UNIVERSIDADE
LUSÓFONA

Aplicação para monitorização de uma central de energia renovável

Trabalho Final de curso

Relatório Final

Autores: Ana Beatriz Pires e Bruno Miguel Soares

Orientadores: Lúcio Studer e Gabriela Soares

Coorientador: João Ramos

Trabalho Final de Curso | LEI | 28/06/2024

www.ulusofona.pt

Direitos de cópia

Resumo

Neste Trabalho Final de Curso pretende-se desenvolver uma aplicação Web para monitorização de uma central de produção de energia solar em que seja possível ser feita a recolha de informação sobre os equipamentos e criação de *dashboards* com indicadores de consumo e desempenho.

Também serão aplicados modelos de ciências de dados para realização de previsões do desempenho esperado dos painéis relativamente a diversos parâmetros como meteorologia e níveis de sujidade nos painéis, comparando essas previsões aos valores efetivamente obtidos.

Para melhor monitoramento dos painéis solares será também desenvolvido um sistema de alertas para que os responsáveis consigam perceber, por exemplo, quando é necessário limpar os painéis, ou quando um inversor está avariado.

Abstract

In this Final Project, the aim is to develop a Web application to monitor a solar energy production facility capable of collecting information about the equipment and creating dashboards displaying consumption and performance indicators.

Additionally, Data Science models will be applied to forecast the expected performance of the solar panels based on numerous parameters such as weather conditions and panel cleanliness. The forecast will then be compared to the actual values of production obtained.

To enhance the monitoring of the solar panels an alert system will also be implemented so that the responsible parties can receive notifications, for example, when cleaning the panels is necessary, or when there's a malfunction on one of the inverters.

Índice

Lista de Figuras

Lista de Tabelas

1 Identificação do Problema

1.1 Enquadramento

Os painéis solares são dispositivos projetados para absorver energia do sol e convertê-la em eletricidade utilizável e o uso desta é cada vez mais uma opção para a produção de energia limpa e sustentável. A demanda para a utilização de painéis solares é especialmente avultada em países cujo clima proporciona uma produção de energia maior pelo alto nível de incidência solar e a não existência de dias chuvosos.

1.1.1 Funcionamento dos painéis solares

Os painéis solares absorvem a luz do sol durante o dia e geram energia. Para melhor eficiência na conversão da radiação solar para energia elétrica é imprescindível que as placas tenham incidência direta/perpendicular da luz, assim, quanto mais radiação direta o painel recebe, mais energia elétrica será gerada. O resultado disso é a produção de corrente elétrica continua que será a seguir enviada para o inversor. O inversor é o coração do sistema. É o responsável por transformar corrente continua, para corrente alternada, que é a energia que utilizamos em nossas casas e empresas.

O medidor de energia bidirecional é um componente essencial para os sistemas de energia solar fotovoltaica conectados à rede. Não só mede a energia consumida, mas também mede a quantidade de energia injetada na rede elétrica. Ou seja, durante o dia o medidor contará quanto de energia foi gerada pelos módulos fotovoltaicos e enviado para a rede elétrica, já durante a noite ou em dias nublados o medidor mede quanto de energia foi consumida na residência. Podemos observar o funcionamento dos painéis solares na Figura 1. [Efei23]

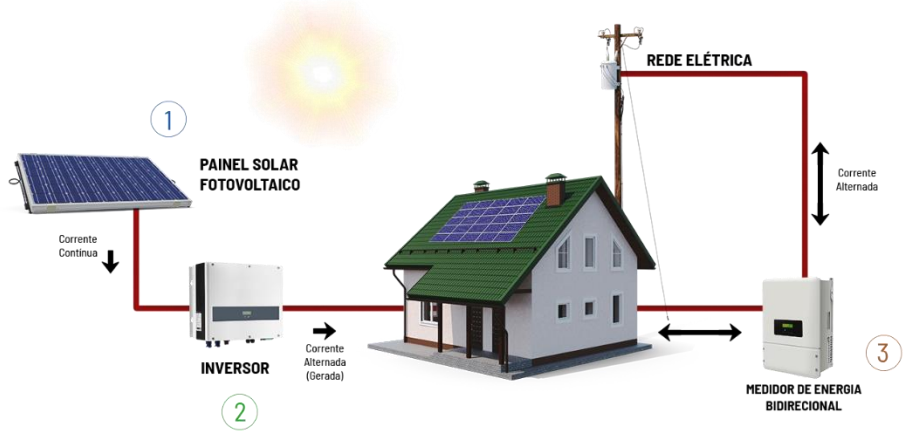

Figura 1 - Funcionamentos dos painéis solares (retirado de [Efei23]).

1.1.2 Monitorização e análise

Atualmente existem softwares para visualização de dados diários do nível de energia produzida, total de receitas conseguidas através dessa produção, e outros parâmetros específicos para que seja possível fazer a monitorização do bom funcionamento destes dispositivos pelos seus responsáveis. A análise dos dados fornecidos pode ser útil para a previsão de produção futura bem como a análise de perda de eficiência devido a problemas nos próprios painéis.

1.2 Objetivos

Pretendemos desenvolver uma aplicação web para a monitorização de uma central de energia solar onde iremos recolher as informações dos painéis solares, a criação de um *dashboard* onde apresentamos os dados e as estatísticas que os painéis solares nos dão, como o consumo e o desempenho de uma forma simples para o cliente e um sistema que retorna alertas quando há avarias de peça, sujidade dos painéis, ou problemas na rede.

1.3 Organização do documento

Este documento estrutura-se nos seguintes capítulos:

- No **Capítulo 1** é feita a identificação do problema e objetivos do presente trabalho.
- No **Capítulo 2** é avaliada a viabilidade e pertinência da solução proposta.
- No **Capítulo 3**, é feito o benchmarking da solução proposta com aplicações que são usadas atualmente e que funcionam de forma semelhante ao proposto.
- No **Capítulo 4**, é feita a análise de séries temporais e modelos preditivos.
- No **Capítulo 5**, é feito o levantamento e analise de requisitos.
- No **Capítulo 6**, é detalhada a solução proposta para o problema identificado.
- No **Capítulo 7**, são definidos os testes funcionais e os testes não funcionais
- No **Capítulo 8**, é apresentado o método, o planeamento do TFC e consequentes resultados.
- No **Capítulo 9**, são apresentadas as conclusões e sugestões para possíveis trabalhos futuros.

2 Viabilidade e Pertinência

2.1 Pertinência

O uso dos painéis solares é bastante vantajoso para quem faz este investimento. Primeiramente, a instalação desta tecnologia pode reduzir significativamente as contas de energia, onde gera lucro a longo prazo para empresas e residências. Além de tudo, a energia solar é inesgotável, sustentável e contribui para a redução das emissões de carbono, tornando-a amiga do ambiente.

A sustentabilidade é crucial para enfrentar os desafios globais, como as mudanças climáticas, a escassez de recursos e a degradação ambiental. Na educação, desempenha um papel vital na promoção da conscientização sobre consumo responsável e a mudança de atitudes para um estilo de vida mais sustentável.

Na Tabela 1 apresentamos os vários tipos de energia e as emissões de CO2 produzidas pelas mesmas. Estes valores são aproximados e podem variar dependendo de vários fatores, incluindo a eficiência da tecnologia utilizada.

Tabela 1 - Tipos de energia e respetivas emissões de CO2, retirado de [IMP24][IMP24]

Tipo de Energia	Emissões de CO2 (gramas por kWh)
Carvão (carvão mineral)	820
Gás natural (combustão)	490
Biomassa	230
Energia Solar (fotovoltaica)	41
Hidroelétrica	24
Energia Eólica	12
Nuclear	12

A produção direta de energia solar (por meio de módulos fotovoltaicos) não emite dióxido de carbono (CO2) durante o seu funcionamento. No entanto, consideramos as emissões associadas à fabricação, transporte e instalação dos painéis solares. O processo de produção desses painéis envolve o uso de energia e materiais, o que pode resultar em emissões de CO2. Apesar das emissões iniciais, a energia solar geralmente compensa essas emissões ao longo da sua vida útil, produzindo eletricidade sem emissões durante a operação.

De momento os dados fornecidos pelos softwares usados atualmente são extremamente técnicos e o utilizador comum não tem grande conhecimento sobre aquilo que lhes é exposto. Encontrou-se então a necessidade de se desenvolver uma aplicação para tornar mais simples a visualização dos diversos dados obtidos e fazer previsões mais precisas de produção futura para deixar os clientes mais informados.

De acordo com um artigo do Jornal de Negócios redigido por Sónia Santos Dias [SOFC23] o mercado para softwares de gestão de carbono vai atingir cinco mil milhões de euros em 2032 e

que este cresça a uma taxa anual de 19,7%. Isto acontece devido a regulamentos e uma maior sensibilidade para com a questão climática, a digitalização das produções energéticas por necessidade das partes interessadas e procura de transparência para obter vantagens competitivas.

2.2 Desenvolvimento Sustentável da ONU

Em 2015, a ONU [ONU24] criou uma definição da Agenda 2030, constituída por 17 Objetivos de Desenvolvimento Sustentável. A Agenda aborda várias dimensões do desenvolvimento sustentável e que promove a paz, a justiça e instituições eficazes. Os ODS têm como base os progressos e lições aprendidas com os 8 Objetivos de Desenvolvimento do Milénio, estabelecidos entre 2000 e 2015, e são fruto do trabalho conjunto de todos.

A agenda 2023 e os 17 ODS são a visão para a humanidade e "uma lista das coisas a fazer em nome dos povos e do planeta".

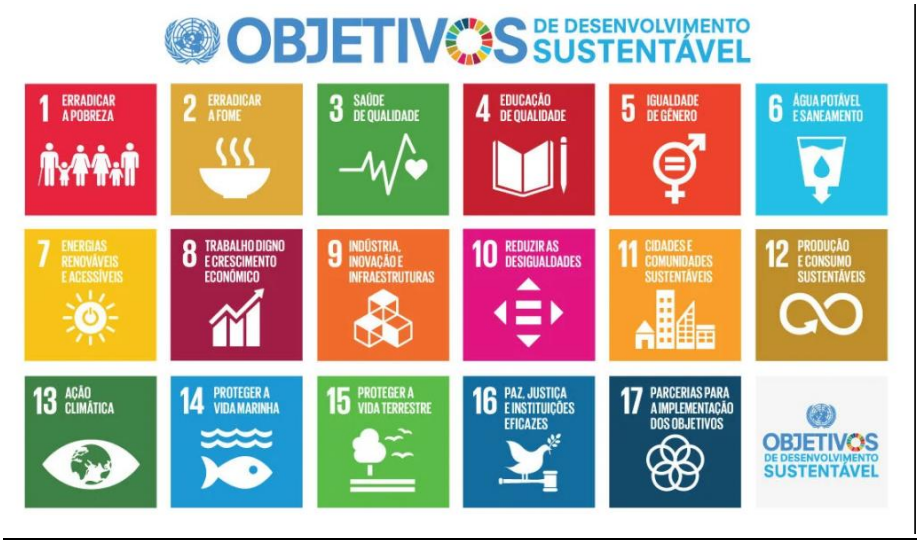

Figura 2 - Objetivos de Desenvolvimento Sustentável

2.2.1 ODS 7 – Energias Renováveis e Acessíveis

O ODS 7 [ODS24] visa garantir o acesso a uma energia acessível, segura, sustentável e moderna para todos. De acordo com o relatório "*Tracking SDG 7: The Energy Progress Report 2023*" [TRAC24], há 675 milhões de pessoas sem acesso a energia elétrica. Este estudo foi feito pelas cinco organizações responsáveis por esta ODS: a Agência Internacional de Energia, a Agência Internacional de Energia Renovável, a Divisão de Estatística das Nações Unidas, o Banco Mundial e a Organização Mundial da Saúde.

Para atingir o ODS 7 até 2023, é necessário investir em fontes de energia limpa, como a solar, eólica e hidroelétrica, e melhorar a produtividade energética. Também é necessário expandir a infraestrutura e melhorar a tecnologia para energia limpa em todos os países em desenvolvimento.

Para percebermos o impacto da falta de energia no mundo, são mostrados os seguintes dados:

Impacto global

675 milhões de pessoas em todo o mundo vivem sem eletricidade.

Outras energias

2,3 bilhões de pessoas cozinham com combustíveis e tecnologias contaminantes.

A Figura 3 mostra os países com mais pessoas afetadas.

Figura 3 - Países com mais pessoas afetadas pela falta de eletricidade

Principais causas

A perspetiva da economia regional incerta, os altos níveis de inflação, as flutuações cambiais, as dificuldades de endividamento e a falta de financiamento.

Existe solução?

No cenário de emissões líquidas zero de 2050, alcançar o acesso universal à eletricidade até 2030 exige um investimento anual de 30 mil milhões de dólares em geração de rede e soluções descentralizadas por meio de programas de fornecimento integrados, inteligentes e eficientes [IBE24].

3 Benchmarking

3.1 Benchmarking de outras aplicações

Neste momento, estão a ser usadas para monitoramento e gerenciamento dos painéis solares duas aplicações distintas: o iSolarCloud da Sungrow, e o SunnyPortal da SMA. Visto a necessidade de tornar as informações destas duas aplicações mais legíveis, achámos então necessário uma análise mais profunda das suas *features*.

Estas são capazes de gerir múltiplos equipamentos numa só aplicação onde são apresentados os dados de produção energética em tempo real e passados (como produção mensal, anual e total) que podem ser exibidos em diversos tipos de grafismo e relatórios conforme a necessidade do utilizador; possui também uma aba de avisos sobre falhas de sistema nos painéis e têm suporte para iOS e Android e páginas Web, como podemos ver na Figura 4 e na Figura 5.

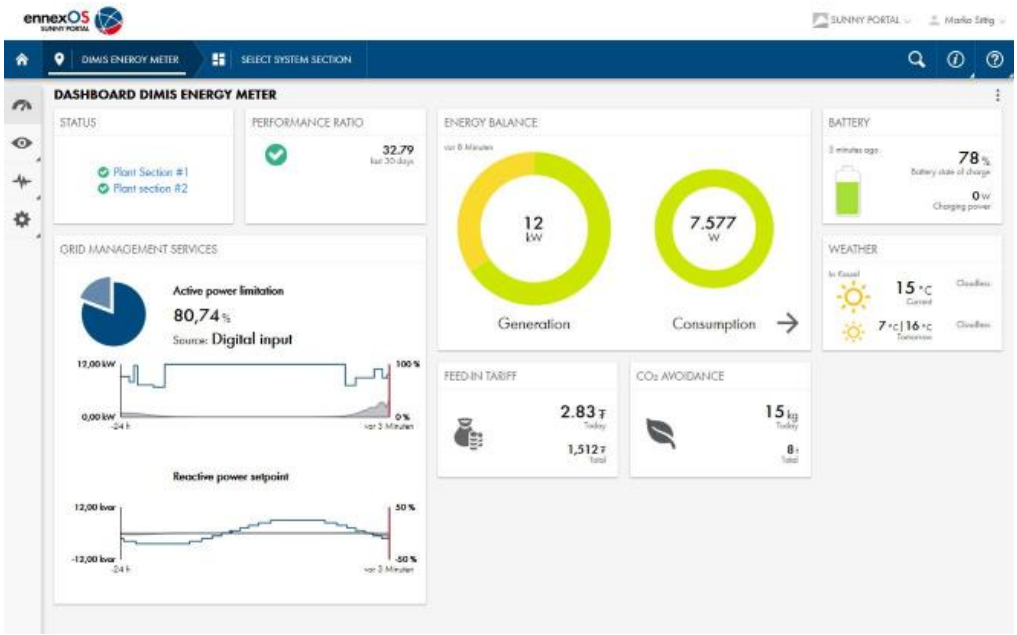

Figura 4 - Aplicação do Sunny Portal, retirado de [SMA23]

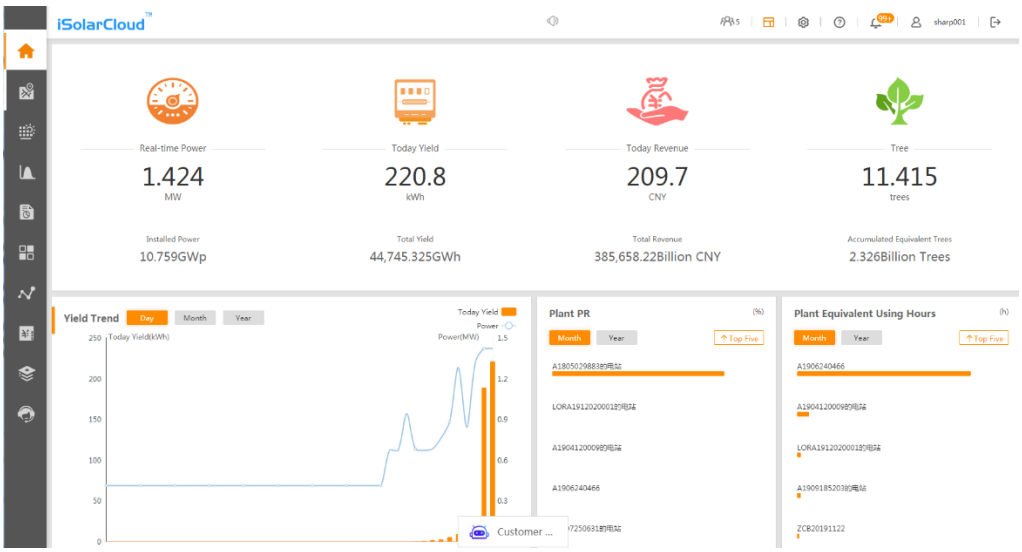

Figura 5 – Aplicação do iSolarCloud, retirado de [ISOL23]

Adicionalmente, neste momento também é usada uma tabela de Excel para se colocarem os valores esperados de produção de diversos anos comparativamente ao valor real de produção, fazer gráficos utilizando esses mesmos dados, e mostrar o dinheiro que era expectável poupar em comparação com o que foi poupado.

