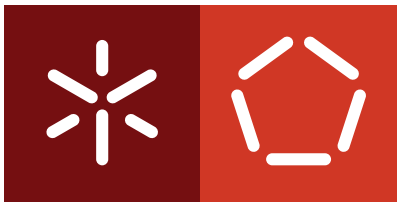
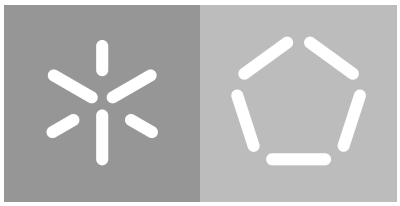


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Diogo Meira Neves

OntoReport:
Gerador de relatórios sobre ontologias

Outubro 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Diogo Meira Neves

OntoReport:
Gerador de relatórios sobre ontologias

Dissertação de mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação de
José Carlos Ramalho

Outubro 2019

Despacho RT - 31 /2019 - Anexo 3

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

AGRADECIMENTOS

A realização da dissertação, apresentada de seguida, contou com o contributo de várias pessoas, que, para eu chegar a esta etapa, e durante o seu desenvolvimento, me apoiaram e deram o seu contributo.

Em primeiro lugar quero agradecer aos meus pais, Ana Paula e Joaquim, sem eles nada disto seria possível. Agradeço tudo que fizeram por mim ao longo destes anos, mas sobretudo agradeço todo o apoio e força que sempre me deram.

Agradeço ao meu orientador, professor José Carlos Ramalho, por me ter orientado e apoiado ao longo desta dissertação, mas também por todo o conhecimento que me transmitiu.

Agradeço também a todos os professores do Mestrado Integrado em Engenharia Informática, da Universidade do Minho, por todo o saber que me foi transmitido.

Agradeço a toda a minha família, pelo apoio constante que me ajudou a chegar aqui.

Agradeço ainda a todos os meus amigos, não só por estarem sempre a meu lado e me terem ajudado e motivado quando precisava, mas também por todos os momentos que me proporcionaram ao longo destes anos.

Por fim, dedico este dissertação à minha madrinha, Amélia, que infelizmente faleceu e não teve oportunidade de me ver alcançar esta etapa. Obrigado por tudo.

Obrigado a todos!
Diogo Neves

Despacho RT - 31 /2019 - Anexo 4

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

RESUMO

Uma ontologia pode ser definida como um modelo de dados, que representa uma descrição de conceitos num determinado domínio. Esta tem como principal objetivo, aumentar a compreensão partilhada num determinado domínio, eliminando as diferenças, sobreposições e incompatibilidades em conceitos, estruturas, entre outros, criando assim uma especificação formal legível por um computador, e explícita, no sentido em que as entidades da ontologia são claramente definidas, distintas e inter-relacionadas entre si.

Tendo em conta o aumento exponencial de dados presentes na WEB, ontologias têm sido cada vez mais usadas como modelo de armazenamento de dados. Este aumento de usabilidade leva a que seja necessário o desenvolvimento de ferramentas que nos permitam criar/editar/analisar ontologias, ou seja, que nos permitam não só interagir com elas, mas também realizar um tratamento dos dados consequentemente recolhidos.

Assim, pretende-se nesta dissertação, desenvolver uma aplicação web capaz de gerar, através de uma especificação, um relatório, referente a uma determinada ontologia acessível através de um determinado endpoint. Por outras palavras, pretende-se a criação de uma aplicação que permita ao utilizador obter, da ontologia, apenas os dados que pretende, em vez de uma quantidade enorme de dados sem qualquer tratamento prévio.

Palavras-chave: Aplicação Web, Ontologia, Relatório de Informação, Web Semântica

ABSTRACT

An ontology can be defined as a data model, which represents a description of concepts in a given domain. Its main objective is to increase the shared understanding in a given domain, eliminating differences, overlaps and incompatibilities in concepts, structures, among others, thus creating a formal specification readable by a computer, and explicit in the sense that the entities of ontology are clearly defined, distinct and interrelated.

Given the exponential increase in data present on the Web, ontologies have been increasingly used as a data storage model. This increase in usability makes it necessary to develop tools that allow us to create / edit / analyse ontologies, that is, that allow us not only to interact with them, but also to perform a treatment of the consequently collected data.

Thus, this dissertation intends to develop a web application capable of generating, through a specification, a report, referring to a certain ontology accessible through a given endpoint. In other words, it is intended to create an application that allows the user to obtain from the ontology only the data he wants instead of a huge amount of data without any previous processing.

Keywords: Web Application, Ontology, Information Report, Semantic Web

CONTEÚDO

LISTA DE FIGURAS

LISTA DE EXCERTOS DE CÓDIGO

ACRÓNIMOS

A

API Application Programming Interface.

C

COM Component Object Model.

CORBA Common Object Request Broker Arquitecture.

CSS Cascading Style Sheets.

CSV Comma-separated values.

D

DCOM Distributed Component Object Mode.

DOM Document Object Model.

F

FOAF Friend of a Friend.

H

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

J

JSON JavaScript Object Notation.

M

MVVM Model-view-viewmode.

O

ORB Object Request Broker.

OWL Ontology Web Language.

R

RDF Resource Description Framework.

RDF(S) Resource Description FrameworkSchema.

REST Representational State Transfer.

RPC Remote Procedure Call.

S

SOAP Simple Object Access Protocol.

SPARQL Simple Protocol and RDF QueryLanguage.

U

UNICODE Universal Character Encoding.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

W

W3C World Wide Web Consortium.

WWW World Wide Web.

X

XML Extensible Markup Language.

1

INTRODUÇÃO

Neste capítulo é apresentado a contextualização, motivação, objetivos e metodologia seguida nesta dissertação. Além disso, é também apresentada a estrutura deste documento.

1.1 CONTEXTUALIZAÇÃO

Nos últimos anos, o aumento exponencial dos dados disponíveis, principalmente no domínio Web, tem conferido uma importância significativa às técnicas de organização da informação, tendo como principal foco um melhor tratamento e organização destes mesmos dados [1]. Se olharmos, por exemplo, para o Facebook, facilmente percebemos que o volume de dados, presente nos seus sistemas de informação, é bastante elevado, o que leva a que haja uma enorme organização destes, de modo a que possam ser consultados e modificados de forma eficiente.

Com o objetivo de concretizar este mesmo tratamento e organização, de dados, torna-se possível evidenciar uma atual existência de diversos tipos de técnicas que podem ser classificadas de diversas formas. Ontologia destaca-se como uma destas técnicas, sendo a mesma uma estrutura que se organiza a partir de conceitos e dos seus relacionamentos, que vem preencher uma das lacunas que encontramos na Internet com o fornecimento de uma estrutura compartilhável e semântica para um determinado domínio.

A principal razão para a escolha de ontologias, por parte da comunidade informática, passa por estas permitirem a partilha e entendimento de informação, num domínio, capaz de realizar a comunicação entre pessoas e computadores, sendo que, as mesmas são também desenvolvidas para facilitar a partilha e reutilização de informação [2]. Isto é, além de a informação contida nas páginas web ser apenas interpretada por pessoas, passa também a ser interpretada por computadores, que conseguem perceber o significado da informação e ajudar, de forma mais efetiva, na procura de informação.

Neste sentido é necessário, também, desenvolver ferramentas que permitam tratar e analisar ontologias, como por exemplo, a criação de uma ferramenta tecnológica que permita gerar relatórios sobre uma ontologia, através de uma especificação dada pelo utilizador, que irá ser o objecto de estudo nesta dissertação.

1.2 MOTIVAÇÃO

Com a enorme disponibilização, em tempo real, de todo, e qualquer, tipo de informação surgem dificuldades no seu tratamento assim como na sua apresentação, tornando-se assim um desafio para os programadores fazer chegar aos utilizadores a informação certa, no momento certo.

Hoje em dia, quando se pensa em ontologias, e se faz um breve pesquisa na Internet sobre ferramentas associadas, encontra-se sobretudo três tipos:

- Editores para criar e manipular ontologias;
- Editores para criar conteúdos *Resource Description Framework (RDF)*;
- Editores para criar query's *Simple Protocol and RDF QueryLanguage (SPARQL)*;

Estas ferramentas são adequadas quando a ação se resume à criação de ontologias e de documentos RDF.

No entanto, existe uma ausência de ferramentas que permitam explorar, de forma simples e prática, por todos os potenciais utilizadores, uma ontologia. A maior parte das ferramentas atuais, que permitem a criação de query's por parte do utilizador, assumem que o utilizador já tem um certo conhecimento base da ontologia em causa, sendo que um utilizador, sem conhecimento prévio, não consegue explorar uma ontologia sem receber uma grande quantidade de dados que não lhe interessa.

Neste sentido, a criação de uma ferramenta web, que permita uma interação dinâmica e iterativa, sobre uma ontologia, por parte do utilizador, torna-se particularmente útil e relevante.

1.3 OBJETIVOS

O principal objetivo desta dissertação consiste na realização de uma aplicação web, que permita, ao utilizador, criar uma especificação de um relatório sobre uma determinada ontologia.

Por outras palavras, esta aplicação irá permitir que um utilizador explore uma ontologia, que desconheça, sem que para isso tenha de efectuar/criar *querys* sobre ela. Irá funcionar como um sistema que se irá adaptar conforme as preferências/escolhas do utilizador. Para tal o utilizador terá de fornecer, à aplicação, um *endpoint* de uma dada ontologia, armazenada em *GraphDB*, que irá funcionar como ponto de partida da análise, e diz respeito a uma ontologia escolhida pelo utilizador. A partir deste *endpoint* irão ser recuperadas as classes, propriedades, atributos e indivíduos, da ontologia.

No final irá ser apresentada uma tabela de resultados, compilada, com a informação pretendida pelo utilizador. Esta tabela irá poder ser exportada e guardada pelo utilizador.

1.4 METODOLOGIA

A metodologia para a elaboração da presente dissertação segue sobretudo os seguintes passos:

1. Estudo teórico sobre ontologias e trabalhos desenvolvidos na área;
2. Estudo sobre os requisitos da aplicação;
3. Estudo sobre qual a estrutura da aplicação;
4. Estudo sobre o tipo e quais as ferramentas a utilizar no desenvolvimento da aplicação;
5. Desenvolvimento de um protótipo da aplicação.

1.5 ESTRUTURA DO DOCUMENTO

Nesta secção é descrita a estrutura do documento. Em todos os capítulos é apresentado uma breve introdução do que é abordado, terminando com um breve resumo do que foi apresentado.

Esta dissertação está assim dividida da seguinte forma: o capítulo atual, onde é feita uma pequena introdução ao tema abordado e, em seguida, a motivação que levou à realização deste trabalho assim como os objetivos a serem alcançados, terminando com a exposição da metodologia utilizada para o realizar.

O capítulo seguinte, capítulo 2, tem enfoque na apresentação do estado de arte, desta dissertação. Neste capítulo, o principal objetivo, passa pela descrição de alguns conceitos, necessários para a compreensão desta dissertação, como por exemplo o conceito de *web* semântica. Também neste capítulo são descritas algumas *frameworks* que provém de trabalhos anteriores realizados nesta área. Ainda no capítulo 2 é apresentado o GraphDB, como sendo uma ferramenta usada para armazenar ontologias, as quais poderão ser acedidas por esta aplicação, seguida de uma breve introdução à definição de serviços web, com ênfase na possíveis estruturas para a sua criação.

De seguida, o capítulo 3 tem por objetivo uma análise de quais os requisitos necessários à elaboração desta aplicação, apresentando-se também a estrutura, sobre a qual assenta o desenvolvimento desta aplicação, terminando com a exposição de quais as tecnologias escolhidas para a sua implementação.

O capítulo 4 apresenta toda a fase de implementação da aplicação, ou seja, é onde está descrito todo o processo de desenvolvimento, desde o lado do servidor até ao lado cliente.

Finalmente, o capítulo 5 contém a conclusão desta dissertação, onde é resumido todo o trabalho realizado e também apresentadas algumas linhas de trabalho futuro.

2

ESTADO DE ARTE

Neste capítulo, irá ser apresentada uma breve introdução ao termo de Web Semântica tal como alguns conceitos a ela relacionada, entre eles a definição de ontologia e algumas linguagens padrão para a sua implementação.

Por conseguinte serão apresentadas algumas ferramentas já desenvolvidas para a criação de ontologias, assim como para o seu tratamento, acrescentando-se ainda a apresentação do GraphDB, como ferramenta de armazenamento.

Por fim, é apresentada a definição de serviços web, com foco nas diferentes estruturas, possíveis de usar, na sua implementação.

2.1 WEB SEMÂNTICA

O *World Wide Web Consortium (W3C)* [1] , criado em 1994, é a principal organização internacional de padronização para a *World Wide Web (WWW)* e tem como objetivo conduzir a Web ao seu potencial máximo[3]. Neste sentido surgiu o conceito de Web Semântica. A Web Semântica, descrita por Tim Berners-Lee, não é uma Web separada mas sim uma extensão da atual, na qual é atribuída à informação um significado bem definido, permitindo uma melhor cooperação entre computadores e pessoas[4].

A partilha de informação, com o mesmo significado para computadores e humanos, requer a existência de marcadores semânticos padrão que sejam interpretados num vocabulário comum. Desta forma, a W3C propôs o aparecimento de novas tecnologias e linguagens tais como *Extensible Markup Language (XML)*, XML Schema, RDF, RDF Schema, SPARQL, e *Ontology Web Language (OWL)*, que permitam garantir a interoperabilidade e cooperação entre sistemas computacionais com o objetivo de desenvolver um modelo tecnológico para a partilha de conhecimento assistido por máquinas.

