Universidad de La Laguna

Development of a web application based on SparkQL queries

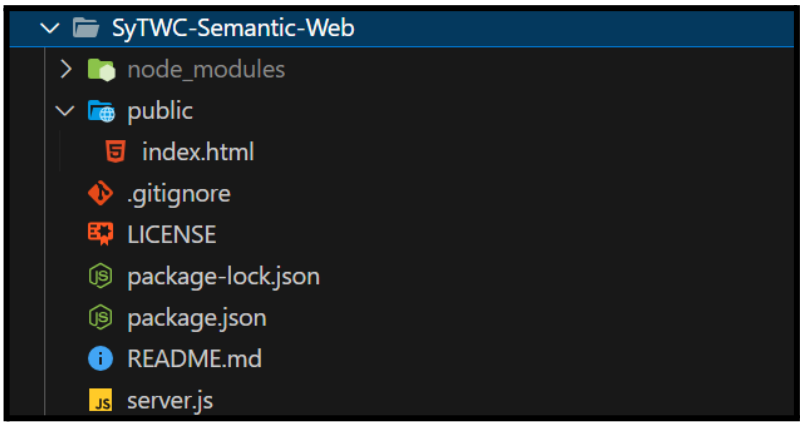
Master's Degree in Computer Engineering - Systems and Web Technologies: Client

Yago Pérez Molanes

This report describes the technical characteristics and operation of a web application dedicated to the Wikidata and DBPedia platforms and based on SparkQL queries with the JavaScript Comunica library.

Technical features

First of all, we have chosen to follow a monolithic development of a web application dedicated to the purpose described above. In the project created (a typical nodejs and javascript project), also available in the following repository, two source code files have been created, an index.html and a server.js, described below:



Server

Our application has a server, as the Comunica library cannot be executed directly on the web browser. The structure of the server is shown below:

```
// Import the necessary modules

const express = require('express');

const                          QueryEngine                          =
require('@comunica/query-sparql').QueryEngine;



// Create a new QueryEngine instance
```

```
const myEngine = new QueryEngine();



// Create a new Express application

const app = express();



// Middleware to parse JSON bodies

app.use(express.json());



// Middleware to serve static files from the "public" directory

app.use(express.static('public'));
```

First the necessary libraries are imported, an instance of the query engine is created and a basic Express server is initialised, middleware is added to handle JSON bodies, and to serve static files, a public directory where our html file is located.

There are 4 endpoints in the application, all of which are handled through GET requests and consist of specific requests for data on the football theme.

```
'/search/spanishteams'

'/search/teamsportstitles'

'/search/teamstadiums'

'/search/city/:city'
```

The first one is used to search for Spanish teams, the second one to search for how many titles each football team has in total, the third one to retrieve the stadiums of each team, and the last one is to search for those players that match

the city that the user enters, therefore in this last endpoint a parameter is handled.

The handling of a specific route is shown below:

```
// Route to search for Spanish football teams

app.get('/search/spanishteams', async (req, res) => {

    // Execute a SPARQL query to fetch Spanish football teams

    const bindingsStream = await myEngine.queryBindings(

        // SPARQL query

        `

        PREFIX wd: <http://www.wikidata.org/entity/>

        PREFIX wdt: <http://www.wikidata.org/prop/direct/>



        SELECT ?team ?teamLabel

        WHERE {

        ?team wdt:P31 wd:Q476028;  // Instance of football team

                wdt:P17 wd:Q29.      // Country: Spain

        ?team rdfs:label ?teamLabel.

        FILTER(LANG(?teamLabel) = "es").

        }

        `, {

                                            sources:        [{type:
'sparql',value:'https://query.wikidata.org/sparql'}]
```

```
        },
    );


    // Array to store the results
    const results = [];


    // Event listener for 'data' event
    bindingsStream.on('data', (binding) => {
        results.push(binding.toString());
    });


    // Event listener for 'end' event
    bindingsStream.on('end', () => {
        res.json(results);
    });


    // Event listener for 'error' event
    bindingsStream.on('error', (error) => {
        console.error(error);
        res.status(500).send('Error executing SPARQL query');
    });
```

```
});
```

Below we explain each query that has been used on the server:

