DESENVOLVIMENTO DE UMA PLATAFORMA *WEB* PARA MONITORAMENTO DE CONSUMO DE ENERGIA ELÉTRICA NO SETOR RESIDENCIAL

Sophia Victória Santos
UFERSA
sophia.santos@alunos.ufersa.edu.br

Fabiana Karla de Oliveira Martins Varella
DET
UFERSA
fkv@ufersa.edu.br

Resumo — O presente trabalho tem como objetivo desenvolver um produto viável de uma plataforma web em Drupal para o monitoramento do consumo de energia elétrica no setor residencial. A partir dos conceitos de linguagem de programação, frameworks e os próprios conceitos de Drupal, construiu-se uma plataforma que permite aos usuários criarem pontos de medição que receberão dados de consumo de energia elétrica através de uma API[1] de forma que o usuário possa realizar o monitoramento do consumo individual e coletivo dos pontos de medição, identificando qual item gera maior custo na conta mensal. A criação da plataforma tem diversos pontos positivos, dentre eles a possiblidade de comparar o valor da tarifa de energia elétrica quando o usuário estiver na convencional ou na branca, além de permitir identificar quais os elementos que geram maior ou menor consumo energético, podendo também verificar os gastos com energia elétrica ao longo do período de um ano. Por fim, o trabalho mostra que a plataforma é bastante efetiva e viável, dando a possibilidade ao usuário em desempenhar as funcionalidades mais básicas de uma plataforma web.

Keywords — Eficiência energética, Consumo de energia elétrica, medição, Plataforma web

I. INTRODUÇÃO

A constante conectividade com a internet e compartilhamento de dados e informações, juntamente com os processos de automação, torna possível o uso mais eficiente de diversos equipamentos, seja para manter um local em uma temperatura específica com menor consumo de energia elétrica, seja para informar quando regar uma planta ou realizar qualquer outra atividade diária.

A multidisciplinaridade presente no conceito da Internet das Coisas (IOT) permite a sua implementação em diversos setores, seja no setor industrial, público ou privado, por exemplo [1]. A partir desse conceito de IOT, o processo de automação residencial se torna mais atrativo de implementar devido à sua possibilidade de tornar processos do dia a dia mais práticos e confortáveis, além de também possibilitar redução nos custos com energia elétrica ao longo do dia. Sua conectividade com a internet, permite maior controle das automações instaladas chamam a atenção e incentivam a aplicação desses processos de automação em residências.

Apesar da preocupação com a redução de consumo de energia elétrica, para que se tenha redução efetiva do valor gasto com a conta de energia elétrica é necessário ter o conhecimento mínimo sobre o gasto na fatura de energia elétrica dos equipamentos [2]. Logo, conhecer o consumo de energia elétrica da residência é um dos pontos mais importantes para proporcionar maior eficiência energética para o local e manter um melhor padrão de consumo.

Tendo em vista que o conhecimento do consumo elétrico de uma residência é um dos pontos mais importantes para poder traçar planos de eficiência energética, ainda faltam opções de plataforma de acesso gratuito ou de curso acessível que possua boa visualização dos dados de consumo de energia elétrica. Essa falta de opção e esse difícil acesso torna, muitas vezes, o processo de automação inviável, ou então, gera um grande esforço para se ter um melhor entendimento do uso residencial de energia elétrica.

A partir do exposto, o presente trabalho tem que como objetivo desenvolver um produto viável de uma plataforma *web* baseada em PHP para visualização e monitoramento de dados de consumo de energia elétrica para o setor residencial.

II. PLATAFORMAS DE MONITORAMENTO

Ao buscar por sistemas de monitoramento de consumo de energia elétrica é bastante comum encontrar projetos voltados para a confecção de *hardware* que coleta os dados de um sistema em específico e salva os dados para serem analisados em programas como *Excel* ou servidores *web* de coleta de dados como, por exemplo, no trabalho desenvolvido por [2], que não mostra muitos conceitos de programação relacionados à plataforma *web.* Em outros casos, em outras literaturas, é possível encontrar trabalhos voltados à salvar em uma página construída puramente em *html* como no trabalho de [3] que apresentam um nível baixo de escalabilidade[2].

Dessa forma, esta seção abordará os principais pontos para a confecção de uma plataforma *web* voltada ao monitoramento e análise de dados referentes ao consumo de energia elétrica, passando pelos conceitos de linguagens de programação e *framework*.

[1] Interface de Programação de Aplicação, uma interface utilizada para conexão entre dois serviços.

[2] Capacidade de um sistema crescer atendendo novas demandas sem necessidade de reestruturar suas funções.

A. Linguagens de programação

Linguagens e lógicas de programação são combinadas para criar diferentes formas de se comunicar com diferentes *hardwares* e então realizar diferentes ações desejadas. Apesar das características da lógica de programação não serem bastante mutáveis, as linguagens de programação são, variando de acordo com sua origem, os paradigmas adotados e os objetivos.

Assim como a humanidade utiliza das formas de linguagens para a comunicação entre diferentes indivíduos existem diferentes tipos de linguagens para a comunicação com e entre *hardwares*. A existência de diferentes significados para uma mesma frase, de acordo com a sintaxe e semântica utilizada, impede a utilização de linguagens como o português para a programação de computadores, sendo necessário um outro tipo que não possuam ambiguidades na sua interpretação [4].

A primeira linguagem de programação surgiu antes dos computadores modernos. Ada Lovelace seria então a primeira programadora da história, onde em 1843 adicionou notas, na tradução do artigo de Luigi Menabra, que descreviam comandos para realizar o cálculo dos números de Bernoulli, sendo esse considerado o primeiro *software* desenvolvido. Com a evolução dos *hardwares* linguagens como o *Assembly* e o FORTRAN passaram a serem utilizadas para a comunicação com as máquinas e repassar os passos a passos que elas devem realizar [5].

O FORTRAN passou a ser considerado, então, uma linguagem de alto nível, que diferentemente das linguagens de baixo nível como o *Assembly* possuem uma maior rapidez no processamento e necessita de menores quantidades de linhas de comando para realizar a mesma ação. A partir das linguagens de alto nível tornou-se possível a definição dos paradigmas que deram origem as linguagens de programação mais avançadas e poderosas.

Por volta de 1960 e 1970 o paradigma que começou a ser amplamente utilizado foi o paradigma de linguagem estruturada que consiste no uso de rotinas e sub-rotinas para estruturar o código [4]. Diferentemente dos métodos de programação utilizados anteriormente, o paradigma estruturado tem como característica dividir o código em diferentes módulos, onde cada módulo é responsável por uma ação específica dentro do código, permitindo uma legibilidade e compreensão muito maior do código escrito, surgindo então o conceito de *clean code* que busca tornar os códigos mais fáceis de entender apenas pela leitura dele. Para a modularização dos códigos são utilizadas funções autorais (sub-rotinas) totalmente separadas do código principal, dessa forma essas funções são chamadas no código principal e passam a realizar a ação determinada.

