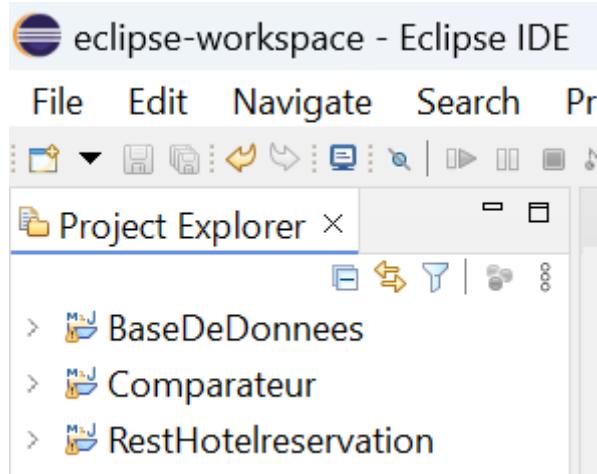




Dans le cadre de ce projet, j'ai développé une application **REST** permettant de gérer des réservations d'hôtels. L'objectif principal était de concevoir une plateforme distribuée, permettant à des agences partenaires de consulter les disponibilités des chambres d'hôtels et de procéder à des réservations via des services web REST.

Pour répondre aux exigences du projet, j'ai structuré le développement en plusieurs étapes essentielles. En suivant la consigne de l'énoncé et afin de répondre à ces exigences, j'ai décidé d'organiser mon travail en **3 principaux packages** :

- **RestHotelreservation** (Version Distribuée + Image)
- **Comparateur**
- **BaseDeDonnees** (Bonus)



### **La Question 1 : Version distribuée-Agence de voyage et hôtels basée sur REST**

Pour répondre à la question 1, j'ai utilisé le package **RestHotelReservation**, qui se concentre sur la gestion des réservations d'hôtels via des services REST.

J'ai utilisé donc Spring Boot, comme framework pour le développement d'applications Java. J'ai opté pour les packages suivant :

- **controller** pour gérer les requêtes entrantes,
- **service** pour encapsuler la logique métier,

- **model** pour définir les entités comme Hotel, Chambre, Agence, et Reservation,
- et enfin **repositories** pour les interactions avec la base de données.

Concernant la base de données, j'ai configuré une base H2 en mémoire dans le fichier **application.properties**. J'ai également défini un **port 8081** pour le serveur et activé la console H2 pour visualiser les données si nécessaire.

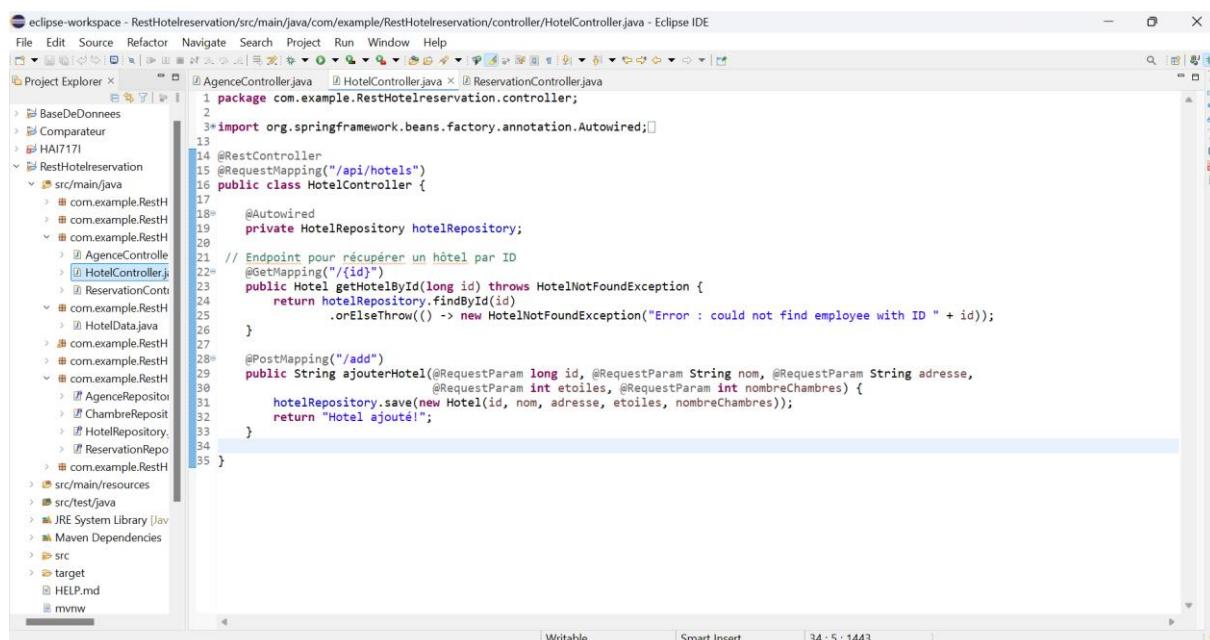
J'ai ensuite conçu les entités principales du projet. Par exemple, l'entité **Hotel** inclut des attributs comme **le nom, l'adresse, le nombre d'étoiles, et le nombre de chambres disponibles**. L'entité **Chambre** capture les informations sur les différents types de chambres, avec des détails tels que le type (**simple, double**), **le prix, la disponibilité**, et **une URL pour afficher une image de la chambre**. J'ai également créé les entités **Agence** pour gérer les partenaires et **Reservation** pour enregistrer les réservations effectuées par les clients.

En parallèle, j'ai développé **les repositories** à l'aide de **JpaRepository**. Par exemple, **le ChambreRepository** contient une méthode personnalisée permettant de **rechercher des chambres disponibles** en fonction de leur type et de leur disponibilité.

Pour encapsuler la logique métier, j'ai implémenté les services **AgenceService** et **ReservationService**.

Ces services sont essentiels pour traiter les demandes des clients. Par exemple, **AgenceService** contient une méthode qui retourne les chambres disponibles, tandis que **ReservationService** vérifie la disponibilité d'une chambre, l'associe à une réservation, et met à jour son statut dans la base de données.

Une fois les services opérationnels, j'ai développé des **endpoints** dans le package **controller**. Ces **endpoints** permettent d'exposer les fonctionnalités sous forme d'API REST. Par exemple, **l'endpoint /api/agence/disponibilite** permet de consulter les chambres disponibles en fonction du type et de la disponibilité. Quant à **l'endpoint /api/reservation/reserver**, il permet aux agences d'effectuer une réservation en fournissant les informations nécessaires comme le type de chambre, le nom du client, et les coordonnées.



```

eclipse-workspace - RestHotelreservation/src/main/java/com/example/RestHotelreservation/controller/HotelController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer X BaseDeDonnees Comparateur HA17171 RestHotelreservation src/main/java com.example.RestH com.example.RestH com.example.RestH AgenceController HotelController ReservationController com.example.RestH HotelData.java com.example.RestH com.example.RestH com.example.RestH AgenceRepository ChambreRepository HotelRepository ReservationRepository com.example.RestH src/main/resources src/test/java JRE System Library [Java] Maven Dependencies src target HELP.md mvnw
AgenceController.java HotelController.java ReservationController.java
1 package com.example.RestHotelreservation.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestParam;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import com.example.RestHotelreservation.model.Hotel;
11 import com.example.RestHotelreservation.repository.HotelRepository;
12
13 /**
14 * Endpoint pour récupérer un hôtel par ID
15 */
16 @RestController
17 @RequestMapping("/api/hotels")
18 public class HotelController {
19     @Autowired
20     private HotelRepository hotelRepository;
21
22     /**
23      * Endpoint pour récupérer un hôtel par ID
24      */
25     @GetMapping("/{id}")
26     public Hotel getHotelById(long id) throws HotelNotFoundException {
27         return hotelRepository.findById(id)
28             .orElseThrow(() -> new HotelNotFoundException("Error : could not find employee with ID " + id));
29     }
30
31     /**
32      * Ajouter un nouvel hôtel
33      */
34     @PostMapping("/add")
35     public String ajouterHotel(@RequestParam long id, @RequestParam String nom, @RequestParam String adresse,
36                               @RequestParam int etoiles, @RequestParam int nombreChambres) {
37         hotelRepository.save(new Hotel(id, nom, adresse, etoiles, nombreChambres));
38         return "Hôtel ajouté!";
39     }
40 }

