WS - Project 2
Football Data Web Service

Alexandre Ribeiro 108122
Guilherme Amorim 107162
Paulo Macedo 102620

2025

1 Introduction

Football is the most watched sport in the world, with millions of players, fans, and clubs spread across continents [1]. This sport generates a huge amount of data, from player statistics to club performance. Analyzing and visualizing these data can provide valuable insights into team interactions, player relationships, and overall trends in the football system.

For that reason, on the first project, available Kaggle football datasets were used to construct a semantic network reflecting relations among football clubs and players with stats and other interesting entities involved, such as football leagues and nations. The process involved structuring data into an interactive network, in which users can query relations in the football world through a web application.

On this project, the focus shifted to the creation of an ontology, designed to describe football entities and then, integrate it with the existing data. In addition, a set of inference rules was developed to support the generation of new relationships among entities, allowing for the identification of implicit connections.

Furthermore, to enrich the dataset, the system was also connected to SPARQL endpoints from Wikidata, retrieving complementary information that was not included on the original datasets.

2 Ontology Definition

The ontology developed for this project models the football domain, including players, clubs, countries, leagues, and statistics. It was defined using RDFS (RDF Schema) and OWL (Web Ontology Language), leveraging the expressive capabilities of both languages to create a robust semantic model.

2.1 Challenges

In the first project, multiple datasets were used to build one large dataset, capturing various entities and the relations between them (see Figure 1 below).

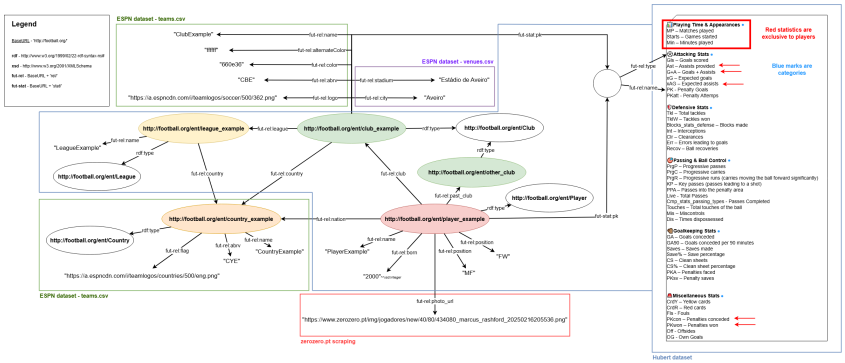

Figure 1: Semantic Network Model - Project 1

This was revealed to be a significant challenge in this second project because of the restructuring that had to be made to separate the semantic context from the data and move it to the ontology. In a major way, the task was successful and we feel that it accomplishes the requirements of the project.

2.2 General Ontology Structure

The ontology is organized around five main classes (see Figure 2):

- `Player` - Represents football players
- `Club` - Represents football clubs or teams
- `Country` - Represents countries that players can represent
- `League` - Represents football leagues or competitions
- `StatisticType` - Represents categories of football statistics

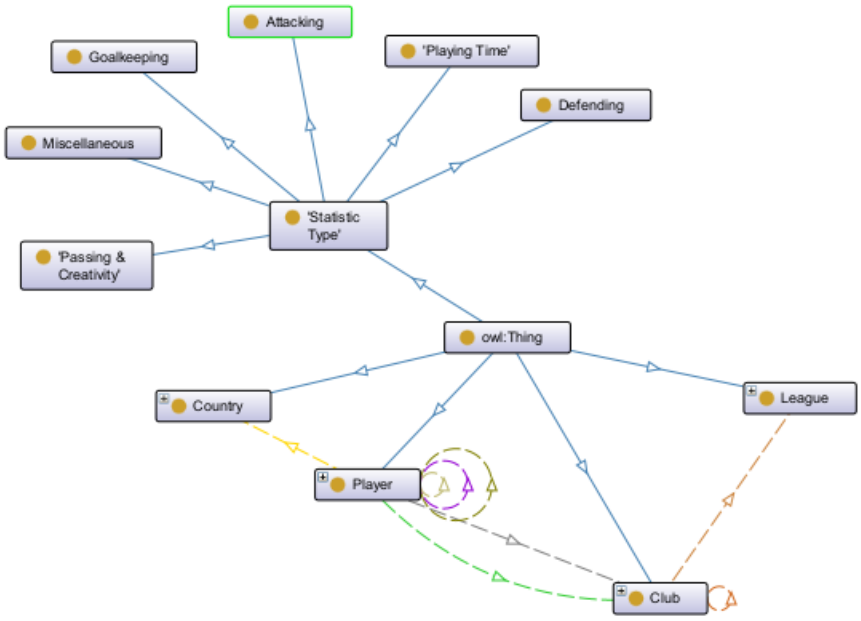

Figure 2: Class Structure (taken from Protégé)

2.3 Class Hierarchy

The ontology uses a well-defined class hierarchy, especially for statistics:

```
StatisticType
 - PlayingTime
 - Attacking
 - Defending
 - Passing (Passing & Creativity)
 - Goalkeeping
 - Miscellaneous
```

Each subclass of `StatisticType` groups related statistical properties, allowing for clear semantic organization of the data.

2.4 Object Properties

Object properties define the relationships between entities (see Figure 3):

Basic Relations:

- `club` - Relates a player to their current club (Functional)
- `nation` - Relates a player to their country (Functional)
- `past_club` - Relates a player to their previous clubs
- `country` - Relates clubs/leagues to their countries (Functional)
- `league` - Relates a club to their league (Functional)

Inferred Relations:

- `teammate` - Players from the same club (Symmetric)
- `compatriot` - Players from the same country (Symmetric)
- `past_teammate` - Players who played for the same club (Symmetric)
- `cityRival` - Rival clubs from the same city and league (Symmetric)

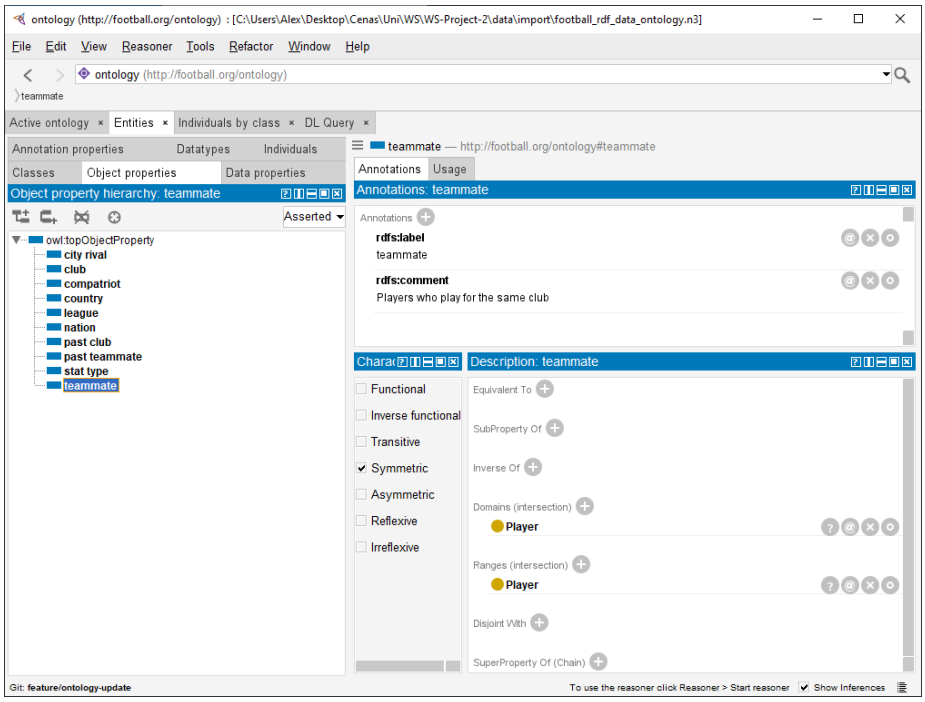

Figure 3: Object Properties (taken from Protégé)

