



HAI704I - Architectures Logicielles Distribuées

Rapport du 3ème projet REST

Démonstration Agence :

<https://drive.google.com/file/d/1qPcmYMxVcPev3nobQyS2Z5BC410evnLd/view?usp=sharing>

Démonstration Comparateur :

<https://drive.google.com/file/d/1S-XASSmG6NW1jF11srHDvVSIGIZ4ST-J/view?usp=sharing>

ALMALLOUHI Mohamad Satea (22011503)
DAYIOGLU Gurgun (22024990)

Table des Matières

Table des Matières	2
Introduction	3
Conception de l'Application	3
UML de HotelService	4
Classe Hotel	4
Classe Offre	5
Classe Client	5
Classe Chambre	5
Classe Reservation	6
UML de AgenceService	6
Classe Agence	7
Classe Offre	7
Classe Hotel	7
Implémentation de l'Application	7
Les Services Exposés de HotelController	8
GET /disponibilites	8
GET /reserver	8
GET /Chambre/{id}	8
Les Services Exposés de AgenceController	9
GET /api/comparable	9
Méthode Auxiliaire invokeOffres	9
GET /api/reserver	9
Les Services Exposés de AgenceWebController	9
GET /Accueil	9
POST /Accueil	9
GET /AfficherOffres	9
GET /AfficherOffre	9
GET /Reservation	10
POST /Reservation	10
GET /Valider	10
GET /Problem	10
Exemple : Réservation d'Une Chambre	10
Comparateur (Trivago)	13
Conclusion	15

Introduction

Ce rapport explore la conception et la mise en œuvre d'un système de réservation d'hôtel à l'aide de framework Spring. En utilisant Spring Boot, nous pouvons facilement construire des services web REST.

Avant de commencer directement par la conception, il est important de comprendre les fonctionnalités demandées. Commençons par en énumérer les principales :

- L'utilisateur doit pouvoir rechercher des hôtels et faire des réservations par l'intermédiaire d'une agence. Naturellement, l'utilisateur est le client d'une agence et n'interagit pas directement avec les hôtels.
- L'agence de voyage achemine ensuite ces demandes vers différents hôtels et renvoie les résultats. Ces demandes doivent être authentifiées.
- L'agence peut appliquer des réductions spécifiques avant de fournir les résultats. Cela signifie que différentes agences peuvent avoir des prix différents pour les mêmes offres.

En prenant compte ces points, nous avons choisi d'utiliser qu'un seul contrôleur pour les hôtels (HotelController) et deux contrôleurs pour les agences (AgenceController et AgenceWebController). Le deuxième contrôleur pour les agences servira l'HTML de l'application web aux clients.

