

Desenvolvimento de sistema de apoio à manutenção preventiva em contexto industrial

GONÇALO FERNANDES SILVA SECO

Julho de 2026

DESENVOLVIMENTO DE SISTEMA DE APOIO À MANUTENÇÃO PREVENTIVA EM CONTEXTO INDUSTRIAL

Gonçalo Fernandes Silva Seco

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Sistemas e Planeamento Industrial

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de
Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Gonçalo Fernandes Silva Seco, Nº 1180849, 1180849@isep.ipp.pt

Orientação científica: Professor Carlos José Campos, crc@isep.ipp.pt

Empresa: Controlar – Sistemas Industriais e Eletrónica LDA

Supervisão: Carlos Junio Rocha, carlosrocha@pt.controlar.com



Departamento de Engenharia Eletrotécnica
Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Sistemas e Planeamento Industrial

2026

Agradecimentos

A realização deste trabalho não teria sido possível sem o apoio de várias pessoas a quem devo um reconhecimento sincero.

Ao Professor Doutor Carlos Campos, orientador deste trabalho, o primeiro agradecimento. A disponibilidade que demonstrou ao longo de todo o processo foi determinante para o rumo e a qualidade do resultado final. O acompanhamento próximo, os comentários críticos e a abertura para discutir cada detalhe da solução foram uma ajuda que excedeu em muito o que seria expectável de um orientador.

À minha namorada, o agradecimento mais sentido. A paciência que demonstrou ao longo deste percurso, os momentos que abdicou e a disponibilidade com que sempre esteve presente foram determinantes para que este trabalho chegasse a bom porto. O seu apoio constante nos momentos de maior pressão valeu mais do que as palavras conseguem expressar.

Aos meus pais, por tudo o que tornaram possível. A dedicação, o esforço e o incentivo que sempre colocaram no meu percurso foram a base sobre a qual este trabalho foi construído. A confiança que depositaram em mim ao longo dos anos é uma dívida que não se líquida com agradecimentos, mas que reconheço com genuína gratidão.

Aos meus colegas de trabalho na Controlar, pela disponibilidade, pelo conhecimento partilhado e pelo acompanhamento ao longo do desenvolvimento da solução. A proximidade com o contexto real de operação industrial, possibilitada pelo trabalho em equipa no dia-a-dia, foi essencial para que a solução respondesse a necessidades concretas e não apenas a requisitos abstratos.

Resumo

Equipamentos industriais requerem manutenção regular para assegurar a sua disponibilidade e fiabilidade ao longo do ciclo de vida. A ausência de dados operacionais estruturados sobre o desgaste e a utilização de componentes instalados nas máquinas de testes desenvolvidas pela Controlar, impede a antecipação de falhas e impossibilita a tomada de decisões técnicas fundamentadas em evidências reais. Estas paragens não planeadas contribuem também para um impacto negativo e dispendioso nos clientes. Para colmatar estes problemas foi desenvolvido um sistema de apoio à manutenção preventiva em contexto industrial, adaptado às limitações da abordagem reativa praticada na Controlar. A solução desenvolvida responde a este problema através de uma arquitetura híbrida composta por dois subsistemas complementares: um sistema local, orientado ao registo e acompanhamento operacional junto da máquina em ambiente offline, e um sistema central online, destinado à consolidação e análise de dados provenientes de múltiplas instalações.

O sistema local foi implementado com uma arquitetura em camadas segundo o padrão MVC, assente em Python com Flask na camada de serviços, MySQL na camada de persistência e uma interface *web* desenvolvida em tecnologias *web* padrão sem dependências externas.

O sistema desenvolvido regista o desgaste dos componentes com base em ciclos de funcionamento reais, gera alertas preventivos organizados em três níveis de severidade, garante rastreabilidade completa através de um log de auditoria associado a cada utilizador autenticado e disponibiliza exportação de dados em formato JSON para integração com o sistema central. O controlo de acesso é implementado por perfis (administrador, operador de manutenção e observador) com aplicação efetiva ao nível dos serviços independentemente da interface. O sistema central agrega os dados exportados pelas instâncias locais, computa estatísticas de vida útil por tipo de componente, incluindo média, percentil cinquenta e percentil noventa, e disponibiliza um relatório de

recomendações parametrizável que apoia decisões de engenharia sobre seleção e dimensionamento de componentes com base em dados reais de operação.

A solução foi validada funcionalmente através de verificação incremental módulo a módulo e implantada na máquina RiTS ControlarS2.8 da Controlar, onde o sistema local operou em integração efetiva com o NI TestStand, acumulando dados reais de utilização. Os dados exportados pela instância local, foram ingeridos pelo sistema central, que calculou estatísticas de vida útil e gerou recomendações de substituição baseadas em observações reais de múltiplos componentes. Os resultados confirmam que o sistema cumpre os requisitos funcionais especificados e responde ao problema identificado, estabelecendo as condições necessárias para a transição de um modelo de manutenção reativo para uma abordagem preventiva orientada por dados, com capacidade de evolução futura para análise preditiva baseada em modelos de fiabilidade.

Palavras-Chave

Manutenção industrial preventiva, Rastreabilidade, Gestão de manutenção; Ciclos de vida de componentes, Análise estatística, Arquitetura híbrida, Indústria 4.0.

Abstract

Industrial equipment requires regular maintenance to ensure its availability and reliability throughout its operational lifecycle. The absence of structured operational data regarding the wear and utilisation of components installed in the test systems developed by Controlar prevents the anticipation of failures and hinders technically sound maintenance decisions based on real operational evidence. These unplanned downtimes also have a significant and costly impact on customers. To address these limitations, a preventive maintenance support system for industrial environments was developed, tailored to overcome the constraints of the reactive maintenance approach currently adopted at Controlar.

The proposed solution is based on a hybrid architecture comprising two complementary subsystems: a local offline system for operational data collection and maintenance tracking at machine level, and a central online system responsible for consolidating and analysing data gathered from multiple installations.

The local system was implemented using a layered architecture following the Model–View–Controller (MVC) design pattern, with Python and Flask providing the service layer, MySQL serving as the persistence layer, and a web-based user interface developed using standard web technologies without external frameworks.

The developed system monitors component wear based on actual operating cycles, generates preventive maintenance alerts organised into three severity levels, ensures full traceability through an audit log associated with each authenticated user, and supports data export in JSON format for integration with the central system. Access control is enforced through role-based permissions (administrator, maintenance operator, and observer), with authorisation implemented at the service layer independently of the user interface. The central system aggregates data exported from local instances, computes component lifetime statistics by component type, including the mean, median, and 90th percentile, and generates configurable recommendation reports to support engineering

decisions regarding component selection and maintenance intervals based on real operational data.

The proposed solution was functionally validated through incremental module-by-module verification and deployed on the Controlar RiTS ControlarS2.8 test system, where the local subsystem operated in full integration with NI TestStand and collected real operational data. The exported data were successfully ingested by the central system, which calculated component lifetime statistics and generated replacement recommendations based on observations collected from multiple components. The results demonstrate that the proposed system satisfies the specified functional requirements and effectively addresses the identified problem, establishing the foundations for the transition from a reactive maintenance strategy to a data-driven preventive maintenance approach, while providing a scalable basis for future predictive maintenance supported by reliability models.

Keywords

Preventive maintenance; industrial maintenance; traceability; maintenance management; component life cycles; statistical analysis; hybrid architecture; Industry 4.0

Índice

AGRADECIMENTOS	I
RESUMO	III
ABSTRACT	V
ÍNDICE.....	IX
ÍNDICE DE FIGURAS.....	XI
ÍNDICE DE TABELAS.....	XV
ACRÓNIMOS	XVII
1. INTRODUÇÃO	1
1.1.CONTEXTUALIZAÇÃO.....	4
1.2.DESCRICÃO DO PROBLEMA.....	4
1.3.OBJETIVOS.....	7
1.4.ABORDAGEM E FASES DO PROJETO.....	9
1.5.CALENDARIZAÇÃO DO PROJETO	10
1.6.ORGANIZAÇÃO DO DOCUMENTO	11
2. MANUTENÇÃO INDUSTRIAL E TECNOLOGIAS ENVOLVIDAS	13
2.1. ESTRATÉGIAS DE MANUTENÇÃO INDUSTRIAL.....	14
2.2.MANUTENÇÃO PREVENTIVA: CONCEITOS E PRÁTICAS.....	17
2.3.LINGUAGENS E FERRAMENTAS UTILIZADAS.....	20
3. ARQUITETURA E DESIGN DA SOLUÇÃO.....	25
3.1.ANÁLISE DE DOMÍNIO.....	25
3.2.ANÁLISE DE REQUISITOS	28
3.2.1.REQUISITOS FUNCIONAIS DO SISTEMA LOCAL	29
3.2.2.REQUISITOS FUNCIONAIS DO SISTEMA CENTRAL.....	31
3.2.3.REQUISITOS NÃO FUNCIONAIS.....	32
3.3.CASOS DE USO.....	34
3.3.1.SISTEMA LOCAL.....	34
3.3.2.SISTEMA CENTRAL.....	35
3.4.DESIGN DA BASE DE DADOS.....	37
3.4.1.BASE DE DADOS DO SISTEMA LOCAL.....	37
3.4.2.BASE DE DADOS DO SISTEMA CENTRAL.....	39

3.5.DESIGN DA SOLUÇÃO DE INTERAÇÃO.....	40
3.5.1.ARQUITETURA GERAL.....	41
3.5.2.DESIGN DO SISTEMA LOCAL.....	41
3.5.3.DESIGN DO SISTEMA CENTRAL.....	43
4. IMPLEMENTAÇÃO DO SISTEMA.....	45
4.1.INTERAÇÃO COM O UTILIZADOR	46
4.1.1.ESTRUTURA E NAVEGAÇÃO	47
4.1.2.VISUALIZAÇÃO DE DADOS E ALERTAS	48
4.2.CAMADA DE SERVIÇOS	52
4.2.1.EXPORTAÇÃO E INTEROPERABILIDADE	54
4.2.2.ESTRUTURA DO JSON DE EXPORTAÇÃO	55
4.2.3.NORMALIZAÇÃO E CATEGORIZAÇÃO NO SISTEMA CENTRAL.....	57
4.2.4.CONSolidAÇÃO Estatística.....	59
4.2.5.GESTÃO DE UTILIZADORES E ENCRIPTAÇÃO DE CREDENCIAIS	60
5. DEMONSTRAÇÃO E RESULTADOS.....	63
5.1.SISTEMA LOCAL DE MANUTENÇÃO	64
5.1.1.RITS COMO CONTEXTO DE APLICAÇÃO	65
5.1.2.CONFIGURAÇÃO DA INSTÂNCIA	65
5.1.3.OPERAÇÃO DE INTEGRAÇÃO COM O TESTSTAND	68
5.1.4.SISTEMA DE ALERTAS	70
5.1.5.RASTREABILIDADE E EXPORTAÇÃO.....	72
5.2.SISTEMA CENTRAL DE MANUTENÇÃO.....	74
5.2.1.AUTENTICAÇÃO E ACESSO	74
5.2.2.DASHBOARD E ATIVIDADE RECENTE	74
5.2.3.INGESTÃO DE DADOS	75
5.2.4.MÁQUINAS E HISTÓRICO DE COMPONENTES	76
5.2.5.ESTATÍSTICAS DE VIDA ÚTIL	77
5.2.6.RELATÓRIO DE RECOMENDAÇÕES	78
5.2.7.GESTÃO DE UTILIZADORES	79
6. CONCLUSÕES E TRABALHO FUTURO.....	81
6.1.CONCLUSÕES	82
6.2.LIMITAÇÕES	84
6.3.TRABALHO FUTURO	85
REFERÊNCIAS DOCUMENTAIS.....	87

Índice de Figuras

Figura 1	Áreas de negócio da Controlar	2
Figura 2	Exemplos de máquinas desenvolvidas pela Controlar	3
Figura 3	Diagrama de Gantt do projeto	11
Figura 4	<i>Technology stack</i> do sistema local e central	24
Figura 5	CMTS (esquerda) e RiTS (direita)	26
Figura 6	Diagrama de fluxo de dados do sistema geral	28
Figura 7	Diagrama de casos de uso do sistema local	34
Figura 8	Diagrama de casos de uso do sistema central	36
Figura 9	Esquema relacional de base de dados do sistema local	37
Figura 10	Esquema relacional de base de dados do sistema central	39
Figura 11	Diagrama do sistema local	42
Figura 12	Diagrama do sistema central	44
Figura 13	Estrutura de ficheiros do sistema geral	46
Figura 14	Interface do sistema local	50
Figura 15	Dashboard sistema central	51
Figura 16	Aba de estatísticas do sistema central	52
Figura 17	Tabela de componentes	58
Figura 18	Avisos na interface do sistema central	60

Figura 19	RiTS	64
Figura 20	Configuração da instância RiTS ControlarS2.8	66
Figura 21	Biblioteca de componentes da instância RiTS	67
Figura 22	Biblioteca de ações da instância RiTS	67
Figura 23	Chamada da DLL ControlarMaintenanceClient com a função IncrementWorkingCycles	68
Figura 24	Histórico de incrementos automáticos gerados pelo TestStand	69
Figura 25	Componentes instalados com ciclos reais acumulados (perfil observador)	70
Figura 26	Três níveis de alerta ativos em simultâneo com gráfico de distribuição	71
Figura 27	Vista de alertas no perfil de observador (sem opção de descarte)	71
Figura 28	Tabela de componentes com codificação cromática por nível de utilização	72
Figura 29	Log de auditoria com operações realizadas na instância RiTS	73
Figura 30	Popup de confirmação de exportação antes do encerramento da sessão	73
Figura 31	Página de autenticação do sistema central	74
Figura 32	Dashboard do sistema central com três máquinas registadas e atividade recente	75
Figura 33	Histórico de uploads com dados de três máquinas distintas	76
Figura 34	Máquinas registadas e componentes da instância RiTS com ciclos de vida concluídos	77
Figura 35	Estatísticas de vida útil por componente com gráficos de média / P50 / P90 e volume de amostras	78

Figura 36 Relatório de recomendações com intervalos de substituição baseados no percentil 90 79

Figura 37 Gestão de utilizadores do sistema central com três perfis diferenciados 80

Índice de Tabelas

Tabela 1	Requisitos funcionais do sistema local	30
Tabela 2	Requisitos funcionais do sistema central	32
Tabela 3	Requisitos não funcionais do sistema em geral	33

Acrónimos

Sigla/Acrónimo	Designação por extenso
API	<i>Application Programming Interface</i>
CEN	<i>European Committee for Standardization</i>
CMTS	<i>Compact Multicell Test System</i>
CMMS	<i>Computerized Maintenance Management System</i>
CSS	<i>Cascading Style Sheets</i>
CSV	<i>Comma-Separated Values</i>
DLL	<i>Dynamic-Link Library</i>
ECMA	<i>European Computer Manufacturers Association</i>
EN	<i>European Standard (Norma Europeia)</i>
EOL	<i>End-of-Line</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
ID	Identificador
IEC	<i>International Electrotechnical Commission</i>
ISO	<i>International Organization for Standardization</i>
JSON	<i>JavaScript Object Notation</i>
MVC	<i>Model-View-Controller</i>
NI	<i>National Instruments</i>
PLC	<i>Programmable Logic Controller</i>
RBAC	<i>Role-Based Access Control</i>
RCM	<i>Reliability-Centered Maintenance</i>
REST / RESTful	<i>Representational State Transfer</i>
RF	<i>Requisito Funcional</i>
RFC	<i>Requisito Funcional do Sistema Central</i>
RiTS	<i>Rotational Simultaneous Test System</i>
RNF	<i>Requisito Não Funcional</i>
SAE	<i>SAE International</i>
SHA	<i>Secure Hash Algorithm</i>
SQL	<i>Structured Query Language</i>
TPM	<i>Total Productive Maintenance</i>
UI	<i>User Interface (Interface do Utilizador)</i>
URL	<i>Uniform Resource Locator</i>
WSGI	<i>Web Server Gateway Interface</i>

1. INTRODUÇÃO

A competitividade da indústria moderna depende, em grande parte, da capacidade de as empresas assegurarem processos produtivos contínuos, eficientes e com níveis elevados de qualidade. Em ambientes industriais onde a produção é fortemente dependente do desempenho de máquinas e equipamentos, a ocorrência de avarias inesperadas pode resultar em paragens não planeadas, desperdício de matéria-prima, incumprimento de prazos e aumento significativo de custos operacionais. Assim, a manutenção industrial assume-se como um fator estratégico, não apenas como uma atividade de suporte, mas como um elemento determinante para a fiabilidade, disponibilidade e sustentabilidade dos sistemas produtivos.

Entre os vários tipos de manutenção, a manutenção preventiva destaca-se pela sua abordagem planeada e sistemática, baseada em intervenções periódicas definidas por tempo, ciclos de utilização ou critérios técnicos. O objetivo principal é reduzir a probabilidade de falhas críticas através de inspeções, substituições e ajustes realizados

antes de ocorrerem avarias. Quando corretamente implementada, esta estratégia contribui para uma maior estabilidade produtiva, prolongamento da vida útil dos componentes, redução de custos com reparações urgentes e aumento da segurança no ambiente de trabalho. Contudo, a eficácia da manutenção preventiva depende diretamente da qualidade e consistência dos registos de utilização, intervenções e histórico de componentes, sendo essencial assegurar rastreabilidade e acessibilidade à informação.

É neste enquadramento que empresas com forte presença no setor industrial, como a Controlar, assumem um papel relevante.



Figura 1 - Áreas de negócio da Controlar

Inserida num contexto tecnológico e orientada para a inovação (Figura 1), a Controlar atua no desenvolvimento e integração de soluções de teste e controlo de qualidade, automação para a indústria, criando soluções à medida dos clientes de forma a oferecer uma produção mais controlada e assertiva, contribuindo para a melhoria contínua de processos e sistemas produtivos. São produzidas na Controlar vários tipos de máquinas, desde máquina

standalone (Linha superior da Figura 2), como máquinas em linha (Linha inferior da Figura 2)



Figura 2 - Exemplos de máquinas desenvolvidas pela Controlar

A sua intervenção em ambientes industriais exige, de forma recorrente, a monitorização e gestão eficaz de equipamentos, bem como a recolha estruturada de dados técnicos que suportem decisões relacionadas com operação, manutenção e otimização.

Além disso, com a crescente adoção de princípios associados à Indústria 4.0, como digitalização, integração de dados e tomada de decisão baseada em informação, torna-se cada vez mais relevante garantir que os processos de manutenção acompanham esta evolução. A manutenção preventiva beneficia especialmente deste contexto, uma vez que a centralização de registos, a normalização de procedimentos e a capacidade de gerar indicadores de utilização e desgaste permitem melhorar a fiabilidade dos equipamentos e apoiar decisões técnicas com maior rigor.

1.1. CONTEXTUALIZAÇÃO

Atualmente, a abordagem predominante em máquinas da Controlar consiste na manutenção reativa, ou seja, a intervenção ocorre apenas após a ocorrência de uma avaria ou falha no sistema. Esta estratégia tem revelado diversas limitações, nomeadamente ao nível da organização do trabalho e da gestão dos recursos disponíveis.

As manutenções reativas são, por natureza, de carácter urgente, uma vez que a paragem inesperada de uma máquina pode comprometer toda uma linha de produção, resultando em custos elevados para os clientes da Controlar. Esta imprevisibilidade cria um ambiente de constante pressão e dificulta a planificação das intervenções técnicas.

Adicionalmente, a escassez de recursos humanos agrava a situação. Nem sempre é possível alocar de imediato uma equipa de suporte técnico, o que pode atrasar significativamente a resolução do problema. Acresce ainda o facto de muitas vezes, os técnicos serem enviados para o local sem conhecimento prévio da causa da avaria, obrigando a uma análise diagnóstica em tempo real, antes de se poder proceder à substituição do componente defeituoso.

Perante este cenário, torna-se evidente a necessidade de repensar a estratégia de manutenção adotada, de forma a corrigir os impactos negativos das falhas e a garantir um funcionamento mais eficiente e sustentável dos sistemas produtivos.

1.2. DESCRIÇÃO DO PROBLEMA

O problema central reside na falta de previsibilidade relativamente ao momento em que uma avaria ou uma paragem de máquina poderá ocorrer. A inexistência de sinais fiáveis e tempestivos sobre o estado de degradação dos componentes conduz, com frequência, a paragens não planeadas, com impactos significativos: perdas económicas decorrentes da interrupção da produção, aumento do tempo de reposição de serviço, e, não raras vezes, agravamento do dano por ausência de manutenção atempada. Acresce

que os tempos médios de vida útil disponibilizados pelos fabricantes, embora úteis como referência, não refletem as condições reais de operação (perfil de carga, contexto ambiental, ciclos efetivamente executados, qualidade das intervenções prévias), sendo por isso frequentemente aplicados de forma desajustada ao caso concreto.

Em muitos contextos industriais, a manutenção mantém-se predominantemente reativa (após falha) ou calendarizada (por tempo de calendário), ambas com limitações claras: a primeira sacrifica disponibilidade e previsibilidade; a segunda ignora a utilização efetiva e o desgaste acumulado dos componentes. A ausência de dados operacionais estruturados, de rastreabilidade das intervenções e de métricas de utilização por componente dificulta a transição para modelos mais informados, baseados em condição e risco.

Neste enquadramento, torna-se necessário instrumentar o processo de manutenção para recolher, organizar e disponibilizar informação relevante ao nível local: catálogo de componentes e ações, associação explícita entre componentes instalados e ações, contagem de ciclos efetivamente executados (*working cycles*), bem como o estado operativo de cada entrada (ativo/desativado/removido). Paralelamente, é essencial garantir rastreabilidade completa (histórico e *logs* de alteração por utilizador), de forma a suportar auditoria, reconstrução de eventos e melhoria contínua.

A aplicação proposta responde a estas necessidades em duas frentes complementares. Localmente, disponibiliza uma interface *web* leve que permite registar e consultar componentes, ações e instalações, incrementar ciclos associados às ações, gerar alertas com base na percentagem de utilização (em função de *working cycles* face a *expected cycles*), e governar o acesso por perfis (administrador, operador, observador). Esta camada assegura funcionamento autónomo (mesmo com conectividade limitada), fornecendo exportação de dados em formato normalizado JSON (*JavaScript Object Notation*) e registando automaticamente operações sensíveis em *log*. Centralmente (segunda parte do trabalho), os dados recolhidos por múltiplas instâncias locais serão agregados numa plataforma online, permitindo filtragem, normalização e análise

transversal de desempenhos em diferentes máquinas/contextos, com vista à estimativa mais robusta de tempos de vida e ao apoio à decisão na seleção de componentes futuros.

Deste modo, o problema deixa de ser encarado como uma ausência genérica de previsibilidade e passa a ser formalizado como um problema de dados: recolher o que realmente interessa, estruturar e relacionar entidades (componentes–ações–instalações), medir utilização de forma objetiva (ciclos), sinalizar risco (alertas por patamares) e acumular histórico suficiente para, numa fase seguinte, inferir comportamentos de falha a partir de observações reais. A solução proposta cria, assim, as condições para evoluir de um modelo reativo/calendário para um modelo informado por evidência, melhorando a disponibilidade dos ativos, a previsibilidade das intervenções e a eficiência global do processo de manutenção.

Perante este diagnóstico, seria expectável considerar a adoção de um sistema de gestão de manutenção assistida por computador CMMS (*Computerized Maintenance Management System*) como resposta ao problema identificado. Estas plataformas constituem uma solução consolidada no mercado industrial, disponibilizando funcionalidades de planeamento de manutenção preventiva, registo de intervenções e controlo de ativos que, em traços gerais, se alinham com as necessidades descritas. Contudo, a análise das soluções disponíveis revelou limitações estruturais que inviabilizam a sua adoção direta no contexto da Controlar [14].

