System Requirements Specification

Index

For

Calorie Tracker Application

Version 1.0

CALORIE TRACKER APPLICATION

System Requirements Specification

1 PROJECT ABSTRACT

The **Calorie Tracker Application** is a ASP.NET MVC 5 with MS SQL Server database connectivity. It enables users to manage various aspects of event planning and organization.

Following is the requirement specifications:

	Calorie Tracker Application
Modules	
1	Calorie
Calorie Module Functionalities	
1	Create an Calorie
2	Update the existing Calorie details
3	Get the Calorie by Id
4	Get all Calories
5	Delete an Calorie

2 Assumptions, Dependencies, Risks / Constraints

2.1 Calorie Constraints

- When fetching an Calorie by ID, if the Calorie ID does not exist, the operation should throw a custom exception.
- When updating an Calorie, if the Calorie ID does not exist, the operation should throw a custom exception.
- When removing an Calorie, if the Calorie 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 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 (Int) Key, Not Null
- Exercise (string) is not null.
- Sets (Int) not null.
- Reps (Int) not null.
- Date (DateTime) not null

3. TEMPLATE CODE STRUCTURE

3.1 Package: CalorieTrackerApp

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	CalorieTracker Controller	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Web.Config	Web.Config file	Contain all Services settings and SQL server Configuration.	Already Implemented

Interface	ICalorieTrackerService, interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	CalorieService 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	ICalorieRepository CalorieRepository CS file and interface.	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially Implemented
Models	Calorie cs file	All Entities/Domain attribute are used for pass the data in controller.	Already Implementation

5.2 Package: CalorieTrackerApp.Tests

Resources

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

6. 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:**
   (CalorieTrackerApp /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**

     1> Create Database CalorieTrackerDb
     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-**
     (CalorieTrackerApp /dotnet-ef database update)

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

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

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

8. **To launch your application, Run the following command to run the application:**
   (CalorieTrackerApp /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:**
    (CalorieTrackerApp.Tests/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.**