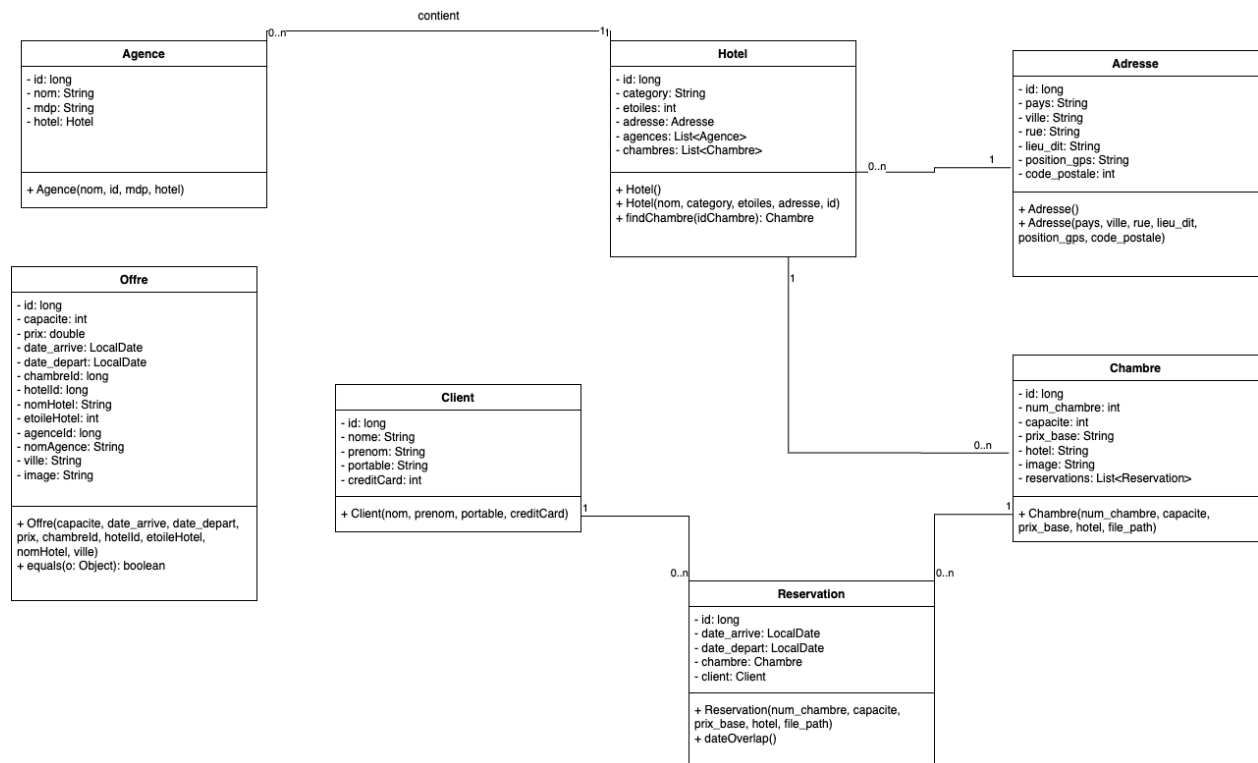
Conception de l'Application

Comme mentionné précédemment, chaque hôtel est envisagé comme un fournisseur de services distinct. Ces services sont accessibles via une API REST, qui permet de réaliser des actions telles que la consultation des disponibilités, la visualisation des détails des chambres, et la gestion des réservations.

Les agences de voyage jouent un rôle crucial en tant que consommateurs des services proposés par les hôtels. Elles demandent des informations et réservent des chambres en nom de leurs clients. Les agences hébergent également un service web qui permettra aux clients d'accéder à une interface où ils pourront faire leurs demandes.

Pour mieux expliquer les détails, examinons les diagrammes UML qui décrivent nos modèles.

UML de HotelService



Classe Hotel

La classe Hotel est au cœur de notre application de réservation. Elle encapsule toutes les informations essentielles d'un hôtel, tels que :

- Identifiant
- Catégorie
- Etoiles
- Adresse
- Agences
- Chambres

Chaque hôtel a une association un-à-un avec la classe Adresse.

La relation un-à-plusieurs (0..n) entre Hotel et Agence montre que les hôtels peuvent être promus par plusieurs agences, permettant ainsi une diversification des canaux de vente et une plus grande visibilité sur le marché.

L'association un-à-plusieurs (0..n) entre Hotel et Chambre illustre que l'hôtel peut offrir plusieurs types de chambres. Chaque chambre est une entité distincte avec sa propre capacité, son prix de base et ses réservations.

Classe Offre

La classe Offre encapsule les détails d'une réservation potentielle, tels que :

- Identifiant
- Capacité : Indique le nombre de personnes que l'offre peut accueillir, une information vitale pour les clients lors de la sélection d'une chambre.
- Prix : Le coût associé à l'offre, qui peut varier en fonction de plusieurs facteurs comme la saison, la demande, et les promotions.
- Dates de Séjour : Définissent la période pendant laquelle la chambre est disponible pour réservation.
- Identifiant de Chambre : Lien direct vers la chambre spécifique à laquelle l'offre se réfère, facilitant la gestion des disponibilités.
- Informations Relatives à l'Hôtel et à l'Agence : Les attributs hotelId, nomHotel, etoileHotel, agenceId, nomAgence, et ville offrent un aperçu complet de l'offre, en associant chaque offre à un hôtel et une agence spécifiques, ainsi qu'à une localisation.
- Visuel de l'Offre / Image : Permet aux clients de voir un aperçu de l'hôtel ou de la chambre avant de réserver.

