



**Politécnico
Castelo Branco**

Escola Superior
de Tecnologia

Sistema inteligente de visão computacional para inspeção de centrais fotovoltaicas

Projeto II

João Filipe Campos Cabrito

20171417

Orientadores

João Manuel Leitão Pires Caldeira

Vasco Nuno da Gama de Jesus Soares

Trabalho apresentado à Escola Superior de Tecnologia de Castelo Branco do Instituto Politécnico de Castelo Branco, realizado sob a orientação científica do Doutor João Manuel Leitão Pires Caldeira e coorientação do Doutor Vasco Nuno da Gama de Jesus Soares, do Instituto Politécnico de Castelo Branco.

Junho 2026

Composição do júri

Presidente do júri

Ângela Cristina Marques de Oliveira Professor Adjunto do Instituto Politécnico de Castelo Branco

Vogais

Doutor João Manuel Leitão Pires Caldeira Professor Adjunto do Instituto Politécnico de Castelo Branco

Doutora, Mónica Isabel Teixeira da Costa Professor Adjunto do Instituto Politécnico de Castelo Branco

Agradecimentos

A concretização deste trabalho não teria sido possível sem o apoio e o contributo de várias pessoas, a quem deixo o meu sincero reconhecimento.

Agradeço, em primeiro lugar, ao meu orientador, Professor Doutor João Manuel Leitão Pires Caldeira, e ao meu coorientador, Professor Doutor Vasco Nuno da Gama de Jesus Soares, pela orientação científica, pela disponibilidade e pelo rigor com que acompanharam todas as fases deste projeto. As suas sugestões e o seu sentido crítico foram determinantes para superar as dificuldades e para a qualidade do trabalho desenvolvido.

Estendo este agradecimento a todos os professores da Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco com quem me cruzei ao longo deste percurso, pelos conhecimentos transmitidos e pela formação que serviu de base à realização deste trabalho.

À minha família, e em particular aos meus pais, agradeço o apoio incondicional, a confiança depositada e os valores que sempre me transmitiram, sem os quais não teria chegado até aqui.

À minha namorada, agradeço a paciência, o incentivo constante e a presença em todos os momentos, sobretudo nos mais exigentes.

Aos meus amigos, agradeço o companheirismo, a motivação e os momentos de descontração que foram essenciais para manter o equilíbrio ao longo desta etapa.

A todos, o meu mais sincero obrigado.

Resumo

A energia solar fotovoltaica tem assumido um papel crescente na transição energética, o que intensifica a necessidade de processos de operação e manutenção mais escaláveis. A inspeção visual tradicional dos módulos é morosa, dependente da interpretação humana e de difícil aplicação frequente em centrais de grande dimensão, sobretudo na deteção de sujidade que reduz a captação de radiação solar. Em continuidade com o Projeto I, no qual se estudaram e avaliaram experimentalmente abordagens de visão computacional para deteção e segmentação de sujidade em imagens RGB de painéis fotovoltaicos, o presente trabalho centra-se na implementação e validação funcional de um protótipo demonstrador.

O objetivo foi desenvolver uma plataforma distribuída que integra a aquisição de imagens no *edge*, a ingestão em *PV*, o processamento assíncrono baseado em *jobs*, a inferência com os modelos YOLOv8x e DINOv2, o mapeamento espacial das deteções sobre o *layout* da planta e a visualização operacional numa aplicação web. A componente de aquisição foi concebida para Raspberry Pi 4 Model B com Camera Module 3, incorporando mecanismos de *spool*, *checksum*, *retries* e idempotência no envio das capturas. A persistência e o cruzamento espacial entre deteções, zonas e painéis recorrem a PostgreSQL/PostGIS, sendo os resultados convertidos em níveis de severidade e em *tickets* de apoio à decisão operacional. A arquitetura foi concebida como *cloud-ready*, preparada para serviços Microsoft Azure, sendo a demonstração executada localmente em contentores Docker.

A validação, de natureza funcional e realizada em ambiente controlado, confirmou a viabilidade do fluxo *end-to-end*, desde a captura ou submissão da imagem até à apresentação dos resultados. O sistema é apresentado como prova de conceito e não como produto final, identificando-se como trabalho futuro a recolha sistemática de imagens em terreno real, o reforço do conjunto de dados e a consolidação da integração em *cloud*.

Palavras chave

Inspeção visual de centrais fotovoltaicas; Visão computacional; Arquitetura distribuída *edge-cloud*; Mapeamento geoespacial; Prova de conceito.

Abstract

Photovoltaic solar energy has played an increasingly important role in the energy transition, which intensifies the need for more scalable operation and maintenance processes. Traditional visual inspection of modules is time-consuming, dependent on human interpretation and difficult to apply frequently in large-scale plants, particularly for detecting soiling that reduces solar radiation capture. Building on Project I, in which computer vision approaches for the detection and segmentation of soiling in RGB images of PV panels were studied and experimentally evaluated, the present work focuses on the implementation and functional validation of a demonstrator prototype.

The aim was to develop a distributed platform integrating image acquisition at the edge, *backend* ingestion, asynchronous job-based processing, inference using the YOLOv8x and DINOv2 models, spatial mapping of detections onto the plant layout, and operational visualisation through a web application. The acquisition component was designed for a Raspberry Pi 4 Model B with a Camera Module 3, incorporating spool, checksum, retry and idempotency mechanisms for capture submission. Persistence and the spatial intersection between detections, zones and panels rely on PostgreSQL/PostGIS, with the results being converted into severity levels and operational decision-support tickets. The architecture was designed to be cloud-ready, prepared for Microsoft Azure services, while the demonstration runs locally in Docker containers.

The validation, functional in nature and carried out in a controlled environment, confirmed the feasibility of the end-to-end flow, from image capture or submission to the presentation of results. The system is presented as a proof of concept rather than a final product, with future work identified as the systematic collection of images in real field conditions, the strengthening of the dataset and the consolidation of cloud integration.

Keywords

Photovoltaic plant inspection; Computer vision; Edge-cloud distributed architecture; Geospatial mapping; Proof of concept.

Índice geral

1.	Introdução.....	1
1.1	Âmbito.....	1
1.2	Definição do Problema	2
1.3	Objetivos.....	4
1.4	Cronograma	5
1.5	Contribuições	6
1.6	Organização do Documento	7
2.	Protótipo	9
2.1	Arquitetura	9
2.2	Componente de Hardware.....	17
2.2.1	Raspberry Pi 4 Model B e Raspberry Pi Camera Module 3.....	18
2.3	Componente de Software.....	20
2.3.1	Serviços edge.....	21
2.3.2	Backend, ingestão e processamento assíncrono	24
2.3.3	Serviço de inferência.....	26
2.3.4	Base de dados e mapeamento geoespacial	28
2.3.5	Aplicação web	31
3.	Testes de Validação	37
4.	Conclusões e Trabalho Futuro	43
5.	Referências.....	46

Índice de figuras

Figura 1 - Arquitetura global do protótipo, preparada para integração cloud/Azure e executada em localhost para demonstração funcional	10
Figura 2 - Fluxograma geral de funcionamento do protótipo	14
Figura 3 - Pipeline funcional do protótipo em ambiente de demonstração, desde a aquisição ou submissão da imagem até à apresentação dos resultados na aplicação web.....	16
Figura 4 - Organização dos serviços no ambiente local Docker	17
Figura 5 - Componentes de hardware considerados para a componente edge do protótipo: Raspberry Pi 4 Model B e Raspberry Pi Camera Module 3	19
Figura 6 - Componentes de hardware utilizados na componente edge do protótipo: Raspberry Pi 4 Model B, Raspberry Pi Camera Module 3 e cartão microSD	20
Figura 7 - Construção dos serviços edge através de Docker Compose no Raspberry Pi.....	22
Figura 8 - Arranque dos serviços edge-capture e edge-uploader no ambiente Raspberry Pi.....	22
Figura 9 - Editor de layout para definição manual de zonas e painéis da central fotovoltaica	30
Figura 10 - Interface de calibração por pontos para alinhamento entre imagem capturada e base-map da planta.....	31
Figura 11 - Dashboard geral da aplicação, com indicadores operacionais e tickets recentes	32
Figura 12 - Visualização da planta fotovoltaica com zonas, painéis e estados operacionais sobrepostos ao base-map	33
Figura 13 - Interface de submissão manual de capturas, permitindo associar uma imagem à planta e à câmara correspondente	34
Figura 14 - Página de tickets com filtragem por planta, estado e severidade ..	35
Figura 15 - Detalhe de uma zona com resultados de inferência YOLOv8x e DINOv2 associados à ocorrência analisada	36
Figura 16 - Diagrama de casos de uso da aplicação web e da componente edge.....	38
Figura 17 - Registo de execução da componente edge durante a captura de imagens com Raspberry Pi Camera Module 3	39

Lista de tabelas

Tabela 1 - Cronograma de tarefas	6
Tabela 2 - Estrutura utilizada para documentação dos testes de validação	41

Lista de abreviaturas, siglas e acrónimos

ACR	Azure Container Registry
API	Application Programming Interface
BD	Base de Dados
CSS	Cascading Style Sheets
DINOv2	Self-Distillation with No Labels, versão 2
HDR	High Dynamic Range
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
OS	Operating System
PV	Photovoltaic
RGB	Red, Green, Blue
SHA-256	Secure Hash Algorithm 256-bit
UI	User Interface
UTC	Coordinated Universal Time
YOLOv8x	You Only Look Once, versão 8x

1. Introdução

Este primeiro capítulo apresenta o enquadramento geral do trabalho desenvolvido no âmbito do Projeto II, clarificando o contexto em que surge o protótipo proposto, o problema que se pretende abordar e os objetivos definidos para a sua concretização. Tendo em consideração o trabalho realizado no Projeto I, dedicado ao estudo e avaliação de técnicas de visão computacional para inspeção de centrais fotovoltaicas, este relatório centra-se na implementação de uma solução funcional, de carácter demonstrador, capaz de integrar aquisição de imagens, processamento automático, inferência, mapeamento geoespacial e visualização operacional. São ainda apresentadas as principais contribuições do trabalho, o planeamento seguido e a organização global do documento.

1.1 Âmbito

A energia solar fotovoltaica tem vindo a assumir um papel cada vez mais relevante no contexto da transição energética, tanto em instalações de grande escala como em sistemas distribuídos de autoconsumo. A expansão deste tipo de infraestrutura aumenta a necessidade de garantir níveis adequados de operação e manutenção, uma vez que o desempenho dos módulos PV depende não só das condições de irradiância, mas também do seu estado físico e superficial. A acumulação de sujidade, poeiras, folhas, excrementos de aves, sombras localizadas ou outros elementos sobre os painéis pode reduzir a captação de radiação solar e comprometer a eficiência energética da instalação.

A inspeção tradicional de centrais fotovoltaicas é, em muitos casos, realizada de forma manual ou através de campanhas periódicas de observação e análise visual. Embora estas abordagens sejam úteis, apresentam limitações quando aplicadas a centrais de maior dimensão, onde o número de módulos torna a inspeção morosa, pouco escalável e dependente da interpretação humana. Além disso, a periodicidade das inspeções pode dificultar a deteção atempada de ocorrências, reduzindo a capacidade de resposta das equipas de operação e manutenção. Neste contexto, a utilização de sistemas automáticos de inspeção visual surge como uma alternativa relevante, permitindo recolher imagens, processá-las de forma sistemática e transformar os resultados em informação operacional.

O presente trabalho surge na continuidade do Projeto I, no qual foi estudada a aplicação de técnicas de visão computacional e aprendizagem profunda à inspeção de centrais fotovoltaicas. Nesse trabalho foram analisadas abordagens baseadas em modelos da família YOLO e em arquiteturas baseadas em transformers, nomeadamente DINOv2, com o objetivo de detetar e segmentar

padrões de sujidade em imagens RGB de módulos PV. Foram ainda identificadas limitações importantes para a aplicação em contexto real, como a influência de reflexos, sombras, variações de iluminação, sujidade difusa e necessidade de conjuntos de dados mais representativos.

Dando continuidade a esse enquadramento, o Projeto II centra-se no desenvolvimento de um protótipo funcional que integra a componente experimental de visão computacional num fluxo completo de inspeção. A solução proposta não se limita à execução isolada dos modelos de inferência, procurando antes organizar um sistema distribuído capaz de suportar a captura ou submissão de imagens, a ingestão *backend*, a criação de jobs assíncronos, a execução de inferência, o mapeamento das deteções sobre o layout da central, a classificação de severidade, a geração de tickets e a visualização dos resultados numa aplicação web.

Assim, o âmbito deste relatório incide sobre a conceção, implementação e validação funcional de uma plataforma demonstradora para inspeção visual de centrais fotovoltaicas. O sistema foi concebido como prova de conceito, integrando componentes de edge, *backend*, serviço de inferência, base de dados geoespacial e interface web. A componente de aquisição no edge encontra-se implementada com recurso a *Raspberry Pi 4 Model B* e *Raspberry Pi Camera Module 3*, permitindo capturar imagens, enviá-las para o *backend*, executar inferência e apresentar resultados na aplicação web. A validação atual combina essa capacidade de captura com imagens de teste controladas, provenientes de fixtures ou de uma pasta local, permitindo testar o pipeline de forma repetível. A principal limitação nesta fase não está na inexistência de captura por hardware, mas na ausência de uma campanha sistemática de recolha em terreno real, numa central fotovoltaica em operação [1].

1.2 Definição do Problema

A operação eficiente de centrais fotovoltaicas depende da capacidade de acompanhar, de forma regular, o estado dos módulos instalados. Embora estes sistemas sejam frequentemente associados a uma exploração relativamente autónoma, a sua produção energética pode ser afetada por fatores externos e superficiais, como a acumulação de poeiras, folhas, excrementos de aves, manchas, sombras localizadas ou outros elementos que reduzem a captação da radiação solar. Estes fenómenos podem não comprometer de imediato o funcionamento global da instalação, mas tendem a provocar perdas de desempenho e a aumentar a necessidade de ações de limpeza e manutenção.

Um dos principais desafios associados a este contexto é a dificuldade de realizar inspeções frequentes, sistemáticas e escaláveis. Em centrais de maior dimensão, a análise manual dos módulos é morosa, depende da disponibilidade

de equipas técnicas e está sujeita à interpretação visual de cada operador. Mesmo quando são recolhidas imagens dos painéis, a sua análise continua muitas vezes dependente de observação humana, o que limita a rapidez de resposta e dificulta a criação de registos consistentes ao longo do tempo. Assim, o problema não está apenas na deteção da sujidade, mas também na transformação dessa deteção em informação útil para a operação e manutenção.

No âmbito do Projeto I, foi estudada a aplicação de técnicas de visão computacional e aprendizagem profunda à inspeção automática de módulos PV, com especial enfoque na deteção e segmentação de sujidade em imagens RGB. Esse trabalho permitiu comparar diferentes abordagens, incluindo modelos da família YOLO e uma solução baseada em DINOv2, avaliando o seu desempenho em cenários de deteção e segmentação. Foram ainda identificadas limitações relevantes, nomeadamente a sensibilidade a reflexos, sombras, variações de iluminação, sujidade difusa e coexistência de classes na mesma imagem [2].

Apesar da importância dessa componente experimental, a aplicação prática de um sistema de inspeção não se esgota na execução de um modelo de inferência. Para que os resultados possam apoiar decisões operacionais, é necessário integrá-los num fluxo mais amplo, capaz de associar cada imagem a uma planta, a uma câmara, a uma zona da central e, quando possível, a painéis específicos. Sem esta integração, a saída do modelo permanece essencialmente visual, isto é, limitada a caixas delimitadoras, máscaras ou classificações, não sendo diretamente convertida em ações concretas de manutenção, alertas ou tickets de intervenção.

Deste modo, o problema abordado no Projeto II consiste em transformar a componente experimental desenvolvida anteriormente num protótipo funcional de plataforma distribuída, capaz de ligar aquisição de imagens, ingestão, processamento assíncrono, inferência, mapeamento geoespacial e visualização operacional. O sistema deve permitir que uma imagem capturada ou submetida seja processada automaticamente, que as deteções sejam normalizadas e relacionadas com o layout da planta, que a severidade seja estimada e que os resultados sejam apresentados numa interface web de apoio à decisão.

Outro aspeto relevante prende-se com a validação da componente de aquisição no *edge*. A solução foi concebida e implementada com um *Raspberry Pi 4 Model B* e uma *Raspberry Pi Camera Module 3*, aproximando o protótipo de um cenário de captura local em contexto de central fotovoltaica. O sistema já permite capturar imagens através do hardware, enviá-las para o backend, executar inferência e apresentar os resultados na interface web [1][3][4]. No entanto, nesta fase, ainda não foi realizada uma campanha de recolha em terreno real, com imagens captadas numa central fotovoltaica em operação e sob condições ambientais variáveis. Por esse motivo, o desenvolvimento e os testes recorrem também a imagens de teste provenientes de *fixtures* ou de uma pasta local, permitindo validar o comportamento do pipeline de forma controlada e repetível.

