

Rapport SOAP

Version Exhaustive (les Quatre Dépôts)



Version Centralisée

Organisation de la solution :

Pour réaliser l'objectif de ce TP SOAP qui est la création d'une application de réservation d'hôtel en ligne, j'ai créé **quatre classes** et **une classe Main** qui sert à exécuter le programme.

1. Classe Hôtel :

Dans cette classe, j'ai déclaré des attributs pour décrire un hôtel (comme son nom, son adresse, son nombre d'étoiles), conformément au diagramme UML. J'ai ensuite créé un constructeur et les accesseurs (getters et setters) pour ces attributs.

J'ai ajouté par la suite des méthodes importantes :

- **Méthode ajoutChambre :**

Cette méthode sert à ajouter des chambres à l'hôtel. Elle initialise une liste de chambres disponibles pour chaque hôtel.

- **Méthode rechercheDisponibilites :**

Cette méthode permet de rechercher les chambres disponibles dans un hôtel en fonction des critères spécifiés par l'utilisateur (comme la date d'arrivée, la date de départ, le nombre de personnes, et le prix maximum).

Elle parcourt la liste des chambres de l'hôtel et retourne uniquement celles qui répondent à ces critères.

Classe Chambre :

Dans cette classe, j'ai déclaré des attributs pour décrire une chambre (comme son type, son numéro, son prix, sa capacité...). J'ai ensuite créé un constructeur ainsi que des getters et setters (accesseurs) pour manipuler ces attributs.

Ensuite, j'ai ajouté une :

Méthode ajoutReservation :

Cette méthode permet d'ajouter une réservation à la liste des réservations pour cette chambre.

Elle stocke les réservations associées à cette chambre dans une liste, ce qui permet de vérifier les disponibilités de la chambre pour des dates données.

Méthode disponibleSurPeriode :

Cette méthode vérifie si la chambre est disponible pendant une période spécifique (définie par une date d'arrivée et une date de départ).

Elle parcourt la liste des réservations existantes de la chambre et vérifie si les dates demandées se chevauchent avec les dates des réservations existantes.

Pour la classe Réservation, j'ai uniquement déclaré les différents attributs correspondant aux informations nécessaires pour décrire une réservation, comme le numéro de réservation, les dates, le prix, et les relations avec l'hôtel, la chambre, et le client.

Concernant la méthode **effectueReservation** :

Cette méthode vérifie si la chambre sélectionnée est vide pour une période précise. Si c'est le cas, cela signifie que la chambre est disponible.

La classe Client : j'ai créé une structure simple pour représenter un client. J'ai déclaré les attributs (numCin, nom, prénom, email et le numéro de téléphone). J'ai également conservé la même structure cohérente pour la création du constructeur et des accesseurs (getters et setters), afin de permettre une gestion organisée et uniforme des données.

Pour la classe Main :

1- La création des instances des objets :

J'ai créé une instance de la classe Hôtel pour représenter l'hôtel dans le système, en fournissant les mêmes informations passées en paramètres dans la classe Hôtel initiale.

J'ai également créé quatre instances de la classe Chambre pour décrire les chambres disponibles dans cet hôtel, chacune avec ses paramètres.

2- Ajout des chambres à l'hôtel :

J'ai utilisé la méthode **ajoutChambre()** pour associer chaque chambre créée à l'hôtel concerné. Cela permet d'enregistrer les chambres dans une liste qui va être utilisée plus tard pour la recherche de disponibilité.

3- Recherche des chambres disponibles :

J'ai défini des critères de recherche pour simuler une recherche d'hôtel. (comme la date d'arrivée, la date de départ, le nombre de personnes, et le prix maximum que le client ne compte pas dépasser.

Grâce à la méthode **rechercheDisponibilites()** de la classe Hôtel, seules les chambres répondant aux critères sont affichées. Pour cela j'ai opté pour la méthode isEmpty dans la

condition **if chambresDisponibles.isEmpty** qui vérifie si la liste des chambres disponibles est vide. Une fois la liste est vide cela signifie qu'il n'y a aucune chambre correspondant aux critères de recherche de l'utilisateur.

4. Interaction avec l'utilisateur via le Scanner :

Une fois la liste des chambres disponibles sera affichée, l'utilisateur pourra effectuer son choix par toute flexibilité pour cela j'ai intégré l'objet Scanner pour permettre à l'utilisateur de faire son choix parmi les résultats affichés. Cette interaction rend le programme dynamique et engageant.

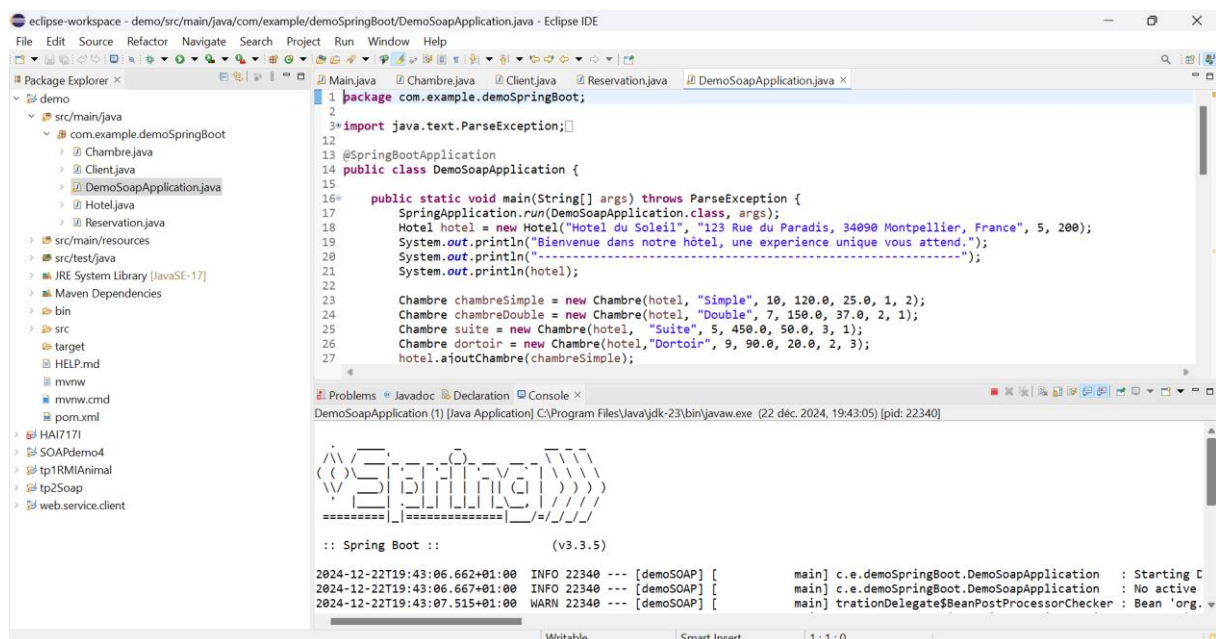
- L'utilisateur peut entrer le numéro correspondant à la chambre qu'il souhaite réserver.
- fois que l'utilisateur a choisi une chambre, j'ai ajouté une étape pour lui demander ses coordonnées bancaires. Une fois les coordonnées sont fournies, une vérification doit être établie pour garantir la validité de ces coordonnées (pas nulles ou vides) avant de finaliser la réservation.

