HAI704I - Architectures Logicielles Distribuées

Rapport du 2nd projet

Démonstration vidéo :
https://www.youtube.com/watch?v=W8zlfpcd-4g

ALMALLOUHI Mohamad Satea (22011503)
DAYIOGLU Gurgun (22024990)

Introduction

Ce rapport présente un système de réservation d'hôtels conçu pour les agences de voyage, mis en œuvre à l'aide de Java API for XML Web Services (JAX-WS). JAX-WS est une technologie qui permet de créer des services web basés sur le protocole SOAP, un protocole standard pour l'échange de messages XML sur Internet.

Le projet se compose de plusieurs packages Java qui définissent les entités clés telles que les *hôtels*, les *chambres*, les *clients* et les *réservations*, ainsi que les interfaces pour les services hôteliers et de réservation. Chaque entité est représentée par une classe et ses attributs et des méthodes spécifiques qui permettent de manipuler les données de réservation. En outre, les interfaces de services définissent les opérations exposées par le serveur web, telles que la recherche de chambres disponibles et la réalisation de réservations.

Le système permet aux agences de voyage de s'interfacer avec les services hôteliers via des classes clients générées par *wsimport* qui utilisent des liens URL pour se connecter aux points de terminaison des services web. Ces points de terminaison sont exposés par des hôtels partenaires, permettant ainsi aux agences de réserver des chambres en temps réel et de bénéficier de réductions spécifiques. Le système gère également les exceptions personnalisées pour répondre aux différents cas d'erreur qui peuvent survenir lors de la réservation.

Conception du système

La structure adoptée pour ce projet repose sur des principes d'architecture orientée services (SOA). L'adoption de JAX-WS et du protocole SOAP s'aligne avec ces principes, car elle permet de définir des interfaces claires entre les services consommés et les clients qui les utilisent.

Cette structure modulaire a plusieurs avantages :

- Encapsulation : Chaque entité (hôtel, chambre, client, réservation) est définie comme une classe distincte avec ses propres attributs et méthodes. Cela permet d'encapsuler et de gérer la complexité, rendant le code plus lisible et plus facile à maintenir.

- Réutilisation : Les interfaces de services, *HotelService* et *ReservationService*, peuvent être implémentées par différents hôtels, permettant ainsi aux agences de voyage de communiquer avec de multiples partenaires via une interface commune.

