ESB - TravEL Expenses GReat Again!

Carlier Maxime - Chevalier Mathias - Eroglu Yasin - Gning Khadim

Introduction

L'objectif de ce lab et de cette solution est de fournir au client un moyen simple et rapide de soumettre une demande de réservation à un système de gestion de voyages. Cette réservation contient différente sous-réservations, un vol, un séjour à l'hôtel et une location d'un véhicule. Bien évidemment, l'entreprise souhaitant fournir ce service ne dispose pas, et ne souhaite pas pour des raisons évidentes, implémenter elle même les services permettant de rechercher chacune de ces trois sous-réservations. La complexité derrière ces services étant titanesque, elle ne pourrait en aucun cas rivaliser avec les grands domaines. Il faut donc être capable d'agréger un ensemble de services externes afin de concevoir à la demande des utilisateurs une demande de voyage complète.

La responsabilité de l'entreprise est donc de fournir cette possibilité à l'utilisateur et de gérer son service de gestion de voyage qui va stocker et conserver ces données pour fournir au client un suivi de ce voyage et un système d'approbation. Nous sommes ici dans un cas finalement assez typique où le besoin d'*Enterprise Service Bus* (ESB) est réel et prend tout son sens. Ce rapport a pour but d'expliquer la manière dont nous avons procédé et les solutions que l'on propose pour notre architecture d'intégration ainsi que les choix et leurs justifications ayant motivé son design.

Choix stratégiques et planification

La première étape nécessaire à la mise en place d'un ESB est d'abord un ensemble de choix stratégiques afin de localiser où les risques sont les plus grands et quelles seront donc les problématiques les plus importantes et les plus urgentes à traiter. Nous avons effectué cette recherche tous ensemble afin de tous avoir les mêmes lignes directrices en tête tout au long de l'implémentation, l'objectif étant de bloquer un design et de s'y tenir. Nous avons aussi proposé une ébauche de notre route principale dès cette première recherche à travers un diagramme. Cela nous a permis à fortiori d'avoir une consistance réelle dans différents compartiments ce qui ne peut qu'être bénéfique au système et à sa maintenabilité. Nous allons donc expliciter nos décisions stratégiques et la manière dont elles se sont directement répercutées sur notre planning et sur la manière dont nous avons abordé l'implémentation de cet ESB.

Décisions stratégiques

La première problématique que nous avons abordé est la communication avec les services externes dont nous n'étions pas maître. L'inconnue sur la qualité de ces services se devait d'être levée le plus tôt possible puisque bien évidemment, notre incapacité à nous connecter aux services externes est totalement bloquant pour la suite de l'intégration. Et c'est donc tout naturellement que nous avons consacré la première semaine à implémenter cette connexion et à découvrir les services externes avec les objets métiers traversants, et ce afin de s'adapter le plus vite possible à ces conditions sur lesquelles nous n'avons finalement pas la main.

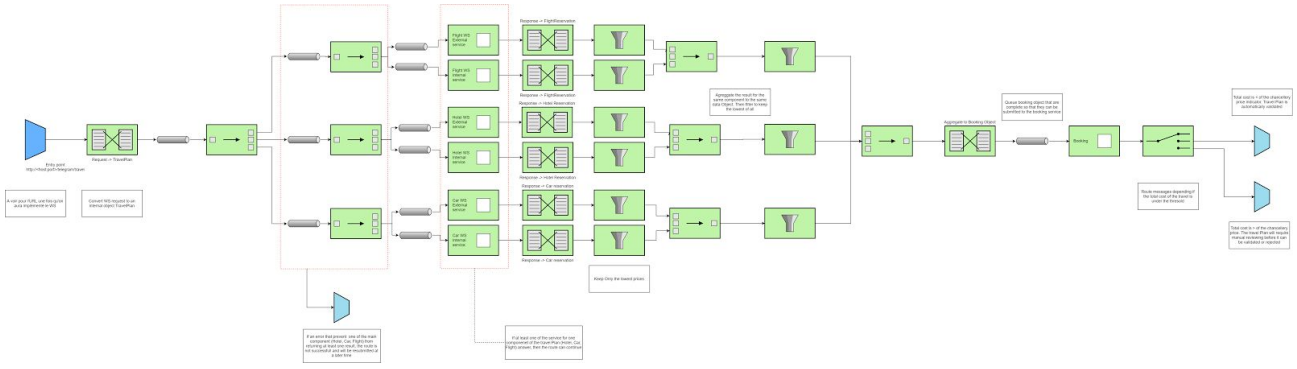
La seconde problématique majeure s'est aussi révélée dès la phase de recherche et de choix décisionnels, il s'agit de l'implémentation du Bus elle même et des Routes qui le composent. En effet, personne dans le groupe n'avait de connaissances avant ce projet, et aucun de nous n'avait développé en Camel ni même approché ce langage. Il était donc primordial de lever cette inconnue technologique le plus rapidement possible. Nous avons donc choisi d'y accorder la deuxième semaine, afin de livrer une version très minimale mais déployable et fonctionnelle le plus vite possible. Malheureusement, de par sa complexité et notre manque de connaissances, nous verrons par la suite que la mise en place du framework Camel s'est révélé plus fastidieuse qu'initialement prévue. En même temps que la mise en place du Bus minimal, les fonctionnalités nécessaires à la réalisation de la route de demande de réservation pour le service de gestion de voyage ont aussi été placées dans la deuxième semaine.

