**🏨 Hotel Room Reservation System**

A full-stack application for managing hotel room reservations.

**Problem Statement:**

A hotel has 97 rooms distributed across 10 floors:

* Floors 1-9: Each floor has 10 rooms, numbered sequentially (e.g., Floor 1: 101-110, Floor 2: 201-210, and so on).
* Floor 10 (Top Floor): Has only 7 rooms, numbered 1001-1007.
* Building Structure:

1. A staircase and lift are located on the left side of the building.
2. Rooms on each floor are arranged sequentially from left to right, with the first room on each floor being closest to the stairs/lift.

* Room Proximity (Travel Time):

1. Horizontal travel: Moving between two adjacent rooms on the same floor takes 1 minute per room.
2. Vertical travel: Moving between floors takes 2 minutes per floor using the stairs/lift.

* Booking Rules:

1. A single guest can book up to 5 rooms at a time.
2. Priority is to book rooms on the same floor first.
3. If rooms are not available on the same floor, Priority is to book rooms that minimize the total travel time between the first and last room in the booking.
4. If the required number of rooms is unavailable on one floor, booking should span across floors, prioritizing rooms that minimize the combined vertical and horizontal travel time.

**Example Scenario:**

1. **Available Rooms:**
   1. Floor 1: 101, 102, 105, 106
   2. Floor 2: 201, 202, 203, 210
   3. Floor 3: 301, 302
2. **A guest wants to book 4 rooms:**
   1. Rooms 101, 102, 105, 106 on Floor 1 will be selected because they minimize total travel time.
3. **If only 2 rooms are available on Floor 1 (e.g., 101, 102):**
   1. The system will select rooms 201, 202 from Floor 2, as this minimizes vertical (2 minutes) and horizontal travel times.

Your task is to create a room reservation system that dynamically calculates the total travel time between booked rooms and optimally assigns rooms based on these rules.

**📌 Project Overview**

This system enables users to **book, visualize, and manage hotel room reservations** while following smart booking rules. The backend efficiently assigns rooms based on proximity, and the frontend provides an interactive UI for seamless booking.

**🛠️ Technologies Used**

**Frontend:**

* **Angular 19** (Standalone Components)
* **TypeScript**
* **Bootstrap** (UI styling)
* **RxJS Signals** (State management)
* **Angular HTTP Client** (API interactions)

**Backend:**

* **Node.js & Express.js**
* **PostgreSQL** (Relational database)
* **pg (Node.js PostgreSQL driver)**
* **dotenv (Environment variables management)**
* **CORS & Body-Parser** (For request handling)

**📂 Database Structure**

The system uses **four tables** to store and manage hotel bookings.

**1️. floors Table**

Stores hotel floor details with a **unique floor number constraint**.

**2️. rooms Table**

* Stores room details and **links them to floors** via floor\_id.
* Ensures **unique room numbers** and tracks **occupancy status** (is\_occupied).

**3️. occupancy Table**

* Maintains current **room occupancy** separately.
* Uses a **foreign key** (room\_id) to reference the rooms table.

**4️. bookings Table**

* Stores **guest bookings** along with the **booked room numbers**.
* Uses an **integer array (integer[])** to store multiple booked rooms in a single record.
* Includes a **timestamp (booking\_time)** to track when a booking was made.

**🚀 Backend Overview**

The backend is built using **Node.js & Express.js**, with PostgreSQL as the database.

**📌 API Endpoints**

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| GET | / | Health check (API status) |
| GET | /rooms | Fetch all available rooms |
| POST | /book | Book a specified number of rooms |
| POST | /reset | Reset all bookings |
| POST | /random-occupancy | Generate random room occupancy |

**📌 Booking Algorithm**

1. **Fetches all available rooms.**
2. **Tries to book all rooms on the same floor first.**
3. **If that’s not possible, selects rooms across floors while minimizing travel time.**
4. **Marks rooms as booked in the database.**

**📌 Travel Time Calculation**

* **Horizontal movement (same floor):** 1 min per room
* **Vertical movement (between floors):** 2 min per floor
* **Minimizes combined travel time across floors.**

**📌 Random Occupancy Generation**

* **Randomly occupies 20-30% of rooms.**
* **Ensures random occupancy does not interfere with new bookings.**

**🎨 Frontend Overview**

The **Angular frontend** provides an interactive UI with real-time room status updates.

**📌 Features**

* **Smart Room Booking**: Selects rooms optimally based on the rules.
* **Dynamic Visualization**: Displays floors and rooms with color-coded statuses.
* **Random Occupancy Simulation**: Allows testing of booking constraints.
* **Reset Functionality**: Clears all bookings for a fresh start.

**📌 UI Breakdown**

* **Lift on the left side** (⬆️) for better navigation.
* **Rooms displayed in rows per floor** (10 rooms per floor, except the top floor with 7 rooms).
* **Color-coded rooms:**
  + 🟩 **Green:** Booked rooms
  + ⚪ **White:** Available rooms
  + 🟨 **Yellow:** Randomly occupied rooms

**📌 Project Structure**

/src

├── app/

│ ├── components/

│ │ ├── booking/

│ │ │ ├── booking.component.ts (Logic for booking UI)

│ │ │ ├── booking.component.html (Hotel layout & controls)

│ │ │ ├── booking.component.scss (Styling)

│ ├── services/

│ │ ├── booking.service.ts (Handles API calls)

│ ├── app.component.ts (Main component)

│ ├── app.component.html (Root template)

├── main.ts (Bootstraps Angular App)