Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Diogo Miguel Melo de Almeida

Desenvolvimento de um datalogger de baixo custo para aquisição de dados de equipamentos em parques solares fotovoltaicos

Maio de 2022

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Diogo Miguel Melo de Almeida

Desenvolvimento de um datalogger de baixo custo para aquisição de dados de equipamentos em parques solares fotovoltaicos

Tese de Mestrado
Engenharia Eletrotécnica - Energia e Automação Industrial

Professor Doutor Paulo Moisés Almeida da Costa
Professor Doutor Paulo Rogério Perfeito Tomé

Monitor do trabalho:
Engenheiro Augusto José Chaves Almeida Casais

Maio de 2022

RESUMO

A monitorização de parques solares fotovoltaicos é, em particular nas instalações de maior potência instalada, uma atividade crucial para assegurar o adequado desempenho técnico e económico destas unidades de produção de eletricidade.

A referida monitorização implica a aquisição de dados relativos ao funcionamento das centrais, em termos de grandezas elétricas e não elétricas (meteorológicas). Estes dados possibilitam o cálculo de indicadores de desempenho (por exemplo energia efetivamente produzida face ao valor expectável) e a deteção de situações de falha. A aquisição dos dados necessários à monitorização contínua de uma central impõe a comunicação com diversos equipamentos instalados no parque fotovoltaico.

O presente trabalho, realizado em parceria com a empresa Voltalia Portugal S.A., tem por objetivo fundamental o desenvolvimento de um sistema de aquisição e registo de dados (*"datalogger"*) provenientes de equipamentos instalados em parques solares fotovoltaicos, nomeadamente inversores, sensores (temperatura, irradiância, tensão e corrente), autómatos e até mesmo servidores. O sistema desenvolvido destina-se, principalmente, a ser utilizado em parques solares mais antigos, cujos equipamentos não possuem acesso a softwares com as funcionalidades pretendidas. O desenvolvimento deste projeto teve como base partes de *dataloggers* anteriormente desenvolvidos pela Voltalia, em particular no que se refere a *drivers* de comunicação com dispositivos instalados nas centrais fotovoltaicas. Estes componentes de programação já existentes foram melhorados e adaptados para a utilização no *datalogger* desenvolvido neste trabalho. Adicionalmente, foi desenvolvido todo o sistema de aquisição, tratamento e armazenamento de dados referentes a grandezas elétricas (potências, tensão elétrica, corrente elétrica, frequência, energia, nível de carga em baterias, etc.) e não elétricas (temperaturas, irradiância, pressão atmosférica, velocidade e direção do vento, humidade relativa, etc.). Além disso, foi também implementada uma interface gráfica que permite visualizar as grandezas relevantes, para além da definição de utilizadores e respetiva atribuição de permissões relacionadas com o uso do sistema. O sistema desenvolvido irá funcionar num *Raspberry Pi*, que por sua vez terá de efetuar uma comunicação com um sistema central de modo a recolher os dados necessários dos equipamentos, esta comunicação será possível através do uso de um serviço de comunicação desenvolvido (aplicação de comunicação) pela própria Voltalia.

ABSTRACT

The monitoring of photovoltaic plants is an important activity to ensure the adequate technical and economic performance of these electricity production units, especially in higher power facilities.

The mentioned monitoring implies the acquisition of data relative to the functioning of the plants in terms of electrical and non-electrical (meteorological) quantities. This data enables the calculation of performances indicators (for example, the energy produced versus its expected value) and the detection of fault situations. The continuous acquisition of photovoltaic plant data requires communication with multiple types of equipment installed on those power plants.

The present work, developed in partnership with Voltalia Portugal S.A., aims to create acquisition and data-logging system from equipment installed in photovoltaic plants, including inverters and sensors (temperature, irradiation, tension and current), automatons and even servers. This project is aimed mainly at older photovoltaic plants whose equipment doesn't possess dedicated software with the required functionalities.

The development of this project used parts of software previously developed by Voltalia, particularly concerning communication drivers with devices installed in photovoltaic plants. These existing programming components were improved and adapted for use in the *datalogger* developed. The system for acquiring, processing, and storing electrical (power, electrical voltage, electrical current, frequency, energy, battery charge level, etc.) and non-electrical (temperatures, irradiance, pressure atmosphere, wind speed, relative humidity, etc.) data was developed. In addition, a graphical interface was implemented, allowing the visualization of the temporal evolution of relevant quantities, the management of alarms, the definition of users and the respective attribution of permissions related to the use of the system. The developed system will run on a Raspberry Pi, which in turn will perform a communication with a central system to collect the necessary data from the equipment, this communication will be possible with the use of a communication service (communication application) developed by Voltalia themselves.

PALAVRAS-CHAVE

Monitorização
Raspberry
Fotovoltaico
Centrais fotovoltaicas
Solar
Base de dados
Interface WEB

KEY WORDS

Monitoring
Raspberry
Photovoltaic
Photovoltaic Plants
Solar
Data Base
WEB Interface

AGRADECIMENTOS

Gostaria de agradecer aos meus orientadores, Professor Doutor Paulo Rogério Perfeito Tomé e Professor Doutor Paulo Moisés Almeida da Costa, pela disponibilidade e orientação prestada ao longo da realização do trabalho de mestrado.

Ao Engenheiro Augusto José Chaves Almeida Casais pelo apoio e orientação prestada.

À Voltalia Portugal S.A. pela oportunidade de poder realizar este projeto

Aos meus amigos pelo apoio que me deram ao longo deste percurso

À minha família, pelo apoio e pelo esforço que fizeram para que eu pudesse alcançar esta meta.

ÍNDICE GERAL

Índice

ÍNDICE DE FIGURAS

1. Introdução

O presente capítulo, dividido em três secções, apresenta o enquadramento do trabalho realizado, enfatizando as vantagens da monitorização de parques solares fotovoltaicos, apresentando os principais objetivos inerentes à elaboração deste projeto e descrevendo a forma como este trabalho se encontra organizado.

1.1 Enquadramento

Uma central fotovoltaica tem como objetivo converter energia solar em eletricidade através do efeito fotovoltaico, o qual consiste em produção de energia elétrica num determinado material (célula fotovoltaica) quando este é exposto à radiação solar.

Estas centrais, tal como se evidencia na Figura 1, são constituídas por equipamentos diversos, incluindo: painéis solares fotovoltaicos, inversores, equipamentos de proteção (disjuntores, fusíveis, descarregadores de sobretensão, interruptores …), condutores elétricos, transformadores de potência e equipamentos de medição (potências, tensão, corrente, temperatura, irradiância, etc.) [1].

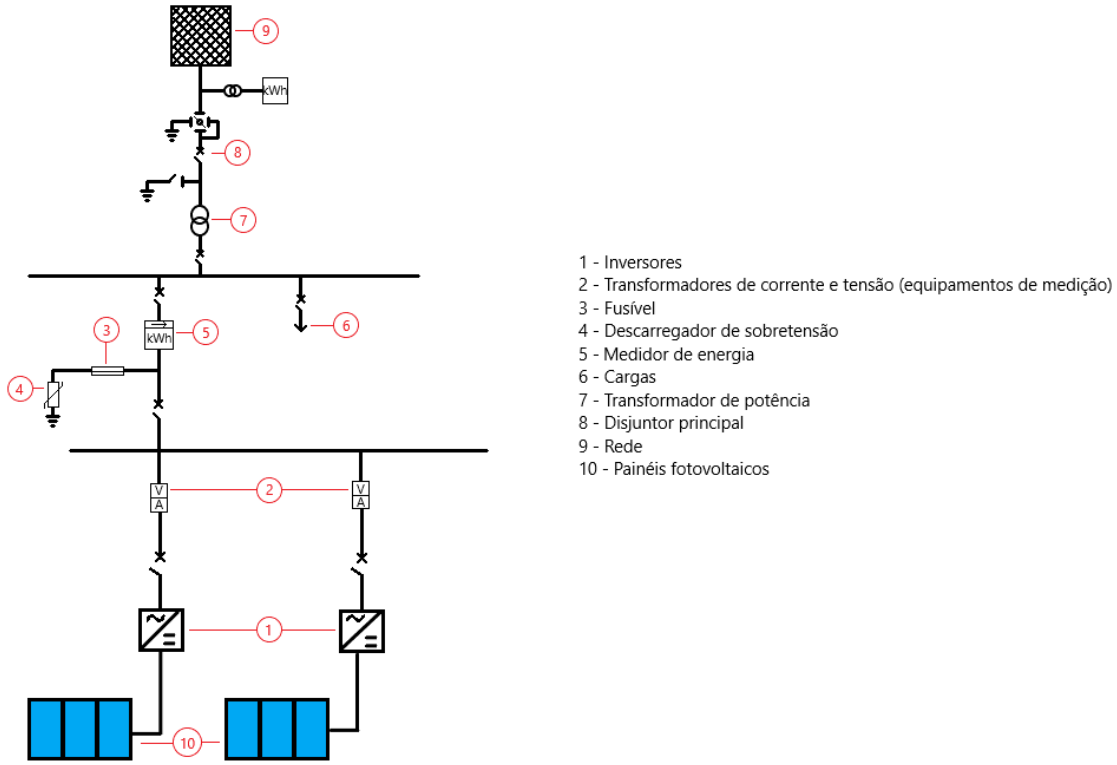


Figura 1 - Constituição genérica de uma central fotovoltaica de grande dimensão

A monotorização das centrais fotovoltaicas é uma tarefa importante uma vez que permite perceber a forma como estas se encontram a ser exploradas (através de indicadores de desempenho), bem como detetar, de forma mais rápida, eventuais situações de falha ou de mau funcionamento parcial. Neste contexto importa realçar que estas centrais podem estar situadas em locais remotos, cujo acesso nem sempre é fácil, o que reforça a importância de se monitorizar à distância o seu funcionamento, contribuindo desta forma para uma melhor gestão das equipas técnicas (manutenção, otimização, etc.) que possam ser enviadas para os locais de implantação das centrais.

A supervisão do desempenho de uma central fotovoltaica implica a aquisição de informação relativa a um conjunto de parâmetros elétricos e não elétricos relacionados com o seu funcionamento, incluindo: potências elétricas, tensão e corrente elétrica, temperatura ambiente, temperatura de células fotovoltaicas, irradiância solar, etc. Com base na informação relativa a estas grandezas é possível comparar, em cada momento, a produção de uma central com aquela que seria esperada, tendo em consideração as condições que se encontrem nesse mesmo momento (temperatura, radiação, etc.). Esta comparação permite inferir desvios injustificados da produção efetiva face ao valor esperado e, desta forma, apoiar a definição de medidas de manutenção preventiva e corretiva. Note-se que os desvios referidos podem ser originados por situações de falha em equipamentos da central fotovoltaica (avaria de inversores,

avaria de painéis fotovoltaicos, etc.), atuação de proteções elétricas, acumulação de sujidade nos painéis fotovoltaicos, possíveis sombreamentos que possam surgir durante a vida útil da central, etc.

Várias alternativas de sistemas destinados à monotorização de centrais de produção fotovoltaica estão atualmente disponíveis no mercado. A Voltalia utiliza, na sua atividade, algumas soluções comerciais, nomeadamente: *Solar-Log*, o *Meteocontrol*, o *Webdyn*, o *SMA Sunny WebBox*, entretanto descontinuado e substituído pelo *SMA Cluster Controller*, o *SmartLogger* da *Huawei*, o *Logger1000* da *Sungrow* e o *Higeco*.

Várias das alternativas referidas no paragrafo anterior preenchem muitos dos requisitos da Voltalia para a monitorização dos parques solares fotovoltaicos, sendo utilizadas em alguns parques. Contudo, estas alternativas apresentam algumas desvantagens importantes, nomeadamente:

i) os custos a suportar pela utilização das soluções existentes, o qual tende a ser elevado;

ii) as dificuldades relacionadas com o facto de algumas das soluções existentes serem limitados no que diz respeito à flexibilidade, pois nem sempre possuem *drivers* compatíveis com alguns dispositivos que integrados em centrais fotovoltaicas;

iii) a dependência de terceiros para a resolução de possíveis problemas que possam ocorrer ao longo da vida útil da central fotovoltaica monitorizada.

Em consequência das dificuldades enumeradas nos pontos anteriores, a Voltalia pretende desenvolver uma solução orientada às suas necessidades. Em particular a Voltalia pretende uma solução em que seja detentora do código fonte que possibilite realização de alterações que se revelem necessárias ao longo do tempo (por exemplo acrescentar novos equipamentos a uma central em funcionamento). É neste contexto que se insere o presente trabalho.

1.2 Motivação e objetivos do trabalho

Como foi referido na secção anterior, a monitorização de parques solares fotovoltaicos reveste-se de particular importância no contexto da sua exploração, existindo diversos sistemas atualmente disponíveis no mercado para a realização dessa monitorização.

Contudo, as ferramentas disponíveis apresentam, frequentemente, elevados custos associados à sua utilização. Por outro lado, essas ferramentas apresentam algumas limitações relacionadas com a flexibilidade para utilização com equipamentos de múltiplos fabricantes e, em particular, com equipamentos instalados em centrais fotovoltaicas mais antigas. Estes equipamentos são, frequentemente, dotados de protocolos de comunicação antigos e proprietários dos fabricantes, as quais dificultam a comunicação com os sistemas de aquisição e registo de dados e com os respetivos softwares. Uma vez que o acesso ao código base destes *softwares* não está, geralmente, disponível, a sua adaptação, assim como permitir desenvolver e implementar esses protocolos de modo a efetuar a comunicação com os dispositivos mais antigos torna-se complexa, dificultando a integração e o uso destes nos sistemas de monitorização.

O presente trabalho tem por objetivo fundamental o desenvolvimento de um *datalogger* de baixo custo para aquisição de dados de equipamentos em parques solares fotovoltaicos (cujo esquema de funcionamento pode ser visto na Figura 2). Este terá também de permitir a apresentação dos dados em tempo real e permitir consultar o histórico de dados, utilizando gráficos ou tabelas adequadas para o efeito. Outra função do sistema consiste na gestão de situações de alarme, possibilitando a respetiva identificação, o registo do momento em que ocorreram e a perceção do estado destes alarmes (se ainda se encontram ativos ou não).

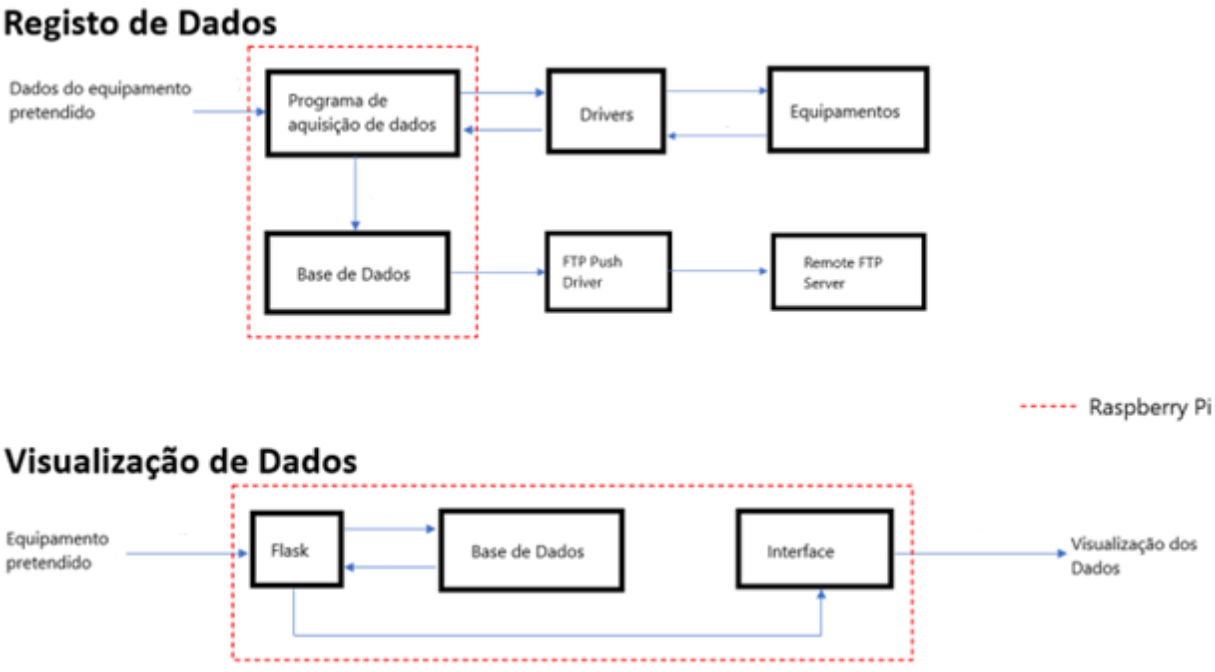


Figura 2 - Esquema de Funcionamento

A aplicação informática deve apresentar uma interface intuitiva e de fácil utilização e comunicar com os dispositivos de uma, ou mais, centrais fotovoltaicas, utilizando a aplicação de comunicação fornecida pela Voltalia S.A. A aplicação será instalada num dispositivo *Raspberry Pi* dotado de um cartão de memória com um máximo de 64GB de armazenamento. Estas caraterísticas colocam algumas limitações sobre o volume de dados que poderão ser armazenados em memória local.

1.3 Organização do Relatório

O presente relatório encontra-se dividida em 5 capítulos, incluindo este e o último dedicado às conclusões e trabalhos futuros.

No Capítulo 2 apresenta-se o estado de arte, oferecendo-nos uma revisão dos sistemas de monitorização existentes atualmente no mercado, bem como o que é utilizado pela Voltalia na monitorização de alguns parques solares fotovoltaicos.

O Capítulo 3 centra-se na discussão dos requisitos estabelecidos pela Voltalia, expondo os principais objetivos a serem cumpridos, a estrutura do sistema que se pretende desenvolver e apresenta-se a metodologia utilizada para a conceção do sistema.

O Capítulo 4 apresenta a implementação do sistema, expondo o funcionamento dos diferentes componentes que tornam possível a recolha e apresentação dos dados. Efetuam-se também testes de modo a apresentar o funcionamento do sistema.

O Capítulo 5 apresenta as conclusões do trabalho, o que feito, bem como o que pode ser realizado no futuro.

2. Monitorização de centrais fotovoltaicas

Nos últimos anos, os sistemas de monotorização assumiram-se como essenciais no âmbito da operação e manutenção de centrais fotovoltaicas, contribuindo para um funcionamento estável e fiável destas instalações. Estes sistemas recolhem dados sobre um conjunto de parâmetros relacionados com a operação de uma central fotovoltaica. Essa informação permite supervisionar o desempenho da central, quer em termos históricos, quer em tempo real, incluindo a gestão de situações de alarme relacionadas, por exemplo, com falhas no funcionamento de componentes que integram a central [2].

2.1 Sistemas de monitorização na literatura

Na literatura podem ser encontradas múltiplas propostas de implementação de sistemas de monitorização para instalações fotovoltaicas. Essas propostas diferem no domínio de aplicabilidade (sistemas fotovoltaicos de maior ou menor potência, monofásicos ou trifásicos, monitorização individual de painéis ou monitorização mais global da central, etc.).

Algumas publicações fazem uma revisão das tecnologias, técnicas e procedimentos que têm vindo a ser adotadas para monitorização de centrais fotovoltaicas [3][4][5]. Em [3], os autores fazem uma revisão das principais técnicas de monitorização, comparando o respetivo desempenho. Em particular, os autores avaliam aspetos relacionados com o tipo e forma de funcionamento dos sensores utilizados para a monitorização, os tipos de controladores utilizados para a aquisição de dados, os meios de transmissão de informação e as formas de armazenamento e tratamento desta. Rahman et al [4], fazem também uma revisão de várias técnicas e procedimentos que podem ser utilizadas na monitorização de parques solares fotovoltaicos. A análise de dados monitorizados baseia-se em análise de arquiteturas e

características de vários sistemas de monitorização desenvolvidos em contexto de investigação e de utilização comercial. Os autores mostram que todos os sistemas discutidos no artigo, desenvolvidos de acordo com requisitos diferenciados, apresentam vantagens e desvantagens relativas. No final, os autores propõem um sistema de monitorização baseado num microcontrolador Arduíno. Indicam que as vantagens de tal sistema residem no facto do Arduíno oferecer um ambiente de programação de fácil compreensão, para além de ser um microcontrolador que permite uma fácil integração de diferentes tipos de sensores e módulos diversos. Os autores ressalvam, contudo, que a plataforma proposta não deve ser considerada para substituir plataformas de monitorização baseadas em dados de satélite, em particular porque os protocolos de comunicação podem não estar disponíveis. Também em [5] se efetua uma revisão do progresso verificado nas tecnologias e procedimentos aplicados à monitorização de sistemas fotovoltaicos, com foco nas formas de processamento de informação e nos protocolos de transmissão de dados.

