Accord d'éthique personnelle concernant les travaux universitaires

Projet de groupe

Nous soumettons ce travail et attestons que nous avons appliqué toutes les règles appropriées de citation et de référencement en usage à l'Université d'Ottawa https://www.uottawa.ca/etudiants-actuels/integrite-academique. Nous attestons que ce travail est conforme au règlement sur l'intégrité académique de l'Université d'Ottawa. Nous comprenons que cette tâche ne sera pas acceptée ou notée si elle est soumise sans la signature de tous les membres du groupe.

MAI ANH HOANG
Nom, lettres majuscules

300278143
Numéro d'étudiant

Signature

04/06/2024
Date

VIET TIEN DANG
Nom, lettres majuscules

300229069
Numéro d'étudiant

Signature

04/06/2024
Date

JASON DER
Nom, lettres majuscules

300311848
Numéro d'étudiant

Signature

04/06/2024
Date

CSI 2532 Final Project - Hotel App

Github Link:
https://github.com/ja-der/Hotel-Project

1. SGBD and Programming Languages Used

Our application is developed using a combination of frontend, backend, and database technologies to provide a comprehensive user solution. In this report, we detail the technologies employed in our implementation and provide insights into their roles in creating a robust and scalable application.

2. Technology Stack

2.1 Frontend

For the frontend development of our application, we chose React.js, a popular JavaScript library for building user interfaces. React.js offers several advantages, including component-based architecture, virtual DOM for efficient rendering.

2.2 Backend

The backend of our application is powered by Node.js, a runtime environment for executing JavaScript code outside a web browser. Alongside Node.js, we utilised Express.js, a minimalist web framework for Node.js, to simplify routing, middleware integration, and request handling, enabling us to implement our SQL queries and user authentication.

2.3 Database Management System (SGBD)

To store and manage our application's data, we opted for PostgreSQL, the tool recommended in class.

3. Installation Guide (Also detailed on the github)

Database Setup:

1. Please create a database, preferably named ehotel, on PostgreSQL.

2. Execute the queries provided in the data.sql file to set up the database schema and initial data.

3. Ensure that the PostgreSQL server is running on port 5432.

Backend

1. Navigate to the /backend directory in the command line interface.

2. Create an .env file in the /backend directory and set the following environment variables:

   POSTGRES_USER=your_postgresql_user
   HOST=localhost
   DATABASE=ehotel
   PASSWORD=your_postgresql_password
   jwtSecret=secret

3. Run npm i to install the required dependencies.

4. Install Nodemon by running npm i nodemon.

5. Start the backend server by running nodemon index.

The backend application should be running at http://localhost:4000.

Front End

1. Navigate to the /frontend directory in the command line interface.

2. Run npm i to install the required modules.

3. Start the frontend part by running npm start.

4. The frontend application should be accessible at http://localhost:3000.

3. Database Definition Language

All the database schema creation queries (DDL) are included in the database folder under /backend/database.