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

SELECT ?team ?teamLabel
WHERE {
?team wdt:P31 wd:Q476028;  // Instance of football team
      wdt:P17 wd:Q29.      // Country: Spain
?team rdfs:label ?teamLabel.
FILTER(LANG(?teamLabel) = "es").
```

The query uses SparkQL, a query language used to retrieve data stored in the Resource Description Format (RDF) commonly used in graph databases and the semantic web.

The consultation does the following:

1. Defines the wd and wdt prefixes, which are used to abbreviate the query URLs.
2. Select two variables, ?team and ?teamLabel
3. In the Where clause, it specifies the conditions that the data must meet to be included in the result. In this case, it looks for resources that are an instance of a football team (wd: Q476028) and that are in Spain (wd: Q29).
4. It then retrieves the label (rdfs: label) of each team and assigns it to the ?teamLabel variable.
5. Then with the expression FILTER(XXX), filter the results to include those devices whose label is in English.

Therefore, this query returns a list of football teams in Spain, along with their labels in Spanish.

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>

SELECT ?teamLabel (COUNT(?championship) AS ?titles) WHERE {
?championship wdt:P31 wd:Q27020041;  # Instancia de "Campeonato de fútbol español"
              wdt:P1346 ?team.       # Ganador
?team wdt:P31 wd:Q476028;           # Instancia de "equipo de fútbol"
      rdfs:label ?teamLabel.
FILTER(LANG(?teamLabel) = "es").
}
GROUP BY ?teamLabel
ORDER BY DESC(?titles)
LIMIT 50
```

The previous query is somewhat more complex than the first one, which is explained below:

This SparkQL query is used to obtain a list of Spanish football teams and the number of football championships they have won.

1. PREFIX: These lines define abbreviations for the URLs used in the query.
2. SELECT ?teamLabel (COUNT(?championchip) AS ?titles) This line selects the variables to be included in the results. In this case, the team labels (?teamLabel) and the number of championships they have won (?titles) are selected.
3. WHERE: This clause specifies the conditions that the data must meet to be included in the results.
4. ?championship wdt:P31 wd:Q27020041 This line selects resources that are an instance of championship football.
5. wdt:P1346 ?team: This line selects the championships where the winner is the ?team.
6. ?team wdt:P31 wd: Q476028: This line selects the teams that are an instance of football team (wd:Q476028).
7. rdfs: label ?teamLabel: This line selects the team label (?teamLabel).
8. rdfs:label ?teamLabel: This line selects the team label (?teamlabel).
9. FILTER(lang(?TteamLabel) = "es"): This line filters the results to include only those teams whose label is in English.
10. GROUP BY ?teamLabel: This line groups the results by the team label.
11. ORDER BY DESC(?titles): This line sorts the results in descending order by the number of titles won.
12. LIMIT 50: This line limits the results to the first 50.

This query returns a list of the top 50 football teams and the number of championships they have won, sorted from highest to lowest.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT ?team ?teamLabel ?stadium ?stadiumLabel
WHERE {
?team a dbo:SportsTeam;
      dbo:ground ?stadium.
?stadium dbo:country dbr:Spain.
?team rdfs:label ?teamLabel.
?stadium rdfs:label ?stadiumLabel.
FILTER(LANG(?teamLabel) = "es" && LANG(?stadiumLabel) = "es")
}
ORDER BY ?teamLabel
```

The above query is about DPpedia and consists of the following:

1. PREFIX: These lines define the abbreviations for the URIs used in the query.
2. SELECT ?team ?teamLabel ?stadum ? stadiumLabel: This line is used to select the variables to be included in the results. In this case, the teams ?team, the team labels ?teamLabel, the stadiums ?stadium and the stadium labels ?stadiumLabel are selected.
3. ?team a dbo:SportsTeam; dbo:ground ?stadium: This line selects resources that are an instance of a sports team (dbo:SportsTeam) and that have a stadium (dbo: ground).
4. ?stadium dbo:country dbr:Spain. This line selects stadiums that are in Spain.
5. ?team rdfs:label ?teamLabel: ?stadium rdfs:label ?stadiumLabel: These lines select the team and stadium labels.
6. FILTER (LANG(?teamLabel) = "es" && LANG(?stadiumLabel) = "es"): This line filters the results to include only those teams and stadiums whose labels are in English.
7. ORDER BY ?teamLabel: This line sorts the results by team label.

Here is the latest consultation:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT *
WHERE {
?athlete a dbo:SoccerPlayer ;
        dbo:birthPlace [ rdfs:label "${city}"@en ] ;
        dbo:number ?number .
}
```