A arquitetura da Web Semântica pode assim ser vista como uma pilha de linguagens, tal como se pode observar na figura seguinte (Figura 2.1). Esta abordagem por camadas assume que as categorias superiores utilizam as capacidades das categorias inferiores e

1 https://www.w3.org/

permite ainda que as categorias inferiores possam ser aplicadas de modo estável, mesmo que as categorias superiores ainda estejam só formuladas.

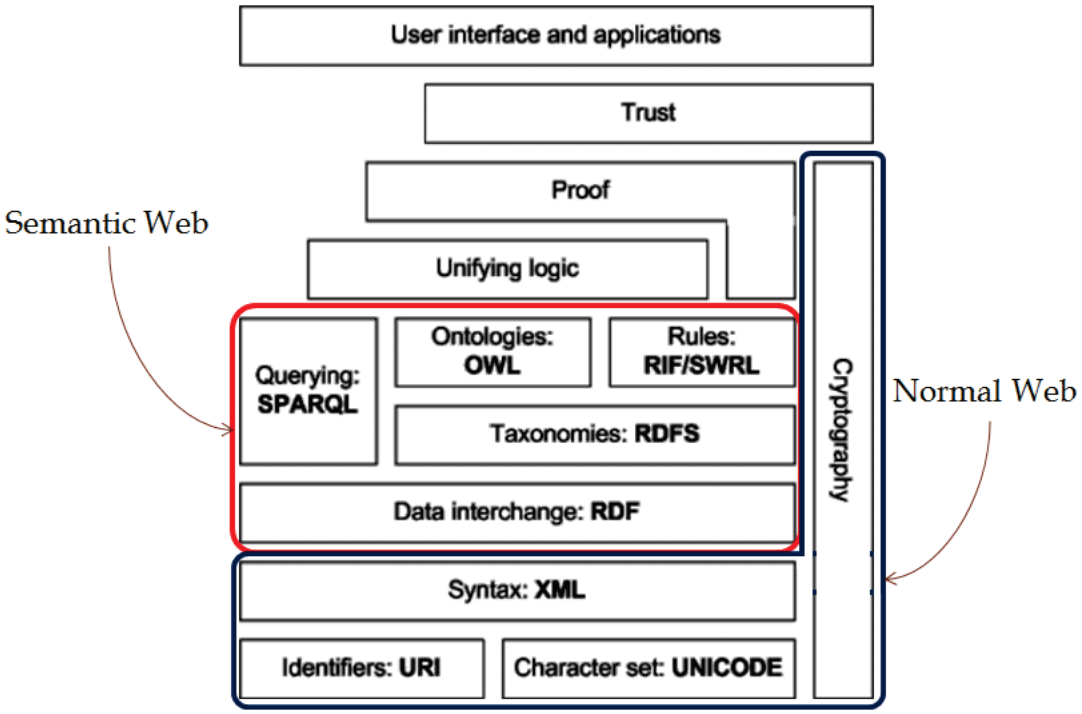


Figura 2.1.: Arquitetura da Web Semântica

Como podemos observar, Figura 2.1, a Web Semântica assenta sobre tecnologias que são conhecidas no domínio da Web. Sem estas tecnologias base não era possível implementar aplicações na Web Semântica.

Assim as tecnologias base são:

- *Uniform Resource Identifier (URI)* : padrão para a identificação de recursos. Cada recurso terá de ser identificado obrigatoriamente por um ou mais URI, que tem como função identificar e simultaneamente localizar o recurso na Web.
- *Universal Character Encoding (UNICODE)* : padrão que permite aos computadores representar e manipular, de forma consistente, texto de qualquer sistema de escrita.
- XML : permite a criação de documentos compostos por informação estruturada.

Na próxima secção irão ser apresentados os conceitos da camada intermédia, que permitem criar aplicações na Web Semântica.

2.1.1 *Ontologia*

As Ontologias são a espinha dorsal da Web Semântica. A definição mais comum de uma ontologia é que esta é "Uma especificação formal e explícita de um conceptualização". Dito isto, uma especificação explícita significa que, os conceitos e relações recebem nomes explícitos, e uma conceptualização refere-se, neste caso, a um modelo abstracto de como as pessoas pensam sobre as coisas no mundo, sendo geralmente restritas a um certo domínio. O facto de ser formal significa que, evita a ambiguidade, afastando-se assim de confundir definições incoerentes[5], e, neste contexto, significa ainda que passa a ser inteligível para as máquinas.

De uma forma mais simples, uma ontologia descreve uma hierarquia de conceitos, de um determinado domínio, relacionados através de relações, sendo que um domínio pode ser definido como uma parte da realidade que forma um assunto de ciência, tecnologia ou modo de uso[6].

Assim, as ontologias podem representar muitas coisas num determinado domínio. Coisas essas que são representadas como conceitos, na ontologia, e que são organizados em classes e subclasses. Cada classe, ou subclasse, irá estar associada a um conjunto de propriedades, que descrevem as características e atributos, bem como as várias restrições. Por fim pode ser composta por instâncias, ou indivíduos, que irão constituir a base de conhecimento da ontologia.

Para um melhor entendimento deste conceito, é representado de seguida um exemplo de uma ontologia, vizualizada através do WebWOWL [2].

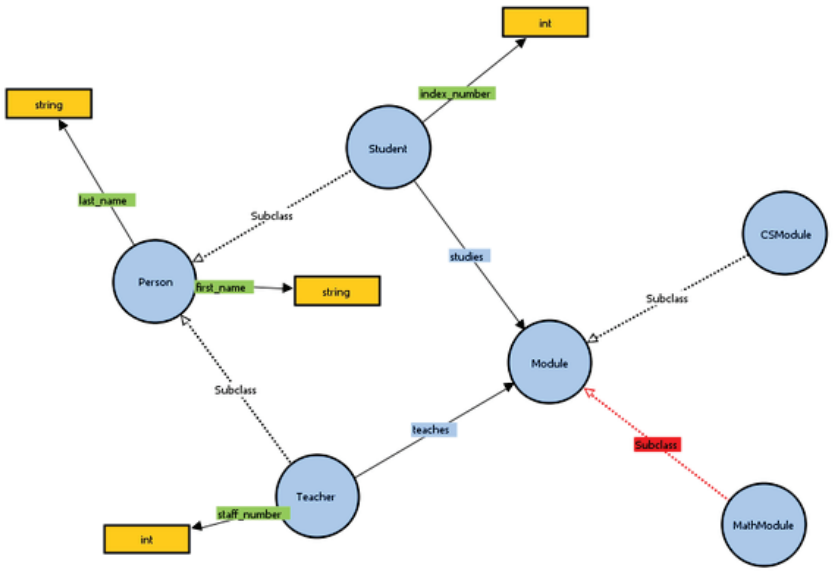


Figura 2.2.: Exemplo de uma ontologia

2 http://www.visualdataweb.de/webvowl/

A figura anterior, Figura 2.2, diz respeito a uma ontologia que podia facilmente armazenar, os conceitos básicos, de uma universidade. É constituída por duas classes: *Person* e *Module*, que irão ter duas subclasses cada uma. As subclasses *Student* e *Teaher*, pertencentes à classe *Person*, e as subclasses *CSModule* e *MathModule*, pertencentes à classe *Module*. Estas classes e subclasses irão assim armazenar, em grupos, os diferentes tipos de indivíduos da ontologia.

De seguida, é possível observar a existência de duas propriedades que definem o relacionamento entre classes e subclasses. A propriedade *Studies*, que se refere ao relacionamento entre um *Student* e um *Module*, que em português corrente pode ser traduzido para "um estudante estuda uma disciplina". A propriedade *Teaches*, que se refere ao relacionamento entre um *Teach* e um *Module*, que se traduz em "um professor lecciona uma disciplina".

É possível também, através da análise da figura 2.2, perceber a existência de propriedades que definem alguns tipos de dados, existentes em cada classe ou subclasse. Por exemplo, a propriedade *first_name*, associada a uma *Person*, significa que naquela classe vamos ter um atributo que contém o nome da pessoa.

Relativamente ao método sobre o qual são construídas as ontologias, não existe um método óptimo para a sua construção. De entre eles os que mais se destacam são:

- Ontology Development 101 : que consiste num guia de tarefas a executar, essas tarefas têm a função de ajudar no desenvolvimento da ontologia, pois orientam esse desenvolvimento até chegar ao objectivo final, uma ontologia completa e funcional [7] ;
- On-to-Knowledge : que consiste num estudo inicial de viabilidade [8];
- Methontology : que consiste num processo baseado na prototipagem de ontologias, baseando-se na evolução das actividades de desenvolvimento [9].

Todos estes métodos permitem a contrução de uma ontologia, no entanto, para o desenvolvimento desta aplicação, o método pelo o qual uma ontologia foi desenvolvida, não é relevante tendo em conta que esta irá funcionar sobre ontologias já desenvolvidas.

2.1.2 *Resource Description Framework (RDF)*

Metadados são estruturas de dados sobre dados que melhoram a descoberta e o acesso a dados. O uso de metadados, entre aplicações, requer convenções apropriadas para a semântica, sintaxe e estrutura. O RDF[3], desenvolvido pela W3C, é uma linguagem que permite a troca e reutilização dos metadados, permitindo assim a interoperabilidade entre sistemas computacionais, de modo a efetuarem trocas de informação na Web. O RDF não descreve a semântica mas fornece uma base comum para a expressar[10].

O RDF usa XML como uma sintaxe comum para a troca e processamento de metadados, e o seu modelo de dados pode ser descrito como um relacionamento entre 3 tipos de objetos:

- **Recurso** - tudo o que é descrito por uma expressão RDF, sempre designados pelo seu URI.
- **Propriedade** - um aspeto específico, característica, atributo ou relação usada para descrever um recurso. Cada propriedade é identificada por um nome.
- **Declaração** - um recurso em conjunto com uma propriedade e o seu valor.

Assim o RDF representa informação através de declarações numa estrutura do tipo Sujeito-Predicado-Objeto. Onde o sujeito é o recurso descrito na declaração, o predicado é a propriedade do recurso escolhido e o objeto é o valor dessa propriedade, constituindo assim os triplos RDF (s, p, o).

Esta estrutura pode ser observada na figura seguinte, Figura 2.3, onde temos a relação entre *John* e a sua data de nascimento, representado através de um triplo RDF.

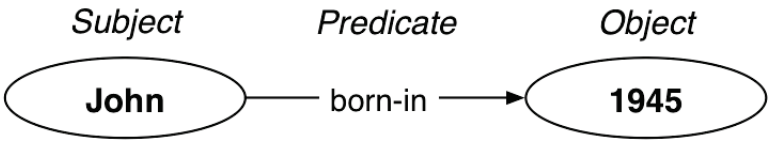


Figura 2.3.: Estrutura do triplo RDF (s,p,o)

3 https://www.w3.org/RDF/

2.1.3 *Resource Description FrameworkSchema (RDF(S))*

O RDF não tem mecanismos para descrever predicados nem suporta a descrição de relações entre predicados e outros recursos, assim surge o *Resource Description FrameworkSchema (RDF(S))*[4], como uma extensão ao RDF.

O modelo de dados do RDF(S) permite construir o conceito de classe dentro de um domínio de informação. O conceito de classe permite agrupar objetos que tem características e comportamentos semelhantes. O conceito de herança permite a objetos herdar algumas propriedades e comportamentos da classe mãe. Através do RDF(S) é assim possível construir ontologias.

No excerto representado de seguida é possível observar a relação entre a classe *horse* e a classe *animal*, representado em RDF(S), algo que não seria possível apenas com recurso a RDF.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base=  "http://www.animals.fake/animals#">
    <rdfs:Class rdf:ID="animal" />
    <rdfs:Class rdf:ID="horse">
        <rdfs:subClassOf rdf:resource="#animal"/>
    </rdfs:Class>
</rdf:RDF>
```

Na figura seguinte, Figura 2.4, é possível observar de forma mais clara a diferença entre RDF e RDF(S). Com RDF é possivel apenas definir tipos de um objecto, com o RDF(S) é possivel definir as diferentes relações entre eles. A parte inferior corresponde assim à parte RDF, e a parte superior ao RDF(S).

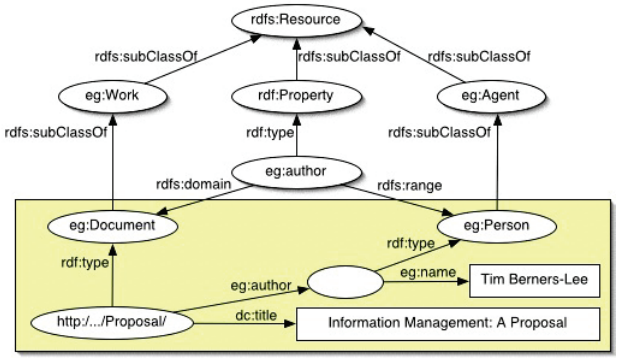


Figura 2.4.: Diferença entre RDF e RDF(S).

4 https://www.w3.org/TR/rdf-schema/

2.1.4 *Simple Protocol and RDF QueryLanguage (SPARQL)*

O SPARQL [5], é uma linguagem de consulta de declarações RDF, que faz também parte das recomendações do W3C. Esta linguagem permite fazer queries que consistem em padrões de triplos, conjunções e disjunções.[11]

A sintaxe da linguagem é semelhante à sintaxe SQL e disponibiliza um conjunto de comandos que permitem retornar dados mas não atualizá-los. Para atualizar ou acrescentar dados e necessário utilizar uma extensão denominada SPARUL.

Exemplo de uma query SPARQL, que permite obter a segunda página, de nomes e e-mails, de pessoas no arquivo *Friend of a Friend (FOAF)* de Tim Berners-Lee, já que cada página tem 10 pessoas:

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE { ?person foaf:name ?name .
    OPTIONAL { ?person foaf:mbox ?email }
} ORDER BY ?name LIMIT 10 OFFSET 10
```

2.1.5 *Ontology Web Language (OWL)*

A OWL é uma uma linguagem desenvolvida pelo W3C, com vista a possibilitar a publicação, extensão e partilha de ontologias na Web, utilizando para isso uma pilha de tecnologias, como por exemplo, RDF, RDF(S), SPARQL, etc.