Os autores de [6] propõem um sistema de baixo custo equipado com diversos sensores (tensão, corrente, irradiância, temperatura, etc.) para monitorização de sistemas fotovoltaicos com o objetivo principal de determinar causas de perdas de eficiência. O sistema utiliza sensores em cada painel fotovoltaico e uma rede de comunicação sem fios, sendo os dados recolhidos transferidos para um centro de controlo no qual se determinam as eficiências e as eventuais causas de um pior desempenho bem como se detetam situações de falha.

Em [7] é proposto um sistema de monitorização baseado num microcontrolador *PIC16F877a* e em *LabVIEW*. O sistema é capaz de simular e monitorizar a tensão e corrente do gerador fotovoltaico bem como informação relativa a gradezas como a temperatura ambiente e a radiação solar. Várias outras propostas de sistemas de monitorização baseadas na utilização do *LabVIEW* têm vindo a ser publicadas, como, por exemplo, a apresentada pelos autores G. Bayrak e M. Cebeci [8], que apresentam uma aplicação num sistema fotovoltaico de potência reduzida (potência instalada de 1,2 kWp). Na implementação utilizam um sistema de aquisição de dados (*DAQ*) baseado numa carta NI USB-6221 para adquirirem informação sobre corrente, tensão e potência produzidas pelo sistema fotovoltaico. A aplicação de monitorização foi desenvolvida com recurso ao *software LabVIEW*. O *LabVIEW* é também utilizado pelos autores de [5] para suportar o desenvolvimento de um sistema de monitorização que, nas palavras dos autores, se caracteriza por ter baixo custo, ser simples, altamente fiável e de elevada precisão. Na implementação do sistema é utilizado um computador (*laptop*) ligado via porta série a um microcontrolador de baixo custo. Os autores desenvolveram uma placa que permite a interface dos vários sensores utilizados com o microcontrolador bem como uma interface gráfica (desenvolvida em *LabVIEW*) para possibilitar a monitorização dos dados recolhidos. A aquisição dos dados dos sensores é efetuada através de uma aplicação escrita em *assembly*, o qual permite o posterior envio para o computador para monitorização e armazenamento.

Em [9] e [10] apresenta-se uma arquitetura de um sistema de informação de suporte ao funcionamento de sistema FV. A arquitetura proposta baseia-se em dois pilares fundamentais no desenvolvimento de sistemas: flexibilidade e segurança. A prova de conceito foi implementada através do desenvolvimento de três instanciações de cada um dos elementos contemplados na arquitetura. O sistema desenvolvido é capaz de disponibilizar informação relativa ao funcionamento de uma central FV, independentemente da sua dimensão, num dispositivo móvel.

Um sistema de monitorização de sistemas fotovoltaicos de pequena dimensão é apresentado em [11] e [12]. Este sistema é caracterizado pelo seu elevado nível de integração, baixo custo e facilidade de instalação na maioria dos sistemas fotovoltaicos de pequena potência (alguns kW). O sistema permite obter, armazenar, tratar e visualizar parâmetros elétricos e meteorológicos que são essenciais à monitorização de geradores fotovoltaicos. A identificação de situações de falha e a elaboração de análises de desempenho são outras características relevantes do sistema proposto. O acesso à informação das instalações monitorizadas pode ser efetuado de forma remota, utilizando uma aplicação web desenvolvida para dispositivos móveis. A integração do sistema de monitorização num sistema centralizado de supervisão é também possível.

2.2 Sistemas de monitorização comerciais

Para além das propostas constantes da literatura científica, existem várias propostas de sistemas de monitorização de utilização comercial, nomeadamente:

2.2.1 Solar-Log

O *Solar-Log*, [13], ilustrado na Figura 3, é um conjunto de *software* e *hardware* para monotorização que permite uma integração com uma grande variedade de marcas de inversores, baterias, sensores meteorológicos e analisadores de energia. Nesta solução é possível observar as potências, o rendimento energético, a temperatura ambiente bem como a meteorologia.

Os equipamentos de registo de dados mais recentes desta marca possuem um sistema de *GPRS* juntamente com uma porta *Ethernet*. Para além disso, possuem também uma série de ligações para outros dispositivos, nomeadamente: portas *USB* para atualização do *firmware* e para realização de *backup* de dados, entradas de pulso (que podem ser utilizadas com contadores de energia externos), uma saída de pulso (passível de ser ligada a um ecrã externo), uma entrada para o alarme antirroubo, uma porta de comunicação *CAN bus* (para, por exemplo, ligar inversores com este tipo de comunicação), uma porta *RS485* a dois condutores com a designação *RS485-A* e uma porta de comunicação *RS485/422* em que possibilita a ligação de

um bus a dois ou a quatro fios, com a designação *RS485/422-B* (que permitem também a ligação com inversores ou acessórios adicionais) [14].

Os *loggers* desta empresa têm um custo que varia entre os 340 € e os 960 € [13][14] (na Figura 3 pode-se observar o *SolarLog base 2000*), para parques fotovoltaicos com potência instalada entre 15 kWp e 2 MWp, isto sem contar com o custo dos restantes equipamentos proprietários que teriam de ser adquiridos para expandir as suas funções. Note-se que estes são as versões base, dotadas apenas de duas portas de comunicação *RS485* ou uma porta *RS422*, duas portas *Ethernet*, duas portas *USB* e uma entrada de pulso. Existem módulos de expansão, caso sejam necessárias mais entradas ou saídas digitais.

Apesar dos muitos benefícios deste sistema, este não permite modificações efetuadas por terceiros. Em consequência, se o utilizador pretender efetuar modificações nas funcionalidades existentes, ou acrescentar novas, tal não é possível. Existem três versões do dispositivo, que são escolhidas mediante a potência nominal da instalação (até 50 kW, de 50 kW a 100 kW e de 100 kW até 2 MW).

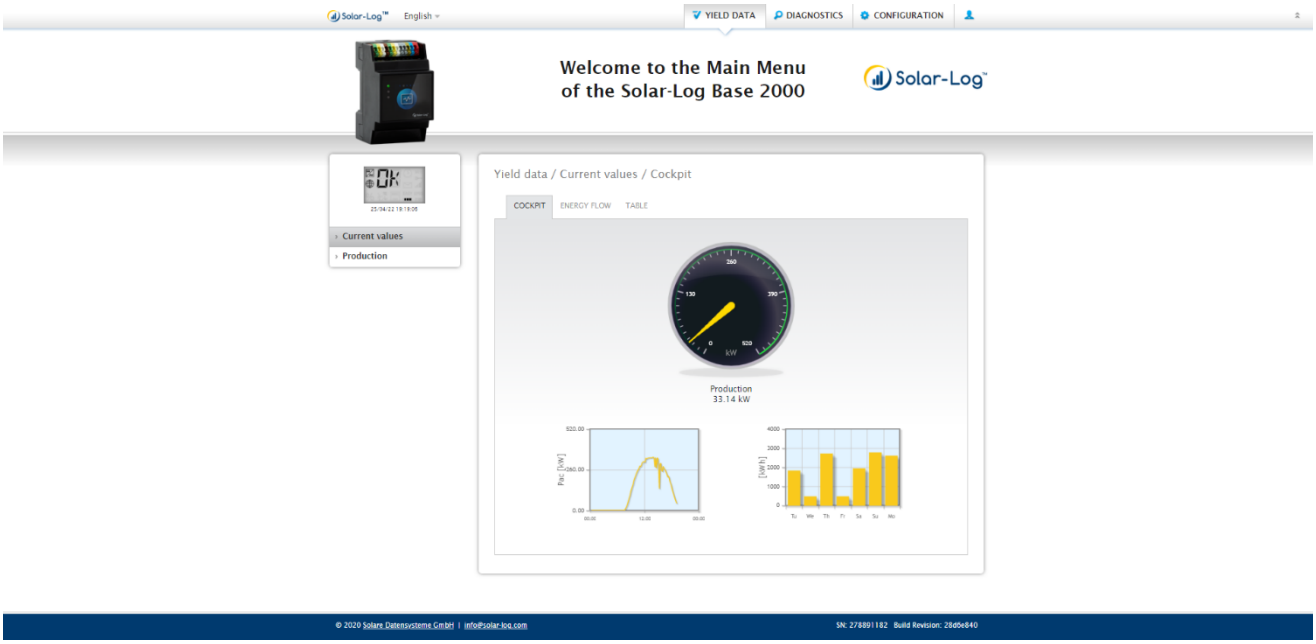


Figura 3 - *Solar-Log Base 2000*

2.2.2 Meteocontrol

A *Meteocontrol*, [17], oferece uma solução de registo de dados flexível com alguma independência, visto ser possível recolher dados dos principais fabricantes de equipamentos, nomeadamente inversores e alguns analisadores de energia. A sua aplicação (parte inferior da Figura 4) oferece um registo de valores, como potências, energia, irradiação, perdas, entre outros parâmetros.

O dispositivo *blue'Log X-Series*, [18], apresentado na parte superior da Figura 4, um conjunto *hardware* e *software* que permite a recolha de dados. Este dispositivo permite comunicação com até 100 equipamentos, podendo armazenar até 100 dias de dados, transferências através de *SFTP/FTP-Push* e oferece compatibilidade com 3300 dispositivos de fabricantes diversos. Este dipositivo possui duas portas de comunicação *RS485*, uma entrada *Ethernet*, uma entrada *CAN*, uma entrada por pulsos e três entradas analógicas (uma de tensão, uma de corrente e a terceira para ligação de uma resistência, por exemplo um *PT1000*). O *blue'Log X-Series* custa cerca de 780 € [19].

A solução *Meteocontrol* apresenta custo de instalação elevado e a uma dependência de terceiros para a inclusão de novos equipamentos ou até mesmo na resolução de possíveis problemas que ocorram com o *software*.

Figura 4 – *blue'Log X-Series*

2.2.3 Webdyn

A *Webdyn*, [20], é uma empresa que fornece um equipamento chamado *WebDyn*, Figura 5, o qual permite o registo de dados relativos à produção elétrica fotovoltaica, radiação solar, temperatura, gestão de alarmes, entre outros. Permite a ligação de até duzentos inversores, uma ligação *Modbus RTU* (*Remote Terminal Unit*) ou *TCP* (*Transmission Control Protocol*) e a ligação com *PLC* (*Programmable Logic Controller*) [21].

O *WebDyn* possui conectividade *3G/2G*, uma porta de comunicação *RS485*, outra *RS485/RS422*, quatro entradas analógicas, seis entradas digitais, duas saídas digitais e uma porta *Ethernet*. O preço deste equipamento ronda os 420 € [22].

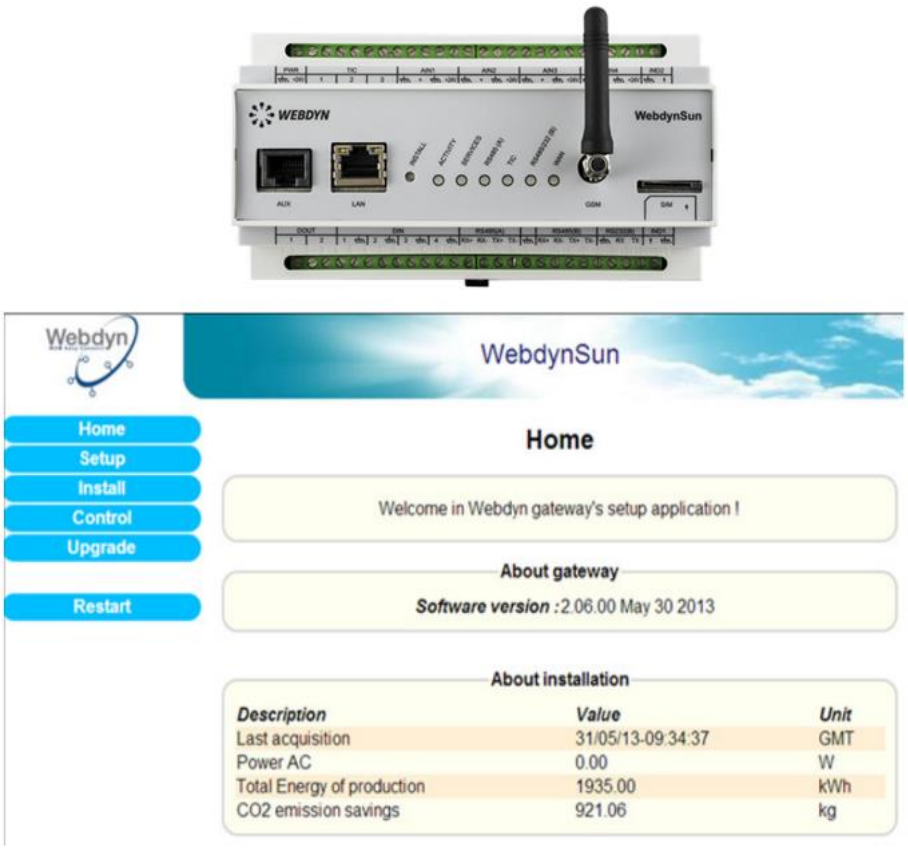


Figura 5 - *Datalogger WebDyn*

2.2.4 *SMA Cluster Controller*

A *SMA*, [23], oferece o *Cluster Controller*, Figura 6. O *Cluster Controller*, por sua vez, é indicado como um equipamento que faz uma monitorização e controlo profissional para grandes parques descentralizados. Ambos os dispositivos oferecem comunicação *Ethernet Modbus*, com o *Cluster Controller* a oferecer comunicação *FTP* e *interfaces* adicionais em *HTTP*. O *Cluster Controller* permite ligação com até um máximo de 75 dispositivos da *SMA* através de uma solução de *Ethernet* baseada em *Fieldbus*. No que diz respeito às entradas adicionais do dispositivo, apenas o *Cluster Controller* possui mais opções, tendo entradas *USB*, entradas e saídas digitais, entradas e saídas analógicas e entradas para medição de temperatura[24].

A *interface* dos dois *loggers*, permite uma visualização de todos os dispositivos conectados através de uma vista hierárquica, entre o *Master* e os *Slaves*. Nesta *interface* é possivel visualizar valores em tempo real, como a temperatura ou a potência, bem como o registo histórico dos mesmos.

O *SMA Cluster Controller* possui um custo aproximado de 850 € [25].

A *SMA*, apesar de oferecer uma excelente qualidade a nível de *software*, apresenta muitas limitações a nível de *hardware*, sendo que é necessária a utilização de dispositivos *SMA* para a comunicação com ambos os *dataloggers*.

Figura 6 - *SMA Cluster Controller*

2.2.5 Huawei Smart Logger

A *Huawei* apresenta o *Smart Logger 3000*, [26], Figura 7, um dispositivo de monotorização que permite a recolha de informação de até 80 inversores e outros 80 equipamentos. O *Smart Logger* possui uma porta *Ethernet*, uma porta USB, entradas e saídas digitais, quatro entradas analógicas, uma entrada de pulsos (para um medidor de energia) e até três portas *RS485* [26].

A *interface* deste dispositivo permite a monitorização simples, que possibilita uma gestão de alarmes, monotorização em tempo real e a elaboração de relatórios de desempenho. O menu desta aplicação possui seis opções: "*Deployment Wizard*", uma função onde se pode definir os parâmetros de implementação, conectar a dispositivos ou ao sistema de gestão; "*Over View*", aqui pode-se observar a informação da central, os alarmes ativos ou a energia produzida num período definido; "*Monitoring*", onde é apresentada informação em tempo real; "*Querry*", onde é obtido o histórico dos dados registados; "*Maintenance*", onde é possível efetuar atualizações de *software*; e por fim, "*Settings*", onde são criadas funções para a monotorização de dados. No caso da última opção é possível, por exemplo, definir uma função que apresente o rendimento diário, o rendimento total e a energia total produzida. É também possível apresentar uma lista com todas as informações referentes a cada inversor ligado a este sistema. No entanto a versão padrão não inclui suporte para dispositivos de terceiros, sendo necessário ter acesso a uma versão especial cujo custo pode chegar aos 80 mil dólares (versão *enterprise*) [27]. Existem duas versões deste equipamento, uma que possui comunicação com *MBUS* (padrão europeu para leitura de água, gás ou contadores de eletricidade) sendo que o custo deste *Smart Logger 3000* ronda os 1495 € [28], a versão sem este modelo de comunicação ronda os 560 € [29]. Esta solução apresenta uma grande dependência da própria *Huawei* para obter atualizações. O facto de ter de subscrever a uma versão *enterprise*, de modo a poder comunicar com dispositivos terceiros também se apresenta como uma desvantagem para esta solução.

Figura 7 - *Smart Logger* 3000 da *Huawei*

2.2.6 Sungrow Logger1000

O *Logger1000* da *Sungrow*, [30], Figura 8, é um sistema de monitorização, que permite a recolha de até 30 dispositivos. Esta solução utiliza uma ligação *WLAN* (*Wireless Local Area Network*), possui três portas *RS485*, cinco entradas digitais, quatro entradas analógicas e uma porta de *Ethernet*. Este sistema possui também uma *Web Interface* que pode ser observada na Figura 8 [31].

A *interface* é capaz de apresentar informação geral relativa ao sistema, monotorização de dispositivos, gestão de alarmes, gestão de dispositivos, registo histórico dos dados, entre outras funções [31]. Na Figura 8, pode-se observar um exemplo desta *interface*, o exemplo dos valores armazenados por um inversor, bem como o rendimento e potência ativa do mesmo.

O *logger1000* tem um custo de aproximadamente 435€, sendo que o custo do *software* não é apresentado. Apesar de esta solução ter um custo apelativo (não contando com o *software*) e possuir um *software* com os requisitos pretendidos, peca na flexibilidade, uma vez que é dependente da *Sungrow* para ajustes de *software* ou para ligação de novos dispositivos [32].

