System Requirements Specification

Index

For

Score Management Application

Version 1.0

SCORE MANAGEMENT APPLICATION

System Requirements Specification

1 PROJECT ABSTRACT

The **Score Management Application** is a Entity Framework 4.8 with MS SQL Server database connectivity. It enables users to manage various aspects of Score management and organization.

Following is the requirement specifications:

	Score Management Application
Modules	
1	Score
Score Module Functionalities	
1	Create an Score
2	Update the existing Score details
3	Get the Score by Id
4	Get all Scores
5	Delete an Score

2 Assumptions, Dependencies, Risks / Constraints

2.1 Score Constraints

- When fetching an Score by ID, if the Score ID does not exist, the operation should throw a custom exception.
- When updating an Score, if the Score ID does not exist, the operation should throw a custom exception.
- When removing an Score, if the Score 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
- Player Name are marked as [Required] to ensure they are not null,
- Player Score (Int) is not null.
- Score int not null
- Game Type string not null
- Notes not null
- Date (Date) of the Score 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 ScoreCONTROLLER

URL Exposed	Purpose
1. /api/Score/GetAllScores Http Method: GET Parameter: - Return: <IEnumerable<Score>>	Fetches all the Scores
2. api/Score/CreateScore Http Method: POST Parameter 1: Score Return: Score	Add a new Score
3. /api/Score/DeleteScore Http Method: DELETE Parameter 1: Int (id) Return: -	Delete Score with given Score id
4./ api/Score/GetScoreById Http Method: GET Parameter 1: Int (id) Return: Score	Fetches the Score with the given id
5. /api/Score/UpdateScore Http Method: PUT Parameter 1: Int (id) Parameter 2: Score Return: Score	Updates existing Score

5. TEMPLATE CODE STRUCTURE

5.1 Package: ScoreManagementApp

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Score 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	IScoreService, interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	ScoreService 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	IScoreRepository ScoreRepository 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	Score cs file	All Entities/Domain attribute are used for pass the data in controller.	Already Implementation

5.2 Package: ScoreManagementApp.Tests

Resources

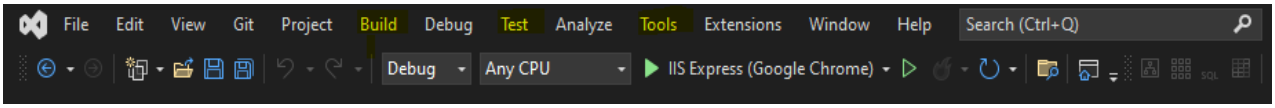
The ScoreManagementApp.Tests project contains all test case classes and functions for code evaluation. Don't edit or change anything inside this project.

6. EXECUTION STEPS TO FOLLOW

1. **After successfully cloning the project template on desktop, you will be able to see folder named with your user id. (e.g. user@gmail.com)**

2. **Go to below path and open solution file with Visual Studio.**
   Path: user@gmail.com > ScoreManagementApp > ScoreManagementApp.sln

3. **All actions such as building, compiling, running the application, and executing test cases will be performed using the Visual Studio interface. Rather than using the command terminal, the necessary operations will be initiated through the buttons, menus, and features available within the Visual Studio IDE.**



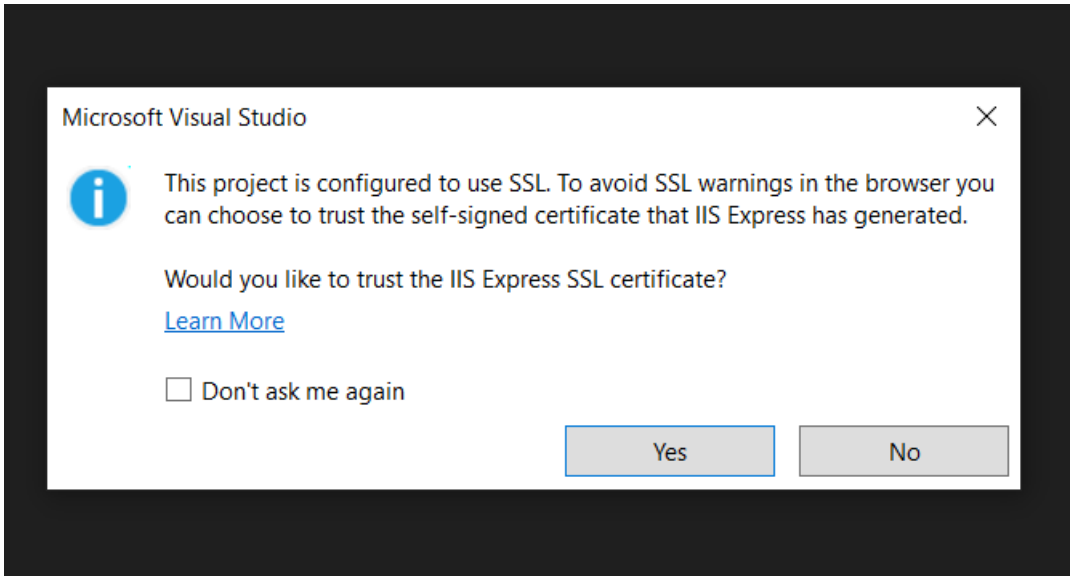
4. **Press Ctrl + S to save your code.**