5. Validation et confirmation de la réservation :

- Si toutes les conditions sont réunies, la réservation est confirmée, et un message avec le numéro de réservation sera affiché.

La version centralisée avec SPRING BOOT

Dans cette phase du projet, j'ai intégré une version simplifiée du service Web à Spring Boot, en commençant par créer une application de base. Cette tentative a permis d'afficher la sortie dans la console, comme illustré ci-dessus. Ce test initial m'a permis de mieux comprendre les flux de travail avec Spring Boot, tout en offrant une perspective sur la facilité d'extension et la gestion des erreurs du serveur, ce qui était bénéfique pour les autres étapes du projet.



```
package com.example.demoSpringBoot;

import java.text.ParseException;

@SpringBootApplication
public class DemoSoapApplication {

    public static void main(String[] args) throws ParseException {
        SpringApplication.run(DemoSoapApplication.class, args);
        Hotel hotel = new Hotel("Hotel du Soleil", "123 Rue du Paradis, 34090 Montpellier, France", 5, 200);
        System.out.println("Bienvenue dans notre hôtel, une experience unique vous attend.");
        System.out.println("-----");
        System.out.println(hotel);

        Chambre chambreSimple = new Chambre(hotel, "Simple", 10, 120.0, 25.0, 1, 2);
        Chambre chambreDouble = new Chambre(hotel, "Double", 7, 150.0, 37.0, 2, 1);
        Chambre suite = new Chambre(hotel, "Suite", 5, 450.0, 50.0, 3, 1);
        Chambre dortoir = new Chambre(hotel, "Dortoir", 9, 90.0, 20.0, 2, 3);
        hotel.ajoutChambre(chambreSimple);
    }
}
```

```

:: Spring Boot ::      (v3.3.5)

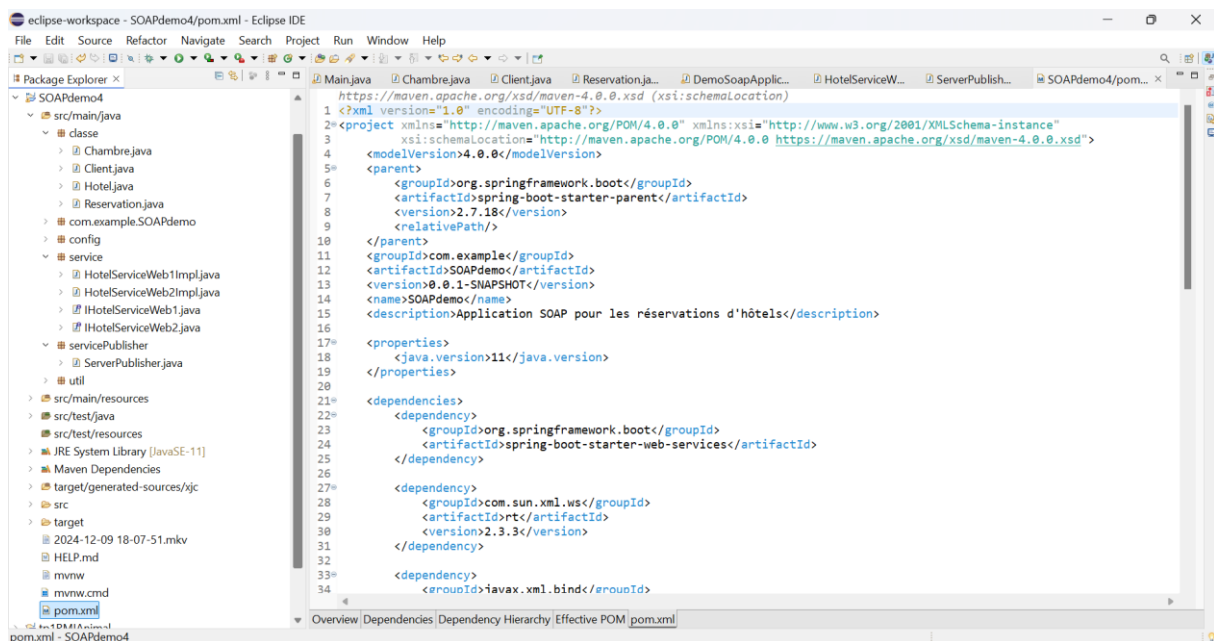
2024-12-22T19:43:06.662+01:00 INFO 22340 --- [demoSOAP] [
2024-12-22T19:43:06.667+01:00 INFO 22340 --- [demoSOAP] [
2024-12-22T19:43:07.515+01:00 WARN 22340 --- [demoSOAP] [
main] c.e.demoSpringBoot.DemoSoapApplication : Starting [
main] c.e.demoSpringBoot.DemoSoapApplication : No active
main] trationDelegate$BeanPostProcessorChecker : Bean 'org.
```

