UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATEUS QUEIROZ CUNHA

NODEGIS: SIMPLIFICANDO O DESENVOLVIMENTO DE APLICAÇÕES WEB DE GEOPROCESSAMENTO

CAMPINA GRANDE - PB

2021

MATEUS QUEIROZ CUNHA

NODEGIS: SIMPLIFICANDO O DESENVOLVIMENTO DE APLICAÇÕES WEB DE GEOPROCESSAMENTO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Cláudio de Souza Baptista

CAMPINA GRANDE - PB

2021

C972n Cunha, Mateus Queiroz.
NodeGIS: simplificando o desenvolvimento de aplicações web de geoprocessamento. / Mateus Queiroz Cunha. – 2021.

13 f.

Orientador: Prof. Dr. Cláudio de Souza Baptista.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Sistemas de informações geográficas web. 2. Geoprocessamento. 3. Aplicações web. 4. Web GIS. 5. Conteinerização. 6. REST – Representational State Transfer. I. Baptista, Cláudio de Souza. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

MATEUS QUEIROZ CUNHA

NODEGIS: SIMPLIFICANDO O DESENVOLVIMENTO DE APLICAÇÕES WEB DE GEOPROCESSAMENTO

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Cláudio de Souza Baptista

Orientador – UASC/CEEI/UFCG

Professora Dra. Francilene Procópio Garcia

Examinadora – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 20 de outubro de 2021.

CAMPINA GRANDE - PB

ABSTRACT

Associated with the ubiquity of spatial information nowadays, several tools which foster the development of web Geographical Information Systems (GIS) have emerged. Although there are a large number of solutions, many are complex, requiring specific skills from developers. Therefore, there is a need for a tool that simplifies the process of developing and publishing a web GIS application, with the advantage of being open source. In our study, we present NodeGIS, an open source tool that provides a graphical interface for the development of web GIS applications, without code writing or complex server configuration. NodeGIS uses a container-based architecture, using REST and facilitating the deployment of a web GIS application. The presented tool allows the user to plot vector maps, perform overlay and customization operations, zooming, panning, tooltip, conventional and spatial attribute queries. NodeGIS can also be used for teaching GIS, requiring no software installation from students.

NodeGIS: Simplificando o Desenvolvimento de Aplicações Web de Geoprocessamento

Mateus Queiroz Cunha
Laboratório de Sistemas de Informação
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
mateus.cunha@ccc.ufcg.edu.br

Cláudio de Souza Baptista
Laboratório de Sistemas de Informação
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
baptista@computacao.ufcg.edu.br

RESUMO

Diante da vasta presença da informação espacial nas aplicações atuais, surgiram várias ferramentas que fomentam o desenvolvimento de aplicações web de Sistemas de Informações Geográficas (GIS). Apesar de haver um grande número de soluções, muitas são complexas, requerendo habilidades específicas dos desenvolvedores. Portanto, há a necessidade de uma ferramenta que simplifique o processo de desenvolver e publicar uma aplicação web GIS, com a vantagem de ser de código aberto. Neste trabalho é apresentado o NodeGIS, uma ferramenta de código aberto que provê uma interface gráfica para o desenvolvimento de aplicações de web GIS, sem escrita de código ou configuração complexa de servidores. O NodeGIS utiliza-se de uma arquitetura baseada em contêineres, fazendo uso de REST, e facilitando o *deployment* de uma aplicação web GIS. A ferramenta apresentada permite ao usuário plotar mapas vetoriais, realizar operações de *overlay* e de personalização de camadas, *zooming*, *panning*, *tooltip*, consultas em atributos convencionais e espaciais. O NodeGIS também pode ser utilizado no ensino de GIS, não necessitando instalação de software por parte dos alunos.

Palavras-chave

GIS, web GIS, conteinerização, REST.

Repositório

https://github.com/mateusqc/node-gis

1 INTRODUÇÃO

Junto com a ubiquidade da informação espacial nas aplicações hodiernas, surgiram várias ferramentas comerciais e livres, bem como APIs que fomentam o desenvolvimento de aplicações de web GIS (Sistema de Informações Geográficas web). Grandes empresas do setor têm provido soluções de web GIS, como por exemplo a Microsoft Location Technologies - Azure Maps[1], Nokia Here SDK[2], Amazon AWS Location API[3], Google Maps API[4], deck.gl[5], Apple Maps API[6], mapbox[7], Uber API[8], ESRI ArcGIS Api[9], QGIS[10], MapServer[11], GeoServer[12], dentre muitas outras soluções. Também muitos gerenciadores de banco de dados provêem suporte nativo à dimensão espacial como os Relacionais: PostgreSQL/PostGIS, MySQL, SQL Server, Oracle, e IBM DB2.; os NewSQL: SAP Hana, CockroachDB e SingleStore; e os NoSQL: MongoDB, CouchDB, Cassandra, dentre outros.

No âmbito de GIS desktop há um bom número de soluções comerciais e livres, sendo QGIS e gvSIG[13] soluções livres de destaque. Todavia, quando partimos para soluções Web GIS [3], existem inúmeras soluções usando servidores de mapas, servidores de banco de dados espaciais e *frontends*, mas com complexidade diversa, exigindo do desenvolvedor habilidades específicas nas diversas tecnologias utilizadas nestas três camadas [2, 4–7]. Falta, portanto, uma ferramenta que simplifique o processo de desenvolver e publicar uma aplicação web GIS, ainda mais de forma gratuita. Este é então o objetivo da ferramenta NodeGIS aqui proposta [1].

O NodeGIS é uma ferramenta de código aberto que provê interface gráfica para o desenvolvimento de um web GIS, sem a necessidade de desenvolver código ou configurar servidores. O *deployment* é feito via contêineres Docker[14], onde são construídas duas imagens, uma para o *frontend* e outra para o *backend*, além de ser utilizado um contêiner PostgreSQL/PostGIS para o funcionamento básico da aplicação.

A ideia é trazer a fácil experiência do desenvolvimento de aplicações desktop GIS para o ambiente Web. Ademais, a ferramenta auxilia no processo de ensino e aprendizagem em geoprocessamento, provendo o professor e alunos de uma ferramenta que permite explorar graficamente e interativamente, diversos recursos de GIS, sem a necessidade de instalação de software, utilizando-se tão somente de um navegador com conexão à internet.

