



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Guilherme Madalena Below

**Desenvolvimento de Sistema de Monitoramento e Análise de Consumo e
Geração de Energia**

Florianópolis
2022

Guilherme Madalena Below

**Desenvolvimento de Sistema de Monitoramento e Análise de Consumo e
Geração de Energia**

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.

Orientador: Prof. Carlos Barros Montez, Dr.

Supervisor: Riscardo Vasconcellos, Eng.

Florianópolis

2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Below, Guilherme

Desenvolvimento de Sistema de Monitoramento e Análise
de Consumo e Geração de Energia / Guilherme Below ;
orientador, Carlos Montez, coorientador, Ricardo
Vasconcellos, 2022.

69 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia de Controle e Automação,
Florianópolis, 2022.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Internet das
Coisas. 3. Energia Solar. 4. Monitoramento. I. Montez,
Carlos . II. Vasconcellos, Ricardo. III. Universidade
Federal de Santa Catarina. Graduação em Engenharia de
Controle e Automação. IV. Título.

Guilherme Madalena Below

**Desenvolvimento de Sistema de Monitoramento e Análise de Consumo e
Geração de Energia**

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 24 de março de 2022.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Carlos Barros Montez, Dr.
Orientador
UFSC/CTC/DAS

Ricardo Vasconcellos, Eng.
Supervisor
CEO da Dayback

Prof. Marcelo de Lellis Costa de Oliveira, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

AGRADECIMENTOS

À minha família, por me apoiar durante toda a graduação e garantir os meios para desenvolvimento de tal trabalho.

Ao supervisor deste trabalho, Ricardo Vasconcellos, por estar sempre disponível para auxiliar no projeto e por me estimular constantemente.

Ao orientador, professor Carlos Montez, pela atenção desde o início do projeto e pelo rápido atendimento às demandas sempre que necessário.

Aos colaboradores da Dayback por manter a infraestrutura necessária para o desenvolvimento do projeto funcionando.

RESUMO

As empresas que produzem inversores para painéis de geração de energia solar fornecem acesso aos dados sobre a geração. Contudo, essas informações, de forma bruta, não permitem uma análise mais detalhada e uma comparação entre consumo e geração, distinguindo energia injetada de auto-consumo. Caso painéis solares de fabricantes variados sejam instalados, o gerenciamento se torna mais complexo. Este projeto visa melhorar uma plataforma que monitora tanto geração quanto consumo de energia, produzindo relatórios e visualização de dados na web. Foram desenvolvidos padrões de implementação de código e telas de monitoramento e análise de dados históricos e de tempo real. Como fonte de dados foram utilizadas APIs de inversores e medidores acoplados a hardware de IoT, para as quais foram criadas interfaces de integração.

Palavras-chave: , Monitoramento, Energia Solar, Internet das Coisas.

ABSTRACT

Inverter manufacturing companies for solar energy panels provide access to generation data, however, none of them offer comparison between energy consumption and generation, distinguishing injected energy from local consumption. In case solar panels from various manufacturers are installed, the management becomes more complex. This project intends to improve a platform that monitors both consumption and generation of energy, outputting a report and allowing for web data visualization. Standards of code implementation and monitoring and analysis screens for energy data both in real time and history were developed. As data sources, inverter APIs and measuring devices connected to IoT hardware were used, for which integration interfaces were made.

Keywords: Monitoring. Solar Energy. Internet of Things.

LISTA DE FIGURAS

Figura 1 – Tela de Login	17
Figura 2 – Dashboard do Dayback Energy	19
Figura 3 – Gerenciamento de plantas de clientes	20
Figura 4 – CMS	21
Figura 5 – Notificação do Telegram	22
Figura 6 – Inicialização de página da web	23
Figura 7 – Modelo Model-View-Controller	27
Figura 8 – Exemplo de comunicação com MQTT	32
Figura 9 – Diagrama de integração	35
Figura 10 – Diagrama de casos de uso do gerador de relatórios	36
Figura 11 – Diagrama de classes do gerador de relatórios	37
Figura 12 – Diagrama de classes da <i>Chart</i>	38
Figura 13 – Diagrama de casos de uso do gerador de relatórios	39
Figura 14 – Diagrama de integração	40
Figura 15 – <i>Dashboard</i> de plantas ABB	41
Figura 16 – <i>Dashboard</i> do cliente ABB	42
Figura 17 – <i>Dashboard</i> do cliente Smappee	43
Figura 18 – Payload instantâneo	44
Figura 19 – Payload consolidado	44
Figura 20 – utilidade de requisição de dados <i>front-end</i>	47
Figura 21 – Controller de Energia	48
Figura 22 – Página de gerador de relatórios	49
Figura 23 – Exemplo de função geradora de <i>template</i>	50
Figura 24 – Função para compilar o PDF	50
Figura 25 – Página de gerador de relatórios	51
Figura 26 – Arquivos da <i>Chart</i>	52
Figura 27 – Propriedades da <i>Chart</i>	53
Figura 28 – Definição de retorno da <i>Chart</i>	54
Figura 29 – Exemplo 1 utilização componente <i>Chart</i>	54
Figura 30 – Exemplo 2 de utilização componente <i>Chart</i>	55
Figura 31 – Criador de gráficos	55
Figura 32 – Tela de status	56
Figura 33 – Nova <i>Dashboard</i>	57
Figura 34 – Nova <i>Dashboard</i> de televisão	58
Figura 35 – Inspetor de dados tempo real	59
Figura 36 – Teste de Confiabilidade	61
Figura 37 – Capa	67

Figura 38 – introdução	67
Figura 39 – Sistema	67
Figura 40 – Geração x Radiação	67
Figura 41 – Consumo Total	68
Figura 42 – Geração total	68
Figura 43 – Fator de potência	68
Figura 44 – Conclusões	68
Figura 45 – Contato	69

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
DBHW	Dayback Hardware
MQTT	Message Queuing Telemetry Treansport
IoT	Internet of Things
TCP	Transmission Control Protocol
API	Application Process Interface
DOM	Document Object Model
HTML	HyperText Markdown Language
CSS	Cascading Style Sheets
ORM	Object Relation Mapping
CLI	Command Line Interface
SQL	Standard Query Language
GUI	Greaphic User Interface
MVC	Model View Controller
GMT	Greenwitch Mean Time zone

SUMÁRIO

1	INTRODUÇÃO	12
1.1	HISTÓRICO DA EMPRESA	12
1.2	MERCADO DE ENERGIA	12
1.3	MODELO DE NEGÓCIO	13
1.4	PROBLEMA E PROPOSTA DE SOLUÇÃO	13
1.5	OBJETIVOS	14
1.6	METODOLOGIA	14
1.7	ESTRUTURA DO DOCUMENTO	14
2	DESCRIÇÃO DO DAYBACK ENERGY	16
2.1	ARQUITETURA DO SISTEMA	16
2.2	AUTENTICAÇÃO	16
2.3	MONITORAMENTO	17
2.4	CMS	20
2.5	NOTIFICAÇÃO	21
2.6	ARMAZENAMENTO	22
3	TECNOLOGIAS E CONCEITOS	23
3.1	APLICAÇÃO WEB	23
3.1.1	HTML, CSS e Javascript	23
3.1.2	React	24
3.1.3	PHP e Laravel	24
3.1.4	Webpack e Babel	25
3.1.5	Tailwind	25
3.1.6	MySQL	25
3.2	CONCEITOS DE ARQUITETURA DE SOFTWARE	26
3.2.1	Modelo MVC	26
3.2.2	ORM Eloquent	27
3.2.3	Formas normais do banco de dados	27
3.2.4	Princípios utilizados no <i>front-end</i>	28
3.2.5	Princípios SOLID	29
3.3	SERVIDOR E INFRAESTRUTURA	30
3.3.1	Amazon Web Services	30
3.3.2	AWS RDS	30
3.3.3	S3	31
3.3.4	EC2	31
3.3.5	MQTT broker	32
3.3.6	AWS SES	33
3.3.7	Telegram	33

4	PROJETO E ARQUITETURA	34
4.1	REQUISITOS FUNCIONAIS	34
4.2	REQUISITOS NÃO FUNCIONAIS	34
4.3	ORGANIZAÇÃO DE SERVIÇOS DE INTEGRAÇÃO	35
4.4	PROJETO DE COMPONENTES FRONT-END	36
4.4.1	Gerador de relatórios	36
4.4.2	Padrão para Componentes	37
4.4.3	Ferramenta de inspeção de dados em tempo real	38
4.4.4	Tela de Status	39
5	INTEGRAÇÃO	40
5.1	ABB AURORA VISION	40
5.2	SMAPPEE	42
5.3	DBHW	43
5.4	FRONIUS	45
5.5	SOLAR EDGE	45
5.6	ARMAZENAMENTO	45
5.7	INTERFACE PADRONIZADA	46
6	IMPLEMENTAÇÃO DE TELAS E FERRAMENTAS	49
6.1	GERADOR DE RELATÓRIOS	49
6.2	CHART	51
6.3	TELA DE STATUS	56
6.4	MUDANÇA NA UI	57
6.5	FERRAMENTA DE INSPEÇÃO DE DADOS EM TEMPO REAL	58
7	RESULTADOS	60
7.1	ANÁLISE	60
7.2	TESTES	61
7.3	IMPACTOS	62
8	CONCLUSÃO	63
8.1	TRABALHOS FUTUROS	63
	REFERÊNCIAS	65
	ANEXO A – MODELO DE RELATÓRIO	67

1 INTRODUÇÃO

Este documento detalha o projeto de fim de curso realizado pelo autor durante seu período de trabalho na Dayback, ao longo da disciplina DAS5511, "Projeto de fim de curso" do curso de Engenharia de Controle e Automação da Universidade Federal de Santa Catarina. O estágio ocorreu no período entre 05/06/2021 e 01/12/2021. Foram cumpridas 30 horas semanais durante 6 meses.

1.1 HISTÓRICO DA EMPRESA

A Dayback Energia Estratégica nasceu como Pensys Tecnologia em 2014 e ganhou diversos prêmios de inovação. No ano de 2016 passou a ser incubada no CELTA, incubadora do CERTI, e focou totalmente seus esforços em produtos para Microgeração de Energia Elétrica por fonte Eólica e Solar. EM 2017, a empresa mudou de estratégia, tomando o nome que tem hoje, Dayback Energia Estratégica, com foco em gerenciamento e monitoramento de energia e integração de sistemas em *Cloud*.

1.2 MERCADO DE ENERGIA

O desperdício de energia no Brasil em um período de dois anos é de aproximadamente 52 bilhões de reais, conforme dados da Associação Brasileira das Empresas de Serviços de Conservação de Energia (ABESCO), para o período entre 2015 e 2017. Para colocar este valor em contexto, ele representa metade da produção de energia da usina de Itaipu no mesmo período (CICLOVIVO, 2019).

Segundo a Confederação Nacional da Indústria (CNI), o setor industrial é responsável pelo consumo de 41 % da energia elétrica do Brasil. Nesse sentido, a relação intensa entre as indústrias e o consumo da energia elétrica é evidente. Isso ilustra a importância de um projeto de eficiência energética para indústrias (GUSTAVO, 2017).

Atualmente, nas unidades fabris do Brasil, cerca de 20 por cento dos motores instalados possuem mais de 25 anos, ao passo que esses são os maiores consumidores de energia nas indústrias. Na maioria dos casos, por exemplo, os motores já foram rebobinados cerca de 7 a 10 vezes (GUSTAVO, 2017). O dimensionamento incorreto de cargas de motores e a utilização de máquinas sem manutenção ou de eficiência baixa são os maiores causadores de ineficiência energética na indústria.

"O desperdício é cobrado em valor de multa na conta de energia, que vem descrita como Energia Reativa Excedente. Os dados referentes a esta multa não são bem explicados, sendo que o valor pode representar até 30% do valor da fatura", explica Fábio Amaral, diretor da Energy Painéis Elétricos, empresa que promove um workshop sobre o assunto no dia 17 de outubro em Curitiba, no Paraná (CICLOVIVO, 2019).

Observando os números e as demandas, percebe-se que há espaço no mercado para se investir em monitoramento e eficiência energética. Esta é uma das razões por que a Dayabck resolveu se especializar nesta área.

1.3 MODELO DE NEGÓCIO

A estratégia de negócio da empresa se baseia em três serviços principais, dois dos quais ainda estão em desenvolvimento: o Dayback Energy, o pegada de carbono e o totem carregador de carros elétricos. O único que está em operação é o Dayback Energy, e este é o tema deste projeto.

A ideia do pegada de carbono consiste em medir energia e gerar créditos de carbono. O totem carregador do carro elétrico é um equipamento que possibilita que se cobre pelo uso de carregadores de carro elétrico. Este equipamento inclui um aplicativo que possui uma versão do cliente, que é responsável pela ativação do totem, e uma versão do administrador, que mostra um relatório de uso e rendimento dos carregadores de um usuário.

O Dayback Energy, foco deste trabalho, consiste em três funcionalidades principais: o monitoramento de consumo e geração de energia, o relatório e o serviço de notificação. Os dados de energia são obtidos por meio de equipamento próprio da empresa, que requer instalação, ou por meio de integração com APIs externas de outros equipamentos, como inversores de equipamentos de geração de energia.

1.4 PROBLEMA E PROPOSTA DE SOLUÇÃO

A Dayback é uma empresa pequena que está tentando crescer em número de clientes. No entanto, a carga de trabalho sobre a equipe da empresa é muito alta. Relatórios mensais devem ser gerados para todos os clientes do Dayback Energy, o que ocupa boa parte da carga de trabalho de alguns dos funcionários. Um aumento súbito na quantidade de clientes poderia facilmente causar uma sobrecarga. Para solucionar este problema, foi proposta a criação de um gerador automático de relatórios, que carregasse os dados de energia dos clientes em páginas pré-configuradas.

A plataforma utiliza dados de energia coletados de diversos equipamentos de marcas diferentes, com especificações diversas. Com o aumento da diversidade de equipamentos, houve dificuldade em se monitorar quando os equipamentos estavam em funcionamento correto e quando estavam sem comunicação ou não operantes. Para resolver este problema, propôs-se a criação de uma interface de status de todos os dispositivos habilitados que pudesse ser visualizada rapidamente pela equipe e que o suporte técnico pudesse ser acionado sem atrasos.

A característica de inovação da empresa cria um efeito negativo de incerteza com relação aos requisitos do sistema. É muito comum que a busca de novos clientes

cause o surgimento de novas demandas de software para o fechamento de contrato. Para que adaptações rápidas sejam possíveis, deve-se escrever o código de forma que ele se mantenha extensível e robusto, para que a adição de novas funcionalidades a componentes antigos não comprometa a qualidade.

1.5 OBJETIVOS

De forma geral, o objetivo do projeto é criar ferramentas automatizadas de suporte aos processos da empresa, visando maior escalabilidade da plataforma e maior flexibilidade e alcance ao se integrar novas tecnologias e desenvolver novas funcionalidades.

As soluções propostas estão enumeradas a seguir:

- Integrar a plataforma com APIs de equipamentos de medição e estabelecer uma interface entre os dados do *back-end* e *front-end*.
- Criar um gerador automático de PDF, a ser utilizado como gerador automático customizável de relatórios em PDF.
- Melhorar o design do *front-end*.
- Criar método de reenvio de dados em caso de perda de internet.
- Desenvolver uma forma de monitorar o status dos pontos de medição de energia.

1.6 METODOLOGIA

O projeto foi uma aprimoração de um sistema que já estava em operação, com atualizações progressivas no software em produção. A maior parte das ferramentas foram desenvolvidas para a utilização pela equipe da Dayback, que é enxuta. Por isso foi possível obter opiniões e sugestões dos próprios usuários do sistema durante o processo de implementação.

A desenvolvimento como um todo foi realizado seguindo a metodologia de projetos Ágil. Foram realizadas reuniões semanais para discutir prioridades e avaliar o trabalho executado. Nelas, eram dadas sugestões e críticas sobre o desenvolvimento do projeto, o que facilitou a elaboração de requisitos de sistema e reduziu a quantidade de retrabalho no final do projeto.

1.7 ESTRUTURA DO DOCUMENTO

- Capítulo 1: Apresentação da empresa e do problema e definição de objetivos
- Capítulo 2: Descrição do Dayback Energy

- Capítulo 3: Tecnologias e Conceitos
- Capítulo 4: Projeto e Arquitetura
- Capítulo 5: Integração com outras tecnologias
- Capítulo 6: Descrição da implementação do projeto
- Capítulo 7: Análise de resultados e testes
- Capítulo 8: Conclusões

2 DESCRIÇÃO DO DAYBACK ENERGY

Este capítulo discorre sobre os detalhes de implementação da plataforma Dayback Energy. Nele são apresentados sua arquitetura, suas funcionalidades e tecnologias de implementação serão descritas.

2.1 ARQUITETURA DO SISTEMA

Há duas formas de interagir com a plataforma:

- requisições *http*
- comandos

Todas as requisições *http* feitas ao *site* são tratadas pelo roteador do Laravel (*framework* para aplicações *web* em *php*). O roteador é o componente redireciona as requisições para a rotina correspondente, que pode ser um envio de página da *web* ou uma chamada de controlador.

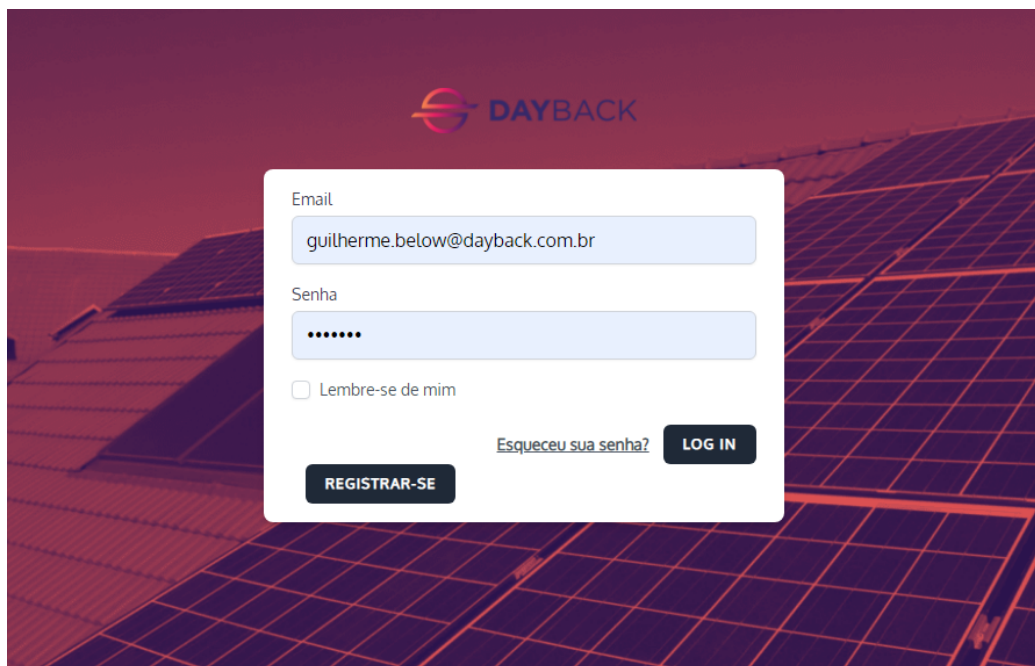
As rotas são protegidas por um *middleware*, que intercepta as requisições e verifica a integridade das mesmas. No caso, o *middleware* foi utilizado para verificar se a requisição pertence a um usuário autenticado.

