Uberoo - Projet final

Membres :
Chennouf Mohammed
Spinelli Aurelien
Wilhelm Andreina

Responsable :
M. Mosser

Novembre 2018

Contents

1 Description du système

Uberoo est un tout nouveau service de livraison de produits alimentaires qui prévoit de créer une nouvelle entreprise à Sophia Antipolis à la mi-novembre. Il permet aux restaurateurs de publier des produits alimentaires (plats individuels ou menus complets) sur une plate-forme Web et aux clients de commander ces produits. Les articles sont ensuite livrés par les coursiers directement sur le lieu de travail du client. L'originalité d'Uberoo, à la différence des plate-formes existantes, réside dans le fait qu'au lieu d'exiger un calendrier de livraison fixe, le système calcule à la volée une heure d'arrivée estimée (ETA) pour chaque livraison.

2 Technologique

- **Java**
- **MongoDB**
- **Cucumber**
- **Gatling**
- **Kafka**
- **PostgreSQL**
- **Html**

3 Version initiale

3.1 Choix de conception et architecture

Pour réaliser une telle application nous allons dans un premier temps vous présenter les fonctionnalités couvertes et les scénarios qui exposeront les besoins des clients. Une fois la base de notre projet établi nous vous présenterons l'architecture générale d'Uberoo.

3.1.1 Fonctionnalités

En ce qui concerne le "scope" initial de notre projet, les fonctionnalités suivantes vont être développées :

- **Catalogue de produits alimentaires :** Nous avons mis en place un système de catalogue qui référence chaque plat suivant une catégorie. Une recherche suivant ces dernières pourra être effectuée.
- **Gestion des commandes :** Cette fonctionnalité permettra de gérer les commandes suivantes si elles ont été confirmé ou annulé.

- **Gestion des livraisons :** Nous aurons ainsi la possibilité de connaître l'avancement d'une livraison. Cette dernière sera une commande assignée à un livreur.
- **Estimation du temps de livraison :** Lorsqu'un client passe une commande le temps de livraison de cette dernière est estimé pour qu'il puisse confirmer ou non la commande.
- **Exposition des livraisons disponibles proches :** Un livreur pourra consulter les livraisons non assignées autour de lui pour sélectionner efficacement celles qui l'intéressent.
- **Gestion des paiements :** Un client pourra payer une commande avant qu'elle lui soit livrée.

3.1.2 Scénarios

Personas :

- *Gail* un étudiant qui vit à **Sophia** qui commande souvent dans un camion snack.
- *Erin* un développeur logiciel dans une grande entreprise.
- *Jordan* un chef de restaurant.
- *Jamie* un livreur.
- *Uberoo* notre système.

Scénario de livraison sans encombre :

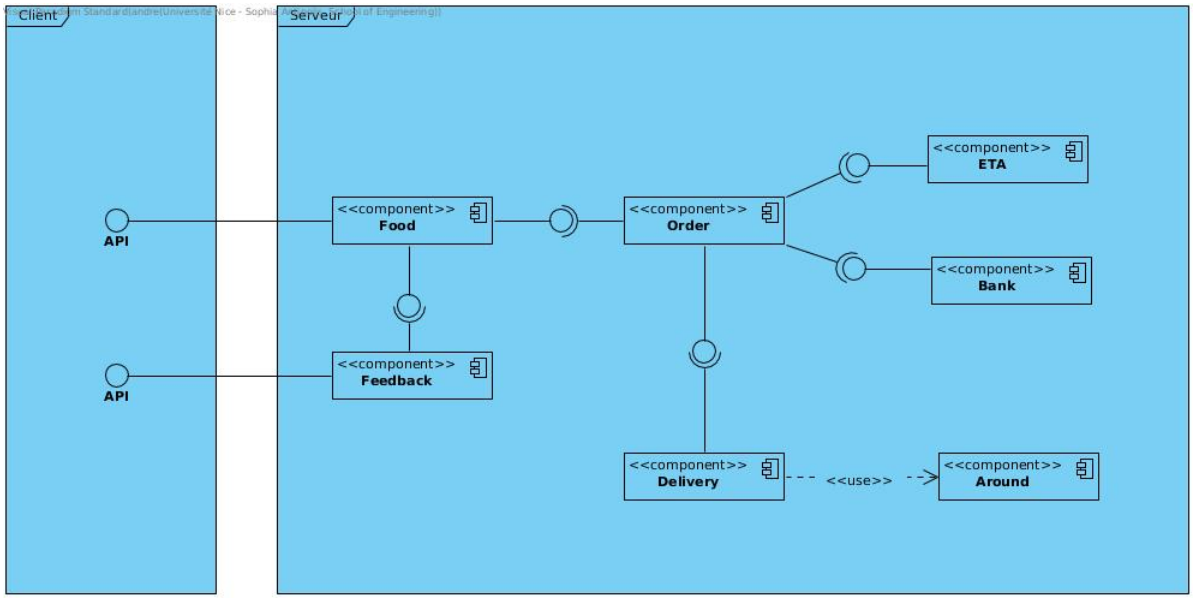
Ce scénario décrit une livraison qui se déroule sans que le livreur n'est de problème.

