



Trabalho Prático #1 | Web Semântica

# Aplicação Pokedex utilizando Web Semântica



Lia Cardoso, 107548, MEI  
Liliana Ribeiro, 108713, MEI  
Francisco Ferreira, 124467, MEI

# Índice

<b>1. Introdução ao tema</b>	<b>3</b>
<b>2. Dados, fontes e sua transformação</b>	<b>4</b>
RDF Model (Ontologia/Design do Schema)	4
Processo de Conversão	6
<b>3. Arquitetura aplicada</b>	<b>7</b>
<b>4. Operações sobre os dados (SPARQL)</b>	<b>11</b>
Construir um novo grafo entre todos os atributos de um pokemon (Construct)	14
<b>5. Funcionalidades da Aplicação (UI)</b>	<b>16</b>
Overview	16
Principais Páginas/Views	17
Página de Pesquisa (/pokemon/search/)	17
Página de Detalhes (/pokemon/stats/{id}/)	17
Outras Funcionalidades / Páginas	18
Tecnologias de Interface	18
Screenshots	18
<b>6. Conclusões</b>	<b>22</b>
<b>7. Configuração para executar a aplicação</b>	<b>23</b>

# 1. Introdução ao tema

O principal objetivo deste projeto consiste em desenvolver um sistema de informação, acessível via web, que permita aos utilizadores explorar e gerir dados de Pokémon com base em tecnologias da Web Semântica. A escolha do universo Pokémon deve-se à riqueza e à organização do respetivo conjunto de dados: cada Pokémon possui atributos bem definidos (nome, tipos, evoluções, habilidades, estatísticas), além de relacionamentos claros que favorecem uma abordagem orientada a grafos. Esta combinação de propriedades torna o Pokémon um excelente caso de estudo para demonstrar como dados interligados (Linked Data) podem oferecer novas perspetivas em comparação com tabelas estáticas ou bases de dados tradicionais.

A criação deste sistema procura solucionar o problema de fornecer uma forma estruturada e interconectada de consultar e gerir a informação de Pokémon. Em vez de se resumir a listagens textuais, o sistema relaciona ativamente as diversas facetas de cada criatura — como tipagem, fraquezas e poderes — permitindo pesquisas mais intuitivas e ligações entre Pokémon que partilham propriedades semelhantes. Adicionalmente, o projeto contempla funcionalidades que vão além de uma simples pesquisa, apresentando batalhas, filtragem avançada, visualização de estatísticas detalhadas e comparações, como de altura e peso entre Pokémons específicos.

Para alcançar estes objetivos, a abordagem adotada baseia-se nos princípios da Web Semântica. Os dados são modelados em RDF, garantindo uma representação em forma de grafo que pode ser armazenada num triplestore GraphDB. As consultas são realizadas em SPARQL, o que oferece a flexibilidade de extrair e combinar a informação de forma mais inteligente do que as típicas bases de dados relacionais. Por fim, o sistema apresenta os resultados através de uma aplicação web em Python, suportada pelo framework Django, proporcionando uma interface amigável que integra funcionalidades de pesquisa, exibição de detalhes, comparação e batalhas.

Nos capítulos seguintes do relatório, descrevem-se os processos de preparação dos dados e a sua transformação em RDF, bem como as consultas SPARQL criadas para responder a cenários de utilização específicos. Aborda-se igualmente a conceção da interface de utilizador e as diferentes secções da aplicação, incluindo o sistema de batalhas, a página de pesquisa com múltiplos filtros, a secção de detalhes de cada Pokémon e o módulo de comparação de estatísticas. Por fim, discute-se a arquitetura global do sistema, refletindo sobre as decisões tecnológicas e metodológicas que sustentam o projeto, bem como as perspetivas de evolução futura.

## 2. Dados, fontes e sua transformação

O conjunto de dados utilizado neste projeto provém de três fontes principais no Kaggle. Uma delas disponibiliza informações abrangentes sobre cada Pokémon (estatísticas, nomes, números de Pokédex, tipos, habilidades, valores de altura e peso, indicativo de status lendário, entre outros atributos) - disponível em <https://www.kaggle.com/datasets/mariotormo/complete-pokemon-dataset-updated-090420> - , originalmente em formato CSV. As outras incluem imagens associadas aos diferentes Pokémon - aos quais se podem aceder através de <https://www.kaggle.com/datasets/kvpratama/pokemon-images-dataset> e <https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types> - , possibilitando uma dimensão mais visual ao sistema. Graças a esta combinação, foi possível estruturar tanto os aspetos numéricos e textuais (como estatísticas de batalha e descrições) quanto as representações gráficas das criaturas. Em termos práticos, a existência de entidades bem definidas (Pokémon, tipos, habilidades, formas de evolução) e dos relacionamentos entre elas fez com que o dataset fosse especialmente adequado à abordagem por grafos, típica da Web Semântica. O facto de o Pokémon ser um tema amplamente documentado, com dados consistentes e relacionamentos claros, reforçou ainda mais a escolha deste universo.

A base de dados de texto continha diversas colunas que iam desde o nome e número do Pokémon na Pokédex até atributos como HP, Attack, Defense, Special Attack, Special Defense, Speed, altura, peso, se é ou não lendário, e indicações sobre tipos primário e secundário. Alguns destes campos eram cruciais para o mapeamento em RDF, pois representam os predicados e objetos na camada semântica. Além disso, a informação de evolução (nome do Pokémon em que evolui) e de habilidades (por exemplo, ability1, ability2) forneceu a oportunidade de criar relacionamentos mais ricos do que as clássicas colunas em tabelas. Nesse sentido, este dataset demonstrou-se ideal porque disponibiliza um conjunto de entidades e atributos que podem ser diretamente traduzidos para classes, propriedades e valores literais em RDF, permitindo ainda a interligação dos dados de forma a evidenciar aspetos como fraquezas, eficácia defensiva/ofensiva e ligações evolutivas.