Os comandos são rotinas que podem ser acionadas pelo *prompt* de comando ou por um agendador de tarefas. O envio de notificação e o gravador de dados são feitos através de comandos agendados.

2.2 AUTENTICAÇÃO

O login do sistema é feito por meio de usuário (e-mail) e senha, como mostrado na Figura 1:

Figura 1 – Tela de Login



Fonte: Arquivo pessoal.

Existem 5 tipos de usuário, definidos por seus “roles” (papéis): Administrador, Integrador, Cliente, Básico e Técnico. As permissões variam dependendo do papel do usuário.

A autenticação se dá pelo método padrão do Laravel, que funciona gerando um CSRF Token quando o usuário faz login. O *token* é gravado em um arquivo dentro das pastas temporárias do servidor e em um *cookie* no *browser* do usuário. Cada vez que uma requisição é feita ao servidor, o *cookie* é enviado junto e a comparação é feita. O *token* expira depois de um período de tempo configurável de inatividade. Esta utilidade existe para evitar que ocorram ataques de cross-site request forgery (CSRF).

2.3 MONITORAMENTO

A *dashboard* é o painel que mostra os dados da API do ABB, Smappee e/ou DBHW. Usuários com papel do tipo Cliente têm acesso a esta página, assim como a todas as que o usuário Básico possui.

A *dashboard* contém três elementos principais:

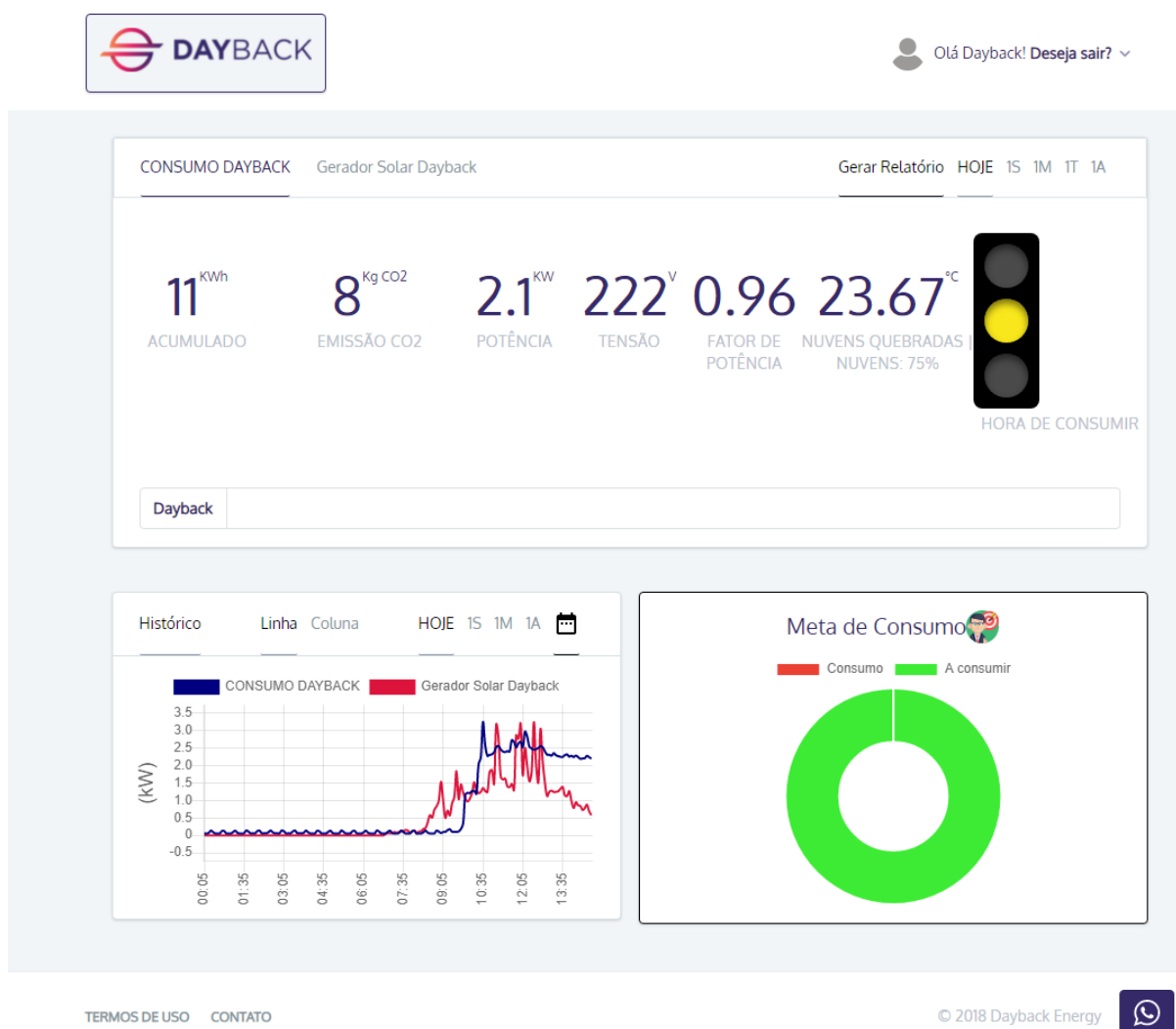
- Consolidado
- Meta de consumo
- Tabela

A meta de consumo é o gráfico à direita na tela mostrada na Figura 2. O valor a consumir é uma média do consumo diário no último mês, enquanto o consumo é o quanto já foi consumido no dia atual.

O consolidado é um espaço para colocação de indicadores. A empresa oferece customização nesta área. É possível que um cliente peça um indicador específico para sua necessidade, mas este não é o padrão do plano de negócios. Os elementos principais do consolidado são:

- Valor acumulado de consumo de energia em kWh e valor equivalente de emissão de CO2 deste valor.
- Valores de potência instantânea, tensão e fator de potência em tempo real. Caso o cliente não possua medidores de potência instantânea, este elemento não aparece.
- Semáforo com 3 estados: verde quando a geração de energia é mais alta que o consumo, amarelo quando o consumo estiver mais alto que a geração e vermelho quando a geração for zero. Caso o cliente não possua geração de energia, o semáforo não aparece na tela.
- clima e tempo: que puxa de uma API do tempo os dados meteorológicos da localização do cliente.

Figura 2 – Dashboard do Dayback Energy



Fonte: Arquivo pessoal.

O consolidado tem opção de mostrar um medidor por vez. A seleção pode ser feita por meio dos botões que aparecem no canto superior esquerdo. Também há a opção de escolher o período de tempo que se quer consultar. No canto superior direito existem opções de “hoje”, uma semana, um mês, um trimestre e um ano. A seleção de período só se aplica aos campos de valor acumulado de geração de energia e CO₂.

O último elemento é a tabela, onde o cliente pode consultar o histórico do perfil de consumo e geração mais detalhadamente. Os dados apresentados na Figura 2 são referentes ao usuário de demonstração, que tem seus medidores instalados no prédio do CELTA, onde a empresa está incubada.

A tabela oferece períodos de consulta, como mostrado na Figura 2. Também existe a opção de escolher um dia específico para a consulta.

2.4 CMS

O content management system (CMS) é o sistema de gerenciamento de clientes e banco de dados da plataforma. A Figura 3 mostra a página de gerenciamento de clientes.

A lista à esquerda mostra cada um dos usuários com papel de *Client*. O botão “Painel” leva à *dashboard* do usuário em questão. O status fica em vermelho caso os medidor principais do cliente estejam desativados.

O mapa à direita mostra todas as localizações das plantas dos clientes.

Figura 3 – Gerenciamento de plantas de clientes

The screenshot displays the Dayback Energy CMS interface. At the top left is the Dayback logo. The top right shows a user profile for 'Olá Ricardo!' with a 'Deseja sair?' link. Below the header is a navigation bar with 'CLIENTES', 'RELATORIO', and 'CMS' tabs. The main content area is divided into three sections: 'Unidades' (MW DE CAPACIDADE INSTALADA), 'Kg CO2' (EMISSÃO EQUIVALENTE), and 'MWh' (GERADO NO TOTAL). Below these are two main panels: 'Clientes' and 'Mapa'. The 'Clientes' panel lists five clients with their installed power and status:

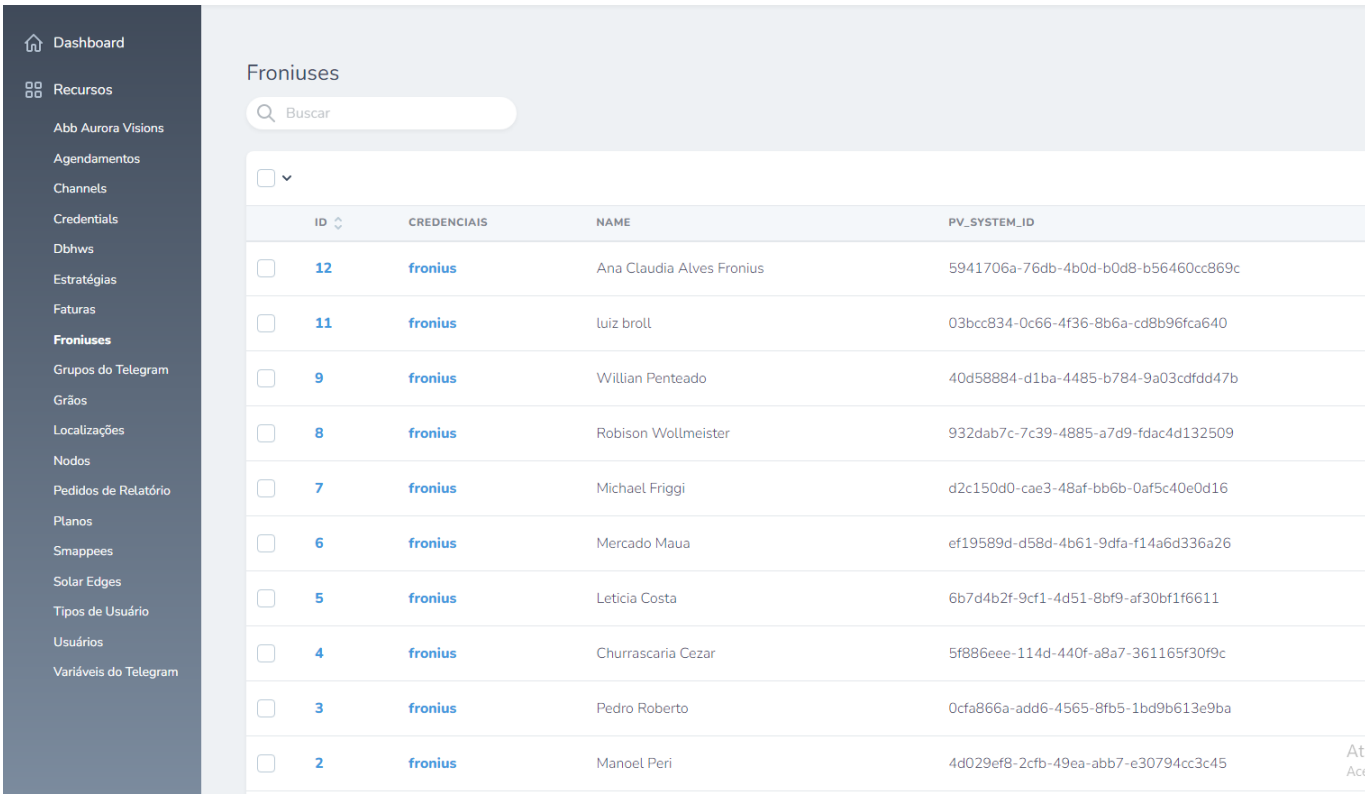
Client	Installed Power (kW)	Status
SU Super Lima	118.8	OK
ST Stangherlin	42.21	OK
CO Cooparcentro	75.4	OK
PO Posto Manivela	81.25	OK
BE Beija Flor	162.15	OK

The 'Mapa' panel shows a Google Maps interface with a location pin on a map of Brazil. A Google Maps error message is displayed: 'Esta página não carregou o Google Maps corretamente. Você é o proprietário deste site? OK'. The map includes standard navigation controls like zoom in (+) and zoom out (-) buttons.

Fonte: Arquivo pessoal.

A Figura 4 mostra uma das páginas de edição de entidades. O sistema foi implementado utilizando um software de integração do ecossistema do Laravel, chamado Nova. Ele já vem com funções de edição, criação e deleção de dados, política de autenticação, suporte a temas de UI.

Figura 4 – CMS



The screenshot displays a web application interface for managing Fronius users. On the left is a dark sidebar with a menu containing items like Dashboard, Recursos, and various system components. The main area is titled 'Froniuses' and features a search bar. Below the search bar is a table with columns for ID, CREDENCIAIS, NAME, and PV_SYSTEM_ID. The table lists 12 users, each with a checkbox for selection. The users are ordered by ID from 12 down to 2.

ID	CREDENCIAIS	NAME	PV_SYSTEM_ID	
<input type="checkbox"/>	12	fronius	Ana Claudia Alves Fronius	5941706a-76db-4b0d-b0d8-b56460cc869c
<input type="checkbox"/>	11	fronius	luiz broll	03bcc834-0c66-4f36-8b6a-cd8b96fca640
<input type="checkbox"/>	9	fronius	Willian Pentead0	40d58884-d1ba-4485-b784-9a03cdfdd47b
<input type="checkbox"/>	8	fronius	Robison Wollmeister	932dab7c-7c39-4885-a7d9-fdac4d132509
<input type="checkbox"/>	7	fronius	Michael Friggi	d2c150d0-cae3-48af-bb6b-0af5c40e0d16
<input type="checkbox"/>	6	fronius	Mercado Maua	ef19589d-d58d-4b61-9dfa-f14a6d336a26
<input type="checkbox"/>	5	fronius	Leticia Costa	6b7d4b2f-9cf1-4d51-8bf9-af30bf1f6611
<input type="checkbox"/>	4	fronius	Churrascaria Cezar	5f886eee-114d-440f-a8a7-361165f30f9c
<input type="checkbox"/>	3	fronius	Pedro Roberto	0cfa866a-add6-4565-8fb5-1bd9b613e9ba
<input type="checkbox"/>	2	fronius	Manoel Peri	4d029ef8-2cfb-49ea-abb7-e30794cc3c45

Fonte: Arquivo pessoal.

2.5 NOTIFICAÇÃO

O Laravel possui um sistema de agendamento de tarefas. Foi configurado que a cada cinco minutos um comando fosse ativado, o que faz com que seja chamada uma rotina de notificação para cada ponto de medição configurado.

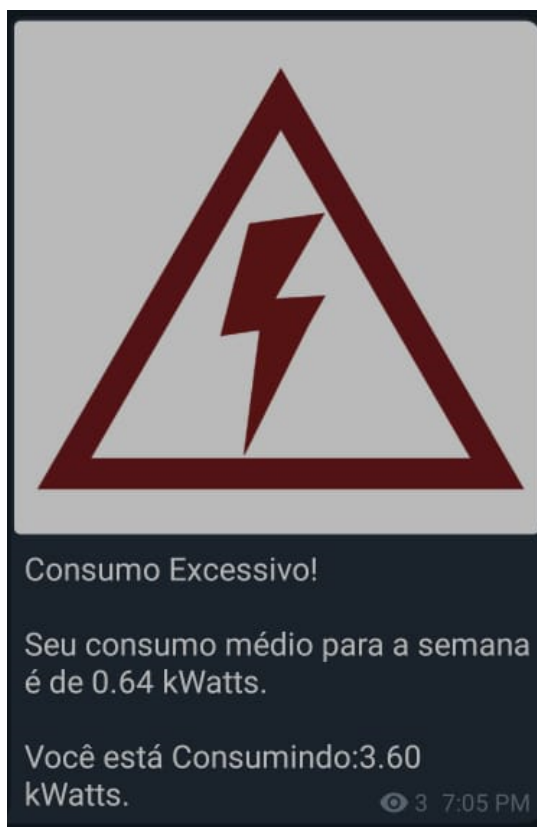
Cada uma das rotinas leva um tempo para ser processada, além disso, alguma delas pode retornar com erro. Para impedir que um erro interrompa o processo inteiro ou que outros comandos realizados no servidor impeçam que as rotinas sejam realizadas, foi implementado um sistema de fila.

O gerenciador de processos do Laravel atribui *workers* às rotinas da fila. A quantidade de *workers* pode ser configurada no “Supervisor” (um programa para Linux). Isto permite que haja processamento paralelo e independência entre as funções, reduzindo a quantidade de erros.

O algoritmo de notificação separa o consumo em dois grupos: noite e dia. O dia geralmente tem um consumo maior que a noite, por isso se faz a separação. São calculadas a média e o desvio padrão dos grupos de dados, em seguida verifica-se se o valor atual de medida é maior que um desvio padrão acima da média. Caso seja, é

enviada uma mensagem de notificação, como mostrado na Figura 5

Figura 5 – Notificação do Telegram



Fonte: Arquivo pessoal.

2.6 ARMAZENAMENTO

O Smappee e ABB Aurora Vision (sistemas de monitoramento integrados à plataforma) dão suas próprias garantias de permanência de dados, mas foi decidido que para garantir o respeito ao contrato do cliente eles também seriam gravados no banco de dados da Dayback.

