Full Stack .Net Web Developer

Project: Bookify Hotel Reservation System with ASP.NET Core

Objective: To build a robust and scalable hotel reservation web application, leveraging design patterns like **N-Tier Architecture**, the **Repository Pattern**, and **Unit of Work** for a clean, maintainable, and professional codebase.

Description: Bookify is a comprehensive hotel booking platform that allows customers to search for available rooms, view room details, and make reservations using an integrated payment system. The application will also feature a powerful admin panel for hotel staff to manage room types, individual rooms, and all customer bookings. The entire system will be built following the **N-Tier architecture** to ensure a clear separation between the UI, business logic, and data access layers.

Design Patterns to Use:

- **N-Tier Architecture**: Structuring the application into a Presentation Layer (UI), a Business Logic Layer (Services), and a Data Access Layer.
- **Repository Pattern**: Abstracting all database operations to provide a clean and consistent API for data access.
- **Unit of Work Pattern**: Ensuring that complex operations, like creating a booking and updating room availability, are completed in a single, atomic transaction to maintain data integrity.
- **Dependency Injection**: Loosely coupling the application's components by injecting services (like repositories) into controllers.

Technologies to Use: ASP.NET Core MVC, Entity Framework Core, ASP.NET Identity, Stripe, JQuery, DataTables, Toaster JS.

Week 1: Architecture Setup and Room Listings

- **N-Tier Architecture Setup**:
  - **Presentation Layer (Bookify.Web)**: The main ASP.NET Core MVC project containing controllers, views, and client-side assets (**JQuery**, etc.).
  - **Business Logic Layer (Bookify.Services)**: A class library to handle business rules, such as checking room availability and calculating reservation costs.
  - **Data Access Layer (Bookify.Data)**: A class library for implementing the **Repository** and **Unit of Work** patterns to communicate with the database.
- **Database Design**:

    - Use **Entity Framework Core** to define the database schema for tables like `Rooms`, `RoomTypes`, `Bookings`, and `Users`.
- **Repository and Unit of Work Implementation**:
    - Create generic and specific repositories (e.g., `RoomRepository`, `BookingRepository`) to abstract CRUD operations. The `DbContext` will function as the **Unit of Work**.
- **User Authentication Setup**:
    - Configure **ASP.NET Identity** to manage user registration, login, and roles (e.g., Customer, Admin).

Deliverables:

- A solution with the complete **N-Tier Architecture** set up.
- A public-facing page that lists available rooms, fetching data through the repository pattern.
- Database schema created with Entity Framework Core Migrations.
- A working user registration and login system.

Week 2: Reservation Flow, Roles, and Admin Panel

- **Reservation Cart Functionality**:
    - Implement a "reservation cart" using Session State where users can temporarily hold a room selection before confirming their booking.
- **Role-Based Access Control (RBAC)**:
    - Use **ASP.NET Identity** roles to secure the admin panel, ensuring only users with the "Admin" role can access it.
- **Admin Dashboard**:
    - Create an admin interface for managing rooms, room types, and viewing all bookings. Utilize **DataTables** to display the information in a user-friendly way. All data operations must go through the **Repository Pattern**.

Deliverables:

- A fully functional reservation flow where users can select a room and dates.
- Secure role-based permissions for admin and customer users.
- An admin dashboard for managing the hotel's rooms and bookings.

Week 3: Booking Confirmation and Stripe Integration

- **Booking Confirmation**:

    - Implement the final checkout and booking confirmation logic. The process of creating a `Booking` record and updating the room's availability must be wrapped in a single transaction using the **Unit of Work** pattern.
- **Stripe Payment Integration**:
    - Integrate the **Stripe** payment gateway to handle payments during the booking confirmation step. Securely process payments and store transaction references.
- **User Profiles and Booking History**:
    - Create a profile page where authenticated users can view their personal details and see a history of their past and upcoming bookings.

Deliverables:

- A complete booking confirmation process with integrated **Stripe** payments.
- A customer profile page with a viewable booking history.
- Thoroughly tested booking and payment functionality.

Week 4: Health Checks, Logging, and Final Polish

- **Implement Health Checks & Logging**:
    - Add **ASP.NET Core Health Checks** to create a health endpoint (e.g., `/health`) that verifies the application's ability to connect to the database.
    - Integrate a structured logging framework (like Serilog) to write detailed, structured logs for requests, errors, and other important application events.
- **UI Enhancements**:
    - Use **JQuery** and **Toaster JS** for a more dynamic and interactive user experience (e.g., pop-up notifications). Refine the site's look and feel with Bootstrap and custom CSS.
- **Final Testing**:
    - Conduct comprehensive end-to-end testing of the entire application to ensure all features are working as expected.

Deliverables:

- A polished and responsive UI for both the customer-facing site and the admin panel.
- A functional `/health` endpoint for monitoring the application's status.
- Structured logging implemented throughout the application for easier debugging.
- A complete, well-tested, and fully documented application.