No restante deste artigo, apresenta-se na seção 2 a arquitetura e as principais funcionalidades do NodeGIS, seguida da seção 3 que descreve como criar um web GIS utilizando o NodeGIS e, posteriormente, a 4 que detalha a avaliação da solução desenvolvida. As considerações finais são objeto da seção 5.

[1] https://azure.microsoft.com/en-us/services/azure-maps/
[2] https://developer.here.com/
[3] https://aws.amazon.com/pt/location/
[4] https://developers.google.com/maps
[5] https://deck.gl/
[6] https://developer.apple.com/maps/
[7] https://www.mapbox.com/
[8] https://developer.uber.com/
[9] https://developers.arcgis.com/
[10] https://www.qgis.org/
[11] https://mapserver.org/
[12] http://geoserver.org/
[13] http://www.gvsig.com/pt/produtos/gvsig-desktop
[14] https://www.docker.com/

2 SOLUÇÃO PROPOSTA

Dentre as principais funcionalidades do NodeGIS destacam-se: adição de camadas vetoriais, consultas convencionais e espaciais; personalização das camadas do mapa; construção de mapas temáticos; *tooltip*; consulta interativa no mapa, onde é possível realizar consultas espaciais complexas de forma facilitada pela interface da aplicação, não havendo necessidade de conhecimentos de SQL; busca de feições no mapa; utilização de múltiplos bancos de dados espaciais; visualização dos dados de uma camada em tabelas, bem como filtros e seleções que podem ser aplicados à estes dados e visualizados no mapa.

Na figura 1, é apresentado um exemplo da interface do NodeGIS contemplando as seguintes camadas: mapa dos municípios da Paraíba, ferrovias e rodovias do estado. Além disso, na figura 1 também é mostrado uma consulta espacial com uma operação espacial de *buffer* da principal ferrovia do estado da Paraíba, destacado em verde.

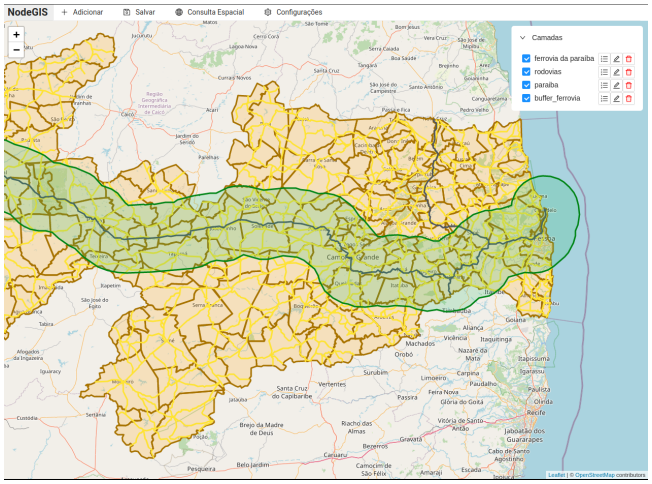

Figura 1: Mapa no NodeGIS contendo os municípios da Paraíba, suas Rodovias, Ferrovias e com operação espacial de *buffer* aplicada

O projeto arquitetural do NodeGIS baseia-se numa arquitetura REST[15], segmentada em *frontend*, *backend* e associada a um ou mais bancos de dados relacionais espaciais, conforme detalhado na figura 2. Serão detalhados nesta seção cada aspecto arquitetural, especificidades e tecnologias utilizadas no desenvolvimento do NodeGIS.

2.1 *Frontend*

O módulo de *frontend* do NodeGIS consiste em uma aplicação React[16], que utiliza uma implementação do padrão Flux[17] de fluxo de dados de *frontend*, apresentado na figura 3. O padrão Flux possui quatro principais elementos: a *View*, que são os componentes React que exibirão os dados da aplicação ao usuário; o *Store*, estruturas

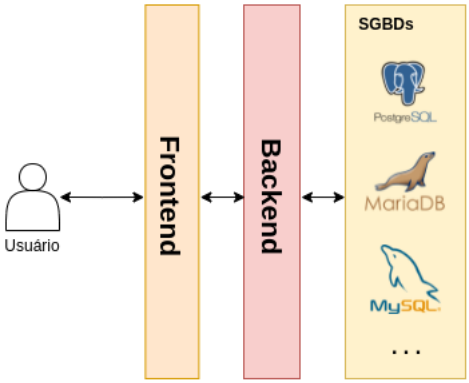

Figura 2: Arquitetura base do NodeGIS

responsáveis por obter, manipular e armazenar estes dados na aplicação; o *Dispatcher*, responsável por gerenciar o fluxo de dados, distribuindo as ações provenientes da *View* para o(s) seu(s) respectivo(s) *Store(s)*; e, por fim, a *Action*, que são funções que carregam consigo dados destinados aos *Stores* e provenientes de ações do usuário.

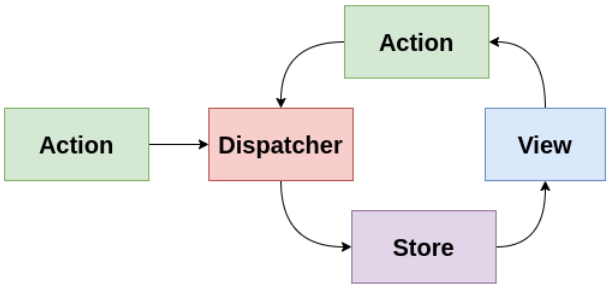

Figura 3: Estrutura de fluxo de dados de *frontend*

Existem diferentes implementações do padrão Flux, porém a utilizada foi em associação com o MobX[18], uma biblioteca de gerenciamento de estado que desempenha o papel da *Action* e do *Dispatcher*, simplificando a implementação do padrão arquitetural. Na figura 4 é apresentado o detalhamento da arquitetura de *frontend* e como os seus componentes se comunicam. A camada do padrão Flux está abstraída, já que é transparente ao desenvolvedor com a utilização do MobX.