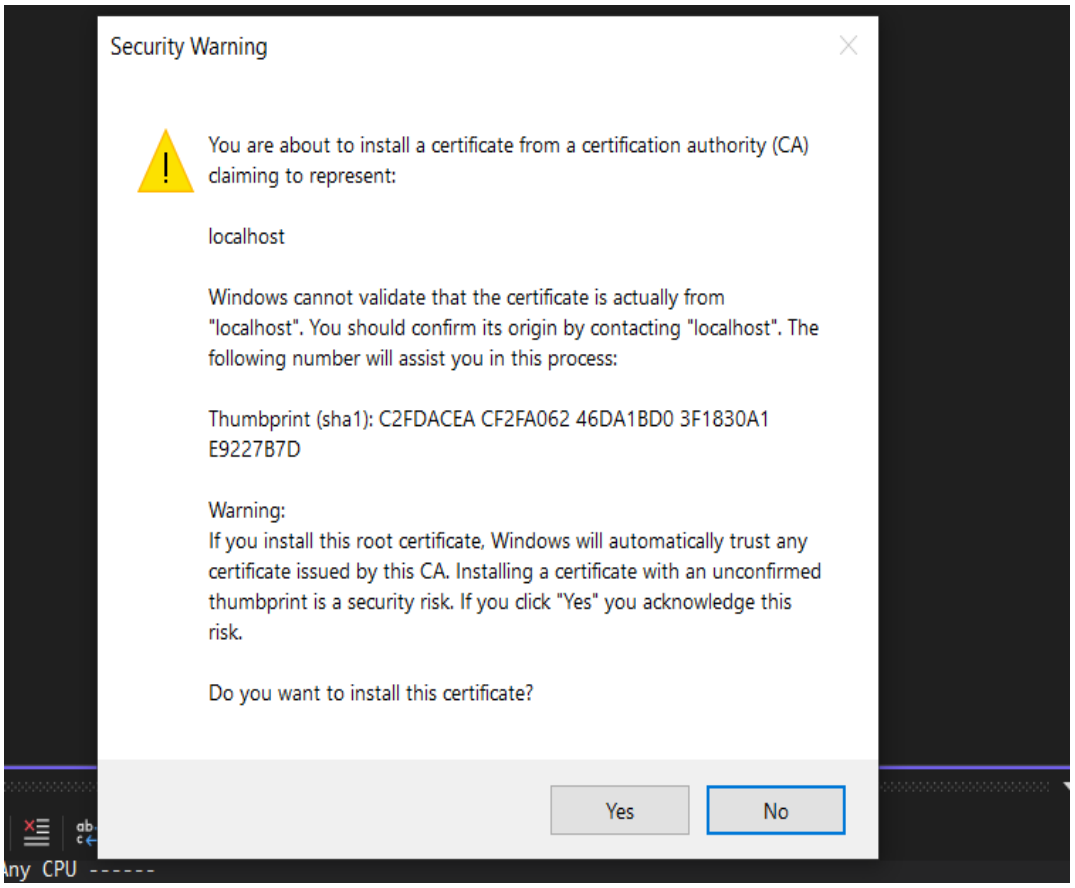
5. **Steps to Apply Migration(Code first approach):**
   - **Go to "Tools" -> "NuGet Package Manager" -> "Package Manager Console" from the top menu bar of Visual Studio.**
   - **After clicking on "Package Manager Console," a new tab should open at the bottom of the Visual Studio window, displaying the Package Manager Console.**
   - **Run following command to apply migration : update-database**

