Universidade Federal de Minas Gerais
Escola de Engenharia
Curso de Graduação em Engenharia de Controle e Automação

Sistema para Controle de Paradas de Equipamentos de Ferramentaria do Setor Automotivo

Guilherme Menezes Magalhães

Orientador: Prof. Víctor Costa da Silva Campos, Dr.
Supervisor: Eng. Felipe Augusto Vitoriano

Belo Horizonte, Julho de 2022

Monografia

Sistema para Controle de Paradas de Equipamentos de Ferramentaria do Setor Automotivo

Monografia submetida à banca examinadora designada pelo Colegiado Didático do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Minas Gerais, como parte dos requisitos para aprovação na atividade Projeto Final de Curso II.

Belo Horizonte, Julho de 2022

Resumo

No setor industrial de ferramentarias, em diversas ocasiões, os *softwares* de gestão apresentam informações em excesso, ou não possuem soluções para problemas específicos da empresa contratante, fatores que comprometem a eficiência do processo. Esta realidade costuma ser bastante comum neste tipo de indústria, e, por isso, para auxiliar uma ferramentaria automobilística, foi desenvolvido uma plataforma *web* que integra vários módulos para cálculo, armazenamento e visualização de dados relacionados a eventos de máquinas de usinagem e seu desempenho. Para isso, dados presentes no banco de dados do cliente são processados e inseridos na plataforma desenvolvida, permitindo encontrar os principais gargalos do processo por meio do histórico de eventos e de um Diagrama de Pareto. Ao final, os indicadores de desempenho eficiência global (OEE), tempo médio para reparos (MTTR), tempo médio entre falhas (MTBF) e disponibilidade inerente são calculados e dispostos de forma gráfica. Para o desenvolvimento desse sistema, uma arquitetura de *software* foi planejada e implementada, de forma que o sistema final fosse de simples entendimento e desenvolvimento, além de ser o mais escalável possível. Ao final do projeto, todos os módulos foram testados e validados na empresa cliente, e a arquitetura planejada foi seguida por completo.

Palavras-chave: software; arquitetura de *software*; indicadores de desempenho; sistema; diagrama de Pareto; banco de dados.

Abstract

In the industrial tooling sector, on several occasions, management software presents an excess of information, or does not have solutions for specific problems of the contracting company, factors that compromise the efficiency of the process. This reality is usually quite common in this type of industry, and therefore, to help an automotive tool shop, a web platform was developed that integrates several modules for calculation, storage and visualization of data related to machining machine events and their performance. To achieve this, data available in the client's database are processed and inserted into the developed platform, allowing the main bottlenecks of the process to be found through the history of events and a Pareto Diagram. At the end, the performance indicators overall equipment efficiency (OEE), mean time to repair (MTTR), mean time between failures (MTBF) and inherent disponibility are calculated and displayed graphically. To development this system, a software architecture was planned and implemented, so that the final system was simple to understand and develop, in addition to being as scalable as possible. At the end of the project, all modules were tested and validated at the client company, and the planned architecture was completely followed.

Keywords: software; software architecture; key performance indicators; system; Pareto diagram; database.

Agradecimentos

Primeiramente, agradeço a meus pais, Gilda e Claudionor, por todo o suporte que me deram durante toda a minha vida, permitindo que eu chegasse a este ponto. Agradeço também à minha namorada Thais, que sempre me deu todo o suporte possível e que me propiciou muita felicidade durante os últimos 5 anos. Por fim, a meu amigo Lucas, que compartilha bons momentos comigo há quase uma década, e aos meus amigos Laura, Michael, Gustavo e Bruno, que fizeram parte de toda a minha trajetória acadêmica, sempre apoiando um ao outro.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este projeto objetivou a automação de um processo rotineiro presente em uma ferramentaria do setor automotivo: a supervisão e análise de paradas de equipamentos. Para isso, foi desenvolvida uma plataforma *web* que fornece tabelas contendo as informações sobre as paradas, como horário, motivo e a operação que estava a ser realizada no momento. Além disso, também são fornecidos indicadores de desempenho para uma melhor gestão do processo por parte dos engenheiros, como eficácia geral do equipamento e indicadores de manutenção para condições de falhas.