Um comando, igual ao descrito para a notificação, foi utilizado nesta seção. Também com processamento paralelo e gerenciamento de erros. O comando é executado somente uma vez por dia, pois as APIs nem sempre estão com os dados atualizados, então se a frequência de leitura fosse muito alta, haveriam instantes de medição perdidos.

3 TECNOLOGIAS E CONCEITOS

Este capítulo tem como objetivo explicar os conceitos utilizados no projeto, que incluem temas de programação, engenharia de software e as tecnologias utilizadas na implementação.

3.1 APLICAÇÃO WEB

A primeira parte do projeto foi criar uma aplicação *web* que fosse capaz de acessar uma API e apresentar os dados em um *web site* em forma de *dashboard*. Nesta sessão, são apresentadas as tecnologias e as razões para a escolha delas.

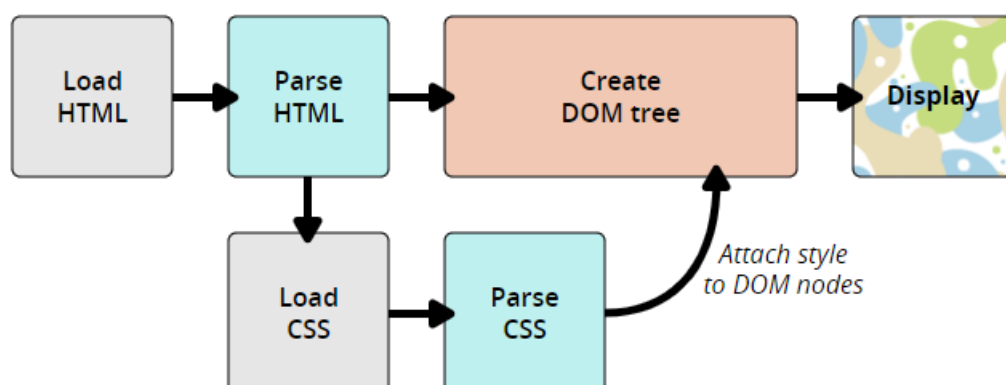
3.1.1 HTML, CSS e Javascript

HTML (Linguagem de Marcação de HiperTexto) é a linguagem mais básica da construção de *web sites*. Ela define a estrutura básica do conteúdo. Por meio de tags como `<head>`, `<body>`, `<div>`, o HTML fornece os objetos que constituem o DOM (Document Object Model) da aplicação (W3SCHOOLS, 2021a).

O CSS (Cascading Style Sheets) é responsável pela estilização dos elementos do DOM. As estilizações podem ser de fonte, disposição de elementos na tela, borda, cores, *backgrounds*, tamanhos dos elementos, animações, entre outras. Todo o *web site* que precisa de qualquer estilização utiliza o CSS (W3SCHOOLS, 2021b).

A Figura 6 apresenta, através de um fluxograma, a interação entre HTML e CSS para a constituição do DOM.

Figura 6 – Inicialização de página da web



Fonte: (MOZILLA, 2021)

O Javascript é a linguagem de programação utilizada para adicionar funções aos elementos do DOM. Podendo adicionar, remover e modificar todos os elementos,

atributos e estilos do DOM, sendo assim, a principal ferramenta para alteração de página da *web* depois do carregamento completo (W3SCHOOLS, 2021c).

Javascript é uma linguagem multi paradigma, suportando estilos de programação imperativo, funcional, orientado a eventos e, mais recentemente, também orientado a objetos (FLANAGAN, 2002). Esta flexibilidade a torna uma linguagem muito versátil. É a linguagem de programação mais utilizada no mundo atualmente (LIU, 2020).

3.1.2 React

React é uma biblioteca de Javascript que pode ser utilizada para fazer pequenas funcionalidades ou servir de base para criar aplicações de *front-end* inteiras. Seu grande diferencial é a sua habilidade de virtualização de DOM, que permite uma sintaxe amigável e versatilidade quanto à decisão de quando atualizar o DOM real. Isto faz com que as aplicações ganhem em desempenho, principalmente em projetos grandes (RAVICHANDRAN, 2020).

O React define componentes, que possuem métodos, variáveis e estados internos. Cada componente pode definir e utilizar outros componentes. A própria aplicação em si é um componente. Esta arquitetura de software favorece composição como forma de reutilização de código.

O fluxo de dados do React é sempre do componente pai para o componente filho na forma de *props*, que funcionam como argumentos de função, o que organiza e simplifica a lógica. Consequentemente, componentes mais altos na hierarquia não devem se importar com o que acontece dentro de componentes mais baixos. Caso seja necessário que o fluxo de dados ocorra de alguma forma que não seja por meio das *props*, deve-se utilizar o React Context, uma biblioteca externa de gerenciamento de estado, como o Redux, ou manipulação direta do DOM. É recomendado na documentação que só se utilize estas ferramentas caso elas sejam realmente necessárias.

Na versão 16.8, o React apresentou uma sintaxe diferente, em que toda a aplicação poderia ser escrita na forma de funções construtoras, e não classes. As manipulações e definições de estado deveriam ser feitas por meio de *hooks*, que servem justamente para que essas funções tenham todas as funcionalidades das classes. Trabalhar com funções foi preferido a usar classes, portanto, esta sintaxe funcional foi utilizada.

3.1.3 PHP e Laravel

Laravel é um framework de PHP para desenvolvimentos de aplicações *web*. Considerado por muitos (GOEL, 2020) um dos melhores *frameworks* de *back-end*, juntamente com Ruby on Rails.

O Laravel possui suporte para login, autenticação, recuperação de senha, envios de e-mail, gerenciamento de eventos, um motor de *templates*, um ORM (Object

Relational Mapping), gerenciamento de execução de processos (filas), um sistema de rotas, middlewares, um CLI (*command line interface*) e um ecossistema muito rico, com diversas bibliotecas com soluções prontas feitas para o Laravel. Mais adiante será explicado sobre cada uma das utilidades do Laravel no capítulo de implementação.

A riqueza de artifícios proporcionada pelo Laravel faz dele uma escolha excelente para projetos com equipes pequenas, em que recursos como tempo e dinheiro devem ser poupados ao máximo.

3.1.4 Webpack e Babel

O Laravel em si não consegue utilizar os arquivos do React, pois este necessita de um compilador para funcionar. Portanto, utiliza-se o Webpack, que é um compilador de Javascript e, opcionalmente, de arquivos de estilo e imagens.

O Laravel Mix é um pacote que disponibiliza uma API para definir passos de produção para a aplicação Laravel, definindo diversos pré-processadores de Javascript e CSS, incluindo um de React.

3.1.5 Tailwind

Tailwind é um framework de CSS baseado em utilidades que providencia classes prontas que podem ser colocadas diretamente na marcação HTML. Por exemplo, “font-bold” transforma a fonte em negrito, “text-xl” deixa o texto no tamanho definido nas configurações como “xl”. Estas classes podem ser compostas simplesmente escrevendo a classe do elemento como uma lista de classes do Tailwind, como class = “font-bold text-xl”. Esta técnica confere agilidade no desenvolvimento, bem como uma forma de organizar os estilos do código (TAILWIND, 2021). A primeira vista, um programador experiente se perguntaria se o Tailwind não é igual a criar os objetos de estilo diretamente nos componentes. Isso não é verdade, pois este tipo de estilização “inline” não permite que se adicione responsividade nem condicionais de eventos. Além disso, o Tailwind também permite que se criem classes de estilo, assim como no método convencional. Outra vantagem é que ele permite que se criem “temas” de classes, definindo tamanhos de tela para os gatilhos de responsividade, paleta de cores, eventos, entre outros, o que configura uma forma de padronização de estilos da aplicação.

3.1.6 MySQL

MySQL é um dos sistemas de banco de dados mais populares da atualidade. A escolha de utilizar um banco de dados relacional foi dada primariamente por causa da facilidade de integração com o app Laravel (LARAVEL, 2021), que já vem com um ORM (Object Relational Mapper). Seguindo o *Dependency Inversion Principle*, que será discutido ainda neste capítulo, um subsistema deve poder ser trocado por outro

sem que o programa perca as propriedades desejadas. O ORM faz justamente isso, possibilita que o mesmo banco de dados possa ser replicado em qualquer sistema de gerenciamento de banco de dados relacional.

3.2 CONCEITOS DE ARQUITETURA DE SOFTWARE

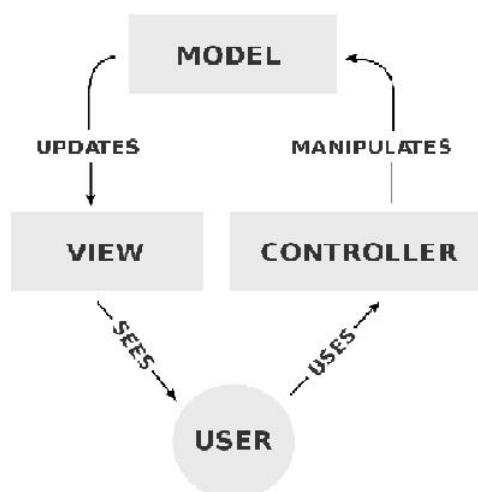
Nesta sessão, serão explicados os conceitos de arquitetura de software principais aplicados durante a criação do projeto. Isto não inclui detalhes de implementação, mas sim a teoria para justificar a organização do projeto, o fluxo de dados dentro da aplicação e os padrões de escrita de funções e classes.

3.2.1 Modelo MVC

O modelo MVC (*Model-View-Controller*) representado na figura 7, é um padrão de desenvolvimento de software adotado por muitas aplicações *web* e programas com GUI (*Graphic User Interface*). O Laravel já vem com suas ferramentas e organização básica de arquivos nos padrões deste modelo. Existem três componentes neste contexto (HOPKINS, 2013):

- **Modelo:** é a estrutura de dados do sistema. Determina quais são as entidades do sistema, a relação entre elas, a comunicação com a camada de permanência de dados (banco de dados) e as regras de negócio.
- **Controlador:** Recebe uma entrada e converte em algum comando para o modelo ou para a *view*. Em sistemas mais modernos, está havendo a separação do controlador em mais partes, ficando o controlador somente como interface para a sua camada no modelo.
- **View:** A *View* é a interface gráfica do usuário. Qualquer elemento que seja renderizado na tela do usuário final faz parte da *View*.

Figura 7 – Modelo Model-View-Controller



Fonte: Wikipedia.

Estes são os conceitos mais importantes do MVC utilizados no projeto

3.2.2 ORM Eloquent

O Eloquent é o ORM (*Object Relation Mapper*) utilizado como padrão pelo Laravel. Com ele, cada tabela do banco de dados pode ser representada por um *Model*, que é usado como interface de interação com a tabela do banco de dados. O Eloquent permite que se adicione, atualize, e remova dados das tabelas por meio da classe correspondente à tabela. Os comandos de acesso ao banco de dados podem ser gerados automaticamente por meio dos métodos dos *Models* ou escritos manualmente. Uma vantagem de se utilizar um ORM é que se pode passar o mapeamento de uma tabela do banco de dados como uma classe através do software, o que é muito prático quando os componentes que dependem da tabela fazem múltiplas interações com ela.

3.2.3 Formas normais do banco de dados

Normalização é o processo de organização de banco de dados a partir de regras visando eliminar redundância de dados e prover flexibilidade em relação a mudanças (MICROSOFT, 2022). As três formas normais mais importantes para o projeto e que foram utilizadas durante a implementação estão descritas a seguir :

- 1NF (Primeira forma normal): Eliminar grupos repetidos de dados na mesma tabela, identificar cada tabela por uma chave primária, criar uma tabela para cada conjunto de dados relacionados.

- 2NF (Segunda forma normal): Criar tabelas separadas para referências que se aplicam a múltiplos valores e relacionar as tabelas por uma chave externa.
- 3NF (Terceira forma normal): Eliminar campos que não dependem da chave. Por exemplo, uma dependência funcional, que não depende da chave, seria de um valor que é múltiplo de outro que não seja a chave primária.

3.2.4 Princípios utilizados no *front-end*

Os princípios utilizados no *front* e *back end* foram, em parte, diferentes, devido à natureza dos *frameworks* e aplicações. Alguns foram utilizados em ambos, como DRY (*don't repeat yourself*) e KISS (*keep it simple, stupid*), pois são largamente aceitos pela comunidade. Apesar da sessão se referir ao *front-end*, os conceitos desta e de outras podem ser sinérgicos ou até similares. A organização foi feita desta forma, pois as ideias foram retiradas da orientação na documentação do React e do livro “Composing Software” (ELLIOT, 2018), que se referem especificamente ao Javascript. Abaixo, estão listados os principais destes conceitos:

- Composição acima de Herança: herança é a forma mais restritiva de reaproveitamento de lógica em programação orientada a objetos. Se baseando em relações do tipo “é um”(ex: pombo é um pássaro). O que leva a diversos problemas, como duplicação por necessidade, o problema gorila/banana e o problema da classe base frágil. Existem três formas principais de composição de objetos: agregação, quando um objeto é propriedade de outro, concatenação, quando um objeto é formado ao adicionar novas propriedades a um objeto, e delegação, quando um objeto delega a outro objeto, ou seja, quando os elementos de um objeto refletem no conteúdo de outro (ELLIOT, 2018). Estes conceitos foram utilizados visando flexibilidade e reutilização eficiente de código.
- Fluxo de dados unidirecional: No React, componentes têm uma hierarquia, em que componentes são encapsulados por outros componentes. Qualquer informação de um componente só pode ser compartilhada com componentes que são encapsulados pelo mesmo. Assim sendo, o fluxo de dados fica em formato de árvore, o que faz com que seja natural a manutenção de um único “detentor da verdade” para qualquer fragmento do estado da aplicação. Esta prática simplifica muito o processo de *debug* (REACT, 2021).
- Imutabilidade de dados: Uma fonte de *bugs* muito prevalente no Javascript em geral são mutações em objetos. Quando se passa um objeto para uma função que precisa alterar os valores passados no objeto, na verdade, está se passando uma referência àquele objeto. Caso o objeto seja alterado durante aquela função, ele será modificado em todos os lugares, mesmo que tenha sido passado como

parâmetro para a função, ou que o objeto seja atribuído a outra variável, pois o que é atribuído à nova variável é a referência, não o objeto em si. Uma boa prática, apoiada pela documentação do React, é se abster de utilizar mutação completamente (ELLIOT, 2018).

- **Funções puras:** funções puras são funções que, dados os argumentos de entrada, sempre têm o mesmo retorno e não possuem efeitos colaterais. Para que estas características sejam alcançadas, a função não deve possuir estado interno (isto é possível em Javascript, pois toda função é um objeto), nem interagir com APIs ou variáveis de estado a que ela possa ter acesso. Efeitos colaterais são, por exemplo, chamadas de API, modificação de estado, gravação de dados, ou qualquer interação com o meio externo que não seja por meio dos argumentos da função. Priorizar funções deste tipo facilita muito a composição de software, testabilidade, e até legibilidade (ELLIOT, 2018).

3.2.5 Princípios SOLID

SOLID é um acrônimo para cinco princípios principais promovidos por Robert C. Martin (MARTIN, 2002), aplicados a programação orientada a objetos. O intuito é tornar o código mais flexível, gerenciável e inteligível. Estes princípios nem sempre são possíveis de serem aplicados por causa de limitações de implementação ou arquitetura, e uns são mais importantes que outros, dependendo da aplicação, mas é importante levá-los em consideração durante o desenvolvimento. Os cinco são apresentados abaixo:

- *Single responsibility principle:* Cada módulo, classe ou função deve ter responsabilidade sobre uma única parte da funcionalidade do programa. Outra forma de dizer é que uma classe deve ter somente uma razão para mudar. (MARTIN, 2002). Se uma classe tem mais de uma responsabilidade, é possível que uma mudança decorrente de uma necessidade criada por uma das responsabilidades impeça a outra de operar, ou mesmo simplesmente interfira com a outra temporariamente, causando mudanças inesperadas no sistema.
- *Open-closed principle:* “Entidades de software (classes, módulos, funções)” devem ser abertas a extensão, mas fechadas a modificação” (MARTIN, 2002). O objetivo é criar módulos que nunca mudam. Se as requisições mudarem, o código já existente deve ser mantido e novo código deve ser adicionado. Deve ser notado que nenhum programa real será completamente fechado a modificação, então estratégias devem ser adotadas para implementar o princípio de forma eficaz.
- *Liskov substitution principle:* “funções que fazem referência a classes base devem ser capazes de utilizar objetos de classes derivadas sem saber disso” (MARTIN,

2002). Para que isso seja possível, uma regra simples pode ser utilizada ao estender o comportamento de objetos: as pré condições da classe/protótipo estendido devem ser mais fracas que as da classe/protótipo base, enquanto as pós condições devem ser mais fortes.

- *Interface segregation principle*: “clientes não devem ser forçados a depender de interfaces que não utilizam” (MARTIN, 2002). Este princípio não foi necessário durante a implementação, pois a herança foi evitada durante o desenvolvimento em favor de composição.
- *Dependency inversion principle*: “módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações”. E “abstrações não devem depender de detalhes, detalhes devem depender de abstrações” (MARTIN, 2002). De forma mais prática, isto pode ser utilizado como a separação do programa em camadas, em que as camadas se comunicam entre si por meio de interfaces bem definidas. Assim, tanto componentes de alta quanto de baixa ordem podem ser reutilizados, mudanças em uma camada que respeitem a interface não causarão problemas em outras, o que facilita futuras alterações no código.

3.3 SERVIDOR E INFRAESTRUTURA

3.3.1 Amazon Web Services

AWS (Amazon Web Services) é a maior plataforma de serviços de computação em nuvem do mundo no momento de digitação deste documento. Em 2017, AWS possuía 33 por cento de toda a *cloud*, comparado com seus competidores Microsoft Azure (18 por cento) e Google Cloud (9 por cento).

Ela fornece uma gama variada de soluções de tecnologia, como máquinas virtuais, banco de dados extensível, inteligência artificial, armazenamento de dados, IoT, envio de e-mail e SMS, e muitos outros. A integração de muitos subsistemas é muito facilitada, com protocolos de segurança, padrões de implementação, distribuição de carga, e outras partes importantes de implementação já convenientemente estruturados. Por isso, toda a plataforma está hospedada na plataforma da AWS.