Por volta de 1980, outro paradigma passou a ser mais difundido, sendo esse a programação orientada a objetos que consiste na interação de diferentes unidades de códigos chamada objeto [4]. A orientação a objetos tenta modelar o máximo possível o código como o mundo real, através de objetos, atributos e ações, cada objeto pode ser considerado como uma criatura diferente que possui características diferentes e realizam ações diferentes. Por exemplo, um objeto poderia ser um gato que possui a característica de tom de pelagem preta e realiza a ações de falar miando, ou então um carro que possui como característica ser da marca Ford e realiza a ação de se locomover.

Todo o conceito de programação orientada a objetos promove uma maior abstração dos conceitos aplicados na escrita de códigos e consequentemente apresenta mais fatores de segurança devido à sua complexidade e aos conceitos de encapsulamento, herança, polimorfismo e interfaces. A partir dos paradigmas surgem linguagens mundialmente utilizadas como Java, JavaScript, Python e PHP. Neste trabalho será tratado mais especificamente sobre o PHP, pois foi a linguagem de programação selecionada para ser utilizada. PHP.

B. PHP

PHP: *Hypertext Prepocessor* é uma linguagem de programação utilizada mundialmente e *open-source*, ou seja, aberta a comunidade e evolui com auxílio dela. De acordo com uma pesquisa realizada pela página *stackoverflow*, comumente utilizada pela comunidade de programadores e estudantes, o PHP é a décima-primeira linguagem mais utilizada, sem distinção de foco de uso em *back* ou *front-end* [6].

O PHP é uma linguagem de programação focada em produtos *web* e apesar das críticas que sofreu quanto a inadequação para manutenção de *softwares* de alta escala, a evolução do PHP que trouxe consigo o paradigma da programação orientada a objetos, uso de bibliotecas e aumento da complexidade em conjunto com a segurança foram aos poucos solucionando alguns problemas e permitindo a linguagem a continuar sendo utilizada em grande escala [7].

Como explicitado, o PHP é uma linguagem que sofreu grandes alterações ao longo dos anos e atualmente está na versão 8.1, porém seu início se deu em 1994 por Ramus Lerdorf, formado por conjuntos de *scripts* escritos na linguagem C que o criador utilizava para verificar os acessos ao seu currículo online [8]. Em 1995 o código fonte se tornou *open-source* e com a ajuda de outros desenvolvedores que se juntaram a comunidade foi lançado em 1997 a segunda versão do PHP. Até que na 5ª versão do PHP, publicada em 2004, foram adicionados os conceitos de orientação a objetos e passou a se consolidar cada vez mais [8].

Ao analisar diferentes linguagens de programação é possível entender alguns motivos do PHP ser bastante utilizado. Entre esses motivos se percebe a facilidade de se aprender a linguagem devido à ausência da necessidade de declarar o tipo das variáveis, chamadas de funções mais simplificadas e uma menor necessidade de memória para processamento.

Apesar de ser possível programar apenas com a linguagem de programação, a utilização de *frameworks* facilita e proporciona novas características para o processo de programação. No caso do PHP é possível encontrar diferentes *frameworks* como Laravel, Drupal e Symfony, cada um possui um foco e abstrações diferentes, porém todos buscam trazer uma maior facilidade no processo de desenvolvimento através de funções nativas que reduzem o esforço do sistema, aumentar a segurança através das abstrações, entre outras peculiaridades de cada um. Na subseção "*C*" será tratado especificamente sobre o *framework* Drupal, que foi selecionado para ser utilizado neste trabalho.

C. Drupal

Utilizando o PHP como base, o Drupal é um *framework open-source* criado em 2000 por dois estudantes da

universidade de Antwerp, Dries Buytaert e Hans Snijder, que participavam de um grupo de oito estudantes e buscavam uma forma de melhorar a comunicação entre si dentro dos dormitórios da universidade. Surgindo assim um *site* de notícias onde cada um poderia compartilhar o que estava fazendo, coisas que gostavam e outros assuntos, até que no início de 2001, foi liberado o código por trás dessa plataforma e se inicia o processo de evolução do framework Drupal [9].

Atualmente o Drupal conta com mais de um milhão de colaboradores na comunidade, sendo esses desde desenvolvedores, designers até mesmo patrocinadores e estrategistas que trabalham junto para criar funcionalidades, documentações, dar suporte e muitas outras ações que trazem evolução para a comunidade, estando atualmente na versão 9 e com previsão para lançamento da versão 10 do Drupal até o final do ano de 2022 [9].

Além das características naturais de um *framework* o Drupal apresenta um sistema de modularidade e conta com uma grande flexibilidade que permitem construir diferentes tipos de *web sites* facilmente com alta taxa de escalabilidade e gerenciamento de *softwares* [10]. Essas características também aumentam seus níveis de segurança, dando uma maior confiabilidade para plataformas feitas a partir do Drupal, vindo a ser utilizado por grandes empresas como a Tesla, Johnson & Johnson, por ex.

Outros pontos atrativos para o uso do Drupal é o core do *framework* ser desenvolvido contendo ferramentas de acessibilidade e tradução de texto em diversas linguagens. Dessa forma, qualquer plataforma construída a partir do Drupal automaticamente já dispõe dessas ferramentas, agregando mais valor ao produto.

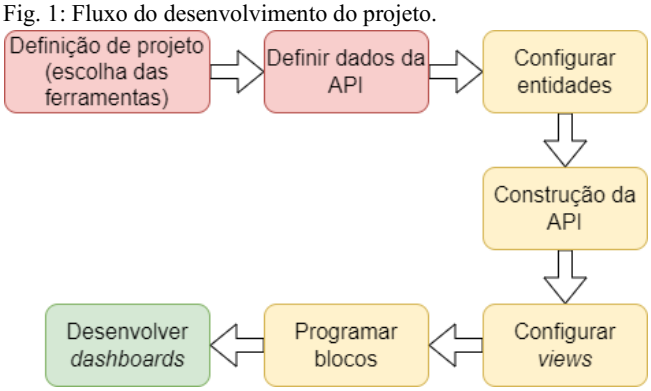
Devido ao Drupal ser um *framework open-source* com bastante contribuição da comunidade e apresentar um esquema de modularidade, é possível construir plataformas completas apenas utilizando códigos já desenvolvidos por outra pessoa, sem a necessidade de realizar configurações muito complexas e apenas realizando o *download* dos módulos e temas disponibilizados na página oficial do Drupal. A facilidade de construir uma página com pouco conhecimento de desenvolvimento *web* torna-se um atrativo para iniciantes.

Apesar de ser possível construir diversas coisas diferentes apenas com os módulos e temas que os autores disponibilizam na página oficial do Drupal, plataformas mais complexas e estilizadas necessitam que o desenvolvedor compreenda como criar módulos e temas, ou seja, como criar funcionalidades e diferentes *layouts*, respectivamente. Desta forma, atendendo às especificidades dos objetivos da plataforma.

III. Metodologia

A presente seção tem como objetivo descrever os passos e etapas necessários para programar e configurar uma plataforma web, em Drupal, de monitoramento do consumo de energia elétrica de uma ou mais residências. Para isso a metodologia será detalhada conforme apresentado na Fig. 1, seguindo pelos passos de definição de projeto programação e visualização do usuário, representados, respectivamente, pelas cores vermelho, laranja e verde.

Fig. 1: Fluxo do desenvolvimento do projeto.



Fonte: Autoria Própria.

