

RAPPORT DE PROJET

SYSTÈME DE GESTION DE
RESERVATION D'HÔTEL

Mai

2024



INTRODUCTION

Introduction

Le contexte

Les informations circulantes dans l'hôtel sont de nature diverse, elles concernent le client depuis son arrivée jusqu'à son départ de l'hôtel, et pour le bon fonctionnement de travail, nous avons pensé à mettre en place un système informatique qui facilitera les tâches de gestion de l'hôtel. L'application à mettre en oeuvre permettra une interaction rigoureuse entre le réceptionniste, le gérant et les clients, en assurant ainsi une bonne gestion de l'hôtel.

Les réservations à l'hôtel sont ouvertes pendant toute l'année aux clients étrangers voulant séjourner et le nombre de ces derniers augmente d'année en année ce qui engendre cumul d'information, les responsables trouvent des difficultés dans leurs travaux, le taux d'erreur ne cesse d'accroître et certaines tâches font qu'elles sont pénibles

à traiter, dont on souligne les problèmes suivants :

- Perte des documents.
- Toutes les procédures sont faites manuellement.
- Mauvais archivage des documents.
- Difficulté et retard lors de la recherche de l'information.
- Perte de temps.

Notre travail vise à réaliser une application réseau pour la gestion d'un hôtel qui consiste à gérer les clients, les chambres, les services, les factures, ainsi que les comptes, qui seront soumis à un seul acteur. Afin d'atteindre les objectifs, l'application à concevoir offrira les espaces suivant :

■ Un espace réservé à l'administrateur de l'application qui lui permet la gestion des clients, la gestion des chambres, la gestion des services, la gestion des factures et la mise à jour de ces derniers (l'ajout, modification, suppression).

■ Un espace réservé aux autres utilisateurs de l'application (employés de l'hôtel par exemple le réceptionniste, ..etc) après avoir un compte (login et mot de passe) configuré par l'administrateur et des privilèges pour avoir accès à leurs fonctionnalités.





Introduction

Besoins fonctionnels

Les besoins fonctionnels se présentent en six (6) grandes parties :

1. Authentification : Cette interface permet à l'utilisateur d'accéder à son interface après une authentification par un login et un mot de passe.
2. Effectuer une recherche : Cette interface permet à un utilisateur d'effectuer une recherche avec la saisie d'un mot clé concernant le sujet de la recherche.
3. Gestion des utilisateurs : Gestion des utilisateurs
4. Gestion des clients : Cette interface offre à l'utilisateur la possibilité de gérer les différents clients de notre application c'est-à-dire de faire des mises à jour sur ces derniers (l'ajout, la modification ou la suppression).
5. Gestion des factures : Cette interface offre à l'utilisateur la possibilité de gérer les différentes factures de l'hôtel c'est-à-dire de faire des mises à jour sur ces derniers (l'ajout, la modification ou la suppression).
6. Gestion des réservations : Cette interface offre à l'utilisateur la possibilité de gérer les différentes réservations de l'hôtel effectuées par les clients c'est-à-dire de faire des mises à jour (l'ajout, la modification ou la suppression) sur ces réservations.
6. Gestion des chambres : Cette interface offre à l'administrateur la possibilité de gérer les différentes chambres d'hôtel c'est-à-dire de faire des mises à jour sur ces derniers (l'ajout, la modification ou la suppression).





Introduction

Besoins non fonctionnels

Les besoins non fonctionnels sont importants car ils agissent de façon indirecte sur le résultat

et sur le rendement de l'utilisateur, ce qui fait qu'ils ne doivent pas être négligés, pour cela il faut répondre aux exigences suivantes :

1. **Fiabilité** : L'application doit fonctionner de façon cohérente sans erreurs et doit être satisfaisante.
2. **Les erreurs** : Les ambiguïtés doivent être signalées par des messages d'erreurs bien organisés pour bien guider l'utilisateur et le familiariser avec notre application.
3. **Ergonomie et bonne Interface** : L'application doit être adaptée à l'utilisateur sans qu'il ne fournisse aucun effort (utilisation claire et facile) de point de vue navigation entre les différentes pages, couleurs et mise en textes utilisés.
4. **Sécurité** : Notre solution doit respecter surtout la confidentialité des données personnelles des clients qui reste l'une des contraintes les plus importantes dans les applications.
5. **Aptitude à la maintenance et la réutilisation** : Le système doit être conforme à une architecture standard et claire permettant sa maintenance et sa réutilisation.
6. **Compatibilité et portabilité** : Une application quel que soit son domaine, son éditeur et son langage de programmation ne peut être fiable qu'avec une compatibilité avec toutes les plateformes et tous les systèmes.

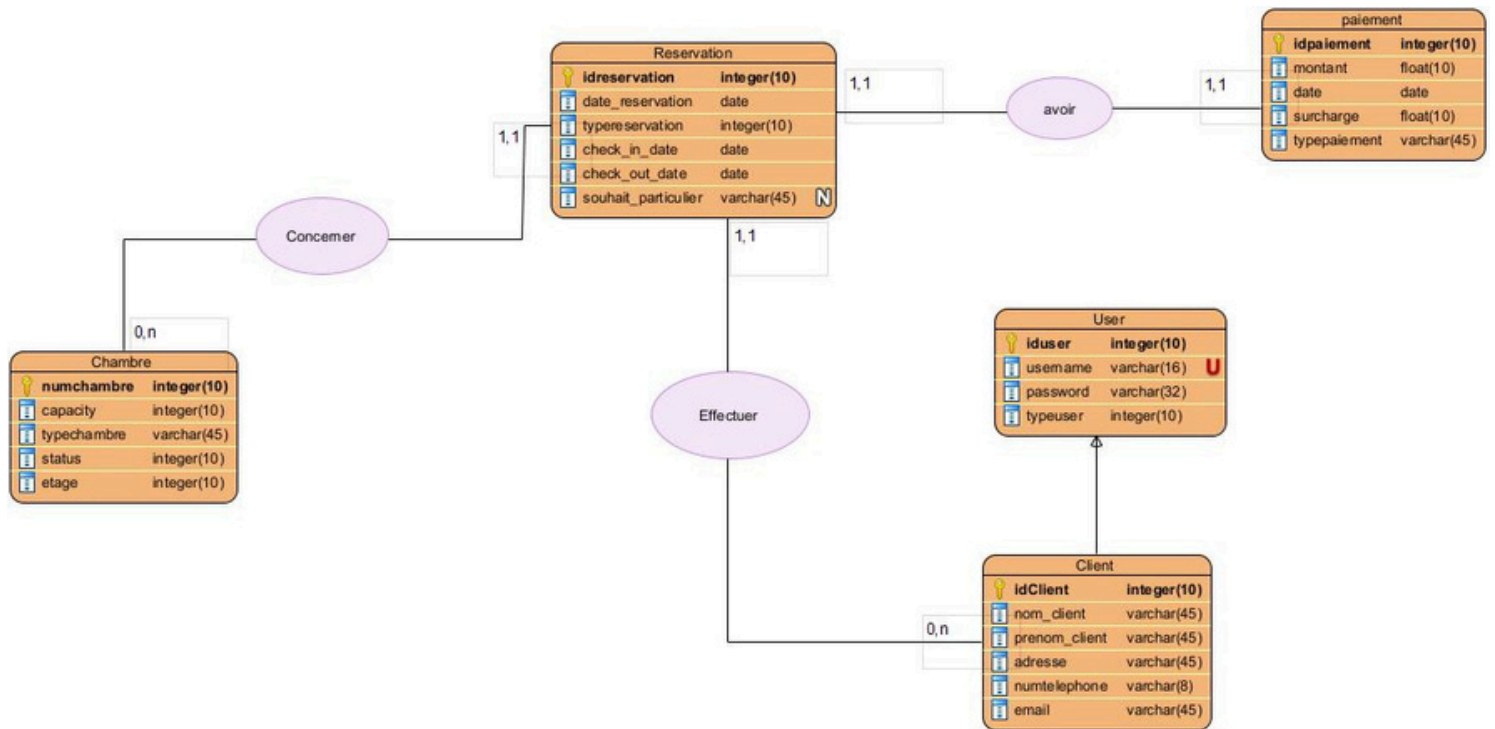


02



**ANALYSE DES
BESOINS**

Modèle entité association / modèle Relationnel



Modèle Relationnel :

- > Chambre (numchambre, capacity , typechambre , status , etage)
- > Reservation(idreservation , date_reservation , type reservation , check_in_date , check_out_date,souhait_particulier , #idchambretereservant)
- > Paiement(idpaiement , montant , date ,surcharge , typepaiement , #idreservation)
- > Client(idClient, nom_client, prenom_client,adresse,numtelephone,email)
- > User(iduser , username , password , typeuser,#idclient)



Requêtes SQL:

Création de la base de données :

```
CREATE DATABASE IF NOT EXISTS hotel;  
USE hotel;
```

"CREATE DATABASE IF NOT EXISTS hotel;" : crée une base de données nommée "hotel" si elle n'existe pas déjà.