A especificação de OWL contém três variantes:

- ***OWL Lite***: suporta as necessidades básicas de uma hierarquia de classificação e restrições simples. Permite assim a contrução de ontologias simples.
- ***OWL DL***: permite uma grande expressividade, e são as mais usadas atualmente.
- ***OWL FULL***: tem a expressividade da DL mas não impõe limitações sobre como os recursos se relacionam, sendo pouco usadas por ser computacionalmente pesadas.

A linguagem OWL é desenhada para ser utilizada por aplicações que necessitem de processar a informação, em vez de apenas a apresentar aos humanos.

5 https://www.w3.org/TR/rdf-sparql-query/

2.2 FERRAMENTAS DE MANIPULAÇÃO

Depois de uma breve introdução ao termo de Web Semântica, nesta secção será apresentada uma exposição de algumas ferramentas que trabalham sobre ontologias, disponíveis atualmente.

2.2.1 *Protégé*

Protégé [6] é um editor de ontologias baseado em Java[7] e uma *framework* de base de conhecimento mantido pela Universidade de Stanford. É também uma plataforma de código aberto de um conjunto de ferramentas para a construção de aplicações, orientadas ao conhecimento, sendo, sem dúvida, a ferramenta mais conhecida pela comunidade de ontologias.

Esta ferramenta disponibiliza uma interface intuitiva e robusta para o desenvolvimento de ontologias, através da manipulação dos seus diversos painéis, possibilitando o desenho hierárquico de classes, a descrição de propriedades, a construção de restrições e regras, a definição de conceitos, entre outros, implementando assim um conjunto de estruturas e ações que suportam a criação, visualização e manipulação de ontologias.[12]

Esta interface pode ser observada na figura 2.5.

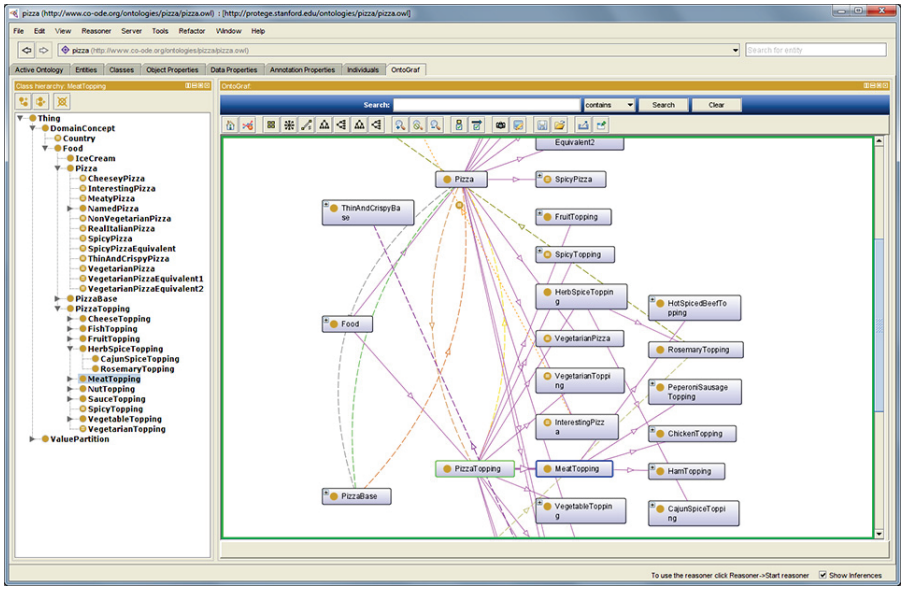


Figura 2.5.: Visualização do Protégé

Através da instalação de *plugins* disponíveis, é ainda possível alargar as suas funcionalidades a outras áreas de acordo com as necessidades específicas do utilizador.

6 https://protege.stanford.edu/
7 https://www.java.com/en/

OWLViz

OWLViz é usado como um *plugin* do Protégé. Permite que um utilizador possa visualizar as classes de uma ontologia e navegar entre elas. Este apresenta um sistema de cores, em que cada cor tem uma característica associada, e permite ao utilizador guardar uma dada visualização no formato de imagem.

ProtégéVOWL

ProtégéVOWL é outro *plugin* do Protégé usado para a visualização do esquema da ontologia. A sua interface está sobretudo dividida em 3 partes, a parte principal que permite a visualização do esquema ontológico, uma barra lateral com os detalhes do elemento selecionado e ainda por alguns controlos que permitem ajustar o layout do grafo. De um modo geral este plugin é muito parecido à ferramenta *WebVOWL*

2.2.2 *WebVOWL*

WebVOWL [8] não é um *plugin* como o ProtégéVOWL, mas sim uma ferramenta independente, sendo implementado em JavaScript. Em vez de estar vinculado a um analisador OWL em particular, o WebVOWL define um esquema *JavaScript Object Notation (JSON)* para as ontologias. Isso torna o WebVOWL independente de um analisador OWL em particular. No entanto, embora o WebVOWL seja implementado em JavaScript, a transformação da ontologia OWL na estrutura necessária JSON também pode ser realizada com outras linguagens de programação.[13]

A interface do utilizador do WebVOWL é semelhante à do ProtégéVOWL, consistindo na visualização de uma ontologia no ecrã principal, os detalhes aparecem numa barra lateral e um menu inferior contendo controlos, filtros e modos, tal como pode ser observado na imagem seguinte.

Os utilizadores podem interagir com a interface de maneira simples, podendo ampliar e desampliar, deslocar o fundo e mover elementos. Além disso, a barra lateral exibe os dados sobre a ontologia visualizada, como o título, namespace, autor e versão, bem como a descrição (se fornecida). Ela também lista anotações e propriedades personalizadas que não aparecem na visualização. Tal como pode ser observado na figura 2.6.

8 http://vowl.visualdataweb.org/webvowl.html

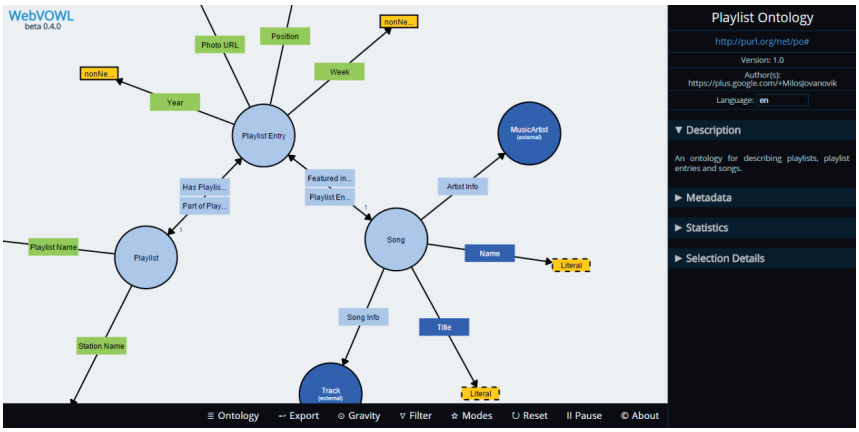


Figura 2.6.: Visualização do WebVOWL

2.2.3 *Apache Jena*

Jena [9] é uma outra *framework* que oferece um conjunto de ferramentas e bibliotecas que simplificam o desenvolvimento de aplicações, na *web semântica*, baseadas em ontologias, sendo um projeto de código aberto desenvolvido pelos pesquisadores da HP Laboratories na Bristol, cidade no Reino Unido.[14]

A arquitetura desta framework pode ser visualizada na figura seguinte.

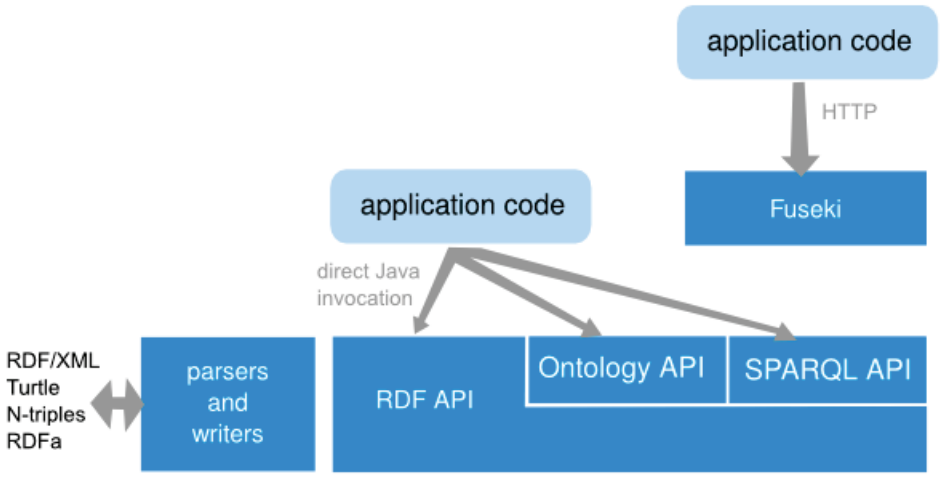


Figura 2.7.: Arquitetura Apache Jena

Esta *framework* funciona através da comunicação com 3 (*Application Programming Interface (API)*)'s. Uma RDF API que permite aceder aos triplos RDF e gráficos e ainda que seja

9 https://jena.apache.org/

possível adicionar ou remover triplos que correspondam aos padrões específicos de uma consulta. Uma SPARQL API que tem a responsabilidade de gerir o SPARQL, tanto para consulta como atualização. Uma Ontology API que permite o suporte de dois tipos de linguagens de ontologias, RDF(S) e OWL.

Além destes recursos, que são todos acedidos via API, hoje em dia torna-se necessário a publicação de dados na Internet. Dito isto, o Fuseki[10] é um servidor que permite esta publicação de dados e ainda a consulta e atualização dos modelos RDF.

A figura 2.8, apresentada de seguida, representa a interface web do Apache Jena Fuseki.

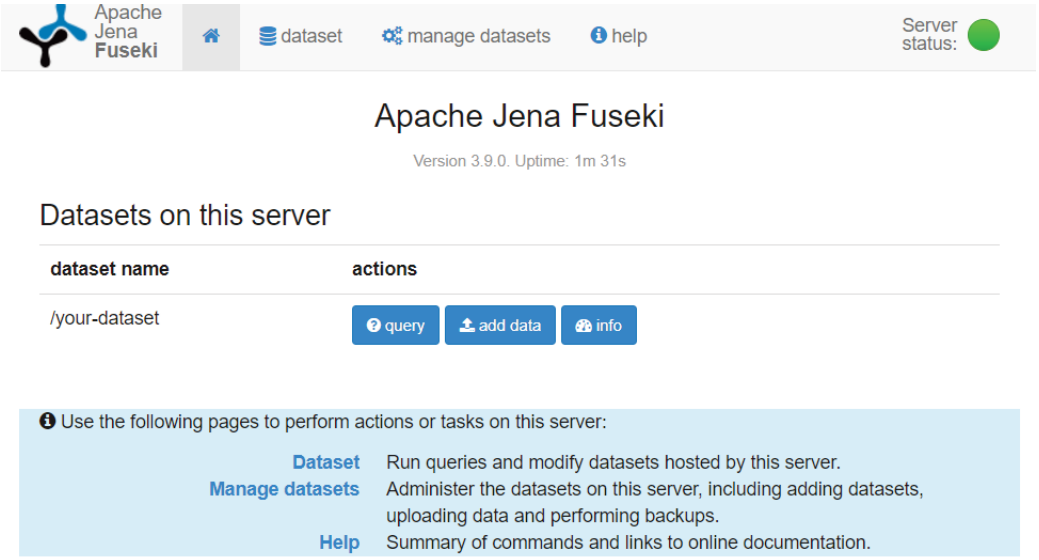


Figura 2.8.: Interface web do Apache Jena Fuseki

2.2.4 *Mobi*

Mobi [11] é uma plataforma gratuita e aberta à comunidade, que trabalha sobre dados criando um ambiente para que qualquer comunidade acelere a descoberta de conhecimento sobre os dados. A plataforma aplica as melhores práticas recomendadas pela W3C para apoiar o crescimento conhecimento em vários domínios.

Para a sua utilização o utilizador deve fazer o seu download, na página oficial, e correr o programa que irá colocar a correr um servidor em *localhost*. Uma vez iniciado a plataforma oferece ao utilizador uma interface web constituída pelos seguintes 7 módulos explicitados de seguida.

10 https://jena.apache.org/documentation/fuseki2/
11 https://mobi.inovexcorp.com/docs/

Editor de ontologias

O editor de ontologias, da plataforma Mobi, é baseado na web e permite ao utilizador um desenvolvimento local, de ontologias.

A visão geral, deste módulo, possui uma página que contém uma lista de todas as ontologias disponíveis no repositório local. Cada ontologia exibe os seus metadados e ainda um conjunto de acções, sendo elas, exclusão de uma ontologia ou gerir as suas permissões. Nesta página é ainda possível filtrar as ontologias, criar novas ontologias e ainda fazer o *upload* de uma determinada ontologia.

Depois de selecionada uma ontologia é aberto um editor que permite ao utilizador fazer as mais diversas alterações à ontologia.

Pedidos de merge

Este módulo permite aos utilizadores criarem representações, de longa duração, de uma fusão entre duas ramificações de um registo, que permite fornecer verificações e antes de se incluir alterações. Funcionando, assim, como reparador de colisões que possam aparecer em determinadas ocasiões.

Ferramenta de mapeamento

A plataforma Mobi oferece uma ferramenta de mapeamento que permite ao utilizador definir definições personalizadas para transformar os dados de entrada no modelo de dados RDF. Para fazer esta transformação a ontologia apenas precisa de estar carregada na plataforma Mobi.