2.5 Datatype Properties

The ontology defines datatype properties for different types of information. Here are some examples (see Figure 4):

Basic Information:

- `name` - Name of the entity (string)
- `position` - Player's position (string)
- `born` - Birth year (integer)
- `photo_url` - Photo URL (string)

Club Properties:

- `stadium` - Club's stadium (string)
- `city` - Club's city (string)
- `logo` - Logo URL (string)
- `color`, `alternateColor` - Club colors (string)

Performance Statistics: Statistical properties are categorized by type:

- **Playing Time:** `mp` (matches played), `starts` (games started), `min` (minutes)
- **Attacking:** `gls` (goals), `ast` (assists), `xg` (expected goals)
- **Defending:** `tkl` (tackles), `int` (interceptions), `clr` (clearances)
- **Passing:** `prgp` (progressive passes), `kp` (key passes)
- **Goalkeeping:** `saves` (saves), `cs` (clean sheets)
- **Miscellaneous:** `crdy` (yellow cards), `crdr` (red cards)

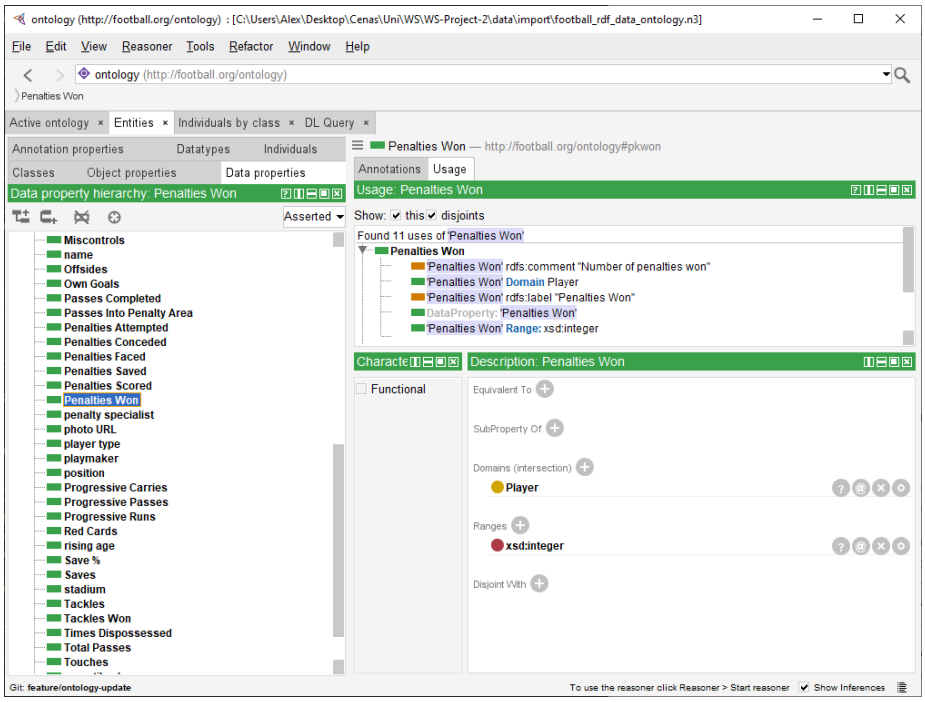

Figure 4: Data Properties (taken from Protégé)

2.6 OWL Features Used

Functional Properties: Several properties are defined as functional (a player has only one current club, one country, etc.):

```
:club a owl:ObjectProperty, owl:FunctionalProperty
:nation a owl:ObjectProperty, owl:FunctionalProperty
```

Symmetric Properties: Bidirectional relations are modeled as symmetric:

```
:teammate a owl:ObjectProperty, owl:SymmetricProperty
:compatriot a owl:ObjectProperty, owl:SymmetricProperty
```

Class Unions: Some properties apply to multiple classes:

```
rdfs:domain [ owl:unionOf ( :Player :Club ) ]
```

Equivalent Properties: For compatibility with different namespaces. This was the solution we used to map the stats from the data to the stats described in the ontology, in order to give them labels and categorize them:

```
stat:mp owl:equivalentProperty :mp
```

2.7 Inferred Properties

The ontology includes properties that can be inferred through rules (see section 3):

- `efficiency` - Efficiency based on goals+assists per 90 minutes
- `veteranStatus` - Veteran status (35+ years old)
- `youngProspect` - Young prospect (under 23 years old)
- `penaltySpecialist` - Penalty specialist (90%+ success rate)
- `playmaker` - Playmaker (many assists and key passes)
- `goalThreat` - Significant goal threat
- `keyPlayer` - Key player (many minutes played)
- etc.

3 SPIN Inferences

SPIN (SPARQL Inferencing Notation) rules are used to automatically derive new knowledge from existing data in the football ontology. These rules enable the system to infer relationships and classifications that are not explicitly stated in the original data, enriching the knowledge base with meaningful insights.

Note: Contrary to what was said in the presentation, the 'teammate' and 'compatriot' relations are not automatically inferred from the ontology, but are instead generated by the SPIN rules described below.

3.1 Player Relationship Rules

3.1.1 Teammate Identification

This rule identifies players who are teammates by checking if they play for the same club (see Figure 30):

```
PREFIX fut-rel: <http://football.org/rel/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player1 ont:teammate ?player2 .
}
WHERE {
    ?player1 a ?class .
    ?player2 a ?class .
    ?class rdfs:subClassOf* ont:Player .
    ?player1 fut-rel:club ?club .
    ?player2 fut-rel:club ?club .
    FILTER(?player1 != ?player2)
}
```

3.1.2 Compatriot Identification

This rule identifies players who are compatriots (from the same country) (see Figure 30):

```
PREFIX fut-rel: <http://football.org/rel/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player1 ont:compatriot ?player2 .
}
WHERE {
    ?player1 a ?class .
    ?player2 a ?class .
    ?class rdfs:subClassOf* ont:Player .
    ?player1 fut-rel:nation ?country .
    ?player2 fut-rel:nation ?country .
    FILTER(?player1 != ?player2)
}
```

3.1.3 Past Teammates

This rule infers past teammate relationships based on shared previous clubs (was inferred, but it is not displayed on the interface due to visual constraints):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fut-rel: <http://football.org/rel/>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player1 ont:pastTeammate ?player2 .
```

```
}
WHERE {
    ?player1 rdf:type ?pClass1 .
    ?pClass1 rdfs:subClassOf* ont:Player .
    ?player2 rdf:type ?pClass2 .
    ?pClass2 rdfs:subClassOf* ont:Player .
    ?player1 fut-rel:past_club ?club .
    ?player2 fut-rel:past_club ?club .
    FILTER(?player1 != ?player2)
}
```

3.2 Player Performance and Classification Rules

3.2.1 Player Efficiency Calculation

This rule calculates player efficiency based on goals and assists per 90 minutes (see Figure 17):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:efficiency ?eff .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player ont:gls ?goals .
    ?player ont:ast ?assists .
    ?player ont:min ?minutes .
    FILTER(?minutes > 0)
    BIND(ROUND((?goals + ?assists) * 90.0 / ?minutes * 100) / 100 AS ?eff)
}
```