- *Gail* consulte le catalogue proposé par un restaurant.
- *Gail* choisi une soupe de ramen dans la catégorie de nourriture asiatique.
- *Uberoo* estime le temps de livraison de la commande de *Gail*.
- *Gail* trouve le temps de livraison convenable et confirme sa commande.
- *Gail* procède au paiement de sa commande pour pouvoir manger dès que la commande arrive.
- *Jordan* consulte les commandes confirmées par les clients pour préparer efficacement les repas.
- *Jamie* veut connaître les livraisons disponibles autour de lui, pour sélectionner une course et commencer la livraison.

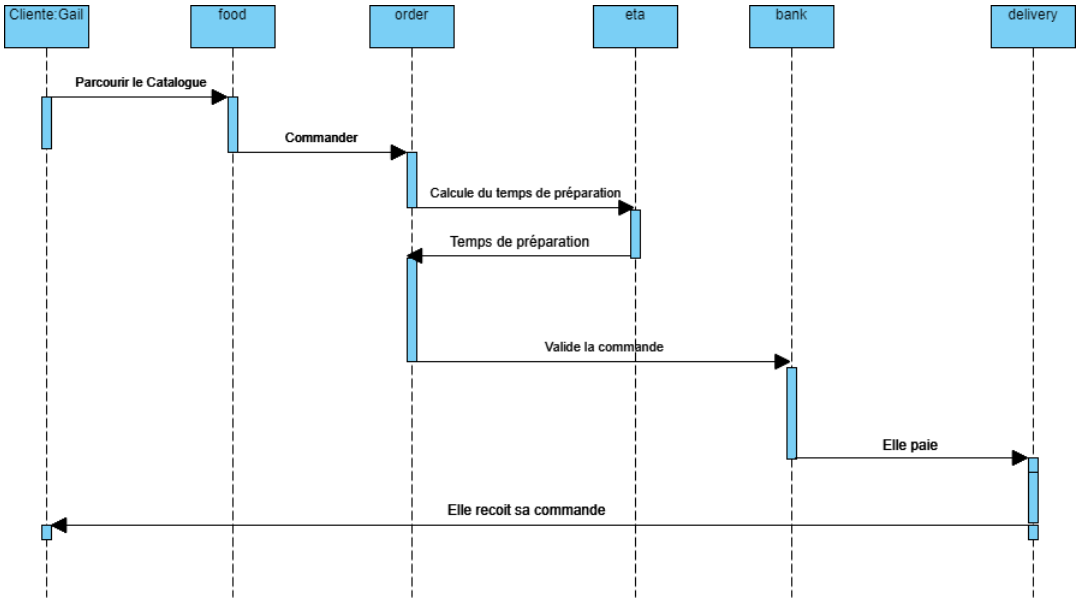
- *Jamie* confirme le bon déroulement de la livraison et pourra être payer.
- *Jordan* peut consulter les commandes qui ont été correctement livrées.