"USE hotel;" spécifie que les opérations suivantes seront effectuées dans le contexte de la base de données "hotel".

```
DROP TABLE IF EXISTS `client`;  
CREATE TABLE `client` (  
  `idclient` int NOT NULL AUTO_INCREMENT,  
  `nom_client` varchar(45) NOT NULL,  
  `prenom_client` varchar(45) NOT NULL,  
  `adresse` varchar(45) DEFAULT NULL,  
  `num_tel` varchar(8) DEFAULT NULL,  
  `email` varchar(45) NOT NULL,  
  PRIMARY KEY (`idclient`)  
);
```




```
-- Supprime la table `client` si elle existe déjà
DROP TABLE IF EXISTS `client`;

-- Crée une nouvelle table `client`
CREATE TABLE `client` (
  -- Colonne pour l'ID client, auto-incrémentée
  `idclient` int NOT NULL AUTO_INCREMENT,

  -- Colonne pour le nom du client, ne peut pas être vide
  `nom_client` varchar(45) NOT NULL,

  -- Colonne pour le prénom du client, ne peut pas être
  vide
  `prenom_client` varchar(45) NOT NULL,


  -- Colonne pour l'adresse du client, peut être NULL par
  défaut
  `adresse` varchar(45) DEFAULT NULL,

  -- Colonne pour le numéro de téléphone du client, peut
  être NULL par défaut
  `num_tel` varchar(8) DEFAULT NULL,

  -- Colonne pour l'adresse e-mail du client, ne peut pas
  être vide
  `email` varchar(45) NOT NULL,

  -- Définit la clé primaire de la table sur la colonne
  `idclient`
  PRIMARY KEY (`idclient`)
);
```





```
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `iduser` int NOT NULL AUTO_INCREMENT,
  `username` varchar(16) NOT NULL,
  `password` varchar(32) NOT NULL,
  `idclient` int DEFAULT NULL,
  `typeuser` int NOT NULL,
  PRIMARY KEY (`iduser`),
  KEY `idclient_idx` (`idclient`),
  CONSTRAINT `idclient` FOREIGN KEY (`idclient`)
REFERENCES `client` (`idclient`)
);
```



```
-- Crée une nouvelle table `user`
CREATE TABLE `user` (
  -- Colonne pour l'ID utilisateur, auto-incrémentée
  `iduser` int NOT NULL AUTO_INCREMENT,

  -- Colonne pour le nom d'utilisateur, ne peut pas être
  vide
  `username` varchar(16) NOT NULL,

  -- Colonne pour le mot de passe, ne peut pas être vide
  `password` varchar(32) NOT NULL,

  -- Colonne pour l'ID client, peut être NULL par défaut
  `idclient` int DEFAULT NULL,


  -- Colonne pour le type d'utilisateur, ne peut pas être
  vide
  `typeuser` int NOT NULL,

  -- Définit la clé primaire de la table sur la colonne
  `iduser`
  PRIMARY KEY (`iduser`),

  -- Crée un index sur la colonne `idclient`
  KEY `idclient_idx` (`idclient`),

  -- Définit une contrainte de clé étrangère sur la colonne
  `idclient`, faisant référence à la table `client`
  CONSTRAINT `idclient` FOREIGN KEY (`idclient`)
  REFERENCES `client` (`idclient`)
);
```





```
DROP TABLE IF EXISTS `chambre`;
CREATE TABLE `chambre` (
  `numchambre` int NOT NULL,
  `capacity` int NOT NULL,
  `typechambre` varchar(45) NOT NULL,
  `status` int NOT NULL,
  `etage` int NOT NULL,
  PRIMARY KEY (`numchambre`)
);
```

```
-- Supprime la table `chambre` si elle existe déjà
DROP TABLE IF EXISTS `chambre`;
```

```
-- Crée une nouvelle table `chambre`
```

```
CREATE TABLE `chambre` (
  -- Colonne pour le numéro de chambre, qui ne peut pas être vide
  `numchambre` int NOT NULL,
```

```
  -- Colonne pour la capacité de la chambre, qui ne peut pas être
  vide
```

```
  `capacity` int NOT NULL,
```

```
  -- Colonne pour le type de chambre, qui ne peut pas être vide
```

```
  `typechambre` varchar(45) NOT NULL,
```

```
  -- Colonne pour le statut de la chambre, qui ne peut pas être vide
```

```
  `status` int NOT NULL,
```

```
  -- Colonne pour l'étage où se trouve la chambre, qui ne peut pas
  être vide
```


```
  `etage` int NOT NULL,
```

```
  -- Définit la clé primaire de la table sur la colonne `numchambre`
```

```
  PRIMARY KEY (`numchambre`)
```


```
)
```





```
DROP TABLE IF EXISTS `reservation`;
CREATE TABLE `reservation` (
  `idreservation` int NOT NULL AUTO_INCREMENT,
  `id_client_reservant` int NOT NULL,
  `date_reservation` date NOT NULL,
  `typereservation` int NOT NULL,
  `check_in_date` date NOT NULL,
  `check_out_date` date NOT NULL,
  `souhait_particulier` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`idreservation`),
  KEY `id_client_reservant_idx` (`id_client_reservant`),
  CONSTRAINT `id_client_reservant` FOREIGN KEY
(`id_client_reservant`) REFERENCES `client` (`idclient`)
);
```





```
-- Supprime la table `reservation` si elle existe déjà
DROP TABLE IF EXISTS `reservation`;
```

```
-- Crée une nouvelle table `reservation`
```

```
CREATE TABLE `reservation` (
```

```
-- Colonne pour l'ID de la réservation, auto-incrémentée
`idreservation` int NOT NULL AUTO_INCREMENT,
```

```
-- Colonne pour l'ID du client effectuant la réservation,
qui ne peut pas être vide
```

```
`id_client_reservant` int NOT NULL,
```

```
-- Colonne pour la date de la réservation, qui ne peut pas
être vide
```

```
`date_reservation` date NOT NULL,
```

```
-- Colonne pour le type de réservation, qui ne peut pas être
vide
```

```
`typereservation` int NOT NULL,
```

```
-- Colonne pour la date d'arrivée, qui ne peut pas être vide
```

```
`check_in_date` date NOT NULL,
```

```
-- Colonne pour la date de départ, qui ne peut pas être
vide
```

```
`check_out_date` date NOT NULL,
```





-- Colonne pour les demandes particulières, qui peut être NULL par défaut

`souhait_particulier` varchar(45) DEFAULT NULL,

-- Définit la clé primaire de la table sur la colonne
`idreservation`


PRIMARY KEY (`idreservation`),

-- Crée un index sur la colonne `id_client_reservant`
KEY `id_client_reservant_idx` (`id_client_reservant`),

-- Définit une contrainte de clé étrangère sur la colonne
`id_client_reservant`, faisant référence à la table `client`

CONSTRAINT `id_client_reservant` FOREIGN KEY
(`id_client_reservant`) REFERENCES `client` (`idclient`)
);





```
DROP TABLE IF EXISTS `res_cham`;
CREATE TABLE `res_cham` (
  `idreservation` int NOT NULL,
  `numchambre` int NOT NULL,
  PRIMARY KEY (`idreservation`,`numchambre`),
  KEY `fk_1Res_Cham_chambre1_idx` (`numchambre`),
  CONSTRAINT `fk_1Res_Cham_chambre1` FOREIGN KEY
(`numchambre`) REFERENCES `chambre` (`numchambre`),
  CONSTRAINT `fk_1Res_Cham_reservation1` FOREIGN KEY
(`idreservation`) REFERENCES `reservation` (`idreservation`)
);
```





-- Supprime la table `res_cham` si elle existe déjà

DROP TABLE IF EXISTS `res_cham`;

-- Crée une nouvelle table `res_cham`

CREATE TABLE `res_cham` (

-- Colonne pour l'ID de la réservation, qui ne peut pas être vide

`idreservation` int NOT NULL,

-- Colonne pour le numéro de chambre, qui ne peut pas être vide

`numchambre` int NOT NULL,

-- Définit la clé primaire de la table sur les colonnes

`idreservation` et `numchambre`

PRIMARY KEY (`idreservation`,`numchambre`),

-- Crée un index sur la colonne `numchambre`

KEY `fk_📄Res_Cham_chambre1_idx` (`numchambre`),

-- Définit une contrainte de clé étrangère sur la colonne

`numchambre`, faisant référence à la table `chambre`


CONSTRAINT `fk_📄Res_Cham_chambre1` FOREIGN KEY
(`numchambre`) REFERENCES `chambre` (`numchambre`),

-- Définit une contrainte de clé étrangère sur la colonne

`idreservation`, faisant référence à la table `reservation`

CONSTRAINT `fk_📄Res_Cham_reservation1` FOREIGN KEY
(`idreservation`) REFERENCES `reservation` (`idreservation`)
);





```
DROP TABLE IF EXISTS `paiement`;
CREATE TABLE `paiement` (
  `idpaiement` int NOT NULL AUTO_INCREMENT,
  `idreservation` int NOT NULL,
  `montant` float NOT NULL,
  `date` date NOT NULL,
  `surcharge` float DEFAULT NULL,
  `typepaiement` varchar(45) NOT NULL,
  PRIMARY KEY (`idpaiement`),
  KEY `idreservation_idx` (`idreservation`)
);
ALTER TABLE `paiement` ADD CONSTRAINT `idreservation`
FOREIGN KEY (`idreservation`) REFERENCES `reservation`
(`idreservation`);
```





-- Supprime la table `paiement` si elle existe déjà

DROP TABLE IF EXISTS `paiement`;

-- Crée une nouvelle table `paiement`

CREATE TABLE `paiement` (

-- Colonne pour l'ID du paiement, auto-incrémentée

`idpaiement` int NOT NULL AUTO_INCREMENT,

-- Colonne pour l'ID de la réservation associée au
paiement, qui ne peut pas être vide

`idreservation` int NOT NULL,

-- Colonne pour le montant du paiement, qui ne peut pas
être vide

`montant` float NOT NULL,

-- Colonne pour la date du paiement, qui ne peut pas être
vide

`date` date NOT NULL,

-- Colonne pour les frais de surcharge, qui peut être NULL
par défaut

`surcharge` float DEFAULT NULL,





```
-- Colonne pour le type de paiement, qui ne peut pas être vide
```

```
`typepaiement` varchar(45) NOT NULL,
```

```
-- Définit la clé primaire de la table sur la colonne  
`idpaiement`
```

```
PRIMARY KEY (`idpaiement`),
```

```
-- Crée un index sur la colonne `idreservation`
```

```
KEY `idreservation_idx` (`idreservation`)
```

```
);
```

```
-- Ajoute une contrainte de clé étrangère sur la colonne  
`idreservation`, faisant référence à la table `reservation`  
ALTER TABLE `paiement` ADD CONSTRAINT `idreservation`  
FOREIGN KEY (`idreservation`) REFERENCES `reservation`  
(`idreservation`);
```



PL/SQL:

Procédure pour trouver les chambres disponibles en fonction des critères spécifiés et affiche les résultats.

```
CREATE OR REPLACE PROCEDURE chambres_disponibles(
    check_in IN DATE,
    check_out IN DATE,
    nb_personnes IN NUMBER
) IS
    chambre_dispo chambre%ROWTYPE;
    cur_count NUMBER := 0;
    CURSOR cur IS
        SELECT *
        FROM chambre
        WHERE capacite >= nb_personnes
        AND numchambre NOT IN (
            SELECT id_chambre_reserve
            FROM reservation
            WHERE (check_in_date BETWEEN check_in AND
check_out)
                OR (check_out_date BETWEEN check_in AND
check_out)
                OR (check_in BETWEEN check_in_date AND
check_out_date)
                OR (check_out BETWEEN check_in_date AND
check_out_date)
        );
BEGIN
    -- Vérification des données
    IF check_in IS NULL OR check_out IS NULL OR
nb_personnes IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Tous les
paramètres doivent être renseignés.');
```

→

```
        END IF;

        IF check_in >= check_out THEN
            RAISE_APPLICATION_ERROR(-20002, 'La date de fin
doit être postérieure à la date de début.');
```

```

IF nb_personnes <= 0 THEN
    RAISE_APPLICATION_ERROR(-20003, 'Le nombre de
    personnes doit être supérieur à 0. ');
END IF;

-- Traitement
OPEN cur;
LOOP
    FETCH cur INTO chambre_dispo;
    EXIT WHEN cur%NOTFOUND;
    -- Afficher les informations sur la chambre
    DBMS_OUTPUT.PUT_LINE('Chambre : ' ||
    chambre_dispo.numchambre || ', Capacité : ' ||
    chambre_dispo.capacite || ', Type : ' ||
    chambre_dispo.typechambre || ', Étage : ' ||
    chambre_dispo.etape || ', Prix par jour : ' ||
    chambre_dispo.prix_par_jour);
    cur_count := cur_count + 1;
END LOOP;
CLOSE cur;

-- Vérifier si aucune chambre n'est disponible
IF cur_count = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Aucune chambre disponible
    pour les critères spécifiés. ');
END IF;
END chambres_disponibles;

```



1. Elle commence par **déclarer** une variable `chambre_dispo` de type `chambre%ROWTYPE` pour stocker les informations d'une chambre et une variable `cur_count` pour compter le nombre de chambres disponibles.
2. Ensuite, elle **définit un curseur** nommé `cur` qui sélectionne toutes les chambres qui ont une capacité supérieure ou égale au nombre de personnes spécifié et qui ne sont pas réservées pendant la période donnée.
3. Les **vérifications** des données d'entrée sont effectuées :
 - Si l'une des données d'entrée est nulle, elle lève une erreur.
 - Si la date de début est postérieure ou égale à la date de fin, elle lève une erreur.
 - Si le nombre de personnes est inférieur ou égal à zéro, elle lève une erreur.
4. Le curseur est ouvert.
5. Une boucle **LOOP est utilisée pour parcourir les résultats du curseur**.
 - Pour chaque chambre disponible, elle affiche les informations de la chambre à l'aide de la fonction `DBMS_OUTPUT.PUT_LINE`.
 - Le compteur `cur_count` est incrémenté à chaque itération.
6. Après avoir parcouru toutes les chambres disponibles, le curseur est fermé.
7. Enfin, elle **vérifie** si des chambres ont été trouvées (`cur_count > 0`). Si aucune chambre n'est disponible, un message est affiché pour en informer.

déclencheur garantit l'intégrité des données en s'assurant qu'une chambre n'est pas réservée pour des périodes qui se chevauchent avec d'autres réservations existantes.


```
CREATE OR REPLACE TRIGGER
check_room_availability
BEFORE INSERT OR UPDATE ON reservation
FOR EACH ROW
DECLARE
    v_overlap_count NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_overlap_count
    FROM reservation
    WHERE id_chambre_reserve =
:new.id_chambre_reserve
    AND (
        (:new.check_in_date BETWEEN check_in_date
AND check_out_date)
        OR (:new.check_out_date BETWEEN
check_in_date AND check_out_date)
        OR (check_in_date BETWEEN :new.check_in_date
AND :new.check_out_date)
        OR (check_out_date BETWEEN
:new.check_in_date AND :new.check_out_date)
    )
    AND idreservation <> NVL(:old.idreservation, 0); --
-- Pour exclure la propre réservation lors de la mise à
jour
```



```
IF v_overlap_count > 0 THEN
    RAISE_APPLICATION_ERROR(-20004, 'La
chambre est déjà réservée pour cette période.');
```

END IF;

END;



1. Lorsqu'une nouvelle réservation est insérée (**INSERT**) ou qu'une réservation existante est mise à jour (**UPDATE**), ce déclencheur est déclenché pour chaque ligne concernée.