A tabela comparativa abaixo pretende demonstrar as diferenças entre as aplicações usadas anteriormente e a aplicação que pretendemos desenvolver neste trabalho bem como providenciar uma visão das melhorias e benefícios de o fazer.

Tabela 2 - Tabela comparativa entre as aplicações antigas e a aplicação nova

Funcionalidades	iSolarCloud	SunnyPortal	Aplicação nova
Apresentar dados históricos e atuais	✓	✓	✓ (de forma mais simplificada)
Sistema de Alertas	✓	✓	✓ (de forma mais simplificada)
Previsões com base em vários parâmetros	✗	✗	✓
Parâmetros ecológicos	✓	✓	✓
Previsão de receita	✗	✗	✓
Receita gerada	✓	✓	✓

4 Estado da arte

4.1 O que é uma série temporal?

A série temporal é um tipo de dados, ou uma estrutura de dados em que a componente do

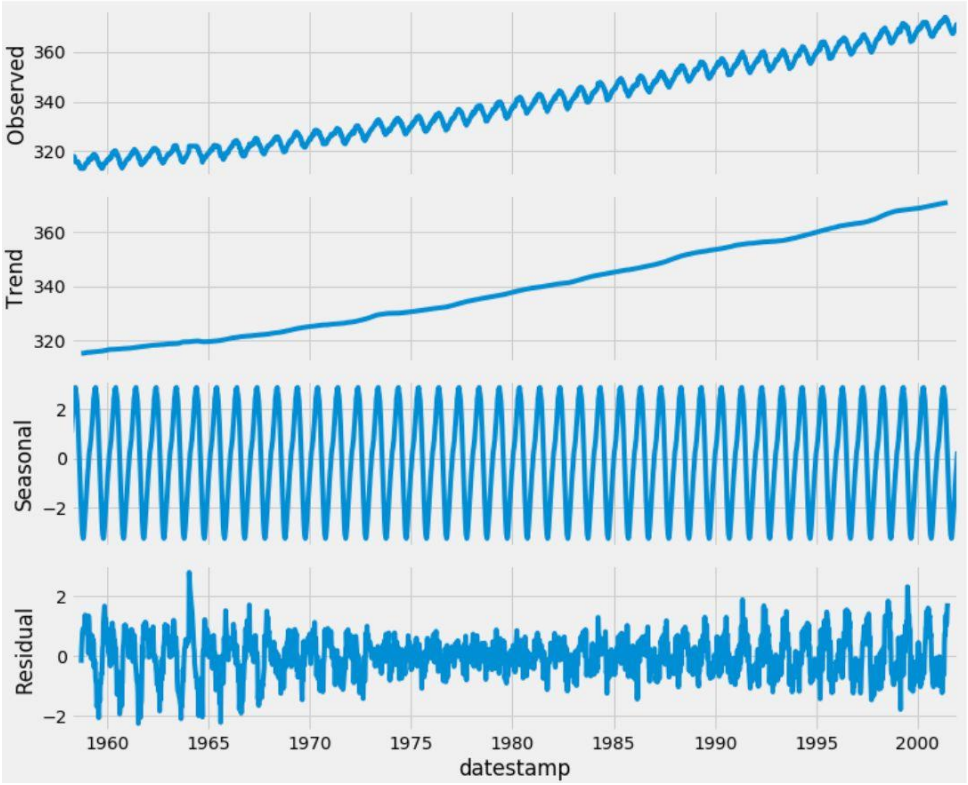

Figura 6 - Componentes das séries temporais

tempo é essencial. E por termos este componente, temos análises, modelos e métodos específicos para séries temporais. Contudo, este tipo de análise não é apenas o ato de coletar dados ao longo do tempo – o que diferencia os dados das séries temporais de outros dados é que esta pode mostrar como as variáveis mudam ao decorrer do tempo, isto é, o tempo é uma medida fundamental pois mostra como os dados se ajustam ao longo dos *data points*, bem como os resultados [Tabl23].

As séries temporais normalmente requerem grandes quantidades de dados para garantir consistência e confiabilidade. Usar um grande conjunto de dados garante que os padrões que são encontrados podem mostrar variância sazonal. Adicionalmente, uma grande quantidade de dados também pode ser usada para fazer previsões futuras com dados que aconteceram no passado.

Estas podem ser descritas em três componentes: a sazonalidade, que descreve o sinal periódico na série temporal; tendência, que descreve se uma série temporal está a aumentar ou a diminuir durante o tempo; ruído que descreve a variabilidade e variância da série temporal. [DCAM23]

Alguns objetivos das séries temporais são por exemplo, compreender a estrutura da série, isto é, conseguir encontrar uma forma de prever valores futuros, identificar tendências como o crescimento ou diminuição de determinado fenômeno ao longo do tempo, identificar padrões

de sazonalidade, ou seja, padrões que se repetem em intervalos fixos de tempo e identificar valores atípicos que não seguem um padrão esperado.

4.2 Previsão de séries temporais e medidas de desempenho

A previsão de séries temporais trata-se de prever o futuro com a maior precisão possível, tendo em contas todas as informações disponíveis, incluindo dados históricos e conhecimento de quaisquer eventos futuros que possam impactar as previsões. Para gerar o gráfico da Figura 7 foram coletados dados reais do próprio TFC para ser feita uma previsão. A azul temos os dados históricos, a verde temos a série observada e a vermelho é representada a previsão.

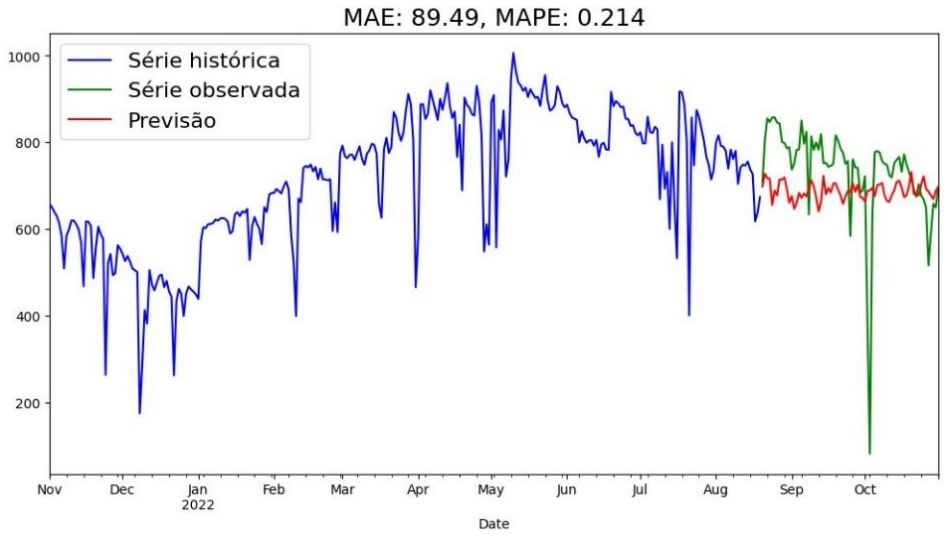

Figura 7 - Exemplo de uso de uma série temporal

Para avaliar o quão bem estas previsões se comportam temos algumas medidas de desempenho como por exemplo o MAE (*Mean Absolute Error*) que calcula a média do erro de previsão absoluto durante o período de previsão e o MAPE (*Mean Absolute Percentage Error*) que é uma métrica em escala que divide o erro absoluto pelo erro real. Os métodos de cálculo de ambas as medidas de desempenho estão representadas na Figura 8.

$$MAE = \frac{\sum_{t=1}^{n} |y_t - \hat{y}_t|}{n} \qquad MAPE = \frac{100}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{y_i}$$

Figura 8 - Fórmula do MAE e do MAPE

4.3 Alguns modelos preditivos

O modelo ARIMA (*Autoregressive Integrated Moving Average*) fornece uma das abordagens para a previsão de séries temporais [TEXb23]. Este modelo usa valores de auto regressão e erros de previsões anteriores em um modelo semelhante a uma regressão.

LightGBM (*Light Gradient Boosting Model*) é uma estrutura que usa algoritmos de aprendizagem baseados em árvore de decisão. Isto é, usa algoritmos baseados em histogramas, que agrupam valores de recursos contínuos em compartilhamentos discretos. A maioria dos algoritmos de

aprendizado de árvores de decisão crescem árvores por profundidade, como exemplificado na Figura 9 que podemos ver a seguir: [GIT23]

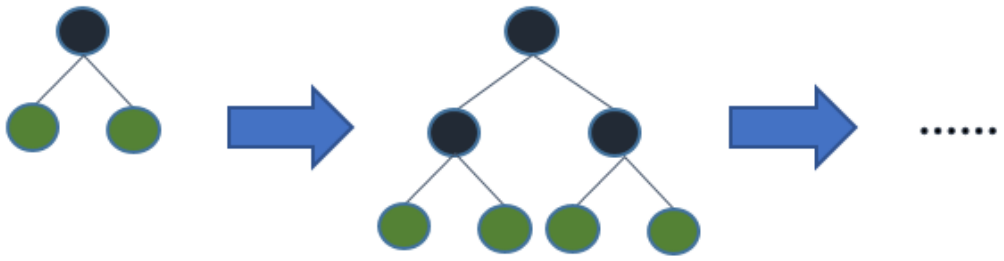

Figura 9 – Funcionamento de outros algoritmos de árvores de decisão

Como verificamos na Figura 10 o LightGBM cria árvores de decisão que crescem em termos de folha, o que significa que dada uma condição, apenas uma única folha é dividida, dependendo do ganho.

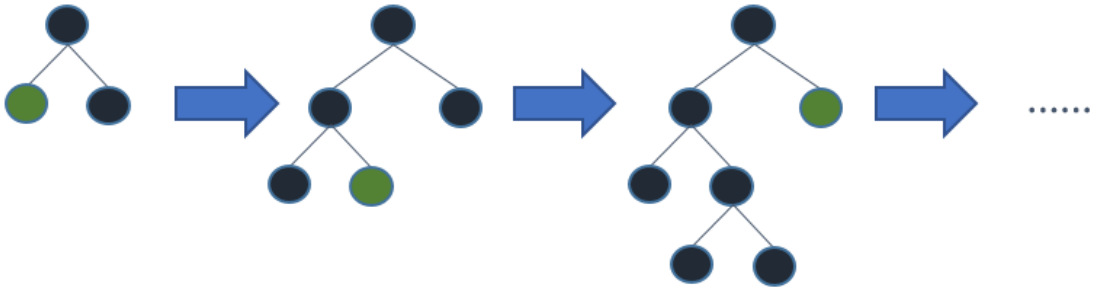

Figura 10 – Funcionamento do LightGBM

O Prophet é um modelo criado originalmente para prever dados diários com sazonalidade semanal e anual, além de efeitos de feriados. Posteriormente, foi estendido para cobrir mais tipos de dados sazonais. Funciona melhor com séries temporais com forte sazonalidade e várias temporadas de dados históricos [TEXa23]. Esta série temporal pode ser considerada um modelo de regressão não linear, e o seu cálculo é efetuado pela fórmula representada na Figura 11 onde g(t) descreve uma tendência linear por partes (ou "termo de crescimento"), é(t) descreve os vários padrões sazonais, h(t) captura os efeitos do feriado e ε(t) é um termo de erro de ruído branco.

$$sim_t = g(t) + \acute{e}(t) + h(t) + \varepsilon_t,$$

Figura 11 - Fórmula do Prophet

Na Tabela 3 representada abaixo podemos ver uma comparação simplificada entre os três modelos mencionados.

Tabela 3 - Comparação entre modelos preditivos (LightGM-ARIMA-Prophet)

Critério	LightGBM	ARIMA	Prophet
Complexidade	Alta	Baixa e Moderada	Moderada
Padrão capturado	Não linear e complexo	Linear e sazonal	Sazonal, feriados e tendências
Eficiência	Alta, otimizado para grandes conjuntos	Moderada	Moderada
Manuseio de Sazonalidade	Pode capturar sazonalidade complexa	Sazonalidade básica	Sazonalidade automática
Manuseio de Feriados	Não automático	Não automático	Automático
Robustez contra *Outliers*	Moderada	Alta	Moderada
Análise Multivariada	Suporta	Não suporta	Não suporta

Posto isto, escolhemos usar o LightGBM visto que é uma estrutura projetada para velocidade e eficiência e porque há a possibilidade de adicionar informações adicionais, semelhantes a uma análise multivariada (como quantidade de dias sem limpeza, informações climáticas, última manutenção dos painéis, etc.) o que é criticamente necessário para o que está previsto nos objetivos do projeto.

5 Levantamento e Análise de Requisitos

5.1 Análise de requisitos

Para este projeto foram levantados e discutidos alguns requisitos funcionais e não funcionais que especificam as funcionalidades do nosso projeto em conjunto com o *stakeholder,* alguém que trabalha nesta área, após reuniões. Estes estão descritos abaixo.

5.1.1 Requisitos Funcionais

Um requisito funcional visa especificar uma funcionalidade específica que o software deve ter, ou seja, algo que o sistema deve ser capaz de fazer.

Listam-se em baixo os requisitos funcionais levantados:

Estrutura	**Sistema de login para perfis de utilizador**
R1	O sistema deverá permitir um acesso diferenciado a funcionalidades com base nos perfis de usuário, nomeadamente, criança, utilizador e técnico

Página Inicial	**Monitoramento e exibição de dados de energia e emissões**
R2	O sistema deverá fornecer aos utilizadores, na página inicial, uma visão abrangente e em tempo real da produção de energia, a energia acumulada naquele dia ou mês, comparação entre o que estamos a produzir e o que estamos a consumir e o impacto ambiental.

Previsões	**Monitoramento de manutenção e economia financeira**
R3	O sistema deverá fornecer previsões críticas para os técnicos e utilizadores sobre a necessidade de manutenção ou limpeza, detalhes sobre a próxima manutenção e a estimativa de economia financeira com base na produção de energia.

Alertas	**Sistema de alertas**
R4	O sistema deverá fornecer alertas oportunos para situações críticas, como o desligamento do inversor, limpezas necessárias, baixa produção de energia e o desligamento da central de energia.

Vantagem Económica	**Visualização das perdas financeiras**
R5	O sistema deverá fornecer uma visualização clara e compreensível das perdas financeiras decorrentes de manutenção e sujidade nos painéis solares, adaptando a apresentação para técnicos, utilizadores e de maneira criativa para crianças e jovens.

Educação	**Secção educativa interativa para consumo e produção de energia**
R6	O sistema deverá ter uma secção dedicada a fornecer informações educativas sobre o consumo e a produção de forma divertida e dinâmica, sem se concentrar apenas em números e dados.

5.1.2 Requisitos não funcionais

Um requisito não funcional visa especificar uma funcionalidade que é característica do sistema, ou seja, são as propriedades específicas ao sistema. Normalmente são qualificados como desempenho, escalabilidade, segurança, entre outros.

Listam-se em baixo os requisitos não funcionais levantados:

Adaptabilidade	**Adaptabilidade a diversos tipos de display**
R7	O sistema deve ser projetado e implementado de forma a garantir uma experiência consistente e eficiente numa variedade de dispositivos, como telemóveis ou computadores.

Usabilidade	**Usabilidade da interface do utilizador**
R8	A interface do sistema deve ser projetada de forma simples e intuitiva, visando facilitar o uso para os utilizadores de diversos perfis e habilidades.

Disponibilidade	**Disponibilidade ininterrupta**
R9	O sistema deve garantir uma disponibilidade contínua, ou seja, 24 horas por dia e 7 dias por semana, para atender às necessidades dos utilizadores a qualquer hora.

Desempenho	**Desempenho e frequência de atualização dos dados**
R10	O sistema deve garantir um desempenho eficiente e uma atualização periódica dos dados, com frequência predefinida, para proporcionar informações em tempo quase real.

Compatibilidade	**Compatibilidade com sistemas fotovoltaicos e inversores**
R11	O sistema deve ser projetado e implementado de maneira a ser compatível com os sistemas fotovoltaicos e com os inversores.

5.2 Casos de Uso

Os diagramas de caso de uso descrevem visualmente as funcionalidades do sistema que vai ser projetado, e vai demonstrar as diferentes interações do utilizador com o sistema. Na Os diagramas de caso de uso descrevem visualmente as funcionalidades do sistema que vai ser projetado, e vai demonstrar as diferentes interações do utilizador com o sistema. Na Figura 12 descrevemos os casos de uso do nosso projeto.

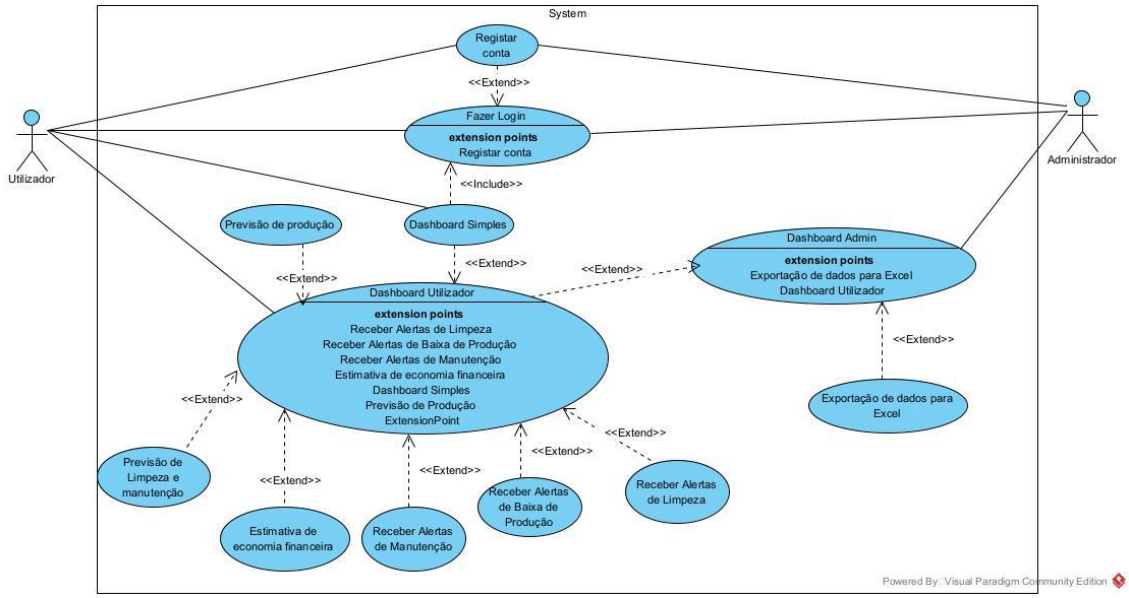

Figura 12 - Diagrama de Caso de Uso

5.3 Diagramas de Atividade

Um diagrama de atividade descreve visualmente o comportamento e as atividades executadas sequencialmente pelo sistema. Da Figura 13 à Figura 17 estão apresentados os diagramas de atividades que formulámos com base nos casos de uso apresentados na secção anterior.