Houve um processo de simplificação da estrutura da AWS. O IoT Core e o DynamoDB não estão mais sendo utilizados, pois foram substituídos por uma rotina dentro do Laravel. A razão para esta substituição foi redução de custos.

3.3.2 AWS RDS

AWS RDS é o banco de dados relacional da Amazon. É onde está hospedado o servidor de MySQL utilizado pela plataforma Dayback Energy. O RDS proporciona:

- elasticidade: aumento ou diminuição de espaço disponível
- segurança para os dados: *backups* periódicos
- controle de acesso: somente máquinas certificadas tem acesso ao BD
- independência de servidor: o servidor pode ser migrado livremente
- distribuição: mais de um servidor pode ter acesso ao mesmo BD.

Todas são características desejáveis para um produto que respeite o requisito de flexibilidade e segurança.

3.3.3 S3

AWS S3 é um sistema de contêineres de armazenamento de dados. Ele possui facilidade de integração com outros serviços da AWS.

Atualmente, o Dayback Energy o utiliza para guardar imagens do *site*, que não poderiam ser salvas eficientemente com nenhuma das outras soluções de armazenamento descritas neste documento. As grandes vantagens de utilizar este serviço são:

- Independência de máquina: poder trocar de máquina virtual livremente
- Segurança: os arquivos não vivem no servidor, o que impede o armazenamento de software malicioso em ambiente sensível.

3.3.4 EC2

O EC2 é o serviço de hospedagem de máquinas virtuais da AWS. O servidor que contém a aplicação é executado em uma destas máquinas. A principal vantagem de se utilizar este serviço é, a facilidade de integração com o resto dos serviços da Amazon. Dentro de cada serviço, é necessário que se configure políticas de utilização, em que somente clientes com determinadas credenciais podem ter acesso à funcionalidade. Com uma máquina virtual da Amazon, todos os passos de autorização ficam mais fáceis.

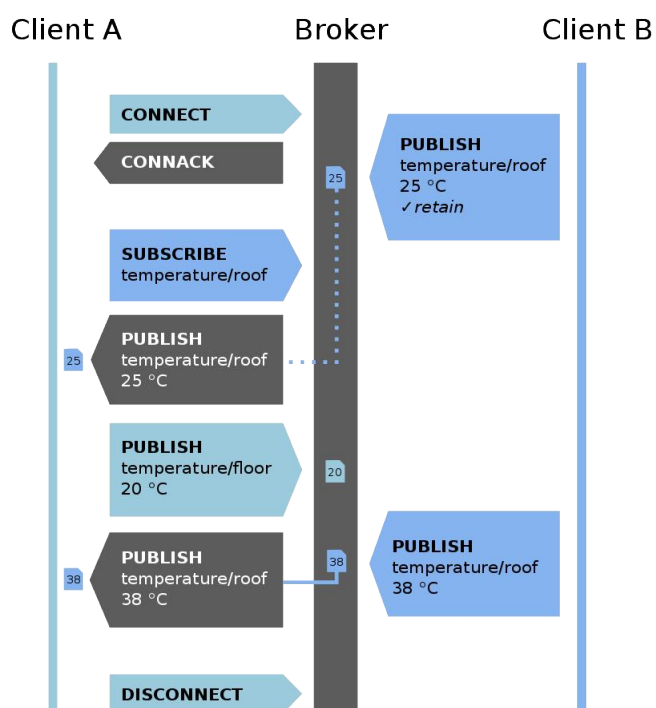
O EC2 também oferece a possibilidade de reservar um IP estático, o que é essencial para o gerenciamento de DNS (Domain Name System) e permissões. O tamanho da máquina virtual também é facilmente configurável, com opções de memória RAM e processamento para suprir quase qualquer demanda quando for necessário.

Atualmente, há duas máquinas em funcionamento no EC2. Uma para o servidor do Dayback Energy e uma para o broker MQTT.

3.3.5 MQTT broker

O Message Queuing Telemetry Transport (MQTT) é um protocolo de rede do tipo publicação-subscrição. Devido à sua "leveza" em relação à quantidade de código, é adequado para a comunicação com dispositivos colocados em localização remota ou com baixa disponibilidade de rede (OASIS, 2021).

Figura 8 – Exemplo de comunicação com MQTT



Fonte: Arquivo pessoal.

O funcionamento se dá como representado na Figura 8. Os dispositivos se conectam ao *broker* como clientes pelo protocolo Transmission Control Protocol (TCP) e se inscrevem em um ou mais canais, chamados de tópicos. Clientes inscritos podem realizar duas ações:

- Enviar mensagens no tópico
- Todos que estão conectados ao tópico recebem as mensagens.

O protocolo foi utilizado para fazer a comunicação entre os dispositivos IoT e a plataforma em tempo real e também para comunicar o DBHW com o gerenciador de IoT da AWS.

Foi criada uma máquina virtual com Linux no EC2 da AWS para conter o *broker*. A implementação se deu com o Mosquitto *broker*. Optou-se pelo Mosquitto por este ser muito leve e ser de código aberto, o que não acarreta em custos adicionais para o

projeto. Outra vantagem é que a documentação da Amazon tem instruções de como realizar esta implementação.

3.3.6 AWS SES

O AWS SES é o *Simple Email Service*. Um serviço da AWS de envio de e-mail. Toda a parte de envio automático de e-mail da plataforma utiliza este serviço. Ele possui uma tolerância de 5000 envios de e-mail gratuitos por mês, o que o torna muito bom para uma *start up*.

3.3.7 Telegram

O Telegram foi utilizado como veículo para as notificações de consumo excessivo. Para cada ponto de notificação, foi criado um grupo do telegram. Em cada um deles, foi adicionado um telegram *bot*. No servidor, foi instalada a biblioteca irazasyed/telegram-bot-sdk. Ao pegar o *token* do *bot* pelo próprio telegram, foi possível configurá-lo no servidor, fazendo a conexão entre o *bot* e a aplicação.

4 PROJETO E ARQUITETURA

Este capítulo é dedicado a especificar os requisitos funcionais e não funcionais do sistema e apresentar o projeto das partes a serem implementadas.

Vale ressaltar que o autor deste documento foi responsável pelo projeto e desenvolvimento integral da aplicação, recebendo auxílio na implementação do gerador de relatórios nos últimos ciclos de desenvolvimento.

4.1 REQUISITOS FUNCIONAIS

Abaixo estão listados os requisitos funcionais levantados:

- Criação de Tela de Status de Monitoramento
- nova versão do MQTT e DBHW
 - Armazenar dados do MQTT no banco de dados
 - Algoritmo de recuperação de dados perdidos em caso de perda de conexão
 - Criar ferramenta de visualização de dados em tempo real
- Criação de gerador semi automático de relatório
 - Adicionar e remover páginas
 - Criar gráficos com dados de clientes e colocá-los no PDF
 - Criar opção de comparação de dados diários
 - Permitir download de PDF
 - Possuir pré visualização do PDF
- Melhorar a interface
 - Manter dados atualizados
 - Criar modo de televisão para a *dashboard*
 - Tornar telas customizáveis

4.2 REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais estabelecem como o projeto deve ser implementado, o que deve garantir, que tipo de infraestrutura deve ser utilizada e que metodologia de desenvolvimento deve ser adotada. Os requisitos não funcionais são:

Confiabilidade do armazenamento: os dados devem ser armazenados de forma segura por tanto tempo quanto o contrato estipular. A confiabilidade deve ser de 100%.

Robustez : o sistema deve ser capaz de superar falhas nos pontos de medição, como quedas de energia e perda de sinal wi-fi.

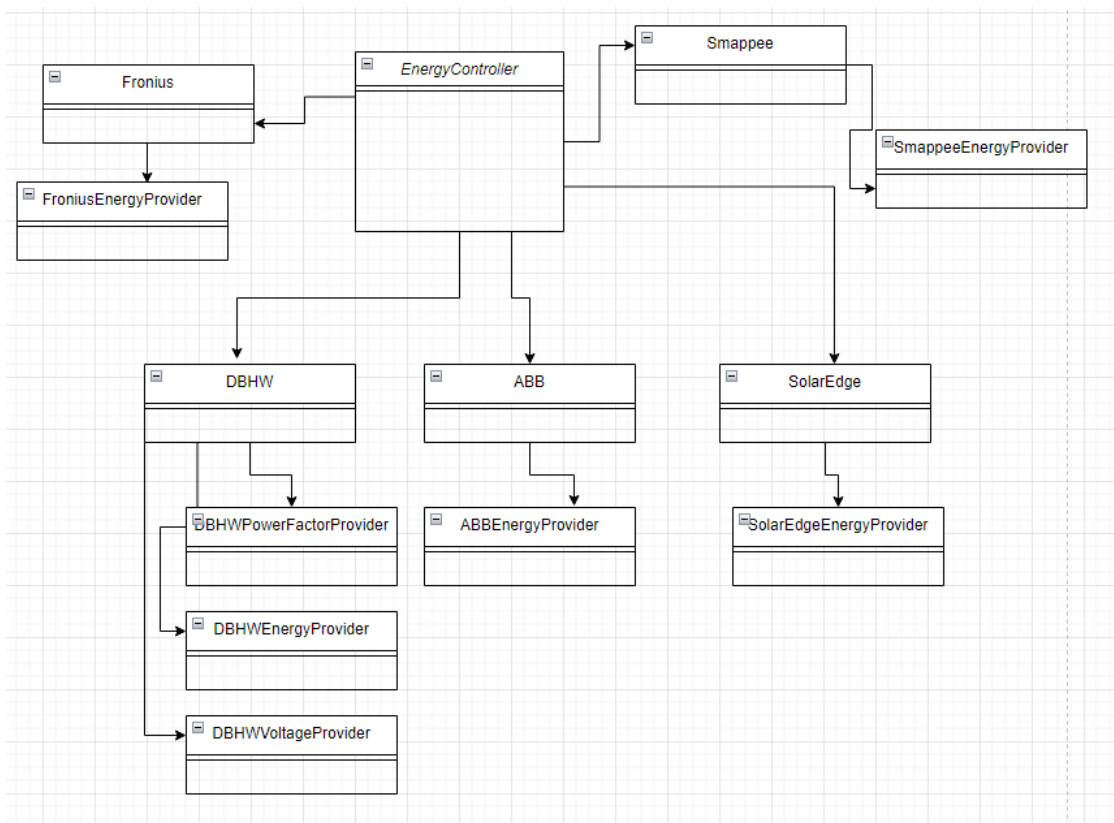
Flexibilidade : o software não deve se limitar a uma única fonte de dados, portanto, uma API ou um hardware IoT deve ser intercambiável com outros.

Escalabilidade : possuir arquitetura de software modular e escalável, aberta a ampliações futuras

4.3 ORGANIZAÇÃO DE SERVIÇOS DE INTEGRAÇÃO

Existe um serviço de integração para cada tipo de fonte de dados, que são as APIs da Fronius, Smappee, ABB, Solar Edge e os dados do banco de dados do DBHW. Cada serviço tem acesso às credenciais de acesso à API ou banco de dados e estabelece a conexão com a fonte de dados. Adicionalmente, cada um possui provedores, que são responsáveis por fazer as requisições especificadas na requisição http. O diagrama de classes é definido na figura 45

Figura 9 – Diagrama de integração



Fonte: Arquivo pessoal.

Nas seções 5.2 a 5.6 são dadas breves descrições dos serviços.

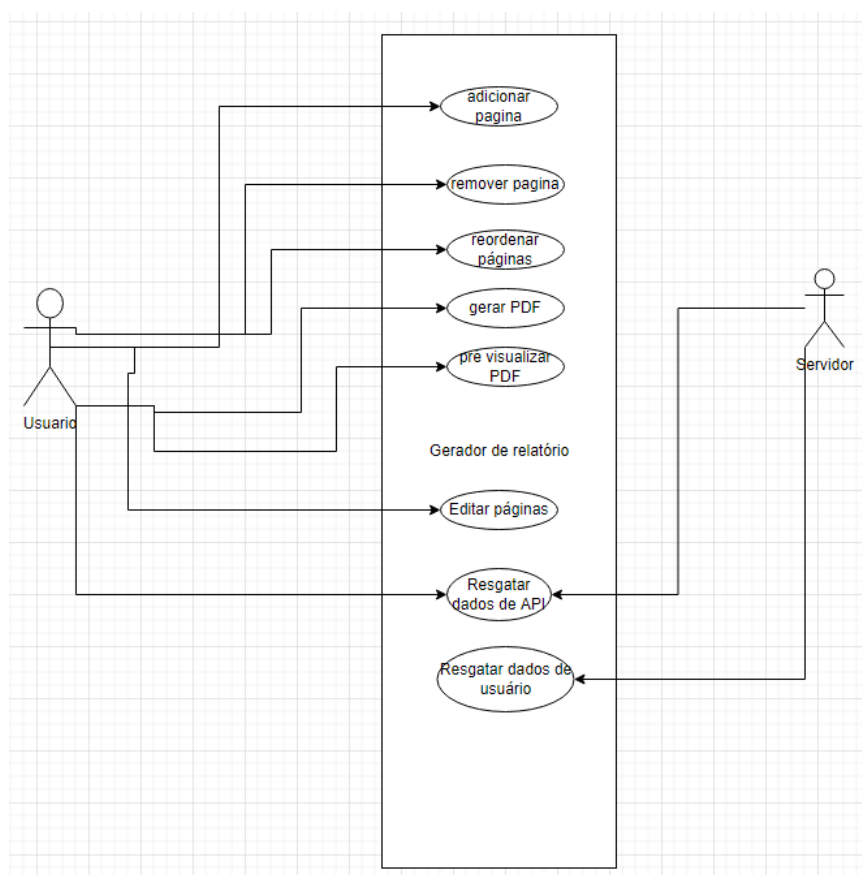
4.4 PROJETO DE COMPONENTES FRONT-END

4.4.1 Gerador de relatórios

As páginas a serem criadas pelo gerador de relatórios existiam como um modelo de relatório da empresa. O modelo específico utilizado para a criação das páginas está mostrado no anexo A.

O diagrama de casos de uso, que relaciona os atores com as ações possíveis do gerador de relatório está mostrado na Figura 10.

Figura 10 – Diagrama de casos de uso do gerador de relatórios



Fonte: Arquivo pessoal.

Na Figura 11, tem-se o diagrama de classes simplificado do gerador de relatórios. O *ReportMaker* é o componente responsável por criar o esqueleto do gerador de relatórios e gerenciar seus componentes. Ele *renderiza* os formulários em um lado da tela e compila as páginas, imprimindo-as no outro lado da página como PDF.

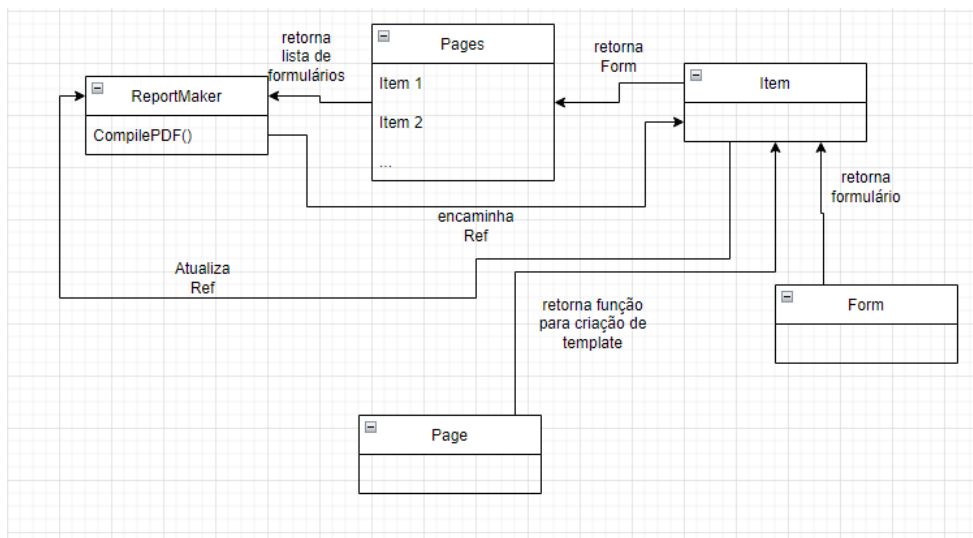
Os dados necessários para a compilação do PDF, como as imagens, não devem fazer parte do estado da aplicação, de modo a evitar atualizações excessivas à interface gráfica. Portanto, coloca-se os dados necessários para a compilação em uma *Ref*

do *React*, que é utilizada como um objeto javascript comum que não será redefinido a cada ciclo de vida da aplicação.

Cada *Item* recebe um identificador único e é adicionado a uma lista de páginas. Eles são responsáveis por gerenciar o conteúdo da parte da *Ref* associada ao seu identificador. Cada um possui um formulário, que representa a parte gráfica do gerenciamento do conteúdo da *Ref*.

As *Pages* são funções

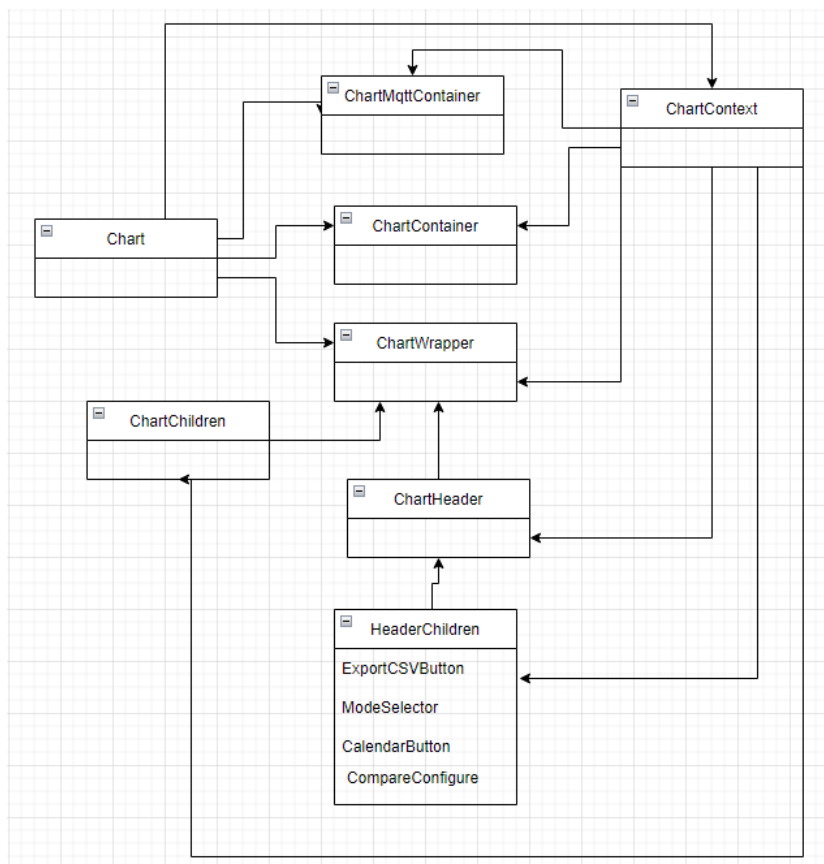
Figura 11 – Diagrama de classes do gerador de relatórios



Fonte: Arquivo pessoal.