### RDF Model (Ontologia/Design do Schema)

Para representar os dados em formato RDF, optou-se por usar a sintaxe Turtle (ficheiros .ttl), que é relativamente legível e fácil de editar ou inspecionar manualmente. Contudo, a mesma modelagem poderia ser serializada em RDF/XML, JSON-LD ou N3 sem afetar a estrutura semântica. O ponto central da ontologia reside na classe de Pokémon (por exemplo, ns1:Pokemon), que agrupa cada criatura identificada por um URI único. À semelhança disso, cada tipo recebeu uma classe e instâncias próprias (ns1:Type), enquanto

as habilidades (caso fossem tratadas como entidades separadas) ganharam a classe ns1:Ability.

Definiram-se diversas propriedades para mapear os campos originais do CSV para a ontologia. Muitos atributos, como altura, peso e estatísticas de batalha, foram tratados como data properties (literal), por exemplo ns1:height (com xsd:float), ns1:hp (xsd:integer), ns1:isLegendary (xsd:boolean) e assim por diante. Os relacionamentos relevantes — como o facto de um Pokémon ter um tipo primário ou secundário — foram modelados por object properties. Por exemplo, ns1:primaryType e ns1:secondaryType ligam instâncias de ns1:Pokemon a instâncias de ns1:Type, valorizando a componente semântica em vez de utilizar meros nomes literais. A mesma lógica foi aplicada às habilidades (ns1:ability1, ns1:ability2), que ligam um Pokémon a uma entidade do tipo ns1:Ability.

No que respeita às evoluções, recorreu-se a uma propriedade como ns1:evolvesTo que aponta de um Pokémon para outro. A informação de efetividade (fraquezas e resistências) foi encapsulada usando nós em branco (blank nodes), ligados a cada Pokémon por meio de ns1:effectiveness. Este nó em branco tem propriedades como ns1:againstFire, ns1:againstWater, etc., guardando valores numéricos (xsd:float) que indicam quão eficaz (ou ineficaz) é determinado tipo ao atacar aquele Pokémon.

Um pequeno fragmento em Turtle ajuda a ilustrar este modelo. Por exemplo, para o Bulbasaur:

```
<http://example.org/pokemon/Pokemon/0> a ns1:Pokemon ;
  ns1:ability1 <http://example.org/pokemon/Ability/overgrow> ;
  ns1:attack 49 ;
  ns1:baseFriendship 70 ;
  ns1:defense 49 ;
  ns1:effectiveness [
    ns1:againstFire "2.0"^^xsd:float ;
    ns1:againstFlying "2.0"^^xsd:float ;
    ns1:againstPsychic "2.0"^^xsd:float ;
    ns1:againstWater "0.5"^^xsd:float
  ] ;
  ns1:generation 1 ;
  ns1:hp 45 ;
  ns1:isLegendary false ;
  ns1:primaryType <http://example.org/pokemon/Type/grass> ;
  ns1:secondaryType <http://example.org/pokemon/Type/poison> ;
  schema:name "Bulbasaur" .
```

Neste exemplo, Bulbasaur é identificado pelo URI

<http://example.org/pokemon/Pokemon/0>, tem ns1:primaryType apontando para <http://example.org/pokemon/Type/grass>, tal como ns1:secondaryType aponta para <http://example.org/pokemon/Type/poison>, além das suas estatísticas e da propriedade

ns1:isLegendary false. Ao usar URIs em vez de literais para tipos e habilidades, torna-se simples anexar mais informações a esses recursos sem precisar alterar a estrutura central do Pokémon.

## Processo de Conversão

A conversão dos dados do CSV do Kaggle para RDF foi feita maioritariamente através de um script em Python que utilizou bibliotecas como pandas (para leitura e manipulação de dados) e rdflib (para geração dos triplos RDF). Inicialmente, cada Pokémon foi identificado com base numa coluna id (por vezes “Unnamed: 0” ou outra similar), garantindo URIs exclusivos para instâncias de ns1:Pokemon (por exemplo, `http://example.org/pokemon/Pokemon/0` para o primeiro Pokémon na lista). Em seguida, foram mapeadas as colunas do CSV para as propriedades definidas na nossa ontologia. Enquanto atributos como altura, peso, estatísticas de ataque e defesa assumiram a forma de literais do tipo `xsd:float` ou `xsd:integer`, as colunas referentes a tipo, habilidades e evoluções deram origem a propriedades de ligação para outras entidades (classes ns1:Type, ns1:Ability e instâncias de ns1:Pokemon).

Ao longo deste processo, foi necessário limpar dados inconsistentes, como entradas vazias ou com valores “nan”. Determinou-se uma forma de tratar dados duplicados, filtrar Pokémon com informação em falta, e criar um esquema de URIs que permanecesse coeso, por exemplo para tipos — `http://example.org/pokemon/Type/{typeName}` — e habilidades — `http://example.org/pokemon/Ability/{abilityName}`. Em relação ao outro dataset de imagens, foram consideradas principalmente referências para associar cada Pokémon à sua imagem correspondente, caso se pretendesse expandir a ontologia e acrescentar uma propriedade como ns1:hasImage.

Uma das maiores dificuldades residiu em manter a coerência entre as diferentes versões do CSV, pois havia colunas que mudavam ligeiramente de nome ou dados que divergiam (por exemplo, Pokémon com forms adicionais). Foi então realizado um processo de unificação dos ficheiros (`pokedex_merged.csv`, `pokedex_final.csv`, etc.) para obter um único CSV robusto. Além disso, foi importante decidir como lidar com casos em que um Pokémon tinha mais de um tipo ou mais de uma habilidade, o que resultou em propriedades repetidas como ns1:ability1 e ns1:ability2. Também houve o cuidado de garantir que cada blank node, usado para a secção de efetividade, agregasse corretamente os pares ns1:againstX “valor”^^xsd:float.