O principal elemento do *frontend* é o mapa, desenvolvido com o auxílio da biblioteca Leaflet[19], que possui uma versão específica[20] para utilização com componentes React. Todos os dados utilizados no mapa estão armazenados em seu respectivo *Store* e são obtidos como respostas às requisições HTTP ao *backend* (realizadas com a utilização da biblioteca Axios[21]) onde os dados geográficos são

[15] https://pt.wikipedia.org/wiki/REST
[16] https://pt-br.reactjs.org/
[17] https://facebook.github.io/flux/
[18] https://mobx.js.org/
[19] https://leafletjs.com/
[20] https://react-leaflet.js.org/
[21] https://www.npmjs.com/package/axios/

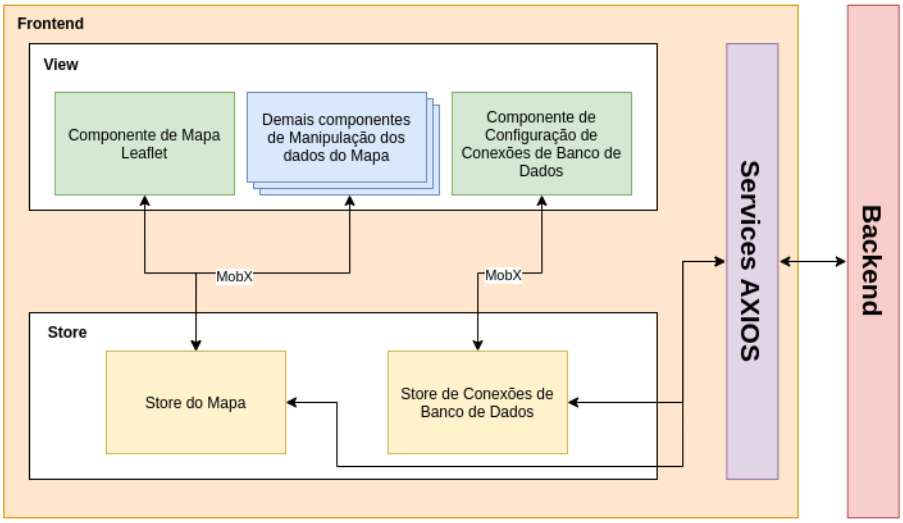

Figura 4: Detalhamento de arquitetura do *frontend*

convertidos no formato GeoJSON[22]. O Leaflet é capaz de interpretar os dados numa camada no mapa diretamente a partir do GeoJSON, não necessitando de nenhum tratamento ou estruturação adicional dos dados geográficos.

No instante em que o usuário seleciona a funcionalidade de adicionar uma nova camada vetorial ao mapa, é feita uma requisição ao *backend* que retorna todas as tabelas do banco de dados que possuem ao menos uma coluna do tipo geometria, bem como quais são as colunas de geometria presentes na tabela. O restante das definições é realizado diretamente no *frontend*, como a definição dos dados que serão exibidos no *tooltip*, a estilização da camada e o nome da camada no mapa.

É possível salvar o estado atual do mapa na aplicação, para que ao fechar e acessar novamente o NodeGIS as camadas já sejam exibidas juntamente com todos os seus metadados associados, tendo assim a mesma estilização e dados definidos para o *tooltip*. Além disso, a interface do NodeGIS é construída com o auxílio da biblioteca de componentes React chamada Ant Design[23].

Os dados de uma dada camada são gravados em uma estrutura do *Store* do mapa, onde cada camada é representada por um objeto JavaScript[24]. O objeto em questão possui formato que segue o padrão exibido na figura 5. Tem-se as seguintes atribuições para cada um dos atributos deste objeto:

- *name*: nome dado à camada;
- *key*: tabela do banco de dados que a camada representa;
- *geometryColumn*: tipo de dado da coluna geométrica utilizada na camada, podendo ser *Polygon*, *LineString* ou *Point*;
- *styles*: objeto contendo as definições de estilo estático da camada (caso o atributo *styleType* seja igual a *static*);
- *styleType*: atributo que define qual o tipo da estilização da camada, se é estático (*static*) ou temático (*choropleth*);
- *displayColumns*: lista de objetos que representam as colunas que terão os seus valores exibidos no *tooltip* das feições da camada no mapa;
- *choroplethStyleDefinition*: objeto contendo as definições de estilo para mapas temáticos (caso o *styleType* seja igual a *choropleth*);
- *data*: GeoJSON retornado por consulta ao banco de dados e recebido como resposta à requisição HTTP feita ao *backend*.

```json
{
  "name": "paraiba",
  "key": "paraiba",
  "type": "polygon",
  "geometryColumn": "geom",
  "styles": {
    "fillColor": "#3388ff",
    "fillOpacity": 0.2,
    "color": "#3388ff",
    "weight": 3,
    "opacity": 1
  },
  "styleType": "static",
  "displayColumns": [
    {
      "column": "nome",
      "label": "Nome",
      "key": "nome"
    }
  ],
  "choroplethStyleDefinition": {
    "colorFunction": null,
    "equal": false,
    "column": null,
    "defaultColor": "#3388ff",
    "values": []
  },
  "data": {}
}
```

Figura 5: Dados de camada vetorial em *frontend* representados em formato JSON

[22] https://geojson.org/
[23] https://ant.design/
[24] https://www.javascript.com/

Estes dados são utilizados na renderização dos elementos no mapa, onde são construídos componentes React do Leaflet. O componente chamado *MapContainer* define o contexto externo do mapa utilizado na aplicação e onde os demais componentes, respectivos a cada elemento adicionado neste mapa, deverão ser renderizados. Os dados das camadas são consumidos a partir do *Store* do mapa, possuindo o formato conforme o exibido na figura 5. O atributo data do objeto de uma camada vetorial é então utilizado para renderizar as feições no mapa por meio do componente *GeoJSON*, que aceita diretamente dados definidos no formato de nome análogo ao seu. Entretanto, para dados do tipo *Point*, decidiu-se utilizar o componente *CircleMarker*, conforme exibido na figura 6, que destaca de forma mais significativa a feição e permite a sua estilização. Ao utilizar o componente *GeoJSON* para este tipo de dado foram exibidos marcadores de estilo fixo, o que poderia confundir o usuário no caso de múltiplas camadas deste tipo presentes no mapa.