A restrição mais determinante prende-se com a conectividade. As máquinas desenvolvidas e mantidas pela Controlar operam, por requisito dos clientes ou por exigências de segurança do processo produtivo, em ambientes fisicamente isolados da internet. A grande maioria dos sistemas CMMS modernos assenta em arquiteturas *cloud* ou depende de conectividade de rede contínua para o funcionamento das suas funcionalidades nucleares, o que os torna incompatíveis com este contexto sem alterações de infraestrutura de custo e risco operacional desproporcionados. Acresce que a necessidade de integração com os modelos de software internos da Controlar e com os sistemas de automação e teste já presentes nas máquinas, nomeadamente o NI TestStand, exigiria um nível de personalização que os sistemas de âmbito geral não suportam de forma

nativa. Finalmente, a lógica de rastreabilidade por ciclos de utilização efetivos, em substituição de uma simples calendarização por tempo, representa um requisito funcional específico que não encontra correspondência direta nas funcionalidades base das soluções comerciais analisadas.

Por estas razões, o desenvolvimento de uma solução à medida apresentou-se como a abordagem mais adequada, permitindo responder de forma precisa ao conjunto de restrições e necessidades identificadas sem comprometer a autonomia operacional das máquinas nem introduzir dependências externas incompatíveis com o ambiente industrial em causa.

1.3. OBJETIVOS

O principal objetivo deste trabalho é o desenvolvimento de um sistema de apoio à manutenção preventiva adaptado ao contexto operacional da Controlar, concebido para superar as limitações estruturais da abordagem reativa atualmente em uso. Para a concretização deste objetivo geral, foram definidas metas específicas que orientaram as decisões de desenvolvimento ao longo do projeto.

Em primeiro lugar, pretende-se criar um sistema de registo e acompanhamento do desgaste de componentes instalados nas máquinas da Controlar, capaz de armazenar, organizar e disponibilizar informação sobre o estado de utilização de cada componente com base em ciclos de funcionamento reais.

Em segundo lugar, o sistema tem como objetivo suportar ativamente uma abordagem preventiva à manutenção, mediante a geração automática de alertas baseados em limiares de utilização. Ao sinalizar componentes cujo nível de desgaste se aproxima de valores críticos, o sistema permite antecipar intervenções antes da ocorrência de avarias, reduzindo a probabilidade de paragens não planeadas e os custos associados a reparações de emergência.

Em terceiro lugar, procura-se garantir rastreabilidade completa do processo de manutenção, através do registo sistemático de intervenções, histórico de alterações e controlo de acessos diferenciado por perfil de utilizador. Esta dimensão é determinante em contexto industrial, onde a capacidade de reconstruir eventos passados, auditando ações e responsabilidades, constitui um requisito de qualidade e segurança operacional.

Em quarto lugar, o sistema visa consolidar dados provenientes de múltiplas instalações numa plataforma central online, permitindo comparar o comportamento de componentes em diferentes máquinas e contextos operacionais. Esta capacidade de análise transversal é particularmente relevante para a Controlar, cuja atividade envolve a instalação e manutenção de equipamentos em múltiplos clientes, e onde a aprendizagem organizacional sobre vida útil e padrões de falha depende necessariamente da agregação de dados distribuídos.

Por fim, a disponibilização de dados reais sobre a durabilidade dos componentes em ambiente operacional constitui igualmente um objetivo do trabalho, com o propósito de complementar as especificações fornecidas pelos fabricantes e apoiar as equipas de projeto na seleção e dimensionamento de componentes futuros. Esta informação, recolhida em condições reais de funcionamento, tem potencial para melhorar a qualidade técnica das decisões de engenharia e contribuir para a redução dos custos de ciclo de vida dos sistemas desenvolvidos pela empresa.

A concretização destes objetivos visa, no fundo, criar as condições necessárias para que a Controlar possa evoluir de um modelo de manutenção predominantemente reativo para um modelo fundamentado em dados, com ganhos mensuráveis em termos de disponibilidade, previsibilidade operacional e eficiência global dos processos de manutenção.

1.4. ABORDAGEM E FASES DO PROJETO

A implementação da solução foi planeada segundo uma abordagem modular, incremental e iterativa, que permite a construção progressiva dos diferentes componentes do sistema com validação faseada e integração controlada. Esta forma de trabalhar foi adotada por se adequar a um contexto industrial onde coexistem requisitos de fiabilidade, rastreabilidade e compatibilidade com sistemas existentes, e onde a introdução de uma solução nova deve minimizar interrupções operacionais. A separação da solução em dois subsistemas complementares, um sistema local orientado à operação junto da máquina, e um sistema central orientado à consolidação e análise de dados.

O desenvolvimento do projeto foi organizado em cinco fases principais, representando a evolução desde o levantamento de requisitos até à disponibilização da solução em ambiente real. A primeira fase, dedicada à análise de requisitos e planeamento, centrou-se no levantamento de necessidades técnicas e funcionais junto das equipas de manutenção e engenharia da Controlar, na identificação dos principais cenários de utilização e na definição das restrições inerentes ao ambiente industrial. Esta fase incluiu ainda a caracterização dos equipamentos e sistemas envolvidos, a definição da estrutura dos dados relevantes e a seleção das tecnologias a utilizar em cada módulo, de forma a garantir compatibilidade entre o sistema local e a futura integração central.

A segunda fase focou-se no desenvolvimento do sistema local, implementando o núcleo operacional da solução com prioridade à robustez, autonomia de funcionamento e rastreabilidade. Os diferentes componentes deste módulo foram desenvolvidos e validados de forma sequencial, garantindo que cada elemento se encontrava estável antes de avançar para o seguinte, numa lógica de integração progressiva e controlada.

A terceira fase correspondeu ao desenvolvimento do sistema central, concebido como repositório agregador e camada de análise sobre dados provenientes de múltiplas origens. Esta fase introduziu desafios adicionais relacionados com a heterogeneidade dos dados de entrada, o controlo de qualidade na ingestão e a capacidade de suportar consultas e visualizações sobre um repositório em crescimento. A preocupação com a

consistência e comparabilidade dos dados constituiu um princípio metodológico central desta fase, dado que a utilidade analítica da plataforma central depende diretamente da qualidade e coerência da informação armazenada.

A quarta fase incidiu sobre os testes e a validação da solução, conduzidos de forma progressiva e alinhados com a integração por fases. Foram considerados testes ao nível dos módulos individuais, testes de integração entre camadas e validação funcional com utilizadores técnicos, recolhendo *feedback* sobre a adequação operacional da solução ao contexto real de utilização. A quinta e última fase contemplou o *deploy* da solução em ambiente real, com verificação do correto funcionamento dos fluxos críticos após instalação.

1.5. CALENDARIZAÇÃO DO PROJETO

O desenvolvimento do projeto decorreu entre março e julho de 2026, seguindo a sequência de fases descrita na secção 1.4. As primeiras seis semanas foram dedicadas à fundamentação do trabalho: duas semanas de estudo do problema junto das equipas da Controlar, seguidas de duas semanas de levantamento do estado da arte e mais duas semanas de definição da arquitetura e do design da solução. A implementação do sistema, incluindo os dois módulos local e central, ocupou o mês seguinte, e o mês posterior foi dedicado à obtenção de resultados e à colocação da solução em produção na máquina RiTS ControlarS2.8.

A redação do relatório não foi tratada como uma fase isolada no final do projeto, mas iniciada em paralelo com a implementação e mantida ao longo de todas as fases subsequentes, sendo consolidada e revista à medida que os resultados ficavam disponíveis. A apresentação final foi preparada na última semana que antecedeu a entrega. A Figura 3 ilustra esta planificação sob a forma de diagrama de Gantt.

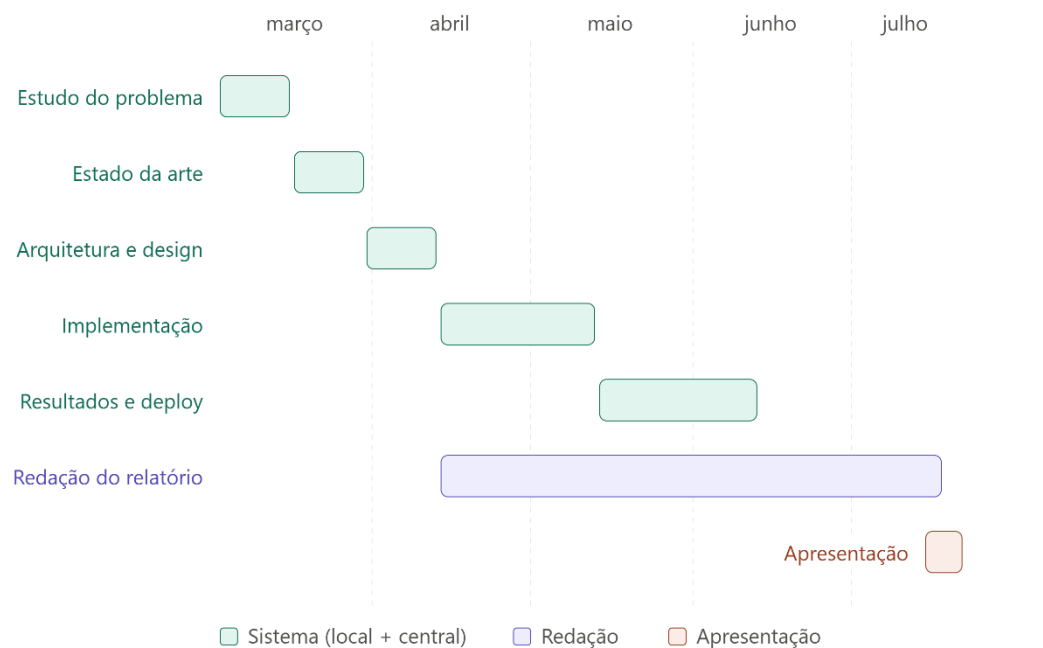


Figura 3 – Diagrama de Gantt do projeto

1.6. ORGANIZAÇÃO DO DOCUMENTO

O presente documento está organizado em seis capítulos, estruturados para acompanhar progressivamente o raciocínio que suporta o desenvolvimento da solução proposta.

O Capítulo 1 apresenta o enquadramento geral do trabalho: o contexto industrial que motiva o estudo, o problema identificado, os objetivos definidos e a abordagem metodológica adotada.

O Capítulo 2 expõe o estado da arte relativo às estratégias de manutenção industrial, com foco nos fundamentos da manutenção preventiva e na ferramenta NI TestStand, e apresenta as linguagens, *frameworks* e ferramentas que compõem o *stack* tecnológico da solução.

O Capítulo 3 documenta a fase de análise e design. Começa com uma análise de domínio que clarifica como as entidades do sistema se relacionam entre si, passa pela especificação de requisitos funcionais e não funcionais de ambos os módulos e pelos casos de uso representativos, e conclui com as decisões de design da base de dados e da arquitetura da solução.

O Capítulo 4 descreve a implementação dos dois módulos, organizada por camada funcional. A secção de *frontend* cobre a estrutura de navegação, o padrão de comunicação assíncrona e os mecanismos de visualização de dados e alertas. A secção de *backend* cobre a camada de rotas Flask, o modelo de acesso à base de dados, a gestão de utilizadores e encriptação de credenciais, a exportação e interoperabilidade entre módulos, a normalização dos dados no sistema central e a consolidação estatística.

O Capítulo 5 apresenta os resultados da aplicação da solução em ambiente real. A primeira secção documenta a instalação e operação do sistema local na máquina RiTS ControlarS2.8 da Controlar, incluindo a integração efetiva com o NI TestStand e o comportamento do sistema de alertas. A segunda secção documenta o sistema central implantado em servidor dedicado, com as estatísticas calculadas e o relatório de recomendações gerado a partir de dados reais de múltiplas instalações.

O Capítulo 6 encerra o documento com as conclusões do trabalho, avaliando em que medida os objetivos propostos foram alcançados, as limitações reconhecidas que condicionam o âmbito atual da solução, e as direções de evolução identificadas para trabalho futuro.

2. MANUTENÇÃO INDUSTRIAL E TECNOLOGIAS ENVOLVIDAS

O presente capítulo enquadra o trabalho desenvolvido à luz do estado do conhecimento nas áreas relevantes para a solução proposta. Numa primeira parte, são apresentadas as principais estratégias de manutenção industrial, desde a manutenção corretiva até às abordagens baseadas na condição e na fiabilidade, situando a manutenção preventiva no contexto mais amplo das práticas de gestão de ativos. Numa segunda parte, aprofundam-se os conceitos, práticas e condições de eficácia específicas da manutenção preventiva, que constituem a base conceptual do sistema desenvolvido. Por fim, são analisadas as linguagens e ferramentas utilizadas para desenvolver este trabalho de forma a conhecer melhor o seu funcionamento e as necessidades da sua criação.

2.1. ESTRATÉGIAS DE MANUTENÇÃO INDUSTRIAL

A manutenção industrial pode ser definida como o conjunto de ações técnicas e de gestão realizadas ao longo do ciclo de vida de um ativo com o objetivo de o manter, ou repor, num estado em que seja capaz de cumprir a função requerida. Esta definição é coerente com a terminologia adotada em normas europeias e internacionais, que procuram uniformizar conceitos e reduzir ambiguidades na documentação técnica e na comparação de práticas entre organizações [1]. Numa perspetiva de gestão, a manutenção não deve ser vista apenas como resposta a falhas, mas como um elemento estruturante do desempenho operacional e da gestão do risco, influenciando diretamente a disponibilidade, a fiabilidade e os custos de operação e de paragem [2].

Em termos conceptuais, a classificação dos tipos de manutenção pode ser organizada de acordo com o momento em que a intervenção é desencadeada e com a informação que suporta essa decisão. Um primeiro grupo corresponde à manutenção corretiva, realizada após a ocorrência de uma falha, visando restaurar a capacidade funcional do equipamento. Apesar de inevitável em qualquer contexto industrial, uma estratégia excessivamente reativa tende a conduzir a paragens não planeadas, maior variabilidade operacional e custos elevados, sobretudo quando a falha implica danos secundários ou necessidade de resposta urgente [2]. A própria normalização do domínio distingue a manutenção corretiva como intervenção posterior ao reconhecimento de uma avaria, sublinhando que o objetivo principal é a reposição do estado requerido e não a otimização do momento de intervenção [1].

Em contraste, a manutenção preventiva corresponde a intervenções planeadas antes da falha, executadas de acordo com intervalos definidos por calendário ou por utilização, como horas de funcionamento, ciclos ou número de operações. A lógica da preventiva é reduzir a probabilidade de falha através de inspeções, substituições programadas, lubrificação e outras tarefas periódicas, assumindo que certos mecanismos de desgaste e degradação podem ser controlados com periodicidade adequada [2]. Este tipo de política é particularmente eficaz quando existe relação relativamente estável entre o nível de utilização e a degradação do componente, permitindo definir intervalos com

base em recomendações do fabricante, histórico de avarias e experiência de operação. Em setores onde a recolha e normalização de dados é crítica, a manutenção preventiva também é importante por facilitar o registo consistente de tarefas e a análise estatística posterior de fiabilidade e custos [3].

Uma evolução da preventiva tradicional surge com a manutenção baseada na condição, onde a decisão de intervir é suportada pelo estado observado do equipamento e por indicadores mensuráveis do seu comportamento. Em vez de assumir intervalos fixos, pretende-se detetar alterações que sinalizem degradação ou aproximação de falha, recorrendo a técnicas de monitorização como análise de vibrações, temperatura, assinaturas elétricas, análise de óleo ou ultrassons. Normas específicas para *condition monitoring* estabelecem orientações gerais para a implementação destes programas, incluindo princípios de seleção de parâmetros, aquisição de dados e interpretação de tendências, reforçando a necessidade de critérios consistentes e comparáveis entre ativos [4]. Neste contexto, é comum distinguir a manutenção baseada na condição de uma vertente mais avançada, frequentemente designada por manutenção preditiva, em que, além do diagnóstico do estado atual, se procura estimar a evolução futura e antecipar o momento ótimo de intervenção, recorrendo a modelos e prognósticos. A literatura de referência sobre diagnósticos e prognósticos em manutenção descreve este processo como dependente da qualidade dos dados recolhidos, de métodos de processamento robustos e de mecanismos de decisão capazes de transformar tendências e indicadores em ações de manutenção concretas, incluindo estimativas de vida útil remanescente [5].

Para além destas categorias mais clássicas, existem metodologias orientadas para selecionar políticas de manutenção em função das consequências da falha e do risco associado. A manutenção centrada na fiabilidade, conhecida como *Reliability-Centered Maintenance*, propõe um processo estruturado para identificar funções do ativo, falhas funcionais, modos de falha e respetivas consequências, e, a partir daí, definir tarefas de manutenção técnica e economicamente justificadas. O valor desta abordagem está em evitar a aplicação indiscriminada de tarefas preventivas onde não são eficazes e, simultaneamente, assegurar que ativos críticos recebem políticas proporcionais ao impacto potencial na segurança, no ambiente, na operação e nos custos [6]. Complementarmente,

em ambientes industriais onde se pretende maximizar eficiência global e reduzir perdas, a Manutenção Produtiva Total (*Total Productive Maintenance*) introduz uma dimensão organizacional e cultural, promovendo a participação ativa dos operadores em atividades de manutenção de primeira linha, como inspeções de rotina e identificação precoce de anomalias. Esta filosofia procura aumentar disponibilidade e estabilidade do processo através de envolvimento, normalização de práticas e melhoria contínua, reduzindo a separação tradicional entre operação e manutenção [7].

Em síntese, os tipos de manutenção não devem ser entendidos como alternativas mutuamente exclusivas, mas como componentes de uma estratégia integrada. Em sistemas industriais reais, é comum coexistirem intervenções corretivas para falhas inevitáveis, ações preventivas para mecanismos de desgaste conhecidos, monitorização baseada na condição para ativos críticos e metodologias orientadas ao risco para priorização e otimização. A qualidade da recolha de dados, a padronização de registos e a capacidade de analisar historicamente falhas e intervenções são fatores decisivos para evoluir de abordagens predominantemente reativas para modelos mais previsíveis e sustentáveis, com ganhos em disponibilidade, custo e segurança [3][5].

É com este propósito que se insere o trabalho desenvolvido: a proposta de um sistema que instrumenta o processo de manutenção preventiva com recolha estruturada de dados operacionais, rastreabilidade de intervenções e mecanismos de alerta baseados em utilização real, criando a base de informação necessária para evoluir de uma lógica reativa para uma abordagem orientada por evidência.

2.2. MANUTENÇÃO PREVENTIVA: CONCEITO E PRÁTICAS

A manutenção preventiva corresponde ao conjunto de intervenções planeadas realizadas antes da ocorrência de uma falha, com o objetivo de reduzir a probabilidade de avaria e manter o equipamento num estado compatível com a função requerida. Na terminologia normalizada da manutenção, esta abordagem distingue-se da manutenção corretiva precisamente por não ser desencadeada por uma falha já verificada, mas por

critérios pré-definidos de tempo, utilização ou estado, integrando-se numa lógica de planeamento e controlo sistemático do desempenho do ativo [1]. Em termos de gestão da fiabilidade, a manutenção preventiva assume um papel central por contribuir para a disponibilidade e para a previsibilidade operacional, permitindo reduzir paragens não planeadas e organizar recursos, peças e janelas de intervenção com menor impacto na produção [1].

A adoção de manutenção preventiva tende a surgir como resposta a três limitações típicas de estratégias predominantemente reativas. Em primeiro lugar, a falha inesperada tem custos indiretos elevados, associados a paragem, perdas de produção e efeitos na qualidade. Em segundo lugar, a intervenção urgente frequentemente implica menor eficiência na execução, maior probabilidade de erros e maior exposição a risco. Em terceiro lugar, a ausência de planeamento dificulta a gestão de sobressalentes e o dimensionamento de equipas, conduzindo a tempos de reparação mais longos e a um comportamento operacional menos estável [2]. Assim, a preventiva é, simultaneamente, uma prática técnica e uma prática de gestão, na medida em que depende de disciplina de registo, planeamento, padronização de procedimentos e acompanhamento por indicadores [9].

Do ponto de vista conceptual, a manutenção preventiva pode ser estruturada em duas modalidades principais. A primeira é a manutenção preventiva baseada no tempo, em que as tarefas são executadas segundo periodicidade calendarizada, como inspeções mensais ou revisões anuais. A segunda é a manutenção preventiva baseada na utilização, em que a execução ocorre após determinado nível de esforço operacional, medido em horas de funcionamento, número de ciclos, número de operações ou outra métrica diretamente ligada ao uso. A escolha entre estas modalidades depende da natureza do mecanismo de degradação e da forma como o desgaste evolui com a operação, sendo particularmente adequada à abordagem por utilização quando o processo produtivo apresenta variabilidade significativa no regime de trabalho do equipamento [2][8]. Em ambos os casos, o princípio é semelhante: estabelecer um limiar de intervenção suficientemente antecipado para reduzir a ocorrência de falhas, mas não tão conservador que origine substituições prematuras e desperdício de vida útil [9].

A definição destes intervalos exige, idealmente, uma base técnica sustentada. Em muitos contextos industriais, é comum iniciar o programa com recomendações do fabricante e experiência de campo, evoluindo progressivamente para uma otimização baseada em dados históricos. Esta evolução é relevante porque nem todos os modos de falha são sensíveis a intervenções periódicas. Estudos clássicos sobre padrões de falha evidenciam que uma parcela significativa das falhas não segue um comportamento estritamente “por desgaste” com aumento contínuo da taxa de falha ao longo do tempo, o que significa que a simples calendarização de substituições não garante, por si só, redução proporcional de avarias [10]. Assim, a manutenção preventiva é mais eficaz quando associada a uma análise de criticidade e a uma identificação explícita dos modos de falha relevantes, assegurando que cada tarefa preventiva tem um propósito claro e uma relação técnica defensável com o risco que pretende mitigar [11].

Na prática, um programa de manutenção preventiva é composto por tarefas de natureza diversa, cuja seleção depende do tipo de ativo, do processo e das consequências de falha. Entre as tarefas mais comuns encontram-se inspeções sistemáticas, lubrificação, reapertos, limpeza técnica, substituição programada de consumíveis, calibração e testes funcionais. A inspeção tem como objetivo identificar sinais precoces de degradação e condições anómalas, funcionando como mecanismo de controlo que pode desencadear intervenções adicionais. A lubrificação e as ações de limpeza técnica reduzem fricção, desgaste e contaminação, sendo frequentemente determinantes em equipamentos rotativos e em sistemas com componentes mecânicos de precisão. A substituição programada aplica-se sobretudo a elementos com comportamento de desgaste relativamente previsível ou cuja falha tem impacto elevado, justificando a troca antes de atingir o limite funcional. A calibração e os testes funcionais são críticos em sensores, atuadores e sistemas de segurança, onde pequenas derivações podem ter impacto relevante na qualidade ou no risco operacional [8][9].

Contudo, a qualidade de um programa preventivo não depende apenas do conjunto de tarefas, mas da forma como estas são especificadas e executadas. A descrição de cada tarefa deve ser suficientemente clara para garantir repetibilidade entre técnicos e turnos, incluindo critérios de aceitação e ações a tomar quando são detetadas anomalias. Um dos

riscos recorrentes em programas preventivos é a “ritualização” das tarefas, em que a execução passa a ser feita por rotina, sem ligação efetiva à redução de falhas. Para mitigar este risco, recomenda-se que o programa seja periodicamente revisto com base em resultados observados, registos de avarias e indicadores de desempenho, ajustando periodicidades, redefinindo tarefas e removendo atividades que não demonstrem valor [9][11].

A gestão do trabalho é outro fator determinante. Mesmo tarefas tecnicamente adequadas podem produzir baixo resultado se forem executadas de forma irregular ou sem preparação. A literatura de boas práticas enfatiza a necessidade de planeamento e programação como parte integrante da manutenção preventiva, assegurando disponibilidade de peças, ferramentas, documentação e tempos de paragem compatíveis com a produção. Esta disciplina melhora a qualidade da execução e reduz o tempo total de intervenção, com impacto direto na disponibilidade do equipamento [9]. Do mesmo modo, a gestão de sobressalentes deve ser coerente com o programa preventivo, evitando tanto ruturas de stock em componentes críticos como excesso de inventário em peças de baixa rotação, mantendo rastreabilidade e controlo de consumo [3][9].