CREATE TABLE QUERIES

```
CREATE DATABASE ehotel;

-- CREATE TABLE QUERIES

-- CREATE CHAIN TABLE
```

```sql
CREATE TABLE Chain (
    ChainID SERIAL PRIMARY KEY,
    ChainName VARCHAR(255) NOT NULL,
    HeadquartersAddress VARCHAR(255) NOT NULL,
    NumberOfHotels INT NOT NULL,
    HeadquartersEmail VARCHAR(255) NOT NULL,
    HeadquartersPhoneNumber VARCHAR(20) NOT NULL
);

-- CREATE HOTEL TABLE
CREATE TABLE Hotel (
    HotelID SERIAL PRIMARY KEY,
    HotelAddress VARCHAR(255) NOT NULL,
    HotelCity VARCHAR(255) NOT NULL,
    HotelPhoneNumber VARCHAR(20) NOT NULL,
    HotelEmail VARCHAR(100) NOT NULL,
    StarRating INT NOT NULL,
    NumberOfRooms INT NOT NULL,
    ChainID INT NOT NULL,
    FOREIGN KEY (ChainID) REFERENCES Chain(ChainID) ON DELETE CASCADE
  );

  -- CREATE ROOM TABLE
CREATE TABLE Room (
    RoomID SERIAL PRIMARY KEY,
    Price DECIMAL(10, 2) NOT NULL,
    Amenities VARCHAR(255) NOT NULL,
    Capacity INT NOT NULL,
    RoomView VARCHAR(255) NOT NULL,
    Extendable VARCHAR(25) NOT NULL,
    Issues VARCHAR(255) NOT NULL,
    HotelID INT NOT NULL,
    ChainID INT NOT NULL,
    FOREIGN KEY (HotelID) REFERENCES Hotel(HotelID) ON DELETE CASCADE,
    FOREIGN KEY (ChainID) REFERENCES Chain(ChainID) ON DELETE CASCADE
);

-- CREATE CLIENT TABLE
CREATE TABLE Client (
    ClientID SERIAL PRIMARY KEY,
    ClientFirstName VARCHAR(20) NOT NULL,
    ClientLastName VARCHAR(20) NOT NULL,
    ClientAddress VARCHAR(255) NOT NULL,
```

```sql
    ClientSSN INT NOT NULL,
    RegistrationDate DATE NOT NULL,
    ClientEmail VARCHAR(100),
    ClientPassword VARCHAR(255)
);

-- CREATE RESERVATION TABLE
CREATE TABLE Reservation (
    ReservationID SERIAL PRIMARY KEY,
    CheckInDate DATE NOT NULL,
    CheckOutDate DATE NOT NULL,
    ClientID INT NOT NULL,
    HotelID INT NOT NULL,
    RoomID INT NOT NULL,
    ChainID INT NOT NULL,
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID) ON DELETE
CASCADE
);
-- CREATE EMPLOYEE TABLE
CREATE TABLE Employee (
    EmployeeID SERIAL PRIMARY KEY,
    EmployeeFirstName VARCHAR(20) NOT NULL,
    EmployeeLastName VARCHAR(20) NOT NULL,
    EmployeeAddress VARCHAR(255) NOT NULL,
    EmployeeEmail VARCHAR(100) NOT NULL,
    EmployeePassword VARCHAR(255) NOT NULL,
    EmployeeSSN INT NOT NULL,
    HotelID INT NOT NULL,
    ChainID INT NOT NULL,
    FOREIGN KEY (HotelID) REFERENCES Hotel(HotelID) ON DELETE CASCADE,
    FOREIGN KEY (ChainID) REFERENCES Chain(ChainID) ON DELETE CASCADE
);

-- CREATE RENTAL TABLE
CREATE TABLE Rental (
    RentalID SERIAL PRIMARY KEY,
    StartDate DATE,
    EndDate DATE,
    ReservationID INT,
    EmployeeID INT NOT NULL,
    ClientID INT NOT NULL,
    ChainID INT NOT NULL,
    HotelID INT NOT NULL,
```

```sql
    RoomID INT NOT NULL,
    FOREIGN KEY (ReservationID) REFERENCES Reservation(ReservationID)
ON DELETE CASCADE,
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID) ON DELETE
CASCADE,
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID) ON DELETE
CASCADE
);

-- CREATE POSITION TABLE
CREATE TABLE Position (
