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

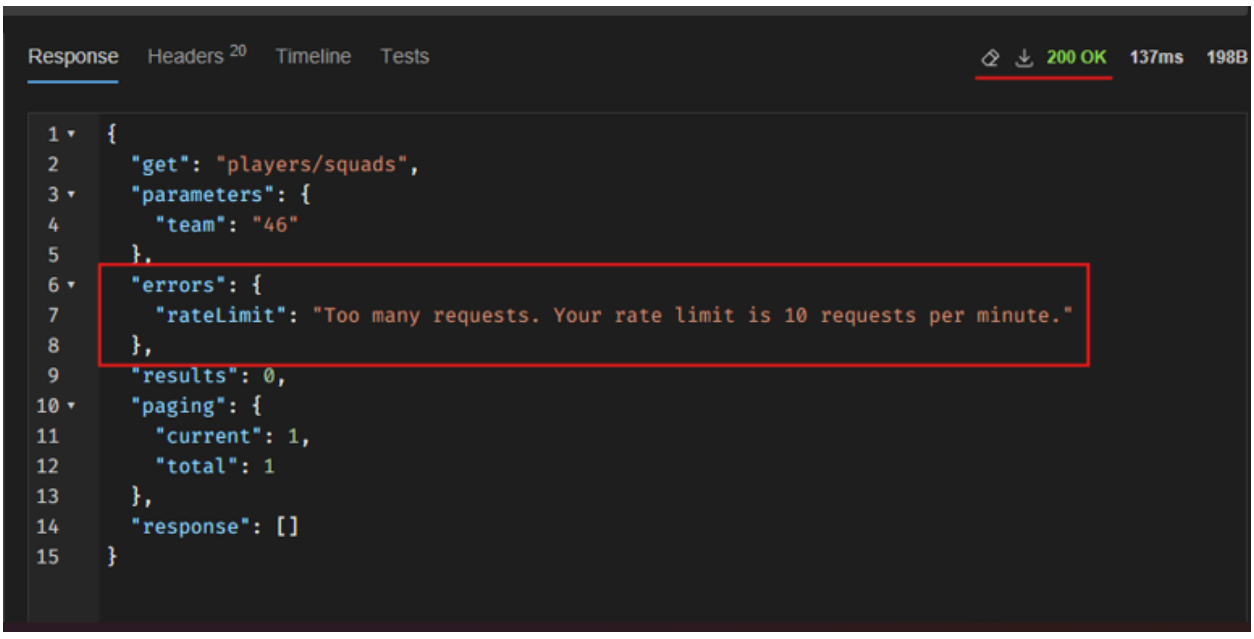
⚠ Challenges Faced

🔗 Third-Party API Limitations

The application integrates with two APIs:

1️⃣ **API-Football** (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).



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

2️⃣ **Football-Data API** (www.football-data.org)

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

3️⃣ **SportMonks API** (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

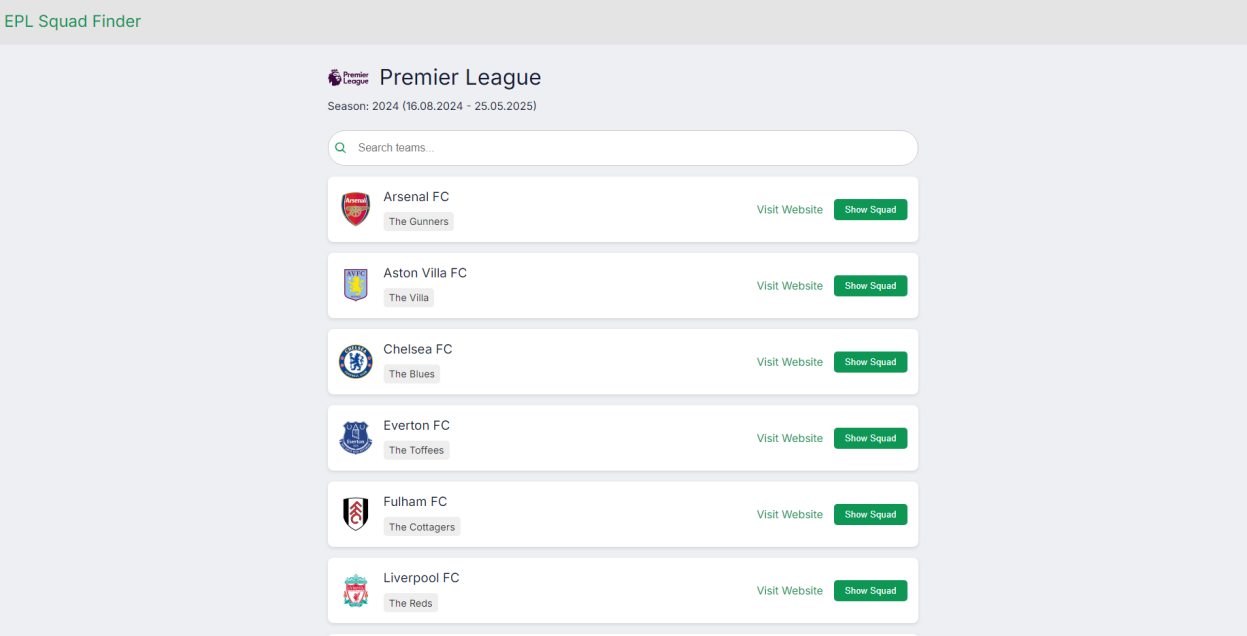
The application consists of **two main pages**:

⚽ 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**.

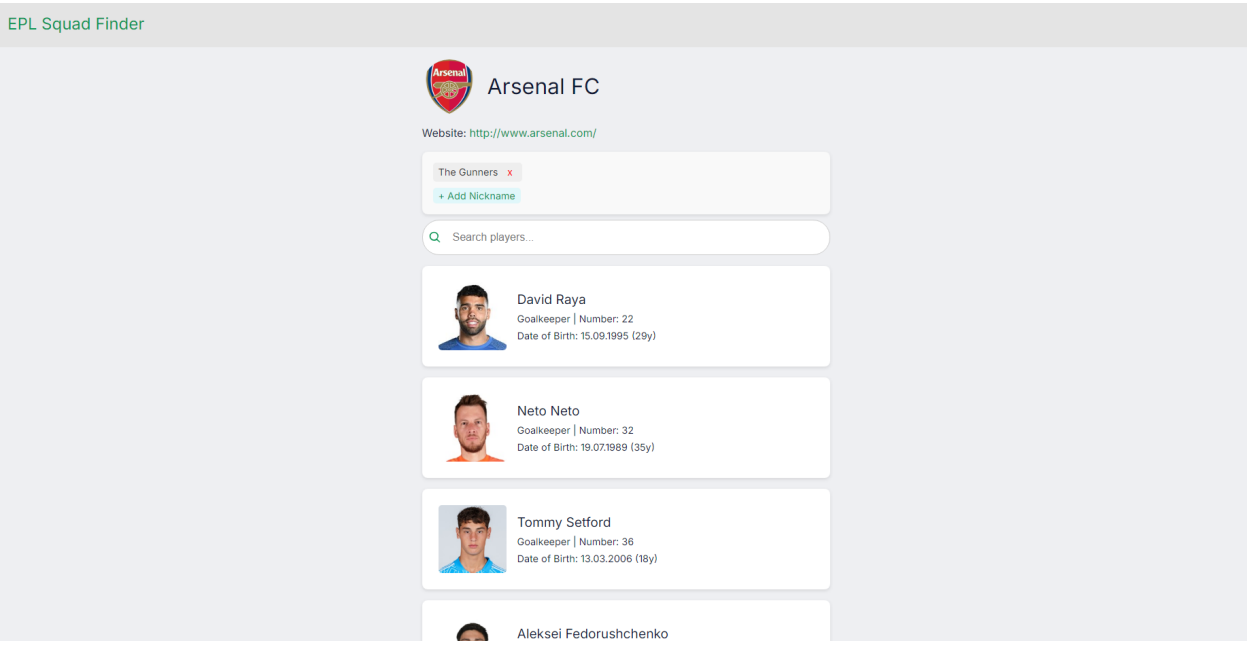


👥 Team Squad Page

📌 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.