4.4.2 Padrão para Componentes

A Figura 12 mostra o diagrama de classes da *Chart*. Esta lógica foi também utilizada em outros componentes, como o calendário e os cartões com valores consolidados. A ideia é que os componentes sejam arquitetados de forma que funcionem como um pacote de node (*npm package*), sendo assim independentes do resto do *software* e muito configuráveis.

Figura 12 – Diagrama de classes da *Chart*

Fonte: Arquivo pessoal.

O componente básico é representado na Figura 12 pelo *Chart*, que declara o contexto e as variáveis de estado globais ao componente. Os elementos *Container* são responsáveis por prover dados ao gráfico. Uma opção de *Container* pode ser declarada aninhada à *Chart* no momento de instanciação.

Os *headers*, declarados no *ChartHeader*, são responsáveis por configurar os dados do componente. O controle do estado do componente por parte do usuário é dado pelos componentes colocados no *header*.

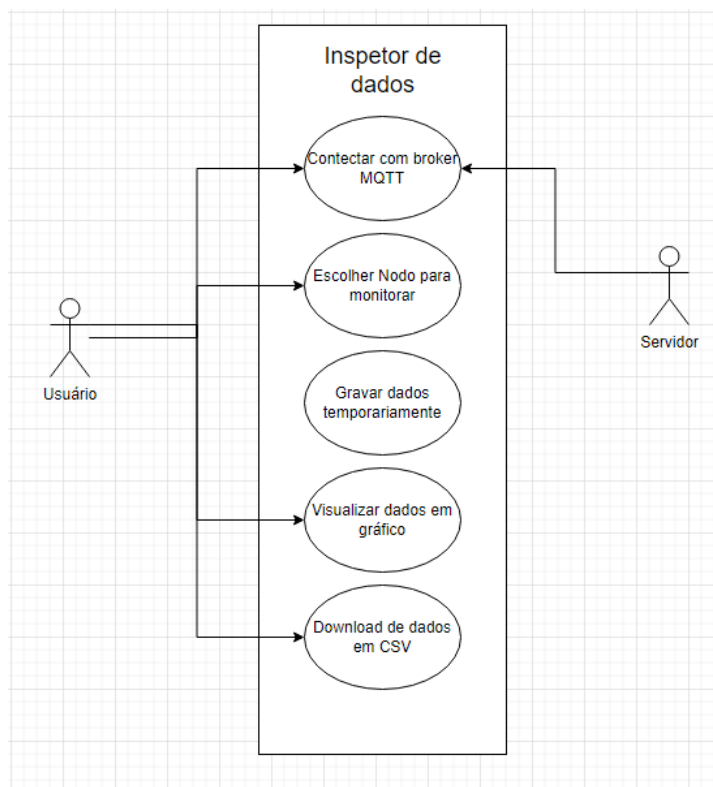
Por fim, o *ChartWrapper* é um componente puramente visual que define os elementos gráficos básicos do componente e aceita elementos aninhados como propriedade.

Esta lógica permite que qualquer componente seja configurável em nível de fonte de dados, aparência visual e opções de interatividade.

4.4.3 Ferramenta de inspeção de dados em tempo real

A ferramenta de inspeção de dados em tempo real possui os casos de uso mostrados na Figura 13.

Figura 13 – Diagrama de casos de uso do gerador de relatórios



Fonte: Arquivo pessoal.

A ferramenta deve permitir conexão com o *broker MQTT* do DBHW e mostrar dados de potência, fator de potência, tensão, corrente frequência em um gráfico. Os dados a serem apresentados devem ser configuráveis e deve ser possível exportá-los em um arquivo CSV.

4.4.4 Tela de Status

As seguintes especificações foram levantadas para a tela de status:

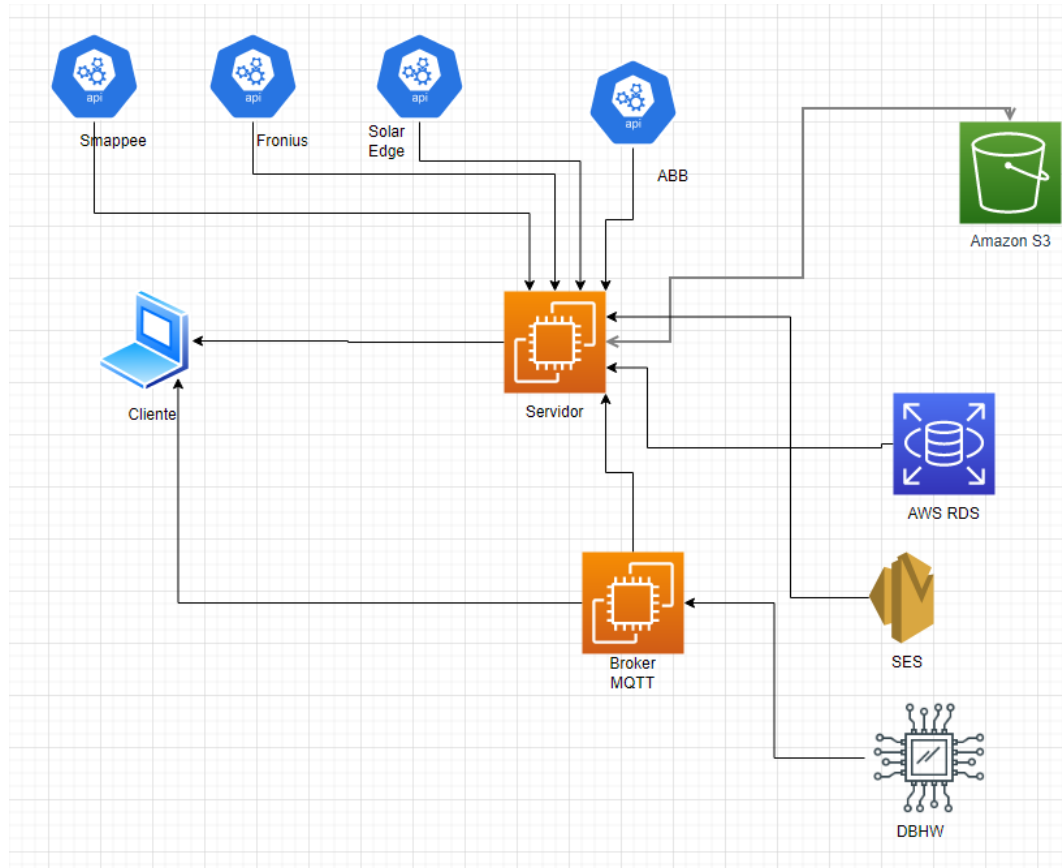
- A tela de status deve mostrar todos os nodos de medição e qual foi a última vez que foram recebidos dados associados a ele. Caso faça mais de 24 horas desde a última medição, o status deve ser o mais alarmante possível.
- Os nodos devem estar agrupados por tipo de medidor.

5 INTEGRAÇÃO

Neste capítulo são descritos os equipamentos e APIs utilizados no projeto e como eles foram integrados à plataforma.

A Figura 14 mostra o fluxo de requisições de serviços do Dayback Energy.

Figura 14 – Diagrama de integração



Fonte: Arquivo pessoal.

Houve uma simplificação da infraestrutura relativa ao estado anterior da plataforma. Isto teve como motivação a redução de custos e simplificação de processos. O MQTT não é mais processado no IoT Core, mas sim por um processo do Laravel. O DynamoDB não é mais utilizado, pois os dados são gravados diretamente no AWS RDS.

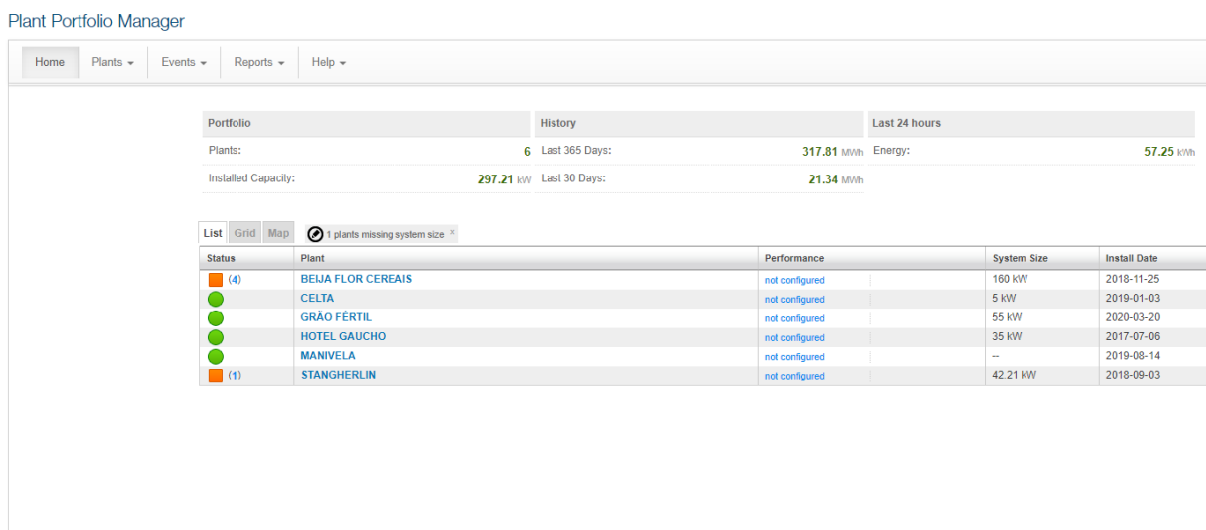
5.1 ABB AURORA VISION

ABB é a fabricante dos inversores que foram utilizados na integração dos painéis solares com o software. Como o próprio inversor já vem com um transmissor de

dados, não foi necessária nenhuma implementação de medição adicional. Isto seria necessário caso fosse desejado saber a geração em tempo real.

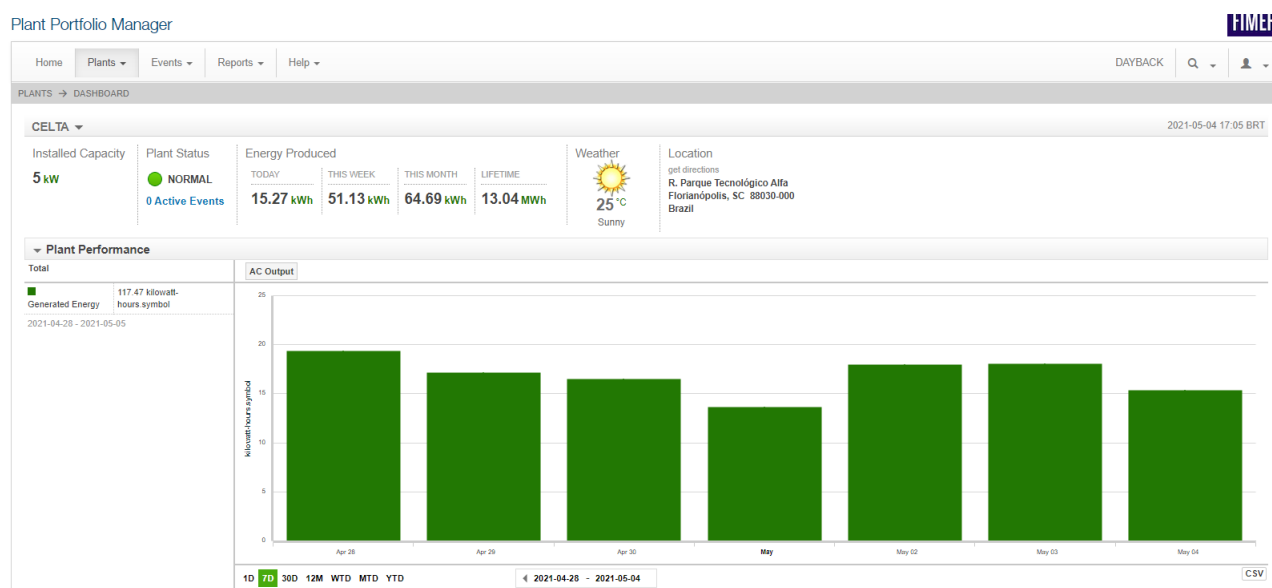
Os dados do inversor vão para o sistema de IoT da ABB, que salva os dados e os disponibiliza na nuvem através da plataforma Aurora Vision. Uma mesma conta na ABB pode conter várias plantas de energia solar, como mostrado na Figura 15.

Figura 15 – *Dashboard* de plantas ABB



Fonte: Arquivo pessoal.

Ao selecionar uma planta, as informações coletadas aparecem em uma *dashboard*, que foi muito útil para comparação de dados durante a validação da plataforma Dayback Energy (Figura 16).

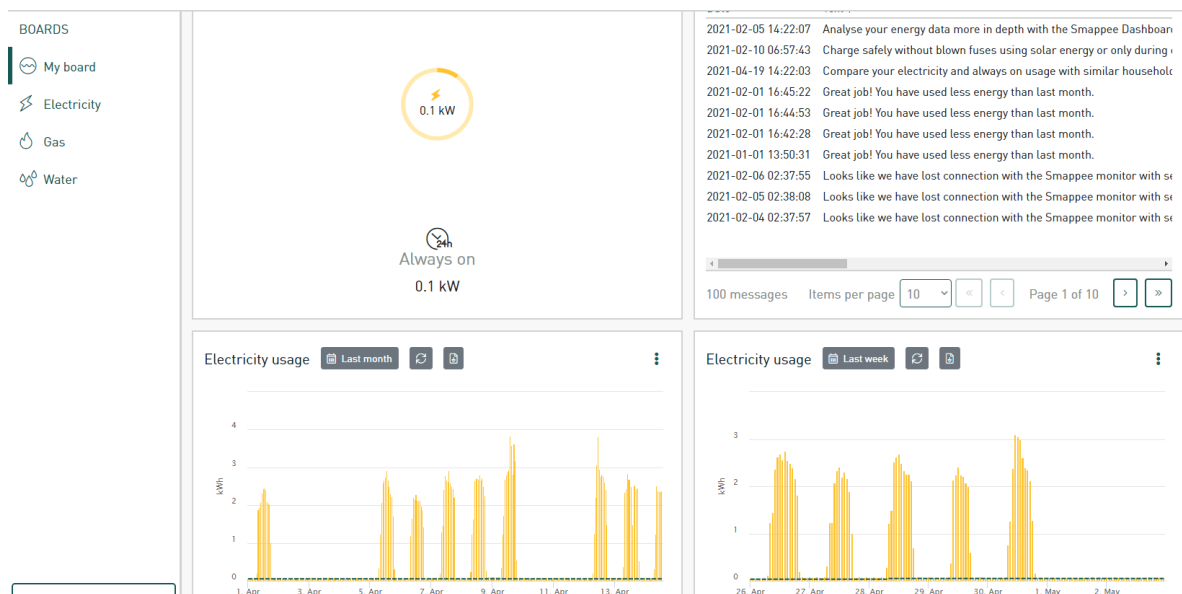
Figura 16 – *Dashboard* do cliente ABB

Fonte: Arquivo pessoal.

Além da *dashboard*, a ABB disponibiliza uma API, que possibilita a transferência de dados para o sistema Dayback Energy.

5.2 SMAPPEE

Smappee é um dispositivo de IoT que lê um medidor e publica os dados para a nuvem. Ele tem suporte para leitura de linha monofásica ou trifásica. Assim como o ABB, o Smappee também disponibiliza uma *dashboard* para acompanhamento de planta (Figura 17).

Figura 17 – *Dashboard* do cliente Smappee

Fonte: Arquivo pessoal.

A solução do Smappee tem muito em comum com a do projeto descrito neste documento, por isso, foi utilizado como exemplo para algumas funcionalidades e também como comparador durante a verificação de resultados.

O Smappee fornece uma API, que foi utilizada tanto para a *dashboard* do Dayback Energy em período de teste quanto para salvar os dados no banco de dados. Outra utilidade do Smappee é a possibilidade de enviar dados em tempo real por protocolo MQTT.

5.3 DBHW

O Dayback Hardware é o hardware de IoT da Dayback. Ele registra leituras de potência, corrente, tensão e fator de potência e envia por protocolo MQTT para um broker MQTT situado em uma EC2 na AWS.

O DBHW fornece dados de Tensão RMS, corrente RMS, frequência, fator de potência e potência instantânea em tempo real para cada uma das fases. Um pacote de dados está mostrado na Figura 18.

Figura 18 – Payload instantâneo

```
{
  "TIMESTAMP": 1646343255,
  "DATA": {
    "IRMS_F1": "2.86",
    "IRMS_F2": "2.87",
    "IRMS_F3": "2.91",
    "VRMS_F1": "228.41",
    "VRMS_F2": "227.48",
    "VRMS_F3": "230.31",
    "FREQ_F1": "60.15",
    "FREQ_F2": "59.88",
    "FREQ_F3": "59.94",
    "PF_F1": "0.96",
    "PF_F2": "0.96",
    "PF_F3": "0.96",
    "AP_F1": "654.36",
    "AP_F2": "653.56",
    "AP_F3": "671.01",
    "RP_F1": "625.91",
    "RP_F2": "626.04",
    "RP_F3": "643.95",
    "CORE TEMP": "85.0",
    "CONNECTED TO": "Drummer Office",
    "WIFI SIGNAL": "50",
    "IP": "192.168.1.2"
  }
}
```

Fonte: Arquivo pessoal.

O pacote de dados consolidado fornece dados de fator de potência médio, potência média(kW), valor do relógio de energia(kWh), e tensão média(V) para cada uma das fases (Figura 13).