Um aspeto particularmente relevante na manutenção preventiva moderna é a integração com metodologias de seleção e otimização de políticas, como a manutenção centrada na fiabilidade. A abordagem RCM (*Reliability-Centered Maintenance*) fornece um enquadramento para decidir quando faz sentido aplicar tarefas preventivas baseadas em periodicidade e quando é preferível recorrer à monitorização por condição, a testes funcionais ou até a estratégias de “*run-to-failure*” em itens não críticos. Este processo reduz a tendência para programas preventivos excessivos e melhora a coerência técnica das tarefas, ligando-as a modos de falha e consequências [11][12]. Em paralelo, a gestão de ativos, enquanto disciplina de alinhamento entre desempenho, custo e risco ao longo do ciclo de vida, reforça a importância de definir objetivos, medir resultados e assegurar melhoria contínua, sendo compatível com programas de preventiva bem estruturados e sustentados em indicadores [13].

Em síntese, a manutenção preventiva representa uma prática essencial para reduzir paragens não planeadas e aumentar previsibilidade, mas o seu sucesso depende de três condições fundamentais: seleção criteriosa de tarefas com base em mecanismos de falha e criticidade; definição rigorosa de procedimentos e critérios de execução; e gestão sistemática do trabalho e dos dados, com revisão periódica orientada a resultados. Quando estas condições são respeitadas, a preventiva deixa de ser apenas um conjunto de rotinas calendarizadas e passa a constituir um mecanismo estruturado de controlo do risco e de melhoria do desempenho operacional [2][9][11].

No contexto específico deste trabalho, estas condições de eficácia da manutenção preventiva traduzem-se em requisitos concretos do sistema a desenvolver: a seleção criteriosa de tarefas corresponde à necessidade de registar e associar componentes a ações de manutenção com critérios técnicos definidos; a definição rigorosa de procedimentos traduz-se nos mecanismos de controlo de estado e rastreabilidade implementados; e a gestão sistemática de dados com revisão periódica orientada a resultados é precisamente o que a plataforma central visa suportar, ao agregar informação de múltiplas instalações para extração de padrões e apoio à decisão.

2.3. LINGUAGENS E FERRAMENTAS UTILIZADAS

Em sistemas de informação para contextos industriais com restrições de conectividade, a escolha do *stack* tecnológico tende a ser conservadora por boas razões: tecnologias maduras e amplamente documentadas reduzem o risco operacional, simplificam a manutenção ao longo do tempo e permitem que o foco de desenvolvimento recaia sobre os requisitos do domínio e não sobre a gestão de dependências externas. Esta secção apresenta as ferramentas e linguagens que suportam a solução desenvolvida, contextualizando cada uma no panorama técnico em que se insere.

Python foi concebido por *Guido van Rossum* no final dos anos 80 e publicado em 1991 [15], com uma filosofia de design centrada na legibilidade e na forma explícita que é apresentada. A ideia central, que o código é lido muito mais vezes do que escrito, moldou

a sintaxe e as convenções da linguagem, tornando-a particularmente adequada a projetos mantidos por equipes reduzidas durante períodos prolongados. No domínio dos serviços web e da integração de sistemas industriais, Python ganhou terreno significativo nas últimas duas décadas [16], em parte pela riqueza do ecossistema de bibliotecas disponíveis via PyPI e em parte pela facilidade com que se integra com ferramentas do ecossistema industrial. Como se descreve adiante. A tipagem dinâmica e a ausência de um ciclo de compilação explícito encurtam o ciclo de iteração, o que tem valor prático em projetos onde os requisitos se vão precisando à medida que o sistema toca o ambiente real.

O NI TestStand é uma plataforma de sequenciamento de testes desenvolvida pela National Instruments, amplamente utilizada em sistemas de teste e validação industriais para orquestrar a execução de sequências de operações sobre o equipamento sob teste [17]. Nas máquinas desenvolvidas pela Controlar, o TestStand é responsável por executar as sequências de teste que constituem o processo produtivo: cada passo da sequência corresponde a uma operação específica da máquina. acionamento de atuadores, leitura de sensores, verificação de parâmetros e é executado um número controlado de vezes ao longo da vida do equipamento. É precisamente esta correspondência entre passos de sequência e operações repetíveis que torna o TestStand o ponto natural de contagem de ciclos de utilização dos componentes. O TestStand suporta um modelo de extensão baseado em adaptadores que permitem invocar código externo, incluindo bibliotecas .NET, diretamente como passos de sequência, sem alterar a estrutura da sequência existente. É este mecanismo que a biblioteca *ControlarMaintenanceClient* utiliza para integrar o incremento de contadores de ciclos no fluxo normal de execução do teste, dispensando qualquer intervenção manual na interface de manutenção durante a operação da máquina.

Flask é um *microframework web* para Python, desenvolvido por *Armin Ronacher* e publicado em 2010 [18]. A palavra "micro" não traduz limitação funcional, mas sim uma filosofia de design deliberadamente distinta das *frameworks* de maior alcance como o Django. Flask disponibiliza as primitivas essenciais com o encaminhamento de pedidos HTTP (*HyperText Transfer Protocol*), gestão do contexto de execução e suporte a extensões opcionais. E deixa ao programador a responsabilidade de decidir sobre o resto. Esta abordagem é bem documentada na literatura de desenvolvimento *web* [18] como

adequada ao desenvolvimento de APIs (*Application Programming Interface*) RESTful (*Representational State Transfer*), em que o servidor expõe operações de domínio sobre recursos bem definidos sem a sobrecarga de funcionalidades que o projeto não requer. O *framework* assenta no standard WSGI (*Web Server Gateway Interface*) [19], garantindo compatibilidade com servidores de produção diversificados. A consequência direta desta leveza é que as decisões de arquitetura ficam do lado do programador. A separação entre camadas, a consistência no tratamento de erros e a organização dos módulos não são impostas pelo *framework*, o que exige disciplina de desenho, mas devolve controlo sobre a estrutura do sistema.

A persistência de dados, em ambos os subsistemas, é implementada em MySQL, um sistema de gestão de bases de dados relacionais de código aberto com origem em 1995 [20], atualmente mantido pela Oracle. O modelo relacional subjacente [21] organiza os dados em tabelas com relações explícitas e garante consistência através de restrições de integridade referencial. O motor de armazenamento InnoDB, adotado por omissão nas versões mais recentes, implementa as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) com suporte a transações e bloqueios ao nível da linha [22]. Uma característica do MySQL com relevância particular para este sistema é o suporte a *stored procedures*: blocos de lógica executados diretamente no motor de persistência que permitem centralizar regras de validação e manipulação dos dados no próprio servidor de base de dados, reduzindo a superfície de exposição e garantindo consistência independentemente da origem dos pedidos [20]. MySQL mantém-se entre os SGBDR (Sistema de Gestão de Banco de Dados Relacional) mais utilizados a nível global [23], com um ecossistema estável de ferramentas de administração e migração.

A interface de utilizador, de ambos os subsistemas, foi implementada com as três tecnologias base da web: HTML5, CSS3 e JavaScript. O HTML5, especificado pelo W3C [24], introduziu um conjunto de melhorias semânticas e APIs nativas que reduziram a dependência de extensões externas para funcionalidades correntes. O CSS3 organiza-se em módulos independentes, entre os quais *Flexbox* e *Grid* [25], que simplificam a construção

de *layouts* responsivos sem recorrer a bibliotecas adicionais. O JavaScript é definido pela especificação *ECMAScript*, mantida pelo *ECMA International* (*European Computer Manufacturers Association*) [26]; a versão *ES6* (2015) introduziu funções assíncronas, módulos nativos e outras funcionalidades que modernizaram o desenvolvimento *front-end* sem quebrar a compatibilidade com versões anteriores. A opção por não utilizar *frameworks* de interface como React ou Vue, abordagem referida na literatura como "*vanilla JS*" [27], elimina riscos associados a atualizações de dependências de terceiros e garante compatibilidade com diferentes versões de navegadores. Para sistemas instalados em ambientes industriais com controlo estrito sobre o *software* disponível, esta característica tem peso real.

O JSON (*JavaScript Object Notation*) foi formalizado por *Douglas Crockford* e publicado nas *RFC 4627* (*Request for Comments*) [28] e, subsequentemente, *RFC 7159* [29], sendo também adotado como standard *ECMA-404* [30]. Trata-se de um formato de intercâmbio de dados baseado em texto, com uma estrutura de pares chave-valor e listas ordenadas que é simultaneamente legível por humanos e processável por máquinas. A adoção do JSON como formato dominante em APIs REST (*Representational State Transfer*) [31] resulta da sua leveza face ao XML (*Extensible Markup Language*), da compatibilidade nativa com JavaScript e do suporte generalizado nas linguagens de programação mais usadas. Em contextos de integração entre sistemas com origens distintas, o JSON funciona bem como formato de exportação e importação precisamente porque não depende de plataformas específicas para ser produzido ou consumido.

A manutenção de um *stack* tecnológico coerente entre os dois subsistemas, Python, *Flask*, MySQL, HTML (*HyperText Markup Language*) /CSS (*Cascading Style Sheets*)/JavaScript e JSON, não é uma coincidência, mas uma decisão de engenharia com consequências práticas. A literatura de engenharia de *software* refere de forma consistente que a diversidade tecnológica desnecessária aumenta a carga cognitiva de manutenção e multiplica os pontos de risco em equipas de dimensão reduzida [32].

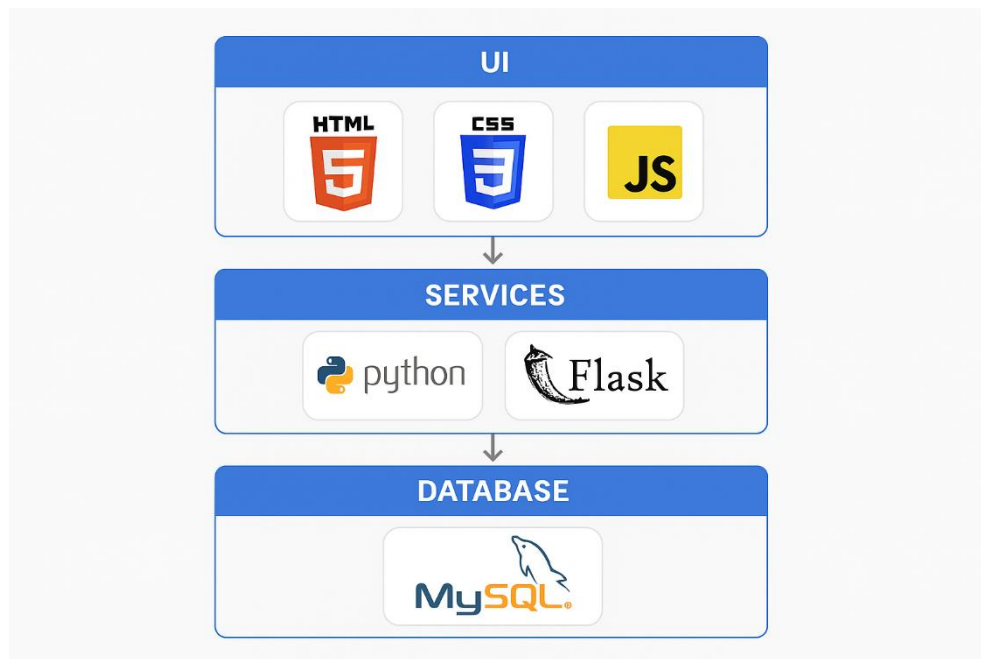


Figura 4 – *Technology stack* do sistema local e central

Partilhar o mesmo *stack* (Figura 4) entre módulos significa que o conhecimento adquirido num contexto é diretamente aplicável noutro, os processos de instalação e configuração são familiares e os problemas de integração são mais fáceis de isolar.

3. ARQUITETURA E DESIGN DA SOLUÇÃO

O presente capítulo documenta as decisões de análise e design que suportam a solução proposta. Parte do levantamento de necessidades e da especificação de requisitos, passa pelos casos de uso que ilustram as interações esperadas entre utilizadores e sistema, detalha o design das bases de dados de cada subsistema e conclui com as decisões de arquitetura que determinam a organização interna e o comportamento global da solução.

3.1. ANÁLISE DE DOMÍNIO

A forma como as entidades do sistema são definidas e estas se relacionam entre si, determina o que o sistema consegue registar, comparar e analisar, assim como o que fica fora do seu alcance. Antes de avançar para a especificação dos requisitos funcionais é necessário clarificar como o domínio do sistema está conceptualmente estruturado.

No contexto desta solução, uma máquina é qualquer equipamento de teste, validação ou controlo de qualidade montado nas instalações de um cliente. Cada máquina é identificada por um código único que combina três parâmetros de contexto, cliente, produto e tipo de máquina. Este código é o elo entre o sistema local a desenvolver, onde os dados são produzidos, e o sistema central também a implementar, onde os dados de múltiplas máquinas são futuramente agregados para análise comparativa. Na Figura 5 temos dois exemplos de máquina a RiTS (*Rotational Simultaneous Test System*) e a CMTS (*Compact Multicell Test System*)

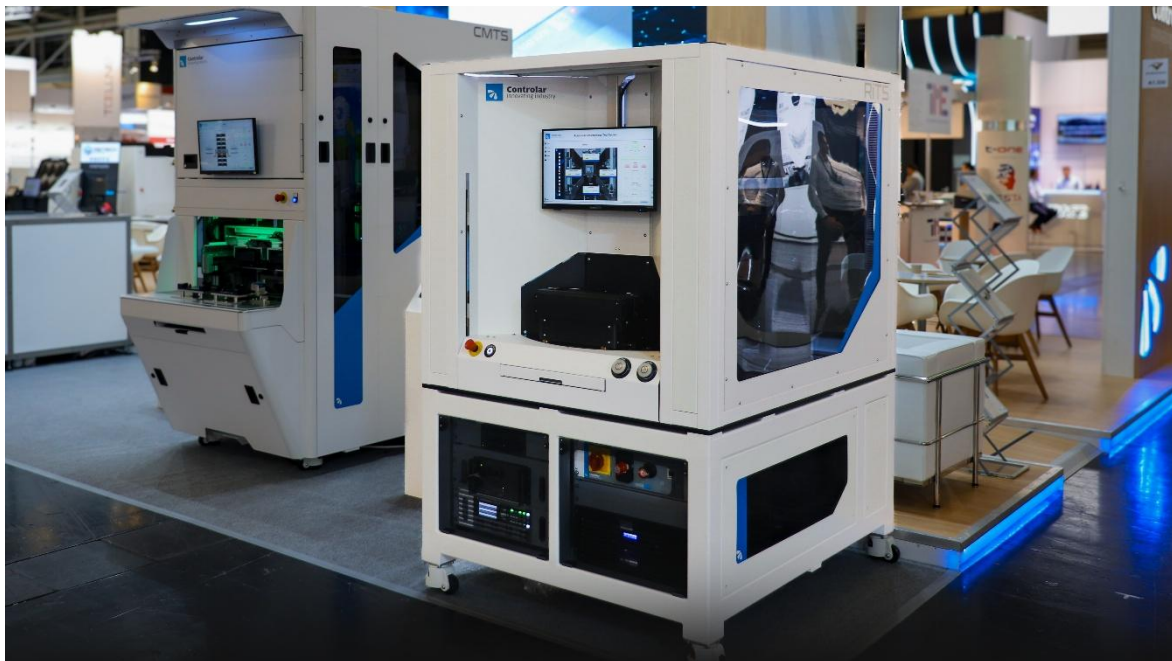


Figura 5 – CMTS (esquerda) e RiTS (direita)

Um componente é qualquer elemento físico da máquina sujeito a desgaste e que pode ser substituído de forma independente. Os componentes são identificados pela combinação de fabricante, modelo e tipo, uma combinação ternária que funciona simultaneamente como chave de identificação na biblioteca e como chave de agrupamento para efeitos estatísticos no sistema central. A mesma identificação tripla em máquinas distintas ou em posições distintas da mesma máquina representa o mesmo tipo de componente, ainda que sejam instâncias físicas diferentes. Uma ação representa uma posição funcional da máquina alvo de manutenção, tipicamente correspondente a um passo de sequência no NI TestStand. As ações têm nomes únicos no contexto de cada instalação e são o ponto de contagem de ciclos: cada vez que a ação é executada, o seu contador é incrementado. A ação não representa o componente que lá está instalado, representa o lugar, independentemente do componente que o ocupa em cada momento.

A separação entre biblioteca e instalação de componentes é a decisão conceptual central do modelo de dados. A biblioteca armazena o que existe e pode ser utilizado, é o catálogo estável de componentes e ações conhecidas pelo sistema. A instalação regista o que está atualmente em serviço e o que aconteceu historicamente ao longo do tempo, é o registo operacional de quais componentes estão em quais posições da máquina, com quantos ciclos acumulados e em que estado de utilização. Esta separação tem uma consequência prática imediata: o mesmo tipo de componente pode ser instalado em múltiplas ações da mesma máquina ou em ações de máquinas completamente distintas, sem duplicar a sua descrição lógica na biblioteca. Cada instalação acumula os seus próprios ciclos de forma independente, mas todas partilham a mesma referência de vida útil esperada definida na biblioteca do sistema.

O ciclo de vida de um componente instalado começa no momento da instalação, com o contador a zero e termina quando o componente é substituído ou retirado. Ao longo desse percurso, o contador cresce a cada execução da ação correspondente, até que a razão entre os ciclos acumulados e os ciclos esperados ultrapasse os limiares que geram alertas com níveis progressivos. Quando o componente é substituído, o registo original é preservado e guardado com o estado de ciclo “concluído”, onde um novo registo começa do zero na mesma posição. O sistema nunca elimina registos de instalações anteriores: a

linha temporal completa de cada posição fica permanentemente disponível, tornando os dados historicamente úteis para análise de fiabilidade.

Esta estrutura viabiliza a análise comparativa ao longo do tempo entre máquinas. Quando os dados de múltiplas instalações são agregados no sistema central, todos os ciclos de vida concluídos por componentes com a mesma tripla de identificação, independentemente da máquina ou cliente onde ocorreram, contribuem para a estimativa da durabilidade real desse tipo de componente. Um componente substituído numa única máquina produz uma única observação, insuficiente para qualquer inferência estatística. O mesmo componente substituído em várias máquinas ao longo do tempo produz um historial representativo. Na Figura 6 podemos ver como os dados fluem no sistema.

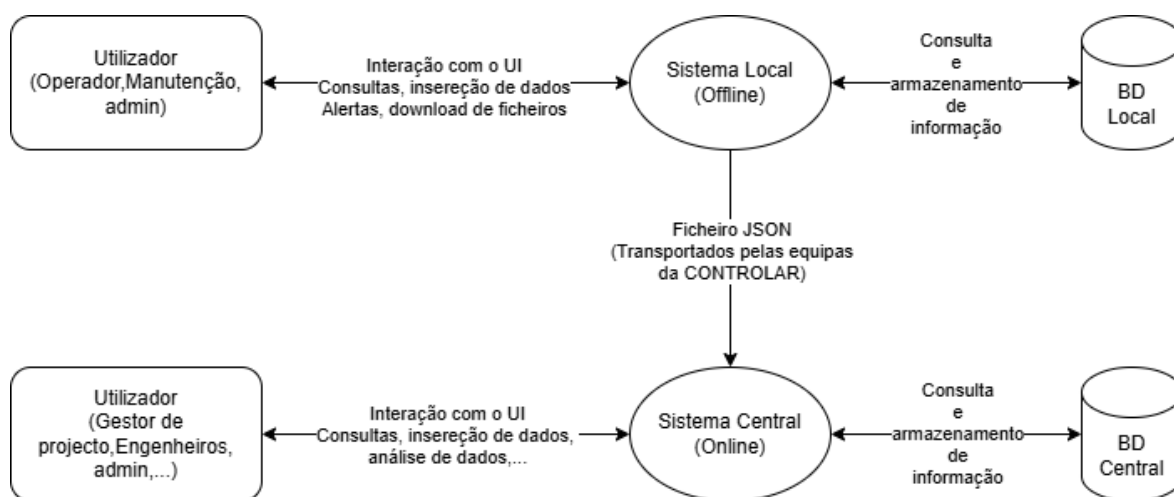


Figura 6 – Diagrama de fluxo de dados do sistema geral

A separação entre o sistema local, orientado ao registo e ao alerta e, o sistema central, orientado à análise e ao conhecimento, reflete exatamente esta lógica: os dados são produzidos localmente, mas o valor analítico emerge da sua agregação.

3.2. ANÁLISE DE REQUISITOS

O levantamento de necessidades foi conduzido junto de três grupos de intervenientes internos da Controlar. As equipas de manutenção e suporte ao produto

contribuíram para a definição dos requisitos operacionais do sistema local, identificando três limitações centrais do processo atual: ausência de registos estruturados sobre o estado dos componentes, dificuldade em antecipar intervenções sem dados de utilização fiáveis e dependência de conhecimento não documentado concentrado nos técnicos mais experientes. As equipas de engenharia e desenvolvimento acrescentaram a perspetiva estratégica: a necessidade de dados reais sobre a durabilidade dos componentes em ambiente de produção para apoiar decisões técnicas de longo prazo e comparar comportamentos entre instalações. Paralelamente, uma análise dos componentes com maior histórico de intervenções nos equipamentos em operação permitiu identificar as categorias com maior criticidade e maior potencial de monitorização baseada em ciclos de utilização.

A convergência destes contributos estabeleceu uma base de requisitos que cobre as dimensões de registo, rastreabilidade, alerta e análise estruturadoras dos dois módulos da solução.

3.2.1. REQUISITOS FUNCIONAIS DO SISTEMA LOCAL

O sistema local serve três perfis de acesso com capacidades progressivamente mais abrangentes: Observador, orientado para consulta; Operador de Manutenção, com capacidade de registo e intervenção; e Administrador, com acesso total incluindo configuração e gestão de utilizadores. Os requisitos foram estruturados em torno das secções funcionais da interface, seguindo o fluxo real de utilização.

No que respeita ao acesso e controlo de permissões, o sistema deve disponibilizar autenticação com mudança de utilizador e adaptação da interface ao perfil carregado, apresentando ou ocultando formulários, botões e secções administrativas. Esta adaptação é uma camada de usabilidade: a autorização efetiva é sempre garantida do lado do servidor, independentemente da origem do pedido.

As bibliotecas de componentes e de ações constituem os dados de referência do sistema. A Biblioteca de Componentes armazena fabricante, modelo, tipo e ciclos esperados de cada componente; a Biblioteca de Ações regista o nome e os ciclos

consumidos por cada ação. A inserção de novos registos está restrita a perfis com permissões de edição e sujeita a validação de campos obrigatórios, com restrição de unicidade sobre o nome da ação que impede duplicados.

A secção de Componentes Instalados representa o núcleo operacional: a aplicação suporta a instalação de componentes por associação a uma ação, a listagem com indicadores de utilização (ciclos acumulados, ciclos esperados, percentagem e estado), a ativação e desativação de componentes sem perda de histórico, e a substituição com preservação do registo anterior e contador a zero no novo.

O sistema de alertas calcula automaticamente a percentagem de utilização e classifica componentes em três níveis de severidade, com sinalização visual diferenciada e comportamento de descarte diferenciado por nível. O histórico de ações e o log de alterações garantem rastreabilidade completa, com o log restrito ao administrador. A exportação em formato JSON, disponível para administrador e operador, produz um ficheiro com a nomenclatura definida pelos parâmetros de configuração, deve ser forçada um pedido de atualização do estado da máquina sempre que se faz *logout* de utilizadores com permissões superiores. As configurações e a gestão de utilizadores são exclusivas do administrador. Na tabela 1 é apresentado um resumo dos requisitos para o sistema local com os perfis responsáveis e localização na página web.