2. Dans la section déclarative du déclencheur, une variable **v_overlap_count est déclarée pour stocker le nombre de réservations** qui se chevauchent avec la nouvelle réservation à ajouter ou à mettre à jour.

3. Une requête `SELECT COUNT(*) INTO v_overlap_count` est utilisée pour **compter le nombre de réservations existantes qui ont des dates de chevauchement avec la nouvelle réservation**. La clause `WHERE` de la requête sélectionne les réservations pour la même chambre que la nouvelle réservation et qui ont des dates de chevauchement avec la nouvelle réservation.

4. **Les dates de chevauchement sont vérifiées de quatre manières différentes :**

- La date de début de la nouvelle réservation tombe entre la date de début et la date de fin d'une réservation existante.
- La date de fin de la nouvelle réservation tombe entre la date de début et la date de fin d'une réservation existante.
- La date de début d'une réservation existante tombe entre la date de début et la date de fin de la nouvelle réservation.
- La date de fin d'une réservation existante tombe entre la date de début et la date de fin de la nouvelle réservation.



5. La condition AND idreservation <>

NVL(:old.idreservation, 0) est utilisée pour exclure la propre réservation lors de la mise à jour. Cela évite de considérer la réservation en cours de mise à jour comme une réservation qui se chevauche avec elle-même.

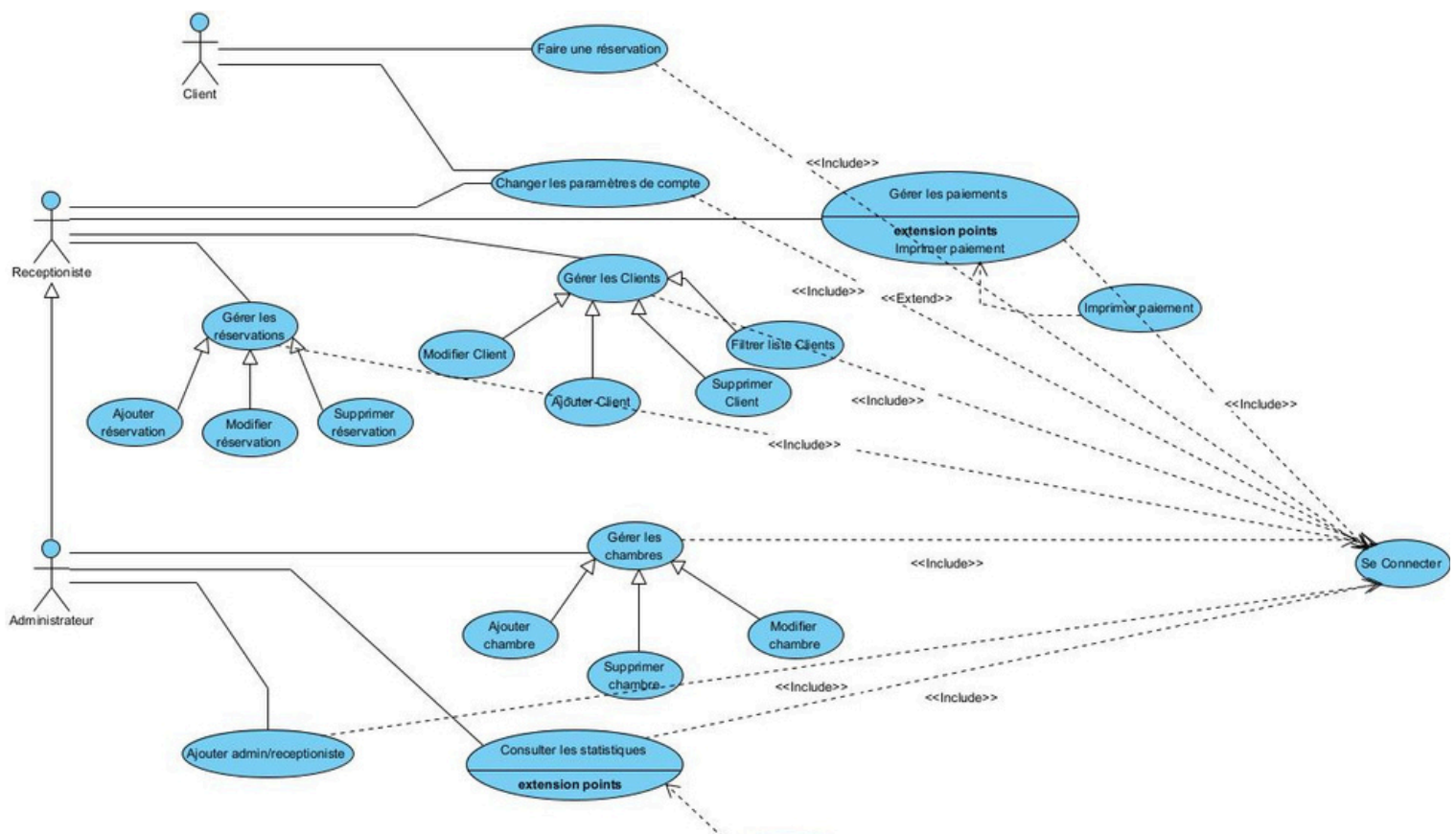
6. Si **v_overlap_count** est supérieur à zéro, cela signifie qu'une chambre est déjà réservée pour la période spécifiée, et une erreur est levée à l'aide de **RAISE_APPLICATION_ERROR** pour indiquer que la chambre n'est pas disponible pour cette période.



03

**ANALYSE DE
DOMAINE ET
CONCEPTION**

Diagramme de cas d'utilisation



Analyse de diagramme de cas d'utilisation

Les cas d'utilisations sont les outils formels qui permettent de consigner et d'examiner les interactions et les dialogues des utilisateurs (acteurs) avec le système. Un cas d'utilisation est une narration qui décrit un scénario appliqué à une utilisation particulière, dans lequel un acteur fournit des entrées et pour lequel le système produit une sortie observable. Un cas d'utilisation doit exprimer ce que doit faire un acteur, sans préjuger de la façon dont cela sera fait, il décrit le comportement du système vu de l'extérieure.

Cas d'utilisation	S'authentifier
Acteurs	Utilisateur
Pré-condition	Aucune
Scénario nominal	[début] 1. Accès à l'application ; 2. Le système affiche le formulaire d'authentification ; 3. L'utilisateur saisit son login et son mot de passe ; -Si les champs sont incomplets Alors Exécuter l'exception -Sinon Aller à (4) ; 4. Le système vérifie la validité des informations fournies -Si les champs sont incorrects Alors Exécuter l'exception ; -Sinon Aller à (5) ; 5. Le système donne l'accès à l'interface correspondante. [fin]
Alternative Exception	Le système affiche un message d'erreur et réaffiche la boîte d'authentification et attend que l'utilisateur ressaisisse ses informations.



Analyse de diagramme de cas d'utilisation

Cas d'utilisation	Gérer les clients.
Acteurs	Réceptionniste/Administrateur
Pré-condition	s'authentifier
Scénario nominal	<p>[début]</p> <ol style="list-style-type: none">1. Accès à l'application ;2. S'authentifier ;3. Le réceptionniste demande le formulaire de gestion des clients ;4. Le système affiche le formulaire ;5. Le réceptionniste effectue l'action souhaitée (ajout, modification, suppression) ; <p>Pour l'ajout :</p> <ul style="list-style-type: none">-Si les champs sont incomplets Alors Exécuter l'exception ;-Sinon Aller à (6) ; <ol style="list-style-type: none">6. Confirmer l'action ; <p>[fin]</p>
Alternative Exception	Le système affiche un message d'erreur et réaffiche le formulaire d'ajout et attend que l'utilisateur ressaisisse ses informations.



Analyse de diagramme de cas d'utilisation

Cas d'utilisation	Gérer les réservations.
Acteurs	Réceptionniste/Administrateur
Pré-condition	s'authentifier
Scénario nominal	<p>[début]</p> <ol style="list-style-type: none">1. Accès à l'application ;2. S'authentifier ;3. Le réceptionniste demande le formulaire de gestion des réservations ;4. Le système affiche le formulaire ;5. Le réceptionniste effectue l'action souhaitée (ajout, modification, suppression) ; <p>Pour l'ajout :</p> <ul style="list-style-type: none">-Si les champs sont incomplets Alors Exécuter l'exception ;-Sinon Aller à (6) ; <ol style="list-style-type: none">6. Confirmer l'action ; <p>[fin]</p>
Alternative Exception	Le système affiche un message d'erreur et réaffiche le formulaire d'ajout et attend que l'utilisateur ressaisisse ses informations.



Analyse de diagramme de cas d'utilisation

Cas d'utilisation	Gérer les chambres.
Acteurs	Administrateur
Pré-condition	s'authentifier
Scénario nominal	<p>[début]</p> <ol style="list-style-type: none">1. Accès à l'application ;2. S'authentifier ;3. L'administrateur demande le formulaire de gestion des chambres ;4. Le système affiche le formulaire ;5. L'administrateur effectue l'action souhaitée (ajout, modification, suppression) ; <p>Pour l'ajout :</p> <ul style="list-style-type: none">-Si les champs sont incomplets Alors Exécuter l'exception-Sinon Aller à (6) ;6. Confirmer l'action ; <p>[fin]</p>
Alternative Exception	Le système affiche un message d'erreur et réaffiche le formulaire d'ajout et attend que l'utilisateur ressaisisse ses informations.



Analyse de diagramme de cas d'utilisation

Cas d'utilisation	Gérer les paiements.
Acteurs	Réceptionniste/Administrateur
Pré-condition	s'authentifier
Scénario nominal	<p>[début]</p> <ol style="list-style-type: none">1. Accès à l'application ;2. S'authentifier ;3. L'administrateur demande le formulaire de gestion des paiements;4. Le système affiche le formulaire ;5. L'administrateur effectue l'action souhaitée (ajout, modification, suppression) ; <p>Pour l'ajout :</p> <ul style="list-style-type: none">-Si les champs sont incomplets Alors Exécuter l'exception-Sinon Aller à (6) ; <ol style="list-style-type: none">6. Confirmer l'action ; <p>[fin]</p>
Alternative Exception	Le système affiche un message d'erreur et réaffiche le formulaire d'ajout et attend que l'utilisateur ressaisisse ses informations.