A plataforma desenvolvida se comporta como um painel de controle para o uso de ativos, fornecendo todos os dados necessários para melhor acompanhamento e eficiência do processo. O projeto buscou, também, garantir o desenvolvimento técnico de forma mais objetiva e padronizada possível, justificando as decisões de implementação com base na literatura, de forma que seja simples a modificação e expansão futura do sistema, garantindo escalabilidade e clareza no entendimento do código.

1.1 Motivação e Justificativa

Com o fenômeno da globalização se tornando cada vez mais presente na sociedade contemporânea, mudanças sociais, econômicas, políticas e ideológicas têm ocorrido de forma rápida e drástica, em que "estamos vivendo uma nova etapa do desenvolvimento do capitalismo marcada pela multiplicação da produção de mercadorias complexas que exigem a utilização de elevada competência profissional nas áreas de gestão empresarial, de materiais e de tecnologias sofisticadas" (ALCOFORADO, 1997). Devido a isso, torna-se cada vez mais necessária a existência e uso de meios que tornem mais eficientes a gestão do processo, algo muito facilitado pelo uso de máquinas e *softwares*.

A sociedade sofreu diversas revoluções nos últimos séculos no âmbito tecnológico. Aproximadamente no ano de 1760, iniciou-se, na Inglaterra, o período conhecido na história como Primeira Revolução Industrial, demarcando o início do processo produtivo

mecânico, principalmente devido à construção de ferrovias e da invenção da máquina a vapor (SCHWAB, 2019). No final do século XIX, foi iniciada a Segunda Revolução Industrial, dando origem aos métodos de produção, como o Fordismo e Taylorismo, que buscavam formas de produzir ao máximo, com o menor custo e o menor tempo possível (RIBEIRO, 2015), bem como o surgimento de diversas novas tecnologias que ainda são utilizadas até os dias atuais, como telefone, motores elétricos e manipulação de ondas de rádio.

Em tempos mais recentes, na década de 1960, ocorreu a Terceira Revolução Industrial, conhecida, também, como revolução digital, marcada pela invenção de tecnologias que fazem o uso de semicondutores, como o silício, permitindo o surgimento de componentes eletrônicos, computadores e, por consequência, a internet (SCHWAB, 2019).

Por último, nos dias atuais, estamos vivendo a Quarta Revolução Industrial, conhecida como Indústria 4.0, caracterizada pela criação e uso de algoritmos de *software* complexos, que fazem o uso de inteligência artificial, e de sensores cada vez menores e que se comunicam entre si e com outros dispositivos por meio de redes de computadores, além de uma automação de processos cada vez mais intensa. Os campos da ciência de dados (*data science*) e o uso de computação em nuvem (*cloud computing*) crescem rapidamente, o que permite a invenção de tecnologias com poder de processamento maior e que auxiliam cada vez mais no cotidiano. Entretanto, o que caracteriza esta presente revolução industrial não é apenas essas tecnologias, mas sim "... a fusão dessas tecnologias e a interação entre os domínios físicos, digitais e biológicos" (SCHWAB, 2019).

Em diversas situações na indústria, os processos são feitos de forma manual, rotineira, ineficiente e dispendiosa, algo que gera demoras nos atendimentos, retrabalhos e uso excessivo de recursos. Esse tipo de acontecimento costuma ser um indício de carência de tecnologias que permitem otimização dos meios de produção e de gestão. Dessa forma, a partir da possibilidade de integração de tecnologias características da Indústria 4.0, torna-se possível a criação de métodos para maior controle de gestão do processo produtivo, permitindo o acompanhamento em tempo real de toda uma fábrica, condensando informações num único local de fácil acesso para os gestores, além de simplificar e agilizar processos manuais realizados por operadores, tornando a troca de informações entre diversos setores da indústria muito mais rápida e eficaz.

1.2 Objetivos do Projeto

A partir das ideias citadas anteriormente, o projeto teve como objetivo:

1. Criar uma plataforma *web* para o recebimento e integração dos dados de apontamento de paradas provenientes da ferramentaria;

2. Especificar uma arquitetura de *software* bem definida, de forma a simplificar futuras expansões do aplicativo, bem como facilitar o entendimento e desenvolvimento do código;

3. Desenvolver módulos de lógica para cálculo de indicadores de desempenho;

4. Desenvolver painéis de controle interativos;

5. Implementar sistema em processo produtivo.

1.3 Local de Realização

O projeto foi desenvolvido completamente em regime remoto, popularmente conhecido como *home office*, em nome da empresa Vitau Automation Sistemas de Automação LTDA.