Para a criação de uma plataforma web que tenha como funcionalidades a realização do registro de consumo de energia elétrica de uma unidade consumidora e monitoramento desse consumo ao longo dos dias e meses é necessário, primeiramente, determinar algumas características do projeto para então determinar como serão programadas as funcionalidades em específico.

Com as especificações do projeto definidas escolheu-se o PHP como linguagem de programação, utilizando o *framework* Drupal que permite a criação de projetos escaláveis, não requerendo pagamento de licença para uso e mesmo assim garantindo maior segurança aos dados armazenados no sistema. Além de o Drupal possuir uma maior facilidade em definir e manusear entidades e realizar conexões com banco de dados, essas características o tornam um *framework* adequado para se trabalhar tendo em vista que as funcionalidades da plataforma envolvem trabalhar com grandes quantidades de dados.

Apesar do projeto ter como objetivo a criação de um produto viável da plataforma de monitoramento, ainda se faz necessário que esta se mantenha *online* a todo momento. Para isso, é necessário utilizar um servidor para hospedar o projeto, garantindo que qualquer usuário possa utilizar seus serviços sempre que necessitar. Nesse caso, escolheu-se utilizar o serviço de hospedagem Phanteon.io [11] que dispõe de dois ambientes gratuitos *sandbox* para usuários conforme relatado na própria página do serviço.

Definidos a linguagem de programação, *framework* e serviço de hospedagem é possível iniciar efetivamente o processo de programação. Sendo assim, esta seção será dividida em duas etapas, de forma que a primeira situará a configuração do ambiente e a construção da API para aquisição de dados de medição de consumo de energia elétrica, e a segunda etapa tratará da construção do monitoramento que o usuário poderá acessar para verificar seu consumo.

A. *Aquisição de dados*

Ao se trabalhar com aquisição de dados o primeiro passo é determinar quais serão os dados que deverão ser armazenados no sistema, de forma que apenas os dados necessários sejam solicitados ao usuário, visando garantir maior eficácia ao sistema; um banco de dados menor; e a plena conformidade com as diretrizes da Lei Geral de Proteção de Dados (LGPD), que se refere ao tratamento de dados pessoais, dispostos em meio físico ou digital, feito por pessoa física ou jurídica [12].

Para a aquisição dos dados de medição do consumo de energia elétrica do usuário escolheu-se salvar as seguintes informações:

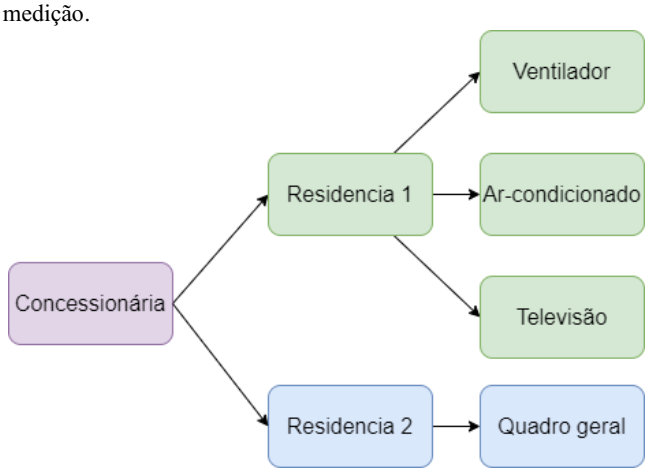
- uid – id do usuário.
- measuring_point_id – id do ponto de medição.
- voltage – tensão do sistema.
- current – A corrente medida pelo sistema.
- insert_date – data de envio dos dados.

Dos dados de medição que serão inseridos no sistema, apenas a data de envio dos dados não deve ser enviada pelo usuário, sendo esse gerado pelo próprio sistema no momento de salvar as informações no banco de dados. O "id do ponto de medição" se faz necessário para indicar qual o ponto de medição que consumiu uma determinada quantidade de potência, enquanto o "id do usuário" age como uma segunda verificação, garantindo que é realmente o ponto de medição do usuário. A tensão e a corrente são utilizadas para calcular a potência de consumo do ponto de medição ou de cada ponto de medição.

Com os dados a serem armazenados definidos e a instância do Drupal criada no Phanteon, iniciou-se as configurações do ambiente para permitir a obtenção dos dados. Como o Drupal já possui internamente um sistema de cadastro de usuários pré-definido que atende a todas as necessidades do projeto (apenas ser inserido um e-mail, *login* e senha para o usuário) não se faz necessário modificar essas configurações, sendo o próximo passo possibilitar a criação dos pontos de medições.

Para desenvolvimento deste projeto os pontos de medição são definidos como os locais dentro de uma residência onde o usuário deseja realizar o monitoramento do consumo de energia elétrica, como por exemplo, um aparelho de ar-condicionado, um ventilador ou até mesmo uma seção de um disjuntor. Como existem diversas concessionárias de energia elétrica no Brasil, a unidade consumidora, no caso, residência deve estar vinculada a uma concessionária de forma que todos os pontos de medição desse domicílio sejam cobrados de acordo com as taxas referentes à concessionária específica. É possível observar a relação adotada na Fig. 2.

Fig. 2: Relação entre concessionárias, residências e pontos de medição.



Fonte: Autoria Própria.

Com as três entidades que serão trabalhadas na plataforma definidas iniciou-se a configuração das concessionárias. Nesta etapa foram definidos os campos:

- Título.
- Tarifa de ponta.
- Tarifa intermediária.
- Horário final de pico.
- Horário final fora de pico 1.
- Horário final fora de pico 2.
- Tarifa convencional.
- Tarifa fora de ponta.
- Horário inicial de pico.
- Horário inicial fora de pico 1.
- Horário inicial fora de pico 2.

Com os campos definidos e a entidade criada é possível criar concessionárias com seus respectivos valores cobrados de taxa e horários para cada zona.

O próximo passo deu-se pela configuração das residências. Como as residências possuem como funcionalidade facilitar a divisão de pontos de medição entre diferentes residências para o mesmo usuário e definir qual concessionária estará vinculada a seus pontos, definiu-se seus campos como:

- Título.
- Tipo de taxa.
- Concessionária.
- Logo.

A "logo" tem como objetivo facilitar a visualização ao buscar todas as residências cadastradas e o tipo de taxa define se a taxa padrão para a residência é convencional ou taxa branca, ou seja, se está de acordo com a modalidade convencional ou a modalidade tarifa branca. Para a última entidade definiu-se seus campos como:

- Título.
- Residência.
- Logo.
- Tipo de medição.

O "tipo de medição" indica se este será de geração ou consumo, enquanto a "logo" tem como objetivo facilitar a visualização do ponto de medição ao buscar todos os pontos de medição de uma residência.

Com as entidades de residência e pontos de medição criadas é possível o usuário criar sozinho os equipamentos relacionados a cada uma dessas entidades.

Com os pontos finalizados deu-se então início a construção da API para que o usuário consiga inserir os dados no sistema. O primeiro passo para a construção da API é a inserção do módulo da comunidade "restui", que cria uma interface de usuário para gerenciar as API's definindo se estarão ativas ou inativas. Em seguida, criou-se um módulo customizado nomeado como "eletricapi" para realizar as configurações da API do tipo POST e salvar os dados fornecidos quando um usuário enviar.

