Université Nice Sophia Antipolis

POLYTECH

Uberoo

Equipe K:

Aghiles Dziri
Alexandre Clement
Amri Haroun
Florian Lehmann

Dans le cadre de ce projet, nous avons choisi de repartir de zéro en prenant en compte les remarques qui nous ont été faites lors de notre dernier rendu.

Environnement technique :

Nous avons choisi d'utiliser des technologies dans lesquels nous étions à l'aise afin de ne pas ajouter de complexité supplémentaire liée à l'apprentissage d'une nouvelle technologie. Ainsi, nous avons utilisé J2EE, EJB, PostgreSQL, Docker (version 3.5).

Week 41:

Diagramme de cas d'utilisation

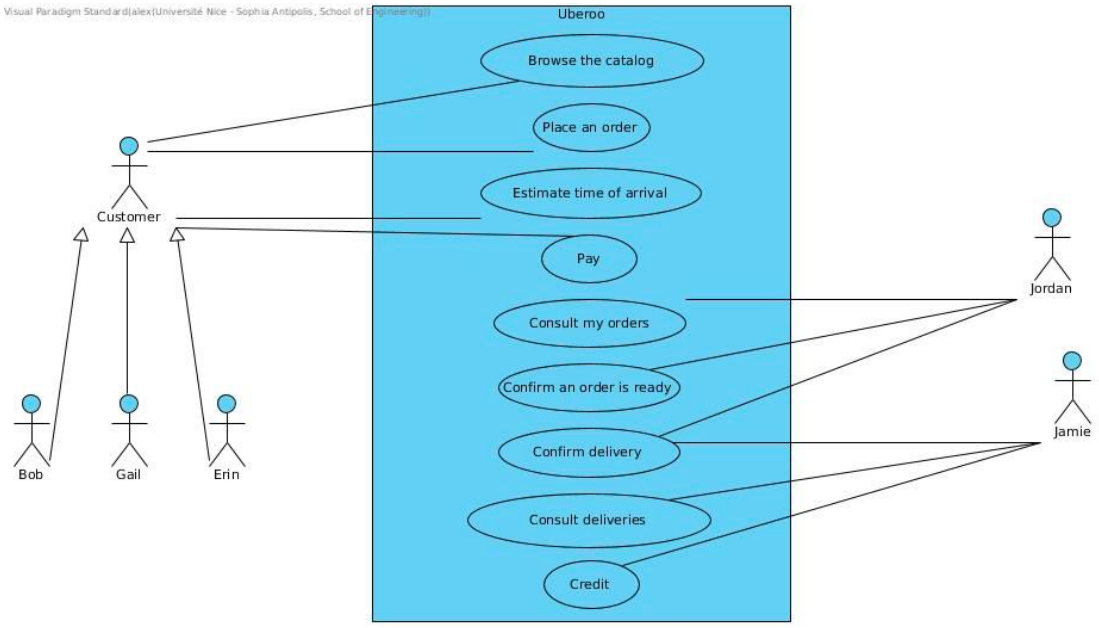

Figure 1 : Diagramme de cas d'utilisation

Architecture

Après analyse des différentes *user story* demandé par le *product owner*, nous avons décidé de réaliser l'architecture microservice suivante :

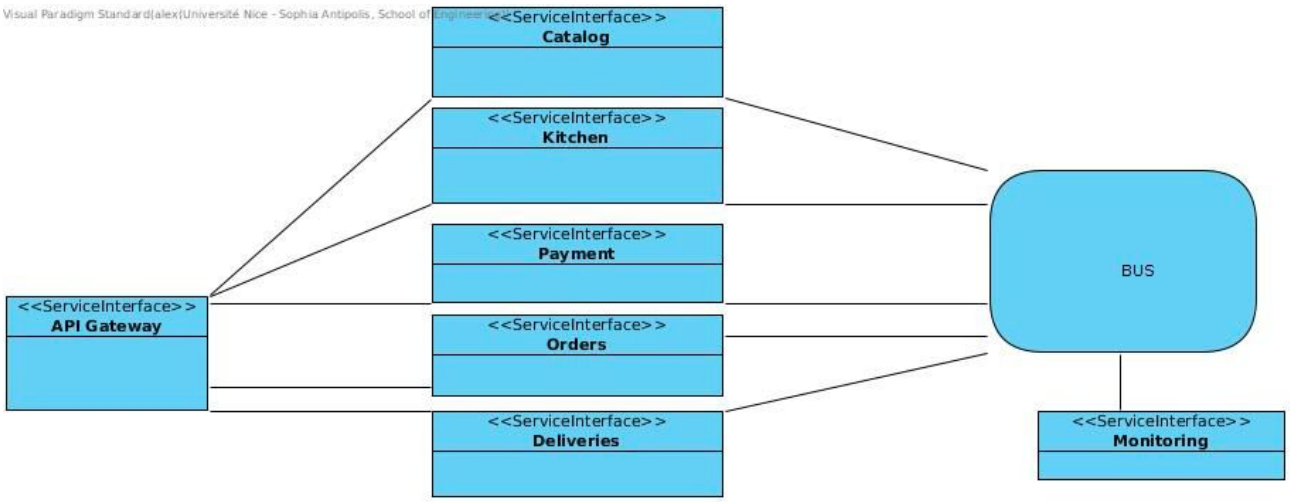

Figure 2 : Représentation des services

Le rôle de chacun de ses microservices est présenté ci-dessous :

- Le service catalogue met à disposition plusieurs routes en rapport avec les produits souvent être commandé.
- Le service Kitchen donne le temps estimé à l'arrivée de chaque commande en prenant en compte le restaurant et le type de plat.
- Le service Payment permet à chaque utilisateur de payer une commande validée.
- Le service order est chargé de centraliser les commandes et de suivre leur statut.
- Le service deliveries permet la gestion des livraisons

Les services communiquent avec le client en REST à l'aide d'une route HTTP et par l'intermédiaire d'un bus Kafka pour les communications internes au système faites à l'aide de message.