La version distribuée du projet avec deux services Web

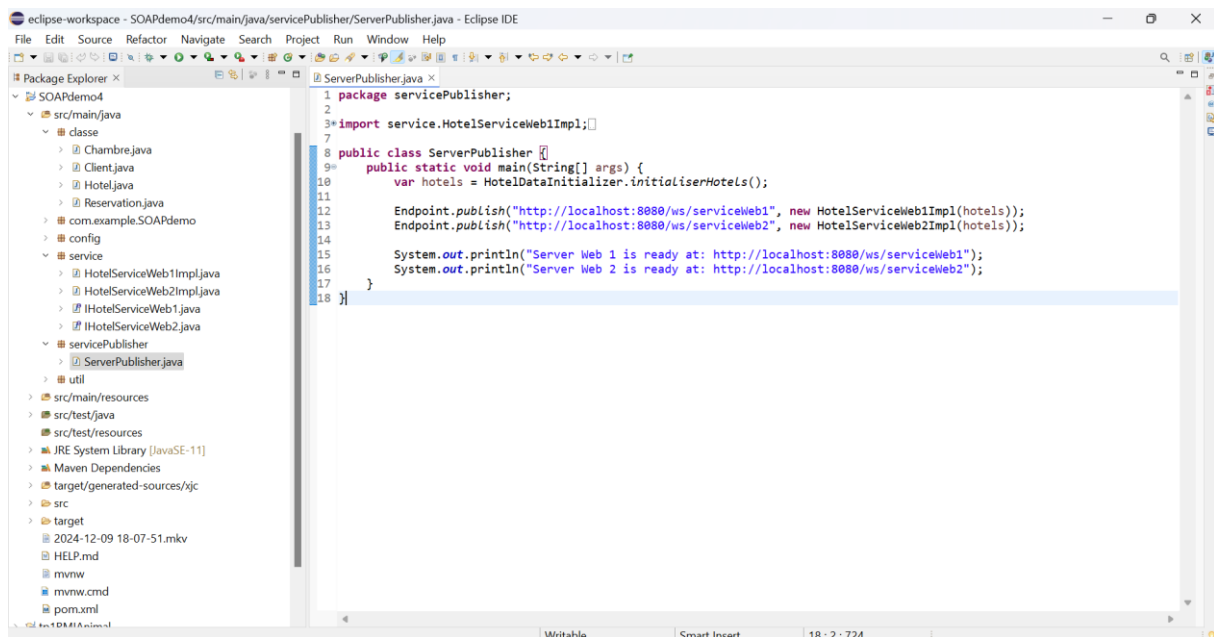
Partie Serveur

Pour la **version distribuée**, j'ai utilisé les mêmes classes que dans ma version centralisée, mais j'ai ajouté deux interfaces et leurs implémentations pour mettre en place un serveur avec deux services Web. L'idée était de créer deux services distincts : un pour la méthode **rechercheDisponibilité** et un autre pour **effectueReservation**. Chaque service disposait de son propre fichier d'interface et d'implémentation, ce qui permettait de bien séparer les responsabilités entre les différents composants.

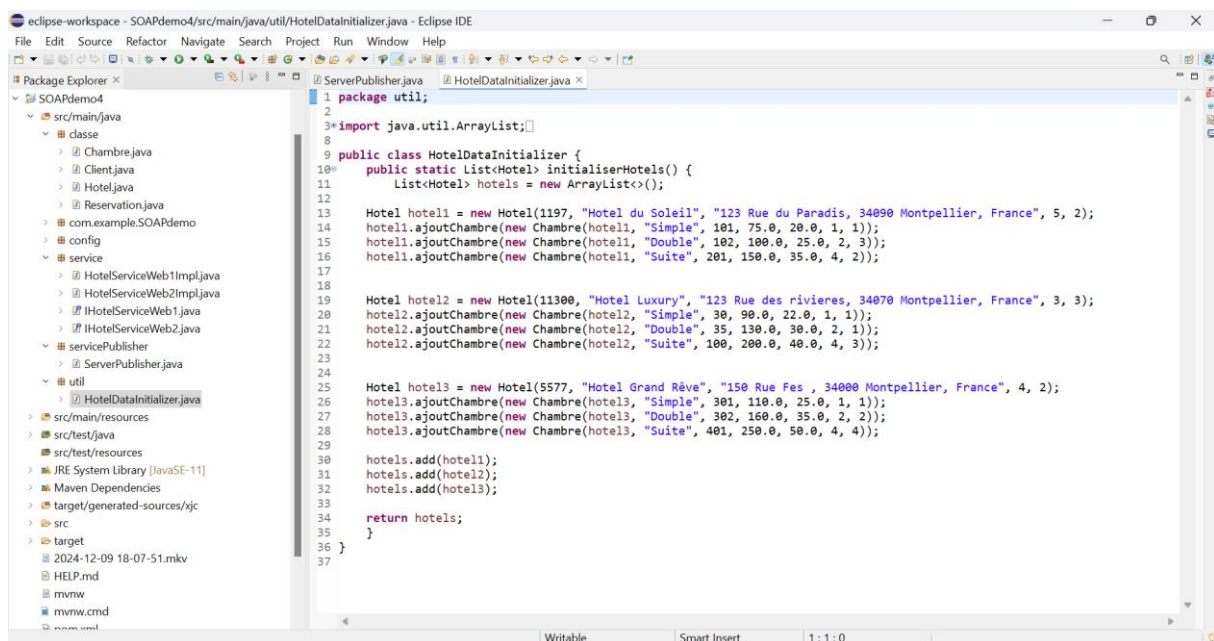
J'ai utilisé les annotations comme **@WebService**, **@WebMethod**, et **@Override** pour définir les services et les méthodes accessibles via SOAP. En outre, j'ai ajouté les dépendances nécessaires à l'intégration du serveur avec les services Web.