A visão inicial deste modulo mostra uma pagina de seleção de mapeamento. Do lado esquerdo é apresentada uma lista de todos os mapeamentos disponíveis, no repositório, assim como os botões para criar e excluir repositórios. Ao clicar num determinado mapeamento é apresentado uma lista de informações do lado direito da página. As informações, do mapeamento, inclui a sua descrição, as classes e propriedades mapeadas, o título da ontologia de origem, juntamente com um menu suspenso com opções para edição, execução, duplicação e *download* do mapeamento.

Gestor de datasets

O gestor de datasets, da plataforma Mobi, permite aos utilizadores criar, editar, limpar e excluir conjuntos de dados dentro de um aplicação.

Esta página exibe uma lista de todos os conjuntos de dados, presentes no repositório local. Cada conjunto de dados exibe informação sobre os metadados e ainda um menu com os botoes editar, limpar e excluir.

Página de pesquisa

O módulo de pesquisa, presente na plataforma Mobi, permite que os utilizadores pesquisem e explorem, de forma rápida, os seus gráficos de conhecimento.

Esta página oferece uma interface intuitiva que permite aos utilizadores navegar pelas entidades definidas na ontologia. È também permitido ao utilizador criar e executar *querys* SPARQL.

Catálogo

Este módulo, presente na plataforma Mobi, permite aos utilizadores publicar dados, descrições de conjuntos de dados, análises e outros recursos.

Página de análise

Esta página de análise, presente na plataforma Mobi, permite que os utilizadores analisem os seus dados usando uma variedade de tabelas e gráficos, que podem ser guardados.

A visão desta página passa por uma lista de análises já efetuadas assim como um botão para a criação de uma nova.

A imagem seguinte representa a visão geral da plataforma Mobi.

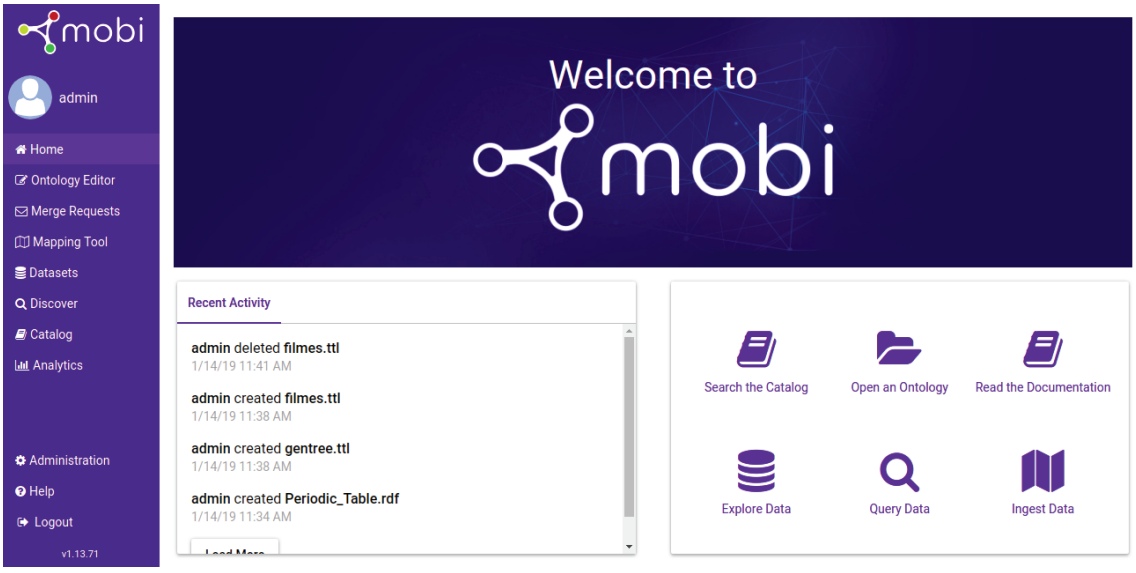


Figura 2.9.: Visão geral da plataforma Mobi

2.3 GRAPHDB

Nos dias de hoje os tipos de base de dados que podem ser divididos em duas categorias: bases de dados relacionais ou base de dados não relacionais.

Numa base de dados relacional os dados são organizados em tabelas, em que cada tabela representa um conjunto de dados num formato especifico. Dentro das tabelas os dados são organizados em colunas e em cada coluna contém um tipo de dado (strings, inteiros...).

Por outro lado, as bases de dados não relacionais, é um tipo de base de dados que não usa o esquema tabular para a organização dos dados. Em vez disso, o armazenamento de dados é adaptado consoante o tipo de dados que se pretende guardar. Por exemplo, os dados podem ser armazenados como pares de chave/valor simples, documentos JSON, ou como um grafo que consiste em arestas e vértices.

O *GraphDB*[12] enquadra-se num tipo de base de dados não relacional, e que, tal como o nome indica, se baseia no armazenamento de dados através de grafos. O armazenamento de dados em grafos baseia-se em apenas duas identidades: vértices e arestas. Em que os vértices representam as entidades e as arestas as relações entre entidades.

A figura seguinte, Figura 2.10, representa a visão de um elemento, o oxigénio, da tabela periódica armazenada em GraphDB. Através da sua expansão é possível observar as suas relações entre as outras entidades. Por exemplo, é possível observar que o oxigénio é um elemento não metálico, pertence ao grupo 16 e ao período 2, etc.

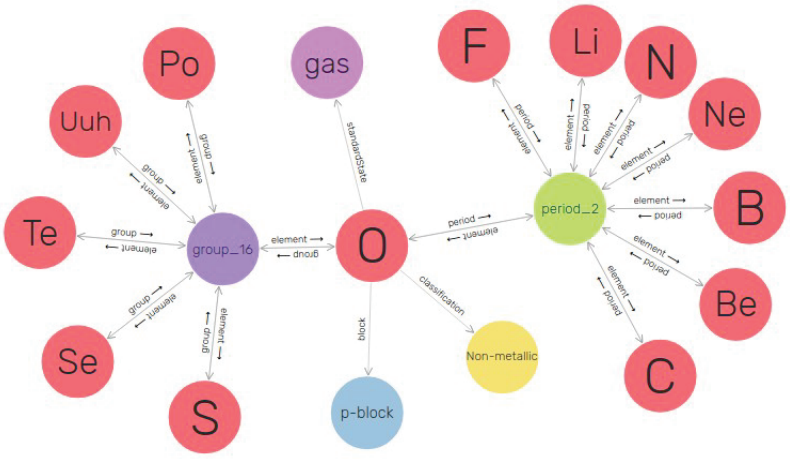


Figura 2.10.: Exemplo de uma ontologia representada no GraphDB

12 http://graphdb.ontotext.com/

2.4 SERVIÇOS WEB

Como já enumerado anteriormente, a taxa de utilização da *Internet* tem crescido imenso, o que leva a um aumento dos serviços web. Ora este aumento levou, inicialmente, a que surgisse a criação de formas de comunicação entre diferentes aplicações, de modo a que aplicações alojadas em máquinas diferentes sejam capazes de comunicar entre si. Este problema foi, inicialmente, resolvido com o desenvolvimento de 3 paradigmas:

- *Remote Procedure Call (RPC)* - este refere-se a um paradigma que atribui aos programas a capacidade de interação com outros através da rede, podendo também um programa invocar métodos num outro espaço de endereçamento [16]. Neste paradigma quando um método é invocado, as tarefas vão ser executadas na máquina de destino, enquanto o processo inicial é suspenso à espera da respetiva resposta, quando esta estiver disponível, é devolvida ao processo inicial e este continua a sua execução normalmente.

- *Common Object Request Broker Arquitecture (CORBA)* - este paradigma tem como objetivo permitir a comunicação entre diferentes tipos de aplicações, em diferentes ambientes e linguagens de desenvolvimento, através da Internet, utilizando para isso um objeto intermédio, *Object Request Broker (ORB)*, que permite aos utilizadores do sistema comunicar com estes sem conhecer a sua localização ou especificações [17].

- *Distributed Component Object Mode (DCOM)* - este paradigma, desenvolvido pela Microsoft, tem como objetivo a comunicação entre objetos alojados em diferentes computadores, quer localizados na mesma rede local quer em outros pontos acessíveis através da Internet [18]. Este funciona de forma semelhante ao CORBA, tendo um objeto intermédio, *Component Object Model (COM)*, que recebe pedidos dos clientes.

Apesar deste paradigmas resultarem, quando falamos de integração de software em ambiente local, quando surgiu a necessidade de comunicação entre aplicações alojadas em diferentes redes, nenhum dos modelos obteve sucesso, principalmente devido à limitação a nível de plataformas e da fraca segurança.

Surgiram assim novos serviços web para resolver este problema, sendo que, entre eles, existem dois que se destacam como sendo os principais:

- *Simple Object Access Protocol (SOAP)* - é um protocolo que tem por principal objetivo a troca de informações entre ambientes distribuídos e descentralizados [19].Através deste protocolo, a comunicação entre os diferentes componentes é realizada com base em mensagens XML, enviadas entre eles através de mensagens *Hypertext Transfer Protocol (HTTP)*. Sendo o XML, um formato standard, que pode ser interpretado por qualquer aplicação, independentemente do ambiente e linguagem de programação utilizada.

- *Representational State Transfer (REST)* - é um protocolo foi desenvolvido especificamente para a criação de serviços web, que tem por objetivo ser independente da plataforma, independente da linguagem, usando o HTTP como modo de funcionamento. A base deste protocolo assenta sobre 6 princípios [20]:
  - Arquitetura Cliente-Servidor - existe a separação das diferentes responsabilidades do sistema em módulos, como por exemplo, a separação entre a interface do sistema e os módulos de serviço;
  - Stateless - todos os pedidos são processados de forma independente, sendo que o servidor não tem conhecimento do estado da aplicação do lado do cliente;
  - Cache - que consiste no armazenamento de dados do lado do cliente, de forma a reduzir a necessidade de requisições ao servidor, otimizando o sistema;
  - Interface Uniforme - esta arquitetura tem foco na simplicidade, acessibilidade, interoperabilidade e na capacidade de descoberta de recursos;
  - Sistema em Camadas - a utilização de diferentes camadas permite ao sistema proteger determinados componentes, fazendo com que nem todos sejam acessíveis de modo público;
  - Code-On-Demand - que permite dotar o servidor da capacidade de transferência de código que pode ser utilizado no cliente.

2.5 Síntese

Tal como enunciado, foi apresentado neste capítulo o estado de arte desta dissertação. Começando pela apresentação de todas as definições, consideradas relevante para o entendimento desta dissertação, como por exemplo a definição de ontologia, expondo-se de seguida algumas ferramentas de manipulação, que existem atualmente neste domínio. Foi ainda apresentada a base de dados, GraphDB, que irá ser utilizada pela aplicação, para importar os dados das ontologias. Por fim, foram ainda apresentadas as diversas arquiteturas disponíveis para o desenvolvimento de serviço web, que será importante para perceber a arquitetura desta aplicação.

Dito isto, tendo já o conhecimento teórico necessário, os próximos capítulos irão focar-se mais no processo de desenvolvimento/implementação da aplicação. Começando com a explicitação do requisitos, onde irão estar mapeadas todas as cláusulas que a aplicação tem de cumprir, seguida das tecnologias a utilizar, desde a camada de *back-end* até ao *front-end*, terminando com o capítulo de implementação, onde irá estar explicado todo o trabalho realizado.

3

ANÁLISE DA APLICAÇÃO

O presente capítulo tem por objetivo uma análise detalhada dos requisitos necessários, seguindo-se da apresentação do modelo de estrutura que irá ser utilizado, para o desenvolvimento da aplicação, terminando com a apresentação de quais as ferramentas escolhidas, e utilizadas, para o desenvolvimento/implementação da mesma.

3.1 REQUISITOS

O objectivo principal desta dissertação, é a construção de uma aplicação que permita, ao utilizador,criar uma especificação de um relatório sobre uma determinada ontologia, logo é necessário levantar a lista de requisitos a cumprir.

Assim, os requisitos, da aplicação, passam por:

- Requisitos gerais
  - Universal - a aplicação deve permitir ao utilizador colocar qualquer *endpoint*, de uma ontologia, desde que esta esteja armazenada em GraphDB;
  - Ajustável - a aplicação deve permitir ao utilizador seleccionar e explorar quaisquer classes, propriedades ou indivíduos da ontologia, fazendo assim a filtragem que achar conveniente;
  - Interface intuitiva — a aplicação web deve ter uma interface que permita ao utilizador de forma simples e intuitiva, utilizar a aplicação;
- Requisitos específicos
  - Quantificação de dados - a aplicação deve apresentar ao utilizador a quantidade de dados do tipo seleccionado, presente na ontologia. Isto é, deve ser apresentado o número de classes, número de propriedades da classe e ainda a quantidade de indivíduos da classe;
  - Histórico de preferências - a aplicação deve permitir ao utilizador ver quais as suas escolhas até ao momento;

- Pesquisa através de um recurso - a aplicação deve permitir ao utilizador explorar um determinado recurso, apresentado como resultado;
- Apresentação de resultados - a aplicação deve mostrar ao utilizador uma tabela compilada com os dados pretendidos;
- Filtragem de resultados - a aplicação deve permitir ao utilizador filtrar qualquer valor na tabela de resultados;
- Histórico de querys efectuadas - a aplicação deve permitir ao utilizador, em qualquer momento, consultar quais as querys SPARQL, já efectuadas sobre a ontologia;
- Exportação de resultados - a aplicação deve permitir ao utilizador exportar os dados;
- Selecção de apenas uma classe - a aplicação só deve permitir a seleção de uma classe ao utilizador, não sendo possível explorar recursos de classes diferentes.
- Seleção de múltiplas propriedades - a aplicação deve permitir ao utilizador selecionar o número de propriedades de classe que pretender.
- Seleção de múltiplos indivíduos - a aplicação deve permitir ao utilizador selecionar o número de indivíduos que pretender.
- Conjugação de classe e propriedades - a aplicação deve permitir ao utilizador obter resultados só com base na classe e propriedades escolhidas.
- Conjugação de classe, propriedades e indivíduos - a aplicação deve permitir ao utilizador obter resultados com base na classe, propriedades e indivíduos escolhidos.