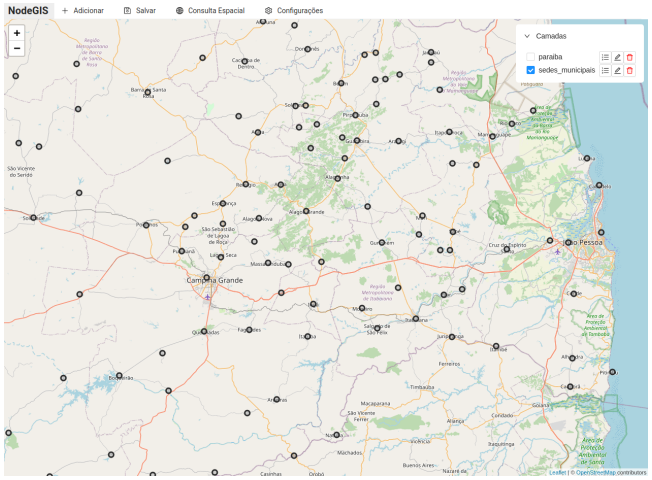

Figura 6: Camada vetorial com dados de geometria do tipo *Point* representados em formato JSON

Outra ação essencial com definições no *frontend* é a realização de consultas espaciais, que podem ser realizadas de duas formas: consultas SQL ou seleção de feições no mapa. A consulta espacial SQL possui um redirecionamento da consulta informada manualmente pelo usuário para o *backend* e, posteriormente, o banco de dados, respondendo-a com o resultado (inclusive erros). Já se tratando da consulta por seleção de feições no mapa, as feições podem ser selecionadas diretamente no mapa, conforme a figura 7, assim como camadas inteiras, de acordo com a operação espacial. Tais operações vão desde as que resultam em outras geometrias, como união e contém, até as que resultam em números, como perímetro, área e distância.

As consultas espaciais possuem a sua estrutura construída no *frontend*, entretanto, dados específicos são enviados ao *backend*, por meio de requisição HTTP, para a realização da consulta propriamente dita. Esta consulta é então construída e posteriormente enviada ao banco de dados.

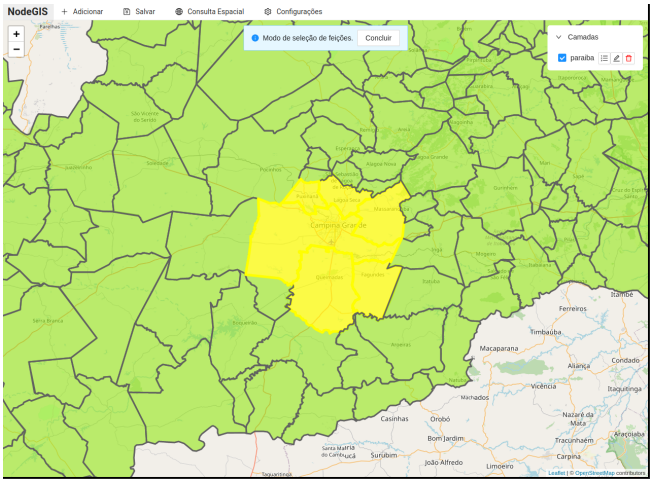

Figura 7: Seleção de feições do mapa para operação espacial de união

2.2 *Backend*

O *backend* do NodeGIS consiste num servidor web capaz de responder requisições HTTP, seguindo um padrão de API REST. Esta API foi construída utilizando NodeJS em conjunto com o *framework* express[25]. A arquitetura utilizada segue conforme apresentado na figura 8, dividindo-se em: rotas, camada de acesso que redireciona as requisições HTTP aos seus respectivos *controllers*; *controllers*, camada que trata as requisições feitas à API, bem como as suas respostas, a partir dos dados vindos do *repository* após seu processamento; *repository*, que armazenam a lógica de construção de cada consulta espacial, bem como o gerenciamento dos demais dados e construção das consultas ao banco de dados em geral; e os clientes de conexão ao(s) banco(s) de dados.

De forma a prover versatilidade e possibilidade de escolha ao usuário, o NodeGIS possui compatibilidade com diferentes SGBDs (Sistemas de Gerenciamento de Bacos de Dados) relacionais espaciais, sendo eles: PostgreSQL[26], MySQL[27], MariaDB[28], SQLite[29] (em conjunto com sua extensão, o SpatiaLite [30]) e CockroachDB[31]. Para conseguir lidar com os diferentes SGBDs, foi implementado um cliente específico para cada tipo suportado, encapsulando o cliente NodeJS já existente para o banco de dados específico, sendo este também responsável por tratar eventuais divergências nas implementações SQL. Além dos bancos de dados que são possíveis de acoplar à aplicação, o *backend* também possui uma instância do SQLite internamente. Esta instância auxilia no gerenciamento dos dados da aplicação, armazenando quais as camadas que o usuário gravou em sua visualização no NodeGIS, além de armazenar os dados de conexão aos demais bancos de dados utilizados.

A maioria dos *repositories* definidos no *backend* apenas redireciona as chamadas à API para o banco de dados por meio de consultas

[25] https://expressjs.com/pt-br/
[26] https://www.postgresql.org/
[27] https://www.mysql.com/
[28] https://mariadb.org/
[29] https://www.sqlite.org/index.html
[30] https://www.gaia-gis.it/fossil/libspatialite/index
[31] https://www.cockroachlabs.com/

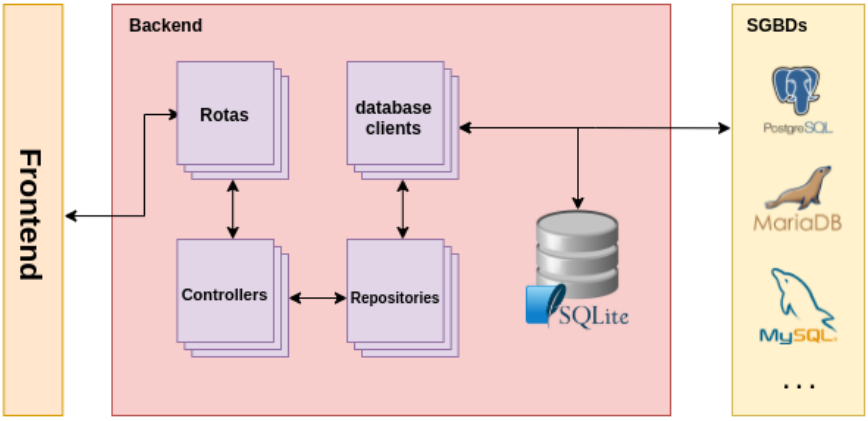

Figura 8: Detalhamento de arquitetura do *backend*

básicas, exceto pelo *queryRepository*, responsável pela lógica de construção das consultas espaciais vindas do *frontend*. Dentre as operações espaciais suportadas, estão: união, diferença, intersecção, contém, cruza, toca, disjunção, intersecta, *buffer*, centróide, área, distância, comprimento e perímetro.