Pour la classe **ServerPublisher**, j'ai déterminé les **endpoints** pour chaque service Web, permettant ainsi d'accéder au serveur et au WSDL associé. Ma classe ServerPublisher ressemble à ceci :



Comme on peut le voir sur la photo, j'ai créé une classe **HotelDataInitializer** où j'initialise les instances de mes objets, ce qui joue le rôle de méthode main

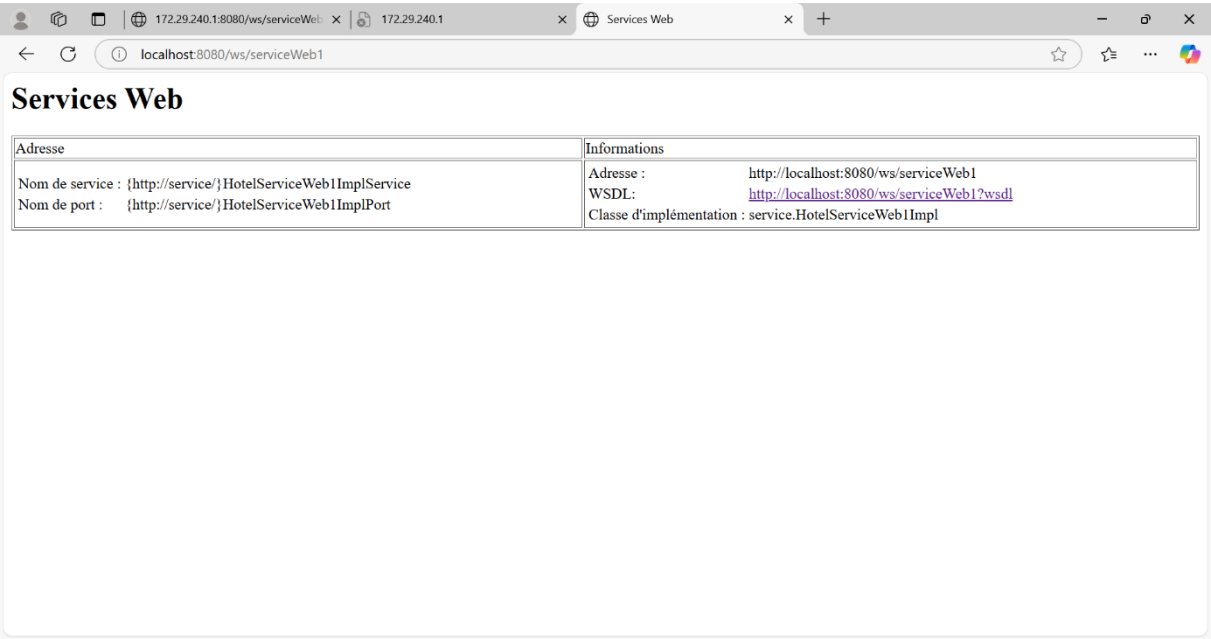
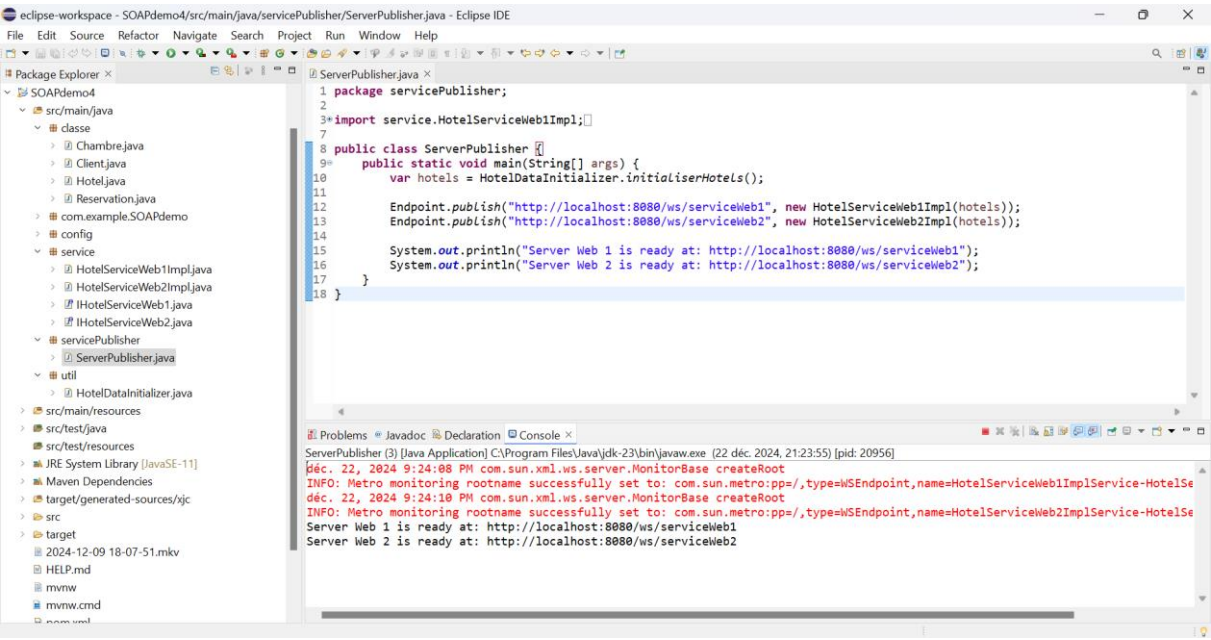


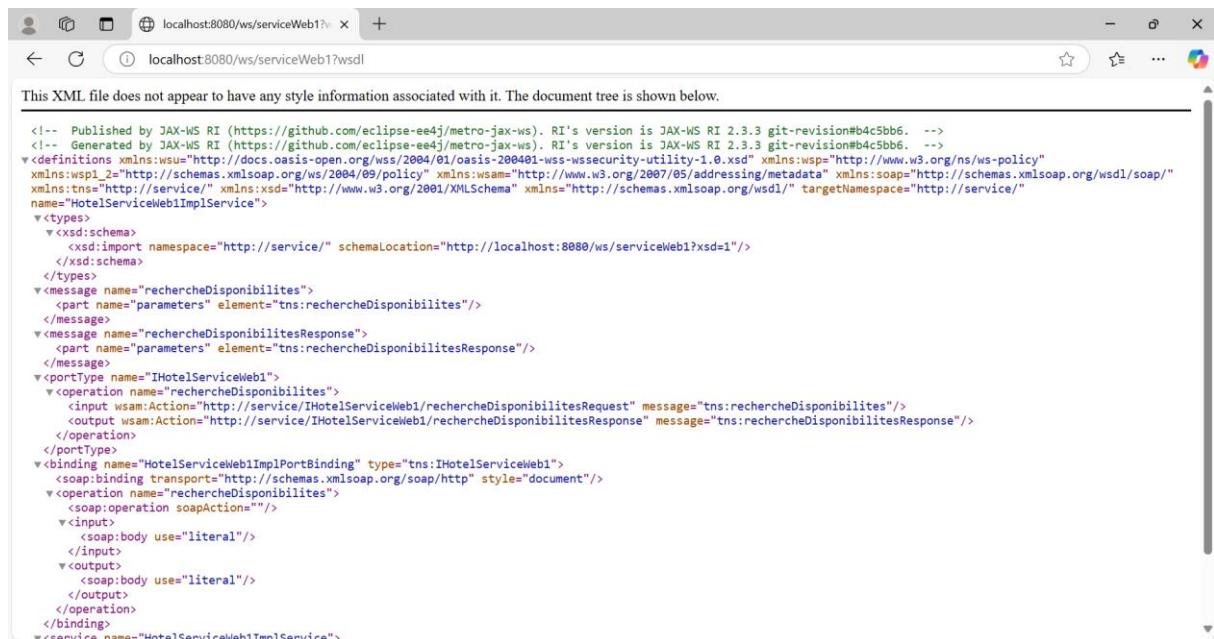
J'ai lancé le serveur en cliquant sur le bouton **Run** dans Eclipse. Une fois le serveur démarré, j'ai pu accéder à l'interface WSDL des services via le navigateur en utilisant les liens suivants :