3.2 ESTRUTURA DA APLICAÇÃO

Depois de elaborada a lista de requisitos, esta secção tem por objectivo apresentar uma visão global daquilo que é a estrutura da aplicação.

Pela análise de requisitos, a solução escolhida, para a estrutura desta aplicação, foi uma arquitectura ”cliente-servidor”, baseado no modelo REST. O modelo ”cliente-servidor”tem por objectivo uma divisão de tarefas, de modo a reduzir a carga do sistema. O servidor irá oferecer uma série de serviços a um determinado utilizador, ou seja, executar uma tarefa pedida pelo utilizador e retornar os dados, por outro lado o cliente é o responsável por solicitar um determinado serviço, ao servidor, através de mensagens. Este é o modelo mais popular na comunidade web, devido ao facto de ser possível a execução de vários clientes sem pôr em causa o funcionamento do sistema.

Assim, a aplicação OntoReport, irá estar constituída pelo cliente web e pelo servidor. O cliente web irá comunicar com o servidor através de pedidos HTTP, que por sua vez, o servidor irá responder através de respostas HTTP. O servidor vai efectuar a ligação à base de dados, efectuando as *queries* necessárias, e retornar os resultados.

A figura 3.1, que se encontra de seguida, representa a estrutura da aplicação OntoReport.

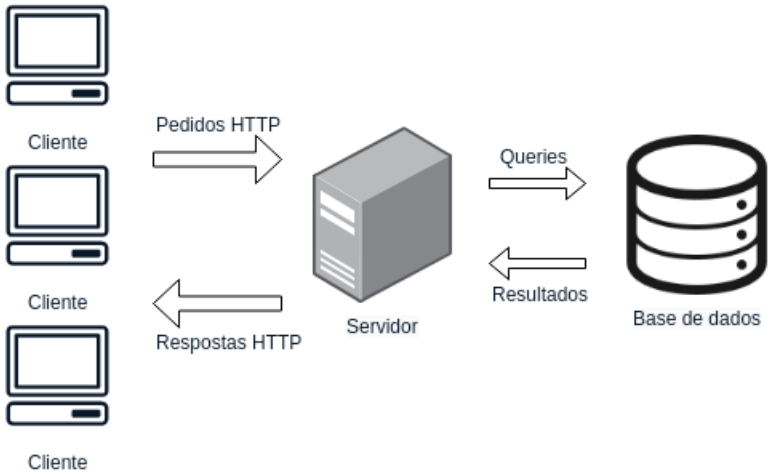


Figura 3.1.: Estrutura da aplicação

3.3 TECNOLOGIAS A UTILIZAR

Depois de elaborada a lista de requisitos e da concepção da estrutura da aplicação, nesta secção irão ser descritas quais as tecnologias escolhidas, e utilizadas, para a implementação da aplicação.

3.3.1 *Escolha ferramenta Back-end*

Uma vez que esta dissertação se baseia no desenvolvimento uma aplicação web, faz sentido utilizar uma plataforma que trabalhe sobre a "linguagem da web", *JavaScript*[1]. Para isso foi escolhido o Node.js [2], que consiste numa plataforma de desenvolvimento baseado no JavaScript Engine V8 da Google, que permite a linguagem JavaScript ser usada num servidor de uma aplicação web.

O principal objectivo do desenvolvimento de aplicações em Node.js passa por se poder criar aplicações rápidas e escaláveis, e ao mesmo tempo tenham boa performance gastando pouca memória.

1 https://www.javascript.com/
2 https://nodejs.org/en/

Para tal acontecer a plataforma baseia-se em 3 conceitos:

- I/O não bloqueante;
- Programação assíncrona orientada a eventos;
- Single-thread.

Ou seja, ao contrário de outras *frameworks* de desenvolvimento, o Node.Js não depende de multithreading para a execução de processos concorrentes, operando apenas em single-thread, através de uma abordagem baseada em eventos e de input e output não bloqueante. O facto de utilizar operações I/O não bloqueantes previne que a aplicação não fique bloqueada aquando a invocação de alguma operação. Isto significa, que o servidor nunca espera pela API para retornar dados. Este tipo de atributos leva a que o Node.Js seja leve e eficiente em comparação a outras *frameworks* que usem I/O bloqueante [21].

Na figura seguinte, 3.2, é possível observar o processamento de operações em Node.js.

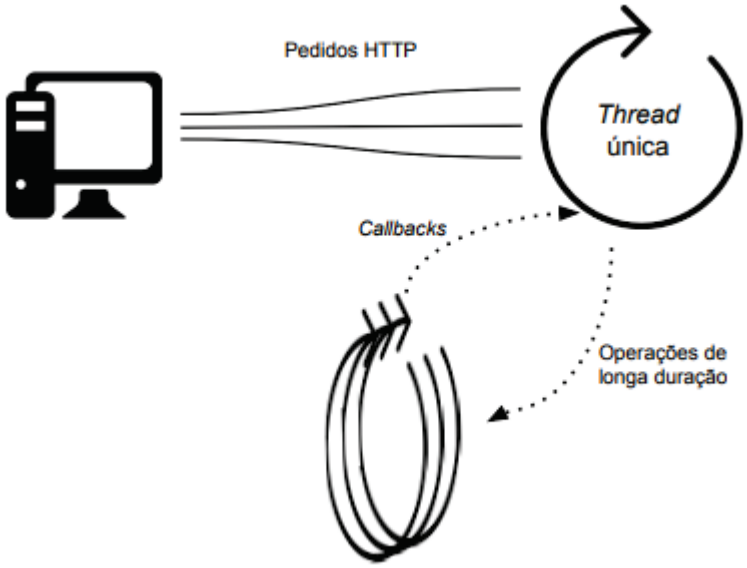


Figura 3.2.: Processamento das operações em Node.js

Portanto, várias operações I/O podem ocorrer em paralelo, e o respetivo *event callback* será invocado assim que a operação termine. Para que tal aconteça, é necessário um *event loop*. O *event loop* é um mecanismo que vai realizar duas tarefas num ciclo contínuo: deteção de eventos e desencadeamento de event handlers. O event loop é apenas uma thread a correr dentro de um processo, pelo que podemos tirar duas ilações: apenas um event handler estará a correr a uma determinada altura e qualquer event handler será executado até ao fim, sem que seja interrompido [22].

A construção deste servidor, Node.js, no contexto desta dissertação, irá ser apresentada no capitulo 4.

3.3.2 *Escolha ferramenta Front-end*

São bastantes as tecnologias existentes no mercado, para o desenvolvimento de interfaces de utilizador, sendo que muitas delas se baseiam em *Javascript*, como por exemplo, Angular.js [3], React.js[4], Vue.js[5], etc. Todas elas têm como objectivo auxiliar o processo de construção de interfaces de utilizador.

Para esta dissertação a ferramenta escolhida foi o Vue.JS. Vue.js é uma ferramenta cuja arquitectura se baseia parcialmente no modelo *Model-view-viewmode (MVVM)*, sendo utilizada para a construção de componentes reactivos. Nesta ferramenta, cada instância do Vue é um ViewModel, a *Document Object Model (DOM)* gerida pelas instâncias do Vue correspondem ao View e os Models são simples objectos de JavaScript, ou *data objects* [23].

Cada componente do Vue.JS segue a seguinte estrutura, que se divide em:

- template - neste caso composto por html, mas que podia ser facilmente utilizada outra linguagem como por exemplo PUG[6];
- script - onde pode ser definido algum comportamento do componente;
- style - onde pode ser definido o *Cascading Style Sheets (CSS)* dos elementos do componente.

. No excerto seguinte, excerto 3.1,é possível observar a estrutura de um componente Vue.js.

```
<template>
  <div class="exemplo">{{ msg }}</div>
</template>
<script>
  export default {
    data () {
      return {
        msg: 'Ola Mundo!'
      }
    }
  }
</script>
<style>
  .exemplo {
    color: red;
  }
</style>
```

Excerto de código 3.1: Exemplo de estrutura de componente Vue.js

3 https://angularjs.org/
4 https://reactjs.org/
5 https://vuejs.org/
6 https://pugjs.org/api/getting-started.html

A escolha desta ferramenta deve-se essencialmente a:

- Ser uma ferramenta sobre a qual já se possuí alguma familiaridade;
- Ser uma ferramenta que possuí uma documentação muito completa;
- Apresentar uma boa *performance* quando comparada a ferramentas similares[24];
- Ser uma ferramenta que está a ter um grande crescimento, sendo cada vez mais adoptada.

Tecnologias web utilizadas

Relativamente às tecnologias web escolhidas, para uso no servidor Vue.Js, foram as seguintes:

- HTML

  O *HyperText Markup Language (HTML)* é uma linguagem para o desenvolvimento de páginas web, ou por outras palavras, é a principal linguagem de marcação na WWW [25]. Através de HTML é possível criar páginas web, documentos HTML, que são interpretados pelos navegadores.

  A sua sintaxe é bastante simples, baseada em tags, tal como acontece no XML, em que cada tag irá representar um elemento da página, tal como pode ser observado na figura 3.3.

```
<html>
<head>
<title>Menu Html e Css</title>
<link rel="stylesheet" type="text/css" href="menu.css">
</head>
<body>
    <div id="menu">
        <ul id="linksmenu">
            <li><a href="">Home</a></li>
            <li><a href="">Postagens</a></li>
            <li><a href="">Contato</a></li>
            <li><a href="">Sobre</a></li>
        </ul>
    </div>
</body>
</html>
```

Figura 3.3.: Exemplo de estrutura html

Esta vai ser usada na construção da estrutura da página, dos componentes Vue.Js.

- JavaScript

  O JavaScript [26] é uma linguagem de programação interpretada e orientada a objectos. Inicialmente foi criada para ser executada apenas do lado do cliente, isto é nos navegadores, no entanto, hoje em dia também já é utilizado do lado do servidor, como por exemplo no Node.Js.

  O uso desta tecnlogia irá ser relevante tanto do lado do servidor Node.Js, como do lado cliente, servidor Vue.Js, visto ser através dela que irão ser analisados os dados retornados do servidor Node.Js.

  CSS

  O CSS, é uma linguagem que trata de descrever o aspecto dos elementos HTML [27]. Ou seja, através desta linguagem é possível manipular os vários elementos HTML, de forma a que o visual da aplicação fique de acordo como o esperado.

  A principal razão pela escolha desta tecnologia passa por:

  - Facilidade de uso
  - Mais simples que outras tecnologias semelhantes
  - Design responsivo
  - Não precisa de licença para utilização.

- JSON

  O JSON, é um formato para troca de dados. A sua estrutura de dados baseia-se, basicamente, na relação chave/valor, isto é, para cada valor representado atribui-se uma chave. Esta estrutura leva a que este formato seja facilmente interpretado por humanos e por máquinas.

  Neste caso, esta tecnologia, irá ser particularmente útil na troca de dados entre o servidor Node.Js e Vue,Js.

- HTTP

  O HTTP, é um protocolo de comunicação, que tem por objectivo efectuar a troca de dados entre o cliente e o servidor.

  Neste protocolo, o servidor Vue.Js envia pedidos, HTTP *requests*, para o servidor Node.Js, que após executar as tarefas necessárias, responde com um HTTP *response* com os dados.

3.4 SÍNTESE

Tal como enunciado, foi apresentado neste capítulo uma análise prévia da aplicação. O primeiro passo foi o desenvolvimento de todos os requisitos necessários, para que a aplicação cumprisse os objetivos iniciais. A lista de objetivos foi dividida em duas partes, a divisão entre requisitos gerais e requisitos específicos, de modo permitir uma melhor interpretação aquando o desenvolvimento da aplicação.

Seguidamente foi definida a estrutura base da aplicação, separando a aplicação em dois, o lado do servidor, responsável pelo acesso a dados, e o lado cliente, responsáveis pela interface e tratamento de dados devolvidos pelo servidor.

O último passo passou pela apresentação das tecnologias escolhidas para o desenvolvimento da aplicação

Dito isto, tendo já o conhecimento teórico necessário, os requisitos e estrutura, da aplicação, e as tecnologias a utilizar, no próximo capitulo irá ser apresentada a fase de implementação da aplicação, desde o lado do servidor até ao lado do cliente.

4

IMPLEMENTAÇÃO

Esta dissertação tem como base a criação de um *front-end* e um *back-end* de uma aplicação web, que possibilite a um utilizador explorar uma ontologia com as suas preferências. Assim, depois de descritos os requisitos, estrutura e tecnologias usadas pela aplicação, no presente capitulo irá ser explicado todo o processo de desenvolvimento/implementação da aplicação. Primeiramente, será apresentada a arquitectura geral da aplicação, onde está representada a introdução das tecnologias escolhidas em conjugação com a estrutura da aplicação. De seguida será explicado, em detalhe, as duas camadas, cliente e servidor, do sistema.

