GUILLEMET Samuel

FERET Nathan

CSC 5002

Middleware and software architecture for distributed applications

Rapport final

Introduction

En réponse à la demande croissante d'amélioration de l'expérience touristique à Paris, la ville prend des mesures importantes pour améliorer le système de vélos VLib de Paris grâce à l'introduction de VlibTour. VlibTour est conçu pour répondre aux besoins des groupes de touristes qui souhaitent explorer les points d'intérêt (POI) renommés de Paris. Cette application offre aux participants la possibilité de créer leurs propres circuits personnalisés en sélectionnant une séquence de points d'intérêt. Une interface permet d'accéder aux cartes des circuits, de partager la localisation en temps réel entre les membres du groupe.

Le cœur de ce projet réside dans le développement de la preuve de concept (POC, se concentrant sur des composants vitaux tels que le stockage de la base de données pour les POI et les visites, un serveur d'émulation de visite pour gérer les mouvements des utilisateurs, la communication via RabbitMQ, l'affichage des cartes en temps réel, et le partage sécurisé des informations via les hôtes virtuels RabbitMQ (vhosts).

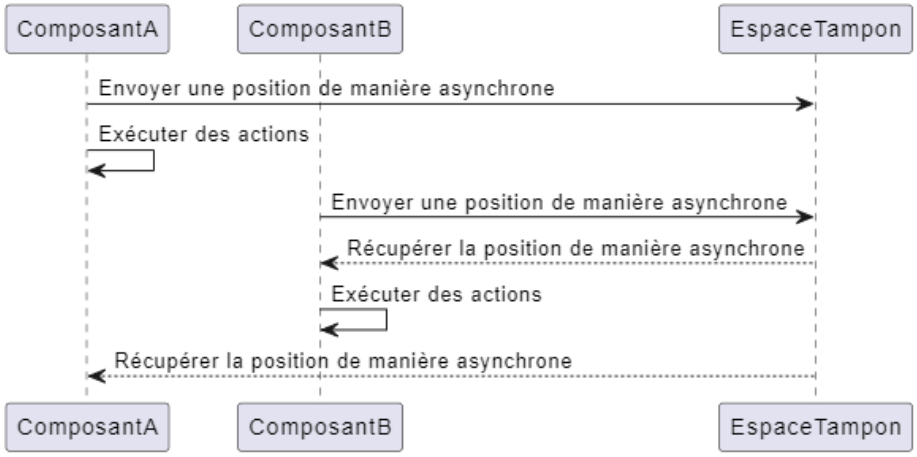
Proof of Concept

Patron de conception et architecture

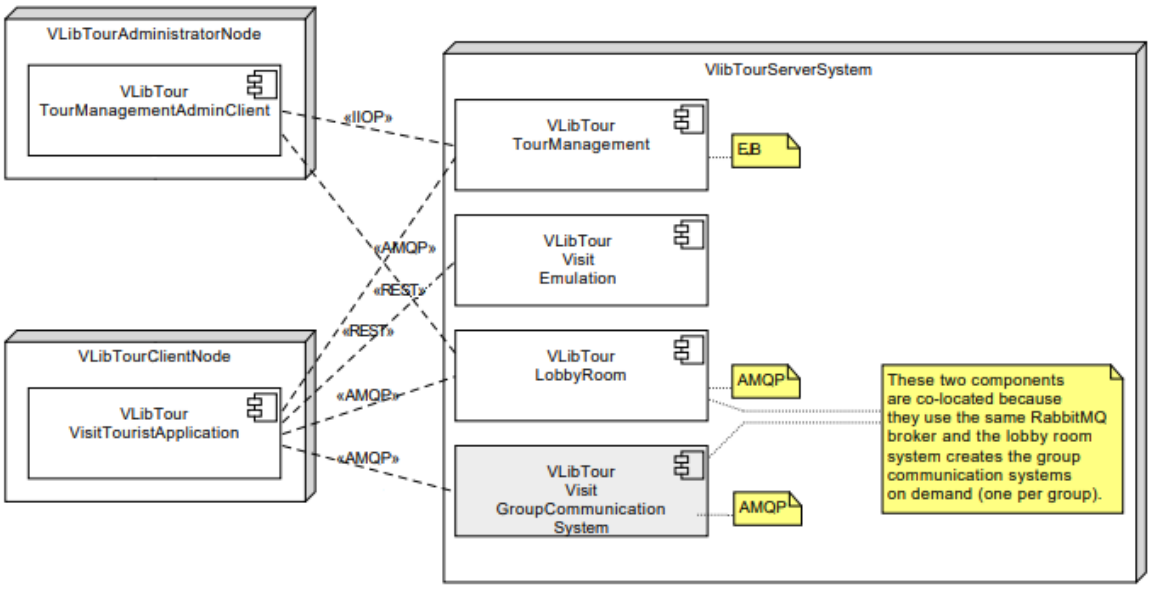
Plusieurs micro-services ont été développés pour gérer les différentes fonctionnalités de l'application. Afin d'interagir avec eux depuis l'application client, le patron de conception **Proxy** a beaucoup été utilisé. Il nous a été utile lorsque nous avons souhaitez ajouter une couche de contrôle et d'abstraction autour d'un objet existant sans que le client ait besoin de connaître tous les détails de cette abstraction. Cela a réussi à améliorer la modularité, la sécurité et les performances de notre application. Nous détaillons plus trad dans le rapport un exemple du patron de conception **Proxy**.

