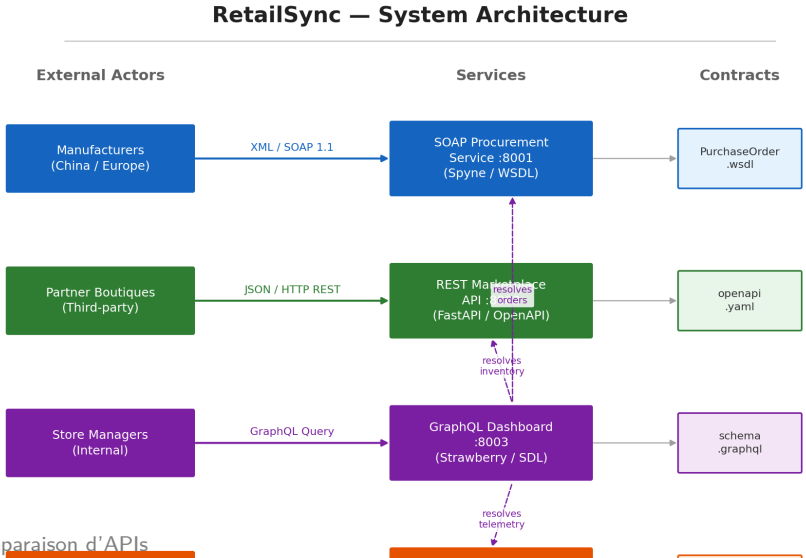
1. **Définir le "Contrat"** – On écrit d'abord un fichier qui décrit comment l'API va marcher.
2. **Coder le serveur** – Le code est créé à partir du contrat, jamais l'inverse.
3. **Tester avec Postman** – Pour vérifier que ça fonctionne bien !

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



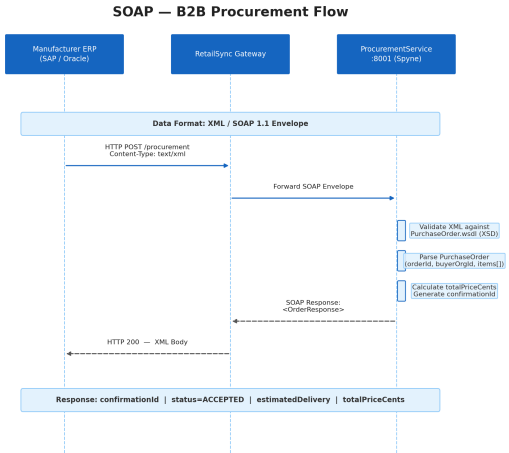
SOAP Module SOAP – Commandes aux usines

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
+ Très sécurisé
– XML dur à lire
– Outils un peu vieux



REST Module REST – Gestion de l'inventaire

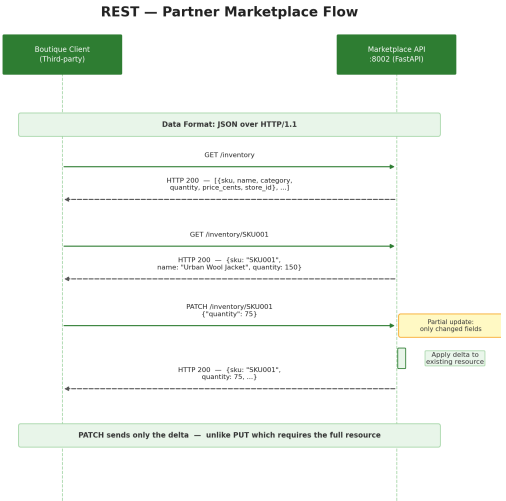
Format : Le grand classique du net (JSON)

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



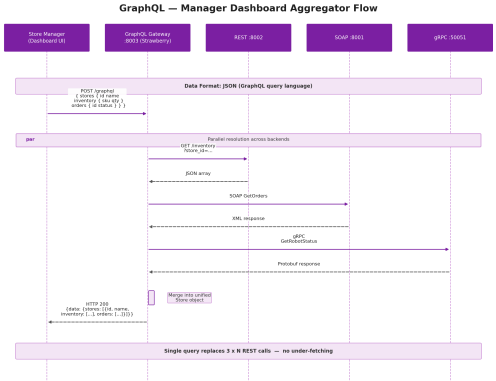
GraphQL Module GraphQL – Le Tableau de Bord