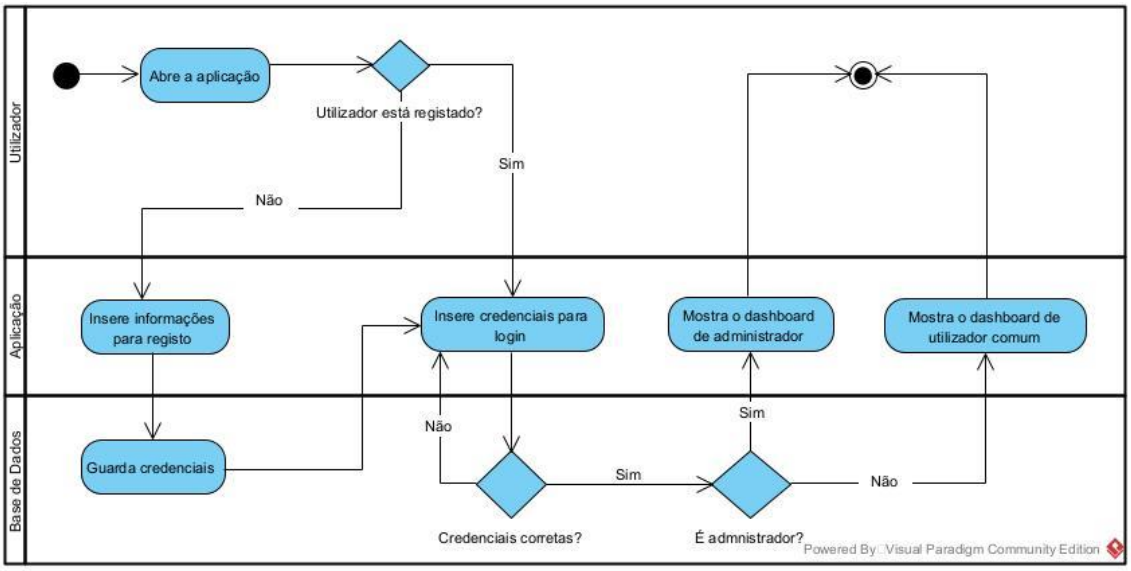

Figura 13 - Diagrama de Atividades, Login e Registo

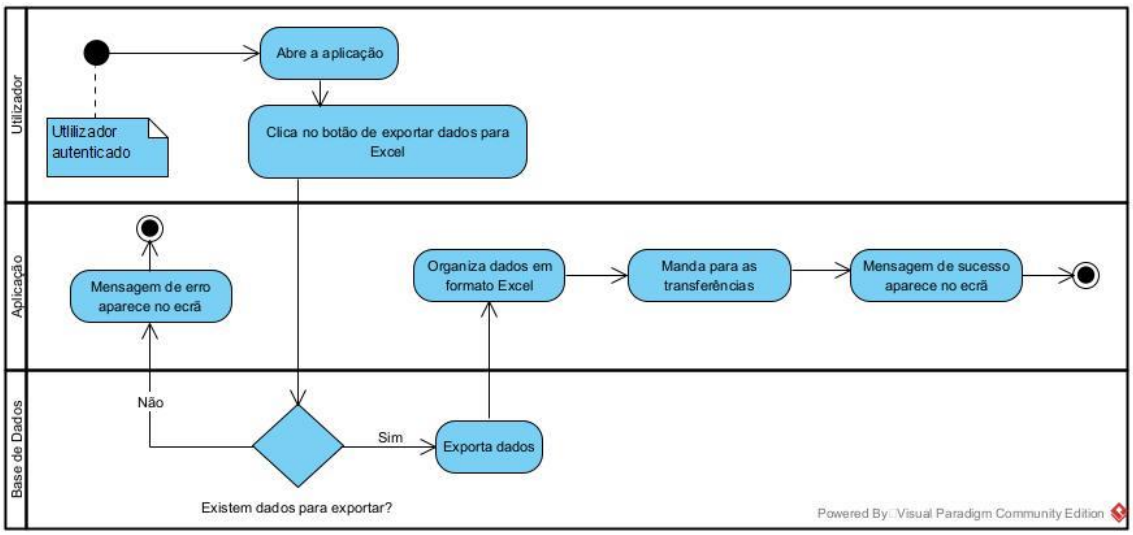

Figura 14 - Diagrama de Atividades, Exportar dados de produção para Excel

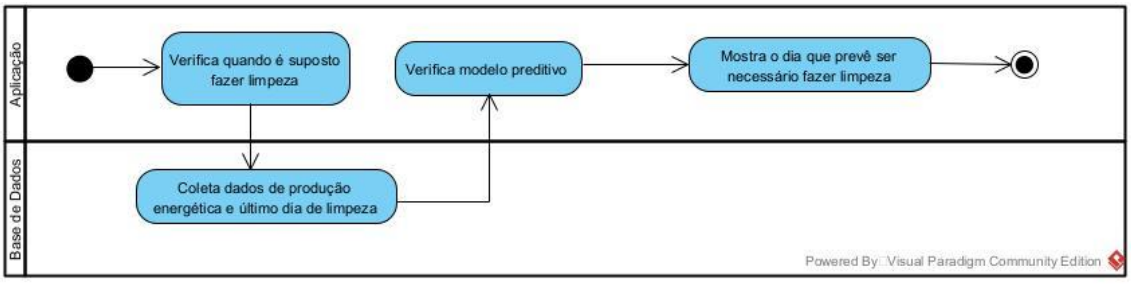

Figura 15 - Diagrama de Atividades, Previsão do dia de limpeza

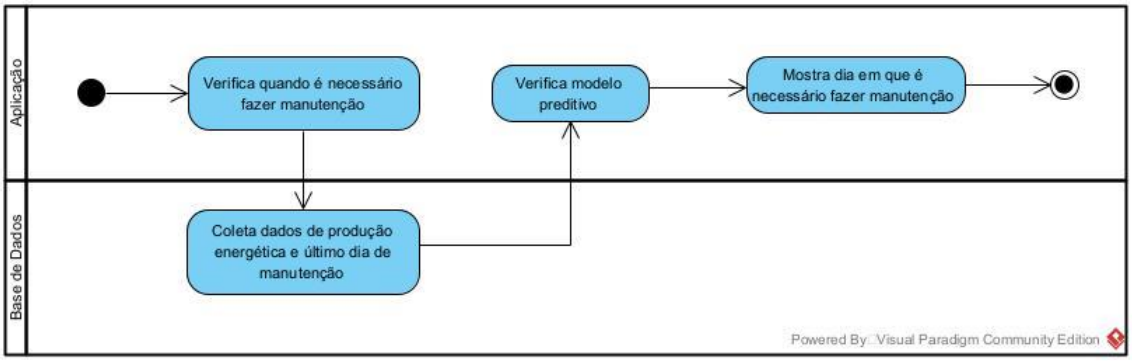

Figura 16 - Diagrama de Atividades, Previsão do dia de manutenção

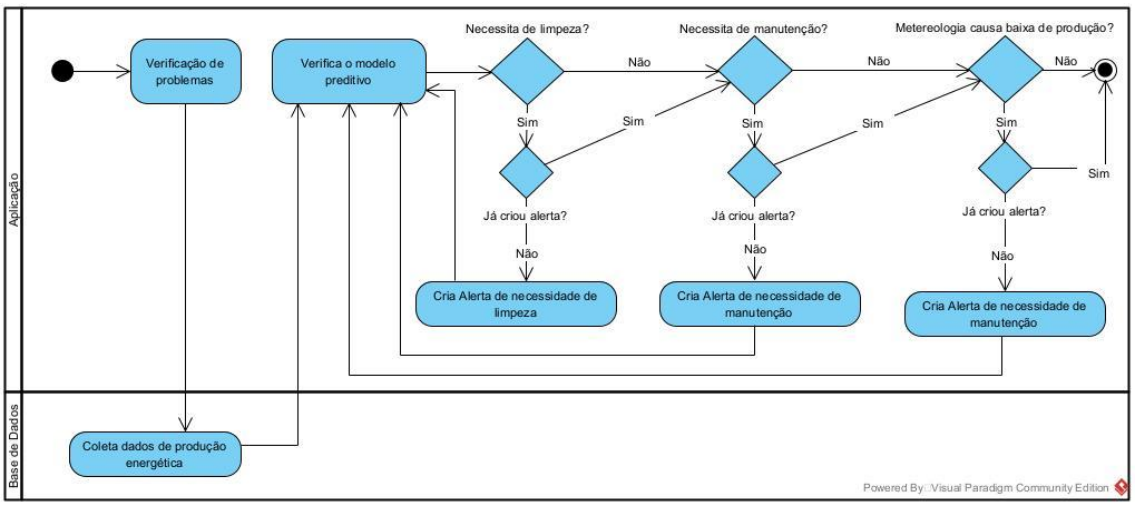

Figura 17 - Diagrama de Atividades, Sistema de Alertas

5.4 Base de Dados

Como os dados necessários para este projeto não têm uma grande complexidade relacional e é necessária uma grande flexibilidade, optámos por utilizar o MongoDB para fazer a gestão dos dados de produção energética diária bem como a temperatura nesse mesmo dia. Estes dados serão extraídos do software antigo. Iremos também guardar informações como os detalhes de cada utilizador, visto que será necessário haver distinção entre os diferentes perfis e informações sobre as datas em que são feitas limpezas e manutenções para que seja possível conseguir alertar os responsáveis quando for necessário limpar ou inspecionar novamente as plantas. Na Figura 18 podemos observar o diagrama de entidade relação para a nossa base de dados.

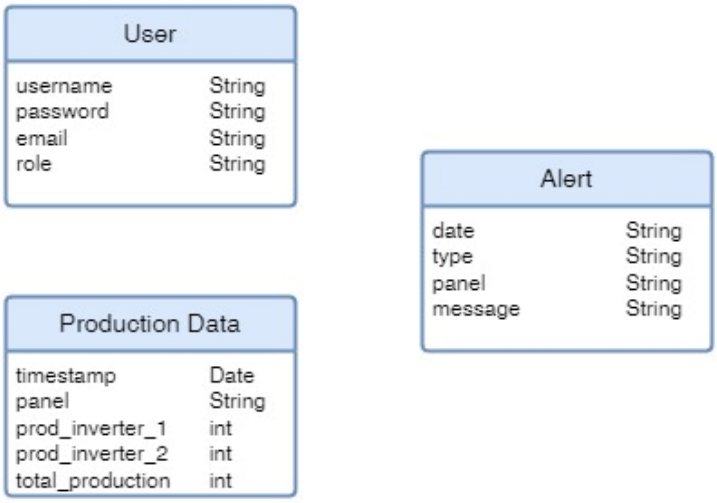

Figura 18 - Diagrama Entidade Relação modelado a partir da API atual

Nesta figura, na entidade "*Production Data*" é colocado a informação de produção do inversor 1 e inversor 2, porque os dados provenientes da API são mostrados assim. Na eventualidade de numa sequência deste projeto se encontrar outra forma de coleção de dados, poderá ser criada uma outra entidade separada para a produção em cada inversor. Esta modelação com entidade extra pode ser vista na Figura 19.

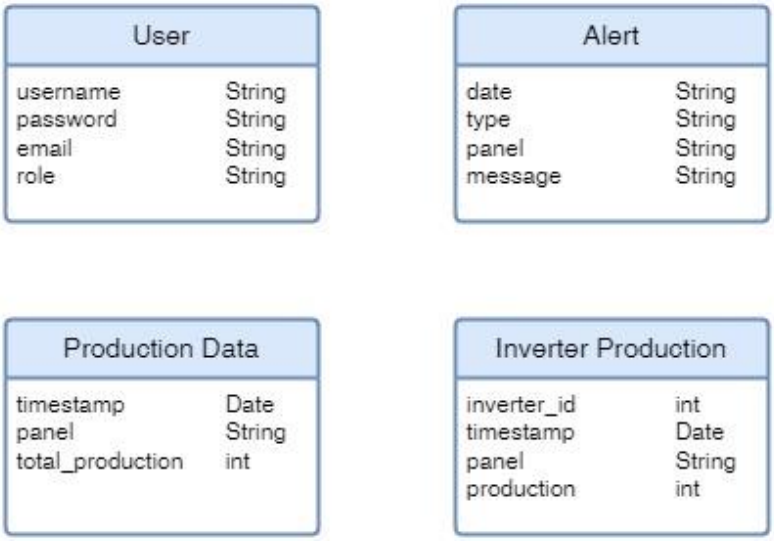

Figura 19 - Diagrama Entidade Relação modelado para a existência de informação extra sobre inversores

5.5 Mockup da aplicação

Visto que é extremamente importante para os *stakeholders* a boa utilização em dispositivos móveis, elaborámos um mockup usando a ferramenta de prototipagem Figma [FIGM24]. Desenvolvemos uma ideia inicial (Figura 20) do que será visto num dashboard principal, um menu que irá ser usado para navegar entre as diversas páginas e um ecrã de login.

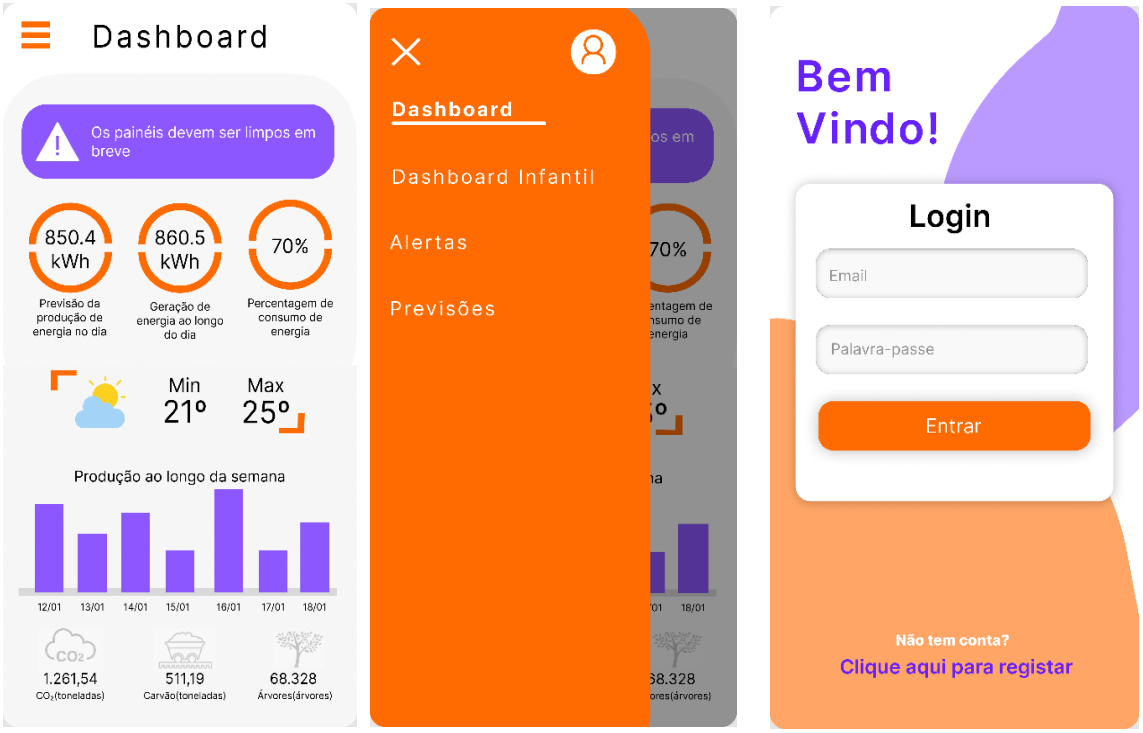

Figura 20 - Mockup da Aplicação

6 Solução Proposta

6.1 Arquitetura

Abaixo na Figura 21 apresentamos a solução inicialmente pensada para o desenvolvimento da nova aplicação para atender às necessidades de monitoramento dos dados provenientes dos painéis solares, previsões de dados de energia produzida face a diversos tipos de fatores, e sistema de alertas.

