

Web Semântica - Formula 1 Pitstop

João Andrade 107969, Tomas Victal 109018, José Gameiro 108840

1 Introdução

O nosso trabalho foca-se no desenvolvimento de um sistema de informação baseado na **Web**, utilizando dados da Fórmula 1. O conjunto de dados que escolhemos abrange informações detalhadas sobre as corridas, temporadas, pilotos, construtores e qualificações para os pilotos e construtores abrangendo desde o início da competição em 1950 até a temporada mais recente de 2024. Estes dados foram extraídos de um dataset que se encontra presente na plataforma **Kaggle** e depois foram transformados para o formato **N3** com auxílio de um *script* de **Python**.

2 Dataset

Os dados utilizados neste projeto provêm do conjunto de dados disponível no Kaggle sobre o *Formula 1 World Championship (1950-2024)* disponível através deste [link](#). Este é composto por múltiplos ficheiros CSV que contêm informações detalhadas sobre circuitos, equipas, pilotos, corridas, resultados e estatísticas relacionadas com a Fórmula 1. Sendo estes:

- **circuits.csv**: Informação sobre circuitos, incluindo ID, nome, localização, coordenadas e URL.
- **constructor_results.csv**: Resultados dos construtores por corrida, incluindo pontos e estado.
- **constructor_standing.csv**: Classificação dos construtores em cada corrida, incluindo posição, pontos e vitórias.
- **constructors.csv**: Informação sobre construtores, incluindo nome, nacionalidade e URL.
- **driver_standing.csv**: Classificação dos pilotos em cada corrida, com pontos, posição e vitórias.
- **drivers.csv**: Informação sobre os pilotos, incluindo nome, número, código, data de nascimento e URL.
- **qualifying.csv**: Resultados das sessões de qualificação, incluindo tempos das voltas.
- **races.csv**: Detalhes das corridas, incluindo ano, circuito, datas das sessões e URL.
- **results.csv**: Resultados das corridas, incluindo posição, pontos, tempos e estatísticas.
- **seasons.csv**: Informação sobre as temporadas, incluindo ano e URL.
- **sprint_results.csv**: Resultados das corridas sprint.
- **status.csv**: Estado dos pilotos e equipas durante as corridas.

A transformação dos dados foi realizada através de um script em Python, cujo objetivo é converter os dados dos ficheiros CSV para o formato RDF (*Resource Description Framework*) na sintaxe **N3**. O processo seguiu os seguintes passos:

1. Leitura de cada ficheiro CSV e determinação do respetivo *namespace* com base no nome do ficheiro.
2. Criação das entidades RDF com identificadores únicos baseados nos valores da primeira coluna de cada ficheiro.

3. Atribuição de relações entre as entidades com base nos identificadores referenciados noutras colunas.
4. Caso alguns valores que se encontrem no ficheiro contenham \N, significa que não contem informação relevante, logo estes dados são descartados.
5. Conversão dos valores numéricos e categóricos para os respetivos tipos RDF.
6. Geração do ficheiro RDF final consolidado.

O código do script Python realiza esta transformação de forma automatizada, garantindo que os dados são convertidos corretamente e mantêm a sua integridade relacional. Cada ficheiro é processado individualmente e as ligações entre entidades são ligadas através do uso de *namespaces* adequados. No final do processo, um ficheiro RDF único é gerado contendo toda a informação estruturada do dataset em N3.

3 Operações Sobre os Dados

3.1 Corridas por Temporada

Na página de cada temporada, queremos mostrar informação sobre todas as corridas que ocorreram nesse ano. Esta informação inclui não só o nome e a data da corrida, mas também dados sobre o condutor (e o respetivo construtor) que venceu a corrida, assim como o condutor que fez a volta mais rápida.

Os dados referentes aos resultados das corridas são do tipo "Result" e contêm informação sobre a posição e a volta mais rápida de cada condutor para cada corrida. O primeiro passo é, então, obter o valor da volta mais rápida para cada corrida, ou seja, obter o valor "mínimo" das voltas mais rápidas de todos os condutores para essa corrida. Para obter esses dados fazemos uma pesquisa por todas as corridas que ocorreram na temporada pretendida e todos os resultados relacionados a essas corridas. De seguida, agrupamos os resultados pelo nome e ID da corrida e fazemos o **MIN** dos valores das voltas mais rápidas:

```
SELECT ?raceId ?raceName (MIN(?fastest) AS ?minFastestLap)
WHERE {{
    ?raceId a type:Race ;
    pred:name ?raceName ;
    pred:year "{year}"^^xsd:int .
    ?result a type:Result ;
    pred:raceId ?raceId ;
    pred:fastestLapTime ?fastest .
}}
```

