Tournament Planner

Introduction

You are volunteering in a **local tennis club**. Luckily, your club has **exactly 32 active members**. In the upcoming week, the yearly club championship will take place. It is your job to implement the software to **organize the matches of the championship**.

In each match, two random club members (=players) play against each other. The winner reaches the next round. This will be repeated until two players reach the final. Here is a table displaying the number of matches per round:

Round	Matches	Total players
1	16	32
2	8	16
3	4	8
4	2	4
5	1	2

Data Model

Your software must store data about **players** and **matches** in a relational database. Consider the following requirements when designing your database model:

- You have to store *players* (=club members participating in the championship). For each player, you have to store the following properties:
  - Firstname (mandatory)
  - Lastname (mandatory)
  - Gender (mandatory)
- You have to store *matches*. For each match, you have to store the following properties:
  - Round number (mandatory; values between 1 and 5, see also table above)
  - References to both players participating in the match (mandatory)
  - Winner of the match (optional; will be set once the match is completed)
- Every record in every table has to have an auto-generated numeric primary key (type `int`)

Requirements for the backend

- Create the data model using **EFcore** in a separate **library**.
- Generate the database (SQL Server or SQLite) using **EFCore migrations**.
- Store the connection string in **appsettings.json**.
- **Seed the players** provided in the file **players.csv**.
- Create an **ASP.NET 5 WebAPI** containing at least one service to access the data layer.
- This service has to implement at least the following functionalities:
  - Add a club member (= player)
  - Add a match between two given club members
  - Set the winner of an existing match
  - Generate match records for the upcoming round:
    * If there are *any* matches in the DB that do *not* have a winner, throw an exception.
    * Count the number of matches to find out the next round.
      · If you find no matches in the DB, the next round is the first round.
      · If you find 16 completed matches, the next round will be the second round.
      · If you find 24 completed matches, the next round will be the third round.
      · If you find 28 completed matches, the next round will be the fourth round.
      · If you find 30 completed matches, the next round will be the fifth round (=final).
      · If there is any other number of matches in the DB, throw an exception.
    * If you generate the first round, generate matches between random players.
    * If you generate any other round, generate matches between random winners of the previous round.
  - Get a list of all matches that do not have a winner yet
  - Delete all matches; this operation will be used during testing and after the championship.
- Add **API endpoints** required to implement your frontend.
- Use **DTOs** when transfering information between frontend and backend.

Requirements for the frontend

- Implement a simple frontend using **Angular**.
- Minimal requirements:
  - Start a new tournament (includes deleting old matches and generating the matches of the first round).
  - Display the pairings (players) of the current round.
  - Mark, if a player is male or female.

- Mark the winner of a match (highlight the winner accordingly).
- If all matches have a winner, start the next round.
- Highlight the winner of the tournament accordingly.