POLYTECH NICE-SOPHIA

Université Nice Sophia Antipolis

Année universitaire : 2019 - 2020

RAPPORT DRONE DELIVERY

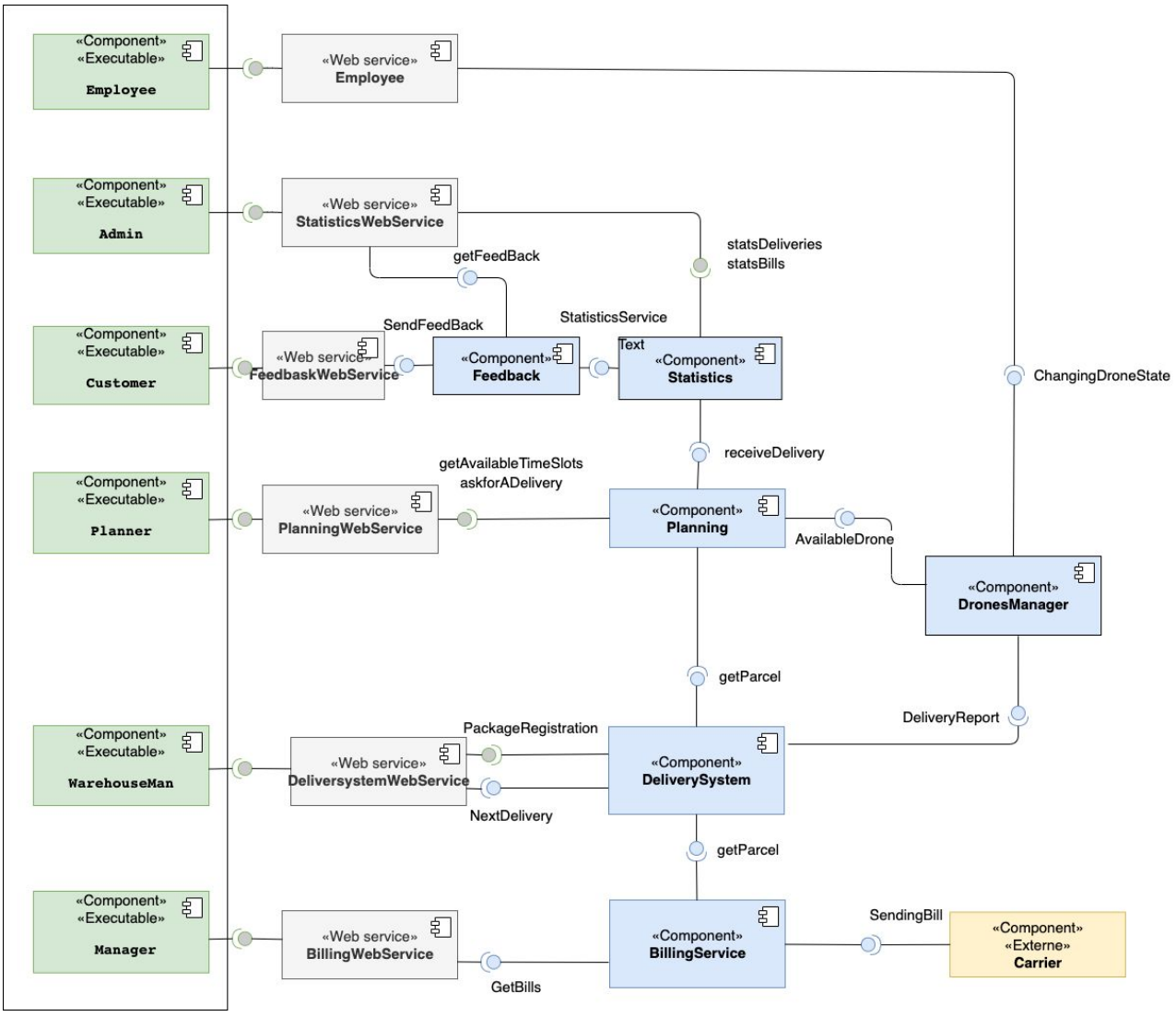
ISA - DEVOPS - SI4

Equipe J :

Florian AINADOU
Yannis FALCO
Maël DELABY
Virgile FANTAUZZI
Remy LARROYE

TABLE DES MATIÈRES

Le diagramme de composants



Notre projet est composé de trois (03) grandes parties, la partie client : en vert pour la console et en gris pour les web services. La partie j2e qui est la partie métier de notre application qui s'occupe de gérer les données et les opérations en bleu. Enfin nous avons la partie en orange qui est un composant externe à notre application.