GROUP BY ?raceId ?raceName

Agora que temos o valor da volta mais rápida para cada corrida, podemos utilizá-lo para filtrar os resultados das corridas e extrair o condutor que obteve a volta mais rápida e o construtor a quem pertencia para essa corrida. Em seguida, obtemos também os seus nomes para os podermos mostrar na página web:

```
?result a type:Result ;
    pred:raceId ?raceId ;
    pred:fastestLapTime ?fastestLap ;
    pred:driverId ?fastestDriverId ;
    pred:constructorId ?fastestConstructorId ;
    pred:position ?position .
```

```
FILTER(?fastestLap = ?minFastestLap)
```

```
?fastestDriverId a type:Driver ;
```

```

    pred:forename ?fastestDriverForename ;
    pred:surname ?fastestDriverSurname .

BIND(CONCAT(?fastestDriverForename, " ", ?fastestDriverSurname) as ?fastestDriverName) .

?fastestConstructorId a type:Constructor ;
    pred:name ?fastestConstructorName .

Para obter o vencedor, a ideia é a mesma: pegamos nos resultados de todas as corridas em que a
posição do condutor é "1" e conseguimos assim facilmente obter a informação sobre o condutor e o
construtor:

?winnerResult a type:Result ;
    pred:raceId ?raceId ;
    pred:fastestLapTime ?winnerfastestLap ;
    pred:driverId ?winnerDriverId ;
    pred:constructorId ?winnerConstructorId ;
    pred:position "1"^^xsd:string .

?winnerDriverId a type:Driver ;
    pred:forename ?winnerDriverForename ;
    pred:surname ?winnerDriverSurname .

BIND(CONCAT(?winnerDriverForename, " ", ?winnerDriverSurname) as ?winnerDriverName) .

?winnerConstructorId a type:Constructor ;
    pred:name ?winnerConstructorName .

```

3.2 Corridas agrupadas por nome

```

PREFIX ns: <{NS}>
PREFIX pred: <{PRED}>
PREFIX type: <{TYPE}>
SELECT ?raceName
(GROUP_CONCAT(CONCAT(STR(?raceId), "__", STR(?year)); SEPARATOR=",") AS ?raceDetails)
WHERE {
    ?raceId a type:Race ;
        pred:name ?raceName ;
        pred:year ?year .
}
GROUP BY ?raceName
LIMIT {LIMIT}
OFFSET {offset}

```

Esta *query* tem como objetivo retornar todas as corridas existentes agrupadas por nome, sendo que existem várias corridas com o mesmo nome mas que ocorreram em anos distintos.

- **PREFIX:** Define os espaços de nomes (*namespaces*) usados na consulta, garantindo que os identificadores RDF sejam referenciados corretamente.
- **SELECT:** Define as variáveis que vão ser retornadas. Neste caso:
 - ?raceName - Nome da corrida.
 - ?raceDetails - Uma *string* agregada contendo o ID da corrida e o ano correspondente.
- **WHERE:** Especifica os padrões que os dados devem satisfazer:
 - ?raceId a type:Race - Filtra apenas entidades do tipo "Race".
 - pred:name ?raceName - Obtém o nome da corrida.

– `pred:year ?year` - Obtém o ano da corrida.

- **GROUP_CONCAT**: Agrupa múltiplos valores numa única string, separando-os por vírgulas. Aqui, é feita a concatenação do ID da corrida com o ano, separados por “_”.
- **GROUP BY**: Agrupa os resultados pelo nome da corrida, garantindo que todas as corridas com o mesmo nome sejam combinadas corretamente.
- **LIMIT e OFFSET**: Permitem paginação dos resultados. **LIMIT** define o número máximo de resultados retornados, enquanto **OFFSET** permite saltar um número específico de resultados para obter diferentes páginas de dados.

Dessa forma, a consulta retorna uma lista de corridas e, para cada uma, uma string agregada contendo os identificadores e anos das edições dessa corrida, facilitando a análise e extração de informações a partir da base RDF.

Com os dados obtidos é feita uma transformação para enviar os dados de forma mais organizada para que a aplicação web os apresente. Onde é feito um ciclo `for` para iterar sobre os dados encontrados e organizá-los no seguinte formato:

```
[
  {
    "raceName": "Name of the Race",
    "raceDetails": [
      {
        "raceId": "URI of the Race",
        "year": "Year of the Race"
      },
      ...
    ]
  },
  ...
]
```

3.3 Corrida Específica

```
PREFIX ns: <http://pitstop.org/driver/>
PREFIX pred: <http://pitstop.org/pred/>
PREFIX type: <http://pitstop.org/type/>

SELECT *
WHERE {
  ns:{driver_id} a type:Driver ;
  pred:forename ?forename ;
  pred:surname ?surname ;
  pred:dob ?dob ;
  pred:nationality ?nationality ;
  pred:url ?url .

  OPTIONAL { ns:{driver_id} pred:number ?number . }
  OPTIONAL { ns:{driver_id} pred:code ?code . }
}
```

Esta query tem como objetivo recuperar informações detalhadas sobre um piloto específico.