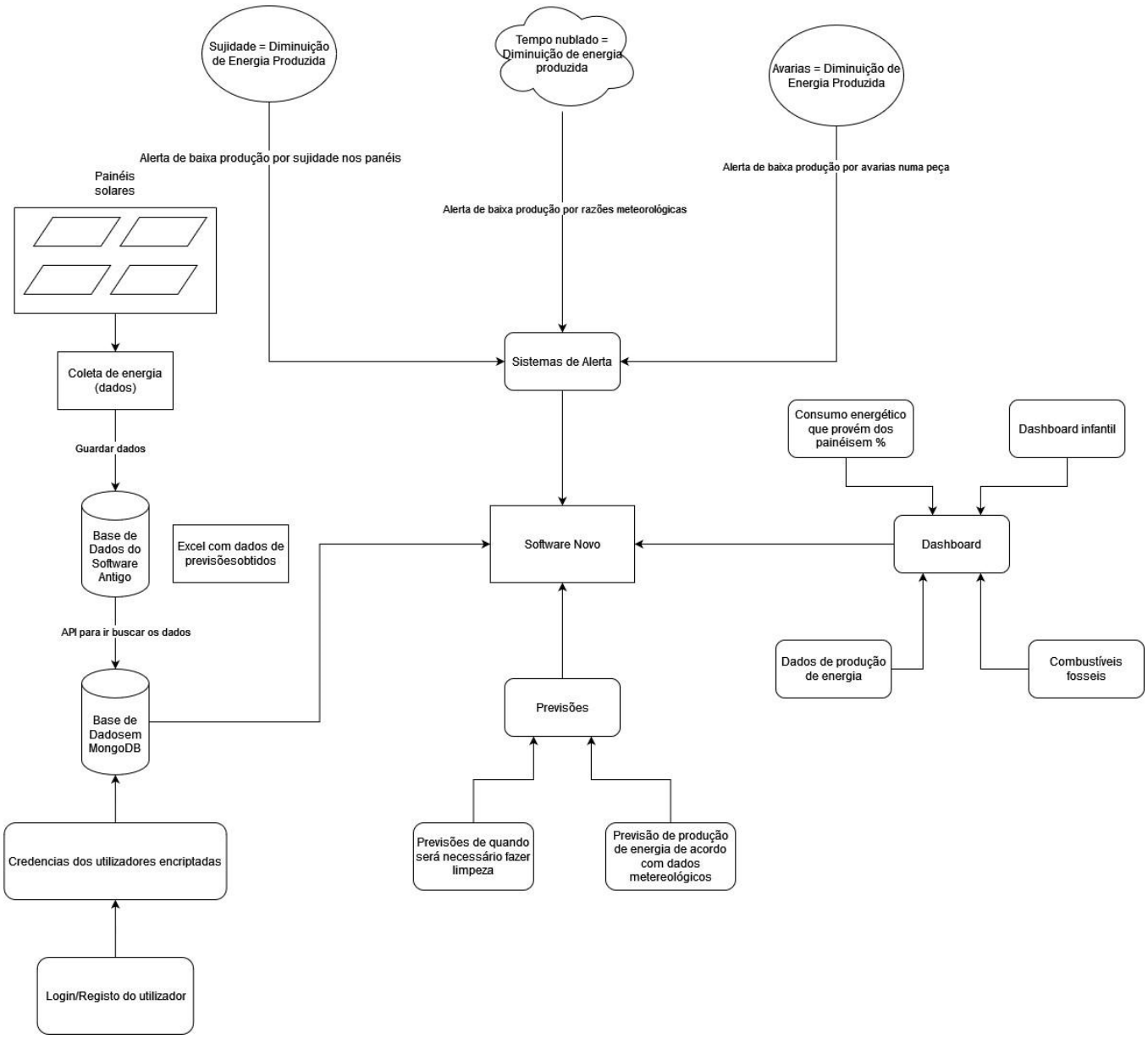

Figura 21 - Arquitetura da solução proposta inicial

Esta arquitetura é composta pelos seguintes componentes principais:

- Módulo de extração de dados, descrito em detalhe na Secção 6.3

- Módulo de carregamento na base de dados, descrito em detalhe na Secção 6.4
- Módulo de desenvolvimento do *front-end*, descrito em detalhe na Secção 6.5
- Módulo de desenvolvimento do *back-end,* descrito em detalhe na Secção 6.6
- Módulo de aplicação da API GoSungrow, descrito em detalhe na Secção 6.7
- Módulo de aperfeiçoamento do modelo preditivo, descrito em detalhe na Secção 6.8

O código fonte da nossa aplicação está no link abaixo:

https://github.com/DEISI-ULHT-TFC-2023-24/TFC-DEISI28-Aplicacao-Para-Monitorizacao-Duma-Central-de-Energia-Renovavel

A arquitetura efetivamente implementada, pode ser vista na figura abaixo.

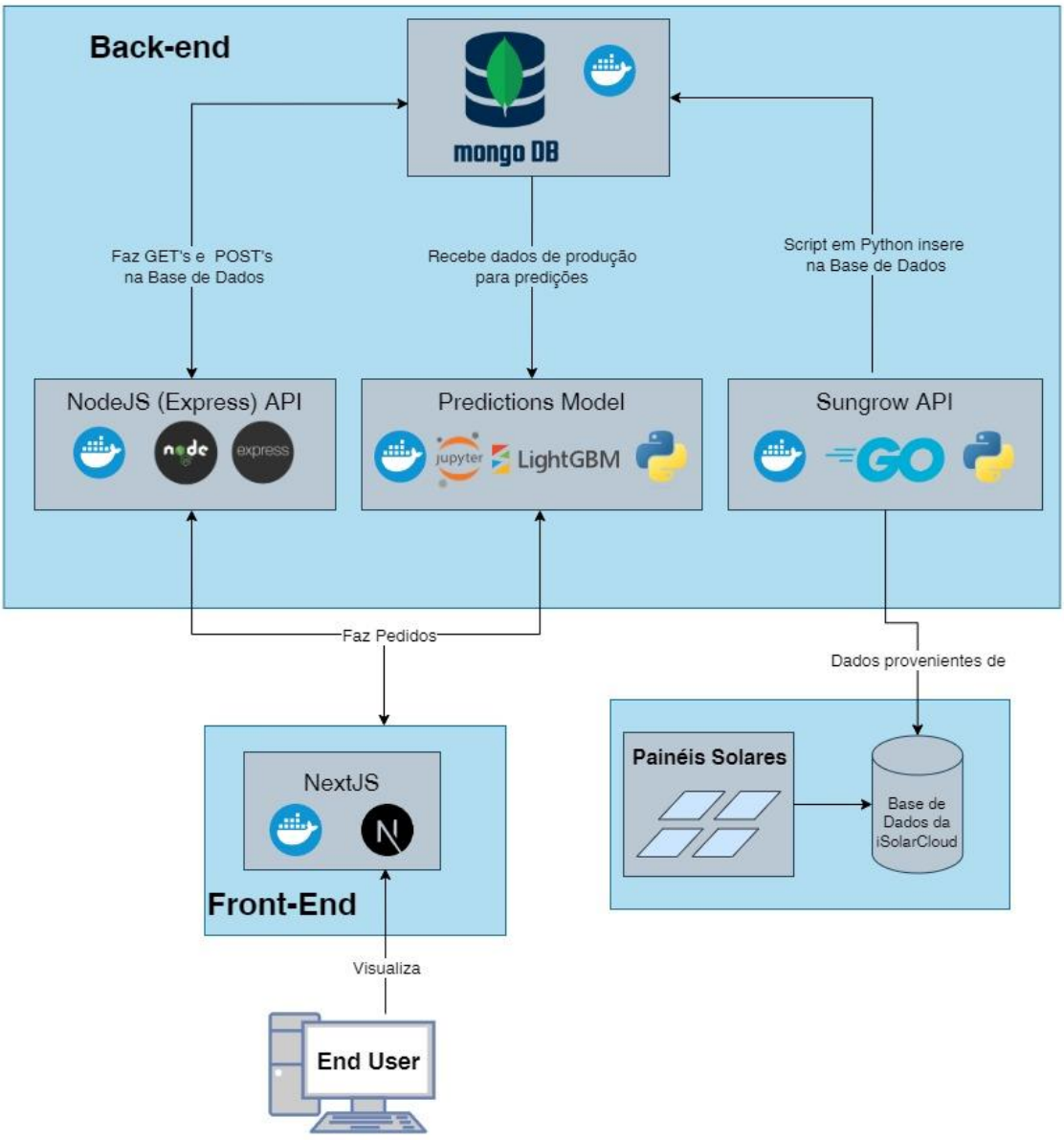

Figura 22 - Arquitetura Implementada

As componentes apresentadas nesta arquitetura são brevemente descritas abaixo, entrando em mais detalhe nas secções seguintes.

- ***Front-end:*** componente conteinerizado e desenvolvida usando NextJS, responsável pela parte visual da aplicação; faz conexão ao *back-end* e mostra ao utilizador de uma maneira que seja esteticamente apelativa;
- ***Back-end:*** Componente cujas subcomponentes estão todas conteinerizadas, responsável pela parte de lógica de negócio, armazenamento, comunicação com a base de dados, modelo de predição e obtenção de dados.
    - **Modelo de predição:** desenvolvido usando Python, Jupyter e LightGBM; responsável por fazer as previsões da produção de energia;
    - **Módulo NodeJS/Express:** desenvolvida para fazer ligação da base de dados com o *front-end* de modo a proporcionar todas as informações necessárias para o desenvolvimento;
    - **Módulo Sungrow *API*:** módulo que faz o *update* da base de dados com os dados de produção provenientes da aplicação antiga;
    - **Módulo de Base de Dados:** Base de dados em MongoDB que guarda informações de produção, utilizadores, alertas, etc.

6.2 Tecnologias e ferramentas utilizadas

Nesta solução são usadas as seguintes tecnologias:

- **Selenium** [SELE23]: um projeto com uma grande variedade de ferramentas que permite a automação de navegadores da Web e fornece extensões para manipular o navegador nativamente tal como um utilizador faria, usando um *Webdriver*. Foi utilizado no início, mas reparou-se lento e ineficaz para a nossa solução proposta.
- **Docker** [DOCK24]**:** Docker é uma plataforma aberta para desenvolvimento, entrega e execução de aplicações. O Docker permite separar aplicações da infraestrutura para que seja possível entregar software rapidamente.
- **MongoDB** [MGDB24]**:** O MongoDB é uma base de dados não relacional orientada a documentos projetada para desenvolvimento de aplicações e escalabilidade.
- **NodeJs** [NODE24]**:** O NodeJS é um ambiente de execução open-source, multiplataforma, com base na engine de Javascript V8 do Google Chrome
- **ExpressJs** [EXJS24]**:** O ExpressJS é uma framework para NodeJS que fornece recursos para a contrução de aplicações móveis e na Web.
- **NextJs**[NEXT24]**:** O NextJS é uma framework de React, permitindo criar aplicações web de alta qualidade com o poder dos componentes React.
- **GoSungrow**[GOS24]**:** O GoSungrow é uma API iSolarCloud escrito em GoLang, onde fornece acesso imediato a todas as chamadas de API e fornece gráfico, tabelas e dados diários, mensais e anuais.

6.3 Extração de dados

Na extração dos nossos dados, decidimos extrair 3 tipos: Geração da energia diária, Clima e Temperatura. Todos os nossos dados são coletados ao fim de cada dia, em intervalos de 5 minutos.

Este processo todo de extração é automatizado e realizado pelo Selenium.

Na Figura 23 conseguimos ver a representação dos 3 tipos de dados no dia 28 de novembro de 2023. A vermelho representa a geração de energia diária, a verde representa o clima e a azul a temperatura.

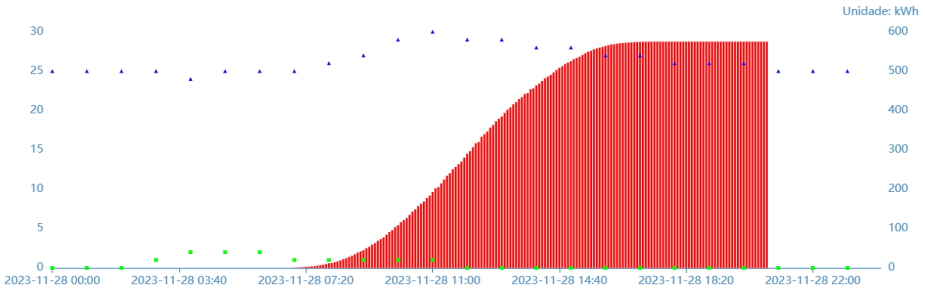

Figura 23 – Representação da energia diária, clima e da temperatura (28-11-2023)

Ao decorrer do nosso projeto, percebemos que o Selenium não seria a nossa melhor opção para a extração dos dados. Além de ser lento, o Selenium utiliza *Web Scraping*, uma solução para aceder aos dados estruturados da web de uma forma automatizada útil quando o site que pretendemos obter os dados não tem uma API, ou caso tenha, forneça apenas um acesso limitado aos dados. Como o website com os dados era alterado constantemente, era necessário alterar o script, o que tornou a utilização do Selenium impossível. Foi necessário encontrar outra alternativa para a coleta dos dados.

Resolvemos usar uma API escrita em GoLang, GoSungrow [GOS24] feita por um utilizador do GitHub, para podermos extrair os nossos dados de forma rápida e eficaz. Os dados são obtidos a através de comandos específicos no Terminal. A explicação mais detalhada da pesquisa feita e modo de uso desta API pode ser vista na Secção 6.7.

6.4 Carregamento de dados

Após a coleta dos dados estes são guardados numa base de dados baseada em MongoDB cujo diagrama de entidade-relação pode ser visto na secção 5.4.

A utilização do MongoDB para este projeto deve-se à falta de complexidade relacional dos dados que vão ser obtidos, a flexibilidade que permite caso seja necessário mudar algum parâmetro, e a sua escalabilidade. Também foi considerado a vertente educativa pelo que nunca tinha sido abordada uma base de dados NoSQL (ou não relacionais).

6.5 Desenvolvimento do *front-end*

Para o desenvolvimento do *front-end* da aplicação estudou-se utilizar React pela facilidade em atualizar e renderizar facilmente e de forma rápida as componentes à medida que os dados mudam, o que se encaixaria na nossa aplicação já que os dados são atualizados muito frequentemente. Percebemos, no entanto, que seria melhor usar o Next.js, pois é uma *framework* em React, e é uma estrutura pronta para a produção rápida de sites na web.

6.5.1 Login

Para o utilizador se autenticar na página de Login, precisámos de criar uma função assíncrona, com o objetivo de realizar o processo de autenticação de utilizadores na aplicação web, onde envia os dados de login, o nome de utilizador e a senha, para um servidor, posteriormente

validando as credenciais fornecidas pelo utilizador. O servidor responde se as credenciais são válidas ou não.

A função é definida como assíncrona ('*async*'), o que permite o uso de '*await*' dentro para lidar com operações assíncronas e recebe dois parâmetros, o '*username*' e a '*password*'. Usar uma função assíncrona, como neste caso, é importante quando se trata de operações que envolvem a comunicação com servidores. Javascript é uma linguagem de execução única (*single-threaded*), o que significa que uma operação demorada pode bloquear a execução de outras operações. Usar uma função assíncrona permite que essas operações ocorram em segundo plano, sem bloquear as restantes operações, melhorando a performance.

Para a tentativa de envio dos dados de login, a função tenta enviar uma solicitação '*POST*', um método utilizado para enviar dados para o servidor, para o endpoint do login, usando a API '*fetch*'. A solicitação é configurada com um *header 'Content-Type: application/json'* para indicar que os dados são enviados em formato JSON. O bo*dy* é construído utilizando um '*JSON.stringify'* para transformar os dados do nome de utilizador e senha em uma string JSON.

No tratamento da resposta do servidor, após enviar a solicitação, a função espera a resposta do servidor ('*await fetch(...)*'). Se a resposta não estiver OK ('*!res.ok*'), a função tenta ler a resposta JSON do servidor e lança um erro com informações detalhadas sobre o status HTTP e a mensagem de erro. Se a resposta estiver OK, a função retorna um array contendo o status HTTP e os dados JSON da resposta do servidor.

No tratamento de erros, se ocorrer algum erro durante o envio da solicitação ou ao processar a resposta, a função captura o erro no bloco '*catch*' e lança um novo erro com a mensagem '*Fetch error*'.

```
export async function validateLoginForm(username : string, password: string) {
  try {
      var res = (await fetch('http://localhost:5000/login',
      {
          method: "POST",
          headers: {
              "Content-Type": "application/json",
          },
          body: JSON.stringify
          ({
              username : username,
              password : password
          })
      }));

      if (!res.ok) {
          const responseData = await res.json();
          throw new Error(`HTTP error! Status: ${res.status}, Message: ${JSON.stringify(responseData)}`);
      }

      return [res.status, await res.json()];

  } catch (error) {
      throw new Error(`Fetch error:`);
  }

}
```

Figura 24 - Função 'ValidateLoginForm' para validar os dados de login

6.5.2 Dashboard

No *dashboard*, temos apresentado um gráfico onde mostra os dados de produção do dia anterior. Para isso, usamos a função '*useEffect*' para buscar os dados de produção a partir do servidor, onde é executada quando o componente é mostrado pela primeira vez. A função obtém os dados de produção do dia anterior e é atualizada.

Dentro do '*useEffect*', é definida uma função assíncrona *'getProduction'* que recebe dois parâmetros: *'startData'* e '*endData'*, representando as datas de início e fim do intervalo de produção a ser obtido. A função utiliza uma solicitação 'GET', um método para obter alguma informação hospedada no servidor, ao *endpoint* '*getProductionData'* no *back-end*.

No tratamento da resposta de servidor, a função vai verificar através da condição, se a resposta foi bem-sucedida ou não, utilizando a mesma dinâmica.