Ao final da construção da API, para a realização de testes, utilizou-se a plataforma POSTMAN para enviar dados aleatórios para o *endpoint* do tipo POST e salvar estes valores no banco de dados, tanto para testes da própria API quanto dos dados que serão mostrados aos usuários finais.

B. *Monitoramento*

Para o monitoramento do consumo de energia elétrica utilizou-se a estrutura de blocos do Drupal para criação de diferentes blocos que representam os dados a serem monitorados para, em seguida, adicionar os blocos nas páginas de monitoramento.

Parar criar blocos customizados que mostrem o valor monetário gasto com energia elétrica de acordo com o usuário que esteja visualizando a página e os dados que ele inseriu no sistema através da API, é necessário criar um módulo customizado. Para isso, foram criados todos os arquivos necessários para um módulo customizado ser aceito pelo Drupal e um arquivo diferente para cada tipo de bloco dentro da pasta Block no módulo.

Para alcançar e desenvolver um produto viável, criou-se sete blocos diferentes, sendo cinco para mostrar o consumo atual de energia elétrica no dia e mês em que a página está sendo visualizada, e um outro bloco para mostrar a tarifa ao longo dos meses do ano atual. Dividiu-se os primeiros cinco blocos entre:

- Consumo no dia para tarifa convencional.
- Consumo no dia para tarifa branca.
- Consumo no mês para tarifa convencional.
- Consumo no mês para tarifa branca.
- Consumo no mês para cada ponto de medição.

Apesar de serem blocos distintos todos possuem o mesmo padrão de busca de dados, mudando apenas algumas condições específicas como data e campo que será analisado para mostrar na tela do usuário.

Para mostrar os dados para o usuário configurou-se os blocos para buscarem todos os dados de medições inseridos no banco de dados, em seguida somar todas os potenciais medidos, dividindo por 60, devido a ser armazenado 1 registro de consumo a cada minuto, totalizando 60 em uma hora, e multiplicando pela tarifa adotada, depois divide o resultado por 1000 para representar o kWh. Dessa forma o sistema mostra o total gasto em kWh de acordo com a Equação (1).

$$Total\ tarifa = \frac{\frac{V \cdot I}{60} Tarifa}{1000} \quad (1)$$

Com os blocos criados definiu-se as páginas que seriam disponibilizadas para o usuário.

Iniciou-se o processo com a criação da *view* das residências cadastradas para os clientes, de acordo com o usuário que abre a página será mostrado o nome da logo de todas as residências cadastradas pelo usuário. Em seguida construiu-se a *view* dos pontos de medição que, assim como a anterior, mostra conteúdos diferentes dependendo de qual usuário está acessando à página, mostrando todos os pontos de medição criados pelo usuário, cada um com o título, logo, id do usuário e id do ponto de medição. Como os dois últimos campos são necessários para o envio dos dados de medição através da API fez-se necessário deixá-los facilmente a mostra para o usuário.

Com ambas as *views* criadas, através das configurações de menu dentro das configurações do Drupal, inseriu-se dois *links* de menu para facilitar o acesso dos usuários às páginas de *views* e de criação de residências e pontos de medição.

Para as páginas de visualizações de consumos de energia elétrica utilizou-se o módulo da comunidade *dashboards* que traz consigo estruturas para customização de páginas de acordo com o modelo desejado, permitindo inserir blocos lado a lado e um embaixo do outro da forma como preferir.

Criou-se então duas *dashboards* de blocos para facilitar a visualização do usuário. Um *dashboard* voltado para análise do consumo de energia elétrica no dia e mês atual, mostrando a comparação dos valores para tarifa convencional e tarifa branca, e o outro *dashboard* com o objetivo de mostrar o consumo para tarifa convencional e para a tarifa branca de cada mês do ano vigente.

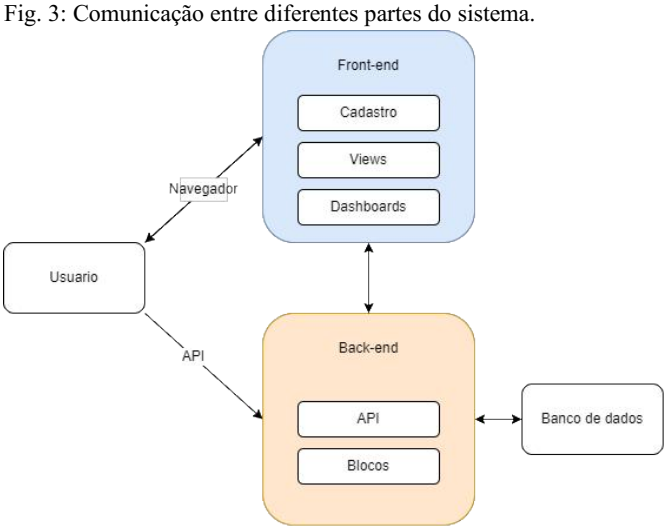
IV. RESULTADOS

Finalizada a etapa da metodologia que descreve os passos, definições e serviços necessários para desenvolver uma plataforma *web*, a presente seção demonstrará as páginas e códigos desenvolvidos no Projeto.

Com o uso das ferramentas para programação em conjunto com a hospedagem do servidor através dos serviços do Phanteon.io, tornou-se possível construir a plataforma *web eletricview* [13].

A presente seção será dividida em duas subseções, sendo a primeira referente ao *front-end* do sistema, parte visual da plataforma, enquanto a segunda o *back-end*, que demonstra os principais códigos necessários para que o usuário consiga visualizar as informações nas páginas do sistema. A comunicação entre ambas as partes do sistema pode ser visualizada na Fig. 3, onde o usuário acessa o *front-end* pelo navegador e a partir do acesso ele se comunica com o *back-end* solicitando os dados para mostrar as informações armazenadas no banco de dados.

Fig. 3: Comunicação entre diferentes partes do sistema.



Fonte: Autoria Própria.

A. *Front-End*

Qualquer usuário do sistema tem acesso à um conjunto de páginas que o permita entender melhor o sistema; seu cadastro; o cadastra dos pontos de medição e a verificação do conjunto como um todo. Dessa forma, o usuário final tem acesso às páginas abaixo descritas:

- Inicial.
- Cadastro.

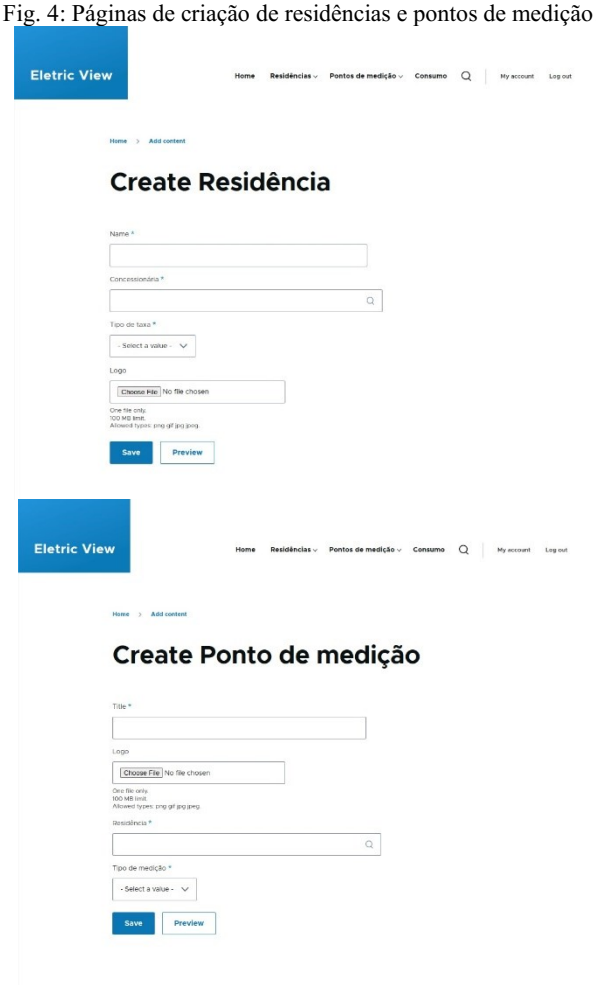
- Login.
- Cadastro de pontos de medição.
- *View* de pontos de medição.
- *Dashboard* consumo ao longo dos meses.
- Cadastro de residência
- *View* de residências.
- *Dashboard* de consumo atual dia e mês.