- **PREFIX**: Define os espaços de nomes (*namespaces*) usados na consulta para garantir que os identificadores RDF sejam referenciados corretamente:
 - `ns:` - Define o espaço de nomes para os pilotos.
 - `pred:` - Define o espaço de nomes para os predicados.

- type: - Define o espaço de nomes para os tipos de entidades.
- **SELECT ***: Indica que a consulta retornará todas as variáveis disponíveis na cláusula **WHERE**.
- **WHERE**: Especifica os padrões que os dados devem satisfazer para um piloto específico identificado por `driver_id`:
 - ns:{driver_id} a type:Driver - Filtra apenas a entidade do tipo **Driver** correspondente ao identificador especificado.
 - pred:forename ?forename - Obtém o primeiro nome do piloto.
 - pred:surname ?surname - Obtém o apelido do piloto.
 - pred:dob ?dob - Obtém a data de nascimento do piloto.
 - pred:nationality ?nationality - Obtém a nacionalidade do piloto.
 - pred:url ?url - Obtém a URL associada ao piloto com informações mais detalhadas.
- **OPTIONAL**: Utilizado para incluir informações opcionais, que podem ou não estar presentes na base de dados:
 - OPTIONAL { ns:{driver_id} pred:number ?number . } - Se disponível, obtém o número do piloto.
 - OPTIONAL { ns:{driver_id} pred:code ?code . } - Se disponível, obtém o código do piloto.

Foi necessário incluir o optional para o número e código de um piloto visto que no dataset nem todos os pilotos apresentam estes campos com \N ou seja, não tem informação, logo é necessário verificar se estes campos existem no piloto especificado.

3.4 Resultados de uma corrida

r refer to

```
PREFIX pred: <http://pitstop.org/pred/>
PREFIX type: <http://pitstop.org/type/>
PREFIX ns: <http://pitstop.org/race/>
SELECT ?driverId ?driverName ?constructorId ?constructorName ?position ?time ?laps
WHERE {
  ?result a type:Result ;
    pred:raceId ns:{race_id} ;
    pred:driverId ?driverId ;
    pred:constructorId ?constructorId ;
    pred:position ?position .
  OPTIONAL { ?result pred:laps ?laps. }
  OPTIONAL { ?result pred:time ?time. }

  ?driverId a type:Driver ;
    pred:forename ?driverForename ;
    pred:surname ?driverSurname .
  BIND(CONCAT(?driverForename, " ", ?driverSurname) AS ?driverName)

  ?constructorId a type:Constructor ;
    pred:name ?constructorName .
}
LIMIT 3
```

Esta query tem como objetivo recuperar informações detalhadas sobre os resultados de uma corrida específica.

- **PREFIX:** Define os espaços de nomes (*namespaces*) usados na consulta para garantir que os identificadores RDF sejam referenciados corretamente
- **SELECT:** Define as variáveis que serão retornadas na consulta:
 - `?driverId` - Identificador do piloto.
 - `?driverName` - Nome completo do piloto.
 - `?constructorId` - Identificador da equipa.
 - `?constructorName` - Nome da equipa.
 - `?position` - Posição final do piloto na corrida.
 - `?time` - Tempo total do piloto na corrida (opcional).
 - `?laps` - Número de voltas completadas pelo piloto (opcional).
- **WHERE:** Especifica os padrões que os dados devem satisfazer:
 - `?result a type:Result` - Filtra apenas entidades do tipo `Result`.
 - `pred:raceId ns:{race_id}` - Seleciona apenas os resultados referentes a uma corrida específica, identificada por `race_id`.
 - `pred:driverId ?driverId` - Obtém o identificador do piloto.
 - `pred:constructorId ?constructorId` - Obtém o identificador da equipa.
 - `pred:position ?position` - Obtém a posição final do piloto na corrida.
 - `OPTIONAL { ?result pred:laps ?laps. }` - Se disponível, obtém o número de voltas completadas pelo piloto.
 - `OPTIONAL { ?result pred:time ?time. }` - Se disponível, obtém o tempo total do piloto na corrida.
- **Relacionamento com pilotos e construtores:**
 - `?driverId a type:Driver` - Garante que o identificador pertence a um piloto.
 - `pred:forename ?driverForename` e `pred:surname ?driverSurname` - Obtém respetivamente o primeiro nome e o apelido do piloto.
 - `BIND(CONCAT(?driverForename, " ", ?driverSurname) AS ?driverName)` - Concatena o primeiro nome e o apelido do piloto em uma única variável `?driverName`.
 - `?constructorId a type:Constructor` - Garante que o identificador pertence a um construtor.
 - `pred:name ?constructorName` - Obtém o nome da construtor.
- **LIMIT:** Define um limite de 3 resultados para a consulta, para obter o pódio da corrida

3.5 Todos os construtores

```
PREFIX pred: <http://pitstop.org/pred/>
PREFIX type: <http://pitstop.org/type>
SELECT ?c ?name ?nationality ?url WHERE {{
    ?c a type:Constructor ;
    pred:name ?name ;
    pred:nationality ?nationality ;
    pred:url ?url .
}}
```

```
LIMIT {LIMIT}
OFFSET {offset}
```

Esta query tem como objetivo recuperar todos os construtores existentes ordenados de forma alfabética.