    JobCode SERIAL PRIMARY KEY,
    JobTitle VARCHAR(255) NOT NULL,
    Responsibilities VARCHAR(1000),
    JobLevel INT,
    HotelID INT NOT NULL,
    ChainID INT NOT NULL,
    FOREIGN KEY (HotelID) REFERENCES Hotel(HotelID) ON DELETE CASCADE,
    FOREIGN KEY (ChainID) REFERENCES Chain(ChainID) ON DELETE CASCADE
);

-- CREATE EMPLOYEEPOSITION TABLE
CREATE TABLE EmployeePosition (
    EmployeeID INT NOT NULL,
    JobCode INT NOT NULL,
    PRIMARY KEY (EmployeeID, JobCode),
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID) ON DELETE
CASCADE,
    FOREIGN KEY (JobCode) REFERENCES Position(JobCode) ON DELETE
CASCADE
);

-- CREATE ARCHIVESRESERVATION TABLE
CREATE TABLE ArchivesReservation (
    ArchiveID SERIAL PRIMARY KEY,
    ReservationID INT,
    CheckinDate DATE NOT NULL,
    CheckoutDate DATE NOT NULL,
    ClientID INT NOT NULL,
    ClientFirstName VARCHAR(20) NOT NULL,
    ClientLastName VARCHAR(20) NOT NULL,
    ClientAddress VARCHAR(255) NOT NULL,
    ClientSSN INT NOT NULL,
```

```sql
    RoomID INT NOT NULL,
    RoomPrice DECIMAL(10, 2) NOT NULL,
    RoomAmenities VARCHAR(255) NOT NULL,
    RoomCapacity INT NOT NULL,
    RoomView VARCHAR(255) NOT NULL,
    RoomExtendable VARCHAR(3) NOT NULL,
    RoomIssues VARCHAR(255) NOT NULL,
    HotelID INT NOT NULL,
    HotelAddress VARCHAR(255) NOT NULL,
    HotelCity VARCHAR(255) NOT NULL,
    HotelPhoneNumber VARCHAR(20) NOT NULL,
    HotelEmail VARCHAR(100) NOT NULL,
    StarRating INT NOT NULL,
    NumberOfRooms INT NOT NULL,
    ChainID INT NOT NULL,
    ChainName VARCHAR(255) NOT NULL,
    HeadquartersAddress VARCHAR(255) NOT NULL,
    NumberOfHotels INT NOT NULL,
    HeadquartersEmail VARCHAR(255) NOT NULL,
    HeadquartersPhoneNumber VARCHAR(20) NOT NULL
);

-- CREATE ARCHIVESRENTAL TABLE
CREATE TABLE ArchivesRental (
    ArchiveID SERIAL PRIMARY KEY,
    RentalID INT NOT NULL,
    ReservationID INT,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    ClientID INT NOT NULL,
    ClientFirstName VARCHAR(20) NOT NULL,
    ClientLastName VARCHAR(20) NOT NULL,
    ClientAddress VARCHAR(255) NOT NULL,
    ClientSSN INT NOT NULL,
    EmployeeID INT NOT NULL,
    EmployeeFirstName VARCHAR(20) NOT NULL,
    EmployeeLastName VARCHAR(20) NOT NULL,
    EmployeeAddress VARCHAR(255) NOT NULL,
    EmployeeSSN INT NOT NULL,
    RoomID INT NOT NULL,
    RoomPrice DECIMAL(10, 2) NOT NULL,
    RoomAmenities VARCHAR(255) NOT NULL,
    RoomCapacity INT NOT NULL,
```

```
    RoomView VARCHAR(255) NOT NULL,
    RoomExtendable VARCHAR(3) NOT NULL,
    RoomIssues VARCHAR(255) NOT NULL,
    HotelID INT NOT NULL,
    HotelAddress VARCHAR(255) NOT NULL,
    HotelCity VARCHAR(255) NOT NULL,
    HotelPhoneNumber VARCHAR(20) NOT NULL,
    HotelEmail VARCHAR(100) NOT NULL,
    StarRating INT NOT NULL,
    NumberOfRooms INT NOT NULL,
    ChainID INT NOT NULL,
    ChainName VARCHAR(255) NOT NULL,
    HeadquartersAddress VARCHAR(255) NOT NULL,
    NumberOfHotels INT NOT NULL,
    HeadquartersEmail VARCHAR(255) NOT NULL,
    HeadquartersPhoneNumber VARCHAR(20) NOT NULL
);
```

