Rapport de projet base de données/réseaux : The Key to Management

Gestion d'une agence d'hôtellerie.

The Key to Management

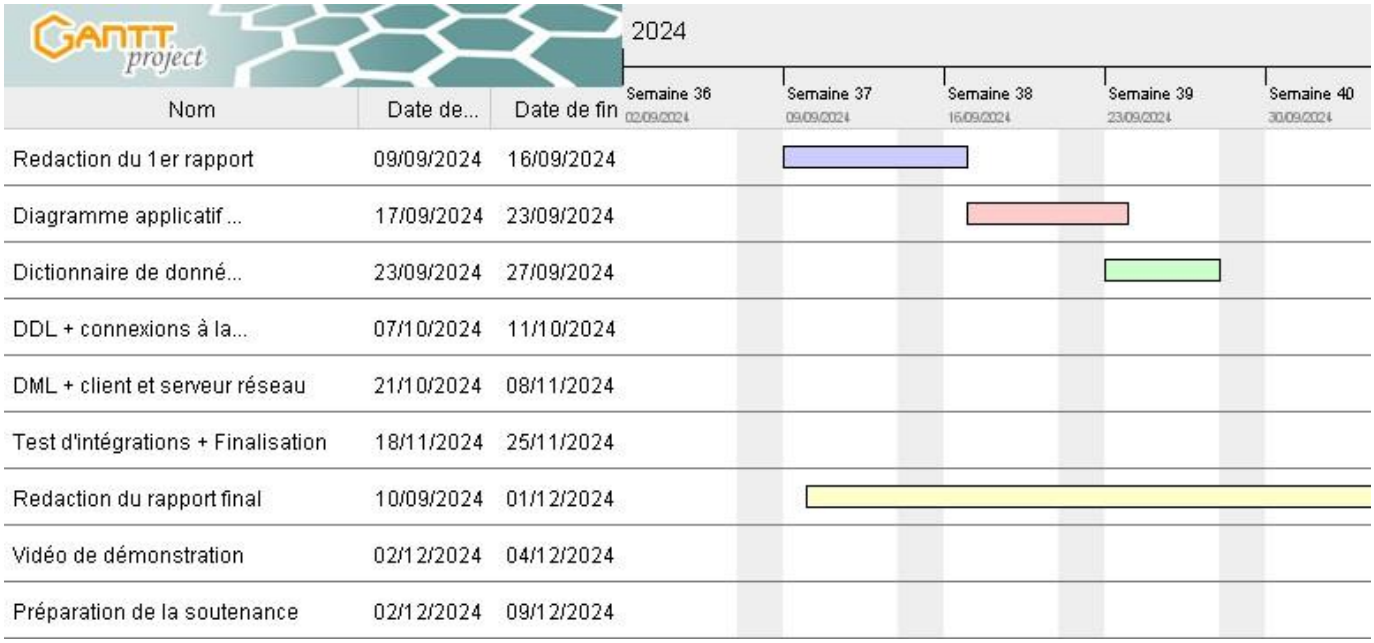
Date	Editeur	Version
12/09/2024	CHEBALLAH, ANAGONOU, HUANG	V1(Contexte)
26/09/2024	CHEBALLAH, ANAGONOU, HUANG	V2(MLD, MCD, Dictionnaire de définition, échanges réseaux)
11/10/2024	CHEBALLAH, ANAGONOU, HUANG	V3(Correction V2 + Serveur et Client + jeu de donnée + Test Serveur et Client)
26/10/2024	CHEBALLAH, ANAGONOU, HUANG	V4 (Correction V3 + liaison bdd avec notre serveur)
30/11/2024	CHEBALLAH, ANAGONOU, HUANG	Final (Correction V4 + Site web et connexion bd avec requêtes)

CERGY PARIS UNIVERSITÉ

Rédigé par :

CHEBALLAH Jawed | ANAGONOU Pirès | HUANG Yanmo

1. Planning



Nom	Date de...	Date de fin
Redaction du 1er rapport	09/09/2024	16/09/2024
Diagramme applicatif ...	17/09/2024	23/09/2024
Dictionnaire de donné...	23/09/2024	27/09/2024
DDL + connexions à la...	07/10/2024	11/10/2024
DML + client et serveur réseau	21/10/2024	08/11/2024
Test d'intégrations + Finalisation	18/11/2024	25/11/2024
Redaction du rapport final	10/09/2024	01/12/2024
Vidéo de démonstration	02/12/2024	04/12/2024
Préparation de la soutenance	02/12/2024	09/12/2024

2. Contexte

Le secteur de l'hôtellerie a évolué considérablement avec l'intégration des technologies numériques, permettant aujourd'hui aux clients de vivre des expériences fluides et personnalisées. Notre projet de gestion hôtelière vise à moderniser les processus de réservation, d'annulation, de modification des réservations, ainsi que la gestion des accès aux chambres, tout en offrant une plateforme web conviviale et intuitive.
L'objectif principal est de simplifier la gestion des réservations pour les clients et de fournir une interface robuste pour les gestionnaires. Grâce à une base de données centralisée, le système gérera efficacement les chambres, les réservations et les comptes utilisateurs. La

sécurité et la confidentialité seront garanties par des protocoles stricts de gestion des identités.

La plateforme permettra aux clients de consulter les disponibilités en temps réel, d'annuler ou de modifier leurs réservations facilement. Une gestion optimisée des arrivées et départs facilitera aussi le travail du personnel hôtelier. Par ailleurs, chaque client pourra créer un compte pour suivre ses réservations, gérer ses préférences et accéder à des offres personnalisées. Le projet inclut également un système sécurisé de gestion d'accès aux chambres via cartes magnétiques ou dispositifs mobiles, répondant aux attentes de modernité et de simplicité des clients.

En résumé, ce projet propose une solution complète et innovante, alliant efficacité, sécurité et personnalisation, pour améliorer l'expérience des clients et des gestionnaires hôteliers.

3. Dictionnaire de données

I. Client

Nom Attribut	TYPE	N/NN	Description/Restriction	Exemple
idClient	**Char(10)**	**NN**	**unique + PK**	**0123456789**
nomClient	**Varchar(40)**	**NN**	**/**	**Dupont**
prénomClient	**Varchar(40)**	**NN**	**/**	**Paul**
mailClient	**Varchar(100)**	**NN**	**Contient @ + unique**	**Paul.dupont@gmail.com**
téléphoneClient	**Char(12)**	**NN**	**« + » + unique**	**+33765426338**
dateDeNaissance	**Date**	**NN**	**<= 100y + >= 18y**	**20/10/2003**

II. Réservation

idReservation	**Char(12)**	**NN**	**unique + PK**	**000123456789**
dateDebut	**Date**	**NN**	**<DateFin**	**01/10/2023**

typeReservation	Char(3)	NN	Enumération : « srp » « enl » (Sur place et En ligne)	srp
etatReservation	Char(3)	NN	Enumération : « rsv » « att » « ann » (Reservé, en Attente et Annulé)	rsv
dateFin	Date	NN	>DateDebut	08/10/2023
numeroCarte	Char(8)	NN	/	12345678

III. Hôtel

idHôtel	Char(8)	NN	unique + PK	12345678
adresse	Varchar(100)	NN	Forme : numeroRue + nom Rue +	75 rue Edmond Jaloux Cergy 95800
			Ville + code postal	
téléphoneHôtel	Char(12)	NN	« + » + unique	+33156453236
mailHôtel	Varchar(100)	NN	Contient @theKTM + unique + « .com »	Jaloux.hotel@theKTM.com