- **SELECT:** Define as variáveis que serão retornadas na consulta:
 - ?c - Identificador único da equipa.
 - ?name - Nome da equipa.
 - ?nationality - Nacionalidade da equipa.
 - ?url - URL associada à equipa.
- **WHERE:** Especifica os padrões que os dados devem satisfazer:
 - ?c a type:Constructor - Filtra apenas entidades do tipo Constructor.
 - pred:name ?name - Obtém o nome da equipa.
 - pred:nationality ?nationality - Obtém a nacionalidade da equipa.
 - pred:url ?url - Obtém a URL da equipa.
- **LIMIT e OFFSET:**
 - LIMIT - Define o número máximo de resultados retornados na consulta, permitindo controlar a quantidade de dados recuperados.
 - OFFSET - Permite pagnar os resultados ao ignorar os primeiros **offset** registros e começar a exibição a partir do próximo conjunto de dados.

3.6 Procurar por Piloto

Esta query tem como objetivo permitir o utilizador pesquisar por pilotos usando o input text box

```
PREFIX pred: <{PRED}>
PREFIX type: <{TYPE}>

SELECT ?driverId ?forename ?surname ?nationality
WHERE {{
    ?driverId a type:Driver ;
    pred:forename ?forename ;
    pred:surname ?surname ;
    pred:nationality ?nationality

    FILTER regex(CONCAT(?forename, " ", ?surname), "{query}", "i") .
}}
LIMIT {LIMIT}
OFFSET {offset}
```

- **PREFIX:** Define os espaços de nomes (*namespaces*) usados na consulta para garantir que os identificadores RDF sejam referenciados corretamente
- **SELECT:** Define as variáveis que serão retornadas na consulta:
 - ?driverId - Identificador do piloto.
 - ?forname - Nome completo do piloto.
 - ?surname - Identificador da equipa.
 - ?nationality - Nome da equipa.
- **WHERE:** Especifica os padrões que os dados devem satisfazer:
 - ?driverId a type:Driver - Filtra apenas entidades do tipo Driver.
 - pred:forname ?forname - Obtém of forname do piloto.
 - pred:surname ?surname - Obtém o surname do piloto.
 - pred:nationality ?nationality - Obtém a nacionalidade do equipa.

- **FILTER**: filtra resultado da query
 - `regex(CONCAT(?forename, " ", ?surname), "query", "i")` - no filter fazemos um concat do forename com o surname e depois verificamos se query esta prensete na string concatenada.
- **LIMIT**: Limita os resultados pelo limite default usado.
- **OFFSET**: Offset usado para retornar paginação.

3.7 INSERTS e DELETES

Nós implementamos a inserção e remoção de dados para as temporadas e para as corridas, tanto na base de dados como na página web. Os DELETES apenas recebem o identificador do recurso e removem-no, juntamente com todas as suas relações. Quanto aos INSERTs, para adicionar uma temporada, basta escolher o ano. Já para adicionar uma corrida, é necessário escolher o identificador do circuito, a data, o nome, a ronda e o ano.

```
INSERT DATA {{
    season:{year} a type:Season ;
    pred:url <{url}> .
}}
```

```
DELETE {{ season:{year} ?p ?o }}
WHERE {{
    season:{year} a type:Season ;
    ?p ?o .
}}
```

4 Aplicação Web

A aplicação web foi desenvolvida em **React**, consumindo endpoints criados em **Django** para obter e manipular as informações necessárias.

A interface desenvolvida apresenta **quatro páginas principais**:

- **Drivers** – Listagem e detalhes dos pilotos.
- **Races** – Informações sobre as corridas realizadas.
- **Construtores** – Dados dos construtores.
- **Seasons** – Temporadas e estatísticas relacionadas.

4.1 Drivers

Para os driver a nossa aplicação contém duas páginas. A página Drivers List [Figure 1](#) apresenta uma lista com todos os pilotos de Fórmula 1 ordenados por nome e um input que permite pesquisar pelo nome dos pilotos [Figure 3](#). A página apresentada na [Figure 2](#) mostra os detalhes de um piloto, bem como as corridas em que participou.

