UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Eustáquio Moreira Santana

Desenvolvimento de um sistema de visualização de sistemas elétricos de distribuição utilizando interface *web* otimizada.

Uberlândia, Brasil

2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Eustáquio Moreira Santana

Desenvolvimento de um sistema de visualização de sistemas elétricos de distribuição utilizando interface *web* otimizada.

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Professor Alan Petrônio Pinheiro

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2022

Guilherme Eustáquio Moreira Santana

Desenvolvimento de um sistema de visualização de sistemas elétricos de distribuição utilizando interface *web* otimizada.

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Professor Alan Petrônio Pinheiro
Orientador

Jefferson Rodrigo de Souza

Marcelo Rodrigues de Sousa

Uberlândia, Brasil
2022

Dedico esse trabalho a Deus, minha família e especialmente meus pais e meu irmão por terem sempre me apoiado neste difícil desafio da minha vida.

Agradecimentos

Agradeço primeiramente à Deus por sempre me guiar e me acompanhar em momentos difíceis e me fortalecer para novos desafios. Agradeço aos meus pais e meu irmão por sempre me entenderem e acreditarem no meu potencial. Agradeço a Universidade Federal de Uberlândia, juntamente com os docentes e FACOM por abrir a minha mente e me preparar para o mercado de trabalho. Agradeço ao meu orientador Alan Petrônio por me guiar e sempre e propor desafios para meu crescimento acadêmico e qualidade do meu TCC. Agradeço a ANEEL por me conceder a oportunidade de desenvolver o Monitor IOT. Este projeto foi financiado pela CEB e ANEEL por meio da proposta de PeD nº PD-05160-1805/2018 .

"São as nossas escolhas que revelam o que realmente somos, muito mais do que as nossas qualidades." **Alvo Dumbledore**

Resumo

A *Internet* das Coisas no ramo de energia tem sido implementada pelo termo *smart grids* e atualmente tem ganhado cada vez mais notoriedade global. A rede elétrica inteligente ideal tem o papel de otimizar o fluxo elétrico, distribuí-la de maneira inteligente e monitorá-la frente a um cenário dinâmico motivado pelas energias renováveis e microgeração distribuída. Motivação esta que se traduz com a necessidade de monitorar cada vez mais elos da rede elétrica. Com isso, é essencial lidar com grande volume de dados heterogêneos advindos de sensores e medidores de energia, monitorá-los para a tomada de decisão em tempo hábil e confiável, além de abstrair inferências nos Sistemas Energéticos. Isso cria a necessidade de uma estrutura para receber estes dados e convertê-los em informação visual intuitiva que facilite a tomada de decisões. Atualmente, há novas ferramentas do mercado que auxiliam o usuário a visualizar dados da rede elétrica. Essas são abrangentes e podem ajudar na tomada de decisões, inferências, armazenamento de histórico, etc. O PowerBI, por exemplo, é capaz de exibir visualizações a partir de uma fonte de dados para o usuário. No entanto, essas ferramentas já existentes no mercado são genéricas, não incorporando algumas peculiaridades da rede elétrica. Portanto, é importante trazer soluções específicas capazes de entregar uma melhor experiência ao usuário para que assim seja possível usá-las com maior eficiência e assertividade nas decisões de operações. Assim, este trabalho propôs uma prova de conceito de um sistema *web* capaz de proporcionar ao usuário funcionalidades que podem auxiliá-lo na tomada de decisões com o monitoramento de milhões de ativos elétricos da rede de distribuição em tempo real.

Palavras-chave: visualização, monitoramento, *internet* das coisas, redes elétricas inteligentes.

Lista de ilustrações

Lista de abreviaturas e siglas

IOT	*Internet of Things*
MIT	*Massachusetts Institute of Technology*
GIS	*Geographical information system*
ANEEL	Agência Nacional de Energia Elétrica
SE	Sistemas de energia
DS	*Data Studio*
Gestal	Gestão de Energia e Utilidades
SQL	*Standard Query Language*
NoSQL	Bancos de dados não relacionais
HTTP	*Hypertext Transfer Protocol*
SSE	*Server Sent Events*
WS	*WebSocket*
API	*Application Programming Interface*
BDGD	Base de Dados Geográfica da Distribuidora
CEB	Companhia Energética de Brasília
SIG-R	Sistema de Informação Geográfica Regulatório
TS	*TypeScript*
MVC	*Model View Controller*
REST	*Representational State Transfer*
JSON	JavaScript *Object Notation*
CORS	*Cross-Origin Resource Sharing*
ORM	*Object-Relational Mapping*
URL	*Uniform Resource Locator*

DOM	*Document Object Model*
JS	Javascript
UCBT	Unidades Consumidoras de Baixa Tensão
HTML	*HyperText Markup Language*

Sumário

1 Introdução

A *Internet* das Coisas tem como um de seus objetivos o intuito de conectar qualquer objeto (ou "coisa") no mundo real a um correspondente no mundo virtual. Nisso, os dados extraídos do objeto real alimentam o modelo virtual, possibilitando a conversão de dados em informações. Embora este seja um conceito já antigo, criado por Kevin Ashton no MIT (*Massachusetts Institute of Technology*) em 1991 (ASHTON et al., 2009), somente agora nessa década que ela tem ganhado mais vigor. Enquanto a *internet* convencional recebe tradicionalmente dados produzidos por seres humanos, a *Internet* das Coisas (vista como versão amadurecida sobre a *internet*) recebe dados produzidos não somente por seres humanos, como também por coisas sensoriadas. Existem vários ramos de IOT (*Internet of Things*) que são implementados para nichos específicos. Este trabalho, por exemplo, trata, ainda que de forma marginal, o emprego de IOT para o setor elétrico.

Com o advento da teoria de IOT, várias aplicações e sistemas foram criadas com esse propósito. Uma delas é conhecida como *smart grid*, a qual tem o conceito de processo de digitalização de toda a rede elétrica que se vê na transmissão, geração e distribuição. Em especial, essas redes elétricas se distribuem por vastos espaços, podendo abranger estados inteiros. Nisso, é comum abranger contingentes superiores a 1 milhão de clientes, cerca de 30 mil unidades transformadoras, ou até 100 mil dispositivos simultaneamente, como é o caso da rede de distribuição elétrica que será usada de referência neste trabalho.

Boa parte dos dispositivos abordados não possuem comunicação em tempo real ou têm dificuldades para isso. Contudo, com a chegada da tecnologia de comunicação, especialmente a focada em IOT, é possível atualmente vincular estes dispositivos na planta elétrica, fazendo-os comunicar com um sistema de virtualização, que corresponde ao sistema de distribuição virtualizado dentro de um sistema de IOT.

Este processo de virtualização não exige somente a comunicação, mas também o tratamento dos dados para informação, padronização dos protocolos, abstração de dispositivos e visualização da informação. A visualização da informação, especialmente por se tratar de ampla escala, precisa buscar como mostrar uma informação de forma amigável juntamente com uma interface gráfica acessível (*web* e *browser*) de vários dispositivos ao mesmo tempo. Neste setor em particular, é muito comum o emprego de agentes que ficam responsáveis por monitorar telas enormes por um longo espaço de tempo e grandes quantidades de dispositivos.

Além disso, com a modernização de navegadores de *internet* como Google Chrome, por exemplo, juntamente com a evolução de linguagem de programação Ecmascript (mais conhecida como JavaScript), é possível atualmente criar interfaces mais complexas e ro-

bustas para o usuário final sem que seja preciso realizar várias instalações de programas em um computador. Em outras palavras, o usuário final terá uma dificuldade menor para acessar a aplicação, facilitando, então, o seu uso.

1.1 Objetivos

Essa pesquisa busca propor uma forma de utilizar a tecnologia de IOT e virtualização da rede elétrica (ou *smart grids*) para visualização da informação em interface *web* e usando recursos GIS (*Geographical information system*). Em outras palavras, usar a ideia de virtualização de objetos reais e extrair informações de maneira visual e centralizada em um mesmo mapa, por exemplo, ou em outras formas de representação gráfica. Nisso, fornecer aos diferentes operadores do sistema elétrico a possibilidade de ver em uma só tela as condições de estados mais básicas desses correspondentes objetos reais por meio dos virtuais em menor granularidade, uma vez que estes estão mais habituados a trabalhar somente com os grandes elementos (como subestações, grandes transformadores, grandes barramentos, circuitos, etc) através de sistemas supervisórios.

Entre os principais desafios a serem tratados neste trabalho, podem-se destacar três principais: (i) estabelecimento de um processo de comunicação entre aplicações eficiente, especialmente com a aplicação de virtualização; (ii) utilizar um *web browser* para visualizar (em tempo real) tantas informações em um mapa, mantendo sua atualização; (iii) deixar com que o usuário crie o seu próprio gráfico com as medidas que desejar com base em um banco de dados. Estes desafios serão tratados neste trabalho de conclusão de curso.

1.2 Justificativa

Atualmente, os sistemas elétricos são monitorados essencialmente em suas grandes unidades como, por exemplo, subestações de grande capacidade, grandes consumidores, etc. Ainda que pequenas unidades (como consumidores residenciais) não tenham suas informações geralmente avaliadas em tela por um operador na prática, elas são registradas apenas em sistemas especializados. Isso porque gerariam grande quantidade de pontos de monitoramento dificultando a supervisão por parte de operadores, especialmente porque os sistemas elétricos geralmente utilizam a visualização através de "diagramas unifilares". Neste caso, muitos itens sendo visualizados ao mesmo tempo tem potencial de prejudicar a avaliação de seus usuários.

Ainda, habitualmente, o elo de ligação entre a informação e o objeto monitorado é fraco ou as vinculações derivadas. Por exemplo, é possível monitorar um transformador e um consumidor. Supondo que estes fazem parte do mesmo circuito, é possível também

inferir informações sobre o cabo elétrico do circuito, ainda que este não esteja sendo monitorado diretamente. A própria vinculação entre transformador e consumidor é também fraca.

De qualquer maneira, é possível inferir informação sobre esse circuito, pois em uma ponta está ligado um transformador e na outra está ligado um consumidor (que também pode ser monitorado). Como é um simples circuito, ainda que ele não seja monitorado indiretamente, é possível inferir informação e saber sobre o seu estado. Por exemplo, se um transformador for monitorado e não são monitoradas as unidades consumidoras vinculadas, sabe-se que ao cair a energia do transformador, automaticamente todas as unidades consumidoras vinculadas podem também cair a energia, mesmo que não sejam monitoradas diretamente. Mesmo assim, portanto, ainda é possível visualizar de forma gráfica GIS este evento da rede elétrica.

Este projeto está contextualizado dentro de um projeto de pesquisa e desenvolvimento na Universidade Federal de Uberlândia firmado com a Companhia Elétrica de Brasília (CEB) através do P&D PD-05160-1805/2018 ANEEL.

2 Revisão Bibliográfica

A utilização da IOT tem crescido nos últimos anos em todos os setores como, por exemplo, o residencial com o controle e medição de consumo energético. Em cidades, para controle de tráfego e segurança. Em sistemas de energia, (o qual será focado neste trabalho) para o gerenciamento de consumo, sustentabilidade e redução de poluição e carbono, segundo (AHMAD; ZHANG, 2020). Além disso, segundo estes autores, o número de dispositivos inteligentes conectados está previsto para 75,4 bilhões em 2025. Atualmente, os dados gerados a partir de dispositivos IOT ultrapassam os de redes sociais, pelo fato de enviarem dados constantemente. Isso pode inclusive requerer uma rede robusta capaz de processar milhões de dispositivos e lidar com milhões de eventos diariamente.

A rede elétrica se beneficia de sistemas avançados de medição para redes inteligentes, contribuindo para o seu crescimento exponencial de dados e informações, segundo (CHEN; CHEN, 2021). O artigo compara os dados obtidos por sistemas de energia (SE) convencionais com os medidores de inteligentes, os quais possuem maiores quantidades, refletindo na qualidade, eficiência e confiabilidade que são levadas a outro patamar. Isto é, o sistema se torna extremamente abundante em dados. Ademais, segundo (SANCHEZ-HIDALGO; CANO, 2018), a importância de se utilizar redes inteligentes no setor elétrico ajudaria lidar com uma grande demanda energética necessária para suprir o mundo no futuro. Isso significa que em 2050 está previsto para a sua geração ser triplicada, requerendo cerca de 40.000 TWh. O artigo também defende que aumentar a capacidade da rede não é o bastante e é preciso que haja monitoramento e análise como complemento. Contudo, implementar um SE não é uma tarefa simples e requer necessidades a serem implementadas para o seu melhor funcionamento.

Em um SE é preciso ter em mente que há algumas necessidades básicas para o seu melhor funcionamento. Isso ajuda a mantê-lo mais eficiente, confiável e robusto para as companhias de energia. Segundo (AHMAD; ZHANG, 2020), o enfoque em controle, monitoramento em tempo real, comunicação, gerenciamento de rede inteligente e automação são pilares importantes para um sistema de energia inteligente e os desafios tecnológicos envolvidos estão na complexidade do sistema e sua interoperabilidade, tolerância e evidenciação de falha em tempo hábil, volume de informação e estabilidade. Neste trabalho o enfoque será em como monitorar em tempo real de maneira estável e agradável para o usuário final um grande volume de dados relacionados a SE. Ainda, segundo (SANCHEZ-HIDALGO; CANO, 2018), uma rede inteligente necessita de ferramentas e sistemas de visualização e painéis de desempenho que atuem em tempo real, além de outras tecnologias que não serão abordadas de maneira aprofundada neste trabalho como medidores, comunicação sem fio, etc. A exibição e visualização de informações úteis para os usuários

de um SE é essencial para o gerenciamento e facilidade de operações no sistema elétrico (CHEN; CHEN, 2021). Apesar disso, este artigo defende que a teoria atual de visualização de dados é retrógrada quando se põe as necessidades de SE do mundo real. Apesar de existirem ferramentas que facilitem, como Power BI, Tableau, Data Studio e outras ferramentas baseadas em *web*, segundo o artigo, são todas para um propósito genérico. Ou seja, aplicações específicas precisam de um maior tratamento em relação a cenários de usuários, aplicação e a possibilidade de diversos métodos de visualização personalizadas, não sendo possível haver uma análise ampla ou resumo da visualização de dados. Adicionalmente, segundo (AHMAD; ZHANG, 2020), no setor de energia há um desafio relacionado ao volume de dados. Isso quer dizer que grandes consumidores obtêm trilhões de dados de diversos dispositivos, sistemas ou sensores, sendo difícil lidar tecnicamente com a velocidade de obtenção desses dados, variedade (dados não estruturados e heterogêneos), confiabilidade e integridade, afetando a proposta de se criar um monitoramento em tempo real adequado para o usuário final.

2.1 Ferramentas de visualização

2.1.1 Power BI

Conforme citado no parágrafo anterior, há ferramentas que ajudam a monitorar redes elétricas de maneira genérica como o Power BI Desktop. Essa permite a junção de várias fontes de dados diferentes como, por exemplo, SQL (*Standard Query Language*) e JSON (JavaScript *Object Notation*). Contudo, apesar deste fato, para grandes quantidades de dados serem salvos no cache a fim de ganhar performance no Power BI ao invés de acessar a fonte de dados diretamente, é preciso que obtenha outros planos pagos.

2.1.2 Data Studio

Data Studio (DS) ou Looker Studio é uma ferramenta gratuita de visualização de dados criado pelo Google. Nela, é possível criar relatórios obtendo dados a partir de bancos de dados como PostgreSQL, MySQL, etc. O compartilhamento de um relatório pode ser feito por um *link, e-mail* ou exposto publicamente a *internet* sem nenhuma cobrança. Outro ponto interessante é a possibilidade de se trabalhar em um relatório com vários usuários ao mesmo tempo.

A taxa de atualização de dados depende da conexão entre o DS e o banco de dados. Em outras palavras, há fatores limitantes como a latência de conexão, por exemplo. Há a possibilidade de se conectar ao *Big Query*, banco de dados da Google de baixo custo indicado para armazenar grandes volumes de dados e focado em análise de dados (GOOGLE, 2022).

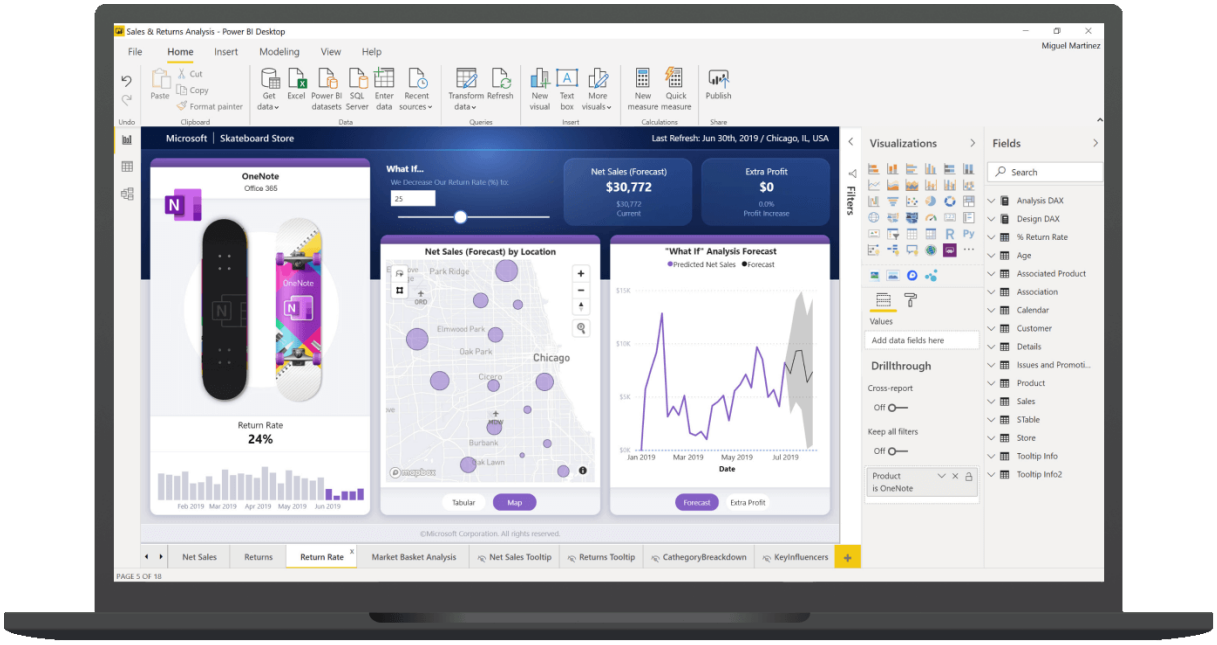

Figura 1 – Ferramenta *Power BI*

PowerBI (2022)

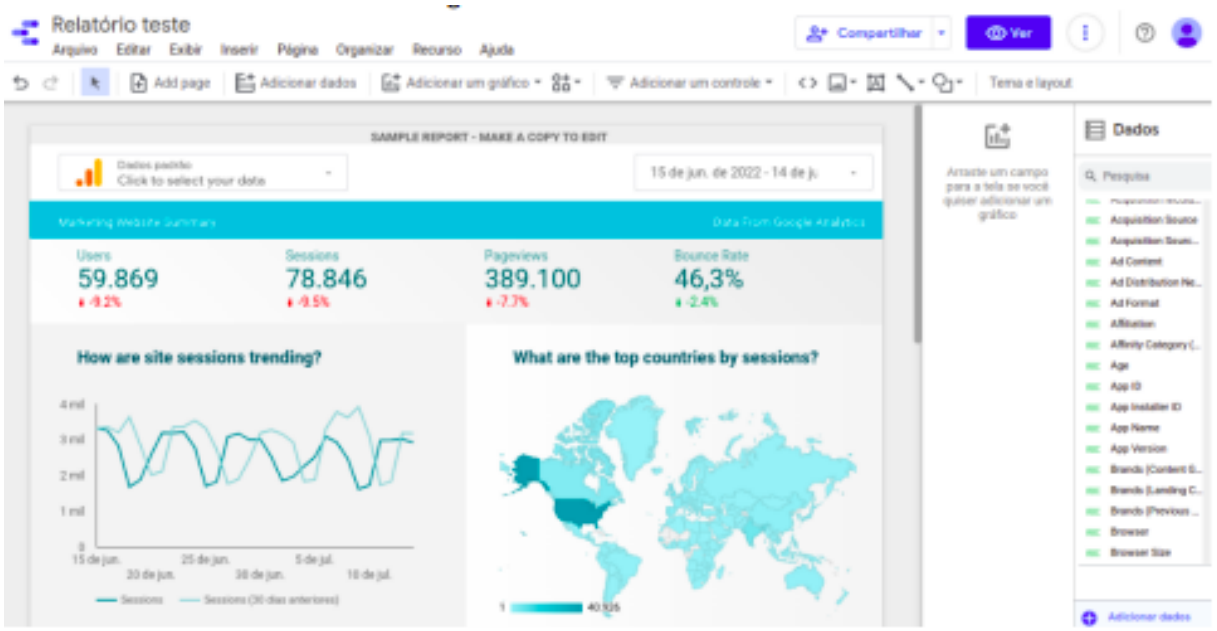

Figura 2 – *Data Studio*

2.1.3 Tableau

Tableau Desktop é uma ferramenta de visualização de dados similar ao Power BI e Data Studio da empresa Sales Force. Com ela, é possível obter diferente fontes de dados como SQL, planilhas, etc. Há a versão gratuita chamada de Tableau Public (TABLEAU, 2022b), sendo possível importar somente arquivos como CSV, JSON, PDF. Caso seja

necessário obter outras fontes de dados como MySQL, é preciso obter a licença paga. Além disso, nessa versão os dados não são armazenados em cache, havendo a necessidade, portanto, de sempre realizar a conexão para obtenção dos dados.