IV. Chambre

idChambre	Char(8)	NN	unique + PK	12345678
numeroChambre	Varchar(4)	NN	/	5
etatChambre	Char(4)	NN	Enum : « rsrv », « free » (Réservé ou libre)	free

V. Type Chambre

idTypeChambre	Char(8)	NN + PK	unique	12345678
nomType	Varchar(40)	NN	/	Suite 2 personnes
Prix	Float	NN	/	875,99

VI. Nettoyage

idNettoyage	Char(8)	NN	unique + PK	12345678
dateNettoyage	Date	NN	>DateFin de réservation ou demande client	03/10/2023
typeNettoyage	Varchar(20)	NN	/	profond

VII. Employé

idEmployé	Char(8)	NN	unique + PK	12345678
nomEmployé	Varchar(40)	NN	/	James
prenomEmployé	Varchar(40)	NN	/	Lebron
telEmployé	Char(10)	NN	« + » + unique	+33689564236
posteEmployé	Varchar(40)	NN	/	Femme de menage
numeroCarte	Char(8)	NN	/	12345678

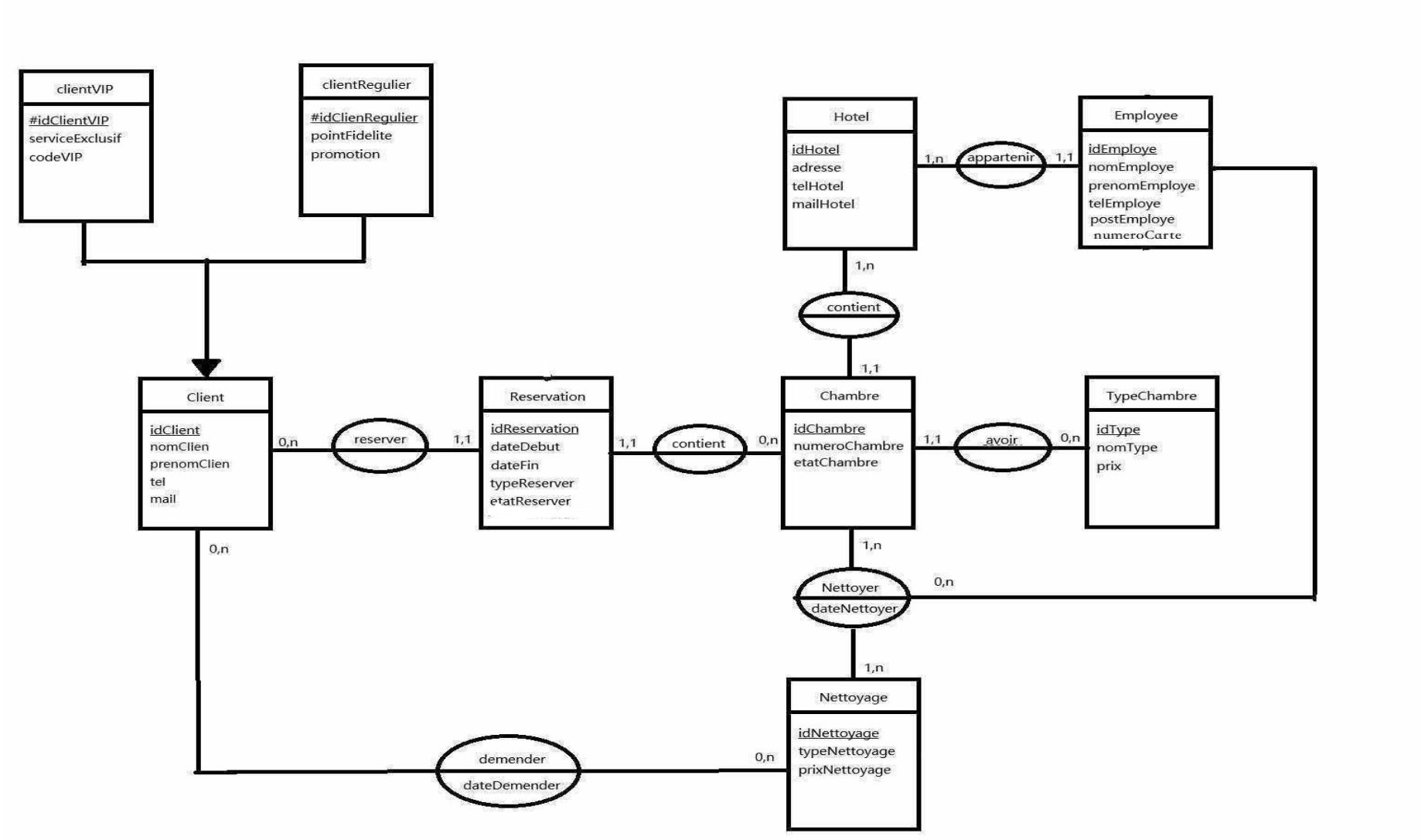
VIII. ClientVIP

idClientVIP	**Char(8)**	**NN**	**unique + PK**	**12345678**
serviceExclusif	**Varchar(40)**	**N**	**/**	**Acces piscine**
codeVIP	**Char(12)**	**NN**	**unique**	**123456789000**

IX. ClientRegulier

idClientRegulier	**Char(8)**	**NN**	**unique + PK**	**12345678**
pointFidélité	**int**	**NN**	**/**	**75000(1eu = 100point)**
promotion	**float**	NN	**0 < promotion < 1**	**0.3(30% de reduction)**

4. MCD


clientVIP
#idClientVIP
serviceExclusif
codeVIP
clientRegulier
#idClienRegulier
pointFidelite
promotion
Hotel
idHotel
adresse
telHotel
mailHotel
1,n
appartenir
1,1
Employee
idEmploye
nomEmploye
prenomEmploye
telEmploye
postEmploye
numeroCarte
1,n
contient
1,1
Client
idClient
nomClien
prenomClien
tel
mail
0,n
reserver
1,1
Reservation
idReservation
dateDebut
dateFin
typeReserver
etatReserver
1,1
contient
0,n
Chambre
idChambre
numeroChambre
etatChambre
1,1
avoir
0,n
TypeChambre
idType
nomType
prix
0,n
1,n
Nettoyer
dateNettoyer
0,n
1,n
Nettoyage
idNettoyage
typeNettoyage
prixNettoyage
demender
dateDemender
0,n

5. MLD

Hôtel (**idHotel**, adresse, telHotel, mailHotel)

Employé (**idEmploye**, nomEmploye, prenomEmploye, telEmploye, posteEmploye, numeroCarte, *#idHotel*)

TypeChambre (**idTypeChambre**, nomType, prix)

Chambre (**idChambre**, numeroChambre, etatChambre, *#idTypeChambre, #idHotel*)

Client (**idClient**, nomClient, prenomClient, tel, mail)

ClientVIP (***#idClientVIP***, serviceExclusif, codeVIP)

ClientRegulier (***#idClientRegulier***, pointFidelite, promotion)

Réservation (**idReservation**, dateDebut, dateFin, typeReserver, etatReserver, *#idClient*, *#idChambre*)

Nettoyage (**idNettoyage**, typeNettoyage, prixNettoyage)

Demande (***#idNettoyage***, ***#idClient***, dateDemander)

Nettoyer (***#idNettoyage***, ***#idChambre***, ***#idEmploye***, dateNettoyer)

6. Echanges réseaux

Nous avons choisi le protocole **TCP** pour les échanges entre le client et le serveur dans ce projet pour les raisons suivantes :

Fiabilité des échanges : Nous avons choisi TCP car TCP garantit que les messages envoyés par le client sont bien reçus par le serveur, sans perte ni corruption de données.