	Name	Nationality	Code	Number
1	Adolf Brudes		U	U
2	Adolfo Cruz		U	U
3	Adrian Sutil		STP	99
4	Adrián Campos		U	U
5	Aguri Suzuki		U	U
6	Al Herman		U	U
7	Al Keller		U	U
8	Al Pease		U	U
9	Alain Prost		U	U
10	Alain de Changy		U	U
11	Alan Jones		U	U

Figure 1: Drivers List

Driver Details

Personal Details

First Name: Jacques
 Last Name: Villeneuve
 Date of Birth: 1971-04-09
 Code: VIL
http://en.wikipedia.org/wiki/Jacques_Villeneuve

Races Won

- Brazilian Grand Prix 1993: Points 10.0
- Spanish Grand Prix 1997: Points 10.0
- British Grand Prix 1997: Points 10.0
- Hungarian Grand Prix 1997: Points 10.0
- Austrian Grand Prix 1997: Points 10.0
- Luxembourg Grand Prix 1997: Points 10.0
- European Grand Prix 1996: Points 10.0
- British Grand Prix 1996: Points 10.0
- Hungarian Grand Prix 1996: Points 10.0

Figure 2: Drivers Details

	Name	Nationality	Code	Number
1	Levi Hamilton		U	U
2	Duncan Hamilton		U	U

Page 1

Figure 3: Drivers Search

4.2 Races

Relativamente às corridas, foi desenvolvida uma página que lista todas as corridas. Ao se clicar num botão com um ícone de uma lupa, outra página é apresentada que mostra todos os anos que a corrida aconteceu [Figure 7](#). Nesta, é possível apagar uma corrida e visualizar informações mais detalhadas sobre uma corrida [Figure 5](#). Por fim, na página que contém todas as corridas que ocorreram num determinado ano é possível criar uma nova corrida [Figure 6](#), em que para isto é necessário especificar:

- O **nome** da corrida;
 - A **data** em que ocorreu ou vai ocorrer;
 - O **ano**;
 - A **ronda**;
 - E escolher um **circuito**, dos fornecidos. Os circuitos apresentados são todos os aqueles que se encontram guardados na base de dados. Para a obtenção dos mesmos desenvolvemos uma query simples que devolve todos os circuitos.
- . Existe uma validação simples para este formulário que simplesmente verifica se todos os campos se encontram preenchidos.

	Name	Last Year	First Year
1	Argentine Grand Prix	1996	1993
2	Australian Grand Prix	2012	1985
3	Austrian Grand Prix	2018	1964
4	Belgian Grand Prix	2019	1950
5	British Grand Prix	2019	1950
6	Canadian Grand Prix	2012	1967
7	Dutch Grand Prix	2012	1952
8	European Grand Prix	2016	1983
9	French Grand Prix	2012	1950
10	German Grand Prix	2019	1951
11	Hungarian Grand Prix	2012	1986

Figure 4: Races List

Race Information		Circuit Information	
Name	Argentine Grand Prix	Name	Autódromo Juan y Oscar Gálvez
Date	1996-04-07	Location	Buenos Aires
Round	3	Coordinates	(-34.6943, -58.4593)
	http://en.wikipedia.org/wiki/1996_Argentine_Grand_Prix	Country	Argentina
			http://en.wikipedia.org/wiki/Aut%C3%B3dromo_Oscar_Alfredo_G%C3%A1lvez

Driver	Team	Number
Jean Alesi	Benetton	14
Damon Hill	Williams	15
Jacques Villeneuve	Williams	12

Drivers Podium

Figure 5: Races Details

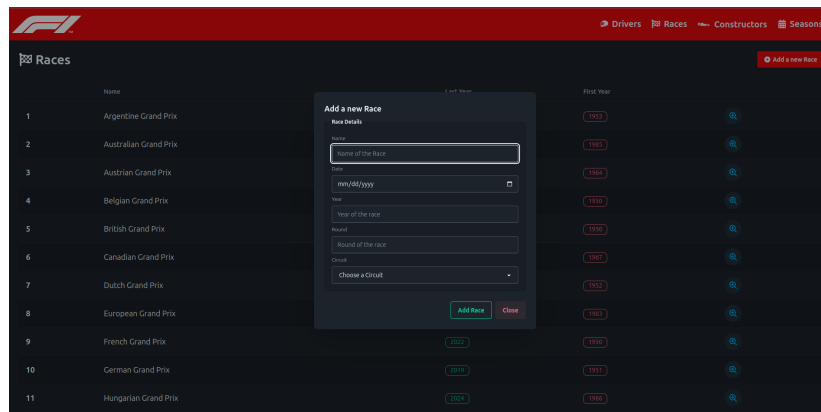


Figure 6: Races Add

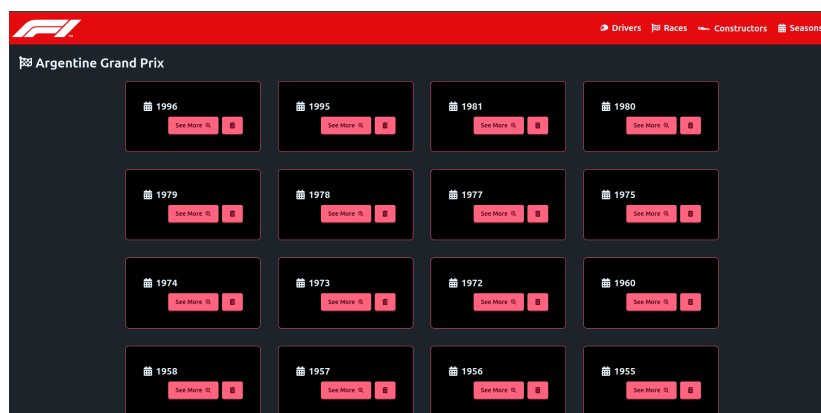


Figure 7: Races por Ano

4.3 Constructors

Para os construtores, desenvolvemos apenas uma página simples que lista os mesmos com algumas informações [Figure 8](#).

