



Universidade do Minho

UNIVERSIDADE DO MINHO
DEPARTAMENTO DE INFORMÁTICA
PROCESSAMENTO E REPRESENTAÇÃO DO CONHECIMENTO

Trabalho Prático Final

Henrique Ribeiro (PG38415)
Maria Pinto (PG39292)

Conteúdo

1	Introdução	2
2	Dados	3
3	Ontologia	4
4	Criação dos Indivíduos	7
5	API de dados	9
6	Interface	13
7	Conclusão	17

Introdução

O objetivo deste trabalho prático consistiu na criação de um *website*, com a formação de uma **ontologia** e criação de informação para demonstração no site. Optou-se por realizar um website em VUE sobre o tão famoso jogo, *League of Legends*.

Após a definição do tema partiu-se para a recolha de dados, para isto utilizaram-se *datasets* disponibilizadas pela *Riot Games*, a empresa criadora do jogo. Em seguida criou-se a estrutura da ontologia: Classes, *Data Properties* e *Object Properties*, o próximo passo focou-se na criação de indíviduos para povoar a ontologia.

Para a povoação, foram utilizados datasets recolhidos previamente, em formato *JSON*, com informação relativa aos *champions*, *items*, *runes* e *summoner spells* que constituem o jogo, e seguidamente converter os dados para formato *TTL*.

O objetivo final seria mostrar essa informação num **website**, de forma estruturada e informativa, possibilitando a apresentação de informações detalhadas sobre os vários componentes do *League of Legends*.

Dados

De maneira a obter os dados necessários sobre o jogo acedeu-se a um *website* criado pela *Riot Games*, <https://developer.riotgames.com/docs/lol>, que permite a *developers* acesso a várias informações relevantes sobre o jogo. No caso deste trabalho prático as informações que são relevantes consistem nas características e detalhes dos diferentes *champions* que são as personagens do jogo assim como itens, *runes* e *summoner spells* que os jogadores podem adquirir para tornar o seu *champion* mais poderoso.

A informação disponibilizada encontra-se em formato *JSON* acompanhada de várias pastas com imagens relativas às informações recolhidas.

📁	champions	03/07/2020 15:01	Pasta de ficheiros
📁	imageAssets	03/07/2020 15:02	Pasta de ficheiros
📁	images	03/07/2020 15:02	Pasta de ficheiros
📄	champion.json	03/07/2020 15:01	JSON file 185 KB
📄	championFull.json	03/07/2020 15:01	JSON file 6 061 KB
📄	item.json	03/07/2020 15:02	JSON file 413 KB
📄	linksjson.txt	03/07/2020 15:02	Documento de tex... 2 KB
📄	runesReforged.json	03/07/2020 15:02	JSON file 45 KB
📄	summoner.json	03/07/2020 15:02	JSON file 29 KB

Figura 2.1: Datasets e imagens retiradas do website

Com isto retiraram-se *datasets* com a informação sobre cada **Champion** (um total de 148 personagens). A cada personagem retirou-se a informação sobre as *tags* que identificam o tipo de combate de cada um, o tipo de ataque, as suas habilidades (4 habilidades únicas + 2 especiais que têm o nome de SummonerSpell), o tipo de energia (*resource*) e também as *skins*. Para além dessas, foram também retirados datasets em formato *JSON* sobre o conjunto de *runes* e também, sobre os itens presentes no jogo, que poderiam tomar o valor de componentes (itens que se podem tornar em outros itens) ou itens inteiros (itens que não se transformam em mais itens). Retiraram-se também várias imagens respetivas a cada *champion*, *item*, *rune* e *summoner spell*.

Ontologia

Feita a pesquisa, foi altura de começar a construir a ontologia no Protegê. Para isso, criaram-se 13 classes: Ability, Champion, ChampionInfo, ChampionSkins, ChampionStats, ChampionTag, Image, Item, ItemType, Resource, Rune e RuneTree. Adicionaram-se também as subclasses Passive, SummonerSpell e Ultimate à classe Ability e também as subclasses Component e FullItem à classe Item, sendo que, um item pode ter esses dois tipos.