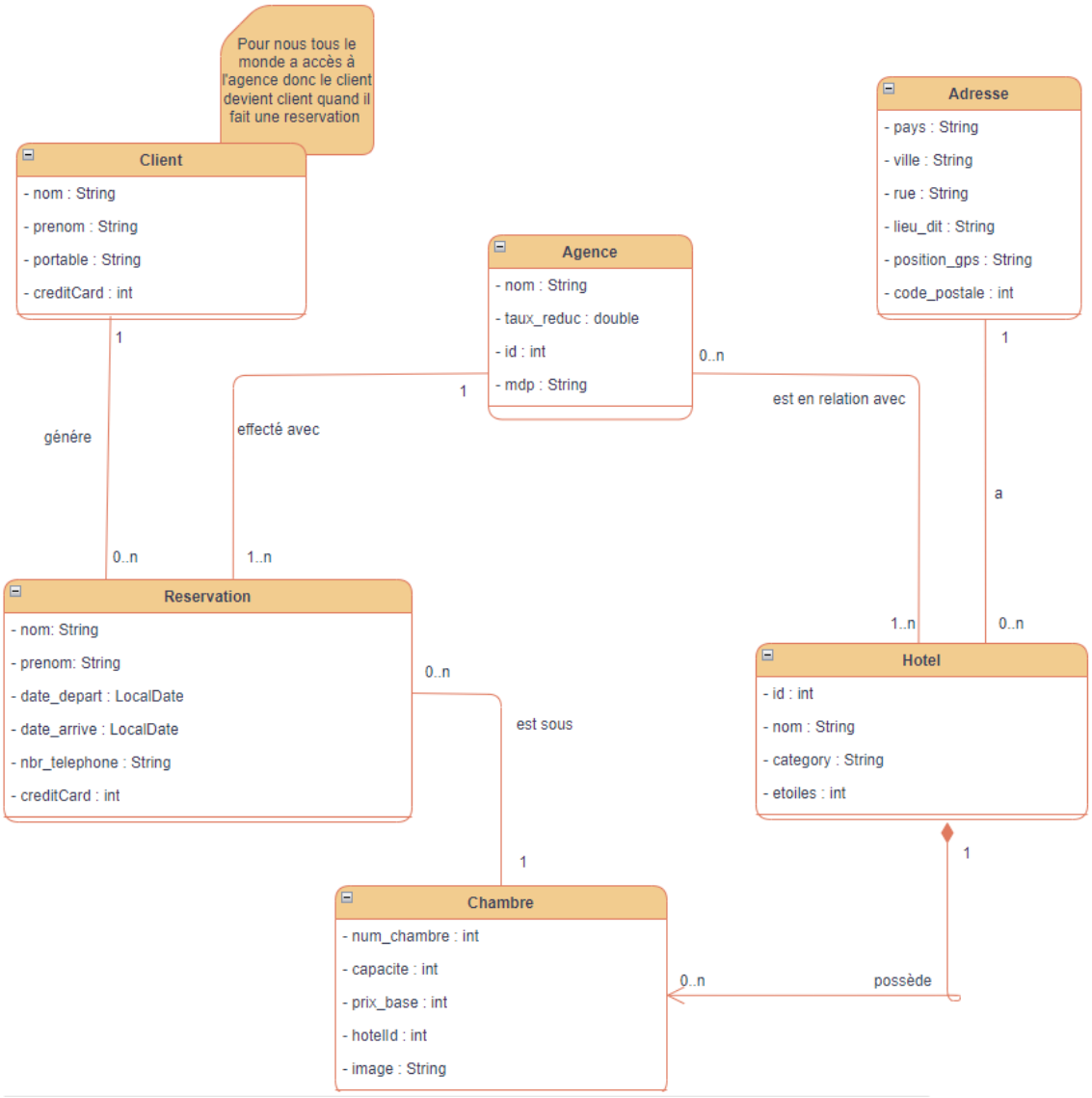
- Interopérabilité : L'utilisation des standards SOAP et XML garantit que le système peut interagir avec une variété d'autres systèmes, même ceux écrits dans d'autres langages de programmation.

- Gestion des exceptions : La définition d'exceptions personnalisées permet une gestion fine des erreurs, facilitant le diagnostic et la récupération en cas de problème lors de la réservation.

Nous ne parlerons pas trop de la première version de l'application, la version non-distribuée, car nous avons réutilisé presque tous ses composants pour créer la version finale et distribuée.

L'important était de s'assurer que nos classes et la structure générale de notre API étaient extensibles. Par exemple, nous avons décidé qu'il était beaucoup plus facile de mettre les réservations dans la classe *Chambre* que de les placer dans une liste plus générique dans la classe *Hotel*. Cela nous a permis de vérifier beaucoup plus rapidement si une chambre était prise ou non pour une période donnée.

