Creating a complete web application involves multiple steps and components. Below is a simplified example to get you started with a basic Room Reservation system using ASP.NET Core. This example assumes you have some familiarity with ASP.NET Core MVC.

### Step 1: Set Up the Project

Create a new ASP.NET Core MVC project using the following steps:

1. Open a command prompt and run the following commands:

```bash

dotnet new mvc -n RoomReservationSystem

cd RoomReservationSystem

```

2. Open the project in your preferred code editor.

### Step 2: Define Models

Create models to represent the entities in your system. In this example, we'll have models for `Room` and `Reservation`.

```csharp

// Models/Room.cs

public class Room

{

public int Id { get; set; }

public string Name { get; set; }

public int Capacity { get; set; }

// Add other properties as needed

}

```

```csharp

// Models/Reservation.cs

public class Reservation

{

public int Id { get; set; }

public int RoomId { get; set; }

public string UserName { get; set; }

public DateTime StartTime { get; set; }

public DateTime EndTime { get; set; }

// Add other properties as needed

}

```

### Step 3: Set Up DbContext

Configure Entity Framework Core to work with your models. Modify the `ApplicationDbContext` class.

```csharp

// Data/ApplicationDbContext.cs

using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext

{

public DbSet<Room> Rooms { get; set; }

public DbSet<Reservation> Reservations { get; set; }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

}

```

### Step 4: Configure Dependency Injection

Register your DbContext and services in the `Startup.cs` file.

```csharp

// Startup.cs

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddControllersWithViews();

// Add other services as needed

}

```

### Step 5: Create Controllers

Create controllers to handle user requests. For simplicity, we'll create a `RoomController` and a `ReservationController`.

```csharp

// Controllers/RoomController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Linq;

public class RoomController : Controller

{

private readonly ApplicationDbContext \_context;

public RoomController(ApplicationDbContext context)

{

\_context = context;

}

public IActionResult Index()

{

var rooms = \_context.Rooms.ToList();

return View(rooms);

}

}

```

```csharp

// Controllers/ReservationController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Linq;

public class ReservationController : Controller

{

private readonly ApplicationDbContext \_context;

public ReservationController(ApplicationDbContext context)

{

\_context = context;

}

public IActionResult Index()

{

var reservations = \_context.Reservations.Include(r => r.Room).ToList();

return View(reservations);

}

}

```

### Step 6: Create Views

Create views to display your data. Create folders named `Views/Room` and `Views/Reservation`, and add `Index.cshtml` views.

```html

<!-- Views/Room/Index.cshtml -->

@model List<Room>

<h2>Rooms</h2>

<ul>

@foreach (var room in Model)

{

<li>@room.Name - Capacity: @room.Capacity</li>

}

</ul>

```

```html

<!-- Views/Reservation/Index.cshtml -->

@model List<Reservation>

<h2>Reservations</h2>

<ul>

@foreach (var reservation in Model)

{

<li>@reservation.UserName reserved @reservation.Room.Name from @reservation.StartTime to @reservation.EndTime</li>

}

</ul>

```

### Step 7: Seed Data

Seed some initial data for testing purposes.

```csharp

// Data/ApplicationDbContextSeed.cs

using System;

using System.Linq;

public static class ApplicationDbContextSeed

{

public static void SeedData(ApplicationDbContext context)

{

if (!context.Rooms.Any())

{

context.Rooms.Add(new Room { Name = "Meeting Room A", Capacity = 10 });

context.Rooms.Add(new Room { Name = "Conference Room B", Capacity = 20 });

context.SaveChanges();

}

if (!context.Reservations.Any())

{

var roomId = context.Rooms.First().Id;

context.Reservations.Add(new Reservation

{

RoomId = roomId,

UserName = "John Doe",

StartTime = DateTime.Now.AddHours(1),

EndTime = DateTime.Now.AddHours(2)

});

context.SaveChanges();

}

}

}

```

Call this method in the `Configure` method of the `Startup.cs` file.

```csharp

// Startup.cs

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

// ... other configurations

using (var scope = app.ApplicationServices.CreateScope())

{

var services = scope.ServiceProvider;

var dbContext = services.GetRequiredService<ApplicationDbContext>();

ApplicationDbContextSeed.SeedData(dbContext);

}

// ... other configurations

}

```

### Step 8: Run the Application

Run the application using the following command:

```bash

dotnet run

```

Visit `https://localhost:5001/Room` and `https://localhost:5001/Reservation` to see the basic views.