```

The screenshot shows two Java code editors in the Eclipse IDE. The left editor contains the `AgenceController.java` file:

```

1 package com.example.RestHotelreservation.controller;
2
3 import java.util.List;
4
5 @RestController
6 @RequestMapping("/api/agence")
7 public class AgenceController {
8
9     @Autowired
10    private AgenceService agenceService;
11
12    @GetMapping("/disponibilite")
13    public List<Chambre> getChambresDisponibilite(@RequestParam String typeChambre, @RequestParam boolean disponible) {
14        return agenceService.consulterDisponibilite(typeChambre, disponible);
15    }
16}

```

The right editor contains the `ReservationController.java` file:

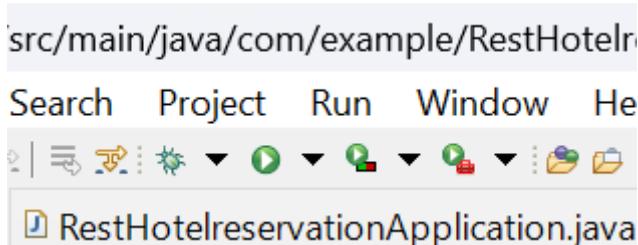
```

1 package com.example.RestHotelreservation.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 @RequestMapping("/api/reservation")
7 public class ReservationController {
8
9     @Autowired
10    private ReservationService reservationService;
11
12    @PostMapping("/reserver")
13    public String reserverChambre(@RequestParam int chambreId,
14                                 @RequestParam String nomClient,
15                                 @RequestParam String clientEmail,
16                                 @RequestParam String identifiantReservation,
17                                 @RequestParam String identifiantHotel,
18                                 @RequestParam String prenomClient,
19                                 @RequestParam String carteBancaire,
20                                 @RequestParam String typeChambre) {
21        return reservationService.reserverChambre(identifiantHotel, identifiantReservation, nomClient, prenomClient, carteBancaire, typeChambre);
22    }
23}

```

## README : Compilation, Exécution et Tests

Pour exécuter l'application **RestHotelReservation**, ouvrez simplement la classe principale `RestHotelreservationApplication.java` et exéutez l'application en cliquant sur le bouton Run (icône verte) dans la barre d'outils d'Eclipse.



**La console affichera ensuite les logs confirmant le démarrage du serveur.**

eclipse-workspace - RestHotelreservation/src/main/java/com/example/RestHotelreservationApplication.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer X RestHotelreservationApplication.java

7  
8 @SpringBootApplication  
9 @EnableTransactionManagement  
10 @ComponentScan(basePackages = {  
11 "com.example.RestHotelreservation.repositories",  
12 "com.example.RestHotelreservation.controller",  
13 "com.example.RestHotelreservation.service",  
14 "com.example.RestHotelreservation.model",  
15 "com.example.RestHotelreservation.data",  
16 "com.example.RestHotelreservation.config"  
17 })  
18  
19 public class RestHotelreservationApplication {  
20  
21\* public static void main(String[] args) {  
22 SpringApplication.run(RestHotelreservationApplication.class, args);  
23 }  
24 }

Console X Problems Debug Shell

RestHotelreservationApplication [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (15 janv. 2025, 06:09:21) [pid: 39412]

=====  
:: Spring Boot :: (v3.4.1)

2025-01-15T06:09:23.197+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] c.e.R.RestHotelreservationApplication : Starti  
2025-01-15T06:09:23.204+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] c.e.R.RestHotelreservationApplication : No act:  
2025-01-15T06:09:23.327+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : Devtoo  
2025-01-15T06:09:23.328+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : For adi  
2025-01-15T06:09:25.296+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Bootst  
2025-01-15T06:09:25.435+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Finish

## Test avec Postman :

- J'ai utilisé Postman pour tester les fonctionnalités REST.
  - **Requête GET** : J'ai testé l'endpoint `/api/agence/disponibilite` pour consulter les chambres disponibles. J'ai utilisé l'URL suivante :

<http://localhost:8081/api/agence/disponibilite?typeChambre=simple&disponible=true>

Cette requête retourne un JSON contenant les chambres disponibles. Voici un exemple de réponse obtenue :

The screenshot shows the Postman interface with a successful HTTP request. The URL is `http://localhost:8081/api/agence/disponibilite?typeChambre=simple&disponible=true`. The response body is displayed in JSON format:

```

1 [           [
2   {           {
3     "id": 2,   "id": 2,
4     "typeChambre": "simple", "typeChambre": "simple",
5     "prix": 300.0, "prix": 300.0,
6     "disponible": true, "disponible": true,
7     "nbLits": 1, "nbLits": 1,
8     "nbPersonnes": 1, "nbPersonnes": 1,
9     "imageUrl": "https://ibb.co/gvvmzff", "imageUrl": "https://ibb.co/gvvmzff",
10    "version": 0 "version": 0
11  }           ]
12 ]

```

On peut changer le TypeChambre dans la requête GET par exemple en choisissant **single**, **double** ou **simple**, pour afficher d'autres disponibilités en fonction de la demande des clients.

- **Requête POST** : J'ai ensuite testé la réservation d'une chambre avec l'endpoint /api/reservation/reserver. Voici l'URL utilisée :

`http://localhost:8081/api/reservation/reserver?chambreId=5&nomClient=Boumezgane&clientEmail=boumeganehajar@gmail.com&identifiantReservation=12345&identifiantHotel=hote1123&prenomClient=Hajar&carteBancaire=1234-5678-9876-5432&typeChambre=single`

- chambreId=5
- nomClient=Boumezgane
- clientEmail=boumezganehajar@gmail.com
- identifiantReservation=12345
- identifiantHotel=hote1123
- prenomClient=Hajar
- carteBancaire=1234-5678-9876-5432
- typeChambre=single

Le serveur a répondu avec un message confirmant la réservation :