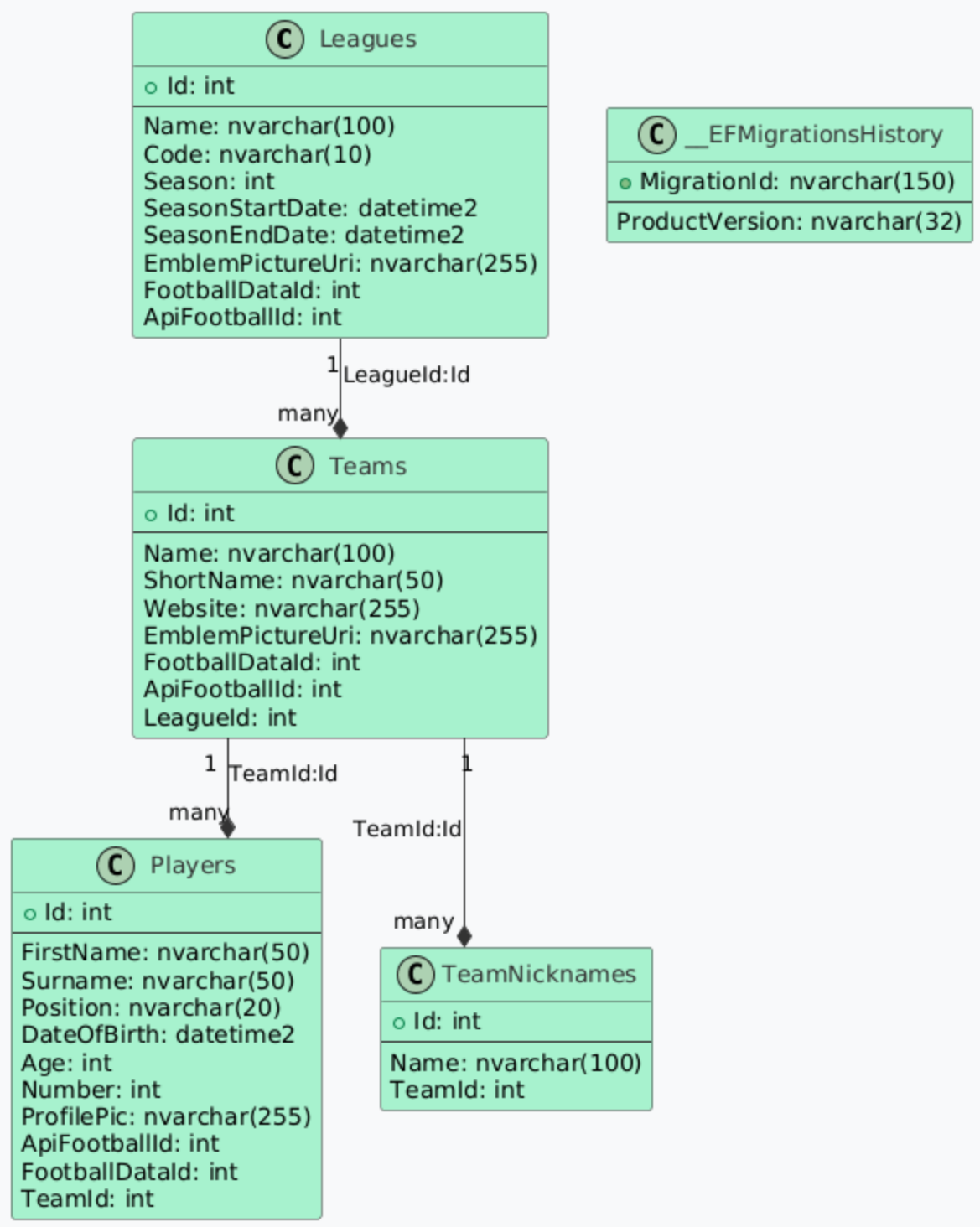


📊 Database Design

🛢️ 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
> - 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