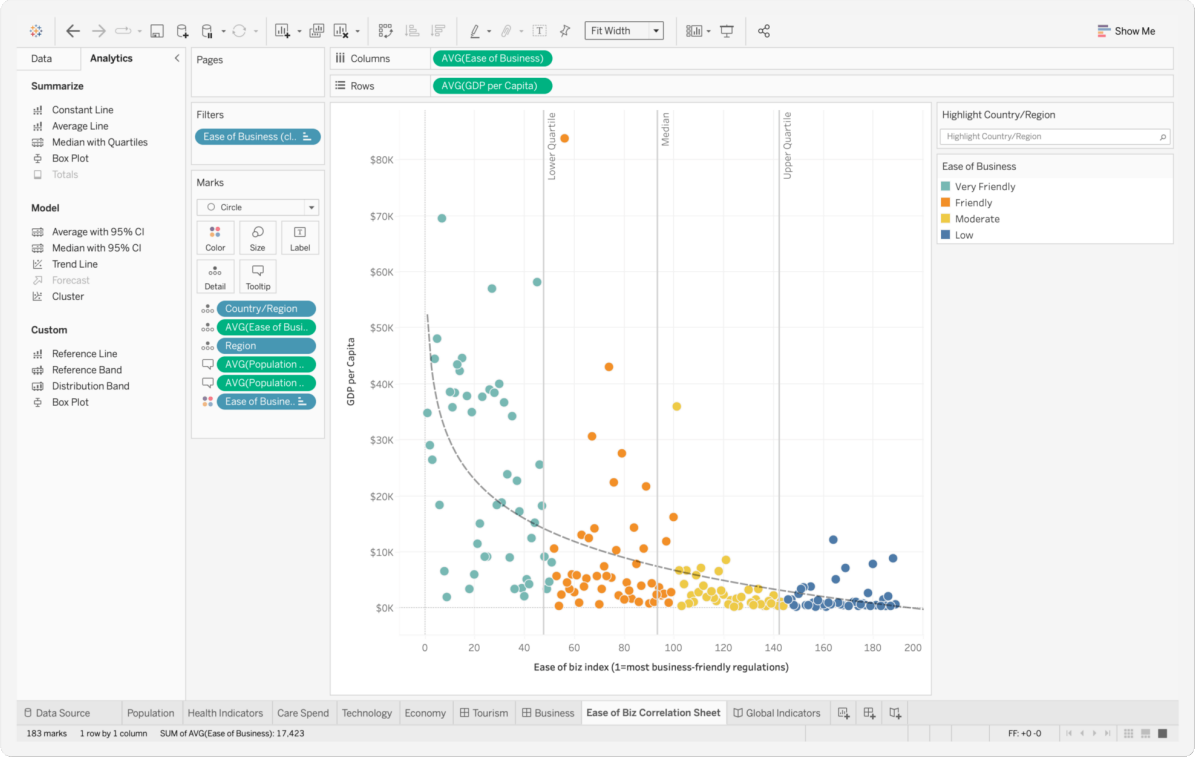

Figura 3 – *Tableau Desktop*

Tableau (2022a)

2.1.4 Gestal

A Gestal (Gestão de Energia e Utilidades) é uma empresa nacional fundada em 1997 e é responsável por fornecer soluções inteligentes no campo de gerenciamento e controle de energia elétrica (GESTAL, 2022a). As suas soluções estão categorizadas em concentrador de dados, controladores, *softwares* e acessórios. Um dos produtos, que é um concentrador de dados identificado por Smart Gate M (Figura 4), é um gerenciador de energia elétrica, utilidades e processo, podendo ser utilizado como concentrador de medições setoriais de maneira sincronizada com a medição da concessionária (GESTAL, 2022b). Esse equipamento é indicado para quem necessita apenas de uma integração com algum sistema ou *hardware* para fins de monitoramento como, por exemplo, rateio da conta de energia elétrica entre os diversos setores de uma instalação. Outro produto, identificado por Smart Gate X (Figura 5) que está na categoria de controlador, tem o papel de controlar medidores, sensores e atuadores de outros fabricantes, por exemplo (GESTAL, 2022c).

Figura 4 – Smart Gate M

GESTAL (2022b)

Figura 5 – Smart Gate X

GESTAL (2022c)

2.2 Tecnologias

Para se criar uma rede inteligente é interessante que seja separado cada responsabilidade em um grupo de tecnologias com o intuito de criar uma arquitetura robusta e fácil de realizar possíveis novas implementações e manutenções. Com isso, é apresentado no artigo (DAKI et al., 2017) que há fases dentro do *Big Data* que delimitam cada papel.

1. A fase de fonte de dados, a qual é relacionada a dispositivos físicos como medidores, sensores, etc.

2. A fase de integração, a qual se comunica diretamente com a anterior, é responsável pela transferência de dados de maneira rápida e confiável.

3. A fase de armazenamento é responsável por salvar em algum banco de dados os dados recebidos.

4. A fase analítica é responsável por explorar, analisar e transformar os dados com o intuito de revelar tendências ou auxiliar em tomada de decisões de alguma companhia, por exemplo.

5. A fase de visualização, a qual será o enfoque deste trabalho, apresenta informação de forma gráfica. Na figura 6 cada fase é representada.

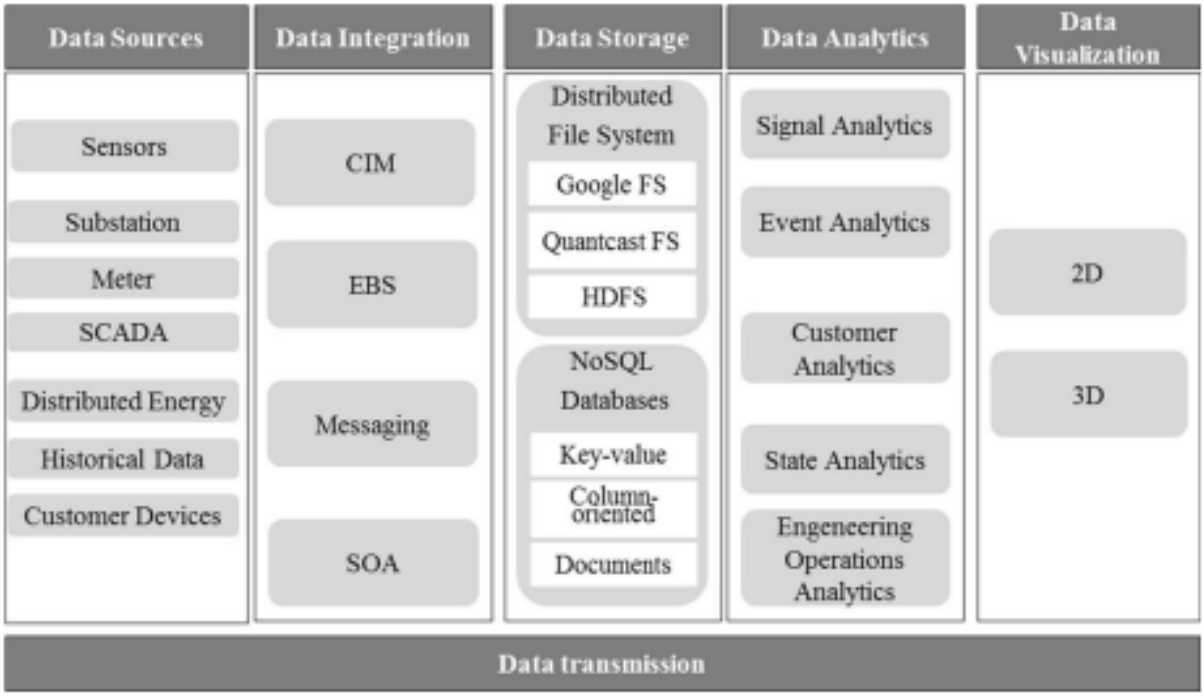

Figura 6 – Fases de *Big Data*
(DAKI et al., 2017)

A visualização é uma conversão de dados numéricos para informação visual, concedendo ao usuário a capacidade de analisar, criticar ou trazer um sentido a informação (SANCHEZ-HIDALGO; CANO, 2018). Segundo o artigo, isso demonstra que ela não é somente um método computacional, e sim um papel importante na área científica, sendo preciso levar em consideração a qualidade da experiência do usuário em que o mesmo deve ser capaz de reagir, interpretar e analisar as informações de maneira rápida e fácil. Para alcançar tal objetivo, é preciso ter atenção em novas tecnologias que possam auxiliar nesse processo.

O grande volume de dados, segundo o artigo (BHATTARAI et al., 2019), é um dos principais desafios enfrentados para a análise de *Big Data*, sendo que a quantidade

de dados gerados pelas concessionárias crescem de maneira exponencial. Por conta deste fato, é importante que seja utilizado tecnologias de armazenamento que possam sanar ou ajudar a reduzir essa pendência. Outro fator abordado por este artigo seria que em decorrência do aumento de dispositivos inteligentes nas concessionárias, o volume de dados gerados por elas nessa categoria seria imenso, havendo a necessidade de encontrar alguma tecnologia ou ferramenta capaz de enviar um grande conjunto de dados em vários sistemas de monitoramento, visualização e análise de maneira rápida sem onerar a performance de toda a rede inteligente. A interface de um sistema *web* também é importante para que o usuário possa monitorar ou visualizar uma grande quantidade de dados de maneira responsiva e confiável. Com o auxílio de bibliotecas e *frameworks* robustas da comunidade, é facilitado o processo de renderizar uma grande quantidade de objetos que representam os dados de maneira performática e agradável ao usuário, por exemplo.

2.2.1 Integração

Para obter os dados gerados por sensores e medidores e enviá-los de maneira rápida e segura, é interessante que existam sistemas de processamento de fluxo de dados que atuam de maneira assíncrona e que sejam capazes de suportar um grande volume de dados. Temos o exemplo do Apache Kafka, plataforma de código aberto com arquitetura de publicação e assinatura, que, segundo o site oficial (KAFKA, 2022), tem a proposta de perda zero de mensagens e grande processamento. Ele também conta com vasta comunidade de usuários e compatibilidade alta com banco de dados conhecidos do mercado (como Postgres, JMS, MongoDB, etc). Além disso, é escalável e com alta disponibilidade quando instalado em clusters, sendo utilizado por sistemas críticos que necessitam de confiabilidade. O Kafka também tem a capacidade de atender com milhões de ponto de dados por segundo (REDHAT, 2022). Dentro de um contexto de *smart grids* utilizando o Kafka, os sensores, medidores e dispositivos são os produtores, os quais seriam responsáveis por gerar as mensagens. Essas mensagens seriam transportadas para o *broker* e partir do mesmo para o tópico indicado. A partir disso, os consumidores, os quais seriam os sistemas de monitoramento, por exemplo, iriam consumir essas mensagens.

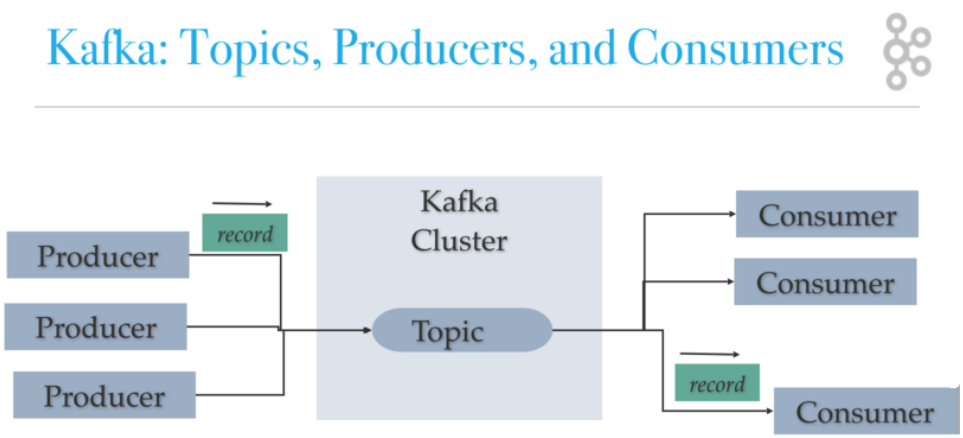

Figura 7 – Arquitetura do Kafka

Azar (2022)

2.2.2 Armazenamento

Conforme citado anteriormente, os dados gerados por sensores, medidores ou sistemas em um SE são heterogêneos e abundantes. Segundo o artigo (SAGIROGLU et al., 2016), os dados de medidores inteligentes, gerenciamentos de ativos são classificados como *big data* pelas concessionárias de energia. Portanto, segundo esses mesmos autores, *big data* é o grupo de dados não manuseados e analisados por ferramentas tradicionais em um intervalo de tempo admissível. Esse conjunto de dados podem ser vídeos, sons, imagens, textos e devem servir para auxiliar em tomadas de decisões. Com isso em mente, novas tecnologias foram criadas com a premissa de enfrentar o desafio de armazenamento de *big data* como NoSQL (bancos de dados não relacionais), sendo projetados para serem adequados às ferramentas analíticas e processamento paralelo. Além disso, bancos de dados não relacionais tem desempenho superior quando é envolvido *big data* em relação aos bancos de dados relacionais, segundo o artigo (ANSARI; VAKILI; BAHRAK, 2019). O MongoDB é um exemplo de banco de dados NoSQL de código aberto baseado em documentos do tipo JSON flexíveis, sendo projetado para ser distribuído em seu núcleo. O mesmo tem suporte de diferentes métodos de replicação *master-slave* com o intuito de realizar várias cópias em servidores, deixando mais simples e rápida a integração com outros sistemas, segundo o mesmo artigo.

2.2.3 Visualização

Para renderizar uma grande quantidade de elementos para um usuário em uma interface *web*, é necessário que seja utilizado técnicas de programação, focando na experiência do usuário. O EcmaScript é o nome oficial da linguagem de programação para navegadores e é conhecida também por JavaScript. Essa linguagem é de *scripting* dinâ-

mica e possui suporte a diversos navegadores de *internet* populares como Chrome, Firefox e Edge. Além disso, a linguagem é multi-paradigma e baseada em protótipos (MOZILLA, 2022a), havendo também a característica de se realizar operações de maneira assíncrona. É possível utilizá-la não somente em navegadores e sim no lado de servidores, utilizando a ferramenta node.js. É interessante utilizar maneiras de não sobrecarregar a *thread* principal, a qual é responsável por renderização dos elementos da interface gráfica e eventos do usuário, evitando, assim, operações de longa duração, sendo uma das estratégias utilizar operações assíncronas (MOZILLA, 2022b). Outra solução seria utilizar *web worker* em que é possível executar comandos em segundo plano com o intuito de manter a *thread* principal sem travamentos para o usuário. Porém há a limitação de não ser possível modificar elementos da interface gráfica e eventos do usuário. Ou seja, caso seja necessário modificar alguma cor da interface gráfica dinamicamente, isso não é possível.

Para haver a comunicação entre o servidor e cliente com a intenção de se obter os dados para o usuário final, há alternativas já conhecidas que podem ser implementadas. Um exemplo disso é realizando uma requisição assíncrona HTTP (*Hypertext Transfer Protocol*) em uma API (*Application Programming Interface*) de tempos em tempos, sendo utilizada, por exemplo, para se obter dados a fim de se plotar em algum gráfico, gerando a impressão de que está sendo atualizado sem a necessidade da ação do usuário. A ação em questão seria o cliente solicitando uma requisição ao servidor e o mesmo sendo um agente passivo.

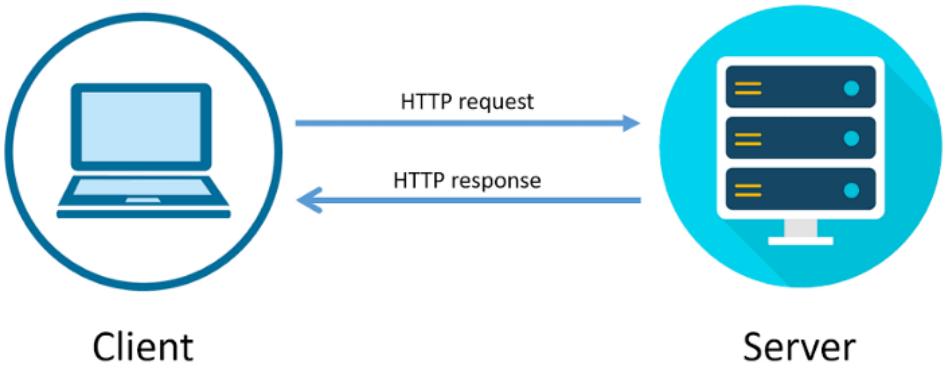

Figura 8 – Requisição HTTP

Wan (2022)

Outra tecnologia HTTP a ser utilizada seria o SSE (*Server Sent Events*), sendo caracterizada por somente o servidor enviar eventos ao cliente e essa comunicação ser unidirecional. Segundo (MOZILLA, 2022c), há a limitação de ser utilizado apenas 6 conexões por domínio entre as abas em navegadores como Chrome, Firefox quando não é ativado o HTTP/2.