Analyse de diagramme de cas d'utilisation

Cas d'utilisation	Faire une demande de réservation
Acteurs	Client
Pré-condition	s'authentifier
Scénario nominal	<p>[début]</p> <ol style="list-style-type: none">1. Accès à l'application ;2. S'authentifier ;3. L'administrateur demande le formulaire de demande de réservation de chambre;4. Le système affiche le formulaire ;5. L'utilisateur remplit tous les champs de la formulaire; <p>Pour l'ajout :</p> <ul style="list-style-type: none">-Si les champs sont incomplets ou bien aucune chambre est disponible Alors Exécuter l'exception-Sinon Aller à (6) ; <ol style="list-style-type: none">6. Confirmer l'action , mettre à jour l'état de chambre et paiements ; <p>[fin]</p>
Alternative Exception	<p>Le système affiche un message d'erreur et réaffiche le formulaire d'ajout et attend que l'utilisateur ressaisisse ses informations.</p>





REALISATION



Les outils de développement

L'étape de réalisation est la dernière de notre projet, elle se présente comme étant l'étape

la plus cruciale vu qu'elle traite l'onglet pratique du projet.

Nous commençons d'abord par une brève illustration de l'environnement de travail ainsi que

l'ensemble des logiciels qu'on a utilisé dans la réalisation de notre application de gestion d'un

hôtel et l'implémentation de base de données, puis nous passons à un aperçu des interfaces les

plus importantes de notre application.

MySQL

MySQL (My Structured Query Language) est un système de gestion de base de données Relationnelles (SGBDR) et basé sur un modèle client – serveur. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde.

Son rôle consiste à stocker et à gérer une ande quantité de données en les organisant sous forme de tables. Le système MySQL doit aussi permettre la manipulation de ces données à travers le langage standard du traitement des bases de données SQL. Les bases de données MySQL sont accessibles en utilisant les langages de programmation , Java, Perl, etc

PHPMysqlAdmin

PHPMysqlAdmin est un logiciel libre, écrit en PHP destiné à gérer l'administration de MySQL sur le World Wide Web . PhpMyAdmin supporte une large gamme d'opérations avec MySQL. Les opérations les plus fréquemment utilisés sont pris en charge par l'interface utilisateur (bases de données de gestion, tables, champs, relations, les index, les utilisateurs, les permissions), alors que vous avez toujours la possibilité d'exécuter directement une instruction SQL.

PhpMyAdmin permet de faire toutes sortes d'opérations comme : Créer et détruire des bases de données (à condition d'avoir les droits) ; Créer, détruire et modifier la description des tables ; Consulter le contenu des tables, modifier certaines lignes ou le détruire



Les outils de développement

Java Development Kit (JDK)

Le kit de développement Java est un environnement de développement de logiciel utilisé pour développer des applications et des applets Java. Il inclut le Java Runtime Environment (JRE), un interprète / chargeur (java), un compilateur (javac), un programme d'archivage (jar), un générateur de documentation (javadoc) et d'autres outils nécessaires au développement Java.

Eclipse

Eclipse est un environnement de développement intégré (IDE) dont le but est de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques. I.B.M. est à l'origine du développement d'Eclipse qui est d'ailleurs toujours le cœur de son outil Websphere Studio Workbench (WSW), lui-même à la base de la famille des derniers outils de développement en Java d'I.B.M. Eclipse utilise énormément le concept de modules nommés "plug-ins" dans son architecture. D'ailleurs, hormis le noyau de la plate-forme nommé "Runtime", tout le reste de la plate-forme est développé sous la forme de plug-ins. Ce concept permet de fournir un mécanisme pour l'extension de la plate-forme et ainsi fournir la possibilité à des tiers de développer des fonctionnalités qui ne sont pas fournies en standard par Eclipse. Les principaux modules fournis en standard avec Eclipse concernent Java mais des modules sont en cours de développement pour d'autres langages notamment C++, Cobol, mais aussi pour d'autres aspects du développement (base de données, conception avec UML, ...). Ils sont tous développés en Java soit par le projet

JDBC

Java DataBase Connectivity est très importante, appelée aussi " passerelle ", est composée d'un ensemble de classes permettant le dialogue entre une application Java et une source de données compatibles SQL (tables relationnelles en général). La passerelle JDBC a été développée de telle façon à permettre à un programme de se connecter à n'importe quelle base de données en utilisant la même syntaxe, c'est-à-dire que la passerelle JDBC est indépendante du SGBD. De plus, JDBC bénéficie des avantages de Java, dont la portabilité du code, ce qui lui vaut en plus d'être indépendant de la base de données d'être indépendant de la plate-forme sur laquelle elle s'exécute.





Les langages de programmation utilisés

Le langage SQL

SQL signifie " Structured Query Language " c'est-à-dire " Langage d'interrogation Structuré ". En fait SQL est un langage complet de gestion de bases de données relationnelles. L'accès aux BDD (bases de données) se fait de façon standard à l'aide de requêtes du langage SQL. Il existe un outil d'administration, PhpMyAdmin, qui nous offre une interface pour manipuler les tables. La connaissance de quelques requêtes permet de répondre à la majorité des besoins de programmation

Définition de Java

Java C'est un langage de programmation orienté objet, développé par Sun Microsystems. Dont le squelette principale est constitué du langage C++. En revanche, ces deux langages sont très différents dans leurs structures (organisation du code et gestion des variables). Ce langage peut être utilisé sur internet pour des petites applications intégrées à la page web (applet) ou encore comme langage serveur (jsp) et la réalisation de jeux fortement basé sur le graphisme. On effectue, il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris) et la création de graphismes élaborés, animés, ainsi que la présentation de texte évolué (hypertexte). L'avantage principal de Java par rapport aux autres langages c'est sa PORTABILITE, le fait qu'un programme Java puisse théoriquement être exécuté sur n'importe quelle plate-forme (type de processeur et système d'exploitation)





La distribution des tâches

Abderrahmen Jedidi :

- Design des interfaces graphique en Figma
- Effectuer la Gestion des réservation
- Changer le mot de passe
- TableModel
- User
- Création des tables
- Modèle entité association de projet
- Rapport de projet

Boussadia Chaabane :

- Conception de projet : Diagramme de cas d'utilisation
- Gestion des chambres
- Gestion des clients
- Page login
- Les interfaces de admin , receptioniste et client
- Dashbord
- Demande de réservation pour le client
- Une partie de Rapport de projet
- Fonction PLSQL

Marzouki Adem :

- Classe connexion de la base données avec le jdbc
- Effectuer la Gestion de paiement , le SignUp , des modifs sur le design
- Réalisation de démo de l'application
- Implémentation des classes : chambre , paiement
- Implémentation de design en Java (templates)



Les requêtes SQL Utilisé en Java

```
public static void updateuserDB(User user) {
    try {
        String query = "UPDATE user SET iduser=? , username=? , password=? ,typeuser=? WHERE iduser=?";
        Connection connection = new Connect().getConnection();
        PreparedStatement preparedStmt = connection.prepareStatement(query);
        preparedStmt.setInt(1, user.getIduser());
        preparedStmt.setString(2, user.getUsername());
        preparedStmt.setString(3, user.getPassword());
        preparedStmt.setInt(4, user.getTypeuser());
        preparedStmt.setInt(5, user.getIduser());
        preparedStmt.executeUpdate();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```
public static void DeleteUser(int idClient, int typeuser) {
    try {
        String query = "Delete from user where idClient=? and usertype=?";
        Connection connection = new Connect().getConnection();
        PreparedStatement preparedStmt = connection.prepareStatement(query);
        preparedStmt.setInt(1, idClient);
        preparedStmt.setInt(2, typeuser);
        preparedStmt.executeUpdate();
        connection.close();
    } catch (
        SQLException e) {
        e.printStackTrace();
    }
}
```



Les requêtes SQL Utilisé en Java

```
try {
    String query = "INSERT INTO USER(username,password,typeuser) VALUES (?, ?, ?);";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setString(1, username);
    preparedStmt.setString(2, cryptPass(password));
    preparedStmt.setInt(3, typeuser);
    preparedStmt.executeUpdate();
    connection.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

```
public static void newUser(String username, int idclient, int typeuser) {
    try {
        String query = "INSERT INTO USER(username,password,idclient,typeuser) VALUES (?, ?, ?, ?);";
        Connection connection = new Connect().getConnection();
        PreparedStatement preparedStmt = connection.prepareStatement(query);
        preparedStmt.setString(1, username);
        preparedStmt.setString(2, cryptPass(username));
        preparedStmt.setInt(3, idclient);
        preparedStmt.setInt(4, typeuser);
        preparedStmt.executeUpdate();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



Les requêtes SQL Utilisé en Java

```
//Verifier qu'il existe dans la base de données :
public User getUserFromDB() {
    try {
        String query = "SELECT * FROM user WHERE username=? and password=?";
        Connection connection = new Connect().getConnection();
        PreparedStatement preparedStmt = connection.prepareStatement(query);
        preparedStmt.setString(1, getUsername());
        preparedStmt.setString(2, getPassword());
        ResultSet resultSet = preparedStmt.executeQuery();
        User utilisateur = null;
        while (resultSet.next()) {
            utilisateur = new User();
            utilisateur.setIduser(resultSet.getInt(1));
            utilisateur.setUsername(resultSet.getString(2));
            utilisateur.setPassword(resultSet.getString(3));
            utilisateur.setIdclient(resultSet.getInt(4));
            utilisateur.setTypeuser(resultSet.getInt(5));
        }
        connection.close();
        return utilisateur;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

```
public static boolean isUsernameExists(String username) {
    try {
        Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/hotel", "root", "");
        String query = "SELECT * FROM user WHERE username=?";
        PreparedStatement statement = connection.prepareStatement(query);
        statement.setString(1, username);
        ResultSet resultSet = statement.executeQuery();
        return resultSet.next();
    } catch (SQLException ex) {
        ex.printStackTrace();
        return true;
    }
}
```



Les requêtes SQL Utilisé en Java

```
// Create user
String insertUserQuery = "INSERT INTO user (username, password, idclient, typeuser) VALUES (?, ?, ?, ?)";
insertUserStatement = connection.prepareStatement(insertUserQuery);
insertUserStatement.setString(1, userName);
insertUserStatement.setString(2, hashedPassword);
insertUserStatement.setInt(3, idClient);
insertUserStatement.setInt(4, type);
insertUserStatement.executeUpdate();

JOptionPane.showMessageDialog((Component) btnNewButton, "Welcome, " + msg + ". Your account is successfully created");
connection.close();
```

```
// Check if the client exists
int idClient = rand.nextInt(1000);
String checkClientQuery = "SELECT idclient FROM client WHERE email = ?";
PreparedStatement checkClientStatement = connection.prepareStatement(checkClientQuery);
checkClientStatement.setString(1, emailId);
ResultSet resultSet = checkClientStatement.executeQuery();
if (resultSet.next()) {
    idClient = resultSet.getInt("idclient");
} else {
    // Create a new client
    String insertClientQuery = "INSERT INTO client (nom_client, prenom_client, adresse, num_tel, email) VALUES (?, ?, ?, ?, ?)";
    PreparedStatement insertClientStatement = connection.prepareStatement(insertClientQuery, Statement.RETURN_GENERATED_KEYS);
    insertClientStatement.setString(1, lastName);
    insertClientStatement.setString(2, firstName);
    insertClientStatement.setString(3, adrs);
    insertClientStatement.setString(4, mobileNumber);
    insertClientStatement.setString(5, emailId);
    insertClientStatement.executeUpdate();
```



Les requêtes SQL Utilisé en Java