4.1 ARQUITECTURA GERAL

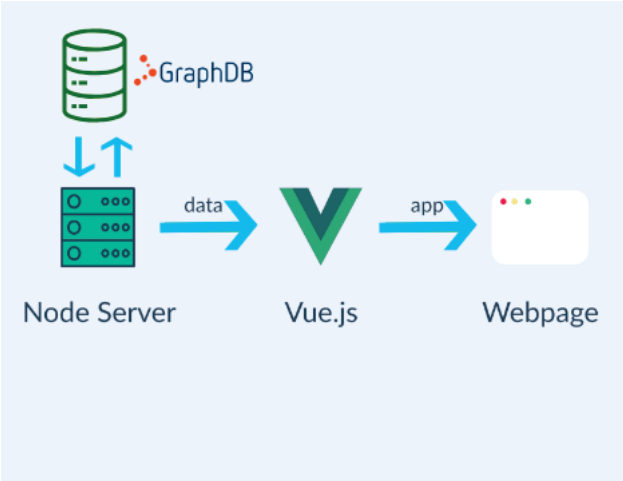


Figura 4.1.: Arquitetura Geral

Como referido no capítulo 3, esta aplicação irá funcionar através de um modelo "cliente-servidor". Através da visualização da figura 4.1, representada anteriormente, podemos verificar essa mesma arquitectura, que assenta em:

- **Servidor Node.js** - que tem por objectivo comunicar com a base de dados, armazenada no GraphDB, de forma a obter os resultados, da ontologia, pedidos pelo utilizador. Desta forma, este servidor funciona como um serviço disponível ao servidor *web*, que tem como principal objetivo a filtragem de resultados. Ou seja, a sua função é efectuar a "ponte"entre a interface e a camada de dados.
- **Servidor Vue.js** - que tem por objectivo fazer o tratamento dos dados, passados através do servidor Node.js, de modo a estes ficarem num formato legível e de fácil percepção, para o utilizador. Aqui é onde é criada toda a camada web do sistema, funcionando como o cliente da aplicação.

Por acréscimo faz ainda parte do sistema, embora de uma forma independente, isto é, não faz parte da aplicação, o GraphDB, que é onde estão armazenadas as ontologias a tratar por parte da aplicação. Ou seja, o servidor irá obter os dados, armazenados em GraphDB, através de um endepoint fornecido pelo utilizador.

4.2 SERVIDOR NODE.JS

Este servidor está organizado de modo a que seja possível haver uma divisão de código consoante a função, isto é, não se vai encontrar o código todo presente num ficheiro, o que ajuda, no futuro, à manutenção do mesmo.

Assim, a estrutura do servidor é a seguinte:

- *Routes*: onde está armazenado o ficheiro que trata do encaminhamento dos pedidos, do servidor web, ou seja da interface *web*, para os *Controllers*, consoante o pedido *Uniform Resource Locator (URL)*;
- *Controllers*: onde está o ficheiro responsável por lidar com os pedidos ao servidor, e devolver uma resposta;
- app.js e server.js: onde é inicializado o servidor

Depois de estar pensada a estrura do servidor, para a construção inicial deste recorreu-se a um comando chamado "npm-init", que quando executado cria o package.json da aplicação. Este ficheiro serve como ponto de partida e nele irão estar contidas todas as informações relevantes da aplicação, isto é, desde a descrição da aplicação até as dependências usadas.

O passo seguinte passou pela procura de um *package* que fizesse a consulta dos dados, armazenados em GraphDB, e para isso, o *package* escolhido foi o "sparql-client-2". Este *package*, que irá assim estar na lista de dependências do servidor, funciona como uma ponte entre este servidor e o GraphDB, levando com ele a *query* a ser executada e retornando os resultados.

Depois de criado o ficheiro package.json, e instaladas as dependências, o próximo passo foi a criação dos ficheiros "app.js"e "server.js", descritos de seguida, para a inicialização do servidor.

Por fim, foram então criadas duas pastas: routes e controllers, também descritas de seguida, onde está definida a camada lógica.

Sparql-client-2

A utilização deste *package* requer apenas dois elementos: o endpoint da ontologia e a query a ser executada. Com este package o servidor consegue estabelecer uma ligação à ontologia e realizar uma query sobre ela, retornando depois os resultados num formato escolhido pelo utilizador. No caso concreto da aplicação, foi definido o retorno em formato JSON, de forma a facilitar a sua leitura por parte do servidor web.

No excerto de código seguinte, excerto 4.1, é apresentado um exemplo de uma invocação, com a utilização deste package, de um pedido de dados sobre a DBpedia [1].

```
const client = new SparqlClient('http://dbpedia.org/sparql'),
            {defaultParameters: {
                format: 'json'
            }
        })
        .register({
            db: 'http://dbpedia.org/resource/',
            dbo: 'http://dbpedia.org/ontology/',
        })
        .query(SPARQL`
          SELECT ?leaderName
           WHERE {
             ${{db: cityName}} dbo:leaderName ?leaderName
           }`)

    (...)
```

Excerto de código 4.1: Exemplo de uso do Sparql-client-2

App.js

Tal como referido atrás este é o ficheiro mais importante da aplicação, estando nele definido todas as configurações do servidor. Começa por a invocação do package Express [2], seguido do redireccionamento das rotas para o ficheiro criado para o efeito na pasta routes.

1 https://wiki.dbpedia.org/
2 https://expressjs.com/

O Express é a *framework* padrão do Node.js que permite, entre outras coisas, receber os pedidos HTTP efetuados ao servidor e enviar a respetiva resposta.

Este ficheiro pode ser observado no excerto seguinte, excerto 4.2.

```
//Basic webserver
    const express = require('express');
    const app = express();
// Routes
    const ontoRoutes = require('./routes/index');
```

Excerto de código 4.2: Ficheiro App.js

Pasta Routes

Esta pasta contém apenas um ficheiro onde estão descritas as rotas da aplicação. Isto é, sempre que é invocado um *get* ou *post*, a este servidor, pedidos efectuado pelo servidor web desenvolvido, este ficheiro decide qual o método que vai ser invocado.

Todos os métodos, possíveis de invocar, encontram-se definidos no ficheiro da pasta *Controllers*, funcionando assim, este ficheiro, contido na pasta *Routes*, apenas como um filtro sobre qual o método a ser chamado.

Assim, as rotas criadas, para o efeito, foram:

- Rota para obter as classes da ontologia

```
    router.post('/classes', indexController.get_ontology_classes);
```

Excerto de código 4.3: Rota para obter as classes da ontologia

- Rota para obter as propriedades da classe escolhida

```
    router.post('/props', indexController.get_ontology_props);
```

Excerto de código 4.4: Rota para obter as propriedades da classe escolhida

- Rota para obter os indivíduos da classe escolhida

```
    router.post('/props_individuo', indexController.
       get_ontology_individuo_props);
```

Excerto de código 4.5: Rota para obter os indivíduos da classe escolhida

- Rota para obter as propriedades da classe e individuo escolhidos

```
router.post('/props_classe_individuo', indexController.
    get_ontology_classe_individuo_props);
```

Excerto de código 4.6: Rota para obter as propriedades da classe e individuo escolhidos

- Rota para obter os recursos da classes e propriedades escolhidas

```
router.post('/results', indexController.get_ontology_results);
```

Excerto de código 4.7: Rota para obter os objetos da classes e propriedades escolhidas

- Rota para obter os resultados de um recurso

```
router.post('/recurso_results', indexController.
    get_ontology_recurso_results);
```

Excerto de código 4.8: Rota para obter os resultados de um recurso

- Rota para obter os recursos da classes e propriedades escolhidas, com filtragem por individuo.

```
router.post('/results_individuo', indexController.
    get_ontology_individuo_results);
```

Excerto de código 4.9: Rota para obter os recursos da classes e propriedades escolhidas, com filtragem por individuo

Pasta Controllers

Esta pasta, contida também apenas por um ficheiro index.js, é onde está definida a ligação entre o servidor e a base de dados, GraphDB., usando para isso o *package* "sparql-client-2".

Para um melhor tratamento dos dados este ficheiro encontra-se dividido em vários métodos, todos eles com funções bem definidas. A principal diferença entre estes será a *query* utilizada por cada um, tendo em conta que irão ser criadas, dinamicamente, com as escolhas do utilizador.

Assim, os métodos criados, para o efeito, foram:

- get_ontology_classes - método que tem por objetivo obter as classes de uma ontologia.

  A query utilizada neste método acaba por ser bastante simples, tendo em conta que irá apenas recolher todas as classes da ontologia.

```
var query = "SELECT ?Classes WHERE { ?Classes rdf:type owl:Class . }
    ORDER BY ASC(?Classes)"
```

Excerto de código 4.10: Query para obter as classes de uma ontologia

- get_ontology_props - método que permite obter as propriedades através de uma classe escolhida.

  Neste método, é preciso ter em conta a escolha anterior do utilizador para a construção da query. Isto é, para o seu funcionamento é preciso saber a classe escolhida pelo utilizador. Esta escolha é enviada através do pedido HTTP efetuado pelo servidor Vue.js.

```
var query = "SELECT DISTINCT ?prop ?class where{ ?p a ?class.  FILTER
     (?class IN (<"+classe+">)) ?p ?prop ?o.} ORDER BY ASC(?prop)"
```

Excerto de código 4.11: Query para obter as propriedades de uma classe

- get_ontology_results - método que permite obter os resultados finais da classe e propriedades escolhidas, sem filtragem por indivíduos.

  Neste método, também é preciso ter em conta as escolhas anteriores do utilizador para a construção da *query*. Neste caso, conjugando a classe escolhida com as propriedades, também escolhidas, pelo utilizador.

```
var query = "SELECT DISTINCT ?id ?Classe " + colunas + "WHERE{?id a ?
    Classe.
FILTER(?Classe IN (<"+ classe+">)). "+ opcoes + " }GROUP BY ?id ?
    Classe " + colunas + "
ORDER BY ?id"
```

Excerto de código 4.12: Query para obter os individuos, de classe e propriedades escolhidas

- get_ontology_individuo_props - método que permite obter os indivíduos de uma determinada classe.

  Neste método, é necessário saber qual a classe escolhida pelo utilizador para recolher os indivíduos. Isto é, de maneira semelhante ao "get_ontology_props ", através da classe escolhida a *query* retorna os dados necessários.

```
var query = "SELECT DISTINCT ?id WHERE{?id a ?Classe. FILTER(?Classe
    IN (<"+classe+">)).}ORDER BY ?id"
```

Excerto de código 4.13: Query para obter os individuos de uma classe

- get_ontology_individuo_results - método para obter os resultados finais da classe e propriedades escolhidas, com filtragem por indivíduos.

  Este método acaba por ser muito semelhante ao get_ontology_results só que com o acréscimo da filtragem por indivíduos escolhidos. Ou seja, vai ser conjugado a escolha da classe, com as propriedades e ainda de indivíduos.

```
var query = "SELECT DISTINCT ?id ?Classe " + colunas + "WHERE{?id a ?
    Classe. FILTER(?Classe IN (<"+ classe+">)). FILTER (?id IN ("+
    individuos+")). "+ opcoes + " }GROUP BY ?id ?Classe " + colunas +
     " ORDER BY ?id"
```

Excerto de código 4.14: Query para obter os indivíduos, de classe, propriedades e indivíduos escolhidos

- get_ontology_recurso_results - método para obter resultados de um recurso especifico.

  Este método diz respeito a explorar um recurso depois de obtidos resultados, ou seja, as restrições irão ser apenas o próprio recurso e propriedades escolhidas.

```
var query = "SELECT DISTINCT ?id " + colunas + "WHERE{ FILTER(?id IN
    (<"+ recurso+">)). "+ opcoes + " }GROUP BY ?id" + colunas + "
    ORDER BY ?id"
```

Excerto de código 4.15: Query para explorar um determinado recurso

Todos os métodos criados seguem uma lógica muito semelhante. A primeira parte do método consiste na ligação à base de dados, através do endpoint dado pelo utilizador, com a ajuda do "sparql-client-2". Depois de a ligação feita é executada uma *query* sobre os dados. Esta *query* é criada dinamicamente com base nas escolhas do utilizador, e assim diferente de método para método, consoante a necessidade de cada um.

Os dados obtidos, por cada método, irão ser retornados ao servidor *web*, que irá efetuar o tratamento necessário para a sua apresentação. O formato de retorno deste é o JSON.

Como exemplo, pode ser observado no Anexo A, o código completo, do método que diz respeito aos dados filtrados por classe, propriedades e indivíduos, ou seja, ao método "get_ontology_individuo_results".

4.3 SERVIDOR VUE.JS

Esta aplicação é constituída também por uma interface web, desenvolvida com recurso ao Vue.js. Neste caso, a aplicação web irá se basear sobretudo em 4 páginas web principais, que irão interagir e comunicar entre si.

O Vue.js, de acordo com a documentação, é composto essencialmente por componentes, que no seu conjunto irão definir a interface. Cada componente é constituído por ações, disponíveis ao utilizador, que quando despoletadas resultam na execução de algo. Os componentes podem comunicar entre si de duas formas:

- Eventos: um componente filho, envia um evento para o componente pai a informar de alterações que tenha sofrido;
- Props: em o componente pai pode passar dados para o componente filho.

No caso concreto desta aplicação, os 4 componentes criados foram então os seguintes:

- Endpoint.vue (Figura 4.2): página inicial da aplicação e onde o utilizador pode inserir o *endpoint* da base de dados a utilizar;

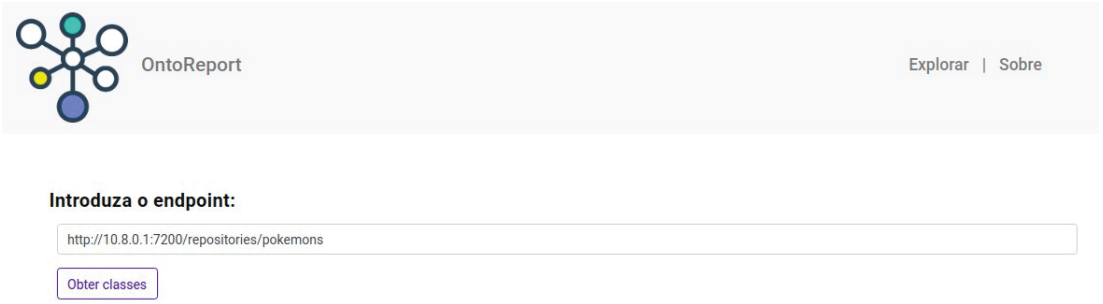


Figura 4.2.: Página inicial - endpoint

Nesta página o utilizador tem de preencher, obrigatoriamente, o campo destinado ao *endpoint*, da ontologia. Se o utilizador não o preencher não lhe é permitido continuar.