A Vitau Automation é uma empresa que busca atuar na vanguarda da Tecnologia 4.0, fornecendo e desenvolvendo soluções e tecnologias para a otimização de processos. Sua equipe criou a Plataforma Vitau IoT, um sistema *online* inteligente para monitoramento, processamento e predição de falhas baseado em dados de ativos e processos industriais. Além disso, fornece, também, serviços de consultoria em automação industrial e desenvolvimento de sistemas industriais integrados de TI/TA, com destaque para sistemas de controle de produção integrados ao sistemas de chão de fábrica e às ferramentas de planejamento corporativo.

A empresa possui uma equipe de especialistas, engenheiros e mestres que atuam na especificação, projeto, desenvolvimento, implantação, comissionamento e manutenção dos sistemas que fornece e desenvolve. Seus projetos estão em execução em empresas pequenas e indústrias multinacionais, órgãos governamentais, com abrangência nacional e internacional (EUA, França e Alemanha).

É uma empresa acelerada pela ABDI - Agência Brasileira de Desenvolvimento Industrial (Ministério da Economia), InovAtiva - o maior programa de aceleração da América latina (Ministério da Economia), FiemgLab (FIEMG-MG), SEED (Governo de Minas Gerais) e Lemonade (FUNDEP - UFMG).

1.4 Estrutura da Monografia

Esta monografia está dividida em seis capítulos:

1. Introdução, apresentando uma visão geral sobre o trabalho, objetivos e a empresa proprietária do projeto;

2. Descrição do processo, em que haverá uma breve explicação de como é o processo presente na ferramentaria, e como funciona o registro de paradas atualmente;

3. Revisão bibliográfica sobre gestão de paradas e manutenção;
4. Metodologias utilizadas e revisão bibliográfica sobre arquitetura de *software*, além dos princípios e termos básicos necessários para entendimento do projeto;
5. Detalhes do desenvolvimento do projeto;
6. Apresentação dos resultados obtidos e testes realizados;
7. Conclusão do trabalho e análise de perspectivas futuras.

Capítulo 2

Descrição do processo

Uma vez que o *software* desenvolvido é destinado ao uso específico de uma ferramentaria automobilística, vale descrever o processo utilizado por ela para maior compreensão da necessidade de se implantar um sistema para gestão de paradas.

2.1 Ferramentarias

As indústrias de ferramentaria se dedicam à produção de ferramentas e moldes que serão utilizados nos processos produtivos de outras indústrias, com peças que realizarão, por exemplo, algum tipo de corte, dobra, conformação ou injeção nos objetos na linha de produção. Elas fazem parte das áreas da metalurgia e usinagem, e abrangem diversos setores industriais, entre os quais podem ser listadas as seguintes as seguintes áreas, mas não sendo limitado a elas (EROMINAS, 2019):

- Automotivo;
- Construção civil;
- Aeroespacial;
- Mineração;
- Agrícola;
- Eletrônica.

Em suma, ferramentarias são indústrias em que a produção visa auxiliar outras indústrias, fornecendo ferramentas que auxiliam na eficiência do processo(EROMINAS, 2019). A ferramentaria cujo projeto desta monografia se destina pertence ao setor automotivo. Nela, são fabricados moldes para injeção plástica e ferramentas para linhas de produção automobilísticas, além de peças utilizadas na montagem de veículos.

2.2 Usinagem

A produção dessas ferramentas é feita por um processo conhecido como usinagem, que é o ato de criar uma peça a partir de uma outra peça bruta maior, normalmente em formato cilíndrico ou retangular, retirando material por meio de uma ferramenta de corte, até se obter o formato e tamanho desejado. Há diversos tipos de operações de usinagem, conforme detalhado nas seções subsequentes (CIMM, 2022).