No final, o script Python exportou todos os triplos para um ficheiro Turtle, fácil de carregar num triplestore GraphDB e de consultar via SPARQL. Com isso, construiu-se um sistema completo, que não apenas representava a informação dos Pokémon de forma semanticamente rica, mas também oferecia a possibilidade de ser estendido com novas classes, propriedades ou relacionamentos, caso se venham a descobrir outros dados úteis do universo Pokémon.

### 3. Arquitetura aplicada

A arquitetura Django conhecida como MVC (Model-View-Controller) no desenvolvimento web é, neste contexto, adaptada para o padrão MVT (Model-View-Template). Nesta abordagem, o Model representa a estrutura dos dados e a interação com a base de dados, o Template é responsável pela apresentação da informação ao utilizador, e a View funciona como a camada de business logic que interage com o modelo e seleciona o template e forms apropriado.

Para além da arquitetura MVT usamos outros ficheiros importantíssimos para manter uma estrutura organizada no código Django. A configuração das rotas é feita através do ficheiro `urls.py`, que define os caminhos das URLs e associa-os às respetivas views. Para aceder aos dados semânticos usamos um ficheiro `sparql.py`, que centraliza as queries SPARQL necessárias para consultar o triplestore. Por outro lado, quando é necessário lidar com formulários HTML de interação com o utilizador, o ficheiro `forms.py` permite a definição estruturada de formulários, com validação automática e integração com os modelos.

Esta separação de responsabilidades promove uma maior organização, reutilização de código e facilita a manutenção da aplicação.

#### Model

Queries de request ao GraphDB Select, Ask, Describe, Construct Insert, Delete

```
@staticmethod
def search_by_name_and_type(name_filter, type_filter):
    escaped_name = name_filter.replace("'", '\\\'')

    query = f"""
PREFIX ex: <http://example.org/pokemon/>
PREFIX sc: <http://schema.org/>

SELECT ?pokemon ?name ?number ?primaryType ?secondaryType
?totalPoints ?megaOf WHERE {{
    ?pokemon a ex:Pokemon ;
        sc:name ?name ;
        ex:pokedexNumber ?number ;
        ex:primaryType ?primaryType ;
        ex:totalPoints ?totalPoints .
    OPTIONAL {{ ?pokemon ex:secondaryType ?secondaryType }}
    OPTIONAL {{ ?pokemon ex:megaEvolutionOf ?megaOf }}

    FILTER (
```



```
CONTAINS(LCASE(?name), LCASE("{escaped_name}")) &&
(
    LCASE(STR(?primaryType)) =
"http://example.org/pokemon/type/{type_filter}" ||
    LCASE(STR(?secondaryType)) =
"http://example.org/pokemon/type/{type_filter}"
)
)
}} ORDER BY ?number
"""

results = run_query(query)
return PokemonManager._parse_results(results)
```

## View

Trata da lógica de negócio da aplicação

```
def search_pokemon(request):
    form = PokemonSearchForm(request.GET or None)
    name_filter = request.GET.get("name", "").strip()
    type_filter = request.GET.get("type", "").strip().lower()
    sort_option = request.GET.get("sort", "")
    pokemons = []

    # Se o formulário for válido, atualiza o name_filter (caso use o
    campo "name" do form).
    if form.is_valid():
        name_filter = form.cleaned_data.get('name', '').strip()

    # Lógica de busca combinando nome e tipo
    if name_filter and type_filter:
        pokemons = PokemonManager.search_by_name_and_type(name_filter,
type_filter)
    elif name_filter:
        pokemons = PokemonManager.search_by_name(name_filter)
    elif type_filter:
        pokemons = PokemonManager.search_by_type(type_filter)
    else:
        pokemons = PokemonManager.get_all_pokemons()
```





```
context = {
    'form': form,
    'name_filter': name_filter,
    'active_type': type_filter,
    'sort_option': sort_option,
    'pokemon_types': POKEMON_TYPES,
}

return render(request, 'pokemon_search_form.html', context)
```

## Template

### Html + CSS para desenvolvimento do frontend

```
<div class="row mb-4 align-items-center filter-sort-section">
    {# Type Filters - Left/Top #}
    <div class="col-lg-9 col-md-8 type-filters">
        <h5 class="filter-title me-3">Filter by type:</h5>
        <div class="type-list">
            {# "All Types" Button - Active when no type is selected #}

            <a href="{% url 'search_pokemon' %}?name={{
name_filter|urlencode }}&sort={{ sort_option }}"
                class="type all-types {% if not active_type
}%}active{% endif %}">
                All
            </a>
            {# Buttons for each specific type #}
            {% for t in pokemon_types %}
                <a href="?type={{ t|urlencode }}{% if name_filter
}%}&name={{ name_filter|urlencode }}{% endif %}&sort={{ sort_option }}"
                    class="type {{ t }} {% if t == active_type
}%}active{% endif %}">
                    {{ t|title }}
                </a>
            {% endfor %}
        </div>
    </div>
```



## Forms

Definir os formulários de interação com o utilizador

```
class PokemonSearchForm(forms.Form):  
    name = forms.CharField(label='Pesquisar por nome', max_length=100,  
required=False)
```