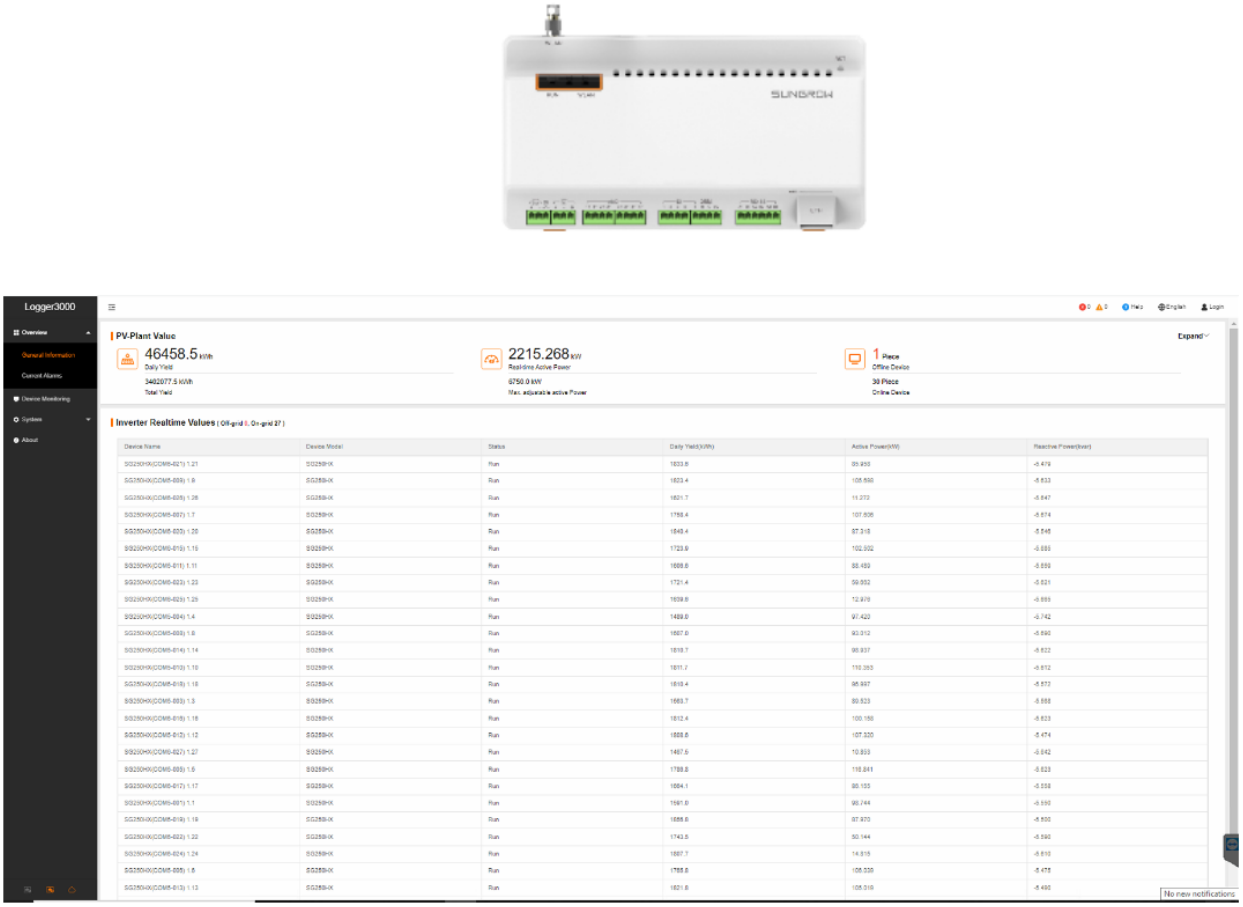


Figura 8 – Sungrow Logger 1000

2.2.7Higeco

A *Higeco*, [33], oferece uma variedade de *dataloggers* que, juntamente com o seu *software* de monotorização e controlo remoto, oferece um ambiente de desenvolvimento integrado, permitindo ao utilizador editar o *software* de acordo com os seus objetivos. O *software* oferece, entre outros, exportação de dados em tempo real, gestão de *drivers*, a criação de páginas (*interfaces*), para além de uma ferramenta de *email* que possibilita o envio de *emails* com relatórios gerados automaticamente. Esta solução possui ainda uma ferramenta de agendamento, comunicação através de exportação por *FTP* (*File Transfer Protocol*) e possui uma gestão de *drivers* que permite desenvolver e editar *drivers* de *modbus RTU* ou *TCP*.

A *Higeco* oferece uma gama de dispositivos capazes de recolha de dados, principalmente usados em *diversas* aplicações *IoT* (*Internet of Things*). Os *dataloggers* possuem portas *RS485*, *RS422* (apenas no modelo mais caro, Figura 9), uma porta *Ethernet* e portas *USB* [34]. Esta solução apresenta-se para um ambiente industrial, oferecendo uma elevada flexibilidade, permitindo ao utilizador a adaptação do sistema aos seus requisitos. O custo o modelo GWC 4DIN varia entre 980 € a 1100 € (apenas para a versão base, de modo a obter as funcionalidades descritas em cima é necessário um custa adicional). Esta solução, no entanto, para além de ser uma das mais dispendiosas, depende também de terceiros para resolução de potenciais problemas que possam ocorrer.

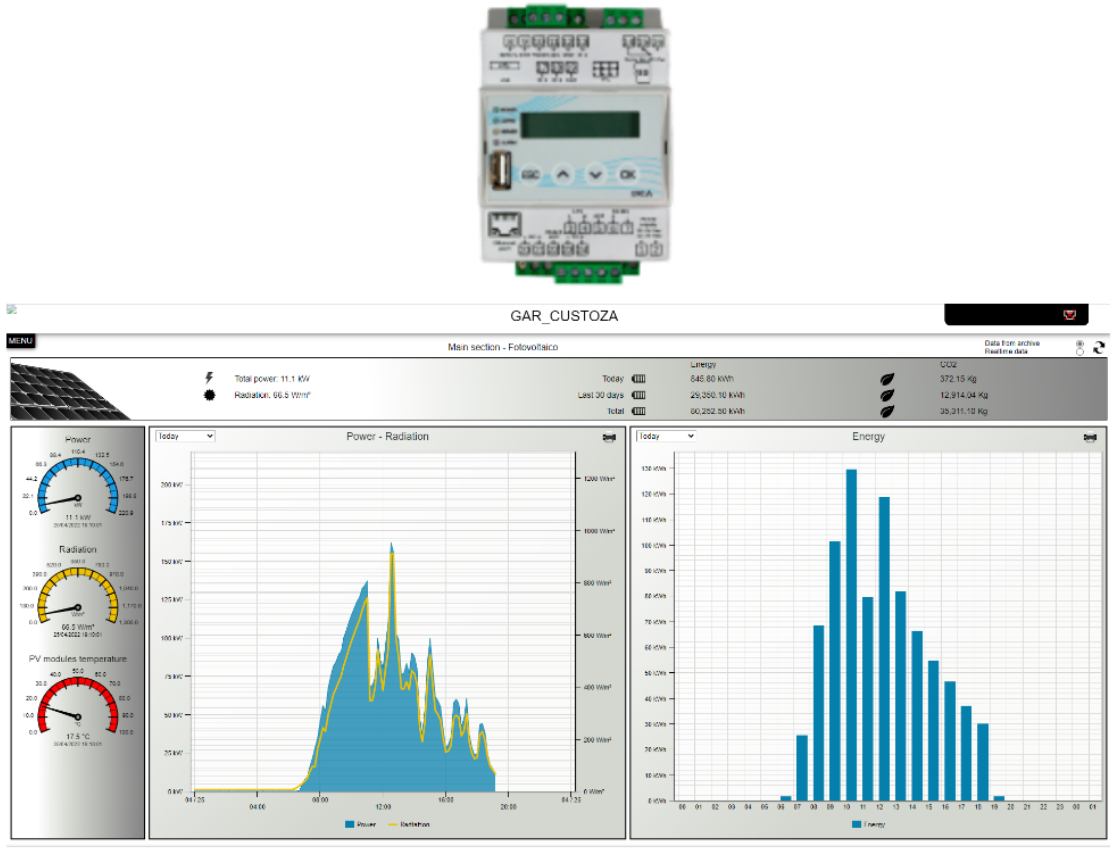


Figura 9 - *Higeco* GWC 4DIN

2.2.8 Comparação das soluções comerciais

A Tabela 1 permite, se forma sintetizada, comparar os diversos sistemas de monitorização comercialmente disponíveis apresentados nas secções anteriores.

Tabela 1 - Comparação entre os sistemas de monitorização comerciais referidos

Nome	**Custo do equipamento**	**Vantagens**	**Desvantagens**
Solar-Log	340-960€	-Sistema especializado para monitorização solar	-Impossibilidade da criação ou modificação de *drivers* -Elevados custos em comparação com outras soluções -Dependência de terceiros para resolução de problemas
Meteocontrol	780€	-Compatibilidade com uma grande variedade de dispositivos -Ligação com até 100 dispositivos -Até 100 dias de armazenamento	-Impossibilidade da criação ou modificação de *drivers* - Dependência de terceiros para resolução de problemas
WebDyn	420€	-Solução de baixo custo em comparação com as outras -Ligação com até 200 inversores	-*Software* básico demasiado limitado e com poucas das funcionalidades pretendidas
Cluster Control	850€	-Ligação com até 50 dispositivos -Monotorização de diversos parques fotovoltaicos -Desenvolvido especificamente para parques fotovoltaicos	-Um dos equipamentos mais dispendiosos - Impossibilidade da criação ou modificação de *drivers* - Dependência de terceiros para resolução de problemas -Necessário o uso de dispositivos da *SMA*
SmartLogger 3000 (Huawei)	390€	- Ligação com até 80 inversores e outros 80 dispositivos -Solução económica a nível de *hardware*	- Impossibilidade da criação ou modificação de *drivers* - Dependência de terceiros para resolução de problemas -Solução dispendiosa a nível de *software*
Logger 1000	435€	-Solução de baixo custo em comparação com as outras	- Impossibilidade da criação ou modificação de *drivers* - Dependência de terceiros para resolução de problemas
Higeco GWC-V2	980€-1100€	-Possibilidade de criação e modificação de *drivers* de *modbus* -Equipamentos desenvolvidos para aplicações *IoT*	- Dependência de terceiros para resolução de problemas - Dependendo da sua utilização pode acrescer custos de *plug-ins* (visualização de dados, envio de dados para servidores *FTP*, etc.)

Os sistemas de monitorização apresentados oferecem muitas das funcionalidades desejadas pela Voltalia para a sua atividade de monitorização de parques solares fotovoltaicos. Contudo, existem alguns requisitos que não são preenchidos pelas soluções existentes, nomeadamente relacionadas com a flexibilidade, resolução de problemas e os custos das respetivas soluções. Com o desenvolvimento de um sistema do qual a Voltalia detenha os direitos, esta tem acesso completo ao código base, facilitando futuras mudanças que se apresentem necessárias. Facilita também a resolução de problemas, deixando de depender de terceiros. Por fim, os custos são significativamente mais reduzidos, pois o *hardware* da solução passa por um *raspberry pi*, que apresenta um custo de cerca de 57€ [35].

3. Caracterização do sistema desenvolver

Analisados alguns dos sistemas existentes no mercado e tendo em consideração a pretensão da Voltalia apresenta-se seguidamente o sistema a desenvolver.

Neste capítulo é apresentado a metodologia utilizada, de modo a desenvolver um projeto no sentido de dotar a Voltalia de um sistema de monitorização que responda às suas necessidades efetivas.

3.1 Requisitos estabelecidos pela Voltalia

A Voltalia pretende ter um sistema desenvolvido especificamente para sua utilização, com acesso a todo o código fonte, podendo assim:

- Fazer alterações que sejam necessárias para ajustar às suas necessidades, incluindo modificações e a inclusão de *drivers* para comunicação com novos dispositivos;

- Comunicar com dispositivos instalados em parques fotovoltaicos mais antigos que sejam dotados de comunicação específica (*drivers* exclusivos), que pode dificultar a comunicação com o *software* de alguns *dataloggers*.

- Ter soluções com menores custos associados (quer de investimento quer de operação).

-Possuir soluções que possam ser instaladas desde pequenas instalações até grandes instalações, permitindo obter o melhor custo benefício para cada uma das instalações.

Assim, a Voltalia pretende que seja criado um sistema informático capaz de funcionar/ser executado num dispositivo *Raspberry Pi* e que possibilite a comunicação da aplicação já desenvolvida pela empresa com uma base de dados (BD). Para além disso, é também um requisito da Voltalia o desenvolvimento de uma *interface Web* que permita a configuração dos equipamentos a recolher os dados e que apresente os valores em tempo real e em histórico (isto é, um registo dos valores num determinado espaço de tempo). A apresentação de registo de valores impõe que estes estejam convenientemente armazenados. Neste caso, pretende-se que seja definida uma estrutura, na qual devem ser armazenados os valores provenientes dos diversos dispositivos que integram as centrais fotovoltaicas. Esta deve ainda permitir o armazenamento de dados relativos a configuração dos dispositivos que constituem as centrais fotovoltaicas, para que seja possível realizar a comunicação (*drivers*) com os dispositivos.

O desenvolvimento deste projeto implica assim a criação de uma *API* (Interface de programação de aplicações), um tipo de *interface* de *software* que irá possibilitar a ligação de vários pedaços de *software*, em concreto a ligação entre a aplicação de comunicação desenvolvida pela Voltalia, a BD desenvolvida e uma *interface WEB* na qual os utilizadores poderão interagir com os dados.

O desenvolvimento do sistema deve, por imposição da Voltalia, ser feito em *Python*, uma vez que a aplicação e comunicação existentes foi também ela desenvolvida nesta linguagem. Por outro lado, os responsáveis por futuras atualizações já estão preparados para o uso desta linguagem.

3.2 Metodologia a utilizar para a resolução do problema

Nesta subseção procura-se enaltecer os métodos encontrados para o desenvolvimento do sistema. Apresentando diversas soluções que solucionem o problema.

3.2.1Arquitetura do sistema e ferramentas utilizadas

A Figura 11 apresenta, esquematicamente, a topologia definida para o sistema desenvolvido, o qual procurou satisfazer os requisitos definidos pela Voltalia.

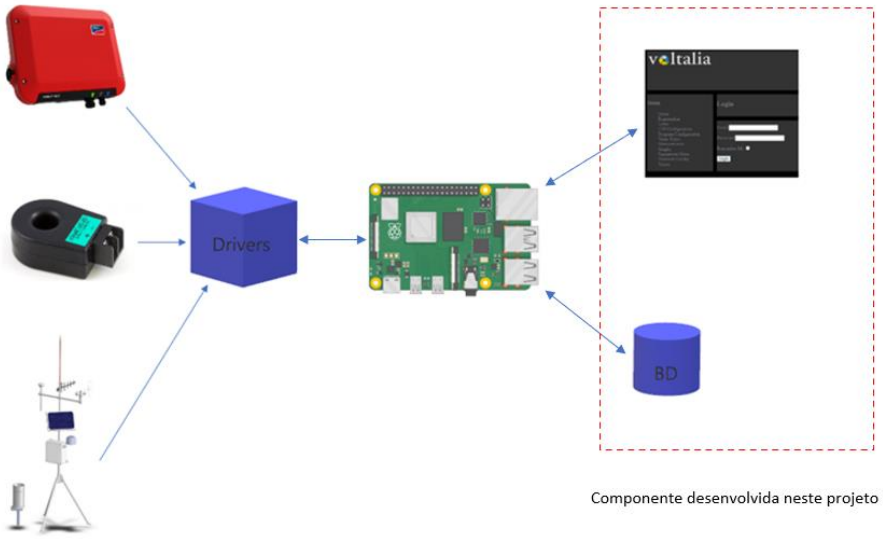


Figura 10 - Esquema do sistema desenvolvido

Sabendo que o dispositivo que irá executar a aplicação informática será um *Raspberry Pi*, o espaço ocupado pela BD terá de ser o menor possível. Para o dispositivo anteriormente referido, existem dois tipos de Sistema de Gestão de Bases de Dados (SGBD): relacionais e não relacionais (*NoSQL*).

De forma a se poder optar por um destes SGBD analisaram-se e testaram-se duas das opções, através de testes de desempenho, designadamente o *PostgreSQL* e o *MongoDB*.

A BD é uma parte crucial do sistema desenvolvido, uma vez que o objetivo principal daquele consiste na recolha (através da utilização de diversos sensores) e armazenamento de dados relativos ao funcionamento de parques solares fotovoltaico. O sistema desenvolvido tem, portanto, de, por um lado, assegurar que os dados recolhidos são armazenados de forma eficiente e, por outro, facultar um tempo de resposta adequado quando se pretende aceder a esses mesmos dados. Importa referir que os dados armazenados serão não apenas dados provenientes dos dispositivos, mas também dados referentes aos utilizadores (por exemplo permissões) ou informações dos dispositivos que sejam necessárias para o serviço de comunicação.

Por fim, será necessária uma *interface*, que torne possível a apresentação dos dados recolhidos, em forma de texto, tabela ou gráficos. Esta *interface* terá de ser intuitiva e bem estruturada, tendo ainda de possuir um menu para facilitar a navegação entre diferentes páginas, sejam de apresentação ou de introdução de dados necessários para ligar aos dispositivos.

O desenvolvimento do sistema apresentado na Figura 10 implicou o uso de um *framework*, de modo a poder construir e implementar uma aplicação *WEB*. Muitos *frameworks* possuem bibliotecas que fornecem acesso a bases de dados, algo necessário para este projeto, entre outros recursos necessários para a criação de uma aplicação. Existem vários, mas os mais conhecidos são o *Django* e o *Flask*. Neste trabalho optou-se pelo uso do *Flask*, uma vez que é já mais bem conhecido do autor.

O *Flask* possui inúmeras bibliotecas que contêm recursos necessários para uso numa empresa, algumas, por exemplo, facilitam o nível de segurança, de modo a dificultar o acesso a pessoas não autorizadas. Devido à popularidade do *Flask*, este possui diversas comunidades ativas que disponibilizam documentação útil que possui informação relevante para o desenvolvimento deste projeto.

3.2.2 Base de Dados

A base de dados é uma componente fundamental do sistema, pois este irá receber grandes volumes de dados que terão armazenados e consultados de forma eficiente. Como tal esta subseção procura comparar dois diferentes tipos de BD, descobrir o mais adequado e apresentar a metodologia para a sua criação.

3.2.2.1 Escolha do Sistema de Gestão de Bases de Dados

Os dois Sistemas de Gestão de Bases de Dados (ou SGBD) que foram, numa primeira análise, considerados como passíveis de serem utilizados no desenvolvimento do projeto foram o *PostgreSQL* e o *MongoDB*. Esta opção inicial resultou do facto de estes SGBD serem dos mais populares e oferecerem serviços gratuitos, o que facilita a pesquisa de documentação relevante para a execução do trabalho. Neste capítulo procura-se fazer uma comparação entre ambos para observar o que melhor se adequa ao sistema.

O *PostgreSQL* [36], é um SGBD relacional desenvolvido em *Open Source*. Uma BD relacional possui diversas vantagens, como a sua simplicidade do seu modelo. Através do uso de tabelas relacionadas umas com as outras através do uso de chaves primárias, pode-se evitar a repetição de dados, evitando também a duplicação de dados. Existe uma facilidade de acesso aos dados mais evidente num modelo racional, o uso do comando *JOIN*, por exemplo, permite a combinação de várias tabelas para se obter os valores que são pretendidos.

O *PostgreSQL* é um dos SGBD grátis mais avançados, desenvolvido em *Open Source*, possuindo características úteis ao trabalho a desenvolver, nomeadamente no que se refere à disponibilização de *views*, que permitem consultar um conjunto de dados de uma ou mais tabelas. Este facto facilita a recolha de dados da BD, sendo que por vezes apenas são necessários alguns dados de uma determinada tabela, ou até mesmo a combinação de dados de duas tabelas.

Adicionalmente, faculta também *stored procedures*, que são um conjunto de comandos que são executados, quando estes são invocados numa aplicação terceira, este processo pode ser usado, por exemplo para a inserção de dados na BD [37][38].

O SGBD *MongoDB* [39], também foi desenvolvido em *Open Source*, sendo classificado como sendo do tipo *NoSQL*. Ao contrário do *PostgreSQL*, o *MongoDB* é um SGBD orientado a documentos, os quais armazenam dados em formato *JSON* (JavaScript Object Notation). Este formato/linguagem possui a vantagem de, na generalidade, ser mais rápida do que o *SQL*. A sua aprendizagem é simples e funciona bem na *Cloud*. No entanto, o *MongoDB* não é uma solução completamente grátis, oferecendo apenas 5GB de armazenamento [40].

Para determinar qual dos SGBD seria o mais adequado ao projeto foram efetuados vários testes. Cada teste demorou 24 horas, sendo que durante esse período foram inseridos dados em duas tabelas (*PostgreSQL*)/coleções (*MongoDB*). Este teste teve como objetivo a observação do comportamento dos SGBD perante a inserção de diferentes volumes de dados. Para o efeito realizaram-se três testes para cada SGBD, nos quais se inseriam dados a cada 5 segundos, 1 minuto e 30 minutos. Os dados inseridos foram do tipo *Float*, uma vez que este tipo de dados é o que ocupa mais espaço. Posteriormente foi também realizado um teste no qual se introduziram, para além de dados do tipo *Float*, *strings*. O objetivo deste teste foi o de avaliar o desempenho num caso mais realista, visto que os dados serão armazenados juntamento com o nome do dispositivo de onde estes são provenientes.