VIEWS

```
--Number of Available Rooms per City
CREATE VIEW AvailableRoomsPerCity AS
SELECT
    HotelCity,
    COUNT(Room.RoomID) AS AvailableRooms
FROM
    Hotel
LEFT JOIN
    Room ON Hotel.HotelID = Room.HotelID
LEFT JOIN
    Reservation ON Room.RoomID = Reservation.RoomID
WHERE
    Reservation.RoomID IS NULL
GROUP BY
    HotelCity;

--Capacity of All Rooms in a Specific Hotel
CREATE OR REPLACE VIEW HotelRoomCapacities AS
```

```sql
SELECT c.ChainID, c.ChainName, c.HeadquartersAddress AS ChainAddress,
h.HotelID, COALESCE(SUM(r.Capacity), 0) AS TotalCapacity
FROM Hotel h
JOIN Chain c ON h.ChainID = c.ChainID
LEFT JOIN Room r ON h.HotelID = r.HotelID
GROUP BY c.ChainID, c.ChainName, c.HeadquartersAddress, h.HotelID;
```

INDEXES

```sql
-- Used for indexing by room price to speed up searches for rooms
within a specific price range:
CREATE INDEX idx_room_price ON Room(Price);

-- Used for Hotel.ChainID to optimize queries filtering hotels by their
chain:
CREATE INDEX idx_hotel_chainid ON Hotel(ChainID);

---- On Client.LastName and Client.FirstName to improve search
performance for clients by name:
CREATE INDEX idx_client_lastname_firstname ON Client(ClientLastName,
ClientFirstName);
```

TRIGGERS

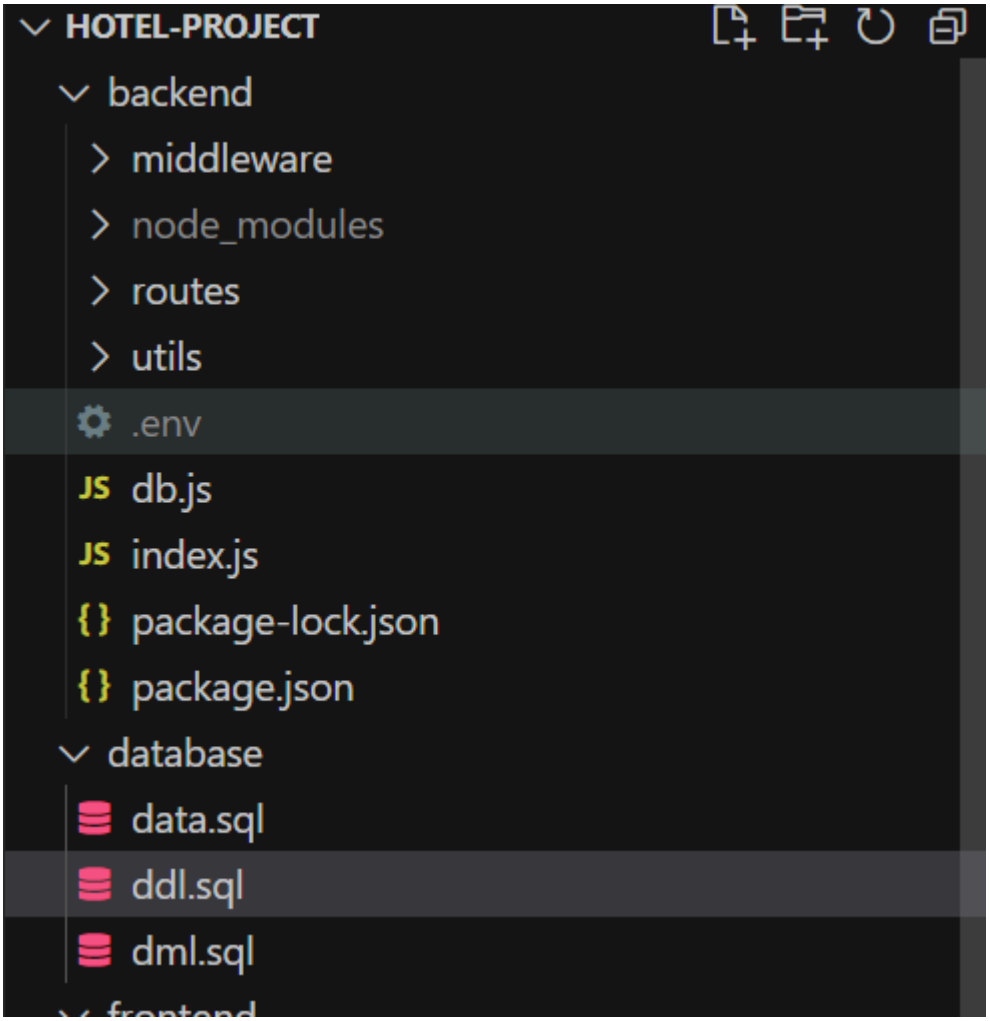
```sql
-- CREATE TRIGGER TO ARCHIVE RESERVATION DATA
CREATE OR REPLACE FUNCTION archive_reservation()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO ArchivesReservation (ReservationID, CheckinDate,
CheckoutDate, ClientID, ClientFirstName, ClientLastName, ClientAddress,
ClientSSN, RoomID, RoomPrice, RoomAmenities, RoomCapacity, RoomView,
RoomExtendable, RoomIssues, HotelID, HotelAddress, HotelCity,
HotelPhoneNumber, HotelEmail, StarRating, NumberOfRooms, ChainID,
ChainName, HeadquartersAddress, NumberOfHotels, HeadquartersEmail,
HeadquartersPhoneNumber)
    SELECT ReservationID, CheckInDate, CheckOutDate,
Reservation.ClientID, ClientFirstName, ClientLastName, ClientAddress,
```

```
ClientSSN, Reservation.RoomID, Price, Amenities, Capacity, RoomView,
Extendable, Issues, Reservation.HotelID, HotelAddress, HotelCity,
HotelPhoneNumber, HotelEmail, StarRating, NumberOfRooms,
Reservation.ChainID, ChainName, HeadquartersAddress, NumberOfHotels,
HeadquartersEmail, HeadquartersPhoneNumber
    FROM Reservation
    JOIN Client ON Reservation.ClientID = Client.ClientID
    JOIN Room ON Reservation.RoomID = Room.RoomID
    JOIN Hotel ON Reservation.HotelID = Hotel.HotelID
    JOIN Chain ON Reservation.ChainID = Chain.ChainID
    WHERE ReservationID = NEW.ReservationID;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER archive_reservation_trigger
