

Capstone Project Submission: Event Management System

Project Title: Event Management System

Name: Maram Rohith

Date: 20th March 2025

Batch Name: .Net React C3

Instructor: Mrs. Jyothi S Patil

Table of Contents:

TABLE OF CONTENTS

1. Problem Definition and Objectives	3
- Brief Overview of Problem Statement	
2. Project Goals and Objectives	4
- Goals	
- Objectives	
3. Frontend and Backend Architecture	5
- Frontend (React)	
- Backend (ASP.NET Core Web API)	
- Database (Microsoft SQL Server)	
4. Components Breakdown	8
- Frontend Breakdown (React)	
- Major Components	
- EventSearch.js	
- EventList.js	
- EventCard.js	
- UserEventDetails.js	
- Navbar.js	
5. Routing (React Router)	10
6. Authentication Mechanism	10
- Authentication Flow	
- User Registration Process	
- User Login Process	
- JWT Token Generation	
7. Database Design & Storage Optimization	13
- Entity-Relationship Diagram (ERD)	
- Optimization Techniques	
8. Installation	15

Problem Definition and Objectives:

Brief Overview of Problem Statement:

In today's fast-paced world, organizing and managing events can be a difficult task for both individuals and organizations. There is a lack of centralized systems that can efficiently handle event creation, management, RSVPs, and notifications for attendees.

The problem arises from the difficulty in coordinating event details, tracking attendees, and ensuring that everyone involved stays informed. Manual processes, such as sending emails, tracking RSVPs through spreadsheets, and communicating last-minute changes, can lead to confusion, missed updates, and poor attendance.

An Event Management System can address these challenges by providing a user-friendly platform for creating, organizing, and managing events. It should include features such as event creation, sending invitations, collecting RSVPs, sending notifications, and updating attendees about changes in real-time. This system would not only improve the user experience but also increase attendance and engagement by keeping everyone on the same page. A platform for users to create and manage events with RSVPs and notifications.

Project Goals and Objectives:

Goals:

1. **Efficient Event Management** - Provide a Centralized platform for users to create, manage and attend events.
2. **User-Friendly Interface** – Ensure a responsive and intuitive UI with React and Bootstrap for smooth user interaction.
3. **Robust Backend** – Use .NET Core to handle API requests, authentication, and authorization for secure login.
4. **Data Storage** – Utilize MS SQL to store and manage event data efficiently.
5. **Email Notifications** – Integrate an email system (e.g. EmailJS) for event confirmations and reminders.

Objectives:

1. **User Authentication** – Secure login/logout system to track users.
2. **Event Creation and Search** – Admin can add, edit, and filter events by name and date.
3. **Responsive UI** – Ensure the app looks good across all screen sizes.
4. **Data Persistence** – Store event details, attendees, and notifications in MS SQL.
5. **Error Handling** – Implement clear error messages and server-side validation.

Frontend and Backend Architecture:

Frontend (React):

React is a JavaScript library for building user interfaces, particularly for single-page applications where you need a fast, interactive user experience. It is developed and maintained by Facebook.

Key Features:

1. **Component-Based Architecture:** React applications are built as a tree of components, which makes them reusable, maintainable, and scalable.
2. **Virtual DOM:** React is known for its efficiency and performance, thanks to its virtual DOM. Changes are first applied to a virtual representation of the UI, then React compares this with the real DOM and updates only the parts that changed.
3. **JSX:** React allows developers to write HTML-like syntax within JavaScript, known as JSX, which makes it easier to build components and work with UI elements.
4. **State & Props:** React uses "state" to store data within a component, and "props" to pass data between components.
5. **Routing:** With React Router, you can map URLs to components and handle page transitions without needing to reload the page. This ensures smooth, dynamic navigation in SPAs.
6. **React Native:** React Native allows developers to build native mobile applications for iOS and Android using React. This enables code sharing between web and mobile platforms.

Backend (ASP.NET Core Web API):

ASP.NET Core is a cross-platform, high-performance framework for building modern web applications and APIs. A Web API in ASP.NET Core is used to expose data and services to client applications, such as web browsers or mobile apps, typically using HTTP requests. ASP.NET Core Web API is commonly used for back-end services that interact with databases, external services etc serving as the backend to a front-end application like React.