This query is used to obtain information on football players born in a specific city. Here is the breakdown of the query:

1. PREFIX: These lines define abbreviations for the URIs to be used in the query.
2. SELECT *: This line selects all variables to be included in the results. In this case, all variables that are defined in the WHERE query will be selected.
3. WHERE: This clause specifies the conditions that the data must meet to be included in the results.
4. ?athlete a dbo:SoccerPlayer ;: This line selects resources that are an instance of football player (dbo:SoccerPlayer).
5. dbo: birthPlace [rdfs:label "${city}"@en]: This line selects football players who were born in a specified city (${city}). Here city is a placeholder that is replaced by the city name when the query is executed.
6. dbo: number ?number: This line selects the number of the football player (dbo: number).

In short, this query returns information about football players who were born in a specified city, including their number.

Finally, the following lines correspond to the server listening opening:

```
const PORT = process.env.PORT || 3000;

app.listen(PORT, () => console.log(`Server started on port
${PORT}`));
```

Client

As we mentioned previously, the design of the application is monolithic, our client and server are centralised, in the public folder of the project are the static files,

which in this case only includes an html file (index.html). We will now comment on the most relevant aspects of it:

```
<h1>Ejemplos de consultas SPARQL</h1>
<div class="form-container">
    <form id="spanishTeamsForm">
        <button type="submit">Lista de equipos de fútbol españoles</button>
    </form>
    <form id="searchForm">
        <button type="submit">Títulos de fútbol de cada equipo</button>
    </form>
    <form id="teamStadiumsForm">
        <button type="submit">Estadios de equipos de fútbol</button>
    </form>
    <form id="citySearchForm">
        <input type="text" id="citySearchTerm" placeholder="Buscar jugadores por ciudad">
        <button type="submit">Buscar</button>
    </form>
    <table id="results">
        <tbody></tbody>
    </table>
```

In the previous screenshot we can see that three buttons and a search engine have been created, in order to be able to execute SparkQL queries, that is why they are assigned unique identifiers so that later from the client's javascript they can be selected and send and receive the data to them:

Here the Javascript code, also included inside the html file:

```
<script>

    document.getElementById('spanishTeamsForm').addEventListener('submit', function(event) {
        event.preventDefault();
        fetch(`/search/spanishteams`)
            .then(response => response.json())
            .then(data => {
                const resultsElement = document.getElementById('results').getElementsByTagName('tbody')[0];
                resultsElement.innerHTML = '';
                data.forEach(item => {
                    const tr = document.createElement('tr');
                    const td = document.createElement('td');
                    td.textContent = item;
                    tr.appendChild(td);
                    resultsElement.appendChild(tr);
                });
            })
            .catch((error) => {
                console.error('Error:', error);
            });
    });
```

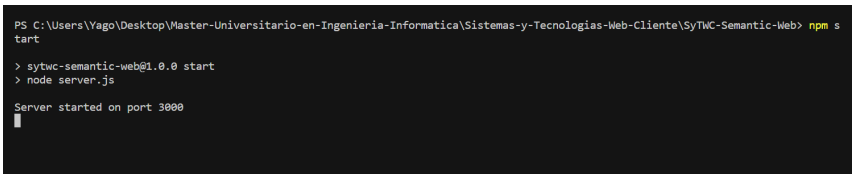
In fact, there are 4 small functions, but we comment only the first one because it is very similar to the rest, basically we add a listening event which is activated when the user clicks on the button, which makes the code inside it to be executed, which is a fetch request (local) to one of the routes that we have defined before, and then inside it the defined response is processed and added to the html element.