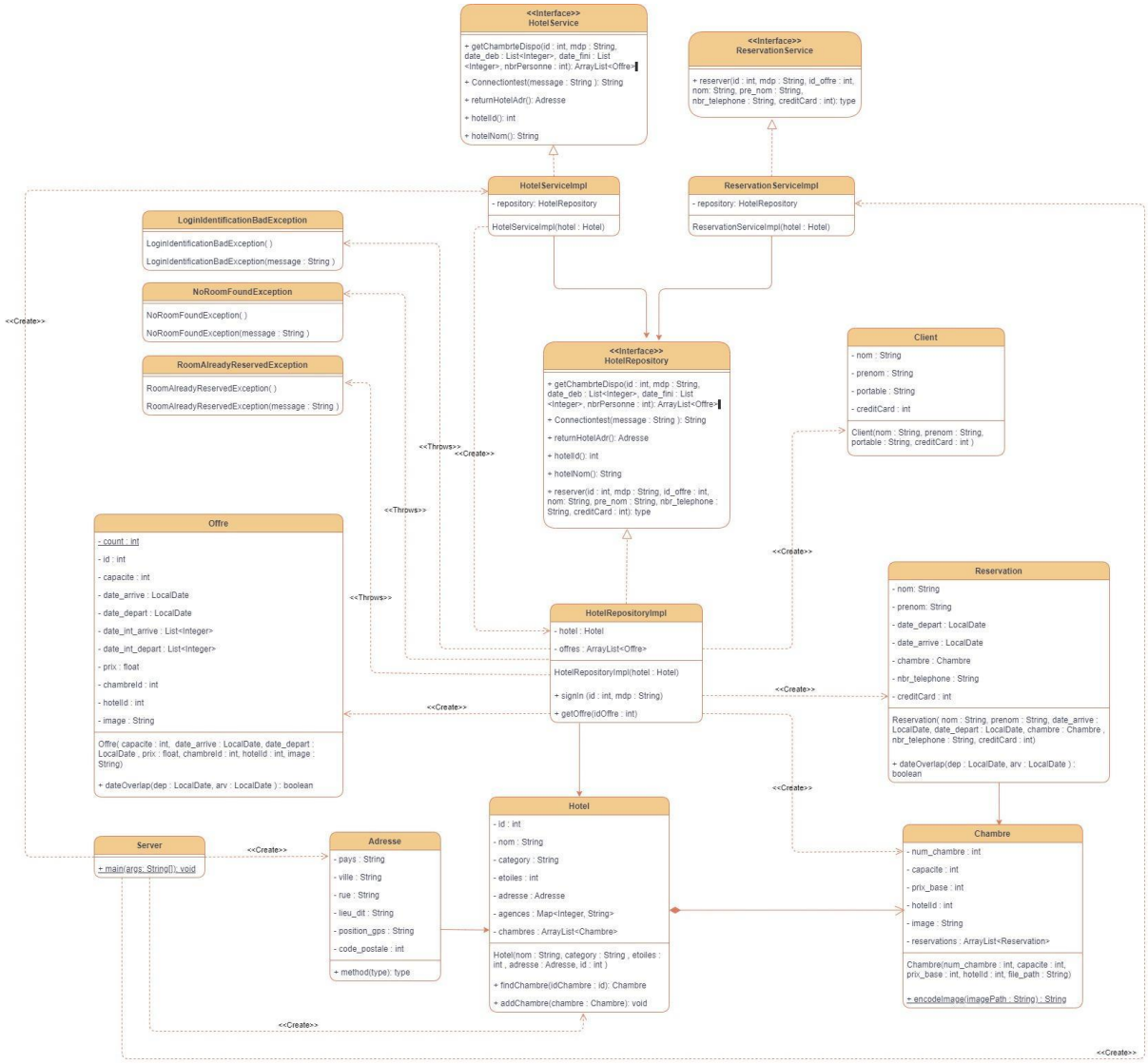
Posons le diagramme UML de la première version de notre application :



Détails de l'implémentation

La version finale de l'implémentation étant assez complexe, il est plus logique de commencer par jeter un coup d'œil à son diagramme UML (ce diagramme peut également être trouvé dans le fichier ZIP).

Ici, nous pouvons voir les principales classes et leurs relations dans le contexte d'une application qui permet aux utilisateurs (ou plutôt aux agences) de faire des réservations :



Le package *hotel.web.service*

Ce package fournit une implémentation des services web pour la réservation d'hôtels. Le système est censé être utilisé par les agences et non par les clients. Les clients feront face aux agences et auront leurs propres interactions. Nous les verrons plus tard.

Services Principaux

HotelServiceImpl et *ReservationServiceImpl* sont les classes centrales qui exposent les services web. Elles implémentent respectivement les interfaces *HotelService* et *ReservationService*.

