
**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**

**CENTRO TECNOLÓGICO**

**COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO**

Dhiego Santos Broetto

# Uma Aplicação Web para Apresentação de Dados de Qualidade de Água do Rio Doce

Vitória, ES

2022

Dhiego Santos Broetto

# Uma Aplicação Web para Apresentação de Dados de Qualidade de Água do Rio Doce

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Ciência da Computação

Orientador: Prof. Dr. João Paulo A. Almeida

Vitória, ES

2022

Dhiego Santos Broetto
Uma Aplicação Web para Apresentação de Dados de Qualidade de Água do Rio Doce/ Dhiego Santos Broetto. – Vitória, ES, 2022-
48 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. João Paulo A. Almeida

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Colegiado do Curso de Ciência da Computação, 2022.

1. Rio Doce. 2. Front-end. 3. Aplicação Web. 4. Dados de Qualidade de Água. I. Almeida, João Paulo Andrade. II. Universidade Federal do Espírito Santo. IV. Uma Aplicação Web para Apresentação de Dados de Qualidade de Água do Rio Doce

CDU 02:141:005.7

Dhiego Santos Broetto

# Uma Aplicação Web para Apresentação de Dados de Qualidade de Água do Rio Doce

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, (dia) de (mês) de (ano):

---

**Prof. Dr. João Paulo A. Almeida**
Orientador

---

**Veruska Carretta Zamborlini**
UFES, Membro interno

---

**Victorio Albani de Carvalho**
IFES, Campus Colatina

Vitória, ES
2022

*Dedico este trabalho a todas as pessoas que fizeram parte da minha trajetória até o presente momento, que me moldaram e ajudaram a ser quem sou hoje, em especial ao meu orientador João Paulo A. Almeida, obrigado.*

# Agradecimentos

Agradeço primeiramente a Deus, por ter me dado a bênção da vida e me proporcionado viver para que eu pudesse enxergar todas as belezas e sentir todas as maravilhosas sensações que o mundo pode proporcionar.

A minha família que me deu amor e estruturas firmes para que eu sempre siga em frente, não importando o desafio e aos meus amigos que durante toda trajetória da minha vida, pessoal e acadêmica, estiveram presentes me apoiando sempre que necessário.

Ao meu orientador João Paulo A. Almeida pela sugestão do tema, pelos ensinamentos a respeito de ontologias, pela dedicação minuciosa nas diversas revisões feitas na monografia e pela paciência nos momentos de crise, sempre sendo sábio no uso das palavras.

Agradeço a todos os professores que me mostraram o prazer de estar inserido na área da informática nos meus primeiros contatos no meu curso Técnico em Informática na instituição CEET Vasco Coutinho.

A todos os professores do curso de Ciência da Computação na Universidade de Vila Velha (UVV), onde apenas três períodos foram o suficiente para eu decidir continuar no curso.

A todos os professores da Universidade Federal do Espírito Santo (UFES), do curso de Ciência da Computação, que iluminaram meu caminho, me fazendo enxergar qual área dentro da computação devo seguir em minha vida.

Muito obrigado a todos, não seria quem sou hoje sem todos vocês.

*"Ninguém vai bater tão duro como a vida, mas não se trata de bater duro. Se trata de quanto você aguenta apanhar e seguir em frente, o quanto você é capaz de aguentar e continuar tentando. É assim que se consegue vencer.*

*(Rocky Balboa)*

# Resumo

Por conta da maior tragédia ambiental do país que ocorreu na barragem de Fundão em Mariana, órgãos ambientais e organizações da sociedade civil (como universidades) intensificaram o monitoramento de vários parâmetros de qualidade de água da bacia do Rio Doce para mensurar os impactos deste incidente.

Para permitir o uso integrado dos dados coletados, pesquisadores de diversas instituições incluindo a UFES, a UFF, a UNESP e o IFES (Campus Colatina) estabeleceram um projeto de infraestrutura de dados baseado nas técnicas da Web Semântica. Seus esforços culminaram no armazenamento dos dados em uma base de triplas RDF (Stardog), baseada em uma ontologia operacional implementada em OWL.

Apesar da disponibilização dos dados em formato padrão na Web Semântica, o acesso a estes dados requer conhecimento técnico especializado. Este projeto descreve a criação de uma aplicação Web (*front-end*), na qual os dados medidos são representados em um mapa geográfico e em uma tabela correspondente. Tais representações podem sofrer mudanças perante filtros que o usuário pode aplicar para melhor visualização em tela.

Para o desenvolvimento do site, foi utilizado o framework de JavaScript Vue.js com o auxílio da biblioteca Vuetify para a montagem do layout e Stardog.js com a responsabilidade de buscar os dados diretamente da base de triplas através de consultas SPARQL e manipulá-los para serem exibidos em tela. Para a exibição do mapa, a biblioteca OpenLayers foi escolhida por se tratar de um projeto *open source* com implementação em JavaScript.

Todo o esforço foi feito para facilitar o acesso à informação no que tange à tragédia ambiental para a população em geral e para estudiosos sobre o tema, podendo até servir como uma ferramenta para futuras pesquisas sobre o tema.

**Palavras-chaves**: Rio Doce. *Front-End*. Aplicação Web. Dados de Qualidade de Água.

# Lista de ilustrações

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| UFES | Universidade Federal do Espírito Santo |
| UFF | Universidade Federal Fluminense |
| UNESP | Universidade Estadual Paulista |
| IFES | Instituto Federal do Espírito Santo |
| RDF | Resource Description Framework |
| OWL | Web Ontology Language |
| SPARQL | SPARQL Protocol and RDF Query Language |
| JSON | JavaScript Object Notation |
| SQL | Standard Query Language |
| SASS | Syntactically Awesome Style Sheets |
| CSS | Cascading Style Sheets |
| UI | User Interface |
| CSV | Comma-Separated Values |
| IRI | Internationalized Resource Identifier |
| URI | Uniform Resource Identifier |
| UCS | The Universal Coded Character Set |
| W3C | World Wide Web Consortium |
| XML | eXtensible Markup Language |
| NEMO | Núcleo de Estudos em Modelagem Conceitual e Ontologias |
| QUDT | Quantity, Unit, Dimension and Type |
| UFO | Unified Foundational Ontology |
| API | Application Programming Interface |
| HTML | HyperText Markup Language |
| SPA | Single-Page Application |

| | |
|---|---|
| OSM | OpenStreetMap |
| DOM | Document Object Model |

# Sumário

# 1 Introdução

Na tarde de 5 de novembro 2015, o Brasil parou para acompanhar e lamentar a considerada maior tragédia ambiental do país, em decorrência do rompimento da barragem de Fundão em Mariana, Minas Gerais. Tragédia essa que foi responsável por ceifar vidas humanas, destruir casas, plantações, além de despejar cerca de 60 milhões de metros cúbicos de rejeitos de minério de ferro por toda extensão da Bacia do Rio Doce (MPMG, 2020).

Para avaliar os impactos do derramamento de rejeitos de mineração no leito dos rios, vários órgãos ambientais e organizações da sociedade civil passaram a monitorar detalhadamente diversos parâmetros de qualidade de água da Bacia do Rio Doce. A grande disponibilidade de dados, porém, vem acompanhada de uma diversidade de formatos diferentes de representação destes dados. Para lidar com este problema de heterogeneidade, instituições de ensino (UFF, UNESP, IFES Campus Colatina) se uniram em um projeto coordenado pela Universidade Federal do Espírito Santo (UFES) para construir uma infraestrutura de *e-science* capaz de importar e apresentar estes dados coletados de uma forma padronizada e centralizada[1].

Os esforços das instituições de ensino culminaram em uma base de dados feita com base em padrões da Web Semântica (como RDF e OWL) e implementação na base de triplas Stardog (STARDOG, 2021), baseada em uma ontologia operacional (ALMEIDA, 2020). O uso dos padrões da Web Semântica nesta infraestrutura visa facilitar o reuso dos dados de medições de qualidade de água por outros sistemas de informação.

Apesar da infraestrutura de dados mitigar os problemas de representação e interpretação de dados, o acesso a estes dados triplificados exige conhecimento técnico especializado (por exemplo na linguagem de consultas SPARQL). Para lidar com essa barreira, propõe-se a criação de *front-ends* de apresentação dos dados disponíveis na infraestrutura, a exemplo do portal proposto em (RABBI; CARVALHO, 2019). Entretanto, com o passar do tempo o referido portal ficou obsoleto por conta das recentes atualizações na ontologia que permeia a base de dados.

O presente trabalho propõe a criação de uma nova aplicação *front-end* Web para organizar as informações centralizadas que foram recolhidas e apresentá-las aos usuários sem exigir deles conhecimento técnico de programação. A aplicação deve extrair as informações contidas na base de dados e apresentá-las de forma intuitiva e direta, com o objetivo de atingir um número maior de pessoas. Para isto, deve explorar a representação dos dados em mapas, a disponibilização dos dados em tabelas, e a possibilidade de reduzir a quantidade de dados de acordo com filtros que possam revelar o interesse do usuário em certos aspectos específicos das medições (por exemplo, parâmetro de qualidade de água, agente responsável pela medição, região geográfica, etc.)

[1] <https://nemo.inf.ufes.br/projetos/integradoce/>

## 1.1 Motivação e Justificativa

O primeiro protótipo do sistema foi elaborado em meados de 2019 (RABBI; CARVALHO, 2019), antes da ontologia operacional estar estável. Com a evolução da ontologia, o portal se tornou obsoleto. Em comparação com o portal desenvolvido anteriormente, o sistema proposto neste trabalho reconstrói o *front-end* com uma tecnologia diferente (Vue.js). Além disso, o *front-end* acessa diretamente os dados no repositório semântico (Stardog) através de um framework em JavaScript chamado Stardog.js (NPM, 2021). Desta forma, a própria base triplificada funciona como *back-end* que executa as consultas e retorna os resultados em formato JSON para apresentação no *front-end*.

## 1.2 Objetivos

Para alcançar este objetivo de substituir o protótipo anterior, alguns objetivos específicos se fazem necessários para que contribuam com o objetivo geral, como: a revisão da literatura, a qual abrange todo conceito de ontologia e seus derivados; o estudo das tecnologias que envolvem todo o desenvolvimento da aplicação Web; a realização de testes para correção de eventuais bugs que possam surgir e por fim propor melhorias para projetos futuros.

## 1.3 Abordagem

Esta subseção apresenta decisões correspondentes aos objetivos apresentados na seção anterior e contribuíram para desenvolvimento do sistema.

### 1.3.1 Estudo da Web Semântica

O estudo e aprendizado a respeito do que se trata a Web Semântica foi mediante aulas da matéria optativa: Tópicos Especiais em Engenharia de Software III (INF09373), ministrada pelo professor Vítor Estevão Silva Souza e através das vídeo aulas disponibilizadas pelo professor no YouTube em seu próprio canal. Ao mesmo tempo em que o artigo (RABBI; CARVALHO, 2019) contribuiu para revisão do conteúdo e aprofundamento da ontologia apresentada em virtude do projeto coordenado pela equipe da UFES.

Assim como no caso do conteúdo de Web Semântica, o conhecimento a respeito de RDF (Resource Description Framework) e o formato de serialização Turtle, foi adquirido por meio das aulas e vídeos, com um acréscimo de páginas na Internet (EIS, 2017). O mesmo pode ser dito acerca do conhecimento sobre consultas SPARQL, porém, a utilização e estruturação de uma consulta SPARQL se assemelha muito à uma em SQL, logo este aprendizado se tornou mais leve por se tratar de uma teoria já vista anteriormente em outros projetos e em matérias no curso.

Tais conhecimentos foram essenciais e de extrema importância para entender a estrutura e ontologia utilizada pela equipe do projeto que organizou e centralizou todos os dados na base de triplas Stardog, com a intenção de ser possível a elaboração das consultas em SPARQL referentes ao projeto.

### 1.3.2 Preparação para o desenvolvimento

O pré-requisito adotado neste projeto foi a prévia instalação do gerenciador de pacotes do Node.js, o npm (SOUZA, 2020), que tem o papel de gerenciar as dependências de um projeto ou na própria criação do mesmo com simples comandos, sem precisar que o desenvolvedor se preocupe com isto.