Assim, o problema central deste trabalho pode ser sintetizado na necessidade de criar uma solução demonstradora que ultrapasse a simples inferência sobre imagens isoladas e organize todo o processo de inspeção visual numa plataforma coerente. Esta plataforma deve assegurar rastreabilidade, tolerância a falhas na submissão de imagens, persistência dos resultados, mapeamento espacial das ocorrências e visualização clara para apoio à operação e manutenção de centrais fotovoltaicas.

1.3 Objetivos

Este projeto tem como objetivo principal desenvolver e validar um protótipo funcional de uma plataforma distribuída para inspeção visual de centrais fotovoltaicas, integrando aquisição de imagens no *edge*, ingestão *backend/cloud*, inferência com modelos de visão computacional, mapeamento espacial sobre o layout da planta e visualização operacional numa aplicação web.

Para atingir este objetivo principal, foram definidos os seguintes objetivos específicos:

- implementar uma componente *edge* preparada para *Raspberry Pi 4 Model B*, com suporte a *pool* local, cálculo de *checksum*, *retries* e *upload* fiável de imagens;
- desenvolver uma API de ingestão responsável por receber capturas, validar metadados, evitar duplicações e criar jobs de processamento;
- integrar modelos de visão computacional previamente estudados, nomeadamente YOLOv8x e DINOv2, num serviço de inferência acessível pelo backend;
- estruturar um pipeline assíncrono de processamento, capaz de acompanhar o estado dos jobs desde a submissão da imagem até à geração de resultados operacionais;
- mapear deteções sobre o layout da central fotovoltaica, recorrendo a *PostgreSQL/PostGIS* para representar zonas, painéis, polígonos e interseções espaciais;
- criar mecanismos de classificação de severidade e geração de tickets, permitindo transformar deteções visuais em elementos acionáveis para operação e manutenção;
- desenvolver uma aplicação web para visualização dos resultados, incluindo *dashboard*, mapa da central, capturas, *tickets* e funcionalidades de administração, layout e calibração;
- validar funcionalmente o fluxo completo com imagens de teste, distinguindo entre funcionalidades implementadas, componentes simuladas/controladas e trabalho futuro.

Dado que o Projeto I incidu sobretudo no enquadramento do problema, revisão da literatura, construção/organização de dados, avaliação experimental e comparação de modelos de visão computacional para deteção e segmentação de sujidade em módulos PV, o foco do Projeto II encontra-se direcionado para a implementação e validação funcional do protótipo demonstrador. Assim, este trabalho não pretende repetir a avaliação teórica e experimental já realizada, mas sim transformar essa base numa solução integrada, capaz de demonstrar um fluxo *end-to-end* de inspeção visual com valor operacional.

1.4 Cronograma

Na **Tabela 1** é apresentado o cronograma das principais tarefas realizadas no âmbito do Projeto II. O planeamento foi organizado de forma a dar continuidade ao trabalho desenvolvido no Projeto I, deslocando o foco da análise experimental dos modelos para a conceção, implementação e validação funcional de um protótipo demonstrador.

Tarefa Nº1 – Definição da arquitetura do protótipo: desenho da arquitetura global da solução, identificação dos principais componentes do sistema e definição do fluxo *end-to-end* de inspeção visual, desde a aquisição ou submissão da imagem até à visualização dos resultados na aplicação *web*.

Tarefa Nº2 – Implementação da componente *edge* e ingestão: desenvolvimento dos mecanismos associados à componente *edge*, incluindo *pool* local, cálculo de *checksum*, *retries* e *upload* de imagens, bem como implementação da API de ingestão responsável por validar metadados, guardar capturas e criar *jobs* de processamento.

Tarefa Nº3 – Integração do processamento assíncrono e serviço de inferência: implementação do fluxo de processamento através de *jobs* assíncronos, integração com o serviço de inferência e normalização dos resultados produzidos pelos modelos de visão computacional, nomeadamente YOLOv8x e DINOv2.

Tarefa Nº4 – Implementação da BD, mapeamento geoespacial e *tickets*: definição do modelo de dados, integração com *PostgreSQL/PostGIS*, representação de plantas, câmaras, zonas e painéis, mapeamento das deteções sobre o *layout* da central e geração de severidade e *tickets* operacionais.

Tarefa Nº5 – Desenvolvimento da aplicação *web*: implementação da interface de utilizador para visualização e gestão da informação produzida pelo sistema, incluindo *dashboard*, mapa da central, página de capturas, listagem de *tickets* e funcionalidades de administração, *layout* e calibração.

Tarefa Nº6 – Testes funcionais, validação e redação do relatório: realização de testes sobre os principais componentes do protótipo, validação do fluxo completo

com imagens de teste, identificação de limitações e redação do relatório final de Projeto II.

Tabela 1 - Cronograma de tarefas

	Mês 1	Mês 2	Mês 3	Mês 4	Mês 5	Mês 6
Tarefa 1						
Tarefa 2						
Tarefa 3						
Tarefa 4						
Tarefa 5						
Tarefa 6						

1.5 Contribuições

O trabalho apresentado no âmbito do Projeto II dá continuidade ao estudo desenvolvido no Projeto I, no qual foi analisada a aplicação de técnicas de visão computacional e aprendizagem profunda à inspeção automática de centrais fotovoltaicas. Enquanto o trabalho anterior se centrou no enquadramento do problema, na análise do estado da arte, na preparação dos dados e na avaliação experimental de modelos, o presente relatório foca-se na conceção, implementação e validação funcional de um protótipo demonstrador.

Consideram-se como principais contribuições do trabalho desenvolvido em Projeto II:

- proposta de uma arquitetura distribuída para inspeção visual de centrais fotovoltaicas, integrando componentes *edge*, *backend*, serviço de inferência, BD geoespacial e aplicação *web*;
- implementação de um fluxo *end-to-end* de inspeção, desde a captura ou submissão da imagem até à ingestão, criação de job assíncrono, inferência, mapeamento espacial, classificação de severidade, geração de *tickets* e visualização na interface;
- desenvolvimento de mecanismos de fiabilidade na componente *edge*, incluindo *pool* local, *checksum*, *retries* e idempotência no envio de capturas;
- integração dos modelos de visão computacional YOLOv8x e DINOv2 num serviço de inferência, permitindo associar os resultados dos modelos a um fluxo operacional de inspeção;

- utilização de *PostgreSQL/PostGIS* para representação de plantas, câmaras, zonas e painéis, bem como para o cruzamento espacial entre detecções e elementos do layout da central;
- desenvolvimento de funcionalidades de calibração e *layout*, permitindo relacionar imagens capturadas com o *base-map* da planta e definir manualmente zonas e painéis relevantes para análise;
- criação de uma aplicação web para apoio à decisão operacional, incluindo *dashboard*, mapa da central, página de capturas, gestão de *tickets* e áreas de administração;
- realização de testes funcionais sobre os principais componentes do protótipo, permitindo validar o fluxo completo em ambiente controlado e identificar limitações e pontos de evolução futura.

Estas contribuições procuram demonstrar a viabilidade de uma solução integrada para inspeção visual de centrais fotovoltaicas, não como produto final, mas como prova de conceito funcional. O protótipo desenvolvido permite transformar resultados de inferência visual em informação estruturada e acionável, aproximando a componente experimental do Projeto I de um cenário de utilização operacional em apoio à manutenção e limpeza seletiva de módulos PV.

1.6 Organização do Documento

Este documento encontra-se organizado em quatro capítulos.

O primeiro capítulo apresenta a introdução ao trabalho desenvolvido, enquadrando o tema da inspeção visual de centrais fotovoltaicas, a definição do problema, os objetivos definidos, o cronograma das tarefas realizadas e as principais contribuições resultantes do Projeto II.

O segundo capítulo descreve o protótipo desenvolvido. Neste capítulo é apresentada a arquitetura global da solução, bem como as suas principais componentes de hardware e software. São descritos os serviços responsáveis pela aquisição ou submissão de imagens, ingestão, processamento assíncrono, inferência, persistência dos resultados, mapeamento geoespacial, geração de tickets e visualização na aplicação *web*.

O terceiro capítulo apresenta os testes de validação realizados ao protótipo. São descritos os testes funcionais aplicados às principais componentes do sistema, incluindo a componente *edge*, a ingestão de capturas, o processamento assíncrono, a integração com o serviço de inferência, o mapeamento em *PostGIS* e a interface *web*. Esta validação tem como objetivo demonstrar o funcionamento do fluxo *end-to-end* em ambiente controlado [5].

Por fim, o quarto capítulo apresenta as conclusões do trabalho desenvolvido e identifica direções de trabalho futuro. São discutidos os principais resultados

obtidos, as limitações do protótipo e os aspetos que deverão ser consolidados numa evolução posterior, nomeadamente a integração completa da captura física no *Raspberry Pi*, a validação com imagens reais em campo, o reforço do *dataset*, a melhoria dos mecanismos de calibração e a integração *cloud* completa [4].

2. Protótipo

Este capítulo apresenta o protótipo desenvolvido no âmbito do Projeto II. A primeira secção descreve a arquitetura global da solução, identificando os principais componentes que constituem o sistema e o fluxo de funcionamento desde a aquisição ou submissão da imagem até à visualização dos resultados na aplicação web. Em seguida, é apresentada a componente de hardware considerada para o cenário de *edge*, baseada em *Raspberry Pi 4 Model B* e *Raspberry Pi Camera Module 3*. Por fim, descreve-se a componente de software implementada, incluindo os serviços de ingestão, processamento assíncrono, inferência, persistência, mapeamento geoespacial, geração de *tickets* e interface web [1][3][4].

2.1 Arquitetura

A **Figura 1** apresenta a arquitetura global do protótipo desenvolvido para inspeção visual de centrais fotovoltaicas. A solução foi concebida como uma plataforma distribuída, organizada em três grandes blocos principais: a componente *edge*, associada à aquisição ou preparação das imagens; a componente backend, responsável pela ingestão, armazenamento, orquestração e processamento; e a componente *frontend*, responsável pela visualização dos resultados e pelo apoio à decisão operacional. Embora a organização dos serviços tenha sido preparada para uma futura integração com infraestrutura *cloud*, nomeadamente através de serviços *Azure*, a demonstração e validação funcional apresentadas neste trabalho são realizadas em ambiente local, com os serviços executados em *localhost* e, quando aplicável, em containers Docker. Esta opção permitiu validar o comportamento do protótipo de forma controlada, reduzindo dependências externas durante a fase de desenvolvimento, sem comprometer a possibilidade de evolução futura para *cloud* [6].

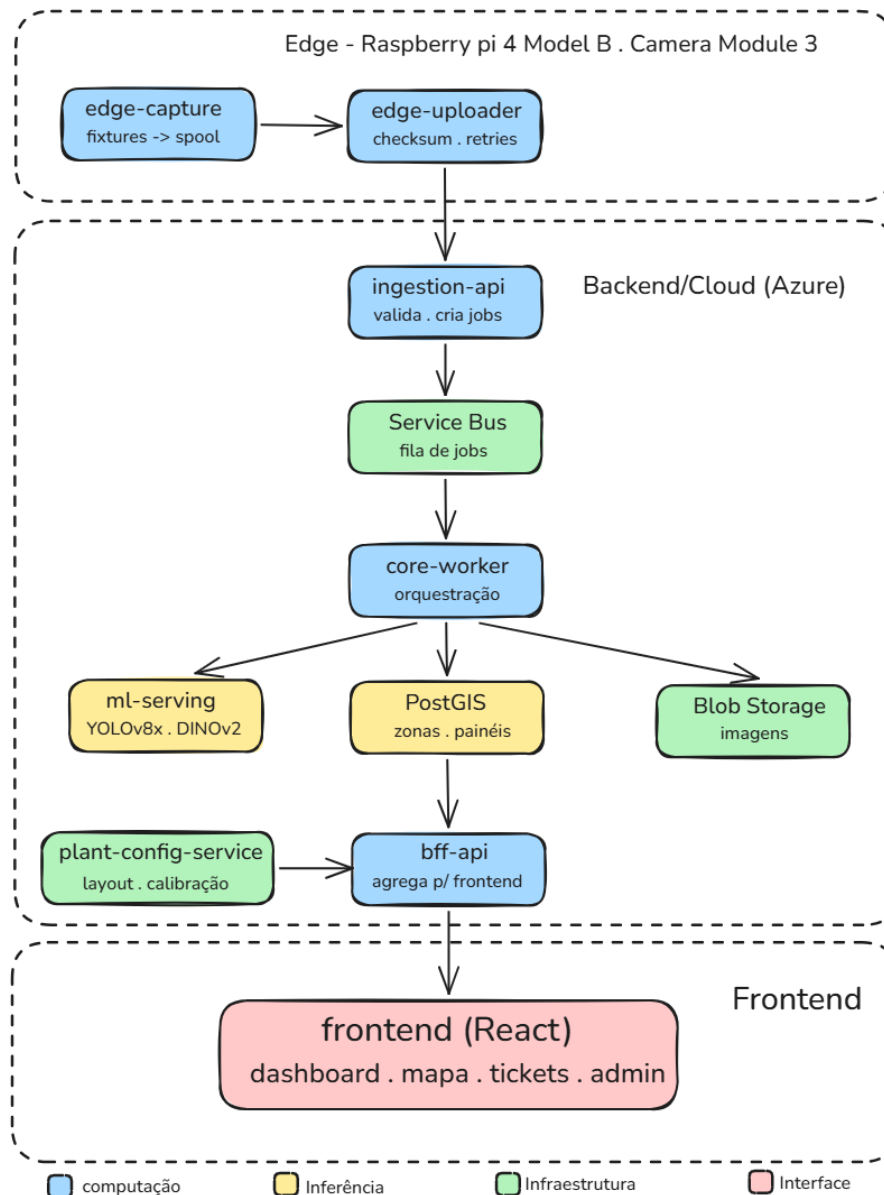


Figura 1 - Arquitetura global do protótipo, preparada para integração cloud/Azure e executada em localhost para demonstração funcional

A componente *edge* foi desenhada para funcionar sobre um *Raspberry Pi 4 Model B* ligado a uma *Raspberry Pi Camera Module 3*, aproximando o protótipo de um cenário real de captura local em centrais fotovoltaicas. O objetivo desta componente é permitir que imagens dos módulos PV possam ser recolhidas junto da instalação e enviadas para o sistema de processamento. Contudo, nesta fase do desenvolvimento, a captura por *hardware* encontra-se funcional no protótipo, mas ainda não foi realizada uma validação sistemática em terreno real, com imagens recolhidas numa central fotovoltaica em operação [1][3][4].

A componente *edge* encontra-se organizada em dois serviços principais: *edge-capture* e *edge-uploader*. O serviço *edge-capture* representa a etapa de aquisição ou simulação da captura, sendo responsável por colocar a imagem e os

respetivos metadados numa área local de *spool*. O serviço *edge-uploader* lê essa *spool*, prepara o envio da imagem para a API de ingestão e aplica mecanismos de fiabilidade, como cálculo ou utilização de *checksum*, *retries* e controlo local de submissões. Estes mecanismos são relevantes porque permitem lidar com falhas temporárias de comunicação e evitam que uma imagem seja perdida caso o backend não esteja momentaneamente disponível.

No momento do envio, cada captura é acompanhada por metadados que permitem identificar a origem e o contexto da imagem, como o identificador da câmara e o instante de captura. Em conjunto com o *checksum*, estes dados suportam a deduplicação no *backend*, evitando que a mesma imagem seja processada mais do que uma vez. Esta preocupação com idempotência é importante num sistema distribuído, uma vez que o reenvio de mensagens ou imagens pode ocorrer de forma legítima quando existem falhas temporárias, *retries* ou interrupções de comunicação.

Na componente *backend*, a *ingestion-api* é responsável por receber as capturas provenientes do edge ou submetidas manualmente através da aplicação web. Esta API valida os metadados recebidos, verifica duplicações com base em informação como *checksum*, *capturedAt* e *cameralId*, guarda a imagem no serviço de armazenamento configurado e cria um job de processamento. Na arquitetura preparada para *cloud*, o armazenamento de imagens corresponde ao *Azure Blob Storage* e a fila de jobs pode ser suportada pelo *Azure Service Bus*. No entanto, para efeitos de demonstração local, estes mecanismos são executados ou simulados em ambiente *localhost*, recorrendo a serviços equivalentes ou configurações locais. Desta forma, foi possível validar o fluxo funcional sem depender da disponibilização integral da infraestrutura *Azure* [6].