Classe Client

La classe Client représente l'utilisateur final de l'application. Ses attributs incluent :

- Identifiant
- Informations Personnelles : Des détails cruciaux pour la création d'un profil client et pour les communications.
- Détails de Paiement : Stockés pour faciliter les transactions rapides.

Classe Chambre

La classe Chambre contient des informations sur les chambres individuelles proposées par les hôtels :

- Numéro et Capacité : Permettent aux clients de choisir la taille et le type de chambre appropriés.

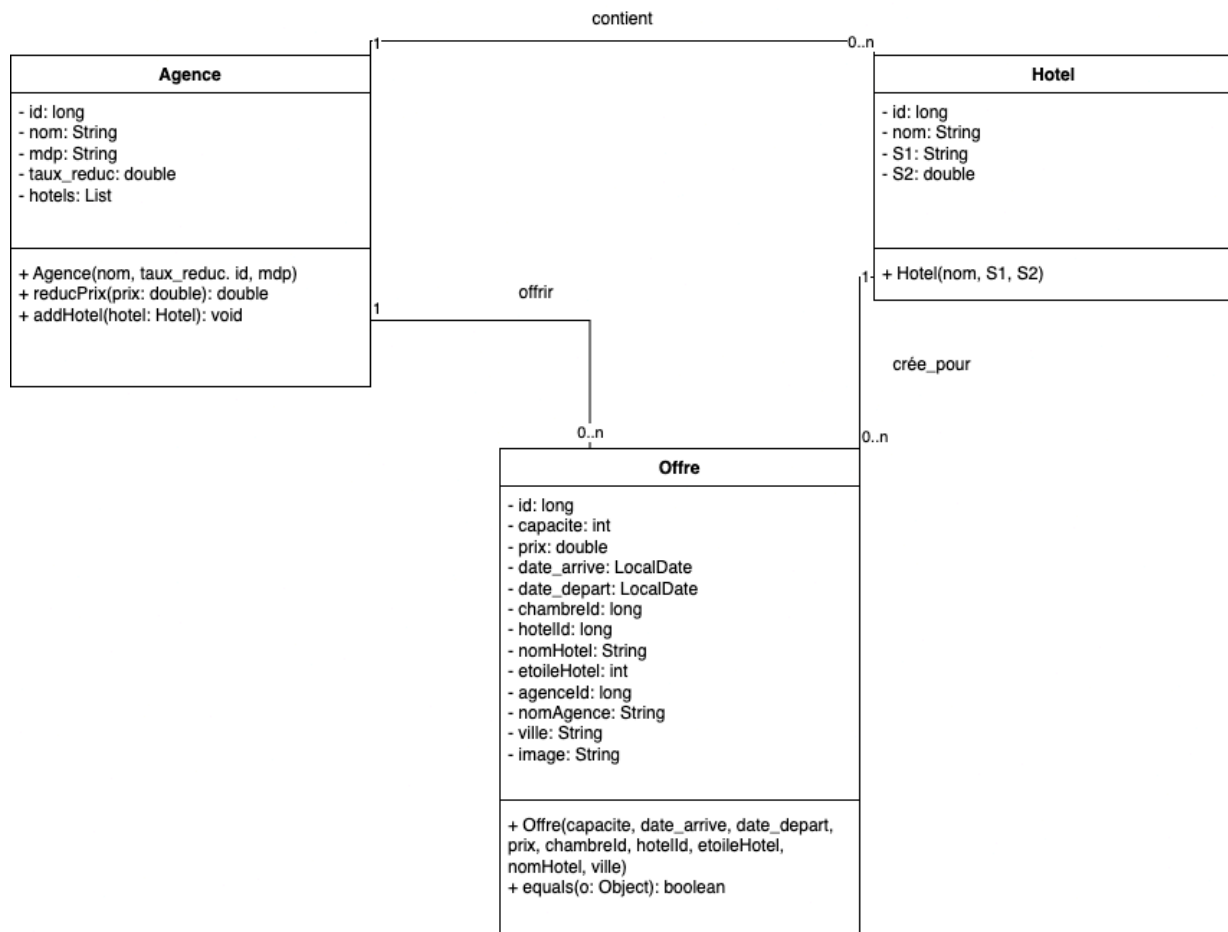
- Prix de Base : Le tarif de départ pour la chambre, avant l'application de réductions ou de suppléments.

