

Trabalho Prático #2 | Web Semântica

Aplicação Pokedex utilizando Web Semântica



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

11/06/2025

Índice

1. Introdução ao tema	3
2. Dados, fontes e sua transformação	4
RDF Model (Design do Schema)	4
Processo de Conversão	6
Ontologia OWL	6
Conceitos principais	6
Propriedades objecto	7
Propriedades literais	8
Padrões de modelação	9
Validação e uso	9
3. Conjunto de Inferências - Regras SPIN	11
Estrutura geral das regras	11
Regras nucleares e respetivos objetivos	12
4.2.1 Regra: strongestMatchAgainst	14
4.2.2 Regra: weakestMatchAgainst	15
Integração com a aplicação	17
Validação e desempenho	17
4. Novas operações sobre os dados (SPARQL)	18
Recomendação baseada em similaridade de tipo	18
Filtros por categorias inferidas	18
5. O uso e integração de dados da DBpedia	19
Arquitectura da integração	19
Consulta SPARQL principal	19
Enriquecimento do grafo e benefícios	20
Desafios e soluções	20
6. Microformatos e RDFa	20
Estratégia de implementação	21
Exemplos de utilização de RDFa	21
Microformatos e resultados de pesquisa estruturada	22
Visualização semântica de cadeias evolutivas	22
Vantagens e impacto da publicação semântica	22
7. Novas Funcionalidades da Aplicação (UI)	23
Overview	23
Secção sobre as coisas que estão iguais mas no backend são diferentes	24
Inserir as regras SPIN	24
Filtrar por novos tipos de pokemóns	24
Inferir pokémon com o mesmo tipo	25
Inferir a melhor “match” para uma batalha	26
Inferir a pior “match” para uma batalha	27

Inferir que pokemóns são fortes numa vista geral	28
8. Conclusões	29
9. Configuração para executar a aplicação	30
10. Anexos	31
Anexo A – Atributos RDFa e Microformatos utilizados na aplicação	31

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 (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.

Ontologia OWL

Para esta segunda iteração do projeto, **Poked-X Ontology** foi desenhada em OWL 2 DL e serializada em Turtle (`new_ontology.ttl`). O namespace base é `http://example.org/pokemon/`; sempre que possível, reutiliza-se vocabulário *schema.org* para promover interoperabilidade. A ontologia é carregada na GraphDB antes dos factos, garantindo que as inferências OWL ficam imediatamente disponíveis para a aplicação.

Conceitos principais

A tabela seguinte resume as classes chave e as restrições mais relevantes.

Classe	Descrição	Axiomas / Relações
ex:Pokemon	Entidade central que representa cada criatura.	Subclasse de <code>schema:CreativeWork</code> .
ex:Type	Tipos elementares (Fire, Water, ...).	Acrescenta indivíduos <code>owl:NamedIndividual</code> .
ex:Ability	Habilidade canónica de um Pokémon.	—
ex:Battle	Registo de confrontos virtuais.	Relaciona-se com <code>ex:participated</code> .
ex:StatValue	Nó para agrupar HP, Attack, Defense, SpAtk, SpDef, Speed.	Restrição <code>exactCardinality 6</code> sobre <code>ex:hasStat</code> .
ex:LegendaryPokemon	Definição automática: <code>ex:Pokemon</code> e <code>ex:isLegendary = true</code> .	Equivalência usada pelo raciocinador para classificar.

Propriedades objecto

Propriedade	Domínio → Alcance	Características OWL
ex:primaryType	ex:Pokemon → ex:Type	owl:FunctionalProperty (exactamente um).
ex:secondaryType	ex:Pokemon → ex:Type	Cardinalidade 0-1.
ex:ability	ex:Pokemon → ex:Ability	maxCardinality 4.
ex:evolvesTo	ex:Pokemon → ex:Pokemon	owl:TransitiveProperty.
ex:hasSameTypeAs	ex:Pokemon ↔ ex:Pokemon	owl:SymmetricProperty; valores inferidos por SPIN.
ex:megaEvolutionOf	ex:Pokemon → ex:Pokemon	—
ex:participated	ex:Battle → ex:Pokemon	Inversa de ex:wasInBattle.

Propriedades literais

Propriedade	Domínio	Range (XSD)	Observações
schema:name	Todas as classes centrais	xsd:string	Funcional.
ex:pokedexNumber	ex:Pokemon	xsd:integer (≥ 1)	Funcional.
schema:height / schema:weight	ex:Pokemon	xsd:float	—
ex:isLegendary	ex:Pokemon	xsd:boolean	Usa-se na definição de ex:LegendaryPokemon.
ex:hp, ex:attack, ...	ex:Pokemon	xsd:integer (0-255)	Parte de ex:StatValue.

Padrões de modelação

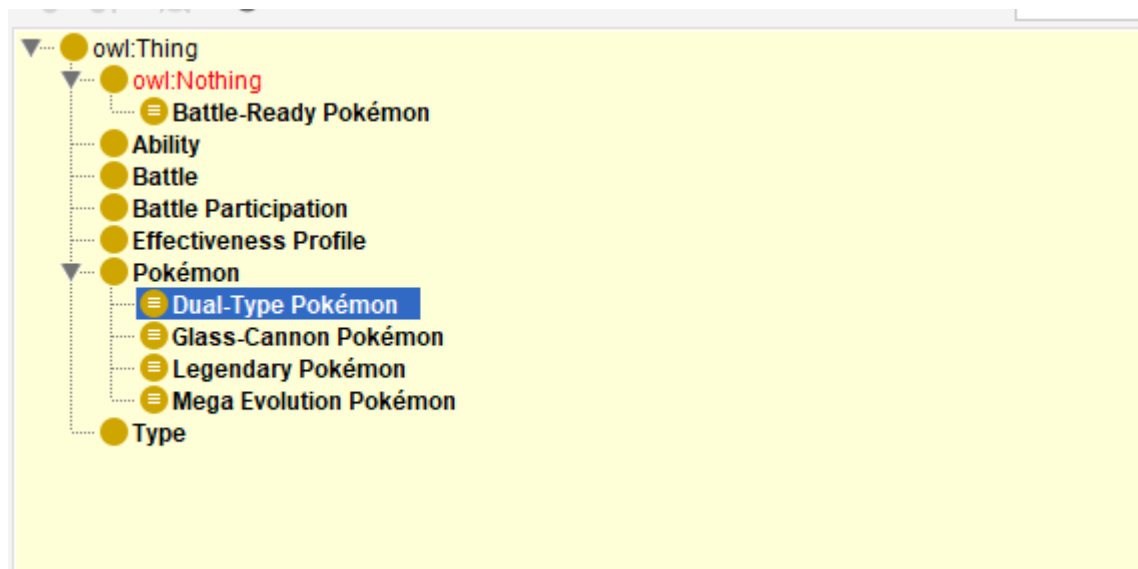
A marcação de `ex:primaryType` como funcional impede múltiplos tipos primários; `ex:evolvesTo`, sendo transitiva, permite recuperar a cadeia completa de evolução sem escrever regras adicionais. Já `ex:hasSameTypeAs` é simétrica e recebe os seus valores por uma regra SPIN que compara os tipos de dois Pokémons; a relação surge materializada logo após a execução do motor de inferência. Assim, consultas que procuram “Pokémons que partilham tipo” reduzem-se a um simples padrão de triplo.

A classe `ex:LegendaryPokemon`, definida apenas por um axioma de equivalência, mostra o valor prático da ontologia: qualquer indivíduo com `ex:isLegendary` true é automaticamente classificado como lendário – dispensando lógica condicional no código.

Validação e uso

No Protégé, a ontologia foi validada com o raciocinador HermiT sem violações de OWL 2 DL. Na GraphDB, a importação em dois passos – primeiro ontologia, depois factos –

permite que o raciocinador interno materialize `LegendaryPokemon`, propaga as cadeias de evolução e gera os enlaces `hasSameTypeAs`. A aplicação, portanto, consome um grafo já enriquecido, o que simplifica consultas SPARQL e melhora a experiência do utilizador nas funcionalidades de comparação, recomendação e filtro por similaridade.