Nos testes originais usaram-se duas tabelas/coleções, uma que tinha apenas um valor a ser introduzido e outra com dez valores. O objetivo desta opção está relacionado com o interesse em observar o consumo da memória produzido pelo aumento do número de colunas e dados contido numa tabela/ficheiro.

Para realizar os testes, foram desenvolvidas aplicações, utilizando o *Flask* (de forma a criar alguns hábitos com o *framework*). Para o teste do *PostgreSQL* foi usada a biblioteca *flask_sqlalchemy*, tendo-se começado por realizar a comunicação com a BD criada para se efetuar o teste, tal como se mostra na Figura 11.

```
11
12  app = Flask(__name__)
13  app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:postgres@localhost:5432/PV5'
14  app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
15
16  db = SQLAlchemy(app)
17  migrate = Migrate(app, db)
```

Figura 11 - Conexão da aplicação *Flask* com a base de dados

Em seguida foram criadas as tabelas, como referido anteriormente, sendo que uma teria uma coluna e a outra teria dez colunas, tal como se mostra na Figura 12. A inserção de dados

nestas tabelas implicou a produção de números aleatórios, os quais foram inseridos nas tabelas através da biblioteca *APScheduler*. Para tal programou-se uma função cujo objetivo seria criar valores aleatórios e inseri-los nas tabelas, sendo este processo executado num determinado intervalo de tempo (como já referido, a cada 5 segundos, 1 minuto e 30 minutos), tal como ilustrado na Figura 13.

```
30  class Test2(db.Model):
31      __tablename__ = 'test2'
32
33      id = db.Column(db.Integer, primary_key=True)
34      t2_var1 = db.Column(db.Float)
35      t2_var2 = db.Column(db.Float)
36      t2_var3 = db.Column(db.Float)
37      t2_var4 = db.Column(db.Float)
38      t2_var5 = db.Column(db.Float)
39      t2_var6 = db.Column(db.Float)
40      t2_var7 = db.Column(db.Float)
41      t2_var8 = db.Column(db.Float)
42      t2_var9 = db.Column(db.Float)
43      t2_var10 = db.Column(db.Float)
44
45      def __repr__(self):
46          return "Test 10 variables"
```

Figura 12 - Criação das tabelas

```
53  @app.route('/test')
54  def test():
55
56      sched = BlockingScheduler()
57
58      def job1():
59          now = datetime.datetime.now()
60          x = random.uniform(0, 5)
61          #n_1 = Test1(t1_var = x)
62          n1 = Test2(t2_var1 = x, t2_var2 = x, t2_var3 = x, t2_var4 = x, t2_var5 = x, t2_var6 = x, t2_var7 = x, t2_var8 = x, t2_var9 = x, t2_var10 = x)
63
64
65          #db.session.add(n_1)
66          db.session.add(n1)
67          db.session.commit()
68
69          print('data transfered at: ')
70          print (now.strftime("%Y-%m-%d %H:%M:%S"))
71
72      sched.add_job(job1, 'interval', minutes=1, start_date='2021-02-27 12:34:00', end_date='2021-02-28 12:34:00')
73
74      sched.start()
75
76      return "Test Finished"
```

Figura 13 - Inserção de dados nas tabelas e envio destas para a base de dados

O teste referente ao *MongoDB*, seguiu o mesmo modelo, usando a biblioteca *pymongo*, a qual permitiu realizar a comunicação com o servidor e a criação das coleções que iriam receber os valores, Figura 14. O *APScheduler* permitiu criar uma rotina de forma a inserir os valores nas coleções, tal como mostra a Figura 15.

```
app = Flask(__name__)
app.config["MONGO_URI"] = "mongodb+srv://test:mongodb@cluster0.fqorf.mongodb.net/test3?retryWrites=true&w=majority"
#app.config["MONGO_URI"] = "mongodb://localhost:27017/"
#myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mongo = PyMongo(app)

#mydb = myclient["test"]
#mycol = mydb["pv1"]

#pv1_collection = mongo.db.pv1
pv2_collection = mongo.db.pv2

```

Figura 14 - Ligação com a base de dados através do *pymongo*

```
@app.route('/test')
def test():

    result = {'result' : 'test successfull'}
    sched = BlockingScheduler()

    def job1():
        dict = {}

        now = datetime.datetime.now()
        x = random.uniform(0, 5)

        dict['val_1'] = x
        dict['val_2'] = x
        dict['val_3'] = x
        dict['val_4'] = x
        dict['val_5'] = x
        dict['val_6'] = x
        dict['val_7'] = x
        dict['val_8'] = x
        dict['val_9'] = x
        dict['val_10'] = x

        #n = mycol.insert_one(dict)
        n = pv2_collection.insert_one(dict)
        print('data transfered at: ')
        print (now.strftime("%Y-%m-%d %H:%M:%S"))

    sched.add_job(job1, 'interval', minutes=1, start_date='2021-07-05 15:28:00', end_date='2021-07-05 15:29:00')

    sched.start()

    return result
```

Figura 15 - Inserção dos dados

No primeiro teste realizado (de 5 segundos) foram gravadas mais de 17 mil linhas, cada uma com um valor aleatório. Na Figura 16 pode-se observar os resultados referentes ao teste do *PostgreSQL*, na primeira tabela (teste com uma coluna), o espaço ocupado foi de 1,156 MB, já a segunda tabela (com dez colunas) ocupou 2,352 MB. Na Figura 17 tem-se os resultados do segundo teste (1 minuto), no qual foram recolhidas quase 900 linhas. Neste caso, o espaço ocupado foi de 56 KB para uma coluna e 80 KB para dez colunas. Por fim, no teste final (30 minutos), presente na Figura 18, foram recolhidas 49 linhas, sendo que o espaço ocupado foi igual nos dois casos, 24 KB.

Na Figura 19 pode-se observar os testes referentes ao *MongoDB*, aqui o espaço ocupado no primeiro teste (de 5 segundos) foi de 590,66 KB, para uma variável e de 2,85 MB, para dez variáveis. No segundo teste (1 minuto), o espaço ocupado por uma variável foi de 9,84 KB e

por dez variáveis foi de 48,66 KB e por fim, no último teste (30 minutos) o espaço ocupado foi de 1,67 KB para uma variável e 8,28 KB para dez variáveis.

Com estes resultados (aglomerados na Tabela 2), pode-se concluir que no que diz respeito a baixas quantidades de dados, o *MongoDB* apresenta os melhores resultados. No entanto, com o escalar do volume de dados, o teste demonstra que existe um aumento muito significativo no espaço ocupado no *MongoDB.* O espaço ocupado passou de 46,66 KB para 2,85 MB no teste de 10 variáveis a cada 5 segundos, em comparação o aumento observado no *PostgreSQL* foi de 80 KB para 2,352 KB. Esta escalada continuaria a acentuar-se com a introdução de ainda mais dados. Ao fim de um mês de recolha de dados, a disparidade entre ambas seria muito mais elevada e iria-se observar um consumo de memória muito elevado. Por este motivo, optou-se por utilizar o *POstgreSQL*.

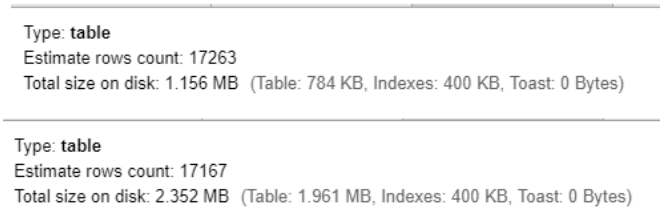


Figura 16 - Teste *PostgreSQL* a cada 5 segundos

Figura 17 - Teste *PostgreSQL* a cada minuto

Figura 18 - Teste ao SGBD *PostgreSQL* a cada 30 minutos

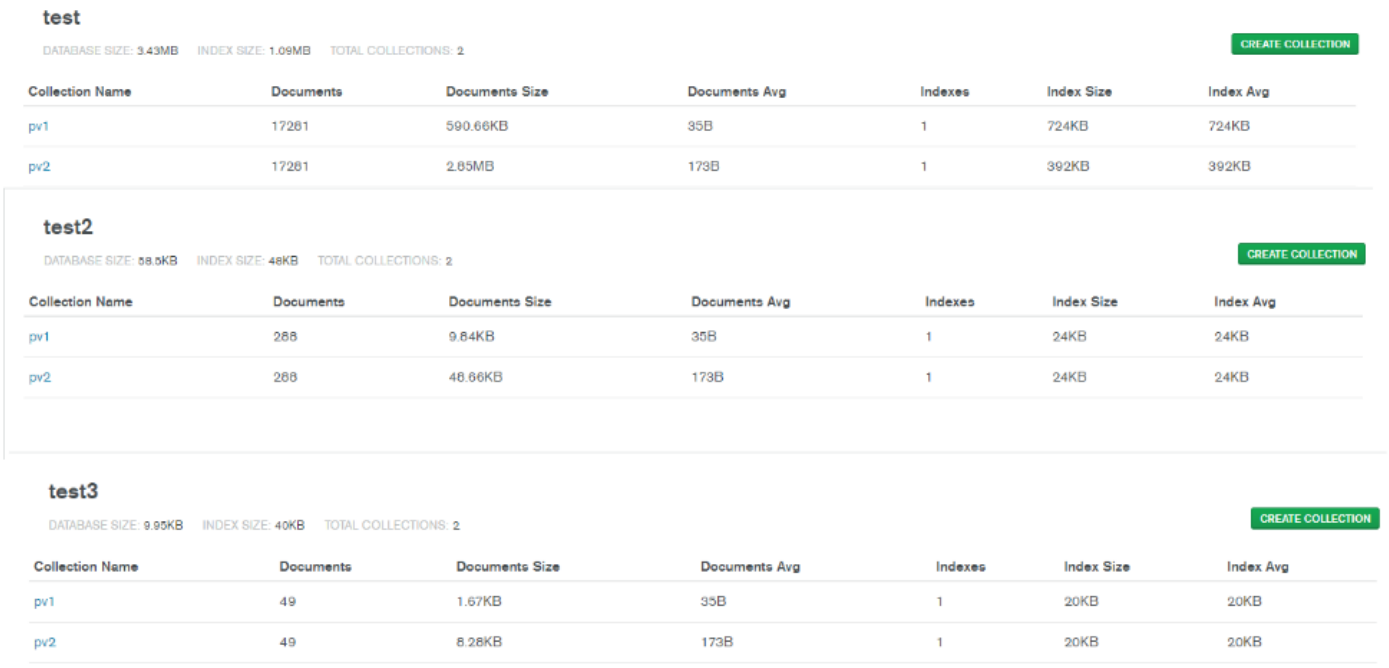


Figura 19 - Testes do ao SGBD *MongoDB*

~

Tabela 2 - Resultados dos testes

		PostgreSQL	MongoDB
1 variável	5 segundos	1,156 MB	590,66 KB
	1 minuto	56 KB	9,64 KB
	30 minutos	24 KB	1,67 KB
10 variáveis	5 segundos	2,352 MB	2,85 MB
	1 minuto	80 KB	46,66 KB
	30 minutos	24 KB	8,28 KB

3.2.2.2 Implementação da base de dados

Ao contrário dos testes, em que os valores foram gerados aleatoriamente dentro da aplicação, a informação será transmitida através da aplicação de comunicação, ou inserida pelo utilizador. Com isto o problema foi dividido em duas partes, a primeira foi a criação da BD, a segunda foi uma maneira de introduzir dados nesta.

3.2.2.3 Criação da base de dados

A Escola Superior de Tecnologia e Gestão de Viseu (ESTGV) possui uma licença para o *PowerDesigner* (neste trabalho foi usada a versão 16.7.1.1), que nos permite a criação do modelo de uma BD de forma gráfica. Este *software* torna o processo de definição de uma BD mais simples.

A BD será composta por sete tabelas principais, com as seguintes finalidades:

- *Equipments_config*, Figuras 20 e 21, o serviço de comunicação necessita de informação para realizar a comunicação com os dispositivos, por exemplo, o endereço de *ip*, ou o tipo de *driver* a ser utilizado. Estes dados serão armazenados neste grupo de tabelas. De modo a prevenir duplicação de dados na introdução de um novo equipamento, todos os dados que possam ser repetidos entre vários equipamentos, como por exemplo, os *drivers* (vários equipamentos podem usar o *ModBus*), foram separados em diferentes tabelas, relacionadas com a *Equipments_config*. Deste modo, no processo de introdução de um novo equipamento, o utilizador não terá de escrever o *driver* que necessita, mas sim aceder à tabela que possui os *drivers* e escolher o que é necessário. Assim previne-se não só a duplicação de informação dentro da BD, como potenciais erros que possam surgir na escrita da informação.
- *Users*, Figura 22, esta tabela irá armazenar os utilizadores que terão acesso ao sistema, esta está relacionada também com outra tabela que possui o nível de permissão que será atribuído a cada utilizador.
- *Measures_config*, Figura 23, aqui serão feitas as configurações para as medidas que irão ser recolhidas, como por exemplo, realizar uma multiplicação, caso o valor obtido não esteja na unidade pretendida, ou realizar um desvio do valor original.
- *ftp_config*, Figura 24, aqui serão introduzidas as configurações necessárias para realizar a comunicação em *FTP* (*File Transfer Protocol*), onde está implementado um *FTP Push*, que passa pelo envio de ficheiros de um *PC* cliente para um servidor *FTP* remoto. Aqui serão necessários o endereço do servidor, bem como a utilizador e palavra-passe.
- *Program_config*, Figura 25, aqui existem dados relativos à aplicação, desde localização da central, país, fuso horário, a informações mais técnicas, por exemplo, durante quantos

dias se mantêm os ficheiros no histórico, ou os *logs* (*days_maintainfiles_zip* e *days_maintain_log*).

- *Alarms*, Figura 26, é nesta tabela que serão armazenados os alarmes, a aplicação e comunicação inclui códigos referentes a diversos alarmes, portanto criou-se uma relação com outra tabela, *error_codes*, aqui encontram-se os códigos associados ao erro que estes representam. Quando ocorrer um alarme, será registado a data de início e data do fim, bem como o código do alarme que foi ativado.
- *Pv_values*, Figura 27, por fim, na tabela serão armazenados os valores de medição provenientes dos diversos dispositivos, esta tabela está relacionada com a *equipments_config*, de modo a poder apresentar qual foi o dispositivo que apresentou os valores e está relacionada com a *measurements*, onde estão os diferentes tipos de medidas e variáveis, Figura 28.