[Services Web](http://localhost:8080/ws/serviceWeb1) - <http://localhost:8080/ws/serviceWeb1>

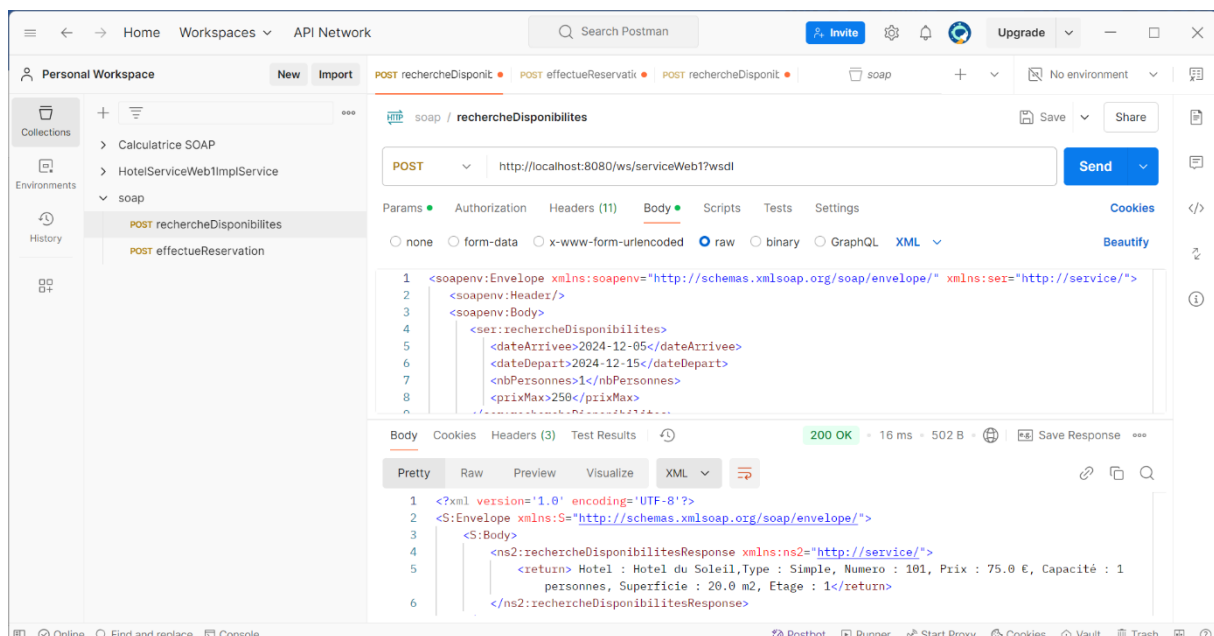
[Services Web](http://localhost:8080/ws/serviceWeb2) - <http://localhost:8080/ws/serviceWeb2>

Cela m'a permis de vérifier que les services étaient bien fonctionnels comme illustré ci-dessous.

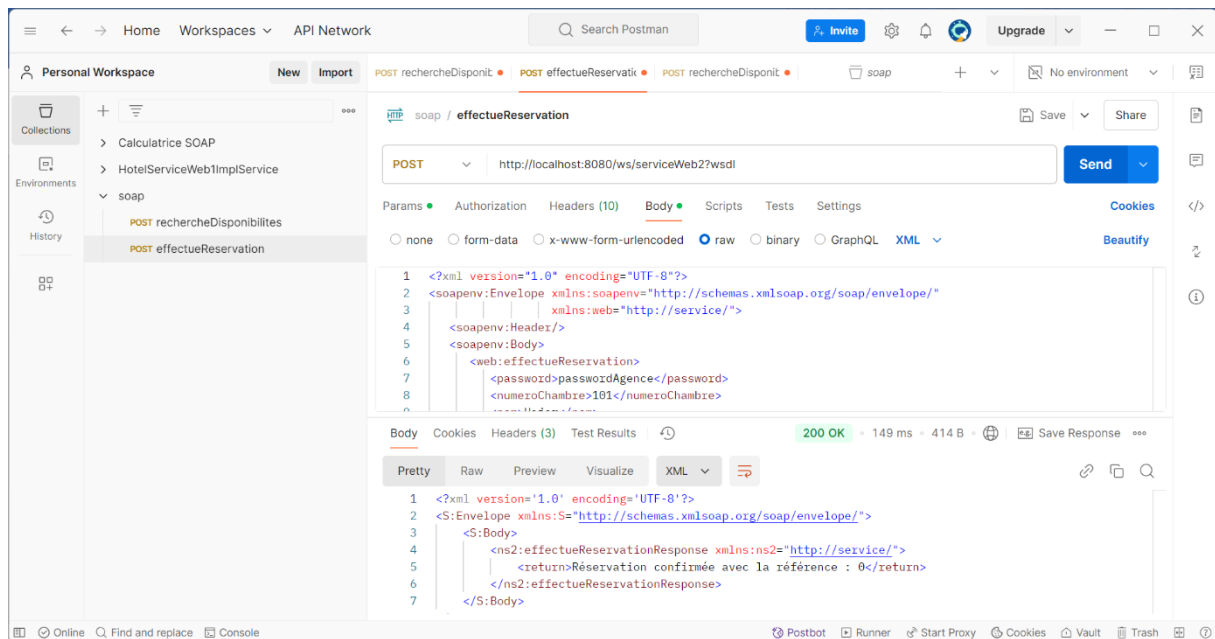




Cela m'a permis également de tester les services dans un environnement local, en interagissant avec eux via des requêtes SOAP à partir de la **console** et **Postman**.