```
int id = 0;
try {
    String query = "select max(idReservation) from Reservation";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    ResultSet resultSet = preparedStmt.executeQuery();
    while (resultSet.next()) {
        id = resultSet.getInt(1);
    }
    connection.close();
    return id;
} catch (SQLException e) {
    e.printStackTrace();
}
```

```
String query = "SELECT COUNT(*) AS nombre_paiement FROM paiement WHERE idreservation = ?";
Connection connection = new Connect().getConnection();
PreparedStatement preparedStmt = connection.prepareStatement(query);
preparedStmt.setInt(1, reservationid);

ResultSet resultSet = preparedStmt.executeQuery();

if (resultSet.next()) {
    int nombreReservations = resultSet.getInt("nombre_paiement");
    return nombreReservations > 0;
}

catch (SQLException e) {
    e.printStackTrace();
}
```



Les requêtes SQL Utilisé en Java

```
String query = "UPDATE reservation SET id_client_reservant=?, date_reservation=?, typereservation=?,  
    check_in_date=?, check_out_date=?, souhait_particulier=?, id_chambre_reserve=? WHERE idReservation=?";  
Connection connection = new Connect().getConnection();  
PreparedStatement preparedStmt = connection.prepareStatement(query);  
preparedStmt.setInt(1, reservation.getClient().getIdclient());  
preparedStmt.setString(2, reservation.getDate_reservation());  
preparedStmt.setInt(3, reservation.getTypereservation());  
preparedStmt.setString(4, reservation.getCheck_in_date());  
preparedStmt.setString(5, reservation.getCheck_out_date());  
preparedStmt.setString(6, reservation.getSouhait_particulier());  
preparedStmt.setInt(7, reservation.getChambre().getNumChambre());  
preparedStmt.setInt(8, reservation.getIdreservation());  
preparedStmt.executeUpdate();  
connection.close();
```

```
try {  
    String query = "INSERT INTO Reservation(id_client_reservant,date_reservation,typereservation,check_in_date,check_out_date,souhait_particulier,id_chambre_reserve) VALUES (?, ?, ?, ?, ?, ?, ?,?)";  
    Connection connection = new Connect().getConnection();  
    PreparedStatement preparedStmt = connection.prepareStatement(query);  
    preparedStmt.setInt(1, reservation.getClient().getIdclient());  
    preparedStmt.setString(2, reservation.getDate_reservation());  
    preparedStmt.setInt(3, reservation.getTypereservation());  
    preparedStmt.setString(4, reservation.getCheck_in_date());  
    preparedStmt.setString(5, reservation.getCheck_out_date());  
    preparedStmt.setString(6, reservation.getSouhait_particulier());  
    preparedStmt.setInt(7, reservation.getChambre().getNumChambre());  
    preparedStmt.executeUpdate();  
    connection.close();  
}
```

```
String query = "Delete from Reservation where idReservation=?";  
Connection connection = new Connect().getConnection();  
PreparedStatement preparedStmt = connection.prepareStatement(query);  
preparedStmt.setInt(1, idReservation);  
preparedStmt.executeUpdate();  
connection.close();
```



Les requêtes SQL Utilisé en Java

```
String query = "SELECT r.*, \r\n"
    + "        cl.nom_client, cl.prenom_client, cl.adresse, cl.num_tel, cl.email\r\n"
    + "FROM reservation r\r\n"
    + "JOIN chambre c ON r.id_chambre_reserve = c.numchambre\r\n"
    + "JOIN client cl ON r.id_client_reservant = cl.idclient;\r\n"
    + "";
Connection connection = new Connect().getConnection();
listReservations = new ArrayList<Reservation>();
if (typereservation != 0)
    query += " and typereservation="
```

```
try {
    String query = "select max(idPaielement) from Paiement";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    ResultSet resultSet = preparedStmt.executeQuery();
    while (resultSet.next()) {
        id = resultSet.getInt(1);
    }
    connection.close();
    return id;
}
```



Les requêtes SQL Utilisé en Java

```
{
String query = "UPDATE paiement SET idreservation=? , montant=? , date=? , surcharge=? , typepaiement=? WHERE
idPaieement=?";
Connection connection = new Connect().getConnection();
PreparedStatement preparedStmt = connection.prepareStatement(query);
preparedStmt.setInt(1, paiement.getIdreservation());
preparedStmt.setDouble(2, paiement.getMontant());
preparedStmt.setString(3, paiement.getDate());
preparedStmt.setDouble(4, paiement.getSurcharge());
preparedStmt.setInt(5, paiement.getTypepaiement());
preparedStmt.setInt(6, paiement.getIdpaiement());
preparedStmt.executeUpdate();
connection.close();
}
```

```
try {
String query = "select max(idPaieement) from Paiement";
Connection connection = new Connect().getConnection();
PreparedStatement preparedStmt = connection.prepareStatement(query);
ResultSet resultSet = preparedStmt.executeQuery();
while (resultSet.next()) {
    id = resultSet.getInt(1);
}
connection.close();
return id;
}
```



Les requêtes SQL Utilisé en Java

```
try {
    String query = "UPDATE paiement SET idreservation=? , montant=? , date=? , surcharge=? , typepaiement=? WHERE idPaielement=?";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, paiement.getIdreservation());
    preparedStmt.setDouble(2, paiement.getMontant());
    preparedStmt.setString(3, paiement.getDate());
    preparedStmt.setDouble(4, paiement.getSurcharge());
    preparedStmt.setInt(5, paiement.getTypepaiement());
    preparedStmt.setInt(6, paiement.getIdpaiement());
    preparedStmt.executeUpdate();
    connection.close();
}
```

```
try {
    String query = "INSERT INTO Paiement(idreservation,montant,date,surcharge,typepaiement) VALUES (?, ?, ?, ?, ?)";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, paiement.getIdreservation());
    preparedStmt.setDouble(2, paiement.getMontant());
    preparedStmt.setString(3, paiement.getDate());
    preparedStmt.setDouble(4, paiement.getSurcharge());
    preparedStmt.setInt(5, paiement.getTypepaiement());
    preparedStmt.executeUpdate();
    connection.close();
}
```

```
try {
    String query = "Delete from Paiement where idPaielement=?";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, idPaielement);
    preparedStmt.executeUpdate();
    connection.close();
}
```



Les requêtes SQL Utilisé en Java

```
String query = "SELECT  
    idpaiement, paiement.idreservation, nom_client, prenom_client, montant, date, surcharge, typepaiement FROM  
    paiement, client, reservation where paiement.idReservation=Reservation.idReservation and  
    Reservation.id_client_reservant=client.idclient";  
Connection connection = new Connect().getConnection();  
listPaielements = new ArrayList<Paielement>();
```

```
String query = "select max(idclient) from Client";  
Connection connection = new Connect().getConnection();  
PreparedStatement preparedStmt = connection.prepareStatement(query);  
ResultSet resultSet = preparedStmt.executeQuery();
```

```
String query = "SELECT idclient, nom_client, prenom_client\r\n" // Determiner top 3 clients reguliers  
+ "FROM client c1\r\n"  
+ "WHERE 3 > ( SELECT count(*)\r\n"  
+ "          FROM client c2\r\n"  
+ "          WHERE (SELECT count(*)\r\n"  
+ "                FROM reservation r3\r\n"  
+ "                WHERE r3.id_client_reservant = c1.idclient) < (SELECT count(*)\r\n"  
+ "                                                                FROM reservation r4\r\n"  
+ "                                                                WHERE r4.id_client_reservant =  
+ "                                                                c2.idclient)\r\n"  
+ "\r\n"  
+ ")\r\n"  
+ "ORDER BY (SELECT count(*)\r\n"  
+ "          FROM reservation r3\r\n"  
+ "          WHERE r3.id_client_reservant = c1.idclient) DESC;";  
Connection connection = new Connect().getConnection();  
listClients = new ArrayList<Client>();  
PreparedStatement preparedStmt = connection.prepareStatement(query);  
ResultSet resultSet = preparedStmt.executeQuery();
```



Les requêtes SQL Utilisé en Java

```
try {
    String query = "UPDATE client SET idclient=?, nom_client=?, prenom_client=?, adresse=?, num_tel=?, email=? WHERE idclient=?";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, client.getIdclient());
    preparedStmt.setString(2, client.getNom_client());
    preparedStmt.setString(3, client.getPrenom_client());
    preparedStmt.setString(4, client.getAdresse());
    preparedStmt.setString(5, client.getNum_tel());
    preparedStmt.setString(6, client.getEmail());
    preparedStmt.setInt(7, client.getIdclient());
    preparedStmt.executeUpdate();
    connection.close();
}
```

```
String query = "SELECT count(*)\r\n"
    + " FROM reservation \r\n"
    + " WHERE id_client_reservant = ?;";
Connection connection = new Connect().getConnection();
PreparedStatement preparedStmt = connection.prepareStatement(query);
preparedStmt.setInt(1, idclient);
ResultSet resultSet = preparedStmt.executeQuery();
while (resultSet.next()) {
    nb = resultSet.getInt(1);
}
connection.close();
return nb;
```



Les requêtes SQL Utilisé en Java

```
String query = "SELECT COUNT(*) AS nombre_reservations FROM reservation WHERE id_client_reservant = ?";
Connection connection = new Connect().getConnection();
PreparedStatement preparedStmt = connection.prepareStatement(query);
preparedStmt.setInt(1, clientId);

ResultSet resultSet = preparedStmt.executeQuery();
```

```
try {
    String query = "INSERT INTO Client(idclient,nom_client,prenom_client,adresse,num_tel,email) VALUES (?, ?, ?, ?, ?, ?)";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, client.getIdclient());
    preparedStmt.setString(2, client.getNom_client());
    preparedStmt.setString(3, client.getPrenom_client());
    preparedStmt.setString(4, client.getAdresse());
    preparedStmt.setString(5, client.getNum_tel());
    preparedStmt.setString(6, client.getEmail());
    preparedStmt.executeUpdate();
    connection.close();
}
```

```
try {
    String query = "INSERT INTO Client(idclient,nom_client,prenom_client,adresse,num_tel,email) VALUES (?, ?, ?, ?, ?, ?)";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, client.getIdclient());
    preparedStmt.setString(2, client.getNom_client());
    preparedStmt.setString(3, client.getPrenom_client());
    preparedStmt.setString(4, client.getAdresse());
    preparedStmt.setString(5, client.getNum_tel());
    preparedStmt.setString(6, client.getEmail());
    preparedStmt.executeUpdate();
    connection.close();
}
```



Les requêtes SQL Utilisé en Java

```
try {
    String query = "Delete from Client where idclient=?";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, idclient);
    preparedStmt.executeUpdate();
    connection.close();
    return true;
}
```

```
try {
    String query = "Delete from Client where idclient=?";
    Connection connection = new Connect().getConnection();
    PreparedStatement preparedStmt = connection.prepareStatement(query);
    preparedStmt.setInt(1, idclient);
    preparedStmt.executeUpdate();
    connection.close();
    return true;
}
```