Com a utilização do npm, foi feita a instalação do Vue.js conforme escolha descrita na seção 1.1. Na instalação do Vue.js, a opção do SASS como pré processador CSS se fez interessante por conta do conhecimento prévio de como se utiliza. Logo após o término da instalação do Vue.js, foi realizada a instalação de bibliotecas e frameworks. Para o desenvolvimento deste site foram embutidos ao projeto o Stardog.js, Vuetify, OpenLayers e VueJsonToCsv (o uso destes será detalhado ao longo deste documento).

### 1.3.3 Definição de funcionalidades da aplicação Web

Como a aplicação Web visa substituir o anterior (RABBI; CARVALHO, 2019), suas funcionalidades foram definidas a partir do primeiro protótipo. Este permite que o usuário manipule os dados por meio de filtros para data de início da medição, data final, escolha de uma lista de agentes medidores e de uma lista dos parâmetros de qualidade de água. Além disto, a seleção de pontos a partir do mapa serve como um dos elementos de filtro espacial para restringir a busca por medições a determinada região geográfica.

### 1.3.4 Reuso de frameworks e bibliotecas

Para construir estas funcionalidades, foram reutilizadas algumas bibliotecas e frameworks para auxiliar no desenvolvimento da aplicação Web, sendo estas: o Vue.js (version 2), empregado para gerenciar e controlar toda a aplicação; o framework UI Vuetify (VUETIFY, 2022) e o pré processador CSS SASS (POPLADE, 2013), responsáveis pelo estilo da página; o framework Stardog.js (NPM, 2021), que corresponde à camada de comunicação com a base de triplas; e o OpenLayers (PAIVA, 2019) que possui diversos métodos para criar e manipular mapas. Por fim, a biblioteca VueJsonToCsv (NPM, 2019) foi empregada para transformar a tabela de dados em um arquivo CSV (Comma-Separated Values) e disponibilizar para download do arquivo.

## 1.4 Organização da Monografia

Além desta introdução, esta monografia é composta por outros quatro capítulos:

- O Capítulo 2 apresenta os aspectos relativos ao conteúdo teórico relevante para o trabalho;
- O Capítulo 3 apresenta os requisitos da aplicação Web e sua arquitetura;
- O Capítulo 4 apresenta a implementação dos componentes da interface com o usuário detalhando o uso do framework Vue.js;
- O Capítulo 5 apresenta as considerações finais do trabalho;

# 2 Referencial Teórico

Serão apresentados neste capítulo os conceitos básicos que fundamentaram o desenvolvimento do sistema Web. O capítulo está dividido nas seguintes seções. A primeira seção 2.1 aborda a Web Semântica e seu impacto no compartilhamento de dados na Internet. A seção 2.2 apresenta a ontologia Integradoce, que foi criada para disponibilização integrada dos dados com os padrões da Web Semântica.

## 2.1 Web Semântica, ontologias e dados conectados

A Web é vasta e complexa, e as informações que nela estão contidas são disponibilizadas em formatos de texto e frequentemente armazenadas em vários bancos de dados com os mais variados formatos. Isto gera uma barreira que dificulta a integração destes dados distintos.

Por conta do problema da falta de estruturação de dados e de padronização, surge a Web Semântica, proposta por Berners-Lee, Hendler e Lassila (2001). O objetivo inicial era transformar a Web (até então focada em conteúdos textuais e multimídia) em uma Web de dados interligados. A Web Semântica logo se torna um movimento colaborativo, com o objetivo de organizar a informação de maneira legível para computadores e máquinas através de padrões de formatação de dados, como o RDF (Resource Description Framework) (W3C, 2014).

Como padrão dos dados, o RDF define um modelo de dados para uma descrição semântica processável por computadores, possibilitando a codificação, o intercâmbio e o reuso de metadados estruturados e a construção de elementos de metadados (SANTOS; CARVALHO, 2007). A estrutura dos dados logo foi organizada em triplas do tipo sujeito-predicado-objeto e cada elemento da tripla é identificado por uma IRI (Internationalized Resource Identifier) (W3C, 2004), então se torna possível a integração dos dados de diversas fontes diferentes com um mesmo modelo abstrato. A Figura 1 representa como se comporta um simples grafo RDF.

Uma IRI trabalha da mesma forma que uma URI (Uniform Resource Identifier) (TECHENTER, 2019), porém com o acréscimo da expansão da quantidade de caracteres disponíveis, utilizando o padrão UCS (The Universal Coded Character Set) (TECHOPEDIA, 2015). Enquanto a URI é limitada a apenas ao padrão US-ASCII. IRIs podem servir como substituição de URIs em casos que o sistema envolvido suporte a utilização do padrão UCS. Portanto, pode-se afirmar que uma URI é uma IRI, mas o contrário não é verdadeiro, conforme Keil (2016).

A representação de dados feitas pelas IRIs foi um caminho adotado pela Web de Dados ou Web Semântica, onde estes possuem o formato padrão de RDF. Estas IRIs permitem que desenvolvedores de conteúdo e usuários identifiquem recursos em seus próprios idiomas W3C (2004), também utilizada para descrever informações estruturadas com o intuito de permitir

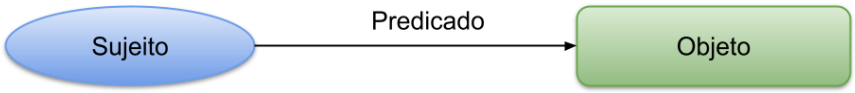


Figura 1 – Exemplo de um grafo RDF.

que aplicativos troquem dados na Web de modo que o significado original seja preservado.

Para a construção de ontologias, é fundamental uma linguagem com semântica bem definida e expressiva o suficiente para descrever inter-relacionamentos complexos e restrições entre objetos (SANTOS; CARVALHO, 2007). A partir deste cenário surge a OWL (Web Ontology Language) (W3C, 2012), que foi baseada nas especificações de RDF. Uma ontologia OWL é capaz de definir relações de taxonomia entre classes, descrições dos atributos de cada classe, propriedades dos tipos de dados apresentados e dos objetos, descrições das relações entre as classes, instâncias das classes e declarações de propriedades.

O RDF é apenas um framework, um modelo de abstração, e conta com diversas serializações (sintaxes concretas textuais). Por exemplo, há uma sintaxe padrão do W3C baseada em XML (MAGALHÃES, 2020). Alternativamente, há uma serialização mais amigável, legível e fácil conhecida como Turtle (BECKETT et al., 2014). (Esta última foi empregada neste trabalho.)

A manipulação e armazenamento dos dados representados por triplas são feitos por bancos chamados de bancos de triplas ou *triplestores*, através da linguagem de consultas chamada SPARQL que se assemelha ao SQL popularmente conhecido de bancos relacionais. O banco Stardog, empregado neste projeto, é uma *triplestore* que possui os dados coletados pelos órgãos ambientais e foram armazenados inteiramente em triplas (inicialmente em um arquivo Turtle que foi importado no Stardog.)

A seguir, ilustramos como funciona um grafo RDF com um exemplo concreto simples extraído do site DBpedia (DBPEDIA, 2022), uma aplicação Web comunitária que extrai dados estruturados de informações criadas a partir de projetos da Wikimedia. O código 2.1 apresenta um exemplo de uma query em SPARQL que faz a busca do município, estado e país onde se encontra a empresa Chocolates Garoto, utilizando o conjunto de dados providos pelo DBpedia. As palavras chaves escritas em letras maiúsculas seguem o padrão adotado pela comunidade (para deixar o código mais legível, a linguagem não é *case-sensitive*.)

As três primeiras linhas da consulta correspondem aos prefixos (*PREFIX*) que foram utilizados. Fazem o papel de servirem como *namespaces* para substituírem partes das IRIs na consulta e deixar a identificação das entidades (sujeitos, predicatos e objetos) mais simples e legível. Os prefixos utilizados neste caso denotam uma estruturação própria da DBPedia (que separa *ontology*, *property* e *resource*).

O *SELECT* desempenha o papel de apresentar quais colunas serão apresentadas no resultado, ou seja, NomeEmpresa, Municipio, Estado e NomePais. Na query em questão foi

Listagem 2.1 – Consulta SPARQL sobre a empresa Chocolates Garoto.