Les données critiques, comme les identifiants des utilisateurs, les numéros de chambre, ou les cartes d'accès, nécessitent une transmission fiable.

Garantie de l'ordre des messages :

TCP assure que les messages sont reçus dans le même ordre que celui dans lequel ils ont été envoyés. Cela est essentiel pour éviter des incohérences dans la validation des informations.

Gestion des erreurs :

En cas de perte de paquets ou de corruption pendant la transmission, TCP prend en charge la retransmission automatique des données. Cela simplifie la gestion des erreurs côté application.

Connexion orientée :

TCP établit une connexion stable entre le client et le serveur avant de permettre les échanges de données. Cette connexion assure que les deux parties sont prêtes à communiquer et qu'aucune donnée ne sera ignorée.

Cas d'usage adapté :

Dans ce projet, les échanges réseau sont centrés sur la validation des informations d'accès et la gestion des permissions. Une perte ou un ordre incorrect des données pourrait entraîner des résultats incorrects ou des failles de sécurité.

Le port par défaut est : **8008**

L'IP par défaut est : **127.0.0.1**

I. Scénario idéal Client

Requête :

```
SELECT chambre.numerochambre, reservation.idreservation
FROM reservation
JOIN chambre ON reservation.idchambre = chambre.idchambre
WHERE reservation.idclient = %s
AND reservation.etatreserver = 'rsv'
```

Explication :

Contexte : Lorsqu'un client se connecte, ses informations (ID client, numéro de chambre, numéro de reservation) doivent être validées.

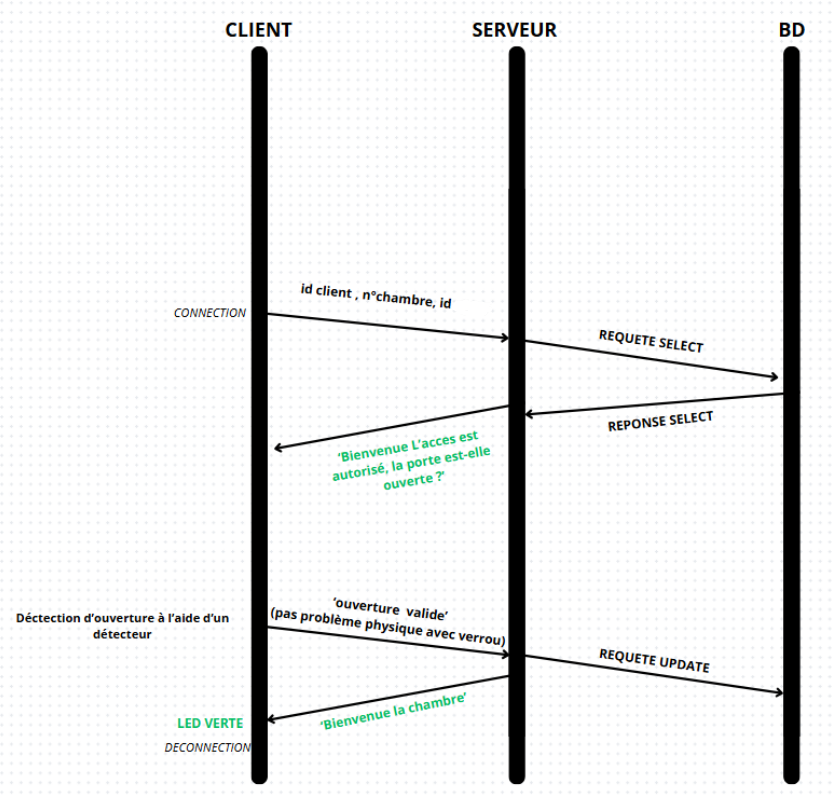
But : Cette requête joint les tables Client, Reservation, et Chambre pour vérifier que :

Le client a bien une réservation associée à son idClient.

Le numéro de chambre et le numéro de reservation fournis par le client correspondent aux données en base.

Résultat attendu :

Toutes les informations correspondent, l'accès est autorisé avec un message de bienvenue.



II. Scénario Client qui se trompe de chambre

Requête :

```
SELECT chambre.numerochambre, reservation.idreservation
FROM reservation
JOIN chambre ON reservation.idchambre = chambre.idchambre
WHERE reservation.idclient = %s
AND reservation.etatreserver = 'rsv'
```

Explication :

Contexte : Lorsqu'un client se connecte, ses informations (ID client, numéro de chambre, numéro de reservation) doivent être validées.

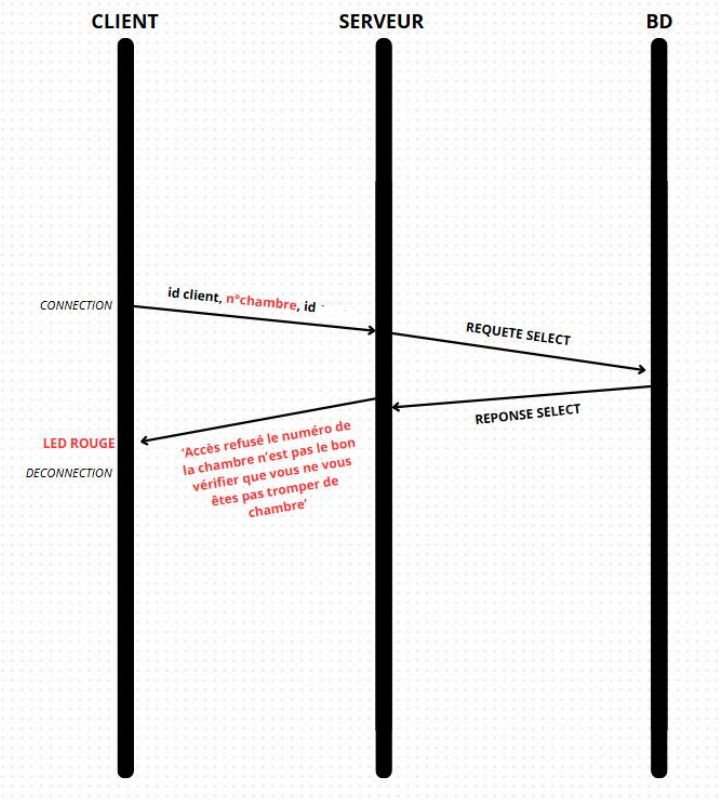
But : Cette requête joint les tables Client, Reservation, et Chambre pour vérifier que :

Le client a bien une réservation associée à son idClient.

Le numéro de chambre et le numéro de reservation fournis par le client correspondent aux données en base.

Résultat attendu :

Le numéro de la chambre ne correspond pas, l'accès est refusé avec un message d'erreur.



III. Scénario idéal Ménage

Requête :

```
SELECT nettoyer.idnettoyage, chambre.numerochambre, employee.idemploye
FROM nettoyer
JOIN chambre ON nettoyer.idchambre = chambre.idchambre
JOIN employee ON nettoyer.idemploye = employee.idemploye
WHERE nettoyer.idnettoyage = %s AND chambre.numerochambre = %s AND
employee.idemploye = %s
```

Explication :

Contexte : Lorsqu'un employé tente d'accéder à une chambre pour effectuer le ménage, ses informations doivent être validées.