2.2.1 Torneamento

É utilizado como matéria prima um objeto metálico cilíndrico, conhecido informalmente na indústria como tarugo, que é cortado por meio de um movimento de translação da máquina, enquanto a peça gira em torno do próprio eixo. A máquina que realiza essa operação é chamada de torno, ilustrado na figura 2.1.

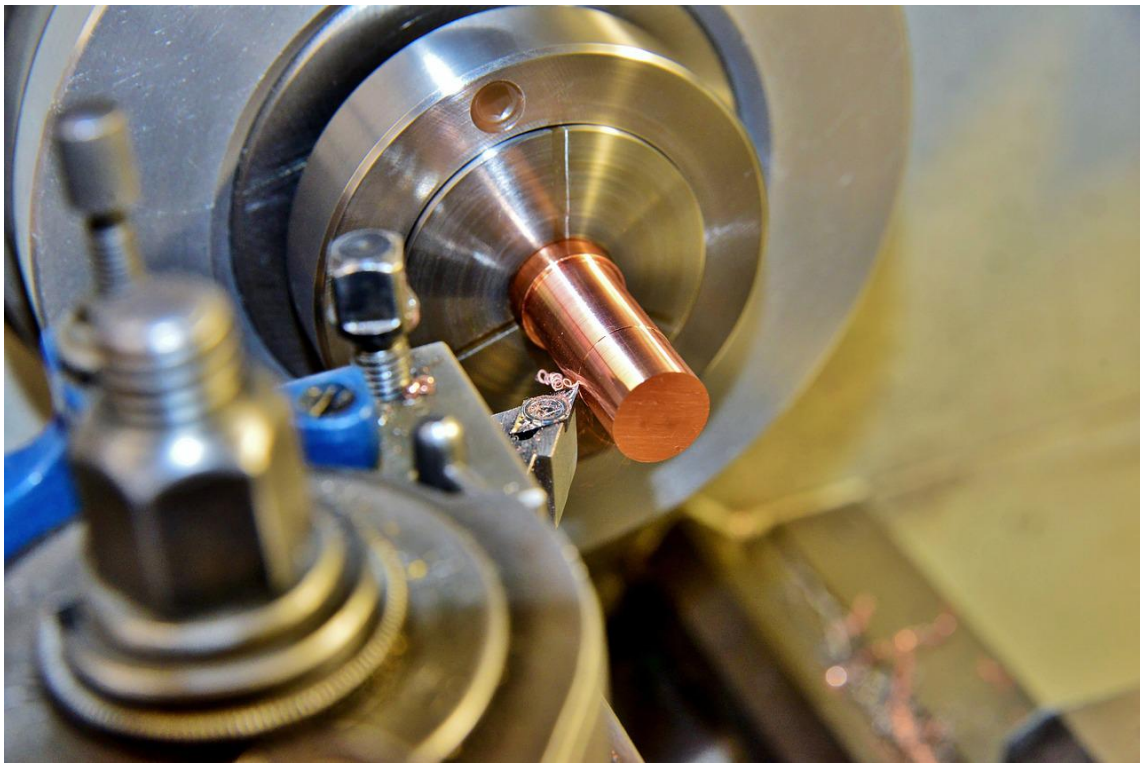


Figura 2.1: Máquina desbastando um tarugo. *Fonte:* Ralph (2019)

2.2.2 Aplainamento

Tem como objetivo moldar superfícies planas por meio de movimentos de translação da máquina em uma peça estática, ou em movimentos de translação da peça com a ferramenta da máquina estática. Uma máquina de aplainamento é mostrada na figura 2.2.