3.2.2 Veteran Status Classification

This rule identifies veteran players (35+ years old) (see Figure 19):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fut-rel: <http://football.org/rel/>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:veteranStatus true .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player fut-rel:born ?birthYear .
    FILTER(2025 - ?birthYear >= 35)
}
```

3.2.3 Young Prospect Identification

This rule identifies young prospects (under 23 years old) (see Figure 20):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fut-rel: <http://football.org/rel/>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:youngProspect true .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player fut-rel:born ?birthYear .
    FILTER(2025 - ?birthYear < 23)
}
```

3.3 Specialized Player Role Rules

3.3.1 Penalty Specialist Classification

This rule identifies penalty specialists with 90%+ success rate (see Figure 23):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fut-rel: <http://football.org/rel/>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:penaltySpecialist true .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player ont:pk ?scored .
    ?player ont:pkatt ?attempted .
    FILTER(?attempted > 0 && (?scored * 1.0 / ?attempted) >= 0.9)
}
```

3.3.2 Playmaker Identification

This rule identifies playmakers based on high assists and key passes relative to games played (see Figure 24):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:playmaker true .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player ont:ast ?assists .
    ?player ont:kp ?keyPasses .
    ?player ont:mp ?matches .
    FILTER(?matches > 0 &&
           (?assists * 1.0 / ?matches) >= 0.3 &&
           (?keyPasses * 1.0 / ?matches) >= 1.5)
}
```

3.3.3 Goal Threat Classification

This rule identifies players who are significant goal threats (see Figure 25):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:goalThreat true .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player ont:gls ?goals .
    ?player ont:mp ?matches .
    FILTER(?matches > 0 && (?goals * 1.0 / ?matches) >= 0.5)
}
```

3.3.4 Key Player Classification

This rule identifies key players based on high average minutes played (see Figure 21):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://football.org/ontology#>
```

```
INSERT {
    ?player ont:keyPlayer true .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player ont:min ?minutes .
    ?player ont:mp ?matches .
    FILTER(?matches > 0 && (?minutes * 1.0 / ?matches) >= 70)
}
```

3.4 Position-Based Classification Rules

3.4.1 Striker Classification

This rule classifies players as strikers based on goal-scoring statistics and position (see Figure 26):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fut-rel: <http://football.org/rel/>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:playerType "Striker" .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player ont:gls ?goals .
    ?player ont:mp ?matches .
    ?player fut-rel:position ?pos .
    FILTER(?matches > 0 &&
           (?goals * 1.0 / ?matches) >= 0.4 &&
           (CONTAINS(LCASE(?pos), "fw") || CONTAINS(LCASE(?pos), "forward")))
}
```

3.4.2 Defensive Midfielder Classification

This rule classifies players as defensive midfielders based on defensive statistics (see Figure 28):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fut-rel: <http://football.org/rel/>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?player ont:playerType "Defensive Midfielder" .
}
WHERE {
    ?player rdf:type ?pClass .
    ?pClass rdfs:subClassOf* ont:Player .
    ?player ont:tkl ?tackles .
    ?player ont:int ?interceptions .
    ?player ont:mp ?matches .
    ?player fut-rel:position ?pos .
    FILTER(?matches > 0 &&
           ((?tackles + ?interceptions) * 1.0 / ?matches) >= 3.0 &&
           CONTAINS(LCASE(?pos), "mf"))
}
```

3.4.3 City Rivals Classification

This rule identifies club rivalries based on same league and city (see Figure 29):

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX fut-rel: <http://football.org/rel/>
PREFIX ont: <http://football.org/ontology#>

INSERT {
    ?club1 ont:cityRival ?club2 .
}
WHERE {
    ?club1 rdf:type ?cClass1 .
    ?cClass1 rdfs:subClassOf* ont:Club .
    ?club2 rdf:type ?cClass2 .
    ?cClass2 rdfs:subClassOf* ont:Club .
    ?club1 fut-rel:league ?league .
    ?club2 fut-rel:league ?league .
    ?club1 fut-rel:city ?city1 .
    ?club2 fut-rel:city ?city2 .
    FILTER(?club1 != ?club2 && ?city1 = ?city2)
}
```

4 New SPARQL Operations

This section presents the new queries used to extract complementary information for the web application. These queries fall into three main categories: **club**, **stadium** and **league**. On each query, the subject (such as a club, stadium or league) is dynamic and replaced with the relevant entity during execution in the web application. However, for the queries shown below, a real example is used in place of the variable to illustrate the expected results.

4.1 Club

4.1.1 Get Club ID from Name

It's necessary to get the corresponding ID for the club at Wikidata to then get all the information needed. Since that the original dataset only contains clubs from top-5 leagues, the filter *FILTER(?league IN (wd:Q324867, wd:Q9448, wd:Q15804, wd:Q13394, wd:Q82595))* has the objective to guarantee that the id returned doesn't belong to an incorrect club. On this example, it's used **"real madrid"** to search for its id.

```
SELECT DISTINCT
    ?club
WHERE {{
    ?club wdt:P31 wd:Q476028 .
    ?club rdfs:label ?label .
    FILTER(LANG(?label) = "en")
    FILTER(CONTAINS(LCASE(?label), "real madrid"))

    ?club wdt:P118 ?league .
    FILTER(?league IN (wd:Q324867, wd:Q9448, wd:Q15804, wd:Q13394, wd:Q82595))
  }}
```

club
wd:Q8682

Figure 5: Get Club Wikidata ID by Name (Output)

4.1.2 Get Club Details from ID

After obtaining the Wikidata ID, it is then possible to retrieve the pretended properties. In the case of properties that have multiple values (e.g., nicknames, sponsors), these values are concatenated in the query and presented in a single line, separated by (";"). For properties whose values include both a name and an image (e.g., sponsors, kit supplier), these are also concatenated, but using the separator ("—"). This data is then processed in the application, which splits the respective fields as needed. On this example it's used the id **wd:Q8682** (Real Madrid ID from the previous query).

```
SELECT
    (GROUP_CONCAT(DISTINCT ?sponsorInfo; separator=";") AS ?sponsors)
    ?kitInfo
    ?officialName
```

```
    (GROUP_CONCAT(DISTINCT ?nickname; separator=";") AS ?nicknames)
    ?audio
    ?inception
    ?presidentInfo
    ?coachInfo
    ?stadiumInfo
    (MAX(?followers) AS ?mediaFollowers)
WHERE {{
    BIND(wd:Q8682 AS ?club)

    OPTIONAL {{
        ?club wdt:P859 ?sponsorEntity .
        ?sponsorEntity rdfs:label ?sponsorName .
        FILTER(LANG(?sponsorName) = "en")

        OPTIONAL {{ ?sponsorEntity wdt:P2910 ?sponsorLogo . }}
        BIND(IF(BOUND(?sponsorLogo),
             CONCAT(?sponsorName, "|", STR(?sponsorLogo)),
             CONCAT(?sponsorName, "|", "")) AS ?sponsorInfo)
    }}
    OPTIONAL {{
        ?club wdt:P5995 ?kitEntity .
        ?kitEntity rdfs:label ?kitName .
        FILTER(LANG(?kitName) = "en")
        OPTIONAL {{ ?kitEntity wdt:P8972 ?kitLogo . }}
        BIND(IF(BOUND(?kitLogo),
                CONCAT(?kitName, "|", STR(?kitLogo)),
                CONCAT(?kitName, "|", "")) AS ?kitInfo)
    }}
    OPTIONAL {{ ?club wdt:P1448 ?officialName . }}
    OPTIONAL {{ ?club wdt:P1449 ?nickname . }}
    OPTIONAL {{ ?club wdt:P443 ?audio . }}