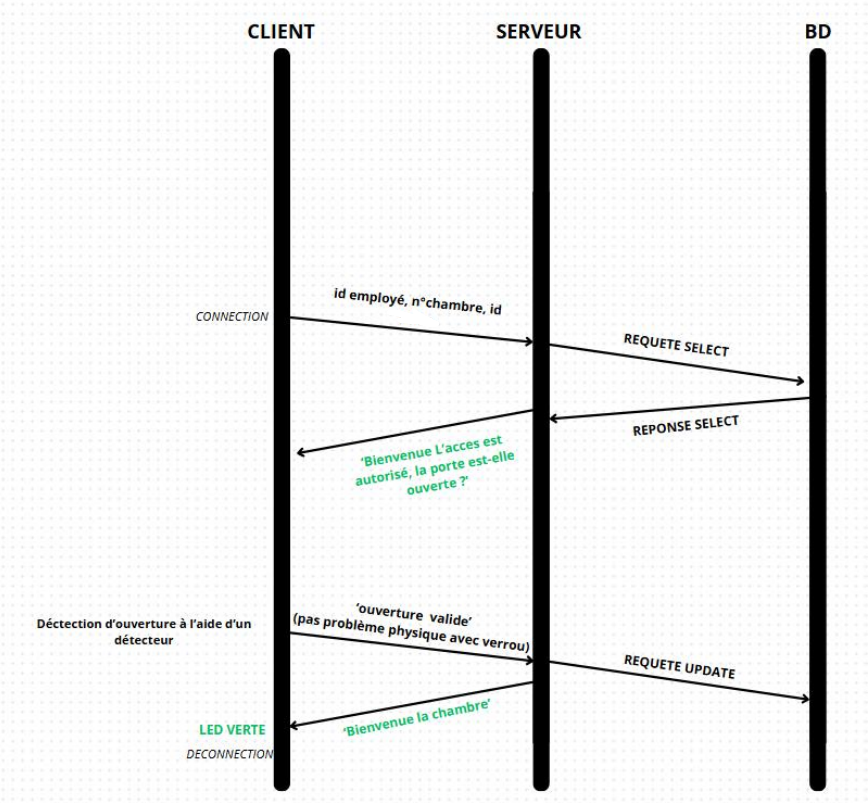
But :

Vérifier que l'idEmploye fourni correspond à une tâche de nettoyage associée dans la base de données.

Vérifier que la chambre (numeroChambre) et la carte d'accès (numeroReservation) correspondent aux données stockées dans la table Nettoyer.

Résultat attendu :

Toutes les informations (id employé, numéro de chambre, numéro de réservation) correspondent, l'accès est autorisé.



IV. Scénario idéal STAFF

Requête :

```
SELECT idemploye
FROM employee
WHERE idemploye = %s
```

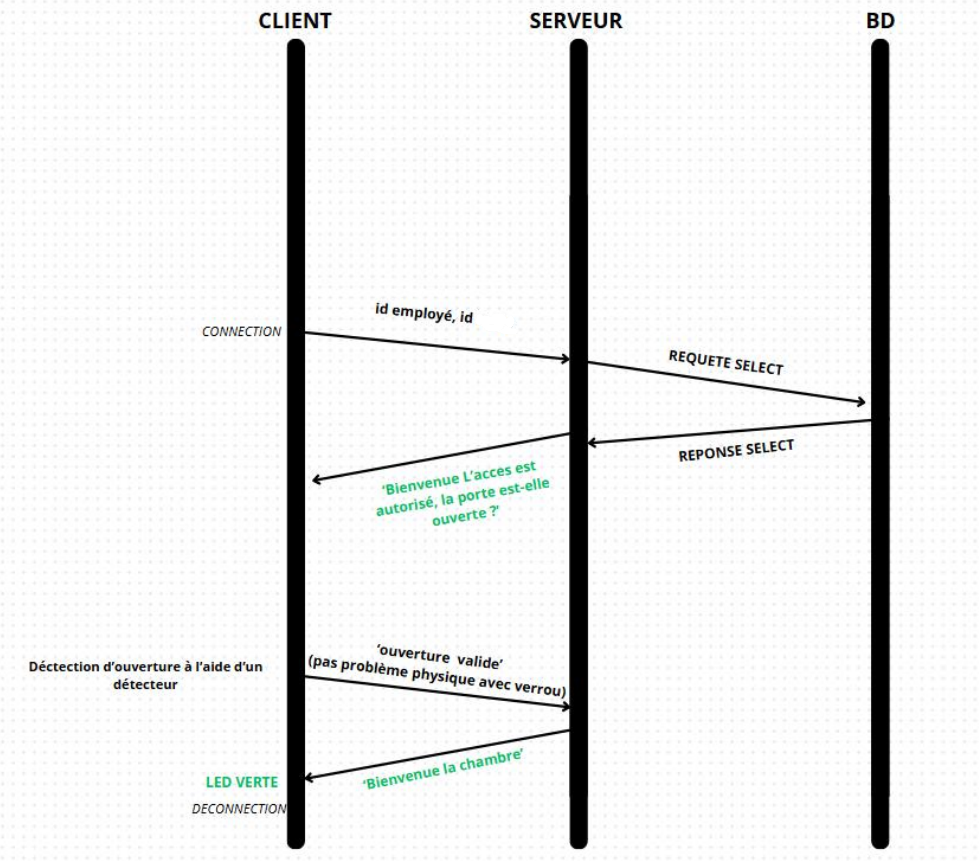
Explication :

Contexte : Lorsqu'un administrateur se connecte, son identifiant doit être validés.

But : Cette requête extrait l'idEmploye.

Résultat attendu :

- Si le numéro de réservation dans la base correspond à celui envoyé par le client, l'accès est autorisé.
- Sinon, le serveur retourne une erreur indiquant que les identifiants sont incorrects.



7. Test du serveur et du client

I. Test du serveur avec netcat :

Pour effectuer nos tests, nous avons utilisé l'outil **Netcat** pour simuler un client et vérifier les interactions avec notre serveur Python.

Étape 1 : Lancer le serveur Python

Tout d'abord, nous démarrons notre serveur Python avec la commande suivante dans un terminal :

```
C:\Users\zijaw\Downloads>python ServeurTest.py
Serveur en écoute sur 127.0.0.1:8008
```

Le serveur se met alors en écoute sur l'adresse **127.0.0.1** et le port **8008**, prêt à recevoir des connexions clients.

Étape 2 : Simuler un client avec Netcat

Pour tester le comportement du serveur, nous utilisons **Netcat** pour émuler un client. Depuis un autre terminal, nous nous connectons au serveur avec la commande suivante :

```
C:\Users\zijaw>echo "user1,101,1001,abcd1234" | ncat 127.0.0.1 8008
Bienvenue L'acces est autorise.
```

Cette commande envoie les informations d'identification sous la forme d'une chaîne de caractères au serveur, où **"admin"** est le nom d'utilisateur, suivi de **room_number**, **portid**, et **caretid**

Étape 3 : Réception et affichage côté serveur

Dès qu'un client se connecte, le serveur Python affiche des informations détaillées pour le débogage, y compris l'adresse IP du client et les données reçues. Par exemple :

```
Nouvelle connexion de  ('127.0.0.1', 54603)
Reçu du client : user1,101,1001,abcd1234
```

Cela nous permet de vérifier que les données sont bien transmises et que le serveur les interprète correctement.