Tabela 1 – Requisitos funcionais sistema local

ID	Requisito funcional (síntese)	Perfis	Secção UI
RF01	Autenticar e mudar de utilizador; definir perfil ativo	Todos	Cabeçalho / <i>Login</i>
RF02	Ajustar interface conforme permissões <i>Role-Based Access Control</i> (RBAC)	Todos	Global
RF03	Inserir e consultar componentes (biblioteca)	Inserir: <i>Admin/Operator</i> ; Consultar: Todos	Biblioteca de Componentes
RF04	Inserir e consultar ações (biblioteca)	Inserir: <i>Admin/Operator</i> ; Consultar: Todos	Biblioteca de Ações
RF05	Instalar componente (associar componente–ação)	<i>Admin/Operator</i>	Componentes Instalados

RF06	Listar componentes instalados e respetivos indicadores	Todos	Componentes Instalados
RF07	Ativar/desativar componente instalado	<i>Admin/Operator</i>	Componentes Instalados
RF08	Gerar e visualizar alertas por nível de severidade	Todos	Alertas
RF09	Exportar tabelas em JSON com nomenclatura definida	<i>Admin/Operator</i>	Exportação (botões por secção)
RF10	Pedido de atualização do estado da máquina no <i>logout</i>	<i>Admin/Operator</i>	<i>Logout</i>
RF11	Definir cliente/produto/tipo de máquina e aplicar na UI/export	<i>Admin</i>	Configurações
RF12	Consultar histórico de ações	Todos	Histórico de Ações
RF13	Consultar logs/auditoria de alterações	<i>Admin</i>	<i>Logs</i>
RF14	Criar/apagar utilizadores e alterar palavras-passe	<i>Admin</i>	Gestão de Utilizadores

3.2.2. REQUISITOS FUNCIONAIS DO SISTEMA CENTRAL

O sistema central tem uma função distinta do local: agregar, normalizar e disponibilizar para análise dados de manutenção provenientes de múltiplas instalações. Esta diferença de propósito implica requisitos centrados na qualidade e consistência dos dados de múltiplas origens, na segurança de uma plataforma multiutilizador e na capacidade de suportar análises sobre volumes em crescimento progressivo.

Em termos de ingestão, o sistema deve aceitar o *upload* de ficheiros JSON exportados pelo sistema local, validar a sua estrutura e conteúdo, registar os metadados de importação e detetar reimportações do mesmo conjunto de dados. O processamento inclui a normalização dos dados para o modelo central e a integração no repositório consolidado. Após armazenamento, a plataforma disponibiliza consulta filtrada por parâmetros relevantes, visualização de métricas e indicadores estatísticos, e exportação de resultados em formato CSV (*Comma-Separated Values*). A administração inclui gestão de utilizadores, auditoria de operações críticas e configuração das regras de importação. O resumo dos requisitos está resumido na Tabela 2.

Tabela 2 – Requisitos funcionais sistema central

ID	Requisito funcional (síntese)	Perfis típicos	Área
RFC01	Autenticar utilizadores e aplicar permissões por perfil	Todos	Segurança
RFC02	Upload/importação de ficheiros JSON (1 ou vários)	Eng./Gestor/Admin	Ingestão
RFC03	Validar estrutura e conteúdo do JSON (<i>schema</i> + regras)	Eng./Gestor/Admin	Ingestão
RFC04	Registar metadados da importação (origem, data, versão, utilizador)	Eng./Gestor/Admin	Ingestão
RFC05	Normalizar/transformar dados para o modelo central	Sistema	Processamento
RFC06	Detetar duplicados/reimportações e evitar inconsistências	Sistema/Admin	Processamento
RFC07	Armazenar dados no repositório central e disponibilizar consulta	Todos (dependendo do perfil)	Dados
RFC08	Pesquisar e filtrar por cliente/produto/máquina/data/componente/ação	Todos (consulta)	Consulta
RFC09	Consultar histórico consolidado multi-máquina	Todos (consulta)	Consulta
RFC10	Gerar indicadores agregados e comparações entre máquinas	Eng./Gestor/Admin	Análise
RFC11	Visualizar resultados (tabelas/gráficos) e navegar por detalhe	Eng./Gestor/Admin	Análise
RFC12	Exportar resultados/dados agregados (JSON/CSV)	Eng./Gestor/Admin	Interoperabilidade
RFC13	Expor <i>endpoints</i> REST para consulta/importação controlada	Sistema/Admin	Interoperabilidade
RFC14	Auditoria: registar ações críticas (<i>imports</i> , <i>admins</i> , configurações)	Admin	Administração
RFC15	Gestão de utilizadores no sistema central	Admin	Administração
RFC16	Gestão de configurações e regras de importação/versões aceites	Admin	Administração

3.2.3. REQUISITOS NÃO FUNCIONAIS

Para além das funcionalidades, a qualidade da solução depende de propriedades transversais que condicionam a sua utilidade em contexto industrial. Os requisitos não funcionais cobrem ambos os módulos, evidenciando onde os contextos divergem.

O desempenho no sistema local está associado à fluidez das operações mais frequentes, como consulta e atualização de tabelas, geração de alertas, num equipamento que coexiste com outras aplicações industriais. No sistema central, a exigência incide sobre

a capacidade de suportar consultas filtradas e importações em lote sem latências significativas à medida que o repositório cresce.

A fiabilidade é crítica em ambos, mas por razões distintas. No sistema local, a prioridade é a consistência dos dados face a falhas e interrupções inesperadas, sem conectividade externa que sirva de *fallback*. No sistema central, a rastreabilidade das importações, incluindo falhas, é determinante para manter a integridade do repositório consolidado.

A segurança assenta em controlo de acesso por perfis e armazenamento de credenciais por *hash* sem exposição em texto simples. No sistema central, a necessidade é mais crítica por envolver dados de múltiplos clientes e instalações. A usabilidade, a manutenibilidade com arquitetura modular, a portabilidade e a rastreabilidade de operações completam o conjunto, com especificações detalhadas na tabela seguinte (Tabela 3).

Tabela 3 – Requisitos não funcionais do sistema em geral

ID	Requisito não funcional	Sistema local	Sistema central
RNF01	Tempos de resposta adequados para operações típicas	✓	✓ (maior carga)
RNF02	Eficiência no consumo de recursos	✓ (máquina local)	✓ (servidor)
RNF03	Robustez a falhas e consistência de dados	✓	✓ (importações)
RNF04	Segurança: autenticação, perfis e proteção de credenciais	✓	✓ (mais crítico)
RNF05	Usabilidade e <i>feedback</i> claro (erros/alertas)	✓	✓ (filtros/ <i>import</i>)
RNF06	Capacidade de Manutenção: arquitetura modular e padrões	✓	✓
RNF07	Capacidade de evolução: compatibilidade entre versões	■ (menos crítico)	✓ (muito relevante)
RNF08	Portabilidade/compatibilidade	✓	✓ (<i>browsers</i>)
RNF09	Auditoria e rastreabilidade	✓	✓ (essencial)

3.3. CASOS DE USO

Os casos de uso ilustram como os utilizadores de cada perfil interagem com as funcionalidades dos dois módulos, complementando a especificação de requisitos com cenários de utilização real.

3.3.1. SISTEMA LOCAL

Os atores do sistema local correspondem aos três perfis definidos nos requisitos: observador, operador de manutenção e administrador. A hierarquia de permissões é cumulativa (herança), conforme ilustrado na Figura 7.

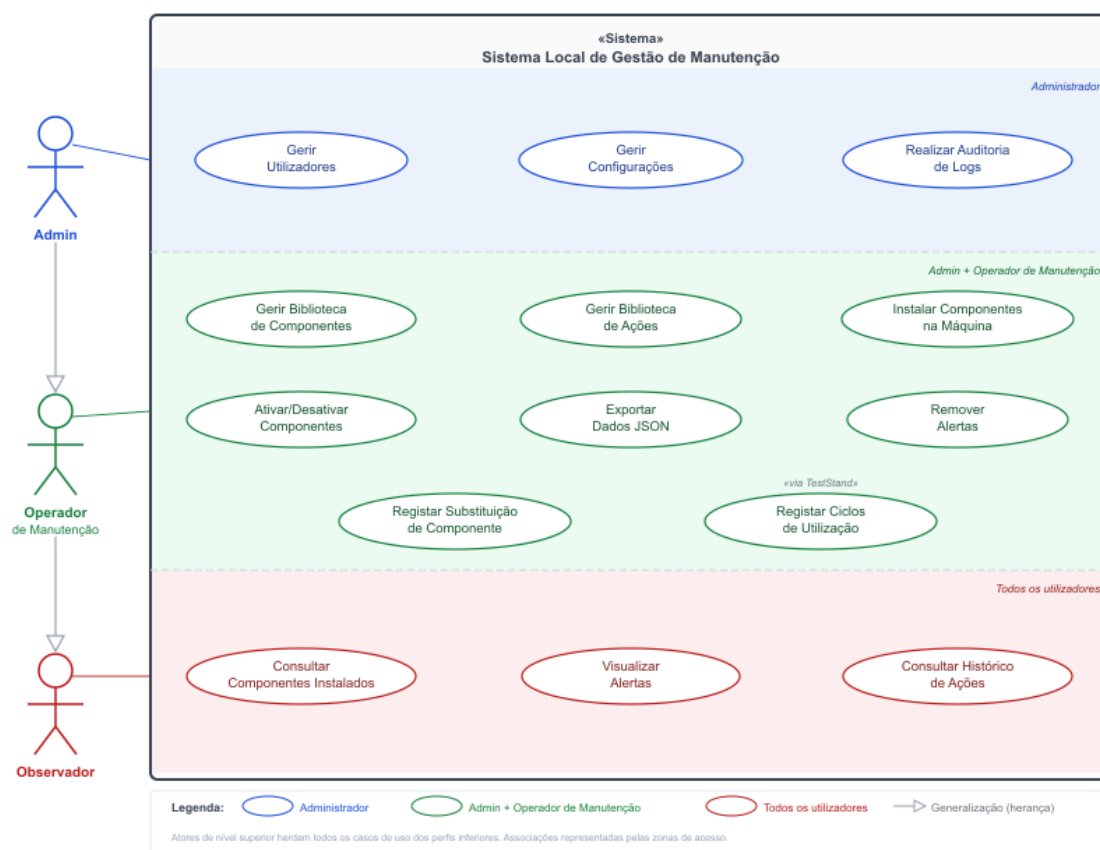


Figura 7 – Diagrama de casos de uso do sistema local

O caso de uso central é o registo de ciclos de utilização de um componente instalado. O sistema recebe automaticamente, através da integração com o *NI TestStand*

via biblioteca .NET, os dados de utilização produzidos durante a operação da máquina, atualizando o contador de *working cycles* sem intervenção manual. Quando um componente atinge o limiar de alerta configurado, o sistema gera automaticamente uma notificação visível a todos os utilizadores, constituindo o mecanismo nuclear de suporte à manutenção preventiva.

Quando o alerta é gerado e a intervenção é decidida, o operador procede à substituição física e regista a ação no sistema. O registo anterior é marcado como removido, preservando o histórico, e um novo registo é criado com o contador reiniciado a zero. O sistema fecha assim o ciclo de manutenção preventiva, desde a deteção da necessidade até ao registo da resolução, com rastreabilidade completa e sem intervenção manual sobre os registos históricos.

A gestão das bibliotecas de componentes e ações precede qualquer instalação: o componente deve existir na biblioteca com o seu limiar de utilização, e a ação correspondente deve estar registada antes de poder ser associada. Esta estrutura de dados de referência garante consistência nos registos ao longo do tempo.

A exportação de dados destina-se ao operador ou administrador e consiste na geração de um ficheiro JSON com o estado atual da máquina. O ficheiro é produzido diretamente a partir da base de dados local no formato normalizado esperado pelo sistema central, sendo posteriormente importado na plataforma central como mecanismo de sincronização assíncrona entre os dois módulos.

3.3.2. SISTEMA CENTRAL

Os atores do sistema central refletem um contexto distinto: engenheiros e gestores da Controlar que necessitam de uma visão consolidada sobre o comportamento dos equipamentos instalados nos clientes, não técnicos a operar junto das máquinas. Os perfis incluem o administrador e utilizadores com perfil analítico, conforme ilustrado na Figura 8.

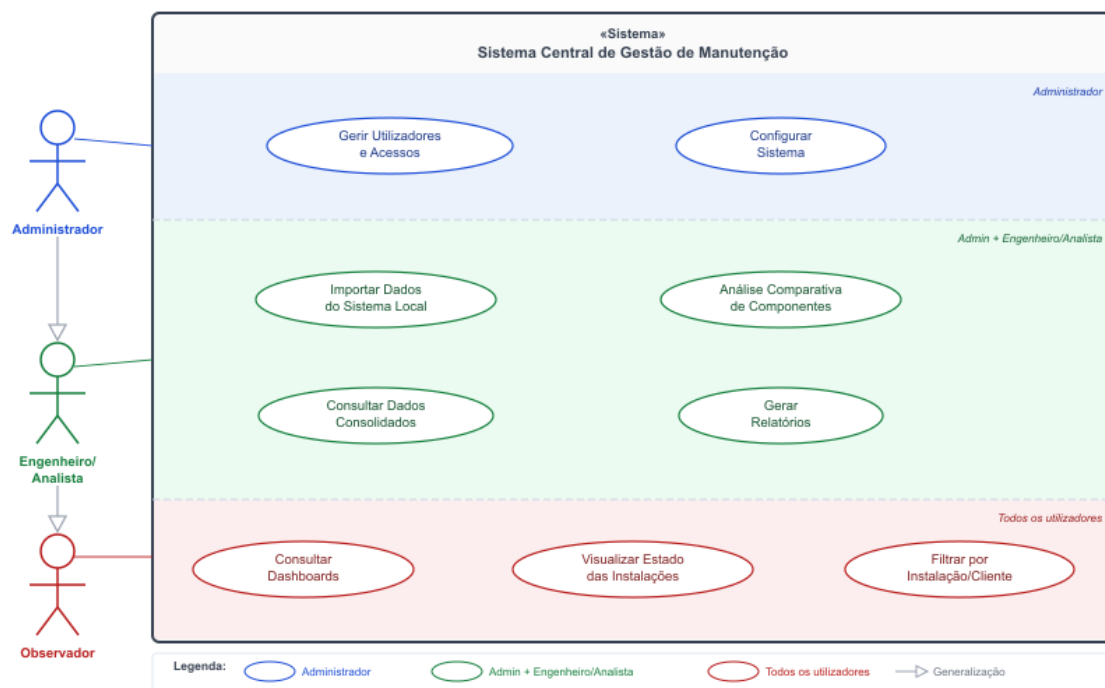


Figura 8 – Diagrama de casos de uso do sistema central

O caso de uso central é a ingestão e consolidação de dados provenientes de múltiplas instalações. O engenheiro/analista importa o ficheiro JSON exportado pelo sistema local, e a plataforma valida, normaliza e integra a informação no repositório consolidado. A qualidade desta ingestão é determinante para a utilidade analítica do sistema, uma vez que a comparabilidade entre instalações depende diretamente da consistência dos dados armazenados.

A consulta de estatísticas de vida útil por componente constitui o principal valor entregue pelo sistema central. O engenheiro acede aos valores de média, percentil 50 e percentil 90 calculados a partir de observações reais de múltiplas instalações, podendo filtrar por fabricante, tipo e número mínimo de amostras. Esta capacidade analítica permite a Controlar acumular conhecimento sobre a durabilidade real dos seus componentes e utilizar essa informação para fundamentar especificações técnicas e decisões de componentes em projetos futuros.

A análise conjunta dos casos de uso dos dois módulos evidencia a complementaridade que estrutura toda a solução: o sistema local responde às necessidades imediatas de rastreabilidade e alerta no ponto de operação, enquanto o sistema central transforma esses dados operacionais em conhecimento organizacional de longo prazo.

3.4. DESIGN DA BASE DE DADOS

A camada de persistência é um componente de design com peso próprio em ambos os subsistemas. As decisões ao nível do esquema relacional, das restrições e da organização das operações por *stored procedures* determinam diretamente a rastreabilidade, a integridade e a evolução futura dos dados.

3.4.1. BASE DE DADOS DO SISTEMA LOCAL

O esquema do sistema local organiza-se em torno de duas entidades de referência e um conjunto de entidades operacionais e de auditoria, conforme apresentado na Figura 9.

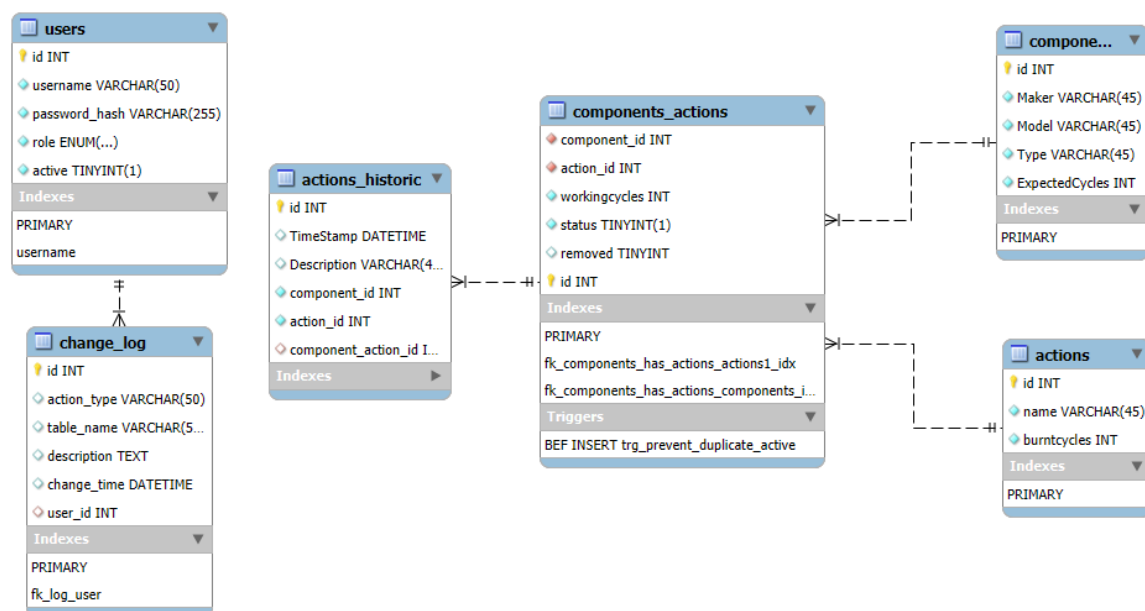


Figura 9 – Esquema relacional de base de dados do sistema local

A tabela *components* é o catálogo de componentes disponíveis para instalação, armazenando fabricante, modelo, tipo e o valor de *expected_cycles* que define o limiar de

vida útil para efeitos de alerta. A tabela *actions* é a biblioteca de ações de manutenção, com restrição de unicidade sobre o campo *name* que impede registos duplicados. Estas duas tabelas constituem os dados de referência estáveis sobre os quais as operações do sistema se apoiam.

A relação entre componentes e ações é materializada em *components_actions*, que representa a instalação efetiva de um componente numa ação específica. A chave primária composta pelos identificadores de componente e ação, com chaves estrangeiras para as tabelas de referência, assegura integridade referencial. Os campos operacionais desta tabela sustentam a funcionalidade central: *working_cycles* mantém o contador acumulado; *status* regista o estado ativo ou inativo; *removed* suporta a remoção lógica, preservando o histórico sem eliminar fisicamente a linha. A consistência desta restrição é reforçada por um *trigger* que atua antes de cada inserção, impedindo entradas ativas duplicadas para o mesmo par componente-ação, mesmo em cenários de acesso concorrente.

A remoção lógica é uma decisão de design deliberada: quando um componente é substituído, o registo anterior fica marcado como removido e um novo registo é criado com o contador a zero. A linha temporal completa das instalações fica preservada, os registos em tabelas dependentes mantêm a sua integridade e o histórico permanece reconstruível para qualquer componente instalado em qualquer momento no passado.

O histórico de intervenções é registado em *actions_historic* com *timestamp* e descrição de cada evento técnico. A tabela *users* armazena credenciais por *hash* e o papel de acesso de cada utilizador. A auditoria funcional fica em *change_log*, que regista para cada operação sensível o utilizador responsável, o *timestamp*, o verbo (*INSERT*, *ACTIVATE*, *DEACTIVATE*, *REPLACE*, *REMOVE*), a entidade alvo e uma descrição contextual. Este mecanismo, alimentado automaticamente pelo procedimento *LogChange* após cada operação relevante, garante rastreabilidade completa sem intervenção manual.

Todas as operações sobre este esquema são realizadas exclusivamente por *stored procedures*, centralizando as regras de manipulação e validação dos dados no motor de persistência e tornando o comportamento do sistema consistente independentemente da origem do pedido.

3.4.2. BASE DE DADOS DO SISTEMA CENTRAL

O esquema do sistema central foi concebido para suportar ingestão de múltiplas origens, reduplicação e cálculo estatístico sobre volumes de dados crescentes, conforme apresentado na Figura 10.

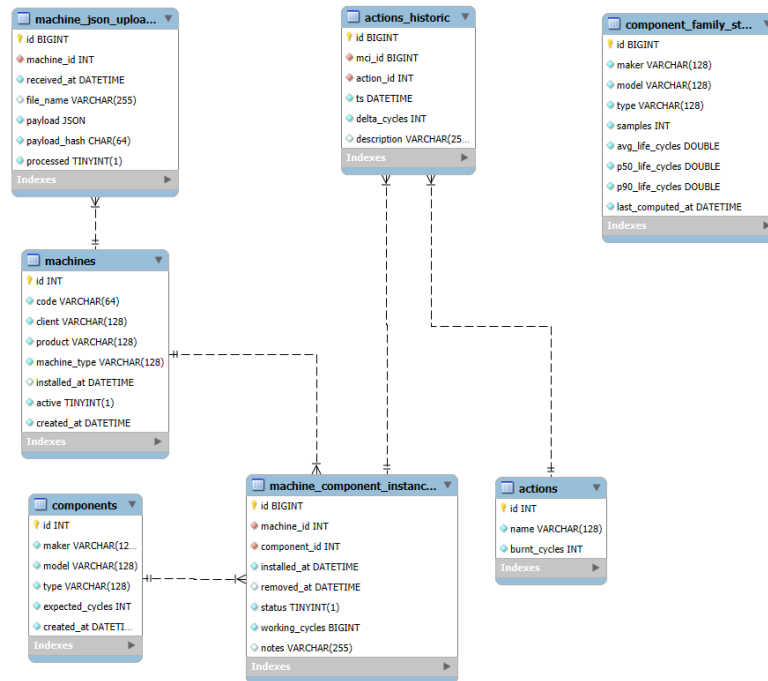


Figura 10 – Esquema relacional de base de dados do sistema central

A tabela *machines* regista cada instalação distinta pelo seu código identificador, que funciona como chave de agrupamento para todos os dados provenientes da mesma origem. A tabela *machine_json_uploads* armazena cada importação recebida com os metadados de origem, data, nome do ficheiro e um *hash* SHA-256 (*Secure Hash Algorithm*) do *payload*. Uma função criptográfica que produz uma representação hexadecimal de 256 bits, determinística e única para cada conteúdo distinto: o mesmo ficheiro produz sempre o mesmo valor, e qualquer alteração ao conteúdo, por mínima que seja, produz um valor completamente diferente e verifica se já existe um registo com esse *hash* para a mesma máquina. O campo *processed* distingue *uploads* recebidos de uploads efetivamente integrados nas tabelas analíticas, suportando um pipeline de ingestão em duas fases: receção e processamento deliberado.

A tabela *machine_component_instances* armazena os componentes extraídos de cada *upload*, associados ao *upload* de origem e à máquina correspondente. Esta rastreabilidade de proveniência é preservada permanentemente: é sempre possível saber de que *upload* e de que máquina um registo específico provém. A tabela *component_family_stats* armazena as estatísticas calculadas por combinação de fabricante, modelo e tipo, a unidade de análise do sistema central, com os valores de média, percentil 50 e percentil 90 dos ciclos de vida observados, o número de amostras que sustentam o cálculo e o momento em que foi executado.

A separação entre o armazenamento de amostras brutas e das estatísticas calculadas é intencional: permite recalculas as métricas a qualquer momento com base nos dados acumulados, sem perder informação, e torna o cálculo auditável, é sempre possível verificar quantas amostras sustentam cada estimativa. O cálculo dos percentis utiliza *window functions* do MySQL, especificamente *PERCENT_RANK* e *ROW_NUMBER*, que permitem computar os percentis de forma eficiente diretamente sobre o conjunto de observações disponíveis para cada componente.