This is a simplified example, and in a real-world scenario, you would need to implement more features, such as user authentication, room reservation logic, error handling, etc.

Certainly! Below are additional steps to enhance your Room Reservation System by adding user authentication, room reservation logic, and basic error handling.

### Step 1: Add Authentication

#### 1.1 Configure Authentication in `Startup.cs`

Update the `ConfigureServices` method to add authentication services.

```csharp

// Startup.cs

public void ConfigureServices(IServiceCollection services)

{

// ... existing configurations

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(options =>

{

options.LoginPath = "/Account/Login";

options.AccessDeniedPath = "/Account/AccessDenied";

});

// ... existing configurations

}

```

#### 1.2 Create an Account Controller

Create an `AccountController` with login and logout actions.

```csharp

// Controllers/AccountController.cs

public class AccountController : Controller

{

public IActionResult Login()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Login(string returnUrl = "/")

{

// Implement login logic here

// Example: Check credentials and sign in the user using SignInManager

return Redirect(returnUrl);

}

[HttpPost]

public async Task<IActionResult> Logout()

{

// Implement logout logic here

// Example: Sign out the user using SignInManager

return RedirectToAction(nameof(HomeController.Index), "Home");

}

}

```

#### 1.3 Create Login View

Create a login view to capture user credentials.

```html

<!-- Views/Account/Login.cshtml -->

<form asp-controller="Account" asp-action="Login" method="post">

<!-- Add input fields for username and password -->

<button type="submit">Login</button>

</form>

```

### Step 2: Add Room Reservation Logic

#### 2.1 Update Reservation Controller

Enhance the `ReservationController` to handle room reservations.

```csharp

// Controllers/ReservationController.cs

[Authorize]

public class ReservationController : Controller

{

private readonly ApplicationDbContext \_context;

public ReservationController(ApplicationDbContext context)

{

\_context = context;

}

public IActionResult Index()

{

var reservations = \_context.Reservations.Include(r => r.Room).ToList();

return View(reservations);

}

[HttpGet]

public IActionResult Reserve(int roomId)

{

var room = \_context.Rooms.FirstOrDefault(r => r.Id == roomId);

if (room == null)

{

return NotFound();

}

return View(new Reservation { RoomId = room.Id, Room = room });

}

[HttpPost]

public IActionResult Reserve(Reservation reservation)

{

if (ModelState.IsValid)

{

// Validate reservation logic (e.g., check for overlapping appointments)

// Save reservation to the database

\_context.Reservations.Add(reservation);

\_context.SaveChanges();

return RedirectToAction(nameof(Index));

}

// If validation fails, return to the reservation form with errors

return View(reservation);

}

}

```

#### 2.2 Update Reservation Views

Create views for listing reservations and making reservations.

```html

<!-- Views/Reservation/Index.cshtml -->

@model List<Reservation>

<h2>Reservations</h2>

<ul>

@foreach (var reservation in Model)

{

<li>@reservation.UserName reserved @reservation.Room.Name from @reservation.StartTime to @reservation.EndTime</li>

}

</ul>

```

```html

<!-- Views/Reservation/Reserve.cshtml -->

@model Reservation

<h2>Reserve Room</h2>

<form asp-controller="Reservation" asp-action="Reserve" method="post">

<!-- Add input fields for reservation details -->

<button type="submit">Reserve</button>

</form>

```

### Step 3: Add Error Handling

#### 3.1 Global Error Handling

Configure global error handling in the `Startup.cs` file.

```csharp

// Startup.cs

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

// ... existing configurations

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAuthentication();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

```

#### 3.2 Custom Error Handling Page

Create a custom error handling view.

```html

<!-- Views/Shared/Error.cshtml -->

<h2>Sorry, an error occurred while processing your request.</h2>

```

### Step 4: Run the Application

Run the application using the following command:

```bash

dotnet run

```

Visit `https://localhost:5001/Room` to see the list of rooms. Login using the `/Account/Login` page, and then you can navigate to `/Reservation/Reserve` to make a reservation.

