



# EPL Squad Finder



## EPL Squad Finder – Overview

### 🎯 Project Goal

This project is a **proof-of-concept (POC)** web application that allows users to search for any **English Premier League team** from the **2024/25 season** and retrieve **detailed squad information**.

### 🚀 Features:

- Search for a team by **name or nickname** (e.g., "Manchester United" or "The Hammers" for West Ham).
- Display squad details, including **profile picture, first name, surname, date of birth, and playing position**.
- User-friendly interface with a **simple UI** for easy team and player lookup.
- Intelligent **data retrieval and matching** from third-party APIs.
- Full deployment with **CI/CD using GitHub Actions & Azure Container Apps**.
- Bonus feature: Users can search teams by nickname and add/remove nicknames dynamically.



## Tech Stack

Category	Technologies
Backend	.NET 9, Entity Framework, Mediator, Refit, Polly, FuzzySharp
Testing	NUnit, AutoFixture, Moq, Shouldly
Frontend	React (Vite), Styled Components
Database	Microsoft SQL Server
DevOps	Docker, Docker Compose, GitHub Actions, Azure Container Apps

## ⚠ Challenges Faced

### 🔗 Third-Party API Limitations

The application integrates with two APIs:

#### 1 API-Football ([dashboard.api-football.com](https://dashboard.api-football.com))

- Free, but has **strict rate limits** (per minute/day).
- Returns **HTTP 200 even for errors** (error messages are inside the response body).

A screenshot of a Postman API response. The response tab is selected, showing a status of 200 OK with a duration of 137ms and a size of 198B. The response body is a JSON object with line numbers. A red box highlights the 'errors' field, which contains a 'rateLimit' message: "Too many requests. Your rate limit is 10 requests per minute."

```
1  {
2    "get": "players/squads",
3    "parameters": {
4      "team": "46"
5    },
6    "errors": {
7      "rateLimit": "Too many requests. Your rate limit is 10 requests per minute."
8    },
9    "results": 0,
10   "paging": {
11     "current": 1,
12     "total": 1
13   },
14   "response": []
15 }
```

- No access to full squad details for the 2024-2025 season in the **free plan**.
- Lacks full squad details.

### 2 Football-Data API ([www.football-data.org](http://www.football-data.org))

- **Lacks full squad details**, making it insufficient on its own.

### 3 SportMonks API ([www.sportmonks.com](http://www.sportmonks.com))

- Considered as an alternative, but **Premier League data is only available in paid plans**.

## ✓ Solution:

- ◆ Developed **custom matching logic** to **merge and match teams/players across APIs**.
- ◆ Implemented **retry policies** using **Polly** to handle failed API requests.

## Data Matching Across APIs

- ◆ **Issue:** Different APIs provide **slightly different team and player names**, making direct matching difficult.
- ◆ **Solution:** Used **FuzzySharp** to compare names based on **similarity scores** to find the best matches.

## UI Overview

The application consists of **two main pages**:

### League Page

- 📌 Displays **all Premier League teams** with **logos, names, and nicknames**.
- 📌 Includes a **search bar** to filter teams.
- 📌 Clicking on a team navigates to its **squad details page**.

Premier League

Season: 2024 (16.08.2024 ~ 25.05.2025)

Search teams...

 Arsenal FC The Gunners	<a href="#">Visit Website</a>	<a href="#">Show Squad</a>
 Aston Villa FC The Villa	<a href="#">Visit Website</a>	<a href="#">Show Squad</a>
 Chelsea FC The Blues	<a href="#">Visit Website</a>	<a href="#">Show Squad</a>
 Everton FC The Toffees	<a href="#">Visit Website</a>	<a href="#">Show Squad</a>
 Fulham FC The Cottagers	<a href="#">Visit Website</a>	<a href="#">Show Squad</a>
 Liverpool FC The Reds	<a href="#">Visit Website</a>	<a href="#">Show Squad</a>

## 👤 Team Squad Page

- 📌 Displays **players in a team** with **photos, names, birthdates, and positions**.
- 📌 Allows adding **custom tags (nicknames)** to teams.
- 📌 Includes a **search feature** for filtering players.