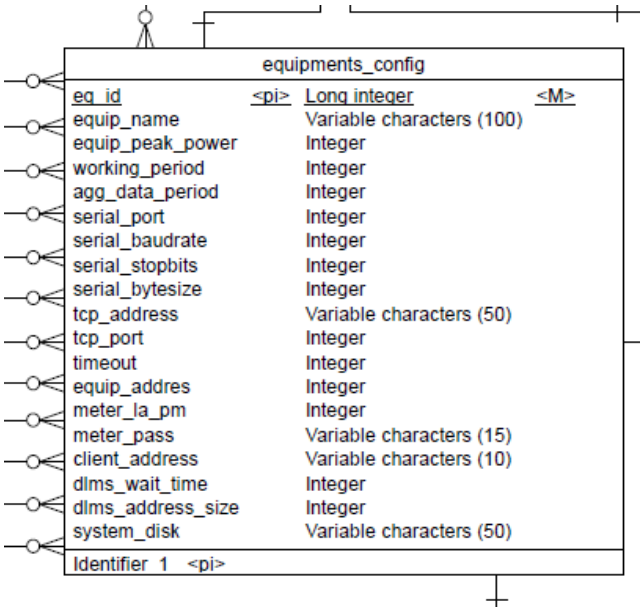


Figura 20 - Tabela *equipments_config*

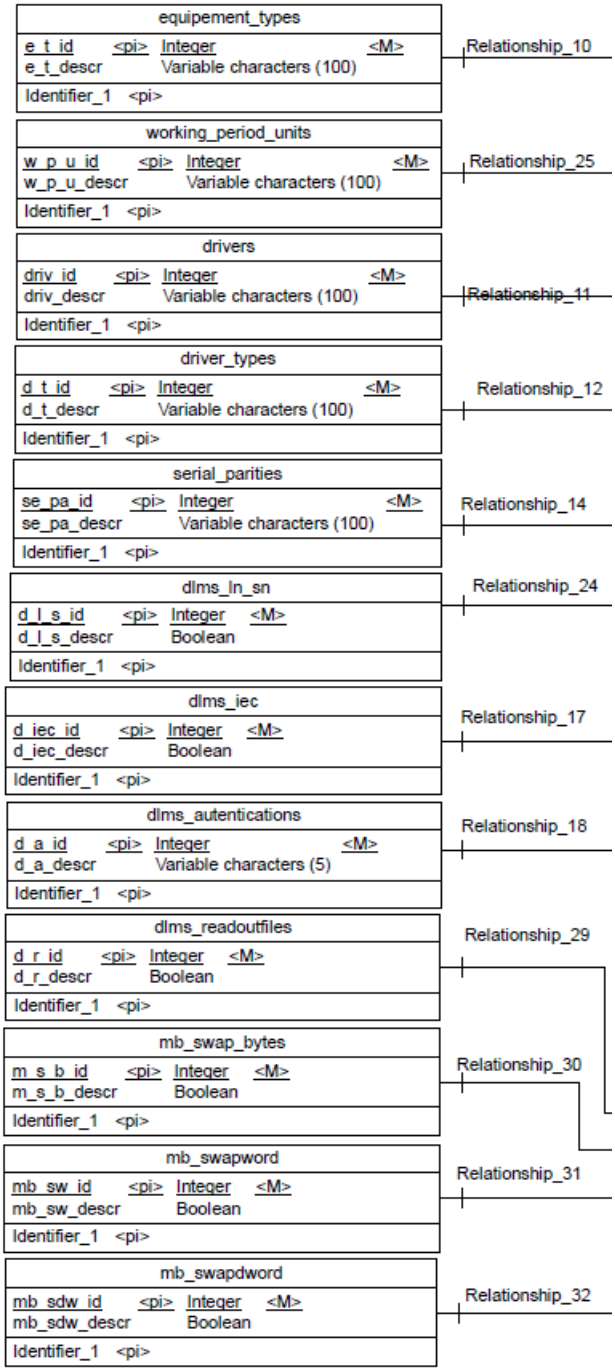


Figura 21 - Tabelas associadas à *equipments_config*

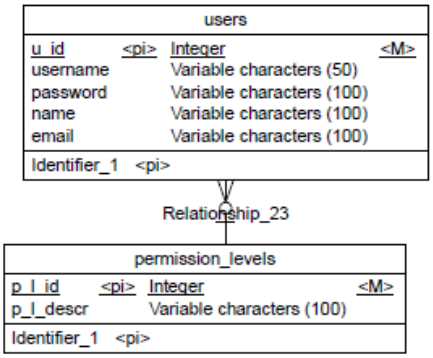


Figura 22 - Tabela referente aos utilizadores

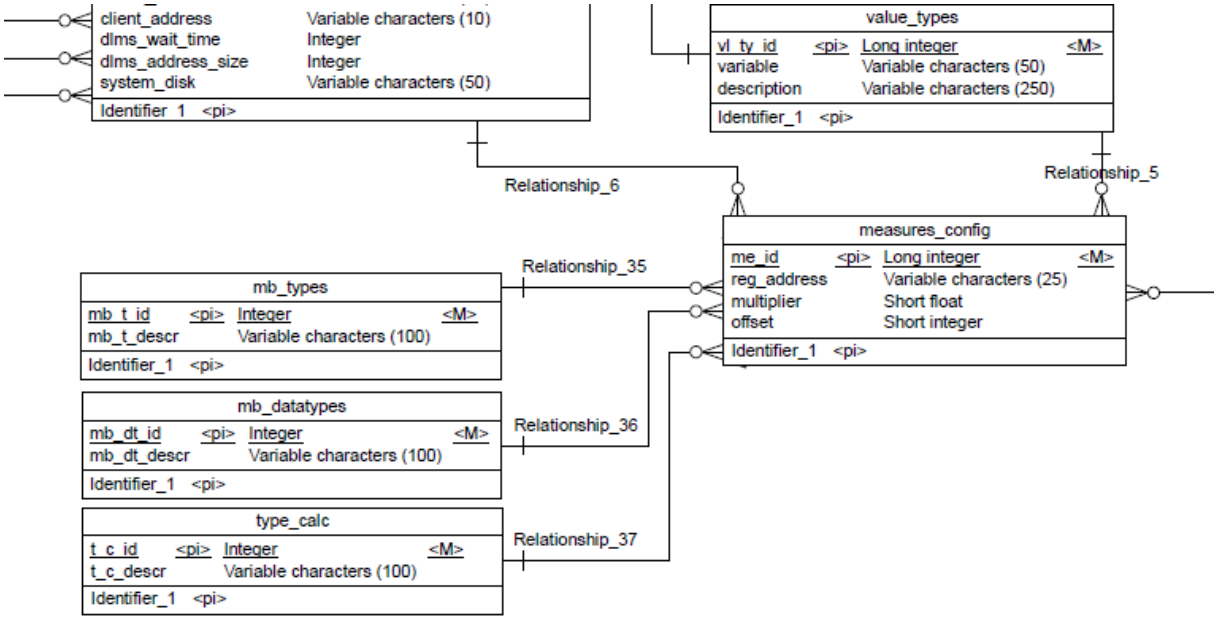


Figura 23 - Tabela refente à configuração das medidas

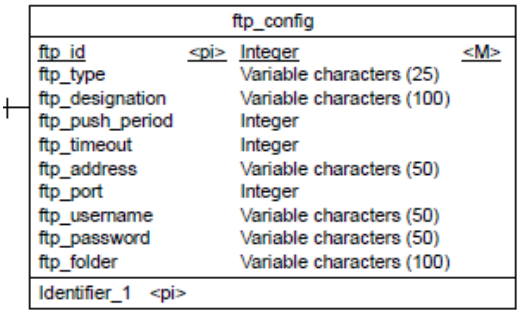


Figura 24 - Tabela referente à configuração do servidor FTP

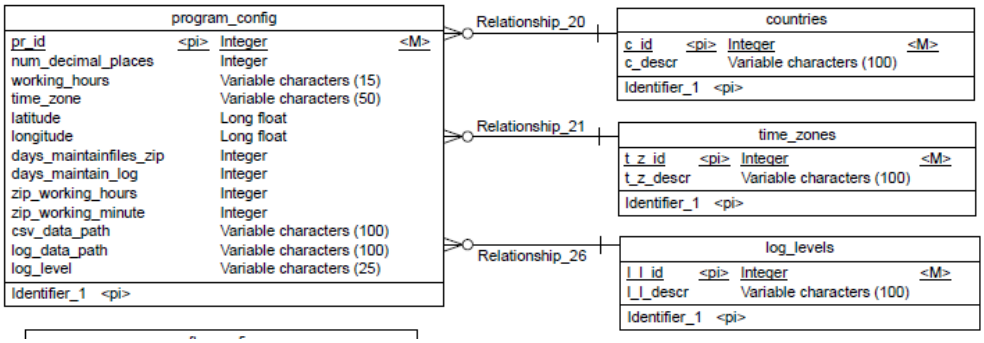


Figura 25 - Tabela referente à configuração da aplicação

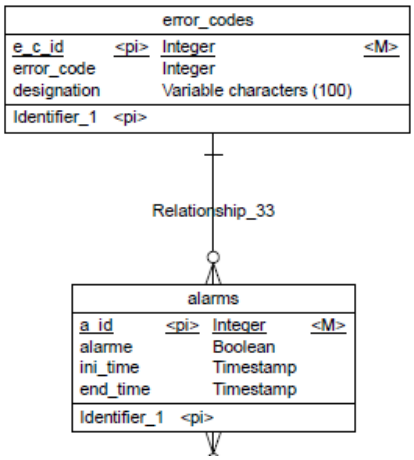


Figura 26 - Tabela referente aos alarmes

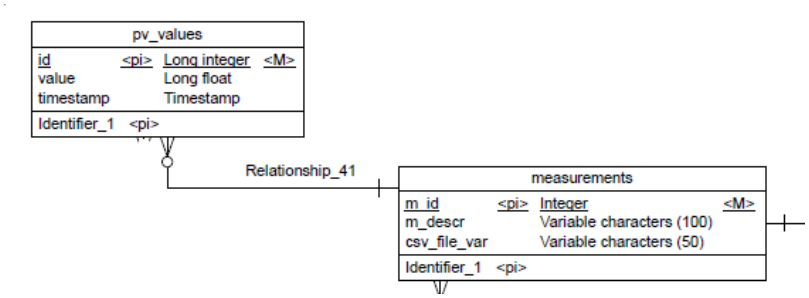


Figura 27 - Tabela que irá armazenar os valores

m_id	vl_ty_id	m_descr	csv_file_var
1	1	Ambient Temperature	AmbientTemp(°C)
2	2	Voltage phase L1	VoltageL1(V)

Figura 28 - Exemplo do que será armazenado dentro da tabela *measurements*

Devido à dimensão do modelo da BD, este encontra-se disponível em formato completo no segundo apêndice.

Para cada tabela foi criada uma sequência. As sequências têm como objetivo, a sequenciação das chaves primárias de cada tabela, deste modo, cada novo valor que seja introduzido na tabela irá receber a numeração adequada automaticamente. Na Figura 29 ilustra-se a criação da sequência referente à chave primária da tabela *equipments_config*, como se pode observar, o valor inicial será de 1, este irá ser incrementado em valores de 1 até um máximo de 10000 (dez mil).

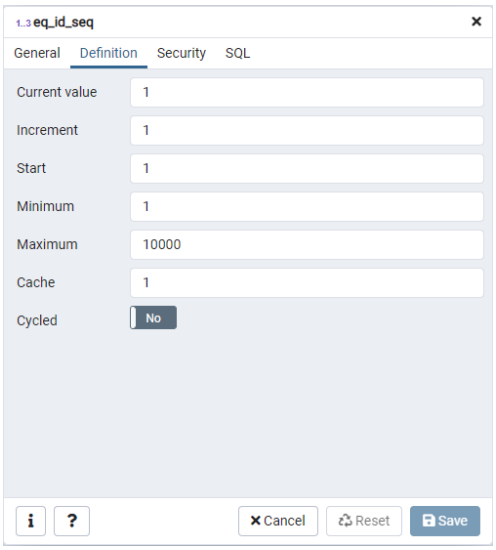


Figura 29 - Criação de uma sequência

Para inserir valores na BD existem dois métodos, o primeiro passa pelo envio de código *SQL* diretamente na aplicação de *python*. Esta método não é aconselhado, pois diminui o desempenho da operação, podendo esta demorar mais tempo. O segundo método, Figuras 30 e 31, utiliza procedimentos para introduzir diretamente na BD, fazendo com que se diminua o número de comunicações com a BD, deste modo, não só se reduz a quantidade de código presente na aplicação, como também se reduz o impacto na performance da operação. Estes tipos de procedimentos serão usados para todas as tabelas. Cada procedimento irá inserir valores na BD. Conforme se pode verificar na Figura 30, utiliza-se uma sequência para a chave da tabela *measures_config*.

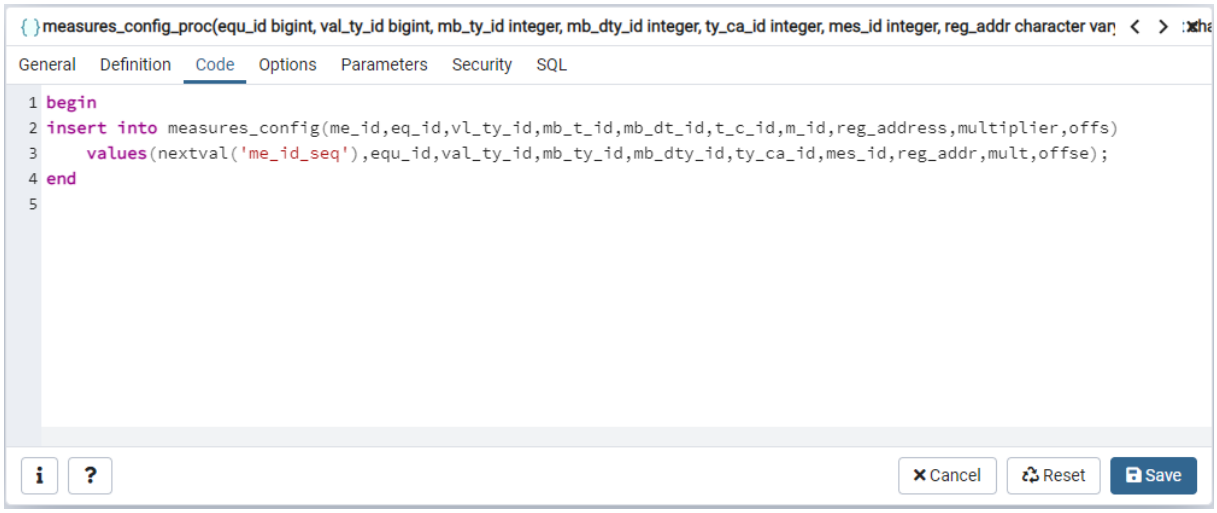


Figura 30 - Procedimento relativo à configuração das medidas

```
cursor.execute("CALL measures_config_proc(%s,%s,%s,%s,%s,%s,%s,%s,%s);", (eq_id, vl_ty_id, mb_t_id, mb_dt_id, t_c_id, m_id, reg_address, multiplier, offs))
```

Figura 31 - Chamamento do procedimento no código

Por último, tem de se recolher valores da BD, para apresentar as medições de um dispositivo, ou outros dados relevantes. Usando o exemplo dos *drivers*, quando se for introduzir um novo equipamento, já existem uma série de *drivers* que estão presentes na BD. Deste modo, é necessário aceder a esta lista e selecionar o *driver* que é pretendido para o novo dispositivo. As vistas acedem a uma tabela e retiram a informação que é necessária desta, na Figura 32 pode-se observar a vista relativa aos *drivers*, aqui são selecionadas as colunas que são pretendidas da tabela dos *drivers*, Figura 33, neste caso será o id (chave primária) dos *drivers* e a descrição do mesmo. Caso seja necessário juntar informação de duas tabelas, usa-se o comando *JOIN*

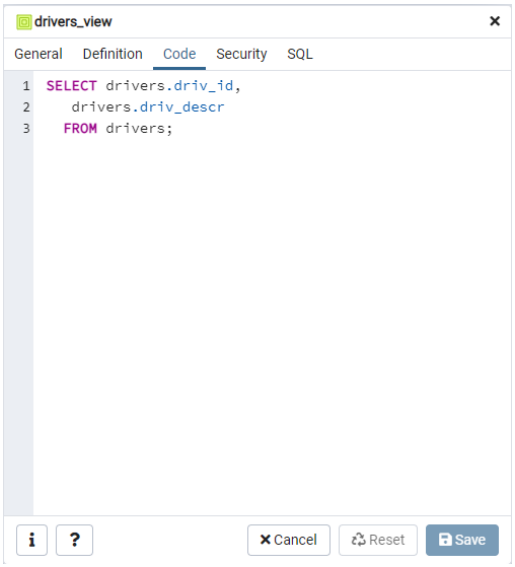


Figura 32 - Vista criada para os *drivers*

	driv_id [PK] integer	driv_descr character varying (100)
1	1	IEC102
2	2	ModusTCP
3	3	ModusUDP
4	4	ModusRTUTCP
5	5	ModusRTU
6	6	ModusASCII
7	7	ModusTCPServer
8	8	MTSensor
9	9	SystemStatus
10	10	DLMSEthernet_TCP
11	11	DLMSEthernet_UDP
12	12	DLMSSerial
13	13	S7PLC

Figura 33 - Tabela dos *drivers*

3.2.3 Desenvolvimento da *Interface*

Para a *interface*, como mencionado anteriormente, foi usado o *framework Flask*. Este possui uma grande variedade de recursos já incluídos na livraria, por exemplo o *flask_sqlalchemy* (usado nos testes). No entanto, este não possui todas as funcionalidades de comunicação desejadas, para tal, foi usado uma livraria mais avançada, o *psycopg2*. Para a criação de endereços neste *framework*, é necessário o uso de decoradores no *python*, estes têm por objetivo adicionar mais funcionalidades a um objeto sem que a sua estrutura seja alterada. Um exemplo do uso destes decoradores pode ser observado nos testes realizados no subcapítulo 3.2.2.1, mais concretamente na Figura 15.

3.2.3.1 Introdução de dados

O *Flask* [41] possui uma livraria adequada à introdução de dados pelo utilizador, o *flask_wtf*. Esta livraria, que juntamente com a livraria *wtforms*, que complementa a anterior, possibilita a criação de formulários que permitem ao utilizador preencher campos numa página *WEB* com dados que são guardados em variáveis no *python*. Para criar e utilizar estes formulários são necessários dois passos. O primeiro passa pela criação de uma função respetiva ao formulário e o segundo chamar esta função no decorador onde este será utilizado.

A livraria *wtforms* possui um recurso intitulado de campos. Estes são importados da livraria e têm como objetivo especificar o tipo de dados que serão passados por aquele campo. O mais usado destes campos é o *StringField*, neste é introduzida uma *string* que é passada para o construtor que armazena esta informação. Na Figura 34, pode-se observar um exemplo deste construtor, um *PasswordField* que armazena a informação de uma palavra-passe no construtor *password*. Nesta figura, pode-se também observar uma *string* com *password* escrita, isto representa o que será apresentado ao utilizador, que é seguido de um campo aonde o utilizador introduz a respetiva informação. Observa-se ainda o *validator*, este indica o requisito para que o campo seja validado. Neste caso a validação ocorre apenas quando é introduzida alguma informação, sendo que este campo não pode ficar em branco. Sendo este um campo *PasswordField*, ao contrário de outros campos, quando o utilizador introduz informação, esta é escondida, aparecendo bolas em vez dos caracteres introduzidos.

```
11      password = PasswordField('Password', validators=[DataRequired()])
```

Figura 34 - Campo *PasswordField*

3.2.4 Recolha de dados dos equipamentos

O objetivo deste *datalogger* passa pela recolha de dados referentes a uma central, para que estes possam ser analisados. Para tal é necessário que ocorra uma comunicação a aplicação de comunicação continua, de modo que exista uma constante recolha de dados. Para que isto ocorra a componente de comunicação não se pode inserir na aplicação principal. Mesmo que o utilizador desligue a *interface*, a recolha de dados tem de permanecer. Caso contrário existiram falhas nos dados recolhidos, deixando de existir informação relativa a todos os instantes da central. A solução passa pela criação de uma segunda aplicação. Esta irá ser executada continuamente, assim não irão ocorrer falhas caso a *interface* seja desligada. Esta aplicação tem de efetuar uma comunicação não só com a BD, mas também com o serviço de comunicação da Voltalia.

3.2.5 Apresentação dos dados

Para a apresentar os dados serão usados dois métodos, tabelas e gráficos. Uma vez armazenados os dados, estes serão consultados pelo utilizador. No entanto têm de ser apresentados de forma organizada de modo a poderem ser consultados e analisados. Para tal serão usadas duas livrarias para cada caso.

Em relação às tabelas, existem muitas livrarias associadas *Python*. Para este trabalho escolheu-se a livraria *plotly*, esta livraria é relativamente simples de implementar e inclui ainda algum nível de customização de modo a poder condizer com o resto do sistema. De modo a criar uma tabela, é necessário criar uma figura. As funções da livraria irão receber duas listas. A primeira, através de uma lista, irá informar quais serão os cabeçalhos da tabela. A segunda irá introduzir os valores na tabela, mais uma vez através do uso de listas.

No que toca aos gráficos, dos diferentes métodos, foi escolhida a livraria *matplotlib*. Uma livraria com reportório de gráficos bastante extenso. Esta livraria, á semelhança das tabelas, também constrói uma figura, aonde serão introduzidos os gráficos. Estes são criados através de listas, onde cada lista será associada a um eixo. Existem equipamentos numa central que podem recolher mais do que uma medida. Portanto os gráficos não podem ser associados a um dispositivo, pois existe mais do que uma informação que tem de ser apresentada. A solução passa pela criação de uma figura para o dispositivo que irá possuir um gráfico para cada medida recolhida.

4. Implementação do sistema

Por fim, neste capítulo, procura-se descrever a implementação do sistema desenvolvido. Para tal, demonstra-se a estrutura do sistema. Desde a *interface* à comunicação efetuada com as diferentes aplicações.

Efetua-se também testes ao sistema criado. Para tal, criou-se um utilizador, efetuou-se o *login*, registou-se um equipamento real e obteve-se as medições do mesmo. Deste modo, é possível observar as interações do sistema. Alguma da informação não é apresentada, de modo a respeitar a confidencialidade para com a Voltalia.

4.1 Estrutura do sistema

Neste subcapítulo procura-se apresentar a estrutura do sistema, desde as vistas que serão apresentadas ao utilizador à recolha de dados. A *interface* funciona através de duas componentes principais. Os formulários que são preenchidos pelo utilizador. E pelas páginas *web*, que são criadas através do *Flask* com o uso de decoradores. Para demonstrar a estrutura da *interface*, usa-se o exemplo de uma página *web* relativa ao registo de um novo utilizador. Demonstra-se também a recolha de dados e a estrutura das figuras.

4.1.1 Criação do formulário para recolha de dados

Na Figura 35, pode-se observar o formulário referente ao registo de novos utilizadores, intitulado de *RegistrationForm*. Este herda as características do *FlaskForm* que nos é fornecido através da livraria *flask_wtf*. Dentro desta função foram criadas as variáveis necessárias para armazenamento da informação relativa a cada utilizador. Como se pode observar na Figura 35,

existem campos que estão a ser invocados, três para ser mais preciso. O primeiro, *StringField*, informa que ali serão inseridas variáveis do tipo *string*. O segundo, *PasswordField*, informa que ali será introduzida uma palavra-passe, o que fará com que no ecrã se esconda o que está a ser escrito, aparecendo bolas em vez dos caracteres introduzidos. Por fim tem-se o *SubmitField*, este irá criar um botão que irá guardar a informação introduzida nas variáveis.

```
7   class RegistrationForm(FlaskForm):
8       username = StringField('Username', validators=[DataRequired()])
9       name = StringField('Name', validators=[DataRequired()])
10      email = StringField('Email', validators=[DataRequired(), Email()])
11      password = PasswordField('Password', validators=[DataRequired()])
12      password2 = PasswordField('Repeat Password', validators=[DataRequired(), EqualTo('password')])
13      submit = SubmitField('Register')
```

Figura 35 - Função *RegistrationForm*

4.1.2 Aplicação *app.py*

Do lado do decorador, Figura 36, valida-se o formulário e efetua-se a ligação à BD, através da livraria *psycopg2*. De seguida, cria-se um *cursor* de modo comunicar com a BD e executar os comandos de *SQL*, associa-se as variáveis recolhidas pelo formulário e executa-se o procedimento, onde serão inseridas as variáveis. Por fim, cometem-se os dados à BD e fecha-se a comunicação. Alerta-se que o código dos formulários acabou por ficar noutra aplicação, intitulado de *forms.py*, de modo a poder organizar melhor o código final, o *p_l_id*, é a variável referente às permissões do utilizador, que será atribuída automaticamente a cada utilizador criado, que neste caso será sempre de visitante. Pode-se também observar o uso da função *generate_password_hash*, da biblioteca *werkzeug*, cujo objetivo é realizar um *hash* na *password*, isto é, através de um algoritmo, esta torna-se numa representação de caracteres embaralhados da mesma.