Classe Reservation

La classe Reservation est liée à la logique de réservation de l'application :

- Informations sur la Réservation : Les dates d'arrivée et de départ (date_arrive, date_depart), ainsi que la référence à la Chambre et au Client, forment la base d'une réservation.
- Méthode dateOverlap : Cette méthode est utilisée pour vérifier les conflits de réservation, assurant qu'aucune chambre n'est doublement réservée pour la même période.

UML de AgenceService



Classe Agence

- Identifiant
- Nom : Le nom commercial de l'agence, utilisé pour la représentation et la communication avec les clients et les hôtels partenaires.
- Mot de Passe : Un élément de sécurité pour authentifier l'accès aux systèmes de gestion internes de l'agence.
- Taux de Réduction : Un avantage compétitif que l'agence peut offrir, permettant de proposer des réservations à des tarifs préférentiels.

Les méthodes comme `addHotel` et `reducPrix` donnent à l'agence la capacité d'élargir son offre et d'appliquer des stratégies tarifaires dynamiques.

La relation offrir entre Agence et Offre illustre que chaque agence peut proposer plusieurs offres. Cette multiplicité est cruciale pour permettre aux agences de proposer un large éventail d'options à leurs clients.

La relation crée_pour entre Hotel et Offre indique que chaque hôtel peut avoir plusieurs offres créées pour lui.

Classe Offre

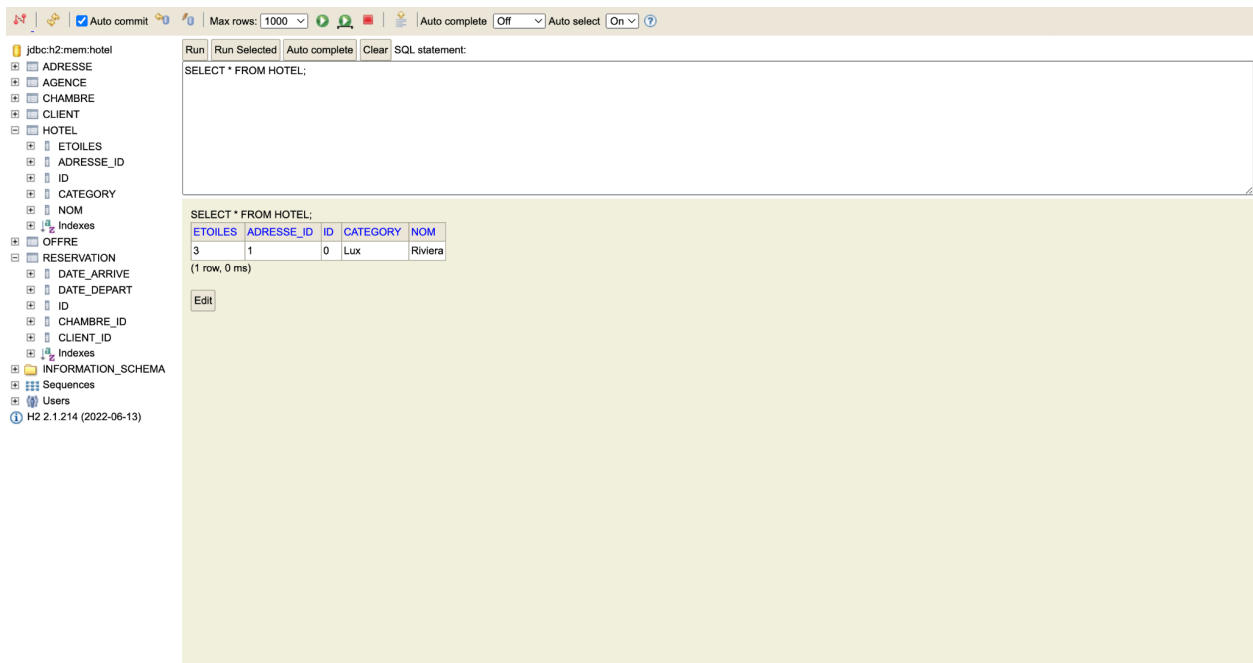
- Identifiant
- Capacité, Prix, et Dates : Ces attributs définissent les caractéristiques essentielles de l'offre telles que la capacité d'accueil, le coût, et la période de séjour.
- Identifiants et Noms Associés : Les attributs comme `hotelId`, `nomHotel`, `etoileHotel`, `agenceId`, et `nomAgence` établissent des liens directs avec les hôtels et agences correspondants, assurant ainsi la traçabilité et la facilitation des réservations.
- Image : Un attribut pour stocker une représentation visuelle de la chambre de l'offre, ce qui est crucial pour attirer et retenir l'attention des clients.