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

SELECT *
WHERE {
    dbr:Garoto dbp:name ?NomeEmpresa ;
               dbo:location ?Municipio .
    ?Municipio dbo:subdivision ?Estado .
    ?Estado dbp:subdivisionName ?NomePais
}
```

utilizado o asterisco (*) como parâmetro para o *SELECT* como uma forma de selecionar toda e qualquer variável utilizada na consulta.

A cláusula *WHERE* é onde são especificadas todas as condições, ou seja, apenas registros que respeitem as condições serão retornados na consulta. Podemos notar a estrutura de triplas aqui, sendo sujeito-predicado-objeto, conforme foi apresentado na Figura 1.

Um padrão que ajuda na construção da consulta é o fato de que não ser necessário reescrever o sujeito caso o mesmo seja alvo de outras condições, basta fazer uso do **';'** para que o SPARQL entenda que o sujeito da próxima condição seja o mesmo da anterior. O **'.'** é utilizado quando o sujeito passa a ser outro.

As variáveis são destacadas na consulta através do caractere inicial **'?'**, e permitem o casamento de padrões com os outros elementos da consulta. No exemplo, temos o trecho **dbr:Garoto dbp:name ?NomeEmpresa**, que selecionará todas as triplas em que haja um sujeito (**dbr:Garoto**) e o predicado (**dbp:name**), de forma que a variável assumirá o valor do objeto em cada tripla selecionada.

O resultado desta consulta é apresentado na Tabela 1. Como se trata de uma empresa e em uma localidade apenas, somente uma linha é apresentada. A Tabela 2 apresenta todas as triplas correspondentes à consulta.

## 2.2 Integradoce

O conceito de Web Semântica ou Web de Dados é amplo, porém este trabalho foi baseado exclusivamente na ontologia Integradoce (ALMEIDA, 2020), criada a partir da necessidade de unificar e padronizar os dados heterogêneos colhidos das medições feitas pelas equipes ambientais.

| NomeEmpresa | Municipio | Estado | NomePais |
|---|---|---|---|
| "Chocolates Garoto S.A."@en | http://dbpedia.org/resource/Vila_Velha | http://dbpedia.org/resource/Espírito_Santo | "Brazil"@en |

Tabela 1 – Resultado da Query 2.1

| Sujeito | Predicado | Objeto |
|---|---|---|
| dbr:Garoto | dbp:name | "Chocolates Garoto S.A."@en |
| dbr:Garoto | dbo:location | http://dbpedia.org/resource/Vila_Velha |
| http://dbpedia.org/resource/Vila_Velha | dbo:subdivision | http://dbpedia.org/resource/Espírito_Santo |
| http://dbpedia.org/resource/Espírito_Santo | dbp:subdivisionName | "Brazil"@en |

Tabela 2 – Representação das triplas da Tabela 1

Para a criação de uma ontologia que una todos os dados referentes às medições feitas por toda extensão da Bacia do Rio Doce, existem certas barreiras que precisam ser rompidas, como a heterogeneidade sintática, já que os dados são dispostos em diferentes formatos e a heterogeneidade semântica, tendo em vista que cada órgão aplica seu próprio vocabulário para descrever os dados referentes às medições feitas pelas equipes técnicas. A plataforma de *e-science* utilizada como fonte de dados para o presente trabalho foi criada com a intenção de unificar e centralizar os dados providenciados pelas equipes[2].

O primeiro passo para o projeto foi a escolha de focar os esforços a resolver o problema da heterogeneidade semântica, utilizando técnicas de engenharia de ontologias para agrupar estes dados de diferentes fontes. Logo foi necessária a criação de uma ontologia de referência (CAMPOS, 2019) sobre os dados ambientais especializada em uma sub ontologia sobre qualidade de água. A ontologia de referência foi então implementada em uma ontologia computacional codificada em OWL (ALMEIDA, 2020).

A Integradoce utiliza como base a ontologia chamada gUFO (ALMEIDA et al., 2019), criada pelo esforço coletivo da equipe do laboratório NEMO (Núcleo de Estudos em Modelagem Conceitual e Ontologias), na UFES, com objetivo de ser uma versão mais leve da ontologia UFO (GUIZZARDI et al., 2015) para aplicações em OWL.

A Figura 2 mostra as principais classes utilizadas no trabalho. As classes se relacionam através de *object properties* correspondendo às associações no diagrama. Além disso, há relações de especialização no diagrama, como é o caso de **Measurement** com **gufo:Event**: **Measurement** herda as *data properties* **hasBeginPointInXSDDateTimeStamp** e **hasEndPointInXSDDateTimeStamp**.

Também é possível enxergar a utilização de classes pertencentes à ontologia gUFO, todas que possuem o prefixo '**gufo:**' no diagrama de classes. Importante destacar que todas as classes utilizadas no trabalho são especializações das classes de gUFO e todas são sub-classes de **owl:Thing**, classe de OWL.

Instâncias da classe **Measurement** representam as medições na base, possuindo relações com **GeographicPoint** e **QualityKind** pelas respectivas *properties* **locatedIn** e **measuredQualityKind**.

**GeographicPoint** por sua vez representa os pontos geográficos referentes às medi-

[2] <https://nemo.inf.ufes.br/projetos/integradoce/>

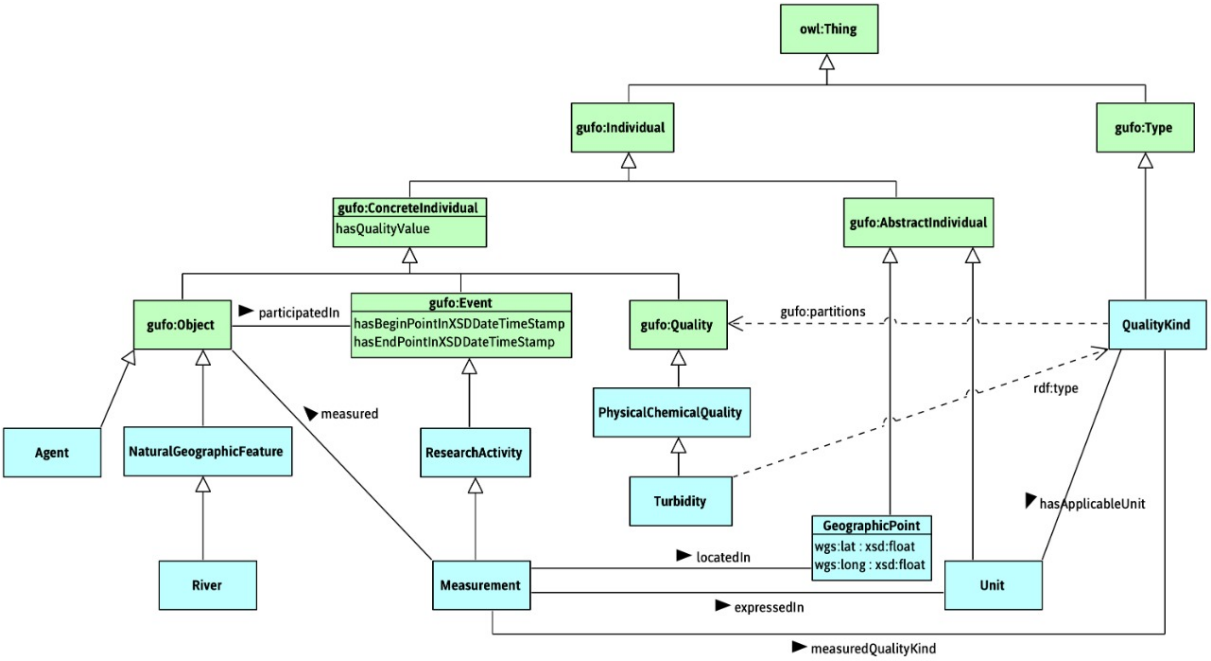


Figura 2 – Diagrama de classes da UML representando as principais classes da Integradoce.[3]

ções realizadas. Esta possui duas *properties* que representam a latitude e longitude (**wgs:lat** e **wgs:long**). Além de estar relacionada a ponto geográfico, uma medição também se relaciona a uma instância de **gufo:Object** através de **measured** para estabelecer a entidade que foi medida. Neste trabalho, a única instância medida é o próprio Rio Doce (uma instância de **River**).

Instâncias da classe **QualityKind** representam os tipos de parâmetros de qualidade de água (subclasses de **Quality**) medidos. Instâncias da classe **Unit** representam as unidades de medida (como **DEG_C** para representar graus Celsius), e são previamente definidas na ontologia. Algumas das unidades são oriundas da ontologia QUDT[4] e recebem o prefixo 'unit'. A *property* **hasApplicableUnit** relaciona uma *QualityKind* à unidade de medida aplicável para aquele tipo de parâmetro de qualidade.

No âmbito da qualidade de água, a classe **gufo:Quality** possui diversas sub-classes relativas aos parâmetros considerados relevantes para o monitoramento da Bacia do Rio Doce (também instâncias de **QualityKind**), porém para melhor visualização, apenas as classes **PhysicalChemicalQuality** e **Turbidity** foram representadas.

Por fim, outra classe que foi utilizada para este projeto foi a **Agent** que, representa os autores responsáveis pelas medições. Neste trabalho foi empregada a *property* **participatedIn**, que liga um **gufo:Object** a um **gufo:Event**. Como **Agent** e **Measurement** são sub-classes de **gufo:Object** e **gufo:Event** respectivamente, **participatedIn** pode ser aplicada a agentes e medições.

A consulta exemplificada na Listagem 2.2 utiliza como base a *triplestore* do projeto, onde

[3] Diagrama fornecido pelo Prof. João Paulo A. Almeida, em março de 2022.

[4] <http://qudt.org>

os dados das medições (*measurements*) estão armazenados. O retorno desta consulta são três medições contendo o parâmetro de qualidade de água, o valor medido, a unidade de medida e o ponto geográfico no mapa de onde foi feita a medição, campos representados no espaço do *SELECT*, conforme apresentado na Tabela 3. O comando *AS* serve apenas como uma forma de nomear a coluna de retorno da consulta, como também é feito em SQL.

Listagem 2.2 – Exemplo de SPARQL que retorna três medições da base Stardog.

```
SELECT (?measurement AS ?Medição)
       (?qualityKind AS ?Característica)
       (?qualityValue AS ?Valor)
       (?unit AS ?Unidade)
       (?geoPoint AS ?Ponto_Geográfico)
WHERE {
        ?measurement a :Measurement ;
                     :measuredQualityKind ?qualityKind ;
                     gufo:hasQualityValue ?qualityValue ;
                     :expressedIn ?unit ;
                     :locatedIn ?geoPoint .
}
LIMIT 3
```

| Medição | Característica | Valor | Unidade | Ponto_Geográfico |
|---|---|---|---|---|
| :AmbientTemperatureMeasurement247659516067172 | :AmbientTemperature | 26.15 | unit:DEG_C | :LAL-01 |
| :AmbientTemperatureMeasurement247659516067249 | :AmbientTemperature | 27.18 | unit:DEG_C | :LAL-01 |
| :AmbientTemperatureMeasurement247659516067326 | :AmbientTemperature | 27.3 | unit:DEG_C | :LAL-01 |

Tabela 3 – Resultado da Query 2.2

No exemplo é possível observar que a estrutura é semelhante à Query 2.1 com algumas poucas alterações. A ausência dos diversos comandos *PREFIX* se dá por estes já estarem cadastrados diretamente na base, isso evita que as queries sejam excessivamente verbosas por conta dos inúmeros prefixos e *namespaces* que possam existir numa só query. Vale salientar a utilização destes prefixos através dos predicados que possuem '**:**' e '**gufo:**' que substituem **http://purl.org/nemo/doce#** e **http://purl.org/nemo/gufo#** respectivamente.

Um detalhe interessante foi uma outra forma de utilização do predicado **rdf:type**, no lugar pode ser usado apenas **'a'**, como visto no primeiro casamento de padrões **?measurement a :Measurement**. O SPARQL permite a utilização de ambos os predicados para o mesmo fim, não existem diferenças entre eles, apenas no intuito de abreviar a query.

O comando *LIMIT* foi o responsável pela consulta ter retornado apenas três elementos, pois como é uma query de exemplo e não possui muitas restrições e casamento de padrões, o resultado traria consigo todas as milhares de medições contidas no banco.

### 2.2.1 Front-end Web com Vue.js

Com o constante crescimento, popularização e consumo da internet, as *webpages* deixaram de ser apenas uma forma de exibir seus dados. Assim então, surgiu a necessidade de

melhorar a experiência do usuário por meio de interfaces mais amigáveis e intuitivas. Dessa forma, nasce o CSS (Cascading Style Sheets) que modifica os elementos em HTML (HyperText Markup Language) com o objetivo de deixar as páginas esteticamente diferentes umas das outras. O JavaScript também foi incorporado às páginas por conta do interesse de aplicar programação nas mesmas. A linguagem funciona como uma espécie de script da Web, que é mundialmente utilizado para fins de processamento de dados para a Web.

Como tudo na computação evolui demasiadamente rápido, logo desenvolver um sistema de informação complexo se tornou uma tarefa árdua e ultrapassada, utilizando apenas HTML, CSS e JavaScript. Por conta disto, os ditos frameworks passaram a ser uma nova forma de programar sites mais dinâmicos com poucos recursos.

Como na programação, tudo pode ser reaproveitado para otimização do tempo, frameworks foram a saída para a evolução da programação Web. Dentre eles, o Vue.js (VUE.JS, 2022b) se apresenta como sendo exclusivo para *front-end*. Nascido em 2014 e criado por Evan You, se trata de um framework progressivo para construção de interfaces, sendo este focado exclusivamente na camada de visual (*visual layer*). Além disso, o Vue.js é capaz de criar aprimoradas *Single-Page Applications* (SPAs), quando utilizadas em conjunto com outros frameworks e bibliotecas de apoio, o desenvolvimento destas se torna muito mais vantajoso e descomplicado. Este framework será empregado nos capítulos subsequentes para desenvolver a aplicação de apresentação de dados de qualidade de água.

# 3 Aplicação Web para Apresentação de Dados

Neste capítulo está descrito o funcionamento do sistema de apresentação de dados como um todo e suas devidas funcionalidades.

## 3.1 Visão geral

A Figura 3 apresenta a arquitetura utilizada no projeto, incluindo o papel das ontologias, das fontes de dados, do repositório semântico e do *front-end*. As fontes de dados são o conjunto das medições feitas pelas equipes que, através das ontologias gUFO e Integradoce, são levados a um único repositório semântico (a *triplestore*), neste caso o Stardog. O Stardog então permite o acesso aos dados através das consultas SPARQL, estas que podem ser utilizadas por uma API de *webservices* ou acessadas diretamente por uma aplicação Web de página única (*front-end* Vue.js), que é o foco do deste trabalho atual.

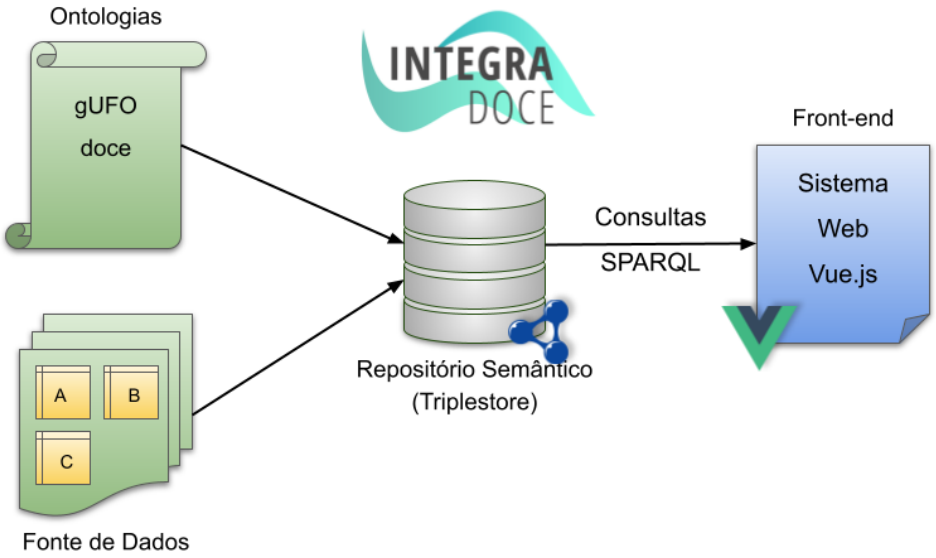


Figura 3 – Arquitetura do projeto Integradoce.

Entre os principais frameworks *front-end* do mercado, o Vue.js é o mais recomendado para iniciantes por ser leve e simples para iniciar um novo projeto (FRIAS, 2020).

O Vue.js permite a composição de componentes, que nada mais é do que a conexão entre diferentes componentes (arquivos .vue), por ser uma abstração que proporciona a cons-

trução de aplicações de larga escala compostas por pequenos componentes, auto-contidos e frequentemente reutilizáveis (VUE.JS, 2022a).

Nessa abordagem, uma *webpage* é composta por diferentes componentes, conforme pode ser observado na Figura 4, na qual os elementos mais escuros da página simbolizam diferentes componentes da página como um todo. A árvore representada à direita é uma forma gráfica de representar a conexão entre diferentes componentes, pois componentes pai possuem comunicação com seus componentes filhos e vice versa.

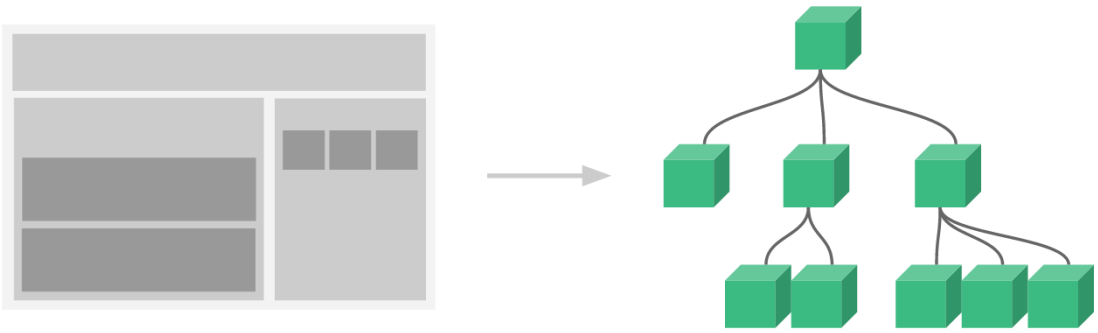

Figura 4 – Composição de componentes.[5]

Já na construção de uma *webpage front-end*, o esqueleto e estilo da página são os pilares mais importantes, sempre focando na UI (User Interface) (PATEL, 2019). UI se remete a usabilidade do site para o usuário final, sua interface precisa ser amigável, intuitiva e padronizada. Tendo em vista este fato, o framework UI Vuetify (VUETIFY, 2022) foi adicionado ao projeto. Este disponibiliza um vasto arsenal de componentes previamente prontos e estilizados, visando padronizar o projeto no quesito de cores, botões e tamanhos de fonte. Para melhorar o desempenho do site no âmbito do estilo das páginas, a presença do pré processador de CSS chamado SASS (POPLADE, 2013) realiza este feito, já que o mesmo apenas compila e carrega os arquivos CSS quando um componente específico é utilizado em tela, diminuindo assim o tempo de resposta com o servidor.

## 3.2 Requisitos do sistema

O sistema possui alguns requisitos a serem alcançados e que nortearam o desenvolvimento. Todos foram levados em consideração para a entrega do protótipo final.

[5] Fonte: Vue.js (2022a), em março de 2022.

- O sistema deve permitir que haja uma conexão com a base de triplas Stardog para obtenção dos dados;
- O sistema deve permitir a visualização do mapa com todos os pontos geográficos presentes na *triplestore* ao abrir a *homepage* pela primeira vez;
- O sistema deve permitir que os dados de medições sejam filtrados conforme campos de data inicial e data final, responsáveis das medições, parâmetros de qualidade de água e um ponto específico do mapa;
- O sistema deve permitir que o usuário possa usufruir de funcionalidades do mapa como *zoom in*, *zoom out*, *drag-and-drop* e maximizar o mapa;
- O sistema deve permitir que o usuário possa selecionar um ponto geográfico presente no mapa e este deve ser integrado aos campos de filtros e considerado como restrição ao realizar uma busca por medições;
- O sistema deve permitir que uma tabela exiba as medições resultantes da busca por filtros: Apresentando os campos de qualidade de água, valor e unidade de medida, data e hora da medição, ponto geográfico no qual esta medição pertence, latitude e longitude do ponto e o responsável pela medição;
- O sistema deve permitir que o usuário possa baixar uma versão da tabela no formato CSV, que além dos campos presentes na tabela, também possui a IRI identificadora de cada medição;
- O sistema deve permitir que o usuário escolha se pretende atualizar ou não a visualização dos pontos no mapa ao filtrar os dados;
- O sistema deve permitir que o usuário possa recarregar o mapa para o estado inicial dele através de um botão para reiniciar o mapa;
- O sistema deve permitir que o usuário possa limpar todos os campos do componente filtro através de um botão.

## 3.3 Arquitetura e Componentes

### 3.3.1 Acesso à base de triplas

Para extrair os dados da *triplestore* e apresentar em tela é necessária a configuração para acesso à base de dados. Entretanto, o sistema atual não possui uma *webservice* ainda que realize este serviço e disponibilize em um serviço de APIs. Logo é indispensável a utilização de uma biblioteca que permite criar essa ponte entre a base de dados e o site, esta biblioteca é chamada de Stardog.js (NPM, 2021).

A biblioteca Stardog.js apresenta um conjunto de métodos para facilitar na criação de queries SPARQL com a utilização de métodos JavaScript que se comunicam diretamente com o banco *triplestore*. O método permite que o resultado SPARQL seja convertido em JSON para facilitar o uso no *front-end*, já que é muito mais acessível a manipulação de JSON com JavaScript.

Listagem 3.1 – Método JavaScript aplicado à consulta na Listagem 2.1.