Autre exemple de test

Si nous simulons un autre utilisateur avec **Netcat** en changeant les informations d'identification, comme ceci :

```
C:\Users\zijaw>echo "admin,0,1001,asds1234" | ncat 127.0.0.1 8008
Erreur portid ou caretid incorrect.
```

II. Test du Client avec netcat :

Pour effectuer les tests de notre **client Java** en simulant un serveur, nous avons utilisé l'outil **Netcat**. Cet outil permet de reproduire le comportement d'un serveur et d'observer les données envoyées par le client.

Étape 1 : Simuler un serveur avec Netcat

Tout d'abord, nous devons émuler un serveur qui écoute sur un port spécifique. Pour cela, nous utilisons **Netcat** en exécutant la commande suivante dans un terminal :

```
C:\Users\zijaw\Downloads>ncat -l -p 8008
admin,0,1001,abcd1234
```

Cette commande place **Netcat** en mode écoute sur le port **8008**, prêt à recevoir les données envoyées par notre client Java.

Étape 2 : Lancer le client Java

Ensuite, nous exécutons notre client Java. Ce client se connecte au serveur **Netcat** et envoie des informations d'identification sous un format prédéfini. Voici un exemple d'exécution du client Java :

```
Connexion établie avec le serveur...
Données envoyées au serveur : admin,0,1001,abcd1234
```

Étape 3 : Répondre au client depuis Netcat

Après avoir reçu les données, vous pouvez simuler une réponse du serveur en tapant directement dans le terminal **Netcat**.

III. Connexion de notre serveur à notre client :

Nous lançons notre serveur qui à un délai de fermeture de 60 secondes sans réponses et nous donnons en paramètres les informations nécessaires pour avoir une réponse du serveur pour les différents utilisateurs que ce soit les employé qui doivent nettoyer, un client ou encore un admin qui a accès à toutes les chambres :

```
Connexion au serveur établie.
Données envoyées : E001    ,102,N001    ,employee
Réponse du serveur : Bienvenue employé E001     ! Tâche de nettoyage autorisée pour la chambre 102.
```

```
Connexion au serveur établie.
Données envoyées : CL2807,101,R001        ,client
Réponse du serveur : Bienvenue client CL2807 !
```

```
Connexion au serveur établie.
Données envoyées : E001    ,None,R001    ,admin
Réponse du serveur : Bienvenue administrateur E001     !
```

8. Connexion à la base de données

I. Comment se connecter à la base de données :

Pour se connecter à la base de données via le site web, nous utilisons directement l'outil proposé sur le site de notre hébergeur alwaysdata qui va nous fournir un outil pour héberger la base de donnée:

Avec cet outil nous pouvons nous connecter sur pgAdmin a la base de données qui sera lié directement à alwaysdata puis à l'aide du php nous utilisons la bibliothèque PDO pour se connecter à PostgreSQL. Cela va permettre de récupérer les données voulus en les stockant pour par la suite les utiliser pour des vérifications ou entrer des nouvelles valeurs dans la base de données. (cf fichier db_connect)

Pour ce qui est de notre serveur python, nous utlisons la bibliothèque psycopg2 qui permet aussi de se connecter à une base PostgreSQL et d'éxécuter des requêtes SQL.

Pour les 2 cas il faut donner en paramètre pour établir la connexion, le nom de l'host, le nom de la base de donnée, le nom de l'utilisateur et le mot de passe de l'utilisateur qui sont tous sur le site de l'hébergeur :
En Python :

```
        # Connexion à la base de données PostgreSQL
```

```
        conn_db = psycopg2.connect("host='postgresql-tktm.alwaysdata.net'
dbname='tktm_bd' user='tktm' password='Pokemon0000!@'")
```

En PHP :

```php
<?php
$host = 'postgresql-tktm.alwaysdata.net';
$dbname = 'tktm_bd';
$username = 'tktm';
$password = 'Pokemon0000!@';

try {
    $pdo = new PDO("pgsql:host=$host;dbname=$dbname", $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    // Utilisez `die` uniquement en développement
    die("Erreur de connexion : " . $e->getMessage());
}
?>
```

II. Les différentes requêtes SQL importantes :

Requêtes SQL dans le PHP :

1. Connexion de l'utilisateur

```sql
SELECT idclient, prenomclient, password
FROM Client
WHERE mail = ?;
```

Explication :

- Cette requête vérifie si un utilisateur existe en fonction de son adresse email (mail).
- Elle récupère l'ID, le prénom et le mot de passe associés pour valider la connexion.
- Utilisée dans le fichier login.php pour authentifier les utilisateurs.

2. Liste des Réservations d'un Client

```sql
SELECT *
FROM Reservation
WHERE idClient = :idClient;
```

Explication :

- Cette requête récupère toutes les réservations associées à un client spécifique (idClient).
- Utilisée dans le tableau de bord pour afficher les réservations du client connecté.

3. Récupération des Chambres Disponibles

```
SELECT Chambre.idchambre, Chambre.numerochambre, TypeChambre.nomtype AS
typechambre, TypeChambre.prix AS prix
FROM Chambre
LEFT JOIN TypeChambre ON Chambre.idtypechambre = TypeChambre.idtypechambre
WHERE Chambre.etatchambre = 'free';
```

- **Explication :**
    - Cette requête récupère toutes les chambres disponibles (etatChambre = 'free') en affichant leur type et leur prix.
    - Utilisée dans view_rooms.php pour lister les chambres disponibles à la réservation.

4. Insertion d'une Nouvelle Réservation

```
INSERT INTO Reservation (idReservation, dateDebut, dateFin, typeReserver,
etatReserver, idClient, idChambre)
VALUES (?, ?, ?, ?, ?, ?, ?);
```

Explication :

- Cette requête ajoute une nouvelle réservation dans la base de données avec les informations suivantes :
    - ID de réservation (idReservation), dates de début et de fin (dateDebut, dateFin), type de réservation (typeReserver), état de réservation (etatReserver), ID du client (idClient), et ID de la chambre (idChambre).
- Utilisée dans view_rooms.php ou dashboard.php pour permettre aux utilisateurs de réserver une chambre.

5. Mise à jour de l'état d'une chambre après une réservation

```
UPDATE Chambre
SET etatChambre = 'rsrv'
WHERE idChambre = ?;
```

Explication :

- Cette requête met à jour l'état de la chambre réservée pour indiquer qu'elle n'est plus disponible (etatChambre = 'rsrv').
- Utilisée après l'insertion d'une réservation pour bloquer la chambre.

6. Liste des Clients (Recherche Dynamique)

```
SELECT *
FROM Client
```

```
WHERE idclient = :search
   OR LOWER(nomclient) LIKE LOWER(:searchLike)
   OR LOWER(prenomclient) LIKE LOWER(:searchLike)
   OR LOWER(mail) LIKE LOWER(:searchLike)
   OR tel LIKE :search;
```

- **Explication :**
    - Cette requête recherche un client en fonction de son ID, son nom, son prénom, son email ou son numéro de téléphone.
    - Utilisée dans l'interface administrateur pour rechercher les clients dynamiquement.

Requêtes SQL dans le serveur Python :

1. **Vérification des Informations d'un Administrateur**

```
SELECT idemploye
FROM employee
WHERE idemploye = %s
```

Explication :

- Cette requête vérifie que l'identifiant de l'employé (idEmploye) et son numéro de carte (numeroCarte) correspondent aux informations stockées.
- Utilisée pour authentifier les administrateurs qui accèdent au système.

2. **Vérification des Informations d'un Client**

```
SELECT chambre.numerochambre, reservation.idreservation
FROM reservation
JOIN chambre ON reservation.idchambre = chambre.idchambre
WHERE reservation.idclient = %s
AND reservation.etatreserver = 'rsv'
```

Explication :

- Cette requête valide qu'un client possède une réservation active pour une chambre donnée, et que son numéro de carte est correct.
- Utilisée pour authentifier les clients qui accèdent à leur chambre.