Classe Hotel

Représente les hôtels pour l'agence.

Implémentation de l'Application

L'un des avantages de l'utilisation de Spring est qu'il contient une base de données intégrée (avec la dépendance *h2database*) et un panneau d'administration, ce qui nous permet d'avoir des données persistantes. En utilisant également *thymeleaf*, il a été assez facile d'ajouter une interface web. Ainsi, dès le départ, nous satisfaisions déjà aux exigences de la question bonus (4) qui nécessitait d'avoir une base de données et une interface.



Les Services Exposés de HotelController

GET /disponibilites

Permet aux clients de consulter les chambres disponibles en fonction de leurs besoins.

L'utilisateur doit fournir l'identifiant de l'agence, un mot de passe, les dates de début et de fin de séjour, ainsi que le nombre de personnes. La méthode `getDisponibilites` commence par vérifier les informations de connexion via la méthode `signIn`. Si la vérification échoue, une exception est levée. Ensuite, elle filtre les chambres disponibles basées sur la capacité et crée des offres correspondantes, qui sont sauvegardées dans le dépôt d'offres.

GET /reserver

Permet de réserver une chambre en spécifiant l'identifiant de l'offre, les informations du client et les détails de la carte de crédit. La méthode `reserver` recherche d'abord l'offre spécifiée. Si elle est trouvée, elle procède à la création ou à la récupération du client, puis crée une réservation. Cette réservation est ensuite enregistrée et l'offre est supprimée du système.

GET /Chambre/{id}

Permet de récupérer les détails d'une chambre spécifique en utilisant son identifiant. La méthode `getChambreById` récupère les informations de la chambre depuis le dépôt de chambres.

Les Services Exposés de AgenceController

GET /api/comparable

Permet de comparer les offres d'hôtels en fonction de critères spécifiques tels que la période de séjour, le nombre de personnes, la ville et le classement par étoiles de l'hôtel. La méthode comparable commence par invoquer toutes les offres disponibles via invokeOffres. Puis, elle filtre ces offres en fonction des critères fournis, applique une réduction sur les prix grâce à la méthode reducPrix de l'agence, et les enregistre.

Méthode Auxiliaire invokeOffres

Cette méthode récupère les offres des hôtels partenaires pour une période donnée et un nombre spécifique de personnes. Elle parcourt tous les hôtels disponibles, construit une URL pour interroger chaque service d'hôtel, et récupère les offres en utilisant RestTemplate. Les offres récupérées sont ensuite traitées et ajoutées à la liste d'offres à retourner.