## Urls

Definição dos urls de request à aplicação

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('', search_pokemon, name='search_pokemon'),  
    path('pokemon/stats/<int:pokemon_id>/', pokemon_stats,  
name='pokemon_stats'),  
    path('battle/', pokemon_selection_view, name='battle'),  
    path('battle/', pokemon_selection_view, name='pokemon_selection'),  
    path('battle/<str:pokemon1_id>/<str:pokemon2_id>/',  
pokemon_battle_view, name='pokemon_battle'),  
    path('fight/delete/<str:battle_id>/', delete_battle_view,  
name='delete_battle'),  
    path('compare/', compare_and_select_pokemon,  
name='compare_pokemon'),  
    path("pokemon/ask/", ask_pokemon_question, name="ask_question"),  
    path('pokemon/export/id/<int:pokemon_id>/', export_pokemon_rdf,  
name='export_pokemon_rdf_by_id'),  
    path('evolution/', all_evolution_chains, name='evolution_chain')  
]
```

## Sparql

Definição do método de request ao Sparql e formatos (Json, N3, Turtle)

```
def run_query_describe(query):  
    sparql = SPARQLWrapper(settings.GRAPHDB_ENDPOINT)  
    sparql.setQuery(query)  
    sparql.setReturnFormat('n3')  
    return sparql.query().convert().decode("utf-8")
```

## 4. Operações sobre os dados (SPARQL)

### Papel do SPARQL e do GraphDB

Para a Poked-X, o GraphDB foi escolhido como triplestore para armazenar o grafo RDF que representa os dados do domínio (neste caso, informações sobre Pokémon). O SPARQL é a linguagem utilizada para interrogar e, quando necessário, manipular esses dados. Através de consultas SPARQL, é possível extrair informação complexa do grafo, como relações entre entidades, filtros específicos ou dados agregados, de forma eficiente e semântica.

### Integração com Django

A integração entre a aplicação Django e o GraphDB é realizada através do endpoint SPARQL HTTP disponibilizado pelo GraphDB, acessível via <http://graphdb:7200/repositories/Poked-X>. As queries são executadas a partir da aplicação Django utilizando a biblioteca SPARQLWrapper, que permite o envio de pedidos SPARQL via HTTP.

### Queries

Ao longo deste projeto, procuramos abordar de forma abrangente os tópicos lecionados na cadeira de Web Semântica, com especial ênfase na aplicação prática dos diferentes tipos de queries SPARQL. Conseguimos atingir este objetivo ao implementar e testar consultas dos tipos SELECT, ASK, DESCRIBE e CONSTRUCT, bem como operações de manipulação de dados como INSERT e DELETE. Abaixo apresentam-se alguns exemplos da utilização destas queries no contexto da nossa aplicação.

### Consulta para a página principal de pesquisa (Select)

Esta query serve para **buscar os Pokémon** a apresentar na página de pesquisa, com suporte para **filtros por nome e tipo**, e ordenação por pokedex Number.

```
@staticmethod
```



```
def search_by_name_and_type(name_filter, type_filter):
    escaped_name = name_filter.replace("'", '\\\'')

    query = f"""
PREFIX ex: <http://example.org/pokemon/>
PREFIX sc: <http://schema.org/>

SELECT ?pokemon ?name ?number ?primaryType ?secondaryType
?totalPoints ?megaOf WHERE {{
    ?pokemon a ex:Pokemon ;
              sc:name ?name ;
              ex:pokedexNumber ?number ;
              ex:primaryType ?primaryType ;
              ex:totalPoints ?totalPoints .
    OPTIONAL {{ ?pokemon ex:secondaryType ?secondaryType }}
    OPTIONAL {{ ?pokemon ex:megaEvolutionOf ?megaOf }}

    FILTER (
        CONTAINS(LCASE(?name), LCASE("{escaped_name}")) &&
        (
            LCASE(STR(?primaryType)) =
"http://example.org/pokemon/type/{type_filter}" ||
            LCASE(STR(?secondaryType)) =
"http://example.org/pokemon/type/{type_filter}"
        )
    )
}} ORDER BY ?number
"""

    results = run_query(query)
    return PokemonManager._parse_results(results)
```

## Perguntas sobre os Pokemons (Ask)

Com esta query é possível perguntar ao website se um pokemon tem um certo atributo (ex. se o pikachu é da segunda geração).

```
@staticmethod
def ask_question_about_pokemon(pokemon_name, property_uri,
value_uri):
    n = pokemon_name.replace("'", '\\\'')
    if property_uri.endswith("generation"):
```



```
query = f"""
PREFIX ex: <http://example.org/pokemon/>
PREFIX sc: <http://schema.org/>
ASK {{
    ?pokemon a ex:Pokemon ;
             sc:name "{n}" ;
             ex:generation {value_uri} .
}}
"""

result = run_query(query)
return result.get("boolean", False)
```

## Obter todos os atributos associados ao Pokémon (Describe)

Esta querie é usada na batalha de pokémons para buscar toda a informação de cada um dos pokemons envolvidos na batalha. A utilização desta querie deve-se ao facto de se necessitar de toda a informação do pokemon, mas também das relações com o nó vazio da effectiveness para posteriormente usar nos cálculos do dano em batalha.

```
def _get_pokemon_data(self, pokemon_id):
    """Fetch Pokémon data from the SPARQL endpoint using DESCRIBE
    query (N3 format)"""
    query = f"""
DESCRIBE <http://example.org/pokemon/Pokemon/{pokemon_id}>
    """

    n3_data = run_query_describe(query)
    g = Graph()
    g.parse(data=n3_data, format="n3")

    subject_uri = f"http://example.org/pokemon/Pokemon/{pokemon_id}"
    pokemon_data = {
        "id": pokemon_id,
        "effectiveness": {}
    }

    effectiveness_node = None

    # (...)
```



```
# Process effectiveness data
    if effectiveness_node:
        for s2, p2, o2 in g:
            if str(s2) == effectiveness_node and "against" in
str(p2):
                type_name =
str(p2).split("/")[-1].replace("against", "").lower()
                pokemon_data["effectiveness"][type_name] = float(o2)

    return pokemon_data
```

Construir um novo grafo entre todos os atributos de um pokemon  
(Construct)

Esta novo grafo serve para no site o utilizador dar download a todos os dados de um  
pokemon.

