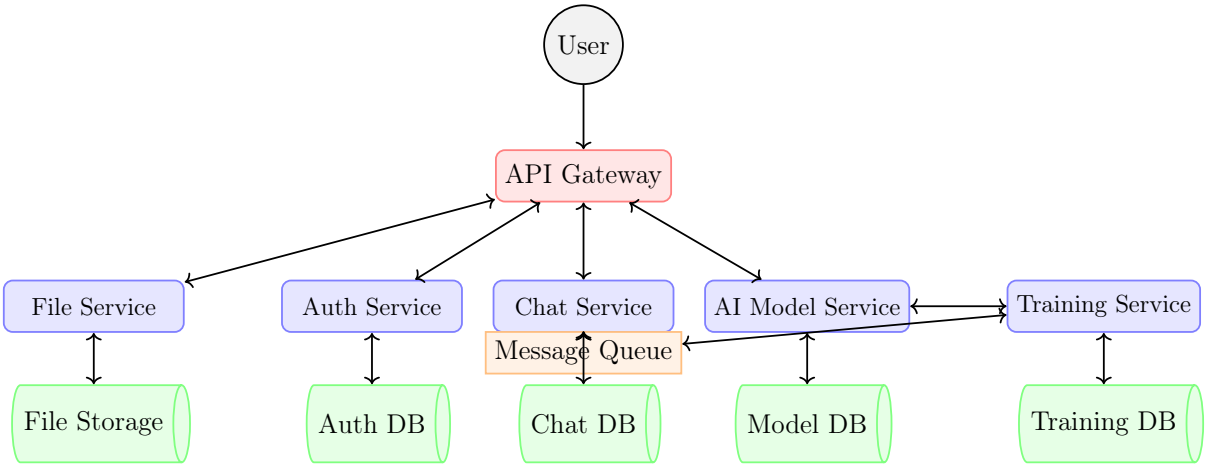
deepseek-gateway

Aness Ben Slama - Angham Regaieg 3IDL02

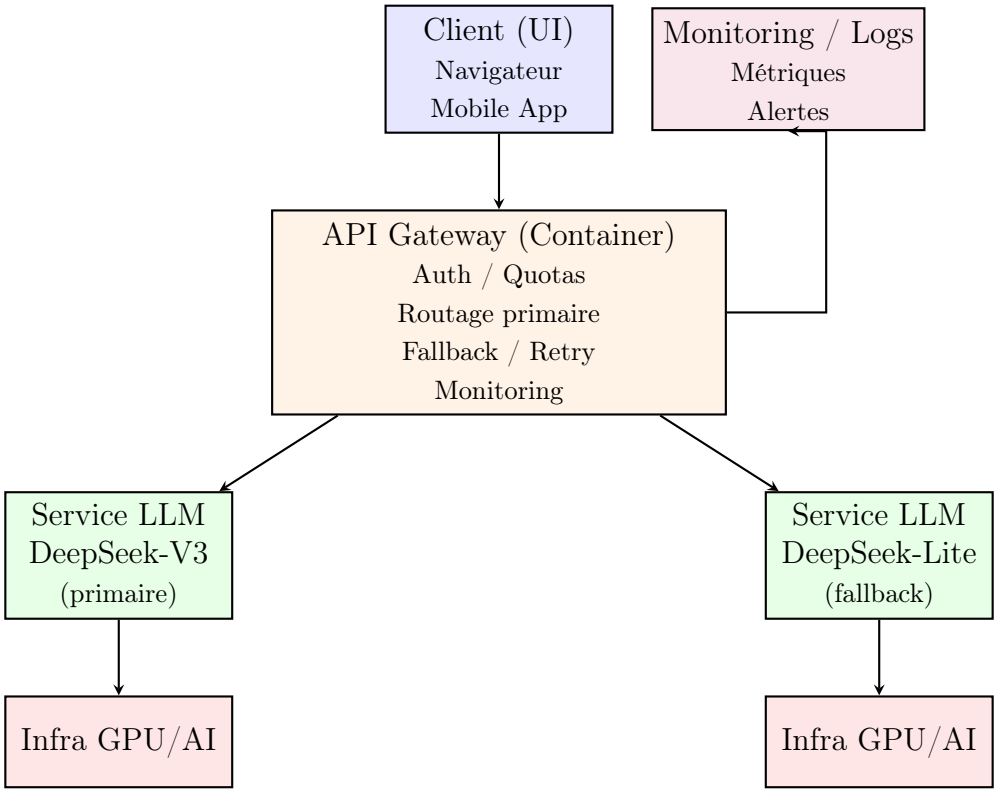
3 Octobre 2025

1 Architecture Gateway

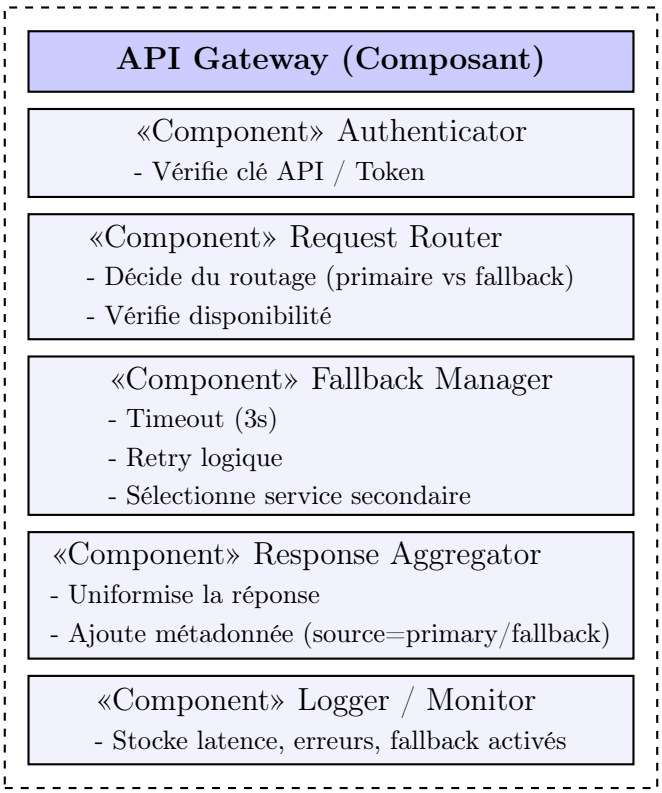
1.1 Architecture générale



1.2 Diagramme conteneurs : Vue externe



1.3 Diagramme de composants : Vue interne



1.4 Algorithme du gateway fallback

Entrées : request
Sorties : response

```
// Étape 1 : Essayer le service principal
response ← Appeler_Service(primary_service, request, timeout)
// Étape 2 : Vérifier si échec ou trop lent
si (response = ERREUR OU response = TIMEOUT) alors