A tabela *users* do sistema central é independente da do sistema local, mantendo credenciais e perfis de acesso específicos desta plataforma. Tal como no sistema local, todas as operações sobre o esquema são realizadas por *stored procedures*, com *IngestMachineJson* como procedimento central de ingestão e *ComputeComponentFamilyStats* como procedimento de cálculo estatístico.

3.5. DESIGN DA SOLUÇÃO DE INTERAÇÃO

As decisões de design descritas nesta secção respondem ao conjunto de restrições identificadas na análise de requisitos: necessidade de operação autónoma sem conectividade, integração com o NI TestStand por ciclos de utilização efetivos, rastreabilidade completa e capacidade de agregar dados de múltiplas instalações para análise comparativa.

3.5.1. ARQUITETURA GERAL

A solução foi concebida como um sistema híbrido composto por dois subsistemas complementares com responsabilidades bem delimitadas. Esta separação não resulta de uma distinção de escala, mas de uma diferença de propósito: o sistema local mantém-se orientado à execução operacional e à evidência transaccional, o que acontece, quando acontece e em que estado, enquanto o sistema central evolui como plataforma de análise e aprendizagem organizacional, orientada à comparabilidade e extração de padrões com robustez estatística crescente.

A sincronização entre os dois módulos é deliberada, controlada e assíncrona: ocorre quando um utilizador autorizado exporta dados do sistema local e os importa na plataforma central. Esta separação garante que a existência do sistema central não introduz nenhum ponto de falha no processo de manutenção operacional, o sistema local funciona de forma completamente autónoma, independentemente do estado ou disponibilidade da plataforma central.

3.5.2. DESIGN DO SISTEMA LOCAL

O sistema local foi estruturado segundo o padrão MVC (*Model–View–Controller*), que promove separação clara de responsabilidades entre camada de apresentação, camada de coordenação e camada de domínio e persistência.

A *View* é a interface *web*, concentrada na apresentação de informação e na recolha de interações. O *Controller* materializa-se no servidor Python que recebe pedidos de qualquer origem autorizada, interface, sequenciadores de teste ou PLCs (*Programmable Logic Controller*), aplica validações e coordena o fluxo de execução. A *Model* encapsula o acesso à base de dados por *stored procedures*, funcionando como guardião da consistência e integridade da informação.

Esta separação não é apenas uma convenção de organização de código. Em sistemas onde a rastreabilidade e a auditabilidade são requisitos operacionais de primeira ordem, centralizar o controlo de fluxo num ponto bem definido e reduzir o acoplamento

entre camadas tem peso real sobre a manutenibilidade e a fiabilidade do sistema a longo prazo.

O controlo de acesso por perfis, administrador, operador de manutenção e observador, é integrado na arquitetura e não apenas na interface: as restrições são aplicadas ao nível dos serviços, independentemente da origem do pedido, seguindo o princípio do menor privilégio. Na Figura 11 sugere em como os módulos do Sistema Local estão divididos.

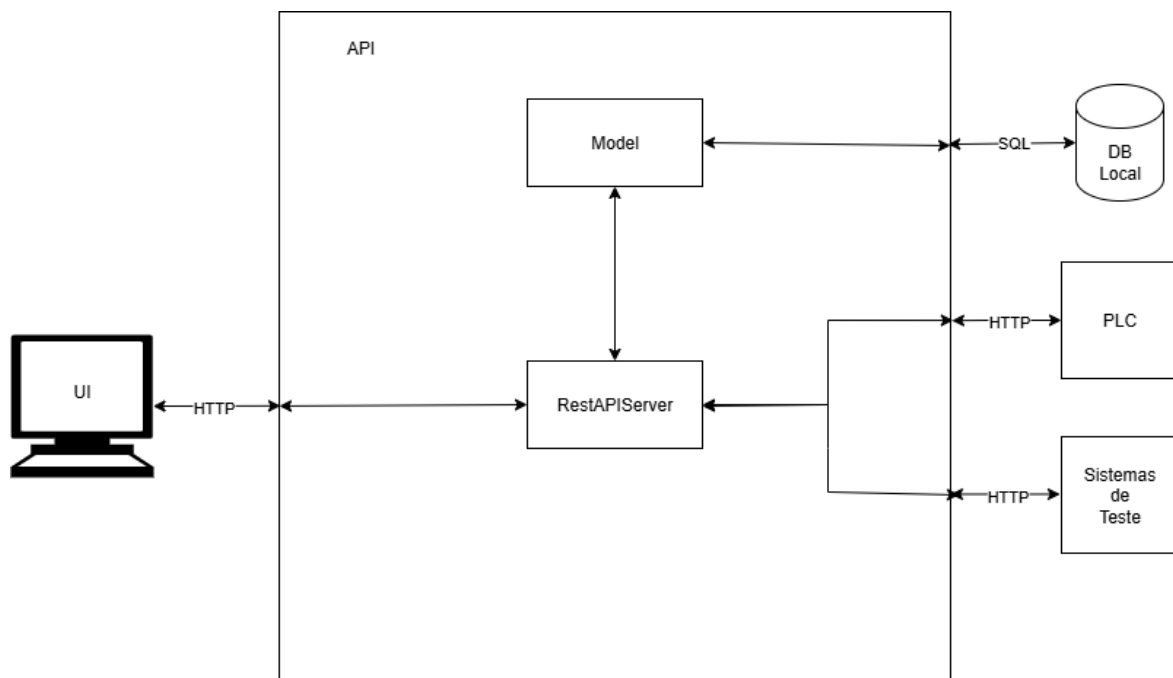


Figura 11 – Diagrama do sistema local

A integração com o NI TestStand é suportada por uma biblioteca .NET (*ControlarMaintenanceClient*) que expõe ao adaptador .NET do TestStand uma interface de tipos nativos, *string*, *int*, *bool*, para incremento de contadores de ciclos. O padrão de utilização segue a convenção habitual de *drivers* de instrumento no TestStand: *Connect* no início da sequência, *IncrementWorkingCycles* ao longo da execução, *Disconnect* no bloco de limpeza.

3.5.3. DESIGN DO SISTEMA CENTRAL

O sistema central pode ser descrito como uma cadeia de valorização de dados organizada em três etapas sequenciais, que asseguram que a qualidade e comparabilidade dos dados ficam estabelecidas antes de qualquer análise.

A ingestão recebe os ficheiros JSON exportados pelos sistemas locais, valida a estrutura e integridade, deteta reimportações por *hash SHA-256* e regista os metadados de origem. O processamento extrai os componentes do *payload*, associa-os à máquina de origem e calcula as estatísticas de vida útil por combinação de fabricante, modelo e tipo, atualizando a tabela de resultados com o conjunto de observações acumulado. A disponibilização expõe os dados consolidados por interface *web* e *endpoints*, com consultas filtradas, visualização de métricas e exportação de resultados.

O padrão MVC do sistema local mantém-se no sistema central, com a separação entre a camada de apresentação, o servidor Flask e o módulo de acesso a dados. A separação entre os dois sistemas ao nível da camada de serviços é assegurada pelo mecanismo de *Blueprint* do Flask, que organiza os *endpoints* do sistema central sob um prefixo dedicado num módulo independente, permitindo que o mesmo processo de servidor sirva ambos os módulos com responsabilidades claramente delimitadas. Na Figura 12 podemos ver os módulos do Sistema Central.

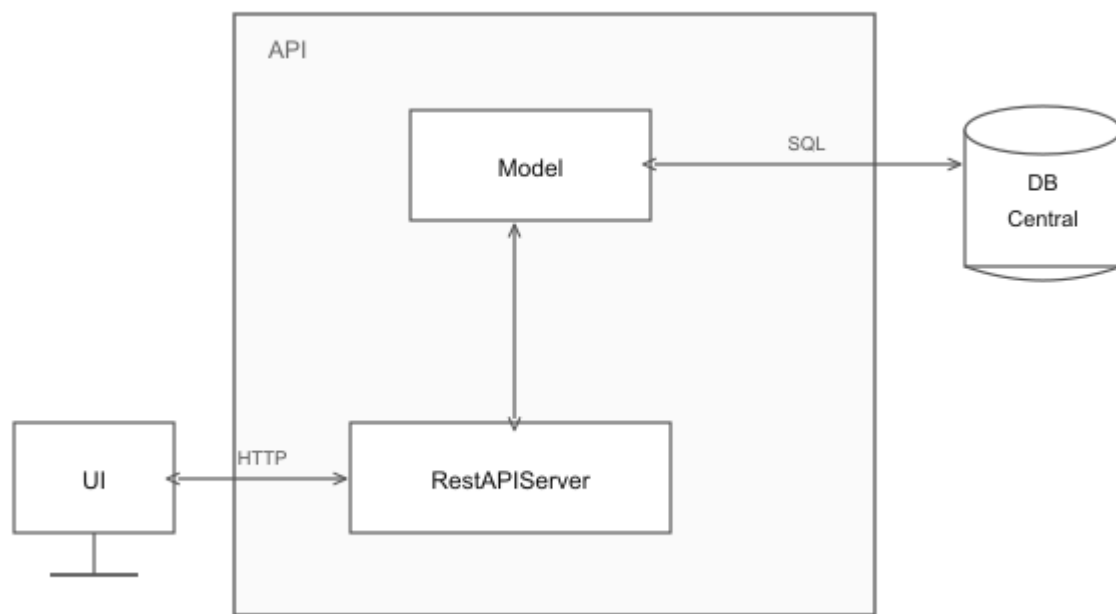


Figura 12 – Diagrama do sistema central

A coerência arquitetural entre os dois subsistemas, mesmo padrão de organização por camadas, mesmo mecanismo de acesso a dados por procedimentos armazenados, mesma estrutura de resposta normalizada, é uma decisão deliberada que reduz a carga cognitiva de manutenção e torna o conhecimento adquirido num módulo diretamente aplicável no outro.

4. IMPLEMENTAÇÃO DO SISTEMA

Este capítulo descreve a concretização da solução, com foco nos mecanismos que determinam o comportamento de cada funcionalidade. A estrutura segue a divisão entre interface de utilizador e camada de serviços, cobrindo em cada secção os dois módulos, local e central, quando partilham abordagem, ou distinguindo-os quando as implementações divergem.

A solução está organizada em dois módulos com diretórios independentes, cada um contendo a camada de servidor Python, o módulo de acesso à base de dados, os ficheiros de *frontend* e os *scripts* de configuração e arranque. A estrutura de ficheiros relevante de cada módulo, excluindo artefactos de compilação, caches e ficheiros utilitários é apresentada na Figura 13 podemos ver a arvore de ficheiros do sistema geral.

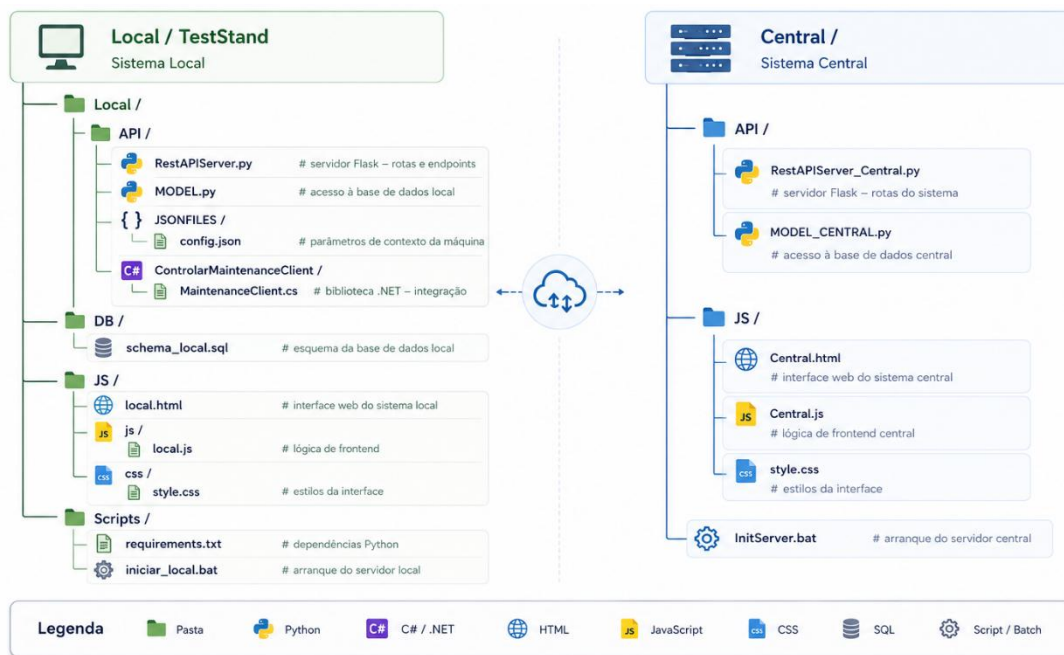


Figura 13 – Estrutura de ficheiros do sistema geral

No sistema local, o ficheiro *RestAPIServer.py* concentra todas as rotas Flask e a lógica de validação de pedidos; o *MODEL.py* encapsula o acesso à base de dados exclusivamente através de *stored procedures*. A biblioteca *ControlarMaintenanceClient*, implementada em C# para o .NET Framework 4.8, expõe ao adaptador .NET do TestStand os métodos de integração necessários para o incremento automático de contadores de ciclos. O ficheiro *config.json* armazena os parâmetros de contexto da instância, cliente, produto e tipo de máquina, que compõem o código identificador utilizado nas exportações. No sistema central, a organização replica o mesmo padrão: *RestAPIServer_Central.py* para as rotas e *MODEL_CENTRAL.py* para o acesso à base de dados, com o *frontend* servido pelos ficheiros *Central.html* e *Central.js*.

4.1. INTERAÇÃO COM O UTILIZADOR

A interface de utilizador de ambos os módulos foi implementada com HTML5, CSS3 e JavaScript sem recurso a *frameworks* externas. Apesar dos dois módulos terem propósitos distintos, o sistema local orientado ao registo e alerta operacional, o sistema

central orientado à análise, partilham a mesma arquitetura de *frontend*, o que torna o conhecimento adquirido num diretamente aplicável no outro. As subsecções seguintes organizam a implementação por camada funcional: primeiro a estrutura de navegação e comunicação com o servidor, depois a visualização de dados e o sistema de alertas.

4.1.1. ESTRUTURA E NAVEGAÇÃO

O padrão adotado em ambos os módulos é o de uma aplicação de página única em que cada secção é um elemento *div* com *display:none* por omissão. A navegação consiste em esconder a secção ativa e mostrar a seleccionada, sem recarregamento da página. Esta abordagem elimina dependências de empacotamento ou compilação e garante compatibilidade com os navegadores presentes nos postos de trabalho industriais sem configuração adicional.

A comunicação com o servidor é feita exclusivamente por chamadas assíncronas com *fetch*, seguindo um padrão uniforme em toda a interface: invoca o *endpoint* correspondente, aguarda a resposta JSON e reconstrói o elemento visual a partir dos dados recebidos. A interface nunca assume que uma operação foi bem-sucedida sem confirmação do servidor; após qualquer escrita solicita imediatamente a listagem atualizada. O excerto seguinte ilustra este padrão aplicado à listagem de componentes instalados, onde os registos removidos são excluídos no cliente antes da renderização e o campo de estado é normalizado de valor inteiro para texto legível.

```
async function fetchInstalledComponents() {
  const res = await fetch(
    'http://127.0.0.1:23456/Maintenance/DB/ListInstalledComponents'
  );
  const data = await res.json();
  return data
    .filter(row => row[row.length - 1] !== 1) // excluir removidos
    .map(row => {
      const newRow = [...row];
      newRow[newRow.length - 2] = newRow[newRow.length - 2] === 1
        ? 'Active' : 'Deactivated';
      newRow.splice(newRow.length - 1, 1);
      return newRow;
    });
}
```

O resultado de cada operação é comunicado através de notificações *toast* geradas dinamicamente, que confirmam o sucesso ou descrevem o erro sem interromper a sessão. Para operações com consequências difíceis de reverter, como o encerramento de sessão quando existem dados por exportar, a interface apresenta um *popup* de confirmação implementado com uma *Promise*: o fluxo fica suspenso até à decisão do utilizador, garantindo que a exportação é concluída antes de a sessão ser encerrada.

A adaptação ao perfil ativo é aplicada no cliente após cada autenticação bem-sucedida. A função *ajustarInterfacePorRole* percorre os elementos marcados com classes de restrição e altera a sua visibilidade conforme o papel recebido do servidor. Formulários de inserção, botões de exportação e secções administrativas são ocultados para perfis sem permissão. Esta adaptação é uma camada de usabilidade; a autorização efetiva é sempre verificada ao nível dos serviços, independentemente da origem do pedido.

4.1.2. VISUALIZAÇÃO DE DADOS E ALERTAS

As tabelas são construídas dinamicamente a partir dos *arrays* JSON devolvidos pelo servidor. Na tabela de componentes instalados, a linha recebe uma classe CSS de cor determinada pela percentagem de utilização: verde abaixo de oitenta por cento, amarelo entre oitenta e noventa, laranja entre noventa e noventa e cinco, vermelho acima. Os componentes desativados são destacados com fundo cinzento-escuro independentemente da percentagem. A coluna de estado, para os perfis de administrador e operador, é substituída por um conjunto de botões de ação, ativar, desativar, remover, substituir, gerados dinamicamente em função do estado atual do componente; para o perfil de observador, a coluna apresenta apenas texto.

A secção de alertas é a que concentra mais lógica calculada no cliente. A função *renderAlertsFromInstalled* recebe a listagem de componentes instalados, já obtida para a tabela principal, sem pedido adicional ao servidor e percorre os componentes ativos extraindo a percentagem de utilização de cada entrada. Com base nessa percentagem, classifica cada componente em três níveis de severidade: baixo a partir de oitenta por cento, médio a partir de noventa, crítico a partir de noventa e cinco. Para cada componente que ultrapassa algum destes limiares, gera dinamicamente um cartão de alerta com a

severidade, o nome do componente, a ação associada e a percentagem atual. O nível máximo de severidade presente determina a cor e o valor numérico do indicador na barra lateral, tornando o estado de alerta visível em qualquer secção da interface sem necessidade de navegar para a secção de alertas.

```
function renderAlertsFromInstalled(data) {
  const alertContainer = document.getElementById('alerts-list');
  alertContainer.innerHTML = '';
  let highestLevel = 0;
  const counter = { low: 0, medium: 0, critical: 0 };

  data
    .filter(row => row[3] === 'Active')
    .forEach(row => {
      const match = row[2].match(/\((\d+)\%/);
      if (!match) return;

      const percent = parseFloat(match[1]);
      let nivel = null, cls = '';

      if (percent >= 95) { level = '⚠️ Crítico'; cls =
'alert-critical'; highestLevel = Math.max(highestLevel, 3);
counter.critical++; }
      else if (percent >= 90) { level = '🔴 Médio'; cls =
'alert-medium'; highestLevel = Math.max(highestLevel, 2);
counter.medium++; }
      else if (percent >= 80) { level = '🟡 Baixo'; cls =
'alert-low'; highestLevel = Math.max(highestLevel, 1);
counter.low++; }

      if (!level) return;

      const div = document.createElement('div');
      div.className = `alert-box ${cls}`;
      div.innerHTML = `<strong>${nivel}</strong> - Componente
<strong>${row[0]}</strong>
na acção <strong>${row[1]}</strong> atingiu
<strong>${percent}%</strong>.`;
      alertContainer.appendChild(div);
    });

  // Actualiza badge e cor do link na sidebar
  const badge = document.getElementById('badge-alerts');
  const total = counter.low + counter.medium + counter.critical;
  badge.textContent = total;
  badge.style.display = total > 0 ? '' : 'none';
  badge.style.background = highestLevel === 3 ? '#e74c3c'
: highestLevel === 2 ? '#e67e22' :
'#f1c40f';

  renderAlertsChart(contagem, total);
}
```

A Figura 14 apresenta a secção de componentes instalados da interface do sistema local, onde são visíveis os indicadores de utilização por linha, a codificação cromática por nível de desgaste e os botões de ação condicionais ao perfil ativo.



Figura 14 – Interface do sistema local

O sistema central partilha o mesmo padrão de construção dinâmica, mas acrescenta visualizações analíticas implementadas com a biblioteca Chart.js. O ecrã inicial após autenticação é um *dashboard* que agrega dados de três *endpoints* existentes e apresenta quatro cartões de indicadores: total de máquinas, total de *uploads*, *uploads* por processar e componentes com estatísticas calculadas. Um aviso é apresentado quando algum componente tem menos de cinco amostras, orientando o utilizador para interpretar as estimativas com o grau de confiança adequado abaixo representado na Figura 15.



Figura 15 – Dashboard sistema central

A secção de estatísticas apresenta dois gráficos gerados com Chart.js. O primeiro é um gráfico de barras agrupadas que mostra, para cada tipo de componente, os valores de média, percentil 50 e percentil 90 lado a lado, tornando imediatamente visível a dispersão entre as três métricas. O segundo é um gráfico de barras horizontais que representa o número de amostras por componente, com codificação cromática que distingue componentes com volume suficiente dos que têm dados insuficientes. As instâncias Chart.js são guardadas em variáveis de módulo e destruídas antes de cada atualização, evitando acumulação de telas em memória. A secção de relatórios inclui igualmente um gráfico de barras horizontais com os componentes recomendados ordenados pelo critério selecionado, complementando a tabela com uma leitura visual da hierarquia de durabilidade relativa, como se pode observar na Figura 16.

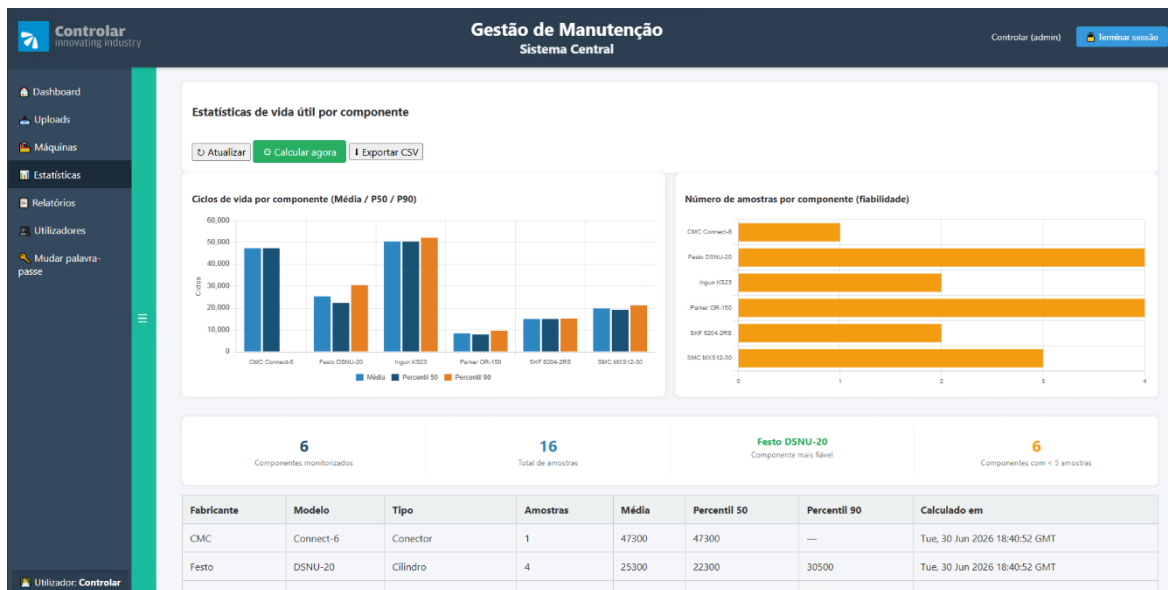


Figura 16 – Aba de estatísticas do sistema central

Em ambos os módulos, a interface resulta das mesmas opções de base: comunicação assíncrona sem recarregamento, renderização dinâmica derivada do estado persistido e controlo de acesso aplicado ao nível dos serviços. No sistema local, estas opções servem a operação diária de registo e alerta junto da máquina; no sistema central, a mesma arquitetura suporta um contexto analítico com maior peso visual e menor frequência de escrita. A distinção de propósito entre os dois módulos não implicou uma distinção de abordagem e essa coerência tem consequências práticas na manutenção e evolução futura de ambos.