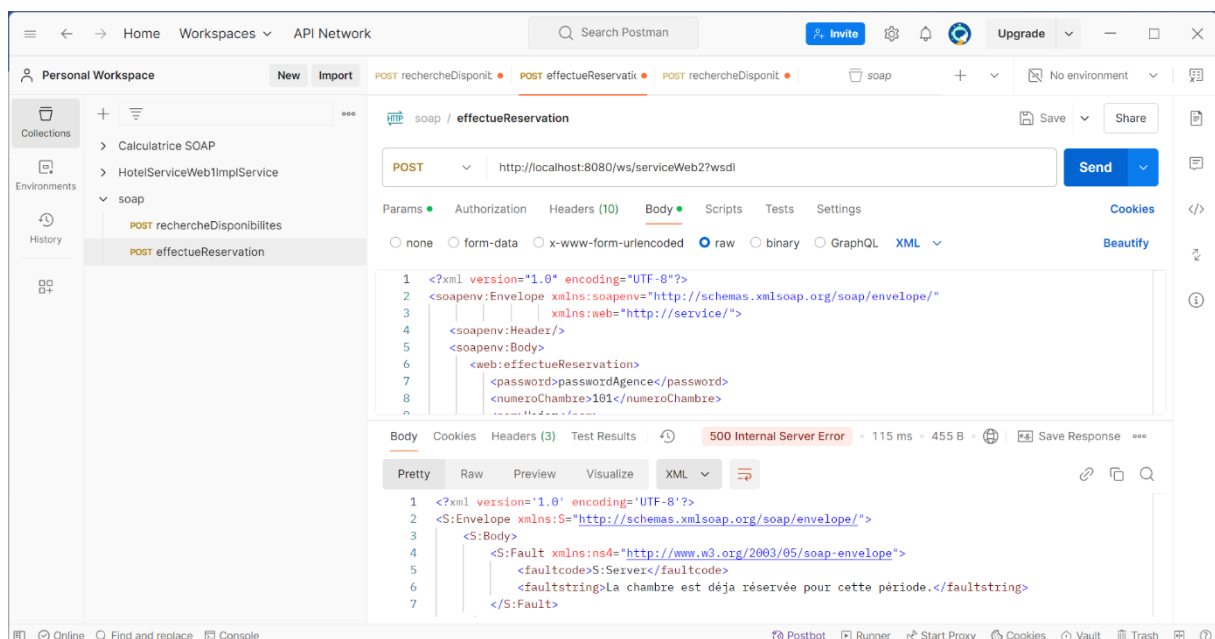


J'ai testé la méthode **rechercheDisponibilites** avec Postman en envoyant une requête SOAP au serveur. Comme on peut le voir dans la capture d'écran ci-dessus, la requête a été correctement envoyée avec les paramètres nécessaires. La réponse du serveur a été reçue, confirmant que le service fonctionne comme prévu. (200OK)

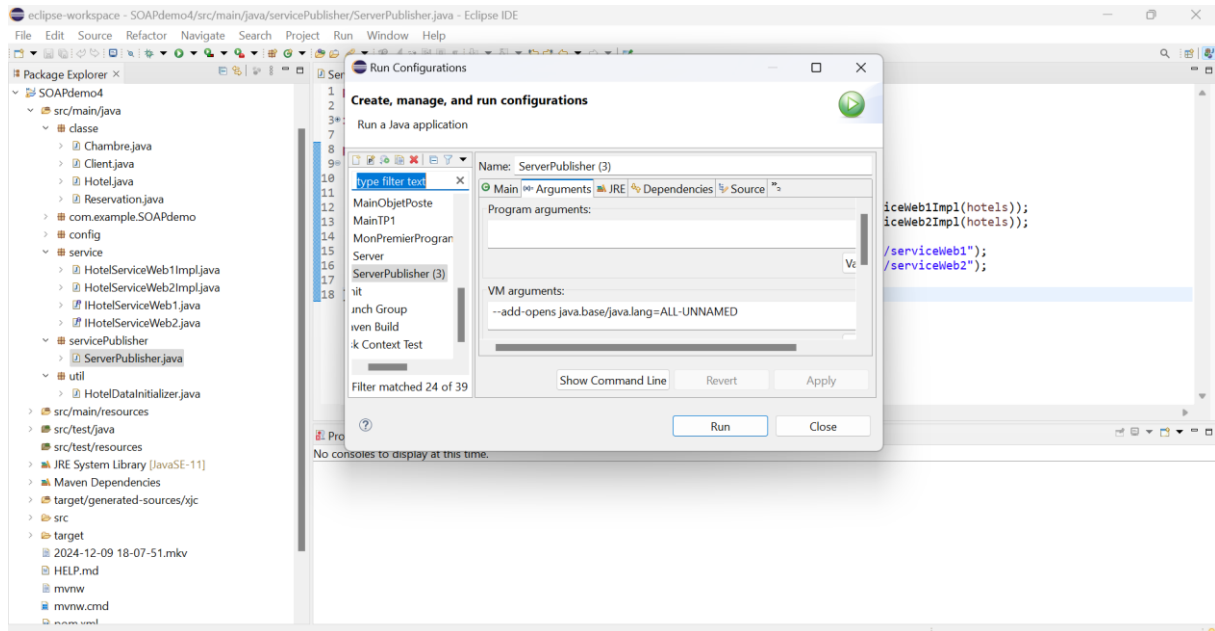


Pour la méthode **effectueReservation**, j'ai envoyé une requête SOAP via Postman et j'ai bien reçu la réponse de confirmation avec la référence de la réservation.

