Assessment: ASP.NET Core MVC Web Application Development

Overview:

This assessment is designed to test your knowledge and skills in developing an ASP.NET Core MVC web application. You will have 4 hours to complete the task, which involves building a functional web application based on a real-world scenario. Focus on applying best practices, including proper use of MVC patterns, data handling, validation, and user experience.

Problem Statement: Event Management System

Scenario:

You have been hired by a company that organizes various events like conferences, workshops, and meetups. They need a web application to manage their events and attendee registrations. The application should allow admins to create and manage events, and allow users to view upcoming events and register for them.

Requirements:

1. **Event Management:**
   - **Create Event:** Admins should be able to create new events with the following details:
     - Event Name
     - Description
     - Date and Time
     - Location
     - Maximum Number of Attendees
     - Price per Ticket (if applicable)
   - **Update Event:** Admins should be able to update existing events.
   - **Delete Event:** Admins should be able to delete events that have not yet occurred.
   - **View Events:** Both admins and users should be able to view a list of upcoming events with details.
2. **Attendee Registration:**
   - **View Event Details:** Users should be able to view detailed information about each event.
   - **Register for an Event:** Users should be able to register for an event. Each user can register only once for a particular event.

- **View My Registrations:** Users should be able to view a list of events they have registered for.
- **Cancel Registration:** Users should be able to cancel their registration before the event date.

3. **User Management:**
   - **User Registration:** Implement a basic user registration and login system.
   - **Role Management:** Differentiate between normal users and admin users.
   - **Authorization:** Admin functionalities should be accessible only to admin users.
4. **Validation & Error Handling:**
   - Implement proper validation for all input fields.
   - Handle errors gracefully and provide meaningful feedback to the user.
5. **Database Design:**
   - Use Entity Framework Core to design and interact with the database.
   - Create appropriate models for Event, User, and Registration.
   - Implement necessary relationships between models (e.g., one-to-many between Event and Registration).
6. **UI/UX:**
   - Use Bootstrap or a similar framework to create a responsive and user-friendly interface.
   - Ensure the application is intuitive and easy to navigate.
7. **Bonus:**
   - Implement search functionality for events based on name, date, or location.
   - Add an option for users to filter events by category (e.g., Conferences, Workshops).
   - Allow users to download a PDF ticket upon successful registration.

Submission:

- Submit the project as a ZIP file containing all source code.
- Include a README file with instructions on how to run the application locally.
- Provide any necessary seed data or scripts to set up the database.

Evaluation Criteria:

- **Functionality:** Does the application meet the requirements? Are all features working as expected?

- **Code Quality:** Is the code well-structured, following MVC principles? Are best practices in place for readability, maintainability, and performance?
- **User Experience:** Is the UI clean and intuitive? Does the application provide a good user experience?
- **Bonus Features:** Implementation of additional features will be considered for extra credit.

Good luck! Make sure to manage your time effectively, and aim to complete the core features before working on any bonus features.

Note: If you need any assistance, you can refer the below document.

Lab Exercise: Building an Event Management System with ASP.NET Core MVC

This lab exercise will guide you step-by-step through the process of building an Event Management System as described in the assessment. By the end of this exercise, you should have a fully functional web application that meets the requirements of the assessment.

Prerequisites:

- Basic knowledge of ASP.NET Core MVC.
- Familiarity with Entity Framework Core for database operations.
- Basic understanding of HTML, CSS, and Bootstrap for UI design.
- A development environment set up with Visual Studio or Visual Studio Code.

Step 1: Set Up the Project

1. **Create a New ASP.NET Core MVC Project:**
   - Open Visual Studio and create a new project.
   - Select "ASP.NET Core Web Application" and choose the MVC template.
   - Name the project EventManagementSystem.
2. **Set Up Entity Framework Core:**
   - Install the necessary NuGet packages for EF Core:

```
Install-Package Microsoft.EntityFrameworkCore

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Tools
```

   - Configure the connection string in appsettings.json:

```
"ConnectionStrings": {

  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=EventManagementSystemDb;Trusted_Connection=True;"

}
```