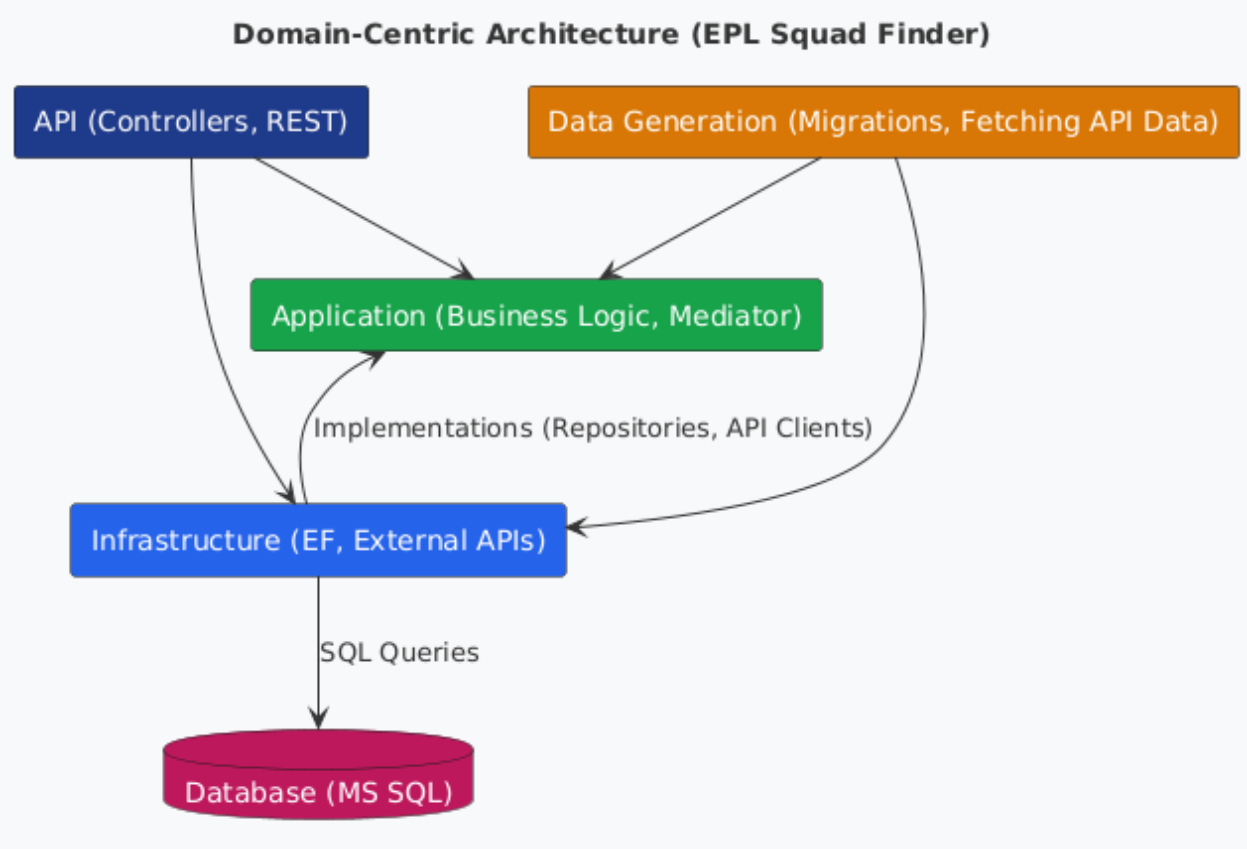
The backend follows a **Domain-Centered architecture**, inspired by **Ports & Adapters (Hexagonal Architecture)**.

🔷 Architectural Layers:

Layer	Responsibilities
API Layer 🎯	Handles HTTP requests and responses. Exposes RESTful endpoints.
Application Layer 🏗️	Implements **business logic**. It is **self-contained** and does not depend on infrastructure.
Infrastructure Layer 🏛️	Manages **database interactions** and **external API calls**.
Data Generation 📩	A separate **console application** 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**:

1️⃣ **Fetch Premier League team data** from **Football-Data API** (`/competitions/{leagueCode}/teams?season={season}`).

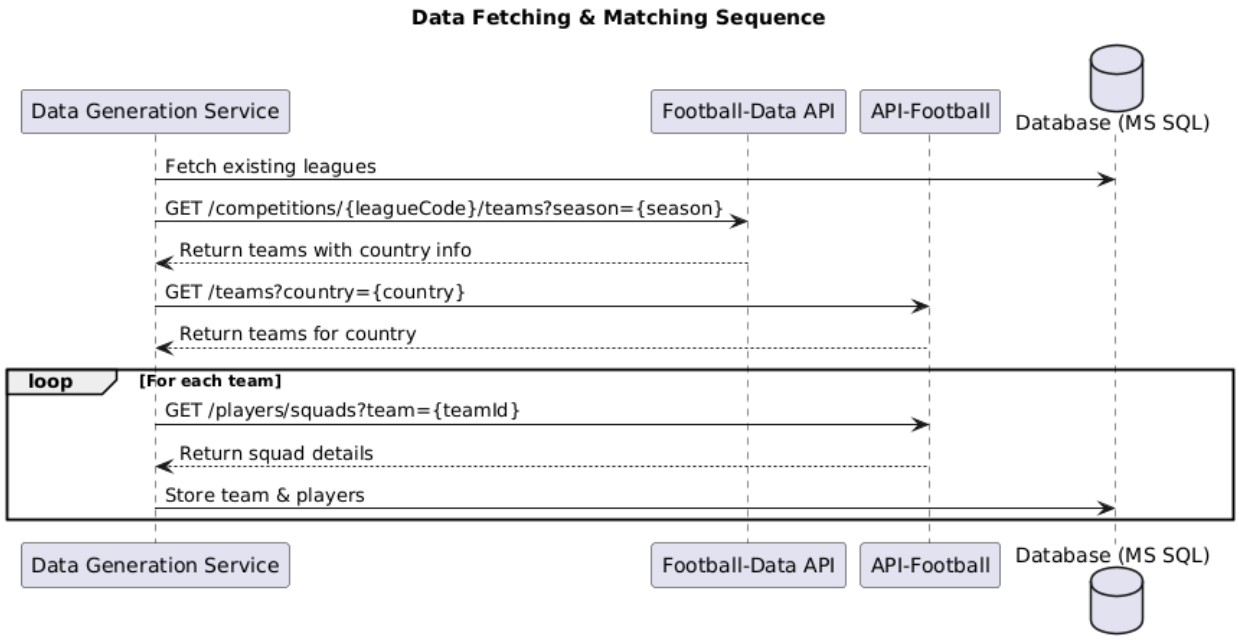
2️⃣ **Fetch corresponding team details** from **API-Football** (`/teams?country={country}`) using **fuzzy matching**.