Ensuite, j'ai essayé de réserver la même chambre pour la même période. Le serveur a correctement renvoyé un message d'erreur indiquant que la chambre était déjà réservée, ce qui montre que le service gère bien les réservations multiples et les conflits de disponibilité.



A noter : J'ai rencontré des erreurs liées à des problèmes de compatibilité avec Java et des configurations de sécurité dans l'environnement d'exécution. C'est j'ai utilisé une commande spécifique pour résoudre ces erreurs : **--add-opens java.base/java.lang=ALL-UNNAMED**. Cette commande a permis de lever les restrictions liées à l'accès aux packages et a corrigé les problèmes de compilation et d'exécution.



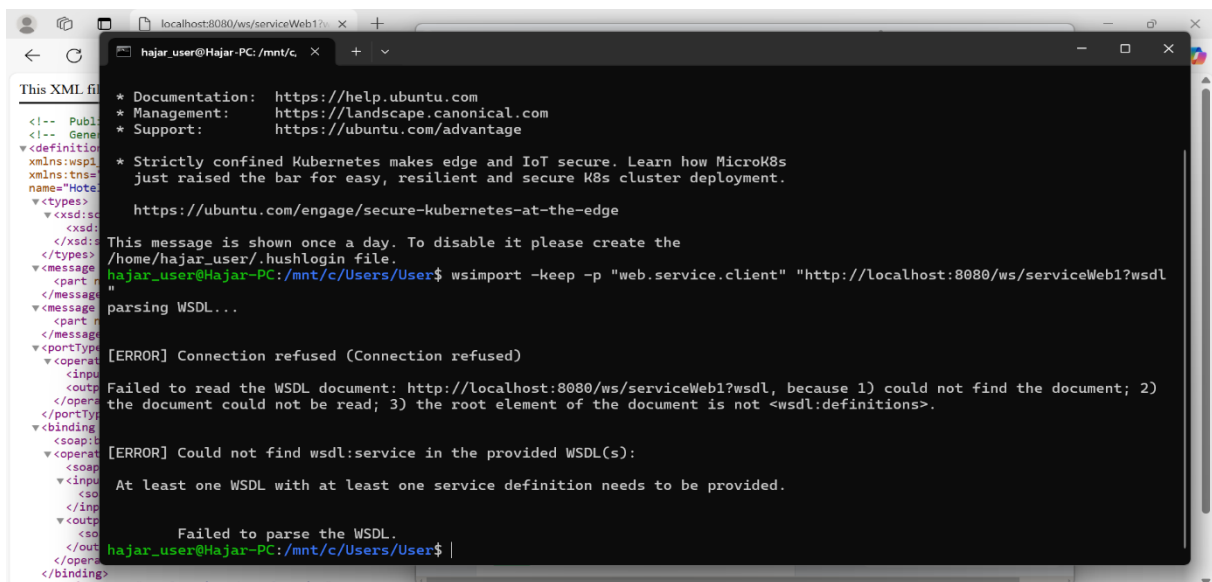
La Version Distribuée - Partie Client

Dans cette phase du projet, j'ai créé un nouveau package pour **le client**. L'objectif était de générer le code nécessaire pour interagir avec les services Web. Pour cela, j'ai utilisé la commande suivant dans le terminal :

```
wsimport -keep -p "web.service.client" "http://localhost:8080/ws/serviceWeb1?wsdl"
```

Cela a permis de compiler et générer les classes nécessaires. Cependant, j'ai rencontré un problème avec l'utilisation de localhost dans la commande, comme l'avait montré **Rima Bachar** dans sa vidéo. Après quelques recherches, j'ai découvert que l'on pouvait utiliser **l'adresse IP au lieu de localhost**, et cette méthode a fonctionné correctement pour moi.

Photo 1 : Cette photo montre la commande avec **localhost** et l'erreur que nous avons rencontrée lors de la dernière séance. La commande `wsimport -keep -p` a échoué en raison de l'impossibilité d'accéder au WSDL via localhost.



Lien vidéo YouTube : <https://youtu.be/IZTarWzFer4?si=FWCRnjAbLUVKOWKr>

Photo 2 : Cette photo montre le succès de la commande avec l'adresse IP de mon ordinateur, ce qui a permis de contourner l'erreur précédente. La compilation a été réalisée avec succès et le code a été généré correctement. Cela a permis de créer les classes nécessaires pour interagir avec le service Web

```
wsimport -keep -p "web.service.client" "http://172.29.240.1:8080/ws/serviceWeb1?wsdl"
```

```
Invite de commandes x hajar_user@Hajar-PC: /mnt/c/ x hajar_user@Hajar-PC: /mnt/c/ + v

Generating code...

Compiling code...

hajar_user@Hajar-PC: /mnt/c/Users/User/desktop/temp_client$
hajar_user@Hajar-PC: /mnt/c/Users/User/desktop/temp_client$ wsimport -keep -p "web.service.client" "http://172.29.240.1:8080/ws/serviceWeb2?wsdl"
parsing WSDL...

Generating code...

Compiling code...

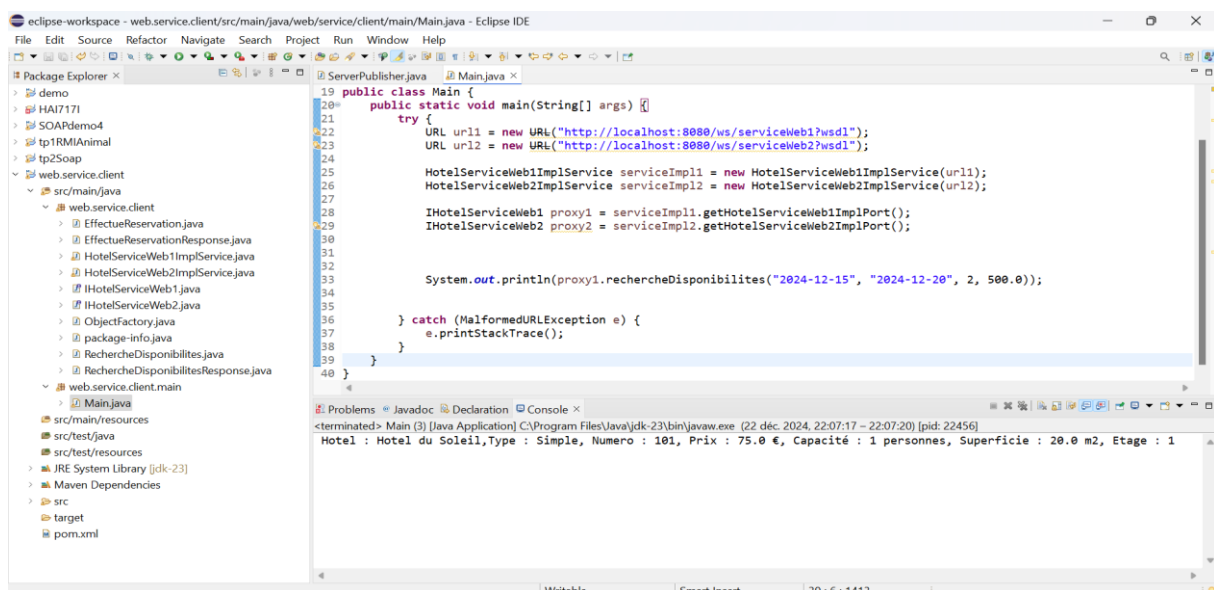
hajar_user@Hajar-PC: /mnt/c/Users/User/desktop/temp_client$ wsimport -keep -p "web.service.client" "http://172.29.240.1:8080/ws/serviceWeb1?wsdl"
parsing WSDL...

Generating code...

Compiling code...

hajar_user@Hajar-PC: /mnt/c/Users/User/desktop/temp_client$
```