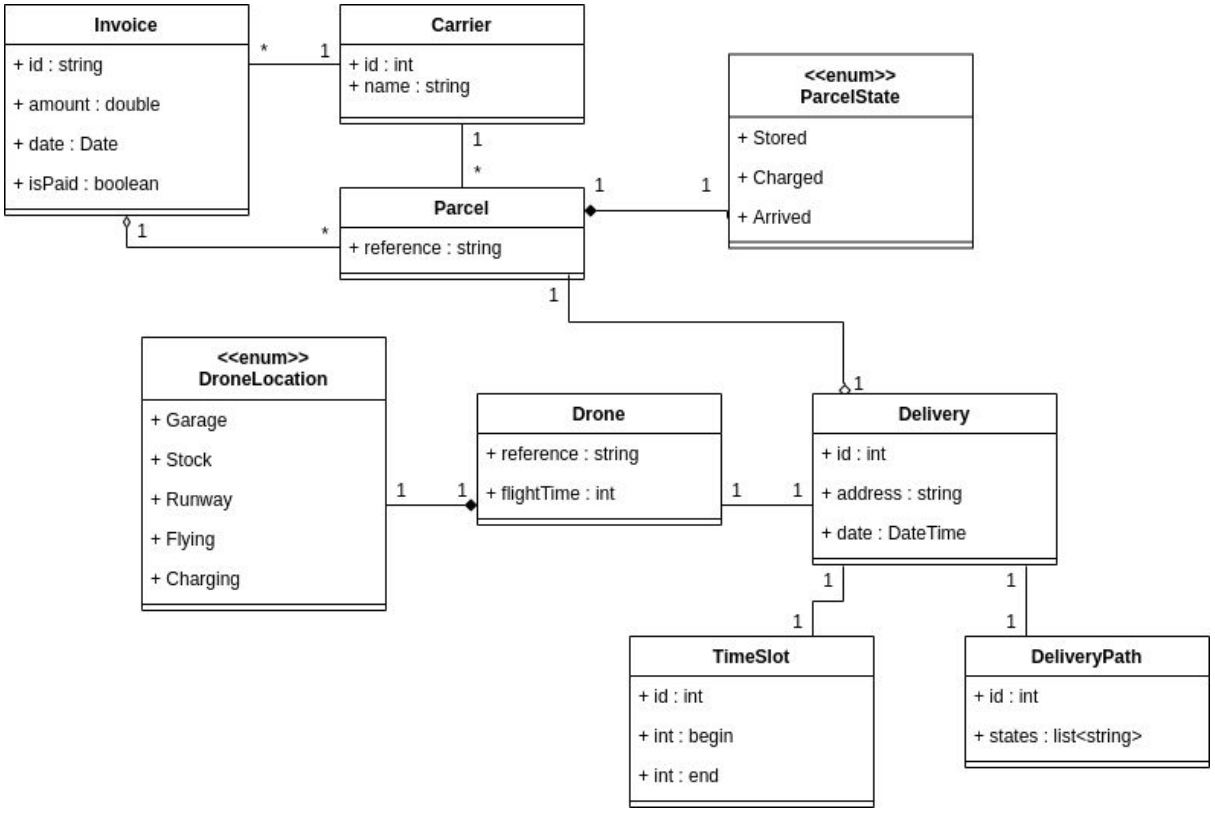
L'ensemble des clients (en vert sur le diagramme de composant) sont regroupés dans notre projet en un seul pour en simplifier l'utilisation. Il serait très simple de les diviser pour que chaque personas ait sa propre interface sur des machines séparées, car nous avons un webservice qui interagit avec la partie j2e de façon distincte pour chacun d'eux.

Notre partie j2e est composée de 6 composants ce qui permet de diviser le code et donc de regrouper les fonctionnalités et objets en fonctions de leur utilité métier. Cela permet de rendre le code plus simple et compréhensible et donc plus simple à faire évoluer et à tester.

Nous avons un composant externe à notre application. Cela permet d'ouvrir notre application à l'extérieur et utilisez des services tiers (envoyer des mails, imprimer sur un serveur d'impression …) cela donne beaucoup de possibilités d'extension pour notre application.

Nos scénarios sont complets de bout en bout c'est à dire traversant notre architecture. L'ensemble de nos composants sont utilisés, ce qui nous permet de tester que toutes nos liaisons entre les composants marchent bien.

Le diagramme de classe



Nous avons un architecture solide, chaque classe est indispensable et est persistante.

Nous avons décidé de représenter la localisation d'un drone grâce à une énumération, cela pourra permettre par la suite d'ajouter des lieux où peut être un drone, il est donc simple d'ajouter des acteurs comme par exemple quelqu'un qui vérifie l'état d'un drone à la sortie du garage.