Key Features:

1. **Cross-Platform:** ASP.NET Core can run on Windows, macOS, and Linux, making it flexible for developers.
2. **RESTful API:** ASP.NET Core Web API typically follows REST principles (Representational State Transfer), where it exposes endpoints (URLs) to interact with resources via HTTP methods like GET, POST, PUT, DELETE.
3. **Dependency Injection:** Built-in support for dependency injection to manage application services and their lifecycle.
4. **Middleware:** ASP.NET Core Web API uses a pipeline of middleware to handle requests and responses, making it easy to add features like authentication, logging, or routing.
5. **Security:** It offers built-in features for securing APIs, such as JWT (JSON Web Token) authentication, OAuth2, and role-based authorization.

Database (Microsoft SQL Server (MS SQL)):

MS SQL Server is a relational database management system (RDBMS) developed by Microsoft. It is used to store and manage data for applications. MS SQL Server is commonly used for applications that require strong data integrity, complex queries, and relationships between data. It's ideal for enterprise applications, data warehousing, and e-commerce platforms.

Key Features:

1. **Structured Data:** MS SQL is ideal for storing structured data in tables and supports relational operations like joins, constraints, and transactions.
2. **Data Integrity & Security:** It supports ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure data integrity and consistency in transactions. It also provides encryption, user roles, and permissions for secure access to data.
3. **Scalability & Performance:** MS SQL Server supports large datasets and offers features like indexing, optimization, and stored procedures to enhance query performance.
4. **Integration:** MS SQL integrates well with .NET applications, making it a popular choice for back-end data storage in ASP.NET Core applications.
5. **Robust :** MS SQL Server provides a robust database solution for storing and querying application data efficiently.

Components Breakdown:

Frontend Breakdown (React):


The frontend is developed using React with React-Bootstrap for styling and UI components. It follows a structured component-based architecture with state management using React hooks.

Major Components:

1. EventSearch.js:

- Allows users to search for events by name and date.
- Uses React state (useState) to manage input fields.
- triggers filtering logic to update the event list dynamically.

Events



2. EventList.js:

- Displays a grid of upcoming events.
- Fetches events from the backend API and updates dynamically.
- Maps over the event data and renders individual EventCard components.

Upcoming Events

party
Date: 19/03/2025
Location: Chennai
party in chennai

wedding
Date: 23/03/2025
Location: Tirupati
wedding in Tirupati

Checking 123
Date: 27/03/2025
Location: Kolkata
Checking in kolkata

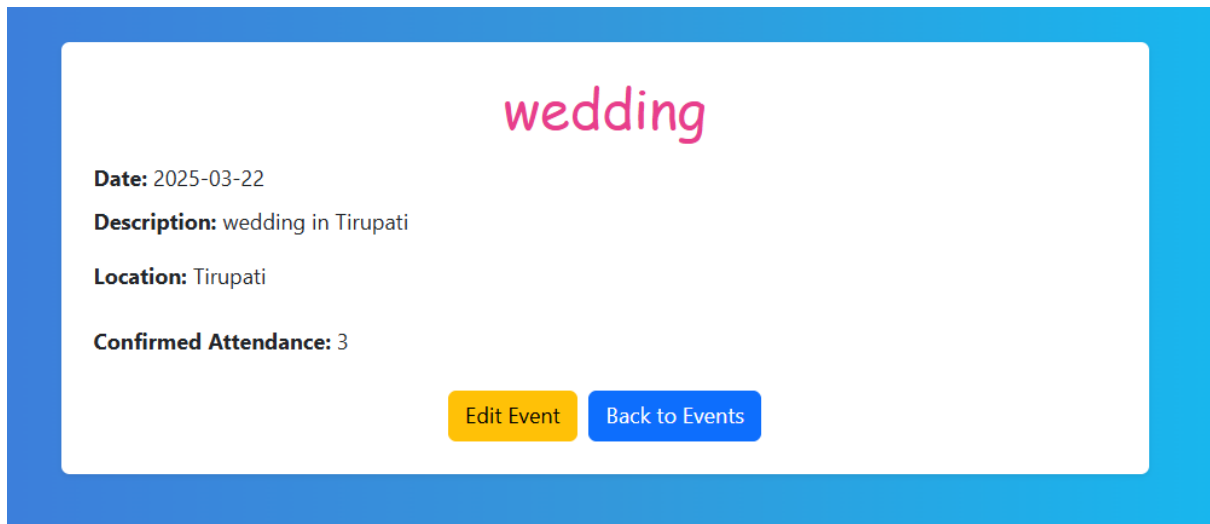
Tirumala Puja
Date: 29/03/2025
Location: Tirupati
Grand Puja in Tiruma