A utilização de jobs assíncronos permite separar a ingestão da imagem do seu processamento. Esta decisão arquitetural evita que a API de ingestão fique bloqueada enquanto decorrem tarefas potencialmente demoradas, como a inferência dos modelos, a geração de resultados visuais ou o cruzamento espacial em *PostGIS* [5]. Assim, após a receção e validação da captura, a *ingestion-api* cria um job em estado inicial e coloca-o na fila de processamento. O processamento posterior fica a cargo do *core-worker*, que consome os jobs disponíveis e executa a sequência necessária para transformar uma imagem submetida em informação operacional.

O *core-worker* desempenha o papel de orquestrador do processamento. Ao consumir um job, atualiza o seu estado e inicia a sequência de operações definida para cada captura. O job começa no estado *PENDING*, passa para *IN_PROGRESS* quando é consumido pelo *worker*, evolui para *INFERRED* após a execução da inferência, para *MAPPED* depois do cruzamento espacial com o layout da planta e para *ALERTED* quando são calculadas severidades e criados ou atualizados tickets. Caso ocorra uma falha durante o processamento, o job pode ser marcado como *FAILED*. Esta máquina de estados permite acompanhar

a evolução de cada captura, garantindo rastreabilidade e facilitando o diagnóstico de erros.

O serviço *ml-serving* disponibiliza os modelos de visão computacional integrados no protótipo, nomeadamente YOLOv8x e DINOv2, estudados anteriormente no âmbito do Projeto I [2]. O YOLOv8x é utilizado para deteção de objetos e produção de *bounding boxes* associadas a padrões de sujidade ou anomalias visíveis nos módulos PV. O DINOv2 é considerado como abordagem complementar, orientada para uma representação visual mais rica, nomeadamente em cenários nos quais a segmentação ou identificação de regiões afetadas possa fornecer informação mais expressiva do que uma simples caixa delimitadora. A integração destes modelos no serviço de inferência permite que os resultados deixem de existir apenas como experiências isoladas e passem a alimentar um fluxo operacional de inspeção.

Depois da inferência, os resultados são normalizados pelo *core-worker*. Esta normalização permite transformar a saída dos modelos em dados consistentes, contendo classes, níveis de confiança, coordenadas, caixas delimitadoras, máscaras ou outros elementos visuais relevantes. A normalização é necessária porque diferentes modelos podem produzir outputs com estruturas distintas. Ao converter esses resultados para um formato comum, o sistema consegue persistir as deteções, relacioná-las com a planta fotovoltaica e apresentá-las de forma coerente na aplicação web.

A BD utilizada no protótipo é *PostgreSQL* com extensão *PostGIS*. Esta componente suporta a persistência dos dados do sistema e permite representar informação geoespacial associada à central fotovoltaica, incluindo plantas, câmaras, zonas, painéis, polígonos e deteções [7]. O *PostGIS* é utilizado para cruzar as deteções produzidas pelos modelos com os elementos definidos no layout da planta, permitindo identificar a zona afetada e, quando possível, o painel ou conjunto de painéis associado à ocorrência. Esta etapa é essencial para transformar resultados visuais em informação operacional, uma vez que uma deteção isolada só se torna útil quando é contextualizada no espaço físico da central [5].

O *plant-config-service* suporta a configuração da planta fotovoltaica, incluindo a gestão de plantas, câmaras, *base-maps*, *layout* e calibração. A calibração permite estabelecer uma relação entre a imagem capturada pela câmara e a representação espacial da planta, enquanto o layout permite definir manualmente zonas e painéis relevantes. Esta abordagem manual e controlada é adequada ao carácter demonstrador do protótipo, uma vez que permite validar o mapeamento espacial antes de introduzir automatismos mais complexos. Assim, a solução privilegia uma configuração verificável, ajustável e compreensível para o utilizador.

A camada de acesso aos dados para a interface é assegurada pela *bff-api*. Este serviço agrega informação proveniente das diferentes componentes do

backend e disponibiliza ao frontend dados já organizados para consumo pela aplicação web. Esta opção simplifica a lógica do lado cliente, uma vez que o frontend não necessita de comunicar diretamente com múltiplos serviços internos. Em vez disso, obtém através da *bff-api* a informação necessária para apresentar indicadores, mapas, capturas, *tickets*, detalhes de deteções e dados de administração.

A **Figura 2** apresenta o funcionamento geral do protótipo, desde a captura ou seleção da imagem na componente *edge* até à disponibilização dos resultados na aplicação web. O fluxo inclui a criação da imagem, o armazenamento temporário na *pool* local, o envio para o backend, a validação dos metadados, a verificação de duplicados, a criação do job assíncrono, a execução da inferência, o mapeamento das deteções sobre o layout da planta e a disponibilização da informação ao utilizador.

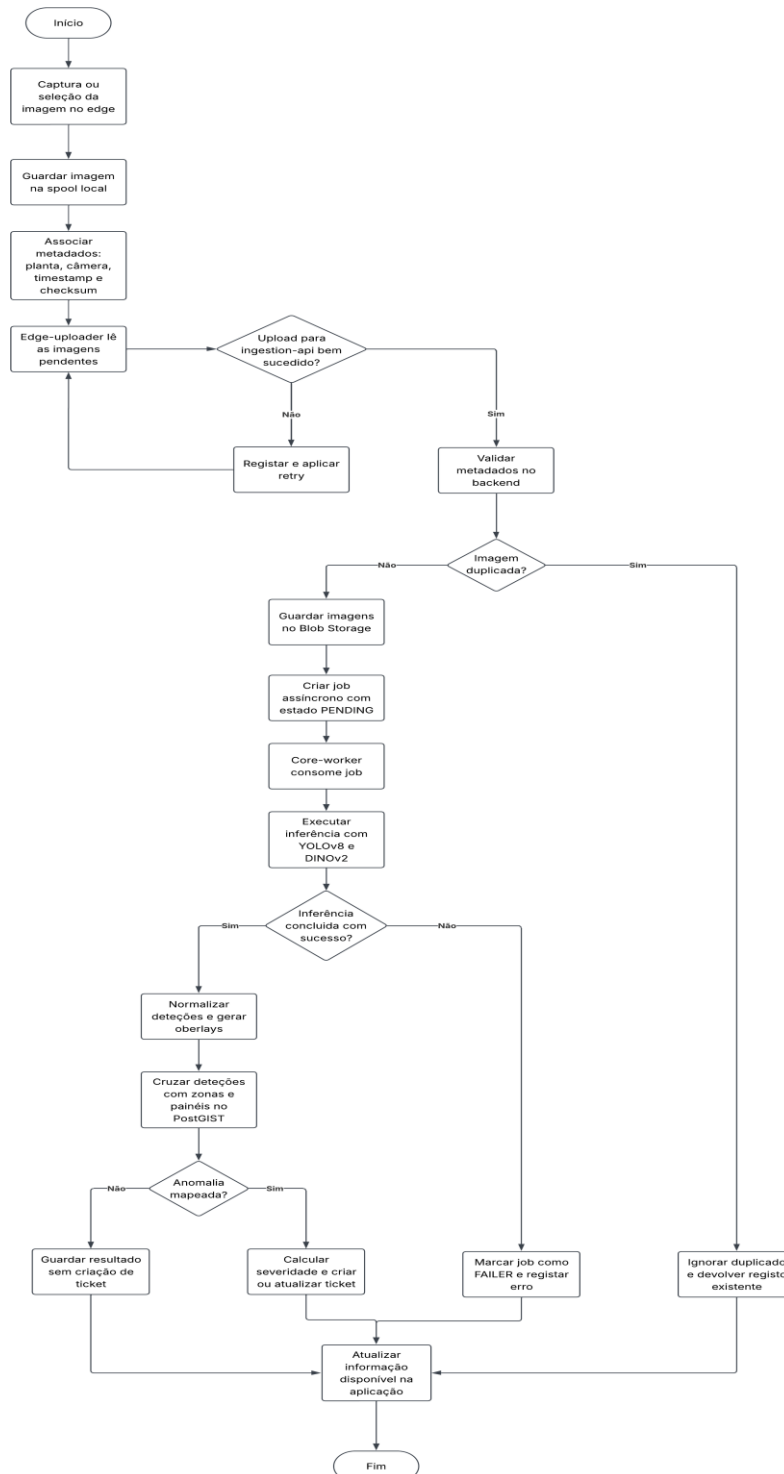


Figura 2 - Fluxograma geral de funcionamento do protótipo

A aplicação frontend foi desenvolvida em *React* e *TypeScript*, funcionando como ponto de contacto entre o utilizador e o sistema. A interface disponibiliza páginas para *dashboard*, mapa, capturas, tickets e administração, permitindo acompanhar o estado das inspeções, consultar resultados de inferência, visualizar deteções sobre o layout da planta, analisar ocorrências por severidade e gerir elementos de configuração [8][9]. Esta camada é particularmente relevante

porque transforma os resultados técnicos do pipeline em informação compreensível para apoio à operação e manutenção.

A **Figura 3** sintetiza o pipeline funcional do protótipo em onze etapas, desde a aquisição no *edge* até à apresentação dos resultados na interface web. O fluxo inicia-se com a captura ou simulação da imagem, seguida da gravação na *spool* local. Posteriormente, o *edge-uploader* envia a captura para a *ingestion-api*, onde ocorre a validação e deduplicação. A imagem é então guardada no armazenamento configurado e é criado um job em estado *PENDING*. O *core-worker* consome esse job, altera o seu estado para *IN_PROGRESS*, invoca o serviço de inferência e, após a obtenção dos resultados, normaliza as deteções. De seguida, o *PostGIS* é utilizado para cruzar as deteções com zonas e painéis, passando o job para o estado *MAPPED* [5]. Por fim, são calculadas severidades, criados ou atualizados tickets e os resultados ficam disponíveis na interface web.

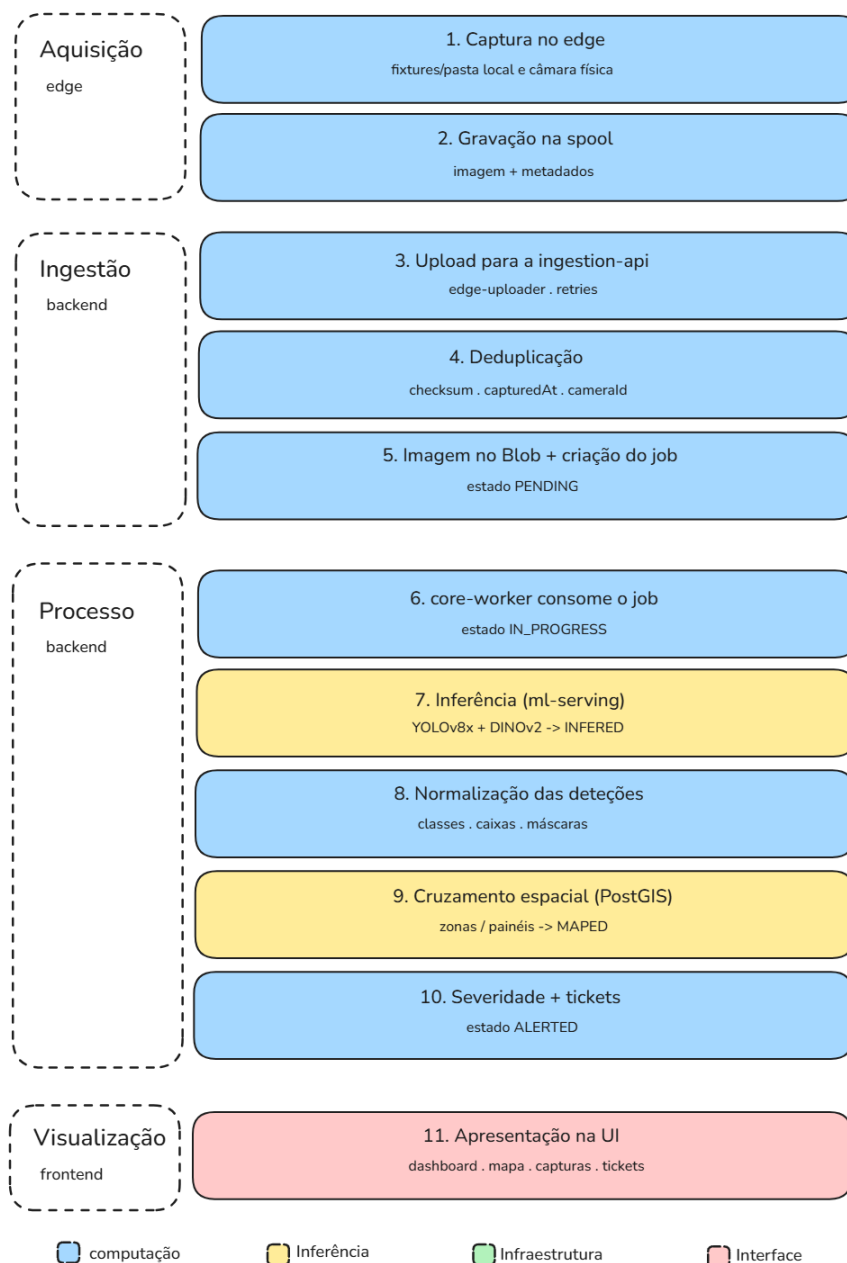


Figura 3 - Pipeline funcional do protótipo em ambiente de demonstração, desde a aquisição ou submissão da imagem até à apresentação dos resultados na aplicação web

De forma global, a arquitetura proposta demonstra que o protótipo não se limita à execução isolada de modelos de visão computacional sobre imagens. Pelo contrário, organiza a inspeção visual como um processo completo, capaz de ligar aquisição, ingestão, processamento assíncrono, inferência, mapeamento geoespacial, geração de *tickets* e visualização operacional. A execução local em *localhost* permite validar esta lógica de forma controlada, enquanto a preparação para serviços *Azure* assegura que a solução mantém uma trajetória clara de evolução para cenários mais próximos de produção [6].

2.2 Componente de Hardware

Esta secção descreve a componente de hardware considerada para o protótipo desenvolvido. A solução foi concebida para funcionar numa lógica de *edge*, em que a aquisição inicial das imagens é realizada junto da central fotovoltaica, antes do envio para o backend. Para esse efeito, foi considerado um *Raspberry Pi 4 Model B*, ligado a uma *Raspberry Pi Camera Module 3*. Esta escolha permite aproximar o protótipo de um cenário de recolha local de imagens, recorrendo a *hardware* compacto, acessível e adequado a sistemas demonstradores [1][3][4].

Para efeitos de desenvolvimento e validação funcional, os serviços do protótipo foram organizados num ambiente local baseado em Docker. Esta abordagem permitiu executar os diferentes componentes de forma isolada, mas integrada, reproduzindo o funcionamento esperado da arquitetura distribuída. A componente *edge* comunica com a *ingestion-api* através de HTTP, enquanto os restantes serviços partilham a BD *PostgreSQL/PostGIS* e o armazenamento local compatível com *Blob Storage* [7][5].

A **Figura 4** representa a organização dos serviços no ambiente local Docker, incluindo os serviços de *backend*, processamento assíncrono, inferência, configuração da planta, agregação de dados para o *frontend*, armazenamento e BD geoespacial [10].

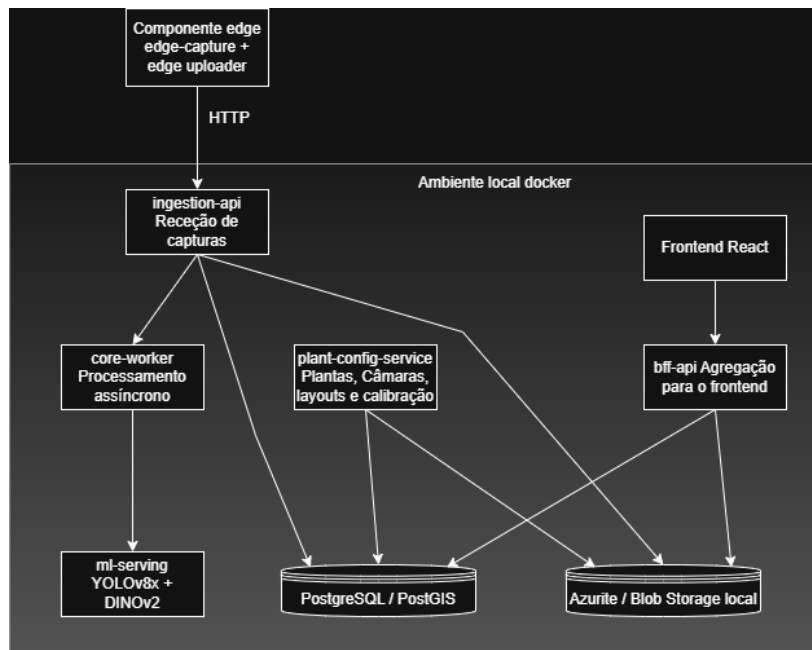


Figura 4 - Organização dos serviços no ambiente local Docker

No entanto, importa clarificar que, embora a componente física de captura se encontre funcional e integrada no fluxo do protótipo, a validação apresentada neste relatório ainda não corresponde a uma campanha de recolha em terreno real, numa central fotovoltaica em operação. O sistema já permite capturar imagens através do hardware considerado, enviá-las para o backend, executar a inferência e apresentar os resultados na aplicação web. Paralelamente, foram utilizadas imagens de teste provenientes de uma pasta local ou *fixtures*, de modo a permitir testes controlados, repetíveis e independentes das condições de campo. Assim, a limitação atual não reside na inexistência de captura por hardware, mas sim na ausência de validação com imagens reais recolhidas no terreno, sob condições ambientais e operacionais próprias de uma central fotovoltaica.