Figura 19 – Payload consolidado

```
{
  "TIMESTAMP": 1646343196,
  "DATA": {
    "PF_F1": "0.96",
    "PF_F2": "0.96",
    "PF_F3": "0.96",
    "RP_F1": "616.98",
    "RP_F2": "627.82",
    "RP_F3": "648.84",
    "KWH_F1": "20.23",
    "KWH_F2": "20.80",
    "KWH_F3": "20.90",
    "VRMS_F1": "228.62",
    "VRMS_F2": "227.90",
    "VRMS_F3": "230.31"
  }
}
```

Fonte: Arquivo pessoal.

Os dados provenientes do pacote de dados consolidado são processados e armazenados no banco de dados, enquanto os dados de tempo real só são guardados

utilizando uma ferramenta de administrador da plataforma que permite gravar os dados de MQTT e mostrá-los em uma tabela, da qual eles podem ser exportados para CSV.

Em caso de perda de conexão, o DBHW armazena os dados em um "buffer", que é enviado pelo MQTT assim que a conexão é reestabelecida. Esses dados são processados pela rotina de gravação de dados e salvos. Subsequentemente, uma mensagem é enviada ao tópico correspondente ao equipamento sinalizando que os dados foram salvos com sucesso. O Hardware então libera o espaço no "buffer" e continua a funcionar normalmente.

5.4 FRONIUS

Fronius, assim como a ABB, é um fabricante de inversores. Os inversores da Fronius também vêm com uma solução de IoT embutida. A placa de integração se conecta à rede Wi-fi local e envia os dados para a nuvem, de onde podem ser resgatados por meio de uma API. A Dayback possui algumas instalações que utilizam inversores Fronius.

Uma desvantagem da API da Fronius é que ela só possui opção de enviar médias de energia de 15 em 15 minutos. Portanto, exige mais manipulação por parte do serviço de integração, além de ter resolução menor do que as outras. Outra desvantagem é que caso períodos maiores forem requisitados, como uma semana com resolução dia a dia, o tamanho da resposta será muito maior, o que torna o serviço mais lento. Isto significa que há uma demanda forte para que se transfira os dados para o banco de dados local. Esta transferência ainda não está sendo realizada no momento, pois a integração foi realizada há pouco tempo.

5.5 SOLAR EDGE

A Solar Edge é outro fabricante de inversores. Algumas das instalações solares da Dayback são realizadas utilizando inversores Solar Edge. A API da Solar Edge é bastante adequada à plataforma. O serviço que faz a requisição de API é bastante simples, pois o formato dos dados são aproximadamente compatíveis com o padrão da plataforma.

5.6 ARMAZENAMENTO

Cada uma dessas APIs possuem interfaces diferentes, o que exige que a plataforma encapsule a classe responsável pela chamada de API em um adaptador para cada uma delas. Cada adaptador é um serviço que recebe o "Model" instanciado correspondente ao dispositivo sobre o qual se deseja obter os dados. Depois, obtém do banco de dados as credenciais necessárias para interagir com a API, então inicia

o cliente HTTP e realiza a requisição com os parâmetros necessários e aguarda a resposta. Por fim, caso a resposta tenha sido válida, transforma o pacote de dados para respeitar uma interface interna de formato de dados e finaliza o cliente HTTP, retornando a resposta.

Todas as APIs, com exceção da Fronius, já possuíam suporte para enviar dados com resolução de 5 em 5 minutos, mas não havia garantias de que os dados seriam salvos nesta resolução no banco de dados deles pelo mesmo período que a Dayback se comprometeu a salvar. Para manter a flexibilidade com relação a isso, foram criadas rotinas que salvam os dados com resolução de 5 em 5 minutos no banco de dados no AWS RDS.

As rotinas de salvar os pacotes provenientes das APIs são realizadas de 5 em 5 minutos. O Laravel vem com suporte para agendamento de comandos. Cada vez que o comando agendado é executado, uma lista de eventos referente aos processos de salvamento de dados é emitida e colocada em uma fila. Uma vantagem de se fazer isso é que cada um dos comandos pode ser tratados individualmente, sem risco de que um erro não esperado comprometa o funcionamento de todos os processos.

A fila é processada por processos assíncronos chamados "Workers", que procuram o tratador de evento correspondente. Os *workers* devem ser configurados no servidor por um serviço gerenciador de processos do Linux chamado "Supervisor". O número máximo de "Workers" é 16. É importante que os processos sejam assíncronos, pois as APIs possuem uma latência que pode vir a se tornar um problema caso uma grande quantidade de chamadas forem feitas e o período total de operação do comando superar 5 minutos. O resultado do processo é armazenado no banco de dados.

5.7 INTERFACE PADRONIZADA

As APIs não seguem nenhum padrão específico de pacote de dados. Se não fosse criado um padrão de dados de energia, teria que haver diferenças na forma de tratar cada um dos *payloads* de cada uma das APIs em todas as camadas desde a requisição dos dados até a visualização no *front-end*, tornando o código desnecessariamente complicado, não escalável e difícil de manter. Para solucionar este problema, foram adicionadas interfaces entre as camadas da aplicação que isolam cada um dos problemas, fazendo com que partes do código que dependem de outras não dependam da implementação, mas sim da interface.

A Figura 20 mostra a interface do lado do *front-end* que define como as requisições dever ser feitas. Nota-se que os parâmetros para requisições de qualquer fonte obedecem o mesmo padrão. "from" e "to", são UNIX *timestamps* que representam o instante de início e o de final de quando os dados devem ser resgatados. Uma vantagem muito grande de se utilizar *timestamps* é que eles são padrão para o mundo

inteiro, não havendo preocupação com fuso horário, já que ele representa a quantidade de milisegundos desde o primeiro de 01/01/1970 GMT. O *node id* é o identificador do nodo, ou ponto de medição que está sendo utilizado, percebe-se que ele não se importa com qual dispositivo ele vai acessar. O ponto de medição é uma referência a um local físico, pouco importando qual equipamento faz a leitura, o que possibilita a troca de equipamento sem a perda de organização de dados. *Aggregation* se refere ao espaçamento temporal entre as medições, como 5 em 5 minutos, hora em hora. *Strategy* é uma opção que flexibiliza a interface permitindo que informações extra acompanhem o *payload*. Por fim, o *cancelToken* é um objeto que é acoplado ao *axios* e faz com que seja possível cancelar requisições quando elas não forem mais utilizadas, como no caso em que um usuário requer dados de energia várias vezes repetidamente, mas só tendo interesse no resultado da última requisição.

Figura 20 – utilidade de requisição de dados *front-end*

```
import axios from 'axios';

export function getData({from, to, node_id, strategy, aggregation, cancelToken=null}){
  let url = '/device/' + node_id + '/request?from=' +
    from + '&to='+ to + '&strategy=' + strategy +
    '&aggregation='+aggregation+(aggregation > 1 ? '' : '&unit=power')

  // url = 'http://usuario.dayback.com.br' + url

  return axios.get(url, {cancelToken: cancelToken})
    .then( resp => {
      return resp
    })
    .catch((e) => {
      if(axios.isCancel(e) ) {
        console.log(e.message, 'request was cancelled');
      }
    })
}
```

Fonte: Arquivo pessoal.

O *Controller*, é o elemento que recebe a requisição *HTTP*. A função do Controller é chamar o serviço que deve entregar o resultado desejado. Cada serviço corresponde a um tipo diferente de medidor. Para o Smappee e ABB existe a opção de se obter dados da API ou do banco de dados, para a Fronius e Solar Edge, existe somente a opção da API por enquanto, e o DBHW possui somente a opção de banco de dados. Cada um dos serviços tem sua forma de tratar os parâmetros definidos na interface representada na Figura 21.

Figura 21 – Controller de Energia

```
9 use Carbon\Carbon;
10
11 class EnergyController extends Controller
12 {
13     public function GetEnergyData (Request $request, $nodeId) {
14         $node = \App\Models\Node::find($nodeId);
15         $device = $node->device;
16
17         switch ($node->device_type) {
18             case 'App\Models\AbbAuroraVision':
19                 try{
20                     $result = \App\Services\AbbAuroraVision\AbbDB::getData($node, $request);
21                 }catch(Exception $e){
22                 }
23
24                 if(isset($result) && $result != []){
25                     return $result;
26                 }
27
28                 $sabb = new \App\Services\AbbAuroraVision\AbbAuroraVisionContext($device);
29                 return $sabb->handleRequest($request);
30                 break;
31             case 'App\Models\Smappee':
32                 try{
33                     $result = \App\Services\Smappee\SmappeeDB::getData($node, $request);
34                 }catch(Exception $e){
35                 }
36
37                 if(isset($result) && $result != []){
38                     return $result;
39                 }
40
41                 $smappee = new \App\Services\Smappee\SmappeeContext($device);
42                 return $smappee->handleRequest($request);
43                 break;
44             case 'App\Models\Dbhw':
45                 $dbhw = new \App\Services\Dbhw\DbhwService($node);
46                 return $dbhw->handleRequest($request);
47                 break;
48             case 'App\Models\Fronius':
49                 return $this->getFroniusData($request, $device);
50             case 'App\Models\SolarEdge':
51                 return $this->getSolarEdgeData($request, $device);
52         }
53     }
54 }
```

Fonte: Arquivo pessoal.

6 IMPLEMENTAÇÃO DE TELAS E FERRAMENTAS

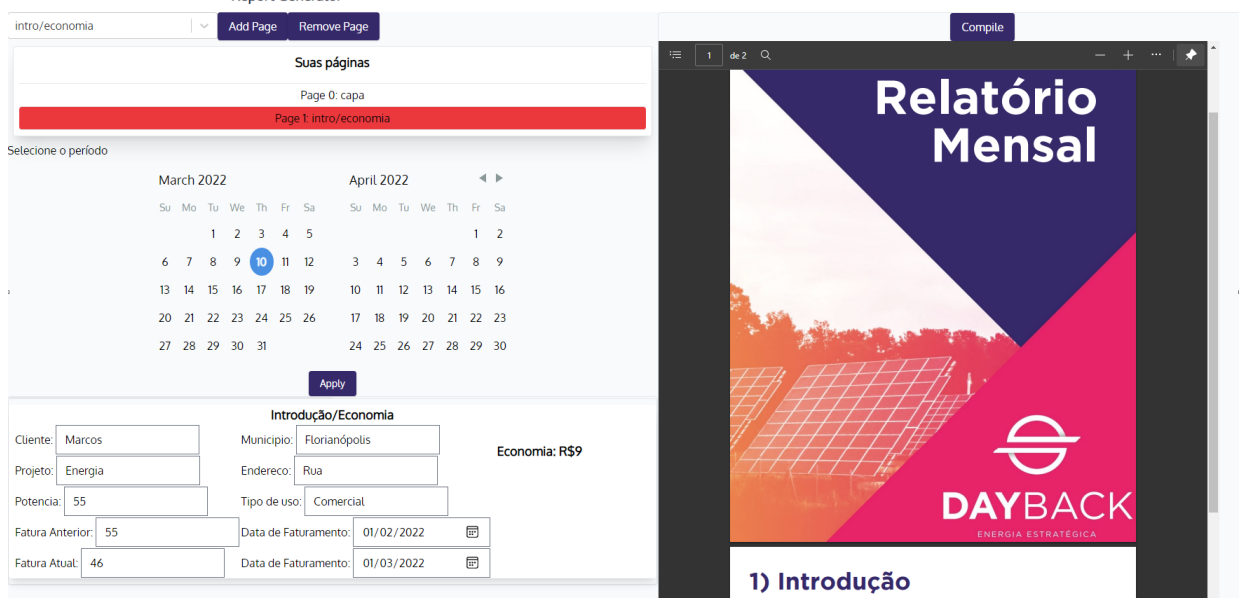
Neste capítulo são descritas as novas funcionalidades desenvolvidas durante o projeto. O foco é descrever como foram implementados os componentes.

6.1 GERADOR DE RELATÓRIOS

O gerador de relatórios é uma ferramenta criada para gerar arquivos PDF a partir de templates editáveis, que podem incluir diferentes tipos de gráficos e imagens e informações úteis sobre a empresa e ações necessárias. A biblioteca de javascript utilizada para transformar os templates em PDF se chama "labelmake". A Figura 22 demonstra como o gerador funciona. À esquerda, ficam os formulários e à direita uma pré-visualização do relatório. O conteúdo do PDF é atualizado quando o botão *compile* é pressionado.

Os formulários editam valores no PDF, servindo para configurar dados específicos do cliente para o qual se deseja fazer o relatório. O PDF é visualizado em um *iframe* (elemento HTML), possuindo funcionalidade de se fazer *download*.

Figura 22 – Página de gerador de relatórios



Fonte: Arquivo pessoal.

Cada uma das páginas é criada por meio de uma função que retorna o componente do formulário e adiciona o *template* a uma *ref*, que funciona como um objeto javascript convencional, não sendo redefinido a cada ciclo de vida do componente. Esta *ref* é atualizada por meio de *hooks*, como mostrado na Figura 23. Cada vez

que o objeto *data* for atualizado, a *ref* será modificada, respeitando o princípio da imutabilidade.

Figura 23 – Exemplo de função geradora de *template*

```
import IntroForm from "../IntroForm";
import IntroPage from "../IntroPage";
import { useState, forwardRef } from "react";

const Intro = forwardRef(({id}, ref) => {
  const [data, setData] = useState({cliente:'', projeto:'', municipio:'', endereco:'', potencia:'', tipoDeUso:''});

  ref.current = { ...ref.current, [id]: IntroPage({data})}

  return(
    <div className='flex border rounded'>
      <IntroForm
        data={data} setData={setData} id={id}
      />
    </div>
  )
})
```

Fonte: Arquivo pessoal.

Cada vez que for pressionado o botão *compile*, será executada a função mostrada na Figura 24. As páginas são numeradas de acordo com a ordem em que elas estão definidas.

Figura 24 – Função para compilar o PDF

```
const compilePDF = async() => {
  //Orders schemas in same order as pages array
  let schemas = {}
  let inputs = []
  pages.forEach((el, index) => {
    const temp = ref.current[el.props.id](index)
    inputs.push(temp.input)
    schemas = {...schemas, ...temp.template}
  })

  const GothamBold = await fetch("GothamBold.ttf").then((res) => res.arrayBuffer());
  const GothamBook = await fetch("GothamBook.ttf").then((res) => res.arrayBuffer());

  const font = { GothamBold,
    GothamBook }

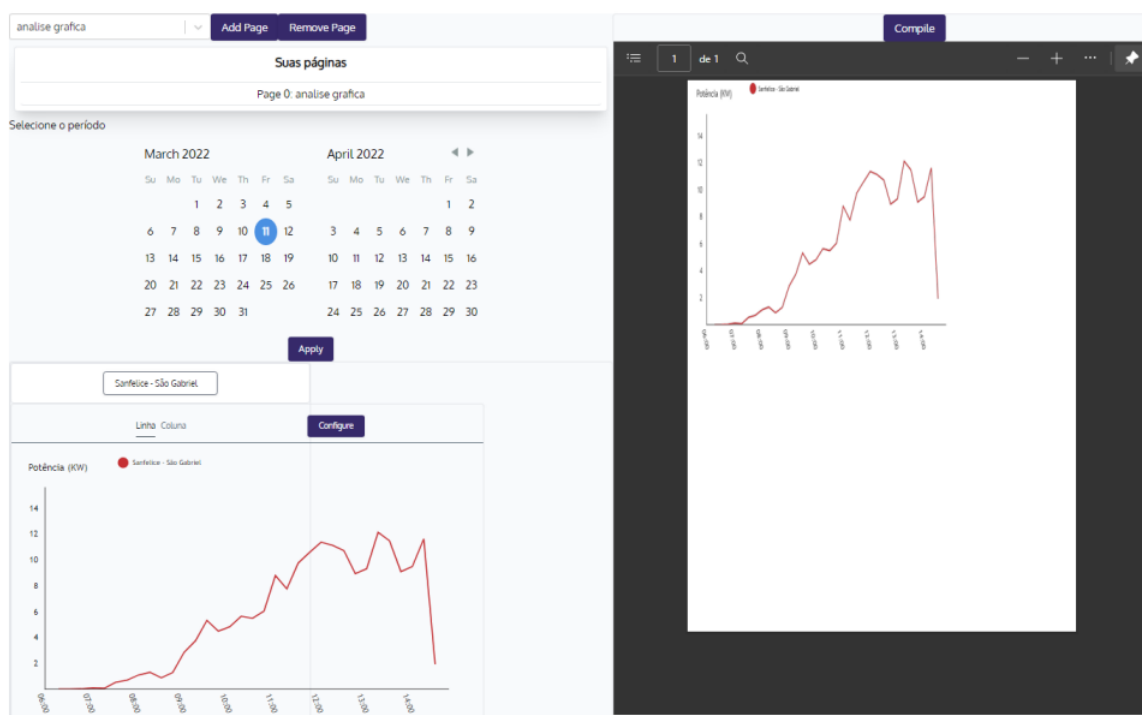
  const template = {
    fontName: 'GothamBook',
    basePdf: { width: 210 * 2, height: 297 * 2 },
    schemas: [schemas]
  };

  const pdf = await labelmake({ template, inputs, font});
  const blob = new Blob([pdf.buffer], { type: "application/pdf" });
  document.getElementById("iframe").src = URL.createObjectURL(blob);
}
```

Fonte: Arquivo pessoal.

O gráfico pode ser adicionado como mostrado na Figura 25. Os gráficos já podem ser inseridos no PDF, porém, as páginas específicas ainda não foram configuradas. O suporte para possibilitar as configurações foi realizado no componente *Chart*, que será descrito na seção 7.2.

Figura 25 – Página de gerador de relatórios



Fonte: Arquivo pessoal.

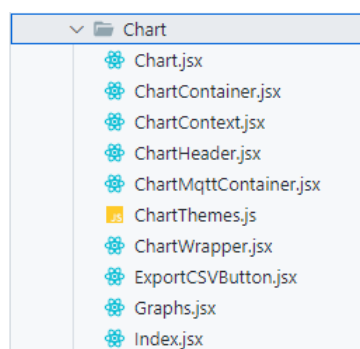
6.2 CHART

A *Chart* é um componente que gera gráficos. Este componente foi utilizado em múltiplas aplicações, exigindo flexibilidade em fonte de dados, estilo, componentes de cabeçalho, como o calendário, botão para fazer *download* de CSV. Para que isso fosse possível, conceitos de composição de software e princípios de programação limpa foram utilizados. Além disso, foi estabelecido um padrão para a confecção de outros componentes customizáveis na plataforma. Esta seção se dedica a explicar o padrão estabelecido e os conceitos utilizados.