Na atualização do estado do componente, os dados de produção recebidos são armazenados no estado do componente utilizando '*setProductionData*'. Os valores de "*Total Production*" são extraídos dos dados e armazenados no estado com '*setTotalProduction'* e a soma dos valores de "*Total Production*" é calculada e armazenada no estado com '*setTotalProductionSum'.*

Caso ocorra algum erro durante o processo de obtenção ou manipulação dos dados, o erro é capturado e registado no console.

```
useEffect(() => {
  async function getProduction(startData: String, endData: String) {
    try {
      const response = await fetch(`http://localhost:5000/getProductionData?startDate=${startData}&endDate=${endData}`);

      if (!response.ok) {
        throw new Error('Failed to fetch production data');
      }

      const data = await response.json();
      setProductionData(data);

      const totalProductionValues = data.map((item:any) => item["Total Production"]);
      setTotalProduction(totalProductionValues);

      var sum = 0;

      totalProductionValues.forEach((p : number) => {
        sum += p;
      });

      setTotalProductionSum(sum);
    } catch (error) {
      console.error('Error fetching data:', error);
    }
  }
```

Figura 25 - Função 'getProduction' para obter os dados de produção do dia anterior

Além do gráfico onde mostra os dados do dia anterior, temos também um gráfico onde mostra os dados de produção acumulada e os de produção total nos últimos 7 dias.

Foi utilizada a mesma dinâmica de programação da função anterior, mas com o diferencial em que calcula a produção acumulada nos últimos 7 dias.

Na atualização do estado, os dados de produção são armazenados no estado do componente utilizando '*setProductionData*' e os valores de "*Total Production*" são extraídos e armazenados no estado com '*setTotalProduction*'.

Para o calculado da produção acumulada em períodos de 7 dias, utilizamos um loop '*forEach*' onde percorre os valores de produção total. Sempre que um valor zero é encontrado, e se a soma acumulada não for zero, a soma é adiciona ao *array* '*prod7days*' e a soma é reiniciada. A produção é acumulada até encontrar um valor zero, que sinaliza o fim do período de produção e a contagem começa novamente, permitindo calcular a produção total entre os períodos. O array '*prod7days*' é então armazenada no estado com '*setTotalProdcution7days'.*

Outro loop '*forEach*' calcula a soma de todos os valores de produção, que é armazenada no estado com '*setTotalProductionSum*'.

```
useEffect(() => {
  async function getProduction(startData: String, endData: String) {
    try {
      const response = await fetch(`http://localhost:5000/getProductionData?startDate=${startData}&endDate=${endData}`);

      if (!response.ok) {
        throw new Error('Failed to fetch production data');
      }

      const data = await response.json();
      setProductionData(data);

      const totalProductionValues = data.map((item:any) => item["Total Production"]);
      setTotalProduction(totalProductionValues);

      var sum7days = 0
      const prod7days : any = [];

      totalProductionValues.forEach((p : number) => {
        if (p == 0 && sum7days != 0){
          prod7days.push(sum7days)
          sum7days=0
        }
        sum7days += p;
      });

      setTotalProduction7days(prod7days)
```

Figura 26 - Função '*getProduction*' para obter os dados de produção dos últimos 7 dias (parte 1)

```
      var sum = 0;

      totalProductionValues.forEach((p : number) => {
        sum += p;
      });

      setTotalProductionSum(sum);

    } catch (error) {
      console.error('Error fetching data:', error);
    }
  }
```

Figura 27 - Função '*getProduction*' para obter os dados de produção dos últimos 7 dias (parte 2)

Para finalizar, temos a informação de meteorologia da cidade e capital de Omã, Muscat, onde ficam situados os painéis solares.

Para obter os dados meteorológicos, é declarada uma função assíncrona que retorna os dados. Para isso, temos três variáveis: a variável '*apiKey*' onde é atribuída o valor da chave da API não exposto para fins de segurança, a variável '*city*' onde é atribuída a cidade onde vamos buscar os dados meteorológicos e por fim a variável 'url' que é construída utilizando a cidade e a chave da API. A URL é configurada para obter os dados meteorológicos no formato métrico (Celsius) a partir de um *endpoint*.

```
export async function getData() {

  let apiKey = process.env.API_WEATHER_KEY;
  let city = 'muscat';
  let url = `http://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`

  var data = await (await fetch(url)).json();

  return data;
}
```

Figura 28 - Função '*getData*' para obter os dados meteorológicos

6.5.3 Painel de Administrador

No nosso painel de administrador, temos uma componente onde mostra as informações do administrador, como o seu *username* e o seu *email*.

A função é definida como uma função assíncrona, responsável por buscar os dados do utilizador e identificar o utilizador com a *role* de "Admin". Inicialmente o '*admin_user* ' é definido como um objeto '*user*' vazio. A função percorre o array de utilizadores e identifica o utilizador cujo papel é "*Admin*" e sendo assim é atribuído à variável. Após identificar o administrador, a função atualiza as variáveis com os valores correspondentes do utilizador administrador.

```
export function AdminInfo() {

    const [adminUsername, setAdminUsername] = useState<String>("");
    const [adminEmail, setAdminEmail] = useState<String>("");


    useEffect (() => {
        async function getAdminUser ()  {
            try {
                const response = await fetch('http://localhost:5000/getAllUsers');

                const data_users = await response.json();

                let admin_user: User = {username: "", password: "", email: "", role: ""};

                data_users.forEach(function(user : User) {
                    if(user.role === "Admin") {
                        admin_user = user;
                    }
                });

                setAdminUsername(admin_user.username);
                setAdminEmail(admin_user.email);

            } catch (error: any) {
                console.error('Error fetching users:', error.message);
            }
        }

        getAdminUser();
    }, [])
```

Figura 29 - Função 'AdminInfo' para obter informações do Adminstrador

O painel contém uma tabela onde mostra todos os utilizadores da aplicação, além de mostrar todos os utilizadores, também dá para registar um utilizador ou deletar um utilizador.

A função '*getUsers*' tem como objetivo, ir buscar, validar e processar os dados do utilizador vindos do servidor.

Solicitamos os dados de utilizador através do '*fetch*' onde envia uma solicitação '*GET*' ao *endpoint.* A resposta depois é verificada com '!*response.ok*' para assegurar que a solicitação foi bem-sucedida e então a resposta JSON é armazenada na variável '*data_users*'. Se não for, um erro é lançado.

Na validação dos dados, verifica se '*data_users*' é um array e se não está vazio, isto tem com objetivo verificar se há ou não utilizadores registados. Se a verificação falhar, um erro é lançado.

Na atualização do estado, se os dados são válidos, os estados '*users*' e '*sortedData*' são atualizados com '*setUsers(data_users)*' e '*setSortedData(data_users)*', para colocar os utilizadores e ordená-los.

Se ocorrer algum erro durante a solicitação ou processamento dos dados, o erro é capturado e uma mensagem de erro é registada no console.

```
useEffect(() => {
  async function getUsers() {
    try {
      const response = await fetch('http://localhost:5000/getAllUsers');

      if (!response.ok) {
        throw new Error('Failed to fetch users: ' + response.status);
      }

      const data_users = await response.json();

      if (!Array.isArray(data_users) || data_users.length === 0) {
        throw new Error('No users found');
      }

      setUsers(data_users);
      setSortedData(data_users);

    } catch (error: any) {
      console.error('Error fetching users:', error.message);
    }
  }

  getUsers();
}, []);
```

Figura 30 - Função 'getUsers' para obter todos os utilizadores

A função '*deleteUser*' é utilizada para enviar uma solicitação de exclusão de um utilizador do servidor. Após a exclusão, se for bem-sucedida, a lista de utilizadores (*'users'*) é atualizada para remover o utilizador excluído. Isto é feito filtrando o array, para excluir o utilizador cujo '*username*' corresponde ao fornecido. O estado '*setUsers*' é atualizado com a nova lista de utilizadores e a função '*setSortedData*' é chamada para atualizar a lista ordenada de utilizador com base nos novos dados e critérios de ordenação e pesquisa.

Se ocorrer algum erro durante a solicitação ou o processamento da resposta, o erro é capturado e uma mensagem de erro é registada no console.

```
const deleteUser = async (username: string) => {
  try {
    const response = await fetch('http://localhost:5000/delete', {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ username })
    });

    if (!response.ok) {
      throw new Error('Failed to delete user');
    }
    const updatedUsers = users.filter(user => user.username !== username);
    setUsers(updatedUsers);
    setSortedData(sortData(updatedUsers, { sortBy, reversed: reverseSortDirection, search }));
  } catch (error) {
    console.error('Error deleting user:', error);
  }
};
```

Figura 31 - Função 'deleteUser' para deletar um utilizador

A função '*RegisterUser*' é utilizada para enviar os dados de registo de um novo utilizador ao servidor, com os dados do '*username*', '*password*', '*email*' e '*role*'.

Se o registo de utilizador foi bem-sucedido, uma mensagem de alerta ('*User registered successfully*') é exibida.

Se ocorrer algum erro durante a solicitação ou o processamento da resposta, o erro é capturado e uma mensagem de erro é registada no console. Uma mensagem de alerta ('*Failed to register user. Please try again.*') é exibida para informar o utilizador.

```
const RegisterUser = async () => {
  try {
    // Enviar os dados do formulário para o servidor
    var response = (await fetch ('http://localhost:5000/register',
    {
      method: "POST",
      headers: {
          "Content-Type": "application/json",
        },
      body: JSON.stringify
      ({
          username : username,
          password : password,
          email : email,
          role : role,
      }),
    }));

    if (!response.ok) {
      throw new Error('Failed to register user');
    }

    alert('User registered successfully');

  } catch (error) {
    console.error('Error registering user:', error);
    alert('Failed to register user. Please try again.');
  }
};
```

Figura 32 - Função 'RegisterUser' para registar um novo utilizador

6.5.4 Aplicação

Em baixo, apresentamos a nossa aplicação, mostrando os principais componentes e aspetos visuais da aplicação. Pela dificuldade de implementação do sistema de coleção de dados automatizado todos os dias, os dados apresentados no *dashboard* são atualmente estáticos. A utilização dos dados estáticos dá-se porque ainda não foi conseguido fazer a automatização total dos dados de produção diários para a base de dados e, para não dar erro ao utilizar dados atuais, foi decidido utilizar dados estáticos. A visualização da temperatura atual considera a localização em Oman, Muscat, usando os dados provenientes da API do OpenWeatherMap [OPWE24].

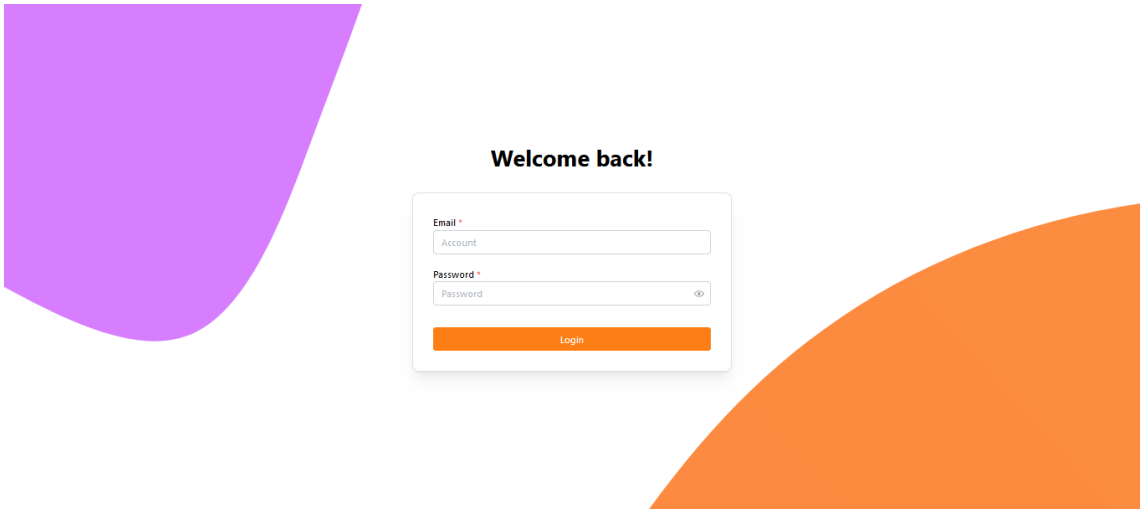

Figura 33 - Página Login

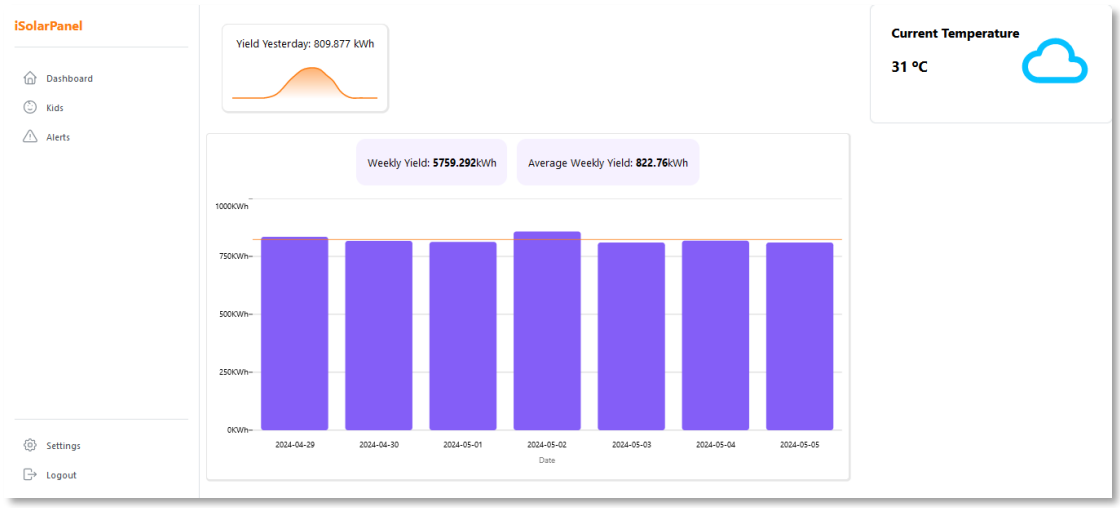

Figura 34 – Página Dashboard

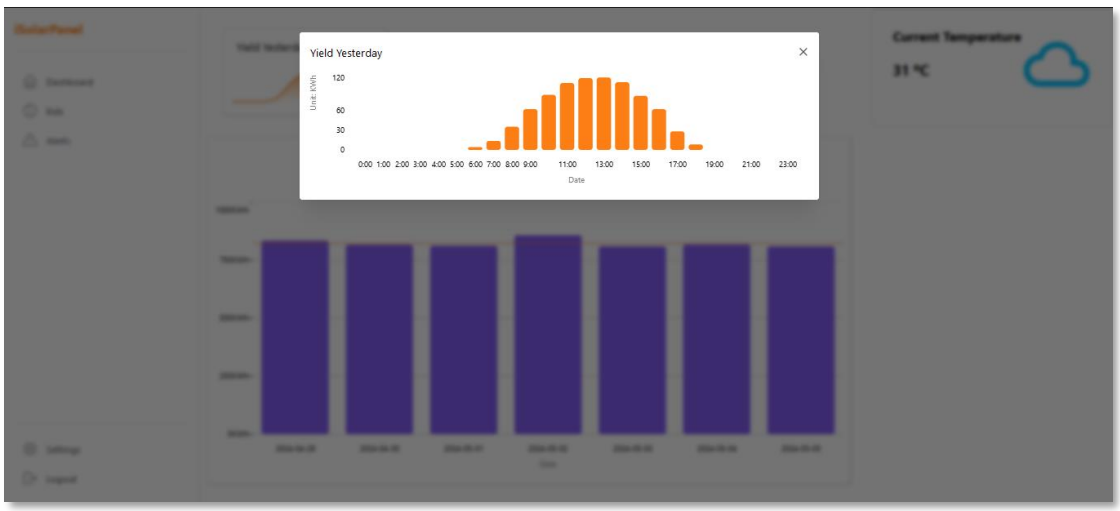

Figura 35 - Página Dashboard

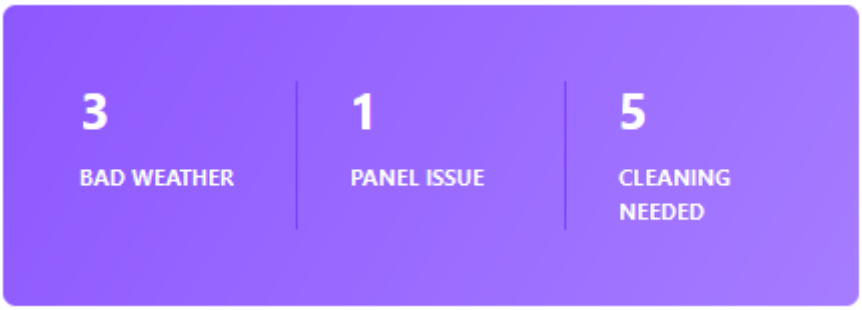

Figura 36 - Tabela de Alertas

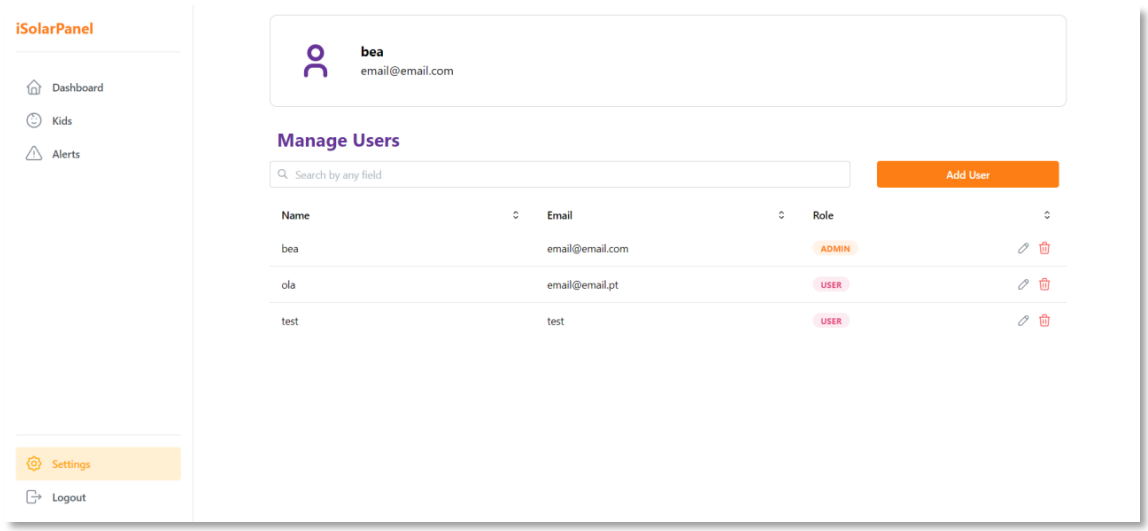

Figura 37 - Página do Administrador

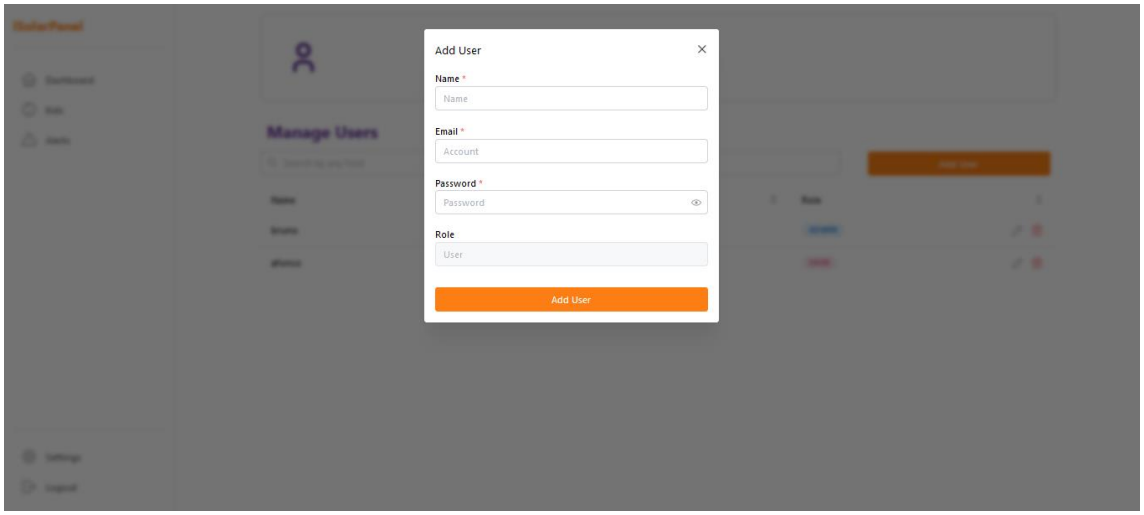

Figura 38 - Página de Adminstrador (Add User)