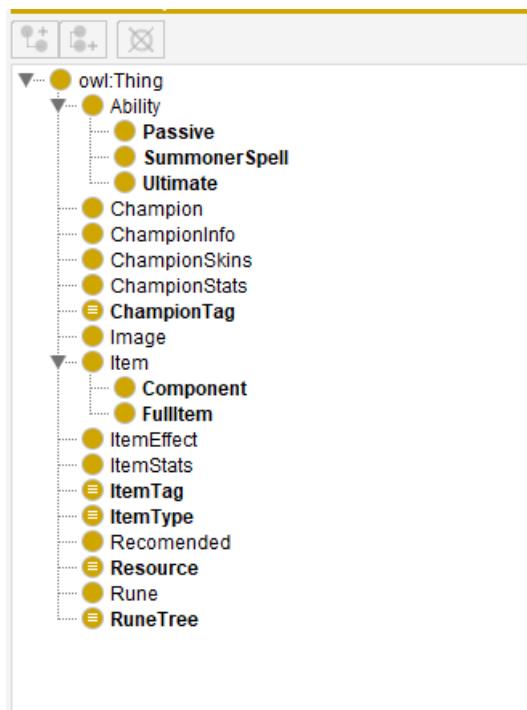


Figura 3.1: Ontologia criada no Protegê

Foram definidas instâncias para várias classes, como o ChampionTag, o ItemTag, o ItemType, o Resource e o RuneTree. Temos o exemplo para a classe ChampionTag, cujas instâncias criadas foram: Assassin, Fighter, Mage, Marksman, Support e Tank.

Seguidamente, foram criadas as *Object properties* relacionadas com as várias classes criadas. No total obtivemos um total de 22 *object properties*.

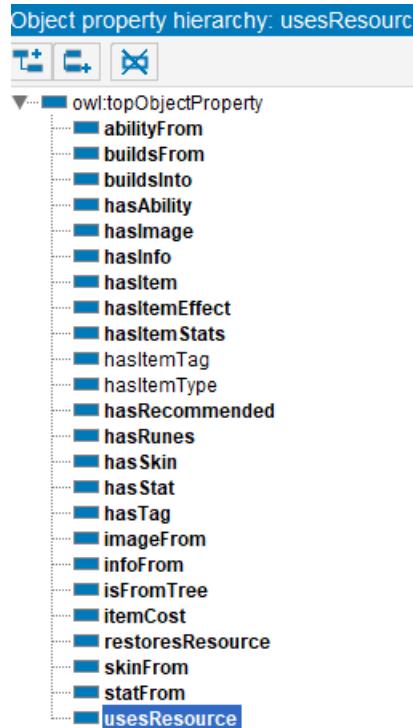


Figura 3.2: Object Properties criadas na Ontologia

Optou-se por criar bastantes ligações sendo que, por exemplo, o *Champion* tinha várias informações associadas, como informação de estilo de jogo, *skins*, *stats* e *tags*, tendo assim sido criadas as ligações *hasInfo*, *hasImage*, *hasSkin*, *hasAbility*, *hasStat* e *hasTag*. Para além das ligações com a classe *Champion*, foram também criadas entre as classes os Itens (Component e FullItem) assim como, entre as Runes e o seu respetivo nóculo da árvore a que pertencia (RuneTree).

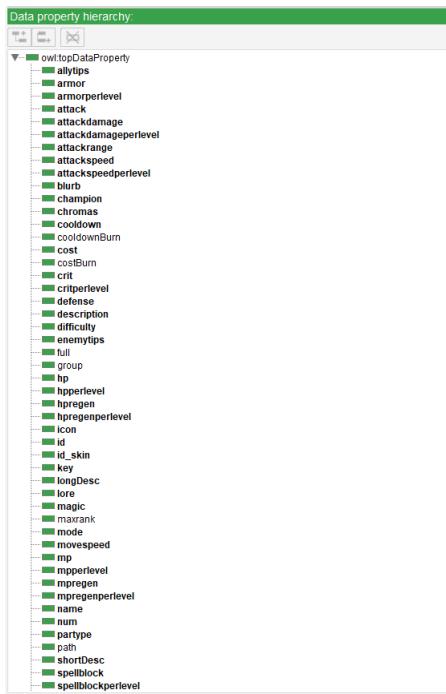


Figura 3.3: Data Properties criadas na Ontologia

Por último, foram criadas as *Data properties* para atribuir a cada classe, ou a mais do que uma classe, como o caso da key, como no exemplo que se segue:

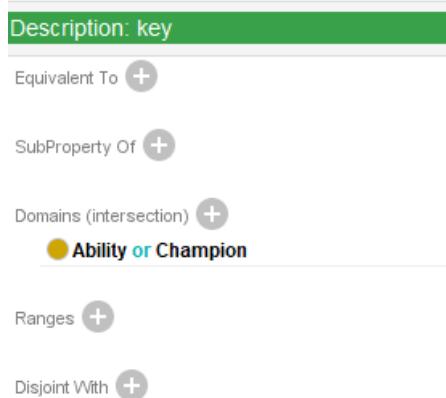


Figura 3.4: Data Properties das classes Ability e Champion

Criação dos Indivíduos

Sendo que tínhamos bastantes classes e informações sobre as mesmas, foram criadas no total 14 scripts na linguagem Python para transformar os dados retirados do dataset JSON (championFull.json, item.json, summoner.json e runesReforged.json) , para formato TTL de forma a povoar a ontologia e a inserir no GraphDB.