Para que a consulta espacial seja construída, o *frontend* realiza uma requisição HTTP ao *backend* contendo um *body* no formato apresentado na figura 9. Este objeto é composto pelas seguintes propriedades: *first*, primeiro parâmetro da consulta espacial; *second*, segundo parâmetro da consulta espacial; e *operation*, uma *string* que representa qual a operação espacial que deverá ser aplicada. Para consultas que não existe comparação entre diferentes feições (ou conjunto de feições), a propriedade *second* poderá estar vazia. Este é o caso de operações como *buffer* e área, por exemplo.

```
{
  "first": {
    "paraiba": {
      "data": [152, 161, 19],
      "geometryColumn": "geom"
    }
  },
  "second": {
    "paraiba": {
      "data": [],
      "geometryColumn": "geom"
    }
  },
  "operation": "contains"
}
```

Figura 9: JSON enviado ao *backend* em consultas espaciais

De forma mais detalhada, as propriedades *first* e *second* armazenam as tabelas utilizadas na consulta como chaves do objeto da propriedade, além de cada uma dessas tabelas também armazenarem propriedades, que são a *data* e a *geometryColumn*. A propriedade data guarda uma lista de identificadores únicos das feições presentes na tabela, chamados de GIDs. Os valores de GID presentes na lista referenciam as feições que devem ser utilizadas na consulta. Caso a lista esteja vazia, a tabela inteira será considerada. A propriedade *geometryColumn*, armazena qual coluna de dados geométricos deve ser referenciada para aquela tabela na consulta a ser realizada.

Para cada um dos parâmetros da consulta espacial é construída uma subconsulta que obtém os dados solicitados, sendo simplesmente uma união das feições solicitadas (caso existam GIDs de feições na propriedade *data*) ou a tabela por completo (caso a propriedade *data* esteja vazia). A partir das subconsultas, a função espacial é aplicada, de acordo com a *string* passada na propriedade operation, direcionando as colunas espaciais de cada subconsulta como parâmetros para a função espacial.

Ao final da construção, esta consulta é envolvida pela função *ST_AsGeoJSON*, de forma a converter os dados de geometria para o formato GeoJSON, tornando mais simples manipular, ler e representar estes dados no mapa (conforme citado na seção anterior).

2.3 Conteinerização

De forma a facilitar as ações voltadas ao *deployment* da aplicação, foram implementadas definições para a conteinerização utilizando o Docker. Tanto o *frontend* quanto o *backend* possuem seus respectivos Dockerfiles, arquivos de configuração da etapa de *build* de contêineres Docker, que definem quais as imagens Docker base e intermediárias serão utilizadas na construção da imagem final, além de parâmetros específicos desta construção.

A primeira imagem base utilizada no *frontend* é a do NodeJS com o Alpine Linux[32], onde é feito o *build* da aplicação React. Este processo ocorre com o auxílio do Webpack[33], um empacotador de módulos JavaScript, bem como o Babel[34], um transpilador JavaScript. O resultado é um conjunto de arquivos JavaScript, CSS e HTML, que são então movidos para o diretório interno de uma imagem NGINX[35], também executando num Alpine Linux. O NGINX será responsável por servir, em forma de resposta às requisições HTTP, o resultado do processo de *build* da aplicação React.

[32] https://www.alpinelinux.org/
[33] https://webpack.js.org/
[34] https://babeljs.io/
[35] https://www.nginx.com/

Já no que diz respeito ao *backend*, foi utilizada a imagem base do NodeJS com Alpine Linux, assim como no *frontend*. Entretanto, por já se tratar de uma aplicação NodeJS, não é necessário utilizar nenhuma imagem Docker adicional para a construção da imagem final do *backend*. O conteúdo do diretório de *backend* no repositório do NodeGIS é copiado diretamente para a imagem Docker, onde é então executado diretamente pelo ambiente NodeJS.

Foi definido um arquivo para auxiliar no processo de construção destas imagens Docker, o Makefile[36]. Este arquivo contém os comandos e parâmetros corretos para a execução do processo de build das imagens, bastando executar o comando *"make build"* na raiz do repositório. Entretanto, para utilizar o NodeGIS por meio de contêineres Docker não é necessário construir as imagens do zero, pois já existem imagens devidamente construídas da aplicação disponibilizadas publicamente no DockerHub[37].

De forma a facilitar o *deployment* do NodeGIS, foram escritos dois *scripts* Shell[38] presentes no diretório raiz do repositório. Um dos *scripts* é direcionado para a execução da aplicação num ambiente local (arquivo chamado *run-application-local.sh*), já o outro para um ambiente de produção com acesso público (arquivo chamado *run-application-prod.sh*). Estes *scripts* já possuem os parâmetros e variáveis de ambiente de cada contêiner configurados corretamente, entretanto também podem ser alterados livremente de acordo com as necessidades específicas do ambiente em que a aplicação será disponibilizada.

Além dos contêineres de *frontend* e *backend*, também foi construído um contêiner PostgreSQL com PostGIS[39] contendo uma base de dados com tabelas que possuem dados espaciais. O banco de dados contido neste contêiner já é configurado pela aplicação como sendo o padrão, já estando adicionado nas configurações assim que a aplicação for executada. Na figura 10, a arquitetura da solução utilizando a conteinerização Docker é apresentada.

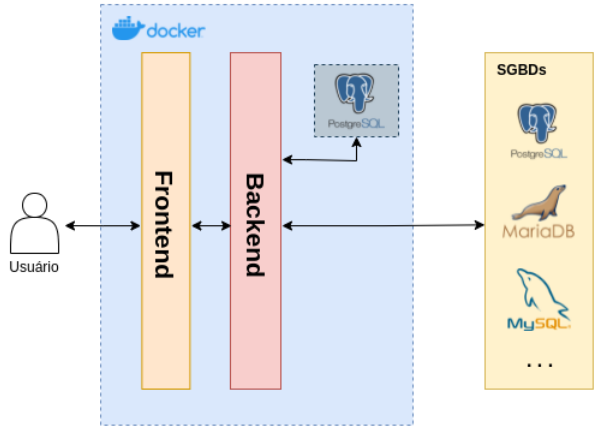

Figura 10: Arquitetura base do NodeGIS utilizando conteinerização Docker

[36]https://pt.wikipedia.org/wiki/Makefile
[37]https://hub.docker.com/
[38]https://pt.wikipedia.org/wiki/Shell_script
[39]https://postgis.net/

3 CRIAÇÃO DE WEB GIS UTILIZANDO O NODEGIS

Um dos focos no desenvolvimento do NodeGIS foi o de ser provida uma aplicação que tivesse uma fácil inicialização, de forma a minimizar impactos de integração entre *frontend*, *backend* e banco de dados, bem como influências do ambiente onde esta aplicação será utilizada (sistema operacional, por exemplo). A conteinerização é fundamental neste processo. A instalação do Docker é o único pré-requisito para a execução do NodeGIS.

