|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PROJET REST | | | | | | | |
|  |  |  |  |  |  |
| Etienne  Martin-Chantereau | | |  | Université de MONTPELLIER  Master Informatique Parcours ICO | | |

|  |  |  |
| --- | --- | --- |
| Framework  Spring  Spring-DATA-JPA  SOMMAIRE  Pages 3-5 : Explication Algorithme Question 1  Page 6-7 : Explication Question 2 Image  Page 7-9 : Explication Question 3 Comparateur  Page 10-12 : Programme Algo  Page 13-14 : Résultat CLI |  | LANGUAGE  JAVA  SQL |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

QUESTION 1 : ALGORITHME CONSULTATION :

    @GetMapping(uri+"/hotel/consultation/id\_agence={identifiantAgence}/password\_agence= {passwordAgence}/date\_arrivee={dateArrivee}/date\_depart={dateDepart}"

            +"/nbPersonne={nbPersonne}/pays={pays}/ville={ville}/etoiles={etoiles}/prixMin ={prixMin}/prixMax={prixMax}")

L’algorithme utilise la méthode hotelService.getHotelByPaysAndVilleAndEtoiles pour obtenir un objet Hotel en utilisant les paramètres pays, ville et etoiles afin d’exécuter une requête sql dérivée via soit l’objet est présent, soit vide.

Dans le cas de présence l’algorithme extrait et stocke le résultat de la requete dans la variable hotelPresent.

Ensuite, l'algorithme filtre la liste des agences partenaires de l'hôtel en utilisant la méthode stream() et la méthode filter(), en cherchant une agence avec l'identifiant spécifié (identifiantAgence) et un mot de passe correspondant (passwordAgence). Si une agence correspondante est trouvée, elle est renvoyée dans un objet Optional<AgencePartenaire>. Sinon, l'objet est vide (absent).

Si un objet AgencePartenaire est présent dans l'objet Optional<AgencePartenaire>, alors le coefficient de promotion de cette agence est extrait et stocké dans la variable coeffPromotion. Sinon, la valeur par défaut de coeffPromotion est définie sur 1.

Ensuite, l'algorithme utilise la méthode chambreService.findChambreByDisponbilitee pour obtenir une liste de chambres disponibles en utilisant les paramètres dateArrivee, dateDepart, coeffPromotion, prixMin, prixMax et nbPersonne.

Cette méthode vient exécuter la requête suivante se trouvant dans la class ChambreRepository :

@Query(value = "SELECT \* FROM chambre WHERE chambre.id\_chambre NOT IN "

+ "(SELECT reservation.id\_chambre FROM reservation WHERE reservation.date\_depart > :dateArrivee AND reservation.date\_arrivee < :dateDepart)"

+ "AND prix\_chambre \* :coeffPromotion BETWEEN :prixMin AND :prixMax AND nombre\_lits >= :nbPersonne",nativeQuery = **true**)

Pour chaque chambre de la liste obtenue, un nouvel objet Offre est créé en utilisant le coefficient de promotion (coeffPromotion) et la chambre, et est ajouté à la liste listOffreDispo.

Enfin la liste est retournée au client.

Si la liste est vide alors le client va afficher un message d’erreur sinon il va afficher l’ensemble des caractéristique de l’offre.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

QUESTION 1 : Algorithme Reservation :

Le client a passée en paramètre l’ensemble des données nécessaire à une réservation, je ne voulais pas que l’agence est accès à tous le mapping de mes classes de l’hotel alors j’ai stocké tout les données dans une hashmap que j’envoie ensuite dans le body de la requete post HTPP.

Il utilise la méthode reservationService.findIfReservationIsPossible pour vérifier si une réservation est possible pour les dates spécifiées dans les paramètres reservation.getFirst("dateArrivee") et reservation.getFirst("dateDepart").

Cette méthode fait appel à la requête sql de la class ReservationRepository qui sélectionne tous les tuples de la table reservation où la date de départ est postérieure à la date spécifiée dans le paramètre dateArrivee et la date d'arrivée est antérieure à la date spécifiée dans le paramètre dateDepart.

@Query(value= "SELECT \* FROM reservation WHERE date\_depart > :dateArrivee AND date\_arrivee < :dateDepart", nativeQuery=**true**)

Cette méthode renvoie une liste vide si la réservation est possible, sinon elle renvoie une liste non vide.