6. **Steps after applying migration :**
   - **Click on "Build"(Top Menu Bar) > "Clean Solution"**
   - **Click on "Build"(Top Menu Bar) > "Rebuild Solution"**
   - **Click on "Build" (Top Menu Bar) > "Build Solution"**

7. **To build your project in Visual Studio, click on "Build" in the top menu, then select "Build Solution" or press Ctrl + Shift + B.**

8. **To launch your application, press F5 or use Ctrl + F5 to start your application without debugging.**
   (Note: If you get below screens regarding SSL certificate, Click on YES for both screens.)

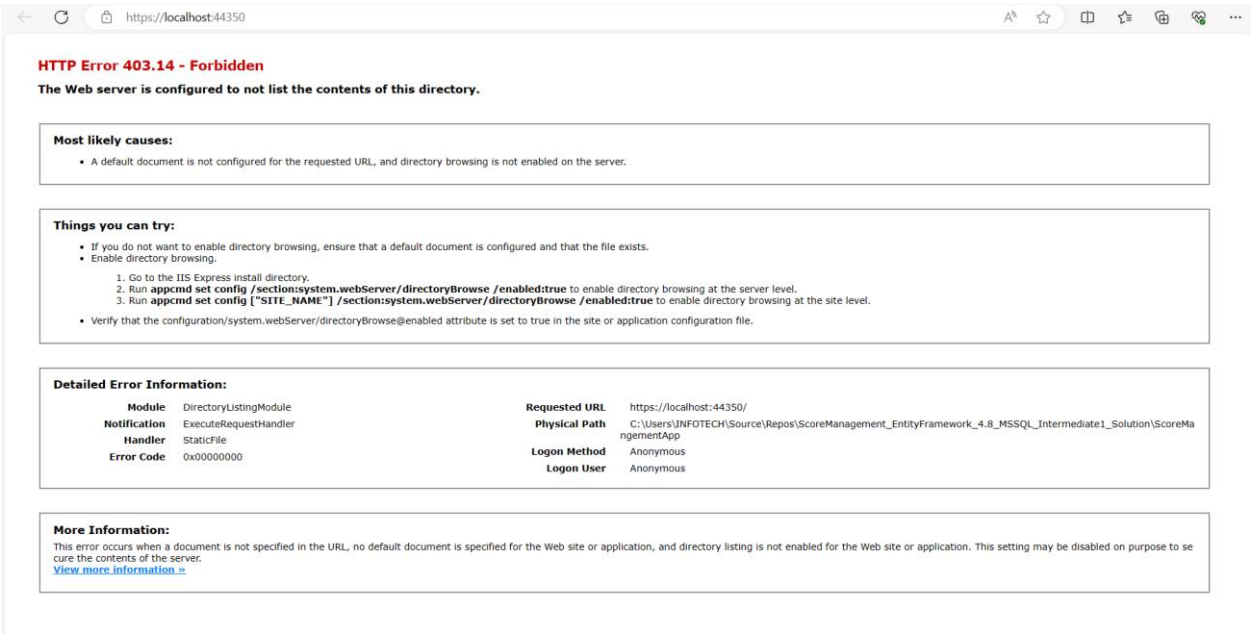




Note: The application will run in the local browser, Run Application again.

