Database Model Documentation

Entities and Relationships(Generated with CHATGPT):

1. User

- **Attributes:**
    - `id`: Integer, auto-incremented, primary key.
    - `firstName`: String.
    - `lastName`: String.
    - `email`: String, unique.
    - `phoneNumber`: String, unique.
    - `profilePicture`: String, optional.
    - `password`: String.
    - `createdAt`: DateTime, auto-generated.
    - `refreshToken`: String, optional.
- **Relationships:**
    - A `User` can have multiple `Hotels` (inverse relation).
    - A `User` can make multiple `Bookings` (inverse relation).
    - A `User` can have multiple `Itinerary` (inverse relation).
    - A `User` can have multiple `Notifications`.

2. Hotel

- **Attributes:**
    - `id`: Integer, auto-incremented, primary key.
    - `name`: String.
    - `address`: String.
    - `location`: String.
    - `starRating`: Float, optional.
    - `images`: Json (storing images, including the logo).
    - `ownerId`: Integer, references `User.id`.
    - `createdAt`: DateTime, auto-generated.
    - `updatedAt`: DateTime, auto-updated.
- **Relationships:**
    - A `Hotel` belongs to one `User` (owner).

    - A `Hotel` can have multiple `RoomTypes`.
    - A `Hotel` can have multiple `Bookings`.

3. RoomType

- **Attributes:**
    - `id`: Integer, auto-incremented, primary key.
    - `name`: String (e.g., twin, double, suite).
    - `amenities`: Json (storing list of amenities).
    - `pricePerNight`: Float.
    - `availableRooms`: Integer.
    - `images`: Json (storing images).
    - `hotelId`: Integer, references `Hotel.id`.
- **Relationships:**
    - A `RoomType` belongs to one `Hotel`.
    - A `RoomType` can have multiple `Bookings`.

4. Booking

- **Attributes:**
    - `id`: Integer, auto-incremented, primary key.
    - `userId`: Integer, references `User.id`.
    - `hotelId`: Integer, references `Hotel.id`.
    - `roomTypeId`: Integer, references `RoomType.id`.
    - `checkInDate`: DateTime.
    - `checkOutDate`: DateTime.
    - `status`: String (e.g., cancelled, confirmed).
    - `createdAt`: DateTime, auto-generated.
    - `updatedAt`: DateTime, auto-updated.
- **Relationships:**
    - A `Booking` belongs to one `User`, one `Hotel`, and one `RoomType`.
    - A `Booking` can have multiple `Itinerary` (inverse relation).

5. Itinerary

- **Attributes:**
    - `id`: Integer, auto-incremented, primary key.

    - `userId`: Integer, references `User.id`.
    - `hotelBooking`: Integer, optional, references `Booking.id`.
    - `flightBooking`: String, optional (could reference an external flight ID from AFS).
    - `finalize`: Boolean, defaults to `false`.
- **Relationships:**
    - An `Itinerary` belongs to one `User` and one `Booking` (optional).

6. Notification

- **Attributes:**
    - `id`: Integer, auto-incremented, primary key.
    - `userId`: Integer, references `User.id`.
    - `message`: String.
    - `isRead`: Boolean, defaults to `false`.
    - `createdAt`: DateTime, auto-generated.
- **Relationships:**
    - A `Notification` belongs to one `User`.

7. City

- **Attributes:**
    - `id`: Integer, auto-incremented, primary key.
    - `name`: String.
    - `country`: String.
- **Relationships:**
    - A `City` can have multiple `Airports`.

8. Airport

- **Attributes:**
    - `Aid`: String, unique.
    - `code`: String.
    - `name`: String.
    - `cityId`: Integer, references `City.id`.
    - `country`: String.

- **Relationships:**
    - An `Airport` belongs to one `City`.

ER Diagram (Relationships Summary)

1. **User** 1 ↔ * **Hotel**
2. **User** 1 ↔ * **Booking**
3. **User** 1 ↔ * **Itinerary**
4. **User** 1 ↔ * **Notification**
5. **Hotel** 1 ↔ * **RoomType**
6. **Hotel** 1 ↔ * **Booking**
7. **RoomType** 1 ↔ * **Booking**
8. **Booking** 1 ↔ * **Itinerary**
9. **City** 1 ↔ * **Airport**

Cedric:

City:
This is the entity which allows us to store each city from the AFS in our database when starting our script. It has an id field which is the primary key and auto increments, the name of the city, the country in which it is and a list of airports present in that said city. We also made sure to make the (City.name, City.country) pairs unique, as some Cities might have the same name.

The relationship between the City and the Airport is many-to-one, meaning that a city can have many Airports while an Airport can only have one City.

Airport:
This is the entity which allows us to store each Airport from the AFS in our database when starting our script. It has an Aid as primary key constituting the String ID from the Airport body the AFS returns. It stores a code, the airport's name, country, and cityId used to reference a City to the Airport.

The relationship between the Airport and City is one-to-many.

Bowale:
User:
This is the entity which allows us to store each users information such as the name, phone number, email and password. Additionally, the user model stores a refresh token which is used help with the backend authentication using JWT. Furthermore,

the refresh token is used to generate a new access token for the user once the previous one that was assigned has expired.

The user has one to many relationships for the following:

- A `User` can have multiple `Hotels` (inverse relation).
- A `User` can make multiple `Bookings` (inverse relation).
- A `User` can have multiple `Itinerary` (inverse relation).
- A `User` can have multiple `Notifications`.

Notifications:
This is an entity that is used to keep track of user notifications. The notifications are generated when there has been an update to the user's booking, such as when they create, cancel or update a hotel booking. Additionally, notifications are also used for both hotel owners and customers. The notification has a unique id which is used to identify the specific notification, and then it has a message option and userId which is the user associated with that notification.

Itinerary:
This entity is used to keep track of the itineraries a user makes. This allows the user to create an itinerary consisting of a flight booking and or a hotel booking. This model keeps track of the hotel booking id, which refers to the booking id the user has for a hotel room. Additionally, this also keeps track of the flight booking reference which refers to a flight booking the user has (which is stored in the AFS API). Both of these are optional because in an itinerary the user can either create one with either a flight or hotel bookin (or both). Additionally, there is also a finalize attribute which is a boolean and it keeps track of whether a user has finalized their booking or not.

Uyiosa:

Room Type:

Defines the different types of rooms available within a hotel (e.g., standard. deluxe) with attributes like name, amenities, price per night, available rooms and images. It links directly to its hotel and supports the calculation of effective availability based on overlapping bookings.

Booking:

Captures a reservation made by a user for a specific room type in a hotel, including check-in and check-out dates and status. It ties together the user, hotel, and room type, enabling management of occupancy and ensuring that room availability is respected.

Hotel:

Represents a physical hotel and includes key details such as its name, address, location, star rating, images (and logo), and timestamps. It is owned by a user and serves as the central entity connecting room types and bookings.