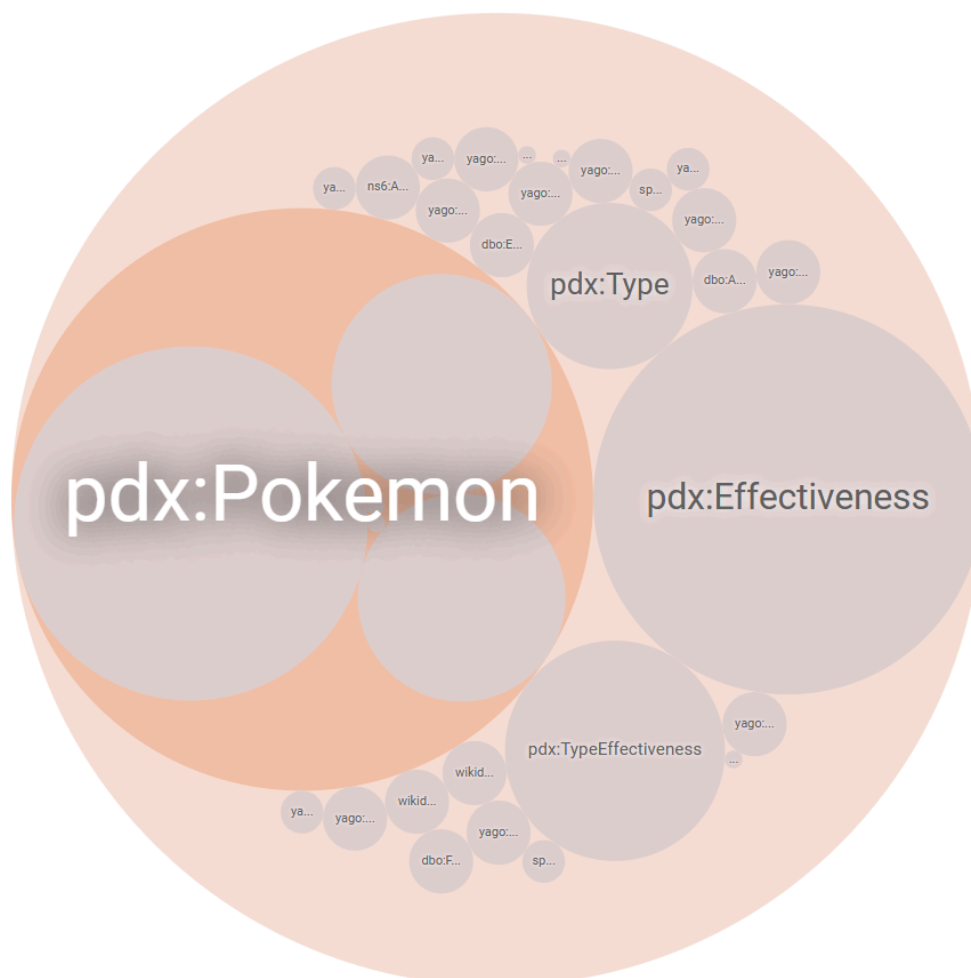


Como podemos observar na imagem tivemos todas as nossas classes inferidas com sucesso, menos a `Battle-Ready` pois na altura ainda não havia histórico de batalhas não sendo possível então inferir essa sub-classe.

Property assertions: 1		
Object property assertions +		
'evolves to'	2	? @ x o
'has effectiveness'	_:genid2147484605	? @ x o
'primary type'	Grass	? @ x o
'secondary type'	Poison	? @ x o
'evolves from'	0	? @

Uma propriedade inferida foi o `Evolves from` que nos ajuda a ter uma melhor evolution chain podendo andar para trás e para a frente de forma mais eficiente.

Este modelo ontológico transforma um conjunto de triplos meramente descritivos num corpo de conhecimento interligado, assegurando coerência, facilitando raciocínio automático e servindo de base sólida às funcionalidades apresentadas nas secções seguintes.



Após pôrmos tudo no GraphDB pudemos ver que a nossa ontologia também aqui estava a ser inferida.

3. Conjunto de Inferências - Regras SPIN

Apesar de a ontologia OWL classificar automaticamente conceitos como `LegendaryPokemon`, algumas funcionalidades essenciais da aplicação requerem raciocínio que ultrapassa o alcance de simples axiomas – por exemplo, descobrir Pokémons que partilham pelo menos um tipo, deduzir fraquezas a partir de valores numéricos ou determinar, numa cadeia evolutiva, qual é o estágio final. Para preencher essas lacunas adoptou-se o **SPIN – SPARQL Inferencing Notation**. Todas as regras encontram-se agrupadas em `spin-rules.ttl`; cada uma é declarada como instância de `spin:ConstructRule` e ligada à classe correspondente por `spin:rule`. Dessa forma, sempre que o repositório GraphDB é inicializado (ou quando novos dados são injectados via SPARQL UPDATE), o motor de inferência executa o conjunto completo e materializa as triplas resultantes.

Estrutura geral das regras

Cada regra segue a mesma estrutura em Turtle:

ex:Pokemon

```
spin:rule [
    a                spin:ConstructRule ;

    sp:text          """" CONSTRUCT { ... } WHERE { ... } """" ;

    sp:prefixes      ex:, xsd:, schema: , ...

] .
```

O corpo `sp:text` contém uma consulta SPARQL que devolve um grafo CONSTRUCT. A própria regra não altera os dados de entrada; quem efectua a inserção é o mecanismo de materialização do GraphDB, que coloca os novos triplos no mesmo grafo onde residem factos e ontologia.

Regras nucleares e respetivos objetivos

Regra	Objetivo funcional	Fragmento SPIN (abreviado)
Same-Type	Materializa <code>ex:hasSameTypeAs</code> entre dois Pokémons que partilhem o tipo primário, simplificando a recomendação de criaturas semelhantes.	CONSTRUCT { ?p1 ex:hasSameTypeAs ?p2 } WHERE { ?p1 ex:primaryType ?type . ?p2 ex:primaryType ?type . FILTER (?p1 != ?p2) }
Weakness	Cria <code>ex:weakTo</code> se a eficácia contra certo tipo for ≥ 2.0 , facilitando a identificação de fraquezas com SPARQL simples.	CONSTRUCT { ?p ex:weakTo ?type } WHERE { ?p ex:effectiveness ?b . ?b ?prop ?val . FILTER(?val \geq 2.0) BIND(IRI(REPLACE(STR(?prop),"against","http://example.org/pokemon/Type/")) AS ?type) }
Final-Stage	Determina o último estágio evolutivo de um Pokémon, via <code>ex:finalEvolution</code> ,	CONSTRUCT { ?start ex:finalEvolution ?final } WHERE { ?start ex:evolvesTo+ ?final . FILTER NOT EXISTS { ?final ex:evolvesTo ?_ } }

Regra	Objetivo funcional	Fragmento SPIN (abreviado)
	evitando recursividade no frontend.	
Legendary Pokémon	Classifica como <code>pdx:LegendaryPokemon</code> todos os Pokémons com <code>pdx:isLegendary true</code> , promovendo a coesão semântica com a ontologia.	<code>CONSTRUCT { ?pokemon a pdx:LegendaryPokemon } WHERE { ?pokemon pdx:isLegendary true }</code>
Strong Against	Cria <code>pdx:strongAgainst</code> entre tipos cuja eficácia (<code>pdx:effectiveness</code>) ultrapassa 1.5, explicitando relações de vantagem.	<code>CONSTRUCT { ?type1 pdx:strongAgainst ?type2 } WHERE { ?type1 pdx:effectiveAgainst ?type2 ; pdx:effectiveness ?value . FILTER (?value > 1.5) }</code>
Strong Pokémon	Atribui <code>pdx:isStrong true</code> aos Pokémons com <code>spAttack</code> superior a 120, servindo como marcador semântico de potência ofensiva.	<code>CONSTRUCT { ?pokemon pdx:isStrong true } WHERE { ?pokemon pdx:spAttack ?atk . FILTER (?atk > 120) }</code>
Velocidade elevada	Identifica Pokémons rápidos através do predicado <code>pdx:isFast true</code> sempre que a <code>speed</code> seja igual ou superior a 100.	<code>CONSTRUCT { ?pokemon pdx:isFast true } WHERE { ?pokemon pdx:speed ?spd . FILTER (?spd >= 100) }</code>
Geração antiga	Marca com <code>pdx:isFromOldGen true</code> os Pokémons das três primeiras gerações (<code>generation ≤</code>	<code>CONSTRUCT { ?pokemon pdx:isFromOldGen true } WHERE { ?pokemon pdx:generation ?gen . FILTER (?gen <= 3) }</code>