De même que dans le précédent projet, nous avons fait le choix d'avoir une base de données pour chaque microservice (Catalog, Kitchen, Payment, Order et Deliveries).

La présence d'une seule base de données pour l'ensemble des services nous aurait permis de gagner en simplicité au détriment d'un plus grand couplage. De plus, il y a un risque qu'un des services modifie la structure des données présente dans la base. Ce changement aurait pour conséquence une coupure de service.
La cohérence peut également être impactée si plusieurs services interagissent avec la même base de données, il y a un 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 données par service.

Cette architecture a plusieurs avantages :

- L'API Gateway qui permet de jouer le rôle d'interface pour l'utilisateur. Elle nous permet ainsi d'exposer toujours les mêmes routes aux utilisateurs sans que ces derniers soient impactés par un changement de route interne.

- Le bus qui permet la communication entre les services. Ainsi, il lui est possible d'avertir plusieurs services d'un événement qui a eu lieu. Le bus permet également de

faciliter le passage à l'échelle et d'assurer la résilience en cas de panne sur un des services.

Le bus est réalisé à l'aide de Kafka. Il contient plusieurs topic auxquels les autres services peuvent s'inscrire comme "order", "kitchen". Les messages contiennent plus d'informations comme le type d'événement : "new_order", "order_delivered".

Diagramme de séquence

La figure 3 représente le flux d'exécution qui est réalisé lors de la commande d'un repas.

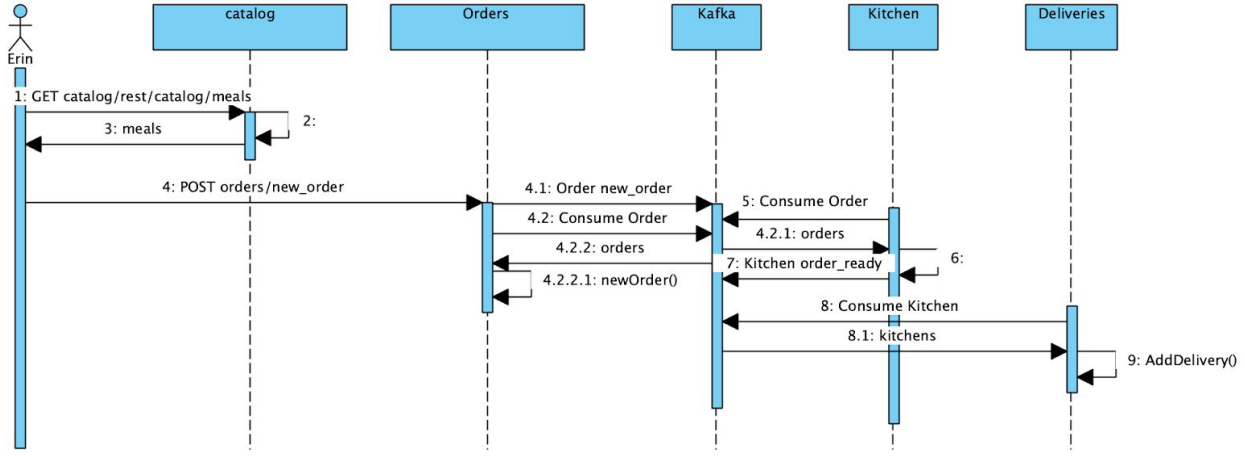

Figure 3 : Commande d'un repas

Services :

Catalogue

Methode HTTP	Route
GET	catalog/rest/catalog/meals
POST	catalog/rest/catalog/meals
GET	catalog/rest/catalog/restaurants

GET	catalog/rest/catalog/categories
GET	catalog/rest/catalog/categories/{categoryName}/meals
GET	catalog/rest/catalog/restaurants/{restaurantName}/meals
GET	catalog/rest/catalog/reviews
GET	catalog/rest/catalog/reviews/restaurants/{restaurantName}
GET	catalog/rest/catalog/reviews/restaurants/{restaurantName}/meals/{mealName}
POST	catalog/rest/catalog/reviews

Kitchen

Methode HTTP	Route
GET	kitchen/rest/kitchen/eta/{restaurant}/{meal}

Payment

Methode HTTP	Route
POST	payment/pay

Orders

Methode HTTP	Route
GET	orders/new_order
POST	orders/restaurants/{restaurant}/orders

Deliveries

Methode HTTP	Route
GET	deliveries/rest/deliveries/deliveries
POST	deliveries/rest/deliveries/deliveries
GET	deliveries/rest/deliveries/deliveries/{longitude}/{latitude}
PUT	deliveries/rest/deliveries/deliveries/{id}/{state}