Rapport CSI 2532 - Livrable 2

Professeur : **Fadi Malek**

Étudiants : Sasha Milne (300366924), Sofia El ouazzani (300367668)

1. SGBD et Langages de Programmation Utilisés

a) Système de Gestion de Base de Données (SGBD)

PostgreSQL : Utilisé pour gérer la base de données relationnelle de l'application.

b) Langages de Programmation

Javascript (JEE - Java Enterprise Edition) : Utilisé pour le backend et la logique métier.
HTML, CSS et JavaScript : Pour le frontend de l'application.
SQL : Pour la création et la gestion de la base de données.

2. Étapes d'Installation de l'Application

Prérequis :
PostgreSQL installé et configuré.
NodeJS installé avec ses dépendances
(https, pg, express, express-session, dot-env)
Maven installé pour la gestion des dépendances.

Github Repository:

Étapes :
1. Configurer la base de données :
2. Importer le fichier ehotelschema.sql dans PostgreSQL pour créer les tables nécessaires.
3. Vérifier que le service PostgreSQL fonctionne et est accessible.

Configurer l'application :
Modifier .env si nécessaire pour ajuster les paramètres de connexion à la base de données.
Vérifier que postgresql-42.5.1.jar (le driver PostgreSQL) est bien ajouté dans WEB-INF/lib/.

Lancer l'application
Exécuter le projet avec : node app.js et accéder à http://localhost:3000 dans un navigateur

3. DDL pour la Création de la Base de Données (Implémentez la base de données)

```
CREATE TABLE ehotelschema.hotel_chain (
   chain_id INT PRIMARY KEY,
   office_address VARCHAR(100) NOT NULL,
   chain_name VARCHAR(100) NOT NULL,
   phone_number VARCHAR(20) NOT NULL,
   email VARCHAR(50) NOT NULL
);
```

```
CREATE TABLE ehotelschema.hotel (
    hotel_id INT PRIMARY KEY,
    chain_id INT NOT NULL,
    hotel_address VARCHAR(100) NOT NULL,
    phone_number VARCHAR(20) NOT NULL,
    email VARCHAR(50) NOT NULL,
    rating DECIMAL(2,1) NOT NULL,
    manager_id INT NOT NULL,
    FOREIGN KEY (chain_id) REFERENCES ehotelschema.hotel_chain(chain_id),
    FOREIGN KEY (manager_id) REFERENCES ehotelschema.employee(SIN)
);

CREATE TABLE ehotelschema.room (
    room_number INT,
    hotel_id INT NOT NULL,
    room_type VARCHAR(50) NOT NULL,
    price DECIMAL(6,2) NOT NULL,
    FOREIGN KEY (hotel_id) REFERENCES ehotelschema.hotel(hotel_id),
    PRIMARY KEY(room_number, hotel_id)
);

CREATE TABLE ehotelschema.client (
    SIN INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    registration_date DATE NOT NULL,
    phone_number VARCHAR(20) NOT NULL,
    email VARCHAR(50) NOT NULL
);

CREATE TABLE ehotelschema.employee (
    SIN INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    employee_role VARCHAR(50) NOT NULL,
    works_at INT NOT NULL,
    email VARCHAR(50) NOT NULL,
    phone_number VARCHAR(20) NOT NULL,
    FOREIGN KEY (works_at) REFERENCES ehotelschema.hotel(hotel_id)
);

CREATE TABLE ehotelschema.reservation (
    reservation_id INT PRIMARY KEY,
    room_number INT NOT NULL,
    hotel_id INT NOT NULL;
    SIN INT NOT NULL,
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    reservation_date DATE NOT NULL,
```

```
   FOREIGN KEY (room_number, hotel_id) REFERENCES ehotelschema.room(room_number,
hotel_id),
   FOREIGN KEY (SIN) REFERENCES ehotelschema.client(SIN)
);
```

4. Insérez dans votre base de données.

Nous avons inséré 5 chaînes, chacune comportant 8 hôtels.