Par ailleurs nous avons utilisé des messages MQTT via RabbitMQ ce qui satisfait le patron de conception **d'appel asynchrone et message en mémoire tampon**. Ce patron de conception est couramment utilisé dans les systèmes distribués, les systèmes de communication inter-processus (IPC). Il contribue à améliorer la performance et la scalabilité de l'application en évitant les blocages inutiles et en permettant aux composants de travailler de manière asynchrone. Par exemple dans notre application il utile pour partager les positions des différents utilisateurs sans blocage de l'application.

Par exemple les composant A et B peuvent continuer de faire des actions sans bloquer pour attendre la position de l'autre.



Architecture du POC



Les 4 composants développés sont les suivants :

Tour Management Le composant de gestion des circuits est utilisé par le voyagiste pour préparer les circuits disponibles proposés aux touristes. Il est également utilisé par les touristes pour découvrir les circuits disponibles.

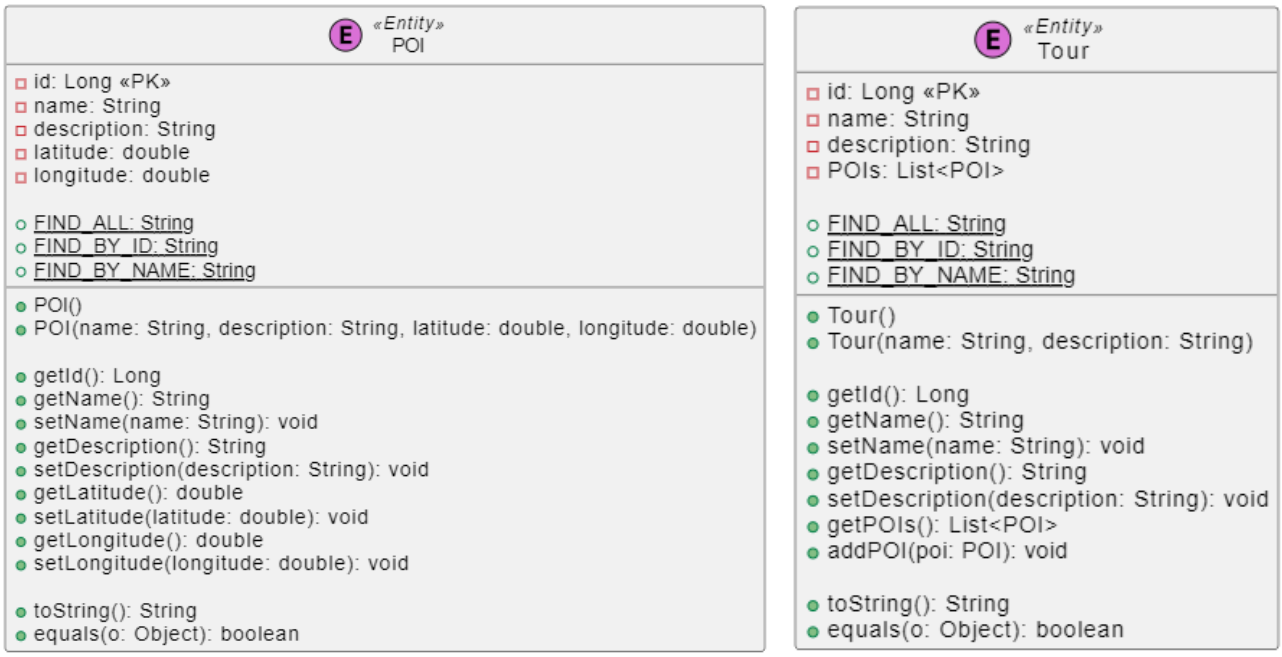
VisitEmulation Ce service est capable de suivre un groupe de touristes pour une visite, en particulier il met à jour le prochain POI pour la visite.