```
@staticmethod
def get_construct_rdf_query_by_id(pokemon_id):
    uri = f"<http://example.org/pokemon/Pokemon/{pokemon_id}>"
    return f"""
PREFIX ex: <http://example.org/pokemon/>
PREFIX sc: <http://schema.org/>

CONSTRUCT {{
    {uri} a ex:Pokemon ;
        sc:name ?name ;

    (...)

    ?effNode
        ex:againstBug ?againstBug ;

    (...)

}}
WHERE {{
    BIND ({uri} AS ?pokemon)
    ?pokemon a ex:Pokemon ;
        sc:name ?name ;
```

```
(...)  
  
OPTIONAL {{  
    ?pokemon ex:effectiveness ?effNode .  
    OPTIONAL {{ ?effNode ex:againstBug ?againstBug }}  
  
    (...)  
  
}}  
}}  
"""
```

## Inserir e Apagar o histórico de batalhas

A aplicação tem também como funcionalidade manter um histórico de batalhas, para isto são criadas novas entradas de triplos na triplestore e apagados caso o utilizador o deseje.

```
def save_battle_result(pokemon1, pokemon2, winner):  
    """Salva o resultado da batalha no banco de dados RDF usando SPARQL  
    INSERT."""  
  
    battle_id = f"battle_{datetime.now().strftime('%Y%m%d%H%M%S')}"  
    battle_uri = f"http://example.org/pokemon/Battle/{battle_id}"  
  
    insert_query = f"""  
PREFIX poke: <http://example.org/pokemon/>  
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>  
  
INSERT DATA {{  
    <{battle_uri}> rdf:type poke:Battle ;  
        poke:date "{datetime.now().isoformat()}"^^xsd:dateTime ;  
        poke:pokemon1  
<http://example.org/pokemon/Pokemon/{pokemon1['id']}> ;  
        poke:pokemon2  
<http://example.org/pokemon/Pokemon/{pokemon2['id']}> ;  
        poke:winner  
<http://example.org/pokemon/Pokemon/{winner['id']}> ;  
        poke:pokemon1Name "{pokemon1['name']}" ;  
        poke:pokemon2Name "{pokemon2['name']}" ;  
        poke:winnerName "{winner['name']}" .  
    }}  
    """
```



```
"""

try:
    result = run_update(insert_query)
    print(f"Insert result: {result}")
    return True
except Exception as e:
    print(f"Error saving battle result: {e}")
    return False

def delete_battle(battle_id):
    """Remove uma batalha específica do RDF."""
    battle_uri = f"http://example.org/pokemon/Battle/{battle_id}"

    delete_query = f"""
PREFIX poke: <http://example.org/pokemon/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

DELETE WHERE {{
    <{battle_uri}> ?p ?o .
}}
"""