This is a basic setup, and you can further enhance it by adding validation, improving the user interface, and incorporating more features based on your organization's requirements.

Certainly! To implement login and logout logic using `SignInManager` in ASP.NET Core Identity, you'll need to interact with the `SignInManager<TUser>` class. Below are the steps to integrate login and logout functionality in your `AccountController`.

### Step 1: Update `AccountController` for Login and Logout

```csharp

// Controllers/AccountController.cs

using Microsoft.AspNetCore.Identity;

public class AccountController : Controller

{

private readonly SignInManager<ApplicationUser> \_signInManager;

public AccountController(SignInManager<ApplicationUser> signInManager)

{

\_signInManager = signInManager;

}

[HttpGet]

public IActionResult Login()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = "/")

{

if (ModelState.IsValid)

{

var result = await \_signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, lockoutOnFailure: false);

if (result.Succeeded)

{

return Redirect(returnUrl);

}

ModelState.AddModelError(string.Empty, "Invalid login attempt");

}

return View(model);

}

[HttpPost]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction(nameof(HomeController.Index), "Home");

}

}

```

### Step 2: Create a Login ViewModel

Create a simple view model to handle login form input.

```csharp

// Models/LoginViewModel.cs

using System.ComponentModel.DataAnnotations;

public class LoginViewModel

{

[Required]

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[Display(Name = "Remember me")]

public bool RememberMe { get; set; }

}

```

### Step 3: Update the Login View

Create a view to display the login form.

```html

<!-- Views/Account/Login.cshtml -->

@model LoginViewModel

<h2>Login</h2>

<form asp-controller="Account" asp-action="Login" method="post">

<div asp-validation-summary="All" class="text-danger"></div>

<div class="form-group">

<label asp-for="Email"></label>

<input asp-for="Email" class="form-control" />

<span asp-validation-for="Email" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Password"></label>

<input asp-for="Password" type="password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<div class="form-check">

<input asp-for="RememberMe" class="form-check-input" />

<label class="form-check-label" asp-for="RememberMe">Remember me</label>

</div>

</div>

<button type="submit" class="btn btn-primary">Login</button>

</form>

```

### Step 4: Ensure ApplicationUser is Configured

Ensure that you have an `ApplicationUser` class configured with ASP.NET Core Identity. This class should be in your `Data` folder.

```csharp

// Data/ApplicationUser.cs

using Microsoft.AspNetCore.Identity;

public class ApplicationUser : IdentityUser

{

// Additional properties if needed

}

```

### Step 5: Update `ConfigureServices` in `Startup.cs`

Ensure that you have `Identity` configured in your `Startup.cs` file.

```csharp

// Startup.cs

public void ConfigureServices(IServiceCollection services)

{

// ... existing configurations

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

// ... existing configurations

}

```

### Step 6: Run the Application

Run the application using the following command:

```bash

dotnet run

```

Visit `https://localhost:5001/Account/Login` to see the login page. Once you're logged in, you can use `https://localhost:5001/Account/Logout` to log out.

Note: Ensure that you have proper views, styles, and additional configurations based on your application structure and design. This example provides a basic setup for login and logout using `SignInManager`.

Certainly! To validate reservation logic and add input fields for reservation details, you'll need to update the `Reserve` actions in your `ReservationController`. Additionally, you should create a model for the reservation details.

### Step 1: Create ReservationDetails Model

Create a model to represent the input fields for reservation details.

```csharp

// Models/ReservationDetailsViewModel.cs

using System;

using System.ComponentModel.DataAnnotations;

public class ReservationDetailsViewModel

{

[Required]

public int RoomId { get; set; }

[Required]

[Display(Name = "Start Time")]

[DataType(DataType.DateTime)]

public DateTime StartTime { get; set; }

[Required]

[Display(Name = "End Time")]

[DataType(DataType.DateTime)]

public DateTime EndTime { get; set; }

}