3.1 Configuração Inicial

Foram disponibilizados, no diretório raiz do repositório[40] da aplicação, *scripts* Shell que aceleram a ação de inicialização da aplicação, já possuindo as definições de parâmetros passados aos contêineres Docker em sua execução. Os scripts são os *"run-application-local.sh"* e *"run-application-prod.sh"*. Ambos realizam esta configuração, entretanto, com parâmetros e propósitos diferentes (conforme a sugestão da nomenclatura, o primeiro deles trata de um ambiente local, enquanto o outro de um ambiente de produção).

Ao iniciar a execução do *script* desejado (com o comando *"bash run-application-local.sh"*, por exemplo), será verificada a presença de três imagens Docker na máquina em questão: a do *frontend*, do *backend* e a do banco de dados PostgreSQL com dados geográficos. Caso estas imagens ainda não existam, serão obtidas do DockerHub, onde estão disponíveis publicamente. Após o *download*, os três contêineres serão inicializados com os seus respectivos parâmetros, sendo exibido no terminal uma saída semelhante à presente na figura 11, contendo o código *hash* de cada contêiner Docker inicializado, bem como a URL de acesso para a aplicação.

```
b6585a13cde8476f40fc9b03ce7237069bb3483242b62333cca66ab38bb25950
19f76c6705000cb65c9f91d1c247b908386929e9f206cc6cd410c52c3bcf07be
a5212c43866b020475871aa4c93034750a95b1d816bde5e9b4b1a09138cfa293
Access application running at: http://localhost:8080
```

Figura 11: Saída do *script* de inicialização do NodeGIS

A execução dos *scritps* é feita da mesma forma, tanto para o ambiente de produção quanto para o local. Entretanto, deve-se atentar que, o NodeGIS executando a partir do *script* de produção depende da configuração da máquina hospedeira para que usuários externos possam acessar. O ambiente em questão deverá possuir um IP público, além de possuir as portas 8080 e 8081 desbloqueadas. Para interromper a execução do NodeGIS, é possível utilizar o *script stop-application.sh*, que utiliza-se dos nomes atribuídos aos contêineres no *script* de execução.

Todos os parâmetros presentes nos *scripts* podem ser alterados de acordo com a necessidade do usuário. Um exemplo deste cenário é quando já se possui um banco de dados contendo dados espaciais, onde a inicialização do contêiner do PostgreSQL torna-se opcional, ou quando deseja-se alterar as portas utilizadas no ambiente de produção.

3.2 Adicionando Camadas Vetoriais

De posse da aplicação devidamente inicializada, o usuário é capaz de utilizar as diversas funcionalidades do NodeGIS por meio do

[40]https://github.com/mateusqc/node-gis

menu superior. A funcionalidade de adicionar camadas vetoriais ao mapa encontra-se no menu “Adicionar”, seguido da opção “Camada Vetorial”. No modal exibido, conforme o destacado na figura 12, o usuário é capaz de selecionar qual tabela será utilizada, bem como a sua coluna espacial. Também pode realizar uma personalização da informação exibida no *tooltip* das feições, bem como do estilo da camada.

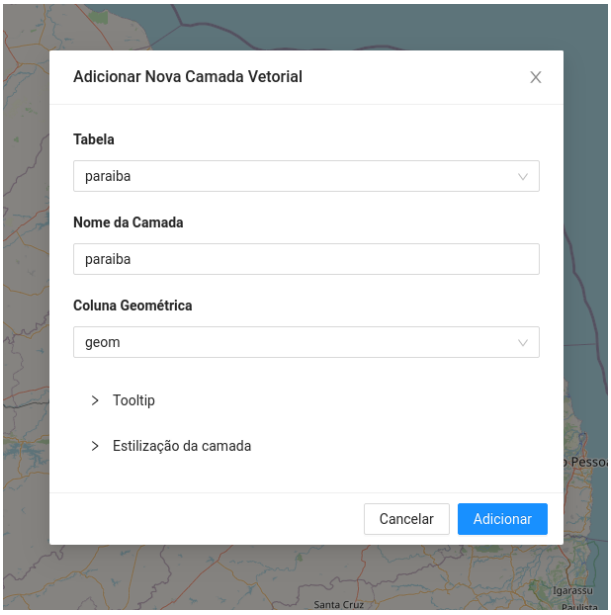

Figura 12: Modal de adição de camadas vetoriais

Ao clicar no botão “Adicionar” sem realizar nenhuma outra definição, a camada será adicionada ao mapa, porém o *tooltip* não será exibido e o estilo receberá cores padrões do sistema, conforme exemplificado na figura 13. O estado atual do mapa poderá ser gravado, sendo então a visualização padrão ao acessar o NodeGIS, onde a funcionalidade se encontra na seção “Salvar”, opção “Estado Atual do Mapa”.

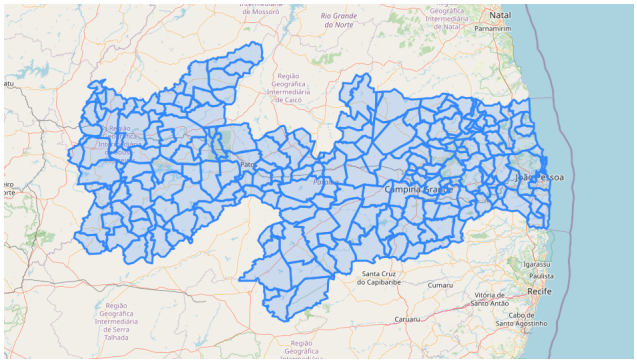

Figura 13: Camada adicionada com definição padrão de estilização

3.3 Realizando Consultas Espaciais

A funcionalidade de consultas espaciais a partir de seleção pode ser acessada no menu “Consulta Espacial”, opção “A Partir de Seleção”. Esta funcionalidade permite que o usuário selecione a operação desejada e os parâmetros necessários para a sua realização (feições ou camadas). Pode-se visualizar na figura 14 a interface de consultas espaciais, na qual já apresenta o resultado de uma consulta espacial de distância entre duas feições do mapa, que foi 383,93 km.