Utilizou-se a biblioteca json para ler o ficheiro JSON, de modo a retirarmos as informações pretendidas.

```
1  # -*- coding: utf-8 -*-
2  import json
3  import re
4
5  champions = []
6  champion = ""
7
8  # Reading data from the file
9  with open('../datasets/championFull.json', encoding='utf-8') as f:
10    data = json.loads(f.read())
11
```

Figura 4.1: Ler o ficheiro JSON

Foi criado um dicionário vazio, que fazia o update de todos os dados contidos no ficheiro JSON. Foram percorridos todos os valores dessa lista, e guardaram-se os valores contidos no "data" num dicionário chamado championsInfo. Foram novamente percorridos os valores contidos nesse dicionário e imprimiram-se os valores conforme o formato dos respetivos indíviduos da ontologia em TTL.

```

dict = {}
dict.update(data)
championsInfo = {}

for value in dict:
    if value == 'data':
        # Get only the champions stored on a new dict
        championsInfo = dict['data']
        # For each item in this new dict iterate
        # Meaning for each champion, get the skins
        for item in championsInfo:
            # championsInfo[item] gets the data for a specific champion
            print("### http://www.tartesdajulia.com/ontologies/LeagueOfLegends#" + item)
            print(": " + item, "rdf:type owl:NamedIndividual ,")
            print("           :Champion ;")
            print('           :hasAbility "' + str(championsInfo[item]['passive']['image']['full'][0:-4]) + ';"')
            for ability in championsInfo[item]['spells']:
                print('           :hasAbility :' + ability['id'] + ';"')
            for recommended in championsInfo[item]['recommended']:
                if recommended['map'] == 'SR' and recommended['mode'] == 'CLASSIC':
                    print('           :hasRecommended :"'
                           + recommended['title'] + ";")
            print('           :hasImage :' + item + ' Image ;')
            print('           :hasInfo :' + item + ' Info ;')
            for skins in championsInfo[item]['skins']:
                print('           :hasSkin :Skin' + skins['id'] + ' ;')
            print('           :hasStat :' + item + ' Stats ;')
            for tags in championsInfo[item]['tags']:
                print('           :hasTag :' + tags + ' ;')
            for allytips in championsInfo[item]['allytips']:
                print('           :allytips "' + allytips + '" ;')
            for enemytips in championsInfo[item]['emytips']:
                print('           :emytips "' + enemytips + '" ;')
            print('           :blurb ' + '""' +
                  championsInfo[item]['blurb'] + '""xsd:string ;')
            print('           :key "' +
                  championsInfo[item]['key'] + '""xsd:positiveInteger ;')
            print('           :lore "' +
                  championsInfo[item]['lore'] + '"" ;')
            print('           :name "' +
                  championsInfo[item]['name'] + '""xsd:string ;')
            print('           :partype "' +
                  championsInfo[item]['partype'] + '"" ;')
            print('           :title "' +
                  championsInfo[item]['title'] + '""xsd:string .\n')

```

Figura 4.2: Passar de formato JSON para formato TTL

Como se pode ver, no caso de extrair o valor do "hasAbility", foi necessário percorrer todos os valores do dicionário championsInfo. De seguida, foram percorridos novamente os valores nesse mesmo dicionário, mas desta vez, com a chave valor "spells". De seguida, foi feita a impressão do valor da chave "id", contida no campo "spells".

Foram feitos desta mesma maneira, os scripts para todas as classes necessárias para a povoação da ontologia (para cada tipo de informação do Champion, para as Runes, Items, Imagens e Summoner Spells).

API de dados

Na criação da API de dados, foram criados 5 ficheiros javascript para as routes: champions, items, ontology, runes e summonerSpells.