S'il est possible de faire une réservation, l'algorithme crée un nouvel objet Reservation en utilisant les dates de réservation spécifiées dans les paramètres reservation.getFirst("dateArrivee") et reservation.getFirst("dateDepart"), ainsi que le nombre de lits spécifié dans reservation.getFirst("nbLitsResa").

Il crée également un nouvel objet Client en utilisant les prénom et nom spécifiés dans les paramètres reservation.getFirst("prenomClient") et reservation.getFirst("nomClient").

Ensuite, l'algorithme vérifie si la carte bancaire spécifiée dans les paramètres reservation.getFirst("numeroCarte"), reservation.getFirst("cardDateExpiration") et reservation.getFirst("cryptogramme") est déjà associée à un client existant en utilisant la méthode carteBancaireService.findCarteBancaireByNumeroANDcryptogrammeANDdateExpiration.

Cette méthode fait appel à une requête sql de CarteBancaireRepository :

@Query(value = "SELECT \* FROM cartebancaire WHERE numero\_carte = :numeroCarte AND date\_expiration= :dateExpiration AND cryptogramme = :cryptogramme", nativeQuery = **true**)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

Cas 1 :

Si la carte bancaire n'est pas trouvée, un nouvel objet CarteBancaire est créé en utilisant les détails de la carte bancaire spécifiés dans les paramètres, et est associé au client nouvellement créé.

La carte bancaire est enregistrée dans la base de données en utilisant la méthode carteBancaireService.createCarteBancaire.

Cas 2 :

Si la carte bancaire est déjà associée à un client existant, alors le client nouvellement créé est associé à cette carte bancaire existante et est enregistré dans la base de données en utilisant la méthode clientService.createClient.

Ensuite, la réservation nouvellement créée est associée au client nouvellement créé ou existant et à la chambre spécifiée dans le paramètre reservation.getFirst("identifiantOffre"), qui est obtenue en utilisant la méthode chambreService.findChambreById. La réservation est enregistrée dans la base de données en utilisant la méthode reservationService.createReservation.

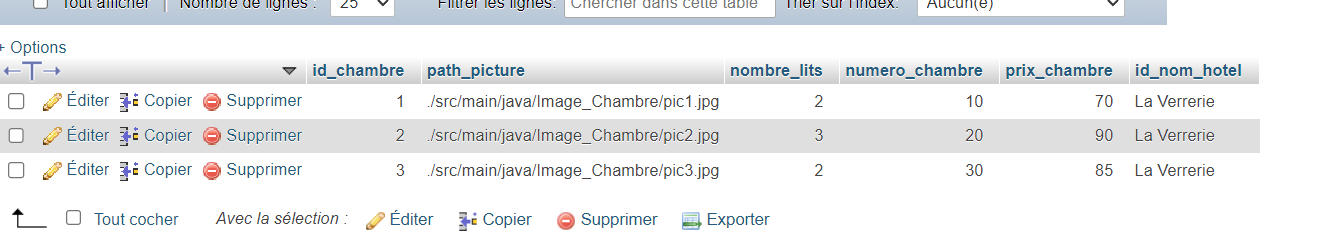
Puis on retourne une string qui renvoie soit les détails de la réservation soit une erreur si une réservation était déjà faite pour ces dates là.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

QUESTION 2 : CONVERSION ET INTÉGRATION D’IMAGE

Pour que le client puisse recevoir une image de la chambre j’ai implémentée et modifiée ma solution la méthode réalisé pour le projet SOAP avec Christophe Houbron. Cela consiste à récupérer le path de la photo côté serveur et de la transformer en List de ByteCode.

De ce fait et partant du postulat qu’il ne faut pas enregistrer des jpegs dans une base de données pour une question de vitesse d’exécution j’ai décidé d’intégrer seulement dans les chambres le chemin d’accès (path) comme ceci :



Ensuite pendant l’exécution de mon algorithme de consultation je peux convertir les paths en ByteCode pour transferable en XML il me suffit simplement de créer un attribut byteCode et une méthode de conversion du path dans ma classe Offre :

**public** **class** Offre **extends** Chambre {

//Attributs

….