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

4️⃣ 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
Exact Name Match	60%	Fuzzy string comparison
Short Name Match	40%	Fuzzy string comparison
Three-Letter Code (TLA) Match	+30 points	Exact match
Stadium Name Match	25%	Fuzzy match (if available)
Youth Team Detection	-50 points	Penalize youth teams

> **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**:

1️⃣ **Normalize player names** (remove dots, lowercase, trim spaces).

2️⃣ **Compute similarity score** between names from both APIs.

3️⃣ Accept matches with a score above 65% (this threshold can be fine-tuned based on data quality 🙂**)**.

4️⃣ **Fallback:** If no match is found, add the player **only from API-Football**.

📡 API Endpoints Used

API	Endpoint	Purpose
Football-Data API	`/competitions/{leagueCode}/teams?season={season}`	Get all teams for a given season
API-Football	`/teams?country={country}`	Get all teams for a country
API-Football	`/players/squads?team={teamId}`	Get squad details for a team

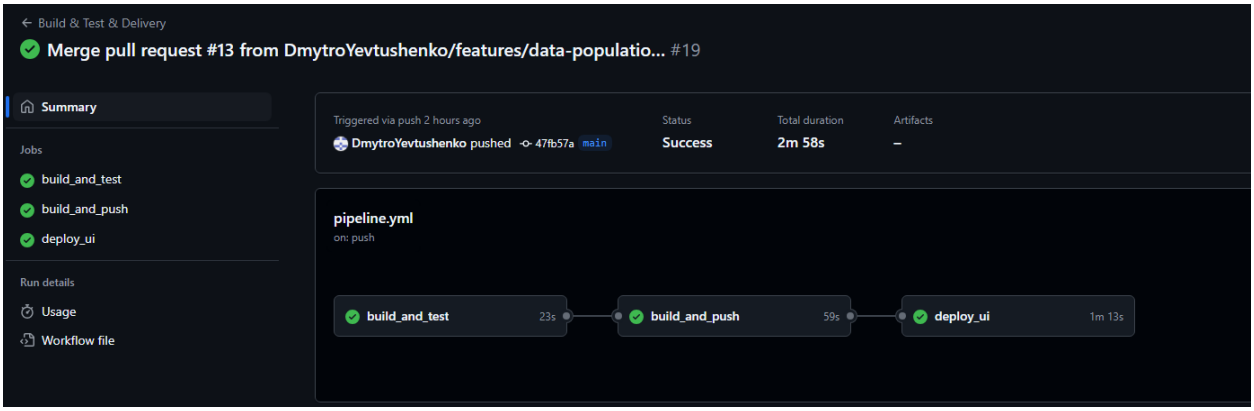
This approach ensures that **data from both APIs is combined efficiently**, resulting in **high-quality team and player information**.

