Rapport SOA
API

Florian LEHMAN

Alexandre Clement

Aghiles DZIRI

Amri Haroun

Equipe K

Ressouce :

Avant de choisir les API que nous allions mettre en place, nous avons dû déterminer le moyen de communication. Pour cela, nous avons dû définir les contraintes liés à la mise en place d'un système tel qu'Uberoo ainsi que sa maintenance.

Les contraintes liées au système Uberoo sont les suivantes :

- **Passage à l'échelle :** Le système doit être capable de passer à l'échelle notamment vers 12h et 19h. En effet, à ces heures le nombre d'utilisateurs est le plus important.

- **Résilience :** La panne d'un des services doit avoir un impact minimal sur notre système. Si le catalogue vient à tomber en panne, la préparation des commandes et leurs livraisons doivent continuer à fonctionner.

Nous avons dès le début exclut les services de type RPC pour les raisons suivantes :

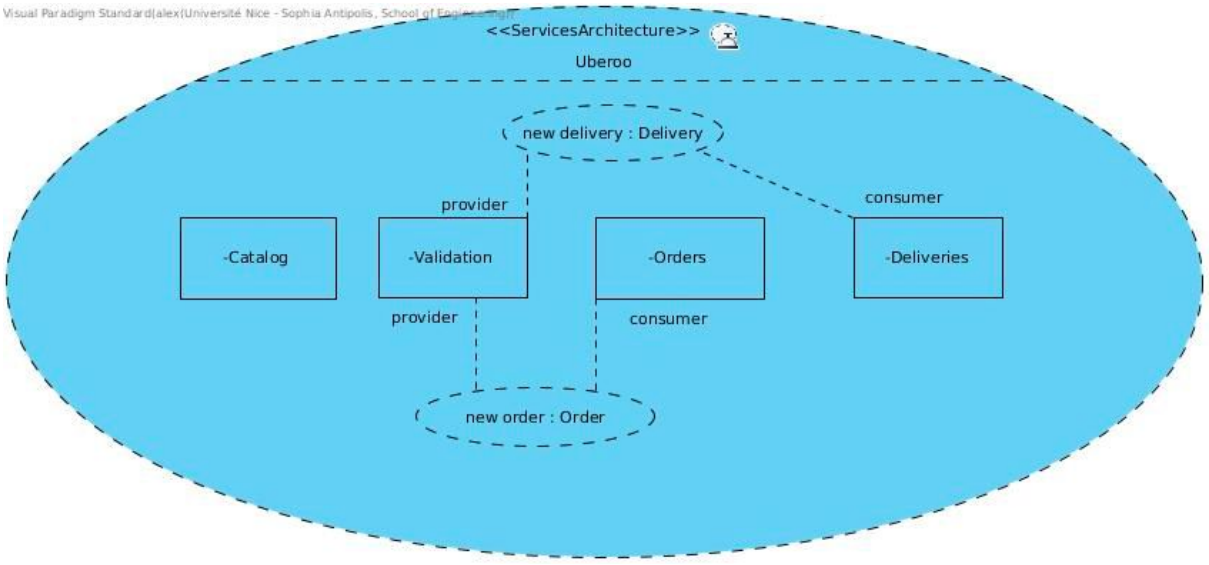
- **Évolutivité :** Ce service est récent et son API risque d'évoluer au fil du temps. Si un service décide d'ajouter de nouvelle fonctionnalités à son API alors il sera obligatoirement nécessaire de régénérer le client. De plus, un client du service Uberoo ne pourra pas utiliser le service tant qu'il n'aura pas mis à jour son application.
- **Ressources :** Il utilise plus de ressources comparé aux autre types (message et ressource) qui est dû au marshalling et à l'unmarshalling des messages.

Contrairement au type précédent, les documents (Message) permettent un couplage faible. Il ne s'adresse pas à un service en particulier mais à un ensemble de services qui aurait souscrit à l'événement produit. Il nous est apparu que les documents étaient plus approprié aux communications asynchrones.

Enfin, le type ressource a les même propriétés que le type Document mis à part qu'il s'adresse directement à un service.

C'est ce dernier type de service que nous avons utilisé dans le cadre du projet.

Choix de design :



Notre Service se compose de 4 différents modules, où chacun offre des services nécessaires au client.

- Le service Catalogue : Permet de fournir le menu qu'on peut commander.
- Le service Commands : Permet de stocker les commandes pour chaque restaurant adhérent au service Uberoo. Chaque restaurant peut ainsi consulter les commandes propres à son magasin.
- Le service Deliveries : Permet d'associer un livreur à une commande déjà faite. Chaque livreur peut consulter les livraisons auquel il est affecté.
- Le service Validation: Permet de fournir à l'utilisateur l'ETA de sa commande. Ce service permet également la validation de sa commande.

Architecture :

Chaque service correspond à une seule fonctionnalité. Chaque service peut-être dupliqué autant de fois que nécessaire. Les services communique entre eux en REST à l'aide d'une route HTTP.

Les modules Catalog, Orders et Deliveries contiennent une base données. La base de données est unique pour chacun des services.

La présence d'une seule base de donné pour l'ensemble des services nous aurait permis de en simplicité au détriomment d'un plus grand couplage. De plus, il y a un risque qu'un des services modifie la base de données. Cette modification pourrait ne pas être compatible avec les autres services. Ce changement aurait pour conséquence une coupure de service.

La cohérence peut également être impacté si plusieurs services interagissent avec la même base de données, il y a risque que deux services tentent de modifier la même ligne. Ces modifications peuvent causer un problème de cohérence.

Pour ces raisons nous avons choisi de mettre en place une base de donnée par service.

Pour la partie delivery, nous avons du faire un choix. D'une part, on peut proposer une API sur laquelle chaque livreur peut envoyer des requêtes pour obtenir les livraisons qu'il doit effectuer. Ceci peut entraîner une surcharge importante du système lorsque le nombre de livreur va augmenter. D'autre part, on peut proposer de notifier le livreur dès que celui-ci reçoit une nouvelle livraison. Cela impose de définir un contrat fort pour les notifications. Comme le projet est sujet à évoluer, et comme le nombre de livreur n'est pour le moment pas assez conséquent pour surcharger le système, on a optez pour la première option consistant à proposer une API sur laquelle le livreur doit envoyer une requête de manière périodique pour savoir si il a une nouvelle livraison à effectuer. Ce choix n'est cependant pas définitif et cette partie devrait surement évoluer par la suite.

Description de l'API avec les autres styles de service :

Resource :

- GET /products (liste de produits du catalogue)
- POST /commands (ajouter une commande)
- GET /commands (liste des commande disponible)
- PUT /commands/{command_id} (modification d'une commande)
- DELETE /commands/{command_id} (supprimer une commande)
- GET /commands/{command_id} (retourne la commande avec id = command_id)
- GET /deleveryman/{id} (liste des livraisons pour le livreure à l'id correspond)
- POST /adddelivery (ajouter une livraison)
- POST /neworder (ajouter une nouveau menu au catalogue)
- GET /restaurants/{restaurantId}/orders (liste des menus d'un restaurant)
- GET /products/{productID}/eta/{address} (ETA de la commande "temps d'attente")
- POST /validate/product (validation d'une commande)