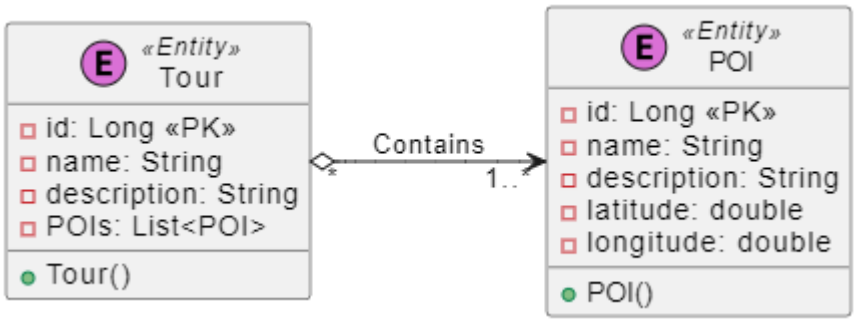
Lobby Room La salle d'accueil est l'endroit où les touristes ont accès au système pour créer une visite et le groupe de participants correspondant. La fonction principale d'une salle d'accueil est de créer un groupe pour une visite et de permettre à un touriste de rejoindre un groupe existant. Rejoindre un groupe/une visite signifie obtenir l'accès au système de communication du groupe de visite correspondant.

Visit Group communication system Le système de communication du groupe de visite est le système responsable de la mise en œuvre d'un système de diffusion fiable aux seuls participants du groupe constitué pour une visite.