2.2.1 Raspberry Pi 4 Model B e Raspberry Pi Camera Module 3

O *Raspberry Pi 4 Model B* foi selecionado como dispositivo de *edge* por apresentar uma boa relação entre custo, dimensão, consumo energético e capacidade de processamento para tarefas de aquisição e preparação de imagens. Trata-se de um computador de placa única, adequado a protótipos de sistemas distribuídos e aplicações IoT, podendo executar serviços locais, scripts de captura, mecanismos de *spool* e processos de comunicação com o backend. No contexto deste trabalho, o *Raspberry Pi* não é utilizado para executar a inferência dos modelos de visão computacional, mas sim como ponto de recolha, organização e submissão das imagens capturadas [1][11].

A utilização do *Raspberry Pi 4 Model B* permite separar a componente de aquisição da componente de processamento mais intensivo. Esta separação é relevante porque os modelos de visão computacional, como YOLOv8x e DINOv2, exigem recursos computacionais superiores aos que se pretende alocar ao dispositivo de *edge* nesta fase do protótipo. Assim, o *Raspberry Pi* atua como dispositivo de captura e envio, enquanto o backend e o serviço *ml-serving* assumem as tarefas de inferência, normalização, mapeamento e geração de resultados operacionais[1][2].

Para a aquisição de imagem foi considerada a *Raspberry Pi Camera Module 3* [3]. Esta câmara é compatível com a plataforma *Raspberry Pi* e dispõe de características adequadas à recolha de imagens RGB em contexto demonstrador, nomeadamente sensor de 12 *megapixels*, suporte para HDR e *autofocus*. Estas características são relevantes para o tipo de aplicação em causa, uma vez que a inspeção visual de módulos PV depende da qualidade da imagem, da nitidez das regiões analisadas e da capacidade de lidar com variações de luminosidade.

No cenário previsto, a câmara seria instalada numa posição fixa, orientada para uma área da central ou para um conjunto de módulos PV. Esta configuração

permite simplificar o problema de mapeamento espacial, uma vez que a perspectiva da câmara permanece estável ao longo do tempo. A partir desta condição, torna-se possível relacionar a imagem capturada com o *base-map* da planta através de calibração por pontos, etapa que será detalhada na componente de software. Em termos práticos, esta relação entre imagem e planta permite que detecções visuais sejam posteriormente associadas a zonas ou painéis específicos.

O *Raspberry Pi* executa *Raspberry Pi OS*, sistema operativo oficial da plataforma, permitindo instalar e configurar os serviços necessários à componente *edge*. A lógica implementada inclui um serviço de captura, responsável por obter imagens através da câmara ou, em testes controlados, a partir de *fixtures*/pasta local, e um serviço de upload, responsável por enviar a captura para o backend. Esta abordagem permite testar o comportamento da *spool*, dos metadados, do *checksum*, dos *retries* e da comunicação com a *ingestion-api*, tanto com imagens capturadas pelo *hardware* como com imagens de teste repetíveis [11].

A **Figura 5** e **Figura 6** apresenta o conjunto de hardware considerado para o protótipo, incluindo o *Raspberry Pi 4 Model B* e a *Raspberry Pi Camera Module 3*.

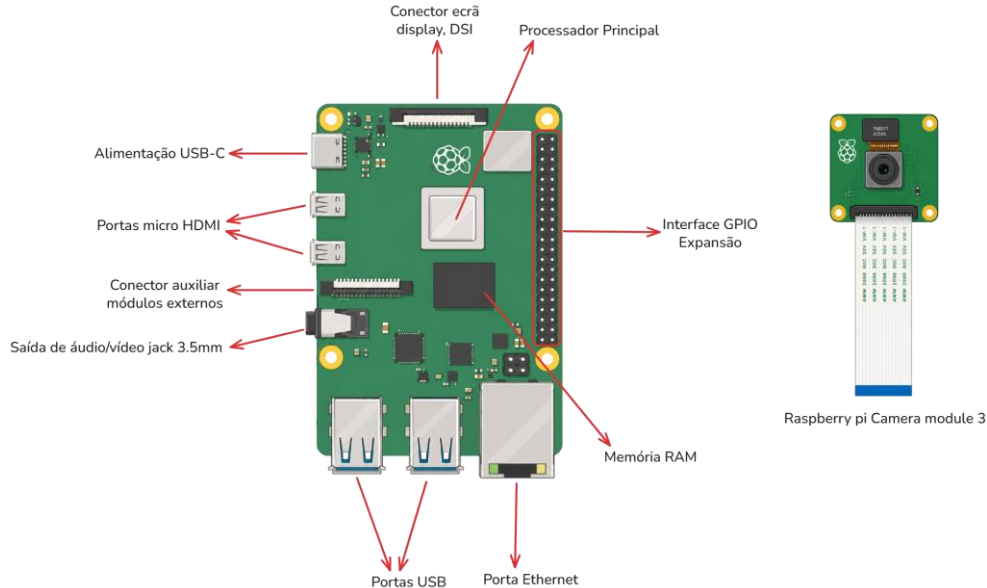


Figura 5 - Componentes de hardware considerados para a componente edge do protótipo: Raspberry Pi 4 Model B e Raspberry Pi Camera Module 3



Figura 6 - Componentes de hardware utilizados na componente edge do protótipo: Raspberry Pi 4 Model B, Raspberry Pi Camera Module 3 e cartão microSD

Apesar de o sistema de captura por hardware se encontrar funcional no protótipo, a sua validação em ambiente de campo permanece como trabalho futuro. O objetivo, numa fase posterior, será realizar uma campanha de recolha numa central fotovoltaica em operação, mantendo o mesmo fluxo de ingestão, processamento, inferência, mapeamento e visualização já validado em *localhost*. Desta forma, o protótipo demonstra a viabilidade funcional da arquitetura, distinguindo a implementação técnica já realizada da validação operacional em condições reais de terreno.

2.3 Componente de Software

A componente de software constitui o núcleo funcional do protótipo desenvolvido. É nesta camada que se concretiza o fluxo de inspeção visual, desde a obtenção ou submissão da imagem até à apresentação dos resultados ao utilizador. Ao contrário de uma abordagem centrada apenas na execução de modelos de visão computacional, o software implementado organiza um processo completo, capaz de receber imagens, validar metadados, criar jobs assíncronos, executar inferência, normalizar deteções, relacionar resultados com o *layout* da central fotovoltaica, calcular severidade, gerar tickets e disponibilizar informação operacional através de uma aplicação web.

A solução foi desenvolvida de forma modular, separando responsabilidades entre diferentes serviços. Esta organização permite isolar a aquisição de imagens, a ingestão, o processamento, a inferência, a persistência, a configuração da planta e a visualização. A separação entre componentes facilita a manutenção do protótipo, permite testar partes do sistema de forma independente e prepara a

solução para uma possível evolução futura para ambientes *cloud*. Embora a arquitetura esteja preparada para integração com serviços *Azure*, a demonstração descrita neste relatório é executada em ambiente local, com recurso a *localhost* e, quando aplicável, a *containers Docker* [6][10].

Do ponto de vista tecnológico, o protótipo integra diferentes ferramentas e *frameworks*, escolhidas de acordo com o papel desempenhado por cada componente. A componente *edge* recorre a scripts e serviços preparados para execução no *Raspberry Pi*. O backend é implementado com serviços desenvolvidos em *Python/FastAPI*, responsáveis pela ingestão, processamento e exposição de dados. A persistência é assegurada por *PostgreSQL* com extensão *PostGIS*, permitindo guardar não só dados relacionais, mas também informação geoespacial associada a zonas, painéis e deteções [1][7][12]. A aplicação web foi desenvolvida com *React* e *TypeScript*, disponibilizando uma interface visual para dashboard, mapa, capturas, tickets e administração [8][9].

Nas subsecções seguintes são descritas as principais partes desta componente de software. Em primeiro lugar, apresentam-se os serviços *edge*, responsáveis pela captura, *spool* e envio das imagens. De seguida, descreve-se o backend, a ingestão e o processamento assíncrono. Posteriormente, é apresentado o serviço de inferência, onde são integrados os modelos *YOLOv8x* e *DINOv2* estudados no Projeto I. Segue-se a descrição da BD e do mapeamento geoespacial com *PostGIS*. Por fim, apresenta-se a aplicação web, que permite transformar os resultados do pipeline técnico em informação compreensível para apoio à operação e manutenção [2][5].

2.3.1 Serviços edge

A componente *edge* representa o ponto inicial do fluxo de inspeção visual. No protótipo desenvolvido, esta componente é responsável pela obtenção da imagem, pela sua organização local e pelo envio para o backend. Embora o sistema tenha sido preparado para funcionar em contexto de central fotovoltaica, com captura através de *Raspberry Pi 4 Model B* e *Raspberry Pi Camera Module 3*, a validação realizada no âmbito deste relatório combina a captura funcional por hardware com imagens de teste provenientes de uma pasta local ou *fixtures*. Esta opção permite testar o comportamento do sistema de forma repetível, sem depender exclusivamente de condições ambientais reais[1][3].

A componente *edge* foi organizada em dois serviços principais: *edge-capture* e *edge-uploader*. O serviço *edge-capture* é responsável pela etapa de aquisição. Quando utilizado com a câmara, este serviço obtém a imagem através do hardware disponível; em cenários de teste, pode também recolher imagens a partir de uma diretoria local previamente definida. Esta flexibilidade é importante para o desenvolvimento do protótipo, uma vez que permite validar o *pipeline* com

imagens controladas e, ao mesmo tempo, manter a possibilidade de utilizar imagens capturadas diretamente pelo dispositivo.

A execução dos serviços *edge* foi validada no *Raspberry Pi* através de *Docker Compose*. A Figura 7 apresenta a construção dos serviços responsáveis pela captura e envio das imagens, enquanto a Figura 8 evidencia o arranque dos containers associados à componente *edge* [1][10].

```
pi@photovoltaicos: local's password:
Linux photovoltaicos 6.12.75-rpi-rpi-v8 #1 SMP PREEMPT Debian 1:6.12.75-1-rpi1 (2026-03-11) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jun 21 22:30:10 2026 from 192.168.1.134
pi@photovoltaicos:~$ curl https://reward-identified-twist-serum.trycloudflare.com/health
pi@photovoltaicos:~$ cd ~/fotovoltaicos_II/edge/edge
-bash: cd: /home/pi/fotovoltaicos_II/edge/edge: No such file or directory
pi@photovoltaicos:~$ cd ~/fotovoltaicos_II/edge
pi@photovoltaicos:~/fotovoltaicos_II/edge$ nano .env
pi@photovoltaicos:~/fotovoltaicos_II/edge$ docker compose -f docker-compose.pi.yml up -d --build edge-uploader

[+] Building 2.8s (22/22) FINISHED
=> [internal] load local bake definitions                                0.0s
=> => reading from stdin 1.04kB                                         0.0s
=> [edge-capture internal] load build definition from Dockerfile        0.1s
=> => transferring dockerfile: 265B                                      0.0s
=> [edge-uploader internal] load build definition from Dockerfile       0.1s
=> => transferring dockerfile: 265B                                      0.0s
=> [edge-uploader internal] load metadata for docker.io/library/python:3 1.2s
=> [edge-uploader internal] load .dockerignore                          0.0s
=> => transferring context: 2B                                           0.0s
=> [edge-capture internal] load .dockerignore                           0.1s
=> => transferring context: 2B                                           0.0s
=> [edge-uploader 1/6] FROM docker.io/library/python:3.12-slim@sha256:d7 0.2s
=> => resolve docker.io/library/python:3.12-slim@sha256:d764629ce0ddd8c7 0.1s
=> [edge-uploader internal] load build context                          0.1s
=> => transferring context: 647B                                         0.1s
=> [edge-capture internal] load build context                           0.0s
=> => transferring context: 682B                                         0.1s
=> CACHED [edge-capture 2/6] WORKDIR /app                               0.0s
=> CACHED [edge-uploader 3/6] COPY requirements.txt .                   0.0s
=> CACHED [edge-uploader 4/6] RUN pip install --no-cache-dir -r requirem 0.0s
=> CACHED [edge-uploader 5/6] COPY run.py .                             0.0s
=> CACHED [edge-uploader 6/6] COPY app/ ./app/                          0.0s
=> CACHED [edge-capture 3/6] COPY requirements.txt .                   0.0s
=> CACHED [edge-capture 4/6] RUN pip install --no-cache-dir -r requirem 0.0s
=> CACHED [edge-capture 5/6] COPY run.py .                             0.0s
=> CACHED [edge-capture 6/6] COPY app/ ./app/                          0.0s
=> [edge-capture] exporting to image                                    0.5s
=> => exporting layers                                                    0.0s
```

Figura 7 - Construção dos serviços *edge* através de *Docker Compose* no *Raspberry Pi*

```
=> => transferring context: 682B                                         0.0s
=> CACHED [edge-capture 2/6] WORKDIR /app                               0.0s
=> CACHED [edge-uploader 3/6] COPY requirements.txt .                   0.0s
=> CACHED [edge-uploader 4/6] RUN pip install --no-cache-dir -r requirem 0.0s
=> CACHED [edge-uploader 5/6] COPY run.py .                             0.0s
=> CACHED [edge-uploader 6/6] COPY app/ ./app/                          0.0s
=> CACHED [edge-capture 3/6] COPY requirements.txt .                   0.0s
=> CACHED [edge-capture 4/6] RUN pip install --no-cache-dir -r requirem 0.0s
=> CACHED [edge-capture 5/6] COPY run.py .                             0.0s
=> CACHED [edge-capture 6/6] COPY app/ ./app/                          0.0s
=> [edge-capture] exporting to image                                    0.5s
=> => exporting layers                                                    0.0s
=> exporting manifest sha256:7a45f81e9ebb43b23f9b6f7647b26bb0f507c08a 0.0s
=> exporting config sha256:a52ec1210820e988c5d3327bc971d15b58e532cf03 0.0s
=> exporting attestation manifest sha256:74f84eafce17656d989a559f8963 0.1s
=> exporting manifest list sha256:5f05b4ab520236a88f6834c29bac0349dfff 0.1s
=> naming to docker.io/library/pv-edge-capture:local                   0.0s
=> unpacking to docker.io/library/pv-edge-capture:local                0.0s
=> [edge-uploader] exporting to image                                    0.5s
=> => exporting layers                                                    0.0s
=> exporting manifest sha256:8ebfe510129e86ac1beb69496744b15c0808b06f 0.0s
=> exporting config sha256:58ffe0b532d5784334e04052565e83748d43a49559 0.0s
=> exporting attestation manifest sha256:d7deecdc3faad131862a0bd3a403 0.1s
=> exporting manifest list sha256:bfb2c035217c1e3dd5f1bda01d09f63671f 0.1s
=> naming to docker.io/library/pv-edge-uploader:local                  0.0s
=> unpacking to docker.io/library/pv-edge-uploader:local               0.0s
=> [edge-capture] resolving provenance for metadata file                0.0s
=> [edge-uploader] resolving provenance for metadata file               0.0s
[+] up 4/4
✔ Image pv-edge-capture:local Built 3.0s
✔ Image pv-edge-uploader:local Built 3.0s
✔ Container edge-edge-capture-1 Started 23.9s
✔ Container edge-edge-uploader-1 Started 13.1s
pi@photovoltaicos:~/fotovoltaicos_II/edge$ cd edge-capture
(.venv) pip@photovoltaicos:~/fotovoltaicos_II/edge/edge-capture$ . .venv/bin/activate
CAMERA_ID=cam-birdeye-01 CAPTURE_SOURCE_MODE=rplicam CAPTURE_INTERVAL_SECONDS=60 SPOOL_ROOT=/var/lib/pv-spool pyth
on run.py
```

Figura 8 - Arranque dos serviços *edge-capture* e *edge-uploader* no ambiente *Raspberry Pi*

Depois de obtida a imagem, o *edge-capture* coloca-a numa área local de *spool*. A *spool* funciona como uma zona intermédia entre a captura e o envio, evitando que a imagem seja perdida caso o backend não esteja disponível no momento da aquisição. Esta decisão torna o sistema mais robusto, pois separa a ação de capturar da ação de transmitir. Em vez de depender de uma comunicação imediata com a API de ingestão, o dispositivo *edge* pode guardar temporariamente a captura e permitir que o serviço de upload a envie quando existirem condições adequadas.