4.2. CAMADA DE SERVIÇOS

A camada de camada de serviços de ambos os módulos foi implementada em Python com Flask, organizando-se em dois ficheiros com responsabilidades bem delimitadas. O ficheiro de rotas (*RestAPIServer*) recebe os pedidos HTTP, aplica validações e encaminha para o modelo; o ficheiro de modelo (*MODEL*) encapsula toda a interação com a base de dados. Esta separação garante que as regras de negócio e as regras de persistência não estão misturadas no mesmo módulo.

O padrão de implementação das rotas é uniforme: validação dos parâmetros recebidos, delegação para o modelo e construção de uma resposta JSON com os campos *success*, *error* e, quando aplicável, os dados solicitados. O excerto seguinte, retirado do módulo de rotas do sistema local, ilustra este padrão numa rota de inserção de ação.

```
@api.route('/Maintenance/DB/InsertAction/<string:Name>/<int:BurntCycles>')
def InsertAction(Name, BurntCycles):
    user_id = request.args.get('user_id', type=int)
    if user_id is None:
        return jsonify({'success': False,
                        'error': 'ID de utilizador em falta'}), 400
    if not Name or BurntCycles <= 0:
        return jsonify({'success': False,
                        'error': 'Dados inválidos'}), 400
    result = md.insert_Action(Name, BurntCycles, user_id)
    return jsonify(result), 200 if result.get('success') else 500
```

O módulo do modelo segue um padrão transacional consistente em todas as operações de escrita: abertura de ligação, invocação do *stored procedure*, *commit* em caso de sucesso, *rollback* em caso de exceção e encerramento da ligação no bloco *finally*. O excerto seguinte, retirado da função *insert_Action* do *MODEL*, ilustra este padrão incluindo o registo automático de auditoria através do procedimento *LogChange*.

```
def insert_Action(inName, inBurntCycles, user_id):
    db = get_db_connection()
    try:
        cur = db.cursor()
        args = [inName, inBurntCycles, 0]
        cur.callproc('InsertAction', args)
        if args[2] != -1:
            desc = f'Inseriu ação {inName} com {inBurntCycles} ciclos.'
        cur.callproc('LogChange', [user_id, 'INSERT', 'Actions', desc])
        db.commit()
        return {'success': True, 'code': args[2]}
    except Exception as e:
        db.rollback()
        return {'success': False, 'error': str(e)}
    finally:
        cur.close()
        db.close()
```

A integração com o NI TestStand ou com qualquer cliente que queira incrementar ciclos, exigiu um *endpoint* específico para incremento de ciclos de trabalho, invocável diretamente por sequenciadores de teste ou pela biblioteca .NET *ControlarMaintenanceClient* sem necessidade de passar pela interface *web*. A rota aceita o nome da ação como parâmetro de URL (*Uniform Resource Locator*) e devolve o novo valor do contador, ou um erro descritivo se a ação não existir. No excerto a seguir temos o excerto de código da definição da função executada pelo *endpoint* referido

```
@api.route('/Maintenance/DB/IncrementWorkingCycles/<string:ActionName>')
def IncrementWorkingCycles(ActionName):
    if not ActionName:
        return jsonify({'success': False,
                        'error': 'Nome da acção em falta'}), 400
    result = md.increment_WorkingCycles(ActionName)
    return jsonify(result), 200 if result.get('success') else 500
```

No modelo, a função *increment_WorkingCycles* chama o *stored procedure IncrementWorkingCycles*, que localiza a ação pelo nome, atualiza o contador na tabela *components_actions* e devolve o novo valor via parâmetro de saída. O retorno inclui sempre os campos *success* e *new_value*, garantindo que o chamador, seja a interface, o *TestStand* ou qualquer outro cliente, recebe uma resposta estruturada que pode interpretar sem ambiguidade.

4.2.1. EXPORTAÇÃO E INTEROPERABILIDADE

A exportação constitui o mecanismo de sincronização entre o sistema local e o sistema central. A operação está disponível para os perfis de administrador e operador e produz um ficheiro JSON com a estrutura normalizada esperada pelo sistema central. O conteúdo é extraído diretamente da base de dados local pela função *export_for_central*, que constrói o *payload* juntando a tabela de componentes com a tabela de instalações.

```

def export_for_central():
    db = get_db_connection()
    try:
        cur = db.cursor(dictionary=True)
        cur.execute("""
            SELECT c.Maker    AS maker,
                   c.Model    AS model,
                   c.Type     AS type,
                   c.ExpectedCycles AS expected_cycles,
                   ca.workingcycles AS working_cycles,
                   ca.status
            FROM components_actions ca
            JOIN components c ON ca.component_id = c.id
        """)
        components = cur.fetchall()
        with open(cfg_path, encoding='utf-8') as f:
            cfg = json.load(f)
            machine_code =
(f"{cfg.get('cliente','')}_{cfg.get('produto','')}_"
 f"{cfg.get('maquina','')}")
            return {'machine_code': machine_code, 'components':
components}
        finally:
            cur.close(); db.close()

```

O ficheiro gerado inclui o código da máquina, derivado dos parâmetros de configuração de cliente, produto e tipo de máquina, e a lista de componentes com todos os atributos necessários para a ingestão central. Os componentes com status ativo são incluídos com o contador de ciclos atual; os com ciclo concluído entram nos cálculos estatísticos do sistema central. A nomenclatura do ficheiro incorpora os parâmetros de contexto, tornando os ficheiros de diferentes instalações imediatamente identificáveis.

4.2.2 ESTRUTURA DO JSON DE EXPORTAÇÃO

O ficheiro produzido pela exportação segue um contrato de interface definido entre os dois módulos. A estrutura raiz contém dois campos obrigatórios: *machine_code*, que identifica de forma única a instalação exportadora, e *components*, que é o *array* de componentes instalados no momento da exportação. O *machine_code* é composto automaticamente pelos parâmetros de configuração de cliente, produto e tipo de máquina, garantindo que ficheiros de diferentes instalações são distinguíveis sem necessidade de

inspeção do conteúdo. O excerto exemplo seguinte ilustra um ficheiro de exportação real com dois componentes, um ativo e um com ciclo de vida concluído.

```
{
  "machine_code": "Controlar_Produto_Maquina01",
  "components": [
    {
      "maker": "SMC",
      "model": "MGPM25-100Z",
      "type": "Guia Linear",
      "expected_cycles": 5000000,
      "working_cycles": 275,
      "status": 1
    },
    {
      "maker": "SMC",
      "model": "CD85F10-80-B",
      "type": "Cilindro",
      "expected_cycles": 5000000,
      "working_cycles": 4987320,
      "status": 0
    }
  ]
}
```

O campo *status* distingue dois estados com significados operacionais distintos. O valor 1 indica que o componente está atualmente instalado e em serviço, o contador de *working_cycles* reflete os ciclos acumulados até ao momento da exportação, mas o ciclo de vida ainda não está concluído. O valor 0 indica que o componente foi substituído e o seu ciclo de vida está terminado, o *working_cycles* representa o número total de ciclos que o componente realizou até ao final da sua vida útil. Esta distinção é fundamental para o sistema central: apenas os componentes com status 0 entram nos cálculos estatísticos de vida útil, porque só estes representam observações completas de um ciclo de vida real. Os componentes com status 1 são armazenados como contexto operacional da máquina, mas não influenciam as estimativas estatísticas, para não introduzir enviesamento com amostras incompletas.

O campo *expected_cycles* é o valor configurado na biblioteca de componentes do sistema local para esse modelo específico, e representa a vida útil esperada segundo a especificação do fabricante ou o critério de substituição adotado pela Controlar. A razão

entre *working_cycles* e *expected_cycles* é o que determina, no sistema local, o nível de alerta gerado. No sistema central, esta mesma razão permite calcular a percentagem de vida útil utilizada no contexto de cada instalação, mas o valor que alimenta as estatísticas é o *working_cycles* absoluto de cada componente com ciclo concluído, não a percentagem, para que os cálculos de percentil reflitam os ciclos reais e não um valor normalizado artificialmente.

4.2.3. NORMALIZAÇÃO E CATEGORIZAÇÃO NO SISTEMA CENTRAL

Quando um ficheiro JSON chega ao *endpoint* de ingestão do sistema central, o *stored procedure IngestMachineJson* executa um conjunto de operações encadeadas numa única transação, garantindo que o repositório central nunca fica num estado parcialmente atualizado. O primeiro passo é a verificação de duplicados: O procedimento calcula o *hash* SHA-256 do *payload* recebido. Se existir, a ingestão é rejeitada com um código de duplicado e a operação termina sem alterar o estado do repositório. Este mecanismo previne que o mesmo ficheiro seja importado acidentalmente mais do que uma vez, situação frequente quando o processo de sincronização é manual e, garante que as estatísticas calculadas refletem observações únicas, como se ilustra no extrato de código a seguir.

```
-- Verificação de duplicado dentro de IngestMachineJson
SELECT COUNT(*) INTO v_dup
FROM machine_json_uploads
WHERE machine_id = v_machine_id
  AND payload_hash = p_payload_hash;

IF v_dup > 0 THEN
  SET p_upload_id = -2; -- código de duplicado
  LEAVE proc;
END IF;
```

Se o *payload* é novo, o procedimento regista ou atualiza a máquina na tabela *machines* pelo seu *machine_code*, cria o registo de *upload* com os metadados de origem, data e *hash*, e extrai cada componente do *array* JSON para a tabela *machine_component_instances*. A extração usa a cláusula *JSON_TABLE* do MySQL, que percorre o *array* de componentes e insere uma linha por entrada sem necessidade de

iteração na camada de aplicação, mantendo a operação atômica e eficiente. No excerto de código abaixo podemos ver como é feita a extração dos componentes do JSON

```
-- Extração dos componentes com JSON_TABLE
INSERT INTO machine_component_instances
(machine_id, upload_id, maker, model, type,
expected_cycles, working_cycles, status)
SELECT
v_machine_id, v_upload_id,
c.maker, c.model, c.type,
c.expected_cycles, c.working_cycles, c.status
FROM JSON_TABLE(
p_payload,
'$.components[*]' COLUMNS (
maker          VARCHAR(100) PATH '$.maker',
model          VARCHAR(100) PATH '$.model',
type           VARCHAR(100) PATH '$.type',
expected_cycles INT          PATH '$.expected_cycles',
working_cycles INT          PATH '$.working_cycles',
status         TINYINT       PATH '$.status'
)
) AS c;
```

A categorização dos componentes para efeitos estatísticos assenta na combinação de três campos: *maker*, *model* e *type*. Esta combinação tripla de identificadores funciona como chave de agrupamento, todos os componentes com o mesmo fabricante, modelo e tipo, independentemente da máquina de onde provieram, são tratados como pertencentes à mesma categoria para efeitos de cálculo de vida útil. A escolha de incluir o tipo na chave de agrupamento garante que variantes do mesmo modelo com características operacionais distintas não são agrupadas artificialmente, o que poderia enviesar as estimativas estatísticas. Na Figura 17, podemos ver como são agrupados os componentes.

Biblioteca de Componentes				
ID	Maker	Model	Type	ExpectedCycles
1	SMC	CD85F10-80-B	Cilindro	5000000
2	SMC	MGPM25-100Z	Cilindro	5000000
3	SMC	DF95C05-40-C	Cilindro	100

Figura 17 – Tabela de componentes

4.2.4. CONSOLIDAÇÃO ESTATÍSTICA

O cálculo das estatísticas de vida útil é a funcionalidade central do sistema central. O *stored procedure* *ComputeComponentFamilyStats* percorre todos os componentes com ciclo de vida concluído (*status* = 0), agrupa-os por combinação de fabricante, modelo e tipo, e calcula média, percentil 50 e percentil 90 dos ciclos observados. O cálculo dos percentis utiliza *window functions* do MySQL (*PERCENT_RANK* e *ROW_NUMBER*) sobre o conjunto de observações disponível para cada combinação. O excerto de código abaixo representa o *store procedure* que executamos para processar as estatísticas de uma família de componentes.

```
-- Fragmento do stored procedure ComputeComponentFamilyStats
INSERT INTO component_family_stats
  (maker, model, type, samples, avg_life_cycles,
   p50_life_cycles, p90_life_cycles, last_computed_at)
SELECT
  maker, model, type,
  COUNT(*) AS samples,
  AVG(working_cycles) AS avg_life_cycles,
  MAX(CASE WHEN pct_rank <= 0.50 THEN working_cycles END) AS
p50_life_cycles,
  MAX(CASE WHEN pct_rank <= 0.90 THEN working_cycles END) AS
p90_life_cycles,
  NOW()
FROM (
  SELECT maker, model, type, working_cycles,
    PERCENT_RANK() OVER (
      PARTITION BY maker, model, type
      ORDER BY working_cycles
    ) AS pct_rank
  FROM machine_component_instances
  WHERE status = 0 -- só ciclos concluídos
) ranked
GROUP BY maker, model, type
ON DUPLICATE KEY UPDATE
  samples = VALUES(samples),
  avg_life_cycles = VALUES(avg_life_cycles),
  p50_life_cycles = VALUES(p50_life_cycles),
  p90_life_cycles = VALUES(p90_life_cycles),
  last_computed_at = VALUES(last_computed_at);
```

A opção de guardar os resultados calculados numa tabela dedicada, em vez de os calcular em cada consulta, serve dois propósitos. Primeiro, o custo do cálculo fica restrito ao momento em que o utilizador decide processar novos *uploads*, sem afetar o

desempenho das consultas de leitura. Segundo, os valores ficam disponíveis para comparação histórica: é sempre possível verificar o que as estatísticas indicavam numa data específica e com quantas amostras foram calculadas, tornando o sistema auditável e não apenas descritivo.

O número de amostras disponível por componente é um indicador relevante na interpretação dos valores calculados. Componentes com poucas amostras têm distribuições estatísticas menos representativas da sua durabilidade real; a interface assinala este facto com uma advertência visual quando o número de amostras é inferior a cinco, orientando o utilizador para interpretar as estimativas com o grau de confiança adequado. Como podemos verificar na Figura 18.



Figura 18 – Avisos na Interface do sistema central

4.2.5. GESTÃO DE UTILIZADORES E ENCRIPTAÇÃO DE CREDENCIAIS

A gestão de utilizadores suporta o modelo de controlo de acesso por perfis que estrutura toda a solução. Três perfis com capacidades progressivamente mais abrangentes, observador, operador de manutenção e administrador, estão disponíveis; o papel de cada utilizador é armazenado na base de dados e propagado pela sessão a todas as operações subsequentes. A criação, desativação e atualização de credenciais estão reservadas ao administrador.

As credenciais nunca são armazenadas em texto simples. No momento da criação de um utilizador, a camada de modelo gera um *hash* seguro com a biblioteca *werkzeug.security* e é esse valor que fica registado na base de dados, tornando as credenciais irrecuperáveis mesmo em caso de acesso indevido ao repositório. A autenticação compara o valor

fornecido com o *hash* armazenado através da função *check_password_hash*, sem que a palavra-passe original circule na aplicação após a criação. O excerto seguinte ilustra a sequência de criação e verificação de credenciais, como podemos observar no excerto abaixo.

```
from werkzeug.security import generate_password_hash,
check_password_hash

# Criação de utilizador – só o hash é guardado
def create_user(username, password, role):
    db = get_db_connection()
    try:
        cur = db.cursor()
        pw_hash = generate_password_hash(password,
method='pbkdf2:sha256')
        args = [username, pw_hash, role, 0]
        result = cur.callproc('CreateUser', args)
        db.commit()
        return {'success': True} if result[3] != -1 \
            else {'success': False, 'error': 'Utilizador já
existe.'}
    except Exception as e:
        db.rollback()
        return {'success': False, 'error': str(e)}
    finally:
        cur.close(); db.close()

# Autenticação – verifica hash sem expor a palavra-passe
def verify_user_credentials(username, password):
    db = get_db_connection()
    try:
        cur = db.cursor()
        cur.execute(
            'SELECT id, password_hash, role FROM users WHERE
username = %s',
            (username,)
        )
        row = cur.fetchone()
        if row and check_password_hash(row[1], password):
            return {'success': True, 'user_id': row[0], 'role':
row[2]}
        return {'success': False, 'error': 'Credenciais inválidas'}
    finally:
        cur.close(); db.close()
```

A atualização de palavra-passe segue um fluxo de verificação em duas etapas: o utilizador fornece a palavra-passe atual, que é verificada contra o *hash* armazenado antes de gerar um novo *hash* e executar a atualização. Este mecanismo impede que uma sessão

aberta sem supervisão seja usada para alterar credenciais sem o conhecimento do titular da conta.

A desativação de um utilizador é realizada de forma lógica através do *stored procedure DeactivateUser*, que atualiza o campo *active* sem eliminar o registo. A listagem de utilizadores, filtrada pelo mesmo campo, deixa de incluir o utilizador desativado, preservando ao mesmo tempo a rastreabilidade das operações que executou durante o período em que esteve ativo.

5. DEMONSTRAÇÃO E RESULTADOS

O presente capítulo documenta a aplicação do sistema desenvolvido num contexto de produção real, utilizando como plataforma de referência a máquina RiTS ControlarS2.8 da Controlar, apresentada na Figura 19.

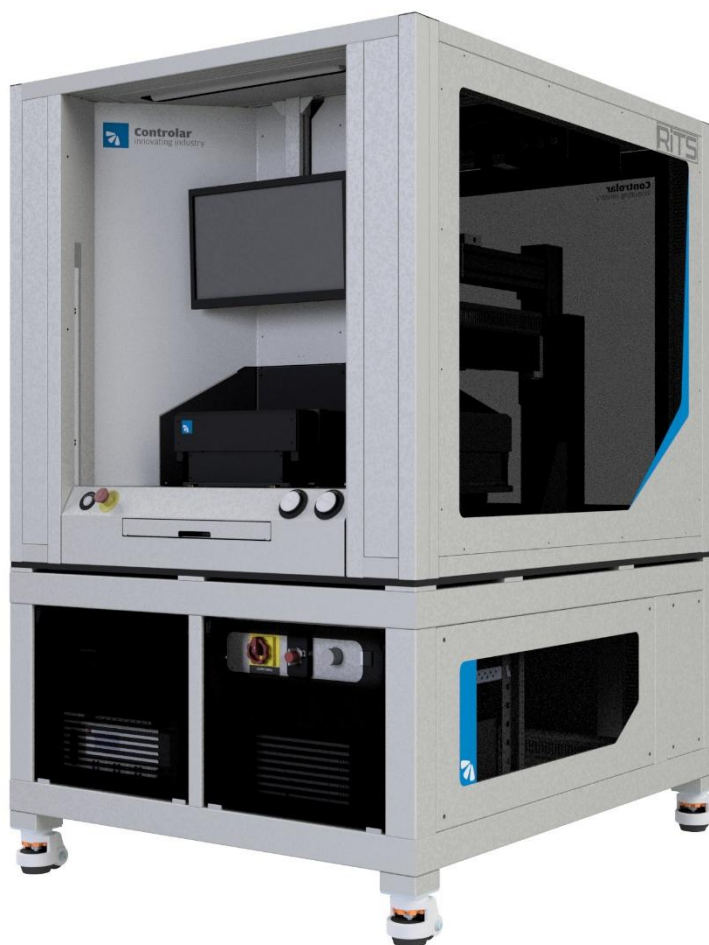


Figura 19 – RiTS

A estrutura segue o fluxo efetivo de utilização: configuração inicial da instância do equipamento, operação com integração ativa com o NI TestStand, demonstração do mecanismo de alertas e rastreabilidade das operações realizadas no sistema central de análise.

5.1. SISTEMA LOCAL DE MANUTENÇÃO

Este capítulo apresenta os resultados da aplicação da solução desenvolvida em contexto real, organizados de acordo com os dois módulos que compõem o sistema. A primeira secção documenta a instalação e operação do sistema local na máquina RiTS ControlarS2.8 da Controlar.

5.1.1. RiTS COMO CONTEXTO DE APLICAÇÃO

O RiTS (*Rotational Simultaneous Test System*) é um sistema automatizado de teste de fim de linha EOL (*End-of-Line*) desenvolvido pela Controlar. Assenta numa plataforma rotativa de quatro posições que permite realizar múltiplos testes em simultâneo, elétricos, óticos e mecânicos, sobre componentes automotivos e eletrónicos, otimizando o fluxo de trabalho industrial e reduzindo os tempos de ciclo em linha de produção. Em cada *step* de rotação, a plataforma sujeita as unidades sob teste a, verificações de luminância e inspeção com visão artificial, testes de háptica e acústica, testes elétricos e de comunicação.

Do ponto de vista da manutenção preventiva, o RiTS é um exemplo bastante representativo precisamente por essa cadência elevada de testes: em cada ciclo de teste implica o acionamento de múltiplos atuadores pneumáticos, cilindros de bloqueio das unidades, conectores, testes com uma Konica Minolta e cilindros de sensibilidade háptica, cujo desgaste acumula de forma diretamente proporcional ao número de itens testadas. A instalação do sistema de gestão de manutenção nesta máquina permitiu validar a integração com o NI TestStand em condições reais e observar o comportamento do sistema durante a sua operação efetiva.

5.1.2. CONFIGURAÇÃO DA INSTÂNCIA

A configuração da instância foi o ponto de partida da instalação. Na secção de configurações, foram definidos os três parâmetros de contexto que compõem o código identificador da máquina: cliente (Controlar), produto (ControlarS2.8) e tipo de máquina (RiTS). Este código “Controlar_ControlarS2.8_RiTS” é o identificador utilizado em todas as exportações JSON e funciona como chave de agrupamento no sistema central. Na Figura 20 podemos ver como se realiza esta configuração da máquina.

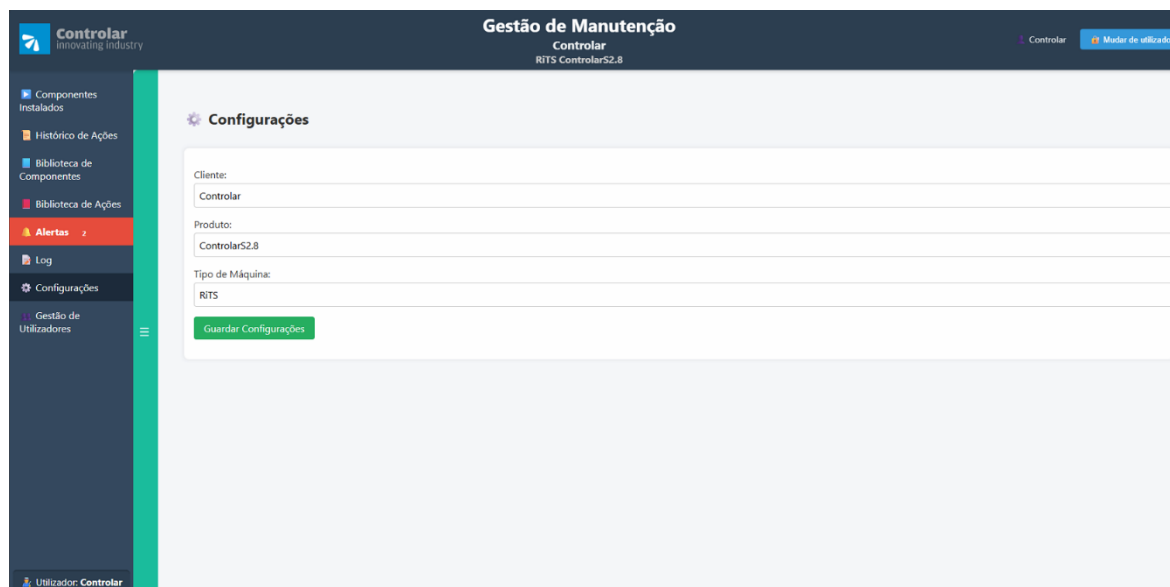


Figura 20 – Configuração da instância RiTS ControlarS2.8

A biblioteca de componentes foi constituída com três entradas correspondentes aos atuadores pneumáticos presentes na máquina. Os cilindros SMC CD85F10-80-B (cilindros de háptica) e SMC MGPM25-100Z (cilindros de conector e bloqueio) foram registados com o valor de ciclos esperados de 5 000 000, correspondente à especificação do fabricante para este tipo de atuador em ambiente industrial. Um terceiro componente, o SMC DF95C05-40-C, foi registado com um limiar de 100 ciclos, deliberadamente baixo, com o propósito de exercitar o sistema de alertas durante o período de instalação sem necessidade de aguardar volumes de operação prolongados. Na Figura 21 tem-se o formulário de inserção de componentes e a lista de componentes na RiTS.