Arsenal FC

Website: <http://www.arsenal.com/>

The Gunners

Search players...

 David Raya Goalkeeper   Number: 22 Date of Birth: 15.09.1995 (29y)
 Neto Neto Goalkeeper   Number: 32 Date of Birth: 19.07.1989 (35y)
 Tommy Setford Goalkeeper   Number: 36 Date of Birth: 13.03.2006 (18y)
 Aleksei Fedorushchenko

# Database Design

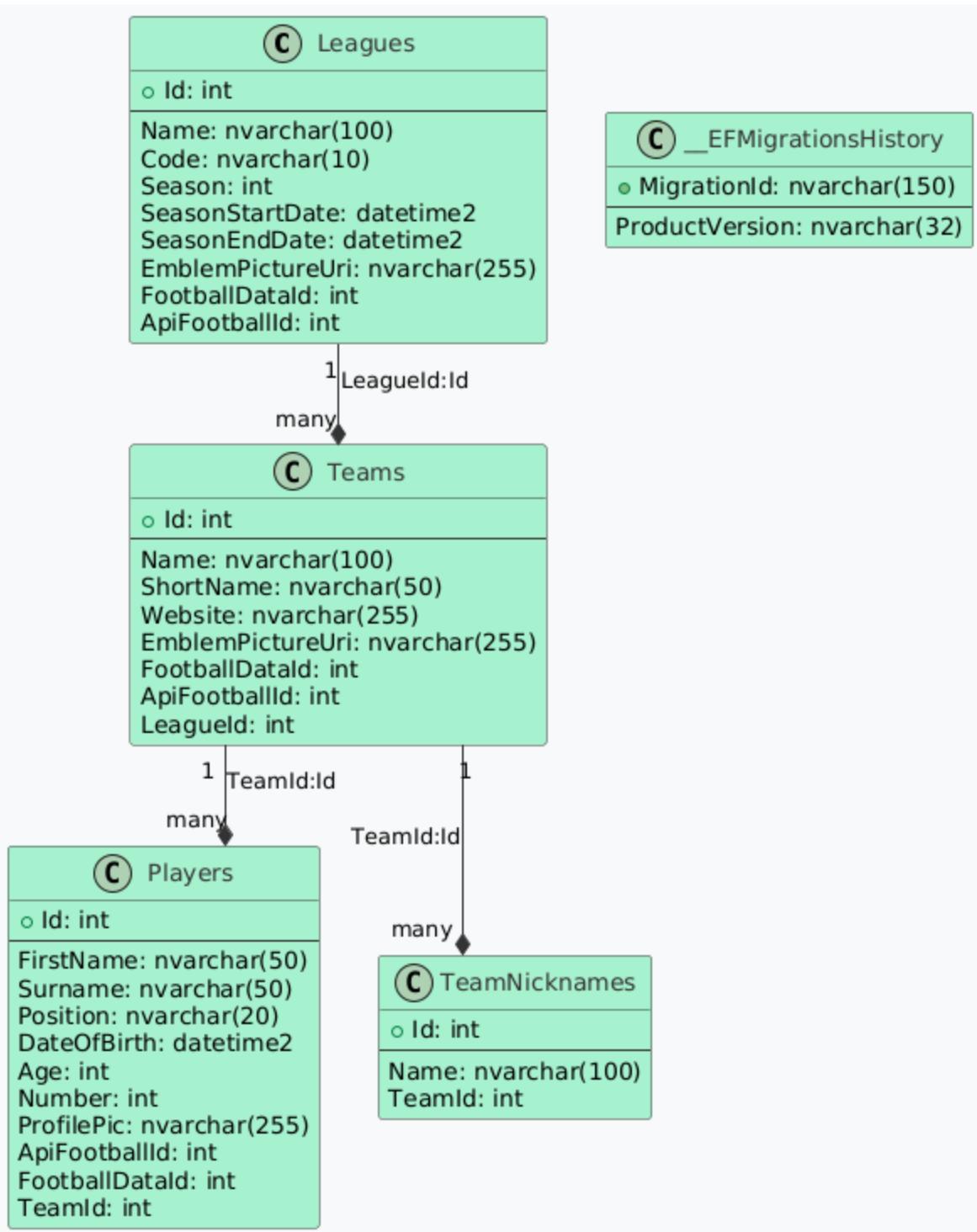
## Database Schema

The application uses **Microsoft SQL Server** as the database. Below is the **diagram** for the database structure:

To view and edit the diagrams, you can use **PlantUML**. The `.pu` files located in the `docs/diagrams/` folder can be opened with:

- The **PlantUML Online Editor**:  
[www.plantuml.com/plantuml/uml](http://www.plantuml.com/plantuml/uml)
- A **local installation** of PlantUML with a compatible IDE (e.g., **IntelliJ IDEA**, **Visual Studio Code** with the **PlantUML extension**).
- The **VSC extension**: [PlantUML for VS Code](#).

To generate the database diagrams, copy the contents of the `.pu` files into the **PlantUML editor** or use a local **PlantUML-compatible IDE** such as **VS Code** or **IntelliJ IDEA** with the appropriate extensions.



## ◆ Key Entities:

- **League** 🏆 – Stores League information.
- **Team** ⚽ – Stores football teams with their details.

- **Player**  – Stores squad details for each team.
  - **TeamNickname**  – Stores alternative names (nicknames) for teams.
- 

## Project Architecture

### Backend Architecture

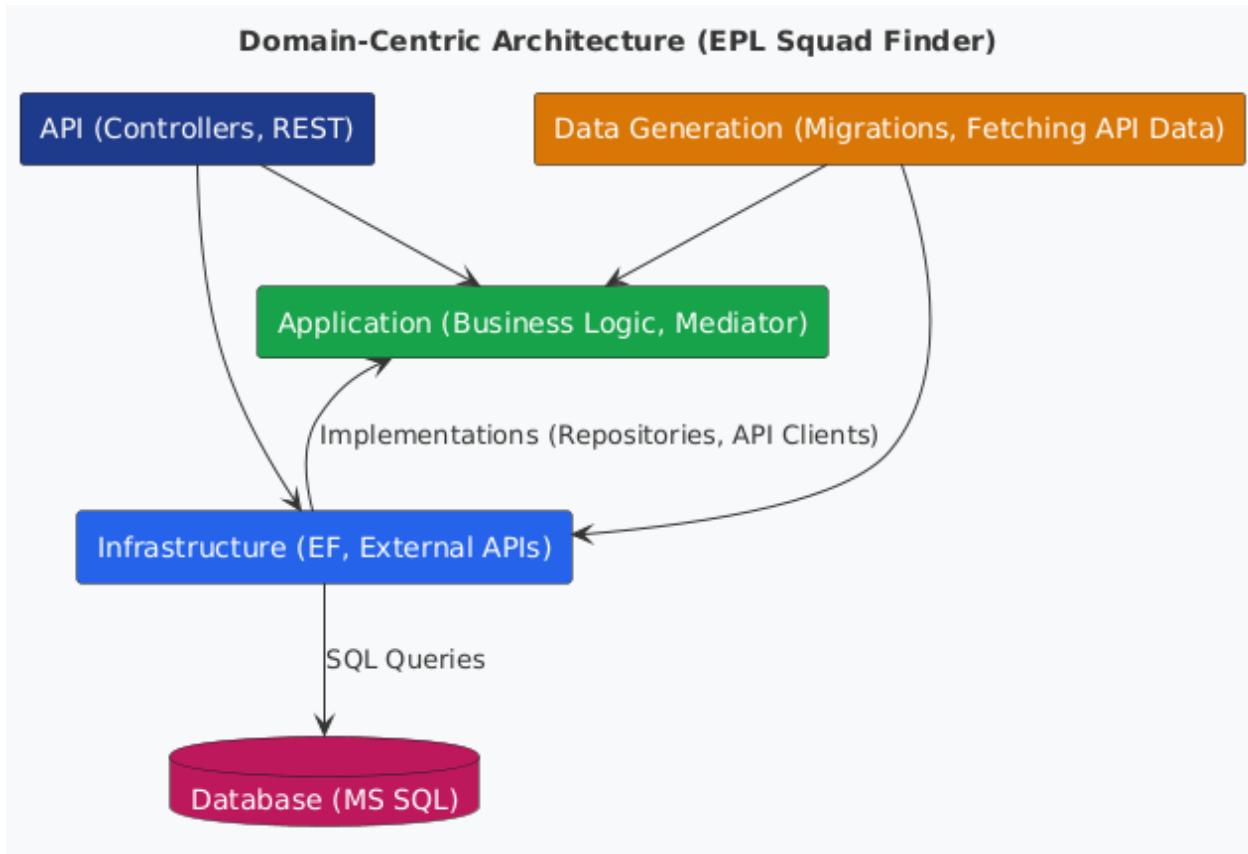
The backend follows a **Domain-Centered architecture**, inspired by **Ports & Adapters (Hexagonal Architecture)**.