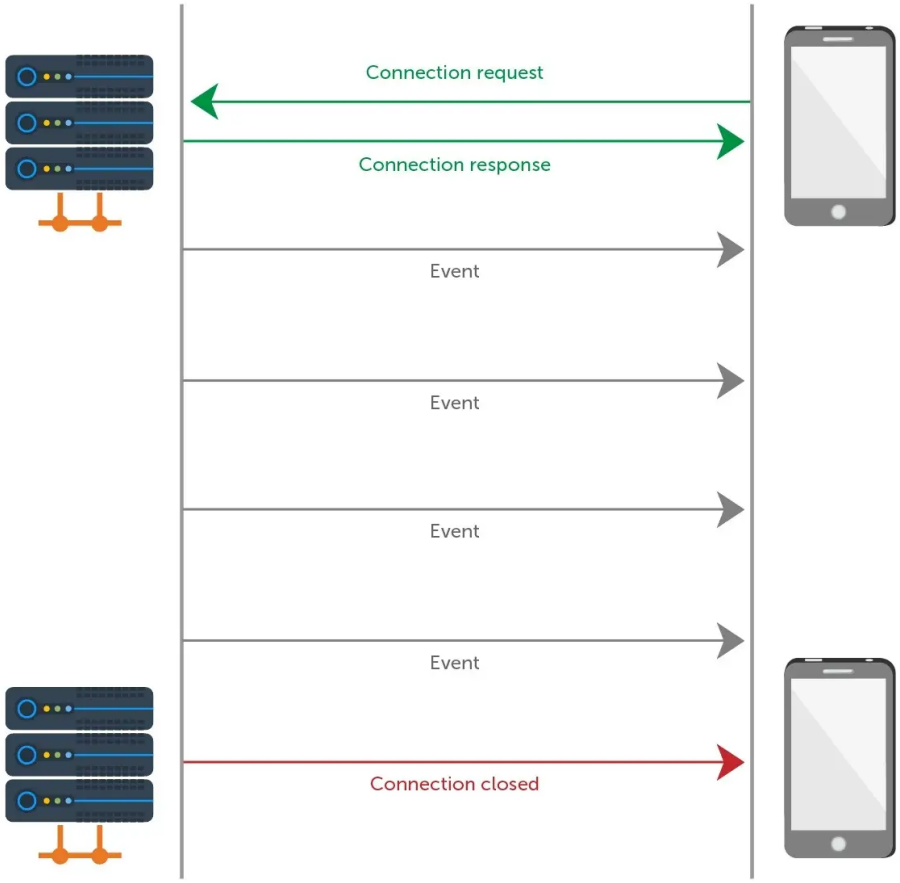

Figura 9 – *Server Sent Event*

Ably (2022)

Tem-se o WS (*websocket*), tecnologia que utiliza HTTP e tem o propósito de realizar comunicação bidirecional, ou seja, tanto o servidor quanto o cliente conseguem se comunicar entre si (LARAVEL-NEWS, 2022). Basicamente os passos até a comunicação bidirecional são: o cliente realiza um *handshake* com o servidor e o mesmo finaliza enviando de volta para o emissor o *handshake* de upgrade no cabeçalho HTTP. Assim, o protocolo deixa de ser HTTP e se torna WS. Diferentemente do HTTP, a conexão entre o servidor e cliente de WS são persistentes sem a sobrecarga no cabeçalho das mensagens, reduzindo, assim, a largura de banda entre ambos.

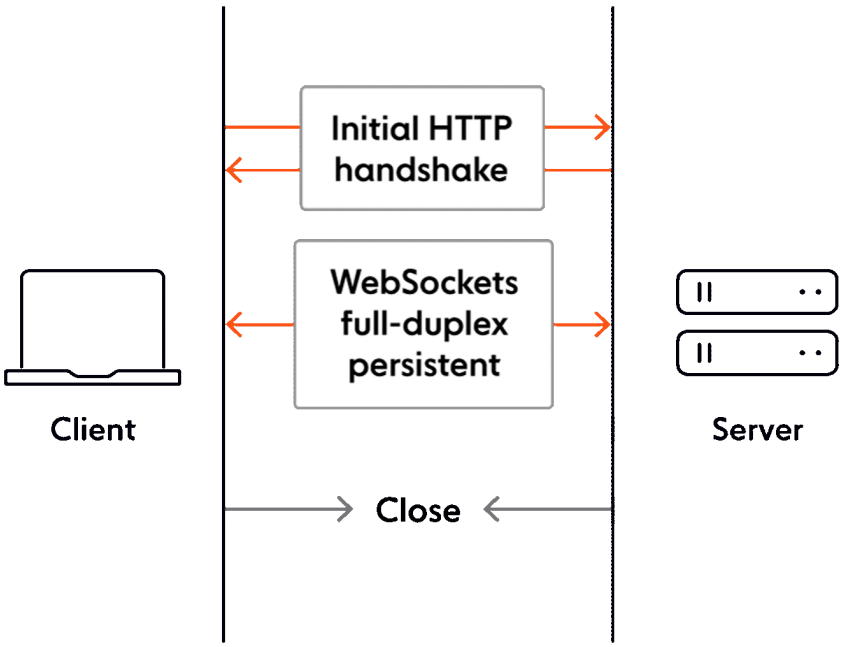

Figura 10 – *Comunicação WebSocket*

Laravel-News (2022)

Quando se é comparado WS com HTTP com *polling* (processo de se conectar ao servidor dentro de um intervalo de tempo), percebe-se que o WS se sai melhor que o HTTP com *polling*. Segundo o artigo (SOEWITO et al., 2019), foi comparado o tráfego entre WS e HTTP, construindo-se um servidor web que obtinha os dados por meio de um sensor e o cliente, dispositivo *smartphone* ou *laptop*, que os adquire a partir desse servidor. Foi concluído que o uso de memória pelo lado do cliente e largura de banda do WS são inferiores ao HTTP com *polling*.

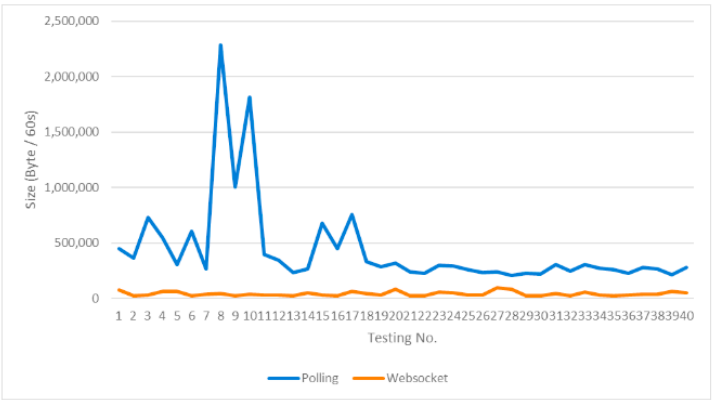

Figura 11 – Comparação largura de banda entre WS e HTTP *Polling*

Soewito et al. (2019)

Outra tecnologia que contribui especificamente na parte gráfica, seja para plotar gráficos, elementos em mapa, objetos tridimensionais, de uma interface web é o WebGL. Essa API de gráficos de baixo nível é compatível entre plataformas, livre de *royalties* e é baseado em OpenGL (KHRONOS, 2022), sendo utilizada para Ecma Script e HTML5 em navegadores como Chrome e Firefox. O segredo basicamente está em utilizar a GPU do dispositivo cliente para plotar ou renderizar um grande volume de dados ou objetos complexos de maneira rápida.

2.3 Resumo

Foi apresentado neste capítulo ferramentas conhecidas no mercado que podem auxiliar no monitoramento e visualização de dados, sendo visível notar que apesar de existirem, é necessário focar em tecnologias específicas feitas em sua concepção para lidar em processar e renderizar grandes volumes de dados com melhor eficiência e experiência para o monitoramento de uma rede elétrica de distribuição.

As tecnologias apresentadas podem auxiliar em lidar com o armazenamento de grandes volumes de dados de maneira rápida e facilitada, juntamente com as tecnologias que sejam capazes de processar e encaminhar milhares mensagens para outros sistemas de maneira rápida e, por fim, apresentá-las em uma interface *web* de maneira rápida e em tempo real.

3 Metodologia e Desenvolvimento

3.1 Apresentação do MonitorIOT

O projeto proposto, designado doravante de 'MonitorIOT', foi desenvolvido com o intuito de ser uma ferramenta para o monitoramento de uma grande quantidade de dados em tempo real de ativos de uma rede elétrica de distribuição. Dentre estes ativos, os mais comuns são: transformadores, unidades consumidoras (baixa, média e alta tensão), circuitos, religadores, reguladores, etc. Tudo em um mapa em padrão GIS personalizado. Estes ativos são obtidos a partir do BDGD (Base de Dados Geográfica da Distribuidora).

O propósito da ferramenta, portanto, é auxiliar os operadores responsáveis pelo monitoramento da rede elétrica de distribuição através de uma interface gráfica mais amigável e escalável que os clássicos diagramas unifilares usados no setor elétrico. O uso do BDGD no projeto teve como objetivo seguir regras e convenções propostas pela ANEEL (Agência Nacional de Energia Elétrica), a qual fornece um documento (ANEEL, 2021) que tem o objetivo de conter um conjunto ínfimo de informações da distribuição, incluindo o SIG-R (Sistema de Informação Geográfica Regulatório).

Além disso, propõe os requisitos mínimos para validação e condução do BDGD, mostra o conteúdo das informações, vencimento e forma de envio à ANEEL e, por fim, demonstra a forma de utilização, juntamente com a publicação da informação relacionadas ao SIG-R. O banco de dados utilizado pelos sistemas que o MonitorIOT consome seguem as especificações do documento na estrutura de dados.

Conforme citado no parágrafo anterior, o sistema desenvolvido obtém seus dados do BDGD. Mas estes são de natureza cadastrais. O sistema também pode receber dados em tempo real de medidores contendo medidas típicas do setor elétrico (corrente, tensão, etc) destes mesmos ativos. Isso porque essa ferramenta foi desenvolvida dentro de um contexto de projeto maior onde neste cenário foi também desenvolvido *hardware* de medição com envio de dados em tempo real. Essa comunicação foi feita por uma "Common API". Essa API *web* permite a conexão à coleções dentro do banco de dados NoSQL MongoDB. Outro sistema, também ligado ao MonitorIOT, é o VPlant, responsável por obter os dados diretamente dos medidores por meio do Apache Kafka ou Node Status. Foi desenvolvido um *back-end* dedicado ao MonitorIOT para obter dados do BDGD por meio de uma API e, com isso, criar gráficos personalizados.

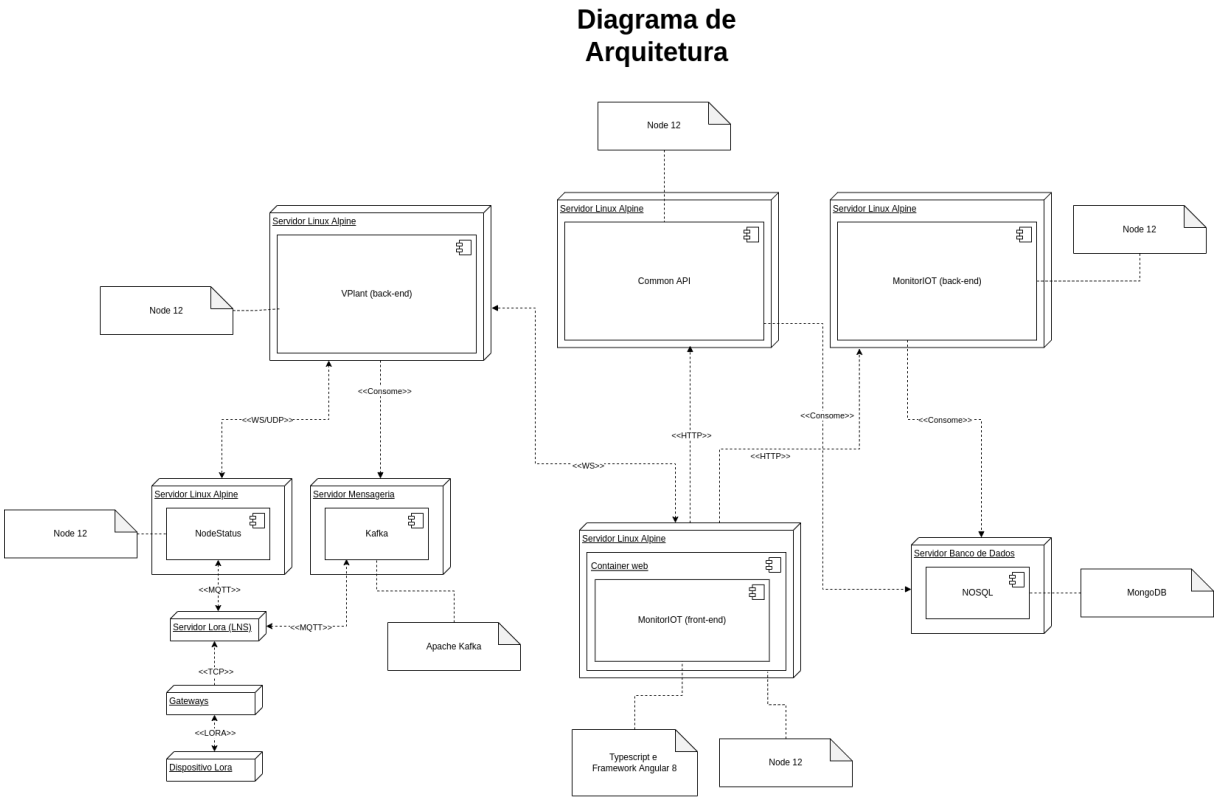

Figura 12 – Visão geral arquitetural

3.2 Tecnologias utilizadas

3.2.1 Interface gráfica

O MonitorIOT é uma aplicação *web* e por isso tem uma arquitetura de *front-end* e *back-end*. A seguir será apresentado quais foram as tecnologias utilizadas para construir a interface gráfica.

- O *front-end* foi desenvolvido na linguagem de programação TS (*TypeScript*) e utiliza o *framework* Angular, mantido atualmente por um time da Google, na versão 8.3. A ideia de sua escolha seria pelo fato de seguir o padrão MVC (*Model View Controller*), algo que é amplamente conhecido, facilitando, por exemplo, na manutenção do código ou alguma adição de funcionalidade.

- Está sendo utilizado o *framework* Bootstrap na versão 4.5.2, ferramenta de código aberto desenvolvido por um *designer* e desenvolvedor na empresa Twitter, com o intuito de facilitar na estilização da ferramenta, juntamente com um *layout* atraente.

- A biblioteca JQuery foi utilizada pelo fato de ser requisito do Bootsrap para funcionar com todos os seus recursos e o pacote JQuery-UI, que estiliza o calendário no momento que é criado um gráfico personalizado.

- Para a utilização do mapa, foi utilizado a extensão da biblioteca Leaflet, chamada Leaflet Glify (FERREIRA, 2020). A essa biblioteca foram feitas alterações a sua forma de renderização, algo que será explicado mais adiante. Para conseguir expandir o mapa na tela inteira, foi necessário utilizar uma extensão da biblioteca Leaflet chamada Leaflet.fullscreen.

- Com o intuito de deixar a navegação do usuário dinâmica e personalizável foi utilizado a biblioteca AngularGridster2 (TIBERIUZULD, 2022), que possibilita ajustar o contexto do MonitorIOT. Por exemplo, ela possibilita criar um elemento dinamicamente qualquer e personalizar seu tamanho tanto na vertical quanto horizontal. Isso incluí movê-los livremente dentro de um espaço da interface gráfica.

- Para a plotagem de gráficos foi utilizada a biblioteca de código aberto Apache ECharts, desenvolvido pelo Apache. Seu objetivo foi de gerar gráficos de maneira facilitada.

- A biblioteca de código aberto LocalForage auxiliou no armazenamento em cache de grandes quantidades de dados.

- Com a intenção de manter algumas funcionalidades assíncronas serem mais simplificadas de se implementar, foi utilizado a biblioteca de código aberto chamada RxJS.

- Por fim, é utilizado o Docker Engine, ferramenta de código aberto, para basicamente virtualizar um sistema operacional de maneira leve, fazendo com que o MonitorIOT seja executado em qualquer servidor ou dispositivo que tenha suporte de maneira facilitada. O sistema operacional utilizado na imagem base do MonitorIOT é o Linux Alpine.

3.2.2 Back-end

A seguir será apresentado quais foram as tecnologias que integraram a construção do *back-end* do MonitorIOT.

- Foi utilizado o NodeJS, *software* de código aberto, permitindo que seja utilizado Javascript.

- A *framework web* de código aberto express é utilizada na versão 4.17.1 para a criação robusta facilitada de API REST (*Representational State Transfer*).

- Com o intuito de manter o código com tipagem nas variáveis e retornos de métodos, foi incluído o uso do TS na versão 4.1.3.
- Tendo em vista que a comunicação entre a interface gráfica do MonitorIOT e o *back-end* utilizam JSON, foi utilizado a biblioteca de código aberto body-parser desenvolvido pelo expressjs.
- Para manter a API acessível a interface gráfica sem o problema de CORS (*Cross-Origin Resource Sharing*), isto é, prevenir com que ela esteja impedida de acessar o *back-end* pelo navegador de *internet*, foi utilizado a biblioteca de código aberto CORS na versão 2.8.5, desenvolvida também por expressjs.
- Foi utilizado um conjunto de bibliotecas de código aberto para lidar com consultas, conexão ao banco e facilitação de ORM (*Object-Relational Mapping*) no MongoDB, sendo a biblioteca de código aberto Mongoose na versão 5.11.15 e Mongoose Paginate para trazer o resultado de uma consulta de maneira paginada.
- Por fim, com o mesmo intuito citado anteriormente sobre os benefícios da virtualização, foi utilizado o Docker também para o *back-end*.

3.3 Estrutura

Após apresentar as tecnologias utilizadas no MonitorIOT, será apresentado como foi organizado o sistema na interface gráfica e *back-end*, havendo a descrição de suas estruturas e concepções, bem como o detalhamento de algum fluxo de algoritmo. A interface gráfica teve um grau de complexidade maior neste trabalho se comparado ao *back-end*, uma vez que houve a necessidade de lidar com um grande número de elementos ao mesmo tempo, preocupando-se sempre em não travar a interface gráfica principal com o intuito de não afetar a experiência do usuário.

3.3.1 Estrutura básica da interface gráfica

A arquitetura proposta do Angular consiste em se aproximar do conhecido padrão MVC, isto é, ele é dividido em módulo, *template*, componente e injetor. Sendo assim, tem-se as seguintes divisões:

- O módulo consiste em englobar o *template* e componente, sendo responsável por encapsular e agrupar cada responsabilidade do sistema.
- O *template* basicamente é responsável pelo código HTML.

- O componente tem o papel de conter em código a regra de negócio do sistema. A comunicação entre o *template* e componente ocorrem por meio da ligação de eventos e ligação de propriedade.
- Os injetores são responsáveis por criar um serviço, o qual tem o papel de realizar a comunicação entre componentes, por exemplo.
- Dentro do módulo *app*, padrão já fornecido pelo *framework*, possui os módulos *dashboard* e *theme*. O primeiro contém diversos componentes que compõem a página como a barra lateral, a barra superior, mapa, etc. O segundo é responsável por alterar o tema do *layout* da interface dinamicamente tanto para uma cor mais escura quanto para uma cor mais clara.
- Na camada de injetores tem-se os serviços que enviam mensagens entre componentes, juntamente com o serviço para realizar uma requisição HTTP para outras aplicações.