    OPTIONAL {{ ?club wdt:P571 ?inception . }}
    OPTIONAL {{
        ?club wdt:P488 ?presidentEntity .
        ?presidentEntity rdfs:label ?presidentName .
        FILTER(LANG(?presidentName) = "en")
        OPTIONAL {{ ?presidentEntity wdt:P18 ?presidentImage . }}
        BIND(IF(BOUND(?presidentImage),
             CONCAT(?presidentName, "|", STR(?presidentImage)),
             CONCAT(?presidentName, "|", "")) AS ?presidentInfo)
    }}
    OPTIONAL {{
        ?club wdt:P286 ?coachEntity .
        ?coachEntity rdfs:label ?coachName .
        FILTER(LANG(?coachName) = "en")
        OPTIONAL {{ ?coachEntity wdt:P18 ?coachImage . }}
        BIND(IF(BOUND(?coachImage),
             CONCAT(?coachName, "|", STR(?coachImage)),
             CONCAT(?coachName, "|", "")) AS ?coachInfo)
    }}
    OPTIONAL {{
        ?club wdt:P115 ?stadiumEntity .
        ?stadiumEntity rdfs:label ?stadiumName .
        FILTER(LANG(?stadiumName) = "en")
        BIND(CONCAT(STR(?stadiumEntity), "|", ?stadiumName) AS ?stadiumInfo)
    }}
    OPTIONAL {{
        ?club p:P8687 ?statement .
        ?statement ps:P8687 ?followers .
    }}
}}
GROUP BY ?kitInfo ?officialName ?audio ?inception ?presidentInfo ?coachInfo ?stadiumInfo
```

sponsors	Emirates|
kitInfo	Adidas AG|http://commons.wikimedia.org/wiki/Special:FilePath/Adidas%202022%20logo.svg
nicknames	Los Blancos
inception	6 de março de 1902
presidentInfo	Florentino Pérez|http://commons.wikimedia.org/wiki/Special:FilePath/25th%20Laureus%20World%20Sports%20Awards%20-%20Red%20Carpet%20-%20Florentino%20P%C3%A9rez%20-%20240422%201948826%20%28cropped%29.jpg
coachInfo	Xabi Alonso|http://commons.wikimedia.org/wiki/Special:FilePath/Los%20Caminos%20de%20Fe%C3%BAtbol.%20Xabi%20Alonso%20%28396667784641%29%20%28cropped%29.jpg
stadiumInfo	http://www.wikidata.org/entity/Q164027|Santiago Bernabéu Stadium
mediaFollowers	108000000

Figure 6: Get Club Details by ID (Output)

4.2 Stadium

4.2.1 Get Stadium Details by ID

Since on the club details query, on the stadium propoerties, the ID is already returned, it's possible to use that to look directly for the stadium information. Just like before, in the case of the stadium events (a property that have multiple values), these are concatenated in the query and presented in a single line, separated by (";"). On this example it's used the id **wd:Q164027** (Santiago Bernabéu Stadium ID).

```
SELECT
    ?label
    ?name
    ?opening
    ?image
    ?location
    ?capacity
    ?categoryName
    (GROUP_CONCAT(DISTINCT ?eventName; separator=";") AS ?events)
WHERE {{
    BIND(wd:Q164027 AS ?stadium)

    ?stadium rdfs:label ?label .
    FILTER(LANG(?label) = "en")

    OPTIONAL {{ ?stadium wdt:P1705 ?name . }}
    OPTIONAL {{ ?stadium wdt:P1619 ?opening . }}
    OPTIONAL {{ ?stadium wdt:P18 ?image . }}
    OPTIONAL {{ ?stadium wdt:P625 ?location . }}
    OPTIONAL {{ ?stadium wdt:P1083 ?capacity . }}
    OPTIONAL {{
        ?stadium wdt:P9803 ?categoryEntity .
        ?categoryEntity rdfs:label ?categoryName .
        FILTER(LANG(?categoryName) = "en")
    }}
    OPTIONAL {{
        ?stadium wdt:P793 ?eventEntity .
        ?eventEntity rdfs:label ?eventName .
        FILTER(LANG(?eventName) = "en")
    }}
}}
GROUP BY ?stadium ?label ?name ?opening ?image ?location ?capacity ?categoryName
```

label	name	opening	image	location	capacity	categoryName	events
Santiago Bernabéu Stadium	Estadio Santiago Bernabéu	14 de dezembro de 1947	commons:Estadio Santiago Bernabéu Madrid.jpg	Point(-3.68835 40.45306)	78297	UEFA stadium category 4	1982 FIFA World Cup;2030 FIFA World Cup

Figure 7: Get Stadium Details by ID (Output)

4.3 League

4.3.1 Get League ID by Name

On the original dataset the only references to leagues are their names associated to a club. So, to get more information about a league, first it's necessary to look for its Wikidata ID, and then directly. Once more, there is a filter (*FILTER(?country IN (wd:Q145, wd:Q29, wd:Q183, wd:Q142, wd:Q38))*) used to guarantee that the league id returned doesn't belong to a league outside of the top-5 leagues. On this example it's used the name **Premier League** (the English Championship).

```
SELECT DISTINCT ?league WHERE {{
    ?league wdt:P31 wd:Q15991303;
    wdt:P17 ?country;
    rdfs:label "Premier League"@en.
    FILTER(?country IN (
        # FILTRAR PAÍSES para evitar outras ligas com o mesmo nome
        wd:Q145, # United Kingdom
        wd:Q29, # Spain
        wd:Q183, # Germany
        wd:Q142, # France
        wd:Q38 # Italy
    ))
}}
```

league
wd:Q9448

Figure 8: Get League ID by Name (Output)

4.3.2 Get League Details from ID

After obtaining the Wikidata ID, it is then possible to retrieve the pretended properties. On this example it's used the id **wd:Q9448** (Premier League ID from the previous query).

```
SELECT ?name ?logo ?numTeams ?website WHERE {{
BIND(wd:Q9448 AS ?league)

?league rdfs:label ?name .
FILTER(LANG(?name) = "en")

OPTIONAL {{ ?league wdt:P154 ?logo. }}
OPTIONAL {{ ?league wdt:P1132 ?numTeams. }}
OPTIONAL {{ ?league wdt:P856 ?website. }}
}}
```

name	logo	numTeams	website
Premier League	commons:Premier League.svg	20	<https://www.premierleague.com/>

Figure 9: Get League Details by ID (Output)

4.3.3 Get League Winners from ID

Also with the Wikidata ID, it is possible to get all the winners of a given championship. The result is given in multiple lines, being each one of them representative of a season. On this example it's used the id **wd:Q9448** (Premier League ID from the first query). In the cases that the winner is not defined (happens on some of the recent examples), the *team1* variable represents the team placed in 1st place and then, the application assume it as the champion.

```
SELECT ?seasonLabel ?winnerLabel ?team1 ?team1Label WHERE {{
    ?season wdt:P3450 wd:Q9448;
            wdt:P580 ?startTime.