3.1.3 Architecture

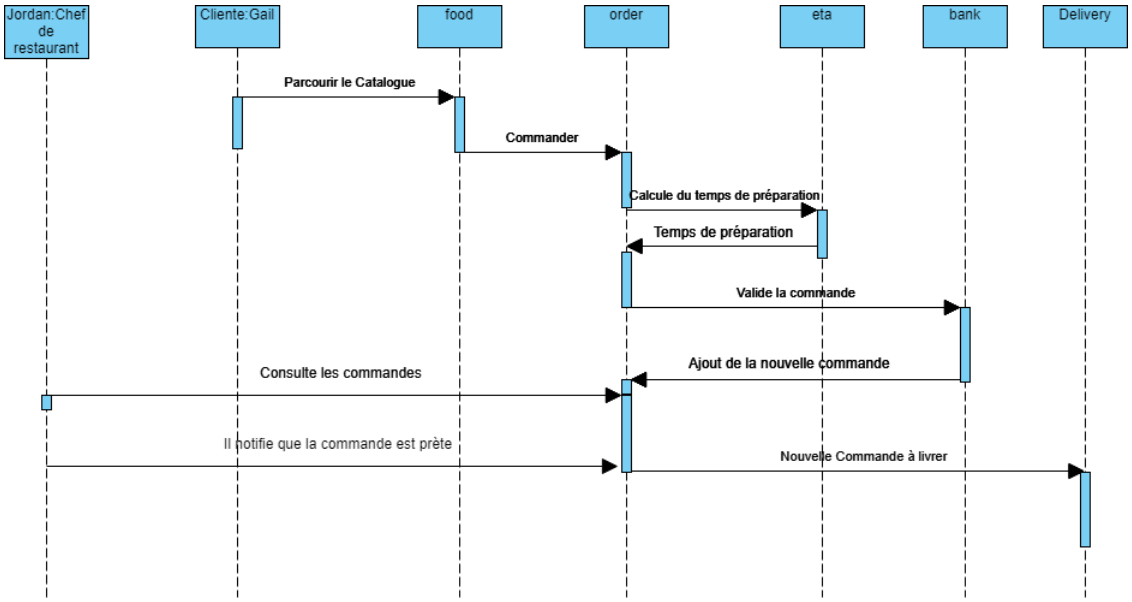
L'architecture générale de notre application est présenté dans ce diagramme de composant :



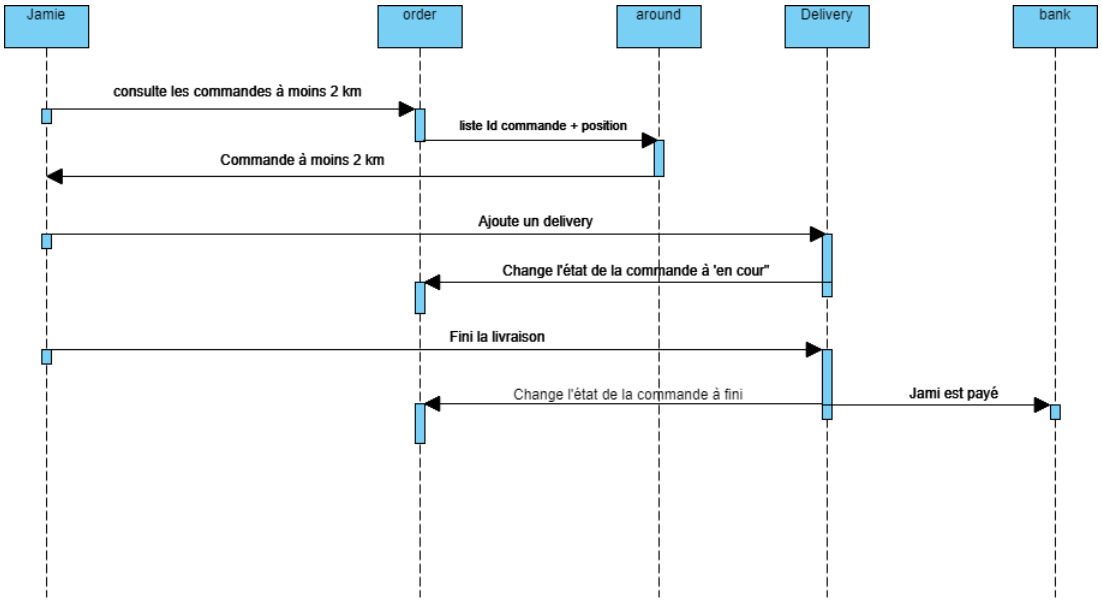
Voici un diagramme de séquence pour nos principaux acteurs du système.
Gail une étudiante :



Jordan le chef de restaurant :



Jamie le coursier :