Get together
Date: 16/03/2025
Location: AT my place
My home

Email prac
Date: 21/03/2025
Location: efsf
vaevsev

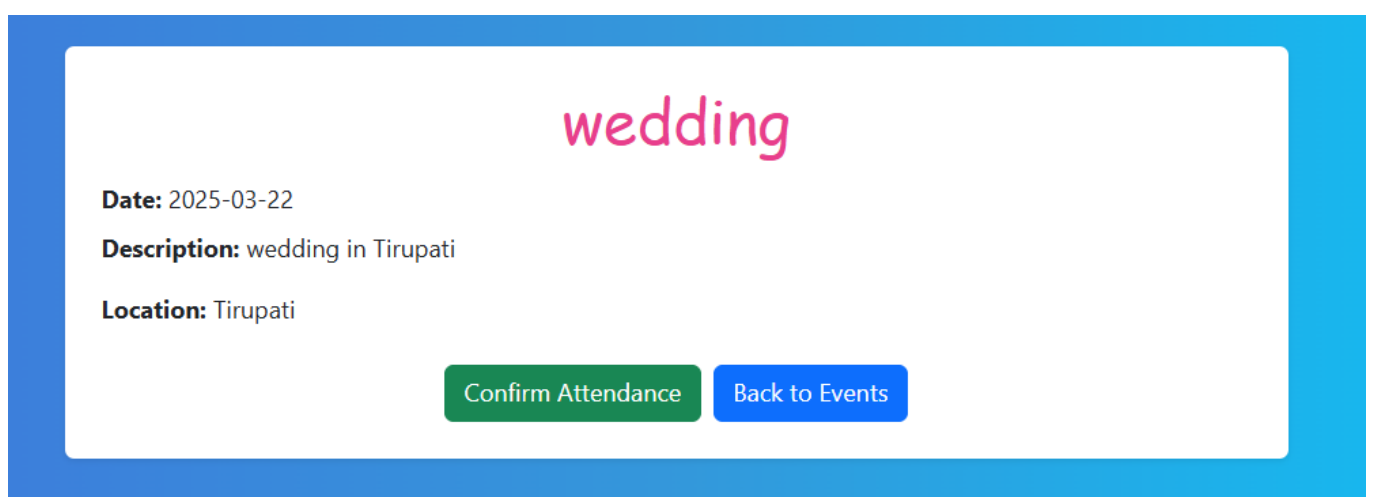
3. EventCard.js:

- Represents a single event card displaying event details.
- Includes a button to view event details.



4. UserEventDetails.js

- Displays detailed information about a selected event.
- Provides an option to confirm attendance.
- Sends a confirmation email using EmailJS.



5. Navbar.js:

- Contains navigation links and a logout button.
- Displays user notifications and authentication status.

6. Routing (React Router):

- Uses **react-router-dom** for navigation between different pages:
- /events → Event listing
- /event/:id → Event details page
- /login → User authentication
- /register → User registration

Authentication Mechanism:

The authentication mechanism for the Event Management System, which is developed using React (Frontend), ASP.NET Core (Backend), and MS SQL Server (Database). The system ensures secure user authentication using JWT (JSON Web Token) authentication.

Authentication Flow:

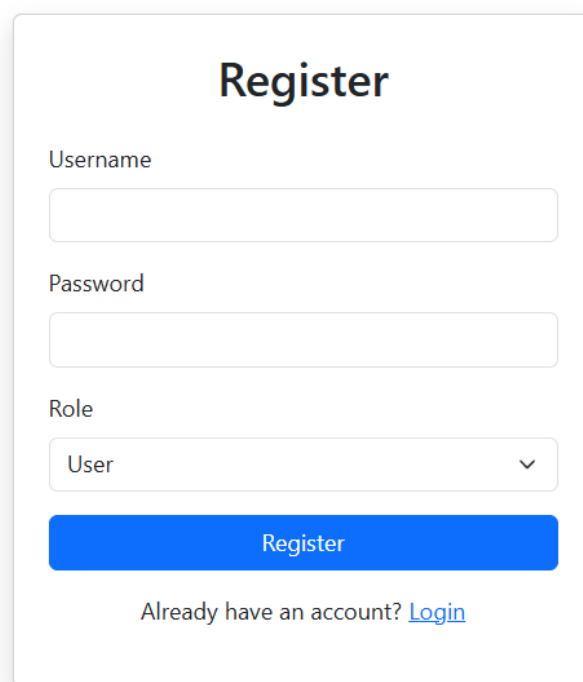
1. **User Registration:** Users sign up by providing their email, password, and other necessary details.
2. **User Login:** Users enter their credentials, which are validated against the database.
3. **Token Generation:** Upon successful authentication, a JWT token is issued.