Além das páginas acessíveis para usuários finais, os usuários administradores conseguem também visualizar as páginas de configurações, cadastrar e editar concessionárias, criar *views* e *dashboards*, entre outras especificações do sistema.

Dentre as telas disponíveis para o usuário as principais são as telas de cadastro e *view* de residências e pontos de medição e as telas referentes as *dashboards* para o monitoramento do gasto com energia elétrica.

As telas de "cadastro" são formulários que o usuário preenche e assim consegue registrar no banco de dados uma nova entidade relacionada ao usuário que cadastrou, permitindo assim utilizar a API para envio dos dados de medição. Os formulários podem ser observados na Fig. 4.

Fig. 4: Páginas de criação de residências e pontos de medição.



Fonte: Print tela desenvolvida.

Como é possível observar na Fig. 4, em ambos os formulários, apenas o campo de imagem não é obrigatório para registrar a nova entidade, sendo os outros campos como concessionária e residência obrigatórios devido à necessidade de identificar os valores cobrados das tarifas de energia elétrica. O campo de "logo" não se faz obrigatório, pois apenas é utilizado para facilitar a visualização da entidade nas *Views* apresentadas na Fig. 5 e Fig. 6.

Fig. 5: *View* de residências (residência sem logo).

Fonte: Print tela desenvolvida.

Na *view* as residências são apresentadas em um esquema de grade com quatro colunas, o mesmo padrão utilizado para a *view* de pontos de medições, que pode ser observado na Fig. 6, assim como pode ser observado a diferença na visualização ao se ter imagens vinculadas as entidades.

Fig. 6: *View* de pontos de medição (pontos com logo).

Fonte: Print tela desenvolvida.

Como não há predefinições de "logos" para o usuário escolher, ele pode utilizar as imagens que sejam mais adequadas a identificar cada ponto de medição. Um ponto de diferença que é possível observar na Fig. 6 é a presença dos campos "uid" e "measuring_point_id", cada um mostrando um valor para cada ponto de medição. Esses pontos são mostrados devido à obrigatoriedade do envio deles na API para o registro do dado de medição. Dessa forma, para que o usuário não fique confuso a respeito de qual valor deve ser enviado para ambos os campos, os dados são mostrados na *view* logo abaixo da logo da respectiva entidade.

A partir do *link* de menu consumo, o usuário consegue acessar as *dashboards*, e ao clicar no nome consumo ele é redirecionado à página de consulta do gasto com o consumo de energia elétrica para valores de referência da tarifa convencional e da tarifa branca, como observado na Fig. 7.

Fig. 7: *Dashboard* de consumo de energia atual.

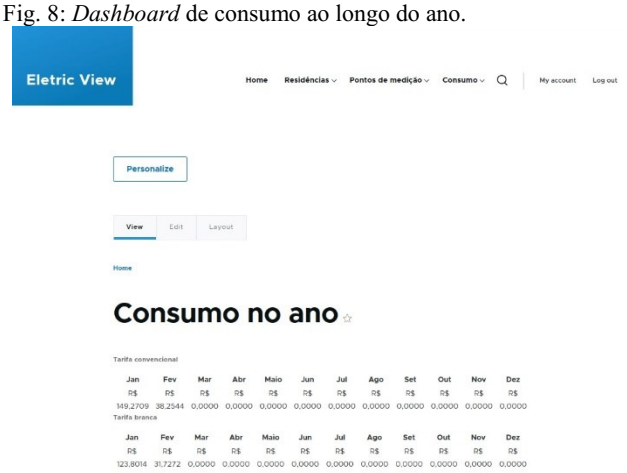
Fonte: Print tela desenvolvida.

A *dashboard* é dividida em duas seções, e a primeira é um comparativo entre o gasto monetário atual no mês e no dia para a tarifa convencional e tarifa branca, permitindo ao usuário comparar qual a melhor tarifa para ser aderida. Para os valores aleatórios utilizados, a tarifa branca demonstra um gasto menor para o usuário, compensando, então, a troca da tarifa.

A segunda seção é um descritivo do gasto de cada ponto de medição no mês atual, no exemplo apresentado na Fig. 7 com os dados salvos através das requisições do POSTMAN, o equipamento de maior consumo no período seria o ar-condicionado com um consumo equivalente à R$ 16,30, enquanto o menor teria sido o consumo da televisão com um gasto de aproximadamente R$ 7,90.

A segunda *dashboard*, referente ao consumo ao longo do ano, pode ser observada na Fig. 8.

Fig. 8: *Dashboard* de consumo ao longo do ano.



Fonte: Print tela desenvolvida.

A *dashboard* apresentada na Fig. 8 tem como objetivo analisar e comparar o consumo total ao longo dos meses do ano. Desta forma, conseguindo verificar quais os meses de maior consumo e quais os meses de menor consumo, permitindo traçar estratégias de redução de gastos e ter uma melhor análise do padrão de consumo do usuário.

A análise comparativa entre a tarifa branca e a tarifa convencional, levando em consideração a de menor gasto ao longo dos meses, possibilita uma avaliação mais abrangente acerca do impacto financeiro de cada modalidade tarifária no consumo de energia elétrica residencial.

À medida que o usuário realiza o envio contínuo dos dados para a plataforma, o sistema salva esses valores, proporcionando uma visualização mais precisa do consumo atual de energia elétrica da residência.

B. *Back-End*

No que se refere à estrutura interna do sistema, ou seja, o conjunto de códigos responsáveis pelo armazenamento, cálculo e exibição dos dados ao usuário, as partes mais relevantes do projeto podem ser divididas em dois módulos distintos. Sendo o primeiro a parte que se encarrega da implementação da API e o segundo é responsável pela criação e configuração dos blocos.

Para estabelecer a comunicação com o sistema é necessário utilizar o *endpoint*

/eletric_rest_api/eletric_resource da API, neste é possível acessar duas funções disponíveis da classe, sendo uma responsável por enviar dados (GET) e outra por salvar dados (POST). Tal implementação é ilustrada na Fig. 9.