```
String query = "SELECT idclient,nom_client,prenom_client,adresse,num_tel,email FROM client";
Connection connection = new Connect().getConnection();
listClients = new ArrayList<Client>();

switch (orderby) {
    case 1:
        query += " Order by nom_client";
        break;
    case 2:
        query += " Order by prenom_client";
        break;
    default:
        break;
}

if (desc != 0)
    query += " DESC";
PreparedStatement preparedStmt = connection.prepareStatement(query);
ResultSet resultSet = preparedStmt.executeQuery();
while (resultSet.next()) {
    Client client = new Client();
    client.setIdclient(resultSet.getInt(1));
    client.setNom_client(resultSet.getString(2));
    client.setPrenom_client(resultSet.getString(3));
    client.setAdresse(resultSet.getString(4));
    client.setNum_tel(resultSet.getString(5));
    client.setEmail(resultSet.getString(6));
}
```

Les requêtes SQL Utilisé en Java

```
String query = "select idclient,nom_client,prenom_client,adresse,num_tel,email from client where idclient = ? ";
Connection connection = new Connect().getConnection();
PreparedStatement preparedStmt = connection.prepareStatement(query);
preparedStmt.setInt(1, idClient);
ResultSet resultSet = preparedStmt.executeQuery();
while (resultSet.next()) {
    Client client = new Client();
    client.setIdclient(resultSet.getInt(1));
    client.setNom_client(resultSet.getString(2));
    client.setPrenom_client(resultSet.getString(3));
    client.setAdresse(resultSet.getString(4));
    client.setNum_tel(resultSet.getString(5));
    client.setEmail(resultSet.getString(6));
    return client;
}
```

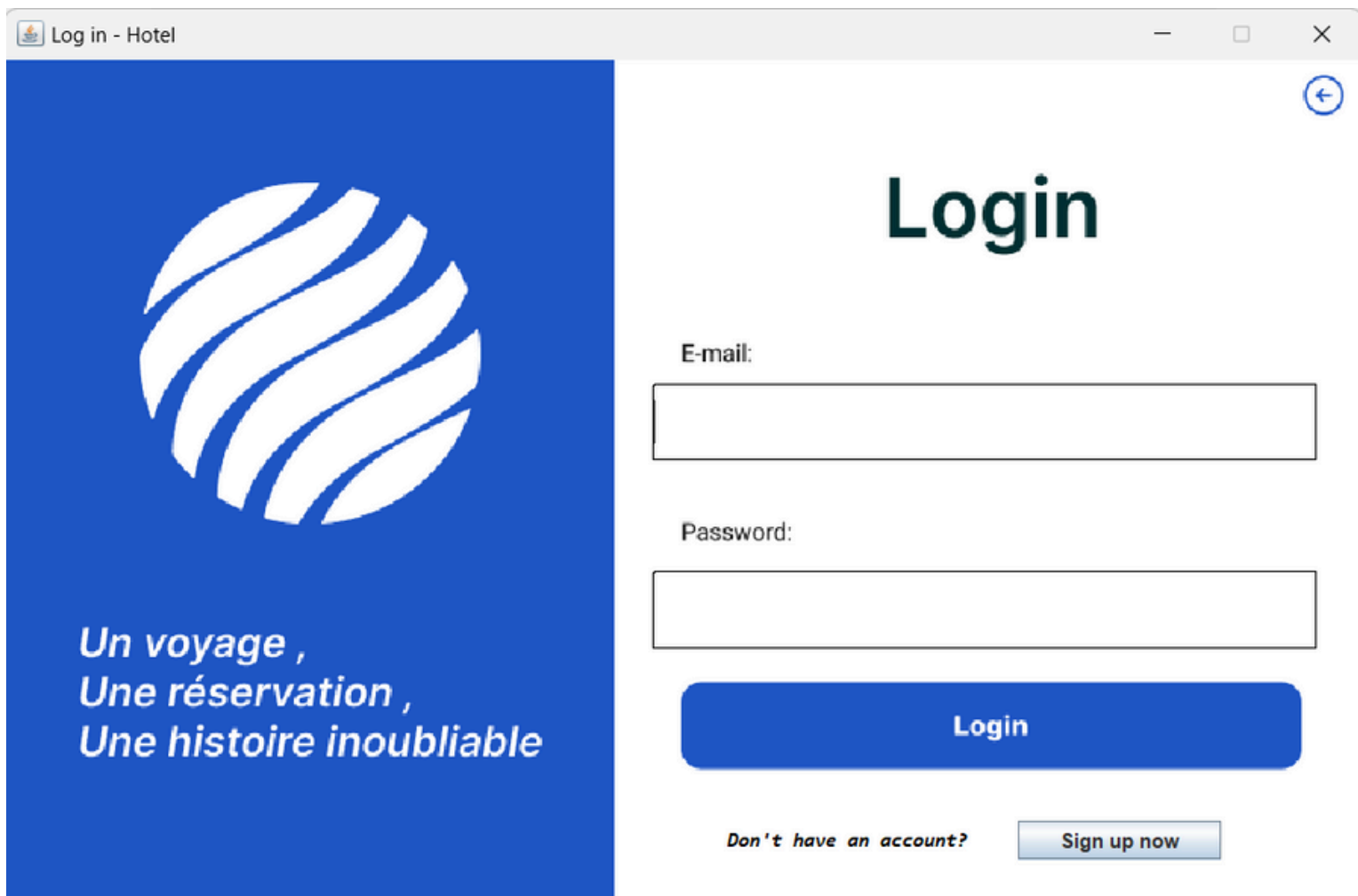
```
String query = "SELECT *\r\n"
    + "FROM chambre\r\n"
    + "WHERE status = 2\r\n"
    + "AND capacity >= ? \r\n"
    + "AND numchambre NOT IN (\r\n"
    + "    SELECT id_chambre_reserve\r\n"
    + "    FROM reservation\r\n"
    + "    WHERE (check_in_date <= ? AND check_out_date >= ?)\r\n"
    + "    OR (check_in_date BETWEEN ? AND ?)\r\n"
    + "    OR (check_out_date BETWEEN ? AND ?)\r\n"
    + ")\r\n"
    + "UNION\r\n"
    + "SELECT *\r\n"
    + "FROM chambre\r\n"
    + "WHERE status = 1\r\n"
    + "AND capacity >= ?";