6.6 Desenvolvimento do back-end

Para o desenvolvimento do *back-end*, considerando que a base de dados escolhida foi MongoDB e a tecnologia equacionada para *front-end* é NextJS, consideramos usar a *stack* MERN (MongoDB, Express, React, Node) [MERN24], usando Express.js e Node.js para lidar com a parte do servidor.

```
const url = 'mongodb://database:27017';

let db;
let users;
let production;
let alerts;

const port = 5000;

const client = await MongoClient.connect(url)

db = client.db('tfc');
users = client.db('tfc').collection('users');
production = client.db('tfc').collection('Production');
alerts = client.db('tfc').collection('alerts');
```

Figura 39 - Ligação à base de dados no *back-end*

No início do desenvolvimento do *back-end* começámos por ligar o servidor, que escuta a porta 5000, e fazer o acesso à base de dados em MongoDB na porta 27017 (a *default*).

Para essa ligação à base de dados, usámos a biblioteca oficial do MongoDB (mongodb) para o Node, no entanto, no momento de criação de *schemas* para a base de dados, esta biblioteca não foi o suficiente pelo que usámos também o mongoose. Os esquemas abaixo representados foram desenhados e pensados para quando é necessário ser inserido um novo utilizador ou um novo alerta.

```
const userSchema = new mongoose.Schema({
  username: String,
  password: String,
  email: String,
  role: String,
});

const alertSchema = new mongoose.Schema({
  date : String,
  type : String,
  panel: String,
  message : String,
})

const UserModel = mongoose.model('users', userSchema);
const AlertModel = mongoose.model('alerts', alertSchema)
```

Figura 40 - *Schemas* para inserção na base de dados

Consoante as necessidades da aplicação, desenvolvemos também vários *endpoints* que podem ser acedidos pelo *front-end*, e que serão explicados nos subtópicos mais abaixo.

Como o nosso *back-end* estava num domínio diferente do *front-end* tivemos alguns problemas com erros do CORS (*Cross-Origin Resource Sharing*), que é um mecanismo que permite recursos restritos em uma página web sejam solicitados por domínios diferentes daquele que solicitou o recurso original, de uma maneira segura e controlada. O CORS protege os usuários de ataques como *Cross-Site Forgery* (CSRF) e o Cross-Site Scripting (XSS).

Usando o ExpressJS, esta implementação é tão simples como fazer a instalação e import da biblioteca 'cors' com o npm, e colocar no código as linhas:

```
const app = express();

app.use(cors());
```

Figura 41 - Implementação do CORS

6.6.1 *Login* , Registo e Deleção de utilizadores

Para as funcionalidades de login, registo e deleção de utilizadores na aplicação foram desenvolvidos três *endpoints* explicados abaixo.

Para o registo, é recebido os parâmetros provenientes de um formulário no *front-end* com as informações do utilizador, nomeadamente o nome de utilizador, a password desejada, o email, e a sua *role*. É verificado então se o utilizador já existe na base de dados pelo seu *username*, caso já exista é retornado o *status code* 400 e uma mensagem a explicar o erro. Caso o utilizador não exista, a sua password é *hashed*, com uma função da biblioteca bcrypt para NodeJS, e salteada 10 vezes (valor passado também a essa função), de seguida o utilizador é criado usando o *schema* referido anteriormente e inserido na base de dados. Por fim, o servidor manda um *status code* 201 com uma mensagem de utilizador registado com sucesso. Caso algum erro ocorra durante todo este processo o servidor envia um *status code* 500.

```
app.post('/register', async (req, res) => {
  try {
    const { username, password, email, role } = req.body;
    const existingUser = await users.findOne({ username });

    if (existingUser) {
      return res.status(400).json({ message: 'User already exists' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new UserModel({ username, password: hashedPassword, email, role });
    await users.insertOne(newUser)

    res.status(201).json({ message: 'User registered successfully' });
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Figura 42 - *Endpoint* de registo de utilizador

Para o login do utilizador a lógica implementada é semelhante. É verificada a existência do utilizador na base de dados com o *username* passado no *body* do pedido, caso não exista retorna um *status code* 404, com mensagem de utilizador não encontrado, depois verifica a validade da password com a função compare do bcrypt, que compara as passwords hashed e retorna um valor *True* ou *False*, caso a password seja inválida o *status code* retornado é um 401. Quando todos os parâmetros estão corretos, é gerado um JSON Web Token, assinado com uma chave secreta que deverá ser colocada num .env na root do projeto, e este token é enviado ao front-end. No caso de erro é enviado um *status code* 500.

```
app.post('/login', async (req, res) => {

  try {
    const { username, password } = req.body;
    const user = await users.findOne({ username });

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    const isPasswordValid = await bcrypt.compare(password, user.password);

    if (!isPasswordValid) {
      return res.status(401).json({ message: 'Invalid password' });
    }

    const token = jwt.sign({ username: user.username, email: user.email, roles: user.roles }, process.env.JWT_SECRET_KEY);

    res.status(200).json({ token });

  } catch (error) {
    res.status(500).json({ message: 'Internal server error'});
  }
});
```

Figura 43 - *Endpoint* de login de utilizador

No *endpoint* de delete é feito o processo similar ao login, mas quando, e só se, o utilizador for encontrado então é feito a sua deleção da base de dados.

```
app.delete('/delete', async (req, res) => {
  try {
    const { username } = req.body;
    const user = await users.findOne({ username });

    if (!user) {
      return res.status(400).json({ message: 'User does not exist' });
    }

    await users.findOneAndDelete({ username });

    res.status(200).json({ message: 'User deleted successfully' });
  } catch (error) {
    console.error('Error deleting user:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Figura 44 - *Endpoint* de Deleção de utilizador

6.6.2 Obtenção de todos os utilizadores

Como se viu necessário obter todos os utilizadores para que possam ser expostos em tabela no *back-end* e para que o administrador possa fazer gestão dos mesmos, ao adicionar ou remover um determinado utilizador, desenvolvemos também um *endpoint* para que se consiga obter a sua totalidade.

Este *endpoint* procura todos os utilizadores e converte a resposta proveniente da base de dados para JSON, com os parâmetros de username, email do utilizador, e respetiva *role*, de forma que possa ser enviada para o front-end e exposta. No caso de existência de algum erro durante este processo, é enviado um *status code* 500 juntamente com a mensagem de erro indicativa de erro no servidor.

```
app.get('/getAllUsers', async (req, res) => {
  try {
    const allUsers = await users.find().toArray()

    const usersToJson = allUsers.map(user => (
      {
        username: user.username,
        email: user.email,
        role: user.role,
      }
    ))

    res.json(usersToJson)
  }
  catch (error) {
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Figura 45 - *Endpoint* obtenção de todos os utilizadores

6.6.3 Alertas

Para as funcionalidades relativas aos alertas foram desenvolvidos 3 *endpoints*: inserção, deleção e obtenção.

Na inserção de alerta o corpo do pedido ao servidor traz informações como a data, o tipo de alerta (meteorológico, avaria, etc.), a mensagem associada ao alerta, e o painel a que o alerta diz respeito. A partir dessa informação é usado o *schema* de Alerta, explicado acima, e é verificado na base de dados se o alerta já foi dado naquele dia a partir do seu tipo, data e painel. Caso já tenha sido inserido esse alerta na base de dados é enviado um *status code* 400 com uma mensagem explicativa da inserção do alerta naquele dia. Caso contrário, o alerta é inserido na base de dados e é retornado um *status code* 200.

```
app.post('/insertAlert', async(req, res) => {
  try {
    const { date, type, message, panel} = req.body;

    const newAlert = new AlertModel({ date, type, message, panel});

    const alreadyAlerted = await alerts.findOne({ date , type , panel});

    if (alreadyAlerted) {
      return res.status(400).json({ message: 'Already inserted this alert today' });
    }

    await alerts.insertOne(newAlert)

    res.status(200).json({ message: 'Alert inserted successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Internal server error' });
  }
})
```

Figura 46 - *Endpoint* de inserção de alertas

Na necessidade de o utilizador deletar um alerta, o *endpoint* de deleção de alerta permite que, através da data, tipo e painel referente ao alerta, se possa encontrar na base de dados e apagá-lo. No caso de o alerta não existir na base de dados o *status code* enviado é o 400, com a respetiva mensagem informativa. Caso exista, é mandado *status code* 200 com a mensagem de alerta eliminado com sucesso. Neste caso, se houver um erro na deleção é explicitado na consola, e enviado ao *front-end* um *status code* 500 com uma mensagem de erro no servidor.

```
app.post('/deleteAlert', async(req, res) => {
  try {
    const { date , type, panel } = req.body;
    const alert = await alerts.findOne({ date, type, panel });

    if (!alert) {
      return res.status(400).json({ message: 'Alert does not exist' });
    }

    await alerts.findOneAndDelete({ date, type, panel });
    res.status(200).json({ message: 'Alert deleted successfully' });

  } catch (error) {
    console.error('Error deleting alert:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
})
```

Figura 47 - *Endpoint* de deleção de Alerta

Para se obter todos os alertas, e posteriormente expor no *front-end* em forma de lista, foi desenvolvido o *endpoint* de obtenção de todos os alertas, que passa por buscar todos os alertas referentes a um determinado painel solar, transformar esse *output* em lista, e mapear essa informação para uma resposta em JSON que será enviada como resposta. Neste caso não verificamos se esta resposta pode ser vazia porque achámos que essa lógica deve ser implementada no *front-end*. No caso de algum erro ocorrer neste processo o código enviado é o 500 com respetiva mensagem de erro.

```
app.get('/getAllAlerts', async (req, res) => {
  try {

    const { panel } = req.query;

    const allAlerts = await alerts.find( {"panel" : panel } ).toArray()

    const alertsToJson = allAlerts.map(alert => (
      {
        date: alert.date,
        type: alert.type,
        message: alert.message,
        panel: alert.panel,
      }
    ))

    res.json(alertsToJson)
  }
  catch (error) {
    res.status(500).json({ message: 'Internal server error' });
  }
});
```

Figura 48 - *Endpoint* de obtenção de todos os alertas

6.6.4 Obtenção de dados de produção na base de dados

Para obter os dados de produção guardados na base de dados, foi criado um *endpoint* que recebe no corpo, uma data de começo e uma data final. Estas datas são colocadas em formato Date() , e é feita uma pesquisa na base de dados por dados de produção entre essas duas datas. Após essa pesquisa, o resultado é convertido numa lista e mandado como resposta em formato JSON, com o *status code* 200. Em caso de erro neste processo, é apresentado na consola o erro de pesquisa de dados e mandado um *status code* 500 com a respetiva mensagem, como resposta ao pedido.

```
app.get('/getProductionData', async (req, res) => {

  try {

    const { startDate, endDate } = req.query;

    const _startDate = new Date(startDate);
    const _endDate = new Date(endDate);

    const prod = await production.find({
      "Timestamp": {
        "$gte": _startDate,
        "$lt": _endDate
      }
    }).toArray();

    res.status(200).json(prod);
  }
  catch (err) {
    console.error('Error retrieving data:', err);
    res.status(500).send('Error retrieving data');
  }

});
```

Figura 49 - *Endpoint* de obtenção de dados de produção guardados na base de dados

6.6.5 Automatização da Inserção de Dados de Produção com Python

Para otimizar o processo de inserção de dados de produção na base de dados, desenvolvemos um script em Python. Este script pré-automatiza a transferência de dados da base de dados do iSolarCloud para a nossa base de dados, garantindo precisão, eficiência e redução de erros humanos. Para transferir os dados para a nossa base de dados, é preciso correr o script. Pensamos numa ideia de fazer uma *crontask* onde, todos os dias, numa certa hora, ele corria o script como uma tarefa, de forma automática, sem precisarmos de correr o script manualmente, mas não conseguimos concretizar essa ideia. A seguir, detalhamos o funcionamento do script, as suas principais funcionalidades e como ele se integra no nosso sistema.

O script é dividido em duas partes. A primeira parte é responsável por coletar os dados de produção dos inversores solares, processá-los e inseri-los no MongoDB.

Na importação das bibliotecas, temos o '*datetime*' e '*timedelta*' para a manipulação dos dados, 'pandas' é utilizado para a manipulação e análise de dados em forma de *DataFrame* e '*result*' é um módulo personalizado que contém funções auxiliares.

Na definição das datas de coleta, a data atual é obtida e, dependendo se é o primeiro dia do mês ou não, são calculadas 'start_Date' e 'end_Date' para definir o intervalo de coleta dos dados.

Os comandos para a coleta de dados, são definidos comandos específicos para coletar dados de dois inversores solares através da API GoSungrow.

Para a coleta e processamento de dados, utilizando a função *'result.command_production'*, os comandos são executados num container Docker para obter os dados de produção.

Na combinação e cálculo da produção total, os *dataframes* dos dois inversores são mesclados com base no *timestamp*, e surge um novo campo, '*Total Production*', onde é calculada a soma de produção dos dois inversores.

Para a conexão com o MongoDB, utilizamos a função '*result.connect_to_mongodb*' e se a conexão for bem-sucedida, os dados são convertidos para uma lista de dicionários e inseridos na coleção '*Production*'.

```python
from datetime import datetime, timedelta
import pandas as pd
import result

try:
    date = datetime.now()

    if date.day == 1:
        # Obter o último dia do mês anterior
        last_day_of_previous_month = date.replace(day=1) - timedelta(days=1)
        start_Date = last_day_of_previous_month.strftime('%Y%m%d')
        end_Date = date.strftime('%Y%m%d')
    else:
        start_Date = (date - timedelta(days=1)).strftime('%Y%m%d')
        end_Date = date.strftime('%Y%m%d')

    command_first_invertor = f"GoSungrow show point data 20240525 20240625 60 995790_1_1_1.p14"
    command_second_invertor = f"GoSungrow show point data 20240525 20240625 60 995790_1_2_1.p14"
    container_name = "api"
    painel="TAISM Solar plant"

    # Criar um DataFrame do pandas com os dados dos comandos dos inversores
    result_1 = result.command_production(container_name, command_first_invertor)
    df_1 = result.append_data_production(result_1.stdout,date.year,1)

    result_2 = result.command_production(container_name, command_second_invertor)
    df_2 = result.append_data_production(result_2.stdout,date.year,2)

    # Mesclar os DataFrames dos inversores
    df_combined = pd.merge(df_1, df_2, on='Timestamp', how='outer')
    df_combined.insert(1, 'Painel', painel)

    # Calcular a produção total dos inversores
    df_combined['Total Production'] = df_combined['Production Inverter 1'] + df_combined['Production Inverter 2']
    #print(df_combined)

    connection = result.connect_to_mongodb()

    if connection is None:
        print("Erro ao conectar ao MongoDB")

    if connection is not None:
        #converte o dataframe para uma lista de dicionarios
        data = df_combined.to_dict(orient="records")
        connection.insert_many(data)
        print("Dados inseridos no MongoDB com sucesso")

except Exception as e:
    print("Erro:", e)
```

Figura 50 - Coleta e Processamento de Dados

A segunda parte do script define funções auxiliares para a coleta, processamento e inserção dos dados.

A função '*connect_to_mongodb*' estabelece uma conexão com o MongoDb e retorna a coleção '*Production*', em caso de falha, é exibido um erro e retorna '*None*'

A função '*command_production*' executa um comando Docker dentro do container e verifica se foi executado com sucesso.

A função '*append_data_production*' processa a saída do comando docker, extraindo os dados de produção. Os dados são formatados num DataFrame, com as colunas '*Timestamp*' e '*Production Inverter X*'.