```
const { Connection, query } = require('stardog');

const conn = new Connection({
  username: 'user',
  password: 'password',
  endpoint: 'stardog_server_url',
});

async function getGarotoLocation() {
    return await query.execute(conn, 'database',
        `PREFIX dbo: <http://dbpedia.org/ontology/>
        PREFIX dbp: <http://dbpedia.org/property/>
        PREFIX dbr: <http://dbpedia.org/resource/>

        SELECT *
        WHERE {
            dbr:Garoto dbp:name ?NomeEmpresa ;
                       dbo:location ?Municipio .
            ?Municipio dbo:subdivision ?Estado .
            ?Estado dbp:subdivisionName ?NomePais
        }`, 'application/sparql-results+json');
}

export {
    getGarotoLocation
}
```

A Listagem 3.1 apresenta a criação de um método para realizar a conexão com a base de triplas Stardog por meio da biblioteca Stardog.js. Desta forma é possível executar a query SPARQL exibida na Listagem 2.1. A primeira linha é a declaração dos métodos/objetos providos pelo Stardog.js para serem aplicados ao código, seriam estes o *Connection* e *query*. Uma instância de *Connection* representa a conexão do sistema com o banco, com dados para *username*, *password* e *endpoint*.

Importante ressaltar que existe uma forma de gerenciamento de *roles* (papéis) e usuários no Stardog, com isto é possível a criação de um usuário vinculado a um *role* de *read-only*. O interesse disto parte do propósito que a *triplestore* deste trabalho tem como objetivo de permitir apenas a leitura de seus dados, ou seja, qualquer acesso a estes dados não ocorreriam de serem alterados, garantindo assim a integridade dos dados. Mais detalhes a respeito do gerenciamento de usuários e seus *roles* estão disponíveis no Stardog [6].

Já o objeto *query* possui um método chamado *execute*, que recebe como parâmetros o objeto da conexão (*conn*), o nome da base de triplas, a *string* com a query, vista na Listagem 2.1

[6] <https://docs.stardog.com/archive/7.5.1/operating-stardog/security/managing-users-and-roles.html>

e por fim, o tipo de retorno (resultados SPARQL serializados em JSON).

### 3.3.2 Componentes da interface com o usuário

A *homepage* (**Home.vue**) possui três componentes diferentes integrados a esta, conforme Figura 5, são eles o componente do mapa (**MapContainer.vue**) apresentado na Figura 6, o componente dos filtros (**Filters.vue**) como mostrado na Figura 8 e o componente da tabela de dados (**MeasurementTable.vue**) como visto na Figura 9. Esta *homepage* faz a ponte entre os três outros componentes, atualizando-os sempre que uma nova informação for alterada através de algum dos três componentes.

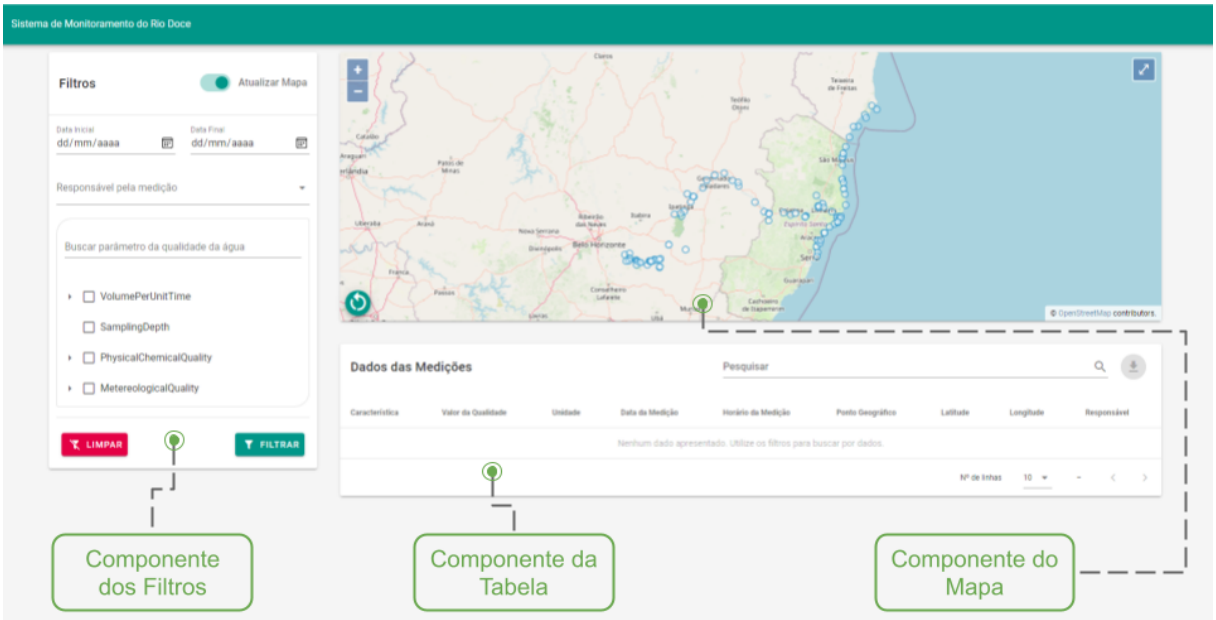


Figura 5 – *Homepage* no primeiro acesso.

Sobre a visualização do mapa geográfico, a biblioteca *open source* utilizada foi a OpenLayers (PAIVA, 2019), que provê uma API para auxiliar na composição e distribuição dos pontos sob o mapa para melhor visualização geográfica. Os *map tiles* (visualização gráfica de um mapa) são providos de outras fontes *open source*, como por exemplo o OpenStreetMap (OSM) (WROCLAWSKI, 2014), que foi o escolhido para este protótipo.

Dentre as funcionalidades do mapa como *zoom in*, *zoom out*, *drag-and-drop* e maximizar o mapa, existem aquelas que também são desenvolvidas a partir do uso da API provida pela biblioteca, como a possibilidade de destacar pontos no mapa dado um conjunto de coordenadas, conforme pode ser visualizado na Figura 6. O botão para reiniciar o mapa é outra funcionalidade que atua no mapa, porém esta é feita totalmente em JavaScript, logo não pertence ao conjunto de funcionalidades da biblioteca em questão. Tal funcionalidade será abordada na seção 4.4.

A seleção de pontos geográficos referentes às medições é feita quando o usuário clica sob algum dos pontos, pequenos círculos destacados em azul sob o mapa, como é possível ver na Figura 7 à esquerda. A imagem à direita apresenta o *card* representante do ponto selecionado

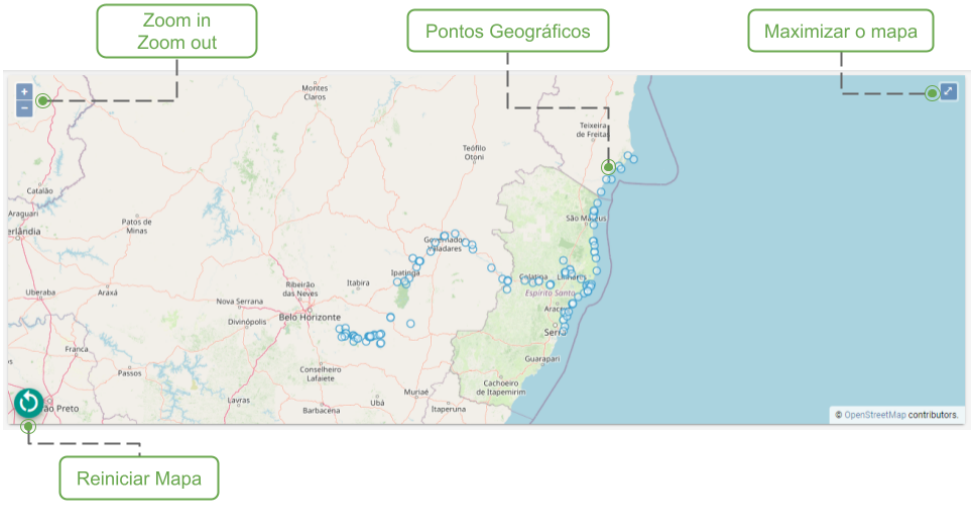


Figura 6 – Mapa inicial com todos os pontos.

no componente dos filtros, agora sendo considerado como um dos campos para a filtragem dos dados.


Figura 7 – Ponto selecionado no mapa à esquerda e apresentado no filtro à direita.

O componente dos filtros, por sua vez, tem o papel de selecionar quais parâmetros serão adotados no momento em que a busca por medições é feita, conforme Figura 8. Os campos que estão presentes no componente são o intervalo entre a data início e final de uma medição, uma listagem de quem foi o agente responsável pela medição, qual parâmetro da qualidade de água desta medição e um ponto específico, este selecionado através do mapa, conforme Figura 7.

Esta versão do protótipo, no momento, possui a possibilidade de seleção de apenas um ponto geográfico para ser utilizado como um dos campos do filtro. Por motivos de tempo para aprendizagem e desenvolvimento com a biblioteca OpenLayers, não foi possível a implementação da funcionalidade de se haver mais de um ponto selecionável ao mesmo tempo.

As listagens de responsáveis e de parâmetros da qualidade de água são buscadas diretamente da base de triplas no momento em que a página é carregada para então serem apresentadas para o usuário para então selecionar quais elementos de cada listagem serão

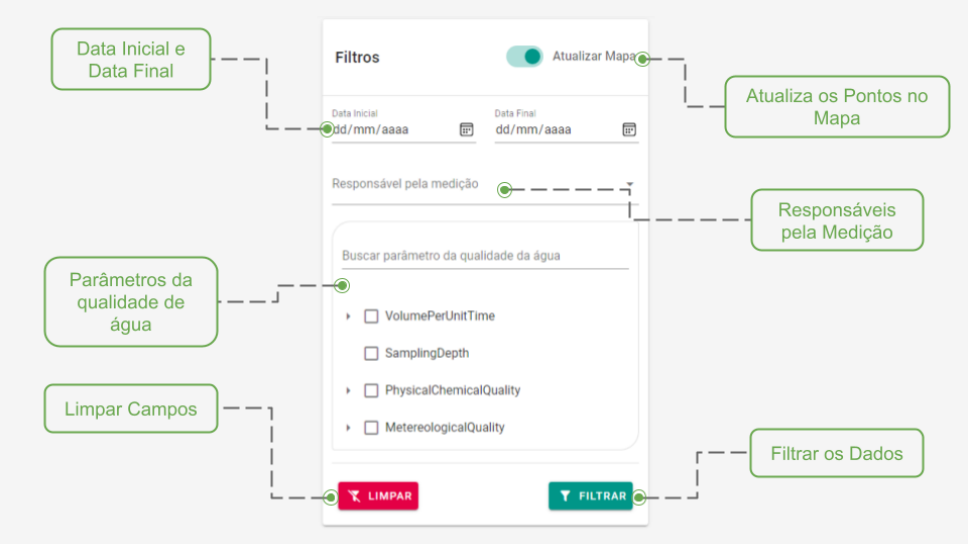


Figura 8 – Componente dos filtros.

incorporados ao filtro no momento da execução da query. No campo da listagem dos parâmetros da qualidade de água existe um campo de busca por nome, no qual é possível o usuário fazer uma busca entre seus elementos para melhor encontrar o campo desejado.

Na base do *card* se encontram os botões para filtrar os dados, ou seja, realizar a busca na base, utilizando as restrições dos campos que foram preenchidos e o botão de limpar todos os campos, recurso útil para caso o usuário pretenda realizar novas buscas com outros parâmetros.

Além disto, o componente possui um interruptor (*switch button*) chamado **Atualizar Mapa**, que opera para que o filtro realizado pelo usuário influencie nos pontos distribuídos no mapa. Por padrão este interruptor permanece ligado, assim, qualquer filtro realizado no sistema, o mapa também sofrerá alterações, podendo diminuir ou aumentar a quantidade de pontos apresentados, dependendo se estes possuem alguma relação a qualquer medição. Caso esteja desabilitado, a visualização do mapa não é alterada.