🚀 Deployment Process

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

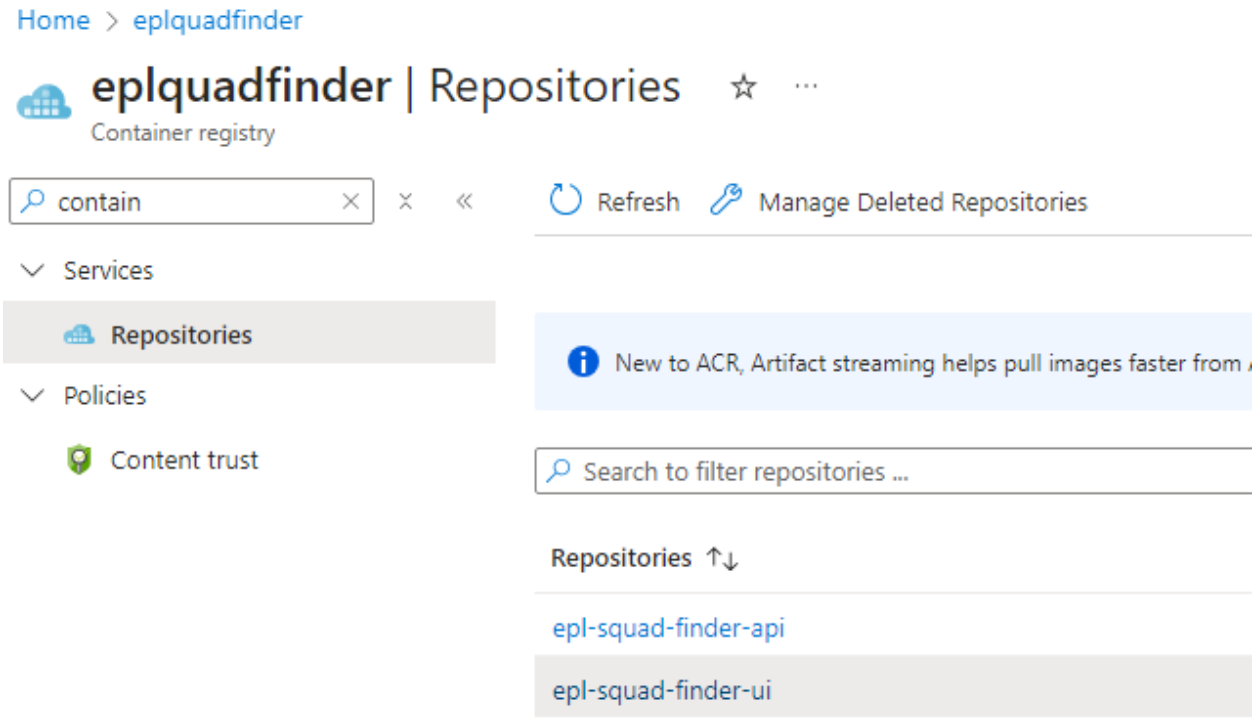
🔷 Deployment Workflow

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

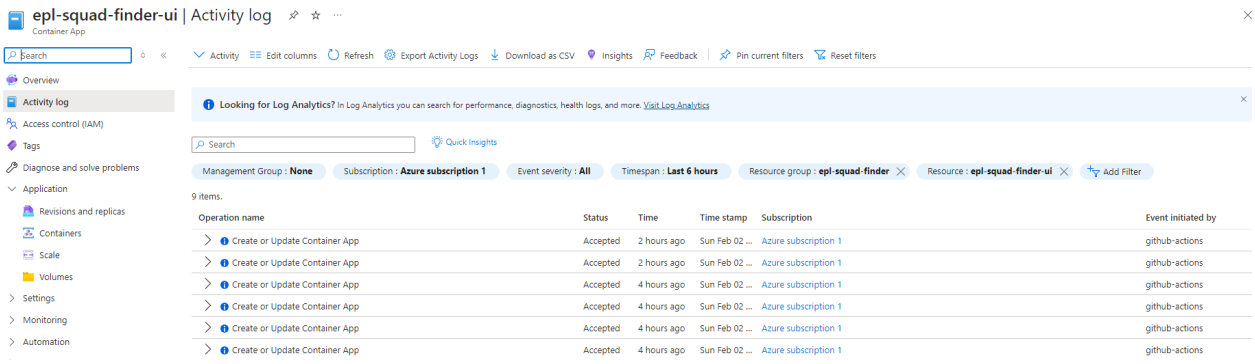


2️⃣ Run project **build** and run **unit tests**

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



4️⃣ Deploy API & UI to Azure Container Apps



🌐 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)

```
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)

1️⃣ 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`

2️⃣ Run API

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

3️⃣ Generate Data (Fetch Teams & Players from APIs)

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

4️⃣ 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`