#### ◆ Architectural Layers:

Layer	Responsibilities
API Layer 	Handles HTTP requests and responses. Exposes RESTful endpoints.
Application Layer 	Implements <b>business logic</b> . It is <b>self-contained</b> and does not depend on infrastructure.
Infrastructure Layer 	Manages <b>database interactions</b> and <b>external API calls</b> .
Data Generation 	A separate <b>console application</b> that applies database migrations and populates team/player data from external APIs.

### Architectural Diagram

The application follows a **Domain-Centered Architecture**, ensuring that **business logic remains independent** from API and infrastructure concerns.



### 💡 Why this approach?

- **Business logic is fully isolated** from infrastructure dependencies.
- **Easy to swap out** APIs, databases, or UI layers without breaking core logic.
- **Improved testability** – the **Application Layer** can be tested **independently** of the database and API services.



## Data Fetching & Matching Logic

Fetching and matching **teams and players** across two different APIs was one of the most complex challenges in this project. The APIs used have **different structures and limitations**, requiring a **custom matching algorithm**.

### ◆ Data Fetching Workflow

The process follows these **steps**:

## 1 Fetch Premier League team data from Football-Data API

(`/competitions/{leagueCode}/teams?season={season}`).

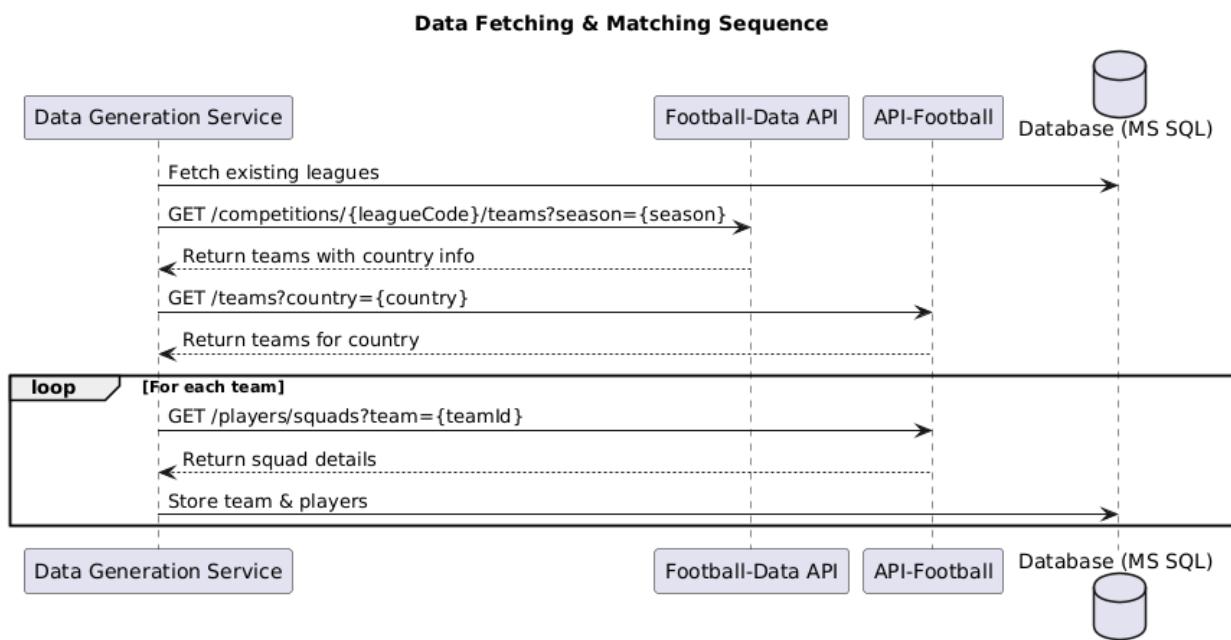
## 2 Fetch corresponding team details from API-Football (`/teams?country={country}`) using fuzzy matching.