    try:
        result = run_update(delete_query)
        print(f"Deleted battle {battle_id}: {result}")
        return True
    except Exception as e:
        print(f"Error deleting battle {battle_id}: {e}")
        return False
```

## 5. Funcionalidades da Aplicação (UI)

### Overview

A aplicação web foi concebida para oferecer aos utilizadores uma forma simples, mas completa, de explorar dados de Pokémon. Partindo de uma perspetiva de utilizador final, é



possível efetuar pesquisas por nome, filtrar resultados pelo tipo desejado, ordenar a lista de Pokémon conforme diversos critérios (como ID ou nome), aceder a páginas de detalhe com estatísticas e atributos específicos, e até mesmo simular batalhas entre Pokémon.

## Principais Páginas/Views

### Página de Pesquisa (/pokemon/search/)

A página de pesquisa é o ponto de entrada principal para o utilizador. Na parte superior, encontra-se um cabeçalho ou secção de destaque onde geralmente aparece o título ou logótipo da aplicação e, logo abaixo, um campo de pesquisa para introduzir o nome (ou parte do nome) de um Pokémon. Há também uma área de filtros de tipo, na qual botões correspondentes aos diversos tipos (Fire, Water, Grass, etc.) permitem restringir a lista de resultados. Um botão “All” restaura a vista para exibir todos os Pokémon disponíveis.

Além disso, esta página oferece um menu de ordenação (ou “sort dropdown”) que o utilizador pode usar para ordenar os resultados por ID, nome (ascendente ou descendente), e por vezes valores de EXP ou outras estatísticas. Assim que o utilizador seleciona um método de ordenação ou insere um texto no campo de pesquisa, a aplicação atualiza a grelha de Pokémon exibidos. Esta grelha apresenta “cartões” individuais para cada Pokémon, incluindo imagem, número da Pokédex, nome, EXP (caso exista) e badges de tipo (primário e secundário). Se existir um número elevado de resultados, a página de pesquisa implementa paginação, permitindo a navegação entre várias páginas de listagem.

### Página de Detalhes (/pokemon/stats/{id}/)

Quando o utilizador clica num cartão de Pokémon na página de pesquisa, é direcionado para a página de detalhes. Aqui, visualiza-se uma informação muito mais completa sobre o Pokémon selecionado. Isso geralmente inclui a imagem (podendo ser estática ou animada), o nome, o número da Pokédex, a altura, o peso, a geração, o estatuto de Pokémon lendário (ou não) e as estatísticas de combate (HP, Attack, Defense, Sp. Attack, Sp. Defense, Speed), habitualmente representadas com barras de preenchimento ou gráficos em radar. Se existirem dados adicionais sobre a evolução, o utilizador poderá ver se o Pokémon evolui para outro e qual o seu sucessor na linha evolutiva (por exemplo, Bulbasaur → Ivysaur → Venusaur). Caso existam propriedades de efetividade — isto é, o quanto o Pokémon sofre dano quando enfrenta tipos específicos (Fire, Water, etc.) — elas surgem sob a forma de valores numéricos, geralmente armazenados como nós em branco no RDF e exibidos na interface de forma clara.

Do ponto de vista técnico, esta página obtém as informações específicas por meio de consultas SPARQL, usando o ID (indicado na rota /stats/{id}/) para filtrar as triplas correspondentes no triplestore. Isso reforça a flexibilidade do sistema, na medida em que não depende de tabelas rígidas, mas sim de dados ligados (Linked Data).

## Outras Funcionalidades / Páginas

Para além das páginas de pesquisa e detalhe, a aplicação pode dispor de outras secções.

- **battle/** onde o utilizador seleciona dois Pokémon para serem confrontados virtualmente, recorrendo às respetivas estatísticas de ataque, defesa e HP para determinar o vencedor.
- **pokemon/ask/** Outra vista comum é a de perguntas com base em SPARQL (/pokemon/ask/), que permite ao utilizador questionar a base semântica sobre aspetos como “O Pikachu é do tipo Electric?” e obter um resultado verdadeiro ou falso consoante os dados existentes no RDF.
- **compare/** onde o utilizador pode escolher até cinco pokemons para compara a sua altura
- **pokemon/export/id/<int:pokemon\_id>/** onde o utilizador pode exportar toda a informação reativa a um pokemon
- **evolution/** onde o utilizador pode visualizar um grafo completo com todas as evoluções de todos os pokemons