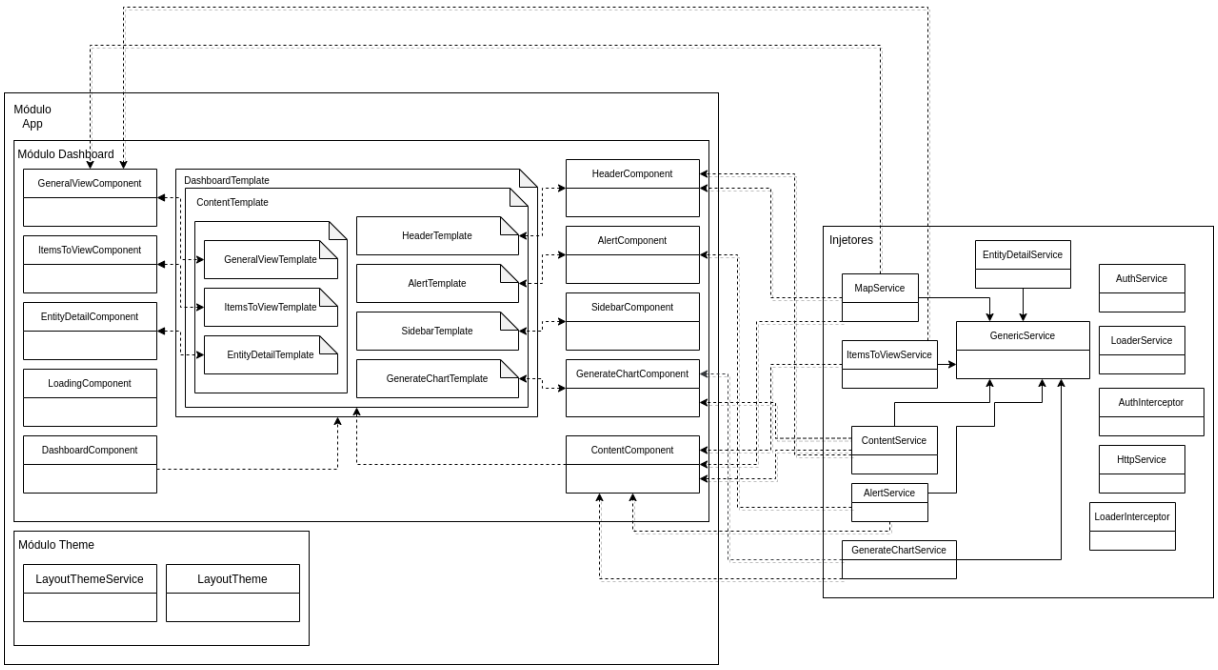

Figura 13 – Arquitetura geral interface gráfica

3.3.2 Estrutura básica do back-end

A arquitetura proposta foi criar camadas dentro do código e definir diferentes responsabilidades para cada uma. Portanto, as camadas foram definidas da seguinte maneira:

- A camada de rota é responsável por expor toda a API e lidar com o mapeamento de URL (*Uniform Resource Locator*) na aplicação para, por exemplo, o *front-end* ter o acesso em serviços de obtenção de dispositivos virtuais por meio do nome, id e alguma chave no formato de texto ou a obtenção de dados que contribuem na construção de um gráfico personalizado.
- A camada controladora é utilizada pela rota e tem o papel no projeto de executar algum tratamento na requisição antes de ser enviado para a camada de serviço.
- A camada de serviço tem o objetivo de executar a regra de negócio e invocar a camada de repositório para buscar informações no banco de dados.
- A camada repositório tem o objetivo unicamente de realizar consultas diretamente no banco de dados.
- A camada domínio tem a responsabilidade de mapear o documento representado no banco de dados em uma classe e é utilizado pela camada de repositório no momento em que é acionado o banco de dados.

- A camada esquema é responsável por mapear e retornar somente atributos necessários a partir de um objeto do resultado de uma consulta quando a camada de serviço é acionada.

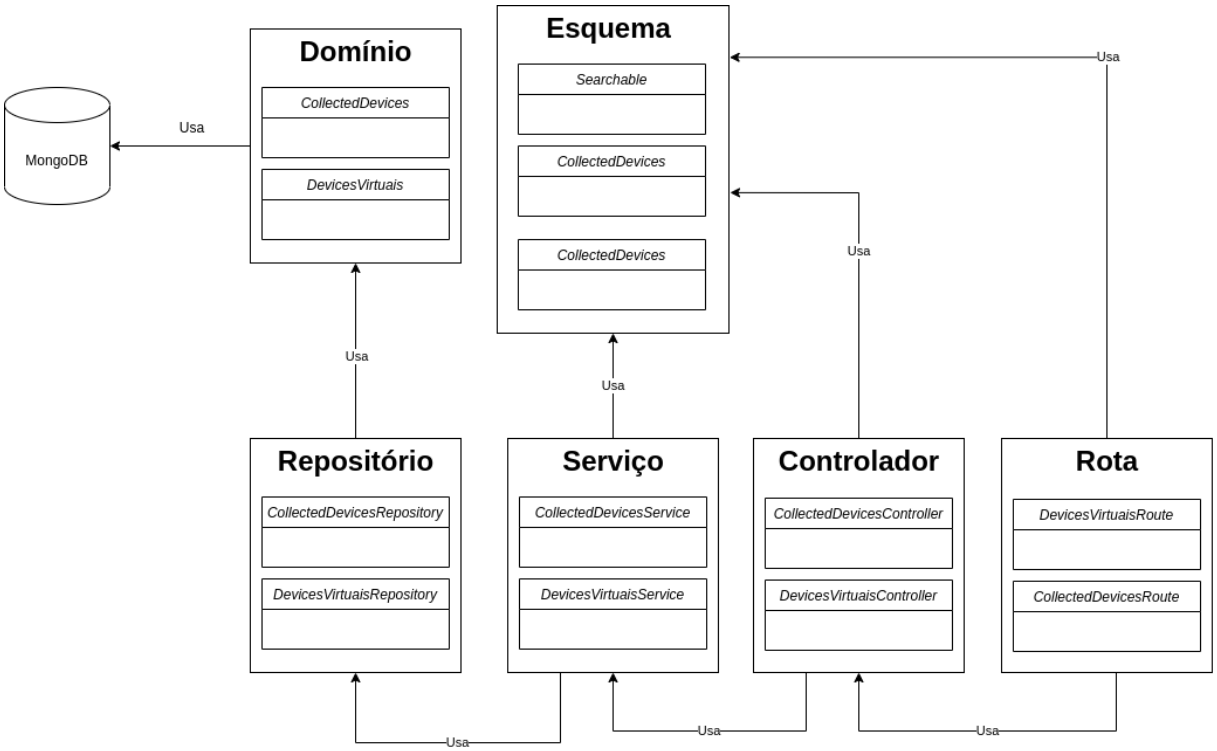

Figura 14 – Arquitetura geral back-end

3.4 Funcionalidades

As duas funcionalidades principais implementadas no MonitorIOT serão descritas com mais detalhe logo adiante. Brevemente, as funcionalidades principais são:

- Plotar elementos no mapa, sendo direcionado para Brasília, local onde estão mapeados os medidores pela CEB e com a função de plotar no mapa vários tipos de transformadores e circuitos.
- Criar gráficos personalizados a partir de uma árvore de diretórios em que contém medidas o qual o usuário pode definir e montar da maneira que desejar.

Foi simulado a transmissão de dados dos medidores para os elementos monitoráveis como UCBT (Unidades Consumidoras de Baixa Tensão) e transformadores de distribuição. A simulação é feita por meio do CommonAPI que gera dados com estados aleatórios ou requeridos através de um *endpoint*. Assim, o VPlant é solicitado e realiza a passagem de dados via conexão WS para o *front-end* quando este realiza o monitoramento no mapa.

Com isso, é refletido na renderização do objeto no mapa e alterado a sua cor de acordo com o estado solicitado. O motivo disso é pelo fato do projeto da CEB possuir apenas dois medidores capazes de transmissão, não demonstrando a real capacidade do MonitorIOT que seria renderizar uma alta quantidade de elementos no mapa preservando a experiência do usuário.

3.4.1 Comunicação entre componentes

Várias partes da interface gráfica utilizam o padrão de projeto *Observer* para se comunicar entre os componentes. Os modais utilizam este padrão para serem acionados quando são solicitados, por exemplo. O sujeito, que nesse caso seria o evento de clicar em um botão, notificaria para o modal (observador) desejado que ele deve ser acionado. O modal, por sua vez, se inscreve no momento do início da instância do componente para ficar escutando caso receba alguma notificação. Como pode se perceber, o modal é um componente que pode ser invocado por outro, no momento que um evento é ocorrido.

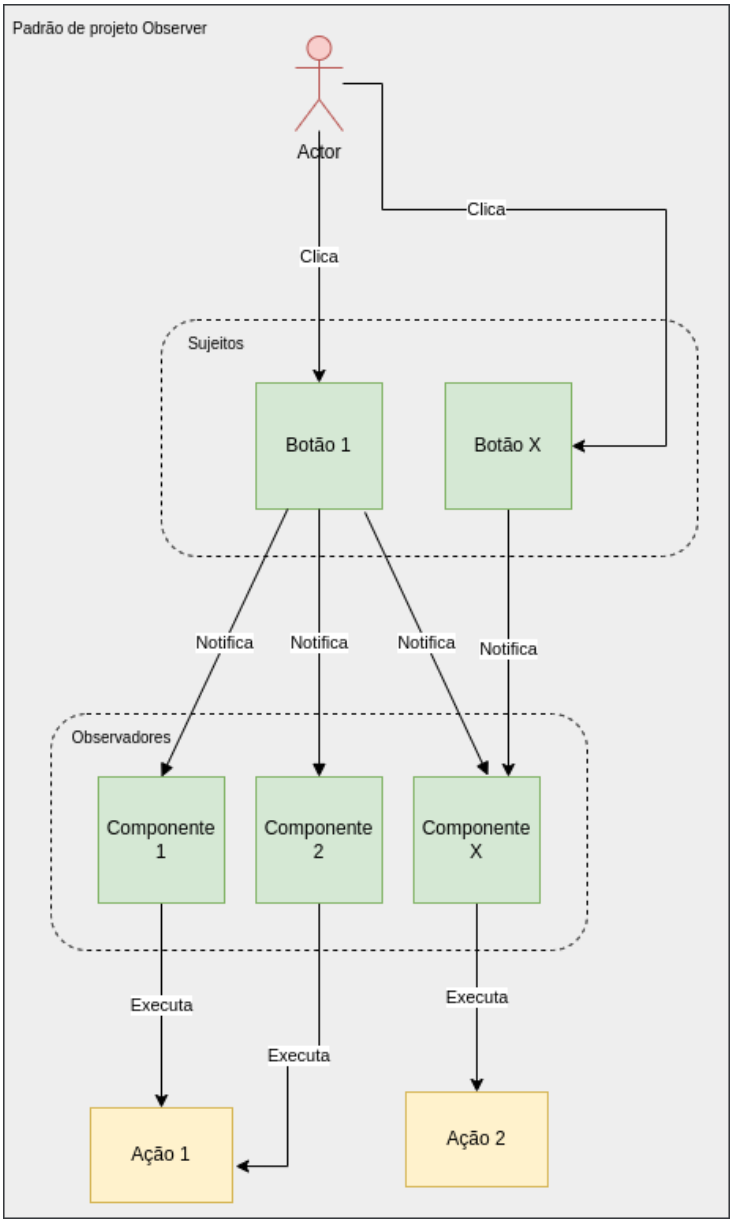

Figura 15 – Exemplo padrão de projeto Observer

3.4.2 Renderização de elementos no mapa

Conforme citado anteriormente, a funcionalidade de renderização de elementos no mapa tem como propósito apresentar para o usuário milhões de objetos plotados nele sem o travamento da interface gráfica. O travamento de interface gráfica no navegador de *internet* é ocorrido pelo fato do JavaScript conter somente uma *thread* principal responsável por renderizar elementos na tela e lidar com o processamento de alguma ação como, por exemplo, um laço de milhares de elementos.

Caso o fluxo passe muito tempo lidando com muitas operações, é possível que a interface gráfica fique congelada enquanto o processamento é ocorrido. Há maneiras de contornar isso como utilizar *web workers*, API responsável por executar um script em

segundo plano separado da *thread* principal (MOZILLA, 2022d), com a limitação de não ser possível de manipular o DOM (*Document Object Model*).

Outra possibilidade seria utilizar a função nativa de intervalo do JS para a cada período em milissegundos executar um trecho do código dentro do tempo proposto. Essa alternativa foi utilizada no sistema e será abordada mais adiante. Houve uma tentativa para o uso do *worker* no sistema, porém não houve resultados satisfatórios pelo fato de que no momento em que há uma cópia de um objeto muito grande, há um congelamento da *thread* principal e consequentemente da interface gráfica, impossibilitando utilizá-lo constantemente para renderizações no mapa.

Para utilizar a funcionalidade em questão é necessário, portanto:

- Clicar no primeiro ícone da esquerda para direita na barra superior do mapa à esquerda.
- É aberto um modal que contém a listagem de todos os itens em que é possível realizar a renderização no mapa.
- Durante esse processo do evento de clicar, é enviado uma notificação para o componente de listagem de itens e feito uma requisição HTTP no servidor CommonAPI a fim de obtê-los propriamente.
- Já fica pré-selecionado a opção de mostrar todos os itens, incluindo os monitoráveis e os não monitoráveis.
- A diferença entre eles é que o primeiro contém elementos monitoráveis e não monitoráveis, enquanto na opção “só monitorados” só possuem monitorados. Os únicos elementos monitorados no momento da produção deste TCC são os transformadores de distribuição e UCBT.

Quando é selecionado um ou mais elementos e clicando no botão “aplicar”, é enviado uma notificação ao componente relacionado ao mapa, juntamente com os elementos selecionados e não selecionados para renderização e, assim, é iniciado o fluxo de renderização no mapa. É iniciado um laço para varrer estes elementos e verificado qual está selecionado para renderização.

Como há fluxos alternativos até o momento de renderização, foi necessário separar o detalhamento para cada caso a fim de facilitar na compreensão de todos as fases até a renderização. No entanto, antes mesmo de abordar propriamente sobre cenários de fluxos, é importante primeiramente apresentar sobre como foi organizado a renderização no mapa, uma vez que há elementos que são convertidos no mapa como linhas, pontos e formas. Assim, foi organizado da seguinte maneira:

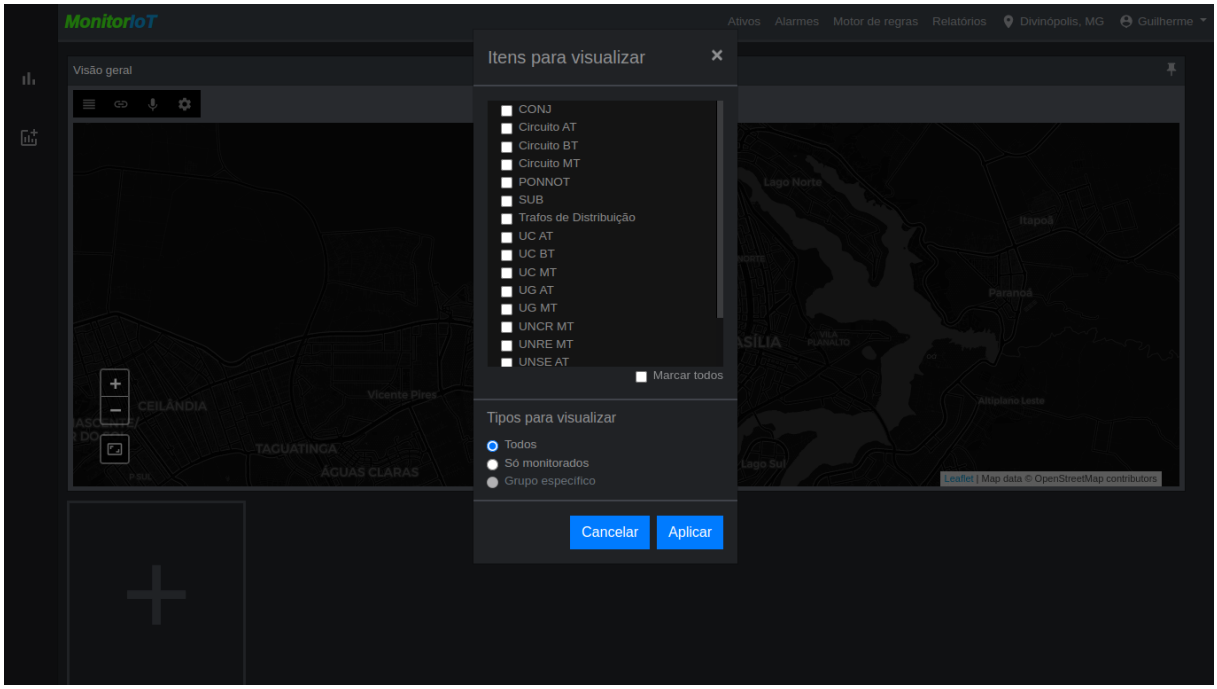

Figura 16 – Menu de itens

- Foi criado um objeto de chave e valor (objeto de camadas) no componente do mapa responsável por conter qual é a categoria do elemento, isto é, se o mesmo é linha, ponto ou forma.
- Os transformadores são classificados como pontos e circuitos como linhas. Todas essas chaves, exceto os pontos monitorados são atribuídas a um valor nulo no momento da inicialização da aplicação.
- Quando é feita uma renderização no leaflet, é retornado um objeto que contém todo os elementos já renderizados, além de métodos como, por exemplo, o de renderizar novamente caso necessite.
- O **objeto de camadas** foi criado com o intuito de prevenir que seja instanciado uma nova renderização em memória toda vez que este for solicitado. Ou seja, caso seja necessário renderizar um elemento do tipo ponto (transformador) que já foi produzido uma vez, não é criado em memória um objeto totalmente novo e sim reaproveitado de forma que apenas seja preciso acionar o método de renderização.

Devido ao fato de haver uma grande quantidade de elementos a serem renderizados em tempo real, ficou inviável manter preso em um laço a resolução de toda a questão de cálculos geométricos dentro da biblioteca Leaflet Glify até o processo de renderização. O motivo disso é porque pode simplesmente travar a navegação do usuário no sistema durante o processamento para casos de monitoramento em tempo real.

A solução disso foi alterar o laço e fazer com que seja executado em um pequeno intervalo de tempo de 20 milissegundos e um limite de 10 milissegundos dentro do laço. Ou seja, a cada 20 milissegundos o laço é executado por um tempo limite de 10 milissegundos. O resultado disso é praticamente a ausência de travamentos na interface gráfica, juntamente com a renderização constante sem afetar a experiência do usuário.

Agora é possível apresentar os diferentes cenários de fluxos até o momento da renderização.