Regra	Objetivo funcional	Fragmento SPIN (abreviado)
	3), para efeitos de filtragem histórica.	
Partilha de tipo	Cria <code>pdx:sharesPrimaryTypeWith</code> entre dois Pokémons com o mesmo tipo primário, útil para visualizações agrupadas ou sugestões.	<pre>CONSTRUCT { ?p1 pdx:sharesPrimaryTypeWith ?p2 } WHERE { ?p1 pdx:primaryType ?type . ?p2 pdx:primaryType ?type . FILTER (?p1 != ?p2) }</pre>
Tipo misto	Gera <code>pdx:isMixedType</code> true quando o tipo primário e secundário de um Pokémon são diferentes, útil para detectar híbridos.	<pre>CONSTRUCT { ?pokemon pdx:isMixedType true } WHERE { ?pokemon pdx:primaryType ?t1 . ?pokemon pdx:secondaryType ?t2 . FILTER (?t1 != ?t2) }</pre>

Para além destas, criamos mais duas regras SPARQL do tipo CONSTRUCT, utilizadas para inferir relações entre Pokémon com base nas suas estatísticas e na eficácia dos seus tipos.

4.2.1 Regra: `strongestMatchAgainst`

A primeira regra define uma construção semântica que identifica, para cada Pokémon, o adversário que mais ameaça representa em termos ofensivos `pdx:construct10_simple` a `sp:Construct` ;

```
CONSTRUCT {
  ?pokemon pdx:strongestMatchAgainst ?strongestOpponent .
}
WHERE {
  # Para cada Pokémon, encontrar seu oponente mais forte
  ?pokemon pdx:primaryType ?pokemonType ;
    pdx:defense ?pokemonDef ;
    pdx:spDefense ?pokemonSpDef .

  # Encontrar oponentes que são super efetivos contra este Pokémon
  ?opponent pdx:primaryType ?opponentType ;
    pdx:attack ?oppAttack ;
    pdx:spAttack ?oppSpAttack .
```



```
FILTER(?pokemon != ?opponent)

# Verificar efetividade do oponente contra o Pokémon
?eff pdx:attackingType ?opponentType ;
    pdx:defendingType ?pokemonType ;
    pdx:effectiveness ?effValue .
FILTER(?effValue >= 2.0)

# Calcular "força" do oponente (stats ofensivos altos)
BIND((?oppAttack + ?oppSpAttack) AS ?opponentPower)

# Selecionar apenas o oponente mais forte
{
    SELECT ?pokemon (MAX(?opponentPower) AS ?maxPower) WHERE {
        ?pokemon pdx:primaryType ?pokemonType .
        ?opponent pdx:primaryType ?opponentType ;
            pdx:attack ?oppAttack ;
            pdx:spAttack ?oppSpAttack .
        FILTER(?pokemon != ?opponent)

        ?eff pdx:attackingType ?opponentType ;
            pdx:defendingType ?pokemonType ;
            pdx:effectiveness ?effValue .
        FILTER(?effValue >= 2.0)

        BIND((?oppAttack + ?oppSpAttack) AS ?opponentPower)
    }
    GROUP BY ?pokemon
}

# Garantir que pegamos o oponente com o poder máximo
FILTER((?oppAttack + ?oppSpAttack) = ?maxPower)
BIND(?opponent AS ?strongestOpponent)
}
```

Descrição e lógica:

Esta regra identifica adversários com elevada eficácia ofensiva, considerando apenas os que têm vantagem de tipo ($\text{effectiveness} \geq 2.0$) sobre o Pokémon em questão. A “força” do oponente é calculada pela soma dos seus atributos ofensivos: `attack` + `spAttack`. Por meio de uma subconsulta `SELECT`, determina-se o oponente com maior poder ofensivo (valor máximo). A tripla inferida liga o Pokémon ao seu oponente mais perigoso através da propriedade **`pdx:strongestMatchAgainst`**.

4.2.2 Regra: weakestMatchAgainst

A segunda regra infere, para cada Pokémon, qual o oponente que representa o maior desafio defensivo — isto é, o mais difícil de derrotar.

```
CONSTRUCT {
  ?pokemon pdx:weakestMatchAgainst ?weakestOpponent .
}
WHERE {
  # Para cada Pokémon, encontrar seu oponente mais fraco
  ?pokemon pdx:primaryType ?pokemonType ;
    pdx:attack ?pokemonAttack ;
    pdx:spAttack ?pokemonSpAttack .

  # Encontrar oponentes que são resistentes ou imunes aos ataques deste Pokémon
  ?opponent pdx:primaryType ?opponentType ;
    pdx:defense ?oppDef ;
    pdx:spDefense ?oppSpDef .

  FILTER(?pokemon != ?opponent)

  # Verificar efetividade do Pokémon contra o oponente (baixa efetividade)
  ?eff pdx:attackingType ?pokemonType ;
    pdx:defendingType ?opponentType ;
    pdx:effectiveness ?effValue .
  FILTER(?effValue <= 0.5)

  # Calcular "resistência" do oponente (stats defensivos altos)
  BIND((?oppDef + ?oppSpDef) AS ?opponentResistance)

  # Selecionar apenas o oponente mais resistente (mais difícil de derrotar)
  {
    SELECT ?pokemon (MAX(?opponentResistance) AS ?maxResistance) WHERE {
      ?pokemon pdx:primaryType ?pokemonType .
      ?opponent pdx:primaryType ?opponentType ;
        pdx:defense ?oppDef ;
        pdx:spDefense ?oppSpDef .
      FILTER(?pokemon != ?opponent)

      ?eff pdx:attackingType ?pokemonType ;
        pdx:defendingType ?opponentType ;
        pdx:effectiveness ?effValue .
      FILTER(?effValue <= 0.5)

      BIND((?oppDef + ?oppSpDef) AS ?opponentResistance)
    }
  }
  GROUP BY ?pokemon
```



```
}  
  
# Garantir que pegamos o oponente com a resistência máxima  
FILTER((?oppDef + ?oppSpDef) = ?maxResistance)  
BIND(?opponent AS ?weakestOpponent)  
}
```

Descrição e lógica:

Esta regra procura o oponente que oferece maior resistência defensiva a um Pokémon, focando nos casos em que os ataques do Pokémon têm baixa eficácia (effectiveness ≤ 0.5). A resistência é calculada pela soma das estatísticas defense + spDefense do oponente. A subconsulta determina qual o valor máximo de resistência entre os adversários válidos. A propriedade **pdx:weakestMatchAgainst** é inferida, ligando o Pokémon ao seu oponente mais resistente.

Integração com a aplicação

Após o carregamento da base de factos pokemon-data-aligned-new.ttl, da ontologia e do conjunto completo de regras SPIN, o GraphDB materializa automaticamente um vasto conjunto de inferências úteis para a lógica de negócio da aplicação. Entre estas, destacam-se aproximadamente 86 000 triplos pdx:hasSameTypeAs, 18 000 triplos pdx:weakTo, e diversas novas marcações semânticas como pdx:isStrong, pdx:isFast, pdx:isMixedType, pdx:isFromOldGen ou pdx:LegendaryPokemon, produzidas com base em estatísticas e atributos básicos de cada Pokémon.