Fig. 9: Funções da classe da API

```
public function get() {
  $select = Database::getConnection()->select('eletricapi');
  $select->fields('eletricapi');
  $results = $select->execute()->fetch();
  $response = json_encode($results);
  return new ResourceResponse($response);
}

/**
 * Responds to entity POST requests.
 *
 * @return \Drupal\rest\ResourceResponse
 *   Return an message.
 */
public function post(Request $request) {
  $data = json_decode($request->getContent());

  $measuring_point = \Drupal::entityTypeManager()->getStorage('node')->load($data->measuring_point_id);
  $measuring_point_uid = $measuring_point->get('uid')->getValue()[0]['value'];
  if ($measuring_point_uid <> $data->uid) {
    $response = ['message' => 'Wrong uid'];
    return new ResourceResponse($response);
  }

  $insert = Database::getConnection()->insert('eletricapi')
    ->fields(['uid', 'measuring_point_id', 'voltage', 'current', 'insert_date'])
    ->values([
      'uid' => $data->uid,
      'measuring_point_id' => $data->measuring_point_id,
      'voltage' => $data->voltage,
      'current' => $data->current,
      'insert_date' => date("Y-m-d H:i:s", time()),
    ]);
  $insert->execute();

  $response = ['message' => 'Finished'];
  return new ResourceResponse($response);
}
```

Fonte: Print tela desenvolvida.

Quando um usuário realiza uma requisição à função *GET*, retorna um valor salvo no banco, e quando a requisição é feita à função *POST*, ele deve enviar os dados da função dentro do corpo da requisição em formato JSON, contendo os campos “uid”, “measuring_point_id”, “voltage” e “current”. Os campos “uid” e “measuring_point_id” são os mesmos verificados na *view* de pontos de medição, enquanto “voltage” é a tensão do sistema (220 ou 110V no Brasil) e “current”, a corrente medida.

Ao receber uma requisição do tipo *POST* com todos os campos necessários dentro do corpo da requisição, o sistema inicia uma conexão com o banco de dados e salva os valores enviados junto à data atual.

Quanto ao módulo responsável por criar os blocos customizados, gerou-se sete arquivos, um responsável por cada uma das funcionalidades:

- Consumo no dia para convencional.
- Consumo no dia para tarifa branca.
- Consumo no mês de cada ponto de medição.
- Consumo total ao longo de cada mês do ano com tarifa branca.
- Consumo no mês para tarifa convencional.
- Consumo no mês para tarifa branca.
- Consumo total ao longo de cada mês do ano comtarifa convencional.

A divisão das funcionalidades em diferentes blocos, quando em comparação a ter um único bloco com todas as funcionalidades, proporciona uma maior facilidade de manutenção dos códigos e maior escalabilidade para o sistema, permitindo acrescentar novas funcionalidades para páginas já existentes ou novas. Diferentes tipos de *dashboards* podem ser criados com diferentes combinações de blocos, além da facilidade de inserir os blocos já criados em outras partes do sistema, como por exemplo a página inicial.

A programação dos blocos de consumo de energia elétrica no dia e no mês tanto para tarifa convencional quanto para tarifa branca segue o mesmo modelo de código, mudando apenas algumas especificações para o objetivo final do bloco. O código referente ao consumo no dia para tarifa convencional pode ser observado na Fig. 10.

Fig. 10: Bloco consumo no dia - tarifa convencional.

```php
/**
 * {@inheritdoc}
 */
public function build() {
  $uid = \Drupal::currentUser()->id();
  $node = \Drupal::entityTypeManager()->getStorage('node')->load(1);
  $conventional_tax = $node->get('field_conventional_tax')->getValue()[0]['value'];
  $date = date('Y-m-d');

  $select = Database::getConnection()->select('eletricapi');
  $select->fields('eletricapi', ['voltage', 'current']);
  $select->condition('eletricapi.uid', $uid, '=');
  $select->condition('eletricapi.insert_date', $date, '>=');
  $results = $select->execute()->fetchAll();

  $potency = 0;
  foreach ($results as $result) {
    $potency += ($result->voltage * $result->current);
  }

  $value = (($potency / 60) * $conventional_tax) / 1000;

  $block['convenc'] = [
    '#markup' => $value,
    '#cache' => ['max-age' => 0],
  ];

  return $block;
}
```

Fonte: Print tela desenvolvida.

A função "*build*" é responsável por construir o bloco e calcular os dados que serão exibidos nele. No caso para a verificação do consumo atual no dia, o sistema busca no banco de dados todos os dados referentes ao consumo que possuam o campo "uid" igual ao "id do usuário" que está acessando a página e tenham como *insert_date* o dia atual. Com os dados, o sistema soma todos os valores e mostra em tela, definindo um cache com tempo de vida zerado para que sempre que a página seja atualizada o sistema mostre os valores mais atuais.

No bloco de consumo no dia para tarifa branca a mesma base do bloco para tarifa convencional é utilizada, porém como a tarifa cobrada varia de acordo com o horário que está sendo consumida a energia elétrica (cobranças na ponta, intermediário e fora de ponta) o sistema verifica o horário que o consumo foi medido e multiplica pela tarifa referente ao horário, somando no final os valores e apresentando em tela com tempo de vida do cache zerado. As diferenças para tarifa branca podem ser observadas na Fig. 11.

Fig.11: Bloco de consumo no dia - tarifa branca

```php
public function build() {
  $uid = \Drupal::currentUser()->id();
  $node = \Drupal::entityTypeManager()->getStorage('node')->load(1);
  $off_peak_tax = $node->get('field_off_peak_tax')->getValue()[0]['value'];
  $intermediary_tax = $node->get('field_intermediary_tax')->getValue()[0]['value'];
  $peak_tax = $node->get('field_peak_tax')->getValue()[0]['value'];
  $date = date('Y-m-d');

  $select = Database::getConnection()->select('eletricapi');
  $select->fields('eletricapi', ['voltage', 'current', 'insert_date']);
  $select->condition('eletricapi.uid', $uid, '=');
  $select->condition('eletricapi.insert_date', $date, '>=');
  $results = $select->execute()->fetchAll();

  $potency = 0;
  foreach ($results as $result) {
    $hour = substr($result->insert_date, 11, 2);
    if ($hour >= 0 && $hour <= 15) {
      $potency += ($result->voltage * $result->current) * $off_peak_tax;
    }
    elseif ($hour >= 16 && $hour <= 17) {
      $potency += ($result->voltage * $result->current) * $intermediary_tax;
    }
    elseif ($hour >= 18 && $hour <= 20) {
      $potency += ($result->voltage * $result->current) * $peak_tax;
    }
    elseif ($hour >= 21 && $hour <= 23) {
      $potency += ($result->voltage * $result->current) * $off_peak_tax;
    }
  }

  $value = ($potency / 60) / 1000;

  $block['convenc'] = [
    '#markup' => $value,
    '#cache' => ['max-age' => 0],
  ];

  return $block;
}
```

Fonte: Print tela desenvolvida.

Para os blocos de consumo no mês, a única diferença em relação aos de consumo no dia é que a data é definida como o primeiro dia do mês, buscando no banco todos os registros com o campo *"insert_date*" maior que o primeiro dia do mês.