    OPTIONAL {{ ?season wdt:P1346 ?winner. }}

    # Participating team com ranking 1
    OPTIONAL {{
        ?season p:P1923 ?teamStatement.
        ?teamStatement ps:P1923 ?team1.
        ?teamStatement pq:P1352 1.
    }}

    BIND(YEAR(?startTime) AS ?startYear)
    FILTER(?startYear < (YEAR(NOW()) ))

    SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}}
ORDER BY DESC(?startYear)
```

seasonLabel	winnerLabel	team1	team1Label
2024–25 Premier League		wd:Q1130849	Liverpool F.C.
2023–24 Premier League		wd:Q9617	Arsenal F.C.
2022–23 Premier League	Manchester City F.C.	wd:Q50602	Manchester City F.C.
2021–22 Premier League	Manchester City F.C.	wd:Q50602	Manchester City F.C.
2020–21 Premier League	Manchester City F.C.	wd:Q50602	Manchester City F.C.
2019–20 Premier League	Liverpool F.C.	wd:Q1130849	Liverpool F.C.
2018–19 Premier League	Manchester City F.C.	wd:Q50602	Manchester City F.C.
2017–18 Premier League	Manchester City F.C.	wd:Q50602	Manchester City F.C.
2016–17 Premier League	Chelsea F.C.	wd:Q9616	Chelsea F.C.

Figure 10: Get League Winners by ID (Output)

5 Wikidata

In the context of external data sources, we had the option to choose between Wikidata and/or DBpedia. However, we decided to focus exclusively on Wikidata, as it provided a much richer and more complete set of information relevant to football. In contrast, DBpedia was less comprehensive and with missing data necessary for our objectives. This made Wikidata the preferred choice for integrating external knowledge into our system.

The original dataset was highly focused on player-related data, offering detailed statistics and attributes primarily for individual footballers. However, it was possible to associate this data with new entities, such as clubs, stadiums, and leagues, by leveraging the available relationships within the dataset. Despite this, the information for these additional entities was quite basic and limited in scope. For this reason, we aimed to enrich the representation of clubs, stadiums and leagues.

5.1 Club

Club is a Wikidata instance of **association football club** [Q476028]. Initially, this entity had the following properties:

- Name
- City
- Stadium
- League

Through Wikidata, we could get more:

- Official Name [P1448] (the name we had before is the most common name)
- Inception Date [P571]
- Nicknames [P1449]
- Head Coach [P286]
- President/Chairperson [P488]
- Social Media Followers [P8687]
- Pronounciation Audio [P443]
- Brands Associated
  - Kits Supplier [P5995]
  - Sponsors [P859]

Beyond this, we could obtain respective stadium and league entities to explore deeply.

5.2 League

League is a Wikidata instance of **association football league** [Q15991303]

On the first project, this was not an entity, but a club property. Now, this has the following properties:

- Name [rdfs:label]
- Number of Participants [P1132]
- Website [P856]
- Logo [P154]
- President/Chairperson [P488]
- Social Media Followers [P8687]

The data about all the winners is obtained by querying each season [Q27020041] which is a **season of the respective league** [P3450]

5.3 Stadium

Stadium is a Wikidata instance of **association football venue** [Q1154710]

On the first project, this was not an entity, but a club property. Now, this has the following properties:

- Name [rdfs:label]
- Image [P18]
- Opening Date [P1619]
- Coordinates [P625]
- Maximum Capacity [P1083]
- UEFA Category [P9803]

6 RDFa and microformats

Microformats and RDFa are methods for adding structured data to HTML, enhancing its semantic meaning and making it machine-readable. Microformats use existing HTML tags and attributes to mark up content in a predefined way, while RDFa extends HTML with attributes to express data using the Resource Description Framework (RDF). We implemented microformats to represent the stadium and league, while RDFa was employed for the club and player.

6.1 RDFa

To enhance the semantic structure of the HTML pages within the application, RDFa (Resource Description Framework in Attributes) was used to embed metadata directly into the HTML markup. RDFa enables the HTML content to be machine-readable, allowing search engines and semantic web agents to extract and understand the relationships and types embedded in the data.

The vocabulary used for RDFa annotations in this project is primarily based on **Schema.org**, a widely adopted schema maintained by major search engines. Schema.org provides a collection of schemas for structured data markup on web pages, allowing developers to label entities such as `Person`, `SportsTeam`, `Place`, and their properties like `name`, `birthPlace`, `memberOf`, and `sport`.

RDFa was applied in multiple templates, particularly those representing football clubs and players.

For instance, a football player page was annotated as follows:

```
<div vocab="https://schema.org/" typeof="Person">
  <!-- Player image -->
  <div property="image" typeof="ImageObject">
    <img property="contentUrl" src="PLAYER_IMAGE_URL" alt="PLAYER_NAME">
  </div>

  <!-- Player name -->
  <h2 property="name">PLAYER_NAME</h2>

  <!-- Player tags as description -->
  <span property="description">Veteran</span>
  <span property="description">Key Player</span>

  <!-- Player nationality and birthDate -->
  <p>
    <span property="nationality">COUNTRY</span><br>
    <span property="birthDate">1999-01-01</span>
  </p>

  <!-- Current club -->
  <div property="memberOf" typeof="SportsTeam">
    <a property="url" href="CLUB_URL">
      <img property="logo" src="CLUB_LOGO_URL" alt="CLUB_NAME">
      <span property="name">CLUB_NAME</span>
    </a>
  </div>
```

```
</div>
```

And the football club's page received the following annotations:

```
<!-- Team block -->
<div vocab="https://schema.org/" typeof="SportsTeam" resource="#TEAM_ID">
  <!-- Logo and Name -->
  <img property="logo" src="TEAM_LOGO_URL" alt="TEAM_NAME">
  <h2 property="name">TEAM_NAME</h2>

  <!-- Address and Stadium -->
  <p>
    <span property="address">TEAM_CITY</span> •
    <span property="location">STADIUM_NAME</span> •
    <span property="foundingDate">1900-01-01</span>
  </p>

  <!-- League/Organization -->
  <p>
    <span property="memberOf" typeof="SportsOrganization">
      <a property="name" href="LEAGUE_URL">LEAGUE_NAME</a>
    </span>
  </p>