3. **Vérification des Informations d'un Employé de Nettoyage**

```
SELECT chambre.numerochambre, reservation.idreservation
FROM reservation
JOIN chambre ON reservation.idchambre = chambre.idchambre
WHERE reservation.idclient = %s
AND reservation.etatreserver = 'rsv'
```

Explication :

- Cette requête vérifie qu'un employé est assigné à une tâche de nettoyage pour une chambre spécifique et valide son numéro de carte.
- Utilisée pour autoriser l'accès des employés aux chambres pour le ménage.

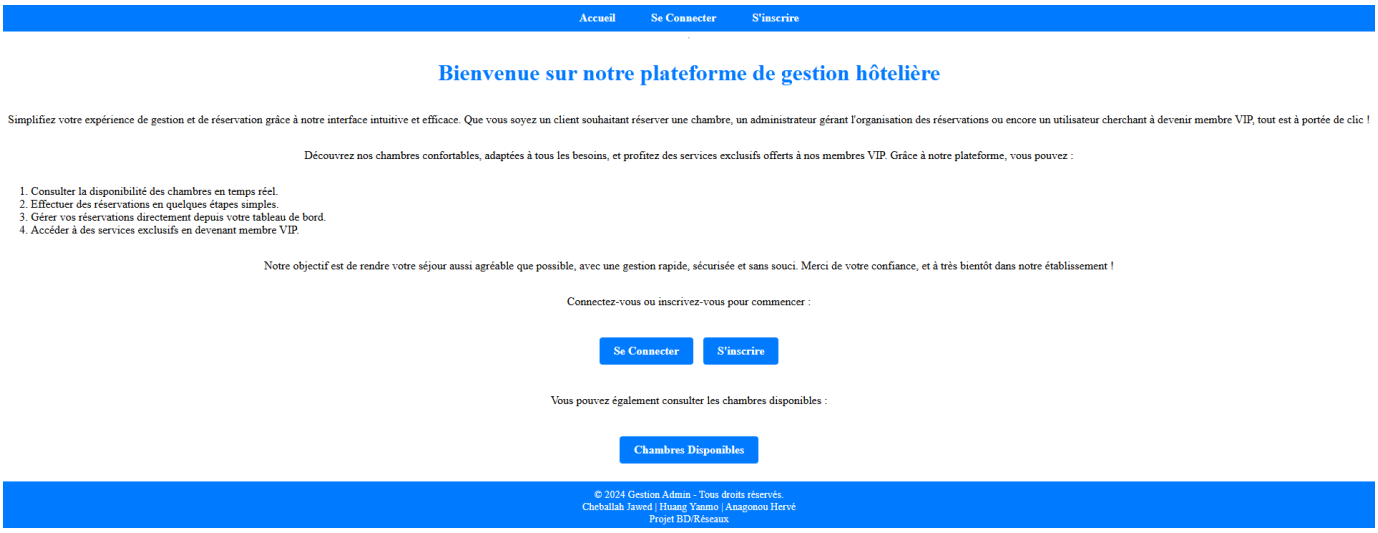
4. **Mise à Jour de l'État d'une Chambre après Nettoyage**

```
UPDATE Chambre
SET etatChambre = 'free'
WHERE idChambre = %s;
```

- **Explication :**
  - Cette requête met à jour l'état d'une chambre après qu'un employé a signalé que le ménage est terminé (etatChambre = 'free').
  - Utilisée pour rendre la chambre disponible à nouveau.

9. Site web

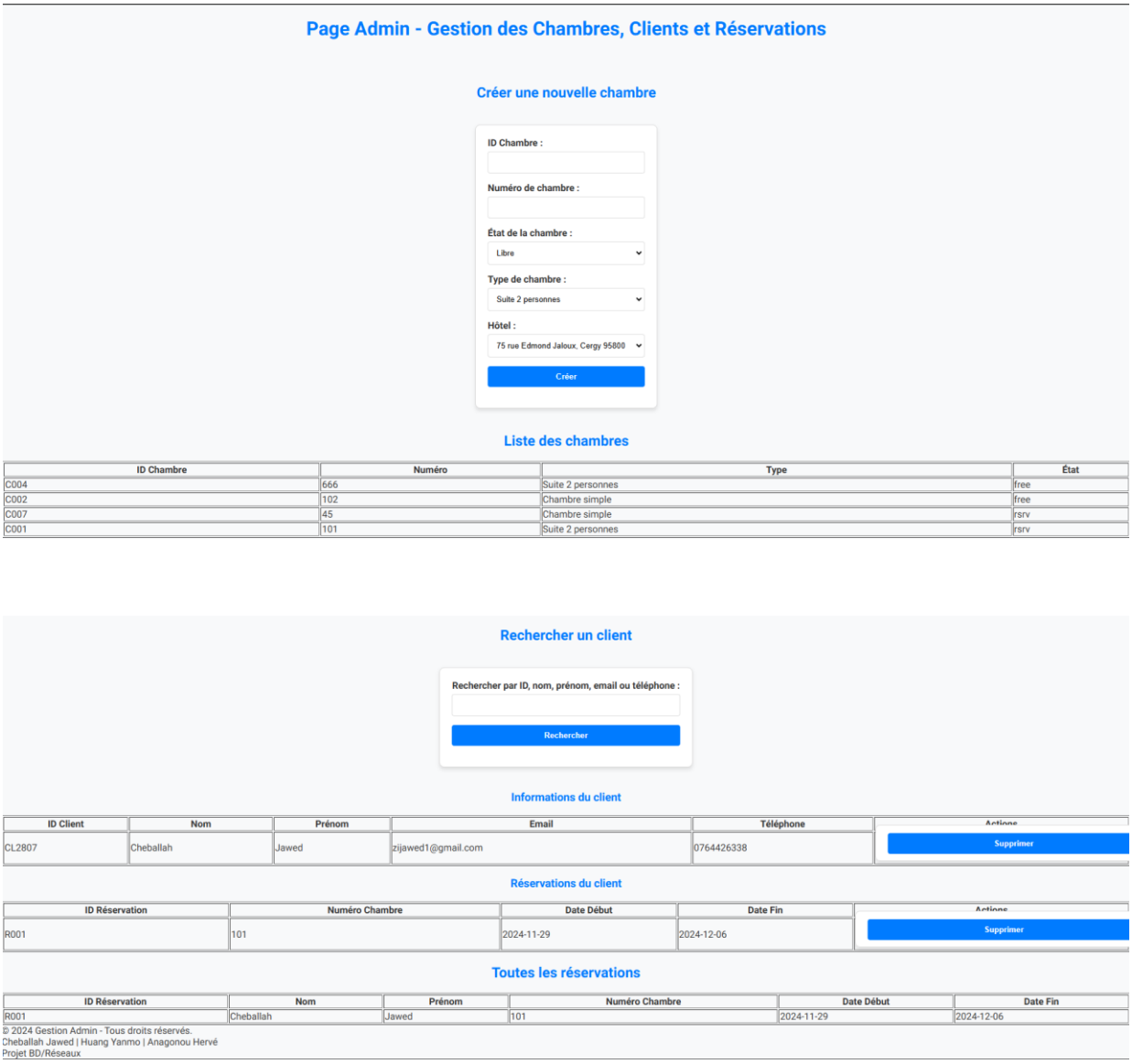
Notre site web contient 2 pages principales :
La première est la page utilisateur avec un espace de connexion, inscription :