Todos os campos são opcionais, logo existe a possibilidade de que uma busca possa trazer mais dados do que a aplicação suporta. Para contornar este problema, existe um limite estabelecido de 200000 (duzentas mil) medições, para que não sobrecarregue o servidor Stardog e ocasione a suspensão da consulta, o que levaria a um retorno sem resultados.

Para os dados da tabela, é possível observar na Figura 9 como os dados das medições são dispostos através de nove colunas, Parâmetro da Qualidade de Água, Valor da Quantidade, Unidade, Data da Medição, Horário da Medição, Ponto Geográfico, Latitude, Longitude e Responsável. Todas essas colunas são ordenáveis ao clicar sob seus cabeçalhos, onde estão descritos seus nomes.

Além da tabela, é possível que o usuário possa realizar uma pesquisa sob os dados ali representados através do campo Pesquisar localizado no espaço superior da tabela. Qualquer dado de qualquer coluna presente na tabela pode servir como objeto de busca deste campo.


Figura 9 – Tabela com dados das medições.

Ao lado do campo de Pesquisar, está o botão que faz o papel de baixar um arquivo CSV com todas as informações presentes na tabela ao ser pressionado, sendo que estas informações podem ser afetadas pela pesquisa. A biblioteca VueJsonToCsv (NPM, 2019) que tem como principal funcionalidade converter todos os dados da tabela em um arquivo CSV, além do que é possível escolher quais colunas estarão presentes no arquivo e que não estão na tabela. Neste caso é importante informar que a coluna de identificador foi omitida da tabela para melhor visualização, porém inclusa no arquivo CSV.

A coluna referente a Pontos Geográficos é composta por links com os nomes dos pontos geográficos de cada medição. Ao clicar sob algum destes pontos, o mapa selecionará o ponto referente a ele no mapa, graficamente da mesma forma que foi demonstrada na Figura 7, ou seja, o ponto é selecionado tanto no mapa quanto no componente de filtros.

Por fim, a tabela possui os recursos padrões de ser possível alterar a quantidade de registros apresentados em uma única página na tabela, sendo as possíveis opções de 5, 10, 15 e todos os registros. Juntamente com a opção de paginação, onde o usuário pode navegar entre as possíveis páginas de registros da tabela.

No total foram utilizados quatro bibliotecas e frameworks para o auxílio do desenvolvimento da aplicação Web, a saber, Vuetify, Stardog.js, OpenLayers e VueJsonToCsv, representados na Figura 10.

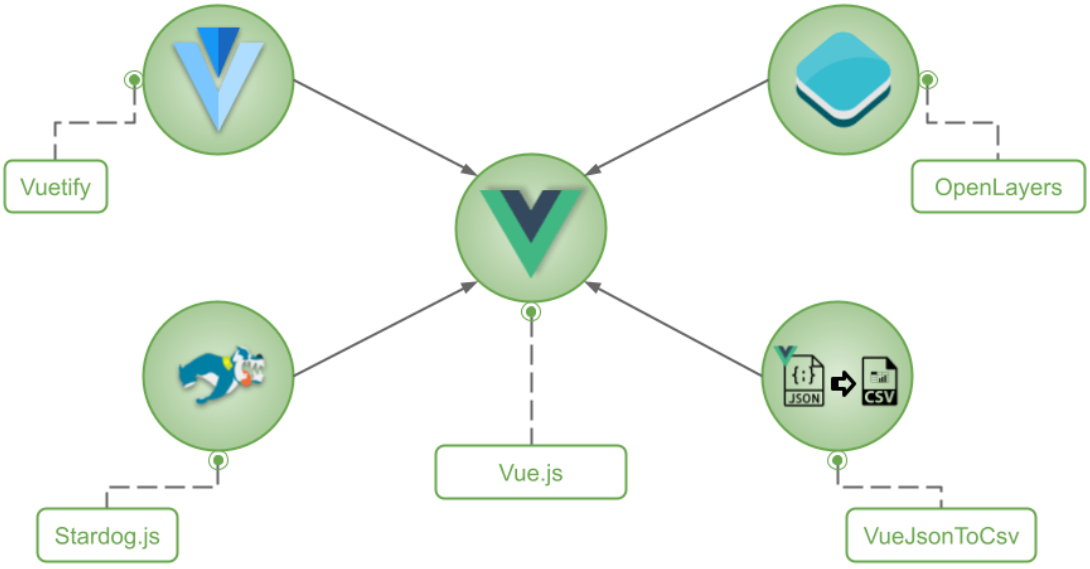


Figura 10 – Bibliotecas e Frameworks utilizados no projeto

# 4 Implementação dos Componentes da Interface

Este capítulo apresenta a implementação dos componentes da interface da aplicação Web, documentando os principais trechos de código destes componentes que foram introduzidos no capítulo anterior.

## 4.1 Visão geral

A estrutura de todo arquivo com código Vue.js é dividida em três principais tags, **<template>** a qual o framework converte em tags HTML no resultado final, **<script>** onde toda lógica programada em JavaScript é descrita para ser vinculada ao sistema no momento em que o código é compilado e a tag **<style>** que comporta o CSS da página/arquivo, sendo opcional este CSS ser *scoped* ou não, ou seja, código CSS pertencer unicamente a este arquivo. Também é possível escolher a linguagem para o **<style>**, que seria algum pré processador. No caso para este trabalho foi feita a escolha do SASS (POPLADE, 2013) (arquivos .scss).

Dentro da tag **<script>**, existem algumas palavras chave denominadas de *options*, que possuem diversas aplicações dentro da instância do Vue. Dentre elas, pode-se destacar os *lifecycle hooks*[7] (pontos de extensão ao longo do ciclo de vida da aplicação) que são executados em determinados momentos, conforme ilustrado na Figura 11; e vários atributos na chamada Options API, por exemplo, *data*[8] ou *assets*[9] que são processados apenas quando a instância do Vue está iniciando sua compilação. Os mecanismos específicos utilizados neste trabalho serão apresentados a seguir, assim que alguma funcionalidade tenha exigido seu uso.

Nas próximas seções serão apresentados trechos de código das funcionalidades mais importantes do sistema, de acordo com os requisitos listados anteriormente na seção 3.2.

## 4.2 Componente relativo ao mapa

O código descrito para a criação do mapa, conforme pode ser observado na Listagem 4.1, faz uso de algumas das *options* já citadas na seção anterior. O *hook mounted* presente aqui é um dos que compõem os *lifecycle hooks*, conforme pode ser visto na Figura 11. A implementação deste componente também usa os atributos *props*, *data* e *methods*.

Os atributos em *props* representam dados transmitidos através de um componente pai

[7] <https://br.vuejs.org/v2/api/index.html#Opcoes-Ciclo-de-Vida>
[8] <https://br.vuejs.org/v2/api/index.html#Opcoes-Dados>
[9] <https://br.vuejs.org/v2/api/index.html#Opcoes-Recursos>

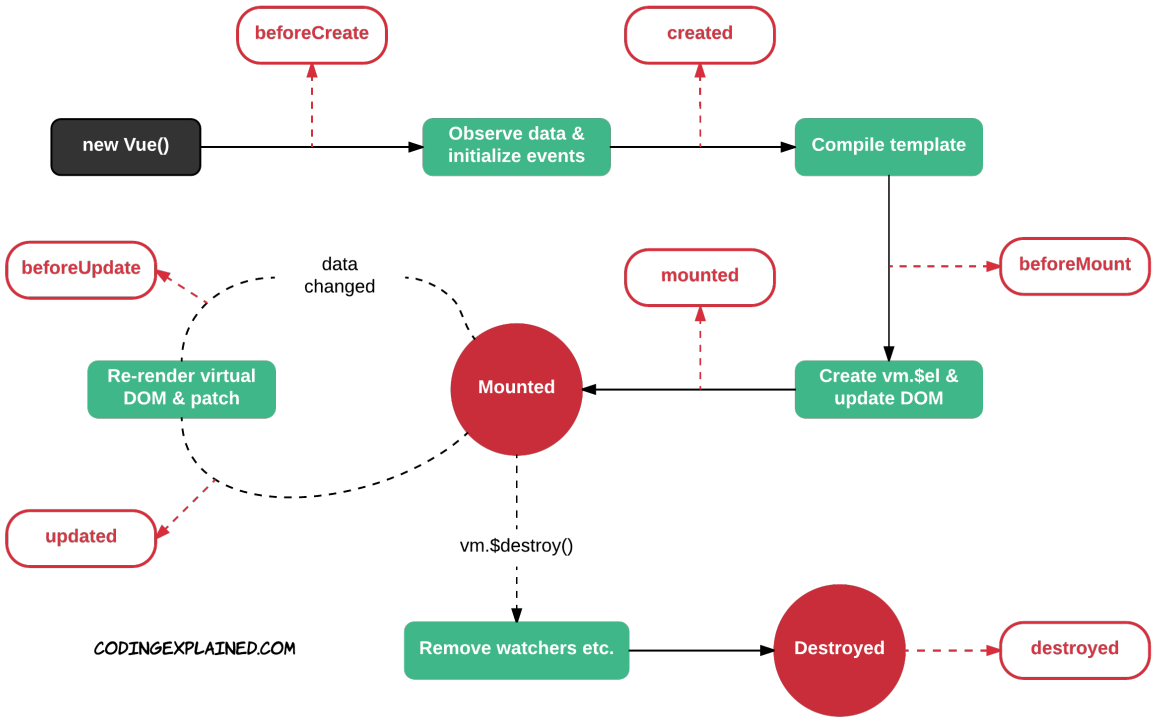


Figura 11 – *Lifecycle hooks* do Vue.js[10]

para o filho, desta forma um componente filho consegue ter acesso a valores antes limitados ao pai. No trabalho apenas o objeto **geojson** foi utilizado para a criação do mapa, este possui consigo todas as coordenadas que foram previamente carregadas no componente pai (**Home.vue**) através do método **getPoints**, na linha 48, que retorna todos os **GeographicPoints** relativos às medições via SPARQL, vide Listagem 4.2.

Os atributos em *data* podem ser utilizados por todo código deste componente, seja na tag **<template>** como em **<script>**. Aqui foram definidos os atributos **map**, **vectorLayer** e **view**, sendo todos estes atributos relativos ao mapa.

O *hook mounted* é executado logo após a instância Vue ter sido montada, quando **vm.$el** é criado, este é um elemento DOM raiz que a instância do Vue gerencia. É importante para quando há necessidade de processamentos mais pesados ou chamadas API, para não influenciar no carregamento da página como um todo, podendo ser exibido uma tela de carregamento no momento em que este processamento está sendo feito. Logo, a construção do mapa foi feita aqui, através da criação dos objetos de **VectorLayer** e **Map** e a chamada para o método que atualiza os pontos dado as coordenadas do objeto **geojson**.

Os atributos de *methods* incluem todos os métodos presentes neste componente, e podem ser chamados em outros *hooks* ou até dentro da seção **<template>** em algumas circunstâncias que serão apresentadas a seguir. Dentro deste *hook* existe o método **updateSource** que projeta

[10] Fonte: Andersen (2017), em março de 2022.

Listagem 4.1 – MapContainer.vue - Criação do mapa.