- Classes.vue (Figura 4.3): página onde o utilizador pode escolher a classe sobre a qual pretende obter informação, sendo possível também fazer desde logo uma filtragem pelos indivíduos;

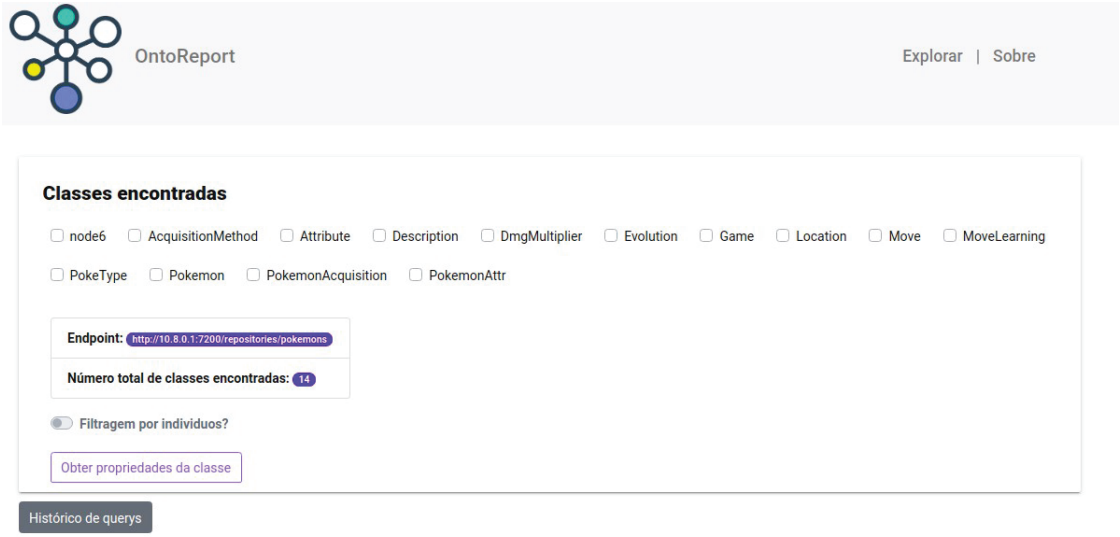


Figura 4.3.: Página seleccionar classe

Nesta página não é permitido ao utilizador seleccionar múltiplas classes. Depois de selecionada a classe o utilizador tem a opção de avançar logo, para a escolha de propriedades, ou fazer um selecção prévia de quais os indivíduos que deseja, tal como demonstrado na Figura 4.4.

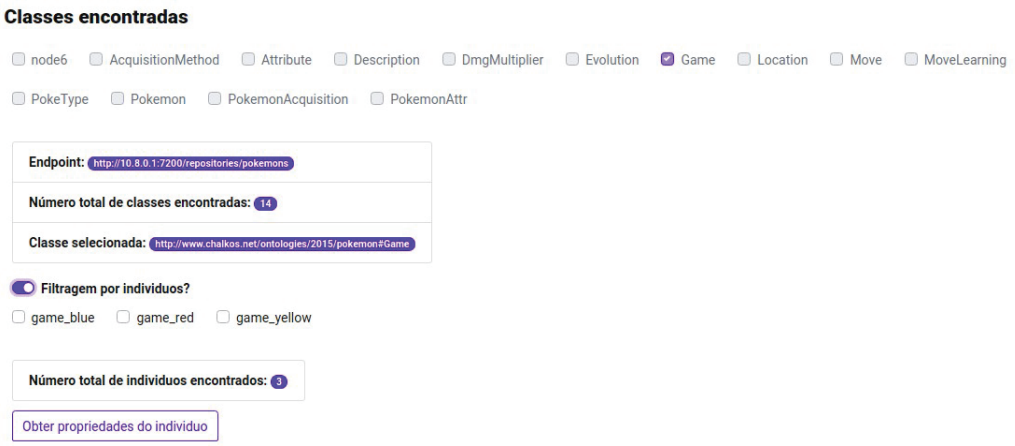


Figura 4.4.: Página seleccionar individuos da classe

- Propriedades.vue (Figura 4.5): depois de escolhida a classe o sistema irá mostrar ao utilizador quais as propriedade daquela classe, sendo possível seleccionar aquelas que pretende;

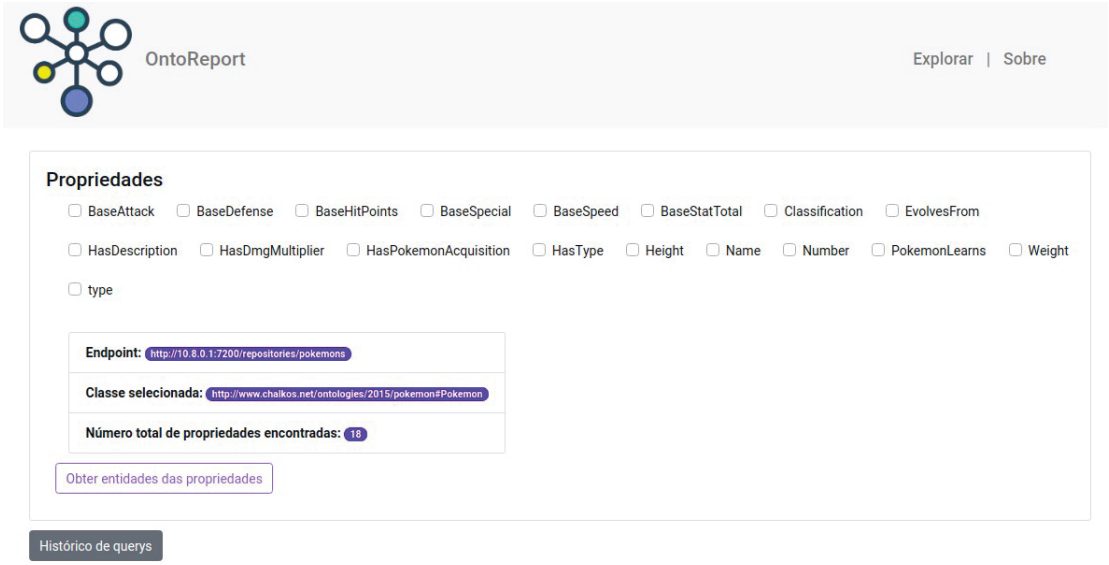
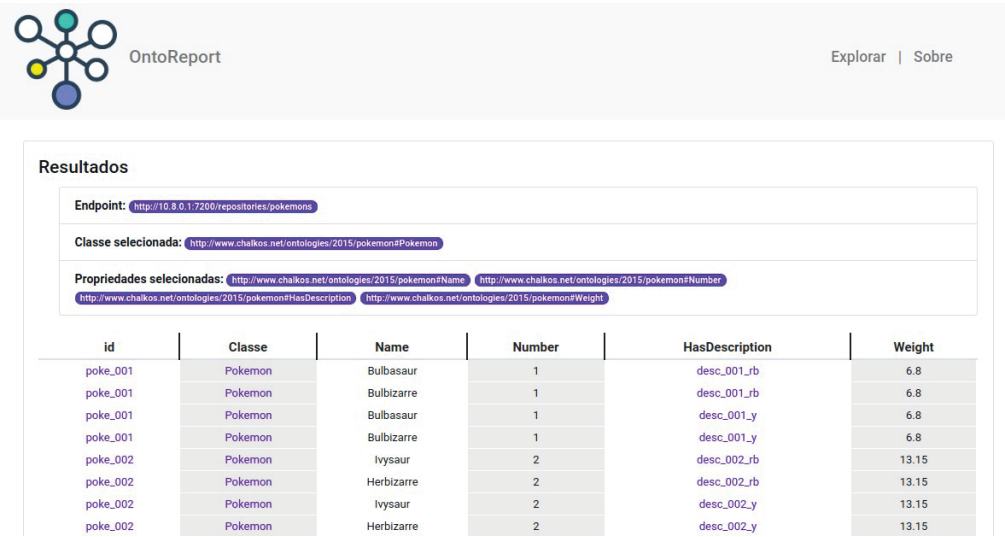


Figura 4.5.: Página seleccionar propriedades

Nesta página é permitido ao utilizador seleccionar múltiplas propriedades, sendo que não é possível avançar sem nenhuma opção seleccionada.

- Resultados.vue (Figura 4.6): depois de escolhida a classe, os indivíduos (opcional), e as propriedades, é apresentado ao utilizador uma página com o agregado de resultados.



id	Classe	Name	Number	HasDescription	Weight
poke_001	Pokemon	Bulbasaur	1	desc_001_rb	6.8
poke_001	Pokemon	Bulbizarre	1	desc_001_rb	6.8
poke_001	Pokemon	Bulbasaur	1	desc_001_y	6.8
poke_001	Pokemon	Bulbizarre	1	desc_001_y	6.8
poke_002	Pokemon	Ivysaur	2	desc_002_rb	13.15
poke_002	Pokemon	Herbizarre	2	desc_002_rb	13.15
poke_002	Pokemon	Ivysaur	2	desc_002_y	13.15
poke_002	Pokemon	Herbizarre	2	desc_002_y	13.15

Figura 4.6.: Página seleccionar propriedades

Os dados apresentados nesta última página podem ser exportados em formato *Comma-separated values (CSV)*.

Como pode ser facilmente identificado, através das figuras anteriores, cada componente desempenha um papel especifico na aplicação, comunicando cada uma individualmente com o servidor, Node.Js, e depois de receber a resposta processa-a de forma adequada.

Terminado este processo o utilizador tem também a oportunidade de explorar um recurso especifico, sendo que para isso apenas precisa de clicar, em cima desse mesmo recurso, sendo redireccionado para a sua página de propriedades.

A cada momento é também permitido ao utilizador ver o histórico de *querys* efetuadas, como demonstra a Figura 4.7.

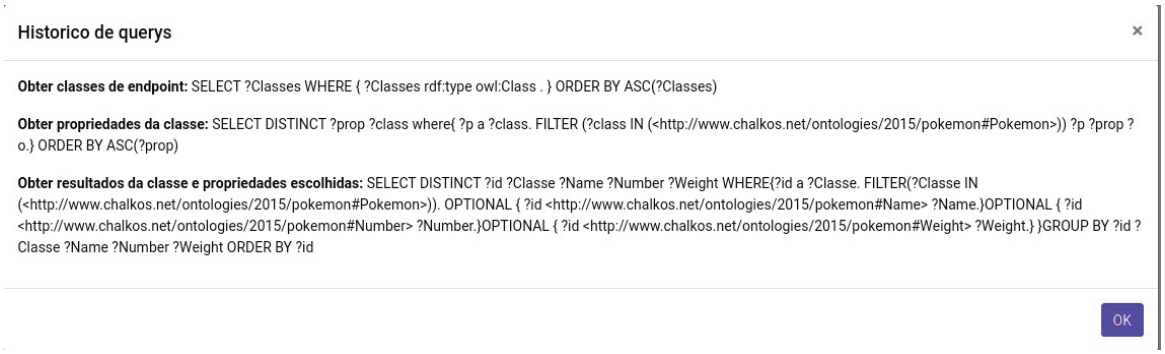


Figura 4.7.: Histórico de querys efetuadas

4.3.1 *Dependência entre componentes*

Apesar de todos os componentes criados serem diferentes, não são totalmente independentes entre si.

Na Figura 4.8, é possível observar a dependência entre os componentes.

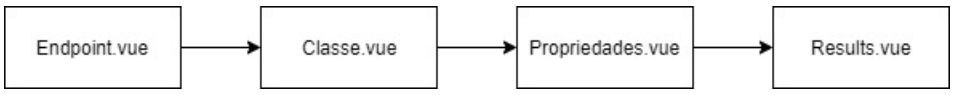


Figura 4.8.: Hierarquia de componentes

Assim, o único componente que não precisa de outro, para ser renderizado, é o "Endpoint.vue", que é onde o utilizador introduz o *endpoint* da ontologia sobre a qual quer obter informação, todos os outros dependem das escolhas anteriores do utilizador.

4.3.2 *Armazenamento de estado*

Para armazenar o estado, tendo em conta que não está a ser considerada a possibilidade de integração de vários utilizadores, foi utilizada a biblioteca Vuex[3]. O Vuex é um padrão de gestão de estado + biblioteca para aplicações Vue.js. Ele funciona como um *store* centralizado para todos os componentes de uma aplicação, com regras garantindo que o estado só possa ser mudado de forma previsível. Na Figura 4.9, é possível observar a estrutura desta ferramenta.

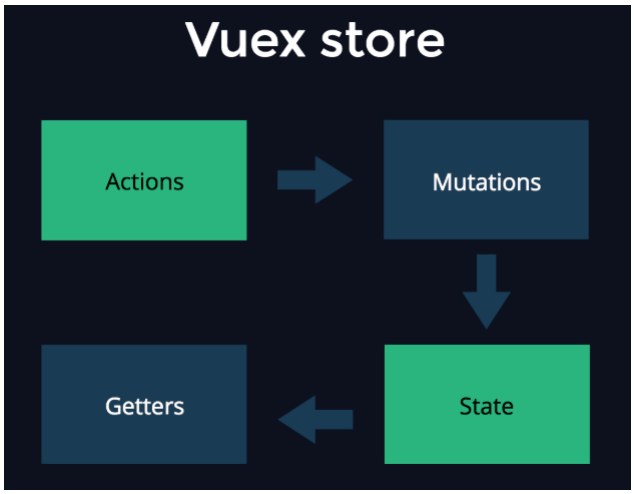


Figura 4.9.: Estrutura do Vuex

No contexto desta aplicação, esta ferramenta irá ser utilizada para guardar o estado da aplicação, consoante as diferentes escolhas do utilizador, isto é, guardando o *endpoint*, as classes, as propriedades e o histórico de *querys*.