O serviço *edge-uploader* é responsável por ler as imagens presentes na *spool* e submetê-las à *ingestion-api*. Antes do envio, a imagem é associada a metadados relevantes, como o identificador da câmara, a data e hora da captura e outros elementos necessários para contextualizar a origem da imagem. Para além disso, é calculado ou utilizado um *checksum*, permitindo identificar a imagem de forma consistente e apoiar a deduplicação no *backend*. Esta abordagem é particularmente relevante em sistemas distribuídos, onde podem ocorrer reenvios devido a falhas temporárias de rede, reinícios do serviço ou indisponibilidade momentânea do *backend*.

A lógica de *retries* implementada no *edge-uploader* permite aumentar a fiabilidade do processo de submissão. Se uma imagem não for enviada com sucesso, permanece na *spool* e pode ser novamente tentada numa execução posterior. Desta forma, evita-se que uma falha temporária provoque perda definitiva de dados. Este comportamento aproxima o protótipo de um cenário operacional mais realista, no qual a conectividade entre o dispositivo *edge* e o backend pode não ser constante ou totalmente previsível.

A idempotência é outro aspeto relevante desta componente. O sistema deve ser capaz de lidar com o envio repetido da mesma imagem sem criar múltiplas capturas equivalentes no *backend*. Para isso, a identificação da imagem através de *checksum*, em conjunto com metadados como *cameraId* e *capturedAt*, permite verificar se uma captura já foi registada anteriormente. Assim, mesmo que o *edge-uploader* volte a enviar a mesma imagem por razões de segurança ou recuperação de falha, o *backend* pode evitar duplicações e manter a consistência dos dados.

No contexto da demonstração realizada em *localhost*, a componente *edge* permite validar o início do fluxo *end-to-end*. A imagem capturada ou selecionada a partir da pasta local é colocada na *spool*, enviada para a *ingestion-api*, registada no backend e posteriormente processada pelos restantes serviços. Esta validação mostra que a componente *edge* não é apenas uma abstração prevista na arquitetura, mas uma parte funcional do protótipo, ainda que a recolha sistemática de imagens em terreno real permaneça como trabalho futuro.

Em síntese, os serviços *edge* asseguram a ligação entre o mundo físico e o sistema de inspeção. A sua função não é executar a inferência dos modelos de visão computacional, mas garantir que as imagens são obtidas, organizadas,

identificadas e enviadas de forma fiável para o backend. Esta separação permite manter o dispositivo *edge* simples e focado na aquisição, enquanto o processamento mais exigente é realizado por serviços *backend* e pelo *ml-serving*.

2.3.2 Backend, ingestão e processamento assíncrono

A componente backend é responsável por receber as capturas provenientes do *edge* ou submetidas manualmente pela aplicação web, organizar o seu processamento e disponibilizar os resultados às restantes camadas do sistema. No protótipo desenvolvido, esta componente assume um papel central, uma vez que estabelece a ligação entre a aquisição de imagens, o serviço de inferência, a BD geoespacial e a interface de visualização. Embora a arquitetura esteja preparada para utilização futura de serviços *Azure*, a validação funcional apresentada neste relatório é realizada em ambiente local, com os serviços executados em *localhost* [6].

A ingestão das imagens é realizada através da *ingestion-api*. Esta API recebe a imagem e os respetivos metadados, como o identificador da câmara, a data e hora de captura e a informação necessária para associar a captura a uma planta fotovoltaica. Antes de aceitar a imagem para processamento, a API valida a consistência dos metadados recebidos e verifica se a captura já foi anteriormente registada. Para esse efeito, são utilizados elementos como o *checksum* da imagem, o instante de captura e o identificador da câmara. Esta verificação permite reduzir duplicações, sobretudo em situações em que o *edge-uploader* repete o envio por razões de segurança ou recuperação após falha.

Após a validação, a imagem é armazenada no serviço de armazenamento configurado para o ambiente em execução. Na arquitetura preparada para cloud, este armazenamento corresponde ao *Azure Blob Storage*; no ambiente local de demonstração, o mesmo comportamento é reproduzido através de configuração local adequada. Esta separação entre o ficheiro de imagem e os restantes dados persistidos permite que a BD guarde apenas metadados e referências para os ficheiros, evitando que imagens sejam armazenadas diretamente nas tabelas relacionais [6].

Depois de armazenada a imagem, a *ingestion-api* cria um job de processamento. Este job representa a unidade de trabalho associada a uma captura e permite acompanhar o seu estado ao longo do pipeline. A criação de jobs assíncronos é uma decisão relevante, porque evita que a API tenha de executar imediatamente tarefas demoradas, como inferência, geração de outputs visuais, cruzamento espacial ou criação de tickets. Assim, a API limita-se a receber, validar, armazenar e registar a captura, deixando o processamento pesado para uma componente própria.

Na arquitetura preparada para *cloud*, a comunicação entre a ingestão e o processamento pode ser suportada pelo *Azure Service Bus* [6]. No ambiente local, esta lógica é executada de forma equivalente, permitindo demonstrar o desacoplamento entre a recepção da imagem e o seu processamento. Esta abordagem aumenta a robustez do sistema, uma vez que permite acumular jobs quando existem múltiplas capturas, processá-los de forma controlada e recuperar mais facilmente de falhas parciais.

O *core-worker* é o serviço responsável por consumir e processar os jobs criados pela *ingestion-api*. Quando um job é selecionado para processamento, o seu estado é atualizado de *PENDING* para *IN_PROGRESS*, indicando que deixou de estar apenas registado e passou a estar em execução. Esta atualização é importante para garantir rastreabilidade e para permitir que a aplicação web apresente ao utilizador uma visão clara sobre o estado de cada captura.

Durante o processamento, o *core-worker* invoca o serviço de inferência, enviando a imagem ou a sua referência para o *ml-serving*. Após receber a resposta dos modelos, o *worker* normaliza os resultados para uma estrutura comum. Esta etapa é necessária porque diferentes modelos podem produzir saídas com formatos distintos, por exemplo *bounding boxes*, máscaras, classes, níveis de confiança ou coordenadas em diferentes sistemas de referência. A normalização permite que estes resultados sejam persistidos e tratados pelas etapas seguintes de forma uniforme.

Concluída a inferência, o job evolui para o estado *INFERRED*. De seguida, o *core-worker* inicia a etapa de mapeamento espacial, relacionando as deteções normalizadas com o layout da central fotovoltaica. Esta operação é suportada pela BD *PostgreSQL/PostGIS* e permite identificar zonas ou painéis afetados. Após esta etapa, o job passa para o estado *MAPPED*, sinalizando que os resultados visuais já foram convertidos em informação espacialmente contextualizada [5][7].

A etapa seguinte consiste no cálculo de severidade e na geração ou atualização de tickets. A severidade permite distinguir ocorrências com diferentes níveis de prioridade, enquanto os *tickets* representam elementos acionáveis para apoio à operação e manutenção. Quando esta fase é concluída, o job passa para o estado *ALERTED*. Caso ocorra uma falha em qualquer uma das etapas, o *job* pode ser marcado como *FAILED*, permitindo identificar problemas e facilitar diagnóstico posterior.

A utilização explícita de estados — *PENDING*, *IN_PROGRESS*, *INFERRED*, *MAPPED*, *ALERTED* e *FAILED* — constitui um mecanismo simples, mas importante, de rastreabilidade. Estes estados permitem acompanhar o percurso de cada imagem no sistema, desde a sua submissão até à disponibilização dos resultados finais. Além disso, facilitam a identificação de falhas, a repetição controlada de processamento e a apresentação de informação compreensível na aplicação *web*.

Em síntese, a componente backend e o processamento assíncrono permitem transformar uma captura numa entidade operacional acompanhável ao longo de todo o pipeline. A *ingestion-api* assegura a receção, validação e registo inicial da imagem, enquanto o *core-worker* executa as etapas de inferência, normalização, mapeamento e geração de alertas. Esta separação de responsabilidades torna o protótipo mais robusto, mais fácil de testar e mais preparado para evoluir de uma demonstração local para uma implantação *cloud*.

2.3.3 Serviço de inferência

O serviço de inferência, designado por *ml-serving*, é responsável por disponibilizar os modelos de visão computacional utilizados pelo protótipo. Esta componente estabelece a ligação entre o pipeline backend e a componente experimental desenvolvida no Projeto I, permitindo que os modelos estudados deixem de ser executados apenas em ambiente de avaliação e passem a integrar um fluxo funcional de inspeção visual.

No âmbito do protótipo, foram considerados os modelos *YOLOv8x* e *DINOv2*, previamente analisados no Projeto I para tarefas de deteção e segmentação de sujidade em módulos PV [2]. A sua integração no serviço *ml-serving* não tem como objetivo repetir a comparação de desempenho já realizada, mas sim demonstrar a forma como estes modelos podem ser utilizados num sistema operacional. Assim, o foco desloca-se da métrica isolada para a integração: receber uma imagem, executar inferência, devolver resultados estruturados e permitir que esses resultados sejam posteriormente mapeados, classificados e apresentados ao utilizador.

O *YOLOv8x* é utilizado como modelo de deteção, produzindo *bounding boxes*, classes e níveis de confiança associados a padrões visíveis na imagem. No contexto da inspeção de centrais fotovoltaicas, estas deteções podem corresponder a zonas de sujidade, manchas, excrementos de aves ou outros elementos visuais relevantes para a operação e manutenção. A sua principal função no protótipo é identificar regiões de interesse na imagem, fornecendo ao backend informação espacial inicial sobre a localização das ocorrências.

O *DINOv2* é considerado como abordagem complementar, orientada para uma representação visual mais rica. Ao contrário de uma deteção baseada apenas em caixas delimitadoras, este tipo de abordagem pode ser útil em cenários onde a geometria da região afetada seja relevante, por exemplo quando a sujidade apresenta contornos irregulares, padrões difusos ou baixa separação visual face ao fundo do painel. No protótipo, o *DINOv2* é integrado como fonte adicional de informação visual, permitindo explorar resultados mais expressivos para posterior análise e visualização.

O funcionamento do serviço de inferência é acionado pelo *core-worker*. Quando um job atinge a fase de processamento, o *worker* envia a imagem, ou a referência para a imagem armazenada, para o *ml-serving*. O serviço executa o modelo configurado e devolve uma resposta contendo as detecções ou regiões identificadas. Esta resposta pode incluir classes previstas, níveis de confiança, coordenadas das *bounding boxes*, máscaras ou outros dados derivados do modelo utilizado.

Uma etapa relevante deste processo é a normalização dos resultados. Como diferentes modelos podem devolver saídas com formatos distintos, o *core-worker* converte a resposta do *ml-serving* para uma estrutura comum, adequada à persistência e ao processamento posterior. Esta estrutura normalizada permite representar detecções de forma consistente, independentemente da origem do modelo, e facilita a sua utilização nas etapas seguintes do *pipeline*, nomeadamente no mapeamento geoespacial em *PostGIS* e na geração de *tickets* [5].

No contexto do protótipo, os resultados da inferência não são tratados como produto final. Uma *bounding box* ou uma máscara, isoladamente, apenas indica uma região visualmente relevante na imagem. Para que esta informação tenha valor operacional, é necessário relacioná-la com o layout da central fotovoltaica, identificar a zona ou painel afetado, calcular severidade e apresentar o resultado de forma compreensível na aplicação *web*. Por esse motivo, o serviço de inferência é apenas uma das componentes do fluxo, sendo complementado pelas etapas de normalização, mapeamento, classificação e visualização.

A integração do *ml-serving* também permite manter a arquitetura modular. Os modelos podem evoluir, ser substituídos ou ajustados sem que seja necessário alterar toda a lógica da aplicação. Esta separação é particularmente importante num domínio como a visão computacional, onde novos modelos, novas versões e novos *datasets* podem melhorar o desempenho ao longo do tempo. Assim, o protótipo fica preparado para incorporar futuras melhorias na componente de inferência, mantendo o mesmo fluxo geral de ingestão, processamento e apresentação dos resultados.

Em síntese, o serviço de inferência transforma os modelos estudados no Projeto I numa componente funcional do sistema desenvolvido em Projeto II. A sua função é processar imagens, identificar regiões de interesse e devolver resultados estruturados ao backend. A partir daí, o restante pipeline converte esses resultados em informação geoespacial, severidade, tickets e elementos visuais disponíveis na interface *web*, aproximando a componente experimental de visão computacional de um cenário de apoio à decisão operacional.

2.3.4 Base de dados e mapeamento geoespacial

A BD do protótipo foi concebida para suportar não apenas a persistência de informação administrativa e operacional, mas também o tratamento espacial necessário ao mapeamento das deteções sobre a planta fotovoltaica. Para esse efeito, foi utilizado *PostgreSQL* com a extensão *PostGIS*, permitindo combinar dados relacionais com dados geoespaciais. Esta escolha é particularmente relevante no contexto do projeto, uma vez que as deteções produzidas pelos modelos de visão computacional só se tornam operacionalmente úteis quando podem ser associadas a zonas, painéis ou áreas específicas da central [7].

A informação persistida inclui entidades como plantas fotovoltaicas, câmaras, capturas, jobs, resultados de inferência, zonas, painéis, severidades e tickets. Cada captura fica associada a uma câmara e a uma planta, permitindo manter o contexto da imagem analisada. Os *jobs* permitem acompanhar o estado do processamento, enquanto as deteções normalizadas representam os resultados produzidos pelo serviço de inferência. As zonas e painéis, por sua vez, são representados através de geometrias, permitindo que o sistema realize operações espaciais sobre o *layout* da instalação.

Um aspeto central desta componente é a distinção entre a imagem capturada e a representação espacial da planta. A imagem obtida pela câmara corresponde a uma perspetiva visual, normalmente influenciada pela posição, orientação e campo de visão da câmara. Já o *base-map* da planta representa uma visão organizada do espaço físico da central, onde podem ser definidas zonas, painéis e outros elementos relevantes. Para relacionar estes dois domínios, o protótipo prevê uma etapa de calibração, baseada na seleção de pontos correspondentes entre a imagem e o mapa da planta.

A calibração por pontos permite calcular uma transformação geométrica entre o plano da imagem e o plano do *base-map*. Esta transformação, frequentemente representada através de uma homografia quando a perspetiva da câmara é fixa, permite projetar coordenadas da imagem para o espaço da planta. No contexto do protótipo, esta abordagem é adequada porque a câmara é considerada numa posição estável, orientada para uma área definida da central fotovoltaica. Assim, as deteções obtidas sobre a imagem podem ser relacionadas com zonas e painéis previamente definidos no *layout*.

O *layout* da planta é definido de forma manual na fase atual do protótipo. Esta decisão foi assumida por ser mais adequada a uma prova de conceito, permitindo criar e ajustar zonas e painéis de forma controlada antes de introduzir mecanismos automáticos mais complexos. O utilizador pode definir áreas relevantes da central, delimitar zonas de inspeção e representar painéis através de polígonos. Esta configuração manual permite validar o comportamento do mapeamento geoespacial com maior transparência, uma vez que os elementos espaciais usados no cruzamento são explícitos e verificáveis.

Após a inferência, as detecções produzidas pelos modelos são normalizadas e convertidas em informação espacial compatível com o modelo de dados. Dependendo do tipo de saída, uma detecção pode ser representada através de uma *bounding box*, de uma máscara ou de uma região aproximada. Estas geometrias são depois cruzadas com as zonas e painéis existentes no *PostGIS*, permitindo identificar que elementos do layout são afetados por cada ocorrência. Desta forma, uma detecção visual deixa de ser apenas um resultado na imagem e passa a ser interpretada como uma ocorrência localizada na planta [5].

O uso de *PostGIS* permite realizar operações como interseção, contenção ou sobreposição entre geometrias. No protótipo, estas operações são utilizadas para determinar se uma detecção intersecta uma zona, se se sobrepõe a um painel ou se se encontra fora do layout configurado. Esta última situação também é relevante, pois permite identificar casos em que a detecção não pode ser associada com confiança a nenhum elemento da planta, seja por calibração incompleta, erro de configuração ou detecção localizada fora da área mapeada.

