Movie Database Management System - Assignment

For the successful accomplishing of the labor course, you need develop a console-based movie management system in C#. You will work with OOP principles, events, delegates, XML/CSV data processing, Entity Framework Core, LINQ queries, and JSON serialization. The system will read movie data from XML and CSV files, upload the data to a SQL Server database using Entity Framework, retrieve data with LINQ queries, and export results into JSON files. You are also required to demonstrate the usage of events and delegates.

You need to upload the completed solution by compressed format into the Moodle system of Óbuda University until 8th of December 2024. The filename needs to be containing your name and Neptun code (e.g. *John_Smith_AAAA123.zip*).

---

1. Project Setup:

- Create a .NET 8.0 Console Application in Visual Studio.
- Add a separate Class Library project for handling Entity Framework Core database operations.
- Install the required NuGet packages for Entity Framework Core and SQL Server.

---

2. XML Data File:

You will use XML file to store movie data that will be loaded into the system.

LargeMovieData.xml

Store a collection of movies in XML format. The structure should include properties like Id, Title, Director, ReleaseYear, Genre, and Rating.

3. Database Setup:

Create a **SQL Server database** named MovieDB. Use **Entity Framework Core** to map the movie data from the XML file into a table named Movies.

The Movies table should include the following columns:

- Id (int, Primary Key)
- Title (string, up to 100 characters)
- Director (string, up to 100 characters)
- ReleaseYear (int)
- Genre (string, up to 50 characters)
- Rating (decimal)

The Entity Framework context should be configured in a **separate class library** project.

4. LINQ Queries:

Create at 15 LINQ queries to fetch data from the **Movies** table.

1. Simple LINQ Queries (5 queries):

- Retrieve all movies directed by "Christopher Nolan".
- Retrieve all movies released in the year 2010.
- Retrieve all movies in the "Sci-Fi" genre.
- Retrieve movies with a rating greater than 8.5.
- Retrieve the top 5 highest-rated movies.

2. Medium Complexity LINQ Queries (5 queries):

- Retrieve all movies released between the years 2000 and 2010.
- Retrieve movies that are in the "Action" genre and have a rating higher than 8.0.
- Retrieve movies sorted by release year in ascending order and then by rating in descending order.
- Count the number of movies in each genre.
- Retrieve the top 3 highest-rated movies directed by "Quentin Tarantino".

3. Complex LINQ Queries (5 queries):

- Group movies by genre and calculate the average rating for each genre.
- Find movies that were released between 1990 and 1999 and have a rating higher than 8.0.
- Retrieve all movies where the title contains the word "King" and the rating is above 8.5.
- Retrieve the top 5 movies for each genre based on rating.
- Find movies directed by "Christopher Nolan" where the rating is above the average rating of all movies in the database.

Use both **Lambda expressions** and **LINQ syntax** for practice.

---

5. Events and Delegates:

In your program, create an event and delegate to notify when the data has been successfully uploaded from XML/CSV files to the SQL Server database. The event should trigger after data upload, printing a success message in the console.

Example:

- Create a delegate that represents the method signature for the event.
- Create an event to notify when the data is uploaded successfully.
- Subscribe to this event in the console application.

6. JSON Export:

After retrieving data from the database using LINQ, serialize the queried data into JSON files. Export different sets of data based on the above queries into separate JSON files.

---

7. Submission Guidelines:

1. **Code Buildability:**
    - Ensure that the project can be **successfully built** and **compiled** without any errors, **otherwise the solution will not be evaluated!**
    - Carefully review your code for common syntax errors, missing references, or incorrect package configurations before submission.
    - Make sure all necessary NuGet packages (e.g., **Entity Framework Core**, **System.Text.Json**, etc.) are correctly installed and included in the solution.
    - If you have added any new libraries or references, make sure they are **restored** correctly and are part of the solution files.
2. **Visual Studio Solution:**
    - Submit the entire **Visual Studio solution**, including:
        - The **Console Application** project.
        - The **Class Library** project containing Entity Framework Core database operations.
        - All the necessary **project files**, including .csproj files, should be included.
    - Ensure that the project includes any relevant **XML** and **CSV** files used for the data upload functionality.
3. **SQL Server Database Script:**
    - If necessary, include a **SQL Server database setup script** (.sql file) to recreate the database schema (e.g., CREATE TABLE statements for the **Movies** table).
    - If Entity Framework Core **Code-First Migrations** are used, ensure that the appropriate migration files are included, and the database can be updated automatically through the code.
4. **Event and Delegate Implementation:**
    - Ensure that events and delegates are properly implemented and triggered during the data upload process.
    - The event notifying successful data upload must be **functional** and should print a confirmation message to the console.

5. **LINQ Queries:**
   - Include all 15 required **LINQ queries** (5 simple, 5 medium, and 5 complex) in the project.
   - These queries should be thoroughly tested and executed without errors.
   - Use both **Lambda expressions** and **Query syntax** for various queries.
6. **JSON Export Functionality:**
   - Ensure that the data retrieved by LINQ queries is correctly **serialized into JSON** files.
   - Each query result must be exported into a **JSON** file, and these files must be automatically created in the project output directory when the application is run.
7. **Project Directory Structure:**
   - The solution should be well-organized, and the **XML** and **CSV** files should be placed in a dedicated folder within the project.
   - The **JSON** output files should be stored in an easily accessible folder, and file paths should be handled properly in the code.
8. **Final Review:**
   - Test the entire application to ensure that it works as expected and performs all tasks without issues.
   - Verify that **all features** are functioning (data upload, event notification, LINQ queries, JSON export).
   - Ensure that the project compiles and runs smoothly on different systems with the required .NET version installed.

By following these guidelines, your submission will ensure that the project is well-structured, functional, and easily testable.