Assim foi definido o seguinte estado da aplicação:

```
state: {
    endpoint: '',
    historico_querys: [],
    classes: [],
    classe_escolhida: '',
    individuos:[],
    individuos_escolhidos:[],
    props: [],
    props_escolhidas: [],
    recurso: '',
    results: null
  }
```

Excerto de código 4.16: Estado da aplicação

3 https://vuex.vuejs.org/

O estado, apresentado anteriormente, irá ser partilhado por todos os componentes da aplicação, que o poderão consultar ou alterar dados. Para melhor gestão do estado, da aplicação, foi criado um ficheiro, que pode ser observado no anexo B, onde estão presentes todos os métodos possíveis para mudança do estado da aplicação. Todos os métodos presentes neste ficheiro, podem ser invocados, de forma simples, pelos diversos componentes da aplicação.

Como exemplo, é apresentado de seguida duas invocações, de métodos deste ficheiro, uma com o objectivo de adicionar elementos, e outra com objetivo de os retirar.

```
    \\muda o endpoint guardado
    this.$store.commit("changeEndpoint", this.endpoint);

    \\limpa as propriedades escolhidas
    this.$store.commit("cleanProps");
```

Excerto de código 4.17: Exemplo de inovação dos métodos Vuex

4.3.3 *Comunicação com o servidor*

Como descrito anteriormente, cada componente irá depender das escolhas anteriores do utilizador, escolhas essas que irão estar armazenadas no estado da aplicação, e irão ser enviadas para o servidor. Uma vez que o Vue.js não tem integradas funcionalidades para pedidos HTTP, a comunicação com o servidor é feita utilizando uma biblioteca externa. Neste caso, foi utilizada a biblioteca Axios [4]. A escolha desta ferramenta recai sobre a familiaridade que havia com a mesma e na sua simplicidade.

Para a utilização desta biblioteca é necessário efetuar o pedido no URL correto, do servidor, e enviar as variáveis necessárias, neste caso recorrendo à *store*.

No exemplo seguinte é possível observar a utilização, desta biblioteca, no caso concreto de obter as classes de uma ontologia.

```
async obterClasses() {
      await axios
        .post("http://localhost:3000/ontologia/classes", {
          endpoint: this.$store.state.endpoint
        })
        .then(response => {
          var classes = new Array();
          classes = response.data.classes;
          var instrucao = response.data.instrucao;
          var query = response.data.query;
          ...
```

4 https://www.npmjs.com/package/axios

```
        ...
      })
      .catch(error => {
        this.loading = false;
        alert("Sem resposta");
        return error;
      });
  }
```

Excerto de código 4.18: Exemplo de pedido Axios.

4.4 SÍNTESE

Tal como enunciado, foi apresentado neste capítulo toda a parte de implementação desta aplicação. Numa primeira fase, está apresentada a arquitectura geral da aplicação, com as diferentes tecnologias escolhidas atribuídas. Do lado do servidor, toda a implementação baseia-se em Node.js, e tem como objetivo o acesso a dados e realização de *querys* sobre estes. Do lado do cliente, toda a implementação baseia-se em Vue.js, e tem como principal objetivo a criação da interface visivel ao utilizador, assim como o tratamento de dados retornados pelos servidor.

Neste capitulo estão também explicadas as diferentes abordagens e ferramentas, tanto do lado *back-end* como do lado *front-end*, que ajudaram à realização desta aplicação, e fizeram com que esta cumprisse os requisitos elaborados.

No capitulo seguinte irão ser apresentadas as conclusões desta dissertação, assim como algumas linhas de trabalho futuro.

5

CONCLUSÕES E TRABALHO FUTURO

Para terminar esta dissertação é apresentado um resumo do que foi abordado na mesma. Além disso são ainda apresentadas algumas linhas para trabalho futuro.

5.1 CONCLUSÕES

O principal objectivo desta dissertação era a construção de uma aplicação que permitisse, a um utilizador, criar uma especificação de um relatório sobre uma determinada ontologia. Por outras palavras, a construção de uma aplicação que permitisse a navegação, sobre uma ontologia, sem que para isso tivesse de efectuar *querys* sobre ela, ou que precisasse de ter algum conhecimento sobre esta.

Numa primeira fase foi elaborado o estado de arte do documento, que consistiu num estudo sobre os principais temas, relacionados com ontologias, assim como o estudo de quais as ferramentas existentes atualmente para a sua manipulação.

O passo seguinte passou pela análise da aplicação a desenvolver. Depois de analisados quais os requisitos, foi necessário definir a estrutura da aplicação. A escolha da estrutura, que se baseou num modelo de "cliente-servidor", acabou por ser relativamente fácil e rápida, tendo em conta que é das estruturas mais utilizadas para a construção deste tipo de aplicações, e também por ser aquela sobre a qual já possuía alguma familiaridade. Nisto, de seguida, foi necessário reflectir sobre a escolha das tecnologias a utilizar, tanto para o lado do cliente como para o lado do servidor. Escolhas essa que acabaram por recair sobre o Node.js, para ser usado no lado do servidor, e o Vue.js para o lado do cliente, que acabaram por se revelar boas escolhas face à rapidez de aprendizagem.

Na fase seguinte foi feita a implementação da aplicação. Esta fase foi a mais demorada, a nível de servidor, principalmente pela necessidade de construção das *querys* necessárias à obtenção de resultados, e a nível de construção da interface, sobretudo devido à necessidade de organização dos dados retornados pelo servidor. Assim, foi criado um protótipo da aplicação, OntoReport, que responde a todos os requisitos elaborados.

Concluindo, pela análise do documento, é possível observar que o objectivo principal, proposto nesta dissertação, foi cumprido. Assim, com a utilização desta aplicação, um

utilizador pode facilmente navegar sobre uma ontologia, através do seu *endpoint*, e retirar os resultados que pretende.

5.2 TRABALHO FUTURO

Apesar desta aplicação, OntoReport, responder a todos os requisitos impostos, há ainda espaço a melhorias futuras.

Assim, são enumerados de seguida alguns pontos que seriam uma mais valia:

- Permitir ao utilizador alterar as suas escolhas, sem que para isso tenha de começar o processo de novo;
- Criar login para possibilitar o armazenamento de informações. Este ponto passaria pela criação de uma base de dados, em que fosse possível armazenar todas as informações de um utilizar;
- Permitir ao utilizador observar os resultados através de um grafo, à semelhança do que acontece em outras ferramentas;
- Criação de vários formatos de exportação de dados;
- Melhorar a interface da aplicação.

BIBLIOGRAFIA

[1] E. A. M. Morais A. P. L. Ambrósio. Ontologias: conceitos, usos, tipos, metodologias, ferramentas e linguagens. Technical report, Instituto de Informática, Universidade Federal de Goiás, December 2007.

[2] Balakrishnan Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[3] Oreste Signore. Representing knowledge in the semantic web. *W3C*, 2009.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[5] Michael Uschold and Michael Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Record*, 33(4), 2004.

[6] Barry Smith, Waclaw Kusnierczyk, Daniel Schober, and Werner Ceusters. Towards a reference terminology for ontology research and development in the biomedical domain. In *KR-MED*, volume 222, 2006.

[7] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, March 2001.

[8] Rudi Studer. On-to-knowledge: Content-driven knowledge management tools through evolving ontologies. 1999.

[9] John Davies, Dieter Fensel, and Frank Harmelen. Towards the semantic web: Ontology-driven knowledge management. 03 2003.

[10] Eric Miller. An introduction to the resource description framework. *D-Lib Magazine*, 4(5), 1998.

[11] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF, 2005.

[12] protege.stanford.edu. Protégé, 2012.

[13] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. Visualizing ontologies with vowl. *Semantic Web*, 7(4):399–419, 2015.

[14] Apache Jena. semantic web framework for java, 2007. http://jena.apache.org/.

[15] Mobi W3C. Web framework for ontologies. http://mobi.inovexcorp.com/.

[16] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, 2(1):39–59, February 1984.

[17] Margaret Rouse. Corba (common object request broker architecture). https://searchsqlserver.techtarget.com/definition/CORBA, 2006.

[18] Markus Horstmann and Mary Kirtland. Dcom architecture. Dcom technical article, Microsoft Corporation, Redmond, WA, 1997.

[19] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (SOAP) 1.1. W3C Note NOTE-SOAP-20000508, World Wide Web Consortium, May 2000.

[20] Bruno Costa, Paulo F. Pires, Flávia Coimbra Delicato, and Paulo Merson. Evaluating a representational state transfer (rest) architecture: What is the impact of rest in my architecture? In *WICSA*, pages 105–114. IEEE Computer Society, 2014.

[21] Kai Lei, Yining Ma, and Zhi Tan. Performance comparison and evaluation of web development technologies in php, python, and node.js. In Xingang Liu, Didier El Baz, Ching-Hsien Hsu, Kai Kang, and Weifeng Chen, editors, *CSE*, pages 661–668. IEEE Computer Society, 2014.

[22] Luís Abreu. Node.js - construção de aplicações web, 2016.

[23] Vue.JS. The vue instance. https://vuejs.org/v2/guide/instance.html.

[24] Javascript frameworks benchmark. https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts/table.html.

[25] W3C. Html5 : a vocabulary and associated apis for html and xhtml, 2012.

[26] MDN web docs. Javascript. https://developer.mozilla.org/pt-PT/docs/Web/JavaScript.

[27] Cascading style sheets. https://www.w3.org/Style/CSS/Overview.en.html.

A

MÉTODO EXEMPLO UTILIZADO NO SERVIDOR

```
//Metodo para obter resultados finais da classe, individuos e propriedades
    escolhidas, com filtragem por individuos
exports.get_ontology_individuo_results = function (req, res, next) {
  var endpoint = req.body.endpoint
  var classe = req.body.classe
  var individuo = req.body.individuo
  var propriedades = req.body.prop
  var opcoes = ""
  var colunas = ""
  var individuos= ""

  for (i in individuo){
    individuos += "<"+individuo[i]+">,";
  }
  individuos =individuos.substring(0, individuos.length - 1)

  for (i in propriedades) {
    colunas += "?" + propriedades[i].split('#')[1] + " ",
    opcoes += "OPTIONAL { ?id <" + propriedades[i] + "> ?" + propriedades[i].
        split('#')[1] + ".}";
  }
  console.log("-------------------------------------------------------------")
  console.log("Endpoint: " + endpoint)
  console.log("Classe: "+classe)
  console.log("Individuos: "+individuos)
  console.log("Querie a efetuar: Obter resultados finais da classe, individuos
      e propriedades escolhidas")
  var query = "SELECT DISTINCT ?id ?Classe " + colunas + "WHERE{?id a ?Classe.
      FILTER(?Classe IN (<"+ classe+">)). FILTER (?id IN ("+individuos+")). "
     + opcoes + " }GROUP BY ?id ?Classe " + colunas + " ORDER BY ?id"
  console.log("---> " + query)
  if (endpoint) {
    client = new SparqlClient(endpoint, {
      defaultParameters: {
        format: 'json'
```

```
      }
    }).register({
      rdf: 'http://www.w3.org/1999/02/22-rdf-syntax-ns#',
      owl: 'http://www.w3.org/2002/07/owl#',
      rdfs: 'http://www.w3.org/2000/01/rdf-schema#'
    })
    client.query(query).execute().then((qres) => {
      console.log("Querie efetuada com sucesso")
      console.log("------------------------------------------------------------"
          )
      return res.status(200).json({
        instrucao: "Obter resultados finais da classe, individuos e
            propriedades escolhidas:",
        query: query,
        result: JSON.parse(JSON.stringify(qres))
      });
    }).catch((error) => {
      console.log("Erro no pedido.")
      console.log('ERRO: ' + error)
      console.log("------------------------------------------------------------"
          )
    })
  } else {
    console.log("Not Endpoint")
    return res.status(500).json({
      message: "Not Endpoint"
    })
  }
}
```

Excerto de código A.1: Método para resultados finais

B

MUTAÇÕES DE ESTADO VUE.JS

```
    mutations: {
        changeEndpoint (state, text) {
            state.endpoint = text
        },
        addQuery(state, text) {
            state.historico_querys.push(text)
        },
        changeClassesEscolhidas(state, classe_escolhida){
            state.classe_escolhida = classe_escolhida
        },
        addClasse(state, text) {
            state.classes.push(text)
        },
        addPropriedade(state, text) {
            state.props.push(text)
        },
        changePropriedadesEscolhidas(state,props_escolhidas){
            state.props_escolhidas= props_escolhidas
        },
        addIndividuo(state,text){
            state.individuos.push(text)
        },
        changeIndividuosEscolhidos(state,individuos_escolhidos){
            state.individuos_escolhidos = individuos_escolhidos
        },
        changeRecurso (state, text) {
            state.recurso = text
        },
        changeResults(state,results){
            state.results = results
        },
        cleanState(state){
            state.endpoint = ''
            state.historico_querys = []
            state.classes = []
```

```
            state.classe_escolhida = ''
            state.props = []
            state.props_escolhidas = []
            state.recurso = ''
            state.results= null
        },
        cleanClasses(state){
            state.classe_escolhida = ''
            state.props_escolhidas = []
            state.recurso = ''
            state.results = null
        },
        cleanProps(state){
            state.props = []
            state.props_escolhidas = []
            state.recurso = ''
            state.results = null
        },
        cleanIndividuos(state){
            state.recurso = ''
            state.individuos = []
            state.individuos_escolhidos = []
            state.results = null
        },
        cleanResults(state){
            state.results = null
        }
    }
```