Code Challange – Ntara

Code Project

You have been provided with a comma-separated value (.csv) file. This data set should remain unedited. Your mission, if you choose to accept it, is as follows:

- Create a single web page in asp.net C# with a textbox and a button that, when clicked, searches a given dataset and returns the entire record of results on screen
- Include a drop-down that defaults to all columns but can be selected to search only one column
- Provide error checking where appropriate
- Please use a technology framework like Angular, React, or Vue to help showcase how you'll interact with the frontend.
- You will be evaluated on functionality not UI design.

Additional Requests:

- Please use a Database (SQLite or In-Memory) is sufficient
- Please use Entity Framework to serve the data
- This is to showcase your code, organization, and standards please showcase them to the team
- Your project should work on any developer's machine when turned in as long as they have Visual Studio and run the project as a web application. If it doesn't work you won't pass. It is that important.
- Provide a write-up about your project. What was your approach? Provide things that you might change about the project if given the opportunity. Observations you made to improve the page. Length of time spent coding the application, not documentation.
- Focus on how a user would use the application
- If you've been asked to do something you haven't mastered yet. It is perfectly fine to explain that in your documentation. We love to learn and grow at our company. So, if you are learning something new is not a negative. Just let us know in the write-up.

User Story for the project:

As a sports announcer, I would like to search for football teams and see their statistical records during a game day announcement.

Candidate: Edgar Vega
Estimated Delivery Date: Fri, Oct 31, 2025
Repository: https://github.com/EVegaGT/FootballTeamWinsWithMascots.

Video: Example of execution

Architecture

The goal of this project was to develop a clean, scalable, and maintainable full-stack application capable of searching and displaying football team statistics.

Backend

We created a new ASP.NET Core 8 project to build a RESTful API, the implementation follows Domain-Driven Design (DDD) and CQRS (Command Query Responsibility Segregation) principles to achieve a strong separation of concerns and ensure that the system can evolve and scale easily, the project architecture leverages MediatR for command and query handling.

1. **FootballTeamWinsWithMascots.Api**
    - Contains the API controllers, dependency injection configurations, and logging setup.
    - Create endpoints for HTTP requests.
    - Uses MediatR to delegate requests to their corresponding handlers in the Application layer.
2. **FootballTeamWinsWithMascots.Application**
    - Implements the CQRS pattern by separating commands (write operations) from queries (read operations).
    - Uses MediatR to define request/response handlers.
    - For this challenge, only the SearchTeamsQueryHandler was required, which filters teams dynamically based on user input.
3. **FootballTeamWinsWithMascots.Domain**
    - Defines the core business model, including entities, interfaces, and domain logic.
    - This layer is pure C#, completely independent from frameworks or external dependencies.
    - Contains interfaces for repositories and business rules.
4. **FootballTeamWinsWithMascots.Infrastructure**
    - Implements Entity Framework Core (EF Core) for data access.
    - Contains DbContext, repositories, and SQLite configuration.

- Handles migrations and seeding of data from a CSV file using CsvHelper.

Frontend

The frontend is developed as an independent React + TypeScript application. It interacts with the backend API through HTTP endpoints, ensuring a clean separation between presentation and business logic.

- Uses React to build a responsive and dynamic single-page application.
- Integrates Material UI (MUI) to provide a clean, user-friendly, and accessible interface.
- Uses swagger-typescript-api to automatically generate the client SDK from the OpenAPI (Swagger)
- API and frontend communicate through HTTP endpoints.

Development Methodology

The development process followed a layered approach to ensure clean boundaries between concerns:

1. Domain first: define entities and repository interfaces.
2. Infrastructure: implement persistence and seeding with EF Core and CsvHelper.
3. Application: implement CQRS with MediatR, focusing on query logic and scalability.
4. API: expose endpoints through minimal controllers that delegate logic to handlers.
5. Frontend: consume the API, generate client code automatically, and design a minimal, intuitive UI.

Times

Task	Time
Create Project	1 hour
Create structure and architecture	2 hours
Configure SQLite and Entity Framework Core	1 hour
Seed database with CSV file using CsvHelper	2 hours
Implement MediatR and configure CQRS pattern	3 hours
Create team search query and handler	1 hour
Add OpenAPI generator and TypeScript client	1 hour
Build React search form	2 hours
Build React result table	1 hour
UI design and layout refinement	1 hour
Configure Visual Studio profiles	1 hour
Total (Coding Time)	15 hours