Controlar

innovating industry

Gestão de Manutenção

Controlar

RITS ControlarS2.8

Controlar

Mudar de usuário

Componentes Instalados

Histórico de Ações

Biblioteca de Componentes

Biblioteca de Ações

Alertas 2

Log

Configurações

Gestão de Utilizadores

Utilizador: Controlar

Adicionar Novo Componente

Fabricante:

Modelo:

Tipo:

Ciclos Esperados:

Inserir Componente

Exportar Biblioteca de Componentes JSON

Biblioteca de Componentes

ID	Maker	Model	Type	ExpectedCycles
1	SMC	CD8SF10-80-B	Cilindro	5000000
2	SMC	MGPM25-100Z	Cilindro	5000000
3	SMC	DF95C05-40-C	Cilindro	100

Figura 21 – Biblioteca de componentes da instância RiTS

A biblioteca de ações foi configurada com doze entradas, cada uma correspondendo a um passo de sequência do NI TestStand que ativa um atuador físico da máquina: *Haptic_1*, *Haptic_2*, *Konica*, quatro posições de *MainConnector* (*Nest1* a *Nest4*) e quatro posições de *LockUUT* (*Nest1* a *Nest4*).

Controlar

innovating industry

Gestão de Manutenção

Controlar

RITS ControlarS2.8

Controlar

Mudar de usuário

Componentes Instalados

Histórico de Ações

Biblioteca de Componentes

Biblioteca de Ações

Alertas 2

Log

Configurações

Gestão de Utilizadores

Utilizador: Controlar

Adicionar Nova Ação

Nome:

Ciclo:

Inserir Ação

Exportar Biblioteca de Ações JSON

Biblioteca de Ações

ID	Name	BurntCycles
1	Haptic_1	1
2	Haptic_2	1
3	Konica	1
4	MainConnector_Nest1	1
5	MainConnector_Nest2	1
6	MainConnector_Nest3	1
7	MainConnector_Nest4	1
8	LockUUT_Nest1	1

Figura 22 – Biblioteca de ações da instância RiTS

Cada ação foi configurada com *BurntCycles* igual a 1, refletindo que cada execução do passo consome um ciclo de utilização do componente instalado nessa posição, como se pode ver na Figura 22.

5.1.3. OPERAÇÃO DE INTEGRAÇÃO TESTSTAND

Com a instância configurada e os doze componentes instalados nas respectivas ações, a máquina foi colocada em operação com a sequência de teste do NI TestStand ativa. A biblioteca .NET *ControlarMaintenanceClient*, integrada como driver nos passos de sequência correspondentes, passou a invocar automaticamente o *endpoint IncrementWorkingCycles* a cada execução, sem qualquer intervenção manual na interface de manutenção. Na Figura 23 podemos ver o step selecionado, que demonstra a chamada da DLL (*Dynamic-Link Library*) para incrementar os *working cycles*

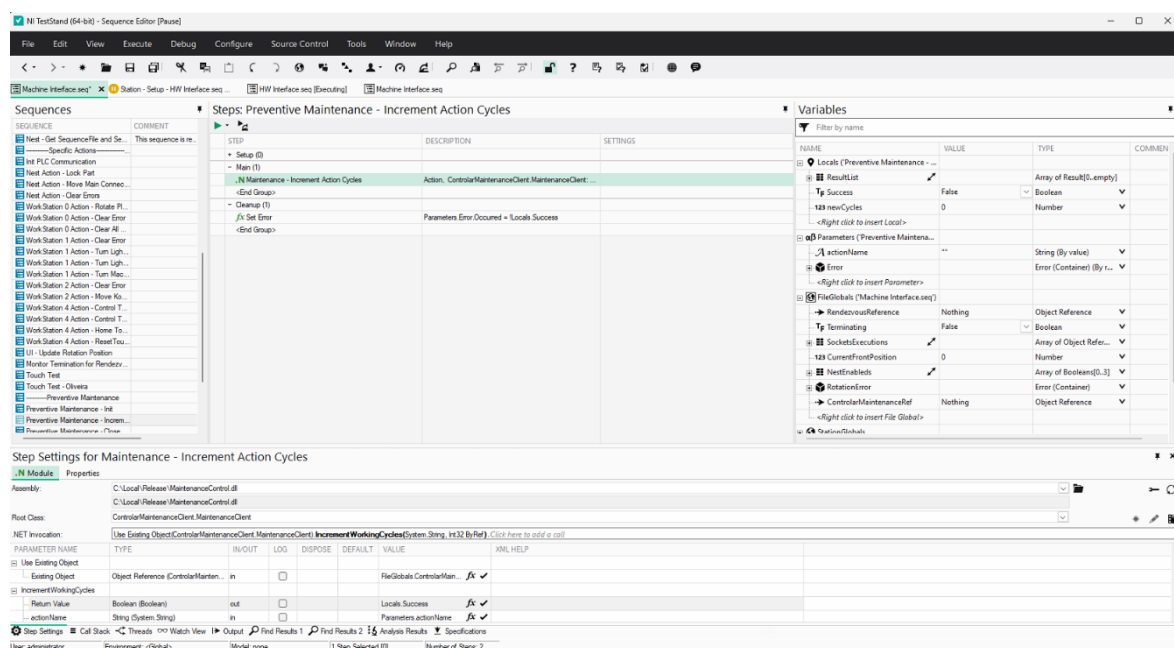


Figura 23 – Chamada da DLL *ControlarMaintenanceClient* com a função *IncrementWorkingCycles*

O histórico de ações registra cada incremento individualmente, com *timestamp* preciso e associação ao componente e ação correspondentes. A Figura 24 mostra uma secção do histórico com centenas de entradas consecutivas de incremento automático para a ação *Konica_2*, geradas pelo NI TestStand durante a operação da máquina. A

cadência dos registos, múltiplas entradas por segundo em determinados momentos, reflete o ritmo de execução da sequência de teste.



The screenshot shows the 'Gestão de Manutenção' application interface. The top header includes the 'Controlar' logo and the title 'Gestão de Manutenção'. A sidebar on the left contains navigation links: 'Componentes Instalados', 'Histórico de Ações', 'Biblioteca de Componentes', 'Biblioteca de Ações', 'Alertas', 'Log', 'Configurações', and 'Gestão de Utilizadores'. The main content area displays the 'Histórico de Ações' table, which lists actions with columns for ID, TimeStamp, Descrição, Component, and Action. Above the table is a button to 'Exportar Histórico de Ações'.

ID	TimeStamp	Descrição	Component	Action
1177	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1178	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1179	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1180	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1181	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1182	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1183	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1184	Tue, 30 Jun 2026 19:03:01 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1147	Tue, 30 Jun 2026 19:03:00 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1148	Tue, 30 Jun 2026 19:03:00 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)
1149	Tue, 30 Jun 2026 19:03:00 GMT	Incremento de 1 ciclos na ação Konica_2	Cilindro, SMC, DF95C05-40-C (3)	Konica_2 (12)

Figura 24 – Histórico de incrementos automáticos gerados pelo NI TestStand

A tabela de componentes instalados, apresentada na Figura 25, mostra os contadores acumulados após o período de operação real. Os cilindros de háptica (*Haptic_1* e *Haptic_2*) registaram 275 e 266 ciclos respetivamente, valores próximos e coerentes com o facto de os dois atuadores operarem em paralelo na mesma fase do ciclo de teste. Os conectores principais (*MainConnector_Nest1* a *Nest4*) acumularam entre 21 e 36 ciclos, e os cilindros de lock (*LockUUT_Nest1* a *Nest4*) entre 23 e 29 ciclos, variações explicadas pela cadência diferente de cada posição na sequência rotativa. Estes valores representam dados reais de operação industrial, produzidos automaticamente pela integração com o NI TestStand sem intervenção manual.



Figura 25 – Componentes instalados com ciclos reais acumulados (perfil observador)

A Figura 25 foi capturada com sessão iniciada no perfil de observador, confirmando o comportamento do controlo de acesso em ambiente real: a tabela apresenta os mesmos dados, mas sem os botões de ação (Desativar, Remover, Substituir), que são exclusivos dos perfis de operador e administrador.

5.1.4. SISTEMA DE ALERTAS

Para verificar o comportamento completo do sistema de alertas sem aguardar os volumes de operação necessários para atingir os limiares nos componentes com 5 000 000 ciclos esperados, o componente SMC DF95C05-40-C foi configurado com *expected_cycles* igual a 100 e instalado em três ações distintas (Konica, Konica_2 e Konica_3), com os contadores incrementados manualmente para valores correspondentes aos três patamares de severidade. Esta abordagem, equivalente à utilização de dados de teste controlados em fase de validação, permitiu exercitar os três níveis de alerta de forma deliberada e verificável.

Com um componente a 80% de utilização (80/100 ciclos), o sistema gerou um alerta baixo; a 90% gerou um alerta médio; a 95% gerou um alerta crítico. A Figura 26 mostra os três alertas ativos em simultâneo, com o gráfico de distribuição por severidade e a codificação cromática correta, vermelho para crítico, laranja para médio, amarelo para

baixo. Os alertas de severidade baixa e média apresentam o botão "Marcar como lido", enquanto o crítico permanece sem essa opção, mantendo-se visível até que o componente seja substituído ou desativado.



Figura 26 – Três níveis de alerta ativos em simultâneo com gráfico de distribuição

A Figura 27 mostra a mesma secção de alertas numa sessão com perfil de observador. Os alertas são visíveis, mas sem qualquer botão de interação, confirmando que a diferenciação de capacidades por perfil se aplica também à gestão de alertas.



Figura 27 – Vista de alertas no perfil de observador (sem opção de descarte)

A tabela de componentes instalados, que podemos observar na Figura 28, reflete os mesmos dados com codificação cromática por linha: vermelho para o componente a 95%, laranja para o de 90% e amarelo para o de 80%.



Figura 28 – Tabela de componentes com codificação cromática por nível de utilização

Permitindo uma leitura imediata do estado de desgaste sem necessidade de navegar para a secção de alertas.

5.1.5. RASTREABILIDADE E EXPORTAÇÃO

Todas as operações realizadas durante a configuração e operação da instância ficaram registadas automaticamente no *log* de auditoria. A Figura 29 mostra uma secção do histórico de alterações, onde são visíveis operações de *INSERT* de componentes e ações, *REPLACE* de um componente substituído, e sequências de *Activate/Deactivate*, cada uma com o utilizador responsável, o *timestamp* preciso e a descrição da operação. O *log* confirma que a cadeia de rastreabilidade funciona desde a primeira operação de configuração até às interações durante a operação.

ID	Utilizador	Descrição	Data	
45	controlar	INSERT	Installed_Components	Instalou componente ID 3 com ação ID 12. 30-06-2026 19:02:38
44	controlar	INSERT	Actions	Inseriu ação Konica_2 com 1 ciclos. 30-06-2026 19:02:04
43	controlar	INSERT	Installed_Components	Instalou componente ID 3 com ação ID 3. 30-06-2026 18:59:58
42	controlar	Deactivate	Installed_Components	Desativou componente 1 pertencente à ação 3. 30-06-2026 18:59:28
41	controlar	REMOVE	Installed_Components	Removeu componente 1 da ação 3. 30-06-2026 18:59:25
40	controlar	Activate	Installed_Components	Activou componente 1 pertencente à ação 3. 30-06-2026 18:59:23
39	controlar	Deactivate	Installed_Components	Desativou componente 1 pertencente à ação 3. 30-06-2026 18:59:20
38	controlar	INSERT	Installed_Components	Instalou componente ID 1 com ação ID 3. 30-06-2026 18:58:04
37	controlar	INSERT	Actions	Inseriu ação Konica com 1 ciclos. 30-06-2026 18:57:58
36	controlar	INSERT	Components	Inseriu componente SMC-DF95C05-40-C do tipo Cilindro com 100 ciclos esperados. 30-06-2026 18:57:01
35	controlar	Activate	Installed_Components	Activou componente 2 pertencente à ação 8. 30-06-2026 18:54:27
34	controlar	Deactivate	Installed_Components	Desativou componente 2 pertencente à ação 8. 30-06-2026 18:54:23
33	controlar	REPLACE	Installed_Components	Substituiu componente 2 na ação 7. 30-06-2026 18:54:14

Figura 29 – Log de auditoria com operações realizadas na instância RiTS

No encerramento de sessão, o sistema apresenta um *popup* de confirmação que questiona se o utilizador pretende exportar o estado atual da máquina antes de sair.

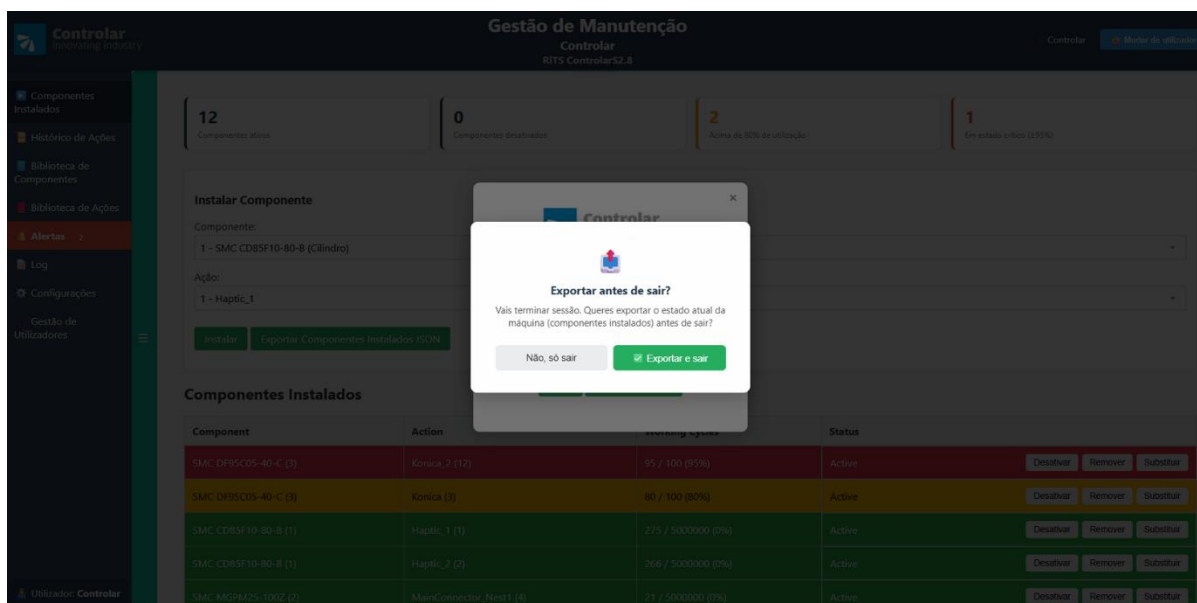


Figura 30 – *Popup* de confirmação de exportação antes do encerramento da sessão

Esta salvaguarda, ilustrada na Figura 30, previne que dados por sincronizar sejam perdidos por um encerramento inadvertido, garantindo que cada sessão de trabalho termina com a oportunidade de atualizar o repositório central.

5.2. SISTEMA CENTRAL DE MANUTENÇÃO

Em paralelo com a instalação do sistema local na RiTS, o sistema central foi implantado num servidor dedicado da Controlar e utilizado para agregar dados de múltiplas máquinas. Esta secção documenta os resultados observados na plataforma central durante o mesmo período.

5.2.1. AUTENTICAÇÃO E ACESSO

O acesso ao sistema central é feito através de uma página de autenticação, que podemos ver na Figura 31, independente do sistema local, servida pelo mesmo servidor Flask mas sob um prefixo de rota dedicado.



Figura 31 – Página de autenticação do sistema central

As credenciais são verificadas contra a tabela de utilizadores do sistema central, com palavra-passe armazenada por *hash pbkdf2:sha256*, de forma análoga ao sistema local.

5.2.2. DASHBOARD E ATIVIDADE RECENTE

O *dashboard* é o ecrã inicial após autenticação e agrega o estado global do repositório central em quatro indicadores: máquinas registadas, *uploads* recebidos, *uploads* por processar e componentes com estatísticas calculadas. No momento da captura, o sistema registava três máquinas distintas, cinco *uploads* todos processados e seis tipos de componentes com estatísticas disponíveis.

A secção de atividade recente lista os *uploads* mais recentes com o código da máquina de origem, o nome do ficheiro, a data de receção e o estado de processamento. São visíveis dois *uploads* reais da RiTS ControlarS2.8, o ficheiro “Controlar_ControlarS2.8_RiTS_Componentes_Instalados.json” exportado diretamente pelo sistema local, confirmando que o pipeline de exportação e ingestão funcionou em ambiente real.

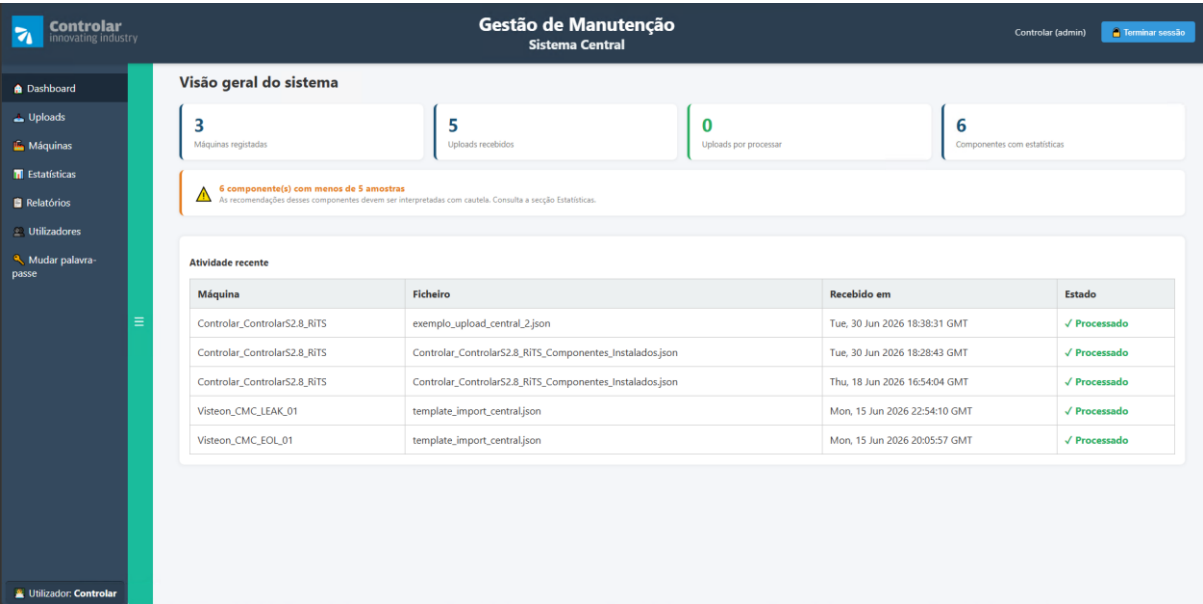


Figura 32 – *Dashboard* do sistema central com três máquinas registadas e atividade recente

O aviso de "6 componentes com menos de 5 amostras" reflete o estado inicial de um repositório em crescimento e demonstra que o mecanismo de alerta de fiabilidade estatística está operacional, como podemos ver na Figura 32.

5.2.3. INGESTÃO DE DADOS

A secção de *uploads* disponibiliza o formulário de importação e o histórico completo de *uploads* recebidos, como podemos ver na Figura 33. O histórico confirma a sequência cronológica das importações: dois *uploads* da máquina Visteon_CMC_EOL_01 e Visteon_CMC_LEAK_01 em Junho de 2026, seguidos de três *uploads* da RiTS em datas distintas, o que demonstra a capacidade do sistema de agregar dados de múltiplas origens e máquinas ao longo do tempo.

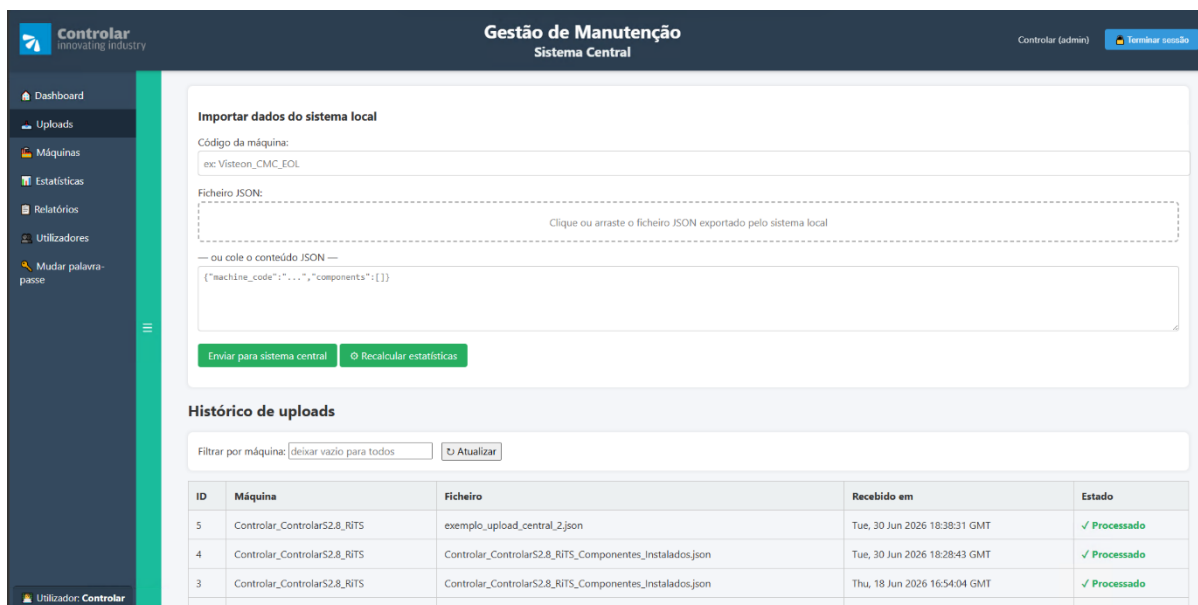


Figura 33 – Histórico de *uploads* com dados de três máquinas distintas

Todos os *uploads* aparecem com estado "Processado", confirmando que a ingestão e extração de componentes para as tabelas analíticas foi concluída com sucesso em todos os casos.

5.2.4. MÁQUINAS E HISTÓRICO DE COMPONENTES

A secção de Máquinas, visível na Figura 34, lista as três instalações registadas com o respetivo código identificador, número de *uploads* e data do último *upload*. A expansão da máquina "Controlar_ControlarS2.8_RITS" revela o historial de componentes agregados pelos *uploads* recebidos: são visíveis registos com status ativo, correspondentes a componentes atualmente em serviço e registos com status substituído, representando ciclos de vida concluídos.



Figura 34 – Máquinas registadas e componentes da instância RiTS com ciclos de vida concluídos. Os componentes substituídos, com utilizações acima de 100%, como o SMC MXS12-30 a 106% e a Ingun KS23 a 104.2%, são os que contribuem para o cálculo estatístico de vida útil real, por representarem ciclos completos com valor observacional concreto.

5.2.5. ESTATÍSTICAS DE VIDA ÚTIL

A secção de estatísticas apresenta os resultados do cálculo de vida útil por tipo de componente, executado sobre o conjunto de observações acumuladas de todas as instalações. Com 16 amostras totais distribuídas por seis tipos de componentes, CMC Connect-6, Festo DSNU-20, Ingun KS23, Parker OR-150, SKF 6204-2RS e SMC MXS12-30, o sistema calculou os valores de média, percentil 50 e percentil 90 para cada família.