De modo a poder comparar as *passwords*, usa-se uma função, da mesma biblioteca, *check_password_hash*, esta foi inserida num ciclo, Figura 37, que irá percorrer todas as *passwords* armazenadas. Se a *password* inserida for válida, a função apresentará um resultado verdadeiro, caso não seja válida, apresentará um resultado falso.

```python
@app.route('/registration', methods=['GET', 'POST'])
def registration():
    registration_form = RegistrationForm(meta={'csrf': False})
    if registration_form.validate_on_submit():
        try:

            # Connect to an existing database
            connection = psycopg2.connect(user="postgres", password="postgres", host="localhost", port="5432", database="PV")

            cursor = connection.cursor()
            p_l_id = 3
            username = registration_form.username.data
            password = generate_password_hash(registration_form.password.data)
            name = registration_form.name.data
            email = registration_form.email.data
            # call stored procedure
            cursor.execute("CALL users_proc(%s,%s,%s,%s,%s);", (p_l_id, username, password, name, email))
            connection.commit()
            cursor.close()


        except (Exception, psycopg2.DatabaseError) as error:
            print("Error while connecting to PostgreSQL", error)

        finally:
            # closing database connection.
            if connection:
                cursor.close()
                connection.close()
                print("PostgreSQL connection is closed")
    return render_template('registration.html', title='Registration', form=registration_form)
```

Figura 36 - Decorador da página de registo

```python
                for r_email,r_password in users:
                    if r_email == t_email:
                        if check_password_hash(r_password, t_password) == True:
                            print(" ")
                            print(users)
                            print(" ")
                            print(r_email)
                            print(" ")
                            print(t_password)
                            print(" ")
                            print("Login successful")
                            return render_template('loginsec.html', title = 'Loginsec')
                        else:
                            print(" ")
                            print(users)
                            print(" ")
                            print(r_email)
                            print(" ")
                            print(t_password)
                            print(" ")
                            print("Login failed")
                            return render_template('loginfai.html', title = 'Loginfai')
```

Figura 37 - Verificação da password introduzida

Com a estrutura concluída, foram necessários os formulários com o código *HTML*, para que o utilizador consiga observar os campos onde serão inseridos os dados requisitados. No fundo da Figura 36, pode-se observar o retorno de uma função, intitulada de *render_template*, que configura o motor de modelos Jinja2 automaticamente. Deste modo, será possível efetuar a processamento digital de um ficheiro *HTML* (*HyperText Markup Language*), neste caso,

registration.html. É necessário também indicar qual o formulário usado, neste caso, *registration_form*. No ficheiro de *HTML*, Figura 38, de modo a poder aceder ao formulário criado, é necessário fazer a associação com o modelo. Para esta associação, será invocada a *hidden_tag*, de forma a renderizar qualquer campo que possa estar escondido, incluindo um campo CSRF (*Cross-site request forgery*), de seguida criam-se parágrafos que irão invocar os campos criados no formulário, obtendo por fim algo semelhante à Figura 39.

```
{% extends "base.html" %}
{% block title %}Registration{% endblock %}
{% block content %}
    <footer>
        <h1>Registration</h1>
    </footer>
    <main>
        <form action="/registration" method="POST">
            {{ form.hidden_tag() }}
            <p>{{ form.username.label }}
            {{ form.username }}</p>
            <p>{{ form.name.label }}
            {{ form.name }}</p>
            <p>{{ form.email.label }}
            {{ form.email }}</p>
            <p>{{ form.password.label }}
            {{ form.password }}</p>
            <p>{{ form.password2.label }}
            {{ form.password2 }}</p>
            <p>{{ form.submit() }}</p>
        </form>
    </main>
{% endblock %}
```

Figura 38 - Formulário do lado *HTML*

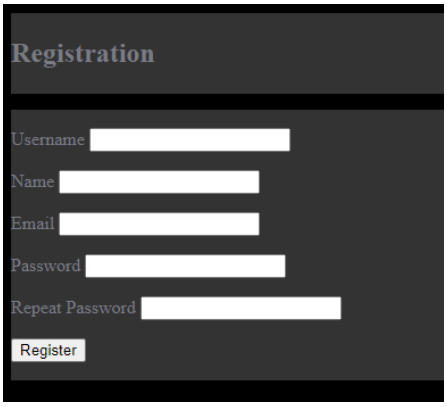


Figura 39 - Vista de registo

4.1.3 Estrutura das páginas *WEB*

Foram criadas páginas *web* para cada situação, nomeadamente: i) registo de um utilizador; ii) login; iii) configuração dos diferentes dispositivos; iv) vista dos gráficos; v) configuração do servidor; vi) configuração do sistema.

Para efetuar uma navegação eficiente, criou-se um menu, que irá estar presente em todas as páginas *web*. Este menu foi criado no ficheiro *base.html*, aqui foram criadas listas com referências a cada página, Figura 40. Para que este menu esteja presente em todas as páginas foi criado um bloco de conteúdo, começando em *{% block content %}* e acabando em *{% endblock %}*. Na Figura 38, pode-se observar que este bloco de conteúdo é preenchido entre as linhas 4 e 22, fazendo a ligação com o ficheiro *base.html*.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <link rel="stylesheet" type="text/css"  href="{{ url_for('static',filename='styles/base.css') }}">
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>PyVBoxLite</title>
9  </head>
10 <body>
11     <section id="page">
12         <header>
13             <img src="{{ url_for('static',filename='MicrosoftTeams-image5.png') }}" class='comp_img'>
14         </header>
15         <nav>
16             <h1>Menu</h1>
17             <ul>
18                 <li><a href="/">Home</a></li>
19                 <li><a href="/registration">Registration</a></li>
20                 <li><a href="/login">Login</a></li>
21                 <li><a href="/ftp">FTP Configuration</a></li>
22                 <li><a href="/program">Program Configuration</a></li>
23                 <li><a href="/valuet">Value Types</a></li>
24                 <li><a href="/measurements">Measurements</a></li>
25                 <li><a href="/graphs">Graphs</a></li>
26                 <li><a href="/equipmentm">Equipment Menu</a></li>
27                 <li><a href="/measCfg">Measures Config</a></li>
28                 <li><a href ="/equipval">Values</a></li>
29             </ul>
30         </nav>
31         {% block content %}
32         {% endblock %}
33     </section>
34 </body>
35 </html>
```

Figura 40 - Ficheiro *base.html*

De modo a manter uma proximidade sistema com outros desenvolvidas pela Voltalia, a mesma pediu que fosse usada uma estrutura semelhante, desde cores a símbolos. Através de *CSS*, criou-se uma estrutura, usando *grid-template-areas*, Figura 41, com o objetivo de criar um *design* organizado. Neste ficheiro foram também alteradas as cores, de modo a coincidirem com o design modelo oferecido pela Voltalia, acabando com o resultado que se pode observar na Figura 42.

```
#page {
    display: grid;
    width: 100%;
    height: 100%;
    grid-template-areas: "head head head head head head head head head"
                         "head  head head head head head head head head"
                         "head  head head head head head head head head"
                         "nav   foot foot foot foot foot foot foot foot"
                         "nav   foot foot foot foot foot foot foot foot"
                         "nav   main main main main main main main main"
                         "nav   main main main main main main main main"
                         "nav   main main main main main main main main"
                         "nav   main main main main main main main main";
    grid-template-rows: 125px 1fr 30px;
    grid-template-columns: 250px 1fr;
}
```

Figura 41 - *grid-template-areas*

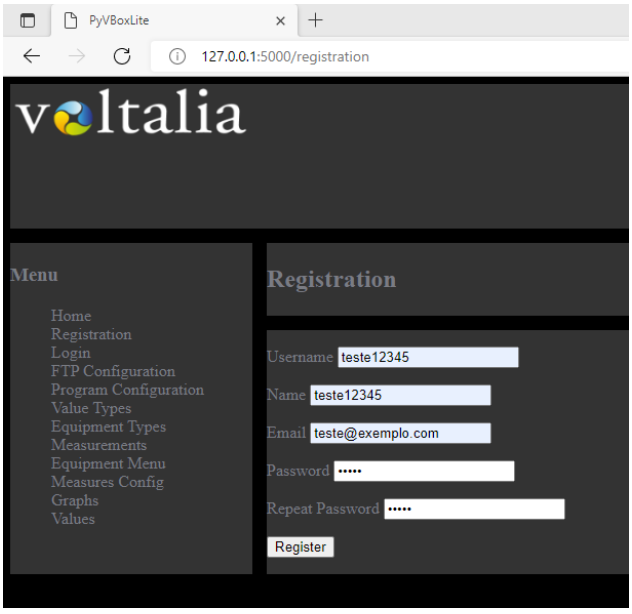


Figura 42 - Vista de registo

4.1.4 Recolha de dados

De modo a recolher os dados, terá de ser efetuada uma comunicação com o serviço de comunicação. Para isso foram criadas duas aplicações auxiliares. A primeira aplicação tem como objetivo a recolha dos dispositivos armazenados na BD. Visto que estes dados são recolhidos no formato de um dicionário e o seviço de comunicação necessita de valores individuais em forma de *string*, é necessário retirar a informação dos dicionários e organizá-la

de modo a ser introduzida na aplicação de comunicação. Na Figura 43 vê-se o exemplo dos ciclos implementados de forma a recolher os dados e armazená-los em variáveis que serão depois introduzidas nas funções da aplicação de comunicação. Estes ciclos terão de percorrer todos os valores recolhendo apenas os necessários, o primeiro ciclo *for* itera os equipamentos e *drivers*, o segundo trata de recolher todos os nomes das medidas recolhidas por este dispositivo para uma lista, o terceiro recolhe o período de agregação dos dados, os seguintes valores seguem este modelo, dependendo do que é necessário pelo serviço de comunicação.

A segunda aplicação trata de criar uma função cujo objetivo passa por armazenar informação na BD, usando um procedimento que insere os dados na tabela *pv_values* esta função substitui outra existente dentro da aplicação de comunicação que irá receber os valores dos dispositivos e estes são armazenados na BD, como se pode observar na Figura 44.

```
62          if eq == eq_t:
63              if driv == 2 or 3 or 4 or 5 or 6 or 7:
64                  for eq_2,name in equip_names:
65                      if eq_2 == eq_t:
66                          equip_name = name
67
68                  #print("equip_name: ", equip_name)
69
70                  header_list = []
71                  for eq_2, header in headers:
72                      if eq_2 == eq_t:
73                          header_list.append(header)
74                  #print("header_list: ", header_list)
75
76
77                  filePath = "C:/FTP"
78
79
80                  for eq_2,adp in agg_data_periods:
81                      if eq_2 == eq_t:
82                          agg_data_period = adp
83                  #print("agg_data_period: ", agg_data_period)
```

Figura 43 - Ciclos de organização dos dados a ser introduzidos na aplicação de comunicação

	v_id [PK] bigint	eq_id bigint	m_id integer	valu double precision	timest timestamp without time zone
1	2	7	8	99	2022-04-06 14:28:54
2	3	7	4	2	2022-04-06 14:28:54
3	4	7	9	101	2022-04-06 14:28:54
4	5	7	6	2	2022-04-06 14:28:54
5	6	7	6	0	2022-04-06 14:28:54
6	7	7	8	100	2022-04-06 14:28:57
7	8	7	4	2	2022-04-06 14:28:57
8	9	7	9	102	2022-04-06 14:28:57
9	10	7	6	2	2022-04-06 14:28:57
10	11	7	6	0	2022-04-06 14:28:57
11	12	7	8	100	2022-04-06 14:29:00
12	13	7	4	2	2022-04-06 14:29:00
13	14	7	9	101	2022-04-06 14:29:00

Figura 44 - Tabela *pv_values* com dados armazenados

4.1.5 Visualização dos dados

Existem duas formas de apresentação dos dados, a primeira passa por uma tabela, na qual estarão presentes todas as variáveis referentes ao dispositivo desejado. A segunda será sob a forma de gráficos, onde cada um dirá respeito a cada medição do dispositivo selecionado.

4.1.5.1 Apresentação dos dados sob a forma de tabelas

Para a criação de tabelas foi necessário a utilização de uma livraria auxiliar, a escolhida foi a *plotly*. Esta livraria permite a criação de tabelas, através de listas fornecidas pelo utilizador. Como mencionado anteriormente, a informação quando recolhida da BD é recebida, não só em formato de dicionário, mas também proveniente de diferentes vistas, fazendo com que se tenha vários dicionários com valores ao invés de uma lista. Para resolver esta situação, será necessário o uso de ciclos de modo a reorganizar os dados de forma a obter as listas com os valores necessários. Para recolher e organizar esta informação foram usadas cinco vistas, vista dos *ids* dos equipamentos, vista das chaves primárias da tabela *pv_values* em relação às chaves primárias dos equipamentos, vista dos *headers* das variáveis, vista dos valores obtidos e vista dos registos da data/hora (sendo que estas últimas 3 vistas são relacionadas com as chaves primárias da tabela *pv_values*).

Os ciclos criados, Figura 45, têm como objetivo recolher as chaves primárias dos valores medidos referentes ao equipamento que for selecionado. Uma vez obtidas estas chaves, vão ser executados os restantes ciclos de forma a obter as listas com informação necessária. No fim cria-se uma figura usando a livraria *plotly*, usando-se as listas e obtendo-se a tabela pretendida, tal como se ilustra na Figura 46.

```
891             cursor = connection.cursor()
892             cursor.execute("SELECT * FROM public.eq_id_view")
893             eq_ids = cursor.fetchall()
894             cursor.execute("SELECT * FROM public.pv_values_view")
895             v_ids = cursor.fetchall()
896             cursor.execute("SELECT * FROM public.m_id_view")
897             measures = cursor.fetchall()
898             cursor.execute("SELECT * FROM public.value_view")
899             values = cursor.fetchall()
900             cursor.execute("SELECT * FROM public.timest_view")
901             timests = cursor.fetchall()
902             cursor.close()
903
904             eq_id = tableval_form.eq_id.data
905
906             valu_temp = ()
907             val_ids = []
908             headers = []
909             vals = []
910             timestamps = []
911
912             for equ_id,v_id in v_ids:
913                 if str(equ_id) == str(eq_id):
914                     val_ids.append(v_id)
915             for val_id in val_ids:
916                 for va_id,me in measures:
917                     if val_id==va_id:
918                         headers.append(me)
919                 for va_id,v in values:
920                     if val_id==va_id:
921                         vals.append(v)
922                 for va_id,t in timests:
923                     if val_id==va_id:
924                         timestamps.append(t)
```

Figura 45 - Organização dos dados em listas

```
926
927             fig = go.Figure(data=[go.Table(header=dict(values=['header', 'value', 'timestamp']),
928                 cells=dict(values=[headers, vals, timestamps]))
929                     ])
930             fig.update_layout(paper_bgcolor="#333333")
931             fig.show()
932
```

Figura 46 - Criação da figura que apresentará as tabelas

4.1.5.1 Apresentação dos dados sob a forma de gráficos

A criação de gráficos foi efetuada recorrendo à livraria *matplotlib*, que permite a criação de uma elevada variedade de gráficos através de listas. No entanto, tal como no subcapítulo anterior, a informação recolhida da BD não se apresenta no formato desejado. Para tal, teve de se usar uma solução semelhante à anterior, de modo a reorganizar a informação proveniente da BD em listas. Contudo, ao contrário das tabelas, aqui pretende-se criar um gráfico para cada medida do dispositivo. Contudo, todos os valores referentes às medições estão presentes dentro da mesma lista. De modo a criar um gráfico para cada variável, é necessário separar estas medições em várias listas, uma para cada gráfico.

Para obter os dados necessários para os gráficos, foram usadas as seguintes listas: vista das chaves primárias dos equipamentos; vista que relaciona as chaves primárias dos equipamentos com as chaves primárias das medidas; vista que relaciona as chaves primárias das medidas com as chaves primárias da tabela *pv_values*; e por fim as vistas dos *headers* e dos valores recolhidos (relacionadas com as chaves primárias da tabela *pv_values*). De seguida criaram-se os ciclos, Figura 47, o objetivo destes passa por determinar as medidas relacionadas com o equipamento selecionado, obter as chaves primarias da tabela *pv_values* a que pertencem essas medidas, atribuir o título do gráfico e preencher a lista com as medidas necessárias. No fim cria-se o gráfico, Figura 48. Este processo será repetido para cada medição (no código demonstrado por *graph*) associada ao equipamento, criando os gráficos no fim do ciclo.

Uma vez que um dispositivo pode possuir uma elevada quantidade de variáveis, decidiu-se colocar todos os gráficos numa só figura, de modo a evitar a criação de várias figuras (que resultaria em várias janelas abertas). De modo a informar sistema de quantos gráficos terão de ser criados, é usada a função *subplot* do *matplotlib*. Nesta função tem de ser introduzido um valor inteiro que indica o número total de gráficos e disposição do gráfico atual, por exemplo se existir um total de três gráficos e o gráfico a ser criado neste momento for segundo, o valor introduzido seria de 132. Para determinar este valor, usada a dimensão da lista *graphs_n*, que possui as medidas associadas a cada dispositivo e o objetivo é criar um gráfico por medida. Visto que se usa esta mesma lista para iterar a criação de cada gráfico, o *index* desta lista determina o gráfico atual a ser criado. Usando o cálculo que pode ser visto na Figura 48 associado à variável *n_fi*, obtemos a informação necessária para ser introduzida na função *subplot*.