## Tecnologias de Interface

A camada de apresentação (UI) fundamenta-se principalmente em HTML e CSS para estruturar e estilizar cada página. O projeto recorre ainda ao framework Django no lado do servidor, que facilita a geração de templates dinâmicos — por exemplo, o Django carrega os dados do RDF via consultas SPARQL e, em seguida, insere-os em cada template para renderizar a UI. Para algumas funcionalidades de estilização e layout, é comum usar componentes do Bootstrap 5, garantindo uma grelha responsiva, botões e dropdowns com comportamento padrão e um design coeso. No lado do cliente, JavaScript cumpre um papel importante para funcionalidades interativas, como a filtragem instantânea por tipos, ordenação dinâmica ou renderização de gráficos (usando bibliotecas como Chart.js). Desta forma, conjuga-se a robustez de Django no back-end, o poder expressivo de SPARQL e RDF, e a flexibilidade de HTML, CSS e JavaScript para proporcionar uma experiência fluida.

## Screenshots

Para ilustrar a interface e todas estas funcionalidades, são incluídas várias capturas de ecrã no relatório. Uma delas mostra a página de pesquisa, realçando a secção de filtros de tipo e tipos de ordenação dos cards e a barra de pesquisa. Outra demonstra a página de detalhes, evidenciando as barras de estatísticas, o número da Pokédex, altura, peso e uma representação das fraquezas e forças do Pokemon.

POKED-X

POKEMON DATABASE

SEARCH

FIGHT

ASK

COMPARE

ABOUT US

Poked-X

All you need to know about pokemons!

Search Pokémon by name...

Filter by type:

All

Normal

Fire

Water

Electric

Grass

Ice

Fighting

Poison

Ground

Psychic

Dragon

Dark

Steel

Fairy

Sort by: ID Asc

#001

Bulbasaur

EXP: 200

Grass

Poison

#002

Ivysaur

EXP: 400

Grass

Poison

#003

Mega Venusaur

EXP: 625

Grass

Poison

#005

Charmander

EXP: 300

Fire

#006

Charmeleon

EXP: 600

Fire

#008

Mega Charizard X

EXP: 634

Fire

Dragon

#009

Mega Charizard Y

EXP: 634

Fire

Dragon

#010

Charizard

EXP: 534

Fire

Dragon

#016

Squirtle

EXP: 314

Water

#018

Wartortle

EXP: 620

Water

#019

Mega Blastoise

EXP: 630

Water

#029

Staryu

EXP: 100

Water

#030

Staryu

EXP: 100

Water

#031

Staryu

EXP: 100

Water

#032

Staryu

EXP: 100

Water

#033

Staryu

EXP: 100

Water

#034

Staryu

EXP: 100

Water

POKED-X

POKEMON DATABASE

SEARCH

FIGHT

ASK

COMPARE

ABOUT US

Poked-X

All you need to know about pokemons!

Bulbasaur

1

Grass

Height: 0.7 m

Weight: 6.9 kg

Generation: 1

Legendary: No

Secondary Type: Poison

HP

Attack

Defense

Sp. Attack

Sp. Defense

Speed

Export RDP

Back to Search

Stats Radar

Bulbasaur Stats

HP

Attack

Defense

Sp. Attack

Sp. Defense

Speed

Type Matchups

Weak Against

Fire

Flying

Ice

Psychic

Strong Against

Electric

Fairy

Fight

Grass

Water

Seguem-se as páginas de simulação de batalhas entre dois pokémons escolhidos pelo utilizador e a página de comparação de alturas entre pokemons, maisuma vez seleccionados pelos utilizadores. Além disto, a página de batalhas também possui um histórico das mesmas.



**POKED-X**  
POKEMON DATABASE

SEARCH

FIGHT

ASK

COMPARE


ABOUT US

Choose Your Pokemon

First Pokemon

Pikachu


Selected: Pikachu



Second Pokemon

Charmander

Selected: Charmander



Start Battle!

**POKED-X**  
POKEMON DATABASE

SEARCH

FIGHT

ASK

COMPARE

ABOUT US

A wild Pokemon appeared...

→ Avançar

Health: 35  
Attack: 55  
Defense: 40

Pikachu

Health: 39  
Attack: 52  
Defense: 43

Charmander

FIGHT!

Pikachu abilities

electric

nan

static

None

Charmander abilities

fire

nan

blaze

None

Battle Historic

No previous battles.

**POKED-X**  
POKEMON DATABASE

SEARCH

FIGHT

ASK





COMPARE

ABOUT US





Compare Pokémon Sizes

Search Pokemon...

Compare



Size Comparison



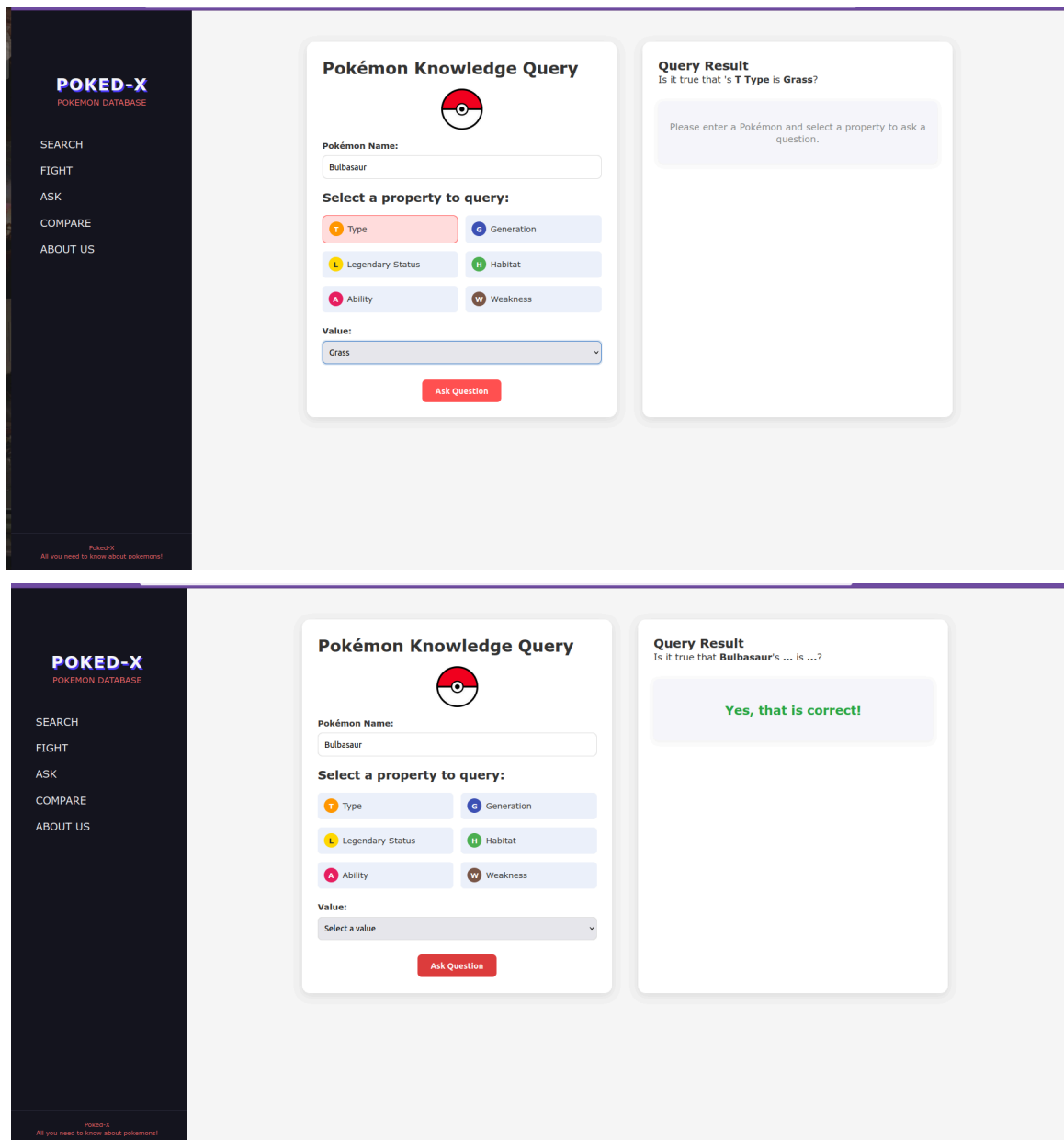
#99 Onix  
Height: 8.8 m  
Weight: 210.0 kg

#131 Lapras  
Height: 2.5 m  
Weight: 220.0 kg

#172 Pichu  
Height: 0.3 m  
Weight: 2.0 kg

#63 Abra  
Height: 0.9 m  
Weight: 19.5 kg

Segue-se uma captura de ecrã com a página de “Pokemon Knowledge Query”, onde os utilizadores podem introduzir um nome de um pokémon e realizar perguntas de “Verdadeiro ou Falso” acerca do mesmo.

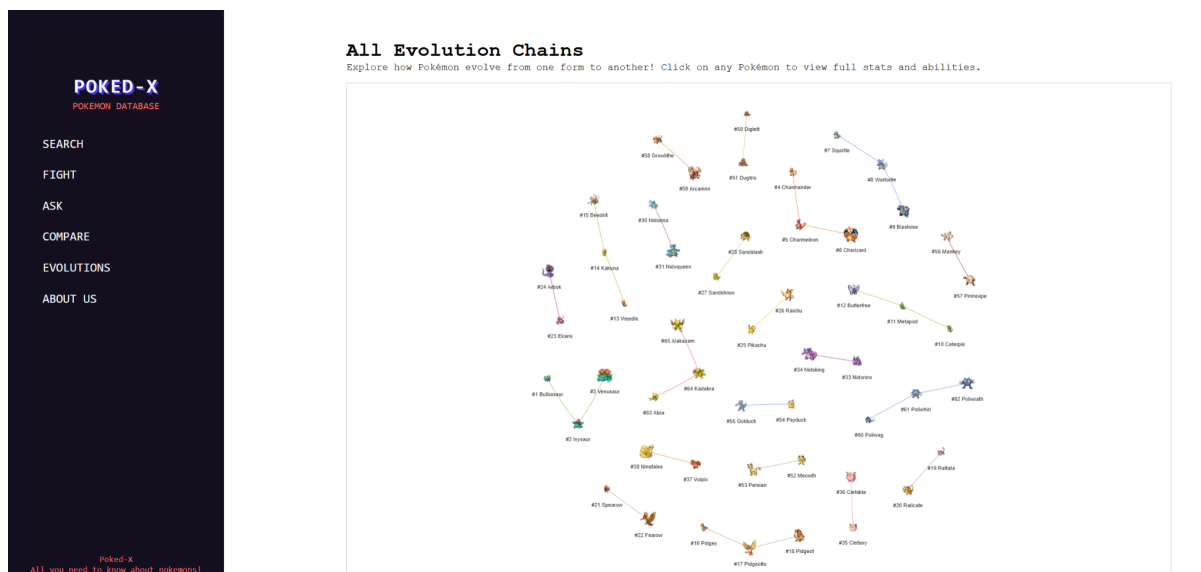


The image displays two screenshots of the "Pokémon Knowledge Query" web application interface. Both screenshots feature a dark sidebar on the left with the "POKED-X" logo and navigation links: SEARCH, FIGHT, ASK, COMPARE, and ABOUT US. The main content area is divided into two panels.