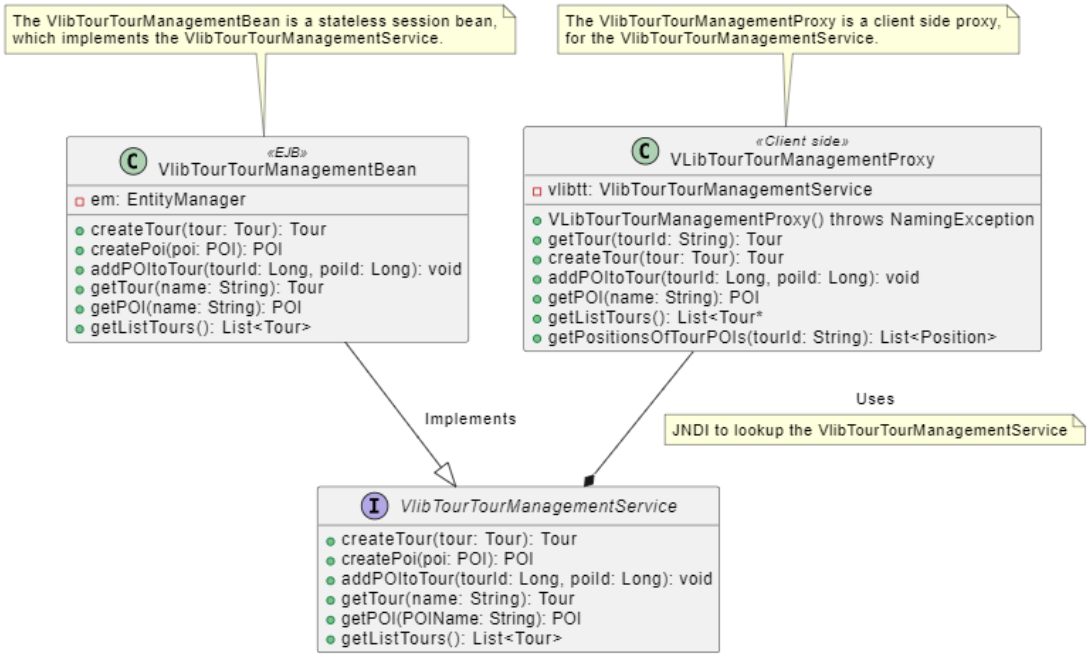
Choix de design

Pour le stockage des données nous avons décidé de stocker les visites ainsi que les point d'intérêts associés.