```
662                 cursor = connection.cursor()
663                 cursor.execute("SELECT * FROM public.eq_id_view")
664                 eq_ids = cursor.fetchall()
665                 cursor.execute("SELECT * FROM public.graphs_view")
666                 graphs = cursor.fetchall()
667                 cursor.execute("SELECT * FROM public.v_id_view")
668                 v_ids = cursor.fetchall()
669                 cursor.execute("SELECT * FROM public.csv_file_var_view")
670                 measures = cursor.fetchall()
671                 cursor.execute("SELECT * FROM public.value_view")
672                 values = cursor.fetchall()
673                 cursor.close()
674 
675                 eq_id = graph_form.eq_id.data
676 
677                 valu_temp = ()
678                 graphs_n = []
679                 val_ids = []
680                 vals = []
681 
682                 for equ_id,m_id in graphs:
683                     if str(equ_id) == str(eq_id):
684                         graphs_n.append(m_id)
685                 for graph in graphs_n:
686                     for m, va_id in v_ids:
687                         if m == graph:
688                             val_ids.append(va_id)
689                     for m, meas in measures:
690                         if m == graph:
691                             header = str(meas)
692                     for val_id in val_ids:
693                         for va_id,v in values:
694                             if val_id==va_id:
695                                 vals.append(float(v))
```

Figura 47 - Organização dos dados para os gráficos

```
682             for equ_id,m_id in graphs:
683                 if str(equ_id) == str(eq_id):
684                     graphs_n.append(m_id)
685             for graph in graphs_n:
686                 for m, va_id in v_ids:
687                     if m == graph:
688                         val_ids.append(va_id)
689                 for m, meas in measures:
690                     if m == graph:
691                         header = str(meas)
692                 for val_id in val_ids:
693                     for va_id,v in values:
694                         if val_id==va_id:
695                             vals.append(float(v))
696 
697                 n1 = len(graphs_n)
698                 n2 = graphs_n.index(graph) + 1
699                 n_fi = 100 + 10 * n1 + n2
700                 plt.subplot(n_fi)
701                 plt.plot(vals)
702                 plt.title(header)
703                 plt.grid(True)
704                 vals = []
705                 val_ids = []
706 
707             plt.show()
```

Figura 48 - Criação dos gráficos

4.2 Testes ao sistema

Nesta subseçãocapítulo apresentam-se as diferentes interações que um utilizador poderá ter com este sistema. Para tal procedeu-se à criação de um utilizador, à introdução de um equipamento real e à análise das medições do equipamento.

4.2.1 Criação de um utilizador

Para efetuar a criação de um utilizador, foi usada a vista de registo, que pode ser observada na Figura 47, juntamente com os dados que foram utilizados para criar este utilizador. Uma vez efetuado este registo, os dados são enviados para a BD. Como mencionado no capítulo 3.3.1, existe um nível de segurança, de modo a proteger a *password* de cada utilizador. De modo a verificar que, não só os dados estão a ser inseridos corretamente, como o *hash* da *password* está também a ser efetuado, foi usado o *pgAdmin*. Aqui acedeu-se á tabela dos utilizadores, Figura 48, e observou-se o conteúdo da mesma. Na linha 4, evidenciada pelo sublinhar vermelho, pode-se observar o utilizador que foi criado. O *hash* da *password*, como se pode observar, foi efetuado, transformando “12345” num conjunto de caracteres embaralhados. Também se pode observar a chave estrangeira *p_l_id*, chave esta que representa a tabela *permission_levels*, aonde

estão presentes os níveis de permissão dos utilizadores. Quando é criado um novo utilizador, este recebe o *p_l_id* de 3, que é equivalente ao nível *visitor*, como se pode observar na Figura 49.

Data Output | Explain | Messages | Notifications

	u_id [PK] integer	p_l_id integer	username character varying (50)	passwrd character varying (200)	nme character varying (100)	email character varying (100)
1	5	3	dimelo	pbkdf2:sha256:150000$9AwQKNCg$f9fe9f0bc2743120b566d301c2952127375b345e482f1de969b541520de85777	Diogo Almeida	dimelo@smth.com
2	6	3	dimelo2	pbkdf2:sha256:150000$GRUM02lj$5acd8e2d72b79bf44dc3ea4af164382a82f516dae650060dc88d76e9b7bf0dde	Diogo Almeida	dimelo2@smth.com
3	7	3	dimelo3	pbkdf2:sha256:150000$GycOygBS$d4d2e183db88a4a533d6be3a7371f49085aced3daa5e331c5c56b208afd83719	Diogo Almeida	dimelo3@smth.com
4	8	3	teste12345	pbkdf2:sha256:150000$2TqK6tWt$2a47044e6d75d27869cc5824b1e1814a73b06c49820ca87d750cc48f8a3380c0	teste12345	teste@exemplo.com

Figura 49 - Tabela *users* preenchida

	p_l_id [PK] integer	p_l_descr character varying (100)
1	1	administrator
2	2	operator
3	3	visitor

Figura 50 - Tabela *permission_levels*

4.2.2 *Login* do utilizador

O próximo passo foi testar o sistema de *login* do sistema. Verificar se quando o utilizador introduz a *password* correta, o sistema é capaz de a reconhecer e associar à informação que está armazenada na BD. Para tal acedeu-se à vista de *login*, Figura 50, aqui, mais uma vez, foram introduzidos os dados do nosso utilizador teste. De modo a verificar se o login foi efetuado, não só se apresenta as imagens da página, Figura 50, como também *prints* do código, Figura 51, de forma a ter uma melhor perceção do que se passa dentro do código. Nestes *prints*, podemos observar que quando a função *check_password_hash* confirma que a *password* está correta, devolve o valor *True*. Na Figura 51, pode-se também observar o tipo de dicionário que é obtido da BD, como se pode observar, apenas se recebe os *emails* e as *passwords*, isto graças às *views* do *PostgreSQL*.

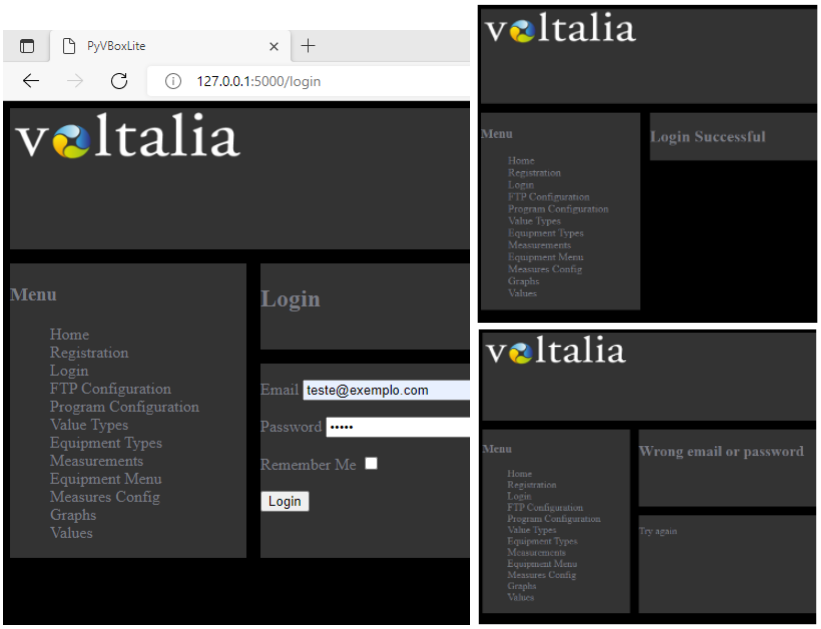


Figura 51 - Vista de *Login*

```
[('dimelc@smth.com', 'pbkdf2:sha256:150000$9AwQKNCg$f9fe9f0bc2743120b566d301c2952127375b345e482f1de969b541520de85777'), ('dimelo2@smth.com', '
pbkdf2:sha256:150000$GRUM02Ij$5acd8e2d72b79bf44dc3ea4af164382a82f516dae650060dc88d76e9b7bf0dde'), ('dimelo3@smth.com', 'pbkdf2:sha256:150000$G
ycOygBS$d4d2e183db88a4a533d6be3a7371f49085aced3daa5e331c5c56b208afd83719'), ('teste@exemplo.com', 'pbkdf2:sha256:150000$2TqK6tWt$2a47044e6d75d
27869cc5824b1e1814a73b06c49820ca87d750cc48f8a3380c0')]

teste@exemplo.com

12345

Login successful
```

```
teste@exemplo.com

1234

Login failed
```

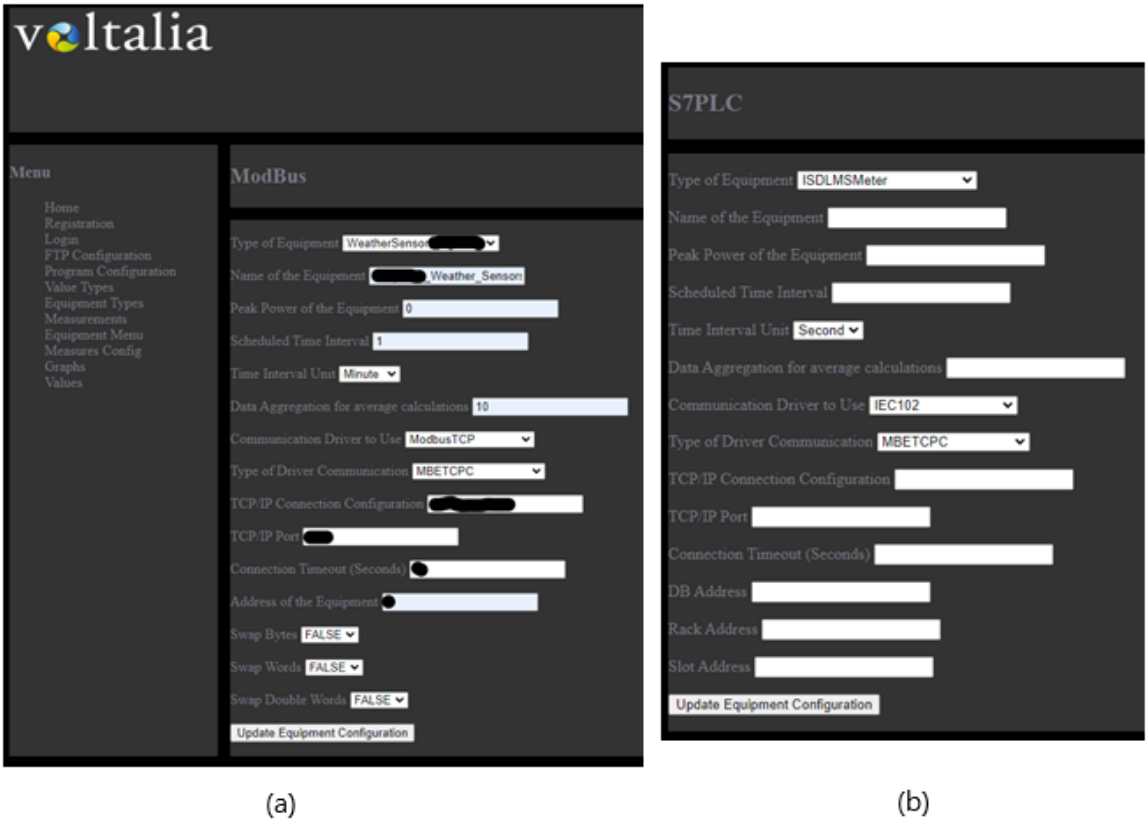
Figura 52 - Resultados do *Login*

4.2.3 Registo de um equipamento

Neste subseção, procura-se introduzir um equipamento, atualmente em funcionamento numa central fotovoltaica situada no sul da Europa. Para tal é necessário seguir alguns passos na inserção deste dispositivo. Primeiro é preciso verificar se a informação relativa ao dispositivo existe na BD. Quando o sistema comunica com um equipamento, é necessário informar não só das características da comunicação do equipamento, mas também a informação das medidas que são registadas pelo mesmo. Após uma verificação, conclui-se que não existia o tipo de

equipamento na BD, bem como as medidas e as suas configurações. Portanto, primeiro inseriu-se o tipo de equipamento na BD, de seguida efetuou-se a configuração do equipamento. Para tal acedeu-se à vista da configuração dos equipamentos, após selecionar o *driver* utilizado pela comunicação (que neste caso foi de *ModBus*), obtém-se a Figura 52 (a), aonde se pode observar a informação que foi introduzida. Visto que cada *driver* requer informação diferente, o formulário que é apresentado varia, conforme podemos observar entre (a) e (b) da Figura 52. Aqui é recolhida toda a informação necessária para que a aplicação de comunicação consiga realizar a comunicação o dispositivo. Como se pode observar estão presentes os *dropdowns*, conforme mencionado em 3.2.2.3, este processo impede que o utilizador introduza a informação manualmente, evitando erros de escrita e a duplicação de valores na BD.

De seguida configurou-se o tipo de valor, a medida e a configuração de cada medida (que no caso deste dispositivo como são obtidas quatro medidas, tem de se efetuar quatro configurações para cada medida), Figura 53. Como se pode observar, as configurações das medidas informam à aplicação de comunicação, não só de informações referentes à comunicação, mas também do tipo de cálculo que é efetuado de modo a obter o valor final.



(a) (b)

Figura 53 - Configuração de equipamentos

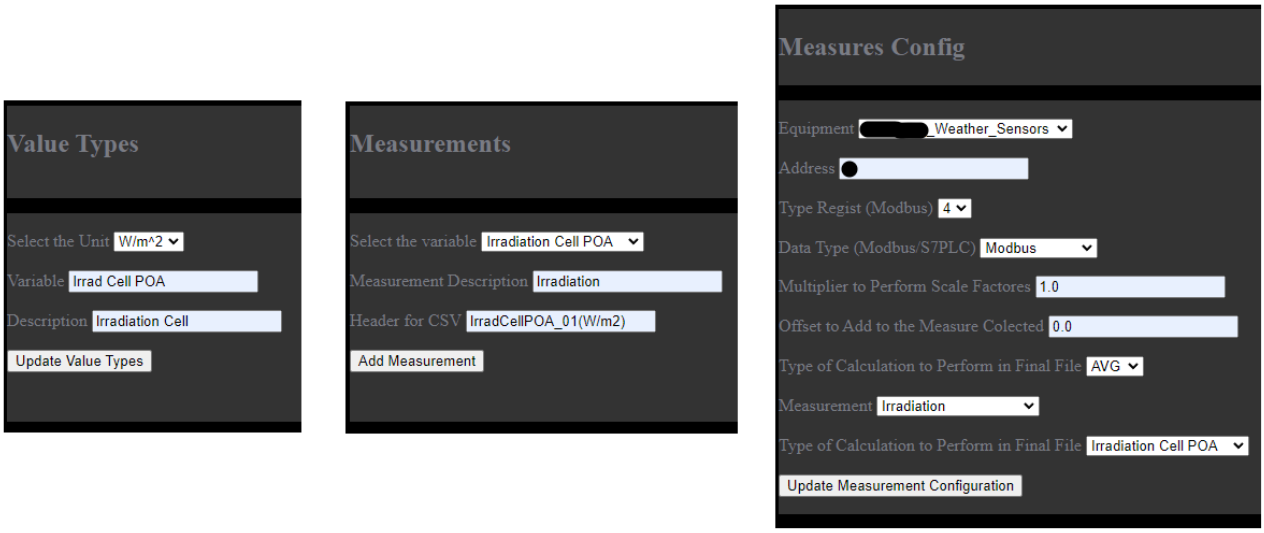


Figura 54 - Configuração das medidas

4.2.4 Conexão com a aplicação do serviço de comunicação

Conforme mencionado no subcapítulo 3.3.2, a comunicação com o serviço de comunicação é efetuada numa aplicação diferente. Isto deve-se ao facto desta comunicação ter de ser efetuada mesmo que a aplicação principal não esteja em funcionamento, caso contrário existiriam quebras na recolha dos dados. De modo a obter uma quantidade aceitável de dados para apresentar neste capítulo, foram recolhidos dados do sensor de irradiação a cada dez minutos durante oito horas, desde o meio-dia até às oito da noite. Assim pode-se obter uma curva da evolução da irradiação que incidiu sobre os painéis neste dia específico.

Para que a aplicação corra continuamente durante oito horas, criou-se um *Schedule*, usando mais uma vez a livraria *apscheduler*, Figura 54. Criou-se um *job* que irá efetuar a comunicação com a aplicação de comunicação, executando as funções necessárias à comunicação a cada dez minutos. Estes dados foram armazenados na tabela *pv_values*, Figura 55.

