Event Management System

Overview

In this assignment, you'll build a small Event Management System with a React frontend and ASP.NET Core Web API backend. The system will allow users to create, view, update, and delete events, as well as register for events.

Duration

Approximately 1 to 2 hours.

Objectives

1. Create a full-stack application with React JS frontend and ASP.NET Core Web API backend
2. Implement proper routing in React using React Router
3. Use Context API for state management
4. Build RESTful APIs with ASP.NET Core
5. Implement database operations using Entity Framework Core

Technical Requirements

Frontend (React JS)

- Create a React application with the following pages:
  - Home page: List of all events
  - Event details page: Display detailed information about a specific event
  - Create/Edit event page: Form to add or edit events
  - Registration page: Form to register for events
- Implement routing using React Router
- Use Context API for managing global state (event data, form submissions)
- Implement responsive design with CSS

Backend (ASP.NET Core Web API)

- Create a RESTful API with the following endpoints:
  - GET /api/events: Get all events
  - GET /api/events/{id}: Get specific event
  - POST /api/events: Create a new event
  - PUT /api/events/{id}: Update an event
  - DELETE /api/events/{id}: Delete an event
  - POST /api/registrations: Register for an event
  - GET /api/events/{id}/registrations: Get all registrations for an event
- Use Entity Framework Core for database operations
- Implement proper error handling and validation

Database Design

- Events table: Id, Title, Description, Date, Location, MaxAttendees
- Registrations table: Id, EventId, Name, Email, RegistrationDate

Step-by-Step Instructions

Backend Setup

1. Create a new ASP.NET Core Web API project
2. Set up Entity Framework Core with the required models
3. Create DbContext and configure database connection
4. Implement controllers for the API endpoints
5. Test endpoints using Swagger or Postman

Frontend Setup

1. Create a new React application
2. Install required packages (react-router-dom, axios)
3. Set up routing structure
4. Create a Context for application state (EventContext)
5. Implement components for each page
6. Connect frontend to backend API

Evaluation Criteria

- Functionality: All features work as expected
- Code quality: Clean, well-structured code with proper error handling
- UI/UX: User-friendly interface with responsive design
- API design: RESTful API with proper endpoints and status codes
- State management: Effective use of Context API

Hints

- For the Context API, create an EventContext to manage event data across components
- Use axios for API calls in React
- Implement loading states and error handling in your components
- For EF Core, use Code-First approach for simplicity
- Consider using Data Transfer Objects (DTOs) for API responses
- Test your API endpoints thoroughly before connecting to the frontend
- Use repository pattern in your ASP.NET Core application

Bonus Challenges

- Implement filtering and sorting options for events
- Add pagination for the events list
- Create a dashboard to show event statistics
- Implement real-time updates using SignalR