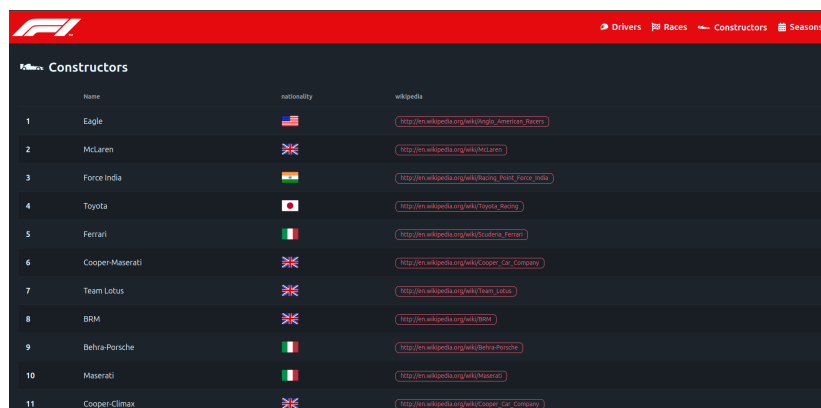
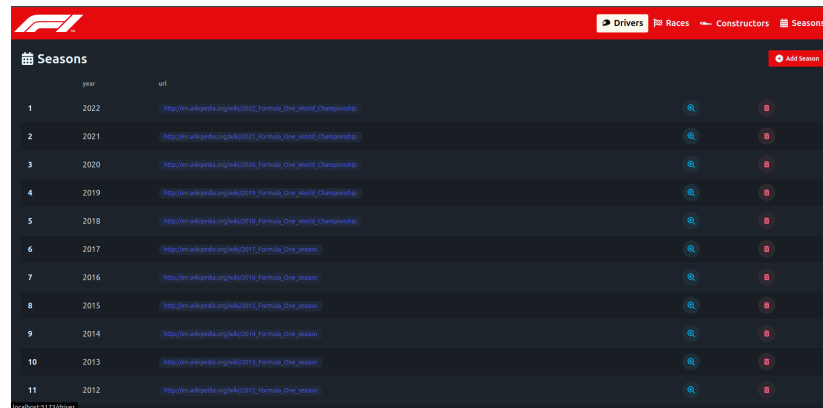


Figure 8: Constructors List

4.4 Seasons

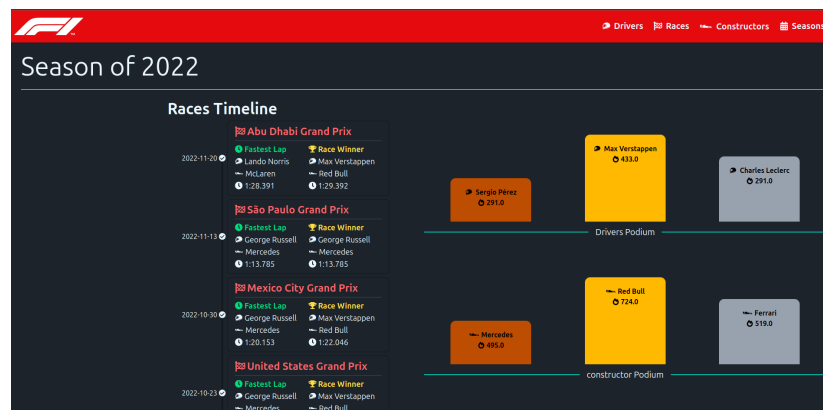
Para apresentar a entidade Épocas, 3 páginas foram criadas:

- **Épocas:** lista todas as épocas existentes [Figure 9](#)
- **Detalhes de uma época:** apresenta todos os detalhes de um época, como todas as corridas que ocorreram numa época. Para cada corrida são apresentadas informações sobre o piloto que teve a volta mais rápida e o vencedor da corrida, bem como os construtores associados. Também é possível observar os três pilotos e construtores que obtiveram mais pontos na época toda [Figure 10](#).
- **Adicionar uma nova Época:** contém um formulário simples com apenas 2 campos de texto um para o **ano** e outro para um **url**. Com a submissão deste formulário uma nova época é criada [Figure 11](#).