HotelServiceImpl fournit l'accès aux informations et à la disponibilité des chambres d'hôtel. Il permet aux utilisateurs de rechercher des chambres en fonction de critères spécifiques et de gérer les données de l'hôtel via le *HotelRepository*.

ReservationServiceImpl est responsable de la gestion des réservations. Il utilise le même référentiel (*HotelRepository*) que *HotelServiceImpl* pour assurer la cohérence des données.

Référentiel de l'Hôtel

HotelRepositoryImpl agit comme une couche d'accès aux données (DAO - Data Access Object). Elle encapsule l'accès et la manipulation des entités *Hotel*, *Chambre*, *Offre* et *Reservation*. C'est ici que se déroule la véritable mise en œuvre des méthodes.

Gestion des Chambres Disponibles

Les chambres sont créées avec des attributs tels que la capacité, le prix de base et une image. Ces chambres sont ensuite utilisées pour créer des offres (*Offre*) qui sont des représentations de la disponibilité des chambres à des dates spécifiques et à un prix donné.

Authentification et Sécurité

Une méthode d'authentification (*signIn*) vérifie les identifiants des agences de voyage, assurant que seules les entités autorisées peuvent accéder aux services de réservation. Cette méthode est utilisée à travers les deux services pour valider les actions des utilisateurs.

Exceptions

Des exceptions personnalisées comme *NoRoomFoundException* et *LoginIdentificationBadException* sont utilisées pour gérer les erreurs liées à la non-disponibilité des chambres et aux échecs d'authentification.

Méthodes Supplémentaires

D'autres méthodes, comme *Connectiontest*, *returnHotelAdr*, *hotelId* et *hotelNom* fournissent des fonctionnalités auxiliaires.

Classe Server

Le fichier *Server* définit le point d'entrée pour le déploiement des services du système. Elle contient la méthode *main* qui est le point de départ pour exécuter l'application serveur. Elle initialise et publie les services web pour plusieurs hôtels en utilisant la classe Endpoint.

Pour chaque hôtel, nous créons une instance de *HotelServiceImpl* qui est ensuite publiée comme un service web à une URL spécifique.

De même, pour chaque service de réservation, nous créons une instance de *ReservationServiceImpl* en utilisant l'instance *HotelServiceImpl* existante. Cela garantit que les deux services partagent le même *HotelRepository*, assurant ainsi la cohérence des données entre les services de l'hôtel et de réservation.

Le package *agence.web.service*

Il contient la logique qui permet à une agence de voyage d'interagir avec les services web des hôtels.

Classe *AgenceImp*

La classe *AgenceImp* est la principale classe de ce package, orchestrant la communication entre les agences et des multiples services web. L'objectif ici est de fournir un accès unifié à une variété de services de réservation et d'opérations hôtelières, tels que la vérification de la disponibilité des chambres et la gestion des réservations.

Le constructeur de *AgenceImp* prend une instance de *Agence* qui contient les détails de l'agence, y compris son nom, son taux de réduction, son identifiant et son mot de passe. Les URLs des services web de chaque hôtel sont stockées au sein de l'instance d'*Agence*. Lors de l'initialisation, *AgenceImp* établit des connexions avec ces services en utilisant les URLs fournies et stocke les proxies des services correspondants dans les listes *hotelS1* et *hotelS2*. Ces proxies permettent à l'agence d'invoquer des opérations distantes de manière abstraite et simplifiée.