  <!-- Players / Squad -->
  <table>
    <tbody>
      <tr typeof="Person" property="athlete" resource="#PLAYER_ID">
        <td>
          <a property="url" href="PLAYER_URL">
            <img property="image" src="PLAYER_IMAGE_URL" alt="PLAYER_NAME">
            <span property="name">PLAYER_NAME</span>
          </a>
        </td>
        <td>
          <span property="jobTitle">Midfielder</span>
        </td>
        <td>
          <span property="nationality">COUNTRY_NAME</span>
        </td>
        <td>
          <span>AGE</span>
        </td>
      </tr>
    </tbody>
  </table>
</div>
```

This markup makes explicit the semantic roles of elements such as the player's name, image, date of birth, nationality, and affiliation to a sports team where (using `Person` type). Additionally, RDFa was used to annotate the sports team entity itself, using the `SportsTeam` type. Elements such as the team's name, logo, founding date, home city, stadium location, and associated league were marked up semantically.

RDFa annotations were verified using the Schema.org validator to ensure that the structure and types conformed to expected standards as seen in Fig. 11.

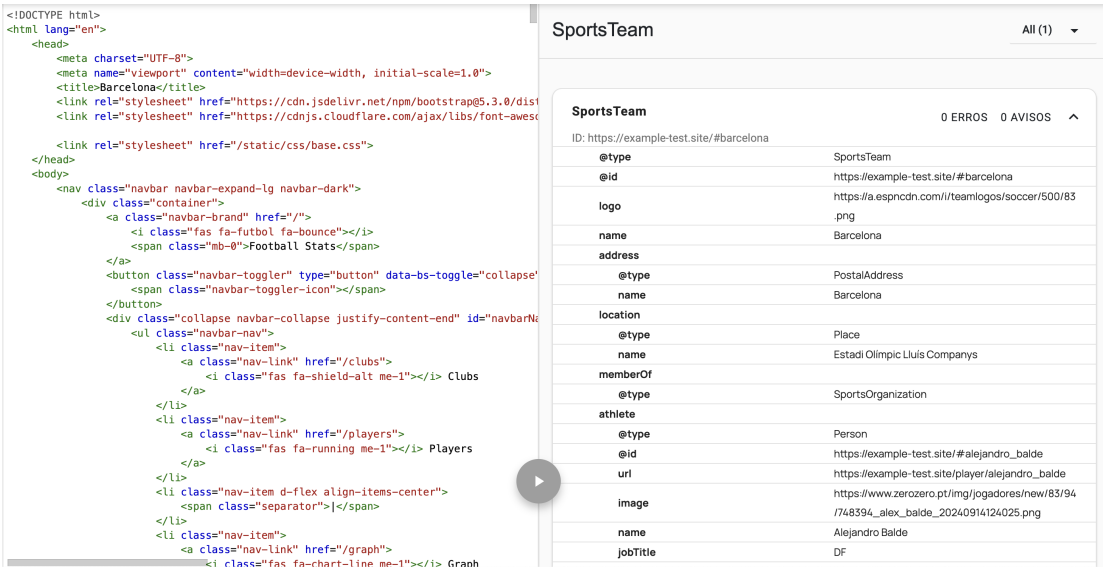

Figure 11: **Schema.org** validator confirming RDFa annotation integrity (Club web page)

6.2 Microformats

In addition to RDFa, Microformats were implemented to provide a different way to add semantic meaning using standard class attributes. Microformats are simple conventions for using HTML to mark up information such as people, organizations, locations, and events.

For this project, microformats were used in the stadium and league template. For stadiums, the `h-card` class was used to describe properties like the name, category, capacity, and opening date using `p-name`, `p-category`, `dt-start`, and other related classes. This not only reinforces the semantic clarity of the content but also improves search engine compatibility and enables certain crawlers to recognize and extract structured data. For leagues, the `h-org` format was used to mark up organizational metadata including logo, name, website, and the number of teams. These annotations help identify the league as an organization and allow semantic agents to extract its attributes easily.

An example of a Microformat annotation for a stadium:

```
  <!-- Venue as h-card -->
<div class="h-card">
```

```
  <!-- Image and Name -->
  <img src="VENUE_IMAGE_URL" alt="VENUE_NAME" class="u-photo">
  <h2 class="p-name">VENUE_NAME</h2>

  <!-- Optional Category -->
  <span class="p-category">Stadium</span>

  <!-- Capacity and Opening Date -->
  <p>
    <span><i class="fas fa-users"></i> 50,000</span>
    •
    <time class="dt-start" datetime="2001-09-01">2001</time>
  </p>

  <!-- Events Section -->
  <div>
    <h3>Notable Events</h3>

    <div class="h-event">
      <p class="p-name">Champions League Final</p>
    </div>

    <div class="h-event">
      <p class="p-name">Concert: The Rolling Stones</p>
    </div>
  </div>
</div>
```

And for a league:

```
  <!-- Organization Card -->
<div class="h-org">
  <!-- Logo and Name -->
  <img src="TEAM_LOGO_URL" alt="TEAM_NAME" class="u-logo">
  <h2 class="p-name">TEAM_NAME</h2>

  <!-- Optional Info -->
  <p>
    <span class="p-note">NUMBER_OF_TEAMS Teams</span>
    •
    <a class="u-url" href="https://example.com">Website</a>
  </p>