Une fois les deux problématiques levées, il s'agissait d'implémenter autour de ces fonctionnalités afin de rendre les services proposés plus efficaces et plus résilients. De plus, il est important de noter que nous avons cherché à avoir un ensemble de fonctionnalités potentiellement restreint et une route Camel totalement fonctionnelle afin de prouver notre capacité à déployer quelque chose de transversal. Une fois ce stade atteint, on démontre une capacité dans le temps à couvrir ensuite le scope de manière plus horizontale.

La solution proposée

Nous allons ici étudier ce qui est, comme cela a été énoncé auparavant, le coeur de notre projet, la route permettant de fournir à l'utilisateur un agrégateur de services de vols, de réservation d'hôtels et de location de voiture, afin de lui offrir la possibilité d'effectuer une demande de voyage complète.

Diagramme



Le diagramme est disponible à la racine de notre répo GIT sous le nom "DataFlow Diagram.png" pour une meilleure visualisation.

Explications

Le premier point qui ressort est la très grande similarité entre les 3 sous routes qui s'occupent de la connexion et de la récupération de données aux services externes. Cela n'est néanmoins pas étonnant puisque leur responsabilité et leur objectif est extrêmement similaire. Seule l'implémentation au niveau du code change puisque les services externes ne fournissent pas tout à fait le même métier (avion, hôtel, voiture...) et ne sont pas tous développés avec le même paradigme.

Une fois la requête reçue en entrée par le bus, elle est directement envoyées aux 3 sous-routes qui fonctionnent donc de la même manière. Chaque sous-route récupère les données liées à la requête dans une queue, avant d'elle même encore dupliquer les 2 données dans 2 chemins différents. En effet, chaque route est tenue d'appeler 2 services (au moins) fournissants des réservations d'une même entité afin de proposer à l'utilisateur la solution la moins chère.

Lorsqu'un des services répond, le contenus de la réponse est récupéré puis utilisé pour construire un de nos objets interne permettant de représenter cette réservation dans un référentiel commun qui sera utilisé par la suite pour comparer la réservation la moins cher pour un même type de réservation, puis encore ensuite pour transformer cet objet dans un format qu'on puisse utiliser pour soumettre un plan de voyage complet au service de review.

Nous avons définie plusieurs routes 'en erreur' qui font que la route globale de soumission d'un plan de voyage se terminera prématurément. Celles-ci sont :

- Lorsqu'une des sous routes globale (Hotel, Car, ou Flights) ne renvoie AUCUN résultat,
- Lorsque le travel plan soumis à un prix excédant les taux de chancellerie.

Nous avons fait le premier choix, car de toute évidence, il n'est pas possible de soumettre un plan de voyage qui ne soit pas complet. Et comme le sujet ne fait pas mention de plan de voyage dont une des composantes est absente (ex: par de réservation de voiture, car pas nécessaire), nous avons interprété cela comme le fait qu'un plan de voyage doit toujours comporter ces 3 composantes.

Le second choix fait partie des contraintes imposées par le sujet, il est dit que si un plan de voyage excède le montant définis par les taux de chancellerie, il doit être mis de côté afin d'être revus manuellement par un manageur après que l'auteur du plan de voyage est fournis de telle explication. Ainsi si le manager n'accepte pas les justifications de son employé, la route se termine en erreur et on suppose qu'un procédé annexe prend la relève afin de régulariser cette situation, mais si le manager valide cet excès, la route se termine en succès et elle poursuit son cours (notification de la validation à l'employé).