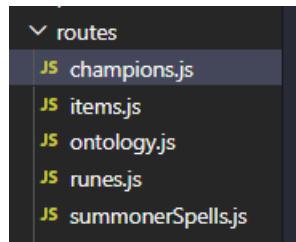


Figura 5.1: Rotas criadas na API

No caso dos champions, foram criadas várias rotas de modo a ir buscar todos os champions, os champions e o seu respetivo nome, cada champion individual, todas as informações individuais referentes a cada champion, entre outras, como podemos ver abaixo:

```
var express = require("express");
var router = express.Router();
var Champion = require("../controllers/champions");

router.get("/", function (req, res) { //Make this present some information...
    Champion.getChampions()
        .then((dados) => res.jsonp(dados))
        .catch((e) => res.status(500).send(`Error listing champions: ${e}`));
});

router.get("/main", function (req, res) { //Make this present some information...
    Champion.getChampionsAndTitle()
        .then((dados) => res.jsonp(dados))
        .catch((e) => res.status(500).send(`Error listing champions: ${e}`));
});

router.get("/:id", function (req, res) {
    Champion.getChampion(req.params.id)
        .then((dados) => res.jsonp(dados))
        .catch((e) => res.status(500).send(`Error listing champion: ${e}`));
});

router.get("/:id/full", function (req, res) {
    Champion.getChampionFull(req.params.id)
        .then((dados) => res.jsonp(dados))
        .catch((e) => res.status(500).send(`Error listing full details: ${e}`));
});
```

Figura 5.2: Rotas do Champion

Seguidamente foram criados os controllers para desta forma serem construídas funções de onde eram retiradas informações do GraphDB, através de queries SPARQL, que iriam buscar os valores pretendidos a serem usados em cada rota. Foi verificado que o repositório LoLDEVELOPMENT continha um total de 56,367 declarações.

```
var prefixes = `<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX noInferences: <http://www.ontotext.com/explicit>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX lol: <http://www.tartesajulia.com/ontologies/LeagueOfLegends#>
`;

var getLink = "http://localhost:7200/repositories/LoLDEVELOPMENT" + "?query=";

Champions.getChampions = async function () {
```

Figura 5.3: Acesso ao repositório criado no GraphDB

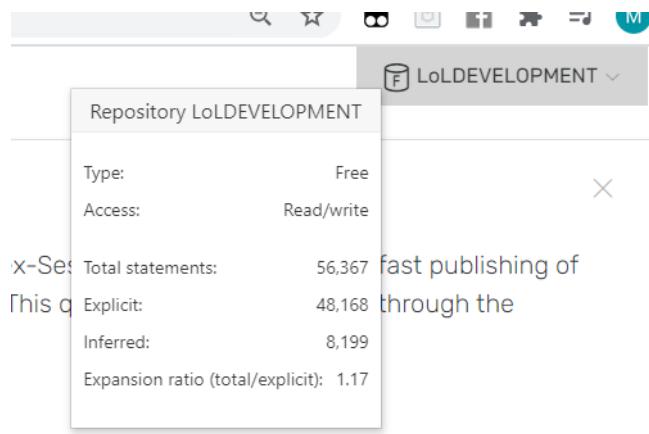


Figura 5.4: Informações do repositório criado no GraphDB

Foram criados 5 controllers no visual studio code: champions.js, items.js, ontology.js, runes.js e summonerSpells.js.

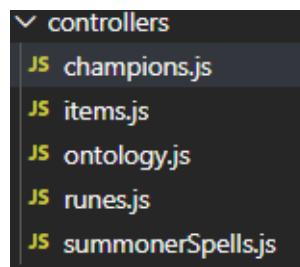


Figura 5.5: Controllers criados na API

Novamente, para o caso dos champions, podemos verificar que foram criadas várias funções de modo a retirar os dados pretendidos. Temos o exemplo da função getChampions, que nos dá o resultado de todos os champions presentes na Ontologia. A função getChampionsAndTile que nos mostra a informação referentes a todos os champions, incluindo a imagem e nome associado. Criou-se outra função chamada getChampion para obter cada champion individualmente, assim como, a função getChampionInfo para obter informação sobre cada champion individual, entre outras variadas funções de modo a ir buscar a informação necessária.

```

Champions.getChampions = async function () {
  var query = `select ?ind where {
    ?i a lol:Champion .
    bind (STRFTIME(STR(?i), '#') AS ?ind).
  } `;
  var encoded = encodeURIComponent(prefixes + query);

  try {
    var response = await axios.get(getLink + encoded);
    return myNormalize(response.data);
  } catch (e) {
    throw e;
  }
};

Champions.getChampionsAndTile = async function () {
  var query = `select ?ind ?imagePath where {
    ?i a lol:Champion .
    ?i lol:hasSkin ?skin .
    ?skin lol:hasImage ?image .
    ?image lol:path ?imagePath .
    FILTER regex(str(?image), "_0Tile").
    bind (STRFTIME(STR(?i), '#') AS ?ind).
  } `;
  var encoded = encodeURIComponent(prefixes + query);

  try {
    var response = await axios.get(getLink + encoded);
  
```

Figura 5.6: Algumas funções criadas no ficheiro champions.js

Foram mostrados só alguns exemplos para o caso dos champions, sendo que, foram criadas várias rotas para cada classe e também bastantes funções para cada classe, que foram bastante úteis para serem usadas na criação da interface e demonstração de informação.