L'objet TimeSlot représente les créneaux horaires qui peuvent être ajoutés afin de mieux planifier les livraisons. Il rend la gestion du problème de scheduling plus simple. En effet, le problème principal était d'arriver à gérer le système dans le temps, afin de donner plus de contrôle à Clissandre. Le but au travers de ce choix est de faire en sorte que les plages

horaires soient définies par Clissandre. Une fois que cela est fait, tout au long de la journée, elle peut insérer des livraisons par rapport au créneaux horaires qu'elle a ajoutés. On considère bien entendu ici que c'est Clissandre elle même qui planifie les livraisons.

La persistance

Dans notre projet nous avons un module nommé entities qui contient tout les objets qui persistent ainsi que les tests vérifiant cette persistance, ces objet sont donc stocké dans une base de donnée grâce à une interface jdbc, cela permet que lorsque le serveur s'éteint puis se rallume il ne perd pas toute les données. Vous pouvez voir ces objets dans notre diagramme de classe (cf partie diagramme de classe).
Dans nos choix d'implémentation de la persistance il y a les suivants :

- Nous avons une classe DeliveryPath qui permet à un drone de savoir par ou il doit passer pour l'acheminement d'un colis afin d'éviter des obstacles tel que des immeubles, on peut se représenter cela comme une route dans Google Maps. La création de cette route se fait manuellement pour le moment mais nous pouvons imaginer que dans une release futur qu'elle soit fait à l'aide d'une api comme google maps. Nous avons décidé de faire persister cette route car cela peut prendre un certain temps de la calculer et cela peut donc être pratique qu'elle soit sauvegarder, nous avons tout de même décider que si une livraison est supprimer cette route l'est aussi, c'est pourquoi nous avons ajouter une opération de cascade de livraison vers la route.
- Une grande partie de nos objet ont un Id généré automatiquement, cela permet de tous les differencier, les seuls objets sans Id sont Drone et Parcel (colis) car ce sont deux choses qui dans la réalité ont une référence unique inscrite dessus.

Les intercepteurs

Nous n'avons pris le temps de ne mettre qu'un seul intercepteur, ce dernier intercepte la requête envoyée au web service DeliverySystemWebService pour la méthode getParcel, il permet de vérifier que l'application demandant un colis spécifique fournisse une référence composé seulement d'un chiffre, si ce n'est pas le cas cet intercepteur retournera alors une exception. Le front gère cette exception en affichant que la référence ne correspond pas au format d'une référence. Cela permet lorsque l'on se trompe dans la référence au lieu de recevoir que le colis n'est pas trouvé, recevoir que le format de la référence est mauvais.
Cet intercepteur permet de faciliter la recherche d'un colis afin qu'on perde moins de temps en se trompant dans une référence.
Dans une prochaine release nous pouvons ajouter des intercepteurs dans un grand nombre d'interfaces de webServices afin de permettre de faire des statistiques plus détaillé et de les faires plus proprement. Nous pouvons aussi afficher les opérations effectué dans le système dans la console sans être invasif.

Les interfaces

DeliverySystem

NextDelivery:

Cette interface permet de demander la prochaine livraison que Marcel le manutentionnaire doit prendre en compte. Il a ainsi toutes les informations relatives non seulement aux colis, mais aussi aux drones qui a été associé à la livraison.

PackageRegistration:

Cette interface devrait permettre de rajouter des colis directement. Mais nous avons décidé de considérer que les données étaient reçues d'un système qui n'est, ici, pas pris en compte.

GetParcel

Cette interface permet de retourner les informations d'un colis à partir de la référence

BillingService

SendingBill

Cette interface permet d'envoyer une facture à un carrier connu du système

CreatingBill

Elle permet de créer une facture à la demande

DroneManager

GettingAvailableDrone:

Cette interface nous donne la possibilité de demander au drone manager de nous retourner un drone disponible pour une livraison. Il renverra le premier drone disponible qu'il retrouve.

DeliveryReport:

Bien que non implémentée, elle devrait permettre d'avoir des retours sur des livraisons faites. Mais c'était plus où moins difficile à implémenter et aurait demandé la modification de plusieurs objets du système.

ChangingDroneState:

Cette interface permettra de changer l'état d'un drone notamment par Garfield ou Charlène dans le cas où ce dernier ne serait pas en état de voler par exemple. Cette future n'était pas réellement prioritaire au vu des tâches qui nous incombent. Nous avons donc décidé de la laisser pour plus tard.