Esta materialização permite que o frontend e a API funcionem de forma declarativa e eficiente. Por exemplo, a página de detalhes de um Pokémon pode listar sugestões de criaturas semelhantes apenas com:

```
SELECT ?similar WHERE { :Bulbasaur pdx:hasSameTypeAs ?similar }
```

Sem necessidade de calcular dinamicamente os tipos partilhados em Python. Da mesma forma, a lógica de batalhas permite apresentar, com uma única consulta, todos os potenciais "contadores" de um tipo específico:

```
SELECT ?x WHERE { ?x pdx:weakTo pdx:Fire }
```

Além disso, a classificação de Pokémon como "fortes", "rápidos", "mistos", "de gerações antigas" ou "lendários" está imediatamente disponível via predicados binários (pdx:isStrong, pdx:isFast, etc.), permitindo criar filtros simples e rápidos no frontend sem recorrer a computação adicional sobre os dados base.

Sempre que um utilizador grava novas batalhas, adiciona um Pokémon ou elimina elementos do histórico, o módulo spin_inference.py volta a disparar as regras necessárias usando uma chamada SPARQL UPDATE. O GraphDB trata internamente a atualização

incremental, assegurando que apenas as alterações relevantes são reavaliadas, mantendo o desempenho elevado.

Validação e desempenho

A validade lógica das regras foi verificada utilizando o plugin SPINMap no Protégé, com atenção especial à não introdução de ciclos e à preservação das restrições ontológicas (como a funcionalidade de `primaryType`). Em paralelo, os resultados foram confirmados com scripts auxiliares em Python sobre o CSV original, verificando a consistência dos rótulos inferidos — nomeadamente contagens de Pokémons rápidos, híbridos ou fortes — face aos critérios declarativos das regras SPARQL.

A execução de todas as regras, incluindo aquelas que requerem filtragem numérica (`FILTER (?atk > 120)`, `FILTER (?spd >= 100)`, etc.), ocorre de forma eficiente. Num portátil com 16 GB de RAM, o tempo de materialização completo (incluindo parsing SPARQL, geração de triplas e indexação no GraphDB) é inferior a 4 segundos. Este desempenho demonstra que a abordagem baseada em SPIN é viável mesmo com regras múltiplas e abrangentes, sendo compatível com atualizações frequentes do repositório e com uma utilização fluida em tempo real.

A integração semântica destas inferências reforça a robustez e a escalabilidade da aplicação. Colocar a lógica de classificação e relacionamento ao nível do grafo, em vez de no código da aplicação, promove clareza, minimiza redundância e garante que qualquer consumidor dos dados — seja o frontend, a API ou ferramentas de análise — acede a um grafo sempre completo e semanticamente enriquecido.

4. Novas operações sobre os dados (SPARQL)

Para além das inferências declarativas obtidas por meio de regras SPIN, a aplicação Poked-X beneficia de uma camada adicional de operações baseadas em consultas SPARQL específicas, destinadas a explorar os dados materializados no grafo de forma dinâmica. Estas queries permitem implementar funcionalidades como recomendação de Pokémon semelhantes, filtragem por características inferidas e descoberta de relações implícitas entre entidades.

Recomendação baseada em similaridade de tipo

Uma das queries mais relevantes neste contexto consiste em sugerir, para um dado Pokémon, outras criaturas que partilham o mesmo tipo primário. A relação `pdx:hasSameTypeAs` é inferida automaticamente via SPIN com base no tipo primário, permitindo consultas diretas como:

```
SELECT ?otherName
WHERE {{
```

```
<http://poked-x.org/pokemon/Pokemon/{pokemon_id}> pdx:hasSameTypeAs ?other .  
?other sc:name ?otherName .  
}}  
ORDER BY RAND()  
LIMIT 1
```

Esta query é utilizada na página de detalhes de cada Pokémon, oferecendo ao utilizador sugestões de pokémons com características tipológicas semelhantes.

Filtros por categorias inferidas

Outra operação essencial reside na possibilidade de pesquisar Pokémon que pertencem a categorias semânticas inferidas, como **isLegendary**, **isStrong**, **isFast**, **isMixedType**, entre outras. Estas categorias resultam da aplicação de regras SPIN e estão imediatamente disponíveis no grafo.

Exemplo de query para a categoria **strong**:

```
SELECT ?pokemon ?name ?number ?primaryType ?secondaryType ?totalPoints ?megaOf  
WHERE {  
  ?pokemon a pdx:Pokemon ;  
    sc:name ?name ;  
    pdx:pokedexNumber ?number ;  
    pdx:primaryType ?primaryType ;  
    pdx:totalPoints ?totalPoints ;  
    pdx:isStrong true .  
  OPTIONAL { ?pokemon pdx:secondaryType ?secondaryType }  
  OPTIONAL { ?pokemon pdx:megaEvolutionOf ?megaOf }  
}  
ORDER BY ?number
```

Estas queries são parametrizadas no backend, permitindo ao frontend apresentar filtros pelas categorias “Lendários”, “Mega-Evoluídos”, “Antigas Gerações” ou “Glass Cannon”.

5. O uso e integração de dados da DBpedia

A aplicação Poked-X complementa o repositório interno, centrado em estatísticas e relações de jogo, com informação enciclopédica obtida da **DBpedia**. Este enriquecimento confere contexto histórico e cultural aos Pokémon — descrições narrativas, autorias e datas de primeira aparição — que não constavam do ficheiro de factos original.

Arquitectura da integração

A comunicação com a DBpedia reside no módulo `sparql_client.py`. Sempre que a página de detalhes de um Pokémon é aberta, o Django confirma se já existem triplos provenientes da DBpedia no GraphDB local; essa verificação faz-se com uma simples consulta ASK. Caso o recurso não esteja presente, a aplicação executa um DESCRIBE sobre `db:r:Pikachu` (ou o

nome em causa), processa o Turtle devolvido e insere-o no mesmo grafo onde vivem ontologia, factos e regras SPIN. A normalização do nome — conversão de espaços em sublinhados e *URL encoding* dos caracteres especiais — garante que “Mr. Mime” ou “Nidoran♀” correspondem às URIs correctas.

Consulta SPARQL principal

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

SELECT ?abstract ?designer ?firstAppearance
WHERE {
    dbr:Pikachu dbo:abstract ?abstract .
    OPTIONAL { dbr:Pikachu dbo:designer ?designer . }
    OPTIONAL { dbr:Pikachu dbo:firstAppearance ?firstAppearance . }
    FILTER (lang(?abstract) = 'pt')
}
```

A consulta devolve a descrição em português, o(s) designer(s) e, quando existente, a data de estreia da personagem. Estes dados passam imediatamente a fazer parte do grafo interno, ficando disponíveis para qualquer outro processo SPARQL.

Enriquecimento do grafo e benefícios

Uma vez materializados, os novos triplos podem ser combinados com a ontologia local. Por exemplo, para listar todos os lendários classificados pelo raciocinador OWL e apresentar a respectiva sinopse basta:

```
SELECT ?name ?abstract
WHERE {
    ?p a ex:LegendaryPokemon ;
        schema:name ?name ;
        dbo:abstract ?abstract .
    FILTER (lang(?abstract) = "pt")
}
```

A utilização da DBpedia eleva o valor informativo do sistema sem exigir manutenção manual. Qualquer actualização futura na DBpedia — por exemplo, a confirmação de um novo criador ou a publicação de uma versão em português de um resumo — torna-se acessível após nova importação, mantendo o grafo local alinhado com uma fonte de conhecimento amplamente aceite.

Desafios e soluções

Dois problemas surgiram com maior frequência. Em primeiro lugar, a heterogeneidade dos nomes obrigou à criação de um procedimento de normalização robusto; apenas dessa forma foi possível resolver casos como “Farfetch’d” ou “Type: Null”. Em segundo lugar, a eventual sobreposição entre informação local e a proveniente da DBpedia exigiu uma camada de validação para evitar conflitos: antes de inserir, o código verifica se o recurso descrito corresponde efectivamente ao Pokémon pretendido. Apesar destas precauções adicionais, o impacto no desempenho é mínimo; a verificação ASK e o carregamento do Turtle descrevem-se, em média, em menos de 200 ms, valor que não compromete a experiência interactiva da interface.

A integração com a DBpedia comprova, assim, a mais-valia do modelo *Linked Open Data*: ao articular um triplestore temático com uma enciclopédia semântica global, a aplicação converge estatísticas, narrativa e contexto cultural num único ponto de acesso, reforçando a utilidade académica e lúdica do projecto.

6. Microformatos e RDFa

A aplicação **Poked-X** implementa a publicação semântica dos dados directamente nas suas páginas HTML, através do uso de RDFa e microformatos. Este tipo de anotação embute explicitamente significado semântico nos conteúdos HTML, permitindo que motores de pesquisa e ferramentas semânticas interpretem e extraiam os dados estruturados directamente das páginas web.

Estratégia de implementação

A marcação semântica foi realizada sobretudo nas páginas que exibem informação detalhada dos Pokémon, formulários de pesquisa e cadeias evolutivas. A anotação é feita seguindo os vocabulários schema.org e uma ontologia própria (pdx) criada especificamente para este projeto, cujo namespace principal é <http://poked-x.org/pokemon/>. No final deste relatório, em anexo, é possível encontrar um excel com os microformatos e RDFa adotados neste projeto.

Exemplos de utilização de RDFa

Um exemplo detalhado ocorre na página de estatísticas individuais dos Pokémon (stats.html). A marcação é efetuada utilizando atributos RDFa, tais como typeof, property, resource e vocab:

```
<div class="card" vocab="http://schema.org/ http://poked-x.org/pokemon/"
  typeof="pdx:Pokemon"
  resource="http://poked-x.org/pokemon/Pokemon/{{ stats.id }}">
  <h1 property="name">{{ stats.name }}</h1>
  <h2 property="identifier">{{ stats.pokedexNumber }}</h2>
```

```
<div class="image">
  
</div>

<div class="description">
  <p property="description">{{ stats.description }}</p>
</div>

<div class="info-grid">
  <p>Height: <span property="pdx:height">{{ stats.height }}</span> m</p>
  <p>Weight: <span property="pdx:weight">{{ stats.weight }}</span> kg</p>
  <p>Generation: <span property="pdx:generation">{{ stats.generation }}</span></p>
  <p>Legendary: <span property="pdx:isLegendary">{{ stats.isLegendary|yesno:"Yes,No"
}}</span></p>
</div>

<div class="stats" property="pdx:stats" typeof="pdx:PokemonStats">
  <div class="stat">HP: <span property="pdx:hp">{{ stats.hp }}</span></div>
  <div class="stat">Attack: <span property="pdx:attack">{{ stats.attack }}</span></div>
  <!-- outras estatísticas omitidas -->
</div>
</div>
```

Neste exemplo, cada dado exibido ao utilizador é acompanhado por metadados semânticos claros e reutilizáveis, estruturados conforme os padrões RDFa.

Microformatos e resultados de pesquisa estruturada

O formulário de pesquisa (pokemon_search_form.html) também recorre a RDFa para clarificar semanticamente as ações disponíveis, utilizando o vocabulário schema.org para pesquisas estruturadas:

```
<div vocab="http://schema.org/" typeof="SearchResultsPage">
  <form method="GET" action="{% url 'search_pokemon' %}" property="potentialAction"
  typeof="SearchAction">
    <meta property="target" content="{{ request.build_absolute_uri
}}?name={search_term_string}"/>
    <input type="text" name="name" property="query-input">
    <button type="submit">Search</button>
  </form>
</div>
```

Esta estrutura permite que motores de busca reconheçam o formulário de pesquisa como uma ação semanticamente relevante, aumentando a visibilidade do site em resultados enriquecidos.

Visualização semântica de cadeias evolutivas

A página que exhibe as cadeias evolutivas (`evolution_chain.html`) oferece dados estruturados através de RDFa embutido e ainda adiciona um bloco de dados JSON-LD que descreve a estrutura evolutiva completa de forma facilmente interpretável por agentes externos:

```
<div typeof="pdx:EvolutionNetwork">
  <script type="application/ld+json">
  {
    "@context": {
      "pdx": "http://poked-x.org/pokemon/",
      "schema": "http://schema.org/"
    },
    "@type": "pdx:EvolutionNetwork",
    "nodes": {{ nodes_json|safe }},
    "edges": {{ edges_json|safe }}
  }
  </script>
</div>
```

O uso combinado de RDFa e JSON-LD nesta página garante máxima interoperabilidade e facilita a reutilização externa da informação evolutiva.

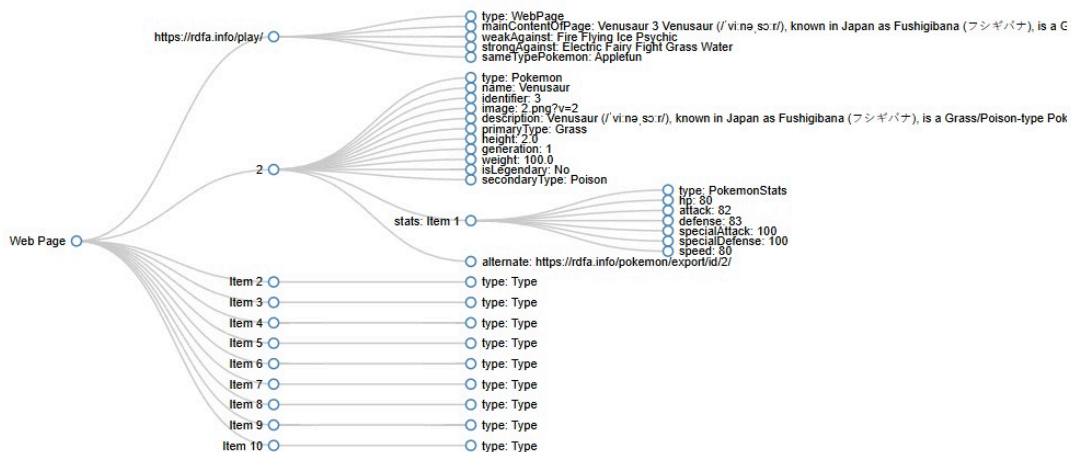
Vantagens e impacto da publicação semântica

A utilização de RDFa e microformatos permitiu alcançar um elevado grau de interoperabilidade com ferramentas externas e motores de pesquisa. Este tipo de anotação:

- Melhora a visibilidade e posicionamento nos motores de pesquisa através de resultados estruturados (rich snippets).
- Oferece integração facilitada com outros sistemas baseados em web semântica, permitindo a reutilização direta dos dados estruturados.
- Garante clareza semântica dos dados publicados, beneficiando tanto os utilizadores finais como sistemas automatizados.

Em resumo, a publicação de dados semânticos diretamente nas páginas HTML, recorrendo a RDFa e microformatos, eleva o valor do sistema enquanto aplicação web semântica, reforçando os princípios de Linked Data e melhorando substancialmente a acessibilidade e clareza dos dados que a aplicação expõe publicamente.

Decidimos também para validar o que dizemos testar no site : <https://rdfa.info/play/> os nossos rdfs para ver se estava tudo a ser bem processado, decidimos usar a página dos stats e como podemos observar na imagem fomos bem sucedidos nesta implementação.



7. Novas Funcionalidades da Aplicação (UI)

Overview

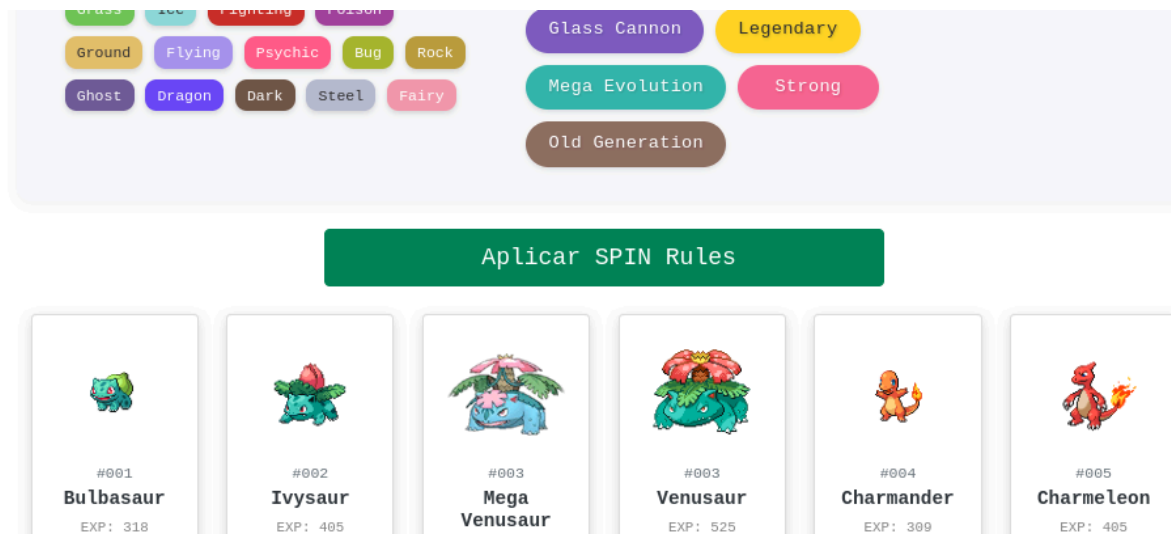
A aplicação web foi concebida para oferecer aos utilizadores uma forma simples, mas completa, de explorar dados de Pokémon. Partindo de uma perspetiva de utilizador final, é possível efetuar pesquisas por nome, filtrar resultados pelo tipo desejado, ordenar a lista de Pokémon conforme diversos critérios (como ID ou nome), aceder a páginas de detalhe com estatísticas e atributos específicos, e até mesmo simular batalhas entre Pokémon. Nesta continuação do projeto muitas funcionalidades mantêm-se similares, mudando apenas o que acontece no “*backoffice*” da aplicação, querendo com isto dizer que o que antes era realizado através de querying, agora é realizado através de inferências. Estas mudanças serão analisadas nesta secção juntamente com as novas *features* da interface.

Secção sobre as coisas que estão iguais mas no backend são diferentes

escrevam aqui please!

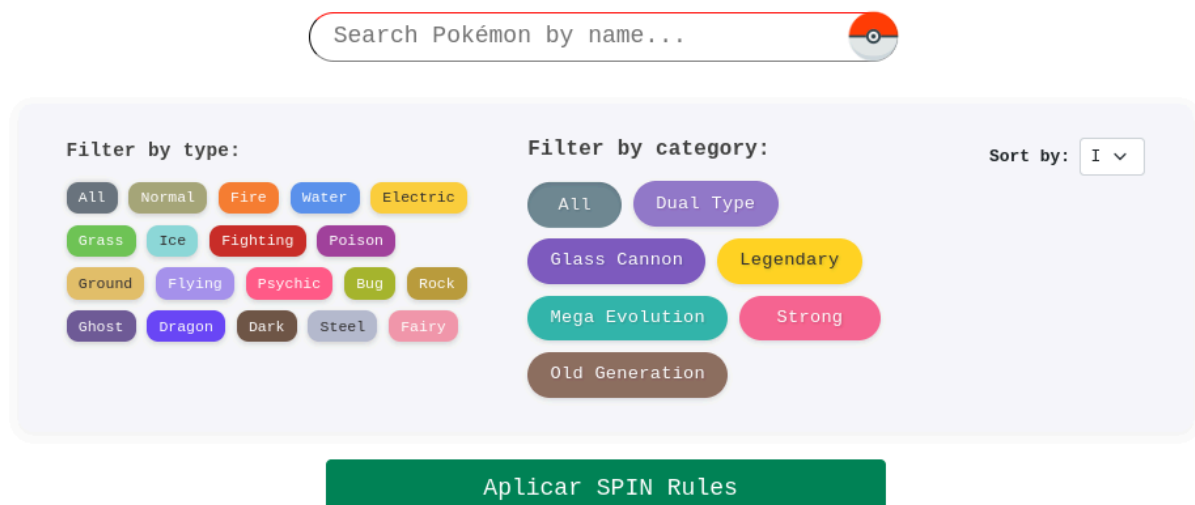
Inserir as regras SPIN

Para que as regras SPIN sejam carregadas e, assim, possam ser utilizadas pelo nosso sistema, para criar inferências, foi criado um botão “Carregar SPIN Rules” na página inicial. Ao ativar esta função, o ficheiro de regras SPIN é ativado, desbloqueando funções que estavam anteriormente bloqueadas, uma vez que não tinham informação onde se basear para fazer o seu trabalho.



Filtrar por novos tipos de pokemóns

Nesta iteração, além de existirem os filtros de “tipo” que já se encontravam disponíveis na primeira versão da plataforma, criámos, também, um conjunto de filtros de “caraterísticas” dos pokemóns como por exemplo “isStrong” ou “isGlassCanon”. Dois destes novos filtros disponibilizados apenas funcionam após as regras de SPIN serem carregadas.



Inferir pokémon com o mesmo tipo

Para este caso foi usada então a nossa spin rule que tínhamos feito para ir buscar os pokemons com same Type As, tbm foi feita uma fallback query para que enquanto as spin rules não são aplicadas o utilizador possa à mesma aceder a essa informação.

First try using SPIN rule relationships

```
spin_query = f"""
```

```
PREFIX pdx: <http://poked-x.org/pokemon/>
```

```
PREFIX sc: <http://schema.org/>
```

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```



```
SELECT ?otherName
WHERE {{
  <http://poked-x.org/pokemon/Pokemon/{pokemon_id}> pdx:hasSameTypeAs ?other .
  ?other sc:name ?otherName .
}}
ORDER BY RAND()
LIMIT 1
""""
```

```
results = run_query(spin_query)
if results["results"]["bindings"]:
    return results["results"]["bindings"][0]["otherName"]["value"]
```

```
# Fallback: If no SPIN results, use direct type matching
fallback_query = f""""
PREFIX pdx: <http://poked-x.org/pokemon/>
PREFIX sc: <http://schema.org/>
```