Puis cela nous amène sur la page dashboard qui nous sert à gérer les réservations. Nous pouvons aussi prendre des réservations sur une autre pages qui va nous montrer la liste des réservations :

La deuxième est une page administrateur qui nous permets de mettre en ligne des chambres, vérifier

les données d'un utilisateur, supprimer et modifier des réservations :



Annexe

1. *Code SQL :*

```
CREATE TABLE Hotel (
    idHotel CHAR(8) PRIMARY KEY,
    adresse VARCHAR(100) NOT NULL,
    telHotel CHAR(12) NOT NULL,
    mailHotel VARCHAR(100) NOT NULL
);
CREATE TABLE Employee (
    idEmploye CHAR(8) PRIMARY KEY,
    nomEmploye VARCHAR(40) NOT NULL,
    prenomEmploye VARCHAR(40) NOT NULL,
    telEmploye CHAR(12) NOT NULL,
    postEmploye VARCHAR(40) NOT NULL,
    idHotel CHAR(8),
    FOREIGN KEY (idHotel) REFERENCES Hotel(idHotel)
);
CREATE TABLE TypeChambre (
    idTypeChambre CHAR(8) PRIMARY KEY,
    nomType VARCHAR(40) NOT NULL,
    prix FLOAT NOT NULL
);
CREATE TABLE Chambre (
    idChambre CHAR(8) PRIMARY KEY,
    numeroChambre VARCHAR(4) NOT NULL,
    etatChambre CHAR(4) NOT NULL CHECK (etatChambre IN ('rsrv',
'free')),
    idTypeChambre CHAR(8),
    idHotel CHAR(8),
    FOREIGN KEY (idTypeChambre) REFERENCES TypeChambre(idTypeChambre),
    FOREIGN KEY (idHotel) REFERENCES Hotel(idHotel)
);
CREATE TABLE Client (
    idClient CHAR(10) PRIMARY KEY,
    nomClient VARCHAR(40) NOT NULL,
    prenomClient VARCHAR(40) NOT NULL,
    tel CHAR(12) NOT NULL,
    mail VARCHAR(100) NOT NULL
);
CREATE TABLE ClientVIP (
    idClient CHAR(10) PRIMARY KEY,
    serviceExclusif VARCHAR(40),
    codeVIP CHAR(12) NOT NULL,
```

```
    FOREIGN KEY (idClient) REFERENCES Client(idClient)
);
CREATE TABLE ClientRegulier (
    idClient CHAR(10) PRIMARY KEY,
    pointFidelite INT NOT NULL,
    promotion FLOAT NOT NULL CHECK (promotion > 0 AND promotion < 1),
    FOREIGN KEY (idClient) REFERENCES Client(idClient)
);
CREATE TABLE Reservation (
    idReservation CHAR(12) PRIMARY KEY,
    dateDebut DATE NOT NULL,
    dateFin DATE NOT NULL,
    typeReserver CHAR(3) NOT NULL CHECK (typeReserver IN ('srp',
'enl')),
    etatReserver CHAR(3) NOT NULL CHECK (etatReserver IN ('rsv',
'att', 'ann')),
    idClient CHAR(10),
    idChambre CHAR(8),
    FOREIGN KEY (idClient) REFERENCES Client(idClient),
    FOREIGN KEY (idChambre) REFERENCES Chambre(idChambre)
);
CREATE TABLE Nettoyage (
    idNettoyage CHAR(8) PRIMARY KEY,
    typeNettoyage VARCHAR(20) NOT NULL,
    prixNettoyage FLOAT NOT NULL
);
CREATE TABLE Demande (
    idDemande CHAR(8) PRIMARY KEY,
    idNettoyage CHAR(8),
    idClient CHAR(10),
    dateDemander DATE NOT NULL,
    FOREIGN KEY (idNettoyage) REFERENCES Nettoyage(idNettoyage),
    FOREIGN KEY (idClient) REFERENCES Client(idClient)
);
CREATE TABLE Nettoyer (
    idNettoyage CHAR(8),
    idChambre CHAR(8),
    idEmploye CHAR(8),
    dateNettoyer DATE NOT NULL,
    PRIMARY KEY (idNettoyage, idChambre, idEmploye),
    FOREIGN KEY (idNettoyage) REFERENCES Nettoyage(idNettoyage),
    FOREIGN KEY (idChambre) REFERENCES Chambre(idChambre),
    FOREIGN KEY (idEmploye) REFERENCES Employee(idEmploye)
);

ALTER TABLE Employee
ADD numeroCarte CHAR(8);
```

```sql
ALTER TABLE Reservation
ADD numeroCarte CHAR(8);
```

2. Jeu de donnée :

```sql
INSERT INTO Hotel (idHotel, adresse, telHotel, mailHotel)
VALUES
('H001', '75 rue Edmond Jaloux, Cergy 95800', '+33156453236',
'jaloux.hotel@theKTM.com'),
('H002', '10 Avenue des Champs, Paris 75008', '+33144783344',
'champs.hotel@theKTM.com');

INSERT INTO Employee (idEmploye, nomEmploye, prenomEmploye,
telEmploye, postEmploye, idHotel)
VALUES
('E001', 'James', 'Lebron', '+33689564236', 'Femme de ménage',
'H001'),
('E002', 'Dupont', 'Marie', '+33612345678', 'Réceptionniste', 'H002');

INSERT INTO TypeChambre (idTypeChambre, nomType, prix)
VALUES
('T001', 'Suite 2 personnes', 875.99),
('T002', 'Chambre simple', 200.50);

INSERT INTO Chambre (idChambre, numeroChambre, etatChambre,
idTypeChambre, idHotel)
VALUES
('C001', '101', 'free', 'T001', 'H001'),
('C002', '102', 'rsrv', 'T002', 'H001'),
('C003', '201', 'free', 'T001', 'H002');

INSERT INTO Client (idClient, nomClient, prenomClient, tel, mail)
VALUES
('CL001', 'Dupont', 'Paul', '+33765426338', 'paul.dupont@gmail.com'),
('CL002', 'Martin', 'Sophie', '+33678965432',
'sophie.martin@yahoo.com');

INSERT INTO ClientVIP (idClient, serviceExclusif, codeVIP)
VALUES
('CL001', 'Accès piscine', 'VIP123456789');
```

```sql
INSERT INTO ClientRegulier (idClient, pointFidelite, promotion)
VALUES
('CL002', 75000, 0.30);

INSERT INTO Reservation (idReservation, dateDebut, dateFin,
typeReserver, etatReserver, idClient, idChambre)
VALUES
('R001', '2024-10-01', '2024-10-08', 'srp', 'rsv', 'CL001', 'C002'),
('R002', '2024-10-15', '2024-10-20', 'enl', 'att', 'CL002', 'C001');

INSERT INTO Nettoyage (idNettoyage, typeNettoyage, prixNettoyage)
VALUES
('N001', 'Profond', 150.00),
('N002', 'Superficiel', 75.00);

INSERT INTO Demande (idDemande, idNettoyage, idClient, dateDemander)
VALUES
('D001', 'N001', 'CL001', '2024-10-02'),
('D002', 'N002', 'CL002', '2024-10-15');

INSERT INTO Nettoyer (idNettoyage, idChambre, idEmploye, dateNettoyer)
VALUES
('N001', 'C002', 'E001', '2024-10-09'),
('N002', 'C001', 'E002', '2024-10-21');
```

