API Service Lab

- Authors: Mohamed Chennouf, Andreina Wilhelm, Aurélien Spinelli, Thibault Bondon

Justification

Service de commande :

Nous avons choisi d'utiliser le paradigme *Document* comme interface du service de commande. Il existe beaucoup de cas utilisations pour ce service :

- le client fait une commande et un temps de livraison lui est transmis
- Le client valide ou pas la commande si le temps de livraison lui convient ou pas
- Le cuisinier doit pouvoir consulter la liste des commandes

Tous ces cas utilisation se prête bien à des évènement tel que :

- ORDER pour faire la commande,
- VALIDATE pour valider ou pas une commande,
- CONSULT pour consulter la liste des commandes

Ce paradigme nous permet, à travers des évènements, de gérer la complexité amené par tous ces cas utilisations avec la même route. Le paradigme Document nous permet aussi de se détacher des procédures utilisées pour se concentrer sur les paramètres qui l'intéresse, ici définis par les différents évènement qu'il souhaite faire.

Service de catalogue de repas:

De plus, nous avons aussi choisi d'utiliser le paradigme *Resource* car un catalogue est une ressources qui contient des repas auquel on pourrai y accéder par une URL : http://localhost:9090/food-service-rest/foods. Ce paradigme nous offre une grande modularité:

- On pourrait ajouter un autre chemins très aisément afin de compléter ce service de catalogue comme par exemple : hhttp://localhost:9090/food-service-rest/foods/id_food pour afficher le detail d'un repas.

Nous avons aussi choisi d'utiliser le paradigme Ressource comme interface du service de catalogue de repas, car il permet d'exposer de façon simple des ressources avec un API standarisé et compréhensible aux clients. Avec REST on réussi à réduire le couplage entre le client et le serveur ce qui permet une évolution plus facile à l'avenir. Dans notre cas il s'agit aussi de ressources sur lesquels on fera surtout de la consultation. Il pourront être manipulé avec d'actions CRUD donc REST c'est idéal puisqu'on peut utiliser les actions déjà définies de HTTP.

Service de livraison :

Dans un premier temps, nous étions parti sur le paradigme RPC pour la livraison car nous pensions qu'une livraison se résumait à la procédure: *livrer(repas, destination)* mais nous nous sommes rendu compte qu'une livraison débute lorsque l'on entre dans le système l'objet à livrer et se finir lorsque le client se fait livrer le repas. De ce fait, il y a là plusieurs cas utilisation :

- Assigner à un livreur le repas à livrer.
- Le livreur doit ensuite mettre dans le système la preuve que la livraison a été faite : facture du client.
- Le patron peut consulter l'état des livraisons en cas de litige par exemple.

Tous ces cas utilisations se prêtent bien à des évènement tel que :

- DELIVER pour ajouter une livraison et l'assigner à un livreur.
- COMPLETE pour marquer la fin d'une livraison et y ajouter la facture.
- CONSULT pour consulter une livraison en particulier.
- LIST pour consulter l'état des livraisons.
- LISTCOMPLETED pour consulter les livraisons effectuées.
- LISTNOTCOMPLETED pour consulter les livraisons en cours.
- DELETE pour supprimer les livraisons

Ce paradigme nous permet de gérer tous ces cas utilisations avec des évènements sur la route: http://localhost:9100/delivery-service-document/delivery

Comme expliqué précédemment , ce paradigme permet de se détacher des procédures utilisées pour se concentrer sur les paramètres qui l'intéresse, ici définis par les différents évènement qu'il souhaite faire.

Model and API

Service de commande :

Pour le service de commande nous nous sommes dit qu'une commande a :

- un id
- un nom de repas
- une adresse de destination
- un nom client
- un status (payé ou pas payé)

De ce fait voici les API qui en résulte :

- Post http://localhost:9080/order-service-document/ordersFood
- Body -> raw -> application/json

L'objet JSON dans le body de la Requête pour faire une commande:

```
{
  "event": "ORDER",
  "orderFood": {
    "id":"23",
    "nameOfFood":"Hot Wings 30 Bucket",
    "nameOfClient":"Aurélien",
    "addressDestination":"3 avenue promenade des anglais"
  }
}
```