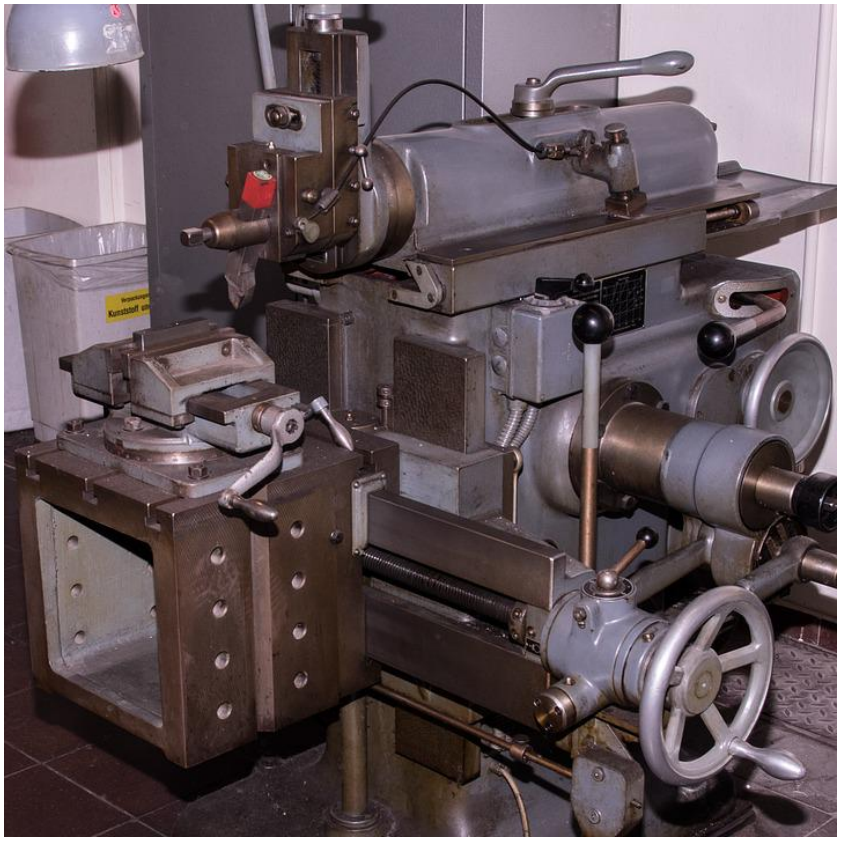


Figura 2.2: Máquina de aplainamento industrial. *Fonte:* Kauer (2019)

2.2.3 Fresamento

Processo realizado por máquinas conhecidas como fresadoras, em que a ferramenta de corte, a fresa, possui diversos gumes e gira enquanto é transladada pela peça com o auxílio de um braço da máquina, num movimento perpendicular ao eixo de rotação. A figura 2.3 mostra uma fresa, e a figura 2.4 apresenta uma máquina fresadora.

2.2.4 Furação

Nesse processo, uma broca é utilizada para criar cavidades cilíndricas ou furos na peça por meio de movimentos de translação e rotação. Além disso, também há uma variação da furação que alarga furos já existentes nas peças por meio do uso de uma ferramenta similar à broca, mas com vários gumes. Uma máquina de furação é apresentada na figura 2.5.

2.2.5 Brochamento

As peças são desbastadas progressivamente em sua superfície, externa ou internamente, por meio de um movimento de translação da máquina chamada de brochadora. Apesar de oferecer um bom acabamento nas peças, cada ferramenta é feita para um uso específico, o

Figura 2.3: Exemplo de fresa. *Fonte:* Schwarzenberger (2014)

que encarece o processo. Um exemplo de peças produzidas por brochamento é mostrado na figura 2.6.

2.2.6 Eletroerosão

Este processo é diferente dos demais devido ao fato de não haver um contato mecânico da peça a ser desbastada e uma ferramenta; o desbaste é feito por meio de descargas elétricas provenientes de eletrodos (PRECISMEC, 2021). A figura 2.7 apresenta uma máquina de eletroerosão.

2.3 Máquinas CNC

Um programa CNC, sigla para Comando Numérico Computadorizado, ou Controle Numérico por Computador, representa um conjunto de instruções de máquina que são utilizadas pelos computadores que controlam máquinas-ferramenta, garantindo uma execução de movimentos sequenciais nos eixos da máquina para que o processo de usinagem seja realizado (CCV INDUSTRIAL, 2017).

Nesse tipo de máquina, o operador atua como um auxílio ao processo, acompanhando o desbaste das peças e realizando trocas de programas, ajustes e calibrações manuais quando necessário.

As máquinas CNC, por serem quase totalmente automáticas, são muito mais caras que máquinas de usinagem mecânicas, e geralmente costumam ser mais presentes em ferramentarias de maior porte.