Un des autres choix que nous avons fait pour le format de notre route, est de laisser l'opportunité de notre route de continuer dans le cas où un des services d'une composante du plan de voyage de ne répond, MAIS à la condition qu'au moins un est retourné une liste d'accommodations (afin de ne pas entrer en conflit avec la specs cité ci-dessus). Nous justifions ce choix par le fait qu'il n'est pas problématique de laisser continuer la route avec un manque de choix d'accommodation, car

quelque soit le nombre de résultat obtenus, il y aura toujours le procédé de fin de route qui compare le prix du plan de voyage avec les taux de chancellerie. Et donc, à ce moment, soit le prix est inférieur et la route s'achève en succès car toutes les contraintes sont respectées (malgré le fait que tous les services n'est pas données leurs liste de résultats, et que nous n'avions de ce fait pas forcément le plan de voyage le moins cher possible) ; soit le prix est supérieur au taux définis, et dans ce cas la route requierera l'approbation d'un manageur. Dans ce cas si, on peut supposer que soit mis en place un mécanisme qui permette de signaler qu'au moment de l'établissement du plan de voyage, un des service d'une composante n'a pas renvoyés de résultat, et qu'ainsi le prix actuel du plan de voyage n'est peut être pas le moins cher possible. Le manager aurait donc ainsi la possibilité de relancer le processus afin que cette fois si le service marche, le plan de voyage soit complété.

Implémentation

Niveau implémentation, nous avons une main route qui va gérer l'agrégation grâce à une stratégie d'agrégation (**implements** AggregationStrategy) et ainsi répartir la demande sur les différents services de notre système. Avant de pouvoir faire une redirection et de lancer le traitement en parallèle, il faut, bien évidemment, formater les données reçu en entrée via des processor (**private static** Processor) afin que nos services puisse les comprendre et procéder en conséquence..

Ces services ont eu aussi leur propre route principale qui envoie la demande sur deux services du même type (voiture, hôtel, vol) mais distinctes, l'une interne et l'autre externe à notre solution. La aussi nous utilison une stratégie d'agrégation afin de traiter en parallèle l'exécution de nos deux service d'hôtel par exemple et pour chacun d'entre eux, nous effectuons un trie afin de sélectionner l'hôtel le moins chére du service.

Une fois l'exécution des services terminé la route principal du service en question récupérer chaque élément sélectionné avant de refaire lui même un trie afin d'en sortir que le plus rentable des deux. Une fois le service ayant fini son exécution, nous revenons sur la main route qui lui va concaténer dans le json le résultat du service en question.

Une fois tout les résultat obtenus, le json est transmise au service de booking qui va lui opérer de son côté.

Problèmes rencontrés

Lors de la réalisation de ce projet, nous avons été confrontés à certains problèmes de plus ou moins grande envergure, qui ont rendu plus lent que prévus, l'ajout de fonctionnalités dans notre projet, ou qui ont simplement rendu impossible la réalisation de celle-ci.

Le premier problème rencontrés a été la documentation de Apache Camel. Celle-ci manque terriblement de raffinement et rend difficile la recherche d'informations et l'implémentation de fonctionnalité. Le fait que la stack ServiceMix soit Open Source est en effet un avantage et une raison pour laquelle la documentation n'est pas complète, mais nous argumentons, que pour une solution Open Source, il est d'autant plus important d'avoir une documentation à jour et complète afin de rivaliser avec les solutions propriétaire qui certe sont payante et entraîne un "Vendor

Lock-in", mais qui propose un support (payant / compris dans l'offre) pour ce genre de problématique.

Le second problème rencontré, a été le manque d'outil afin de travailler dans un environnement convenable. En effet, dû à l'amplitude du procédé de développement, il n'est plus possible de travailler avec un Debugger ce qui rend la traque des problèmes possible seulement avec des printout et de logs disséminés un peu partout. Nous avons exploré les pistes de Remote Debugging proposé par Intellij, mais vu la quantité de couche logicielle à traverser nous n'y somme pas arrivé.

Le troisième problème a été

Répartition

Nous avons essayé de séparer les tâches afin d'avoir une charge de travail répartie équitablement entre les 4 étudiants, 25% chacun donc.

- Carlier Maxime:
  - Sous-route hôtel
- Chevalier Mathias:
  - Services externes de *booking*
- Eroglu Yasin:
  - Sous-route voiture
- Gning Khadim:
  - Sous-route avions

L'intégration globale dans la route principale, tout comme les phases de design ont été appréhendées en commun, et chacun a apporté son tra

Conclusion

Finalement, les problématiques d'intégration ont été résolues à travers un *Enterprise Service Bus* qui met en oeuvre une route principale avec énormément de responsabilités qui a donc été au niveau de l'implémentation découpées en sous-routes. Cette route étant au coeur du métier, elle se doit non seulement d'être efficace, elle se doit aussi d'être résiliente. C'est donc tout naturellement que notre effort s'est concentré sur ce point dur tout au long du projet, quitte à occulter d'autres problématiques.