## 3 Retrieve squad details for matched teams from API-Football (`/players/squads?team={teamId}`).

## 4 Store teams and players in the database, ensuring that only matches with a similarity score above the defined threshold are saved.

## Data Fetching & Matching Flowchart

Here's a **flowchart** outlining the data retrieval process:



## 🔍 Matching Algorithm

Since **Football-Data API** and **API-Football** return slightly different team and player names, **direct matches are not always possible**. To solve this, we implemented a **custom matching algorithm**:

## ◆ Team Matching Strategy

Each team from **Football-Data API** is compared against all teams from **API-Football** using a **scoring system**:

Criteria	Weight (%)	Matching Method
<b>Exact Name Match</b>	<code>match_ratio *60%</code>	Fuzzy string comparison
<b>Short Name Match</b>	<code>match_ratio *40%</code>	Fuzzy string comparison
<b>Three-Letter Code (TLA) Match</b>	+30 points	Exact match
<b>Stadium Name Match</b>	<code>match_ratio *25%</code>	Fuzzy match (if available)
<b>Youth Team Detection</b>	-50 points	Penalize youth teams

◆ Where `match_ratio` is a value in range (0-100), calculated using fuzzy string comparison methods (`Fuzz.PartialRatio`).

**Threshold:** A match is only accepted if the score is **above 90% for teams** and **above 65% for players**.

## ◆ Player Matching Strategy

Players are matched based on **name similarity** using **FuzzySharp**:

- 1 **Normalize player names** (remove dots, lowercase, trim spaces).
- 2 **Compute similarity score** between names from both APIs.
- 3 Accept matches with a score above 65% (this threshold can be fine-tuned based on data quality 😊).
- 4 **Fallback:** If no match is found, add the player **only from API-Football**.

## API Endpoints Used

API	Endpoint	Purpose
<b>Football-Data API</b>	<code>/competitions/{leagueCode}/teams?season={season}</code>	Get all teams for a given season

<b>API-Football</b>	<code>/teams?country={country}</code>	Get all teams for a country
<b>API-Football</b>	<code>/players/squads?team={teamId}</code>	Get squad details for a team

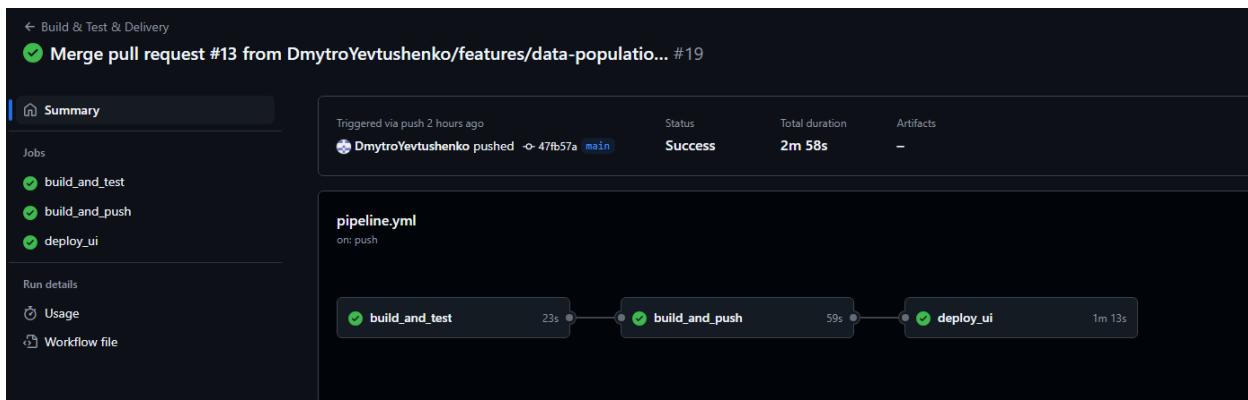
This approach ensures that **data from both APIs is combined efficiently**, resulting in **high-quality team and player information**.