```
<!-- ------------------ MapContainer.vue ------------------ -->
<script>
  export default {
    props: { geojson: Object },
    data: () => ({
      map: null,
      vectorLayer: null,
      view: null
    }),
    mounted() {
      this.vectorLayer = new VectorLayer({
          source: new VectorSource({ features: [] })
      });
      this.map = new Map({
        target: this.$refs["map-root"],
        layers: [
          new TileLayer({ source: new OSM() }),
          this.vectorLayer
        ],
        view: this.view,
      });
      var fullscreen = new FullScreen();
      this.map.addControl(fullscreen);
      this.updateSource(this.geojson);
    },
    methods: {
      updateSource(geojson) {
        const view = this.map.getView();
        const source = this.vectorLayer.getSource();
        const features = new GeoJSON({ featureProjection: "EPSG:3857" }).
            readFeatures(geojson);
        source.clear();
        source.addFeatures(features);
        if(features.length > 5) { view.fit(source.getExtent()); }
      }
    }
  }
</script>
<!-- ------------------ Home.vue ------------------ -->
<template>
  <map-container
    /* ... */
    :geojson="geojson"
  ></map-container>
</template>
<script>
  methods: {
    async getData() {
      await getPoints().then(resultPoints => { /* ... */ });
    },
  }
</script>
```

Listagem 4.2 – Método de busca por todos os pontos.

```
async function getPoints() {
  return await query.execute(conn, 'database', 'SELECT *
  WHERE
  {
    ?point rdf:type :GeographicPoint .
    ?point wgs:lat ?latitude .
    ?point wgs:long ?longitude .
    ?point rdfs:label ?label .
    ?point rdfs:comment ?comment ;
    FILTER EXISTS {
      SELECT ?point WHERE {
        ?measurement a :Measurement ;
                     :locatedIn ?point .
      }
    }
  }', 'application/sparql-results+json');
}
```

os pontos no mapa, dadas as coordenadas dos pontos presentes no objeto **geojson**. O método também reposiciona a janela de visualização do mapa caso haja mais que 5 pontos geográficos apresentados, para evitar que o mapa aplique um *zoom* muito próximo ao ponto, adotando assim uma melhor visualização do mapa.

## 4.3 Componente relativo aos filtros

Na Listagem 4.3 é feita toda lógica por trás do envio dos filtros para realizar a query SPARQL. Quase todo código permanece na classe **Filters.vue**, porém o método para chamar a busca por medições pertence ao **Home.vue**.

Aqui já é possível observar a utilização da tag **<template>** e das tags disponibilizadas pelo Vuetify, neste caso são todas que possuem o prefixo **'v-'**. O ato de incorporar um outro componente a este é feito mediante ao **import**, sendo declarado no *asset* **components**. No exemplo demonstrado foram adicionados os componentes **Agents** e **WaterQualities**.

No Vue.js, diretivas são atributos das tags prefixadas com **'v-'** para indicar que são atributos especiais providos pelo Vue, aplicam comportamento especial de reatividade ao DOM renderizado. A diretiva apresentada neste código é a **'v-model'**, encontrada nos componentes do Vuetify **v-switch** e **v-text-field**, apresenta a interligação de mão dupla (*two-way binding*) entre a caixa de texto e o estado da aplicação, ou seja, o estado no DOM e no atributo são alterados ao mesmo tempo. Essa diretiva **'v-on'** ou **'@'** já executa o método a ela vinculada quando certa condição proposta é alcançada. Utilizando a linha 4 da Listagem 4.3 como exemplo, o método atrelado ao **@click** é disparado quando o componente **v-switch** recebe um clique do usuário.

O *data watch* é ativado quando o atributo ali apresentado sofre alguma alteração, assim executando o trecho de código descrito, conforme visto na linha 29 da Listagem 4.3. Logo que

Listagem 4.3 – Filters.vue | Home.vue - Campos do filtro.

```
<!-- ------------------ Filter.vue ------------------ -->
<template>
  <v-card width="100%" class="mx-auto">
    <v-card-title class="header"> Filtros <v-switch v-model="updateMap" inset
        primary label="Atualizar Mapa" @click="$emit('updateMap', updateMap)" /></v-
        card-title>
    <v-form ref="form">
      <v-text-field label="Data Inicial" v-model="filters.initialDate" type="date"
          />
      <v-text-field label="Data Final" v-model="filters.finalDate" type="date" />
      <agents @selectedAgents="getSelectedAgents" ref="agents" />
      <water-qualities @selectedWaterQualities="getSelectedWaterQualities" ref="
          waterQualities" />
      <v-card-actions>
        <v-btn @click="applyFilters" color="primary" class="ml-auto mr-0">
          <v-icon left>mdi-filter</v-icon> Filtrar
        </v-btn>
      </v-card-actions>
    </v-form>
  </v-card>
</template>
<script>
  import Agents from "./Agents.vue";
  import WaterQualities from "./WaterQualities.vue";
  export default {
    components: { WaterQualities, Agents },
    data() { return {
        filters: {initialDate:'',finalDate:'',agents:[],waterQualities:[],geoPoint:'
            '},
        updateMap: true,
        point: undefined,
    }; },
    props: { selectedPoint: Object },
    watch: { selectedPoint(value) { this.point = value; } },
    methods: {
      applyFilters() {
        if(this.point) { this.filters.geoPoint = this.point.get('point'); }
        else { this.filters.geoPoint = ""; }
        this.$emit("filters", { ...this.filters }, this.updateMap);
      },
      getSelectedAgents(selectedAgents) { this.filters.agents = selectedAgents; },
      getSelectedWaterQualities(selectedWaterQualities) { this.filters.
          waterQualities = selectedWaterQualities; },
    }
  };
</script>
<!-- ------------------ Home.vue ------------------ -->
<template>
  <filters @filters="applyFilters" />
</template>
<script>
  async applyFilters(filters, updateMap, updateTable = true) {
    await getMeasurementsWithFilters(filters).then(resultPoints => { /* ... */ });
  }
</script>
```

**selectedPoint** recebe um novo valor, seu método em *watch* é ativado, executando o código **this.point = value;**, o atributo **value** sendo o novo valor atribuido ao objeto.

Existem outros componentes incorporados ao **Filters.vue**, são eles o **Agent.vue** e **WaterQualities.vue**, que apenas fazem a busca no banco através das consultas apresentadas na Listagem 4.4 e criam a estrutura de **<v-select>** para o **Agent.vue** e a **<v-treeview>** para **WaterQualities.vue**, estes sendo incorporados ao componente pai, **Filters.vue**.

Listagem 4.4 – Método de busca por todos os Agents e WaterQualities.