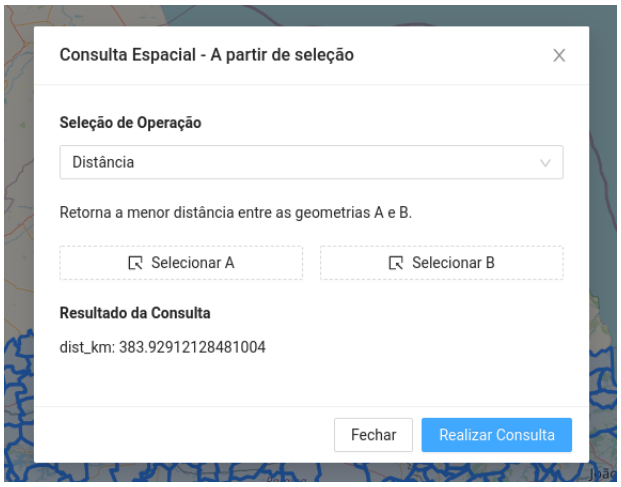

Figura 14: Interface de execução de consultas espaciais, além de resultado de uma consulta espacial de distância

A partir da mesma interface, podem ser realizadas consultas que resultam em geometrias, como a operação de um *buffer*. Por exemplo, ao realizar um *buffer* de 20 km na geometria do município de Campina Grande, será obtido um resultado tal qual exibido na figura 15. Note que o estilo da feição pôde ser alterado para uma melhor visualização. Também é possível gravar o resultado de consultas espaciais em novas tabelas no banco de dados, havendo assim a possibilidade de utilizá-lo em novas consultas, bem como salvar como a visualização padrão do NodeGIS.

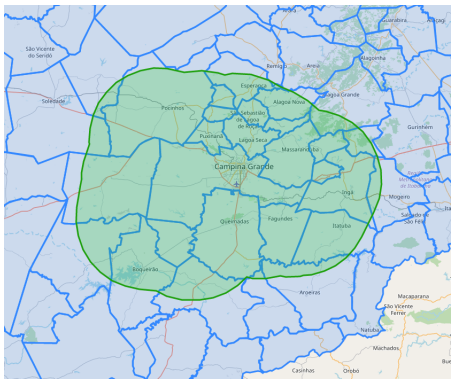

Figura 15: Camada contendo resultado de consulta espacial com operação de *buffer*

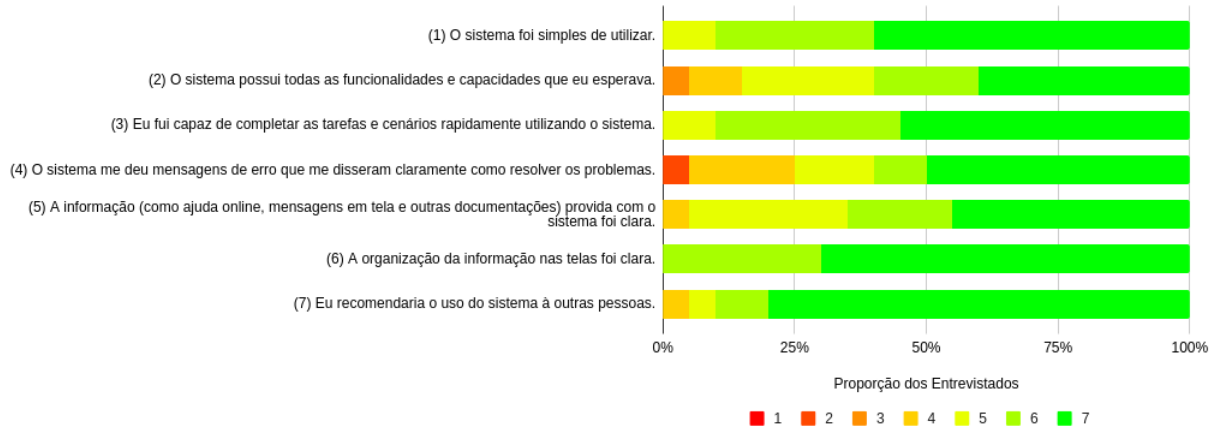

Figura 16: Respostas do Questionário de Avaliação do NodeGIS

Desta forma, o usuário poderá de forma facilitada e rápida: inicializar o NodeGIS, adicionar camadas vetoriais ao mapa, realizar consultas espaciais e salvar visualizações geradas a partir de tabelas existentes ou de consultas obtidas com o próprio NodeGIS.

4 AVALIAÇÃO DA FERRAMENTA

De forma a avaliar o NodeGIS, foi realizada uma avaliação com 20 usuários, utilizando-se de um questionário. Foi feita uma uma adaptação do *Post-Study System Usability Questionnaire*[41] (PSSUQ), em sua versão 3, para a construção do questionário, de forma a se ter apenas questionamentos que sejam relevantes de acordo com o cenário apresentado para o NodeGIS.

O objetivo final da aplicação deste questionário é obter métricas a respeito da satisfação de diferentes usuários com a ferramenta. O questionário contém oito afirmações, nas quais os usuários devem classificá-las de acordo com as suas respectivas experiências ao utilizar a ferramenta. Foram utilizadas as seguintes afirmações no questionário:

(1) Eu possuo conhecimentos de GIS (Sistemas de Informações Geográficas).
(2) O sistema foi simples de utilizar.
(3) O sistema possui todas as funcionalidades e capacidades que eu esperava.
(4) Eu fui capaz de completar as tarefas e cenários rapidamente utilizando o sistema.
(5) O sistema me deu mensagens de erro que me disseram claramente como resolver os problemas.
(6) A informação (como ajuda online, mensagens em tela e outras documentações) provida com o sistema foi clara.
(7) A organização da informação nas telas foi clara.
(8) Você recomendaria o uso do sistema a outras pessoas.

Cada uma das afirmações foram avaliadas com uma pontuação de 1 a 7, onde o 1 significa discordo totalmente e o 7 concordo totalmente. Ao iniciar o processo de avaliação, os usuários tiveram acesso a um vídeo descritivo[42] contendo as principais funcionalidades do NodeGIS. Além disso, foram repassadas aos entrevistados duas atividades simples: adicionar uma camada vetorial ao mapa e realizar uma consulta espacial com a mesma. Em seguida, a utilização da ferramenta foi livre e sem supervisão.