**private** **byte**[] picOfChamber;

//Constructeur

**public** Offre(**float** coeffPromotion, Chambre chambre) **throws** IOException {

**super**(….)

….

setPicOfChamberByPathPicture(chambre.getPicOfChamberByPathPicture());

}

//Méthodes

**public** **void** setPicOfChamberByPathPicture(String pathPicure) **throws** IOException {

File fichier = **new** File(pathPicure);

BufferedImage fileToByteCode = ImageIO.*read*(fichier);

ByteArrayOutputStream ByteForXML = **new** ByteArrayOutputStream();

ImageIO.*write*(fileToByteCode, "jpg", ByteForXML);

**this**.picOfChamber = ByteForXML.toByteArray();

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Ainsi le corps de ma requête http get contiendra du byteCode :

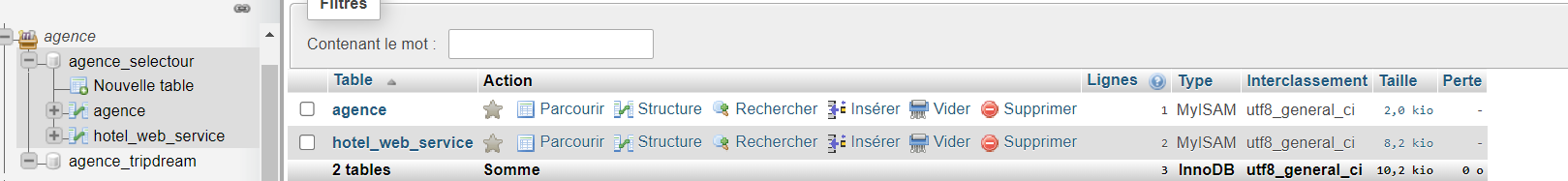
Une image contenant texte

Description générée automatiquement

QUESTION 3 : COMPARATEUR SERVICE WEB :

@GetMapping(***uri***+"/offre/ville={ville}/date\_arrivee={dateArrivee}/date\_depart={dateDepart}/nbPersonne={nbPersonne}/etoiles={etoiles}")

Ici c’est le comparateur qui sera le client et utilisateur final des données, transmise par l’agence, ce qui veut dire qu’il faut convertir l’agence comme un client/serveur. En effet le client va utilisée à la fois le webservice des hotels donc il sera client des hotels, et il sera serveur car il va projeter son propre serviceWeb.

Pour cela on modife les projets agences en leurs créeant une base de donnée : 

La table agence va contenir toute les informations d’une agence, et la table hotel\_web\_service l’url des webservices des différents hotels.

Pour faire simple on modifie le service web consultation de la question des hotels en leur retirant les paramètres de pays et du prix.

@GetMapping(***uri***+"/hotel/consultation/id\_agence={identifiantAgence}/password\_agence={passwordAgence}/date\_arrivee={dateArrivee}/date\_depart={dateDepart}"+"/nbPersonne={nbPersonne}/ville={ville}/etoiles={etoile

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

Ceci vient changer légèrement la requête de recherche d’hotel et des chambres comme suivant :

-Pour hotel :

Optional<Hotel> optHotel hotelService.getHotelByVilleAndEtoiles(ville, etoiles)

Pour la requête de recherche des disponibilités dans le repository des chambres :

"SELECT \* FROM chambre WHERE chambre.id\_chambre NOT IN "

+ "(SELECT reservation.id\_chambre FROM reservation WHERE reservation.date\_depart > :dateArrivee AND reservation.date\_arrivee < :dateDepart)"

+ "AND nombre\_lits >= :nbPersonne".

Voici la trace de l’appel des différents service web avec l’agence Selectour :

L’utilisateur va entrée toutes informations dans la CLI du Comparateur, qui va appeler le service web d’une agence avec l’url suivant (méthode GET) .

<http://localhost:6666/agence/api/offre/ville=Gaillac/date_arrivee=2023-02-17/date_depart=2023-02-18/nbPersonne=2/etoiles=3>

L’agence va ensuite récupérer l’identification de l’agence dans son webService (AgenceController):

Optional<Agence> optAgence = agenceService.getAgence();

Agence agence = optAgence.get();

ArrayList<Offre> listOffreDispo = **new** ArrayList<>();

List<String> listHotelWebServiceURL = **new** ArrayList<>();

listHotelWebServiceURL = hotelWebServiceURLService.getAllURL();

Puis pour chaque URL le programme va appeler le webService des hotels enrichie des données d’identifiant de l’agence et du motdepasse de celle-ci pour pouvoir calculer le prix avec réduction des chambres :

**for** (String hotelWebServiceURL : listHotelWebServiceURL) {

String uri = hotelWebServiceURL+"consultation/id\_agence="+agence.getIdentifiant()+"/password\_agence="+agence.getPassword()+"/date\_arrivee="+dateArrivee+"/date\_depart="+dateDepart+"/nbPersonne="+nbPersonne+"/ville="+ville+"/etoiles="+etoiles;