Figura 2.4: Máquina fresadora. *Fonte:* Ralph (2018)

2.4 Forma de coleta de dados

Na indústria cliente, todas as máquinas que têm seus dados coletados são CNC e possuem sensores que captam os estados trabalhando ou parado. Após a detecção da alteração de estado da máquina, o sensor envia um sinal elétrico para uma rede de comunicação conectada a um computador, que processa o sinal, transforma as informações numa mensagem de formato padronizado e insere em um banco de dados administrado por um sistema integrado de gestão empresarial (*Enterprise Resource Planning*, ou ERP).

Por meio do *software* de gestão, são geradas planilhas Excel contendo informações sobre os eventos de funcionamento ou de parada, e, com isso, são extraídas informações de produtividade. É possível ver um exemplo ilustrativo na figura 2.8. Essa imagem mostra um formato de planilha e de dados similar, mas não igual, ao utilizado pelo cliente, de forma a garantir a confidencialidade do processo real.

Figura 2.5: Máquina de furação. *Fonte:* Pixabay (2016a)

Figura 2.6: Peças de aço produzidas por brochamento. *Fonte:* Pixabay (2016b)

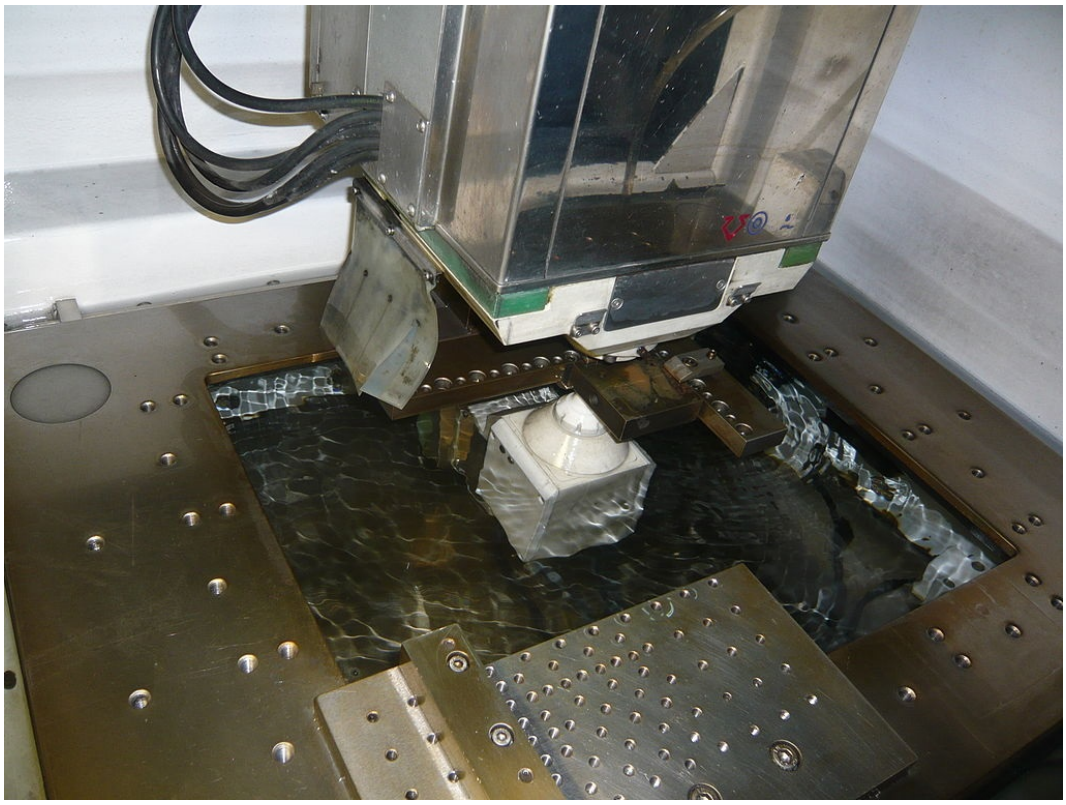


Figura 2.7: Máquina de eletroerosão. *Fonte:* Leonard G. (2006)