4. **Token Storage:** The token is stored on the client-side (localStorage/sessionStorage) for subsequent API requests.
5. **Protected API Calls:** The token is sent in the request headers for authentication.
6. **Token Validation:** The backend verifies the token before processing the request.
7. **Session Expiry & Refresh:** Tokens have an expiry time, and users need to log in again after expiration.

User Registration Process:

Endpoint: /api/auth/register

- User submits registration details (email, password, name, etc.).
- Passwords are hashed using **ASP.NET Core Identity (bcrypt)**.
- The user details are stored in the **MS SQL database**.
- Response: Success message or error message if the email is already in use.



Register

Username

Password

Role

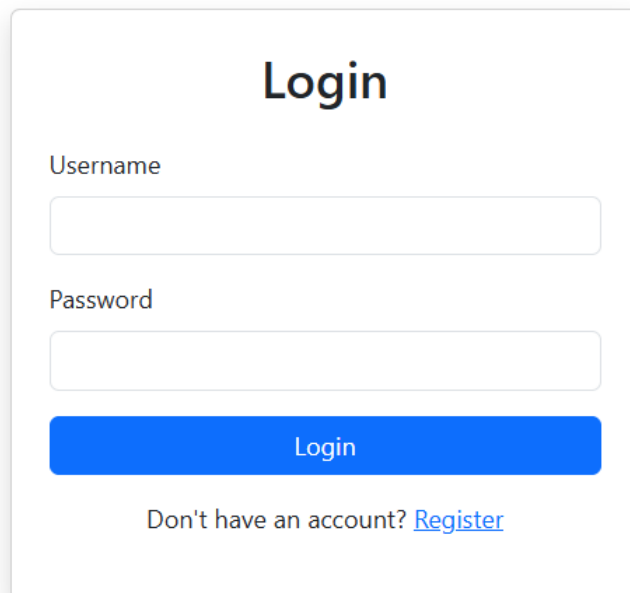
[Register](#)

Already have an account? [Login](#)

User Login Process:

Endpoint: /api/auth/login

- User submits email and password.
- Password is verified against the stored hashed password.
- If authentication is successful, a **JWT token** is generated and returned.
- Response: Token along with user details (excluding password).

A login form with a white background and a subtle shadow. At the top, the word "Login" is centered in a large, bold, dark font. Below it, the label "Username" is positioned above a white text input field with a light gray border. Further down, the label "Password" is positioned above another white text input field with a light gray border. Below the password field is a solid blue button with the word "Login" in white text. At the bottom of the form, the text "Don't have an account? [Register](#)" is displayed, with "Register" being a blue hyperlink.