**Réservation réussie pour Boumezgane Hajar dans la chambre double**

The screenshot shows the Postman interface. On the left, there's a sidebar with 'Personal Workspace' containing collections like 'Calculatrice SOAP', 'HotelServiceWebImplService', and 'REST'. Under 'REST', 'POST POST' is selected. The main area shows a POST request to 'http://localhost:8081/api/reservation/reserver?chambreId=5&nomClient=Boumezgane&clientEmail=t...'. The 'Body' tab is active, showing a JSON payload:

```

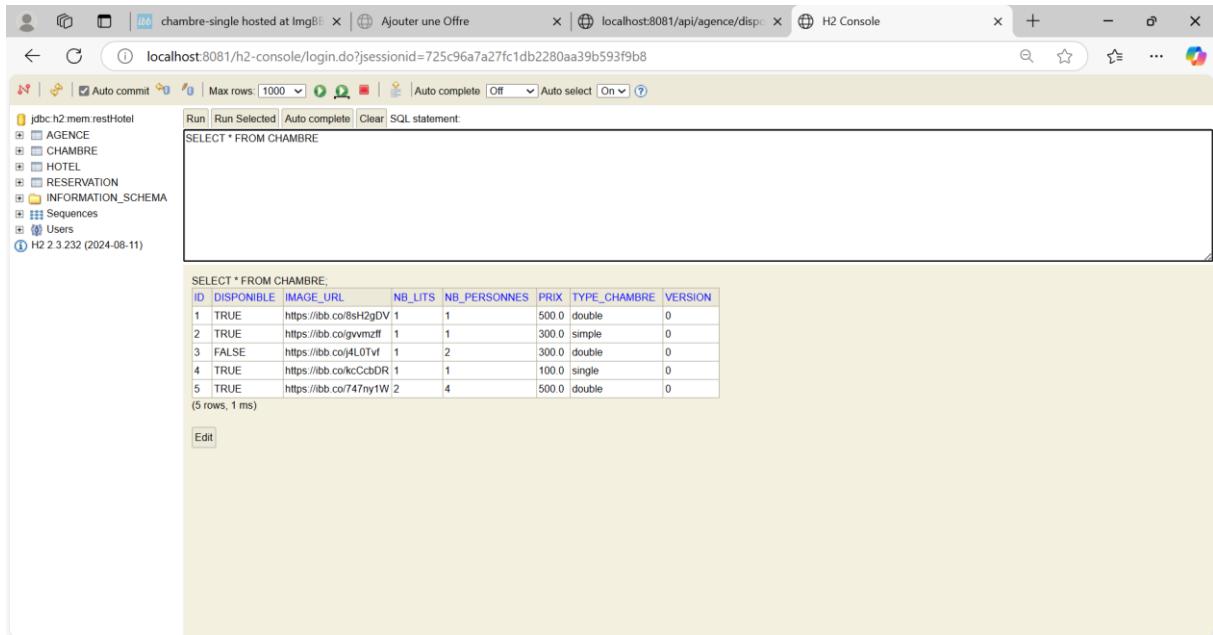
1 "chambreId": 5,
2 "nomClient": "Boumezgane",
3 "clientEmail": "boumezganehajar@gmail.com",
4 "identifiantReservation": "12345",
5 "identifiantHotel": "hotel123",
6 "prenomClient": "Hajar",
7 "carteBancaire": "1234-5678-9876-5432",
8 "typeChambre": "simple"
9

```

The response status is '200 OK' with a duration of '2.64 s' and a size of '230 B'. The response body is: 'Réservation réussie pour Boumezgane Hajar dans la chambre simple'.

Pour accéder à la base de données, comme mentionné dans la photo, il suffit de cliquer sur ce lien [H2 Console](#) et d'utiliser **jdbc:h2:mem:restHotel** comme JDBC URL, avec un mot de passe vide pour se connecter.

The screenshot shows the H2 Console login page. The URL is 'localhost:8081/h2-console/login.jsp?jsessionid=725c96a7a27fc1db2280aa39b593f9b8'. The page has a 'Login' header and a 'Saved Settings: Generic H2 (Embedded)' dropdown. The 'Setting Name' field contains 'Generic H2 (Embedded)'. Below it, 'Driver Class' is set to 'org.h2.Driver', 'JDBC URL' is set to 'jdbc:h2:mem:restHotel', 'User Name' is 'sa', and 'Password' is empty. There are 'Connect' and 'Test Connection' buttons at the bottom.



**À noter :** qu'avant de tester le programme, il est nécessaire de lancer le serveur dans Eclipse en utilisant l'option Run :

- Pour le projet **RestHotelReservation**, le serveur fonctionne sur le port 8081.
- Pour le projet **Comparateur**, le serveur fonctionne sur le port 8083.
- Pour le projet **BaseDeDonnees**, le serveur fonctionne sur le port 8085.

## La Question 2 : Version distribuée- service web intègre des images

Pour répondre à la Question 2, j'ai modifié le projet **RestHotelReservation** afin de permettre l'intégration d'URL d'images pour les chambres, comme demandé dans l'énoncé. Ces URL servent à illustrer les chambres disponibles et à améliorer l'expérience utilisateur en proposant une visualisation des chambres lors de la consultation des disponibilités.

### Pour l'ajout des URL d'images

J'ai utilisé la plateforme **ImgBB** pour héberger les images des différentes chambres. Cette plateforme permet de générer des liens directs vers les images téléchargées, ce qui facilite leur intégration dans le projet. Pour chaque type de chambre (simple, double, suite), j'ai ajouté une URL unique dans l'entité Chambre.

Par exemple, dans l'entité Chambre, j'ai ajouté un **attribut imageUrl** pour stocker l'URL de l'image associée à chaque type de chambre. Voici une portion du code :

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure under "RestHotelreservation".
- Code Editor:** Displays the content of `Chambre.java`. The code defines a class `Chambre` with fields: `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`, `private int id;`, `private String typeChambre;`, `private double prix;`, `private boolean disponible;`, `private int nbLits;`, `private int nbPersonnes;`, `private String imageUrl;`, and `@Version`.
- Console:** Shows the output of the application startup and some Spring Boot logs.

```
1 package com.example.RestHotelreservation.model;
2
3 import jakarta.persistence.Entity;
4
5 @Entity
6 public class Chambre {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private int id;
11
12    private String typeChambre; // (Simple, Double, Suite)
13
14    private double prix;
15
16    private boolean disponible;
17
18    private int nbLits;
19
20    private int nbPersonnes;
21
22    private String imageUrl;
23
24    @Version
25
26}
```

```
RestHotelreservationApplication [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (15 janv. 2025, 06:09:21) [pid: 39412]
:: Spring Boot ::      (v3.4.1)

2025-01-15T06:09:23.197+01:00  INFO 39412 --- [RestHotelreservation] [ restartedMain] c.e.R.RestHotelreservationApplication : Start
2025-01-15T06:09:23.204+01:00  INFO 39412 --- [RestHotelreservation] [ restartedMain] c.e.R.RestHotelreservationApplication : No ac...
2025-01-15T06:09:23.327+01:00  INFO 39412 --- [RestHotelreservation] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devto...
2025-01-15T06:09:23.328+01:00  INFO 39412 --- [RestHotelreservation] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For ad...
```

```
private String imageUrl; // Nouveau Attribut pour l'image
```

### **Pour la mise à jour de la base de données :**

Dans la classe **HotelData**, j'ai préchargé la base de données avec des chambres, en ajoutant les URL d'images correspondantes. Voici une image :

eclipse-workspace - RestHotelreservation/src/main/java/com/example/RestHotelreservation/data/HotelData.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer X RestHotelreservationApplication.java HotelData.java Chambre.java

BaseDeDonnees Comparateur HAI7171 RestHotelreservation src/main/java com.example.RestHotelreservation RestHotelreservationApplication TransactionConfig.java com.example.RestHotelreservation AgenceController.java HotelController.java ReservationController.java com.example.RestHotelreservation HotelData.java com.example.RestHotelreservation HotelException.java HotelNotFoundedException.java Agence.java Chambre.java Hotel.java Reservation.java com.example.RestHotelreservation AgenceRepository.java ChambreRepository.java HotelRepository.java ReservationRepository.java com.example.RestHotelreservation AnnoncesService.java

```
public CommandLineRunner initDatabase(HotelRepository hotelRepository, ChambreRepository chambreRepository) {
    return args -> {
        logger.info("Preloading database with hotels");
        hotelRepository.save(new Hotel(13579L, "Hotel Lux", "123 Rue de la Ville", 5, 2));
        hotelRepository.save(new Hotel(13555L, "Hotel Relax", "456 Avenue de Paris", 4, 3));
        logger.info("Preloading complete");

        logger.info("Preloading database with rooms");
        //String typeChambre, double prix, boolean disponible, int nbLits, int nbPersonnes)
        chambreRepository.save(new Chambre("double", 500, true, 1, 1, "https://ibb.co/8sH2g0V"));
        chambreRepository.save(new Chambre("simple", 300, true, 1, 1, "https://ibb.co/gvVmzff"));
        chambreRepository.save(new Chambre("double", 300, false, 1, 2, "https://ibb.co/j4L0TvF"));
        chambreRepository.save(new Chambre("single", 100, true, 1, 1, "https://ibb.co/kCcCcbDR"));
        chambreRepository.save(new Chambre("double", 500, true, 2, 4, "https://ibb.co/747ny1W"));
        logger.info("Preloading database with rooms");
    };
}
```