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



gRPC Module gRPC – Les Robots de l'Entrepôt

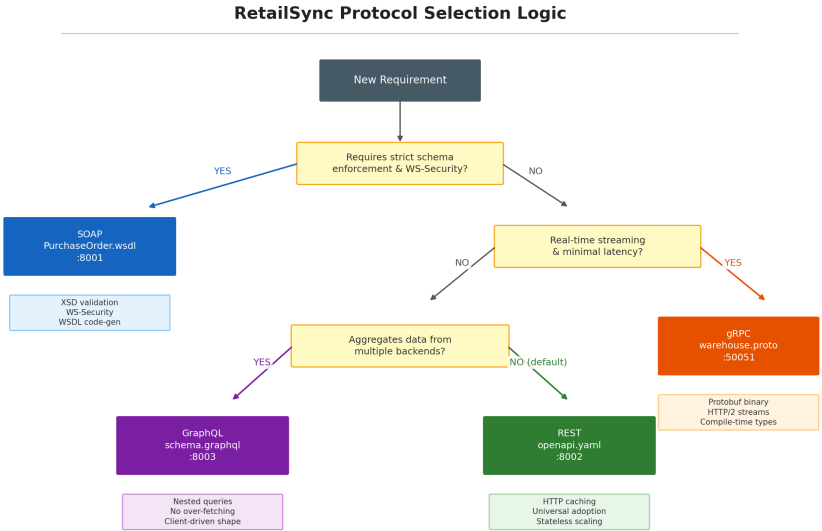
Format : Binaire ultra léger (Protobuf)

Pourquoi on l'a choisi :

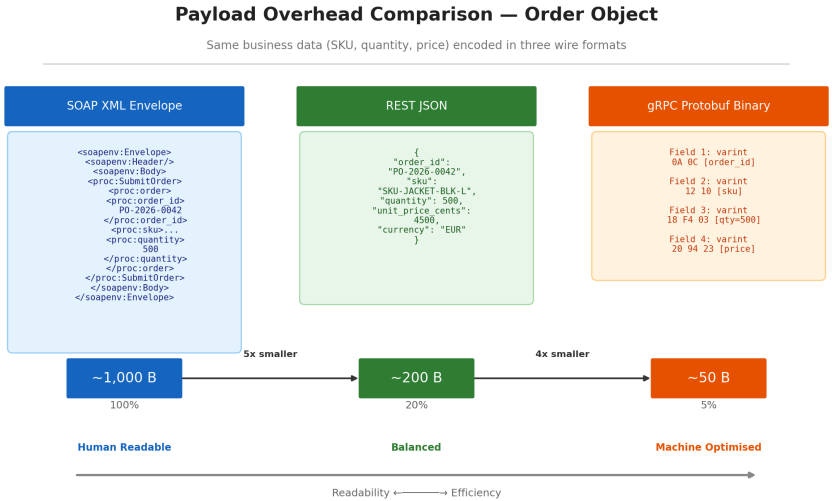
- **Microscopique :** Un message fait 40 octets (contre 250 en REST). On gagne 6x de vitesse!
- **Temps réel :** Le serveur et le robot se parlent en continu sans refaire de nouvelle connexion (streaming bidirectionnel).

+ Ultra rapide / léger
+ Streaming réel
– Illisible sans outil
– Difficile pour le web

Pourquoi telle ou telle API ? (Arbre de décision)



Comparatif de taille des messages



GraphQL, le "Chef d'Orchestre"

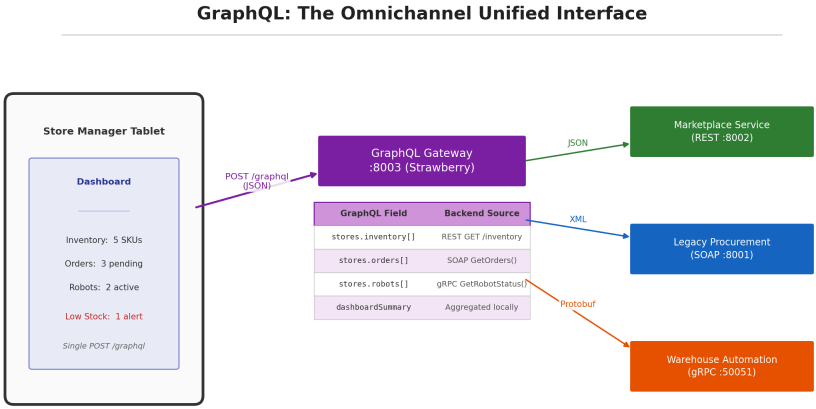


Tableau de Synthèse

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**.

Point d'Amélioration : Le Rôle de l'API Gateway

Le problème actuel (Pédagogique) :

- Notre projet expose 4 serveurs sur **4 ports différents** (8001, 8002, 8003, 50051).
- C'est parfait pour comprendre comment chaque protocole fonctionne en isolation.

La solution en production (Le monde réel) :

- On utiliserait une **Passerelle d'API (API Gateway)**.
- Elle écoute sur **un seul port unifié** (ex: port 443 pour HTTPS) et agit comme routeur central :
  - `/procurement` → Redirigé vers le serveur SOAP.
  - `/inventory` → Redirigé vers le serveur REST.
- Résultat : L'architecture *Microservices* complexe en arrière-plan est invisible pour le monde extérieur !

Que faut-il retenir de ce POC ?

1. **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.
2. **REST est notre valeur sûre.** C'est le choix facile et intelligent pour 80% des projets classiques.
3. **Le contrat avant le code.** Créer des règles (fichiers wsdl, proto, openapi) rend l'équipe de développement beaucoup plus autonome et réduit les erreurs.
4. **GraphQL n'efface pas REST.** Il se met par-dessus, pour rendre l'interface web plus simple à utiliser sans tout casser derrière.
5. **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