O resultado do mapeamento geoespacial alimenta a etapa de classificação de severidade. A severidade pode depender de fatores como a classe detetada, a área afetada, a confiança do modelo, a quantidade de detecções numa zona ou a sua relação com painéis específicos. No âmbito deste protótipo, o objetivo não é definir uma política definitiva de manutenção, mas demonstrar que os resultados visuais podem ser transformados em indicadores acionáveis. Assim, a severidade funciona como uma camada intermédia entre a inferência técnica e a decisão operacional.

A partir da severidade e do contexto espacial da ocorrência, o sistema pode criar ou atualizar tickets. Estes tickets representam elementos de acompanhamento para operação e manutenção, permitindo organizar as ocorrências por planta, zona, painel, estado e prioridade. Esta funcionalidade é importante porque aproxima o protótipo de um cenário real de utilização, no qual as detecções automáticas devem ser convertidas em tarefas ou alertas compreensíveis para equipas técnicas.

A componente de mapeamento geoespacial constitui, por isso, uma das principais contribuições do protótipo. Enquanto a inferência identifica regiões visualmente relevantes, o *PostGIS* permite contextualizar essas regiões no espaço físico da central. Esta ligação entre visão computacional e informação espacial transforma o resultado do modelo em conhecimento operacional, apoiando a identificação de áreas afetadas e a priorização de intervenções [5].

Em síntese, a BD e o mapeamento geoespacial permitem estruturar a informação do sistema de forma persistente, rastreável e espacialmente contextualizada. A utilização de *PostgreSQL/PostGIS* possibilita representar zonas, painéis, polígonos e detecções, bem como calcular relações espaciais entre estes elementos. Esta componente é essencial para que o protótipo deixe de ser

apenas um sistema de análise de imagens e passe a funcionar como uma plataforma de suporte à decisão operacional em centrais fotovoltaicas [5][7].

A configuração espacial da planta é realizada através de funcionalidades de administração disponibilizadas na aplicação *web*. A **Figura 9** apresenta o editor de layout, onde é possível definir zonas e painéis sobre o *base-map* da planta fotovoltaica. Esta etapa é relevante porque permite associar posteriormente as deteções produzidas pelos modelos a elementos físicos concretos da instalação.

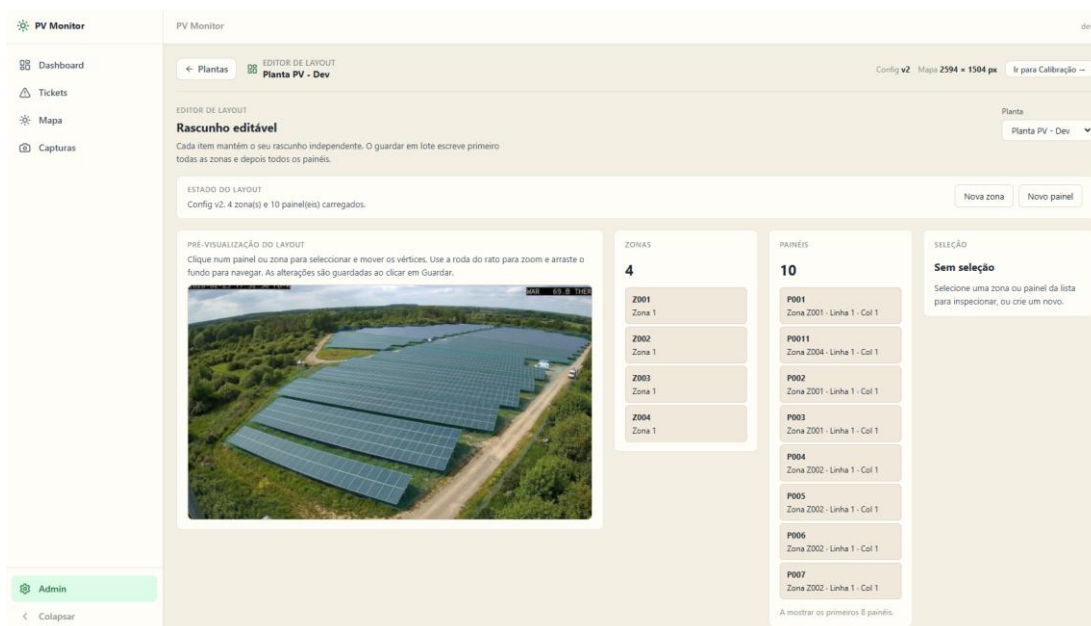


Figura 9 - Editor de layout para definição manual de zonas e painéis da central fotovoltaica

Para além da definição do layout, o protótipo inclui uma interface de calibração por pontos. A **Figura 10** apresenta esta funcionalidade, que permite estabelecer a correspondência entre a imagem capturada pela câmara e o *base-map* da planta. Esta calibração é necessária para projetar as deteções obtidas na imagem para o espaço da planta e permitir o cruzamento espacial através do *PostgreSQL/PostGIS* [5][7].

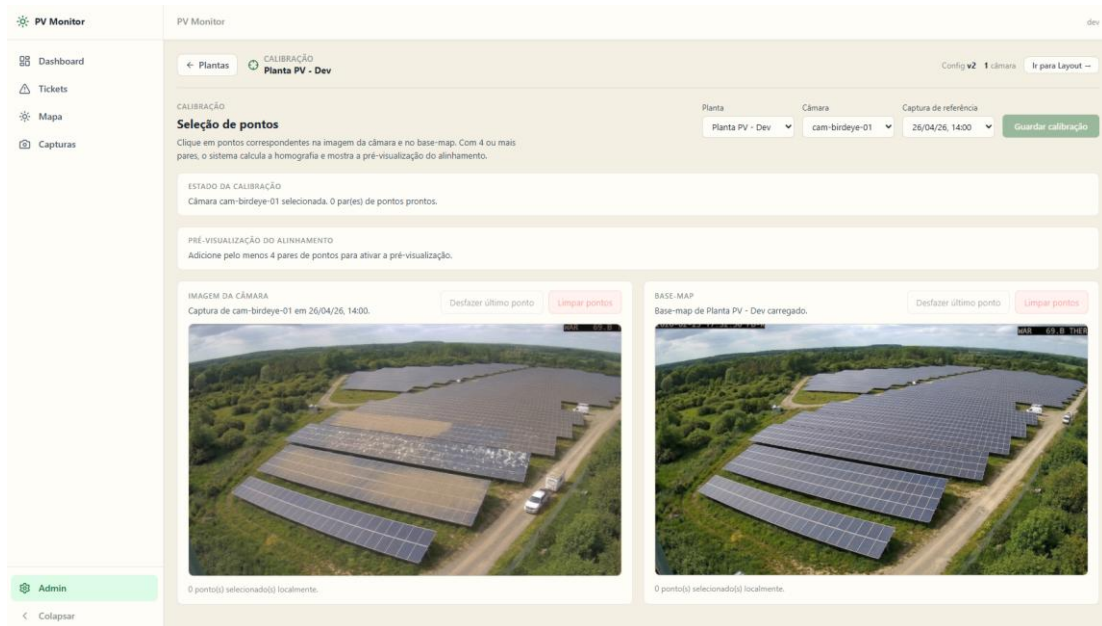


Figura 10 - Interface de calibração por pontos para alinhamento entre imagem capturada e base-map da planta

2.3.5 Aplicação web

A aplicação web constitui a camada de interação entre o utilizador e o protótipo desenvolvido. O seu objetivo é transformar os dados produzidos pelo pipeline técnico — capturas, jobs, deteções, mapeamento geoespacial, severidades e tickets — em informação visualmente compreensível e útil para apoio à operação e manutenção de centrais fotovoltaicas. Esta camada é particularmente relevante porque permite que os resultados dos modelos de visão computacional deixem de ser analisados apenas como outputs técnicos e passem a ser apresentados num contexto operacional.

O frontend foi desenvolvido com *React* e *TypeScript*, permitindo estruturar a aplicação em componentes reutilizáveis e tipados [8][9]. A utilização de Vite facilita o desenvolvimento e a execução local da aplicação, enquanto bibliotecas como *TanStack Query* apoiam a comunicação com a API e a gestão de dados assíncronos no lado cliente [13]. A interface utiliza *Tailwind CSS* para a organização visual e estilização dos elementos, *Recharts* para a apresentação de indicadores e gráficos, e *Lucide React* para a utilização de ícones consistentes ao longo da aplicação. Estas tecnologias foram escolhidas por permitirem desenvolver uma interface modular, responsiva e adequada a uma prova de conceito funcional.

A aplicação comunica com o backend através da *bff-api*, que agrega os dados necessários ao frontend. Esta opção evita que a interface tenha de conhecer diretamente todos os serviços internos do sistema, simplificando o consumo de informação. Assim, a aplicação web obtém dados já organizados para apresentar

indicadores, mapas, capturas, *tickets*, resultados de inferência e informação de configuração. Esta separação contribui para manter o frontend focado na visualização e interação, enquanto a lógica de processamento permanece no backend.

A página inicial da aplicação corresponde ao *dashboard*. Esta vista apresenta uma síntese do estado operacional do sistema, incluindo indicadores sobre capturas, deteções, tickets recentes e níveis de severidade. O dashboard permite ao utilizador obter rapidamente uma visão geral da situação da central, identificando se existem ocorrências pendentes, zonas com maior número de deteções ou *tickets* que exigem atenção. Esta página funciona como ponto de entrada para a análise, orientando o utilizador para as áreas que necessitam de consulta mais detalhada.

A aplicação *web* inclui um *dashboard* inicial, apresentado na **Figura 11**, onde são disponibilizados indicadores gerais sobre o estado do sistema, como tickets abertos, ocorrências críticas, jobs recentes e capturas registadas. Esta vista permite ao utilizador obter uma leitura rápida do estado operacional da planta.

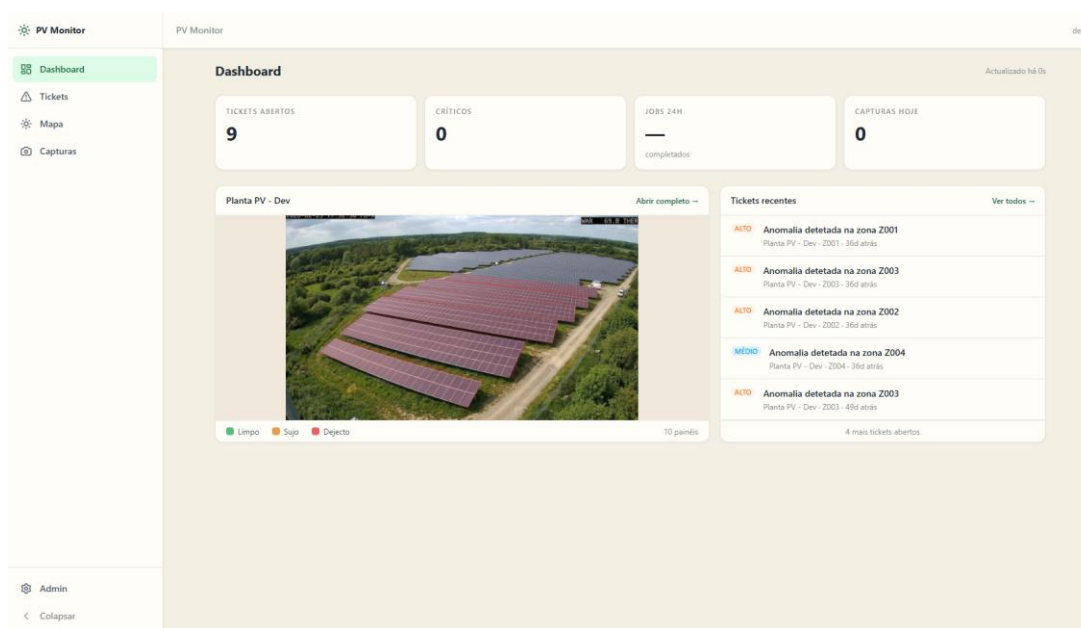


Figura 11 - Dashboard geral da aplicação, com indicadores operacionais e tickets recentes

A página de mapa permite visualizar a planta fotovoltaica e os elementos espaciais configurados no sistema. Nesta vista, podem ser apresentados o *base-map* da planta, as zonas definidas, os painéis representados no layout e os estados associados às ocorrências. A sobreposição de informação visual sobre a planta permite compreender rapidamente onde se localizam as deteções e que áreas da instalação podem estar afetadas. Esta funcionalidade é uma das mais

importantes da aplicação, uma vez que materializa o resultado do mapeamento geoespacial efetuado no *backend* com recurso ao *PostGIS*.

A **Figura 12** apresenta a vista de mapa da planta fotovoltaica. Nesta interface, o utilizador pode visualizar o *base-map* da instalação, as zonas e painéis configurados, bem como a respetiva classificação operacional. Esta vista é central para interpretar espacialmente os resultados da inferência.

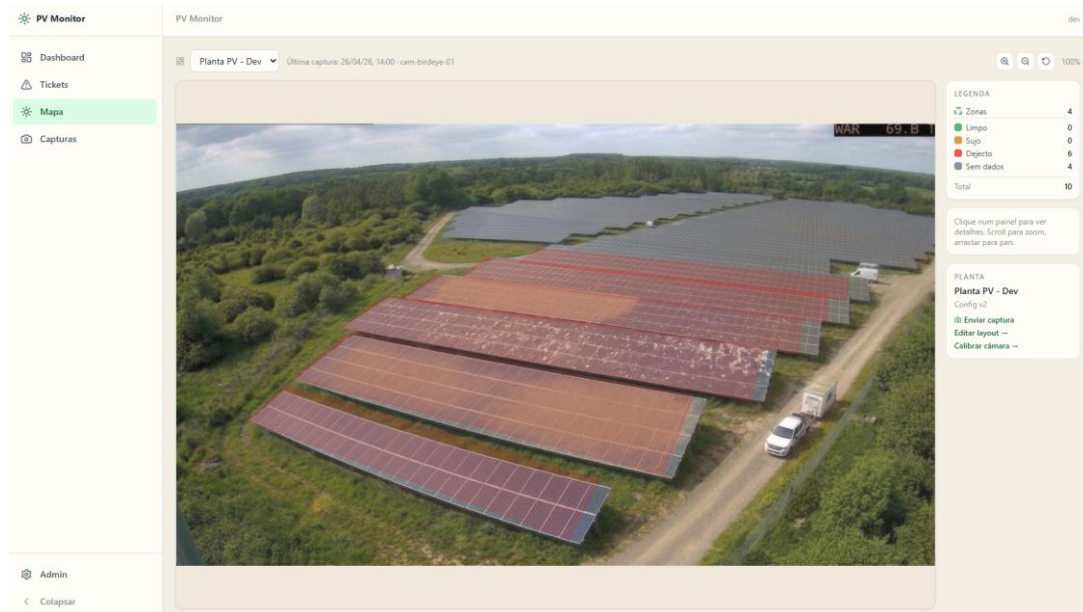


Figura 12 - Visualização da planta fotovoltaica com zonas, painéis e estados operacionais sobrepostos ao base-map

A página de capturas permite consultar as imagens submetidas ao sistema, bem como os respetivos metadados, estado de processamento e resultados associados. Esta vista dá acesso à informação relativa à câmara, data de captura, planta, job correspondente e inferências realizadas. Para além disso, a aplicação permite a submissão manual de imagens, funcionalidade útil para testes controlados, validação do pipeline e comparação entre diferentes cenários de entrada. Esta submissão manual complementa a componente *edge*, permitindo testar o comportamento do backend sem depender exclusivamente de uma captura em tempo real.

Para além da captura automática pela componente *edge*, o sistema permite a submissão manual de imagens através da interface *web*. A **Figura 13** apresenta o formulário de nova captura, no qual o utilizador pode selecionar a planta, a câmara e o ficheiro de imagem a enviar para processamento.

