Technical Assessment Case Study: Internal Resource Booking System

Project Title: Internal Resource Booking System

Role: Junior ASP.NET Core Developer

Objective:

This case study assesses your ability to design, develop, and implement a functional web application using ASP.NET Core. You will demonstrate your understanding of fundamental concepts including C#, Entity Framework Core, database interaction, server-side logic, and basic front-end development. Your code quality, problem-solving skills, and adherence to best practices will also be evaluated.

Scenario:

Our company currently manages shared resources (e.g., meeting rooms, company vehicles, specialized equipment) through manual processes, leading to frequent double-bookings and confusion. We need a simple, intuitive web application that allows employees to view available resources and book them for specific time slots.

Part 1: Core Functionality - Resource & Booking Management

Deliverables for Part 1:

1. **Project Setup:**
    - Create a new ASP.NET Core Web Application. You may choose between **MVC** or **Razor Pages** based on your preference and explain your reasoning during the final review.
    - Configure the project to use **Entity Framework Core** with a **SQL Server LocalDB** or **SQLite** database.
    - Ensure proper configuration in appsettings.json for database connection.
    - Implement basic dependency injection for services.
2. **Resource Management:**
    - **Model Definition:** Create a C# model named Resource with the following properties:
        - Id (Primary Key, integer)
        - Name (string, e.g., "Meeting Room Alpha", "Company Car 1")
        - Description (string, e.g., "Large room with projector and whiteboard", "Compact sedan")

        - Location (string, e.g., "3rd Floor, West Wing", "Parking Bay 5")
        - Capacity (integer, e.g., 10 for a meeting room)
        - IsAvailable (boolean, indicates if the resource is generally available for booking, e.g., true unless under maintenance)
    - **Database Context:** Create an ApplicationDbContext class inheriting from DbContext and add a DbSet<Resource>.
    - **Migrations:** Generate and apply Entity Framework Core migrations to create the Resources table in your database.
    - **CRUD Operations for Resources:**
        - Implement **Create** functionality: A form to add new resources to the system.
        - Implement **Read** functionality: A page that lists all resources, showing their key details. Include a separate page for a detailed view of a single resource.
        - Implement **Update** functionality: A form to edit existing resource details.
        - Implement **Delete** functionality: A way to remove resources from the system.
    - **Server-Side Validation:** Implement basic server-side validation for Resource properties (e.g., Name is required, Capacity must be a positive number).
3. **Booking Model & Basic Display:**
    - **Model Definition:** Create a C# model named Booking with the following properties:
        - Id (Primary Key, integer)
        - ResourceId (Foreign Key to Resource's Id)
        - StartTime (DateTime)
        - EndTime (DateTime)
        - BookedBy (string, representing the name of the person making the booking, e.g., "Jane Doe")
        - Purpose (string, e.g., "Client Pitch", "Team Stand-up")

- **Relationships:** Configure the one-to-many relationship between Resource and Booking in your ApplicationDbContext.
- **Migrations:** Generate and apply new Entity Framework Core migrations for the Bookings table.
- **Display Bookings:**
  - Create a page that lists all existing bookings in the system.
  - On the Resource detail page (or resource list), display a list of all *upcoming* bookings associated with that specific resource.

Part 2: Booking Logic, UI/UX, and Polish

Deliverables for Part 2:

1. **Booking Creation & Validation:**
   - **Create Booking Form:** Develop a user-friendly form to allow users to create a new Booking.
     - The form should allow selection of an available Resource (e.g., via a dropdown list).
     - Inputs for StartTime, EndTime, BookedBy, and Purpose.
   - **Client-Side Validation (Recommended):** Implement basic client-side validation for form inputs (e.g., required fields).
   - **Server-Side Validation:**
     - StartTime and EndTime are required.
     - EndTime must be strictly after StartTime.
     - Purpose is required.
   - **Booking Conflict Logic (CRITICAL):** This is a key assessment point. When a new booking is attempted:
     - Implement robust server-side logic to check for **time conflicts**. A resource cannot be booked if it's already reserved for any part of the requested StartTime to EndTime range.
     - If a conflict is detected, prevent the booking and display a clear, user-friendly error message to the user (e.g., "This resource is already booked during the requested time. Please choose another slot or resource, or adjust your times.").

2. **User Interface (UI) & User Experience (UX) Improvements:**
    - Apply basic styling to your application (e.g., using Bootstrap or simple custom CSS). The goal is readability and a functional layout, not professional design.
    - Ensure forms are intuitive with clear labels and appropriate input types (e.g., type="datetime-local" for date/time inputs).
    - Implement clear navigation between different sections of the application (e.g., "View Resources", "View Bookings", "Add Resource", "Add Booking").
3. **Code Quality & Best Practices:**
    - **Error Handling:** Implement basic error handling (e.g., try-catch blocks for database operations) to prevent application crashes and provide graceful degradation.
    - **Code Organization:** Ensure your code is logically structured, easy to read, and follows C# and .NET coding conventions.
    - **Comments:** Add comments where necessary to explain complex logic or non-obvious code sections.
    - **Version Control:** Use Git throughout the development process. Make regular, meaningful commits.

Optional / Stretch Goals (If Time Permits):

- Allow users to edit or cancel existing bookings.
- Implement a basic search or filter function for resources or bookings (e.g., filter by resource name, date).
- Add simple data seeding to pre-populate your database with a few resources when the application starts.
- Create a simple "Dashboard" page showing current day's bookings or upcoming bookings.

Submission & Evaluation:

What to Submit:

- A link to a Git repository (e.g., GitHub, GitLab, Bitbucket) containing your complete project. Ensure all necessary files are included and the commit history is clear.

- A brief README.md file in your repository explaining how to set up and run the application.
- A zipped file, containing your source code, database and screenshots of your running application that should be uploaded on WeTransfer. The WeTransfer link should be submitted on the form

Evaluation Focus:

Your solution will be evaluated based on the following criteria:

- **Functionality:** Does the application meet all the core requirements? Is the booking conflict logic robust?
- **ASP.NET Core Proficiency:** Correct use of MVC/Razor Pages, controllers/page models, views, routing, and model binding. Understanding of dependency injection.
- **C# Language Skills:** Clean, readable, and idiomatic C# code. Effective use of data structures and control flow.
- **Entity Framework Core:** Proper model design, relationship configuration, migrations, and efficient data querying (LINQ).
- **Database Interaction:** Accuracy and efficiency of CRUD operations.
- **Problem-Solving:** The effectiveness and elegance of your solution to the booking conflict challenge. How you handled potential edge cases.
- **Code Quality & Practices:** Code organization, naming conventions, error handling, and comments.
- **Version Control:** Your Git commit history will be reviewed for regular commits and meaningful messages.
- **User Experience:** Usability and clarity of the application's interface.

Final Review:

Upon completion, you will be asked to:

1. Demonstrate your working application.
2. Walk us through your code, explaining your design choices and implementations.
3. Answer questions regarding your solution and the technologies used.