Para mais informação sobre a API é possível consultar o ficheiro **API.md** que contem uma lista das rotas utilizadas assim como que tipo de informação cada rota contém.

The screenshot shows a dark-themed API documentation page with sections for different endpoints:

- /ontology**: Described as "Provides information about the ontology such as classes, individuals etc...". Sub-routes shown: /ontology, /classes, /individuals.
- /champions**: Described as "Provides information and details about champions or a specific champion.". Sub-routes shown: /champion, /:id, /skins, /abilities, /recommended, /images, /tags, /info, /details, /tips, /full → Full details about a champion, /abilities, /:id, /skins, /:id.
- /item**: Sub-routes shown: /items, /:id, /buildsInto, /buildsFrom.

Figura 5.7: Parte de API.md

Interface

Criada a API foi altura de criar a interface em VUE JS. Criou-se uma página inicial com o logótipo do jogo, "League of Legends", e vários botões para aceder à página dos champions, runes, itens e summoner spells.

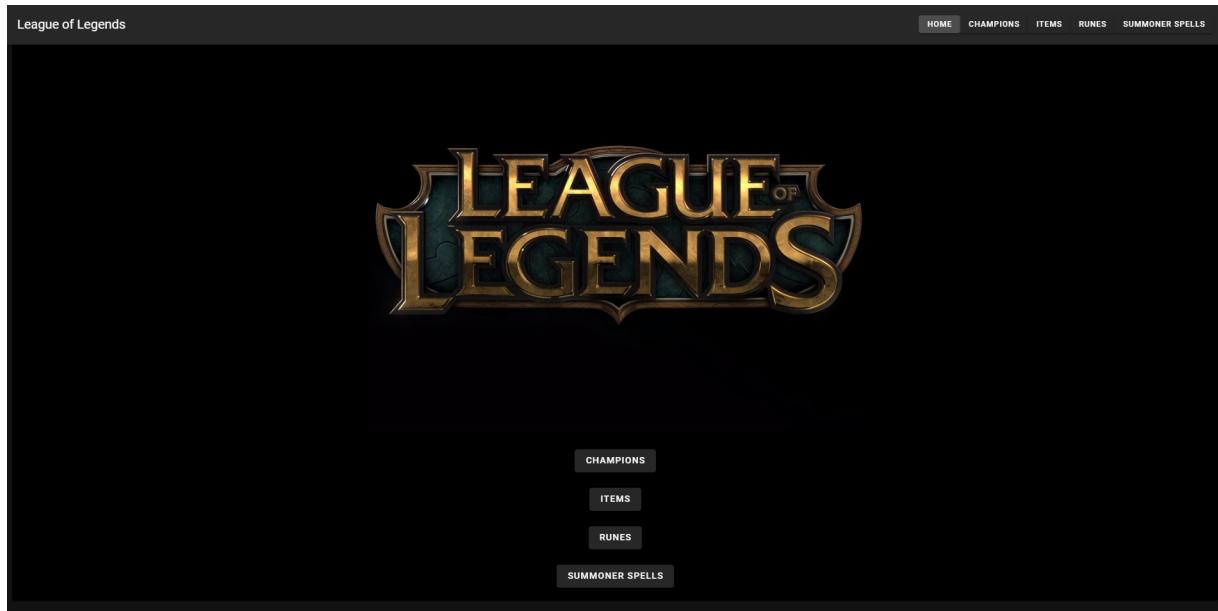


Figura 6.1: Página Inicial

Na páginas dos champions, construiu-se um template com a imagem de todos os champions, com a sua respetiva imagem e nome, através da rota /champion/main. Para aceder à página individual de cada champion, basta clicar com o rato por cima da imagem pretendida.