3.2 Nos micro-services et leurs justifications

- **Service de catalogue :** Ce service pourra consulter les plats proposés par un restaurant et également en ajouté.
- **Service de commande :** Le client pourra passer une commande dans ce service.
- **Service de livraison :** Un livreur sera assigné à une commande de client. Ce service permettra aussi de consulter les livraisons en cours et terminées.
- **Service de feedback :** Le client pourra laisser de feedbacks des repas commandés.
- **Service ETA :** Calculera le temps de livraison lorsque le client passe une commande.
- **Service "arround me" :** Ce service exposera les courses présentes les plus proches du livreur.
- **Service de paiement :** Un client pourra payer une commande avant qu'elle lui soit livrée.

3.2.1 catalogue en RESSOURCE

Nous avons décidé de faire un service pour le catalogue pour les raisons suivantes: Contrairement aux autres services, le catalogue aura bien plus de lecture que d'écriture. En général un restaurant ne modifie que très rarement sa carte. Pour

cette raison nous devons choisir une base de données efficace pour les lectures. Le fait d'avoir très peu de modifications nous permet également de ne pas être obligé d'opter pour une base de données à forte cohérence. Pour un tel service nous avons décidé d'utiliser un paradigme RESSOURCE. Ce paradigme permet de définir une interface uniforme pour un repas : chaque ressource(le repas) est identifiée unitairement, les ressources ont des représentations définies. Cela nous permettra d'implémenter facilement le concept de catégorie.

3.2.2 livraisons et commandes en DOCUMENT

En ce qui concerne les services de livraisons et commandes, nous avons décidé de faire deux services distincts car on peut déjà s'imaginer deux cas distinct :

- Une commandes pourra se représenter sous forme de "panier".
- Une livraison pourra être suivi dans son avancement.

Néanmoins, une livraison peut être vue comme un état particulier d'une commande, mais le fait de séparer ces deux services nous permettra de pouvoir "scaler" lors du tracking d'une livraison. De plus, la livraison fait intervenir un livreur en charge d'une commande ce qui expose un contexte borné distinct. Pour ces deux services nous allons avoir un nombre important de lecture comme d'écriture. Par exemple une commande devra être au moins consultée une fois par le restaurateur, et une livraison par le livreur. De plus notre base de données dans ce cas devra être fortement cohérente puisque les données manipulées auront des éventuelles modifications qui seront importantes de connaître.
De plus, les états d'une commandes et d'une livraison de sont pas les mêmes. Et le fait d'avoir choisi ce style d'architecture (document) nous permettra d'ajouter facilement la gestion de nouvel événements.

3.2.3 ETA RPC

Pour estimer le temps de livraison pour un commande, nous avons décidé de faire un service d'ETA distinct de celui des livraisons et commandes. Car en effet, il fait intervenir les deux. Il doit pouvoir évaluer suivant les livreurs et la listes des commandes. Ce service devra donc effectuer de nombreuses lectures sur la base de données des commandes et livraisons pour ensuite calculer le temps.

3.2.4 Payement et Feedback DOCUMENT

En ce qui concerne le service de paiement le style document a été choisi, car il s'agit d'une série d'événements pour compléter le paiement des repas. Tout d'abord, il faut créer le paiement et en suit le valider. On donne aussi la possibilité de consulter tous les paiements réalisés. Le service feedback a aussi été crée un utilisant le style document, car on peut offrir diffèrent évènements comme la possibilité aux clients de faire un feedback et la possibilité aux chefs de les consulter pour améliorer ses repas.

3.2.5 "Around me" RPC

Par ailleurs, le service "around me" à été créé en utilisant un style RPC, car il s'agit d'un simple calcul qui prend les commandes à livrées et les filtres par rapport à la position du coursier.

De manière générale, la base de données choisi pour chaque service devra être avoir une bonne scalabilité pour supporter les pics de charges d'une application tel qu'Uberoo. En effet, une utilisation de cette application sera beaucoup plus important lors des heures de repas.

4 Première évolution

4.1 Choix de conception et architecture

Pour réaliser la première évolution de l'application nous allons vous présenter les nouvelles fonctionnalités couvertes et comment nous les avons ajoutées.

4.1.1 Fonctionnalités

- **Tracker une livraison :** Un client pourra tracker une livraison et connaître en temps réel la position du livreur.
- **Commentaires des client :** Un client pourra émettre un commentaire sur un produit. Ce commentaire pourra être consulté par le patron du restaurant pour éventuellement amélioré son produit.