    AFFICHER " Service principal indisponible → fallback activé"
    // Étape 3 : Essayer service de repli
    response ← Appeler_Service(fallback_service, request, timeout)
    // Étape 4 : Vérifier encore
    si (response = ERREUR OU response = TIMEOUT) alors
        AFFICHER " Tous les services sont indisponibles"
        response ← "Erreur système - réessayez plus tard"
    fin
    response.source ← "fallback"
fin
response.source ← "primary"
// Étape 5 : Retourner la réponse finale
response
```

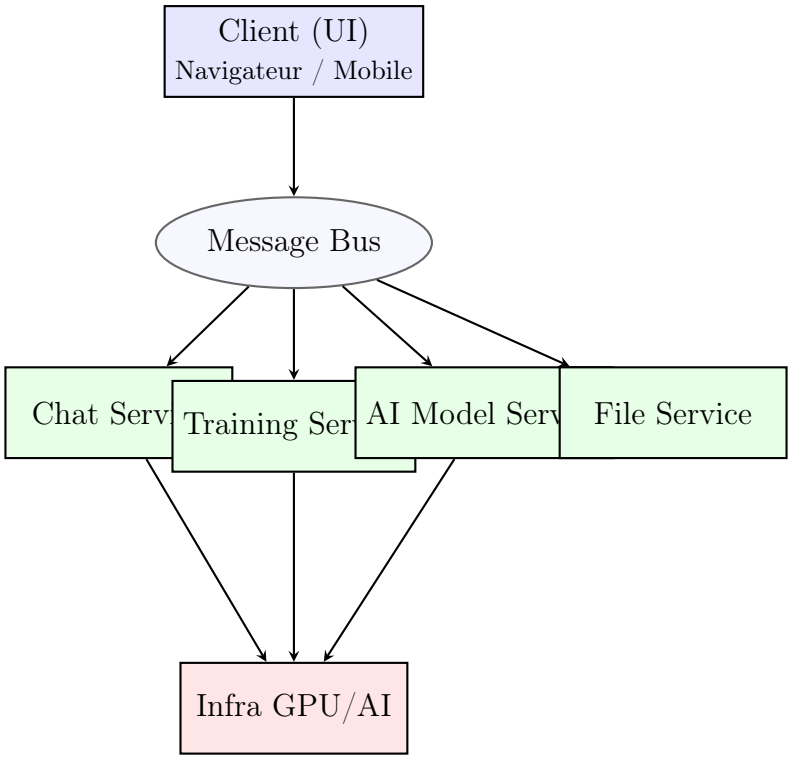
Algorithme 1 : Algorithme de Fallback Gateway