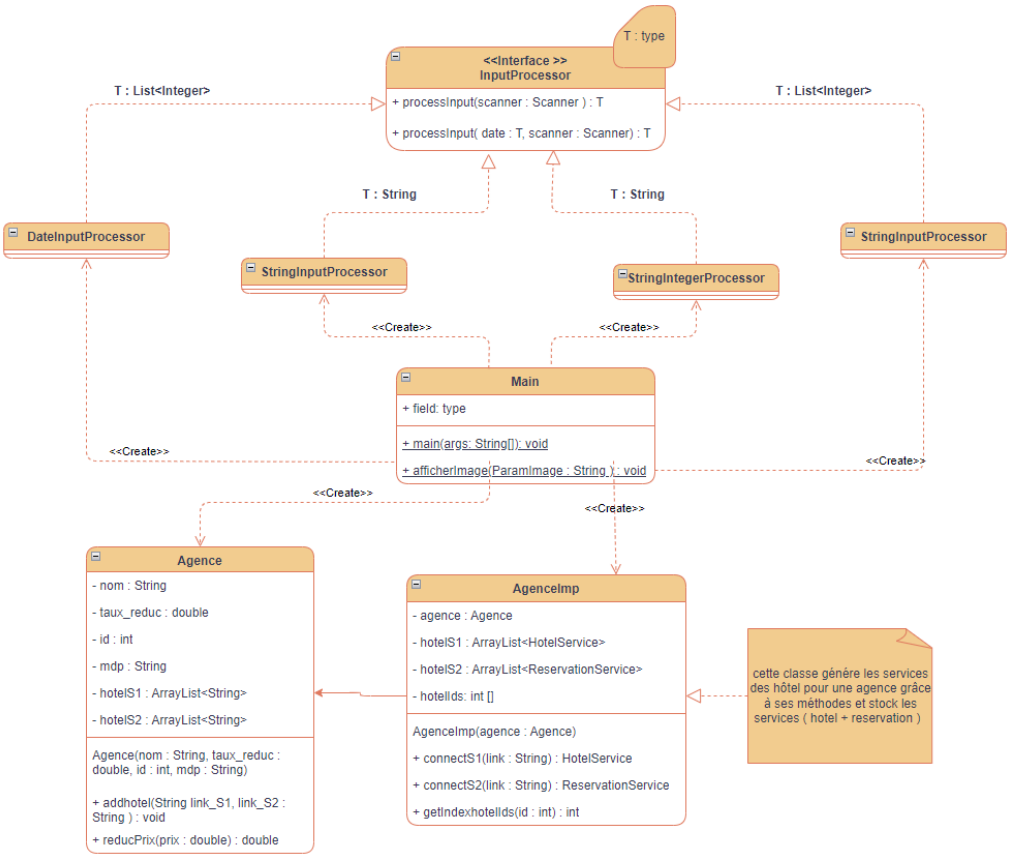
Connexions

La méthode *connectS1* crée une connexion à un service web d'opérations hôtelières, tandis que *connectS2* se connecte à un service de réservation. Ces méthodes retournent un proxy qui peut être utilisé pour effectuer des appels de méthode à distance. Ces connexions sont essentielles pour permettre à l'agence d'interagir avec les hôtels de manière dynamique et flexible.

Le tableau *hotelIds* joue un rôle de référence pour les identifiants des hôtels qui sont utilisés pour accéder aux services spécifiques à un hôtel donné. La méthode *getIndexhotelIds* permet de retrouver l'indice associé à un identifiant d'hôtel dans ce tableau, ce qui facilite la correspondance entre les identifiants et les services.

Classe Main (client)

La classe *Main* joue le rôle de point d'entrée pour l'interface client. Elle orchestre l'interaction entre l'utilisateur et les services web. L'aspect le plus notable de cette classe est l'utilisation de différents processeurs d'entrée (InputProcessor) pour faciliter et guider l'utilisateur à travers diverses étapes de la réservation.



Notre structure présente de multiples avantages :

- Simplicité : La séparation claire des services de réservation et d'opérations hôtelières dans deux listes distinctes simplifie leur gestion.

- Flexibilité : La création dynamique de connexions aux services web permet à l'agence de s'adapter facilement aux changements d'URLs ou d'ajouter de nouveaux hôtels au système.

- Cohérence des Données : En utilisant un tableau pour lier les identifiants des hôtels aux services, le système assure que les réservations sont effectuées avec les bonnes références, réduisant ainsi les erreurs.