## 🚀 Deployment Process

The project is deployed using **Azure Container Apps** with **CI/CD via GitHub Actions**.

### ◆ Deployment Workflow

- 1 Push to the main branch on GitHub → Triggers GitHub Actions



- 2 Run project **build** and run **unit tests**

- 3 Build Docker images for API & UI and push to **Azure Container Registry**

The screenshot shows the Azure Container Registry (ACR) interface for the 'eplquadfinder' registry. The left sidebar includes 'Services' (Repositories selected), 'Policies', and 'Content trust'. A search bar at the top contains 'contain'. A message box says 'New to ACR, Artifact streaming helps pull images faster from ...'. Below the message is a search bar with 'Search to filter repositories ...'. The main area lists two repositories: 'epl-squad-finder-api' and 'epl-squad-finder-ui'.

## 4 Deploy API & UI to Azure Container Apps

The screenshot shows the Azure Container App activity log for the 'epl-squad-finder-ui' app. The left sidebar includes 'Overview', 'Activity log' (selected), 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Containers', 'Scale', 'Volumes', 'Settings', 'Monitoring', and 'Automation'. The main area displays a table of activity logs with columns: Operation name, Status, Time, Time stamp, Subscription, and Event initiated by. All entries show 'Accepted' status, '2 hours ago' time, 'Sun Feb 02 ...' timestamp, 'Azure subscription 1' subscription, and 'github-actions' as the event initiator.

## Frontend Deployment

The UI has been successfully deployed to **Azure Container Apps**, and a **publicly accessible URL** is available:

### EPL Squad Finder UI - Azure Hosted

#### Current Status:

- The **frontend is fully functional**, but it is currently pointing to a **local API instance** for testing purposes.

- The next step is to deploy the backend API to Azure and **update the frontend configuration** to use the deployed API.
- 

## Running Locally

To test the application **locally**, follow these steps:

### ◆ 1. Prerequisites

Before running the application, make sure you have installed:

-  [Docker](#) (Recommended)
  -  [.NET 9 SDK](#)
  -  [Node.js 18+](#)
- 

### ◆ 2. Clone the Repository

```
git clone https://github.com/DmytroYevtushenko/epl-squad-finder.git  
cd epl-squad-finder
```

### ◆ 3. Run with Docker (Recommended)

Navigate to the `docker` directory and run the following command:

```
cd docker  
docker compose up --build
```

 This will automatically start:

- Microsoft SQL Server** (database)
- Console App** (fetches API data & applies migrations)
- Backend API** (<http://localhost:5290>)
- Frontend UI** (<http://localhost:3000>)

Note:

The **console app requires API keys** to be set in `compose.yaml`.

If you prefer not to register for API keys, you can manually **seed the database** with:

```
docker-compose --profile manual run db-seed
```

This will execute the `seed.sql` script inside the database container.

## ◆ 4. Run Manually (Without Docker)

### 1 Setup Database

- Install **SQL Server**
- Update **connection strings** in:
  - `epl-squad-finder-api/EplSquadFinder.DataGeneration/appsettings.json`
  - `epl-squad-finder-api/EplSquadFinder.Api/appsettings.json`

### 2 Run API

```
cd epl-squad-finder-api/EplSquadFinder.Api  
dotnet build  
dotnet run
```

### 3 Generate Data (Fetch Teams & Players from APIs)

```
cd epl-squad-finder-api/EplSquadFinder.DataGeneration  
dotnet build  
dotnet run
```

### 4 Run Frontend

```
cd epl-squad-finder-ui  
npm install  
npm run dev
```

## ◆ 5. Open the UI

Once everything is running, open a browser and go to:



<http://localhost:3000>

- **Frontend** → <http://localhost:3000>
- **API** → <http://localhost:5290>