Operation of the application

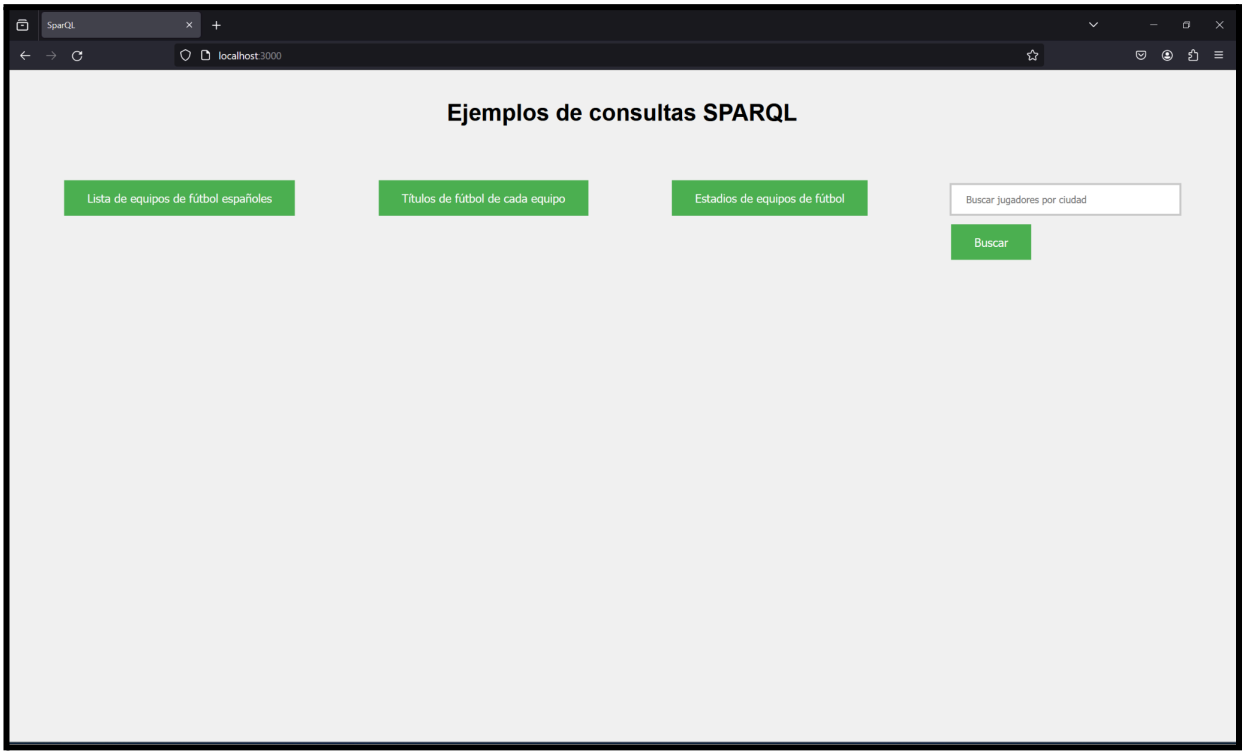
The web application can be tested in two ways, locally, or through a URL where the application is deployed.

Local operation

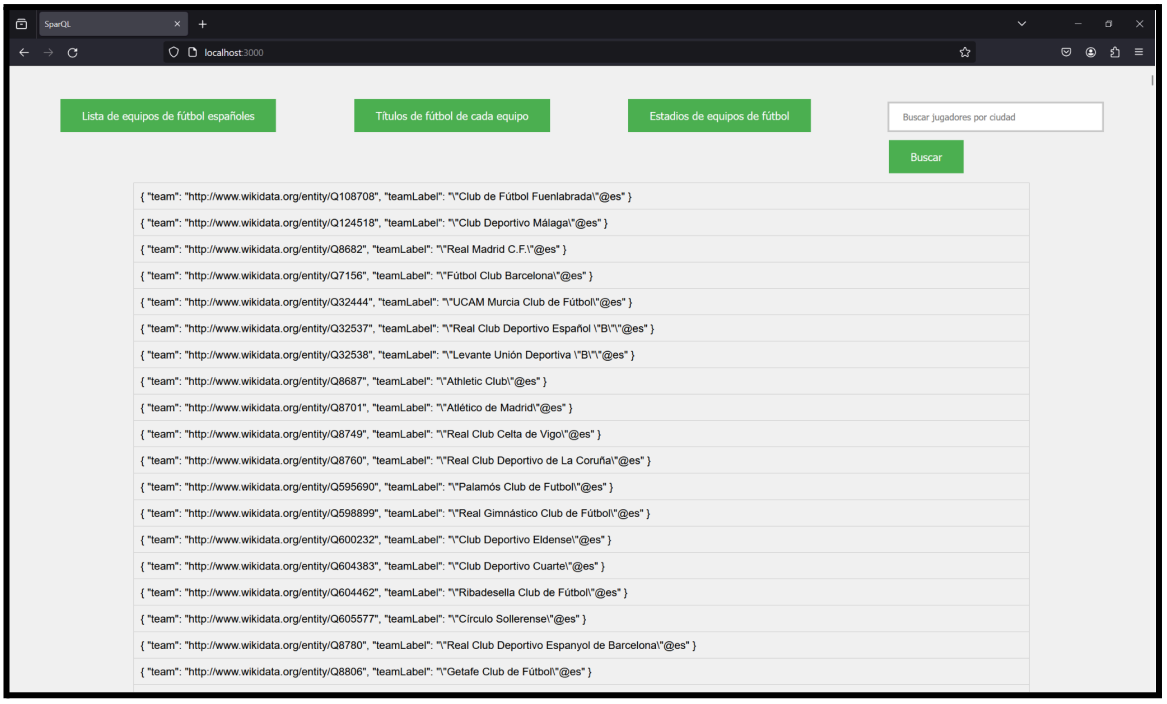
To run the application what would be needed would be to download the github repository and once we are inside the directory install the dependencies with npm install, and once the node_modules folder is added to the project, we run npm start, which starts the express server, we would see something like the following:



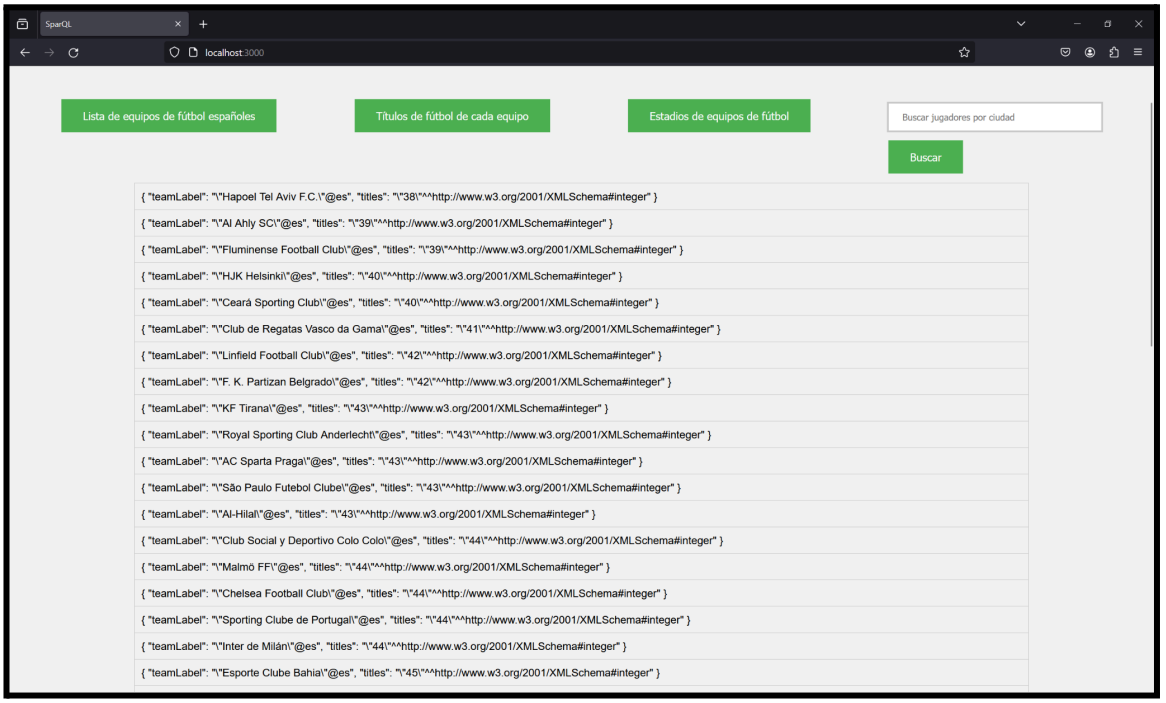
We have to access in our browser to the address http://localhost:3000:



List of Spanish football teams:



Football titles of each team:



Search for football player by city:

Cloud deployment

To test our application in the cloud, simply go to the following address:

- https://sytwc-semantic-web.onrender.com/

And the operation would be the same as locally.

Bibliography

1. Render web services deployment platform.
2. Wikidata
3. DBpedia
4. Comunica Library
5. SparkQL