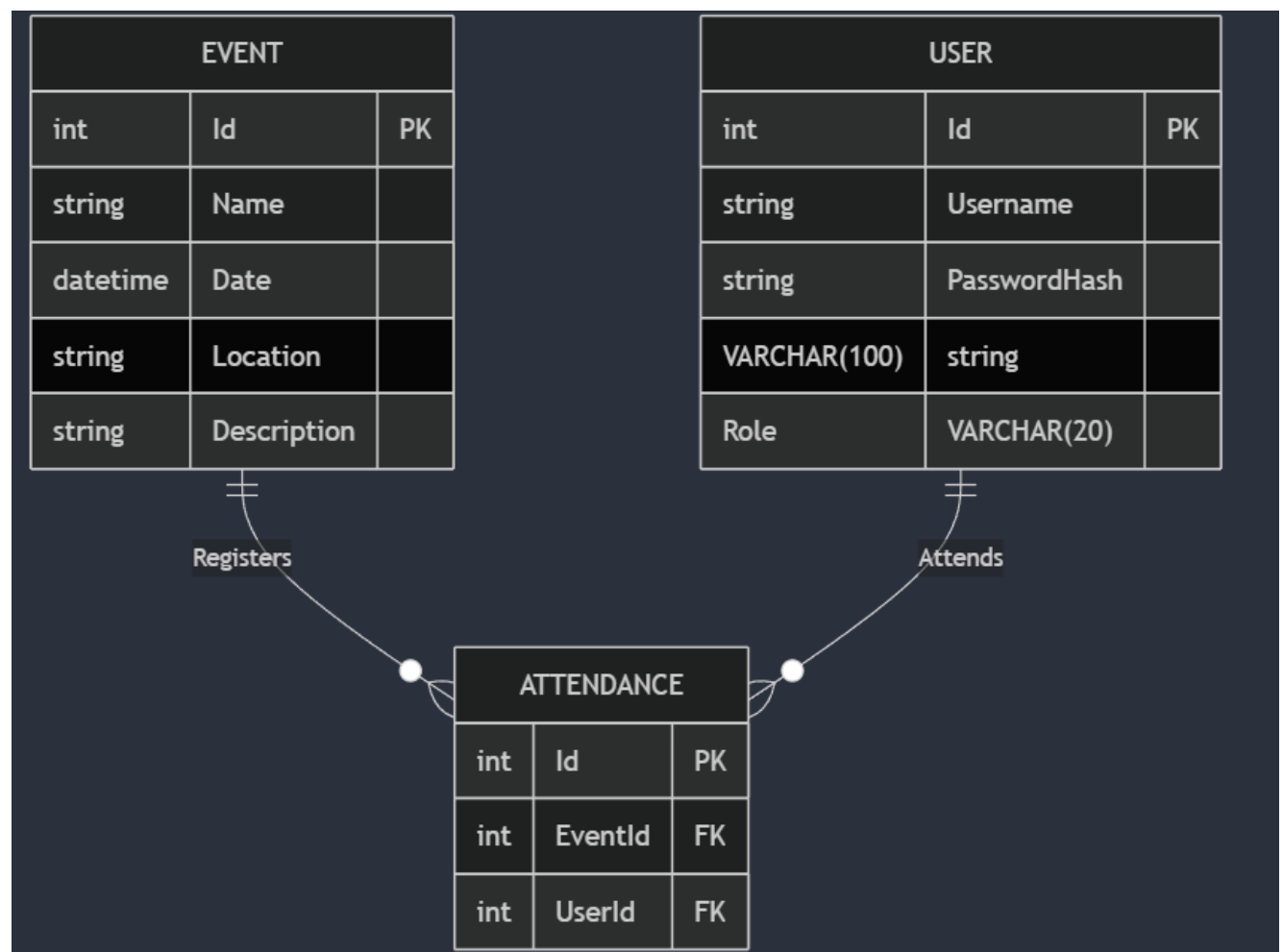
JWT Token Generation:

- The JWT token is generated using ASP.NET Core Authentication & Authorization.

- It consists of three parts:
 1. Header: Specifies the token type and algorithm (e.g., HS256).
 2. Payload: Contains user ID, roles, and expiration time.
 3. Signature: Ensures token integrity using a secret key.

Database Design & Storage Optimization

Entity-Relationship Diagram (ERD):



Optimization techniques for efficient queries:

1. Indexing Strategies:

- Indexes are crucial for speeding up data retrieval. Proper indexing significantly reduces the need for full table scans.

2. Query Tuning Best Practices:

- Optimizing individual queries is essential for efficient data retrieval.

3. Database Design Considerations:

- A well-designed database contributes significantly to query efficiency.

4. Server Configuration and Maintenance:

- Proper server configuration and maintenance are vital for optimal database performance.

5. Stored Procedures for Performance Optimization

- Use Stored Procedures to optimize complex queries and reduce network overhead.

Installing this Project in your System:

For React:

The required React File is provided with the name:

Frontend_EventManagementSystem.zip

1st step: cd - to the project directory and npm install. (Should install the node modules)

2nd step: npm start.

For ASP .Net Core Api:

The required React File is provided with the name:

Backend_EventManagementSystem.zip

1st step: In appsettings.json – Change the server name to current system database server name.

2nd step: go to Debug -> Start Debugging (or) F5

For Database:

The required SQL File is provided with the name

Database_Schema_EventManagementSystem.sql

Notes:

- If the port number is running different other than in <https://localhost:7013> for the backend files i.e. ASP .NET Core Web API(MVC). Please change the port number accordingly in React Files i.e.

1. EventDetails.js
2. Login.js
3. Register.js
4. UserEventDetails.js
5. Redux -> actions -> EventAction.js

The Port number need to changed in all these files.

- The migration files are already present in the Migrations folder. So we can start the application directly by clicking f5. If in any case the backend is not connected to database, Please use these commands in Package Manager Console to add the migration files:

➔ add-migration FinalMigration

➔ update-database

This Should add the necessary migration files and automatically create the Database and tables in MS SQL. Try this step only if the sql file does not create the database and its tables.

- After starting the application:
 1. First Register using valid email and password as Admin or User
 2. Then Login using the credentials used for Register.
 3. Admin can perform CRUD operations while the user cannot.
 4. In the Search section please provide the name of the event and **Double Click** on search button.

5. To again view all the events **Clear** the search and Click on Search button again.
6. The Date functionality in Search may not work so please avoid this functionality.