Connection connection = new Connect().getConnection();
listRooms = new ArrayList<Chambre>();
PreparedStatement preparedStmt = connection.prepareStatement(query);
preparedStmt.setString(2, CheckOut);
preparedStmt.setString(5, CheckOut);
preparedStmt.setString(7, CheckOut);
preparedStmt.setString(3, CheckIn);
```



Interfaces Java:

Login :



The screenshot shows a Java Swing window titled "Log in - Hotel". The window is divided into two main sections. On the left, there is a blue rectangular area containing a white circular logo with diagonal stripes. Below the logo, the text "Un voyage ,
Une réservation ,
Une histoire inoubliable" is displayed in white. On the right, the background is white. At the top right of this section is a blue circular icon with a white arrow pointing left. Below this icon, the word "Login" is written in a large, bold, dark blue font. Underneath "Login", there are two labels: "E-mail:" and "Password:". Each label is followed by a white rectangular input field with a thin black border. Below the "Password:" input field is a blue rounded rectangular button with the word "Login" in white. At the bottom left of the white section, the text "Don't have an account?" is displayed in a small, italicized font. To the right of this text is a light blue rounded rectangular button with the text "Sign up now" in a small, bold font.

Log in - Hotel

←

Login

E-mail:

Password:

Login

Don't have an account? Sign up now



Sign Up:

Sign Up

— □ ×

Explore. Dream. Discover – Your journey begins here.

SIGN UP

Nom :	Username :
<input type="text"/>	<input type="text"/>
Prenom :	Email :
<input type="text"/>	<input type="text"/>
Adresse :	Password :
<input type="text"/>	<input type="text"/>
Numero de telephone :	Password confirmation :
<input type="text"/>	<input type="text"/>





Sign Up



I've an account

Login



Change Password :





Change Password

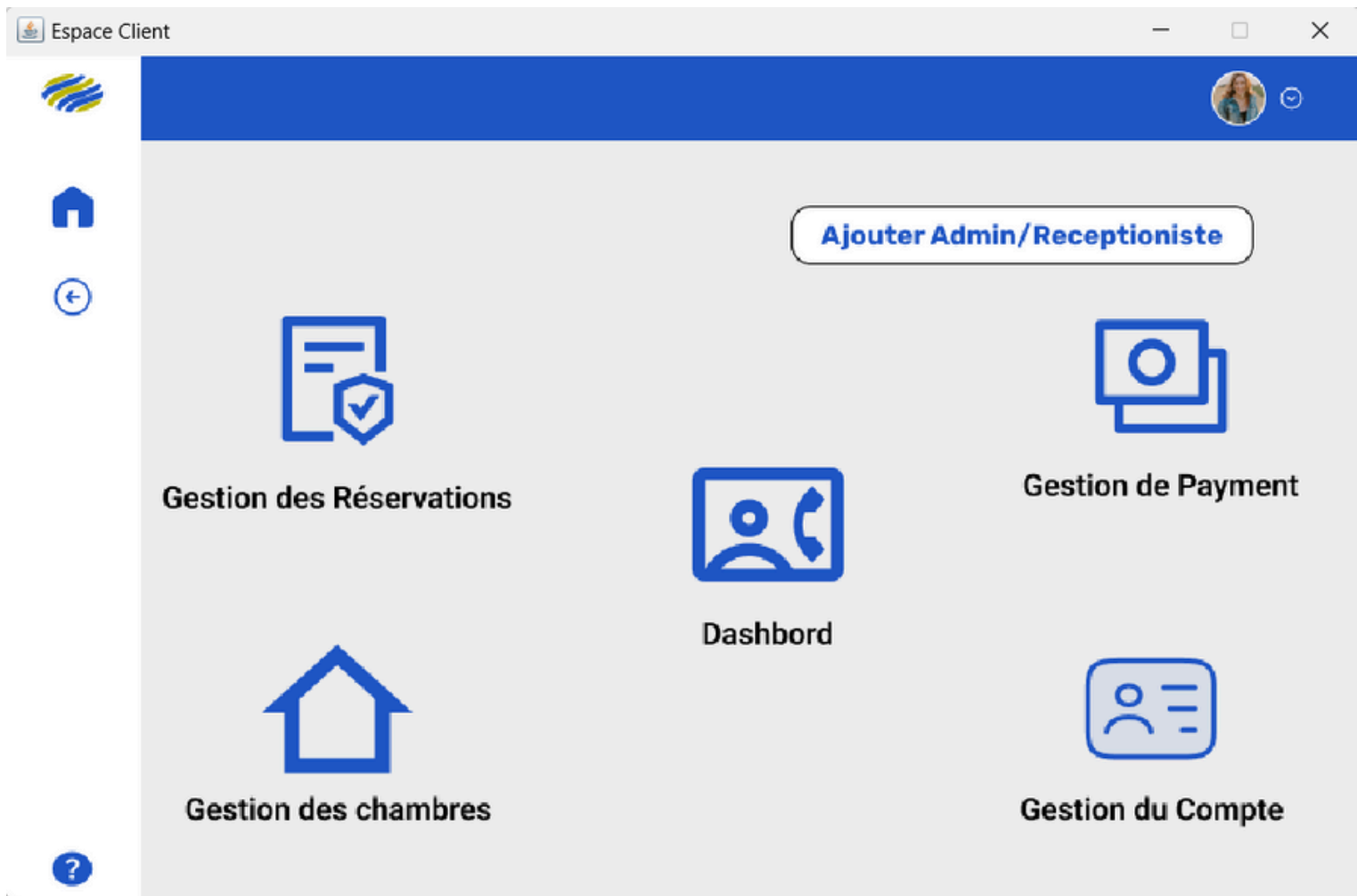
Old Password

New Password

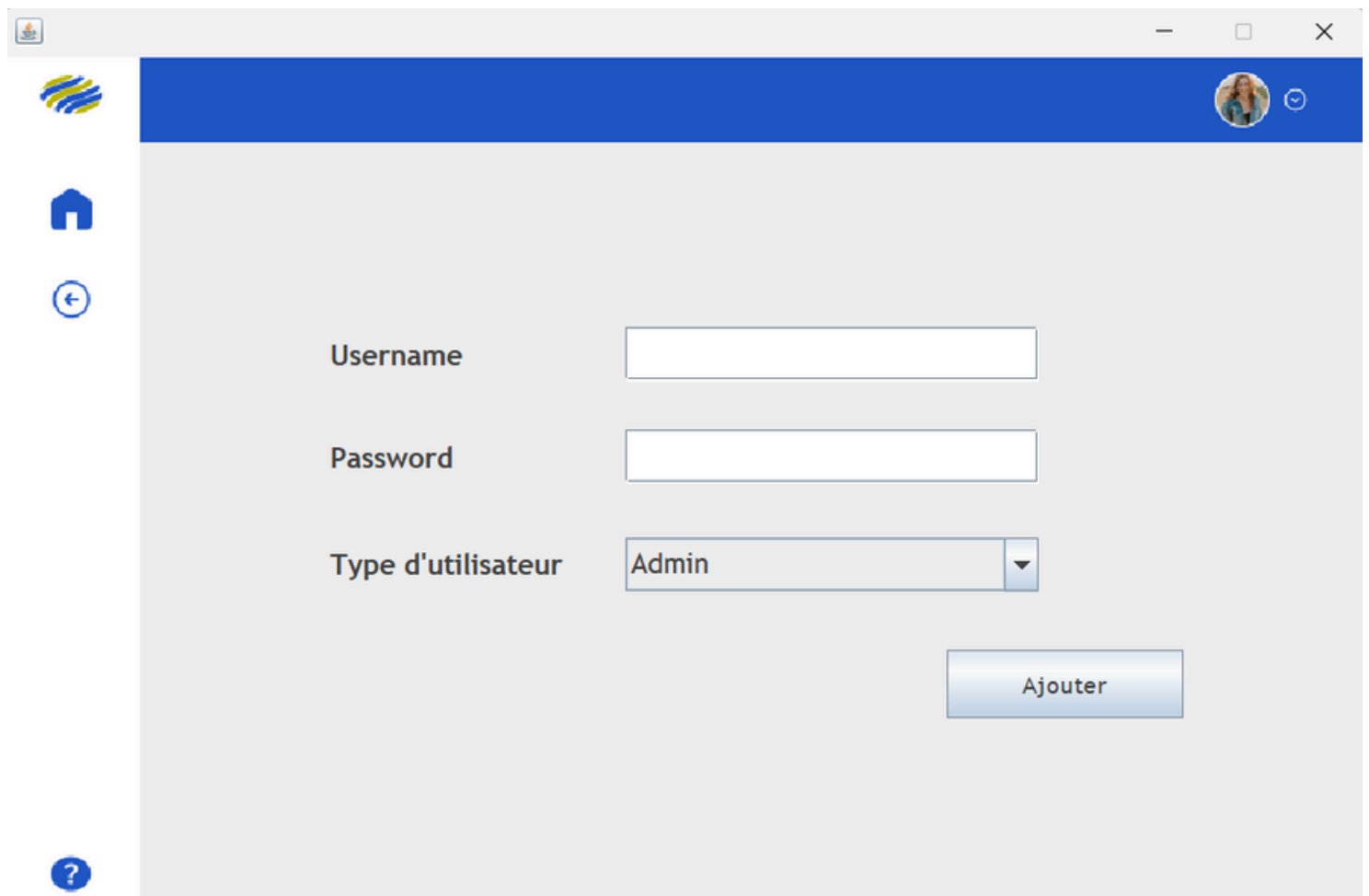
Change Password



Dashboard Admin :



Ajout :



The screenshot shows a web application window with a blue header bar. On the left, there is a vertical sidebar with four icons: a logo, a home icon, a back icon, and a help icon. The main content area has a light gray background. It contains three input fields: a text box for 'Username', a text box for 'Password', and a dropdown menu for 'Type d'utilisateur' with 'Admin' selected. A blue 'Ajouter' button is located at the bottom right of the form.

Username




Password


Type d'utilisateur Admin ▼

Ajouter



Chambre:



Id Chambre:

Filters :

Reg Type

Order by

Ascending

New

Delete

Refresh

Save Changes

Reservation

id_client

nom_client

prenom_client

date_res

type

check_in