year	url
2022	http://en.wikipedia.org/wiki/2022_Formula_One_world_Championship
2021	http://en.wikipedia.org/wiki/2021_Formula_One_world_Championship
2020	http://en.wikipedia.org/wiki/2020_Formula_One_world_Championship
2019	http://en.wikipedia.org/wiki/2019_Formula_One_world_Championship
2018	http://en.wikipedia.org/wiki/2018_Formula_One_world_Championship
2017	http://en.wikipedia.org/wiki/2017_Formula_One_season
2016	http://en.wikipedia.org/wiki/2016_Formula_One_season
2015	http://en.wikipedia.org/wiki/2015_Formula_One_season
2014	http://en.wikipedia.org/wiki/2014_Formula_One_season
2013	http://en.wikipedia.org/wiki/2013_Formula_One_season
2012	http://en.wikipedia.org/wiki/2012_Formula_One_season

Figure 9: Seasons List



Race	Date	Fastest Lap	Race Winner	Podium
Abu Dhabi Grand Prix	2022-11-20	Lando Norris McLaren 1:28.391	Max Verstappen Red Bull 1:29.392	Sergio Pérez (291.0), Max Verstappen (433.0), Charles Leclerc (391.0)
São Paulo Grand Prix	2022-11-13	George Russell Mercedes 1:13.795	George Russell Mercedes 1:13.795	
Mexico City Grand Prix	2022-10-30	George Russell Mercedes 1:20.153	Max Verstappen Red Bull 1:22.046	Mercedes (485.0), Red Bull (724.0), Ferrari (519.0)
United States Grand Prix	2022-10-23	George Russell Mercedes	Max Verstappen Red Bull	

Figure 10: Seasons Details

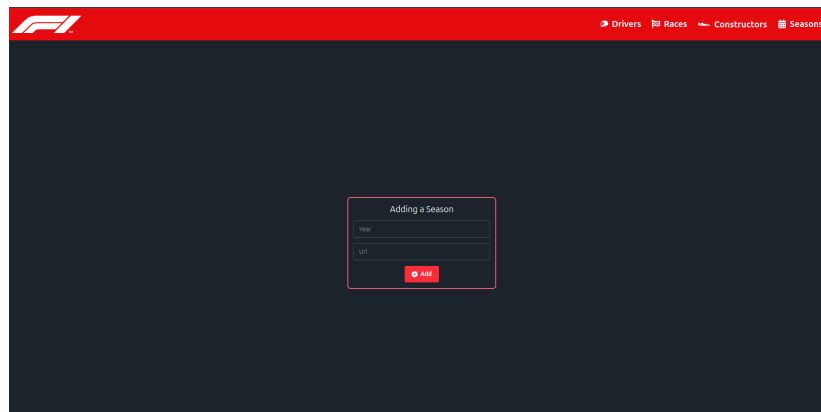


Figure 11: Seasons Add

5 Conclusão

O desenvolvimento deste trabalho permitiu-nos aprofundar os principais conceitos abordados na disciplina de Web Semântica, explorando as funcionalidades da linguagem de consulta SPARQL e do formato de dados N3.

A utilização de Django e React proporcionou um desenvolvimento modular e eficiente, aproveitando bibliotecas que facilitaram a implementação. Esta abordagem permitiu criar uma estrutura modular e organizada, garantindo uma melhor experiência na manipulação dos dados.

Além disso, este projeto reforçou a importância da Web Semântica na organização e reutilização de informação estruturada em relações. A flexibilidade do formato N3 mostrou ser importante para a adição de novos dados sem a necessidade de uma estrutura fixa.

Por fim, cumprimos os objetivos estabelecidos por nós para o trabalho, desenvolvendo queries SPARQL que variam em complexidade, desde queries simples como **SELECT**, **DELETE** e **INSERT** até operações mais avançadas com **GROUP BY**. Dessa forma, conseguimos aplicar a maioria das funcionalidades do SPARQL apresentadas em aula, melhorando o nosso conhecimento na área.

6 Organização do código

O nosso projeto apresenta diversas pastas:

- **data:** Apresenta os ficheiros *.csv* com dados relativos a Fórmula 1;
- **docs:** Apresenta o enunciado para o projeto e também o relatório final;
- **src:** Apresenta outras pastas com o código desenvolvido.

Dentro da pasta **src** encontra-se um ficheiro designado *data_converter.py* que contém toda a lógica explicada na secção 2, para a transformação dos dados em **CSV** para **RDF/N3**.

Também se encontra um ficheiro *docker-compose.yml* que define e configura três serviços dentro de um ambiente **Docker**:

- **vite(aplicação web):** utiliza o ficheiro *dockerfile* (que se encontra dentro do projeto da aplicação web) para construir uma imagem que irá conter a aplicação web. Mapeia a porta **5173** do container para a mesma porta da máquina, permitindo acesso à mesma através dessa porta. Define também dois volumes, um que monta a pasta local com o projeto da aplicação web dentro do container, permitindo que mudanças no código do frontend sejam refletidas sem necessidade de reconstrução. E outro que anula o diretório **node_modules** dentro do container, para impedir que módulos instalados dentro do container sejam sobrescritos pelo sistema de ficheiros local. Por fim uma variável de ambiente é especificada que contém um URL para a RESTAPI.