	A	B	C	D	E	F	G	H	I	J
1	**Máquina**	**Início**	**Fim**	**Tempo total**	**Estado**	**Motivo**			**Horas**	
2	Fresadora 1	01/01/2022 13:00:00	01/01/2022 13:05:00	00:05:00	Trabalhando	Usinagem		**Tempo trabalhando**	00:18:56	
3	Fresadora 1	01/01/2022 13:05:01	01/01/2022 13:15:00	00:09:59	Parada	Troca de ferramenta		**Tempo parada**	00:33:55	
4	Fresadora 1	01/01/2022 13:15:01	01/01/2022 13:20:00	00:04:59	Trabalhando	Usinagem		**Tempo total**	00:52:51	
5	Fresadora 1	01/01/2022 13:20:01	01/01/2022 13:25:00	00:04:59	Parada	Mudança de modo				
6	Fresadora 1	01/01/2022 13:25:01	01/01/2022 13:27:00	00:01:59	Trabalhando	Usinagem				
7	Fresadora 1	01/01/2022 13:27:01	01/01/2022 13:30:00	00:02:59	Parada	Mudança de modo		**Eficiência**	35.82%	
8	Fresadora 1	01/01/2022 13:30:01	01/01/2022 13:35:00	00:04:59	Trabalhando	Usinagem				
9	Fresadora 1	01/01/2022 13:35:01	01/01/2022 13:48:00	00:12:59	Parada	Máquina travou				
10	Fresadora 1	01/01/2022 13:48:01	01/01/2022 13:50:00	00:01:59	Trabalhando	Usinagem				
11	Fresadora 1	01/01/2022 13:50:01	01/01/2022 13:53:00	00:02:59	Parada	Troca de ferramenta				
12										

Figura 2.8: Imagem ilustrativa de uma planilha de eventos. *Fonte:* elaboração própria

Capítulo 3

Revisão bibliográfica

Em diversas situações na indústria, os processos são feitos de forma manual, rotineira, ineficiente e dispendiosa, algo que gera demoras nos atendimentos, retrabalhos e uso excessivo de recursos. Esse tipo de acontecimento costuma ser um indício de carência de tecnologias que permitem a otimização dos meios de produção e de gestão.

Este projeto busca realizar a automação de um processo rotineiro presente em uma ferramentaria do setor automotivo: a supervisão e análise de paradas de equipamentos. Para isso, foi desenvolvida uma plataforma *web* que fornece tabelas contendo as informações sobre as paradas, como horário, motivo e a operação que estava a ser realizada no momento. Além disso, também são fornecidos indicadores de desempenho para uma melhor gestão do processo por parte dos engenheiros, como eficiência geral do equipamento e indicadores de manutenção para condições de falhas. A plataforma desenvolvida se comporta como um painel de controle para o uso de ativos, num molde similar ao mostrado na figura 3.1, fornecendo todos os dados necessários para um melhor acompanhamento e uma melhor eficiência do processo.

3.1 Importância da gestão de paradas

Nos meios de produção contemporâneos, devido à demanda de quantidade, velocidade de fabricação e complexidade dos produtos que podemos observar na sociedade, torna-se praticamente impossível produzir sem o uso de máquinas. Entretanto, muitos dos equipamentos industriais ainda não são completamente autônomos, dependendo da interação humana em um ou mais partes do processo e, por isso, nem sempre funcionam a todo momento. Além disso, mesmo aqueles que são completamente autônomos ainda são sujeitos a desgastes, falhas, acidentes de trabalho e eventos imprevistos que podem vir a causar paradas no processo produtivo. Por isso, torna-se impossível ter informações de eficiência, produtividade, disponibilidade, controle do estoque e custos, entre diversas outras variáveis do processo importantes para a tomada de decisão, sem o controle de quando a