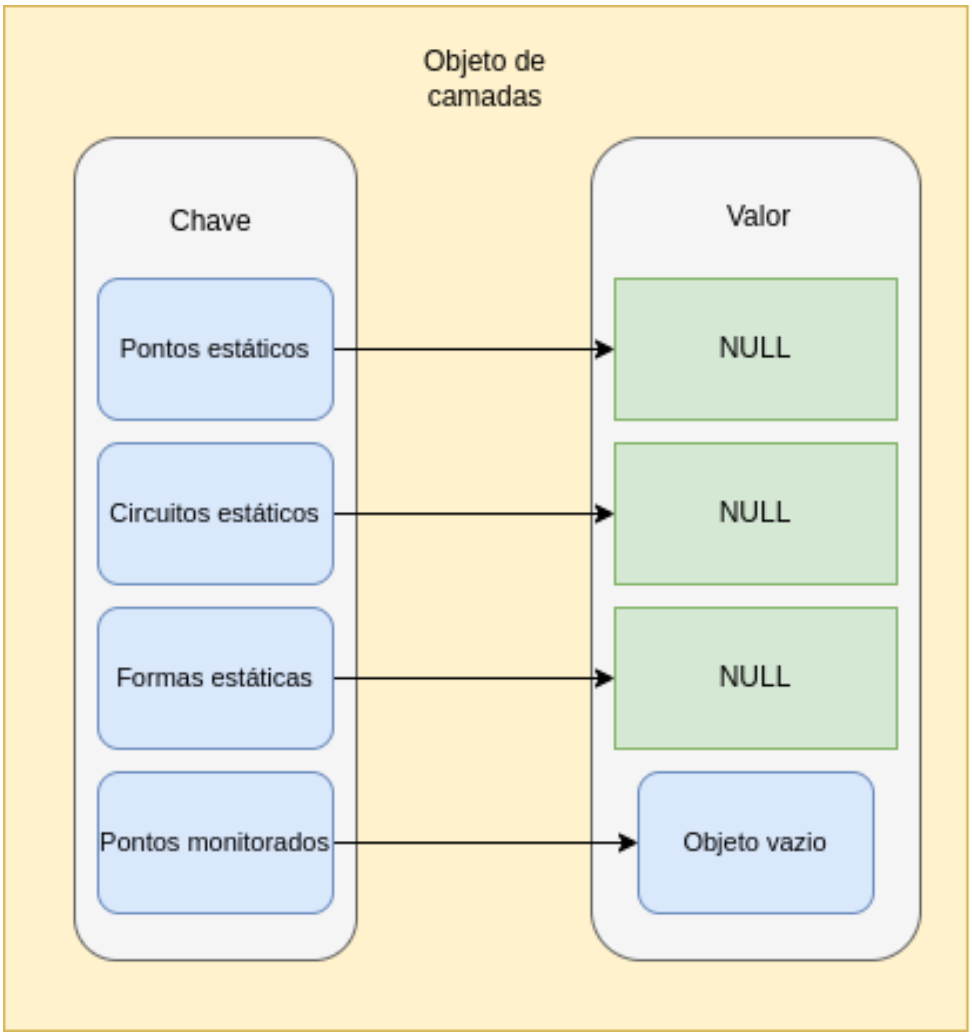

Figura 17 – Objeto de chave e valor representando as camadas

3.4.2.1 Renderizando pela primeira vez um elemento (não monitorado)

Continuando no ponto em que o laço é iniciado e partindo somente sobre os elementos selecionados, o fluxo é executado:

1. É verificado no **objeto de estado de renderização**, responsável por armazenar o estado de renderização dos elementos no mapa, se o elemento na iteração em questão já está renderizado no mapa ou não.

2. Caso não tenha sido renderizado ainda, é verificado se os dados do elemento já foram obtidos no servidor e salvos em disco.

3. Caso tenham sido, são armazenados na memória RAM dentro do **objeto de elementos de servidor**. Este objeto tem o papel de evitar com que seja feito uma requisição nova no servidor para cada nova renderização.

4. Em seguida, é verificado no **objeto de camadas** se já foi feito pelo menos alguma instância de renderização no mapa da mesma categoria desse elemento.

5. Caso positivo, é concatenado no **objeto de camadas**, sendo a chave a categoria do elemento, esses novos dados e, assim, renderizados no mapa. Nota-se que antes mesmo de efetuar a renderização, é realizado um filtro no objeto citado para garantir que seja renderizado somente o que foi solicitado. Em outras palavras, é possível que neste objeto tenham elementos que não devem ser mais renderizados, dependendo da decisão selecionada no modal de listagem de itens.

6. Caso não haja dados em disco, é realizado uma requisição HTTP para o servidor CommonAPI solicitando os dados geográficos e, após isso, armazenados em disco.

7. Como os dados do servidor são obtidos de forma paginada com o intuito de diminuir o tráfego e melhorar a performance do mesmo, é utilizado múltiplas requisições HTTP de maneira assíncrona percorrendo toda a paginação possível, concatenando os dados e salvando-os em disco posteriormente.

8. O serviço em questão requer como argumentos o tipo do item e o índice para obtenção dos dados. A informação de quantidade de páginas de cada elemento é obtida já anteriormente no serviço de listagem de itens quando se é aberto o modal e passado ao enviar a notificação para o mapa.

9. É utilizado o forkJoin da biblioteca RxJS para executar essas requisições de maneira sequencial e, após a finalização, é enviado a notificação para realizar a somatória faltante de requisição no objeto de chave e valor (objeto gerenciador de requisições) que gerencia isso para fins de sincronia.

10. O **objeto gerenciador de requisições** foi criado com o intuito de contabilizar pela categoria do elemento selecionado a solicitação de renderização. Porém, para que seja sincronizado todos os elementos de uma vez, é preciso aguardar com que as múltiplas requisições de outros elementos finalizem.

11. A biblioteca utilizada para armazenar os dados em disco conforme citado anteriormente é o LocalForage, pois o tamanho dos dados que podem vir do servidor é capaz de alcançar até 80 *megabytes* e utilizar a API LocalStorage, padrão do navegador

como Google Chrome, por exemplo, não haveria a possibilidade de armazenar mais que 5 *megabytes* (GOOGLE, 2015).

12. A legenda de itens estáticos é gerada, exibindo por cores elementos representados no mapa.

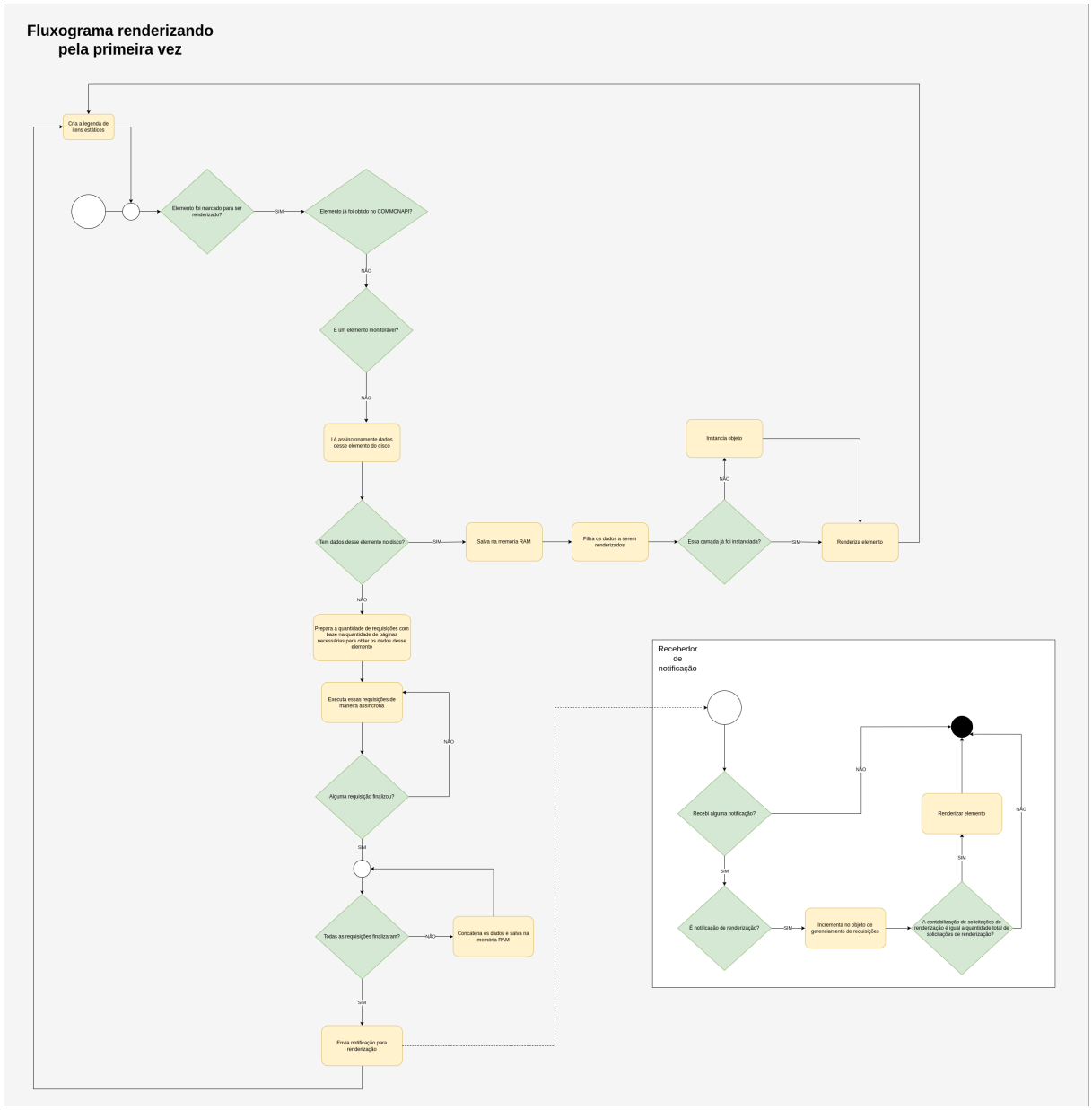

Figura 18 – Fluxograma de renderização pela primeira vez (não monitorados)

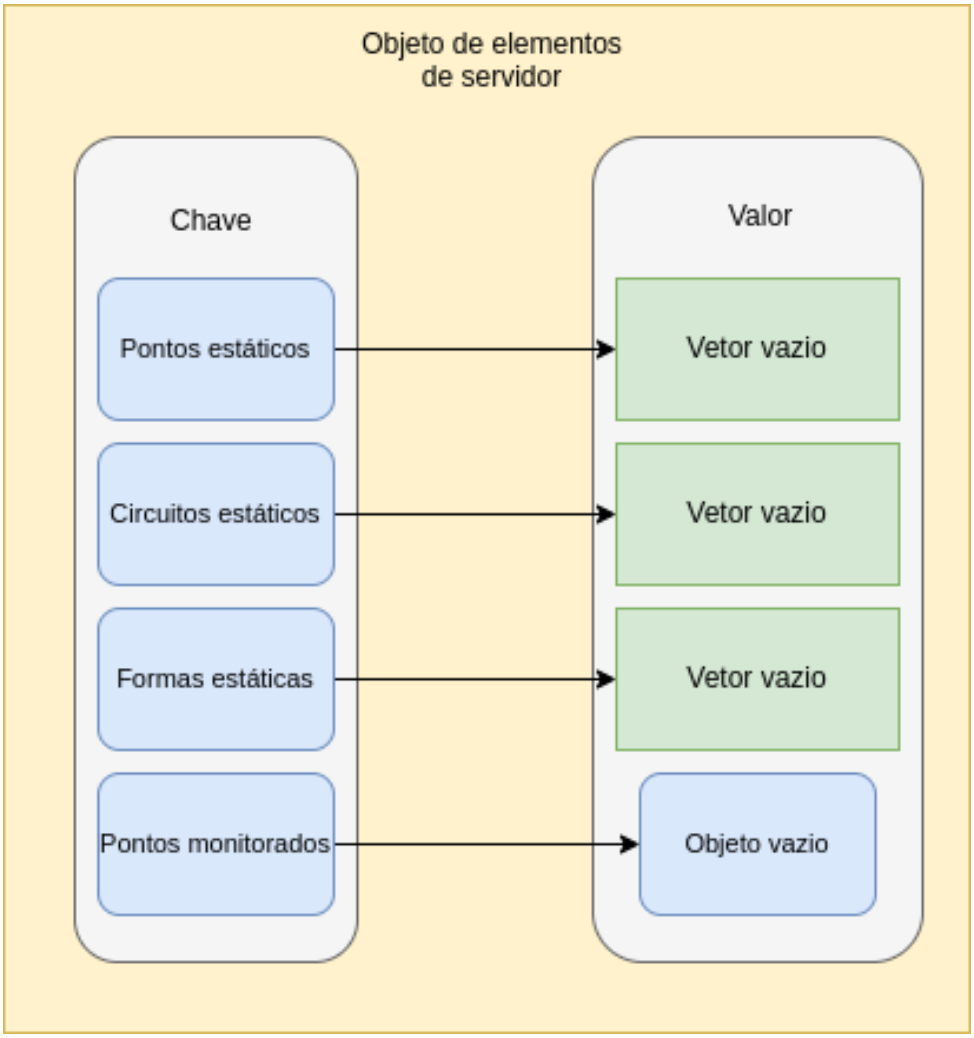

Figura 19 – Objeto responsável por salvar os dados obtidos do servidor

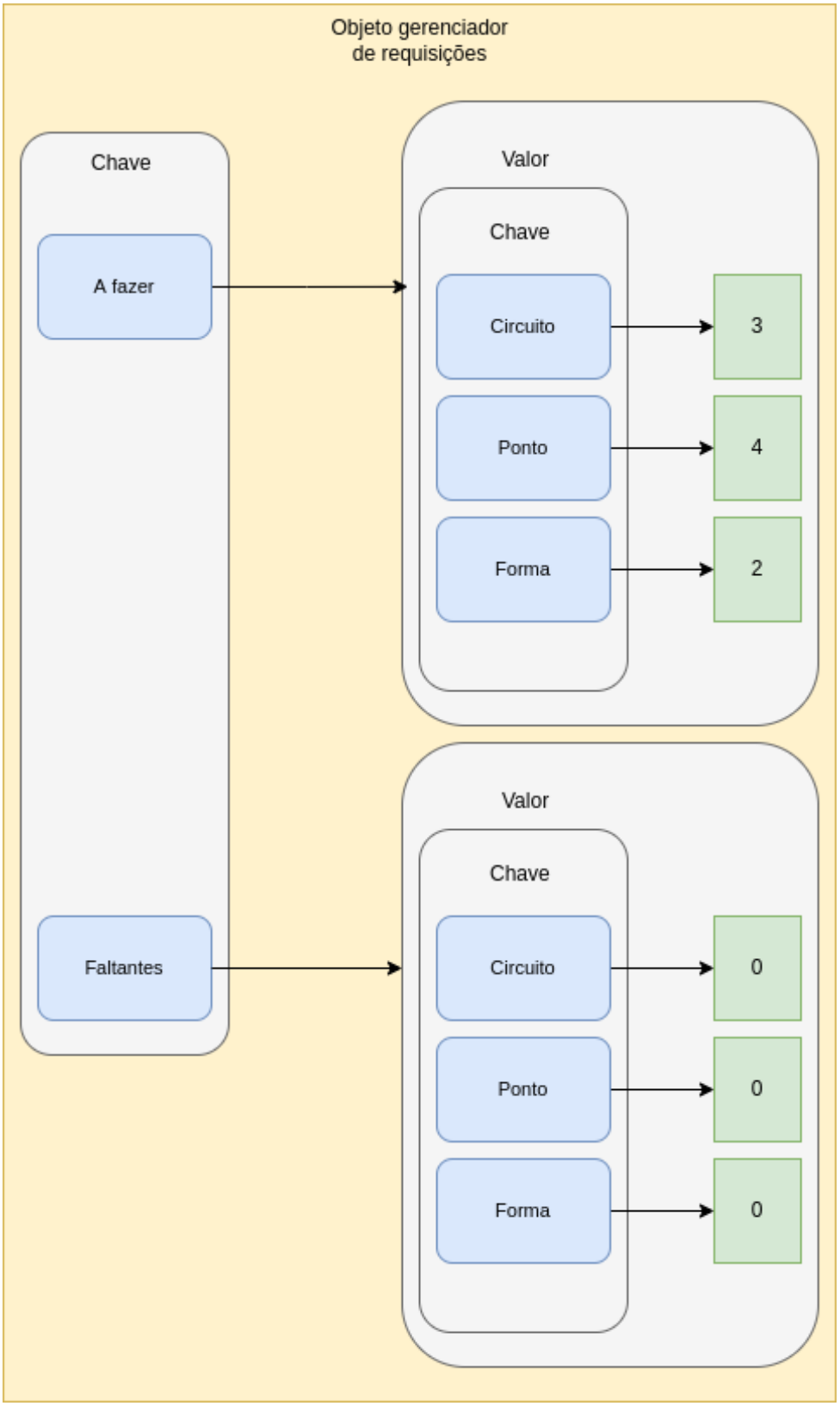

Figura 20 – Objeto responsável por gerenciar as requisições

3.4.2.2 Renderizando um elemento novamente (não monitorado)

Caso seja necessário renderizar novamente um elemento no mapa depois de fechar a página do sistema ou remover os elementos já renderizados, são necessários:

1. É verificado no **objeto de estado de renderização** se a chave correspondente do elemento existe.
2. Caso exista, é renderizado novamente o transformador utilizando a instância leaflet no valor do **objeto de camadas**.
3. Caso contrário, é verificado em disco se o transformador desejado existe. Caso positivo, é resgatado do disco e armazenado no **objeto de elementos do servidor**. Após isso, é criado uma nova instância e armazenada no **objeto de camadas**. Feito isso, é realizada a renderização no mapa.
4. Por fim, como o elemento não é monitorável, o fluxo é finalizado e a legenda é renderizada no mapa também.