```
async function getAgents() {
  return await query.execute(conn, 'database', 'SELECT ?agent
  WHERE { ?agent a :Agent . }', 'application/sparql-results+json');
}

async function getWaterQualities() {
  return await query.execute(conn, 'database', 'SELECT *
  WHERE {
    ?subQuality rdfs:subClassOf* <http://purl.org/nemo/gufo#Quality> .
    ?subQuality rdfs:subClassOf ?superQuality .
    ?subQuality rdfs:label ?label
    FILTER ( lang(?label) = 'en' )
  } ORDER BY ?superQuality', 'application/sparql-results+json');
}
```

O componente **v-btn** possui a diretiva que ao ser clicado, executa o método **applyFilters**, desta forma o sistema verifica se o ponto geográfico faz parte dos filtros para assim o incluir no objeto **filters**, tendo em vista que o ponto geográfico só pode ser selecionado no componente **MapContainer.vue** ou **MeasurementTable.vue**, conforme apresentado nas Figuras 7 e 9 e Listagem 4.9.

Desta forma o **$emit** é invocado, transferindo o objeto **filters** e o atributo booleano **updateMap** para o componente **Home.vue**, utilizando a palavra-chave **"filters"**. Este **$emit** tem a função de ativar um evento em um componente pai através de uma palavra-chave. Nele é possível enviar quaisquer parâmetros como função de *callback*. O componente **Home.vue** então recebe esses parâmetros e os envia para o método que o foi atribuído, o **async applyFilters**, que por sua vez faz a chamada do método **getMeasurementsWithFilters** para assim realizar a query SPARQL e retornar as medições através dos filtros, conforme Listagem 4.5. Assim então é apresentado os dados, preenchendo a tabela de dados, conforme visto na Figura 9.

O sistema também possui uma forma de limpar os campos do componente **Filter.vue** para melhor usabilidade do sistema, como é demonstrado na Listagem 4.6. Quando o botão **Limpar** é clicado, o método **cleanFilters** é acionado, atribuindo valores vazios à todos os atributos. Como os objetos dos componentes **Agents.vue** e **WaterQualities.vue** não podem ser acessados diretamente pelo **Filters.vue**, foram então invocados os métodos internos de cada, o **cleanAgents** e **cleanWaterQualities**, através do identificador de referência **$refs**. Para isto, é preciso que cada cada componente filho possua um atributo ref, assim é possível que o componente pai tenha permissão para acessar as instâncias dos filhos.

Listagem 4.5 – Retorna todas as medições aplicando os filtros.

```
async function getMeasurementsWithFilters(params) {
  let agentsCondition = getAgentCondition(params.agents);
  let waterQualitiesCondition = getWaterQualitiesCondition(params.waterQualities);
  return await query.execute(conn, 'database', 'SELECT *
  WHERE
  {
    ?measurement rdf:type :Measurement ;
                 gufo:hasBeginPointInXSDDateTimeStamp ?initialDate ;
                 gufo:hasEndPointInXSDDateTimeStamp ?finalDate ;
                 :measuredQualityKind ?qualityKind ;
                 gufo:hasQualityValue ?qualityValue ;
                 :locatedIn ?geoPoint ;
                 :expressedIn ?unit .
    ?geoPoint wgs:lat ?latitude ;
              wgs:long ?longitude ;
              rdfs:comment ?geoComment ;
              rdfs:label ?geoLabel .
    ?agent gufo:participatedIn ?measurement .
    ?qualityKind rdfs:label ?qualityKindLabel .
    FILTER (( lang(?qualityKindLabel) = 'en' )
      ${params.geoPoint ? "&& (?geoPoint = <" + params.geoPoint + ">)" : ""}
      ${params.initialDate ? "&& (?initialDate > \"" + params.initialDate + "\"^^xsd
          :dateTime)" : ""}
      ${params.finalDate ? "&& (?finalDate < \"" + params.finalDate + "\"^^xsd:
          dateTime)" : ""}
      ${agentsCondition}
      ${waterQualitiesCondition}
    ) .
  }
  ORDER BY ?geopoint', 'application/sparql-results+json', { limit: 200000 });
}
function getAgentCondition(agents) {
  let agentsCondition = "";
  if(agents.length) {
    for (let index = 0; index < agents.length; index++) {
      if(index == 0) { agentsCondition += "&& (?agent = <" + agents[index] + ">"; }
      else { agentsCondition += " || ?agent = <" + agents[index] + ">"; }
    }
    agentsCondition += ")";
  }
  return agentsCondition;
}
function getWaterQualitiesCondition(waterQualities) {
  let waterQualitiesCondition = "";
  if(waterQualities.length) {
    for (let index = 0; index < waterQualities.length; index++) {
      if(index == 0) { waterQualitiesCondition += "&& (?qualityKind = <" +
          waterQualities[index] + ">"; }
      else { waterQualitiesCondition += " || ?qualityKind = <" + waterQualities[
          index] + ">"; }
    }
    waterQualitiesCondition += ")";
  }
  return waterQualitiesCondition;
}
```

Listagem 4.6 – Filters.vue | Agents.vue | WaterQualities.vue - Limpar campos.

```
<!-- ------------------ Filters.vue ------------------ -->
<template>
  <agents ref="agents" />
  <water-qualities ref="waterQualities" />
  <v-btn color="secondary" @click="cleanFilters">
    <v-icon left>mdi-filter-off</v-icon> Limpar
  </v-btn>
</template>
<script>
  methods: {
    cleanFilters() {
      this.filters.initialDate = '';
      this.filters.finalDate = '';
      this.filters.agents = [];
      this.filters.waterQualities = [];
      this.deletePoint();
      this.$refs.agents.cleanAgents();
      this.$refs.waterQualities.cleanWaterQualities();
    }
  }
</script>
<!-- ------------------ Agents.vue ------------------ -->
<script>
  methods: { cleanAgents() { this.selectedAgents = []; } }
</script>
<!-- ------------------ WaterQualities.vue ------------------ -->
<script>
  methods: { cleanWaterQualities() { this.selectedWaterQualities = []; } }
</script>
```

## Seleção de pontos

A seleção de pontos é um processo em que são necessários quatro componentes devido a suas duas diferentes possibilidades de selecionar um ponto, são estes o **MapContainer.vue**, **Home.vue**, **Filters.vue** e **MeasurementTable.vue**, conforme demonstrado na Listagem 4.7. Existe a possibilidade do usuário selecionar um ponto geográfico direto do mapa ao clicar em um dos círculos azuis dispostos nele, vide Figura 6 ou em algum dos links apresentados na coluna Ponto Geográfico da tabela, conforme mostrado na Figura 9.

Quando o usuário clica em um dos pontos do mapa, o sistema então dispara o evento **click** a partir do objeto **map**, este então envia a seu pai (**Home,vue**) por meio de um $emit o objeto do ponto **clickedPoint** através da palavra-chave **"geoPoint"**. O **Home.vue** recebe então o objeto já atribuindo ao atributo **point** para ativar o *data props* do **Filters.vue**. Assim o ponto é exibido no componente **Filters.vue** e incluso aos campos do filtro assim que uma nova busca for feita.

Já para a seleção diretamente da tabela, o usuário clica em um link da coluna Ponto Geográfico e o método **geoPointClick** é disparado, enviando a latitude, longitude e label do ponto via **$emit** do **"openPopupMap"**. O **Home.vue** recebe os valores enviados e já altera o seu atributo **popupCoordinates**, assim ativando o *data* de **MapContainer.vue**. Aqui o componente faz o envio do mesmo **$emit** do **"geoPoint"**, enviando consigo o atributo **selectedPoint**, mesmo **$emit** utilizado ao clicar sob algum dos pontos no mapa, assim repetindo

todos os passos do primeiro caso.

Listagem 4.7 – MapContainer.vue | Home.vue | Filters.vue - Seleção de pontos.

```
<!-- ------------------- MapContainer.vue ------------------- -->
<script>
  mounted() {
    this.map.on("click", ev => {
      const clickedPoint = this.map.forEachFeatureAtPixel(ev.pixel, feature =>
          feature);
      if (clickedPoint) { this.$emit("geoPoint", clickedPoint); }
    });
  }
</script>
<script>
    data() { return { point: undefined }; },
    watch: { popupCoordinates: { this.$emit("geoPoint", selectedPoint); } }
</script>
<!-- ------------------- Home.vue ------------------- -->
<template>
  <filters :selectedPoint="point" />
  <map-container ref="mapContainer" @geoPoint="point = $event" :popupCoordinates="
      popupCoordinates"/>
  <measurement-table @openPopupMap="popupCoordinates = $event" />
</template>
<!-- ------------------- Filters.vue ------------------- -->
<script>
  props: { selectedPoint: Object },
  watch: { selectedPoint(value) { this.point = value; } },
  methods: {
    applyFilters() {
      if(this.point) { this.filters.geoPoint = this.point.get('point'); }
      else { this.filters.geoPoint = ""; }
      this.$emit("filters", { ...this.filters }, this.updateMap);
    }
  }
</script>
<!-- ------------------- MeasurementTable.vue ------------------- -->
<template>
  <template v-slot:item.geoLabel="{ item }">
    <a @click="geoPointClick(item.latitude, item.longitude, item.geoLabel)">{{ item.
        geoLabel }}</a>
  </template>
</template>
<script>
  methods: { geoPointClick(lat, long, geoLabel) { this.$emit('openPopupMap', [long,
      lat, geoLabel]); } }
</script>
```

## 4.4 Componente relativo à tabela de dados

A tabela de dados foi desenvolvida no componente **MeasurementTable.vue**, conforme visto na Listagem 4.8. Toda estrutura desta tabela foi baseada na própria documentação do Vuetify para criação de uma **v-datatable** [11].

A diretiva **'v-bind:'** ou apenas a abreviação **':'** funciona de maneira semelhante à **'v-model'**, porém esta apenas renderiza no DOM os valores atribuídos a eles por meio de alguma variável. Na tabela é possível enxergar o uso dessa diretiva em **:headers**, **:items** e **:search**,

[11] <https://vuetifyjs.com/en/components/data-tables/>

sendo estes o cabeçalho, corpo e campo de busca da tabela respectivamente.

A diretiva **'v-slot'** permite que se possa fazer alterações em templates preestabelecidos pelo seu fornecedor. Neste caso foi feita a substituição da exibição das colunas **measurementDate**, **measurementTime** e **geoLabel** da tabela proveniente do Vuetify, afim de personalizar os atributos apresentados nela.

Como a tabela reage a qualquer alteração nos dados, sendo constantemente atualizada, o objeto **items** recebe novos valores no momento em que o *prop* **measurements** sofre alguma alteração vinda do seu componente pai, **Home.vue**. O objeto recebe para cada atributo seu um valor, correspondente às colunas da tabela. Porém o único atributo omitido na tabela é **measurement**, por se tratar do campo identificador da medição, não havendo necessidade de apresentá-los em tela.

A *datatable* também possui uma outra funcionalidade, que é a possibilidade de se fazer o *download* das informações da tabela em um arquivo CSV (Comma-Separated Values). A Listagem 4.9 mostra que para isto foi necessária a importação da biblioteca VueJsonToCsv (NPM, 2019) para este componente. Nela existem as diretivas **:json-data** que recebe os dados nos quais serão transformados em CSV e **:csv-title** para o nome do arquivo, que utiliza o método **fileName** como parâmetro.

Os dados da tabela presentes no **filteredItems** são alterados a partir do momento em que alguma alteração é feita na tabela, o **v-data-table** executa o **@current-items**, no qual chama o método **getFiltered**.

O ato de recarregar o mapa para sua forma inicial é um ato simples, vide Listagem 4.10, apenas é feito um **$emit** para o componente **Home.vue** como **"rebootMap"**, conforme visto na linha 3. Isto invoca novamente o método **getData**, linha 11, que resulta na atualização do mapa com todos os pontos geográficos da base de triplas, utilizando o método visto na Listagem 4.2.

Listagem 4.8 – MeasurementTable.vue - Tabela de medições.