```python
import subprocess
import pandas as pd
import pymongo
import pymongo.errors
import pymongo.mongo_client

def connect_to_mongodb():
    try:
        client = pymongo.MongoClient("mongodb://localhost:27017")
        db = client['tfc']
        collection = db['Production']
        return collection

    except pymongo.errors.ConnectionFailure as e:
        print("Erro ao conectar ao MongoDB:", e)
        return None


def command_production(container_name,comando_docker):
    output = subprocess.run(
            ['docker', 'exec', container_name, '/bin/sh', '-c', comando_docker],
            capture_output=True,
            text=True,
            check=True,
            encoding='utf-8'
    )

    output.check_returncode()

    return output


def append_data_production(output,year,id_inverter):
    if output is None:
        raise RuntimeError("O comando Docker não produziu saída.")

    lines = output.splitlines()
    data = []

    for line in lines:
        if line.startswith(f'| {year}'):  # Identificar linhas com dados
            parts = line.split('|')
            timestamp = parts[1].strip()
            value = parts[3].strip()
            if value == '--':
                value = 0
            else:
                value = int(value)
            data.append((timestamp, value))

    return pd.DataFrame(data, columns=['Timestamp',f'Production Inverter {id_inverter}'])
```

Figura 51 - Definição de Funções Auxiliares

6.7 API iSolarCloud

Para conseguirmos obter com rapidez e facilidade os dados precisaríamos de uma *API* pública do iSolarCloud, mas a inexistência dessa levou-nos a tentar encontrar novas opções.

Encontrámos, assim, uma API iSolarCloud escrito em GoLang atualizada pelos inversores SunGrow. Esta API fornece acesso imediato a todas as chamadas de API por meio de uma estrutura simples de *GET/PUT*, gráficos de quaisquer pontos de dados, diários, mensais e anuais com granularidade de 5 minutos a 1 hora e saída de dados para diversos formatos de ficheiro. Para utilizar esta API foi necessário compilar o projeto do repositório do GitHub, o que gerou um ficheiro, e utilizando comandos dentro do terminal, podemos ter acesso às informações de produção vindas diretamente da base de dados da iSolarCloud de forma praticamente instantânea para depois inserir-se na base de dados. Este processo está exemplificado na Figura 52.

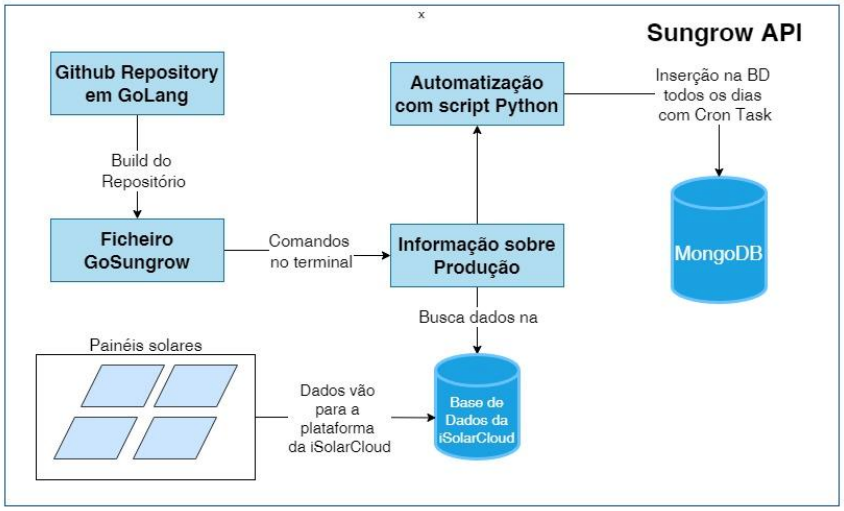

Figura 52 - Processo de compilação e uso da Sungrow API

A API está configurada no Docker, com a vantagem de haver um maior isolamento, portabilidade e escalabilidade.

Configurar esta *API* tomou bastante tempo e foi algo difícil. Primeiramente decidimos configurar a *API* localmente para efeitos de teste. No entanto, a *API* não era atualizada há algum tempo e surgiam erros de "*er_invalid_key*" ou "*Request is not encrypted*". Posteriormente, encontrámos um outro fork desse projeto no GitHub onde estava resolvido este problema. Depois de já estar configurado localmente, precisaríamos de configurar no Docker. Obtivemos alguns problemas no *Dockerfile* e também novamente problemas de "*Request is not encrypted*", mas isto foi resolvido fazendo a compilação do projeto em sistema Linux.

6.8 Aperfeiçoamento do modelo preditivo

Para o aperfeiçoamento do nosso modelo preditivo decidimos testar o modelo ARIMA e o modelo LightGBM para entendermos qual modelo tem o melhor desempenho na previsão dos nossos dados.

6.8.1 ARIMA

Conforme a figura abaixo, está representado a produção de energia ao longo de 3 semanas. A azul é usada para treinar o modelo, o verde para testar o modelo e o vermelho é a sua previsão.

Para o teste do modelo ARIMA foi usada a biblioteca Sktime com o método AutoARIMA [SKTA24], com um intervalo de confiança de 0.9 (representado a cinzento no gráfico), e usámos os mesmos dados de produção de energia que o LightGBM para treinar e testar o modelo. O AutoARIMA faz a seleção dos parâmetros '*p*', '*q*' e '*d*' automaticamente, e mostra-nos o resultado.

No nosso modelo, o AutoARIMA prevê bastante mal a produção de energia nos 5 dias seguintes. Assume que todos os dias, seja manhã ou noite, tem uma produção de energia constante. O modelo não está a capturar adequadamente a sazonalidade dos dados, por isso prevê valores constantes.

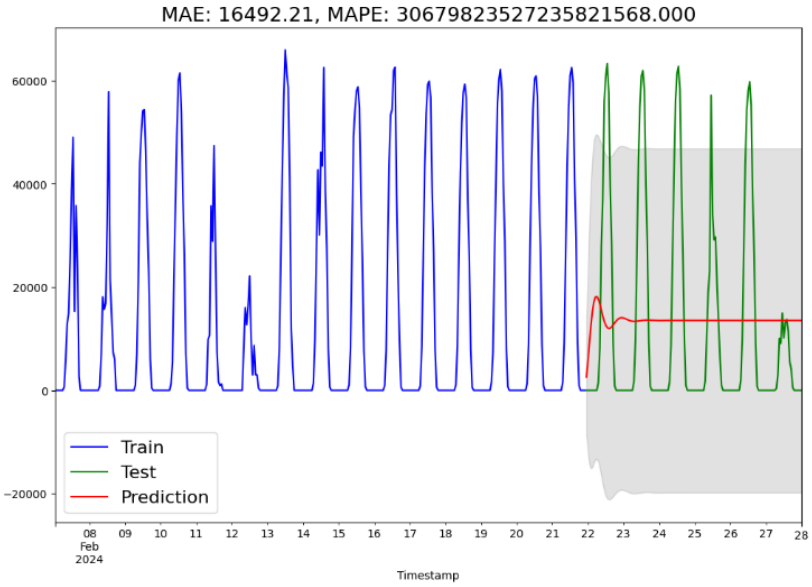

Figura 53 - Modelo preditivo ARIMA para a previsão da produção de energia

6.8.2 LightGBM

Conforme a figura abaixo, está representado a produção de energia ao longo de 3 semanas. A azul é usada para treinar o modelo, o verde para testar o modelo e o vermelho é a sua previsão.

Considerando a sazonalidade diária dos dados começámos primeiramente por colocar *features* de *lag* (ou variáveis de atraso), estas *features* envolvem deslocar os valores das variáveis ao longo do tempo, criando colunas no conjunto de dados representativas do valor da variável no passado.

Após uma análise ao tipo de problema e perspetiva de negócio, concluímos que o modelo deveria ter uma previsão mais pessimista na maioria dos casos, já que é melhor o cliente poder ver que produziu mais energia do que o valor que tinha sido previsto.

No nosso modelo, o LightGBM prevê bem a produção de energia nos 5 dias seguintes. Reparamos que os dados previstos são bastante semelhantes aos dados observados, ou seja, os dados que estamos a prever são muitos semelhantes aos dados reais de produção de energia.

O LightGBM é um modelo que captura muito bem a sazonalidade dos nossos dados devido à sua complexidade, ao contrário do modelo ARIMA observado na figura acima, onde os dados previstos eram constantes. Caso quiséssemos afinar ainda mais o modelo o LightGBM também seria bastante mais vantajoso, visto que podemos adicionar mais *features* e tentar tornar as previsões ainda mais precisas.

Uma vez que afinar o modelo demoraria mais tempo e ainda necessitamos de criar a aplicação, decidimos deixar o modelo desta forma, mas este pode ser afinado em trabalhos futuros.

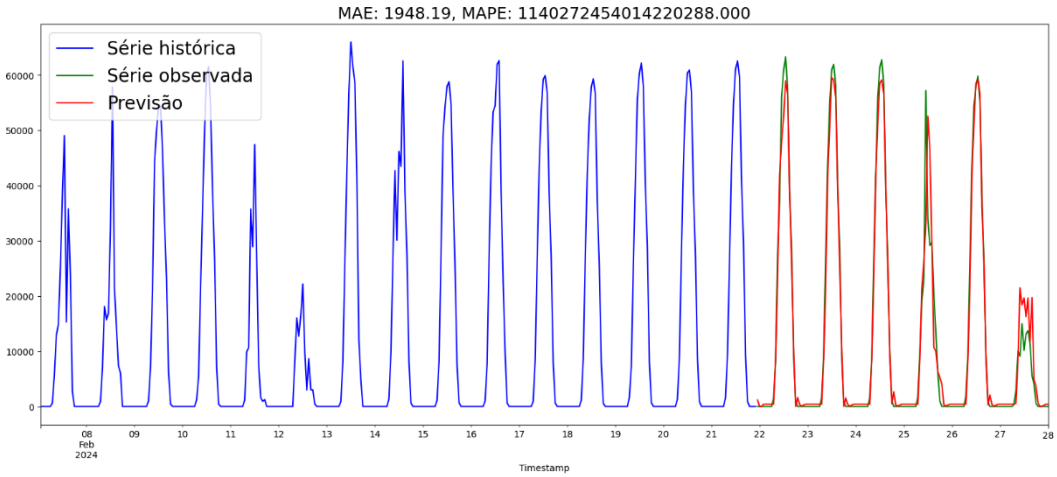

Figura 54 - Modelo preditivo LightGBM para a previsão da produção de energia

7 Casos de Teste

Os casos de teste são os cenários usados para medir a funcionalidade do aplicativo num conjunto de determinadas ações ou condições para verificar os resultados esperados. Em outras palavras, um caso de teste é um conjunto de ações executadas para autenticar a funcionalidade do aplicativo.

7.1 Testes funcionais

Um teste funcional verifica cada função da aplicação, pode ser feito manualmente e é baseado nos requisitos do cliente. Tem como objetivo validar as ações do programa.

Estrutura	Cenário de teste: Sistema de login para perfis de utilizador
T1	**Objetivo:** Verificar se o sistema de login permite acesso diferenciado a funcionalidades com base nos perfis de utilizador: jovem, utilizador e técnico. **Pré-condições:** Existem contas criadas com os diferentes perfis Cada perfil tem a sua funcionalidade especifica associada **Cenário: Login com perfil de Jovem** O jovem acede a página de login Insere as suas credenciais Clica Login O sistema redireciona o jovem para o dashboard com acesso apenas às funcionalidades permitidas para o perfil de jovem **Cenário: Login com perfil de Utilizador** O utilizador acede a página de login Insere as suas credenciais Clica Login O sistema redireciona o utilizador para o dashboard com acesso apenas às funcionalidades permitidas para o perfil de utilizador **Cenário: Login com perfil de Técnico** O técnico acede a página de login Insere as suas credenciais Clica Login O sistema redireciona o técnico para o dashboard com acesso apenas às funcionalidades permitidas para o perfil de técnico

Página Inicial	Cenário de teste: Monitoramento e exibição de dados de energia e emissões
T2	**Objetivo:** Verificar se o sistema fornece aos utilizadores, no *dashboard*, uma visão abrangente e em tempo real dos dados de acordo com o perfil associado. **Pré-condições:** Existem dados disponíveis para exibição na página inicial. Cada perfil tem um *dashboard* personalizado com as funcionalidades especificas associadas. **Cenário: Acesso à Página Inicial por um Jovem:** O jovem acede a página inicial do sistema. O sistema exibe o *dashboard* com as informações referentes ao perfil do jovem. **Cenário: Acesso à Página Inicial por um Utilizador:** O utilizador acede a página inicial do sistema. O sistema exibe o *dashboard* com as informações referentes ao perfil do utilizador. **Cenário: Acesso à Página Inicial por um Técnico:** O técnico acede a página inicial do sistema. O sistema exibe o *dashboard* com as informações referentes ao perfil do técnico.

Previsões	Cenário de teste: Monitoramento de manutenção e economia financeira
T3	**Objetivo:** Verificar se o sistema fornece previsões críticas para técnicos sobre a necessidade de manutenção e detalhes sobre a próxima manutenção, e técnicos e utilizadores sobre a necessidade para limpeza, e a estimativa de economia financeira com base na produção de energia. **Pré-condições:** Existem dados de manutenção/limpeza e produção de energia para análise. As previsões críticas estão configuradas corretamente no sistema. **Cenário: Visualização de Previsões para Técnicos:** O Técnico acede a secção de previsões na aplicação.

	O técnico visualiza as previsões de necessidade de manutenção ou limpeza. O técnico visualiza os detalhes sobre a próxima manutenção. O sistema exibe as previsões com precisão. **Cenário: Visualização de Previsões para Utilizadores:** O Utilizador acede a secção de previsões na aplicação. O utilizador visualiza as previsões de necessidade de limpeza. O sistema exibe as previsões com precisão.

Alertas	Cenário de teste: Sistema de Alertas
T4	**Objetivo:** Verificar se o sistema fornece alertas oportunos para situações críticas, como interrupções do inversor, necessidade de limpeza, baixa produção de energia e o encerramento da central de energia. Além disso, verificar se o sistema de alerta de limpeza é determinado com base nas previsões dos dados e na relação entre a diminuição da produção ao longo do tempo e a sujidade. **Pré-condições:** Os dados de produção de energia, previsões e relação entre a diminuição da produção e a sujidade estão disponíveis no sistema. **Cenário: Alerta de Interrupção do Inversor:** O sistema deteta uma interrupção no inversor. O sistema envia um alerta oportuno para os utilizadores relevantes, informando sobre a interrupção do inversor e fornecendo detalhes sobre a localização e o motivo da interrupção. **Cenário: Alerta de Necessidade de Limpeza:** Com base nas previsões dos dados, o sistema determina que uma limpeza é necessária devido à diminuição da produção de energia ao longo do tempo. O sistema envia um alerta oportuno para os utilizadores relevantes, informando sobre a localização da sujidade **Cenário: Alerta de Baixa Produção de Energia:** O sistema deteta uma baixa produção de energia em comparação aos padrões esperados.

	O sistema envia um alerta oportuno para os utilizadores relevantes, alertando sobre a baixa produção de energia e fornecendo detalhes sobre as possíveis causas e ações corretivas necessárias. **Cenário: Alerta de Encerramento da Central de Energia** O sistema deteta que a central de energia está a ser encerrada. O sistema envia um alerta oportuno para os utilizadores relevantes, informando sobre o encerramento da central de energia.

Vantagem Económica	Cenário de teste: Visualização das perdas financeiras
T5	**Objetivo:** Verificar se o sistema fornece uma visualização clara e compreensível das perdas financeiras decorrentes de manutenção e sujidade nos painéis solares, adaptando a apresentação para técnicos, utilizadores e de maneira criativa para os jovens. **Pré-condições:** Os dados de perdas financeiras decorrentes de manutenção e sujidade nos painéis solares estão disponíveis no sistema. **Cenário: Visualização para Técnicos:** Um técnico acede a secção de visualização de perdas financeiras. O técnico analisa os gráficos e relatórios disponíveis sobre as perdas financeiras decorrentes de manutenção e sujidade nos painéis solares. O sistema apresenta os dados de forma detalhada e técnica, incluindo análises de custos, impacto financeiro ao longo do tempo e recomendações para mitigação de perdas. **Cenário: Visualização para Utilizadores:** Um utilizador acede a secção de visualização de perdas financeiras. O utilizador visualiza os gráficos e relatórios sobre as perdas financeiras decorrentes de manutenção e sujidade nos paneis solares. O sistema apresenta os dados de forma mais simplificada e compreensível para o utilizador, incluindo informações sobre custos associados, impacto financeiro mensal ou anual e sugestões para redução de perdas. **Cenário: Visualização Criativa para Jovens:** Uma jovem acede a secção de visualização de perdas financeiras. O jovem interage com uma apresentação visualmente atrativa e educativa sobre as perdas financeiras decorrentes de manutenção e sujidade nos painéis solares.

	O sistema utiliza elementos gráficos, animações, etc, para transmitir de forma criativa o conceito de perdas financeiras aos jovens, de maneira que seja divertida e educativa.

Educação	Cenário de teste: Secção Educacional Interativa
T6	**Objetivo:** Verificar se o sistema possui uma secção dedicada a fornecer informações educativas sobre o consumo e a produção de energia de forma divertida e dinâmica, sem se concentrar apenas em números e dados. **Pré-condições:** A secção educativa interativa está disponível e acessível aos utilizadores **Cenário: Acesso à Secção Educacional:** Um utilizador acede a secção educativa interativa no sistema. O sistema redireciona o utilizador para a secção educacional, onde informações sobre consumo e produção de energia são apresentadas de forma atraente e interativa. **Cenário: Conteúdo Divertido e Dinâmico:** O utilizador interage com o conteúdo dinâmico na secção educativa, conteúdos interativos ou outras formas de conteúdo O Sistema apresenta o conteúdo de maneira dinâmica e educativa, transmitindo conceitos sobre consumo e produção de energia. **Cenário: Envolvimento do Utilizador:** O utilizador é incentivado a participar de atividades interativas. O utilizador interage com as atividades interativas. O sistema fornece feedback imediato e incentiva a participação do utilizador, tornando a aprendizagem sobre consumo e produção de energia uma experiência envolvente e motivadora.

7.2 Testes não funcionais