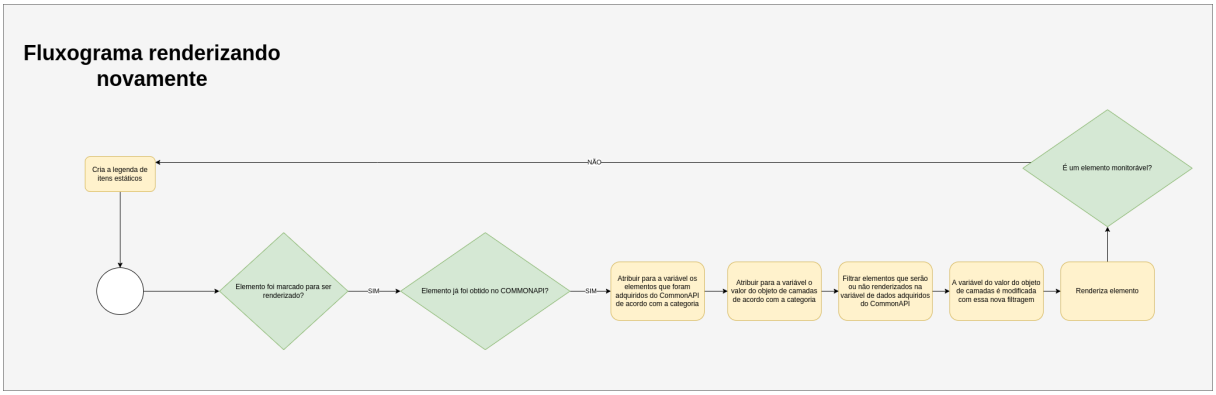

Figura 21 – Fluxograma de quando elemento é renderizado novamente (não monitorados)

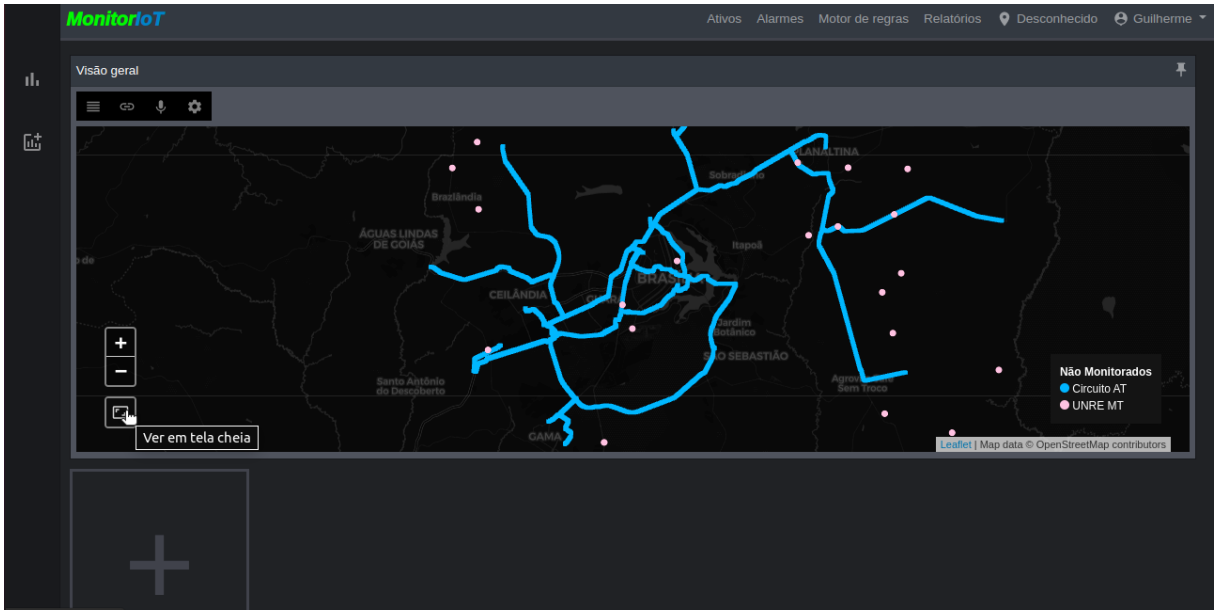

Figura 22 – Elementos de diferentes categorias renderizados no mapa (não monitorados)

3.4.2.3 Renderizando um elemento pela primeira vez (monitorado)

O fluxo de renderizar um elemento monitorado pela primeira vez é semelhante ao tópico 3.4.2.1. As principais diferenças comparadas ao outro fluxo são:

1. Caso os dados não estejam no disco, é estabelecido uma conexão WS para o servidor *back-end* VPlant. Essa conexão tem o objetivo de obter todos os objetos do elemento em seu último estado. A biblioteca para comunicação WS foi desenvolvida por (FERREIRA, 2022) e implementada para se obter os dados do VPlant de maneira paginada, bem como reduzir a grande quantidade de volume de dados recebidos. Para isso, os dados enviados ou recebidos por meio da biblioteca são binários, conforme configurado, e convertidos em objetos internamente, de modo transparente para as aplicações que a utilizam.

2. Os dados são recebidos em partes e à medida que eles chegam no MonitorIOT, são concatenados no **objeto de elementos de servidor** com base na chave desejada. Uma função *callback*, responsável por indicar que não há mais dados a serem recebidos e finalizar a conexão WS, é inserida como parâmetro no momento de estabelecer a conexão WS.

3. A renderização é executada e a instância dela é salva no **objeto de camadas** na chave de pontos estáticos. É realizado um filtro no valor deste objeto e obtido somente as coordenadas que são monitoráveis com base no nome do elemento e salvos em um novo **vetor de coordenadas monitoradas**.

4. Conforme é inserido os dados no **vetor de coordenadas monitoradas**, um novo **objeto de código e índice** de chave (código do elemento) e valor (índice da posição do dado inserido no **vetor de coordenadas monitoradas**) é criado para ser inserido durante essa iteração.

5. É estabelecido, portanto, uma nova conexão WS persistente no VPlant, responsável por obter os dados de coordenadas juntamente com os estados atualizados de cada um.

6. Esses dados são percorridos e para cada iteração é obtido a posição com base na chave do **objeto de código e índice**. Assim, essa posição é utilizada para resgatar no vetor de coordenadas monitoradas o estado que deve ser alterado.

7. Por fim, é realizado a renderização já com os elementos com os seus respectivos estados atualizados.

8. O motivo da utilização do **objeto código e índice** é para não ser necessário com que o **vetor de coordenadas monitoradas** seja varrido até que seja encontrado o valor respectivo a fim de alterar seu o estado. Logo, no pior caso de complexidade poderia ser necessário percorrer cerca de 1 milhões de elementos no caso de, por exemplo, do UCBT, congelando a interface gráfica do usuário.

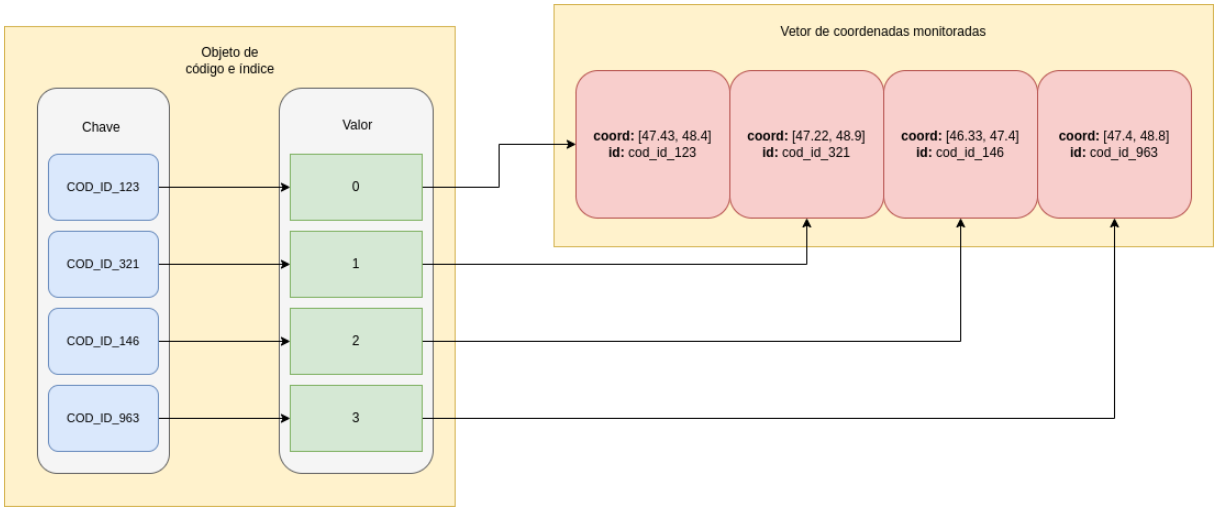

Figura 23 – Relação do objeto de código e índice e vetor de coordenadas monitoradas

Sobre o momento da renderização, é verificado na chave dos monitoráveis no **objeto de camadas** se possui algum objeto com valor internamente. Diferentemente das outras chaves que os valores correspondem a nulo ou a um vetor, essa particularmente possui um outro novo objeto que utiliza como chave o nome do elemento e o valor propriamente os pontos.

Portanto, cada elemento monitorável que for renderizar possui sua camada própria isolada das restantes. A razão por ter feito isso é devido ao fato de se fosse renderizar na camada estática, geraria um efeito indesejado no mapa onde os elementos desaparecem e reaparecem por causa da renderização constante necessária para os monitoráveis.

Isso significa que para cada mudança de estado é removido todos os elementos plotados no mapa e renderizado-os novamente com suas respectivas alterações, levando um tempo considerável para todo esse processo. Com isso em mente, é verificado nesse objeto se a instância foi criada ou não. Caso não seja, é instanciado a camada no valor da chave do **objeto de camadas** e renderizada no mapa. Feito isso, estes objetos renderizados com estado atualizado sobrepõem os estáticos já presentes no mapa exatamente na mesma coordenada, gerando a impressão que foi alterado o estado daquele ponto no mapa.

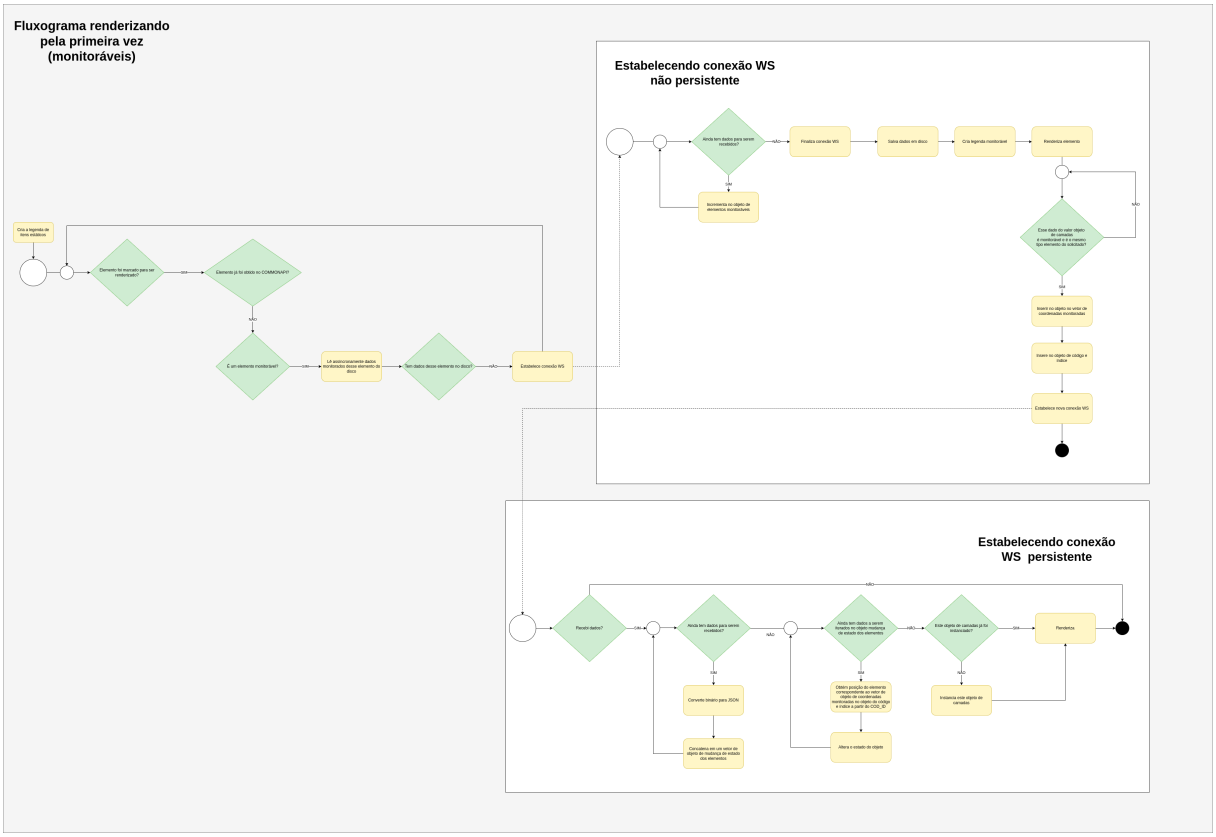

Figura 24 – Renderizando pela primeira vez (monitorados)

3.4.2.4 Renderizando um elemento novamente (monitorado)

Caso o elemento já tenha sido renderizado, o fluxo é semelhante ao que foi citado anteriormente aos não monitorados com a diferença de após feito todo o processo de renderização, é estabelecido uma conexão persistente WS e seguido exatamente o fluxo de renderização constante à medida que são enviados dados ao MonitorIOT citado no tópico anterior.

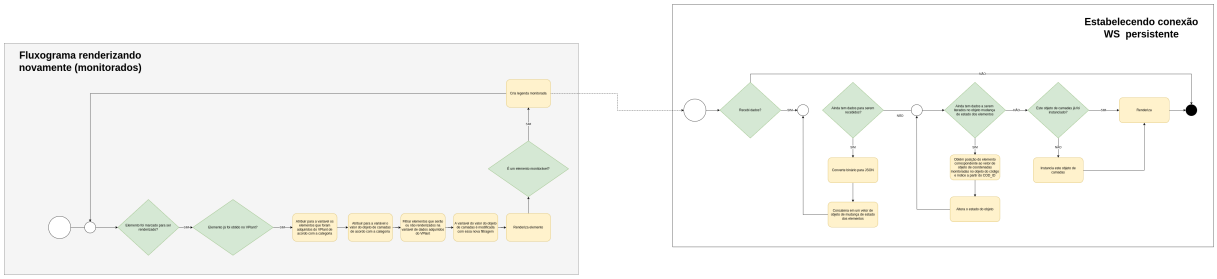

Figura 25 – Renderizando novamente (monitorados)

3.4.3 Criação de gráficos

A criação de gráficos no MonitorIOT, conforme mencionado anteriormente, teve o propósito de possibilitar ao usuário a criar gráficos com diferentes tipos de medidas e deixá-los à vista na página inicial.

Para executar essa funcionalidade são necessárias:

- Inicialmente clicar no ícone com o símbolo positivo. Feito isso, um novo modal é apresentado na tela e como forma de manter organizado para o usuário seguir o fluxo até o momento da criação do gráfico propriamente, foi enumerado cada parte da tela.

- O número 1 indica a fonte de dados, responsável por receber o título do gráfico, o tipo de dados desejado e o valor da fonte de dados (um identificador, por exemplo). A funcionalidade de fonte de dados utiliza o *back-end* do MonitorIOT por meio de uma requisição HTTP para obter mais facilmente um identificador.

- O número 2 indica a dados a serem exibidos, responsável por receber quais serão as variáveis utilizadas para a exibição do mapa. É obtida internamente pelo MonitorIOT, isto é, existe um vetor que inclui objetos de chave e valor que contém todas essas opções em memória. É executado uma recursão com o objetivo de percorrer por esse vetor e exibir os elementos na tela.

- O número 3 indica a lista de variáveis, responsável por listar as curvas que foram selecionadas, normalizá-las e apagá-las caso necessário. A lista de variáveis depende do vetor de objetos citado acima para que possa ser mostrado em ordem hierárquica de acordo com o que foi selecionado. Quando se é clicado em algum item dentro dos dados a serem exibidos do tipo selecionável, é invocado uma função responsável por percorrer a partir do item atual os nós ancestrais, concatenando o seu título a cada nível da recursão para que seja exibido na tela.

- Por fim, o número 4 indica as configurações do gráfico, responsável por definir o tipo de gráfico que será exibido no sistema, juntamente com o período dos dados

que devem ser apresentados. Sobre o tipo de gráfico para este trabalho de conclusão foi disponibilizado apenas a série temporal. Tem-se a possibilidade de selecionar o tipo de gráfico, porém até o momento existe apenas o tipo temporal. Outro campo utilizado que influencia diretamente a consulta dos dados no BDGD pelo MonitorIOT é o de data, responsável por filtrar a partir dela até o momento atual os dados necessários. Todos esses parâmetros desse campo são inseridos como argumentos para a chamada do serviço.

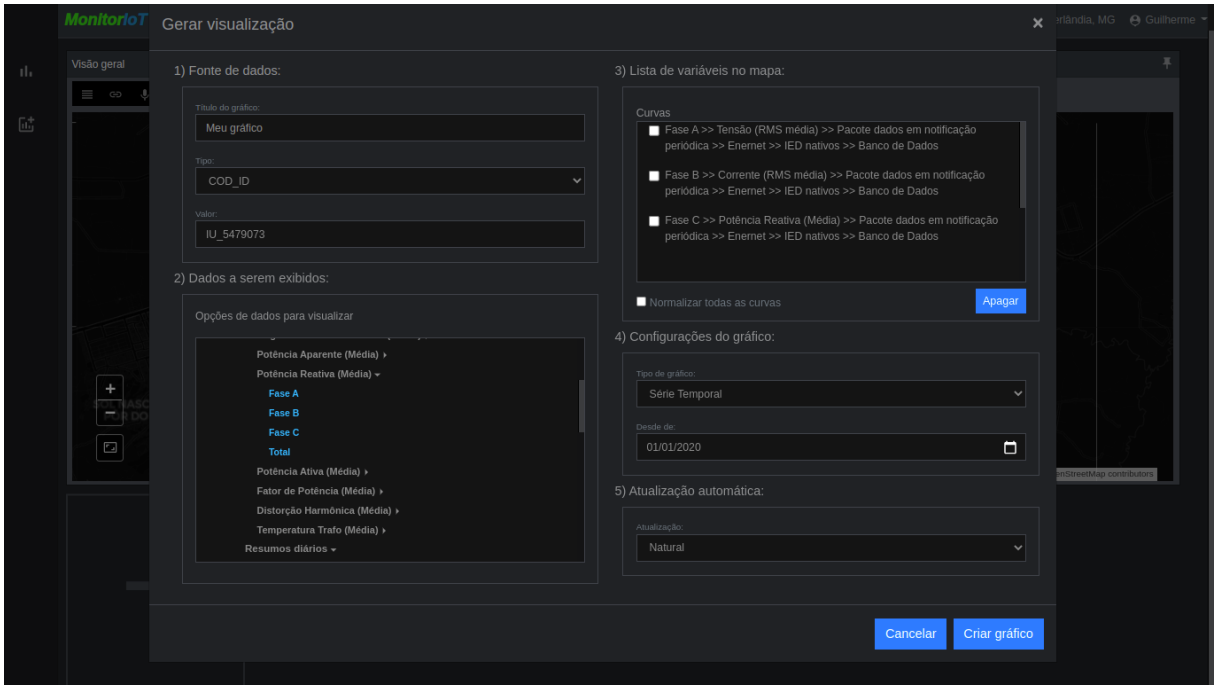

Figura 26 – Tela de criação de visualização personalizada

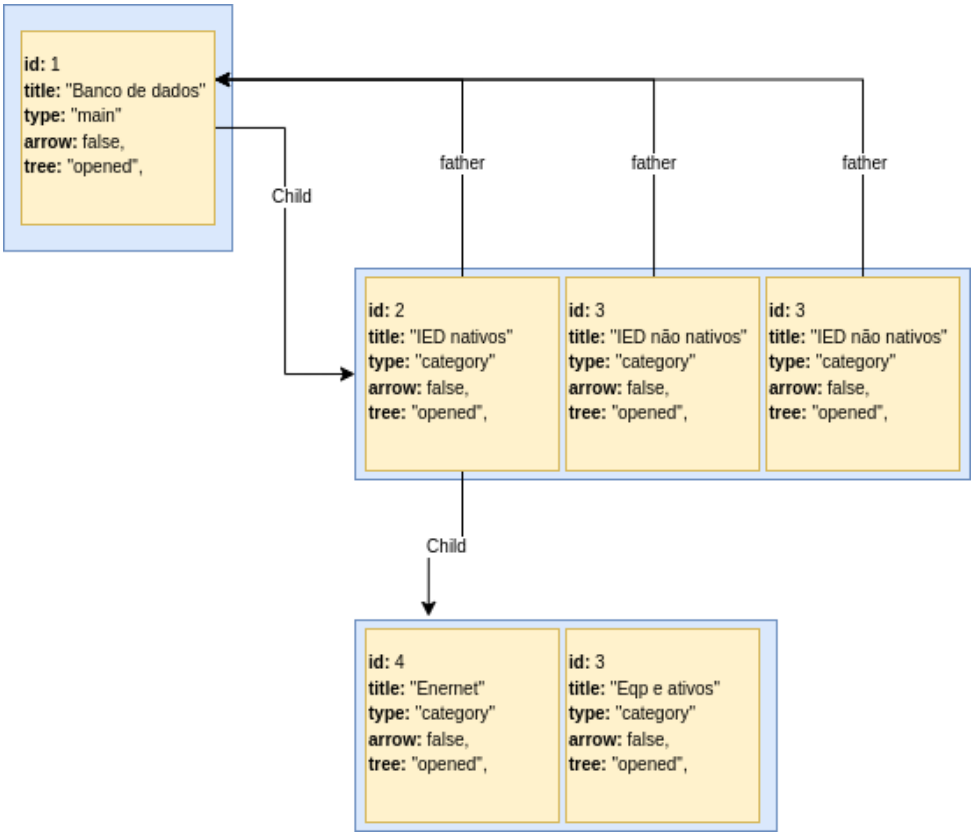

Figura 27 – Vetor de objetos do diretório

3.4.3.1 Plotagem do gráfico

Após realizar todas as configurações citadas nos tópicos anteriores, é realizado múltiplas requisições HTTP no MonitorIOT *back-end* para o serviço de gerar visualização. Essas requisições têm o intuito de retornar propriamente os valores que devem ser plotados no gráfico com base na lista de variável.