Console X Problems Debug Shell

RestHotelreservationApplication [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (15 janv. 2025, 06:09:21) [pid: 39412]

:: Spring Boot :: (v3.4.1)

2025-01-15T06:09:23.197+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] c.e.R.RestHotelreservationApplication : Startin...  
2025-01-15T06:09:23.204+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] c.e.R.RestHotelreservationApplication : No act...  
2025-01-15T06:09:23.327+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtoo...  
2025-01-15T06:09:23.328+01:00 INFO 39412 --- [RestHotelreservation] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For ad...

Grâce à cette modification, l'endpoint **/api/agence/disponibilite** retourne désormais une réponse JSON incluant l'URL de l'image associée à chaque chambre. Voici une image montrant la réponse obtenue dans Postman avec une requête GET :

The screenshot shows the Postman interface with a successful GET request. The URL is `http://localhost:8081/api/agence/disponibilite?typeChambre=simple&disponible=true`. The response body is a JSON array with one element:

```

1 [ {
2   "id": 2,
3   "typeChambre": "simple",
4   "prix": 300.0,
5   "disponible": true,
6   "nbLits": 1,
7   "nbPersonnes": 1,
8   "imageUri": "https://ibb.co/gvvmzff",
9   "version": 0
10 }
11 ]
12

```

En cliquant sur le lien obtenu dans Postman, on peut visualiser directement l'image associée. Une fois que l'on clique sur le lien, toutes les informations relatives à cette image sont affichées dans Postman après avoir envoyé la requête GET.

The screenshot shows the Postman interface with the image URL `https://ibb.co/gvvmzff`. The response body is the HTML content of the image page:

```

1 <!DOCTYPE HTML>
2 <html xml:lang="en" lang="en" dir="ltr" class="device-nonmobile tone-light no-js" prefix="og: http://
3   ogp.me/ns#>
4 <head>
5   <meta charset="utf-8">
6   <meta name="apple-mobile-web-app-status-bar-style" content="black">
7   <meta name="apple-mobile-web-app-capable" content="yes">
8   <meta name="viewport" content="width=device-width, initial-scale=1">
9   <meta name="google" content="notranslate" />
10  <meta name="description" content="Image chambre-single hosted at ImgBB">
11  <title>chambre-single hosted at ImgBB - ImgBB</title>
12  <link rel="preconnect" href="https://simgbb.com">
13  <link rel="alternate" href="https://fonte.morlaais.be/>

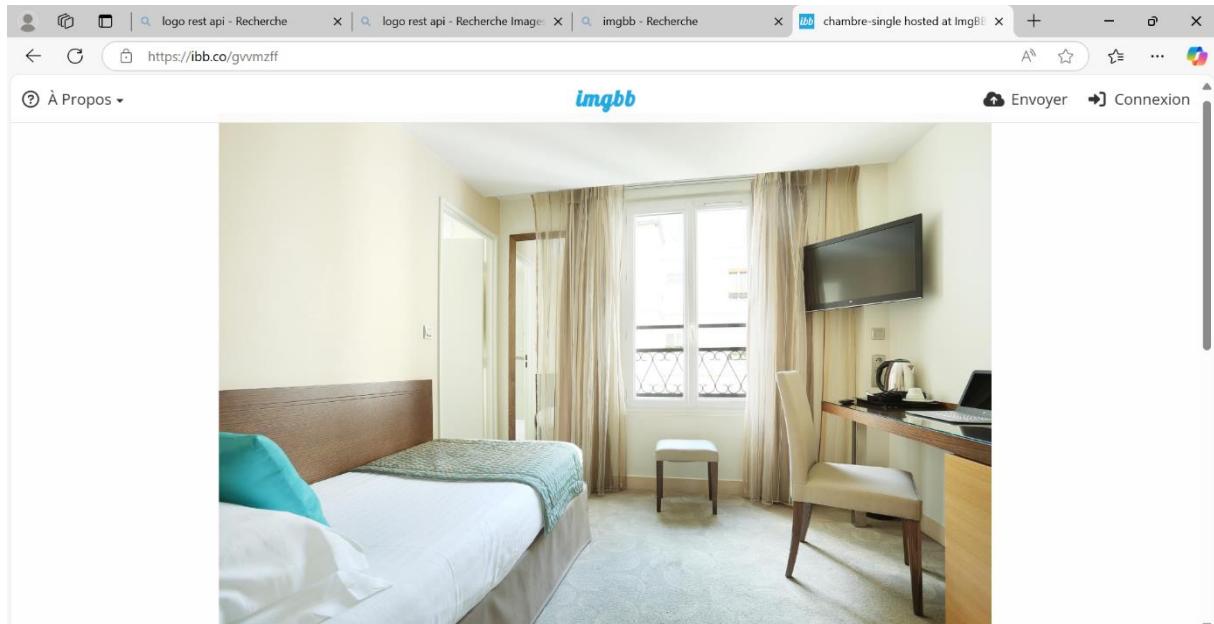
```

En outre, le client peut copier ce lien « [chambre-single hosted at ImgBB — ImgBB](https://ibb.co/gvvmzff) » dans un navigateur pour afficher la photo correspondant à la disponibilité existante.

Par exemple, dans cette recherche, j'ai spécifié une chambre simple. En copiant le lien de l'image dans le navigateur, j'ai pu visualiser l'image de la chambre proposée.

Vous pouvez également tester les autres images existantes dans la base de données en modifiant simplement le paramètre dans l'URL, par exemple en remplaçant simple par double pour afficher les images des chambres doubles.

Voici la photo obtenue grâce à ce lien.



### La Question 3 : Un service web REST proposé par les agences de voyage pour comparateurs

Pour répondre à cette question, j'ai créé un nouveau projet séparé intitulé **Comparateur**. Ce projet repose sur une architecture Spring Boot déployée sur le port 8083 et comprend les **quatre principaux packages** :

Le premier, Comparateur, contient la classe principale **ComparateurApplication**, responsable du démarrage de l'application. **Le package Controller** regroupe les endpoints REST, dont la classe **HotelController** qui gère les requêtes des utilisateurs. Ensuite, le **package Model** inclut quatre entités principales : **Agence**, qui représente une agence de voyage ; **Hotel**, qui contient les informations sur les hôtels ; **Offre**, qui lie une agence à un hôtel en spécifiant un prix proposé ; et **Reservation**, qui gère les informations relatives aux réservations. Enfin, **le package Service** contient la logique métier dans la classe **HotelService**, qui est responsable du stockage des données et de la gestion des opérations sur les offres.

The screenshot shows the Eclipse IDE interface. The top bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The Project Explorer on the left lists the project structure: BaseDeDonnees, Comparateur (containing src/main/java and src/main/resources), and static. The src/main/java folder contains sub-packages like com.example.Comparateur, com.example.Comparateur.controller, com.example.Comparateur.model, and com.example.Comparateur.service. The code editor on the right displays the main class ComparateurApplication.java:

```
1 package com.example.Comparateur;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 @ComponentScan(basePackages = {
8     "com.example.Comparateur.controller",
9     "com.example.Comparateur.service",
10    "com.example.Comparateur.model"
11 })
12
13 public class ComparateurApplication {
14
15     public static void main(String[] args) {
16         SpringApplication.run(ComparateurApplication.class, args);
17     }
18 }
19
20
```

Dans ce projet Comparateur, il faut mentionner que les classes **HotelController** et **HotelService** jouent un rôle central dans la gestion des offres d'hôtels et des réservations.