O bloco referente ao consumo no mês de cada ponto de medição se diferencia dos outros blocos por também buscar no banco o "id do ponto de medição", buscando em seguida o nome da entidade que possui o mesmo "id". Com base no nome do ponto de medição o sistema agrupa os valores calculados de potência e monta um vetor onde a chave do elemento é o nome do ponto de medição e o valor é o somatório de todos os valores. Com o vetor criado esses valores são repassados para um código de estilização que os posiciona em formato de tabela. O código referente ao bloco pode ser observado na Fig. 12.

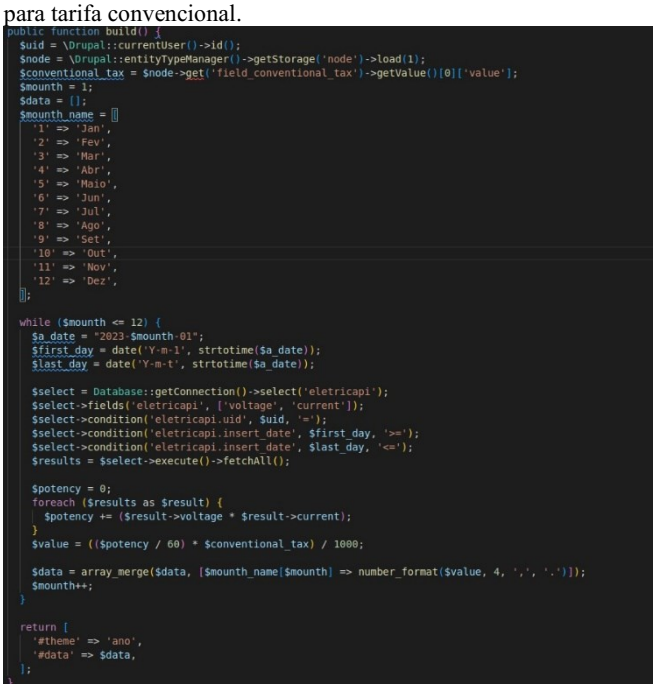
Fig.12: Bloco de medição de consumo no mês por ponto de medição

```php
/**
 * {@inheritdoc}
 */
public function build() {
  $uid = \Drupal::currentUser()->id();
  $node = \Drupal::entityTypeManager()->getStorage('node')->load(1);
  $conventional_tax = $node->get('field_conventional_tax')->getValue()[0]['value'];
  $date = date('Y-m-1');

  $select = Database::getConnection()->select('eletricapi');
  $select->fields('eletricapi', ['measuring_point_id', 'voltage', 'current']);
  $select->condition('eletricapi.uid', $uid, '=');
  $select->condition('eletricapi.insert_date', $date, '>=');
  $results = $select->execute()->fetchAll();

  $data = [];
  foreach ($results as $result) {
    $potency = ($result->voltage * $result->current);
    $value = (($potency / 60) * $conventional_tax) / 1000;
    $measuring_point = \Drupal::entityTypeManager()->getStorage('node')->load($result->measuring_point_id);
    $measuring_point_name = $measuring_point->get('title')->getValue()[0]['value'];

    if (isset($data[$measuring_point_name])) {
      $data[$measuring_point_name] += $value;
    }
    else {
      $data = array_merge($data, [$measuring_point_name => number_format($value, 4, ',', '.')]);
    }
  }

  return [
    '#theme' => 'hello_block',
    '#data' => $data,
  ];
}
```

Fonte: Print tela desenvolvida.

Para o penúltimo bloco criado, tem-se a especificação de que deve ser apresentado o gasto monetário com a energia elétrica para cada mês do ano, considerando a tarifa convencional. Este bloco se baseia no modo de separação de valores por nome do ponto de medição apresentado na Fig. 12, porém ao invés de separar pelo nome, o sistema realiza a separação através do mês da data presente no campo "*insert_date*" fazendo uma consulta no banco de todos os registros feitos por aquele usuário entre o primeiro e último dia de cada mês, salvando os valores em um vetor onde a chave de cada elemento é o nome do mês. Após o vetor ser montado, seus elementos são passados para um código que os dispõe em formato de tabela, como pode ser observado na Fig. 13.

Fig. 13: Bloco de gasto com energia elétrica em cada mês do ano para tarifa convencional.

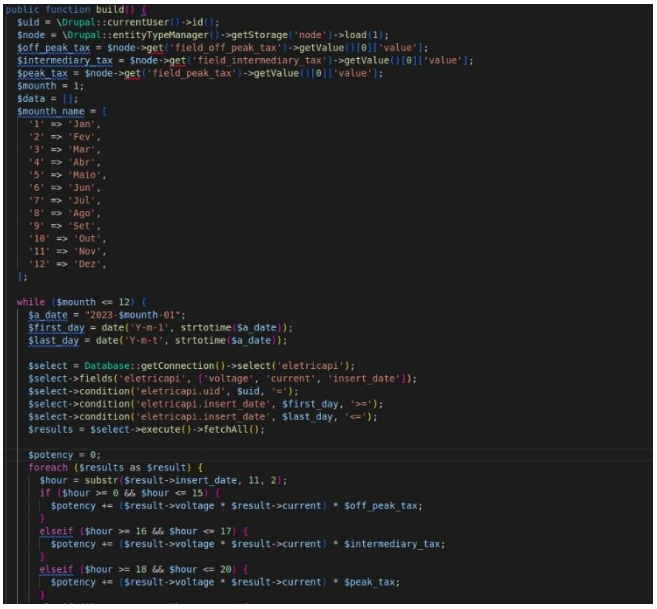


```
public function build() {
  $uid = \Drupal::currentUser()->id();
  $node = \Drupal::entityTypeManager()->getStorage('node')->load(1);
  $conventional_tax = $node->get('field_conventional_tax')->getValue()[0]['value'];
  $mounth = 1;
  $data = [];
  $mounth_name = [
    '1' => 'Jan',
    '2' => 'Fev',
    '3' => 'Mar',
    '4' => 'Abr',
    '5' => 'Maio',
    '6' => 'Jun',
    '7' => 'Jul',
    '8' => 'Ago',
    '9' => 'Set',
    '10' => 'Out',
    '11' => 'Nov',
    '12' => 'Dez',
  ];

  while ($mounth <= 12) {
    $a_date = "2023-$mounth-01";
    $first_day = date('Y-m-1', strtotime($a_date));
    $last_day = date('Y-m-t', strtotime($a_date));

    $select = Database::getConnection()->select('eletricapi');
    $select->fields('eletricapi', ['voltage', 'current']);
    $select->condition('eletricapi.uid', $uid, '=');
    $select->condition('eletricapi.insert_date', $first_day, '>=');
    $select->condition('eletricapi.insert_date', $last_day, '<=');
    $results = $select->execute()->fetchAll();

    $potency = 0;
    foreach ($results as $result) {
      $potency += ($result->voltage * $result->current);
    }
    $value = (($potency / 60) * $conventional_tax) / 1000;

    $data = array_merge($data, [$mounth_name[$mounth] => number_format($value, 4, ',', '.')]);
    $mounth++;
  }

  return [
    '#theme' => 'ano',
    '#data' => $data,
  ];
}
```

Fonte: Print tela desenvolvida.