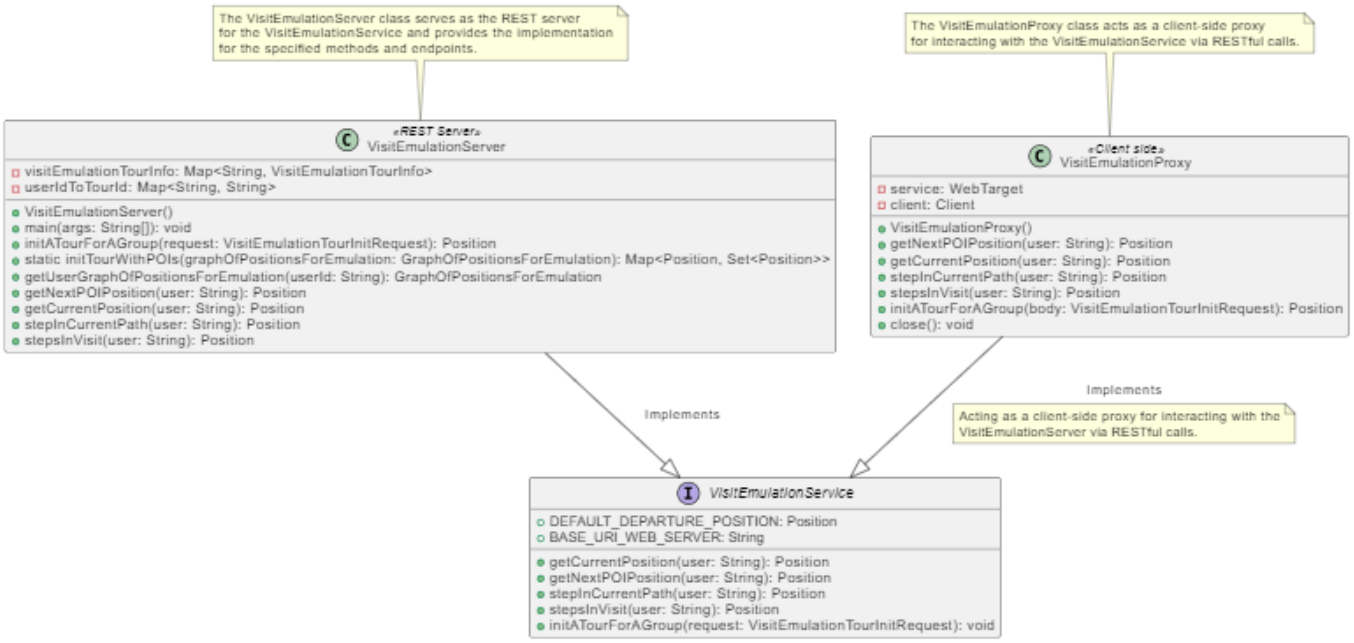


Ces objets sont gérés par le composant **TourManagement :**



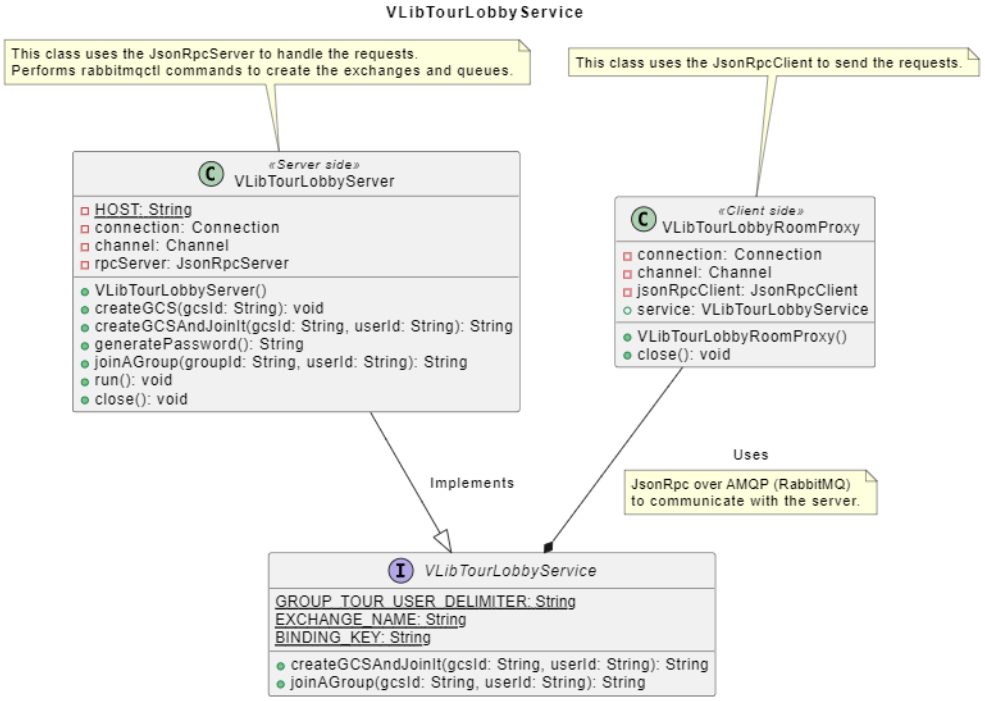
Le design sélectionné pour ce composant consiste à utiliser une *Bean* qui s'exécute dans Glassfish et qui exploite une base de données Glassfish.

On retrouve après le composant **VisitEmulation** qui est utile pour gérer le déplacement des utilisateurs entre les différents points d'intérêt. Il convient de préciser que ce composant est utile dans un but de démonstration car dans l'application réel les utilisateurs se déplace réellement dans Paris.



On communique en utilisant une API *REST* pour récupérer les différentes informations.

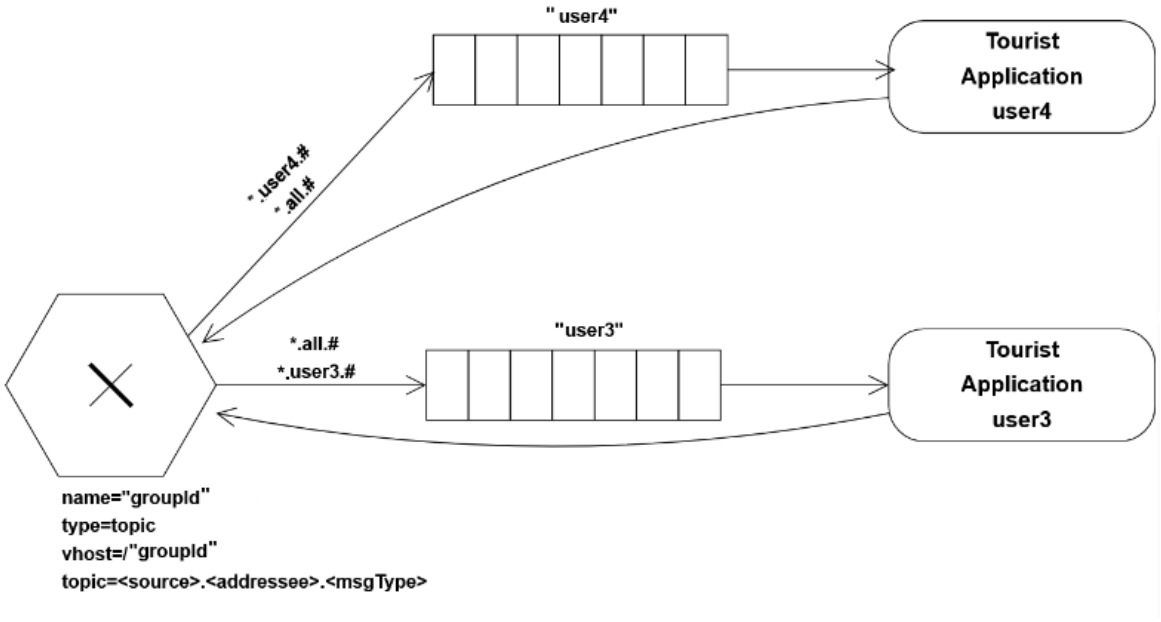
Ensuite on a le composant **Lobby Room,** dont l'objectif est de sécuriser notre application en restreignant l'accès des utilisateurs à un groupe de communication spécifique. Nous ne voulons pas que tout le monde ait accès aux informations de tout le monde. Pour cela, nous avons décidé d'utiliser les vhosts de RabbitMQ. Pour cela nous avons utilisé *JsonRPC* avec RabbitMQ.