L'objet JSON dans le body de la Requête pour valider une commande:

```
{"event": "VALIDATE", "id":"23", "validate":true}
```

L'objet JSON dans le body de la Requête pour refuser une commande:

```
{"event": "VALIDATE", "id":"23", "validate":false}
```

L'objet JSON dans le body de la Requête pour avoir toutes les commandes:

```
{ "event": "CONSULT"}
```

L'objet JSON dans le body de la Requête pour purger la base de données:

```
{ "event": "PURGE"}
```

Service ressource catalogue de repas

Pour le service ressource nous avons mis un moyen ajouter des repas dans le catalogue :

- Post http://localhost:9090/food-service-rest/foods
- Body -> raw -> application/json

L'objet JSON dans le body:

```
{
 "food":
   {
    "name": "tarte aux pommes",
    "description": "Tarte sucrée, faite d'une pâte feuilletée garnie de pommes émincées.",
    "price": 3.5
   }
}
```

La Requête pour afficher le catalogue :

- Get http://localhost:9090/food-service-rest/foods

Service de livraison :

Pour le service de livraison nous nous sommes dit qu'une livraison a :

- un id
- un client
- un commande (qui contient l'adresse, le nom du repas ….)
- un status (en cours de livraison, livré …)

De ce fait voici les API qui en résulte :

- Post http://localhost:9100/delivery-service-document/delivery
- Body -> raw -> application/json

L'objet JSON dans le body de la Requête pour créer une livraison et l'assigner:

```
{
 "event" : "DELIVER",
 "delivery" : {
   "deliveryMan": "Escobar",
   "delivered": "false",
   "id": "6",
   "idOrder": "521"
   }
}
```

L'objet JSON dans le body de la Requête pour compléter une livraison:

```
{"event": "COMPLETE", "id":"6"}
```

L'objet JSON dans le body de la Requête pour consulter une livraison:

```
{"event": "CONSULT", "id":"6"}
```

L'objet JSON dans le body de la Requête pour avoir toutes les livraisons:

```
{"event": "LIST"}
```

L'objet JSON dans le body de la Requête pour avoir toutes les livraisons terminées:

```
{"event": "LISTCOMPLETED"}
```

L'objet JSON dans le body de la Requête pour avoir toutes les livraisons non terminées:

```
{"event": "LISTNOTCOMPLETED"}
```

L'objet JSON dans le body de la Requête pour supprimer une livraison:

```
{"event": "DELETE", "id":"6"}
```

Les 2 autres paradigmes

Service de commande en RPC :

RPC se porte bien avec des services pouvant se traduire par un verbe pour la commande on aurait eu plusieurs procédure tel que :

- Post url pour commander(...)
- Post url valider(...)
- Post url consulter(...)
- Post url purger(...)

Service de commande en Ressource :

Pour le service de commande en ressource, il aurait fallu considérer une commande comme une ressource avec :

-Get url pour avoir les commandes
-Post url pour ajouter des commandes

Service de livraison en RPC :

RPC se porte bien avec des services pouvant se traduire par un verbe pour la commande on aurait eu plusieurs procédure tel que :

- Post url pour livrer(...)
- Post url pour validerLivraisons(...)
- Post url pour listerLesLivrasons(...)

Service de livraison en Ressource :

Pour le service de commande en ressource, il aurait fallu considérer une livraison comme une ressource avec :

-Get url pour avoir la liste des livraisons
-Post url pour valider une livraison
-Post url pour livrer

Service catalogue en RPC :

RPC se porte bien avec des services pouvant se traduire par un verbe pour la commande on aurait eu plusieurs procédure tel que :

- Post url pour AfficherCatalogue(...)
- Post url pour AjouterCatalogue(...)

Service catalogue en document :

Avec document il aurait fallu créer des event au catalogue tel que :

{ “event”: CONSULT } pour afficher la liste des repas

{ “event”: “ADDFOOD”, “food”:{...} } pour ajouter un repas

Tests

Nous avons des tests acceptations qui décrivent des scénario utilisation de nos services. Nous avons aussi des tests gatling qui nous permet de valider que nos service peuvent etre utilisé par plusieurs utilisateurs simultanés .