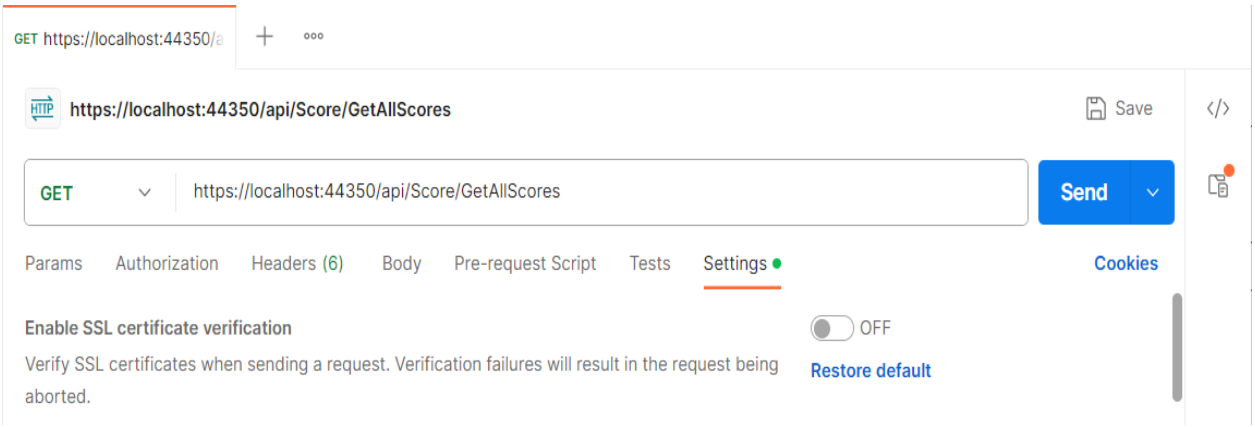
9. **To test any applications on a browser, use the internal browser in the workspace.**

(Note: This screen shows that your application is running on local browser, and the base URL is https://localhost:44350)



10. **To test any Restful application, you can use POSTMAN.**

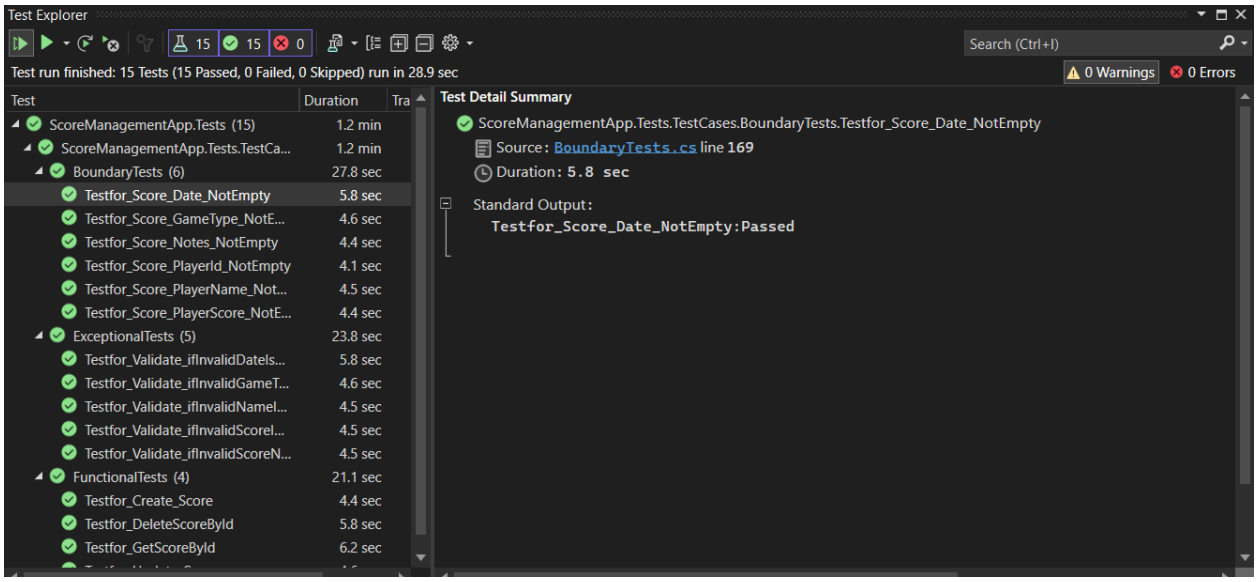
(Note: Before sending the request from postman you need to disable "Enable SSL certificate verification" from settings.)



Use same base URL link from local browser to test on postman.

11. **To run test cases in your project in Visual Studio, click on "Test" -> "Run All Tests" in the top menu. (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).



12. **Steps to push changes to GitHub:**
    - **Go to "View" -> "Git Changes" from the top menu bar of Visual Studio.**
    - **In the "Changes" window on the right side of Visual Studio, you'll see the modified files.**
    - **Enter any commit message in the "Message" box at the top of the window, and click on “Commit All” button.**
    - **After committing your changes, Click the "Push" button (Up Arrow Button) to push your committed changes to the GitHub repository.**

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 follow step-12 compulsorily. 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 follow step-12 compulsorily, 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.**