Exemple pour l’hotel de Gaillac :

http://localhost:8080/hotelservice/api/hotel/consultation/id\_agence=1/password\_agence=selectour/date\_arrivee=2023-02-17/date\_depart=2023-02-18/nbPersonne=2/ville=Gaillac/etoiles=3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

Les données récupérées seront par exemples les suivantes :

    {

        "idChambre": 0,

        "numero": 10,

        "nbLits": 2,

        "prixChambre": 70.0,

        "listReservation": [],

        "disponibilitee": [

            "14-01-2023",

            "15-01-2023",

            "17-01-2023",

            "15-01-2024"

        ],

        "prixChambreWithReduction": 69.0,

        "identifiant": 1,

        "nomhotel": "La Verrerie",

        "adressehotel": "1 Rue de l'Égalité",

        "etoileshotel": 3,

        "picOfChamber": **…. Volontairement Réduit pour l’exemple car les données sont longues.**

    },

Il faut donc enrichir la réponse des webServices des hotels avec le nom de l’agence comme suivant :

**if** (Arrays.*asList*(listOffre).isEmpty()) {

System.***out***.println("Pour "+ hotelWebServiceURL+"nous n'avons rien trouvé");

System.***out***.println();

}**else** {

Arrays.*asList*(listOffre).forEach(System.***out***::println);

**for** (Offre offre : Arrays.*asList*(listOffre)) {

System.***out***.println(offre);

offre.setNomAgence(agence.getNom());

listOffreDispo.add(offre);

}

}

.

Algorithme Consultation CLASS OFFRECONTROLLER :

@RestController

    @GetMapping(uri+"/hotel/consultation/id\_agence={identifiantAgence}/password\_agence={passwordAgence}/date\_arrivee={dateArrivee}/date\_depart={dateDepart}"

            + "/nbPersonne={nbPersonne}/pays={pays}/ville={ville}/etoiles={etoiles}/prixMin={prixMin}/prixMax={prixMax}")

    public ArrayList<Offre> consultationOffre(@PathVariable int identifiantAgence, @PathVariable String passwordAgence,

            @PathVariable int nbPersonne, @PathVariable String dateArrivee, @PathVariable String dateDepart,

            @PathVariable String pays, @PathVariable String ville, @PathVariable int etoiles,

            @PathVariable float prixMin, @PathVariable float prixMax) throws IOException{

        float coeffPromotion;

        ArrayList<Offre> listOffreDispo = new ArrayList<>();

        Optional<Hotel> optHotel = hotelService.getHotelByPaysAndVilleAndEtoiles(pays, ville, etoiles);

        if (optHotel.isPresent()) {

            Hotel hotelPresent = optHotel.get();

            Optional<AgencePartenaire> filterAgencePartenaire = hotelPresent.getAgencesPartenaire().stream()

                    .filter(agencePartenaire -> agencePartenaire.getIdAgence() == identifiantAgence

                    && agencePartenaire.getAgencePartenairePKID().getPasswordAgence().equals(passwordAgence))

                    .findFirst();

            if (filterAgencePartenaire.isPresent()) {

                coeffPromotion = filterAgencePartenaire.get().getCoeffPromotion();

            }else {

                coeffPromotion = 1;

            }

            List<Chambre> listChambreDispo = chambreService.findChambreByDisponbilitee(dateArrivee, dateDepart, coeffPromotion, prixMin, prixMax, nbPersonne);

            for (Chambre chambre : listChambreDispo) {

                Offre offre = new Offre(coeffPromotion, chambre);

                listOffreDispo.add(offre);

            }

        }

        System.out.println(listOffreDispo.toString());

        return listOffreDispo;

    }

Algorithme Reservation :

@PostMapping(uri+"/reservation")

    public String doReservation(@RequestBody MultiValueMap<String, String> reservation) {

            String response = "ok";

//      Test Agence

            if (reservationService.findIfReservationIsPossible(reservation.getFirst("dateArrivee"),

                    reservation.getFirst("dateDepart")).isEmpty()) {

                Reservation reservationToSave = new Reservation(reservation.getFirst("dateArrivee"),

                        reservation.getFirst("dateDepart"), Integer.parseInt(reservation.getFirst("nbLitsResa")));