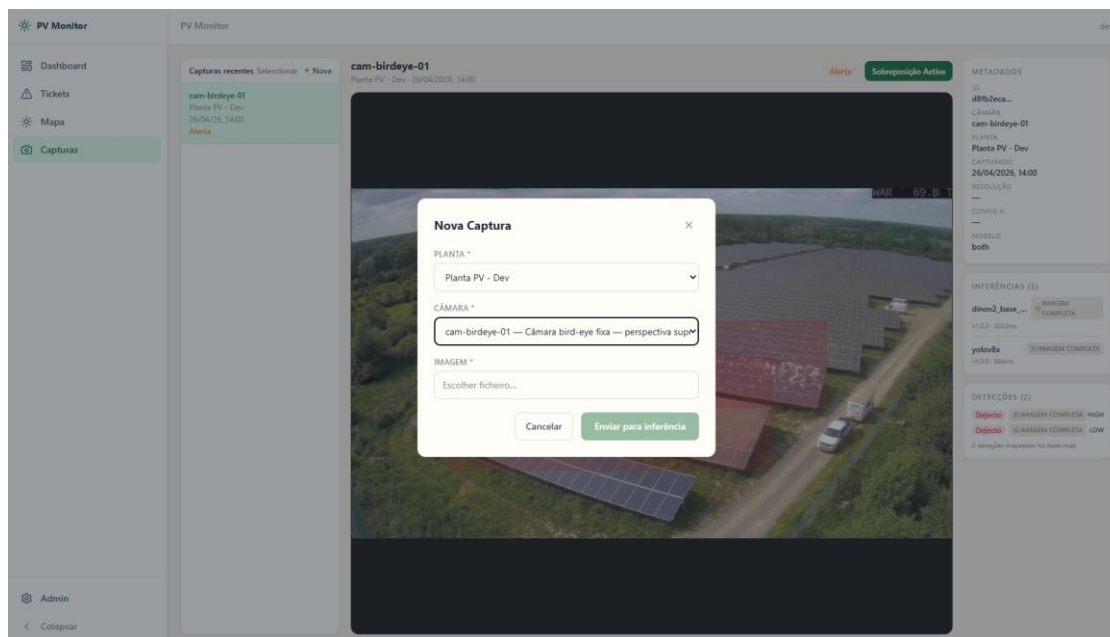


Figura 13 - Interface de submissão manual de capturas, permitindo associar uma imagem à planta e à câmara correspondente

A página de *tickets* apresenta as ocorrências geradas ou atualizadas a partir das deteções e do mapeamento espacial. Cada ticket representa uma situação que pode exigir análise ou intervenção, sendo caracterizado por informação como planta, zona, painel, severidade, estado e data de criação ou atualização. A existência de filtros por estado, severidade ou planta facilita a consulta e permite priorizar ocorrências. Esta funcionalidade aproxima o protótipo de um cenário de operação e manutenção, no qual os resultados automáticos devem ser organizados em elementos acionáveis.

As ocorrências geradas pelo sistema são apresentadas na página de *tickets*, ilustrada na **Figura 14**. Esta página permite consultar anomalias identificadas, filtrar resultados por planta, estado e severidade, e acompanhar a informação operacional associada a cada ocorrência.

SEV	TÍTULO	PLANTA	ZONA	ESTADO	DETECTADO
ALTO	Anomalia detetada na zona Z001	Planta PV - Dev	Z001	Aberto	10/05/26, 13:41
ALTO	Anomalia detetada na zona Z003	Planta PV - Dev	Z003	Aberto	10/05/26, 13:39
ALTO	Anomalia detetada na zona Z002	Planta PV - Dev	Z002	Aberto	10/05/26, 13:39
MÉDIO	Anomalia detetada na zona Z004	Planta PV - Dev	Z004	Aberto	10/05/26, 13:39
ALTO	Anomalia detetada na zona Z003	Planta PV - Dev	Z003	Aberto	27/04/26, 17:40
ALTO	Anomalia detetada na zona Z001	Planta PV - Dev	Z001	Aberto	27/04/26, 17:40
ALTO	Anomalia detetada na zona Z002	Planta PV - Dev	Z002	Aberto	27/04/26, 17:40
ALTO	Anomalia detetada na zona Z004	Planta PV - Dev	Z004	Aberto	27/04/26, 17:40
ALTO	Anomalia detetada na zona Z001	Planta PV - Dev	Z001	Aberto	26/04/26, 15:00

Figura 14 - Página de tickets com filtragem por planta, estado e severidade

A aplicação inclui ainda funcionalidades de administração, layout e calibração. A área de *layout* permite criar e editar zonas e painéis, definindo manualmente os elementos espaciais que serão utilizados no cruzamento com as deteções. Esta abordagem é adequada ao carácter demonstrador do protótipo, uma vez que permite configurar a planta de forma controlada e verificar visualmente a correspondência entre os elementos definidos e o *base-map*. A área de calibração permite selecionar pontos correspondentes entre a imagem capturada e a representação da planta, suportando o alinhamento necessário para o mapeamento das deteções.

Outra funcionalidade relevante é a consulta detalhada de zonas ou ocorrências. Esta vista permite relacionar os resultados da inferência com a zona ou painel afetado, apresentando deteções provenientes dos modelos utilizados, como *YOLOv8x* e *DINOv2* [2]. Ao disponibilizar este nível de detalhe, a aplicação permite compreender a origem visual de um ticket ou de uma classificação de severidade, contribuindo para a transparência do processo e para a validação funcional do protótipo.

Por fim, a **Figura 15** apresenta o detalhe de uma zona com resultados de inferência. Nesta vista são mostradas as deteções associadas aos modelos integrados no protótipo, incluindo resultados provenientes do *YOLOv8x* e do *DINOv2*. Esta visualização permite analisar a localização das anomalias e compreender a informação que suporta a criação dos *tickets*.

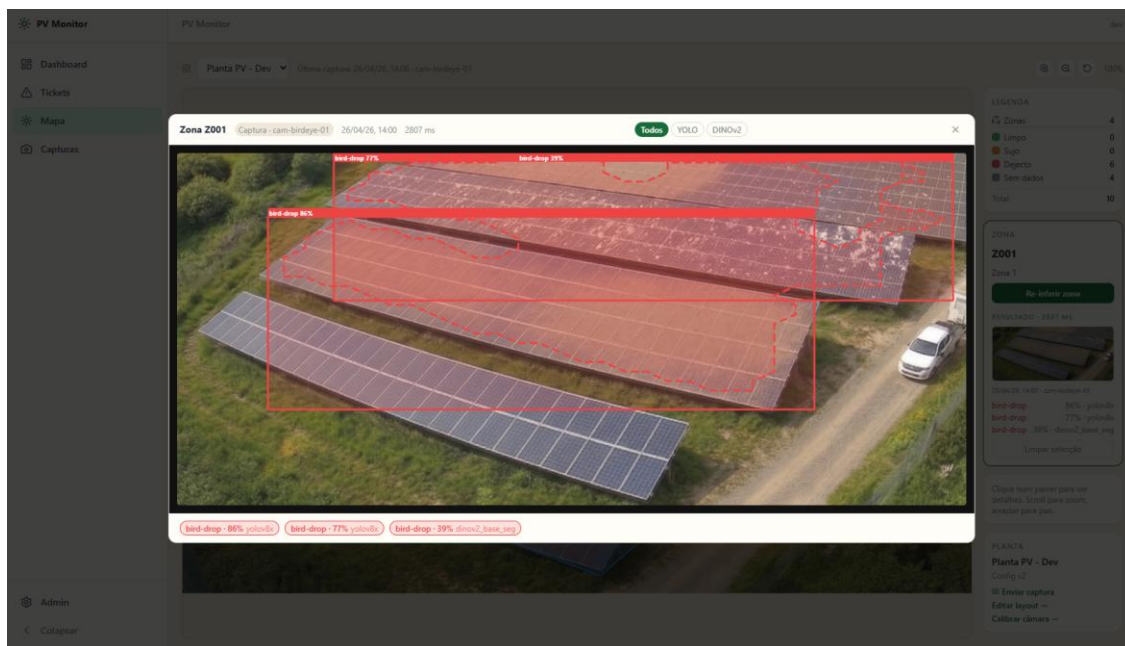


Figura 15 - Detalhe de uma zona com resultados de inferência YOLOv8x e DINOv2 associados à ocorrência analisada

Do ponto de vista funcional, a aplicação web demonstra a integração entre as várias componentes do sistema. O utilizador consegue acompanhar capturas, consultar o estado dos jobs, visualizar deteções, analisar a sua localização na planta, consultar *tickets* e gerir elementos de configuração. Desta forma, a interface não funciona apenas como uma camada estética, mas como uma ferramenta de suporte à decisão operacional, tornando acessível a informação produzida pelo pipeline de visão computacional e mapeamento geoespacial.

Em síntese, o frontend permite fechar o ciclo *end-to-end* do protótipo. A imagem capturada ou submetida é processada pelo backend, analisada pelo serviço de inferência, contextualizada no espaço da planta através de *PostGIS* e apresentada ao utilizador sob a forma de indicadores, mapas, capturas, *tickets* e detalhes operacionais [5]. Esta camada evidencia o principal objetivo do Projeto II: transformar a componente experimental estudada no Projeto I numa solução integrada, demonstrável e orientada para utilização prática em inspeção visual de centrais fotovoltaicas.

3. Testes de Validação

Neste capítulo são apresentados os testes de validação realizados ao protótipo desenvolvido. O objetivo principal foi confirmar o funcionamento do sistema enquanto prova de conceito, verificando se as diferentes componentes implementadas conseguem operar de forma integrada — desde a captura ou submissão da imagem até à apresentação dos resultados na aplicação web.

A validação realizada não pretendeu repetir a avaliação experimental dos modelos de visão computacional, uma vez que essa análise foi desenvolvida no âmbito do Projeto I. Neste relatório, o foco incide sobre a validação funcional do protótipo, isto é, sobre a capacidade do sistema para executar o fluxo *end-to-end* de inspeção visual automatizada. Foram assim observadas as etapas de captura, armazenamento temporário, envio para o backend, ingestão, criação de job, inferência, normalização dos resultados, mapeamento geoespacial, geração de *tickets* e visualização operacional.

Os testes foram realizados em ambiente controlado, com todos os serviços executados localmente através de containers Docker. Embora a arquitetura esteja preparada para integração com serviços *Azure*, nomeadamente *Blob Storage*, *Service Bus* e *Container Apps*, a validação descrita neste capítulo centra-se no funcionamento local do sistema [6]. Esta opção permitiu controlar os dados de entrada, repetir os testes de forma determinística e observar o comportamento de cada componente sem depender da disponibilização integral da infraestrutura *cloud*.

Para a validação foram utilizados dois tipos de imagens. Por um lado, recorreu-se a imagens de teste provenientes de *fixtures* locais, permitindo repetir cenários de forma controlada. Por outro lado, foram consideradas imagens obtidas através da componente sobre painéis PV reais, confirmando que o sistema consegue iniciar o fluxo a partir de captura *edge* por hardware. No entanto, o conjunto de imagens recolhidas em contexto real é ainda limitado, pelo que a validação apresentada deve ser entendida como validação funcional do protótipo e não como uma campanha sistemática de avaliação em terreno.

A primeira etapa validada correspondeu à captura da imagem pela componente *edge*. Nesta fase, o *Raspberry Pi*, em conjunto com a *Raspberry Pi Camera Module 3*, obtém imagens e inicia o fluxo de processamento. Para efeitos de teste, o sistema permite igualmente utilizar imagens previamente colocadas numa pasta local, garantindo maior controlo sobre os cenários analisados. Após a captura ou seleção, a imagem é colocada numa área de *pool* juntamente com os metadados necessários para a sua identificação, nomeadamente o identificador da planta, a câmara, o *timestamp* UTC e o *checksum* SHA-256 [1][3][14].

A aplicação web foi desenvolvida para permitir a consulta operacional dos resultados produzidos pelo sistema e a configuração dos elementos necessários

ao mapeamento espacial. Para além do utilizador da interface, foi considerada também a componente *edge*, responsável pelo envio automático de capturas para processamento.

A **Figura 16** apresenta os principais casos de uso do protótipo. O utilizador pode consultar o *dashboard*, visualizar o mapa da planta, consultar capturas, submeter uma nova captura, configurar a planta, o layout e a calibração, e consultar tickets. A componente *edge* envia capturas automaticamente, dando início ao processamento que gera resultados visuais, atualiza o mapa e cria ou atualiza *tickets* operacionais.

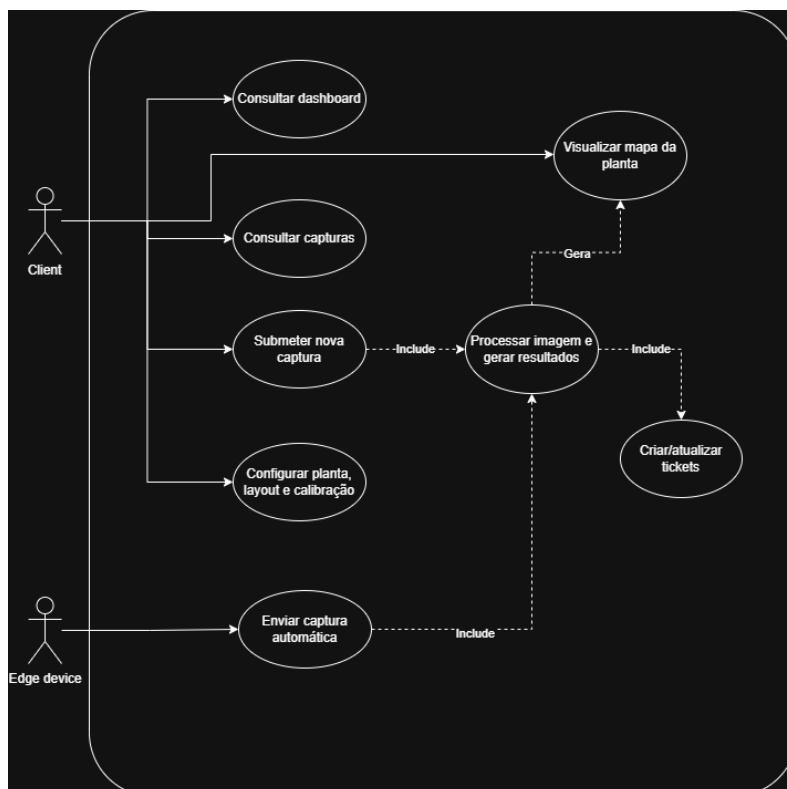


Figura 16 - Diagrama de casos de uso da aplicação web e da componente edge

A **Figura 17** apresenta um registo de execução da componente *edge*, no qual é possível observar a utilização do comando *rpicam-still*, a geração de identificadores de captura e a escrita das imagens na diretoria de *spool*. Esta evidência confirma que o protótipo consegue iniciar o fluxo a partir de captura por hardware, embora a validação sistemática em campo permaneça como trabalho futuro.

```

pi@photovoltaicos: ~/fotovoltaicos_II/edge/edge-capture
File Edit Tabs Help
pi@photovoltaicos:~/fotovoltaicos_II/edge/edge-capture $ .venv/bin/activate
(.venv) pi@photovoltaicos:~/fotovoltaicos_II/edge/edge-capture $ PLANT_ID=00000000-0000-0000-0000-000000000001 CAMERA_ID=cam-birdeye-01 CAPTURE_SOURCE_MODE=rpical CAP
TURE_INTERVAL_SECONDS=60 SPOOL_ROOT=/var/lib/pv-spool python run.py
2026-06-22 12:19:33,698 INFO edge-capture edge-capture started source_mode=rpical fixture_directory=/fixtures interval_seconds=60 pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:19:33,699 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-0d66cc6f-2288-4381-98fc-c4efcd75181.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:19:36,947 INFO edge-capture capture written capture_id=020ad65b-ddf8-43c1-b2f3-0f1a5be598ae source=rpical-still pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:20:36,948 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-ccl16f5db-75c6-4bf2-82b4-ce749aa397e6.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:21:39,725 INFO edge-capture capture written capture_id=9aa181b8-2ba3-481a-bfaf-117727d8fb2e source=rpical-still pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:21:39,725 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-bbd7658-832f-45e6-8e5a-e0f8d24b43a3.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:22:42,634 INFO edge-capture capture written capture_id=7363b1ab-07c7-49d4-9336-8674a03e749 source=rpical-still pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:22:42,635 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-696e6a40-613a-4686-becd-a3a97f8ff076.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:23:45,474 INFO edge-capture capture written capture_id=de3836eb-0873-41ce-93a1-34448402f132 source=rpical-still pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:23:45,475 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-2773ebc3-f891-42b0-94cf-d2b6e749c11b.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:24:48,302 INFO edge-capture capture written capture_id=b606fd55-5e2c-4520-a9b-9247a9c0347c source=rpical-still pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:24:48,303 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-d80b5cc7-c0bc-4f46-8b13-8851d2a1efdc.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:25:51,177 INFO edge-capture capture written capture_id=9d79265b-e395-4bab-9a68-44285e7b0eae source=rpical-still pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:25:51,178 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-4f1ce077-63a6-4198-a120-45084a446254.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:26:54,051 INFO edge-capture capture written capture_id=c4fdf170-08a2-44e3-b256-5a6de627e9a5 source=rpical-still pending_dir=/var/lib/pv-spool/pending
2026-06-22 12:26:54,052 INFO edge-capture capturing image with rpical command=rpical-still -o /var/lib/pv-spool/tmp/rpical-2d00d6c9-56eb-4a56-81ac-c74b5ab49a1b.jpg --
timeout 2000 -quality 90 --nopreview
2026-06-22 12:26:56,912 INFO edge-capture capture written capture_id=817cf429-e3d3-429f-8019-424f8e0b3531 source=rpical-still pending_dir=/var/lib/pv-spool/pending

```

Figura 17 - Registo de execução da componente edge durante a captura de imagens com Raspberry Pi Camera Module 3

De seguida, foi validado o envio da imagem para a *ingestion-api*. O *edge-uploader* lê as imagens presentes na *spool* por ordem temporal, associa os metadados correspondentes e procede ao envio para o *PV* através de um pedido HTTP *multipart*. Em caso de falha de rede, o sistema aplica mecanismos de repetição e mantém a captura disponível para novo envio, evitando que uma falha temporária provoque perda definitiva de dados.