J'ai utilisé la classe HotelController pour exposer trois **endpoints REST** principaux, qui permettent d'interagir avec les données des offres et des réservations :

- **Endpoint /offres** : Ce endpoint utilise la méthode getOffres() pour récupérer et retourner une liste complète des offres d'hôtels sous forme de JSON. Les utilisateurs peuvent ainsi consulter toutes les offres disponibles sans appliquer de filtres.
- **Endpoint /comparaison** : Avec ce point d'entrée, les utilisateurs peuvent spécifier une ville et un budget maximal via des paramètres. La méthode comparerOffres() filtre ensuite les données en fonction des critères fournis et retourne uniquement les offres pertinentes.
- **Endpoint /réservations** : J'ai utilisé la méthode reserver() pour permettre aux utilisateurs d'effectuer une réservation. Cette méthode prend les détails d'une offre dans le corps de la requête (via POST), enregistre la réservation et renvoie un message de confirmation.

```

eclipse-workspace - Comparateur/src/main/java/com/example/Comparateur/controller/HotelController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer Comparator ApplicationComparator HotelController.java HotelService.java
13 @RestController
14 @RequestMapping("/api")
15 public class HotelController {
16
17    @Autowired
18    private HotelService hotelService;
19
20    private List<Reservation> reservations = new ArrayList<>();
21
22    @GetMapping("/offres")
23    public List<Offre> getOffres() {
24        return hotelService.getOffres();
25    }
26
27    @GetMapping("/comparaison")
28    public List<Offre> comparerOffres(@RequestParam String ville, @RequestParam(required = false) Double prix) {
29        return hotelService.comparerOffres(ville, prix);
30    }
31
32    @PostMapping("/reservations")
33    public String reserver(@RequestBody Reservation reservation) {
34        reservations.add(reservation);
35        return "Reservation effectuée pour " + reservation.getClient() + " à " + reservation.getOffre().getHotel().getNom();
36    }

```

Console Problems Debug Shell  
ComparateurApplication [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (15 janv. 2025, 08:14:03) [pid: 26644]

La classe **HotelService** c'est pour gérer toute la logique métier, j'ai structuré cette classe avec :

**Initialisation des données** ou j'ai simulé trois agences et cinq hôtels avec leurs informations de base (nom, ville, prix). Chaque hôtel est associé à une agence avec un prix proposé.

### Gestion des offres :

- La méthode `getOffres()` retourne toutes les offres disponibles.
- La méthode `comparerOffres()` prend en compte la ville et le budget maximal pour retourner uniquement les offres qui respectent ces critères.

```

eclipse-workspace - Comparateur/src/main/java/com/example/Comparateur/service/HotelService.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer Comparator ApplicationComparator HotelController.java HotelService.java Offre.java Hotel.java Reservation.java
12 @Service
13 public class HotelService {
14
15     private List<Offre> offres = new ArrayList<>();
16
17    public HotelService() {
18        Agence agence1 = new Agence("Agence GoVoyage");
19        Agence agence2 = new Agence("Agence Promovacances");
20        Agence agence3 = new Agence("Agence TripInTouch");
21
22        Hotel hotel1 = new Hotel(1, "Hôtel Royal Luxe", "Paris", 100);
23        Hotel hotel2 = new Hotel(2, "Grand Hôtel", "Paris", 150);
24        Hotel hotel3 = new Hotel(3, "Hôtel Relax", "Montpellier", 120);
25        Hotel hotel4 = new Hotel(4, "Hôtel Luxury", "Montpellier", 80);
26        Hotel hotel5 = new Hotel(5, "Hôtel Blanca", "Casablanca", 90);
27
28        offres.add(new Offre(hotel1, agence1, 90));
29        offres.add(new Offre(hotel2, agence2, 140));
30        offres.add(new Offre(hotel3, agence1, 115));
31        offres.add(new Offre(hotel4, agence3, 75));
32        offres.add(new Offre(hotel5, agence3, 100));
33
34    }
35
36    public List<Offre> getOffres() {
37        return offres;
38    }

```

Console Problems Debug Shell  
ComparateurApplication [Java Application] C:\Program Files\Java\jdk-23\bin\javaw.exe (15 janv. 2025, 08:14:03) [pid: 26644]

### Pour les fonctionnalités Implémentées :

La première fonctionnalité est la possibilité de **consulter toutes les offres d'hôtels** via un endpoint REST accessible à l'adresse /api/offres. Cette fonctionnalité renvoie une liste comprenant le nom des hôtels, leur ville, leur prix direct et le prix proposé par les agences, souvent inférieur au tarif direct de l'hôtel.

Une deuxième fonctionnalité est **la comparaison des offres** basée sur des critères donnés. En accédant à l'endpoint /api/comparaison, l'utilisateur peut spécifier une ville et un prix maximal. Le système retourne alors une liste filtrée des offres qui correspondent aux critères, permettant ainsi de visualiser les agences offrant les tarifs les plus compétitifs pour une destination donnée.

Enfin, l'application permet également **de réserver une offre** via l'endpoint /api/reservations. En soumettant une requête avec les informations de l'offre choisie, une réservation est créée, et un message de confirmation est retourné à l'utilisateur, tel que :

« **Réservation effectuée pour Hajar Boumezgane à Hôtel Royal Luxe.** »

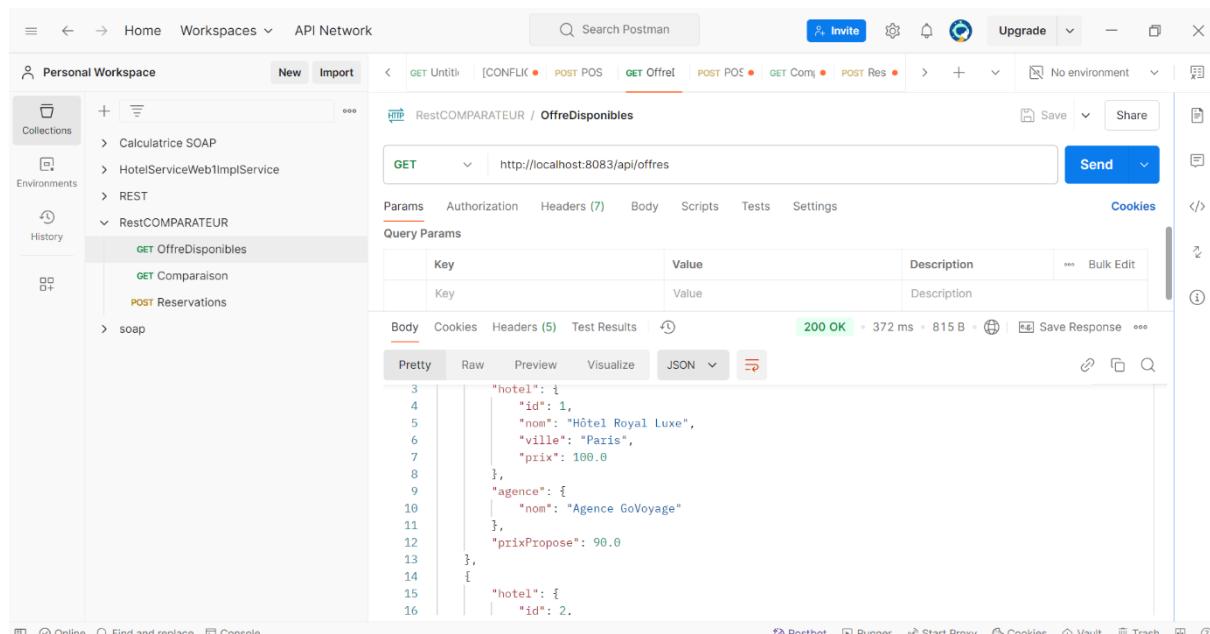
#### - Exécution et Résultats

Pour exécuter le projet, il suffit d'ouvrir la classe ComparateurApplication dans l'environnement de développement Eclipse et de la lancer avec l'option Run. Une fois démarrée, la console confirme le lancement du serveur Spring Boot sur le port 8083, prêt à recevoir des requêtes.