Planning

AskForAdelivery

Cette interface est celle qui permet de demander la création d'une livraison par Clissandre. A partir de commandes plutôt simples. Pour ce faire, il faudra connaître notamment la référence du colis et choisir une page horaire parmi celles disponibles.

GetAvailableTimeSlots:

On considère ici les timeSlots comme des créneaux horaires de 2h ajoutés par l'admin pour permettre de placer des livraisons. Le but étant de moduler les planifications, cette interface renverra uniquement les créneaux horaires disponibles.

Feedback

sendFeedback

Cette interface est celle qui permet de demander la création d'un feedback dans la database par le client. Nous lui donnons en paramètre un Feedback (un nom et une note)

getFeedbacks

Nous pouvons grâce à cette interface récupérer la totalité des feedbacks qui ont été faits, cette interface est notamment utilisé afin de faire des statistiques sur les retours des utilisateurs

Statistics

getStatsBills et getStatsDeliveries

Bien que non utilisées dans l'application, ces interfaces permettent d'obtenir des statistiques sur les factures et les livraisons. Pour les ajouts de futurs statistiques nous rajouterons des fonctions dans ces interfaces dans le composant statistique

StateFull and Stateless Components

Pour ce qu'il en est des différents composants (beans), nous étions partis sur la création de Stateful Components. Mais nous nous sommes rendus comptes que nous n'avions pas réellement besoin de faire perdurer l'état des composants à partir du moment où le singleton "**Database**" nous permettait de garder toutes les informations qu'on souhaitait dans la session courante. Il n'était pas nécessaires par exemple de garder l'état du "**DeliverySystemBean"** pour garder une trace des différentes livraisons créées. Par contre, une de nos fonctionnalités a bouleversé la stabilité de notre système. En effet, la méthode **getNextDelivery** de l'interface "**deliverySystem**" nous permet de récupérer la prochaine livraison. Pour le coup le principale problème était de faire persister une liste et de réussir à récupérer le suivant dans cette liste à chaque fois. Ce que nous n'avons malheureusement pas su comment gérer. Pour le coup, cette partie du système nous n'avons pas réussi à la faire persister. Ce problème est probablement dû à une mécompréhension de l'utilisation sur l'utilisation du singleton "**Database**". Au départ, nous avions vu le singleton comme une base de données.

Synthèse

Nous avons été très contraint en temps, nous avons tous été occupés par différents projets et nous avons rencontré des problèmes pendant de longues semaines au début. Tout n'a donc pas été entièrement implémenté, mais nous avons fait les meilleurs choix possible afin de proposer des scénarios pour un maximum d'utilisateur. Nous avons aussi proposé la meilleure qualité sur ce que nous avons fait par le biais de tests Cucumber et unitaire. Nous avons un backlog toujours en ligne sur GitHub.

Nous avons découpé et responsabiliser au mieux notre code, chaque fonctionnalité différente est dans un composant qui lui est propre afin que nous puissions agir indépendamment sur chaque composant. Cela offre une meilleure lecture du projet ainsi qu'une maintenabilité améliorée.

Nous avons créé tous les composants et interfaces nécessaires au fonctionnement complet du système, et chacun comporte tous les dossiers nécessaires à son bon fonctionnement. Il est donc très simple d'ajouter des fonctionnalités au projet en interagissant directement avec le code du composant associé ou même d'ajouter un composant au projet en suivant le modèle de ceux déjà en place.

Pour le front, tous les utilisateurs sont présents dans un seul et même client, afin de matcher avec le "vrai monde" il faudrait découper un client par utilisateur, mais nous avons fait ce choix afin de rendre les tests plus simples et d'avoir un meilleur visuel de nos résultats, ce n'était donc pas une de nos priorité de le découper tout de suite.

Pour le J2E, nous avons la volonté de lui léguer la totalité des calculs, afin d'alléger le front un maximum. Nous avons aussi pour objectif de ne pas envoyer d'objet complet au front mais seulement les données traitées de ceux-ci. Nous n'avons pas eu le temps de respecter ce choix au maximum. Pour les statistiques (seulement) les objets feedbacks sont renvoyés au client qui fera ensuite les calculs. Nous avons conscience que ce n'est pas le meilleur choix, et dans nos priorités futurs, nous transférerons ces opérations côté back.

Pour le .NET nous avons implémenté la connexion de bout à bout et cela fonctionne correctement (et c'est testé). Les données envoyées ne sont pour le moment pas des plus pertinentes, mais comme la liaison est faite et que nous arrivons à communiquer nos objets métiers, nous pourrons facilement matcher à un cahier des charges plus complexe ou à des demandes plus variées.