GET /api/reserver

Permet de procéder à la réservation d'une offre spécifique. Il récupère l'offre et les informations de l'hôtel concerné et envoie une requête de réservation à l'hôtel via une URL construite. La réponse obtenue (probablement l'identifiant de la réservation) est ensuite retournée au client.

Les Services Exposés de AgenceWebController

GET /Accueil

Affiche la page d'accueil où l'utilisateur peut saisir ses préférences pour la recherche d'hôtels.

POST /Accueil

Traite les informations fournies par l'utilisateur (dates, étoiles, nombre de personnes, ville) et redirige vers la page d'affichage des offres si les données sont valides.

GET /AfficherOffres

Affiche la page avec les offres correspondant aux critères de l'utilisateur.

GET /AfficherOffre

Permet de sélectionner une offre pour la réservation et redirige vers la page de réservation.

GET /Reservation

Affiche la page de réservation où l'utilisateur peut entrer ses informations personnelles.

POST /Reservation

Traite les informations du client et effectue la réservation via la méthode réserver. Redirige vers une page de validation ou d'erreur selon le résultat.

GET /Valider

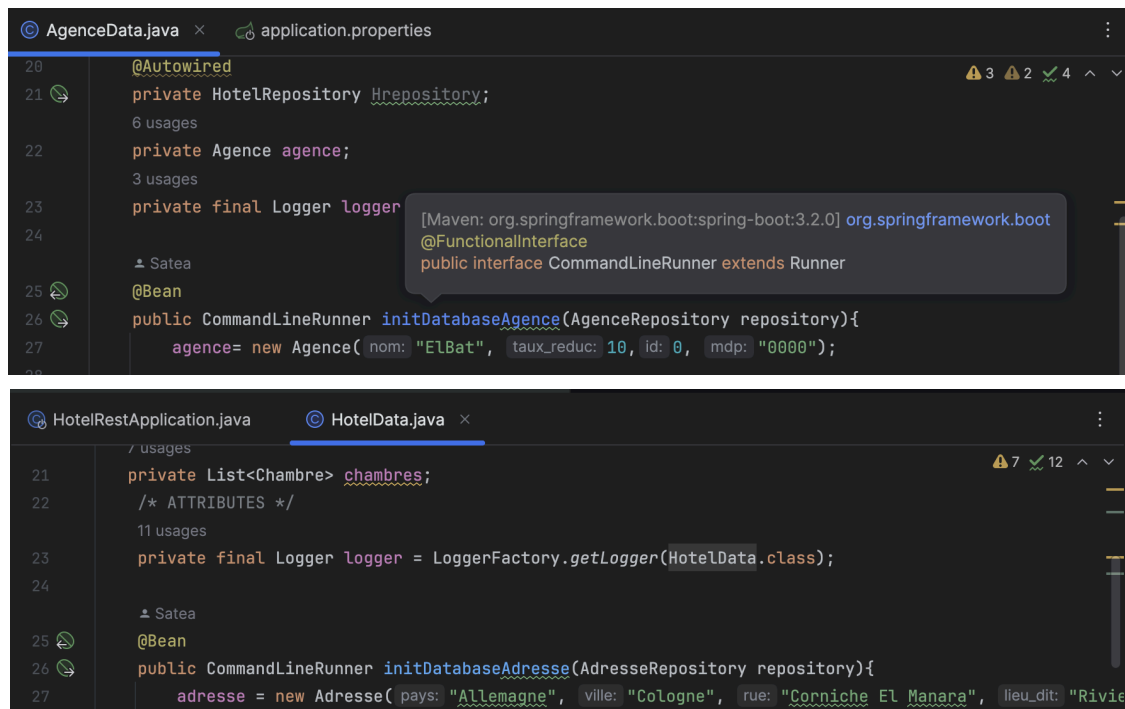
Affiche une page de confirmation après une réservation réussie.

GET /Problem

Affiche une page d'erreur en cas de problème dans le processus de réservation.