Chaque hôtel dispose également de 5 chambres.

Les scripts permettant de générer les commandes SQL sont disponibles sur le dépôt GitHub, sous database/sql-generators.

Pour vérifier que la base de donnée est peuplée

```
SELECT * FROM ehotelschema.hotel_chain;
SELECT * FROM ehotelschema.room;
SELECT * FROM ehotelschema.
```

5. Trigger Functions

```
CREATE OR REPLACE FUNCTION update_registration_date()
RETURNS TRIGGER AS $$
BEGIN
   NEW.date_enregistrement = CURRENT_DATE;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_client_registration
BEFORE UPDATE ON ehotelschema.client
FOR EACH ROW
EXECUTE FUNCTION update_registration_date();


CREATE OR REPLACE FUNCTION check_reservation_dates()
RETURNS TRIGGER AS $$
BEGIN
   IF NEW.date_de_fin <= NEW.date_de_debut THEN
      RAISE EXCEPTION 'La date de fin doit être postérieure à la date de début.';
   END IF;
   RETURN NEW;
END;
```

$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_reservation_dates
BEFORE INSERT OR UPDATE ON ehotelschema.reservation
FOR EACH ROW
EXECUTE FUNCTION check_reservation_dates();

CREATE TRIGGER validate_reservation_dates
BEFORE INSERT OR UPDATE ON ehotelschema.reservation

5. Indices

CREATE INDEX idx_room_hotel_id ON ehotelschema.room (hotel_id);
Accélère la recherche de chambres par hôtel (commun pour les annonces)

CREATE INDEX idx_reservation_availability
ON ehotelschema.reservation (hotel_id, room_number, check_in_date, check_out_date);
Accélère la vérification de la disponibilité des chambres

CREATE UNIQUE INDEX idx_client_email ON ehotelschema.client(email);
Nous avons besoin de « email » comme index car nous effectuons beaucoup de recherches par e-mail lorsque nous utilisons la fonction de connexion

6. Application Web

On a pas réussi à implémenter toutes les fonctionnalités mais les utilisateurs (employee/client) peuvent connecter, créer et voir les réservations, modifier leurs profils, etc.

We finished the reservation with the client, and both client and employee have their own dashboards. A client can make a registration to any room if it has a valid session. Unfortunately, we were a small group and did not have the time to make a comprehensive UI for the complete functionality of the e-hotel application. The rest of the database is populated and complete, which means it would be a matter of implementing another 5-10 html pages and the backend logic to support them. If the reservations pages don't work you can use these existing accounts.

Client login:
email: a.sasha.milne@gmail.com
SIN: 0

Employee Login
email: admin@ehotel.ca
SIN: 0

7. Vues Sql

Vue 1: la première vue est le nombre de chambres disponibles par zone.

```
CREATE OR REPLACE VIEW ehotelschema.available_rooms_by_area AS
SELECT h.hotel_address,
      COUNT(*) AS available_rooms
FROM ehotelschema.room r
JOIN ehotelschema.hotel h ON r.hotel_id = h.hotel_id
WHERE NOT EXISTS (
  SELECT 1
  FROM ehotelschema.reservation res
  WHERE res.hotel_id = r.hotel_id
    AND res.room_number = r.room_number
    AND CURRENT_DATE < res.check_out_date
    AND CURRENT_DATE >= res.check_in_date
)
GROUP BY h.hotel_address;
```

Vue 2: la deuxième vue est la capacité de toutes les chambres d’un hôtel spécifique.

```
CREATE OR REPLACE VIEW ehotelschema.total_capacity_per_hotel AS
SELECT
  h.hotel_id,
  h.hotel_address,
  SUM(r.capacity) AS total_capacity
FROM ehotelschema.room r
JOIN ehotelschema.hotel h ON r.hotel_id = h.hotel_id
GROUP BY h.hotel_id, h.hotel_address;
```