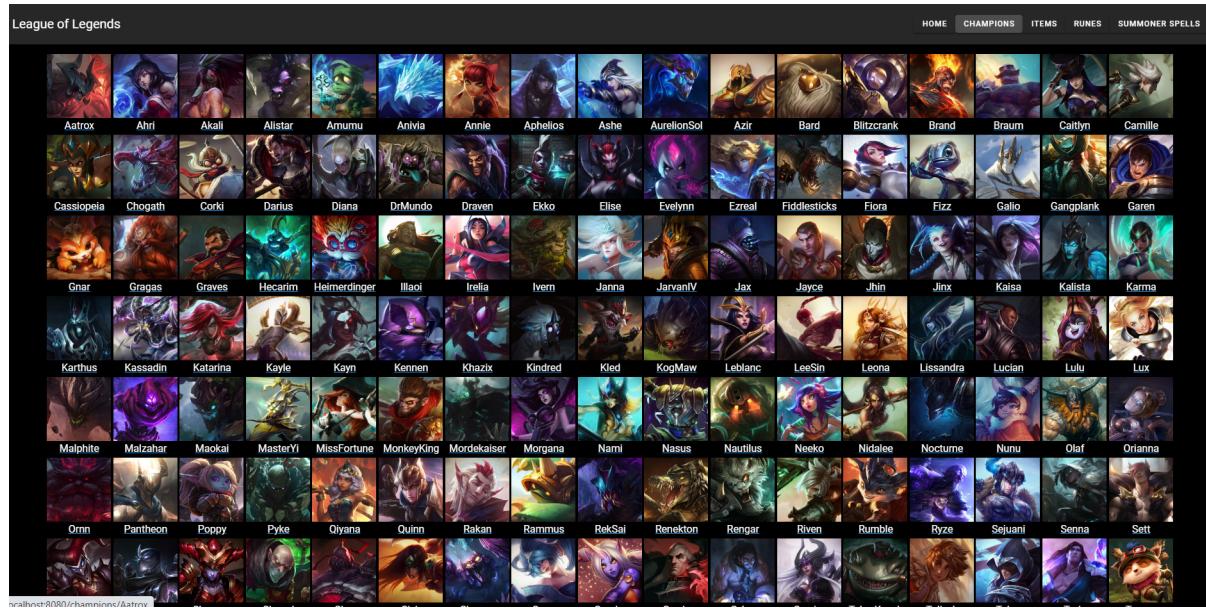


Figura 6.2: Página dos Champions

Na página individual de cada champion, foi utilizada a rota /champion/:id/full, onde se encontra toda a informação relativamente a cada champion. Foram retiradas informação da classe Champion, como o título, lore, partype, entre outras. Foi também retirada a informação da ligação "hasTag" e "hasInfo", onde foram mostradas todas as tags e informações relacionadas com esse champion. Encontra-se também um painel de expansão com as várias informações sobre as habilidades, usando assim a ligação "hasAbility". Nessas painéis, é possível visualizar informação correspondente às habilidades, como a imagem da habilidade, o nome e a descrição. Têm-se também acesso a um conjunto de imagens desse mesmo champion, através de um slide show.

A screenshot of the League of Legends individual champion page for Aatrox, the Darkin Blade. The page has a dark theme with a large image of Aatrox on the right. On the left, there is a sidebar with tabs for Overview, Abilities, and Items. The Overview tab is selected, showing Aatrox's title, class (Fighter/Tank), resource used (Blood Well), and stats (Attack: 8, Defense: 4, Magic: 3, Difficulty: 4). It also includes his lore: Once honored defenders of Shurima against the Void, Aatrox and his brethren would eventually become an even greater threat to Runeterra, and were defeated only by cunning mortal sorcery. But after centuries of imprisonment, Aatrox was the first to find freedom once more, corrupting and transforming those foolish enough to try and wield the magical weapon that contained his essence. Now, with stolen flesh, he walks Runeterra in a brutal approximation of his previous form, seeking an apocalyptic and long overdue vengeance. Below the lore, there are sections for Umbral Dash, The Darkin Blade, World Ender, Infernal Chains, and Deathbringer Stance, each with a small preview image and a description. At the bottom of the sidebar, there is a section for Summoner Spells.

Figura 6.3: Página Individual de cada Champion

Relativamente aos itens, é possível visualizar todos os itens deste jogo, e também filtrar por tipo: defense, attack, movement, tools e magic. Para além disso, pode-se clicar sobre o item e ver mais informações individuais sobre cada um, como o nome e a descrição. Nesse pop-up que abre quando clicamos sobre um item, é também possível ver os itens relacionados com esse mesmo item.