   - Add a DbContext class named ApplicationDbContext in the Data folder:

```
public class ApplicationDbContext : DbContext

{

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

        : base(options)
    {
    }

    public DbSet<Event> Events { get; set; }
    public DbSet<Registration> Registrations { get; set; }
    public DbSet<ApplicationUser> Users { get; set; }
}
```

3. **Create Models:**
   - In the Models folder, create three classes: Event, Registration, and ApplicationUser:

```
public class Event
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public DateTime DateAndTime { get; set; }
    public string Location { get; set; }
    public int MaxAttendees { get; set; }
    public decimal? Price { get; set; }

    public ICollection<Registration> Registrations { get; set; }
}

public class Registration
{
    public int Id { get; set; }
    public int EventId { get; set; }
    public string UserId { get; set; }

    public Event Event { get; set; }
    public ApplicationUser User { get; set; }
}
```

```
public class ApplicationUser : IdentityUser
{
    public ICollection<Registration> Registrations { get; set; }
}
```

- Add relationships between Event and Registration models.

4. **Apply Migrations and Create the Database:**
   - Run the following commands in the Package Manager Console:

```
Add-Migration InitialCreate
Update-Database
```

Step 2: Implement User Management

1. **Set Up Identity:**
   - Modify Startup.cs to configure identity services:

```
services.AddDefaultIdentity<ApplicationUser>()
        .AddRoles<IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>();
```

   - Scaffold the Identity UI for user registration and login:

```
dotnet aspnet-codegenerator identity -dc ApplicationDbContext
```

2. **Add Roles:**
   - Seed the database with admin and user roles in the ApplicationDbContext class:

```
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    builder.Entity<IdentityRole>().HasData(
        new IdentityRole { Name = "Admin", NormalizedName = "ADMIN" },
        new IdentityRole { Name = "User", NormalizedName = "USER" }
    );
}
```

   - Update the database:

```
Update-Database
```

3. **Restrict Access to Admin Features:**

- Use [Authorize(Roles = "Admin")] attribute to restrict access to admin actions in the controller.

Step 3: Implement Event Management Features

1. **Create an EventsController:**
   - Scaffold a new controller named EventsController with views using EF.
   - Add actions for creating, updating, and deleting events:

```
[Authorize(Roles = "Admin")]
public async Task<IActionResult> Create(Event @event)
{
    if (ModelState.IsValid)
    {
        _context.Add(@event);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(@event);
}
```

2. **Implement Event Listing:**
   - Modify the Index action to display a list of upcoming events to both admins and users:

```
public async Task<IActionResult> Index()
{
    var events = await _context.Events.ToListAsync();
    return View(events);
}
```

3. **Create Views for Event Management:**
   - Customize the generated views for creating, editing, and listing events using Bootstrap for a responsive UI.

Step 4: Implement Attendee Registration

1. **Create a RegistrationsController:**

- Scaffold a new controller named RegistrationsController:

```
public async Task<IActionResult> Register(int eventId)
{
    var registration = new Registration
    {
        EventId = eventId,
        UserId = _userManager.GetUserId(User)
    };

    _context.Add(registration);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(MyRegistrations));
}
```

2. **Create Views for Registrations:**
   - Implement a view for users to see their registrations.
   - Add a button on the event details page to allow users to register for the event.

Step 5: Validation and Error Handling

1. **Add Model Validation:**
   - Use data annotations to enforce validation rules in the Event and Registration models:

```
[Required]
public string Name { get; set; }

[Range(1, 1000)]
public int MaxAttendees { get; set; }
```

2. **Handle Errors Gracefully:**
   - Add try-catch blocks where necessary to handle exceptions.
   - Provide user-friendly error messages.

Step 6: Bonus Features (Optional)

1. **Implement Search Functionality:**

- Add a search bar on the event listing page to filter events by name, date, or location.

2. **Implement Event Categories:**
    - Add a category field to the Event model.
    - Allow users to filter events by category.
3. **Generate PDF Tickets:**
    - Use a library like iTextSharp to generate PDF tickets for registered users.

Step 7: Final Touches

1. **Test the Application:**
    - Thoroughly test all features to ensure they work as expected.
    - Make sure the UI is responsive and user-friendly.
2. **Prepare for Submission:**
    - Clean up your code, remove any unnecessary files, and ensure everything is well-documented.
    - Create a README file with instructions on how to set up and run the application.