Foi também verificado o mecanismo de deduplicação. A *ingestion-api* utiliza uma restrição de unicidade baseada na combinação entre *checksum* e identificador da câmara, garantindo que a mesma captura não é registada mais do que uma vez. Quando é recebido um pedido duplicado, a API devolve uma resposta de sucesso com indicação de idempotência, sem criar registos adicionais na BD.

Após a receção da imagem, a *ingestion-api* valida os dados recebidos, armazena o ficheiro no armazenamento configurado para o ambiente de teste, insere o registo da captura e cria um *job* de processamento no estado *PENDING*. Este *job* permite acompanhar o percurso da imagem ao longo do *pipeline*. Quando o *core-worker* consome o *job*, o estado transita para *IN_PROGRESS*, indicando que a captura entrou na fase de processamento. Esta transição foi validada através da observação, dos registos, do sistema e da consulta dos dados apresentados na aplicação web.

Na etapa seguinte, o *core-worker* invoca o serviço de inferência, designado por *ml-serving*, que executa os modelos de visão computacional integrados no protótipo. O YOLOv8x é utilizado para deteção de regiões de interesse, enquanto o DINOv2 funciona como abordagem complementar para obtenção de informação visual mais rica. O serviço devolve os resultados num contrato *JSON* normalizado, incluindo classes identificadas, níveis de confiança, *bounding boxes*

e, quando aplicável, informação espacial adicional associada às regiões analisadas.

Após a receção da resposta, o *core-worker* normaliza os resultados: filtra deteções abaixo de um limiar de confiança configurável, uniformiza os nomes das classes e gera uma imagem de *overlay* com as deteções desenhadas sobre a captura original. Esta imagem derivada é armazenada na área de resultados derivados, permitindo a sua consulta posterior na aplicação web. Os resultados visuais da inferência encontram-se ilustrados na **Figura 15**, onde é apresentado o detalhe de uma zona com deteções associadas aos modelos YOLOv8x e DINOv2.

Depois da inferência, foi validado o mapeamento das deteções sobre o *layout* da planta fotovoltaica. Esta etapa utiliza a calibração previamente definida para projetar as coordenadas das deteções do espaço da imagem para o espaço do *base-map* da planta. Em seguida, o PostgreSQL/PostGIS realiza testes de interseção espacial entre as geometrias das deteções mapeadas e os polígonos dos painéis e zonas configurados [5][7].

Durante os testes, verificou-se que as deteções eram associadas às zonas e painéis correspondentes, e que o sistema permitia distinguir situações em que a deteção não intersectava nenhum elemento do *layout*. A visualização deste mapeamento encontra-se representada na **Figura 12**, através da sobreposição de zonas, painéis e estados operacionais sobre o *base-map* da planta.

Com base no resultado do mapeamento, o sistema calcula a severidade da ocorrência, considerando critérios como a área afetada, o nível de confiança e a zona ou painel associado, e cria ou atualiza *tickets* operacionais. Esta funcionalidade foi validada pela observação dos *tickets* gerados na aplicação web, verificando a sua associação à planta, zona e nível de severidade correspondentes.

Esta etapa é relevante porque demonstra que o protótipo não se limita a produzir resultados visuais, mas converte-os em elementos acionáveis para apoio à operação e manutenção. A página de *tickets*, apresentada na **Figura 14**, evidencia a organização das ocorrências por severidade, planta, zona, estado e data de deteção.

Por fim, foi validada a apresentação dos resultados na aplicação web. O *frontend* permite consultar o *dashboard* geral, o mapa da planta com sobreposição de deteções, o detalhe de cada captura com *overlay* de inferência, a lista de *tickets* com filtros e histórico, e as áreas de administração de *layout* e calibração. Durante os testes, verificou-se que a informação produzida pelo PV era apresentada de forma coerente e que o utilizador conseguia acompanhar o estado das inspeções e consultar os resultados associados a cada captura. Esta validação é apoiada pelas figuras da aplicação web apresentadas na secção 2.3.5.

A validação foi organizada de acordo com as principais etapas do *pipeline*, desde a captura da imagem até à apresentação dos resultados na aplicação *web*. A **Tabela 2** sintetiza os principais testes funcionais realizados ao protótipo. Esta tabela não pretende apresentar uma avaliação quantitativa do desempenho dos modelos, mas sim organizar a validação do fluxo implementado, identificando os testes realizados, o objetivo de cada teste, o resultado esperado, o resultado observado e o respetivo estado de validação.

Tabela 2 - Estrutura utilizada para documentação dos testes de validação

Teste	Objetivo	Resultado esperado	Resultado obtido	Estado
Captura no edge	Validar a obtenção da imagem	Imagem capturada ou selecionada	Imagem obtida pelo sistema	Validado
Spool local	Validar armazenamento temporário	Imagem e metadados guardados	Captura registada na spool	Validado
Upload para ingestion-api	Validar envio para o backend	Imagem enviada com metadados	Imagem recebida pela API	Validado
Deduplicação	Evitar registos repetidos	Captura duplicada não é recriada	A API devolve 200 com idempotent=true; captura não duplicada na BD	Validado
Ingestão	Validar receção e registo	Captura aceite pelo backend	Captura registada no sistema	Validado
Criação do job	Validar criação do processamento	Job criado em estado PENDING	Job associado à captura	Validado
Processamento assíncrono	Validar evolução do job	Estados atualizados no pipeline	Job processado pelo worker	Validado
Inferência	Validar chamada ao ml-serving	Deteções devolvidas pelo modelo	Resultados associados à imagem	Validado
Normalização	Uniformizar resultados	Deteções em formato comum	Deteções normalizadas	Validado
Mapeamento PostGIS	Associar deteções ao layout	Zona/painel identificado	Deteções mapeadas no layout	Validado
Severidade	Classificar a ocorrência	Severidade atribuída	Severidade apresentada	Validado
Tickets	Gerar ocorrência operacional	Ticket criado ou atualizado	Ticket apresentado na UI	Validado
Aplicação web	Validar apresentação dos dados	Resultados visíveis na interface	Dados apresentados na UI	Validado

De forma geral, os testes realizados permitiram confirmar o funcionamento integrado do protótipo em ambiente controlado. A imagem capturada ou submetida é recebida pelo backend, associada a um *job*, processada pelo serviço de inferência, relacionada com o layout da planta e apresentada na interface *web* sob a forma de resultados visuais, severidade e tickets. Esta validação demonstra a viabilidade funcional da solução proposta enquanto protótipo demonstrador.

Importa, no entanto, reconhecer algumas limitações. O número de imagens captadas em contexto real é ainda reduzido e não foi realizada uma campanha extensa numa central fotovoltaica em operação. Fatores como variações de luminosidade, reflexos, sombras, sujidade difusa e diferentes ângulos de captura poderão influenciar o desempenho do sistema em ambiente real. Assim, os testes apresentados validam o funcionamento do *pipeline*, mas não substituem uma avaliação operacional futura em terreno.

A validação foi organizada de forma sistemática, distinguindo entre funcionalidades implementadas, componentes testadas em ambiente controlado e limitações a considerar em trabalho futuro. Os testes descritos devem, por isso, ser interpretados como validação funcional do protótipo e não como avaliação operacional definitiva em contexto real de central fotovoltaica.

4. Conclusões e Trabalho Futuro

O presente trabalho teve como objetivo desenvolver e validar um protótipo funcional de uma plataforma distribuída para inspeção visual de centrais fotovoltaicas. Dando continuidade ao Projeto I, no qual foram estudadas e avaliadas abordagens de visão computacional para detecção e segmentação de sujidade em módulos PV, o Projeto II centrou-se na transformação dessa componente experimental numa solução integrada, capaz de suportar um fluxo end-to-end de inspeção visual.

O protótipo desenvolvido permitiu demonstrar a viabilidade de uma arquitetura modular, composta por componentes de *edge*, *backend*, serviço de inferência, BD geoespacial e aplicação web. A solução implementada permite capturar ou submeter imagens, organizar o seu envio através de mecanismos de *spool*, *checksum* e *retries*, receber as capturas no backend, criar jobs assíncronos, executar inferência com modelos de visão computacional, normalizar detecções, mapear resultados sobre o *layout* da planta, calcular severidade, gerar tickets e apresentar a informação numa interface web.

Uma das principais contribuições do trabalho consistiu na integração dos modelos de visão computacional num fluxo operacional mais amplo. Em vez de apresentar apenas resultados isolados de inferência, como *bounding boxes*, máscaras ou classificações, o sistema desenvolvido procura contextualizar esses resultados no espaço físico da central fotovoltaica. Para esse efeito, a utilização de *PostgreSQL/PostGIS* revelou-se relevante, permitindo relacionar detecções com zonas e painéis definidos no *layout* da planta. Esta etapa é essencial para transformar informação visual em informação operacional, útil para apoio à decisão [5][7].

Outra contribuição importante foi a implementação de uma aplicação web orientada para a visualização e acompanhamento dos resultados. A interface permite consultar indicadores gerais, visualizar a planta e os elementos configurados, analisar capturas processadas, acompanhar tickets e aceder a funcionalidades de administração, layout e calibração. Desta forma, o protótipo aproxima a componente técnica de visão computacional de um cenário de utilização por equipas de operação e manutenção.

Os testes realizados permitiram validar funcionalmente o *pipeline* em ambiente controlado. Foi possível confirmar que uma imagem capturada ou submetida é recebida pelo backend, associada a um job, processada pelo serviço de inferência, relacionada com o *layout* da planta e apresentada na aplicação web sob a forma de resultados visuais, severidade e tickets. Esta validação demonstra que o protótipo cumpre o seu objetivo enquanto prova de conceito, embora não deva ser interpretado como um produto final ou como uma solução pronta para produção.

Apesar dos resultados obtidos, existem limitações que importa reconhecer. A validação foi realizada sobretudo em ambiente local, com serviços executados em localhost e, quando aplicável, através de containers *Docker*. Embora a arquitetura esteja preparada para integração com serviços *Azure*, essa componente deverá ser consolidada em fases posteriores. Além disso, o número de imagens recolhidas em contexto real é ainda limitado, não tendo sido realizada uma campanha extensa numa central fotovoltaica em operação. Esta limitação impede que os resultados sejam interpretados como uma avaliação operacional completa em terreno [6].

A componente de captura por hardware encontra-se funcional no protótipo, mas deverá ser validada com maior profundidade em ambiente real. A realização de testes em campo permitirá observar o comportamento do sistema perante condições ambientais variáveis, como alterações de luminosidade, reflexos, sombras, diferentes ângulos de captura, sujidade difusa e variações na orientação dos painéis. Estes fatores poderão influenciar tanto a qualidade das imagens como o desempenho dos modelos de inferência e a fiabilidade do mapeamento espacial.

Outra limitação prende-se com a calibração e o layout da planta. O protótipo privilegia uma abordagem manual e controlada, baseada na definição de zonas, painéis e pontos de calibração. Esta opção é adequada para uma prova de conceito, pois permite maior transparência e controlo sobre o mapeamento geoespacial. No entanto, numa evolução futura, será importante melhorar os mecanismos de calibração, reduzir o esforço manual de configuração e explorar formas de automatizar parcialmente o alinhamento entre imagem capturada e *base-map* da planta.

Como trabalho futuro, considera-se prioritária a realização de uma campanha de recolha em terreno real, utilizando o *Raspberry Pi* e a câmara em contexto de central fotovoltaica. Esta etapa permitirá reunir um conjunto de imagens mais representativo, validar o comportamento do sistema em condições reais e identificar ajustes necessários ao posicionamento da câmara, à calibração, ao processamento e à interface. A recolha de dados reais será também importante para reforçar o *dataset* e melhorar a capacidade dos modelos perante diferentes tipos de sujidade, iluminação e condições ambientais [1].

Será igualmente relevante consolidar a integração *cloud* da solução. Embora a arquitetura tenha sido preparada para serviços *Azure*, como *Blob Storage*, *Service Bus*, *Container Apps*, *ACR*, *Key Vault* e *Azure Monitor*, a validação descrita neste trabalho foi centrada em ambiente local. Uma evolução futura deverá passar pela implantação completa destes serviços, permitindo avaliar aspetos como escalabilidade, monitorização, segurança, gestão de erros, filas de mensagens, *dead-letter queues* e observabilidade do sistema [6].

Do ponto de vista da segurança e gestão de utilizadores, poderá ainda ser explorada a integração com mecanismos de autenticação e autorização, como

Entra ID e controlo de acessos baseado em papéis. Esta evolução seria importante caso o protótipo fosse aproximado de um cenário real de utilização, no qual diferentes perfis de utilizador poderiam ter permissões distintas para consultar resultados, gerir *tickets*, alterar layouts ou configurar câmaras.

Ao nível funcional, o sistema poderá ser expandido com relatórios para equipas de operação e manutenção, mecanismos de priorização mais avançados, comparação contínua entre modelos, histórico temporal de ocorrências e suporte a novas classes de anomalias. Para além da sujidade, poderão ser consideradas outras situações relevantes em centrais fotovoltaicas, como sombras persistentes, danos visíveis, objetos sobre os módulos ou padrões anómalos associados a degradação superficial.

Em conclusão, o Projeto II permitiu transformar o trabalho experimental desenvolvido no Projeto I numa solução demonstradora integrada. O protótipo desenvolvido evidencia que é possível articular captura de imagens, *PV*, inferência, mapeamento geoespacial, geração de tickets e visualização numa aplicação *web*, criando uma base funcional para apoio à inspeção visual de centrais fotovoltaicas. Embora existam limitações e trabalho futuro a realizar, a solução apresentada demonstra a viabilidade técnica da abordagem e constitui um ponto de partida consistente para evoluções posteriores em contexto real.

5. Referências

- [1] “Raspberry Pi 4 Model B,” 2026.
- [2] “Explore Ultralytics YOLOv8 - Documentação Ultralytics YOLO.” Accessed: Dec. 06, 2025. [Online]. Available: <https://docs.ultralytics.com/pt/models/yolov8>
- [3] “Buy a Raspberry Pi Camera Module 3 – Raspberry Pi.” Accessed: Jun. 22, 2026. [Online]. Available: <https://www.raspberrypi.com/products/camera-module-3/>
- [4] “Camera software - Raspberry Pi Documentation.” Accessed: Jun. 22, 2026. [Online]. Available: https://www.raspberrypi.com/documentation/computers/camera_software.html
- [5] “Documentation | PostGIS.” Accessed: Jun. 22, 2026. [Online]. Available: <https://postgis.net/documentation/>
- [6] “Azure documentation | Microsoft Learn.” Accessed: Jun. 22, 2026. [Online]. Available: <https://learn.microsoft.com/en-us/azure/?product=popular>
- [7] “PostgreSQL: Documentation.” Accessed: Jun. 22, 2026. [Online]. Available: <https://www.postgresql.org/docs/>
- [8] “React.” Accessed: Jun. 22, 2026. [Online]. Available: <https://react.dev/>
- [9] “TypeScript: JavaScript With Syntax For Types.” Accessed: Apr. 02, 2026. [Online]. Available: <https://www.typescriptlang.org/>
- [10] “Docker Compose | Docker Docs.” Accessed: Jun. 22, 2026. [Online]. Available: <https://docs.docker.com/compose/>
- [11] “Raspberry Pi OS - Raspberry Pi Documentation.” Accessed: Jun. 22, 2026. [Online]. Available: <https://www.raspberrypi.com/documentation/computers/os.html>
- [12] “FastAPI - FastAPI.” Accessed: Jun. 22, 2026. [Online]. Available: <https://fastapi.tiangolo.com/>
- [13] “TanStack | The open-source application stack for the web.” Accessed: Jun. 22, 2026. [Online]. Available: <https://tanstack.com/>
- [14] “RFC 6234 - US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF).” Accessed: Jun. 22, 2026. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc6234>