Enfin le dernier composant qui a été développé est le **Visit Group communication system.** Il est important de noter que ce composant s'exécute côté client de l'application. Son rôle principal est de gérer la communication entre les différents utilisateurs du groupe, que ce soit pour partager les positions des utilisateurs ou pour permettre des échanges de messages en communication individuelle (1 à 1). Pour cela nous utilisons RabbitMQ sur le vhost précédemment créé par le composant **Lobby Room** et on publie les données en suivant un schéma de clé de routage bien spécifique.

Par exemple si Joe veut envoyer sa position à tout le monde il va utiliser la clé de routage « joe.all.position » qui contient son propre nom, joe, pour savoir qui envoi le message, all, pour définir que tout le monde doit recevoir ce message, et enfin position, pour savoir qu'il partage une position. Chaque utilisateur de ce groupe de communication s'abonne aux clés de routages suivantes : « *.all.# » pour recevoir tous les message qui concernent tous les utilisateurs et « *.own-user-id.# » pour les message qui le concerne seulement.

On retrouve alors ce schéma de fonctionnement :



Choix de technologies

Le projet utilise docker pour lancer les différents composants comme Glassfish ou RabbitMQ.

Comment tester l'application

Pour tester l'application vous pouvez vous référer au différents READMEs de l'application mais le plus simple est de lancer le script *run_scenario_w_mapviewer.sh* qui démarrera la démonstration du code.

Propositions architecturales pour les exigences extra fonctionnelles

En prenant en considération notre architecture actuelle, nous pouvons aborder la question de la sécurité de manière plus approfondie. Pour garantir la sécurité des informations partagées, nous avons mis en place une solution efficace en utilisant les vhosts de RabbitMQ. Cela permet de restreindre l'accès aux données aux utilisateurs autorisés. Cependant, il est également essentiel de prendre en compte la sécurité de l'accès aux données stockées dans la base de données. Pour cela, des mesures de sécurité additionnelles doivent être mises en place afin de garantir l'intégrité et la confidentialité des données stockées.

Ensuite pour l'exigence d'interopérabilité, il faudrait définir un schéma OpenAPI par exemple pour les services REST, l'utilisation de RabbitMQ est facile à déplacer d'un langage à un autre.

Informations supplémentaires

En complément de la réalisation du POC limité à la visite des Daltons, nous avons entrepris une refonte complète du service d'émulation de visite REST. Cette refonte vise à rendre le service flexible et adapté à une utilisation avec n'importe quel circuit touristique, quelle que soit la composition du groupe. Cette évolution est essentielle pour rendre notre application plus polyvalente et ouverte à une plus grande variété d'utilisations.

En outre, il est important de souligner que, dans l'application cliente, nous n'avons pas incorporé de référence spécifique à l'exemple du POC. Cette décision découle de notre vision selon laquelle notre système a été conçu avec la perspective de son extension à une échelle beaucoup plus vaste que le simple proof of concept initial. Cette conception permettra une adaptation aisée à de futurs cas d'utilisation et à une croissance potentielle de notre application. En résumé, notre approche est axée sur la flexibilité et la scalabilité, ouvrant ainsi la voie à de nouvelles opportunités et applications.