O gráfico de barras agrupadas, que podemos ver na Figura 35, mostra a dispersão entre as três métricas por componente: o Festo DSNU-20 apresenta o P90 mais elevado (30 500 ciclos), sendo identificado pelo sistema como o componente mais fiável do conjunto. O gráfico de barras horizontais à direita representa o número de amostras por componente, com todos os seis tipos a apresentar menos de cinco amostras, situação esperada numa fase inicial de operação, assinalada pelo sistema com o aviso de cautela no *dashboard* e com codificação a laranja nas barras.

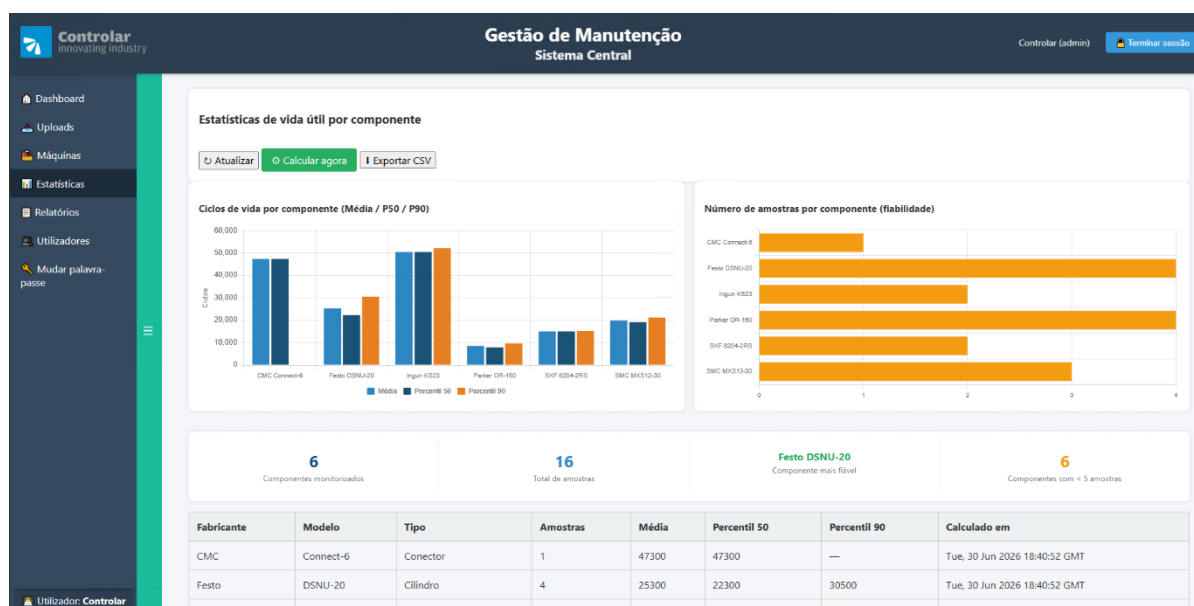


Figura 35 – Estatísticas de vida útil por componente com gráficos de média / P50 / P90 e volume de amostras

À medida que mais instalações exportarem dados e mais substituições forem registadas, a base estatística crescerá e a fiabilidade das estimativas aumentará progressivamente.

5.2.6. RELATÓRIO DE RECOMENDAÇÕES

O relatório de recomendações, presente na Figura 36, constitui a saída analítica de maior valor prático do sistema central. Filtrado por um mínimo de três amostras e ordenado pelo percentil 90 em critério conservador, o relatório identificou três tipos de componentes com dados suficientes para emitir recomendações fundamentadas: o Parker OR-150 (vedante, 4 amostras), o SMC MXS12-30 (guia linear, 3 amostras) e o Festo DSNU-20 (cilindro, 4 amostras).

As recomendações geradas são concretas e acionáveis: "Substituir após 9 600 ciclos" para o Parker OR-150, "Substituir após 21 200 ciclos" para o SMC MXS12-30, e "Substituir após 30 500 ciclos" para o Festo DSNU-20. Estes valores derivam do percentil 90 das observações reais disponíveis, o que significa que 90% das amostras registadas foram substituídas antes de atingir esse número de ciclos, uma estimativa conservadora e diretamente defensável pelos dados que a sustentam.

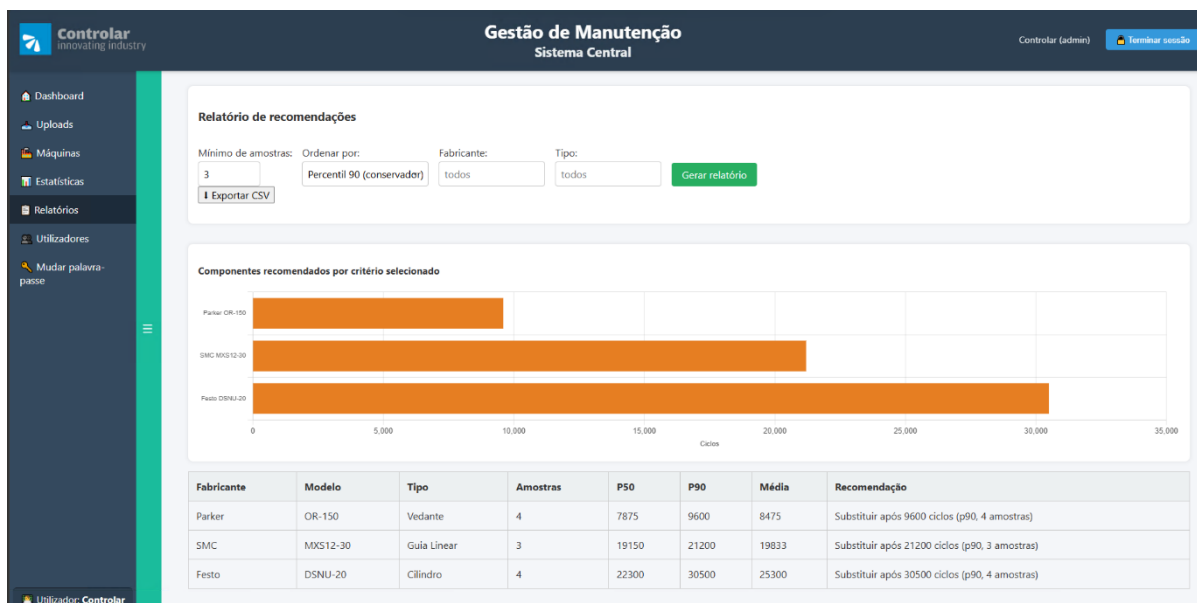


Figura 36 – Relatório de recomendações com intervalos de substituição baseados no percentil 90

O gráfico de barras horizontais permite comparar visualmente os intervalos de substituição entre componentes, complementando a tabela com uma leitura imediata da hierarquia de durabilidade relativa.

5.2.7. GESTÃO DE UTILIZADORES

A gestão de utilizadores do sistema central suporta três perfis distintos, administrador, analista e observador, com responsabilidades diferenciadas. No momento da captura da Figura 37, quatro utilizadores estavam registados: dois com perfil de administrador, um analista e um observador. A diferenciação de perfis no sistema central segue a mesma lógica do sistema local: os administradores têm acesso à configuração e gestão de utilizadores, os analistas podem importar dados e calcular estatísticas, e os observadores têm acesso de leitura às secções analíticas sem capacidade de modificação.

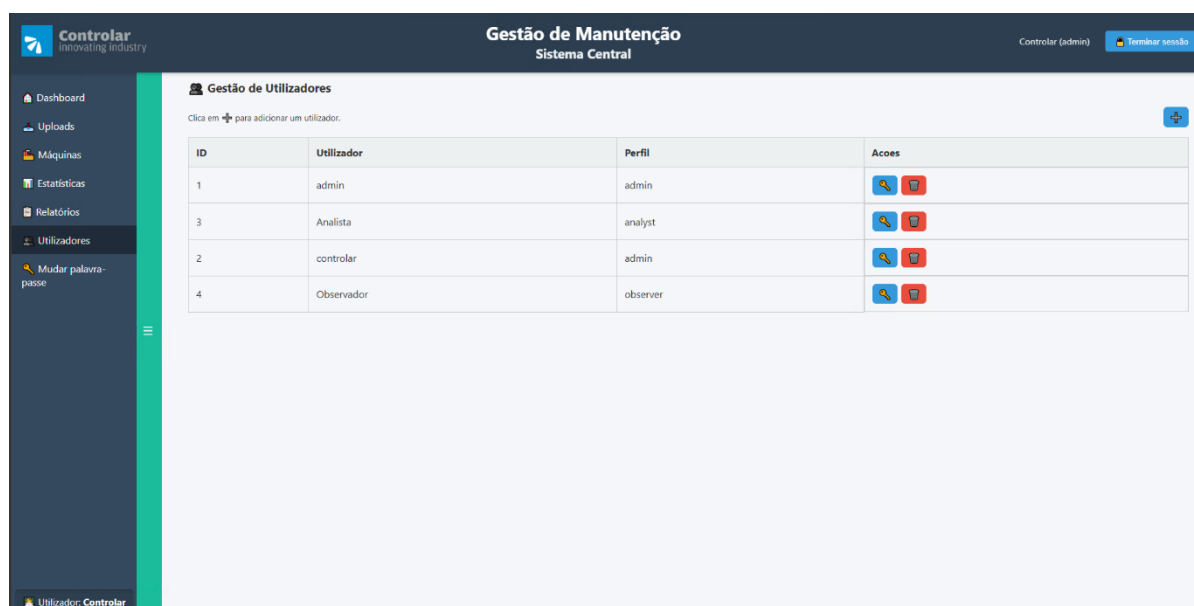


Figura 37 – Gestão de utilizadores do sistema central com três perfis diferenciados

A possibilidade de alterar a palavra-passe está disponível para todos os perfis através da secção dedicada, que exige a confirmação da credencial atual antes de aceitar uma nova, mecanismo de segurança que impede alterações não autorizadas em sessões desacompanhadas, de forma consistente com o comportamento implementado no sistema local.

6. CONCLUSÕES E TRABALHO FUTURO

O presente capítulo encerra o documento com uma reflexão sobre o trabalho desenvolvido, avaliando em que medida os objetivos inicialmente propostos foram alcançados e qual o contributo efetivo da solução para o problema identificado. São apresentadas as conclusões do trabalho, as limitações reconhecidas que condicionam o seu âmbito e aplicabilidade atual e, as direções de evolução identificadas para trabalho futuro, que representam tanto as continuações naturais do que foi implementado como as melhorias necessárias para a adoção plena da solução em ambiente de produção industrial.

6.1. CONCLUSÕES

O trabalho desenvolvido produziu uma solução funcional de apoio à manutenção preventiva adaptada ao contexto operacional da Controlar, respondendo ao problema central identificado na fase de análise: a ausência de dados estruturados sobre o desgaste e a utilização dos componentes instalados nas máquinas, que impedia a antecipação de falhas e mantinha as equipas de manutenção presas a um regime predominantemente reativo. Revisitando os objetivos definidos na secção 1.3, é possível verificar que cada uma das metas específicas traçadas no início do projeto encontra correspondência direta no resultado alcançado.

Relativamente à necessidade de criar uma forma de registo e acompanhamento do desgaste de componentes com base em ciclos de funcionamento reais, foi concretizado através do sistema local, que regista automaticamente cada ciclo executado através da integração com o NI TestStand, sem qualquer intervenção manual sobre os contadores. Este mecanismo substitui a estimativa subjetiva de desgaste por uma medição objetiva e contínua, condição prévia para qualquer decisão de manutenção fundamentada em evidência.

O suporte ativo a uma abordagem preventiva através de alertas automáticos, foi igualmente atingido: o sistema classifica os componentes em três níveis de severidade com base em limiares de utilização, tornando visível, em qualquer secção da interface, o estado de desgaste que anteriormente só seria perceptível após a ocorrência de uma avaria. Esta antecipação é, no fundo, o cerne da transição de um modelo reativo para um modelo preventivo.

A capacidade de garantir rastreabilidade completa do processo de manutenção, foi assegurado pelo log de auditoria associado a cada utilizador autenticado e pelo controlo de acessos diferenciado por perfil, que em conjunto tornam qualquer operação sensível reconstituível e atribuível a um responsável. Em ambientes industriais onde a responsabilização e a auditoria são requisitos de qualidade, esta capacidade tem valor que ultrapassa a mera conveniência técnica.

Em relação à consolidação de dados de múltiplas instalações numa plataforma central, foi alcançado com o sistema central, que ingere, normaliza e agrega os dados exportados pelas instâncias locais, permitindo pela primeira vez à Controlar comparar o comportamento de componentes equivalentes em diferentes máquinas e clientes. Permite uma capacidade de análise transversal que a empresa não possuía anteriormente.

Por ultimo, ser capaz de disponibilizar dados reais sobre a durabilidade dos componentes para apoiar decisões de engenharia, materializou-se no relatório de recomendações baseado no Percentil 90, que oferece às equipas de projeto uma alternativa fundamentada em observações reais às especificações genéricas dos fabricantes, com potencial de reduzir custos de ciclo de vida em decisões futuras de seleção e dimensionamento de componentes.

Em termos mais amplos, o trabalho demonstra que é possível concretizar a transição de um modelo de manutenção reativo para um modelo orientado por dados sem recorrer a soluções CMMS comerciais de âmbito geral, respondendo com uma arquitetura à medida às restrições concretas do ambiente industrial da Controlar, neste caso o isolamento de rede, integração com sistemas de automação existentes e necessidade de funcionalidades específicas não cobertas por plataformas genéricas. A separação em dois módulos com responsabilidades distintas e sincronização assíncrona revelou-se uma decisão arquitetural sólida, que garante autonomia operacional no ponto de utilização e permite que o valor analítico do sistema cresça de forma sustentada à medida que o volume de dados acumulado aumenta. Mais do que uma resposta pontual a uma limitação técnica, a solução desenvolvida estabelece a infraestrutura de dados sobre a qual a Controlar pode continuar a evoluir, progressivamente, rumo a uma manutenção cada vez mais preditiva e baseada em evidência.

6.2. LIMITAÇÕES

Como em qualquer solução na sua primeira fase de implementação em ambiente real, há aspetos que naturalmente ficam para uma próxima etapa de maturação, mais do que limitações no sentido estrito do termo.

No que respeita à validação em produção, o sistema local foi instalado e operou na máquina RiTS ControlarS2.8 durante um período ainda relativamente curto face à vida útil esperada dos componentes de maior duração, pelo que o comportamento do sistema de alertas foi essencialmente demonstrado com um componente configurado com um limiar reduzido, à semelhança de um cenário de teste controlado. Esta opção permitiu validar de forma clara e verificável o funcionamento dos três níveis de severidade, ainda que a observação de alertas gerados por desgaste real acumulado ao longo de ciclos de vida completos fique reservada para um período de operação mais alargado. De forma semelhante, a validação centrou-se numa única instalação, pelo que o comportamento do sistema sob utilização simultânea por múltiplos utilizadores e com o volume de dados correspondente a uma frota mais alargada de máquinas constitui uma extensão natural e desejável do trabalho.

Ao nível do sistema central, a versão atual assenta em estatísticas descritivas (média, percentil 50 e percentil 90) que já representam um avanço significativo face à ausência total de dados que caracterizava a situação de partida, mas que podem ser complementadas, numa fase seguinte, por modelos matemáticos de fiabilidade capazes de captar a estrutura probabilística da vida útil de cada componente.

Importa ainda notar que a robustez das métricas produzidas pelo sistema central acompanha, como seria expectável em qualquer sistema de análise baseado em observações reais, o volume de dados acumulado: com um número ainda reduzido de amostras por tipo de componente, as estimativas devem ser interpretadas com a cautela que o próprio sistema já assinala através do aviso de amostragem insuficiente. Esta característica não constitui uma fragilidade da solução, mas antes uma propriedade inerente a este tipo de abordagem, que tende a ganhar robustez de forma natural à medida

que mais instalações contribuem com dados ao longo do tempo. Sendo, aliás, essa a lógica de crescimento que a arquitetura do sistema central foi especificamente concebida para suportar.

6.3. TRABALHO FUTURO

A direção de evolução mais significativa identificada para o sistema central é a incorporação de modelos matemáticos de análise de fiabilidade que permitam transcender as estatísticas descritivas atualmente disponíveis. A aplicação de distribuições paramétricas de tempo de falha, como a distribuição de amplamente utilizada em engenharia de fiabilidade industrial, permitiria modelar a estrutura probabilística da vida útil de cada tipo de componente e derivar estimativas de probabilidade de falha em função dos ciclos acumulados com base nos dados históricos recolhidos. Esta capacidade tornaria possível calcular intervalos de substituição otimizados em função de critérios de risco configuráveis por tipo de componente e contexto operacional, evoluindo de uma abordagem de recomendação baseada em percentis observados para uma abordagem de gestão do risco fundamentada em modelos de fiabilidade. Complementarmente, a incorporação de modelos de regressão que relacionem características do contexto operacional com a vida útil observada permitiria refinar as recomendações para além do tipo de componente, tornando-as sensíveis ao perfil de carga e às condições de utilização de cada instalação específica.

A extensão da validação em contexto de produção real a um conjunto mais alargado de máquinas e utilizadores constitui o passo seguinte determinante para a adoção efetiva da solução. Esta extensão implicaria a realização de testes de aceitação formais com as equipas de manutenção e engenharia da Controlar em condições reais de operação continuada, a verificação do comportamento do sistema com dados operacionais provenientes de múltiplas máquinas durante um período de tempo representativo e, a avaliação do desempenho sob condições de utilização concorrente. Os resultados desta validação permitiriam identificar ajustes de usabilidade, lacunas de funcionalidade não

antecipadas na fase de desenvolvimento e requisitos de desempenho que não são perceptíveis numa instalação única com um período de operação reduzido.

Por fim, a evolução da solução a longo prazo poderá incluir o desenvolvimento de uma interface adaptada a dispositivos móveis orientada aos técnicos de manutenção em contexto de campo, a integração com sistemas de gestão da qualidade e de rastreabilidade já existentes nas instalações dos clientes da Controlar e, a exploração de mecanismos de alerta proactivo ao nível da frota que, com base nos padrões estatísticos acumulados no sistema central, sinalizem componentes em risco de falha antes de atingirem os limiares de alerta definidos localmente. Estas extensões posicionariam a solução numa trajetória de evolução alinhada com os princípios da Indústria 4.0, em que a manutenção preventiva baseada em dados históricos evolui progressivamente para uma abordagem preditiva baseada em modelos e na análise contínua do estado da frota instalada.

Referências documentais

- [1] **CEN—EN 13306:** Maintenance—Maintenance terminology. European Committee for Standardization, 2010.
- [2] **IEC—IEC 60300-3-14:** Dependability management—Part 3-14: Application guide—Maintenance and maintenance support. International Electrotechnical Commission, 2004.
- [3] **ISO—ISO 14224:** Petroleum, petrochemical and natural gas industries—Collection and exchange of reliability and maintenance data for equipment. International Organization for Standardization, 2016.
- [4] **ISO—ISO 17359:** Condition monitoring and diagnostics of machines—General guidelines. International Organization for Standardization, 2018.
- [5] **JARDINE, A. K. S.; LIN, D.; BANJEVIC, D.**—A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, Vol. 20, n. º 7, 2006, pp. 1483–1510.
- [6] **MOUBRAY, John** — Reliability-Centred Maintenance. 2.ª edição. Industrial Press, 1997. ISBN 0-8311-3146-2.
- [7] **NAKAJIMA, Seiichi** — Introduction to TPM: Total Productive Maintenance. Productivity Press, 1988. ISBN 0-915299-23-2.
- [8] **MOBLEY, R. Keith**—Maintenance Fundamentals. Butterworth-Heinemann, 2004. ISBN 978-0750677981.
- [9] **GULATI, Ramesh**—Maintenance and Reliability Best Practices. Second Edition. Industrial Press, 2012. ISBN 9780831134341.
- [10] **NOWLAN, F. Stanley; HEAP, Howard F.**—Reliability-Centered Maintenance. Report (United Airlines), 1978.

- [11] **IEC—IEC 60300-3-11:** Dependability management—Part 3-11: Application guide—Reliability centred maintenance. International Electrotechnical Commission, 2009.
- [12] **SAE—SAE JA1011:** Evaluation Criteria for Reliability-Centered Maintenance (RCM) Processes. SAE International, 1999 (revisões posteriores).
- [13] **ISO—ISO 55000:** Asset management—Overview, principles and terminology. International Organization for Standardization, 2024.
- [14] **SHANKAR, Lakshmi; SINGH, Chandan Deep; SINGH, Ranjit** — Impact of implementation of CMMS for enhancing the performance of manufacturing industries. International Journal of System Assurance Engineering and Management, Vol. 14, 2023, pp. 1599–1620. DOI: 10.1007/s13198-021-01480-6.
- [15] **VAN ROSSUM, Guido; DRAKE, Fred L.** — Python 3 Reference Manual. Scotts Valley, CA: CreateSpace, 2009.
- [16] **TIOBE SOFTWARE BV** — TIOBE Index. TIOBE Software, 2025. [Online]. Disponível em: <https://www.tiobe.com/tiobe-index/>
- [17] **NATIONAL INSTRUMENTS** — NI TestStand Documentation. National Instruments Corporation, 2023. [Online]. Disponível em: <https://www.ni.com/documentation/en/teststand/>
- [18] **GRINBERG, Miguel** — Flask Web Development: Developing Web Applications with Python. 2.^a edição. Sebastopol, CA: O'Reilly Media, 2018. ISBN 9781491991732.
- [19] **EBY, Philip J.** — PEP 3333: Python Web Server Gateway Interface v1.0.1. Python Software Foundation, 2010. [Online]. Disponível em: <https://peps.python.org/pep-3333/>
- [20] **ORACLE CORPORATION** — MySQL 8.0 Reference Manual. Oracle Corporation, 2024. [Online]. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/>

- [21] **CODD, Edgar F.** — A relational model of data for large shared data banks. Communications of the ACM, Vol. 13, n.º 6, 1970, pp. 377–387. DOI: 10.1145/362384.362685.
- [22] **SCHWARTZ, Baron; ZAITSEV, Peter; TKACHENKO, Vadim** — High Performance MySQL. 3.ª edição. Sebastopol, CA: O'Reilly Media, 2012. ISBN 9781449314293.
- [23] **DB-ENGINES** — DB-Engines Ranking of Relational DBMS. Solid IT GmbH, 2025. [Online]. Disponível em: <https://db-engines.com/en/ranking/relational+dbms>
- [24] **WHATWG** — HTML Living Standard. Web Hypertext Application Technology Working Group, 2024. [Online]. Disponível em: <https://html.spec.whatwg.org/>
- [25] **W3C** — CSS Flexible Box Layout Module Level 1. W3C Candidate Recommendation. World Wide Web Consortium, 2018. [Online]. Disponível em: <https://www.w3.org/TR/css-flexbox-1/>
- [26] **ECMA INTERNATIONAL** — ECMAScript 2015 Language Specification (ECMA-262, 6th Edition). ECMA International, 2015. [Online]. Disponível em: <https://www.ecma-international.org/ecma-262/6.0/>
- [27] **FLANAGAN, David** — JavaScript: The Definitive Guide. 7.ª edição. Sebastopol, CA: O'Reilly Media, 2020. ISBN 9781491952023.
- [28] **CROCKFORD, Douglas** — RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON). IETF, 2006. [Online]. Disponível em: <https://www.rfc-editor.org/rfc/rfc4627>
- [29] **BRAY, Tim (Ed.)** — RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format. IETF, 2014. [Online]. Disponível em: <https://www.rfc-editor.org/rfc/rfc7159>
- [30] **ECMA INTERNATIONAL** — ECMA-404: The JSON Data Interchange Standard. 2.ª edição. ECMA International, 2017. [Online]. Disponível em: <https://www.ecma-international.org/publications-and-standards/standards/ecma-404/>

- [31] **FIELDING, Roy Thomas** — Architectural Styles and the Design of Network-based Software Architectures. Tese de Doutorado, Department of Information and Computer Science, University of California, Irvine, 2000. [Online]. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [32] **SOMMERVILLE, Ian** — Software Engineering. 10.ª edição. Harlow: sPearson Education, 2016. ISBN 9781292096131.