                Client clientToSave = new Client(reservation.getFirst("prenomClient"),reservation.getFirst("nomClient"));

                if (!carteBancaireService.findCarteBancaireByNumeroANDcryptogrammeANDdateExpiration(reservation.getFirst("numeroCarte"), reservation.getFirst("cardDateExpiration"),

                        Integer.parseInt(reservation.getFirst("cryptogramme"))).isPresent()) {

                    CarteBancaire cbToSave = new CarteBancaire(reservation.getFirst("numeroCarte"),reservation.getFirst("cardDateExpiration"),

                            Integer.parseInt(reservation.getFirst("cryptogramme")));

                    clientToSave.setCarteBancaire(cbToSave);

                    cbToSave.setClient(clientToSave);

                    carteBancaireService.createCarteBancaire(cbToSave);

                } else {

                    CarteBancaire cbAlreadyBind = carteBancaireService.findCarteBancaireByNumeroANDcryptogrammeANDdateExpiration(

                                    reservation.getFirst("numeroCarte"), reservation.getFirst("cardDateExpiration"),

                                    Integer.parseInt(reservation.getFirst("cryptogramme"))).get();

                    clientToSave.setCarteBancaire(cbAlreadyBind);

                    clientService.createClient(clientToSave);

                }

                reservationToSave.setClient(clientToSave);

                reservationToSave.setChambre(chambreService

                        .findChambreById(Integer.parseInt(reservation.getFirst("identifiantOffre"))).get());

                reservationService.createReservation(reservationToSave);

                response = reservationToSave.toString();

            } else {

                response = "Erreur l'offre est déjà réservée pour les dates demandées";

            }

            return response;

        }

Algorithme COMPARATEUR

Côté Agence

@GetMapping(***uri***+"/offre/ville={ville}/date\_arrivee={dateArrivee}/date\_depart={dateDepart}"

+ "/nbPersonne={nbPersonne}/etoiles={etoiles}")

**public** ArrayList<Offre> consultationOffre(@PathVariable **int** nbPersonne, @PathVariable String dateArrivee, @PathVariable String dateDepart,

@PathVariable String ville, @PathVariable **int** etoiles) **throws** IOException{

//Je récupère mon agence

Optional<Agence> optAgence = agenceService.getAgence();

Agence agence = optAgence.get();

ArrayList<Offre> listOffreDispo = **new** ArrayList<>();

List<String> listHotelWebServiceURL = **new** ArrayList<>();

listHotelWebServiceURL = hotelWebServiceURLService.getAllURL();

**for** (String hotelWebServiceURL : listHotelWebServiceURL) {

String uri = hotelWebServiceURL+"consultation/id\_agence="+agence.getIdentifiant()+"/password\_agence="+agence.getPassword()+"/date\_arrivee="+dateArrivee

+"/date\_depart="+dateDepart+"/nbPersonne="+nbPersonne+"/ville="+ville+"/etoiles="+etoiles;

Offre [] listOffre = proxy.getForObject(uri, Offre[].**class**);

//J'ajoute le résultat à l'arraylist

System.***out***.println("test");

System.***out***.println(listOffre.toString());

**if** (Arrays.*asList*(listOffre).isEmpty()) {

System.***out***.println("Pour "+ hotelWebServiceURL+"nous n'avons rien trouvé");

System.***out***.println();

}**else** {

Arrays.*asList*(listOffre).forEach(System.***out***::println);

**for** (Offre offre : Arrays.*asList*(listOffre)) {

System.***out***.println(offre);

offre.setNomAgence(agence.getNom());

listOffreDispo.add(offre);

}

}

//Pour chaque résultat je rajoute les informations manquantes.

}

**return** listOffreDispo;

}

Côté Hotel les seuls changement notable par rapport à consultation sont les paramètres de recherche d’hotel :

Optional<Hotel> optHotel = hotelService.getHotelByVilleAndEtoiles(ville, etoiles);

Et de la disponibilitée des chambres :

List<Chambre> listChambreDispo = chambreService.findChambreByDisponbilitee(dateArrivee, dateDepart, nbPersonne);

Resultat Question 1 Consultation :

NOTA BENE le nom des villes et pays des hotels est récupérée par l’appel d’un webService respectif pour chacune des informations.

Une image contenant texte

Description générée automatiquement

Resultat Question 1 Reservation :

Une image contenant texte

Description générée automatiquement

Resultat Question 2 :

Une image contenant texte, intérieur

Description générée automatiquement

Resultat Question 3 :