Durante cada finalização de requisição, os dados são formatados para que possam ser utilizados na plotagem do gráfico da biblioteca E-Charts. Após isso, é enviado uma notificação juntamente com os dados obtidos para o componente de conteúdo (local em que se encontra o mapa e os gráficos construídos). Este componente recebe os dados, adiciona em um vetor de gráficos e a biblioteca os renderiza no mapa. Este vetor é do tipo da biblioteca Gridster, responsável por mover os elementos no espaço do conteúdo livremente, além de conter os dados formatados que são recebidos da notificação.

Caso seja necessário editar este gráfico, basta clicar no ícone de engrenagem próximo a barra de título da plotagem e, assim, é enviado uma notificação ao componente de criação de gráficos juntamente com os dados (título, data, etc) previamente enviados juntamente com a mensagem que seria para a sua edição. Com isso, a tela é preenchida com esses dados sendo possível realizar alguma modificação e realizar a plotagem novamente.

É possível também deixar o gráfico em tela cheia clicando no ícone de expansão na plotagem. Com isso, é invocado uma função presente no objeto documento nativo do DOM. Outra possibilidade é excluir o gráfico que está plotado clicando no ícone de lixeira, sendo removido da lista de gráficos plotados.

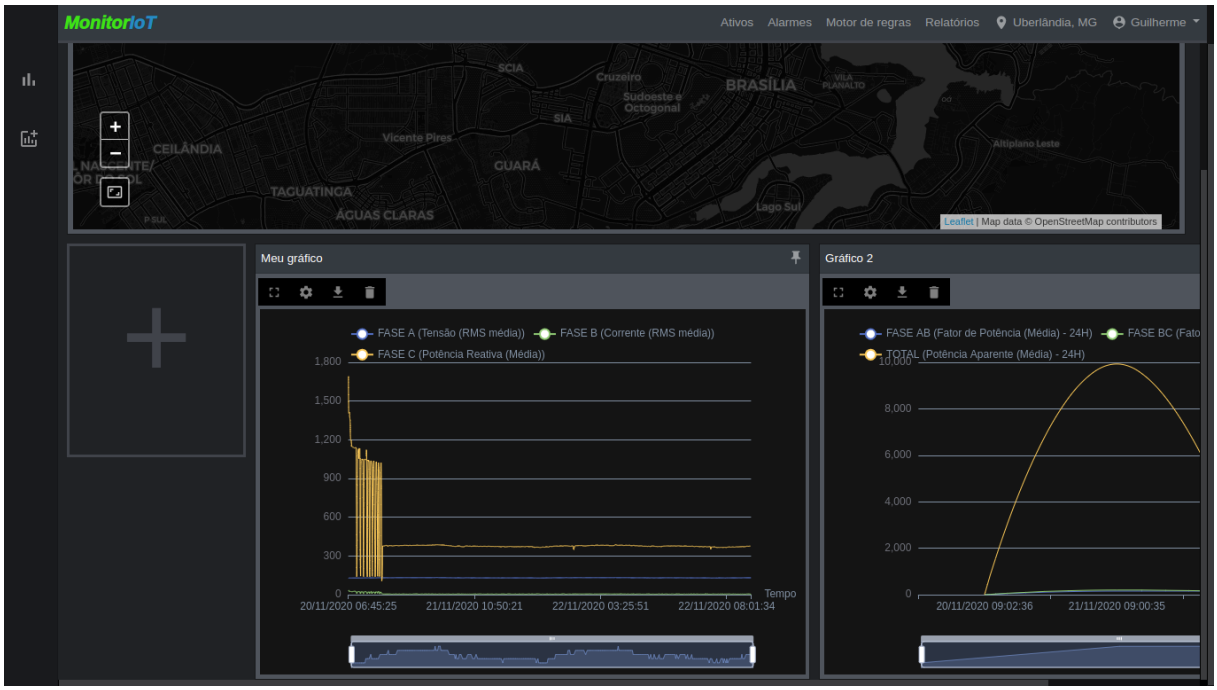

Figura 28 – Tela de gráficos plotados

3.5 Resumo

Neste capítulo foram apresentados a arquitetura geral e estrutura do MonitorIOT, quais sistemas externos ele utiliza, juntamente com a sua estrutura de dados. As tecnologias utilizadas no projeto foram apresentadas, juntamente com o motivo de suas utilizações. Por fim, foram apresentadas as duas principais funcionalidades com detalhes (mapa e criação de gráficos), juntamente com fluxogramas e capturas de telas reais do MonitorIOT.

4 Resultados e discussões

4.1 Resultados

Neste capítulo são apresentados alguns dos resultados obtidos pela modelagem proposta. Em especial, é dado enfoque em cenários que produzem funcionalidades de mapa GIS e criação de gráficos.

Em relação ao mapa, a primeira possibilidade ofertada ao usuário é expandir a tela para o modo 'tela cheia'. Isso ajuda o usuário a manter o foco no estado dos ativos. Para executar tal função é preciso clicar no ícone de expansão na parte inferior a esquerda em que fica o mapa.

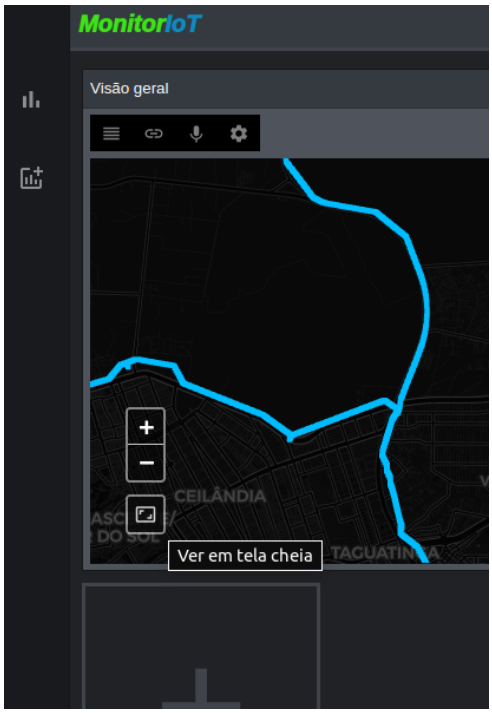

Figura 29 – Opção de tela cheia no mapa

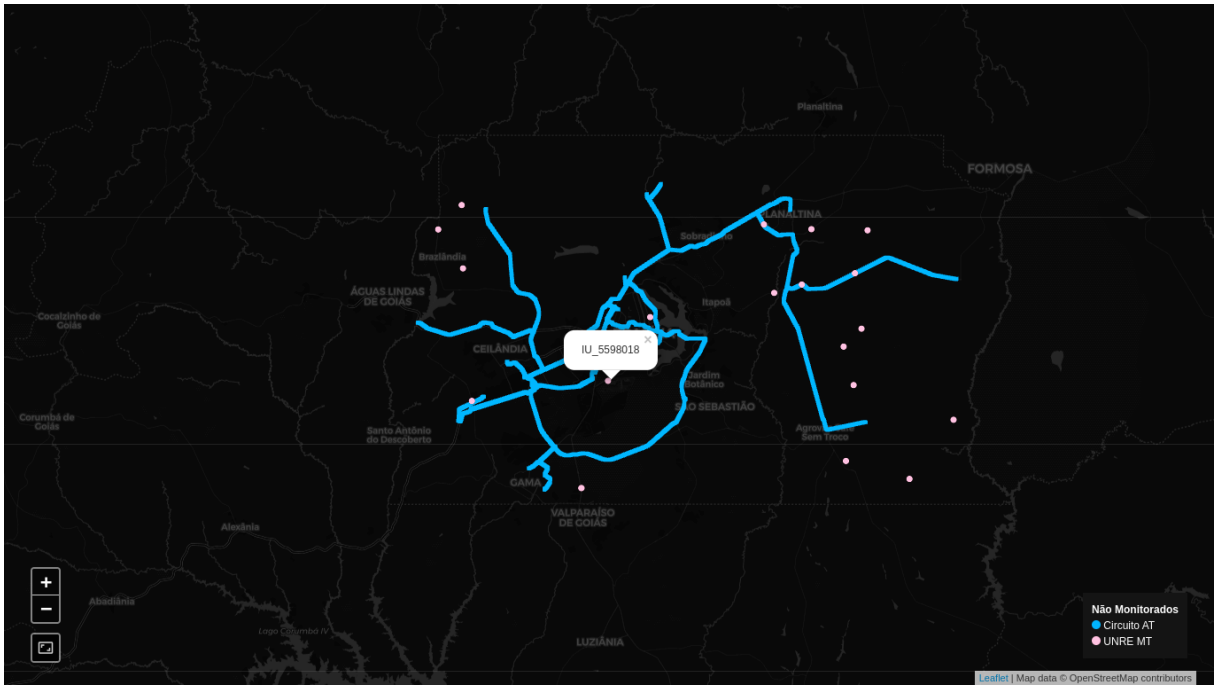

Figura 30 – Mapa em tela cheia

A plotagem de mais de 1 milhão de ativos monitoráveis do tipo UCBT no mapa é possível sem travar a interface do usuário e com relativa "baixa latência", ao alterar a localização no mapa com o mouse.

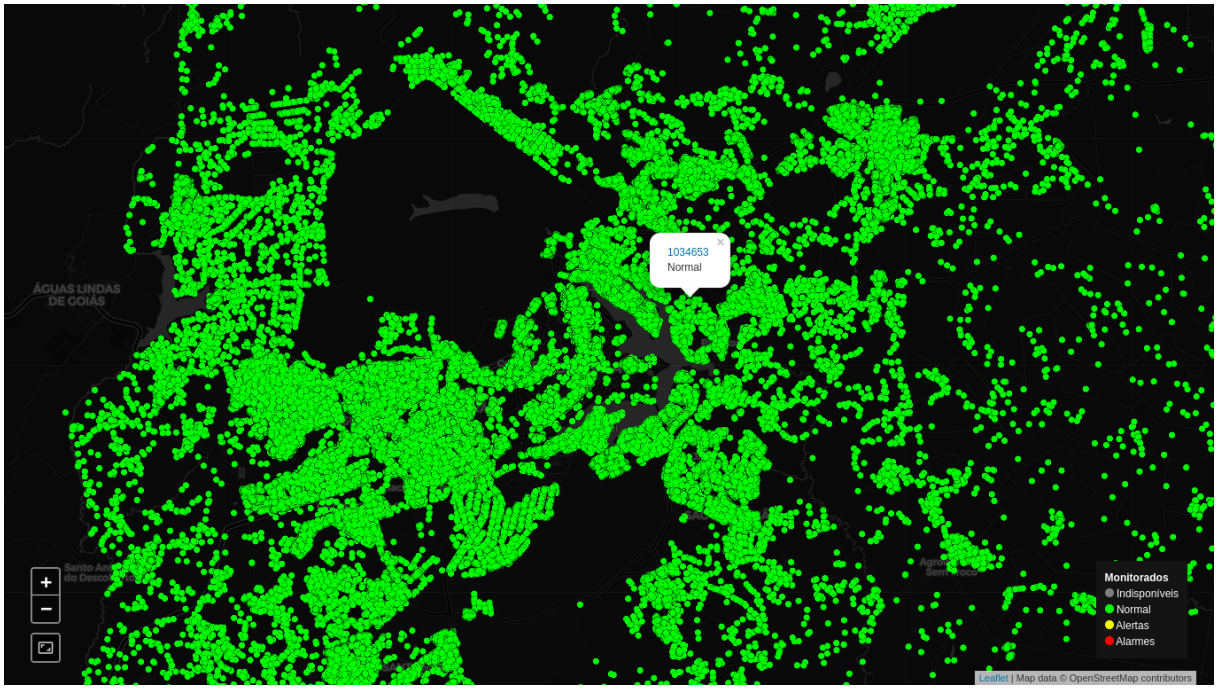

Figura 31 – Mapa com elementos UCBT

A plotagem de mais de 28 mil objetos monitoráveis do tipo transformadores de distribuição no mapa é possível também com as mesmas características que a UCBT.

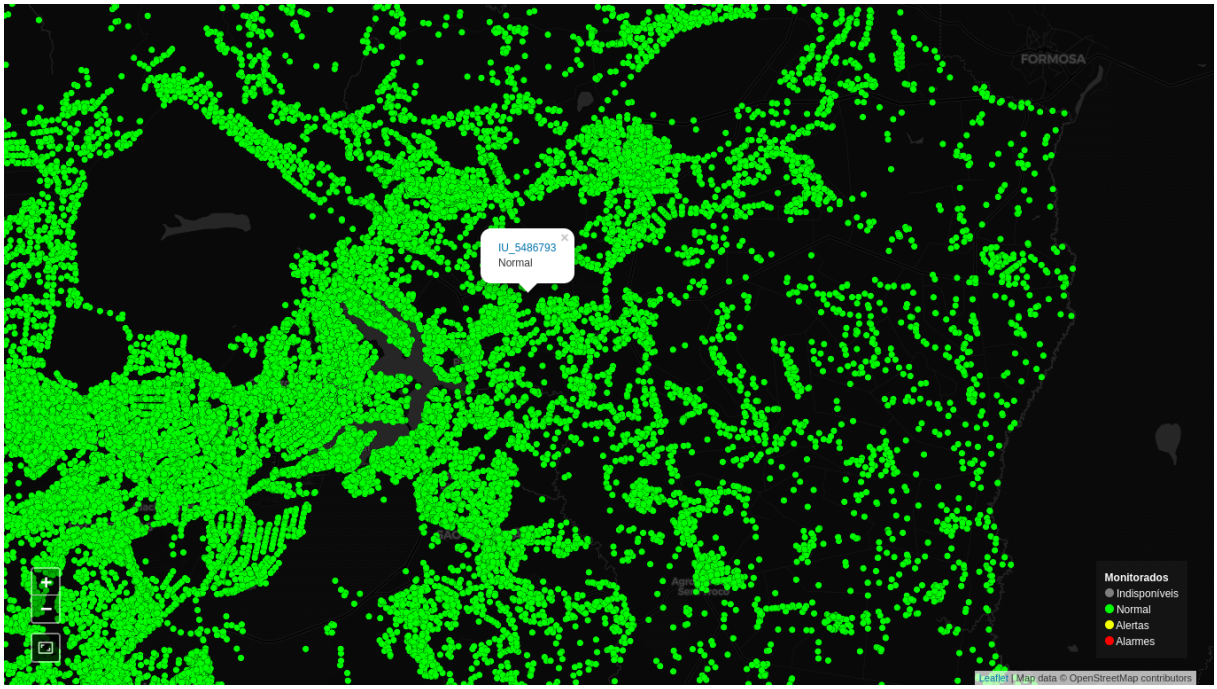

Figura 32 – transformadores de distribuição monitorados

Outra funcionalidade ligada ao mapa seria plotar todos os elementos não monitoráveis e monitoráveis de uma vez no mapa sem alterar a performance da navegação do mesmo.

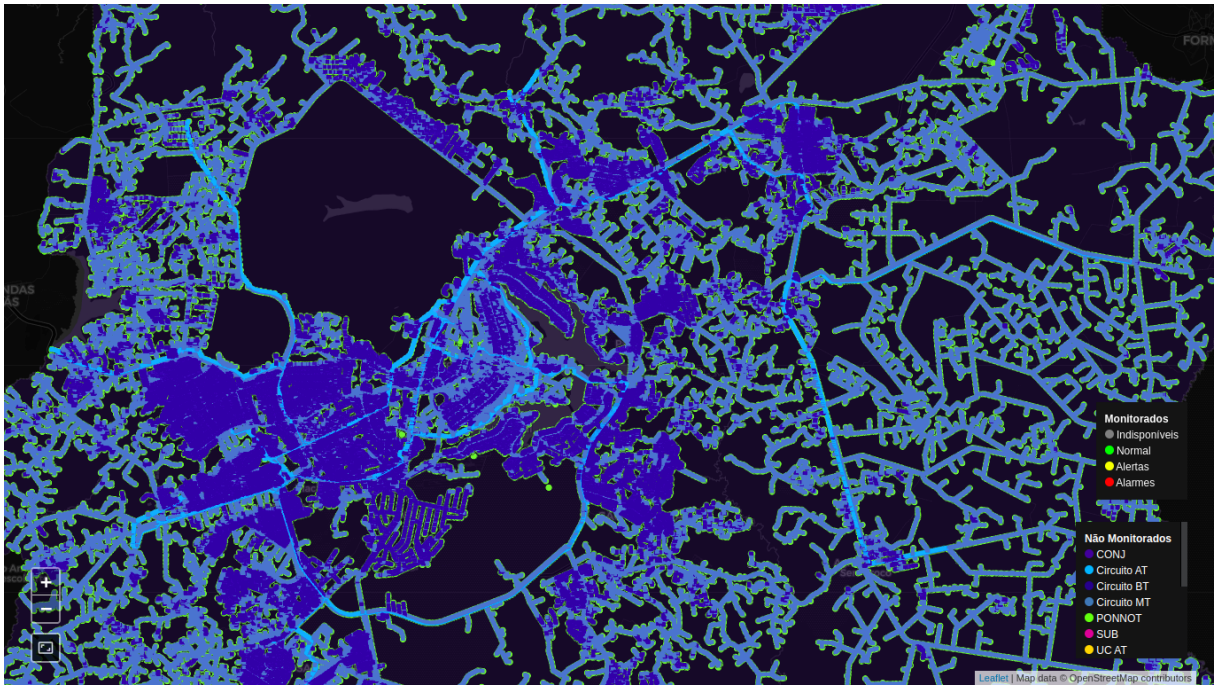

Figura 33 – Mapa em tela cheia com todos os elementos plotados

É possível criar na funcionalidade de criação de gráficos um que tenha diferentes tipos de variáveis distintas, sendo possível correlacioná-las e abstrair alguma possível

análise entre elas. Com isso, para criar um gráfico, é preciso clicar no botão que contém o símbolo de adição na tela e, assim, será aberto um modal que contém as opções possíveis para o usuário criá-lo.