Une fois la commande exécutée avec succès, j'ai généré les fichiers .java et .class. En suivant les étapes de la vidéo, j'ai copié les fichiers .java dans mon projet. Par la suite, j'ai créé une classe Main pour instancier les objets et le proxy nécessaire à l'interaction avec les services Web.



```
eclipse-workspace - web.service.client/src/main/java/web/service/client/main/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer x
demo
HAI7171
SOAPdemo4
tp1RMAnimal
tp2Soap
web.service.client
  src/main/java
    web.service.client
      EffectueReservation.java
      EffectueReservationResponse.java
      HotelServiceWeb1ImplService.java
      HotelServiceWeb2ImplService.java
      IHotelServiceWeb1.java
      IHotelServiceWeb2.java
      ObjectFactory.java
      package-info.java
      RechercheDisponibilites.java
      RechercheDisponibilitesResponse.java
    web.service.client.main
      Main.java
src/main/resources
src/test/java
src/test/resources
JRE System Library [jdk-23]
Maven Dependencies
src
target
pom.xml

ServerPublisher.java Main.java x
19 public class Main {
20     public static void main(String[] args) {
21         try {
22             URL url1 = new URL("http://localhost:8080/ws/serviceWeb1?wsdl");
23             URL url2 = new URL("http://localhost:8080/ws/serviceWeb2?wsdl");
24
25             HotelServiceWeb1ImplService serviceImpl1 = new HotelServiceWeb1ImplService(url1);
26             HotelServiceWeb2ImplService serviceImpl2 = new HotelServiceWeb2ImplService(url2);
27
28             IHotelServiceWeb1 proxy1 = serviceImpl1.getHotelServiceWeb1ImplPort();
29             IHotelServiceWeb2 proxy2 = serviceImpl2.getHotelServiceWeb2ImplPort();
30
31
32             System.out.println(proxy1.rechercheDisponibilites("2024-12-15", "2024-12-20", 2, 500.0));
33
34
35         } catch (MalformedURLException e) {
36             e.printStackTrace();
37         }
38     }
39 }
40 }

Problems Javadoc Declaration Console x
<terminated> Main (3) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (22 déc. 2024, 22:07:17 - 22:07:20) [pid: 22456]
Hotel : Hotel du Soleil, Type : Simple, Numero : 101, Prix : 75.0 €, Capacité : 1 personnes, Superficie : 20.0 m2, Etage : 1
```

Remarque :

Pour le client, j'ai réussi à implémenter la méthode **rechercheDisponibilites** et à afficher les résultats dans la console. Nous pouvons même ajuster les paramètres en fonction de nos recherches. Cependant pour la méthode **effectueReservation**, j'ai rencontré un problème avec le traitement des dates. Je n'ai pas pu faire fonctionner l'algorithme de gestion des dates, donc j'ai commenté la méthode **effectueReservation**.

Il me semble que la solution consiste à transformer les dates en String, comme c'est le cas dans ma méthode **rechercheDisponibilites**, où les dates sont traitées sous forme de String.

J'ai essayé cette modification, mais elle génère des erreurs dans ma version du serveur, qui gère plusieurs dates.

```
19 public class Main {
20     public static void main(String[] args) {
21         try {
22             URL url1 = new URL("http://localhost:8080/ws/serviceWeb1?wsdl");
23             URL url2 = new URL("http://localhost:8080/ws/serviceWeb2?wsdl");
24
25             HotelServiceWeb1ImplService serviceImpl1 = new HotelServiceWeb1ImplService(url1);
26             HotelServiceWeb2ImplService serviceImpl2 = new HotelServiceWeb2ImplService(url2);
27
28             IHotelServiceWeb1 proxy1 = serviceImpl1.getHotelServiceWeb1ImplPort();
29             IHotelServiceWeb2 proxy2 = serviceImpl2.getHotelServiceWeb2ImplPort();
30
31
32             System.out.println(proxy1.rechercheDisponibilites("2024-12-15", "2024-12-20", 2, 500.0));
33             /*System.out.println(proxy2.effectueReservation
34              * /("password123", 101, "Hajar Boumezgane", "hajar@email.com",
35              * /("0711757179", dateArriveeDate, dateDepartDate));*/
36
37         } catch (MalformedURLException e) {
38             e.printStackTrace();
39         }
40     }
41 }
```

Problems | Javadoc | Declaration | Console x

<terminated> Main (3) [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (22 déc. 2024, 22:17:58 - 22:18:00) [pid: 17608]

Hotel : Hotel du Soleil, Type : Simple, Numero : 101, Prix : 75.0 €, Capacité : 1 personnes, Superficie : 20.0 m2, Etage : 1

Remarque : Pour exécuter le client, il faut d'abord lancer le serveur.

Hajar Boumezgane