Exemple : Réservation d'Une Chambre

Nous allons maintenant illustrer comment marche l'application web des agences avec un exemple. Dans cet exemple, nous avons une agence appelée ElBat et quelques hôtels dans la ville allemande de Cologne. Voici notre fichier AgenceData où nous ajoutons ces données :



```
AgenceData.java
20 @Autowired
21 private HotelRepository hrepository;
22 private Agence agence;
23 private final Logger logger
24
25 @Bean
26 public CommandLineRunner initDatabaseAgence(AgenceRepository repository){
27     agence= new Agence( nom: "ElBat", taux_reduc: 10, id: 0, mdp: "0000");
28
HotelRestApplication.java
21 private List<Chambre> chambres;
22 /* ATTRIBUTES */
23 private final Logger logger = LoggerFactory.getLogger(HotelData.class);
24
25 @Bean
26 public CommandLineRunner initDatabaseAdresse(AdresseRepository repository){
27     adresse = new Adresse( pays: "Allemagne", ville: "Cologne", rue: "Corniche El Manara", lieu_dit: "Rivie
HotelData.java
21 private List<Chambre> chambres;
22 /* ATTRIBUTES */
23 private final Logger logger = LoggerFactory.getLogger(HotelData.class);
24
25 @Bean
26 public CommandLineRunner initDatabaseAdresse(AdresseRepository repository){
27     adresse = new Adresse( pays: "Allemagne", ville: "Cologne", rue: "Corniche El Manara", lieu_dit: "Rivie
```

Nous pouvons maintenant aller sur le site web de cette agence (GET /Accueil).

Bienvenue à ElBat!

Ville:

Cologne

Etoile minimum:

1

Nombre de personnes:

2

Date de début:








14/12/2023

Date de fin:

15/12/2023

Rechercher

Après avoir rempli les paramètres (POST /Accueil), explorons les offres (GET /AfficherOffres) et en choisissons une (GET /Reservation).

Hotel	etoile	Capacite	Prix	Date de debut	Date de fin	Image	Action
4 Seasons	5	4	900.0	2023-12-14	2023-12-15		<div>Reserver</div>
4 Seasons	5	4	900.0	2023-12-14	2023-12-15		<div>Reserver</div>
4 Seasons	5	2	180.0	2023-12-14	2023-12-15		<div>Reserver</div>
4 Seasons	5	2	180.0	2023-12-14	2023-12-15		<div>Reserver</div>
Riviera	3	4	1350.0	2023-12-14	2023-12-15		<div>Reserver</div>
Riviera	3	4	1350.0	2023-12-14	2023-12-15		<div>Reserver</div>
Riviera	3	2	450.0	2023-12-14	2023-12-15		<div>Reserver</div>

Nous complétons le formulaire de réservation et finalement, si tout va bien, nous recevons l'identifiant correspondant.

Formulaire de Réservation

Nom:

Prénom:

Numéro de Portable:

Numéro de Carte Bancaire:

Soumettre

Votre reservation a été prise en compte, votre ID= 1!

Faire une autre reservation

Nous pouvons également voir cette réservation dans la base de données de l'hôtel correspondant.

jdbc:h2:mem:hotel

ADRESSE

AGENCE

CHAMBRE

CLIENT

HOTEL

OFFRE

RESERVATION

DATE_ARRIVE

DATE_DEPART

ID

CHAMBRE_ID

CLIENT_ID

Indexes

INFORMATION_SCHEMA

Sequences

Users

H2 2.1.214 (2022-06-13)

Run

Run Selected

Auto complete

Clear

SQL statement:

SELECT * FROM RESERVATION;

SELECT * FROM RESERVATION;

DATE_ARRIVE	DATE_DEPART	ID	CHAMBRE_ID	CLIENT_ID
2023-12-14	2023-12-15	1	3	2

(1 row, 4 ms)

Edit

La prochaine fois qu'on veut faire une réservation, nous ne verrons plus cette chambre dans les mêmes dates comme elle n'est plus disponible.

```

public static boolean dateValide(Chambre c, LocalDate dep, LocalDate arv) {
    for (Reservation r: c.getReservations()) {
        if(r.dateOverlap(dep, arv)) {
            return false;
        }
    }
    return true;
}

```

Comparateur (Trivago)

Dans cette partie, nous essayons de construire un agrégateur qui obtient les prix de plusieurs agences et renvoie les résultats qu'il obtient tout en filtrant selon différentes propriétés.