```

### Step 2: Update ReservationController

Update the `Reserve` actions in your `ReservationController` to include validation logic for overlapping appointments.

```csharp

// Controllers/ReservationController.cs

[Authorize]

public class ReservationController : Controller

{

private readonly ApplicationDbContext \_context;

public ReservationController(ApplicationDbContext context)

{

\_context = context;

}

public IActionResult Index()

{

var reservations = \_context.Reservations.Include(r => r.Room).ToList();

return View(reservations);

}

[HttpGet]

public IActionResult Reserve(int roomId)

{

var room = \_context.Rooms.FirstOrDefault(r => r.Id == roomId);

if (room == null)

{

return NotFound();

}

var reservationDetails = new ReservationDetailsViewModel

{

RoomId = room.Id,

StartTime = DateTime.Now,

EndTime = DateTime.Now.AddHours(1)

};

return View(reservationDetails);

}

[HttpPost]

public IActionResult Reserve(ReservationDetailsViewModel reservationDetails)

{

if (ModelState.IsValid)

{

// Validate reservation logic (check for overlapping appointments)

if (IsOverlapping(reservationDetails.RoomId, reservationDetails.StartTime, reservationDetails.EndTime))

{

ModelState.AddModelError(string.Empty, "This room is already reserved for the selected time.");

return View(reservationDetails);

}

// Save reservation to the database

var reservation = new Reservation

{

RoomId = reservationDetails.RoomId,

StartTime = reservationDetails.StartTime,

EndTime = reservationDetails.EndTime,

UserName = User.Identity.Name // assuming user authentication is set up

};

\_context.Reservations.Add(reservation);

\_context.SaveChanges();

return RedirectToAction(nameof(Index));

}

// If validation fails, return to the reservation form with errors

return View(reservationDetails);

}

private bool IsOverlapping(int roomId, DateTime startTime, DateTime endTime)

{

// Check if there is any existing reservation for the selected room and overlapping time

return \_context.Reservations.Any(r =>

r.RoomId == roomId &&

((startTime >= r.StartTime && startTime < r.EndTime) || (endTime > r.StartTime && endTime <= r.EndTime)));

}

}

```

### Step 3: Update the Reserve View

Update the `Reserve.cshtml` view to include the input fields for reservation details.

```html

<!-- Views/Reservation/Reserve.cshtml -->

@model ReservationDetailsViewModel

<h2>Reserve Room</h2>

<form asp-controller="Reservation" asp-action="Reserve" method="post">

<div asp-validation-summary="All" class="text-danger"></div>

<div class="form-group">

<label asp-for="RoomId">Room</label>

<input asp-for="RoomId" class="form-control" readonly />

<span asp-validation-for="RoomId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="StartTime"></label>

<input asp-for="StartTime" type="datetime-local" class="form-control" />

<span asp-validation-for="StartTime" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="EndTime"></label>

<input asp-for="EndTime" type="datetime-local" class="form-control" />

<span asp-validation-for="EndTime" class="text-danger"></span>

</div>

<button type="submit" class="btn btn-primary">Reserve</button>

</form>

```

This example assumes that you have set up user authentication. If not, you may need to adjust the `UserName` property in the `Reserve` action to match your authentication setup.

Now, when a user attempts to reserve a room, the system checks for overlapping appointments and displays appropriate error messages. The input fields for reservation details include the room, start time, and end time. Adjust the code based on your specific requirements and authentication setup.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Severity | Code | Description | Project | File | Line | Suppression State |
| Error | CS0104 | 'IHostingEnvironment' is an ambiguous reference between 'Microsoft.AspNetCore.Hosting.IHostingEnvironment' and 'Microsoft.Extensions.Hosting.IHostingEnvironment' | RoomReservationApp | G:\Pebble Test\RoomReservationApp\Startup.cs | 14 | Active |
|  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Severity | Code | Description | Project | File | Line | Suppression State |
| Error (active) | CS1061 | 'Reservation' does not contain a definition for 'Room' and no accessible extension method 'Room' accepting a first argument of type 'Reservation' could be found (are you missing a using directive or an assembly reference?) | RoomReservationApp | g:\Pebble Test\RoomReservationApp\Views\Reservation\Index.cshtml | 13 |  |