Certainly! Below is an outline for a document you can use for your hotel room reservation system project.

**Hotel Room Reservation System**

**Project Overview**

The goal of this project is to develop a hotel room reservation system for a hotel with 97 rooms distributed across 10 floors. The system should allow guests to book rooms with a priority for minimizing travel time. Additionally, the system must feature functionality to view available rooms, generate random occupancy, and reset all bookings.

**Features**

1. **View Available Rooms**: Users can view the current status of all rooms (available or booked).
2. **Book Rooms**: Users can book between 1 and 5 rooms at a time, with priority for booking rooms on the same floor. If not enough rooms are available on a single floor, the system will select rooms that minimize travel time, both horizontally and vertically.
3. **Random Occupancy Generator**: This feature simulates random room bookings for testing purposes.
4. **Reset Booking**: This button resets the entire booking system, setting all rooms back to "available."

**System Design**

**1. Hotel Structure**

The hotel consists of 97 rooms distributed across 10 floors:

* **Floors 1 to 9**: Each floor has 10 rooms, numbered sequentially (e.g., Floor 1: 101-110, Floor 2: 201-210, etc.).
* **Floor 10**: The top floor has 7 rooms numbered 1001-1007.

Rooms are numbered as follows:

* **Floor 1**: 101, 102, ..., 110
* **Floor 2**: 201, 202, ..., 210
* **Floor 10**: 1001, 1002, ..., 1007

**2. Room Booking Rules**

* **Booking Capacity**: A user can book up to 5 rooms at once.
* **Same Floor Priority**: Rooms should be booked on the same floor when possible.
* **Minimizing Travel Time**: If rooms on the same floor aren't available, the system should prioritize rooms that minimize both vertical and horizontal travel time.

**3. Travel Time Calculation**

* **Horizontal Travel**: Moving between adjacent rooms on the same floor takes 1 minute per room.
* **Vertical Travel**: Moving between floors takes 2 minutes per floor.

**4. Endpoints**

* **GET /rooms**: Returns the current status of all rooms (available/occupied).
* **POST /book**: Books a specified number of rooms (between 1 and 5). The system will prioritize booking rooms on the same floor first, then across floors if necessary.
* **POST /reset**: Resets all room statuses to "available".
* **POST /generate**: Generates random occupancy for all rooms.

**Technologies Used**

1. **Flask**: Python web framework used for handling the backend, including routing, API endpoints, and the logic for room bookings.
2. **JavaScript (Frontend)**: Used for handling user interactions such as booking rooms, viewing available rooms, and resetting the booking.
3. **HTML/CSS**: Used for structuring and styling the frontend page.
4. **Python (Backend)**: Contains the logic for room booking, travel time calculation, and occupancy generation.

**Functionality Overview**

**1. Frontend**

The frontend is a simple web page where users can:

* View available rooms.
* Book rooms by entering the number of rooms.
* Reset all bookings.
* Generate random occupancy.

The web page also features an interactive display of room statuses and booking results.

**2. Backend**

The backend handles the logic for:

* **Room Status**: Tracks the availability of each room (either "available" or "booked").
* **Booking Process**: When a booking request is received, the system prioritizes rooms on the same floor or those that minimize travel time.
* **Random Occupancy**: The system can simulate random bookings of rooms.
* **Reset Booking**: Resets all room statuses to "available".

**3. Room Booking Algorithm**

* The system first checks if enough rooms are available on the same floor.
* If not enough rooms are available, the system attempts to minimize travel time by selecting rooms across floors.
* It calculates the vertical and horizontal travel time, choosing the combination that minimizes the total travel time.

**API Endpoints**

**1. GET /rooms**

Returns the current status of all rooms in JSON format.

**Example Response:**

{

"101": "available",

"102": "booked",

"103": "available",

"201": "available",

"202": "occupied",

...

}

**2. POST /book**

Books a specified number of rooms and returns the rooms booked and the total travel time.

**Request Body:**

{

"num\_rooms": 3

}

**Example Response:**

{

"booked\_rooms": ["101", "102", "103"],

"total\_travel\_time": 3

}

**3. POST /reset**

Resets the status of all rooms to "available".

**Example Response:**

{

"message": "Room statuses reset successfully."

}

**4. POST /generate**

Generates random occupancy for all rooms.

**Example Response:**

{

"message": "Random occupancy generated."

}

**Frontend Code (HTML + JavaScript)**

The frontend is a simple page with buttons and input fields for interacting with the backend. Users can view rooms, book rooms, reset the booking, and generate random occupancy.

**Example HTML and JavaScript:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hotel Reservation System</title>

<script>

let roomsHidden = false;

let bookingResultHidden = false;

async function fetchRooms() {

const response = await fetch('/rooms');

const data = await response.json();

const roomsElement = document.getElementById('rooms');

if (roomsHidden) {

roomsElement.innerText = JSON.stringify(data, null, 2);

roomsHidden = false;

} else {

roomsElement.innerText = '';

roomsHidden = true;

}

}

async function bookRooms() {

const numRooms = document.getElementById('numRooms').value;

const response = await fetch('/book', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ num\_rooms: parseInt(numRooms) })

});

const data = await response.json();

const bookingResultElement = document.getElementById('bookingResult');

if (bookingResultHidden) {

bookingResultElement.innerText = JSON.stringify(data, null, 2);

bookingResultHidden = false;

} else {

bookingResultElement.innerText = '';

bookingResultHidden = true;

}

}

async function resetRooms() {

const response = await fetch('/reset', {

method: 'POST',

headers: { 'Content-Type': 'application/json' }

});

const data = await response.json();

alert(data.message);

}

async function generateRandomOccupancy() {

const response = await fetch('/generate', {

method: 'POST',

headers: { 'Content-Type': 'application/json' }

});

const data = await response.json();

alert(data.message);

}

</script>

</head>

<body>

<h1>Hotel Reservation System</h1>

<button onclick="fetchRooms()">View Rooms</button>

<pre id="rooms"></pre>

<h3>Book Rooms</h3>

<input type="number" id="numRooms" placeholder="Enter number of rooms" />

<button onclick="bookRooms()">Book</button>

<pre id="bookingResult"></pre>

<h3>Actions</h3>

<button onclick="resetRooms()">Reset Booking</button>

<button onclick="generateRandomOccupancy()">Generate Random Occupancy</button>

</body>

</html>

**Conclusion**

This hotel room reservation system efficiently handles booking requests, prioritizing minimal travel time, and also allows for random occupancy generation and the ability to reset the entire booking. It is built using Flask for the backend, JavaScript for the frontend, and can be tested using various room configurations to ensure optimal booking performance.

Feel free to modify the document as needed and include any additional details specific to your development process or implementation!