AFTER INSERT ON Reservation
FOR EACH ROW
EXECUTE FUNCTION archive_reservation();

-- CREATE TRIGGER TO ARCHIVE RENTAL DATA
CREATE OR REPLACE FUNCTION archive_rental()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO ArchivesRental (RentalID, ReservationID, StartDate,
EndDate, ClientID, ClientFirstName, ClientLastName, ClientAddress,
ClientSSN, EmployeeID, EmployeeFirstName, EmployeeLastName,
EmployeeAddress, EmployeeSSN, RoomID, RoomPrice, RoomAmenities,
RoomCapacity, RoomView, RoomExtendable, RoomIssues, HotelID,
HotelAddress, HotelCity, HotelPhoneNumber, HotelEmail, StarRating,
NumberOfRooms, ChainID, ChainName, HeadquartersAddress, NumberOfHotels,
HeadquartersEmail, HeadquartersPhoneNumber)
    SELECT RentalID, Rental.ReservationID, StartDate, EndDate,
Rental.ClientID, ClientFirstName, ClientLastName, ClientAddress,
ClientSSN, Rental.EmployeeID, EmployeeFirstName, EmployeeLastName,
EmployeeAddress, EmployeeSSN, Rental.RoomID, Price, Amenities,
Capacity, RoomView, Extendable, Issues, Rental.HotelID, HotelAddress,
HotelCity, HotelPhoneNumber, HotelEmail, StarRating, NumberOfRooms,
Rental.ChainID, ChainName, HeadquartersAddress, NumberOfHotels,
HeadquartersEmail, HeadquartersPhoneNumber
    FROM Rental
    JOIN Client ON Rental.ClientID = Client.ClientID
    JOIN Employee ON Rental.EmployeeID = Employee.EmployeeID
```

```
    JOIN Room ON Rental.RoomID = Room.RoomID
    JOIN Hotel ON Rental.HotelID = Hotel.HotelID
    JOIN Chain ON Rental.ChainID = Chain.ChainID
    WHERE RentalID = NEW.RentalID;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER archive_rental_trigger
AFTER INSERT ON Rental
FOR EACH ROW
EXECUTE FUNCTION archive_rental();
```

4. Data Manipulation Language (Too Large to Include Here)

The data.sql file contains the Data Manipulation Language (DML) queries along with initial data insertion queries. These queries support all the functionalities of our application.



* Above shows where the DLL and DML queries can be found within the project directory

5. SQL Queries

4 Examples of SQL queries implemented and used in the web application

```
"SELECT chainid, chainname FROM chain"
```

```
"SELECT hotelid, hotelcity FROM hotel"
```

```
SELECT
    r.*,
    h.HotelID,
    h.HotelCity,
    h.HotelAddress,
    h.HotelPhoneNumber,
    h.StarRating,
    c.ChainName
FROM
    Room r
INNER JOIN
    Hotel h ON r.HotelID = h.HotelID
INNER JOIN
    Chain c ON r.ChainID = c.ChainID
```

```
const rental = await pool.query('SELECT * FROM rental WHERE
ReservationID = $1', [reservationID]);
```