```
<template>
  <v-data-table :headers="headers" :items="items" :search="search">
    <template v-slot:item.measurementDate="{ item }"><span>{{ new Date(item.
        measurementDate).toLocaleDateString('pt-BR') }}</span></template>
    <template v-slot:item.measurementTime="{ item }"><span>{{ new Date(item.
        measurementTime).toLocaleTimeString(['pt-BR'], {hour: '2-digit', minute:'2-
        digit'}) }}</span></template>
    <template v-slot:item.geoLabel="{ item }"><a @click="geoPointClick(item.latitude
        , item.longitude, item.geoLabel)" v-bind="attrs" v-on="on">{{ item.geoLabel
        }}</a></template>
  </v-data-table>
</template>
<script>
export default {
  data() {
    return {
      headers: [
        {text:"ParâmetrodaQualidadedaÁgua",align:"start",value:"qualityKind"},
        {text:"ValordaQualidade",value:"qualityValue"},
        {text:"Unidade",value:"unit"},
        {text:"DatadaMedição",value:"measurementDate"},
        {text:"HoráriodaMedição",value:"measurementTime"},
        {text:"PontoGeográfico",value:"geoLabel"},
        {text:"Latitude",value:"latitude"},
        {text:"Longitude",value:"longitude"},
        {text:"Responsável",value:"agent"}
      ],
      search: "", items: [], filteredItems: [],
    };
  },
  props: { measurements: Array, load: Boolean },
  watch: {
    measurements(measurementList) {
      this.items = [];
      measurementList.forEach(measurement => {
        let dateTimeValue = new Date(measurement.initialDate.value);
        this.items.push({
          measurement: this.getURIName(measurement.measurement.value),
          qualityKind: measurement.qualityKindLabel.value,
          qualityValue: measurement.qualityValue.value,
          unit: this.getUnitName(measurement.unit.value),
          measurementDate: dateTimeValue,
          measurementTime: dateTimeValue,
          geoLabel: measurement.geoLabel.value,
          latitude: measurement.latitude.value,
          longitude: measurement.longitude.value,
          agent: this.getURIName(measurement.agent.value)
        });
      });
    }
  },
  methods: {
    geoPointClick(lat, long, geoLabel) { this.$emit('openPopupMap', [long, lat,
        geoLabel]); },
    getURIName(value) { return value.split("#")[1]; },
    getUnitName(value) { return value.split("unit/")[1]; }
  },
};
</script>
```

Listagem 4.9 – MeasurementTable.vue - Exportação do CSV.

```
<template>
  <vue-json-to-csv :json-data="filteredItems" :csv-title="fileName()" class="d-flex
      justify-end ml-3">
    <v-btn class="mx-2" fab small color="primary"><v-icon dark>mdi-download</v-icon
       ></v-btn>
  </vue-json-to-csv>
  <v-data-table
      /* ... */
      @current-items="getFiltered" />
</template>
<script>
import VueJsonToCsv from 'vue-json-to-csv'
export default {
  components: {
    VueJsonToCsv,
  },
  data() { return { filteredItems: [] }; },
  methods: {
    fileName() {
      var today = new Date();
      var date = today.getDate() + '-' + (today.getMonth()+1)+'-'+ today.getFullYear
          () + '-' + today.getTime();
      return 'datatable-' + date;
    },
    getFiltered(items) { this.filteredItems = items; }
  },
}
</script>
```

Listagem 4.10 – MapContainer.vue | Home.vue - Recarregar o mapa para o estado inicial.

```
<!-- ------------------ MapContainer.vue ------------------ -->
<template>
  <v-btn fab small color="primary" @click="$emit('rebootMap')">
    <v-icon large color="darken-2 green">mdi-restart</v-icon>
  </v-btn>
</template>
<!-- ------------------ Home.vue ------------------ -->
<template>
  <map-container
    /* ... */
    @rebootMap="getData()"
  ></map-container>
</template>
<script>
export default {
  methods: {
    async getData() {
      await getPoints().then(resultPoints => { /*... */ });
    },
  },
}
</script>
```

# 5 Conclusão

Este trabalho visou a produção de um sistema alternativo ao *front-end* desenvolvido anteriormente por Rabbi e Carvalho (2019) para a apresentação de dados de qualidade de água do Rio Doce. Esta meta foi concluída com êxito, com a implementação de todas as principais funcionalidades, com a utilização e integração do banco de triplas Stardog em um projeto em Vue.js, a criação de um mapa geográfico para apresentar todos os pontos de medição, uma tabela de dados com as informações das medições encontradas e por fim, um filtro que disponibiliza campos de data, responsável pela medição, parâmetro de qualidade de água e ponto geográfico para a realização de buscas mais refinadas. Todas estas funcionalidades são detalhadas e aprofundadas no capítulo 3.

## 5.1 Considerações finais

Pode-se destacar a meta atingida de aprendizagem sobre ontologias em geral e sobre a ontologia computacional que foi a protagonista deste trabalho, a Integradoce. Além disto, vale ser citado também o aprendizado sobre a realização de consultas através da linguagem SPARQL e a utilização do Vue.js integrado com diversas bibliotecas e frameworks, a saber, Vuetify, Stardog.js, OpenLayers e VueJsonToCSV, para no fim construir o protótipo funcional, totalmente vinculado com os dados provenientes do Stardog.

Algumas poucas limitações foram encontradas no caminho, como diversas tentativas frustradas de desenvolver um método para deixar a cargo do usuário quais colunas da tabela de dados seriam exibidas, porém o componente **<v-data-table>** possui pouca margem para personalização. Além das várias horas perdidas para tentar implementar um sistema de coloração nos pontos presentes no mapa, com o intuito de apresentar para o usuário quais áreas foram mais afetadas dado algum parâmetro de qualidade de água. Mas a API do OpenLayers é um tanto quanto complexa de se utilizar e sua documentação não é tão clara, portanto foi necessário abandonar a ideia por enquanto.

Uma funcionalidade que não foi desenvolvida por conta do tempo e conhecimento da biblioteca foi a de haver a possibilidade de selecionar vários pontos geográficos direto do mapa para serem utilizados no filtro. Novamente, por utilizar a API do OpenLayers que não possui uma documentação muito boa, esta ideia acabou sendo descartada, por falta de tempo e experiência com a ferramenta.

Trabalhar com este sistema foi uma tarefa no mínimo prazerosa e desafiadora, por tratar de tecnologias pouco abordadas nas disciplinas cursadas, que são as ontologias e consultas na linguagem SPARQL.

## 5.2 Trabalhos futuros

Como trabalhos futuros, podem ser destacados:

- A avaliação da aplicação com usuários finais para *feedback* de usabilidade e demais melhorias;
- O possível desenvolvimento de um *back-end* REST para simplificar o acesso aos dados da base Stardog;
- A implementação de uma funcionalidade no mapa que possibilite existir um *heatmap* (mapa de calor) referentes aos parâmetros de qualidade de água;
- A possibilidade de formatação da tabela de dados para permitir que o usuário possa configurar quais colunas ele queira exibir ou não;
- O desenvolvimento da funcionalidade de selecionar mais do que um ponto geográfico presente no mapa para utilização no filtro.

# Referências


ALMEIDA, J. P. A. The doce water quality ontology. 05 2020. Disponível em: <https://purl.org/nemo/doc/doce>. Citado 3 vezes nas páginas 13, 19 e 20.

ALMEIDA, J. P. A. et al. gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO). 2019. Disponível em: <https://nemo-ufes.github.io/gufo/>. Citado na página 20.

ANDERSEN, B. Vue Instance Lifecycle Hooks. 04 2017. Disponível em: <https://codingexplained.com/coding/front-end/vue-js/vue-instance-lifecycle-hooks>. Citado na página 34.

BECKETT, D. et al. Terse RDF Triple Language. 02 2014. Disponível em: <https://www.w3.org/TR/turtle/>. Citado na página 18.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. *Scientific american*, Springer Nature, v. 284, n. 5, p. 34–43, 2001. Citado na página 17.

CAMPOS, P. M. C. Designing a Network of Reference Ontologies for the Integration of Water Quality Data. Tese de Mestrado, Universidade Federal do Espírito Santo, 2019. Citado na página 20.

DBPEDIA. Global and Unified Access to Knowledge Graphs. 2022. Disponível em: <https://dbpedia.org/>. Citado na página 18.

EIS, D. O básico sobre Sparql e Turtle. 05 2017. Disponível em: <https://tableless.com.br/o-basico-sobre-sparql-turtle/>. Citado na página 14.

FRIAS, T. React vs Vue vs Angular: qual escolher? 04 2020. Disponível em: <https://blog.geekhunter.com.br/react-vs-vue-vs-angular-qual-escolher/>. Citado na página 24.

GUIZZARDI, G. et al. Towards ontological foundations for conceptual modeling: The unified foundational ontology (UFO) story. *Applied Ontology (Online)*, v. 10, p. 259–271, 2015. ISSN 18758533. Citado na página 20.

KEIL, J. M. IRI, URI, URL, URN and their differences. 11 2016. Disponível em: <https://fusion.cs.uni-jena.de/fusion/2016/11/18/iri-uri-url-urn-and-their-differences/>. Citado na página 17.

MAGALHÃES, A. L. O que é e para que serve o arquivo XML. 01 2020. Disponível em: <https://canaltech.com.br/software/xml-o-que-e/>. Citado na página 18.

MPMG. Rompimento da barragem de Fundão, em Mariana: resultados e desafios cinco anos após o desastre. *Ministério Público de Minas Gerais - MPMG*, Minas Gerais, 10 2020. Disponível em: <https://www.mpmg.mp.br/areas-de-atuacao/defesa-do-cidadao/direitos-humanos/noticias/rompimento-da-barragem-de-fundao-em-mariana-resultados-e-desafios-cinco-anos-apos-o-desastre.htm>. Citado na página 13.

NPM. vue-json-to-csv. 07 2019. Disponível em: <https://www.npmjs.com/package/vue-json-to-csv>. Citado 3 vezes nas páginas 15, 31 e 42.

NPM. Stardog.js. 05 2021. Disponível em: <https://www.npmjs.com/package/stardog>. Citado 3 vezes nas páginas 14, 15 e 26.

PAIVA, A. Artigo: o que é OpenLayers e sua importância em soluções de webmapping. 02 2019. Disponível em: <https://mundogeo.com/2019/02/11/artigo-o-que-e-openlayers-e-sua-importancia-em-solucoes-de-webmapping/>. Citado 2 vezes nas páginas 15 e 28.

PATEL, N. UI: Descubra O Que É e Qual A Importância Da User Interface. 05 2019. Disponível em: <https://neilpatel.com/br/blog/ui-o-que-e/>. Citado na página 25.

POPLADE, T. O que é Sass? Entenda esse outro método de escrever CSS. 06 2013. Disponível em: <https://tableless.com.br/sass-um-outro-metodo-de-escrever-css/>. Citado 3 vezes nas páginas 15, 25 e 33.

RABBI, V. T.; CARVALHO, V. A. d. Portal de dados sobre a qualidade da água do rio Doce: Um protótipo funcional. Revista IFES Ciência, 2019. Citado 4 vezes nas páginas 13, 14, 15 e 45.

SANTOS, F. C.; CARVALHO, C. L. d. Aplicações de Suporte à Web Semântica. Instituto de Informática, Universidade Federal de Goiás, p. 6–8, 12 2007. Citado 2 vezes nas páginas 17 e 18.

SOUZA, I. d. Saiba o que é NPM (Node Package Manager) e como instalar. 09 2020. Disponível em: <https://rockcontent.com/br/blog/npm/>. Citado na página 15.

STARDOG. The Enterprise Knowledge Graph Platform | Stardog. Virginia, 2021. Disponível em: <https://www.stardog.com>. Citado na página 13.

TECHENTER. IRI, URI, URL, URN and their differences. 01 2019. Disponível em: <https://techenter.com.br/o-que-sao-uri-url-e-urn/>. Citado na página 17.

TECHOPEDIA. Universal Coded Character Set (UCS). 07 2015. Disponível em: <https://www.techopedia.com/definition/9808/universal-coded-character-set-ucs>. Citado na página 17.

VUE.JS. Composição com Componentes. 2022. Disponível em: <https://br.vuejs.org/v2/guide/index.html#Composicao-com-Componentes>. Citado na página 25.

VUE.JS. Introdução — Vue.js. 2022. Disponível em: <https://br.vuejs.org/v2/guide/index.html>. Citado na página 23.

VUETIFY. Vuetify - A Material Design Framework for Vue.js. 2022. Disponível em: <https://vuetifyjs.com/en/>. Citado 2 vezes nas páginas 15 e 25.

W3C. World Wide Web Consortium Supports the IETF URI Standard and IRI Proposed Standard. 2004. Disponível em: <https://www.w3.org/2004/11/uri-iri-pressrelease>. Citado na página 17.

W3C. Web Ontology Language (OWL). 12 2012. Disponível em: <https://www.w3.org/OWL>. Citado na página 18.

W3C. Resource Description Framework (RDF). 02 2014. Disponível em: <https://www.w3.org/RDF>. Citado na página 17.

WROCLAWSKI, S. Por que o mundo precisa do OpenStreetMap. 01 2014. Disponível em: <https://gizmodo.uol.com.br/analise-openstreetmap/>. Citado na página 28.