#### - Tests avec Postman :

Dans Postman, j'ai testé les différents endpoints pour valider leur bon fonctionnement.

**La première c'est les Offres** : En accédant à /api/offres avec une requête GET, j'ai obtenu une liste complète des offres disponibles. Par exemple, une offre pour un hôtel à Paris affichait un prix proposé par une agence, inférieur au prix direct de l'hôtel.



The screenshot shows the Postman application interface. On the left, the sidebar displays collections, environments, and history. In the center, a collection named "RestCOMPARATEUR" is selected, and a specific endpoint "GET OffreDisponibles" is highlighted. The request URL is set to "http://localhost:8083/api/offres". The response status is "200 OK" with a duration of 372 ms and a size of 815 B. The response body is displayed in a JSONpretty-printed format:

```
3 |     "hotel": {
4 |       "id": 1,
5 |       "nom": "Hôtel Royal Luxe",
6 |       "ville": "Paris",
7 |       "prix": 100.0
8 |     },
9 |     "agence": {
10 |       "nom": "Agence GoVoyage"
11 |     },
12 |     "prixPropose": 90.0
13 |
14 |
15 |     "hotel": {
16 |       "id": 2,
```

**La deuxième c'est pour tester la Comparaison**, j'ai utilisé /api/comparaison avec les paramètres ville=Montpellier et prix=150. Le système a retourné deux offres disponibles dans

cette ville respectant le critère de prix. Les résultats mettaient en évidence les agences proposant des tarifs avantageux et compétitifs pour chaque hôtel, rendant ainsi la comparaison claire et utile pour l'utilisateur.

```

{
  "hotel": {
    "id": 3,
    "nom": "Hotel Relax",
    "ville": "Montpellier",
    "prix": 120.0
  },
  "agence": {
    "nom": "Agence GoVoyage"
  },
  "prixPropose": 115.0
}

```

Enfin, pour valider la fonctionnalité de **Réservation**, j'ai envoyé une requête POST à /api/reservations avec une offre spécifique. Le système a confirmé la réservation avec un message indiquant le nom du client et de l'hôtel réservé.

```

{
  "offre": {
    "hotel": {
      "id": 1,
      "nom": "Hôtel Royal Luxe",
      "ville": "Paris",
      "prix": 100
    },
    "agence": {
      "nom": "Agence GoVoyage"
    }
  }
}

```

1 Réservation effectuée pour Hajar Boumezgane à Hôtel Royal Luxe

#### La Question 4 : Version distribuée- Base de données et interface graphique (Bonus)

Pour réaliser un travail complet, j'ai également essayé de répondre à la question 4, qui concernait l'intégration de bases de données persistantes aux services web ainsi que la proposition d'interfaces graphiques.

## Pour l'intégration des bases de données :

- J'ai configuré une base de données H2 persistante pour stocker les données, en modifiant le fichier **application.properties**. Cela permet de conserver les données même après le redémarrage de l'application.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like HotelController.java, OffreController.java, ReservationController.java, Agence.java, Hotel.java, Offre.java, and Reservation.java under com.example.BaseD.
- application.properties:** Content of the configuration file:

```
spring.application.name=BaseDeDonnees
spring.datasource.url=jdbc:h2:file:./hotelBaseDonnee
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true
server.port=8085
logging.level.org.springframework=DEBUG
```
- Console:** Shows the application's startup logs:

```
2025-01-15T09:39:48.696+01:00 DEBUG 25860 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance for: org.springframework.context.annotation.AutowiredByTypeAnnotationBeanPostProcessor
2025-01-15T09:39:48.698+01:00 DEBUG 25860 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Autowiring by type for bean 'org.springframework.context.annotation.ConfigurationClassPostProcessor': creating shared instance
2025-01-15T09:39:48.699+01:00 DEBUG 25860 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance for: org.springframework.context.annotation.ConfigurationClassPostProcessor
2025-01-15T09:39:48.701+01:00 DEBUG 25860 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance for: org.springframework.context.annotation.ConfigurationProcessor
2025-01-15T09:39:48.702+01:00 DEBUG 25860 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance for: org.springframework.context.annotation.ConfigurationProcessor
2025-01-15T09:39:48.704+01:00 DEBUG 25860 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance for: org.springframework.context.annotation.ConfigurationProcessor
2025-01-15T09:39:48.706+01:00 DEBUG 25860 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Autowiring by type for bean 'org.springframework.context.annotation.ConfigurationProcessor': creating shared instance
```

- J'ai utilisé un fichier **data.sql** pour initialiser automatiquement les données nécessaires (hôtels, agences, offres) à chaque démarrage.

The screenshot shows the Eclipse IDE interface with the following details:

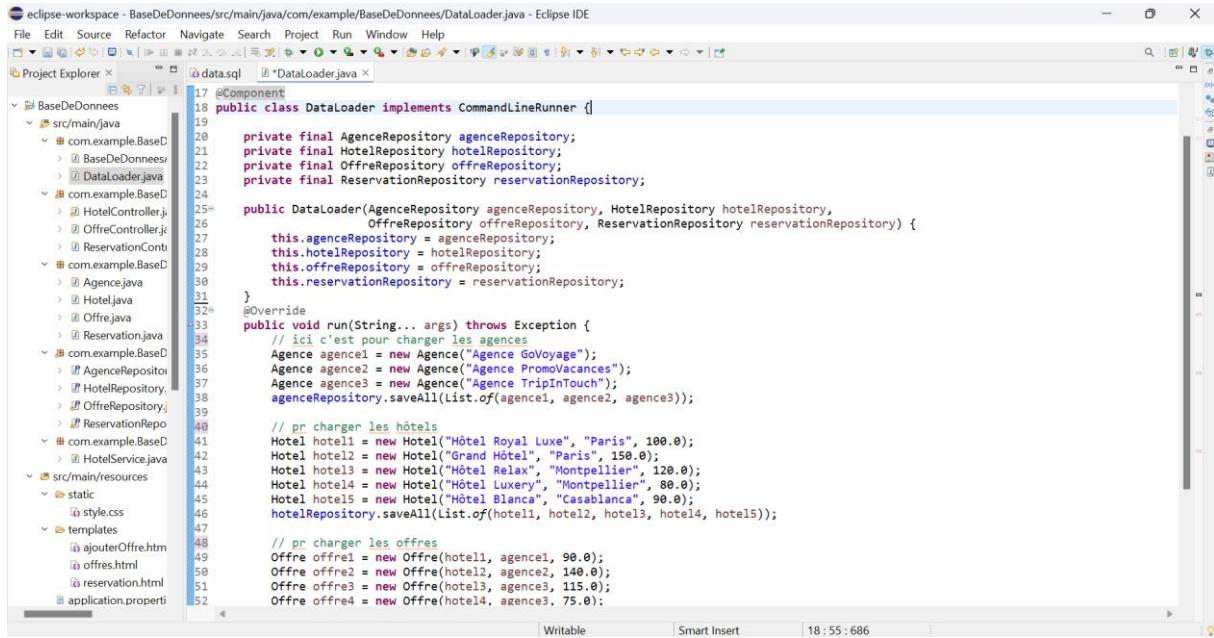
- Project Explorer:** Shows the project structure with files like ReservationController.java, Agence.java, Hotel.java, Offre.java, and Reservation.java under com.example.BaseD.
- data.sql:** Content of the SQL script:

```
-- Insertion dans AGENCE
INSERT INTO AGENCE (id, nom) VALUES (1, 'Agence GoVoyage');
INSERT INTO AGENCE (id, nom) VALUES (2, 'Agence PromoVacances');
INSERT INTO AGENCE (id, nom) VALUES (3, 'Agence TripInTouch');

-- Insertion dans HOTEL
INSERT INTO HOTEL (id, nom, ville, prix) VALUES (1, 'Hôtel Royal Luxe', 'Paris', 100.0);
INSERT INTO HOTEL (id, nom, ville, prix) VALUES (2, 'Grand Hôtel', 'Paris', 150.0);
INSERT INTO HOTEL (id, nom, ville, prix) VALUES (3, 'Hôtel Relax', 'Montpellier', 120.0);
INSERT INTO HOTEL (id, nom, ville, prix) VALUES (4, 'Hôtel Luxury', 'Montpellier', 80.0);
INSERT INTO HOTEL (id, nom, ville, prix) VALUES (5, 'Hôtel Blanca', 'Casablanca', 90.0);

-- Insertion dans OFFRE
INSERT INTO OFFRE (id, prix_propose, hotel_id, agence_id) VALUES (1, 90.0, 1, 1);
INSERT INTO OFFRE (id, prix_propose, hotel_id, agence_id) VALUES (2, 140.0, 2, 2);
INSERT INTO OFFRE (id, prix_propose, hotel_id, agence_id) VALUES (3, 115.0, 3, 3);
INSERT INTO OFFRE (id, prix_propose, hotel_id, agence_id) VALUES (4, 75.0, 4, 3);
INSERT INTO OFFRE (id, prix_propose, hotel_id, agence_id) VALUES (5, 90.0, 5, 1);
```
- Console:** Shows the application's startup logs, identical to the previous screenshot.

- J'ai également implémenté un composant **DataLoader** pour insérer dynamiquement des données via le code Java, comme alternative à l'utilisation de scripts SQL.



```

eclipse-workspace - BaseDeDonnees/src/main/java/com/example/BaseDeDonnees/DataLoader.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer X data.sql DataLoader.java
BaseDeDonnees
src/main/java
com.example.BaseD
BaseDeDonnees
DataLoader.java
com.example.BaseD
HotelController.java
OffreController.java
ReservationController
com.example.BaseD
Agence.java
Hotel.java
Offre.java
Reservation.java
com.example.BaseD
AgenceRepository
HotelRepository
OffreRepository
ReservationRepository
src/main/resources
static
style.css
templates
ajouterOffre.htm
offres.html
reservation.html
application.properties
data.sql

```

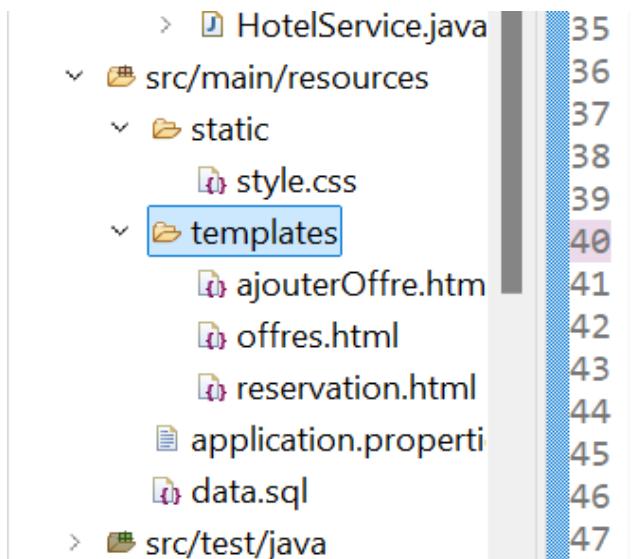
```

17 @Component
18 public class DataLoader implements CommandLineRunner {
19
20     private final AgenceRepository agenceRepository;
21     private final HotelRepository hotelRepository;
22     private final OffreRepository offreRepository;
23     private final ReservationRepository reservationRepository;
24
25     public DataLoader(AgenceRepository agenceRepository, HotelRepository hotelRepository,
26                      OffreRepository offreRepository, ReservationRepository reservationRepository) {
27         this.agenceRepository = agenceRepository;
28         this.hotelRepository = hotelRepository;
29         this.offreRepository = offreRepository;
30         this.reservationRepository = reservationRepository;
31     }
32     @Override
33     public void run(String... args) throws Exception {
34         // ici c'est pour charger les agences
35         Agence agence1 = new Agence("Agence GoVoyage");
36         Agence agence2 = new Agence("Agence Promovacances");
37         Agence agence3 = new Agence("Agence TripInTouch");
38         agenceRepository.saveAll(List.of(agence1, agence2, agence3));
39
40         // pr charger les hôtels
41         Hotel hotel1 = new Hotel("Hôtel Royal Luxe", "Paris", 100.0);
42         Hotel hotel2 = new Hotel("Grand Hôtel", "Paris", 150.0);
43         Hotel hotel3 = new Hotel("Hôtel Relax", "Montpellier", 120.0);
44         Hotel hotel4 = new Hotel("Hôtel Luxery", "Montpellier", 80.0);
45         Hotel hotel5 = new Hotel("Hôtel Blanca", "Casablanca", 90.0);
46         hotelRepository.saveAll(List.of(hotel1, hotel2, hotel3, hotel4, hotels));
47
48         // pr charger les offres
49         Offre offre1 = new Offre(hotel1, agence1, 90.0);
50         Offre offre2 = new Offre(hotel2, agence2, 140.0);
51         Offre offre3 = new Offre(hotel3, agence3, 115.0);
52         Offre offre4 = new Offre(hotel4, agence3, 75.0);
53     }
54 }

```

## Pour la création d'interfaces graphiques :

- J'ai développé deux pages HTML principales en utilisant Thymeleaf pour intégrer dynamiquement les données :



- Une page pour afficher les offres (nom des hôtels, agences, prix proposés) avec une barre de recherche permettant de filtrer par ville et prix maximal.

eclipse-workspace - BaseDeDonnees/src/main/resources/templates/offres.html - Eclipse IDE

```

File Edit Source Navigate Search Project Run Window Help
Project Explorer X Data.sql DataLoader.java offres.html
1 <!DOCTYPE html>
2 <html xmlns="http://www.thymeleaf.org">
3<head>
4   <title>Liste des Offres</title>
5   <link rel="stylesheet" href="#" style="css">
6 </head>
7<body>
8   <h1>Liste des Offres</h1>
9   <form action="#" method="get">
10    <input type="text" name="ville" placeholder="Ville">
11    <input type="number" name="prix" placeholder="Prix Maximum">
12    <button type="submit" class="btn-vert">Rechercher</button>
13  </form>
14  <table>
15    <tr>
16      <th>Nom de l'Hôtel</th>
17      <th>Agence</th>
18      <th>Prix Proposé</th>
19    </tr>
20    <tr th:each="offre : $offres">
21      <td th:text="${offre.hotel.nom}">Nom de l'Hôtel</td>
22      <td th:text="${offre.agence.nom}">Agence</td>
23      <td th:text="${offre.prixPropose}">Prix Proposé</td>
24    </tr>
25  </table>
26  <div class="ajouter-btn">
27    <a href="#">Ajouter une Offre</a>
28  </div>
29 </body>
30 </html>