Este componente gera um elemento SVG no *browser* utilizando uma biblioteca chamada *Victory Chart*, uma das mais utilizadas dentre as gratuitas. A forma de se definir os componentes do gráfico e configurar os parâmetros se assemelhou ao padrão utilizado no Dayback Energy, suas funcionalidades também eram suficientes para suprir a demanda, portanto, foi escolhida.

Na Figura 26, mostra-se o padrão de organização do componente.

- Tem-se o componente *Chart*, que encapsula todos os outros elementos, recebendo propriedades e componentes aninhados, assim como definindo a interface do componente.
- O *ChartContext* é onde se define um *React Context*, que foi muito importante para viabilizar comunicação entre componentes, independente de quando ou por quem eles forem instanciados.
- O *ChartContainer* e o *ChartMqttContainer* são os provedores de dados, em que o *ChartContainer* faz requisições http e formata os dados para o gráfico e o *ChartMqttContainer* se conecta diretamente ao *broker* MQTT somente como cliente. As opções podem ser adicionadas à *Chart*, exclusivamente.
- O *ChartWrapper* é onde o layout do gráfico é definido, com seus eixos, cores, eventos.
- O *Index* importa e exporta de volta todos os arquivos de exportação da *Chart*. Ele é útil, pois reduz a quantidade de linhas de código utilizadas para importar os componentes da *Chart* e serve como sumário para se saber tudo o que existe.
- O *Graphs* contém os elementos que representam os dados no gráfico. Separando-se desta forma, é possível trocar a forma de apresentação sem recorrer a uso de condicionais no *ChartWrapper*.

Figura 26 – Arquivos da *Chart*

Fonte: Arquivo Pessoal.

A Figura 27 mostra as propriedades da *Chart*. Percebe-se que quase todas as propriedades possuem um valor padrão. Se o número de propriedades aumentasse muito, seria ruim se o utilizador tivesse que saber o que cada uma delas deve ser, então adiciona-se um padrão para o caso de uso mais comum.

Figura 27 – Propriedades da Chart

```
export function Chart ({
  nodes=[],
  children,
  container=<ChartContainer />,
  defaultAggregation=1,
  defaultMode = 'Lines',
  defaultPeriod = {from: startOfDay.getTime(), to:endOfDay.getTime(), tag:'hoje'},
  width,
  height,
  theme={lightTheme},
  Graphs,
  containerComponent,
}){
```

Fonte: Arquivo Pessoal.

A Figura 28 mostra o uso do *ChartContext*. Todos os valores dentro da propriedade *values* no *ChartProvider* serão disponibilizados para os componentes aninhados. Isso é especialmente útil quando um componente precisa de propriedades de dois lugares diferentes, pois só há uma oportunidade de se passar propriedades para um componente, que é quando ele é definido. Outra opção seria utilizar componentes de ordem maior, mas isto aumentaria a complexidade do código desnecessariamente.

Figura 28 – Definição de retorno da *Chart*

```

return (
  <ChartProvider value={{
    period,
    setPeriod,
    handleCalendarDateSelect,
    setAggregation,
    aggregation,
    mode,
    handleModeButtonClick,
    chartData,
    freezeData,
    setFreezeData,
    setChartData,
    filteredChartData,
    setFilteredChartData,
    nodes,
    setLoading,
    day, setDay, theme
  }} >
    <>
      <div>
        {children}
      </div>
      <ChartWrapper chartData={filteredChartData} mode={mode}
        aggregation={aggregation} loading={loading} period={period} Graphs={Graphs}
        width={width} height={height} day={day} theme={theme} containerComponent={containerComponent}
      />
      {container}
    </>
  </ChartProvider>
)

```

Fonte: Arquivo Pessoal.

Na Figura 29 aparece um exemplo de utilização da *Chart*. Os nodos, tema e tipos de gráfico são passados como propriedades, enquanto três elementos de cabeçalho são adicionados: o seletor de modo, que permite comutar entre gráfico de barras ou de linha, o tema, que no caso é o escuro.

Fica bastante fácil de se adicionar ou retirar elementos, não precisando depender de renderização condicional, o que mantém o código muito mais limpo.

Figura 29 – Exemplo 1 utilização componente *Chart*

```

<div className="flex-grow flex-col md:ml-36 md:-mt-12 md:pb-4 md:mb-5 ">
  <div className='w-64 md:w-96 lg:w-144 xl:w-192 2xl:224 3xl:240 rounded shadow-xl rounded bg-slate-800 p-2 md:p-4'>
    <Chart nodes={activeNodes} theme={darkTheme} Graphs=[[Lines, Bars]]>
      <ChartHeader >
        <ModeSelector />
        <ExportCSVButton />
        <CalendarButton />
      </ChartHeader>
    </Chart>
  </div>
</div>

```

Fonte: Arquivo Pessoal.

Na Figura 30 está a declaração de um componente sem nenhum item interativo. Este tipo de uso é bastante comum no gerador de relatórios.

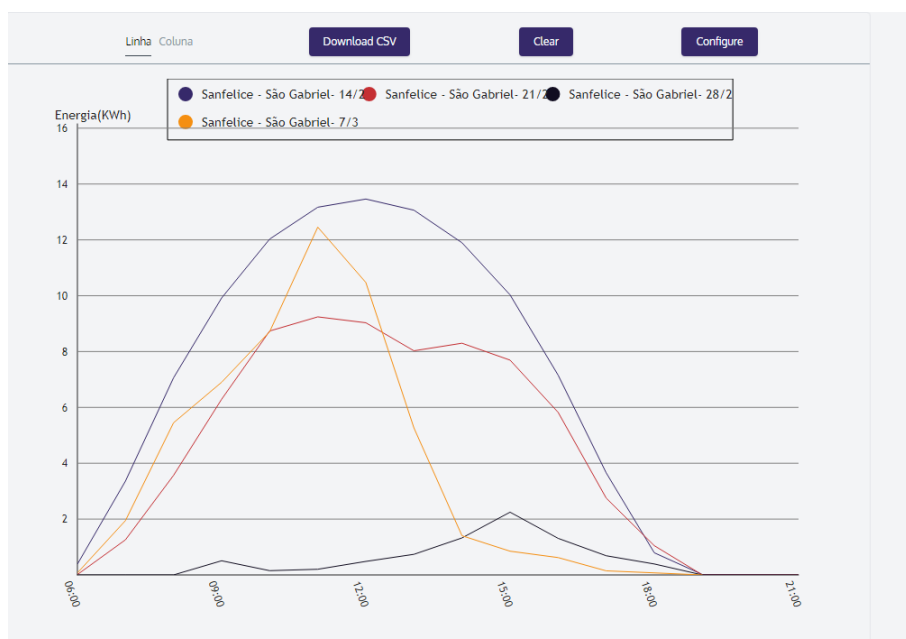
Figura 30 – Exemplo 2 de utilização componente *Chart*

```
<div className="flex bg-white bg-opacity-50 flex-col shadow border border-grey rounded mx-3">
  <Chart nodes={newActiveNodes}
    defaultAggregation={3}
    defaultMode='bar'
    defaultPeriod = {{from: startOfMonth.getTime(), to:now.getTime(), tag:'m'}}
    width={700}
    height={300}
  >
</Chart>
</div>
```

Fonte: Arquivo Pessoal.

A ferramenta da Figura 31 mostra o criador de gráficos, que se diferencia dos outros por ser capaz de dividir dados provenientes do mesmo nodo em segmentos diferentes. No caso na figura, todas as segundas feiras do mês estão apresentadas com dados de hora em hora. Este era um tipo de visualização de dados que sempre foi trabalhoso de se obter para a geração de relatórios. Agora pode ser criado em poucos segundos.

Figura 31 – Criador de gráficos

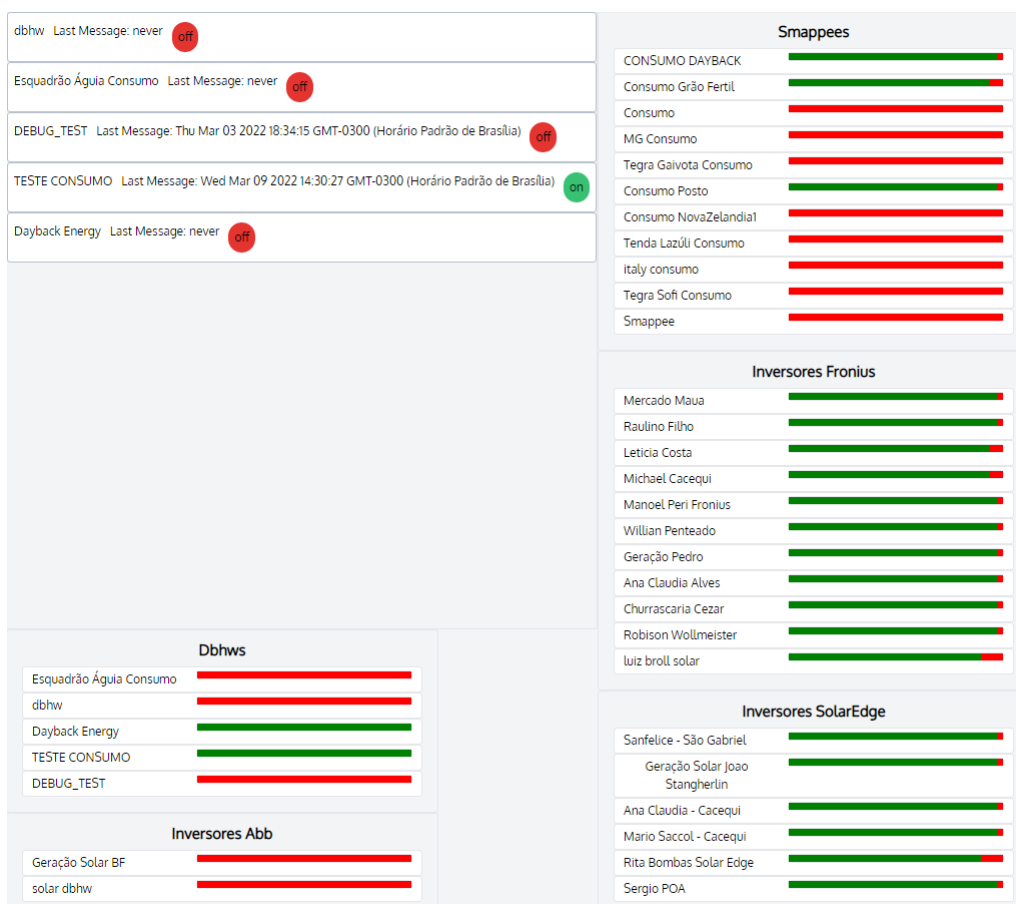


Fonte: Arquivo pessoal.

6.3 TELA DE STATUS

Anteriormente, a forma de se visualizar se as plantas de geração solar e medidores estavam funcionando era olhando a plataforma que disponibiliza os dados do próprio inversor. Com o aumento do número de fontes diferentes de dados, fez-se necessário a criação de uma tela que mostrasse quais medidores estão *online* e quais não estão. A tela em si está mostrada na Figura 32.

Figura 32 – Tela de status



Fonte: Arquivo Pessoal.

Cada um dos itens que possuem uma barra ao lado são nodos. A barra indica há quanto tempo a última atualização foi executada, sendo que a barra está completamente verde se a última medição tem horário igual ao presente e completamente vermelha caso não haja sinal por mais de um dia. Vale notar que as barras também podem estar vermelhas caso não haja dispositivo associado ao nodo.

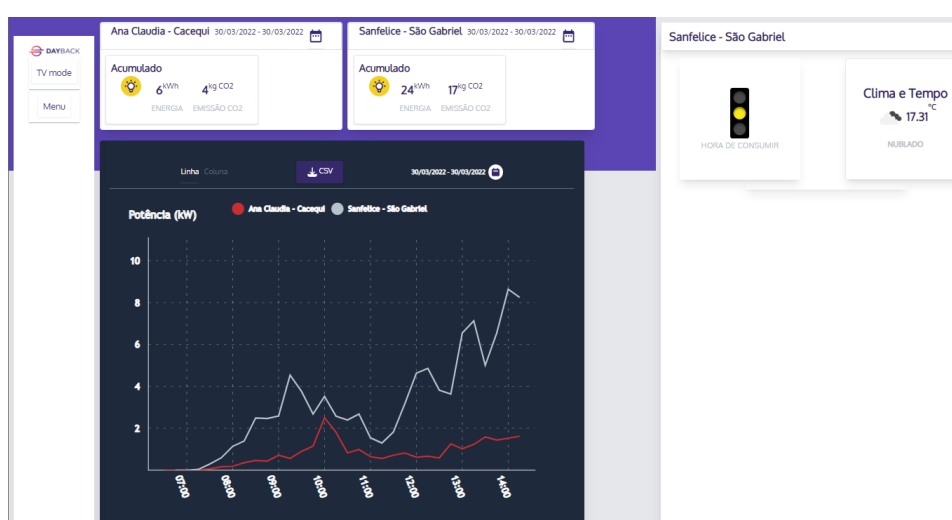
A forma de atualização dos dados ainda é manual, exigindo que se recarregue a página. Soluções como SWR (*Stale While Revalidating*) estão sendo estudadas para o melhoramento deste quesito.

6.4 MUDANÇA NA UI

Foram coletadas opiniões de outros funcionários e da chefia de empresa. A tela da Figura 33 representa o resultado final. Cada cliente possui uma configuração feita pela empresa de quais elementos devem aparecer. O menu à esquerda permite que tais elementos sejam adicionados à tela.

No momento, esta funcionalidade só está habilitando ou desabilitando a visualização de pontos de medição. O gráfico possui uma sobreposição dos dados de todos os nodos selecionados. A maior diferença de um cliente para o outro no momento é se ele possui medição de geração, consumo ou ambas.

Figura 33 – Nova *Dashboard*

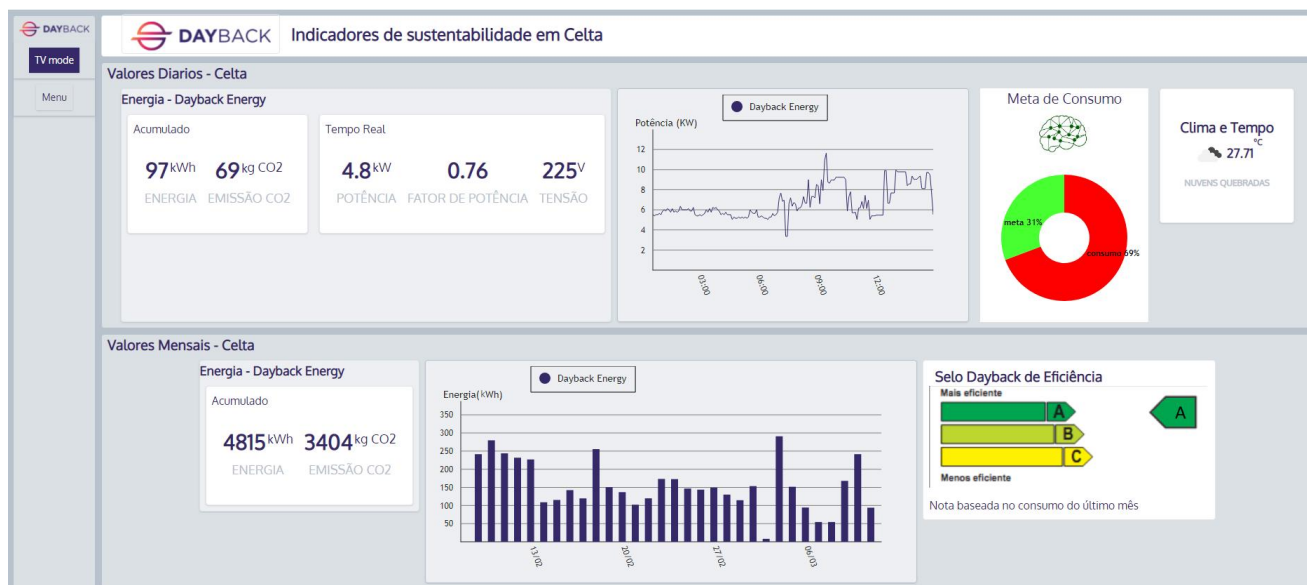


Fonte: Arquivo Pessoal.

A Figura 34 mostra o *dashboard* de televisão. A maior diferença desta tela para a outra é que não existem componentes interativos nesta tela. A parte superior mostra dados relativos ao dia dos nodo marcado como principal: o total de kWh e de equivalente de kg de CO2 emitido no dia, o gráfico do mesmo nodo, a meta de consumo, que é calculada baseada nos valores do último mês, estipulando que o consumo diário deve ser menor que 95% da média do mês, por último está o estado do clima e tempo.

Na parte inferior são mostrados o total de consumo no mês e o gráfico diário. O selo de eficiência ainda é uma funcionalidade experimental e está sendo desenvolvido por outras pessoas da equipe.

Figura 34 – Nova Dashboard de televisão



Fonte: Arquivo Pessoal.

6.5 FERRAMENTA DE INSPEÇÃO DE DADOS EM TEMPO REAL

Esta é uma tela disponível a usuários administradores que serve para realizar diagnósticos quando os dados provenientes do banco de dados não possuem resolução suficiente. Um exemplo da tela é mostrado na Figura 35. Ao se selecionar um nodo para monitorar, o gráfico começa a ser preenchido com cada atualização MQTT.