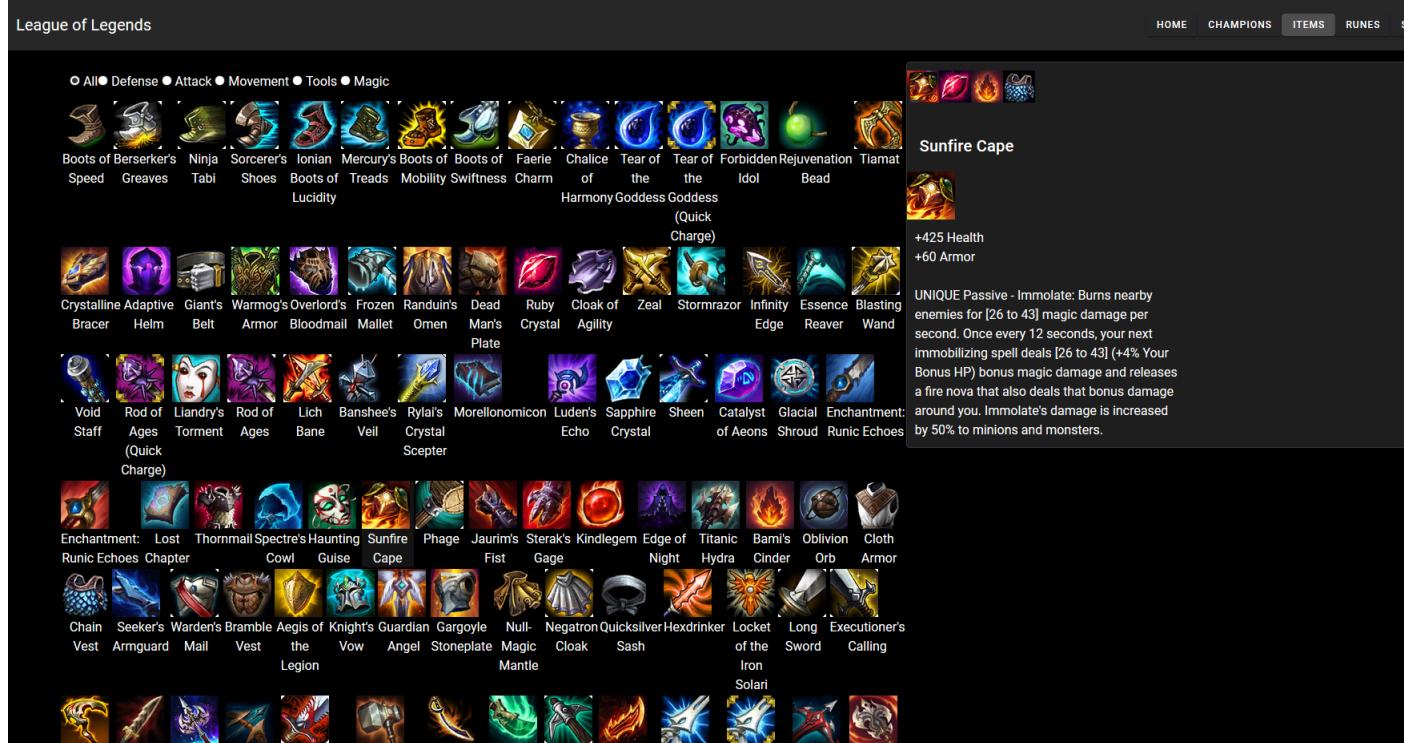


Figura 6.4: Página dos itens

Na páginas runes, foi utilizada a rota /rune/runeAndTree, para termos acesso a todas as runes e também à arvore que pertenciam. Dividiu-se a página em 5 tipos de árvores de runes: Domination, Precision, Inspiration, Resolve e Sorcery. Cada árvore tem a foto das suas runas associadas. É possível também clicar em cima de cada runa e abrir o pop-up com informação relevante sobre a mesma, como o seu nome e a sua descrição.

Finalmente, na página dos Summoner Spells, são mostradas todas as summoner spells existentes e também, clicando sobre o mesmo, é possível ver mais informação em detalhe.