O último bloco é o complementar ao apresentado na Fig. 13, este bloco demonstra o gasto com energia elétrica para cada mês do ano vigente, porém considerando a tarifa branca. Para realizar a soma dos valores referentes às potências, o sistema analisa qual o horário que a medição foi registrada no banco de dados e multiplica a potência registrada pela taxa referente ao horário, seja de pico, fora de pico ou intermediário, no restante o bloco realiza o mesmo processo do apresentado na Fig. 13. Este bloco de consumo de energia elétrica ao longo do ano para tarifa branca pode ser observado na Fig. 14.

Fig.14: Bloco de gasto com energia elétrica em cada mês do ano para tarifa branca.



```
public function build() {
  $uid = \Drupal::currentUser()->id();
  $node = \Drupal::entityTypeManager()->getStorage('node')->load(1);
  $off_peak_tax = $node->get('field_off_peak_tax')->getValue()[0]['value'];
  $intermediary_tax = $node->get('field_intermediary_tax')->getValue()[0]['value'];
  $peak_tax = $node->get('field_peak_tax')->getValue()[0]['value'];
  $mounth = 1;
  $data = [];
  $mounth_name = [
    '1' => 'Jan',
    '2' => 'Fev',
    '3' => 'Mar',
    '4' => 'Abr',
    '5' => 'Maio',
    '6' => 'Jun',
    '7' => 'Jul',
    '8' => 'Ago',
    '9' => 'Set',
    '10' => 'Out',
    '11' => 'Nov',
    '12' => 'Dez',
  ];

  while ($mounth <= 12) {
    $a_date = "2023-$mounth-01";
    $first_day = date('Y-m-1', strtotime($a_date));
    $last_day = date('Y-m-t', strtotime($a_date));

    $select = Database::getConnection()->select('eletricapi');
    $select->fields('eletricapi', ['voltage', 'current', 'insert_date']);
    $select->condition('eletricapi.uid', $uid, '=');
    $select->condition('eletricapi.insert_date', $first_day, '>=');
    $select->condition('eletricapi.insert_date', $last_day, '<=');
    $results = $select->execute()->fetchAll();

    $potency = 0;
    foreach ($results as $result) {
      $hour = substr($result->insert_date, 11, 2);
      if ($hour >= 0 && $hour <= 15) {
        $potency += ($result->voltage * $result->current) * $off_peak_tax;
      }
      elseif ($hour >= 16 && $hour <= 17) {
        $potency += ($result->voltage * $result->current) * $intermediary_tax;
      }
      elseif ($hour >= 18 && $hour <= 20) {
        $potency += ($result->voltage * $result->current) * $peak_tax;
      }
```

Fonte: Print tela desenvolvida.

V. CONSIDERAÇÕES FINAIS

A partir dos resultados obtidos é possível concluir que a plataforma é bastante efetiva e viável, dando a possibilidade ao usuário em desempenhar as funcionalidades mais básicas de uma plataforma *web*, como realizar o cadastro, cadastrar suas entidades, e também, desempenhar as funcionalidades necessárias para uma plataforma de monitoria, permitindo ao usuário registrar os dados de consumo de energia elétrica, assim como visualiza-los, permitindo também realizar tomadas de decisões com base nos valores visualizados.

Pode-se destacar como um dos pontos mais importantes para a plataforma, a possibilidade de identificar quais equipamentos consomem mais energia elétrica e tornam a fatura de energia elétrica mais elevada. Um outro ponto de destaque na conclusão deste trabalho, refere-se à possibilidade de o usuário identificar e tomar a decisão sobre a escolha da tarifa de energia mais adequada à sua respectiva residência. Logo, a plataforma permitirá ao usuário o poder de decisão sobre manter ou trocar a modalidade tarifária a partir das diferenças entre os valores monetários cobrados em suas respectivas tarifas.

Além do valor como um produto viável o projeto se mostra um grande facilitador para permitir que mais pessoas se interessem em conhecer seu consumo energético e como ele tem se distribuído pela casa, seja em relação a cômodos ou equipamentos específicos.

Apesar de já ser um facilitador por si só, a plataforma depende de outros projetos e incentivos voltados para a parte de *hardware* com sensores de corrente ou potência elétrica para que mais usuários sejam atraídos devido ao fato desse não ser um conhecimento tão difundido e simples de ser realizado, podendo ser um grande fator impeditivo.

Tendo em vista a escalabilidade do projeto desenvolvido, recomenda-se para trabalhos futuros a criação de gráficos para representar os valores consumidos, principalmente na análise anual do consumo, que permitem uma maior visibilidade do consumo de energia elétrica pelo próprio usuário, além da

criação de novas *dashboards* como a visualização do consumo em horários e dias específicos para pontos de medição selecionados.

REFERÊNCIAS

[1] KUMAR, Sachin. TIWARI, Prayag. ZYMBLER, Mikhail. Internet of Things is a revolutionary approach for future technology enhancement: a review. Journal of Big Data, 2019.

[2] PUTRA, Leonard. Et al. Design and Implementation of Web Based Home Electrical Appliance Monitoring, Diagnosing, and Controlling System. Procedia Computer Science, 2015, v. 59, p. 34-44.

[3] SCHULER, Ricardo. SISTEMA DE MONITORAMENTO DE CONSUMO DE ENERGIA ELÉTRICA PARA CIRCUITOS MONOFÁSICOS DE BAIXA TENSÃO. 2020. 95 p. Monografia (Bacharel em engenharia elétrica) - UNIVERSIDADE DO VALE DO TAQUARI, Lajeado, 2020.

[4] BERTOLINI, Cristiano et al. LINGUAGEM DE PROGRAMAÇÃO I. 1. ed. Santa Maria, RS: NÚCLEO DE TECNOLOGIA EDUCACIONAL, 2019. 114 p.

[5] SEABRA, Rodrigo Duarte; DRUMMOND, Isabela Neves; GOMES, Fernando Coelho. Análise Comparativa de Linguagens de Programação a partir de Problemas Clássicos da Computação. Revista de Sistemas e Computação, Salvador, ano 2018, v. 8, n. 1, p. 56-76, jun. 2018.

[6] Developer survey. [S. l.], 2 ago. 2021. Disponível em: https://insights.stackoverflow.com/survey/2021. Acesso em: 23 out. 2022.

[7] KUNDA, Douglas; SIAME, Alinaswe. Evolution of PHP Applications: A Systematic Literature Review. International Journal of Recent Contributions from Engineering, Science & IT (iJES), [s. l.], ano 2017, v. 5, n. 1, p. 28-39.

[8] História do PHP [S. 1.], [S. D.]. Disponível em: https://www.php.net/manual/pt_BR/history.php.php. Acesso em: 23 out. 2022.

[9] Our history. [S. l.], [S. D]. Disponível em: https://www.drupal.org/about/history. Acesso em: 23 out. 2022.

[10] About. [S. l.], [S. D]. Disponível em: https://www.drupal.org/about. Acesso em: 23 out. 2022.

[11] Register. [S. l.], [S. D]. Disponível em: https://phanteon.io/register. Acesso em: 11 out. 2022.

[12] Lei Geral de Proteção de Dados Pessoais (LGPD). [S. l.], [S. D]. Disponível em: https://www.gov.br/cidadania/pt-br/acesso-a-informacao/lgpd. Acesso em: 29 jan. 2023.

[13] EletricView. [S. l.], [S. D]. Disponível em: https://dev-eletricview.pantheonsite.io/. Acesso em: 29 jan. 2023.