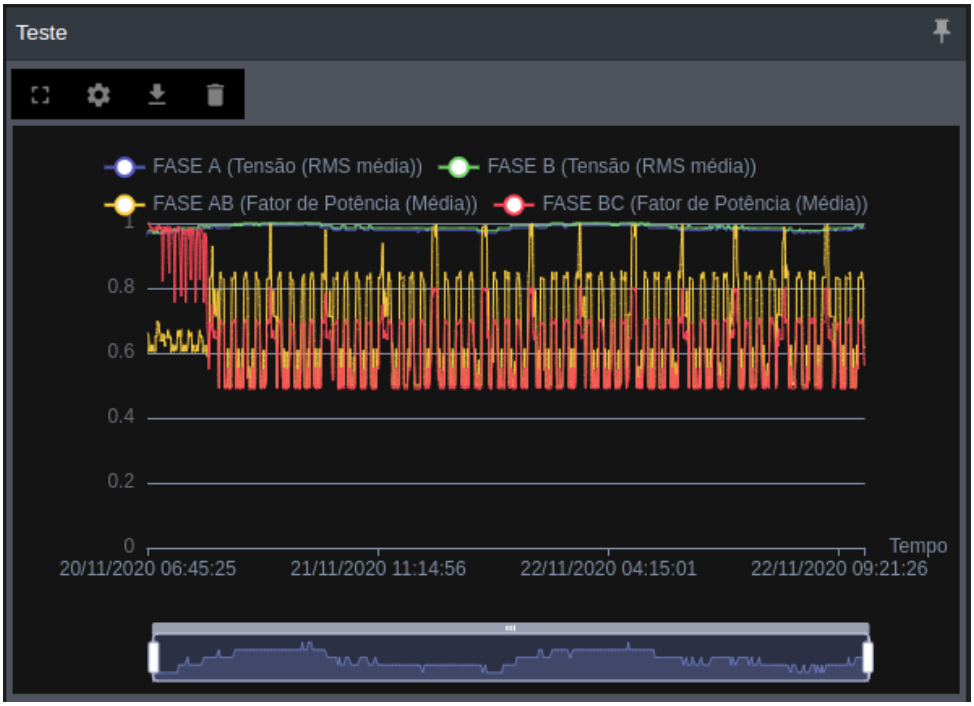

Figura 34 – Gráfico personalizado criado pelo usuário

Além disso, é possível criar múltiplos gráficos diferentes e movê-los ou ajustar o tamanho dos mesmos da maneira que desejar na tela principal, garantindo, assim, uma flexibilidade ao usuário. O usuário também é capaz de movimentar o mapa principal pela tela. Portanto, para ativar tal funcionalidade de mover os elementos é preciso clicar no ícone de alfinete na parte superior à direita da barra.

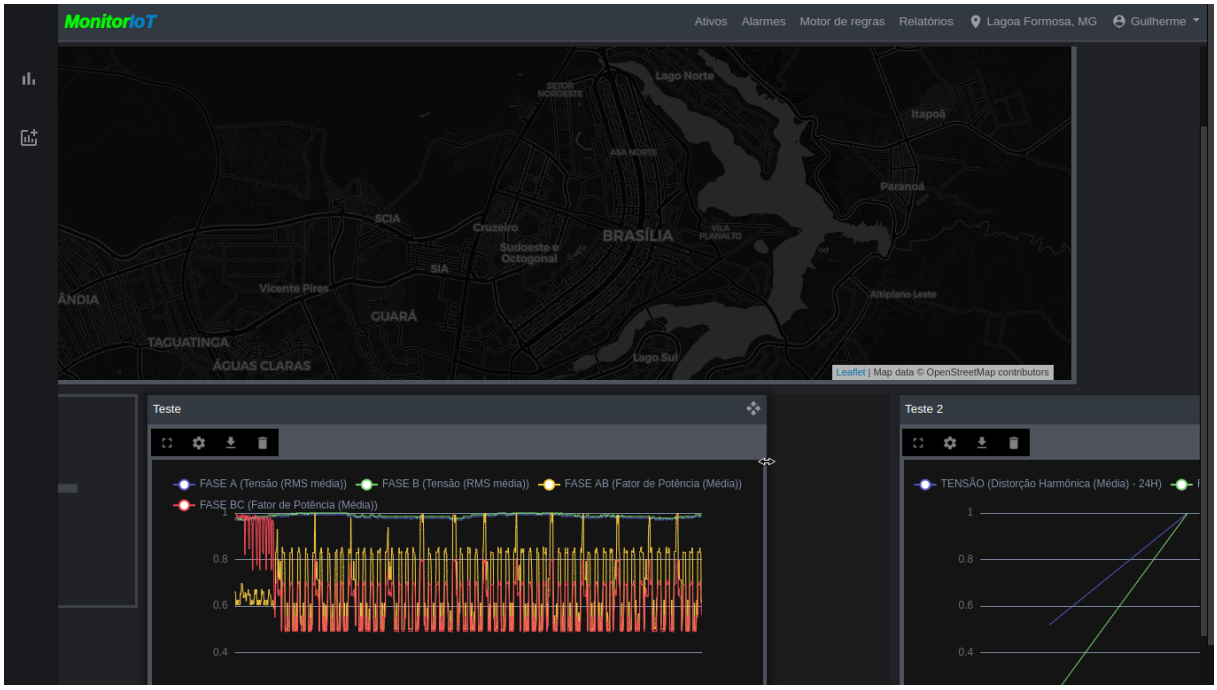

Figura 35 – Ajustando gráfico

Por fim, em relação à criação de gráficos, é possível expandí-lo em tela cheia ou exportá-lo para a máquina em formato PNG (Gráficos Portáteis de Rede).

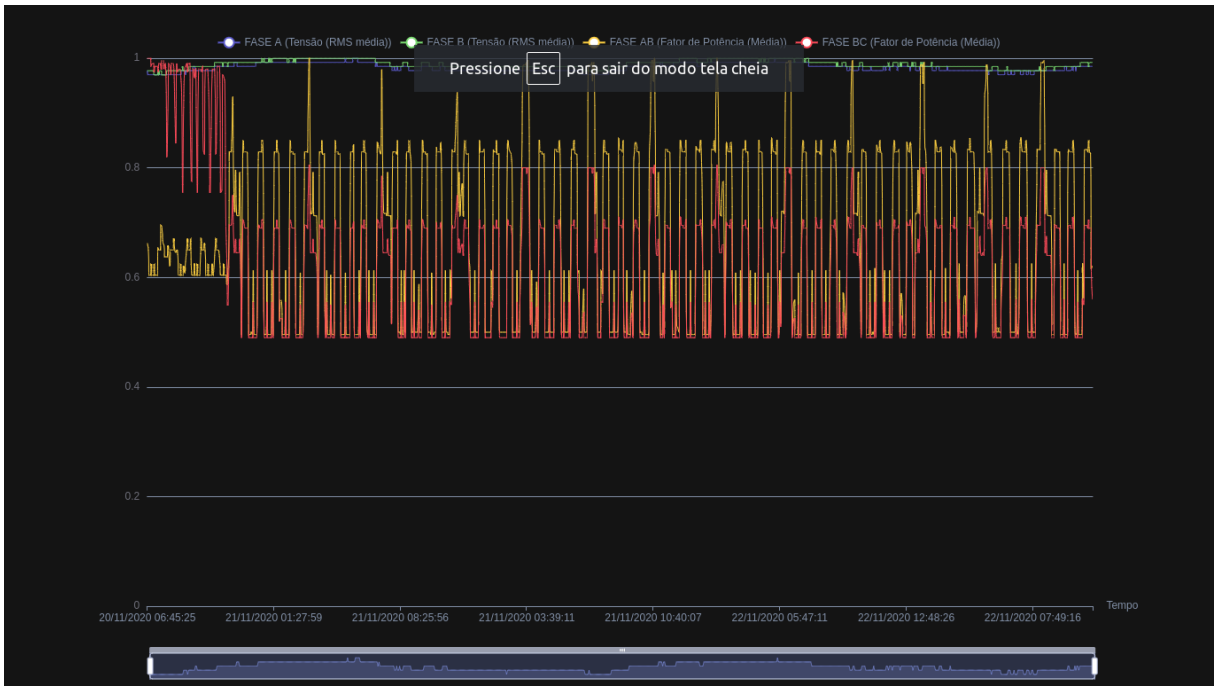

Figura 36 – Gráfico em tela cheia

4.2 Discussões

A tecnologia utilizada neste trabalho foi viável para conseguir renderizar milhões de elementos, monitorar e atualizar os seus estados sem o travamento da interface gráfica para o usuário e sem a latência ao movê-lo. Além disso, foi possível salvar em cache os elementos que não são monitoráveis em disco do cliente, reduzindo por consequência a carga gerada no servidor CommonAPI.

A solução de se criar o gráfico de maneira personalizada pode auxiliar o operador a buscar correlações de acordo com as curvas apresentadas, juntamente com a flexibilidade de movê-lo ou aumentá-lo na tela principal.

A utilização do GIS para *smart grids* em um navegador de *internet* e dinâmico contribuiu monitorar grandes plantas (muitos pontos), havendo responsividade no seu uso. Isso significa que não é necessário estar em uma sala de supervisório para realizar um monitoramento, sendo possível até de casa em um computador convencional.

Houve alguns limitantes nessa pesquisa em relação a medição de desempenho da renderização. São eles:

- Não foram feitos testes de desempenho para ver quantos elementos são possíveis renderizar em tempo desejável.
- Não foi feita análise de desempenho quanto ao uso de memória e cache.
- Não foi realizado análise de desempenho quanto ao tipo de dados empregado para representar as informações.
- Não foi utilizado a versão mais recente do Angular.

4.3 Resumo e discussão geral do capítulo

Foi apresentado nesse capítulo quais foram os resultados de cada funcionalidade juntamente com cenários possíveis de uso pelo usuário. Além disso, foi exposto também a discussão em torno de redução de carga no servidor CommonAPI, baixa latência ao mover o mapa e sem travamento na interface gráfica na funcionalidade de mapas e a importância de criação de gráficos personalizados. Além disso, foi exposto a importância da utilização do GIS para *smart grids*, bem como as limitações existentes nesse TCC.

5 Conclusão e trabalhos futuros

5.1 Conclusão

Este trabalho procurou produzir uma prova de conceito de um sistema *web* capaz de auxiliar no monitoramento de ativos de plantas de distribuição elétricas em ambiente *web* e padrão GIS com interface flexível e configurável pelo usuário. Isso tem potencial para que as tomadas de decisões sejam mais responsivas pelo fato de a visualização do mapa não comprometer na experiência do usuário. Além disso, a possibilidade de organizar a posição e tamanho dos gráficos e mapa na tela da maneira que o usuário desejar pode favorecer o uso desta proposta. Portanto, as vantagens que a arquitetura do sistema e funcionalidades apresentam diretamente para o usuário, servidor e desenvolvedor são:

- Flexibilidade ao manusear o sistema.
- Responsividade na tomada de decisões.
- Possibilidade de análises com base em curvas diferentes ou iguais.
- Diminuição de carga no servidor CommonAPI e consequentemente no MongoDB devido a implementação de cache para o cliente.
- Facilidade de executar manutenções e implementações novas no código do front-end e *back-end* por utilizar o modelo MVC.

Todavia, a conclusão mais proeminente produzida por este trabalho é que a associação de tecnologias aqui empregada permitiu a construção de uma tecnologia de visualização via *web-browser* em padrão GIS com até 1,1 milhões de ativos representados graficamente. Logo, a proposta é tecnicamente viável como alternativa ao clássico modelo de sistemas supervisórios usando representações por diagramas unifilares.

Não foi testado a renderização com mais ativos pelo fato da base de dados BDGD possuir somente até 1,1 milhões de ativos. Assim, se houvessem mais ativos disponíveis para a renderização seria possível realizar novos testes no sistema para aferir a sua responsividade.

O código da interface gráfica do MonitorIOT (EQUIPEPROJETOCEBIOT, 2022b) se encontra em um repositório do Bitbucket privado da UFU e é utilizado o Git para controle de versionamento. O mesmo caso é o *back-end* do MonitorIOT (EQUIPEPROJETOCEBIOT, 2022a) que se encontra também de maneira privada no Bitbucket.

5.2 Trabalhos futuros

Apesar de existir a funcionalidade de organizar os elementos da tela de maneira que desejar, é interessante persistir isso para cada conta de usuário, trazendo a capacidade de que cada um tem a sua própria organização da melhor e mais eficiente maneira para monitoramento.

Outra funcionalidade que pode ser necessária é a possibilidade de criar mais de um mapa com cada elemento diferente plotado. É interessante também ter a capacidade de instanciar uma tela a parte com a finalidade de inserir mais elementos e organizá-los em cada tela.

É interessante haver testes no sistema que medem o desempenho, consumo de memória, estresse e limite. Além disso, é importante estimar a latência de atualização da mudança de estado dos elementos até a renderização no mapa para o usuário.

Para que a biblioteca leaflet renderize os elementos de uma maneira mais rápida, é possível que seja modificado em como é calculado a geometria dos pontos e linhas. Portanto, é importante que seja simplificado a maneira que se é calculado a geometria, contribuindo, assim, para a redução do uso da CPU, a qual lida com cálculos complexos.

É importante sempre atualizar a *framework*, e bibliotecas Angular utilizado no projeto para uma versão mais recente e estável. Isso possibilita novos recursos e melhorias que podem afetar diretamente o MonitorIOT, mantendo-o atualizado e modernizado.

Referências

ABLY. *Server-Sent Events (SSE): A Conceptual Deep Dive*. 2022. Disponível em: <https://ably.com/topic/server-sent-events>. Citado na página 25.

AHMAD, T.; ZHANG, D. A critical review of comparative global historical energy consumption and future demand: The story told so far. *Energy Reports*, Elsevier, v. 6, p. 1973–1991, 2020. Citado 2 vezes nas páginas 16 e 17.

ANEEL. *Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional – PRODIST | Módulo 10 – Sistema de Informação Geográfica Regulatório*. 2021. Disponível em: <https://www2.aneel.gov.br/cedoc/aren2021937_prodist_modulo_10_v3.pdf>. Citado na página 28.

ANSARI, M. H.; VAKILI, V. T.; BAHRAK, B. Evaluation of big data frameworks for analysis of smart grids. *Journal of Big Data*, Springer, v. 6, n. 1, p. 1–14, 2019. Citado na página 23.

ASHTON, K. et al. That 'internet of things' thing. *RFID journal*, Hauppauge, New York, v. 22, n. 7, p. 97–114, 2009. Citado na página 13.

AZAR, J.-P. *Kafka Architecture*. 2022. Disponível em: <https://dzone.com/articles/kafka-architecture>. Citado na página 23.

BHATTARAI, B. P. et al. Big data analytics in smart grids: state-of-the-art, challenges, opportunities, and future directions. *IET Smart Grid*, Wiley Online Library, v. 2, n. 2, p. 141–154, 2019. Citado na página 21.

CHEN, X.; CHEN, X. Data visualization in smart grid and low-carbon energy systems: A review. *International Transactions on Electrical Energy Systems*, Wiley Online Library, v. 31, n. 7, p. e12889, 2021. Citado 2 vezes nas páginas 16 e 17.

DAKI, H. et al. Big data management in smart grid: concepts, requirements and implementation. *Journal of Big Data*, SpringerOpen, v. 4, n. 1, p. 1–19, 2017. Citado 2 vezes nas páginas 20 e 21.

EQUIPEPROJETOCEBIOT. *Código back-end MonitorIOT*. 2022. Disponível em: <https://bitbucket.org/equipeprojetocebiot/monitoriot_back/src/hml/>. Citado na página 58.

EQUIPEPROJETOCEBIOT. *Código interface gráfica MonitorIOT*. 2022. Disponível em: <https://bitbucket.org/equipeprojetocebiot/monitoriot_front/src/hml/>. Citado na página 58.

FERREIRA, D. de O. *Leaflet.glify Github*. 2020. Disponível em: <https://github.com/daniel-oliv/Leaflet.glify/>. Citado na página 30.

FERREIRA, D. de O. *Proposta de um Sistema de IoT para Redes de Distribuição: Entidades Virtuais, Visualização de Informação e Inteligência Artificial Aplicadas a Smart Grids*. 119 p. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Uberlândia, 2022. Citado na página 45.

GESTAL. *GESTAL - Gestão de Energia e Utilidades :: A GESTAL*. 2022. Disponível em: <https://www.gestal.com/gestal>. Citado na página 19.

GESTAL. *GESTAL - Gestão de Energia e Utilidades :: Smart Gate M*. 2022. Disponível em: <https://www.gestal.com/produtos/smart-gate-m>. Citado 2 vezes nas páginas 19 e 20.

GESTAL. *GESTAL - Gestão de Energia e Utilidades :: Smart Gate X*. 2022. Disponível em: <https://www.gestal.com/produtos/smart-gate-x>. Citado 2 vezes nas páginas 19 e 20.

GOOGLE. *Managing HTML5 Offline Storage*. 2015. Disponível em: <https://developer.chrome.com/docs/apps/offline_storage/>. Citado na página 41.

GOOGLE. *Conectar-se ao Google BigQuery - Ajuda do Looker Studio*. 2022. Disponível em: <https://support.google.com/looker-studio/answer/6370296?hl=pt-BR#zippy=%2Cneste-artigo>. Citado na página 17.

KAFKA, A. *Apache Kafka*. 2022. Disponível em: <https://kafka.apache.org/>. Citado na página 22.

KHRONOS. *WebGL Overview - The Khronos Group Inc*. 2022. Disponível em: <https://www.khronos.org/webgl/>. Citado na página 27.

LARAVEL-NEWS. *The WebSocket Handbook: learn about the technology behind the realtime web*. 2022. Disponível em: <https://laravel-news.com/websocket-handbook>. Citado 2 vezes nas páginas 25 e 26.

MOZILLA. *Javascript | MDN*. 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Citado na página 24.

MOZILLA. *Main thread - MDN Web Docs Glossary*. 2022. Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/Main_thread>. Citado na página 24.

MOZILLA. *Using server sent events*. 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/Server-sent_events/Using_server-sent_events>. Citado na página 24.

MOZILLA. *Using Web Workers*. 2022. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers>. Citado na página 37.

POWERBI, M. *Power BI Desktop*. 2022. Disponível em: <https://powerbi.microsoft.com/pt-br/desktop/>. Citado na página 18.

REDHAT. *What is Apache Kafka?* 2022. Disponível em: <https://www.redhat.com/en/topics/integration/what-is-apache-kafka>. Citado na página 22.

SAGIROGLU, S. et al. Big data issues in smart grid systems. In: IEEE. *2016 IEEE international conference on renewable energy research and applications (ICRERA)*. [S.l.], 2016. p. 1007–1012. Citado na página 23.

SANCHEZ-HIDALGO, M.-A.; CANO, M.-D. A survey on visual data representation for smart grids control and monitoring. *Sustainable Energy, Grids and Networks*, Elsevier, v. 16, p. 351–369, 2018. Citado 2 vezes nas páginas 16 e 21.

SOEWITO, B. et al. Websocket to support real time smart home applications. *Procedia Computer Science*, Elsevier, v. 157, p. 560–566, 2019. Citado na página 26.

TABLEAU. *Tableau Desktop*. 2022. Disponível em: <https://www.tableau.com/pt-br/products/desktop>. Citado na página 19.

TABLEAU. *Tableau Public*. 2022. Disponível em: <https://www.tableau.com/pt-br/products/public>. Citado na página 18.

TIBERIUZULD. *tiberiuzuId/angular-gridster2*. 2022. Disponível em: <https://github.com/tiberiuzuld/angular-gridster2>. Citado na página 30.

WAN, K. *Sending Type-safe HTTP Requests With Go*. 2022. Disponível em: <https://betterprogramming.pub/sending-type-safe-http-requests-with-go-eb5bd1f91558>. Citado na página 24.