```

- Le résultat visible dans le navigateur sur le lien suivant : [Liste des Offres](http://localhost:8085/offres)  
<http://localhost:8085/offres>

Nom de l'Hôtel	Agence	Prix Proposé
Hôtel Royal Luxe	Agence GoVoyage	90.0
Grand Hôtel	Agence PromoVacances	140.0
Hôtel Relax	Agence TripInTouch	115.0
Hôtel Luxury	Agence TripInTouch	75.0
Hôtel Blanca	Agence GoVoyage	90.0

A green 'Ajouter une Offre' button is located at the bottom of the table."/>

- Une page pour **ajouter une nouvelle offre** via un formulaire, où l'utilisateur peut sélectionner un hôtel et une agence existants, puis définir un prix proposé.

eclipse-workspace - BaseDeDonnees/src/main/resources/templates/ajouterOffre.html - Eclipse IDE

```

File Edit Source Navigate Search Project Run Window Help
Project Explorer x Data.sql DataLoader.java offres.html ajouterOffre.html x
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3<head>
4   <title>Ajouter une Offre</title>
5   <link rel="stylesheet" href="/style.css">
6 </head>
7 <body>
8   <h1>Ajouter une Offre</h1>
9   <form action="/offres/ajouter" method="post">
10    <label for="hotel">Nom de l'Hôtel :</label>
11    <select name="hotel" id="hotel" required>
12      <option th:each="hotel : ${hotels}" th:value="${hotel.id}" th:text="${hotel.nom}"></option>
13    </select>
14
15    <label for="agence">Nom de l'Agence :</label>
16    <select name="agence" id="agence" required>
17      <option th:each="agence : ${agences}" th:value="${agence.id}" th:text="${agence.nom}"></option>
18    </select>
19
20    <label for="prix">Pri Proposé :</label>
21    <input type="number" name="prix" id="prix" required>
22
23    <button type="submit">Ajouter l'Offre</button>
24  </form>
25
26  <a href="/offres">Retour à la liste des offres</a>
27 </body>
28 </html>

```

- Le résultat est disponible sur le lien suivant : [Ajouter une Offre](http://localhost:8085/offres/ajouter)  
<http://localhost:8085/offres/ajouter>

Ajouter une Offre

Nom de l'Hôtel :

Hôtel Royal Luxe

Nom de l'Agence :

Agence GoVoyage

Prix Proposé :

AJOUTER L'OFFRE

[Retour à la liste des offres](#)

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure for "BaseDeDonnees".
- Code Editor:** Displays the content of `BaseDeDonneesApplication.java`:

```

1 package com.example.BaseDeDonnees;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication(scanBasePackages = {"com.example.BaseDeDonnees",
6 "com.example.BaseDeDonnees.controller",
7 "com.example.BaseDeDonnees.service",
8 "com.example.BaseDeDonnees.repository",
9 "com.example.BaseDeDonnees.model"})
10
11 public class BaseDeDonneesApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(BaseDeDonneesApplication.class, args);
15     }
16
17 }
18
19 }
20

```

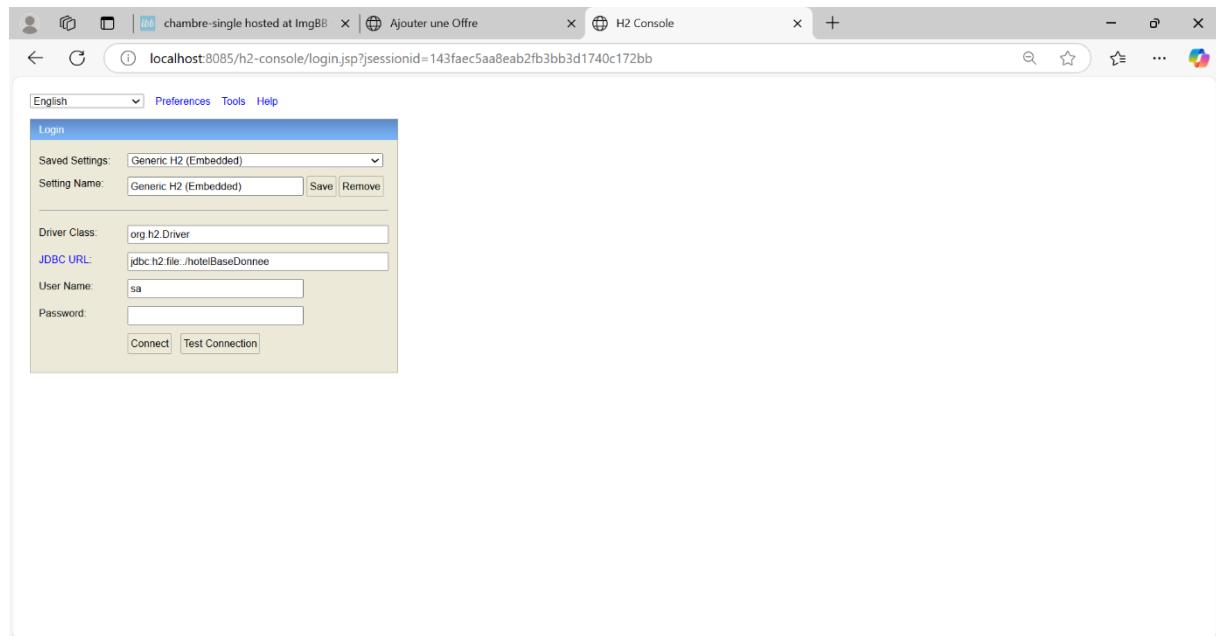
- Java Console:** Shows the application's log output:

```

2025-01-15T11:46:52.136+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.137+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.137+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.138+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.139+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Autowiring by type from bean 'BaseDeDonnees'
2025-01-15T11:46:52.140+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.141+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Autowiring by type from bean 'BaseDeDonnees'
2025-01-15T11:46:52.142+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.143+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.144+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.145+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.146+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.147+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.148+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance
2025-01-15T11:46:52.149+01:00 DEBUG 19100 --- [BaseDeDonnees] [main] o.s.b.f.s.DefaultListableBeanFactory : Creating shared instance

```

Pour accéder à la base de données, comme mentionné dans la photo, il suffit de cliquer sur ce lien [H2 Console](#) et d'utiliser `jdbc:h2:file:/hotelBaseDonnee` comme JDBC URL, avec un mot de passe vide pour se connecter.



The screenshot shows the H2 Console interface with the title bar "H2 Console" and "Offres d'Hôtels". The URL is "localhost:8085/h2-console/login.do?sessionid=ac6e394881da0c30850909deea9b4031". The left sidebar lists database objects: AGENCE, HOTEL, OFFRE, RESERVATION, INFORMATION\_SCHEMA, and Users. The main area contains a SQL statement: "SELECT \* FROM OFFRE". Below it is a table with 5 rows of data:

ID	PRIX_PROPOSE	AGENCE_ID	HOTEL_ID
161	90.0	164	166
162	140.0	165	167
163	115.0	166	168
164	75.0	166	169
165	90.0	164	170

(5 rows, 2 ms)

**Edit**

**À noter :** qu'avant de tester le programme, il est nécessaire de lancer le serveur dans Eclipse en utilisant l'option Run :

- Pour le projet **RestHotelReservation**, le serveur fonctionne sur le port 8081.
- Pour le projet **Comparateur**, le serveur fonctionne sur le port 8083.
- Pour le projet **BaseDeDonnees**, le serveur fonctionne sur le port 8085.

**Hajar BOUMEZGANE**