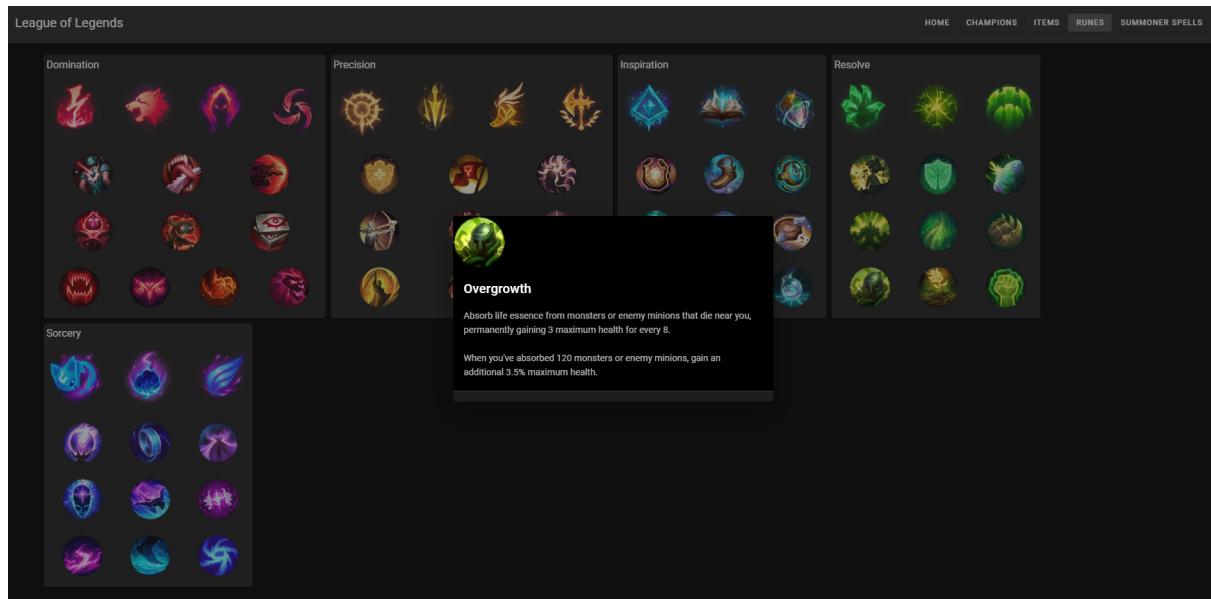


Figura 6.5: Página das Runas

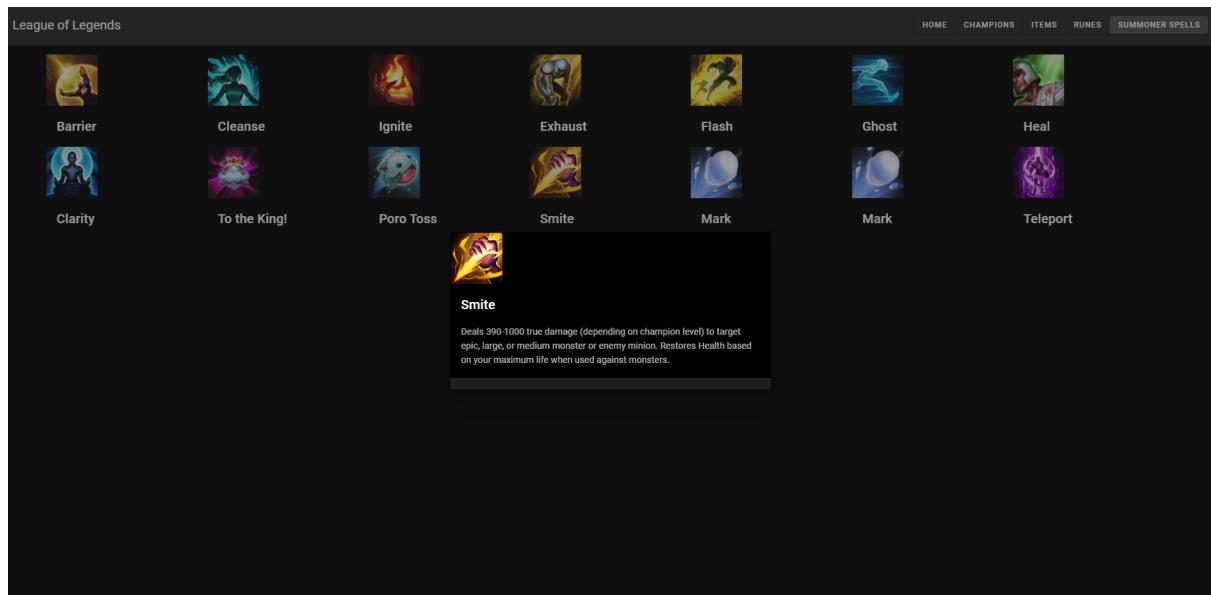


Figura 6.6: Página dos Summoner Spells

Conclusão

Com a realização deste trabalho, foi possível aprender muito mais sobre ontologias, desde ir buscar informação necessária, construir a ontologia em si e povoá-la, sendo que, tivemos que ir sempre adaptando a nossa ontologia à informação que se pretendia, e as ligações necessárias para completar a mesma.

Foi um trabalho bastante trabalhoso, mas também foi bastante necessário para nós aprendermos mais sobre ontologias, *VUE JS*, e também serviu para aprendermos mais sobre a linguagem *Python*.

Pensamos que foi uma mais valia para nós, sendo que deu para aprofundar os nossos conhecimentos, para além das aulas lecionadas, uma vez que esta ontologia continha bastante informação sobre o tema em questão.

Em suma, o trabalho desenvolvido foi concluído com sucesso, na medida em que, conseguiu-se realizar o pretendido, neste caso, criação de uma ontologia, povoação da mesma, criação de uma *API* e interface onde seriam mostradas as informações mais importantes.

A nível de melhorias, conclui-se que se poderia ter mostrado ainda mais informação e ainda adicionar mais páginas com informação que conseguimos retirar mas não nos foi possível desenvolver a mesma na interface.