En principe, le comparateur est comme une méta-agence, car au lieu de faire directement des requêtes aux hôtels, il fait ces requêtes aux différentes agences qui vont faire leurs propres requêtes aux hôtels. Nous pouvons donc justement réutiliser le même code que AgenceWebController. C'est assez de modifier la méthode comparable pour pouvoir agréger toutes les réponses des agences et les retourner.

Examinons l'ancien code où nous faisons nos requêtes un par un aux hôtels pour récupérer les offres :

```

public List<Offre> comparable(LocalDate dateDebut, LocalDate dateFin, int
nombrePersonnes, String ville, int etoile) throws JsonProcessingException {
    List <Offre> offres = invokeOffres(dateDebut, dateFin, nombrePersonnes);
    List <Offre> offreFiltre = new ArrayList<>();
    for (Offre offre: offres){

if(offre.getVille().equalsIgnoreCase(ville) && offre.getEtoileHotel() >= etoile){
        offreFiltre.add(offre);

offre.setPrix(Math.round(agence.reducPrix(offre.getPrix())*100.0)/100.0);
    }
    }
    Orepository.saveAll(offreFiltre);
    return offreFiltre; //control the info sent OR not
}

public List<Offre> invokeOffres(LocalDate dateDebut, LocalDate dateFin, int
nbrPersonne) throws JsonProcessingException {
    agence = Orepository.getReferenceById(0L);
    List<Hotel> hotels = Hrepository.findAll();
    List<Offre> offres = new ArrayList<>();
    for(Hotel hotel: hotels) {

```

```

        String url = hotel.getS1() + "id=" + agence.getId() + "&mdp=" +
agence.getMdp() + "&dateDebut=" + dateDebut.toString() + "&dateFin=" +
dateFin.toString() + "&nombrePersonnes=" + nbrPersonne;
        String response = proxy.getForObject(url, String.class);
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.registerModule(new JavaTimeModule()); //Pour configurer
l'objetMapprer pour qu'il utilise le module qui prend en charge le localdate
        try {
            List<Offre> offresTemp = objectMapper.readValue(response, new
TypeReference<List<Offre>>() {});
            for(Offre offre: offresTemp){
                offre.setHotelId(hotel.getId());
            }
            offres.addAll(offresTemp);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return offres;
}

```

Regardons maintenant le code modifié où nous contactons chaque agence :

```

public List<Offre> comparable(LocalDate dateDebut, LocalDate dateFin, int
nombrePersonnes, String ville, int etoile) throws JsonProcessingException {
    List <Agence> agences= Arepository.findAll();
    List<Offre> offres= new ArrayList<>();
    for(Agence agence: agences) {
        String url = agence.getS1() + "&dateDebut=" + dateDebut.toString() +
"&dateFin=" + dateFin.toString() + "&nombrePersonnes=" + nombrePersonnes+
"&ville="+ ville+"&etoile="+etoile;
        String response = proxy.getForObject(url, String.class);
        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.registerModule(new JavaTimeModule()); //Pour configurer
l'objetMapprer pour qu'il utilise le module qui prend en charge le localdate
        try {
            offres.addAll(objectMapper.readValue(response, new
TypeReference<List<Offre>>() {}));
        } catch (Exception e) {
            e.printStackTrace();
        }
        for (Offre offre: offres){
            offre.setAgenceId(agence.getId());
        }
    }
}

```

```
Orepository.saveAll(offres);  
return offres; //control the info sent OR not  
}
```

Conclusion

Nous avons utilisé le framework Spring pour construire 3 applications différentes utilisant des API REST qui permettent aux clients d'une agence ou d'un service comparateur de chercher des chambres et faire des réservations.