**Top Screenshot:** The "Pokémon Knowledge Query" panel shows the "Pokémon Name" field filled with "Bulbasaur". Under "Select a property to query:", the "Type" button is selected. The "Values:" dropdown menu is set to "Grass". An "Ask Question" button is at the bottom. The "Query Result" panel on the right displays the question: "Is it true that 's T Type is Grass?" and a placeholder text: "Please enter a Pokémon and select a property to ask a question."

**Bottom Screenshot:** The "Pokémon Knowledge Query" panel shows the "Pokémon Name" field filled with "Bulbasaur". Under "Select a property to query:", the "Generation" button is selected. The "Values:" dropdown menu is set to "Select a value". An "Ask Question" button is at the bottom. The "Query Result" panel on the right displays the question: "Is it true that Bulbasaur's ... is ...?" and a green response: "Yes, that is correct!"

E por último, temos a página das evoluções onde podemos observar num formato de grafo as evoluções dos pokémons de forma interativa, podendo também aceder à página de stats de um pokémon à escolha.



## 6. Conclusões

O projeto desenvolvido cumpriu com sucesso o objetivo inicial de criar uma aplicação web baseada em tecnologias da Web Semântica para gestão e exploração estruturada de dados sobre Pokémon. Através da conversão eficaz do dataset Kaggle em RDF, da utilização de SPARQL para consultas avançadas, e da integração destes elementos numa interface web amigável construída com Python e Django, conseguimos demonstrar o potencial das tecnologias semânticas para além das bases de dados tradicionais.

Entre os principais feitos destacam-se a modelação rigorosa e semântica dos dados em RDF, possibilitando consultas complexas e interligações ricas entre as entidades do universo Pokémon. As funcionalidades implementadas, tais como pesquisa avançada, visualização detalhada das estatísticas dos Pokémon, comparação interativa e simulação de batalhas, demonstraram claramente a aplicabilidade prática e o valor acrescido proporcionado pela abordagem semântica.

Durante o desenvolvimento, enfrentamos desafios relevantes, como a manutenção da coerência dos dados durante a conversão para RDF, o aperfeiçoamento das consultas SPARQL para garantir eficiência e precisão, e a integração fluida destas consultas no framework Django. Outro desafio significativo foi assegurar uma interface intuitiva e visualmente agradável, especialmente em áreas complexas como a visualização das evoluções, comparações estatísticas e batalhas.

Este projeto revelou-se uma excelente oportunidade para aprofundar conhecimentos sobre os princípios da Web Semântica, RDF, SPARQL, e a utilização de GraphDB em contextos reais, assim como a sua integração prática com aplicações web modernas através do Django.

Quanto a perspectivas futuras, sugerimos a inclusão de funcionalidades adicionais como a possibilidade de realizar alterações diretamente no RDF via SPARQL UPDATE, implementar pesquisas mais complexas e interativas, integração com datasets externos como DBpedia ou Wikidata para enriquecer ainda mais os dados, melhorias na gestão de erros, adição de gestão de utilizadores e um contínuo aperfeiçoamento da interface para melhorar a experiência do utilizador.

## 7. Configuração para executar a aplicação

Para executar o projeto localmente e aceder a todas as funcionalidades de pesquisa, batalhas e perguntas via SPARQL, é necessário realizar os passos abaixo.

**Pré-requisitos:** Docker, Docker Compose, Git, Navegador Web

### Passo 1. Obter o código-fonte

Clone o repositório: `git clone https://github.com/ftferreira02/WS-Poked-X-Project.git`

Entre no diretório do projeto:

```
cd WS-Poked-X-Project
```

### Passo 2. Construir e iniciar os containeres

A partir da raiz do projeto, execute: `docker-compose up --build`

### Passo 3. Verificar o estado dos serviços

O GraphDB deve estar acessível em `http://localhost:7200`

O Django deve responder em `http://localhost:8000/`

### Passo 4. Configurar o GraphDB (primeira execução)

Aceda a `http://localhost:7200` no navegador.

Entre da secção “Import” e de seguida selecione a opção “Upload RDF Files”.

Selecione o ficheiro “pokemon-rdf.ttl” que se encontra em `app/dataBase` e, depois de o carregar, selecione a opção “The default graph”.

Por fim, selecione “Import”.

### Passo 5. Aceder à aplicação

Com todos os serviços em execução, abre o navegador e visita:

`http://localhost:8000`

Caso tenhas problemas de ligação, confirma que o container Django está configurado para usar a porta 8000 e que não há conflito local.