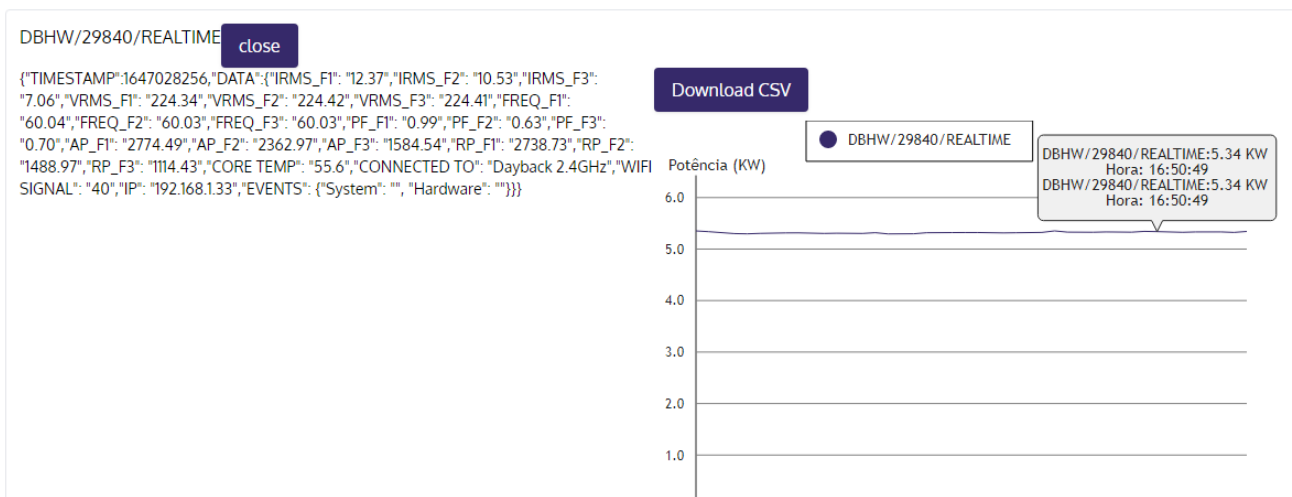
A conexão com MQTT é realizada ao se utilizar o *ChartMQTTContainer* como propriedade da *Chart*. Nessa tela, ainda falta adicionar opções para se imprimir tensão, corrente, fator de potência e frequência no gráfico.

Figura 35 – Inspetor de dados tempo real

Escolha um nodo para monitorar

Dayback Energy

Apply



Fonte: Arquivo Pessoal.

7 RESULTADOS

Neste capítulo, os resultados do projeto são analisados e discutidos. Primeiro, a verificação do cumprimento dos requisitos levantados no Capítulo 4 é feita. Em sequência, serão descritos os testes realizados. No final, serão descritos os impactos do projeto na empresa.

7.1 ANÁLISE

Uma forma de se avaliar os resultados é comparar se os requisitos foram cumpridos.

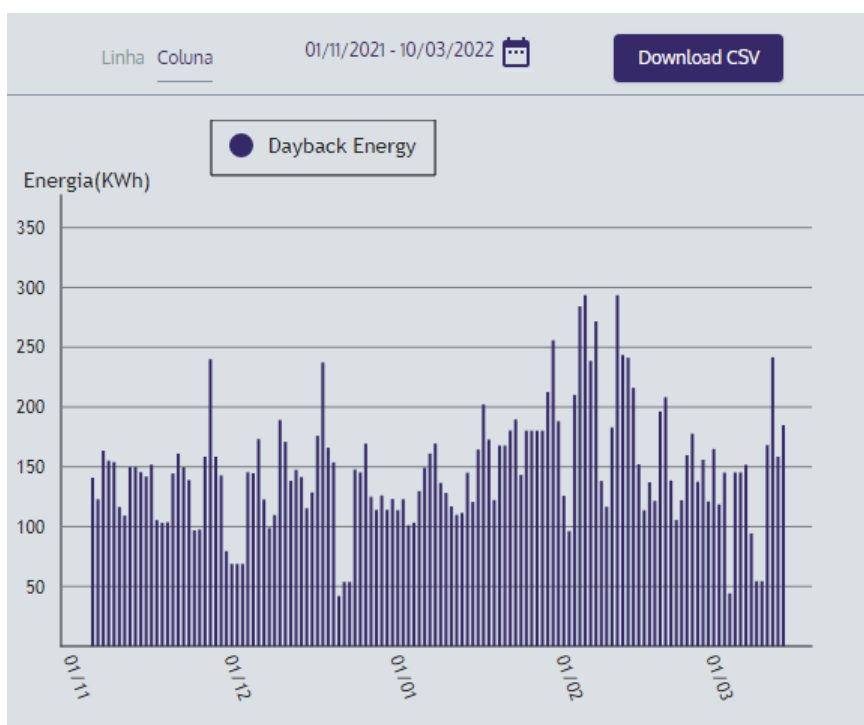
- Foi criada uma rotina para armazenar os dados do MQTT no banco de dados. A primeira versão está em funcionamento gravando dados do prédio do Celta, em Florianópolis. Não foram encontradas falhas no histórico de dados ao se realizar uma inspeção visual nos gráficos, o que sugere que o requisito de confiabilidade no armazenamento foi cumprido.
- Foi desenvolvido um algoritmo de recuperação de dados históricos para casos de perda de conexão. Este foi finalizado recentemente, portanto, não houve tempo para confirmar o funcionamento na prática, porém, os testes locais foram funcionais e satisfatórios, cumprindo o requisito não funcional de robustez.
- A ferramenta de visualização de dados em tempo real foi criada e está em funcionamento em produção. Ela possui funcionalidade de visualizar a mensagem do MQTT e também de fazer *download* de dados. Ainda faltam algumas opções de configuração.
- O gerador semi automático de relatório foi implementado. As funcionalidades que poderiam ser utilizadas parcialmente foram criadas e utilizadas pela equipe que confecciona os relatórios.
- O gerador de relatório possui opções de *download* e pré visualização.
- A funcionalidade de adicionar, remover e reordenar páginas está implementada.
- A interface foi melhorada esteticamente, incluindo responsividade e aproveitamento do espaço.
- O modo de televisão foi implementado e está sendo utilizado por clientes.
- Nas diversas partes do software foram criadas interfaces que isolam as camadas da aplicação. Também foram implementadas técnicas que permitem que componentes sejam reutilizados mais facilmente. Componentes foram reutilizados e customizados múltiplas vezes, indicando que o software é bastante flexível.

- O banco de dados está na terceira forma normal, o que previne problemas de duplicidade de dados e perda de performance.
- A tela de status, a modularidade do código e o gerador de relatórios aumentaram a capacidade de suportar clientes da empresa, confirmando o requisito de escalabilidade.
- A ausência de testes automatizados de funcionalidades é um fator negativo, que ameaça os requisitos de escalabilidade e robustez. Este fator é atenuado pelo fato de que o software estava sendo desenvolvido por somente uma pessoa, porém, um novo colaborador se juntou à equipe e outros devem se juntar no futuro.

7.2 TESTES

O DBHW foi instalado em novembro de 2021, a Figura 36 mostra o histórico dia a dia, retirado do banco de dados. Pode-se perceber que não houveram falhas significativas de hardware ou de software em um período de três meses.

Figura 36 – Teste de Confiabilidade



Fonte: Arquivo Pessoal.

O gerador de relatórios foi testado página a página, depois em configurações variadas. O resultado final correspondeu ao esperado. Todas as páginas funciona-

ram independentemente, portanto, testes individuais foram suficientes para garantir o funcionamento.

A tela de status depende do funcionamento das APIs. Testes individuais foram feitos com cada uma delas periodicamente durante o desenvolvimento. Houve casos em que as chamadas de API não funcionaram. A razão por trás disso em quase todos os casos foi indisponibilidade de dados por causa de perda de conexão wi-fi.

O download de CSV de gráficos de diversos formatos diferentes foi realizado e os ajustes necessários foram feitos. Modificações no formato dos dados ainda podem comprometer seu funcionamento, portanto, é possível que a funcionalidade deva ser retrabalhada.

Testes de responsividade foram realizados com o gráfico. Estes testes são visuais, mas muitos dos casos de uso foram contemplados, modificando tamanho de tela e quantidade de nodos ativos por *dashboard*.

7.3 IMPACTOS

Cada parte do projeto teve um impacto diferente:

- O gerador de relatórios facilitou a geração de gráficos que anteriormente eram criados com tabelas de dados no excel, agilizando muito a produção de relatórios.
- A tela de status tornou muito mais fácil a identificação de equipamentos que estejam *offline*, diminuindo o tempo entre a ocorrência do problema e a iniciativa para resolver, o que faz com que a empresa seja capaz de identificar mais problemas antes que os clientes os percebam.
- A ferramenta de inspeção de dados de tempo real não foi utilizada consistentemente desde a sua criação, parte por falta de demanda, parte por não possuir todas as funcionalidades ainda.
- A existência da tela de status e do gerador de relatório abriu uma nova oportunidade de negócio para a empresa, que é de monitorar clientes com muitas usinas elétricas, fornecendo uma solução integrada.

8 CONCLUSÃO

Dados os resultados apresentados, considera-se que o projeto atingiu seus objetivos. Mesmo nos casos em que a implementação não foi totalmente concluída, a implementação parcial teve resultados positivos na empresa tanto no quesito de operações internas quanto nas possibilidades de produtos que podem ser oferecidos.

Quanto à relação do projeto com o curso de Engenharia de Controle e Automação, os conceitos aprendidos em "Fundamentos de Sistemas de Bancos de Dados" foram bastante úteis, a disciplina de "Introdução à Inteligência Artificial para Automação", apesar de não ter sido utilizada na implementação do projeto, foi útil para analisar possibilidades de análise de dados para produtos futuros. "Fundamentos da Estrutura de Dados" foi útil como treino para produção dos algoritmos, as disciplinas relacionadas com elétrica e eletrônica ajudaram a ter uma noção dos equipamentos de hardware da empresa.

A parte mais difícil do projeto foi a definição de tarefas, pois o ambiente de inovação é muito caótico, múltiplos projetos são iniciados paralelamente e atrasos de implementação fazem com que oportunidades passem e esforço seja perdido, atrasando os outros projetos ainda mais.

8.1 TRABALHOS FUTUROS

Dadas as perspectivas de negócio da empresa e as opiniões dos usuários e testadores do projeto, foram levantados os seguintes trabalhos para complementar o trabalho:

- Automatizar gerador de relatório para extrair todas as informações necessárias para a geração do PDF diretamente do banco de dados, o que exigirá ampliamiento das informações de cadastro de usuários. Com esses dados, criar e disponibilizar opção de criar padrões de relatório com páginas pré-definidas, que consiga criar um novo PDF com poucos cliques.
- Definir o tipo de usuário "Integrador", que representa um cliente que possui muitas usinas de energia. Para ele, serão oferecidas ferramentas adicionais, como a tela de status para as usinas dele, notificação de status de usina e informações comparativas entre as usinas no tempo.
- Salvar dados de APIs da Fronius, Solar Edge e possíveis futuras outras no banco de dados, consolidando dados antigos e gravando-os em tableas separadas para reduzir o tamanho do banco de dados e melhorar a velocidade de carregamento de dados para períodos longos.

- Transferir a infraestrutura para a Oracle, pois a empresa conseguiu créditos na cloud deles, o que reduziria o custo dos serviços de nuvem.

REFERÊNCIAS

CICLOVIVO, Redação. 2019. Disponível em:

<https://ciclovivo.com.br/planeta/energia/brasil-desperdica-r71-milhoes-em-energia-por-dia/#:~:text=Mais%20de%2071%20milh%C3%B5es%20de,anos%20de%202015%20a%202017..>

ELLIOT, Eric. **Composing Software: An Exploration of Functional Programming and Object Composition in JavaScript**. [S.l.: s.n.], 2018.

FLANAGAN, David. **JavaScript: The Definitive Guide**. [S.l.: s.n.], 2002.

GOEL, Aman. 2020. Disponível em:

<https://hackr.io/blog/web-development-frameworks>.

GUSTAVO, Luis. 2017. Disponível em:

<https://fluxoconsultoria.poli.ufrj.br/blog/eficiencia-energetica-para-industrias/>.

HOPKINS, Callum. 2013. Disponível em:

<https://www.sitepoint.com/the-mvc-pattern-and-php-1/>.

LARAVEL. 2021. Disponível em:

<https://laravel.com/docs/8.x/database#configuration>.

LIU, Shanhong. 2020. Disponível em:

<https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/>.

MARTIN, Robert C. **Agile Software Development, Principles, Patterns, and Practices**. [S.l.: s.n.], 2002.

MICROSOFT. 2022. Disponível em: <https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description>.

MOZILLA. 2021. Disponível em: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/How_CSS_works.

OASIS. 2021. Disponível em:

<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.

RAVICHANDRAN, Adhithi. 2020. Disponível em:

<https://adhithiravi.medium.com/react-virtual-dom-explained-in-simple-english-fc2d0b277bc5#:~:>

[text=React%20uses%20virtual%20DOM%20to,rendering%20of%20the%20UI..](https://adhithiravi.medium.com/react-virtual-dom-explained-in-simple-english-fc2d0b277bc5#:~:)

REACT. 2021. Disponível em:

<https://pt-br.reactjs.org/docs/thinking-in-react.html>.

TAILWIND. 2021. Disponível em: <https://tailwindcss.com/docs/utility-first>.

W3SCHOOLS. 2021a. Disponível em:

https://www.w3schools.com/html/html_intro.asp.

W3SCHOOLS. 2021b. Disponível em:

https://www.w3schools.com/css/css_intro.asp.

W3SCHOOLS. 2021c. Disponível em:

https://www.w3schools.com/js/js_intro.asp.

ANEXO A – MODELO DE RELATÓRIO

Figura 37 – Capa



Fonte: Arquivo pessoal.

Figura 38 – introdução

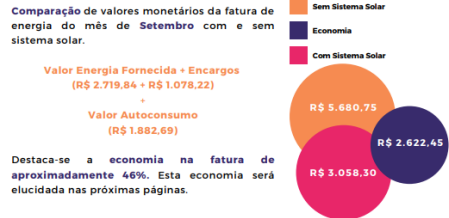
1) Introdução

O presente relatório tem como objetivo apresentar os resultados de **geração e consumo de energia** durante os dias 01/09/21 e 30/09/21, referente ao cliente:

Cliente: Grão Fértil Comercio de Insumos Agropecuario LTDA
Projeto: Sistema fotovoltaico
Município: São Vicente - RS
Endereço: Rua Osvaldo Aranha, 1790
Potência: 55,2 kWp
Tipo de Uso: Industrial

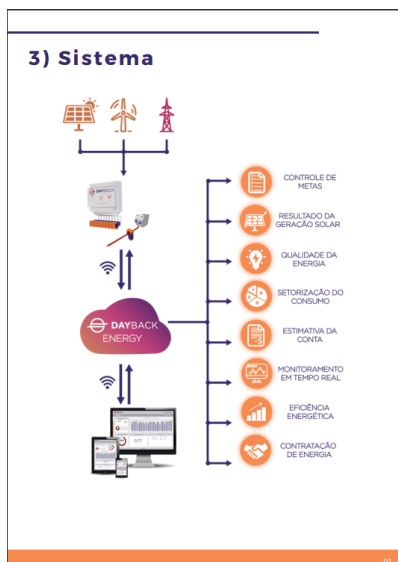
Importante informar que tais dados foram obtidos através do sistema de monitoramento Dayback Energy.

2) Economia



Fonte: Arquivo pessoal.

Figura 39 – Sistema



Fonte: Arquivo pessoal.

Figura 40 – Geração x Radiação

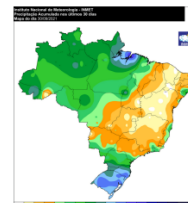
4) Geração x Radiação

O gráfico abaixo apresenta os **valores diários** de geração monitorados entre os dias 01/09/21 e 30/09/21.



A linha em laranja apresenta o nível de **radiação solar** durante o período monitorado, é possível observar que a linha acompanha os dados diários de **geração solar**. Ou seja, conforme a previsão do tempo (chuva, céu encoberto) afeta a **radiação solar**, consequentemente, impacta a **geração de energia solar**.

A imagem abaixo apresenta a **precipitação de chuva** nos dias correspondentes ao período da fatura, neste mês a precipitação foi menor e a geração solar maior.



Fonte: Arquivo pessoal.

Figura 41 – Consumo Total

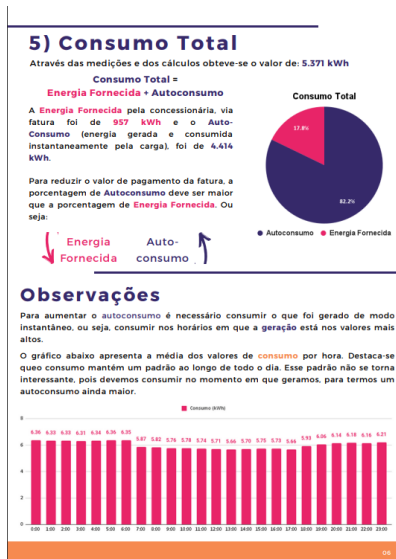


Figura 42 – Geração total

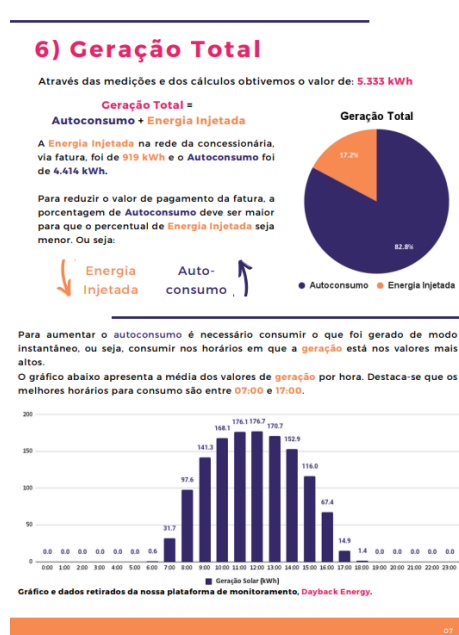


Figura 43 – Fator de potência

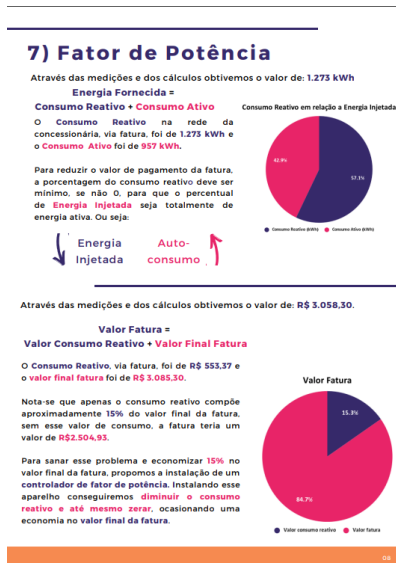


Figura 44 – Conclusões



Figura 45 – Contato



Fonte: Arquivo pessoal.