2 Architecture micro service parallèle

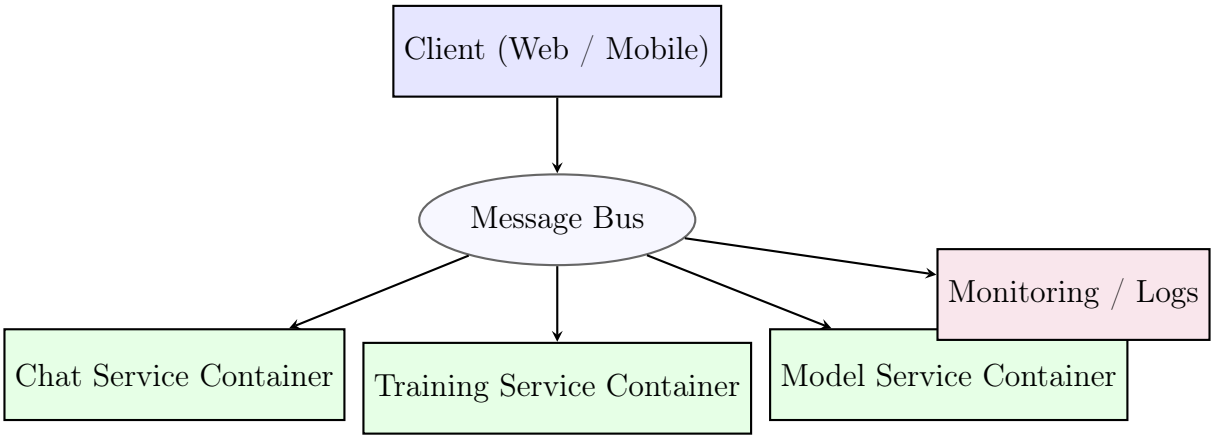
2.1 Architecture générale

— **Microservice parallèle :** On peut l'expliquer par exemple lorsque le client envoie une requête qui est difficile à répondre, le chat service et training service travaille en parallèle afin de fournir une réponse meilleure.

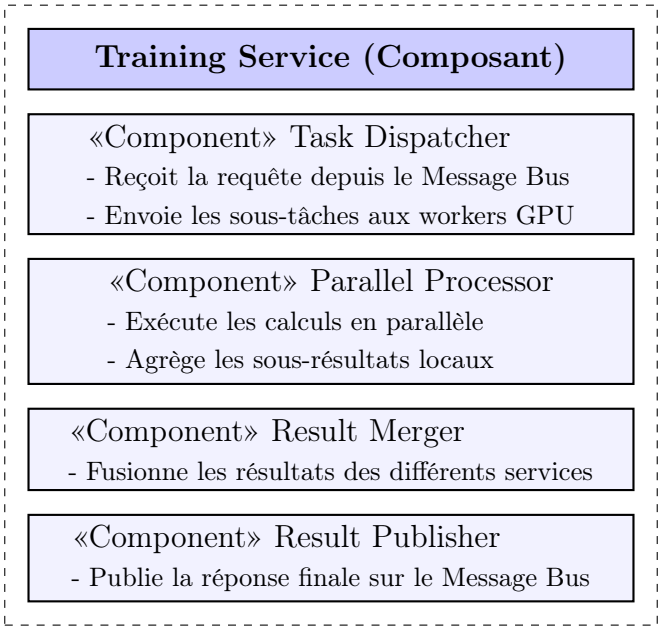
— **Message bus :** permet d'orchestrer et de synchroniser les flux entre services



2.2 Diagramme Conteneurs



2.3 Diagramme Composants



2.4 Diagramme de Code — Orchestration simplifiée

Entrées : request
Sorties : final_response

```
// Publier la requête sur le bus de messages
Publish(MessageBus, request)
// Attendre les réponses des services parallèles
responses ← CollectResponses([Chat, Training, Model], timeout)
// Agréger les réponses reçues
final_response ← Merge(responses)
// Retourner la réponse combinée
final_response
```

Algorithme 2 : Algorithme d'Orchestration Parallèle

2.5 Avantages et Limites

— **Avantages :**
  — Exécution simultanée de plusieurs services → rapidité accrue.
  — Réponses plus riches grâce à la collaboration entre microservices.
  — Haute scalabilité horizontale (ajout facile de nouveaux services).

— **Limites :**
  — Synchronisation et agrégation des résultats complexes.
  — Charge réseau importante sur le Message Bus.
  — Gestion d'erreurs plus délicate (pannes partielles possibles).

3 Architecture optimale

D'après l'analyse précédente des 2 architectures on a remarqué que :

— **Gateway :** apporte centralisation, sécurité et capacité de fallback — mais nécessite redondance et scalabilité pour éviter les SPOF (Single Point Of Failure).

— **L'architecture parallèle :** augmente la qualité et la rapidité des réponses, mais demande une orchestration robuste, de la résilience côté message bus et une gestion stricte des ressources.

Ceci nous a conduit à combiner les deux solutions, ce qui nous permet de tirer parti des forces de chacune :

— **Gateway :** offre un point d'entrée unique sécurisé pour le client, simplifiant l'authentification et la gestion des accès.

— **Message bus :** assure une orchestration fluide et une synchronisation efficace entre les services, tout en renforçant la résilience et la scalabilité de l'architecture.