3. Code serveur :

```python
import socket
import psycopg2
import logging
import time

# Configurations globales
HOST = "127.0.0.1"
PORT = 8008
BUFFER_SIZE = 1024
INACTIVITY_TIMEOUT = 60  # Temps en secondes avant fermeture pour
inactivité
```

```
# Configurer les logs pour le serveur
logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s', filename='server.log')

def handle_client(conn, addr, cursor):
    """Gestion des interactions avec un client ou un employé."""
    try:
        logging.info(f"Nouvelle connexion de {addr}")
        conn.settimeout(5)

        # Réception des données
        try:
            data = conn.recv(BUFFER_SIZE).decode("utf-8").strip()
            if not data:
                raise ValueError("Données vides reçues du client")
            logging.info(f"Données reçues : {data}")
        except socket.timeout:
            logging.warning(f"Client {addr} n'a pas répondu dans les
délais. Connexion fermée.")
            conn.close()
            return
        except (socket.error, ValueError) as e:
            logging.error(f"Erreur lors de la réception des données du
client {addr} : {str(e)}")
            conn.close()
            return

        # Analyse et validation des données
        try:
            userid, room_number, idreservation, role = data.split(',')
            if len(userid) > 10 or len(room_number) > 4 or
len(idreservation) > 12 or len(role) > 10:
                raise ValueError("Données trop longues")
        except ValueError as e:
            logging.error(f"Données mal formatées ou invalides de la
part du client {addr} : {str(e)}")
            conn.sendall("Erreur : Données invalides.".encode("utf-
8"))
            return

        # Gestion des rôles
        if role == "admin":
            # Administrateur
            cursor.execute("""
                SELECT idemploye
                FROM employee
                WHERE idemploye = %s
```

```
            """, (userid,))
            result = cursor.fetchone()

            if result:
                response = f"Bienvenue administrateur {userid} !"
            else:
                response = "Erreur : Identifiant administrateur
incorrect."

        elif role == "client":
            # Client
            cursor.execute("""
                SELECT chambre.numerochambre,
reservation.idreservation
                FROM reservation
                JOIN chambre ON reservation.idchambre =
chambre.idchambre
                WHERE reservation.idclient = %s
                  AND reservation.etatreserver = 'rsv'
            """, (userid,))
            result = cursor.fetchone()

            if result:
                db_room_number, db_id_reservation = result
                if db_room_number == room_number and db_id_reservation
== idreservation:
                    response = f"Bienvenue client {userid} !"
                else:
                    response = "Erreur : Numéro de chambre ou
réservation incorrect."
            else:
                response = "Erreur : Aucun enregistrement trouvé pour
cet identifiant client."

        elif role == "employee":
            # Employé (Nettoyage)
            cursor.execute("""
                SELECT nettoyer.idnettoyage, chambre.numerochambre,
employee.idemploye
                FROM nettoyer
                JOIN chambre ON nettoyer.idchambre = chambre.idchambre
                JOIN employee ON nettoyer.idemploye =
employee.idemploye
                WHERE nettoyer.idnettoyage = %s AND
chambre.numerochambre = %s AND employee.idemploye = %s
            """, (idreservation, room_number, userid))
            result = cursor.fetchone()
```

```
            if result:
                response = f"Bienvenue employé {userid} ! Tâche de
nettoyage autorisée pour la chambre {room_number}."
            else:
                response = "Erreur : Aucun enregistrement trouvé ou
données incorrectes."

        else:
            response = "Erreur : Rôle non reconnu. Les rôles valides
sont : admin, client, employee."

        # Envoi de la réponse au client
        conn.sendall(response.encode("utf-8"))
        logging.info(f"Réponse envoyée à {addr} : {response}")

    except Exception as e:
        logging.error(f"Erreur inattendue avec le client {addr} :
{str(e)}")
        conn.sendall("Erreur : Problème dans la
communication.".encode("utf-8"))
    finally:
        conn.close()
        logging.info(f"Connexion fermée avec {addr}")

def main():
    """Démarrer le serveur."""
    try:
        # Connexion à la base de données PostgreSQL
        conn_db = psycopg2.connect("host='postgresql-
tktm.alwaysdata.net' dbname='tktm_bd' user='tktm'
password='Pokemon0000!@'")
        cursor = conn_db.cursor()

        # Création du socket serveur
        server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        try:
            server_socket.bind((HOST, PORT))
        except OSError as e:
            logging.error(f"Port {PORT} déjà utilisé. Veuillez
vérifier.")
            return

        server_socket.listen(5)
        logging.info(f"Serveur démarré sur {HOST}:{PORT}")
```

```
        last_activity_time = time.time()  # Enregistre l'heure de la
dernière activité

        while True:
            # Vérification du temps d'inactivité
            current_time = time.time()
            if current_time - last_activity_time > INACTIVITY_TIMEOUT:
                logging.info("Fermeture du serveur après période
d'inactivité.")
                break

            try:
                # Accepter une connexion client avec un timeout ajusté
                server_socket.settimeout(INACTIVITY_TIMEOUT -
(current_time - last_activity_time))
                conn, addr = server_socket.accept()
                last_activity_time = time.time()  # Mise à jour de
l'heure de la dernière activité
                handle_client(conn, addr, cursor)
            except socket.timeout:
                continue  # Aucun client ne s'est connecté, retourner
à la boucle principale

    except psycopg2.Error as db_error:
        logging.error(f"Erreur base de données : {str(db_error)}")
    except Exception as e:
        logging.error(f"Erreur serveur : {str(e)}")
    finally:
        if conn_db:
            conn_db.close()
        server_socket.close()
        logging.info("Serveur arrêté.")

if __name__ == "__main__":
    main()
```

4. Code client :

```
package com.example.credentials;

import java.io.*;
```

```
import java.net.Socket;
import java.net.SocketTimeoutException;

public class Client {
    private String serverAddress;
    private int serverPort;

    // Constructeur
    public Client(String serverAddress, int serverPort) {
        this.serverAddress = serverAddress;
        this.serverPort = serverPort;
    }

    // Méthode pour envoyer les informations d'identification
    public void sendCredentials(UserCredentials credentials) {
        try (Socket socket = new Socket(serverAddress, serverPort)) {
            socket.setSoTimeout(5000); // Timeout de 5 secondes pour la
lecture

            try (BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
                 BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()))) {

                System.out.println("Connexion au serveur établie.");

                // Format des données à envoyer
                String data = credentials.getFormattedCredentials();
                if (data.length() > 100) {
                    System.err.println("Erreur : Les données sont trop
volumineuses pour être envoyées.");
                    return;
                }

                writer.write(data);
                writer.newLine();
                writer.flush();
                System.out.println("Données envoyées : " + data);

                // Lecture de la réponse du serveur
                try {
                    String response = reader.readLine();
                    if (response == null) {
                        System.err.println("Erreur : Le serveur a fermé la
connexion.");
                    } else {
```

```
                        System.out.println("Réponse du serveur : " +
response);
                    }
                } catch (SocketTimeoutException e) {
                    System.err.println("Erreur : Le serveur met trop de temps
à répondre.");
                }

            }
        } catch (IOException e) {
            System.err.println("Erreur lors de la communication avec le
serveur : " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        // Paramètres du serveur
        String serverIP = "127.0.0.1";
        int serverPort = 8008;

        // Informations d'identification
        UserCredentials credentials = new UserCredentials("CL2807", "101",
"R001", "client");

        // Envoi des données
        Client client = new Client(serverIP, serverPort);
        client.sendCredentials(credentials);
    }
}
```