  <!-- Winners Section -->
  <div>
    <h3>Winners</h3>
    <div class="h-card">
      <small class="dt-start">2023</small>
      <h5 class="p-name">Winning Team Name</h5>
    </div>
  </div>
</div>
```

While Microformats are useful for many common entities, there were occasional mismatches between the available vocabulary and the desired domain-specific semantics. For example, the use of `dt-start` to represent the founding date of a stadium is a practical workaround, even though it is originally

intended for events. Despite these minor limitations, there was general compatibility between Microformats and the majority of the content types used in the application.

We used a microformats parser (Microformats Parser) to assess the semantic value of the generated code. This allowed us to extract and identify all entities, with the semantic content presented in JSON format, as shown in Figure 12

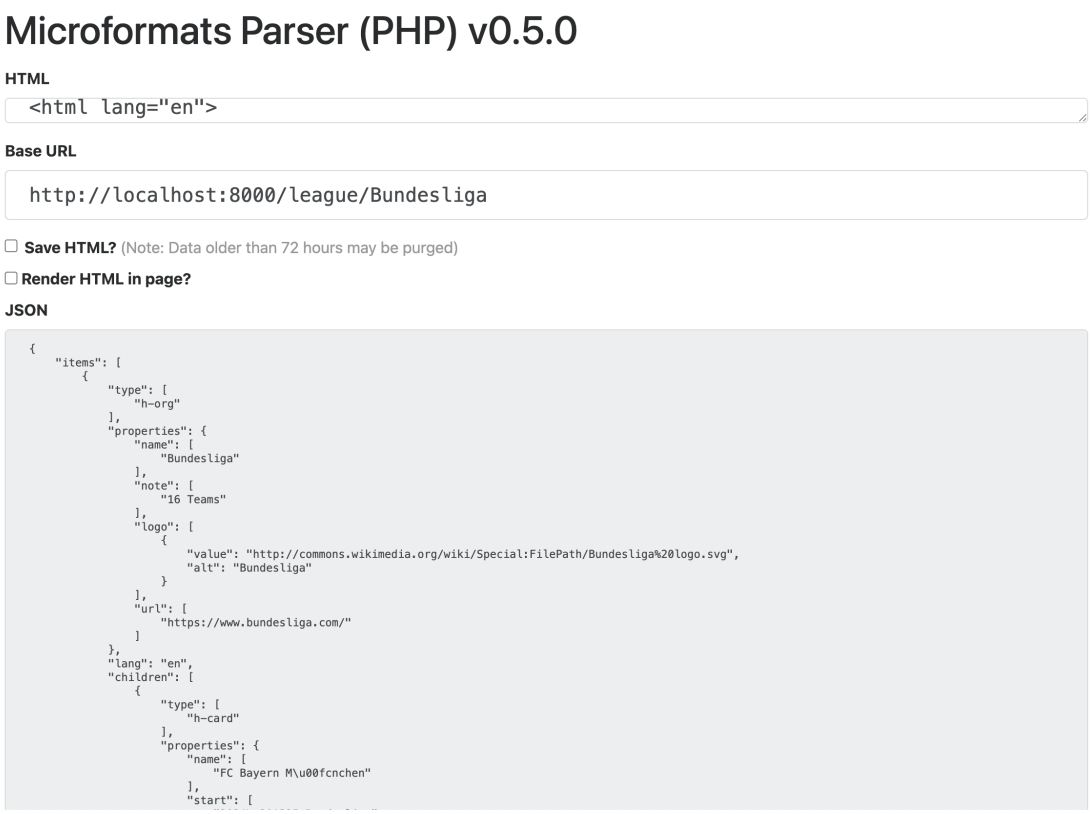

Figure 12: Semantic content extracted by the **Microformats Parser**, displayed in JSON format for a league web page.

7 New features

7.1 Integration Features

7.1.1 Club Page

The Club page provides an in-depth overview of a specific football club, presenting essential information about the team, including its logo, location, stadium and league. This page was updated and now also provides information about foundation date, nicknames for the team, associated brands, manager, president and, in some cases, an audio player with the correct pronounciation of the club.

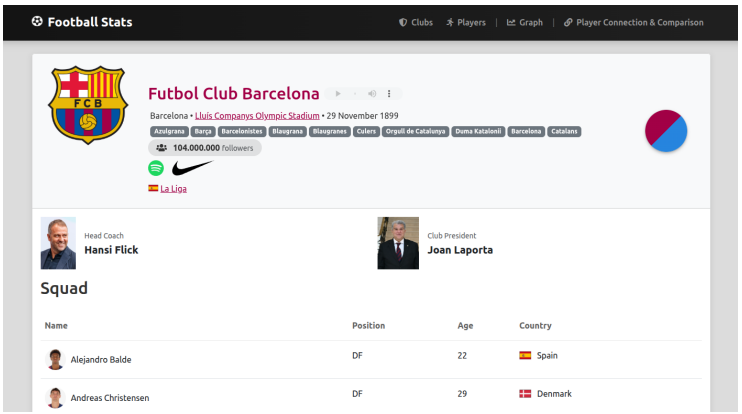

Figure 13: Club Page Updates

7.1.2 League Page

The League page is a new page that provides an overview of a specific football league, presenting basic information about it, including its logo, number of teams and website.

7.1.3 Stadium Page

The Stadium page is a new page that provides an overview of a specific stadium, presenting basic information about it, including an image, UEFA category, capacity, foundation date and location. Besides that, some stadiums present a list of important events hosted by it.

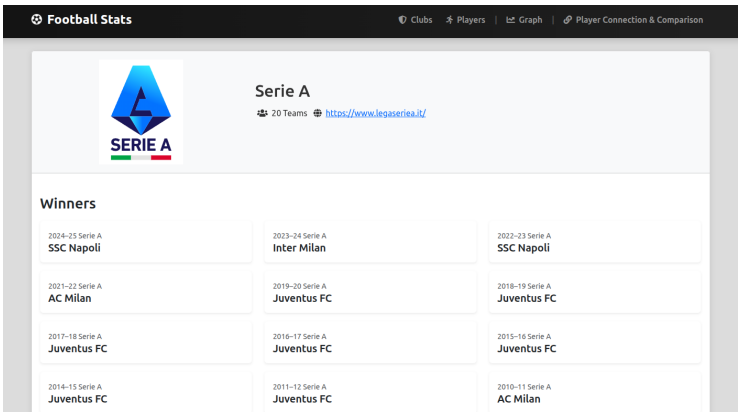

Figure 14: League Page

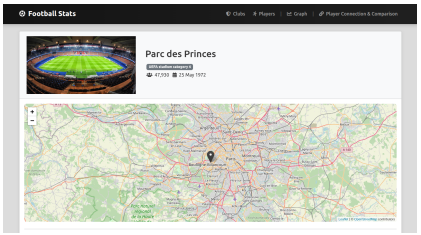

Figure 15: Stadium Page (basic info)

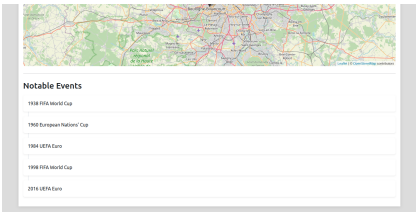

Figure 16: Stadium Page (important events)

7.2 Spin Rules Features

Some SPIN rules were created to enable the inference of new relationships and properties within the football ontology, enriching the dataset with implicit knowledge that is not directly stated in the original data.

7.2.1 Player Efficiency

This is a numeric value calculated by the average goals contributions (goals or assists) by 90 minutes. The player also gains the label "High Efficiency"

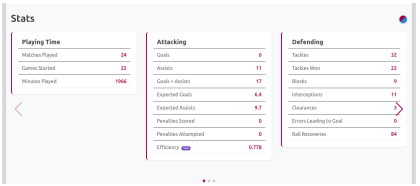

Figure 17: Player Efficiency

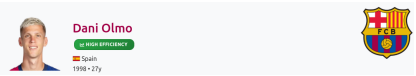

Figure 18: High Efficiency Player

7.2.2 Player Game Styles and Status

The rules and queries used to determine the player game styles (and status) are described in more detail in section 3, however the images are accompanied by a small explanation of said rules.

Identify Veterans

A player is considered a "Veteran" if he is older than 35 years old

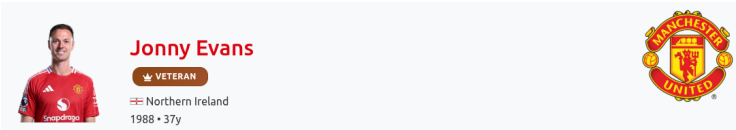

Figure 19: Veteran Player

Identify Young Prospects

A player is considered a "Young Prospect" if he is younger than 23 years old

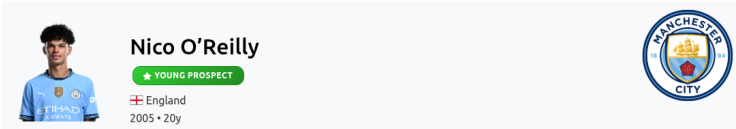

Figure 20: Young Prospect Player

Identify Key Players

A player is considered a "Key Player" if he plays, on average, more than 70 minutes per game

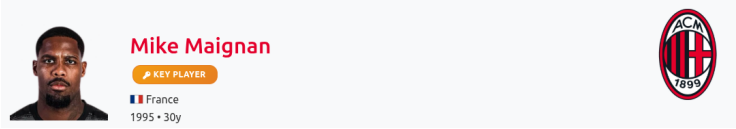

Figure 21: Key Player

Identify Disciplinary Risk Players

A player is considered a "Disciplinary Risk" if he has a high rate of cards.

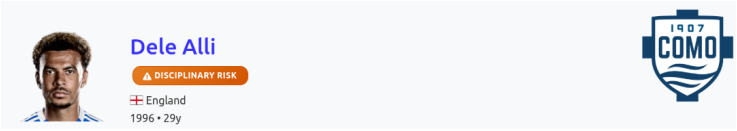

Figure 22: Disciplinary Risk Player

Identify Penalty Specialists Players

A player is considered a "Penalty Specialist" if he plays, on average, more than 70 minutes per game

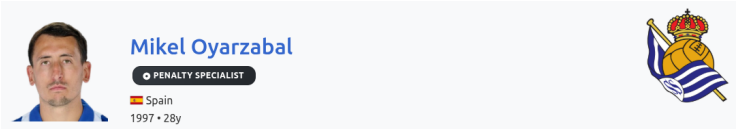

Figure 23: Penalty Specialist Player

Identify Playmakers Players

A player is considered a "Playmaker" if he performs a high number of assists and key passes

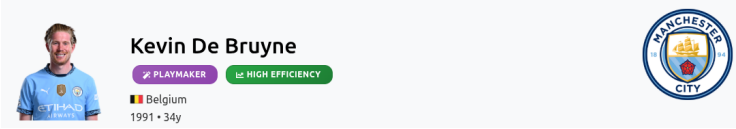

Figure 24: Playmaker Player

Identify Goal Threats Players

A player is considered a "Goal Threat" if he scores more than 1 goal for each 2 matches

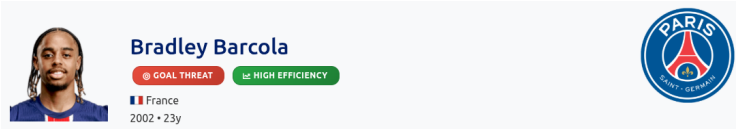

Figure 25: Goal Threat Player

Identify Strikers

Similar to the previous rule, a player is classified as a "Striker" if he scores more than 1 goal for each 2 matches, with the difference being that he plays the "Forward" position instead

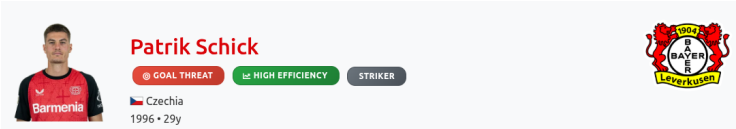

Figure 26: Striker Player

Identify Defensive Midfielders

A player is classified as a "Defensive Midfielder" if he has a high tackles rate and he plays the "Midfielder" position

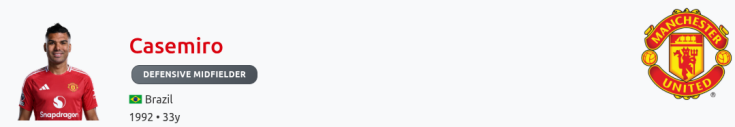

Figure 27: Defensive Midfielder Player

Identify Versatiles

A player is considered "Versatile" if he performs very well in both attacking and defensive scenarios.

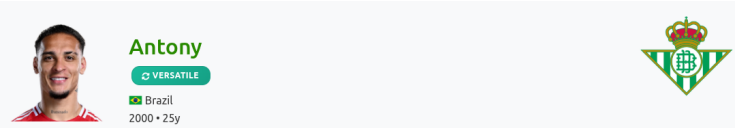

Figure 28: Versatile Player

7.2.3 Relationships

Identify Rival Clubs

A club is considered a "Rival Club" of another club if they play on the same city and league

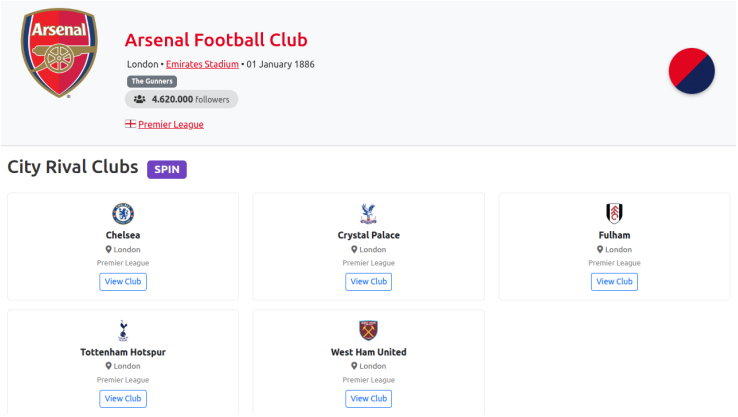

Figure 29: Rival Club

Identify Teammates and Compatriots

Two players are considered teammates if they play for the same club, and are considered compatriots if they are from the same country

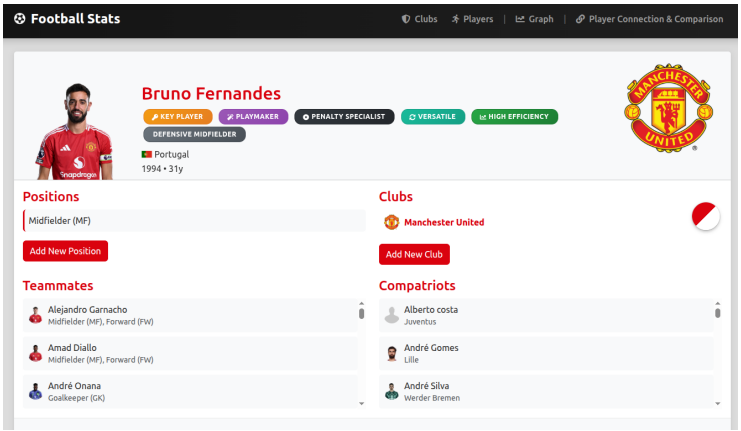

Figure 30: Player's teammates and compatriots inferred by ontology

7.2.4 Player Connection And Comparison

The player connection/comparison feature was updated to determine connections through player game styles and status

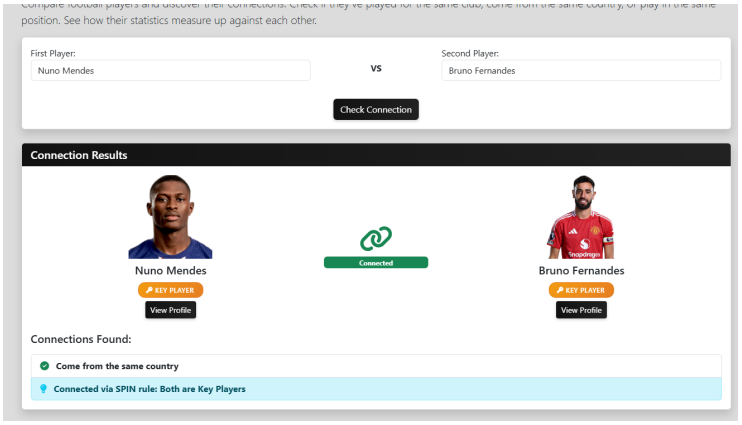

Figure 31: Comparison between two 'Key Players'

8 Conclusions

This project presented an improved approach to modeling data in the football domain, by developing an ontology using RDFS and OWL. It was possible to semantically represent key football entities, players, clubs, leagues, stadiums, and statistics, as well as explicit and inferred relationships among them, alongside other object and data properties.

The use of SPIN rules enabled the inference of implicit knowledge, such as identifying play styles and deducting relationships. The integration with Wikidata enriched the dataset, providing a more complete representation of some entities.

In addition, incorporating semantic web standards like RDFa and Microformats into the web interface gave semantic value to the content, which is machine-readable and further compatible with search engines and semantic tools.

Overall, this project marks a significant improvement over the first one, in terms of data depth, but mainly on technical terms, since it is demonstrated the effectiveness of semantic technologies and the semantic web on the football context.

Configurations

1. Extract the Zip File

After downloading the project, extract the zip file and navigate to the project directory:

2. Set Up Django App