```
179    sched.add_job(job1, 'interval', minutes=10, start_date='2022-05-04 12:10:00', end_date='2022-05-04 20:10:00')
```

Figura 55 - *Schedule* criado para recolher os valores

	v_id [PK] bigint	eq_id bigint	m_id integer	valu double precision	timest timestamp without time zone
1	37	7	8	975	2022-05-04 11:10:00
2	38	7	4	2	2022-05-04 11:10:00
3	39	7	9	860	2022-05-04 11:10:00
4	40	7	6	2	2022-05-04 11:10:00
5	41	7	6	0	2022-05-04 11:10:00
6	42	7	8	1152	2022-05-04 11:20:00
7	43	7	4	2	2022-05-04 11:20:00
8	44	7	9	1007	2022-05-04 11:20:00
9	45	7	6	2	2022-05-04 11:20:00
10	46	7	6	0	2022-05-04 11:20:00
11	47	7	8	802	2022-05-04 11:30:00
12	48	7	4	2	2022-05-04 11:30:00
13	49	7	9	711	2022-05-04 11:30:00

Figura 56 - Tabela *pv_values* com os valores recolhidos do sensor

4.2.5 Apresentação dos resultados

Para apresentar os resultados são usados dois métodos, uma tabela que apresenta todos os valores armazenados de cada dispositivo e gráficos que dizem respeito a cada medição do dispositivo. Visto que no subcapítulo anterior recolhemos oito horas de dados de um sensor, aqui serão usados esses dados para construir a tabela e os gráficos.

4.2.5.1 Tabelas

Para a visualização de tabelas, como foi discutido no capítulo 3.3.3.1, é criada uma tabela que recebe todas as variáveis fornecidas pelo dispositivo. Para aceder a esta tabela é necessário selecionar a vista dos *Values*, Figura 56. Aqui seleciona-se o dispositivo pretendido e o sistema irá percorrer os ciclos criados de forma a recolher e organizar a informação da BD de forma a criar a tabela com os valores finais, que pode ser observada na Figura 57. Nesta tabela pode-se observar as medições efetuadas do sensor, à célula de irradiação referente ao *Plan of Array* (*POA*), irradiação no plano dos módulos (quando estes têm a mesma inclinação e a mesma orientação). A célula de irradiação referente ao *Global horizontal irradiation* (*GHI*), irradiação incidida sob uma superfície horizontal. Por fim, o estado do equipamento.

Figura 57 - Seleção do dispositivo para a tabela

header	value	timestamp
IrradCellPOA_01(W/m2)	975	2022-05-04T11:10:00
Equip_Status_01	2	2022-05-04T11:10:00
IrradCellGHI_02(W/m2)	860	2022-05-04T11:10:00
Equip_Status_02	2	2022-05-04T11:10:00
Equip_Status_02	0	2022-05-04T11:10:00
IrradCellPOA_01(W/m2)	1152	2022-05-04T11:20:00
Equip_Status_01	2	2022-05-04T11:20:00
IrradCellGHI_02(W/m2)	1007	2022-05-04T11:20:00
Equip_Status_02	2	2022-05-04T11:20:00
Equip_Status_02	0	2022-05-04T11:20:00
IrradCellPOA_01(W/m2)	802	2022-05-04T11:30:00
Equip_Status_01	2	2022-05-04T11:30:00
IrradCellGHI_02(W/m2)	711	2022-05-04T11:30:00
Equip_Status_02	2	2022-05-04T11:30:00
Equip_Status_02	0	2022-05-04T11:30:00
IrradCellPOA_01(W/m2)	272	2022-05-04T11:40:00
Equip_Status_01	2	2022-05-04T11:40:00
IrradCellGHI_02(W/m2)	253	2022-05-04T11:40:00
Equip_Status_02	2	2022-05-04T11:40:00
Equip_Status_02	0	2022-05-04T11:40:00
IrradCellPOA_01(W/m2)	1180	2022-05-04T11:50:00
Equip_Status_01	2	2022-05-04T11:50:00
IrradCellGHI_02(W/m2)	1008	2022-05-04T11:50:00
Equip_Status_02	2	2022-05-04T11:50:00

Figura 58 - Tabela criada com as medições do sensor

4.2.5.2 Gráficos

A visualização dos gráficos é possível através da vista *graphs*, aonde é selecionado o equipamento pretendido, neste caso será usado o sensor que foi adicionado ao sistema. Como mencionado no capítulo 3.3.3.2, uma vez que o equipamento seja selecionado, o sistema irá percorrer uma série de ciclos, de modo a recolher e organizar os dados. No entanto, como se pode observar, duas das variáveis recolhidas não são apresentadas, o estado do equipamento. Isto deve-se ao facto da apresentação desta informação sob o formato de gráfico não ser relevante. Nos gráficos, Figura 58, pode-se observar a evolução da irradiação deste sensor em particular durante uma tarde de maio. O eixo dos xx destes gráficos observa-se a data das medições no formato dia-mês-hora. No eixo dos yy temos o valor da irradiação em W/m^2.

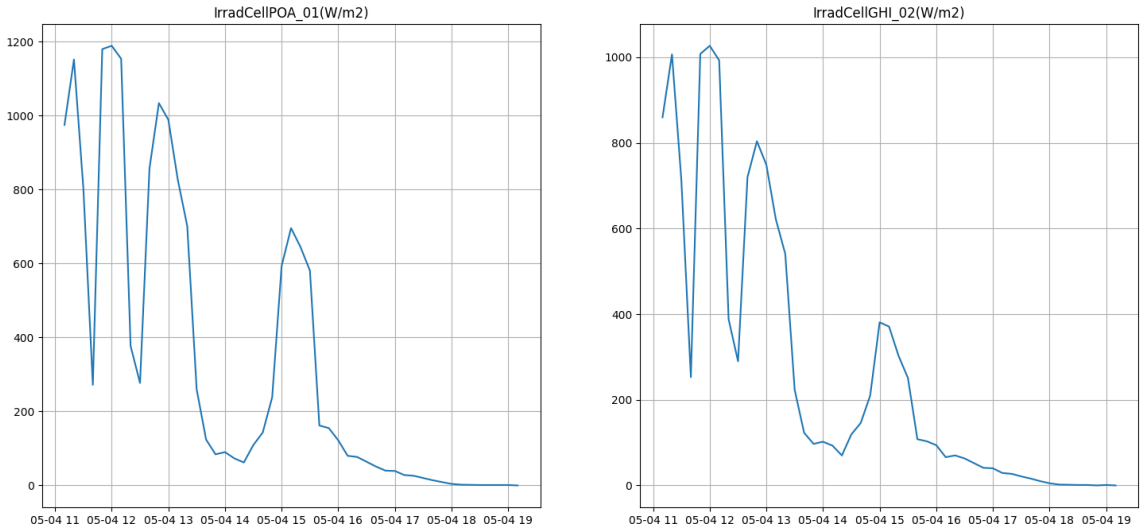


Figura 59 - Gráficos criados com as medidas registadas do sensor

4.2.6 Análise dos resultados

Nesta subseção analisa-se os resultados obtidos do sensor de irradiação. Para tal compara-se com um gráfico da irradiação esperada na mesma altura do ano para a mesma região, que se pode observar na Figura 59. O primeiro pormenor que mais se evidencia são as oscilações dos gráficos da Figura 58. Consultando a previsão do tempo do mesmo dia da região da central, Figura 60, podemos observar que foi um dia nublado, o que resulta em sombreados que

diminuem a intensidade da irradiação, conforme compravam os gráficos. Apesar disto, pode-se verificar que a evolução da curva é relativamente próxima do que seria esperado.

Com base nestes dados obtidos, pode-se determinar uma aproximação dos níveis de produção que foram obtidos neste dia. Caso esses níveis não estejam próximos desta previsão, pode-se concluir que existe algum problema na produção. Se este for o caso, envia-se uma equipa com o objetivo de reparar potenciais problemas.

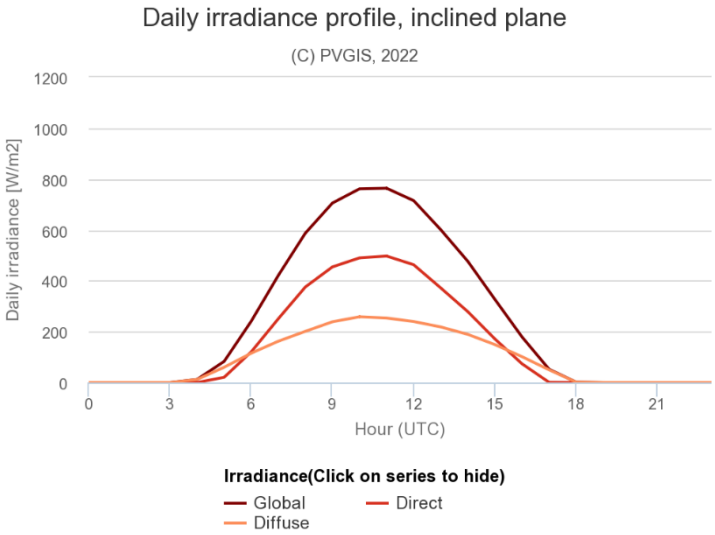


Figura 60 - Previsão da irradiação [42]

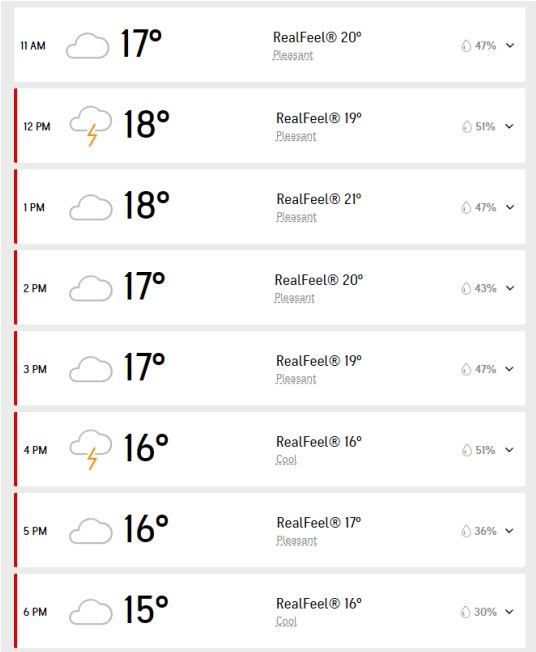


Figura 61 - Previsão do tempo [43]

Por fim, analisa-se o espaço ocupado por estas oito horas de recolha de dados, Figura 62. Foram ocupados 112 KB de memória, sendo recolhidas 240 linhas de medições. Imagine-se que existem vinte dispositivos ligados a este sistema a recolherem o mesmo tipo de medições. Pode-se supor que serão recolhidas 18480 linhas nas mesmas oito horas, 55440 linhas num dia inteiro. Nos testes realizados no capítulo 3.2.2.1, o teste com dez variáveis armazenado em dez colunas (a tabela *pv_values* possui cinco colunas) ocupou 2,352 MB em 17281 linhas. Supondo que a tabela com vinte dispositivos ocupa cerca do triplo do espaço, por dia seriam ocupados cerca de 7 MB, o que resultaria em 211 MB por mês (mais ou menos 2,6 GB por ano). Visto que o sistema terá um espaço total de 64 GB, existe uma grande margem para o armazenamento dos dados.

Figura 62 - Espaço ocupado pela tabela dos valores recolhidos

5. Conclusões

A recolha de dados referente ao funcionamento de uma central fotovoltaica é crucial nos tempos modernos, não só para avaliar a performance da mesma, mas também para prever as suas produções. Uma central equipada com este sistema deixa de necessitar de sistemas terceiros, para efetuar recolha de dados, sendo que estes são muito mais dispendiosos. Assim, o custo quer de manutenção quer de operação irá ser mais reduzido enquanto se garante uma recolha de dados eficiente. Assim, é possível efetuar reparações de possíveis avarias, maximizando a produtividade e a aumentando a prevenção de eventuais problemas.

5.1 Trabalho desenvolvido

Foi possível desenvolver um sistema de baixo custo capaz de recolher dados de equipamentos presentes em centrais fotovoltaicas.

O sistema desenvolvido foi capaz de realizar uma comunicação eficiente com uma BD robusta, capaz de receber elevados volumes de dados e pronta para comunicações eficientes com outras aplicações. Este apresenta diversos recursos que, não só facilitam a comunicação com suavizam o armazenamento e visualização dos dados.

Desenvolveu-se também uma aplicação informática capaz, não só, de comunicar com a BD, armazenando e recolhendo dados, como também comunicar com o serviço de comunicação desenvolvido pela Voltalia. Através destes sistemas, é possível a introdução de novos equipamentos que sejam acrescentados à central. Tudo isto mantendo o sistema flexível. Caso sejam introduzidos novos drivers no sistema de comunicação, estes poderão ser configurados no sistema desenvolvido, oferecendo a este uma elevada flexibilidade. Tendo a Voltalia acesso

à totalidade do código, oferece-lhes também uma maior liberdade para a resolução de eventuais problemas que possam surgir.

Foi desenvolvida também uma *interface WEB*, seguindo o modelo de outras desenvolvida pela Voltalia. Esta oferece ao utilizador meios de configuração de equipamentos ou mesmo do próprio programa. Bem como a apresentação dos dados, sejam estes em forma de tabela e/ou gráfico.

Foi possível apresentar medições obtidas de um sensor conectado a uma central no sul da Europa. Demonstrando um pouco a potencialidade deste sistema, através da análise dos dados obtidos. Foi possível observar as variações de produção devido às condições climatéricas que se apresentavam nesse dia.

No entanto ocorreram vários desafios, desde o tempo que levou à criação da BD, devido à falta de experiência na área, bem como o nível de exigência da aplicação informática, a qual era necessário uma comunicação e interligação eficaz com o serviço de comunicação.

5.2 Trabalhos futuros

Este trabalho deixou espaço para diversos trabalhos futuros. De modo a garantir uma aplicação que esteja ao nível das atuais do mercado, será necessário implementar diversos recursos, como por exemplo:

- Uma gestão de alarmes eficaz, a aplicação de comunicação atualmente envia erros em forma de códigos, mas com os recursos atuais, poderia ser feito uma aplicação que fosse capaz de enviar relatórios de alarmes ao invés de códigos, de modo a obter uma informação mais detalhada sobre potenciais falhas;

- Melhoramento do sistema de *login*, durante o desenvolvimento da aplicação, tentou-se usar a biblioteca *flask_login*, no entanto ocorreram diversos erros na validação do utilizador, pelo que este método acabou por ser descartado;

- Introdução de *Widgets*, o uso destes na página inicial por exemplo torna possível o fornecimento de informação adicional, sem ter de navegar entre páginas para obter o mesmo;

- Implementação de um *FTP* de modo a puder atualizar e apresentar os dados em tempo real;

- Implementação de um algoritmo de inteligência artificial para verificação da qualidade de dados recolhida, utilizando aprendizagem automática (ou *machine learning*);

- Implementar uma *interface* com o mapa de calor (*Heat Map*) da disponibilidade de cada equipamento.

REFERÊNCIAS

[1] A. V. da Cruz Granja, “Estudo e Optimização de Central Fotovoltaica 1MW,” 2017.

[2] R. R. P. dos Santos, “Monitorização de desempenho e deteção de anomalias em centrais fotovoltaicas,” 2014.

[3] S. R. Madeti and S. N. Singh, “Monitoring system for photovoltaic plants: A review,” *Renewable and Sustainable Energy Reviews*, vol. 67, pp. 1180–1207, Jan. 2017, doi: 10.1016/j.rser.2016.09.088.

[4] M. M. Rahman, J. Selvaraj, N. A. Rahim, and M. Hasanuzzaman, “Global modern monitoring systems for PV based power generation: A review,” *Renewable and Sustainable Energy Reviews*, vol. 82, no. P3, pp. 4142–4158, 2018.

[5] S. Ansari, A. Ayob, M. S. H. Lipu, M. H. M. Saad, and A. Hussain, “A Review of Monitoring Technologies for Solar PV Systems Using Data Processing Modules and Transmission Protocols: Progress, Challenges and Prospects,” *Sustainability*, vol. 13, no. 15, Art. no. 15, Jan. 2021, doi: 10.3390/su13158120.

[6] B. Andò, S. Baglio, A. Pistorio, G. M. Tina, and C. Ventura, “Sentinella: Smart Monitoring of Photovoltaic Systems at Panel Level,” *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 8, pp. 2188–2199, Aug. 2015, doi: 10.1109/TIM.2014.2386931.

[7] M. Anwari, M. M. Dom, and M. Rashid, “Small Scale PV Monitoring System Software Design,” 2011, doi: 10.1016/J.EGYPRO.2011.10.079.

[8] G. Bayrak and M. Cebeci, “Monitoring a grid connected PV power generation system with labview,” in *2013 International Conference on Renewable Energy Research and Applications (ICRERA)*, Oct. 2013, pp. 562–567. doi: 10.1109/ICRERA.2013.6749819.

[9] A. N. Pereira, P. Tomé, P. M. Costa, and J. Pascoal, “Architecture of information system for monitoring of photovoltaic plants,” in *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, Jun. 2014, pp. 1–6. doi: 10.1109/CISTI.2014.6876863.

[10] A. P. N. Pereira, “Plataforma gráfica de monitorização de produção FV para sistemas móveis Android,” masterThesis, Instituto Politécnico de Viseu. Escola Superior de Tecnologia e Gestão de Viseu, 2014. Accessed: Mar. 07, 2022. [Online]. Available: https://repositorio.ipv.pt/handle/10400.19/2529

[11] D. Correia, P. Tomé, P. M. Costa, and L. Marques, “Monitoring system for small sized photovoltaic power plants,” in *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, Jun. 2016, pp. 1–6. doi: 10.1109/CISTI.2016.7521454.

[12] D. L. T. da C. Correia, “Sistema de monitorização para instalações fotovoltaicas de pequena dimensão,” masterThesis, Instituto Politécnico de Viseu. Escola Superior de Tecnologia e Gestão de Viseu, 2016. Accessed: Mar. 07, 2022. [Online]. Available: https://repositorio.ipv.pt/handle/10400.19/4039

[13] S.-L. GmbH, “Monitoring,” Mar. 11, 2019. https://www.solar-log.com/en/solutions-service/monitoring/ (accessed Mar. 07, 2022).

[14] “SolarLog_Manual_3x_EN.pdf.” Accessed: Mar. 07, 2022. [Online]. Available: https://www.solar-log.com/manuals/manuals/en_GB/SolarLog_Manual_3x_EN.pdf

[15] “SolarLog Base 15,” *mg-solar-shop*, 2022. https://www.mg-solar-shop.com/solarlog-base-15 (accessed Mar. 07, 2022).

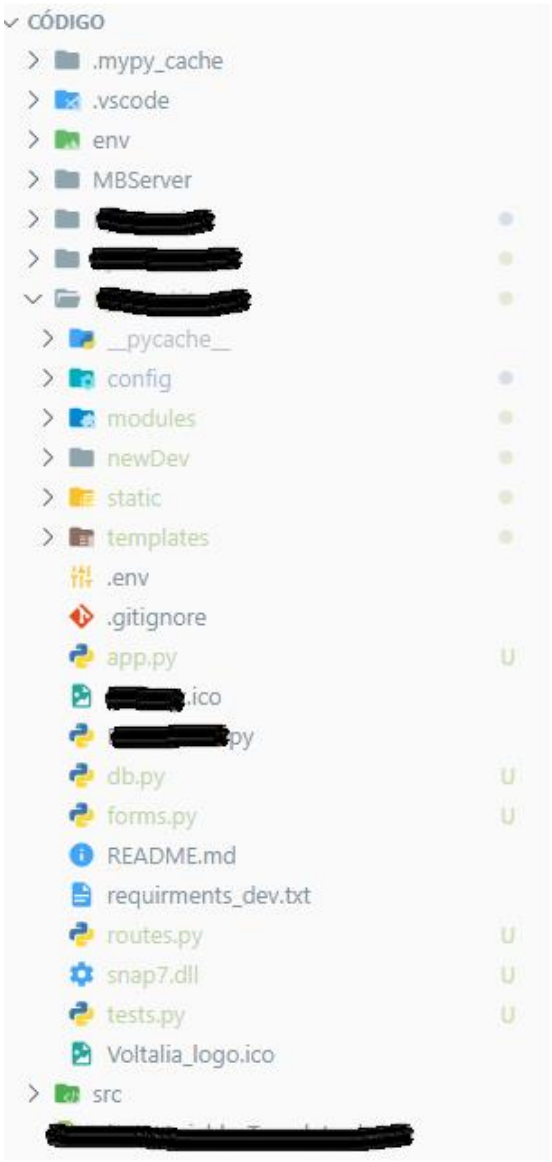
[16] “SolarLog Base 2000,” *mg-solar-shop*, 2022. https://www.mg-solar-shop.com/solarlog-base-2000 (accessed Mar. 21, 2022).

[17] meteocontrol, “meteocontrol,” 2022. https://www.meteocontrol.com/en/ (accessed Mar. 07, 2022).
[18] “DB_blueLog_XM_XC_en.pdf.” Accessed: Mar. 07, 2022. [Online]. Available: https://www.meteocontrol.com/fileadmin/Daten/Dokumente/EN/1_Photovoltaik_Monitoring/1_Produkte/blue_Log_XM_XC/DB_blueLog_XM_XC_en.pdf
[19] “Monitoring & maintenance meteocontrol GmbH - XC-200 Parkregler / plant controller,” *Secondsol der Photovoltaik Marktplatz*, 2022. httpshttps://www.secondsol.com/en/anzeige/24336/accessories/monitoring/monitoring-maintenance/meteocontrol-gmbh/xc-200-parkregler-plant-controller (accessed Mar. 07, 2022).
[20] “M2M Monitoring and IoT solutions provider,” *Webdyn*. https://www.webdyn.com/ (accessed Mar. 21, 2022).
[21] “WebdynSun,” *Webdyn*. https://www.webdyn.com/product/webdynsun/ (accessed Mar. 07, 2022).
[22] “Webdynsun Data Loggers,” *indiamart.com*. https://www.indiamart.com/proddetail/webdynsun-data-loggers-20316222097.html (accessed Mar. 07, 2022).
[23] S. S. T. AG, “SMA Solar Technology AG - Inverter & Photovoltaics solutions.” https://www.sma.de/en.html (accessed Apr. 28, 2022).
[24] “SPortal-WB-CC-BA-US_en-27.pdf.” Accessed: Mar. 07, 2022. [Online]. Available: https://www.sunnyportal.com/Documents/SPortal-WB-CC-BA-US_en-27.pdf
[25] “SMA Cluster Controller,” 2021. https://www.europe-solarstore.com/sma-cluster-controller.html?gclid=CjwKCAjwq9mLBhB2EiwAuYdMtQNNxp0b43mCvluy0x0edShvU61Nyu3wQbXC2_nPgUQPTcZd4NecUBoCdJ0QAvD_BwE (accessed Mar. 07, 2022).
[26] “SmartLogger_3000A_manual.pdf.” Accessed: Apr. 28, 2022. [Online]. Available: https://autosolar.es/pdf/SmartLogger_3000A_manual.pdf
[27] “Huawei NET ECO Price - Huawei Price List 2022,” 2022. https://itprice.com/huawei-price-list/net%20eco.html (accessed Mar. 07, 2022).
[28] “HUAWEI Smart Logger 3000A03,” 2021. https://www.europe-solarstore.com/huawei-smart-logger-3000a03.html (accessed Apr. 28, 2022).
[29] “HUAWEI Smart Logger 3000A01,” 2021. https://www.europe-solarstore.com/huawei-smart-logger-3000a01.html (accessed Apr. 28, 2022).
[30] “The World’s Most Bankable Inverter Brand | SUNGROW HOME,” 2022. https://en.sungrowpower.com/ (accessed Mar. 21, 2022).
[31] “Sungrow-data-logger-1000_1000B-usert-manual.pdf.” Accessed: Mar. 07, 2022. [Online]. Available: https://www.europe-solarstore.com/download/Sungrow/Sungrow-data-logger-1000_1000B-usert-manual.pdf
[32] “Sungrow SGR-Logger1000,” *Phenix LED Saves*. https://ledsaves.org/product/sungrow-sgr-logger1000/ (accessed Mar. 07, 2022).
[33] “Software - Industrial monitoring and remote control - IoT - Industry 4.0,” Jul. 21, 2020. https://www.higeco.com/software/ (accessed Mar. 07, 2022).
[34] “Prodotti - Monitoraggio e telecontrollo industriale - IoT - Industria 4.0,” Jul. 20, 2020. https://www.higeco.com/prodotti/ (accessed Mar. 07, 2022).
[35] “RASPBERRY PI Microcomputador Raspberry Pi 4 Model B 1.5GHz 2GB - com WiFi 2.4/5GHz + Bluetooth 5.0 + PoE.” https://mauser.pt/catalog/product_info.php?products_id=096-7401 (accessed Jun. 08, 2022).

[36] P. G. D. Group, “PostgreSQL,” *PostgreSQL*, Mar. 07, 2022. https://www.postgresql.org/ (accessed Mar. 07, 2022).
[37] “PostgreSQL: About,” 2022. https://www.postgresql.org/about/ (accessed Mar. 07, 2022).
[38] “PostgreSQL,” *Wikipedia*. Mar. 04, 2022. Accessed: Mar. 07, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=PostgreSQL&oldid=1075196265
[39] “MongoDB: The Application Data Platform,” *MongoDB*, 2021. https://www.mongodb.com (accessed Mar. 07, 2022).
[40] “MongoDB,” *Wikipedia*. Jan. 11, 2022. Accessed: Mar. 07, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MongoDB&oldid=1065102604
[41] “Quickstart — Flask Documentation (2.0.x).” https://flask.palletsprojects.com/en/2.0.x/quickstart/ (accessed Mar. 09, 2022).
[42] “JRC Photovoltaic Geographical Information System (PVGIS) - European Commission,” Oct. 15, 2019. https://re.jrc.ec.europa.eu/pvg_tools/en/#DR (accessed May 04, 2022).
[43] “Local, National, & Global Daily Weather Forecast | AccuWeather,” 2021. https://www.accuweather.com/ (accessed May 04, 2022).

APÊNDICES

APÊNDICE 1 - ESTRUTURA DA APLICAÇÃO



APÊNDICE 2 – ESTRUTURA DA BASE DE DADOS