Events Management Application: Two-Tier Architecture Report

1. Architecture Overview

The Events Management Application is designed using a Two-Tier Architecture, where the frontend (client) and backend (server) operate independently but communicate directly. This structure separates the presentation layer (user interface) from the business logic and data access layer (backend and database), ensuring better maintainability and scalability.

System Components

- Frontend (Client Layer): Developed using HTML, CSS, and JavaScript. It interacts with the backend via RESTful API requests.
- Backend (Application Layer): Built with ASP.NET Core Web API, handling HTTP requests, executing business logic, and interfacing with the database.
- Database (Data Layer): SQL Server stores all event-related data, managed using Entity Framework Core.

2. Implementation Steps

1. Backend Development (ASP.NET Core Web API)

- Created a new ASP.NET Core Web API project.
- Set up Entity Framework Core (EF Core) for database interactions.
- Configured SQL Server as the database provider.
- Defined the Event model with properties: Id, Name, Location, and Date.
- Implemented ApplicationDbContext to manage database access.
- Created EventsController with CRUD operations using RESTful API principles.
- Applied EF Core migrations to generate and update the database schema.

2. Database Configuration

- Defined the connection string in appsettings.json to connect to SQL Server.
- Run dotnet ef database update to apply migrations and create the database.
- Verified database structure using SQL Server Management Studio (SSMS).

3. Frontend Development (HTML, CSS, JavaScript)

- Developed a simple HTML structure to allow users to manage events.
- Styled the interface using CSS.
- Used JavaScript (Fetch API) to make asynchronous calls to the backend.
- Implemented dynamic UI updates to reflect event creation, updates, and deletions.
- Handled form validation to prevent empty event submissions.

4. API Integration and Testing

- Connected the frontend with the backend by making AJAX requests to API endpoints.
- Implemented CORS (Cross-Origin Resource Sharing) to allow frontend-backend communication.

3. Advantages and Challenges Faced

Advantages

- Separation of Concerns: Clear distinction between frontend and backend improves code maintainability.
- Scalability: The frontend and backend can be deployed and scaled independently.
- Reusability: The backend API can be used by multiple client applications (e.g., mobile, desktop).
- Improved Security: The database is not exposed to the client directly.

Challenges Faced

- CORS Issues: Initially, cross-origin resource sharing had to be enabled to allow frontend-backend communication.
- Database Connection Issues: SQL Server configurations required fine-tuning for proper connectivity.
- State Management: Since frontend and backend are separate, managing session states requires additional mechanisms.

4. Comparison with Other Architectures

Architecture	Description	Pros	Cons
Single-Tier	All components in one system	Simple, fast, easy to develop	Not scalable, harder to maintain
Two-Tier	Frontend and Backend are separate	Better separation, scalable	Requires proper API integration
Three-Tier	UI, API, and Database separated	Highly scalable, secure	More complex to implement

Conclusion

The Two-Tier Architecture used in the Events Management Application successfully separates the frontend from the backend while maintaining direct interaction with the database. This approach improves modularity, scalability, and maintainability while posing challenges such as CORS handling and database configuration. It provides a cleaner and more flexible solution while avoiding the complexity of a full Three-Tier approach.