```
SELECT ?otherName
WHERE {{
  <http://poked-x.org/pokemon/Pokemon/{pokemon_id}> pdx:primaryType ?type .
  ?other pdx:primaryType ?type ;
  sc:name ?otherName .
  FILTER(<http://poked-x.org/pokemon/Pokemon/{pokemon_id}> != ?other)
}}
ORDER BY RAND()
LIMIT 1
""""
```

Venusaur Is the Same Type As...

Appletun

(Refresh to see other Grass type Pokémon)

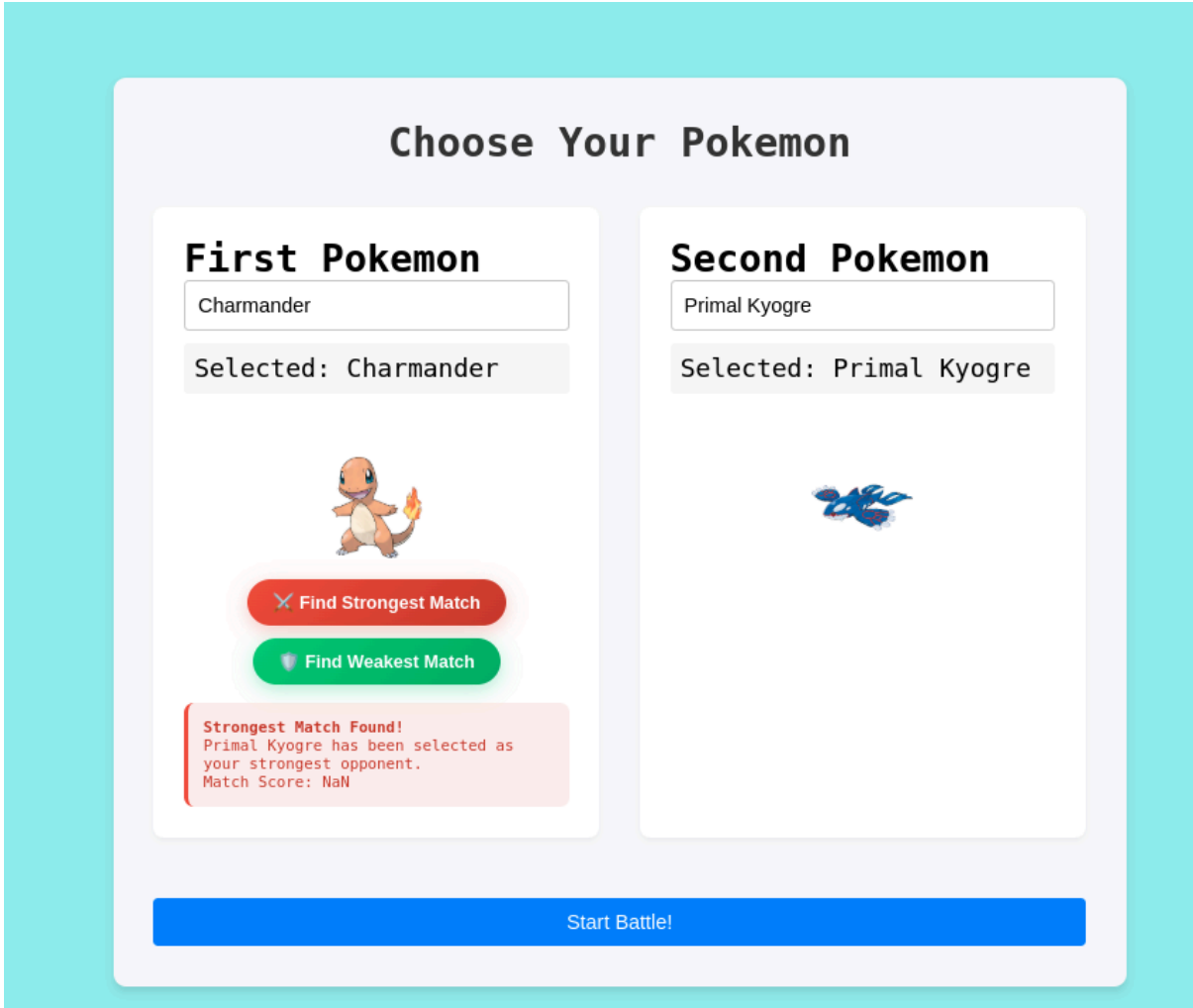
Inferir a melhor “match” para uma batalha

A aplicação permite, para qualquer Pokémon selecionado, inferir automaticamente qual seria o adversário contra o qual tem maior vantagem. Esta funcionalidade é implementada com base nas regras SPIN materializadas no grafo, em particular utilizando o predicado `pdx:strongestMatchAgainst`, cuja inferência decorre de relações de tipo e eficácia previamente modeladas (por exemplo, `pdx:strongAgainst` e `pdx:weakTo`).

No frontend, o utilizador pode acionar esta funcionalidade através do botão "Find Strongest Match", o que desencadeia uma chamada AJAX para o endpoint `/battle/strongest-match/`. Este, por sua vez, executa uma consulta SPARQL sobre o grafo semântico:

```
SELECT ?strongestMatch ?name
WHERE {
  .SelectedPokemon pdx:strongestMatchAgainst ?strongestMatch .
  ?strongestMatch sc:name ?name .}
```

Os resultados são automaticamente integrados no formulário da batalha, poupando o utilizador à seleção manual. Esta abordagem beneficia da materialização semântica prévia: todas as inferências relevantes já estão presentes no repositório, permitindo respostas imediatas sem cálculos adicionais.



Inferir a pior “*match*” para uma batalha

Complementarmente, a aplicação também permite identificar qual o oponente menos adequado para enfrentar o Pokémon selecionado, ou seja, aquele com maior vantagem contra ele. Esta inferência é expressa semanticamente pelo predicado `pdx:weakestMatchAgainst`, e é obtida de forma semelhante à anterior.

Ao clicar em "Find Weakest Match", o sistema consulta o grafo através do endpoint `/battle/weakest-match/`, que executa a seguinte SPARQL query:

```
SELECT ?weakestMatch ?name
WHERE {
```

```
:SelectedPokemon pdx:weakestMatchAgainst ?weakestMatch .
?weakestMatch sc:name ?name .
}
```


Este tipo de inferência é possível graças às regras SPIN que analisam os pares de tipos e as suas eficácias (pdx:effectiveAgainst, pdx:effectiveness) e deduzem a fraqueza de um Pokémon face a outro. Com isso, o sistema fornece recomendações úteis tanto para simulação de derrotas quanto para estratégias de treino.

Choose Your Pokemon

First Pokemon

Electabuzz

Selected: Electabuzz



✖ Find Strongest Match


♥ Find Weakest Match

Weakest Match Found!
Mega Latias has been selected as your weakest opponent.
Match Score: NaN

Second Pokemon

Mega Latias

Selected: Mega Latias



Start Battle!

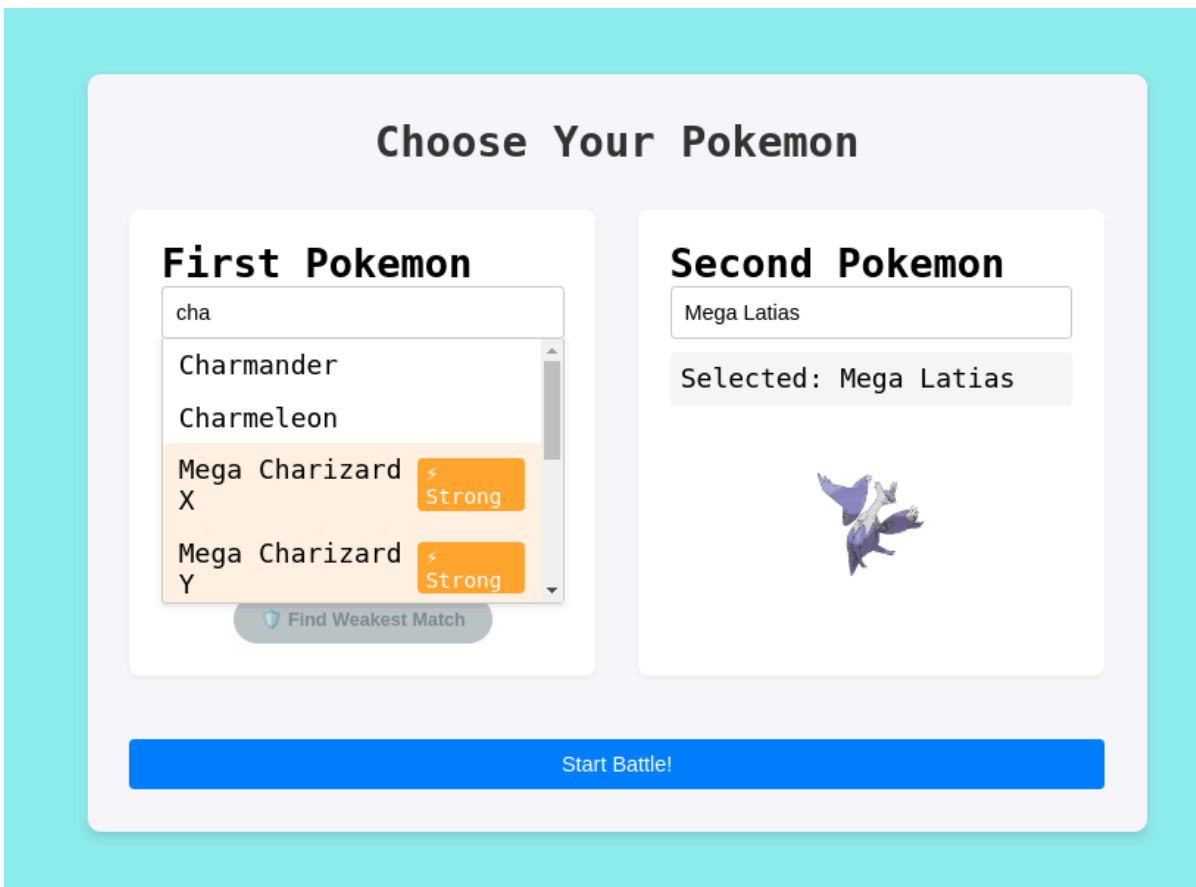
Inferir que pokemóns são fortes numa vista geral

Para além das relações entre pares de Pokémon, o sistema também é capaz de identificar Pokémon "fortes" numa perspetiva geral, com base nos seus atributos estatísticos. A regra SPIN responsável por essa inferência marca como pdx:isStrong true todos os Pokémon cujo valor de pdx:spAttack ultrapasse 120 — um limiar arbitrado com base na distribuição dos dados originais.

Estes dados são materializados no grafo e consultados no backend para realçar visualmente os Pokémons mais poderosos na interface de seleção (pokemon_selection.html). A consulta utilizada é a seguinte:

```
SELECT ?pokemon ?name
WHERE {
  ?pokemon pdx:isStrong true ;
    sc:name ?name .
}
```

Com isso, os Pokémons fortes são apresentados com um distintivo visual ("⚡ Strong") ao lado do nome, permitindo ao utilizador reconhecê-los imediatamente durante o processo de seleção para a batalha. Esta abordagem favorece a usabilidade e reforça o papel das regras SPIN como suporte à lógica de negócio.



8. Conclusões

Neste projeto, a aplicação POKED-X baseada na Web Semântica demonstrou com sucesso como a utilização de RDF, OWL e SPIN pode transformar um conjunto de dados estáticos numa plataforma dinâmica e interativa. Ao empregar estas tecnologias, foi possível enriquecer significativamente os dados originais sobre Pokémon, permitindo consultas inteligentes, inferências automáticas e uma integração harmoniosa com fontes externas como a DBpedia.

A criação de uma ontologia robusta permitiu modelar claramente os relacionamentos e características dos Pokémon, facilitando a execução eficiente de regras SPIN para inferências semânticas complexas, como identificar adversários fortes e fracos em batalhas simuladas. A aplicação beneficia não apenas da riqueza semântica proporcionada pela abordagem escolhida, mas também de um aumento substancial em termos de usabilidade e valor informativo para os utilizadores finais.

Além disso, a implementação de RDFa e microformatos garantiu que os dados estruturados da aplicação fossem visíveis e utilizáveis não apenas internamente, mas também por motores de busca e ferramentas externas.

Embora alguns desafios tenham surgido durante o desenvolvimento, como a integração coerente de dados externos, a manutenção da consistência interna do sistema, as soluções propostas mostraram-se eficazes e escaláveis e ter ideias de como melhorar o UI com os nossos dados sentimos que alcançamos o objetivo proposto embora algumas melhorias pudessem ainda ser implementadas.

9. Configuração para executar a aplicação

Pré-requisitos

- Python \geq 3.10 (recomenda-se 3.12)
- pip e virtualenv (ou `python -m venv`)
- GraphDB (ed. free) instalado localmente
- Git
- Navegador Web recente

Passo 1 — Obter o código-fonte

```
git clone https://github.com/ftferreira02/WS-Poked-X-Project.git
cd WS-Poked-X-Project
```

Passo 2 — Criar o ambiente Python

```
python -m venv venv          # cria ambiente virtual
source venv/bin/activate     # Mac/Linux
venv\Scripts\activate        # Windows
```

```
pip install -r requirements.txt
```

Passo 3 — Configurar e arrancar o GraphDB

1. Inicie o GraphDB (o script graphdb ou graphdb.bat, conforme o SO).
 2. Abra `http://localhost:7200` no navegador.
 3. Crie um repositório (*Poked-X*, regra de inferência: OWL 2 RL).
 4. No separador Import, escolha Upload RDF Files e carregue dois ficheiros, pela ordem:
 1. `ontology_new.ttl` → opção *“The default graph”*
 2. `pokemon-data-aligned-new.ttl` → *“The default graph”*
 3. `type_effectiveness_data.ttl` → *“The default graph”*
 5. Clique Import e aguarde até a barra ficar verde.
-

Passo 4 — Migrar a BD Django e lançar o servidor

```
python manage.py migrate    # aplica migrações locais (auth, sessions, ...)  
python manage.py runserver 8000
```

O Django responderá em `http://localhost:8000/`.

Passo 5 — Aceder à aplicação

- Interface Web → `http://localhost:8000/`
- GraphDB Workbench → `http://localhost:7200`

10. Anexos

Anexo A – Atributos RDFa e Microformatos utilizados na aplicação

Ficheiro	Tag HTML	Atributo	Valor
stats.html	div	typeof	pdx:Pokemon
stats.html	div	resource	http://poked-x.org/pokemon/Pokemon/{{ stats.id }}
stats.html	div	vocab	http://schema.org/ http://poked-x.org/pokemon/
stats.html	h1	property	name
stats.html	h2	property	identifier
stats.html	img	property	image
stats.html	p	property	description
stats.html	span	property	dbo:designer
stats.html	span	property	dbo:firstAppearance
stats.html	div	property	pdx:primaryType
stats.html	span	property	pdx:height
stats.html	span	property	pdx:generation
stats.html	span	property	pdx:weight
stats.html	span	property	pdx:isLegendary
stats.html	span	property	pdx:secondaryType
stats.html	div	typeof	pdx:PokemonStats
stats.html	div	property	pdx:stats
stats.html	div	property	pdx:hp
stats.html	div	property	pdx:attack
stats.html	div	property	pdx:defense
stats.html	div	property	pdx:specialAttack
stats.html	div	property	pdx:specialDefense
stats.html	div	property	pdx:speed
stats.html	div	property	pdx:weakAgainst

stats.html	span	typeof	pdx:Type
stats.html	div	property	pdx:strongAgainst
stats.html	span	typeof	pdx:Type
stats.html	p	property	pdx:sameTypePokemon
pokemon_search_form.html	div	typeof	SearchResultsPage
pokemon_search_form.html	div	vocab	http://schema.org/
pokemon_search_form.html	form	typeof	SearchAction
pokemon_search_form.html	form	property	potentialAction
pokemon_search_form.html	meta	property	target
pokemon_search_form.html	input	property	query-input
pokemon_search_form.html	div	property	filter
pokemon_search_form.html	a	property	filterValue
pokemon_search_form.html	select	property	sortOption
pokemon_search_form.html	div	typeof	ItemList
pokemon_search_form.html	div	property	itemListElement
pokemon_search_form.html	div	typeof	pdx:Pokemon
pokemon_search_form.html	div	resource	http://poked-x.org/pokemon/Pokemon/{pokemon.id}
pokemon_search_form.html	img	property	image
pokemon_search_form.html	p	property	identifier
pokemon_search_form.html	h5	property	name
pokemon_search_form.html	p	property	pdx:experiencePoints
pokemon_search_form.html	span	property	pdx:primaryType
pokemon_search_form.html	span	property	pdx:secondaryType
pokemon_search_form.html	nav	property	pagination
evolution_chain.html	div	typeof	pdx:EvolutionChainCollection



evolution_chain.html	div	vocab	http://schema.org/ http://poked-x.org/pokemon/
evolution_chain.html	h1	property	name
evolution_chain.html	p	property	description
evolution_chain.html	div	typeof	pdx:EvolutionNetwork
evolution_chain.html	div	property	pdx:evolutionNetwork
base.html	html	prefix	schema: http://schema.org/ pdx: http://poked-x.org/pokemon/ dbo: http://dbpedia.org/ontology/
base.html	body	typeof	schema:WebPage
base.html	div	property	schema:mainContentOfPage