- **django (REST API):** utiliza o ficheiro *dockerfile* (que se encontra dentro do projeto da REST API). Mapeia a porta **8000** do container para a mesma porta na máquina local, permitindo acesso ao serviço através dessa mesma porta. São configuradas duas variáveis de ambiente uma que define o endpoint onde o backend se irá conectar ao **GraphDB** e outra que indica o nome do repositório dentro do **GraphDB** onde os dados RDF serão armazenados (neste caso designa-se de *fl-pitstop*). Por fim monta a pasta local que contém o código desenvolvido no container, permitindo atualizações no código sem reconstruir o mesmo.
- **graphDB (base de dados RDF):** utiliza a imagem oficial *ontotext/graphdb* na versão **10.8.3** para o GraphDB. Define o nome e hostname do container, permitindo que outros serviços o encontrem dentro da rede interna do Docker. Mapeia a porta **7200** para permitir acesso ao **GraphDB** através de uma interface web. Monta uma pasta local dentro do container no diretório */opt/graphdb/home/data*, garantindo persistência dos dados RDF mesmo que o container seja removido ou reiniciado.

Ainda se encontram 4 diretórios

- **output:** Contém o ficheiro com os dados em **RDF/N3** (*data.n3*);
- **graphdb:** Contém todos os dados relativos ao *GraphDB*. Este diretório encontra-se mapeado para o container com a imagem do *GraphDB* para que exista persistência dos dados, ou seja, qualquer configuração que seja feita/adicionada na interface do *GraphDB* esta é depois guardada neste diretório. Isto faz com que sempre que se é executado o container com a imagem do *GraphDB*, um repositório criado é criado e os dados são inseridos de forma automática;
- **fl-frontend:** Contém todo o código desenvolvido para a aplicação Web, utilizando a framework *React + Vite*;
- **fl-backend:** Contém todo o código desenvolvido para a API, utilizando a framework *Django*

6.1 Organização do projeto da aplicação web

Neste projeto encontram-se vários ficheiros e diretórios:

- **routes.jsx:** contém todas as rotas configuradas para as páginas web;
- **services/:** contém serviços que por si apresentam métodos que fazem chamadas à REST API para a obtenção de dados;
- **pages/:** apresenta o código para todas as páginas web desenvolvidas e já apresentadas neste relatório;
- **components/** apresenta o código desenvolvido para componentes que foram utilizados em diversas páginas.

Todas as dependências externas usadas, bem como as suas versões, encontram-se listadas no ficheiro *package.json*.

6.2 Organização do projeto Django

Neste projeto também se encontram diversos ficheiros e diretórios:

- **fl_pitstop/urls.py:** contém a nomenclatura de todos os endpoints disponíveis para a API, onde é especificado para cada um uma função do ficheiro *views.py*;
- **fl_pitstop/settings.py:** apresenta todas as configurações do projeto **Django**;
- **fl_pitstop/graph_db.py:** define uma classe que serve como uma interface para interagir com um repositório na **GraphDB**, permitindo executar queries e updates em SPARQL através da API do GraphDB.

- **app/views.py**: contém todas as **views** para cada endpoint, cada view comunica com um determinado serviço para evitar que exista um acesso direto à base de dados em cada view;
- **app/services/**: apresenta todos os serviços para cada entidade, em que, depois dos dados serem recuperados da base de dados, é feito um processo de transformação a estes para que contenham um formato mais organizado para enviar para a aplicação web;
- **app/repositories/**: contém repositórios por entidade, que executam **QUERIES**, **UPDATES** ou **DELETES** na base de dados.

7 Como Correr

Para executar é necessário ter instalado as frameworks **Docker** e **Docker-Compose**. A instalação encontra-se explicada através destes links, um para o [docker](#) e outro para o [docker-compose](#)

Com estas frameworks instaladas, basta executar os seguintes comandos:

```
git clone https://github.com/zegameiro/WS_First_Assignment
cd WS_First_Assignment/src
docker compose up --build
```

Os diferentes serviços estarão disponíveis através dos seguintes links:

- **frontend**: [localhost:5173](#)
- **GraphDB**: [localhost:7200](#)
- **Backend**: [localhost:8000](#)

Para executar o ficheiro que transforma os dados de **CSV** para **RDF/N3**, basta seguir os seguintes comandos:

```
cd WS_First_Assignment/src
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python3 data_converter.py
```

Quando o programa acabar de executar, o ficheiro gerado com os dados em **RDF/N3** irá encontrar-se no diretório **output/**.