4.1.2 Scénario

Scénario supplémentaire de la première évolution:

- *Gail* s'impatiente et track *Jamie* pour connaitre sa position en temps réel.
- *Gail* poste un commentaire négatif sur un plat qu'il a reçu.
- *Jordan* consulte les commentaires d'un plat précis pour pouvoir améliorer ce dernier.

4.2 Évolution des microservices

Pour répondre aux besoins du client de "tracker" une livraison, nous avons ajouter la gestions d'un évènement dans le service de livraison. Ainsi, lorsqu'un client voudra connaître la position du livreur notre service lui renverra en temps réel sa position géographique.
Par contre pour la fonctionnalité de feedback, qui est lié au repas, nous avons fait un service séparé car contrairement au catalogue de repas, le nombre d'écriture sera beaucoup plus important. Le feedback fera référence aux repas. La base de données choisi pour ce service doit être efficace en lecture/écriture et une forte cohérence ne sera pas nécessaire. Nous avons implémenté ce service en document mais le paradigme Ressource aurait pu nous apporter une représentation définie des feedback qui ont dans notre cas tous la même forme.

5 Seconde évolution

5.1 Choix de conception et architecture

Pour réaliser la seconde évolution de l'application nous allons vous présenter les nouvelles fonctionnalités couvertes et comment nous les avons ajoutées.

5.1.1 Fonctionnalités

- **Signaler un problème :** Un livreur pourra à tout moment signaler un problème durant l'acheminement d'une livraison.

5.1.2 Scénario

Scénario supplémentaire de la seconde évolution:

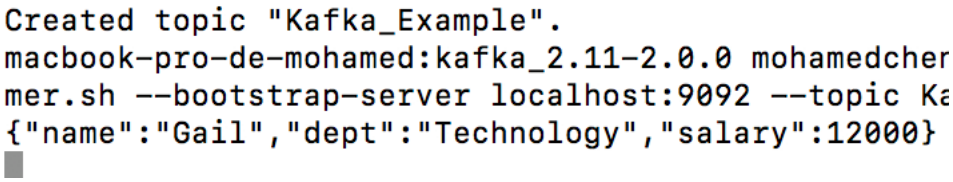
- *Jamie* a un accident sur la route et signal qu'il ne pourra pas terminé sa livraison.
- *Uberoo* assigne la livraison à un autre livreur disponible.

5.2 Modification du microservices de livraison

De la même façon que le tracking, pour ajouter cette nouvelle fonctionnalité, nous n'avons pas eu besoin de grande modification. Un problème qui survient lors d'une livraison est ainsi un événement supplémentaire que l'on va devoir gérer dans notre service de livraison.
Notre service est implémenté dans la paradigme document. L'ajout du problème a donc été facilement réalisable.
Néanmoins, nous ne réalisons que la déclaration d'un problème et non pas le remplacement de livreur... Ce qui pourrait être fait lorsque nous captons l'événement signalant un problème, en notifiant un livreur disponible.

6 Bus de message

Pour s'initier à Kafka, nous avons débuté par utilisation de Kafka en local avec un service producer. Producer envoie un TOPIC : Kafka_Exemple et un message de type "User". Grâce à l'utilisation de Kafka en local, il est possible de créer un consumer avec:
bin/kafka-console-consumer.sh –bootstrap-server localhost:9092 –topic Kafka_Example –from-beginning
Exemple de resulat avec l'envoie du User Gail



console consumer du topic Kafka_Example

Le service Producer est dans notre projet mais nous le considérons pas dans Uberoo. Il nous a permis de nous familiariser avec Kafka.

Pour ajouter un Bus de message dans notre système, nous avons créé un service orderkafka et un service deliverykafka nous permettant de mettre en place notre intégration entre le service ordre et delivery sans impacter nos services actuels. Le service orderkafka envoie un topic ordre et un objet Order, celui-ci est consommé par deliveryKafka.

Étant donné notre infériorité numérique (3 étudiants) et le temps mis à la compréhension et l'utilisation de Kafka, il nous a été difficile d'implémenter plus d'évent Kafka. Notre principal problème fut la création du consumer et notamment la désérialisation du message reçu.