check_out

souhait

idCha

1

Boussadia

Chaabane

2024-04-12

In Person

2024-05-05

2024-05-07

1

2

Flen

Fleni

2024-04-13

In Person

2026-04-30

2026-07-16

1

Num Room

Capacity

Type

Status

Floor

PricePerDay

1

12

Normal

Occupied

2

12.45

2

3





Normal

Free

2

2.5

Imprimer:



numChambre	Capacite	TypeChambre	etage	Prix_par_jour	Nombre de demand..
1	12	Normal	2	12.45	2
2	3	Normal	2	2.5	0

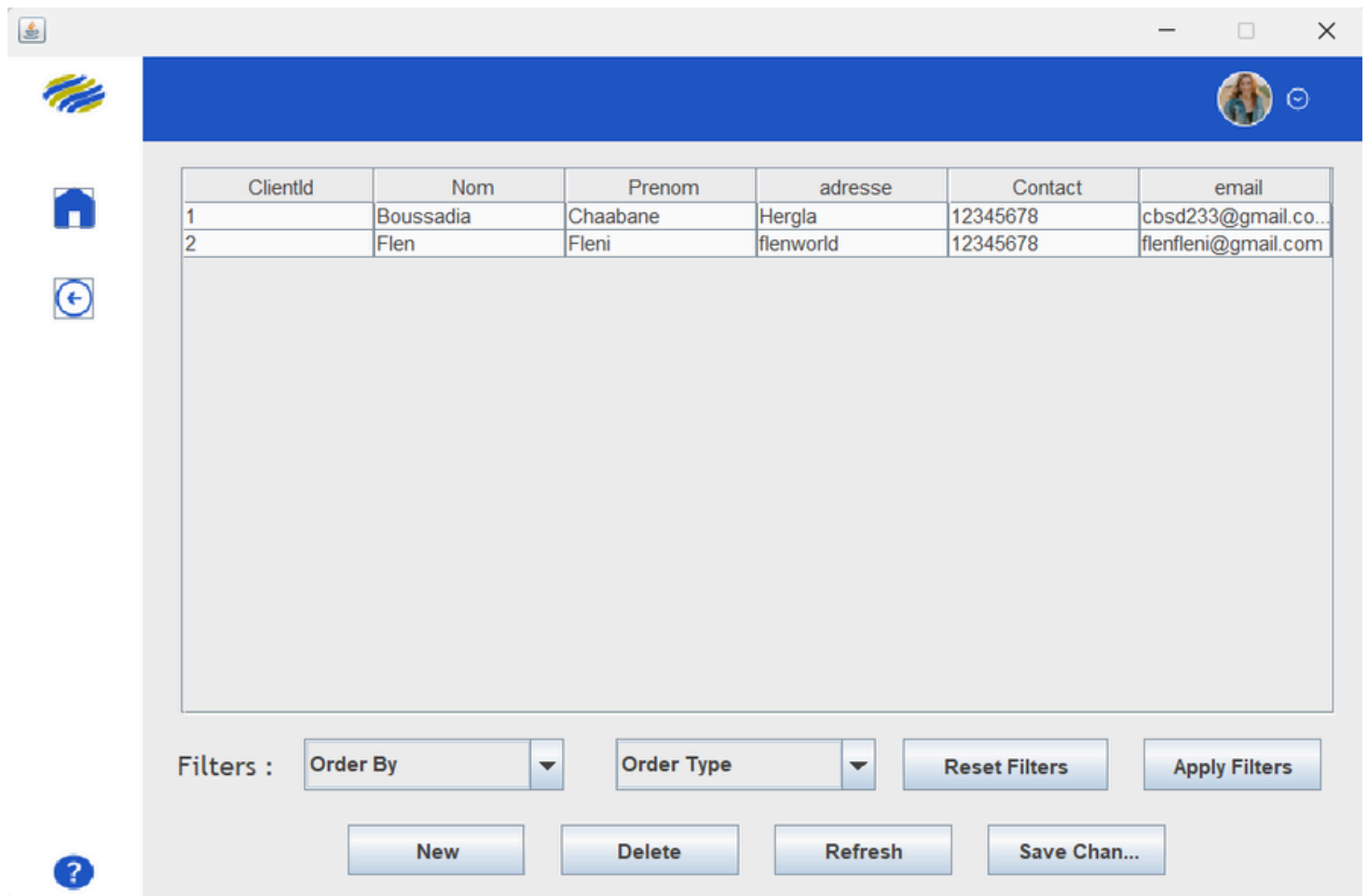
idclient	nom_client	prenom_client	Nombre de reservations
1	Boussadia	Chaabane	1
2	Flen	Fleni	1

Exporter en CSV

Imprimer



Ajout Client :



The screenshot shows a web application interface for client management. The interface includes a sidebar with navigation icons, a main content area with a table of clients, and a bottom toolbar with filters and action buttons.

Table Data:





ClientId	Nom	Prenom	adresse	Contact	email
1	Boussadia	Chaabane	Hergla	12345678	cbsd233@gmail.co...
2	Flen	Fleni	flenworld	12345678	flenfleni@gmail.com

Filters: Order By [v] Order Type [v] [Reset Filters] [Apply Filters]

Actions: [New] [Delete] [Refresh] [Save Chan...]



Paielement :



idpaiement	idreservation	nom	prenom	montant	date	surcharge	typepaiement
							In Person
							In Person
							Online

Filters :

Ascending

Order by

Paielement Ty...

Reset Filters

Apply Filters

New

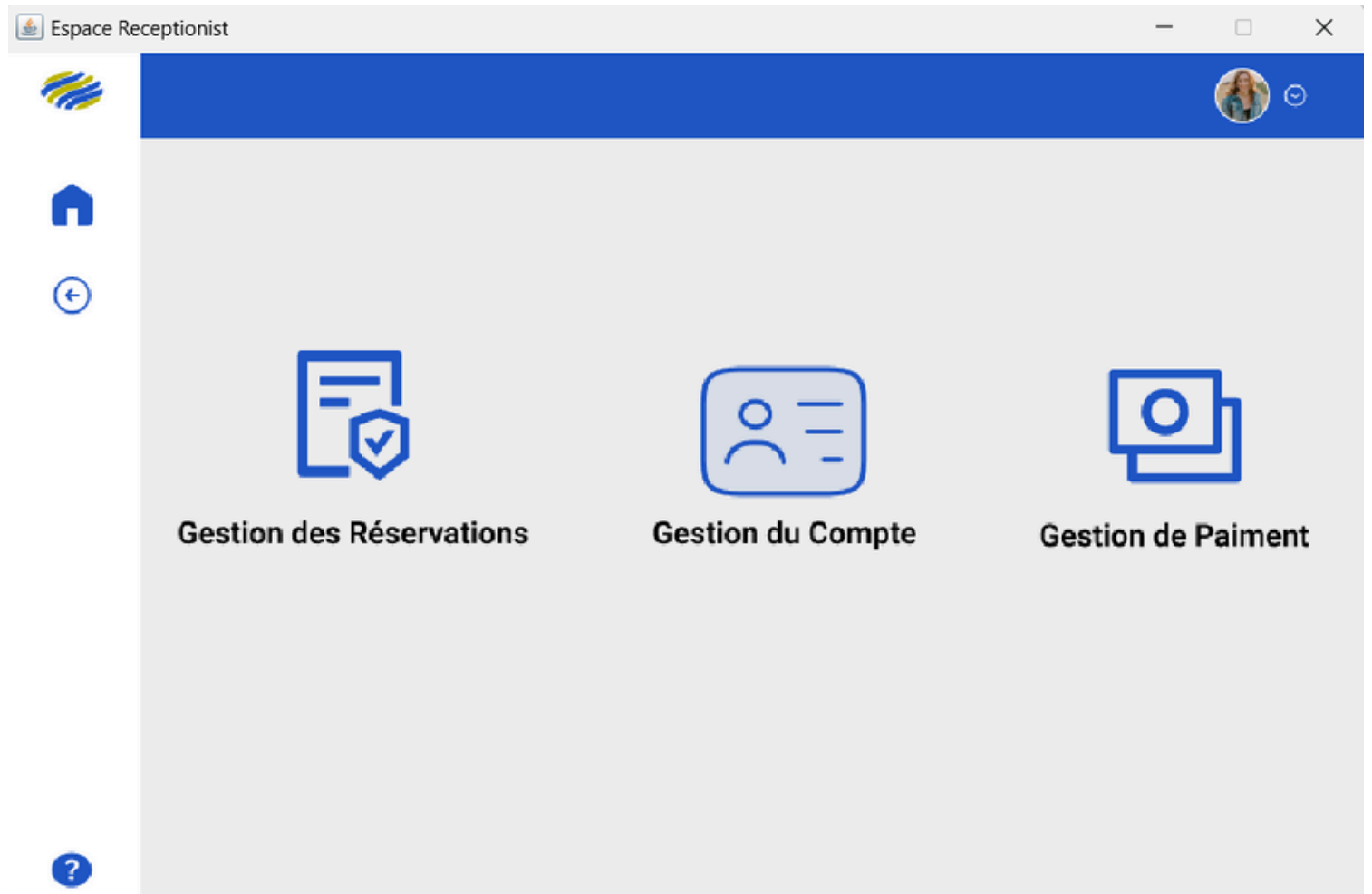
Delete

Refresh

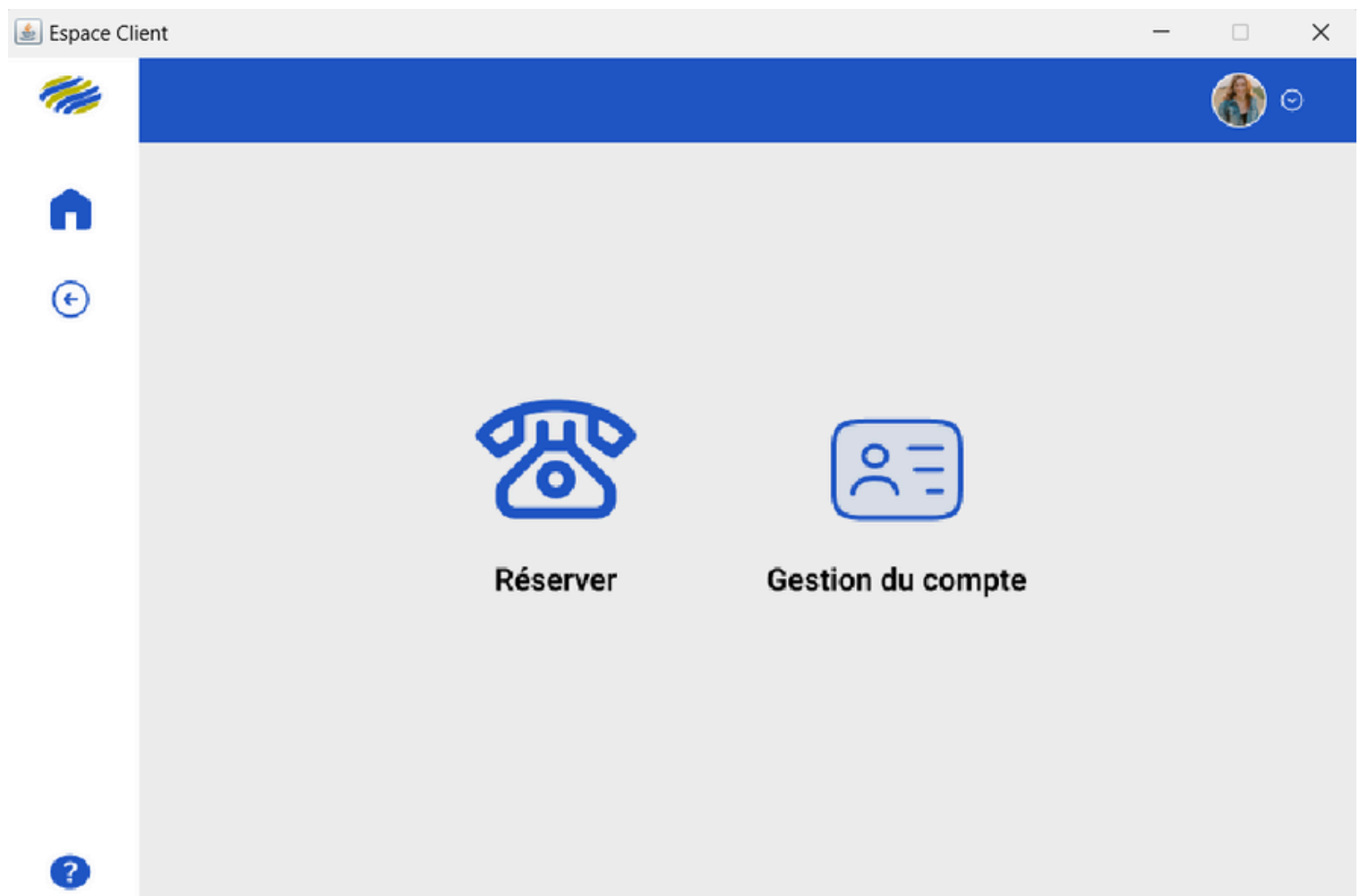
Save Changes



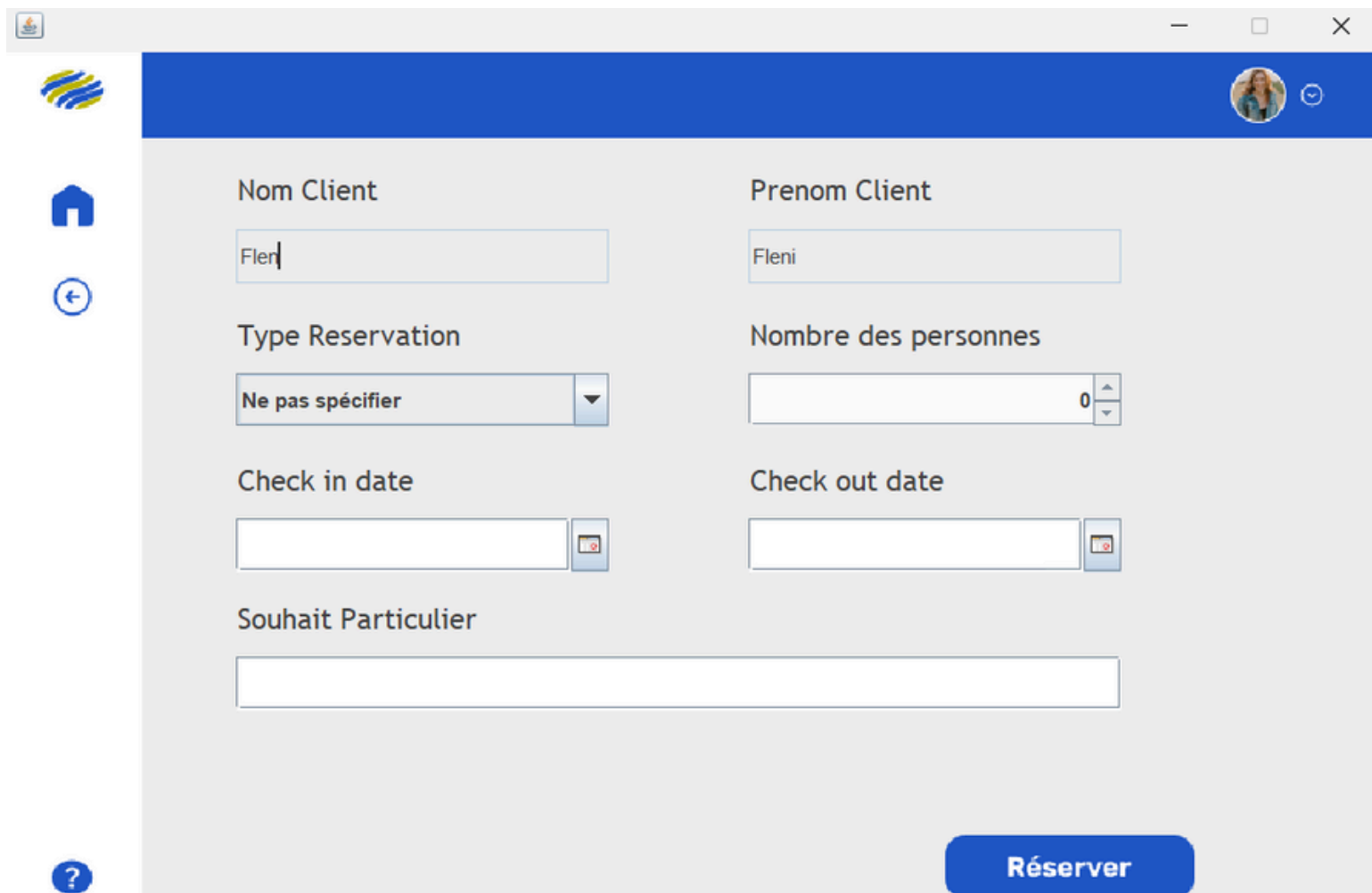
Dashboard Receptionist :



Dashboard Client :



Réserver:



The image shows a web application window for making a reservation. The window has a blue header bar with a logo on the left and a user profile icon on the right. The main content area is light gray and contains several form fields arranged in two columns. The left column includes fields for 'Nom Client' (with the text 'Fler' entered), 'Type Reservation' (a dropdown menu set to 'Ne pas spécifier'), 'Check in date' (with a calendar icon), and 'Souhait Particulier' (a long text area). The right column includes fields for 'Prenom Client' (with the text 'Fleni' entered), 'Nombre des personnes' (a numeric input set to '0'), and 'Check out date' (with a calendar icon). At the bottom right of the form is a blue button labeled 'Réserver'. On the far left of the window, there is a vertical sidebar with three icons: a house, a left arrow, and a question mark.

Nom Client	Prenom Client
<input type="text" value="Fler"/>	<input type="text" value="Fleni"/>
Type Reservation	Nombre des personnes
<input type="text" value="Ne pas spécifier"/>	<input type="text" value="0"/>
Check in date	Check out date
<input type="text"/>	<input type="text"/>
Souhait Particulier	
<input type="text"/>	

Réserver