Voltando a atenção para a figura 17, que contém o resultado da Pergunta 1, pode-se perceber que um pouco mais de 50% dos entrevistados se declararam conhecedores das técnicas de GIS, o que nos proporciona um bom balanceamento na presença de pessoas que já tiveram contato com outras ferramentas de GIS e as que não.

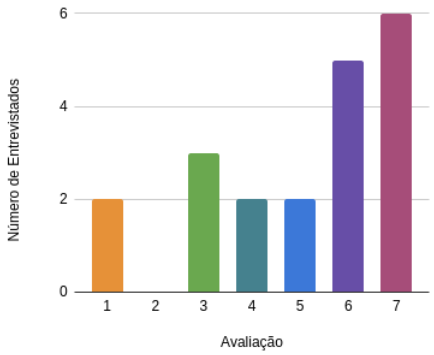

Figura 17: Avaliações da Pergunta "Eu possuo conhecimentos de GIS (Sistemas de Informações Geográficas)."

Os demais resultados da avaliação podem ser visualizados no gráfico presente na figura 16. Cerca de 75% das avaliações de cada afirmação tiveram pontuação maior ou igual à 5, o que representa um ótimo resultado. Além disso, a pontuação 7 foi maioria nas avaliações de cada afirmação, quando não, empatou com as pontuações 6 ou 5.

A grande ocorrência das pontuações próximas à 7 reflete na média geral da avaliação obtida com o questionário, que foi de 6,27 (desconsiderando a primeira afirmação, dado que a mesma não

[41] https://uiuxtrend.com/pssuq-post-study-system-usability-questionnaire/

[42] https://youtu.be/SZYesxzrBIc

representa a experiência do usuário diretamente com o NodeGIS). O quão mais próxima de 7 estiver esta média geral, maior foi a satisfação dos usuários ao utilizar o NodeGIS.

5 CONCLUSÃO

O desenvolvimento de aplicações web GIS é uma demanda cada vez mais crescente. Neste trabalho foi apresentado uma ferramenta de desenvolvimento de aplicações de web GIS que, de forma simplificada, permite plotar mapas vetoriais, *overlay* e personalização de camadas, *zooming*, *panning*, *tooltip*, realizar consultas em atributos convencionais e diversas operações espaciais sobre os dados. A solução proposta é de código aberto e utiliza-se de contêineres Docker para facilitar sua implantação. Assim, também pode ser utilizada para o ensino de GIS sem a necessidade de instalação de software por parte dos alunos.

Um dos desafios encontrados foi a renderização dinâmica e gerenciamento dos dados das camadas vetoriais em frontend, o que pode resultar em uma implementação complexa quando se trata de bibliotecas JavaScript "puras". A versão para React da biblioteca Leaflet foi essencial para contornar este problema, contribuindo para uma implementação de fácil manutenção e evolução.

Foi utilizada uma abordagem sequencial de desenvolvimento das funcionalidades, com o auxílio do Trello para especificação, organização e priorização das atividades. Esta especificação e divisão das atividades foi feita de forma a ter-se funcionalidades que pudessem ser implementadas de forma individual, agregando à solução como um todo conforme fossem concluídas, fazendo com que o NodeGIS já fosse utilizável desde a conclusão da primeira atividade especificada. O processo de desenvolvimento não foi fragmentado em sprints, prática comum em processos que seguem uma metodologia ágil, porém é possível considerar cada atividade desenvolvida como uma entrega individual.

Como trabalhos futuros pretende-se integrar à ferramenta visualizações 3D, nuvem de pontos e *raster*.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por toda a força e sabedoria, mantendo-me perseverante na minha caminhada. Agradeço ao Professor Doutor Cláudio de Souza Baptista, meu mestre, orientador e amigo, por todos os ensinamentos, oportunidades e confiança ao longo de todo o nosso contato, que me enriqueceram enquanto profissional e pessoa. Agradeço ao corpo docente do curso de Ciência da Computação, bem como aos demais funcionários da Universidade Federal de Campina Grande (UFCG), por me proporcionarem toda a estrutura necessária para uma formação de qualidade.

Agradeço aos amigos que construí durante a minha jornada na UFCG, em especial aqueles do Laboratório de Sistemas de Informação, por todos os conhecimentos partilhados e apoio ao longo desses anos. Agradeço também aos meus pais e irmãs, por todo o amor, apoio, ensinamentos e por sempre acreditarem em mim. Por fim, agradeço à minha noiva, por sempre me motivar a dar o meu melhor em tudo que faço e, sobretudo, por toda a paciência diante de todos os sacrifícios necessários na minha jornada acadêmica.

REFERÊNCIAS

[1] T Edwin Chow. 2008. The Potential of Maps APIs for Internet GIS Applications. *Transactions in GIS* 12, 2 (April 2008), 179–191. https://doi.org/10.1111/j.1467-9671.2008.01094.x

[2] Aleksandar Milosavikevic. 2008. An application framework for rapid development for web based GIS : Ginis Web. In *Geospatial Services and Applications for the Internet*, John Sample, Kevin Shaw, Shengru Tu, and Mahdi Abdelguerfi (Eds.). Springer, 48–71. https://doi.org/10.1007/978-0-387-74674-6_3

[3] David Moretz. 2017. Internet GIS. In *Encyclopedia of GIS*. Springer International Publishing, 1074–1081. https://doi.org/10.1007/978-3-319-17885-1_648

[4] Edward Nash, Peter Korduan, Simon Abele, and Gobe Hobona. 2008. Design Requirements for an AJAX and Web-Service Based Generic Internet GIS Client. In *11th AGILE International Conference on Geographic Information Science*. University of Girona, Spain. https://agile-online.org/conference_paper/cds/agile_2008/pdf/82_doc.pdf

[5] A.C. Paiva, E.R. da Silva, F.L. Leite, and C. de Souza Baptista. 2004. A multiresolution approach for Internet GIS applications. In *Proceedings. 15th International Workshop on Database and Expert Systems Applications, 2004.* IEEE. https://doi.org/10.1109/dexa.2004.1333575

[6] Fang Yin and Min Feng. 2009. A WebGIS Framework for Vector Geospatial Data Sharing Based on Open Source Projects. In *Proceedings of the 2009 International Symposium on Web Information Systems and Applications (WISA'09)*. 124–127. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.9559&rep=rep1&type=pdf

[7] Danlin Yu and Jingyuan Yin. 2011. Internet GIS and System Dynamic Modeling in Urban Public Safety and Security Studies: A Conceptual Framework. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 207–216. https://doi.org/10.1007/978-3-642-20539-2_23