```
cd ws_project_1
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Before proceeding to the next step, ensure that **GraphDB** is running and accessible at `http://localhost:7200`.

3. Import Data and Create Repositories

First, create a GraphDB repository for the football data. **Important:** The repository must be named `football`.

Once the repository is created, import both the RDF data and the ontology:

- `data/import/football_rdf_data.nt` contains the RDF dataset for football.
- `data/import/ontology/football_ontology.n3 contains the ontology.`

4. Run Django

Execute the following command:

```
GRAPHDB_ENDPOINT=http://localhost:7200 python3 manage.py runserver
```

The environment variable `GRAPHDB_ENDPOINT` must point to your running GraphDB instance. If you are using a non-default port or host, adjust the URL accordingly.

5. Other Tests

In order to import both the data and the ontology into Protégé, there is a file, `data/import/football_rdf_data_ontology.n3`, that serves that same purpose.

The SPIN rules used for this project are located in:

- `data/import/ontology/football_spin_rules.n3`

In order to see how they are applied, it is possible to do so in the following files:

- `ws_project_1/app/utils/spin_client.py`
- `ws_project_1/app/utils/spin_queries.py`

Note: the name of the folder is 'ws_project_1' because both projects are available on GitHub and the second is a fork of the first.

6. Possible Errors

In some pages, messages like "Some additional information could not be loaded from external sources" may appear, due to errors fetching data from Wikidata. If that happens, refresh the page (this is not a coding issue from our part, it's a problem with wikidata and their API).

References

[1] The most popular sports in the world - worldatlas, 2025. URL `https://www.worldatlas.com/articles/what-are-the-most-popular-sports-in-the-world.html`.