System Requirements Specification

Index

For

Event Management System

Version 1.0

EVENT MANAGEMENT SYSTEM

System Requirements Specification

1 PROJECT ABSTRACT

The **Event Management System** is a .Net Core RESTful Web API 3.1 with MS SQL Server database connectivity. It enables users to manage various aspects of event planning and organization.

Following is the requirement specifications:

	Event Management System
Modules	
1	Event
Event Module Functionalities	
1	Create an Event
2	Update the existing Event details
3	Get the Event by Id
4	Get all Events
5	Delete an Event
6	Search for Events by Name
7	Search for Events by Status

2 Assumptions, Dependencies, Risks / Constraints

2.1 Event Constraints

- When fetching an Event by ID, if the event ID does not exist, the operation should throw a custom exception.
- When updating an Event, if the event ID does not exist, the operation should throw a custom exception.
- When removing an Event, if the event ID does not exist, the operation should throw a custom exception.

Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in ViewModel classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3 Business Validations

- Id (Long) Key, Not Null
- Name (String) of the event is not null, min 3 and max 20 characters.
- Description (String) of the event is not null and should be between 5 and 200 characters in length.
- Status (String) of the event is not null.

4 REST ENDPOINTS

Rest End-points to be exposed in the controller along with method details for the same to be created

4.1 EVENTCONTROLLER

URL Exposed	Purpose
1. /events Http Method: GET Parameter: - Return: <IEnumerable<Events >>	Fetches all the events
2. /events Http Method: POST Parameter 1: Event Return: Event	Add a new event
3. /events/{id} Http Method: DELETE Parameter 1: Long (id) Return: -	Delete events with given events id
4. /events/{id} Http Method: GET Parameter 1: Long (id) Return: Event	Fetches the event with the given id
5. /events/{id} Http Method: PUT Parameter 1: Long (id) Parameter 2: EventViewModel Return: Event	Updates existing event
6. /events/searchByName?name={name} Http Method: GET Parameter 1: String (name) Return: <IEnumerable<Events >>	Fetches the event with the given name
7. /events/searchByStatus?status={status} Http Method: GET Parameter 1: String (status) Return: <IEnumerable<Events >>	Fetches the event with the given status

5. TEMPLATE CODE STRUCTURE

5.1 Package: EventManagement

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Event Controller	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

5.2 Package: EventManagement.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	IEventService, interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	EventService CS file file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially Implemented
Repository	IEventRepository EventRepository CS file and interface.	All these interfaces and class files contain all CRUD operation code for the database. Need to provide	Partially Implemented

		implementation for service related functionalities	
ViewModels	EventViewModel,	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Partially Implemented

5.3 Package: EventManagement.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	EventManagementDbContext cs file	All database Connection and collection setting class	Already Implemented

5.4 Package: EventManagement.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	Event CS file	All Entities/Domain attribute are used for pass the data in controller. Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Already Implemented

5.5 Package: EventManagement.Tests

Resources

The EventManagement.Tests project contains all test case classes and functions for code evaluation. Don't edit or change anything inside this project.

6. CONSIDERATIONS

A. There is no roles in this application
B. You can perform the following possible action

Event

7. EXECUTION STEPS TO FOLLOW

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**

2. **To open the command terminal the test takers, need to go to the Application menu (Three horizontal lines at left top) Terminal → New Terminal.**

3. **On command prompt, cd into your project folder (cd <Your-Project-folder>).**

4. **To connect SQL server from terminal:**
   (EventManagement /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**

     1> Create Database EventManagementDb
     2> Go

5. **Steps to Apply Migration(Code first approach):**
   - **Press Ctrl+C to get back to command prompt**
   - **Run following command to apply migration-**
     (EventManagement /dotnet-ef database update)

6. **To check whether migrations are applied from terminal:**
   (EventManagement /sqlcmd -S localhost -U sa -P pass@word1)

   1> Use EventManagementDb
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go

7. **To build your project use command:**
   (EventManagement /dotnet build)

8. **To launch your application, Run the following command to run the application:**
   (EventManagement/dotnet run)

9. **This editor Auto Saves the code.**

10. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**

11. **To test web-based applications on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

    Note: The application will not run in the local browser

12. **To run the test cases in CMD, Run the following command to test the application:**
    (EventManagement/dotnet test --logger "console;verbosity=detailed")
    (You can run this command multiple times to identify the test case status, and refactor code to make maximum test cases passed before final submission)

13. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**

14. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**

15. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**