Command-Line Interface

Expliquons maintenant à quoi ressemble une interaction entre le client et le service.

```
Entrez l'indice de la'agence : 1
Les villes disponibles:
0)Cologne / Germany
1)Homs / Syrie
Entrez l'indice de la ville : 1
La date d'arrivée - Entrez la date (jour mois année) : 23 03 2024
La date de départ - Entrez la date (jour mois année) : 23 03 2025
Entrez le nombre de personnes : 5
le nombre de personnes invalide. Veuillez entrer un nombre entre 1 et 4.
```

Un client choisit d'abord son agence préférée. Il peut ensuite filtrer les hôtels pour trouver celui qui lui convient. Avant de procéder à une réservation, il a la possibilité de voir une image de la chambre en question.

```
Entrez le nombre de personnes : 3
0) Hotel:Al-Safir, Prix:3600.0, capacite:4, disponiblite:[23, 3, 2025] - [23, 3, 2027]
1) Hotel:Ghornata, Prix:4500.0, capacite:4, disponiblite:[23, 3, 2025] - [23, 3, 2027]
2) Hotel:Ghornata, Prix:4500.0, capacite:4, disponiblite:[23, 3, 2025] - [23, 3, 2027]
3) Hotel:Al-Hamra, Prix:5400.0, capacite:4, disponiblite:[23, 3, 2025] - [23, 3, 2027]
4) Hotel:Al-Hamra, Prix:5400.0, capacite:4, disponiblite:[23, 3, 2025] - [23, 3, 2027]
Entrez le nombre de l'offre a reserver : 1
Entrez 0 Si vous rentrer en arrière et 1 si vous continuez la reservation : 1
```

Intégration des images:

L'intégration des images joue un rôle crucial en enrichissant l'expérience de l'utilisateur. Cette intégration s'articule autour de deux aspects principaux : le stockage et la transmission des images, et leur affichage côté client.

Stockage et transmission des images

Les images des chambres d’un hôtel sont gérées à deux niveaux.

Au début, chaque *Chambre* est associée à un chemin d'accès à une image. Lors de son instanciation, l'image fournie est encodée en Base64. Cette méthode de codage transforme les données binaires de l'image en une chaîne de caractères ASCII, adaptée pour être transmise sur Internet.

Et pendant que le serveur envoie les offres, qui sont des représentations des chambres disponibles pour réservation, il inclut également l'image de la chambre correspondante.

Affichage des images au coté client

La méthode *afficherImage* dans la classe Main est utilisée pour le décodage et l'affichage des images. Cette méthode prend la chaîne encodée en Base64, la décode pour récupérer les données binaires, puis crée une image affichable à partir de ces données. Cette image est ensuite présentée dans l'interface utilisateur, permettant aux utilisateurs de visualiser la chambre ou l'offre sélectionnée.

Difficultés rencontrées

La restriction de SOAP à ne pas accepter les attributs publics a posé un défi initial. Cela a nécessité une compréhension plus profonde du fonctionnement de SOAP, et a mené à des blocages initiaux dans le développement. La résolution de ce problème a impliqué l'adoption de la pratique standard en Java de définir des attributs privés avec des accesseurs (getters) et des mutateurs (setters), ainsi que des constructeurs vides.

Une difficulté majeure a été la gestion des références circulaires entre les entités *Hotel* et *Chambre*. Une telle interdépendance pose des problèmes dans les systèmes basés sur SOAP, où une entité ne peut pas avoir une référence directe à une autre si cette dernière a une référence réciproque.

Conclusion

En menant à bien ce projet, nous avons acquis des compétences précieuses et nous avons approfondi notre compréhension de plusieurs aspects clés du développement de logiciels distribués. L'utilisation de Java combinée à SOAP et WSDL nous a offert une expérience pratique dans la création des applications robustes et interopérables.

Cette expérience a été enrichissante non seulement en termes de maîtrise technique mais aussi pour notre capacité à résoudre des problèmes complexes et à adapter notre conception aux exigences spécifiques des services web. Nous avons également appris l'importance de la validation des entrées utilisateur et de la gestion des exceptions dans un environnement distribué.