Um teste não funcional verifica aspetos não funcionais da aplicação, como o desempenho, usabilidade, confiabilidade, etc. São baseados nas expectativas do cliente e tem como objetivo validar o desempenho do programa.

Adaptabilidade	Cenário de teste: Adaptabilidade a Diferentes Tipos de Display
T7	**Objetivo:** Verificar se o sistema é projetado e implementado de forma a garantir uma experiência consistente e eficiente em uma variedade de dispositivos, como smartphone, tablets ou computadores. **Cenário: Acesso via Dispositivo Móvel:** Um utilizador acede o sistema utilizando um dispositivo móvel, como um smartphone ou tablet. O utilizador navega pelas diferentes secções e funcionalidades do sistema. Os elementos da interface do utilizador se adaptam ao tamanho e orientação do dispositivo móvel, garantindo uma experiência de usuário otimizada e sem comprometer a usabilidade ou funcionalidade do sistema. **Cenário: Acesso via Computador Desktop:** Um utilizador acede o sistema utilizando um computador desktop ou laptop. O utilizador explora as diferentes partes e funcionalidades do sistema. Os elementos da interface do utilizador se ajustam ao tamanho da tela do computador e são dispostos de forma a proporcionar uma experiência de usuário confortável e eficiente, independentemente da resolução da tela ou do navegador utilizado.

Usabilidade	Cenário de teste: Usabilidade da interface do utilizador
T8	**Objetivo:** Verificar se a interface do sistema é projetada de forma simples e intuitiva, facilitando o uso para os utilizadores de diversos perfis e habilidades. **Cenário: Navegação na Interface:** Um utilizador acede a interface do sistema. O utilizador navega pelas diferentes seções e funcionalidades da interface. O layout da interface é claro e organizado, facilitando a localização e acesso às diferentes funcionalidades do sistema. Os elementos de navegação, como menus e botões, são facilmente identificáveis e intuitivos de usar. **Cenário: Realização de Tarefas Básicas:** Um utilizador realiza tarefas básicas, como fazer login, visualizar informações e realizar ações simples.

	As tarefas básicas são concluídas de forma rápida e sem dificuldades, indicando que a interface é intuitiva e fácil de usar para utilizadores com diferentes níveis de habilidade e experiência. **Cenário: Feedback e Orientações:** Um utilizador realiza uma ação na interface do sistema. O sistema fornece feedback imediato sobre a ação realizada e orientações claras sobre os próximos passos. O feedback fornecido pelo sistema é claro e informativo, ajudando os usuários a entenderem o resultado de suas ações e a navegarem pela interface de forma eficiente. **Cenário: Acessibilidade:** Um utilizador com necessidades especiais acede a interface do sistema. Um utiliza ferramentas de acessibilidade, como leitores de tela ou teclas de atalho. A interface do sistema é acessível e suporta ferramentas de acessibilidade, permitindo que utilizadores com diferentes necessidades possam utilizar o sistema de forma eficaz.
Disponibilidade	Cenário de teste: Disponibilidade ininterrupta
T9	**Objetivo:** Verificar se o sistema garante uma disponibilidade contínua, 24 horas por dia e 7 dias por semana, para atender às necessidades dos usuários a qualquer hora. **Cenário: Acesso ao Sistema Durante o Horário de Pico:** Durante um horário de pico, um grande número de utilizadores acede simultaneamente ao sistema. O sistema responde a todas as solicitações dos utilizadores sem falhas significativas, garantindo que todos possam aceder e utilizar o sistema conforme necessário, mesmo durante períodos de alta demanda. **Cenário: Deteção e Resposta a Incidentes:** Um incidente inesperado causa uma interrupção no funcionamento do sistema. A equipe de operações do sistema é alertada sobre o incidente e toma medidas imediatas para resolver o problema. O tempo de inatividade é minimizado e o sistema é restaurado o mais rápido possível, garantindo uma disponibilidade contínua para os utilizadores.

Desempenho	Cenário de teste: Desempenho e frequência de atualização dos dados
T10	**Objetivo:** Verificar se o sistema garante um desempenho eficiente e uma atualização periódica dos dados, com frequência predefinida, para proporcionar informações em tempo real. **Cenário: Atualização dos Dados em Tempo Real:** O sistema é configurado para atualizar os dados com uma frequência em tempo real O sistema atualiza os dados de acordo com a frequência Os dados são atualizados no sistema e garante que as informações sejam sempre atualizadas. **Cenário: Desempenho Eficiente durante Picos de Atividade:** Durante o pico de atividade, um grande número de utilizadores acede ao sistema simultaneamente. Os utilizadores realizam operações que exigem processamento de dados. O sistema mantém um desempenho eficiente, processando rapidamente as solicitações dos utilizadores.

Compatibilidade	Cenário de teste: Compatibilidade com sistemas fotovoltaicos e inversores
T11	**Objetivo:** Verificar se o sistema é projetado e implementado de maneira a ser compatível com os sistemas fotovoltaicos e com os inversores. **Cenário: Compatibilidade com Sistemas Fotovoltaicos** O sistema é compatível com o sistema fotovoltaico instalado. Os dados de produção de energia gerados pelo sistema são transmitidos pelo sistema. O sistema é capaz de receber e processar de produção de energia, sem qualquer tipo de problema. **Cenário: Compatibilidade com Inversores** O sistema é compatível para se comunicar com os inversores do sistema fotovoltaico Os dados relacionados ao desempenho dos inversores são transmitidos pelo sistema. O sistema é capaz de se comunicar bem com os inversores, sem qualquer tipo de problema.

8 Método, planeamento e Resultados

Analisando as especificações do nosso projeto e o certo nível de criatividade que poderemos vir a ter decidimos seguir o Modelo Incremental no desenvolvimento do Trabalho Final de Curso para termos a liberdade de fazer ajustes caso necessário bem como receber *feedback* frequentemente ao decorrer do tempo.

Apresentamos seguidamente as tarefas planeadas ao longo do trabalho bem como os entregáveis definidos pelo Conselho Académico:

- Tarefas
  - **T1** – Identificação do problema
  - **T2** – Viabilidade e pertinência
  - **T3** – Benchmarking
  - **T4** – Análise e replicação dos gráficos de Excel
  - **T5** – Aplicação do modelo preditivo
  - **T6** – Levantamento e análise de requisitos
  - **T7** – Diagramas de Casos de Uso ou descrição de cenários de aplicação
  - **T8** – Desenvolvimento da aplicação
  - **T9** – Testes e validação
- Entregáveis
  - **R1** – Relatório intercalar do 1º Semestre
  - **R2** – Relatório intermédio
  - **R3** – Relatório intercalar do 2º Semestre
  - **S1** - Versão funcional da solução desenvolvida
  - **R4** – Relatório final
  - **S2** – Versão operacional

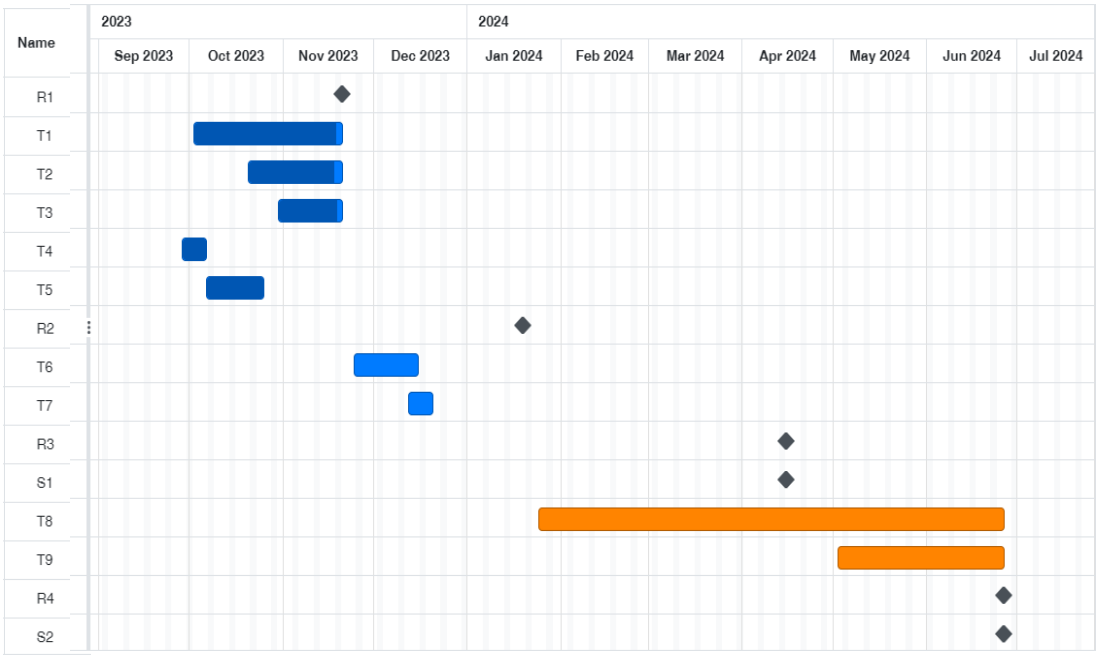

Figura 55 – Calendarização das tarefas propostas

Durante a realização deste trabalho, encontrámos bastantes adversidades tanto a nível pessoal quanto a nível do seguimento do projeto.

Analisando o calendário proposto para a elaboração do projeto, conseguimos ver que houve bastantes progressos nomeadamente ao que toca ao desenvolvimento do modelo preditivo, ao qual foram atingidos resultados excelentes, mas que ficámos aquém das expectativas na questão do desenvolvimento da aplicação e consequentes testes, já que inicialmente achávamos que os dados de produção seriam de fácil acesso, o que não foi o caso e obrigou a que o projeto tivesse uma componente de engenharia de dados que não estava prevista na elaboração dos requisitos e planeamentos iniciais – as soluções que encontrámos (como o Selenium) também se mostraram ineficazes para a resolução deste problema o que acabou por estender ainda mais esta componente; configuração geral de containers com especial atenção à configuração do Docker para o LightGBM que foi necessário porque esta ferramenta não corria no Windows o que acabou por levar mais tempo que o expectável; problemas de configuração e estudo do funcionamento da API do GoSungrow, proveniente de um repositório do GitHub que acabámos por encontrar.

O atraso do desenvolvimento da aplicação também se deveu a problemas a nível pessoal e de saúde de ambos os elementos do grupo, que impediu que prosseguíssemos o trabalho nos moldes temporais que tínhamos previsto, e que causou uma maior sobrecarga num curto espaço de tempo o que acabou por baixar a produtividade de ambos.

Quanto ao desenvolvimento efetivo do projeto, tínhamos inicialmente planeado uma plataforma com *dashboard*, sistema de alertas, gestão de utilizadores, e previsão da produção, que acabámos por perceber que eram demasiados requisitos para encaixar no espaço de tempo que tínhamos, e que acabou por ficar ainda mais curto pelos motivos mencionados acima, pelo que decidimos usar o modelo MVP (Minimum Viable Product), ou seja, tivemos que priorizar tarefas, reorganizar o nosso modo de trabalho e focarmo-nos mais em produzir bons resultados no que considerámos essencial como: o modelo preditivo, a elaboração do *back-end* da plataforma, com todos os *endpoints* que achámos necessários relativamente a todos os requisitos, e elaboração de uma parte do *front-end* nomeadamente o dashboard, uma página de login, e uma página para gerenciamento de utilizadores em que se pode ver a lista total dos mesmos e fazer o registo na base de dados. Esta priorização revelou ser uma boa aposta porque nos conseguimos aprofundar um pouco mais nas tecnologias que nos propusemos a aprender, e conseguimos avançar bastante nestas componentes, deixando sempre espaço para possíveis melhorias futuras em outros trabalhos.

9 Conclusão e trabalhos futuros

9.1 Conclusão

O nosso projeto teve pontos que nos deixaram entusiasmados para a sua realização nomeadamente a temática ambiental e sustentável, o aprendizado de outras linguagens de programação, tecnologias e aprofundamento de conhecimentos em Data Science e modelos de previsão como o LightGBM e o ARIMA.

Para o desenvolvimento deste projeto retirámos bastantes conhecimentos prévios das seguintes Unidades Curriculares:

- Programação Web, foi utilizado como base para aprofundarmos os nossos conhecimentos no desenvolvimento de plataformas web;
- Engenharia de Software, útil para definir requisitos da aplicação;
- Data Science, como base para o estudo de dados, e a elaboração de modelos preditivos;

Apesar de não termos atingido todos os objetivos iniciais, consideramos que todo o processo de aprendizagem enriquece este trabalho e abre portas a serem feitas melhorias em vários pontos.

9.2 Trabalhos futuros

Em relação a trabalhos futuros, existem várias frentes que podem ser melhoradas e funcionalidades novas a implementar, como por exemplo:

- **Modelo preditivo**: o modelo pode ser afinado acrescentando novas *features* e parâmetros para analisar os dados, também podem ser acrescentadas informações como inferir quando houve uma limpeza, e outros fatores externos;
- **Back-end:** pode ser aprofundado a implementação do *back-end* dando especial atenção à *feature* de alertas;
- **Front-end:** podem ser acrescentados mais informações no *back-end* nomeadamente o *dashboard* não só para o utilizador comum, mas para jovens e a páginas de alertas;

Bibliografia

[ULHT21] Universidade Lusófona de Humanidades e Tecnologia, www.ulusofona.pt, acedido em Out. 2021.

[Efei23] Efeito Solar, https://efeitosolar.com/2020/03/09/entenda-o-sistema-solar-fotovoltaico-2/, acedido em Out. 2023.

[SOFC23] Jornal de Negócios, https://www.jornaldenegocios.pt/sustentabilidade/governacao/detalhe/mercado-de-software-de-gestao-de-carbono-vai-crescer-cerca-de-20-ao-ano, acedido em Nov. 2023

[DCAM23] Campus Datacamp, https://campus.datacamp.com/courses/visualizing-time-series-data-in-python/seasonality-trend-and-noise?ex=6 , acedido em Nov. 2023

[SMA23] SMA,https://www.sma-sunny.com/en/ennexos-how-does-smas-energy-management-platform-work/, acedido em Nov. 2023.

[ISOL23] iSolarCloud, http://base.isolarcloud.com:8181/docs/a1-0/d3.md, acedido em Nov. 2023.

[Tabl23] Tableu, https://www.tableau.com/learn/articles/time-series-analysis, acedido em Nov. 2023.

[Med23] Medium,https://medium.com/data-hackers/series-temporais-parte-1-a0e75a512e72, acedido em Nov. 2023

[GIT23] Github, https://github.com/microsoft/LightGBM/blob/master/docs/Features.rst, acedido em Nov. 2023

[TEXa23] oTexts, https://otexts.com/fpp3/prophet.html, acedido em Nov. 2023

[TEXb23] oTexts, https://otexts.com/fpp3/arima.html, acedido em Nov. 2023

[SELE23] Selenium , https://www.selenium.dev/documentation/, acedido em Nov. 2023

[DOCK24] Docker, https://www.docker.com/ , acedido em Jan. 2024

[MGDB24] MongoDB , https://www.mongodb.com/en-us , acedido em Jan. 2024

[ONU24] ONU, https://unric.org/pt/objetivos-de-desenvolvimento-sustentavel/, acedido em Jan. 2024

[ODS24] ODS7, https://unric.org/pt/objetivo-7-energias-renovaveis-e-acessiveis/, acedido em Jan. 2024

[IBE24] Iberdrola, https://www.iberdrola.com/sustentabilidade/comprometidos-objetivos-desenvolvimento-sustentavel/ods-7-energias-renovaveis-e-acessiveis, acedido em Jan. 2024

[IMP24] ImpactNinja, https://impactful.ninja/energy-sources-with-the-lowest-carbon-footprint/, acedido em Jan. 2024

[MERN24] MongoDB , https://www.mongodb.com/mern-stack, acedido em Jan. 2024

[RCJS24] React, https://legacy.reactjs.org/, acedido em Jan. 2024

[NODE24] NodeJs, https://nodejs.org/en, acedido em Jan. 2024

[EXJS24] ExpressJS, https://expressjs.com/, acedido em Jan. 2024

[FIGM24] Figma, https://www.figma.com/ui-design-tool/, acedido em Jan. 2024

[TRAC24] Trackingsdg7, https://trackingsdg7.esmap.org/downloads, acedido em Jan. 2024

[NEXT24] NextJs, https://nextjs.org/, acedido em Abril. 2024

[GOS24] GoSungrow, https://github.com/MickMake/GoSungrow, acedido em Abril. 2024

[SKTA24] Sktime, https://www.sktime.net/en/v0.19.0/api_reference/auto_generated/sktime.forecasting.arima.AutoARIMA.html, acedido em Abril 2024

[OPWE24] OpenWeatherMap https://openweathermap.org/api , acedido em Junho 2024

Glossário

LEI Licenciatura em Engenharia Informática

LIG Licenciatura em Informática de Gestão

TFC Trabalho Final de Curso

ARIMA Autoregressive Integrated Moving Average

LightGBM Light Gradient Boosting Model

MAE Mean Absolute Error

MAPE Mean Absolute Percentage Error

ONU Organização das Nações Unidas

ODS Objetivos de Desenvolvimento Sustentável

MERN MongoDB, Express, React, Node

CORS Cross-Origin Resource Sharing