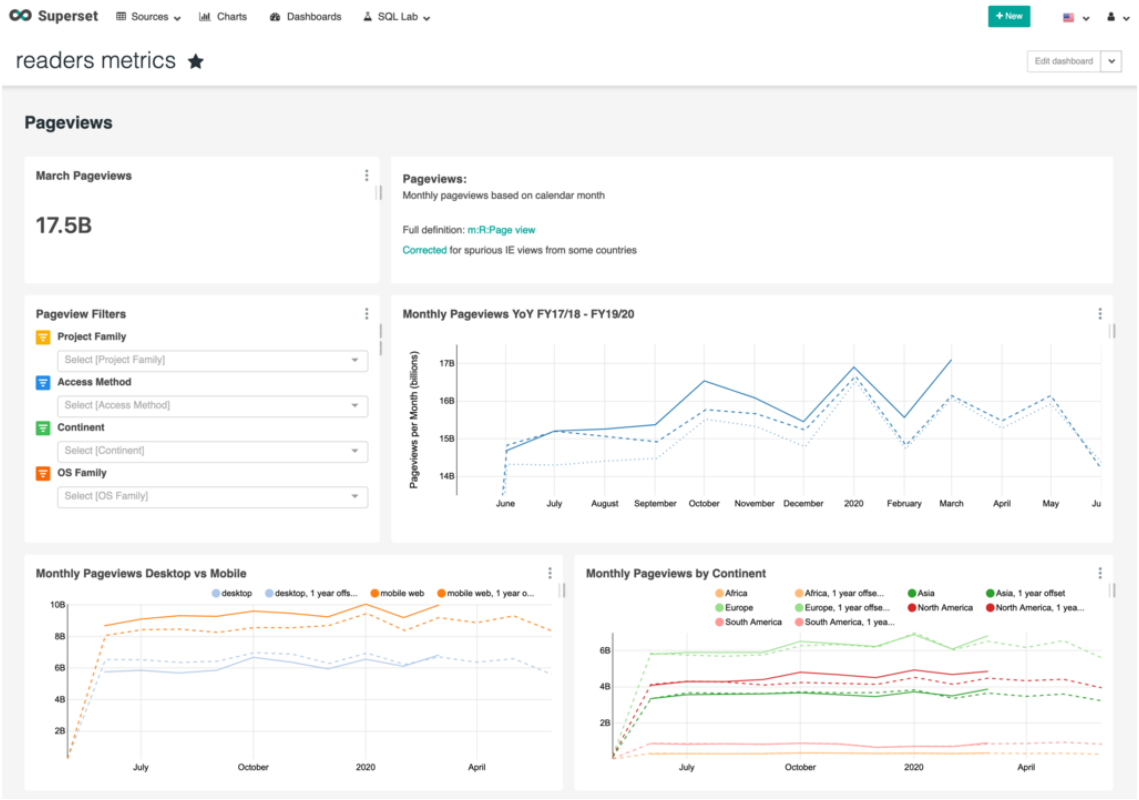


Figura 3.1: Exemplo de painel de controle com diversos indicadores de desempenho e gráficos que caracterizam um processo. *Fonte:* Popov (2020)

máquina funcionou e de quando ela não trabalhou.

Para os casos em que a parada não é por motivo de manutenção, como intervalos para refeições, troca de turnos de trabalho e ausência de um operador, o controle do tempo de funcionamento da máquina é importante para definir gargalos no processo. Por exemplo, numa ferramentaria automotiva, há diversas fresas, de tamanhos diferentes, para serem colocadas nas máquinas para realizar a usinagem, e cada molde a ser fabricado pode exigir um tamanho diferente de fresa para garantir uma precisão específica nos detalhes da peça. Por isso, um operador pode precisar interromper o processo em determinado momento para a troca da fresa na máquina, caracterizando uma parada na produção. Caso essa parada seja muito longa, é necessário haver um replanejamento de parte do processo para tentar agilizá-lo, como deixar as peças numa posição próxima para facilitar seu encontro, obter ferramentas melhores para realizar a troca da fresa de forma mais rápida e até mesmo dirigir a peça a ser usinada para outra fresadora que está equipada com a fresa necessária, de forma a não ser necessário realizar uma troca de peça.

Já nas situações caracterizadas como uma parada de manutenção, que são eventos de prevenção de falhas ou para restaurar a máquina ao seu ponto de operação normal após alguma outra indisponibilidade (FOGLIATO; RIBEIRO, 2009), a situação é, muitas vezes, mais delicada. Uma indisponibilidade por quebra ou erro operacional, por exemplo, pode fazer com que a máquina não trabalhe por horas, ou até dias. Isso, normalmente,

gera prejuízos para a empresa pela falta de produção e atrasos de compromissos (NETO; LIMA, 2002), além dos custos de reparo. Por isso, é necessário não apenas apontar o tempo que a máquina não trabalhou para fins de controle dos indicadores de desempenho, mas é preciso, também, programar e controlar o tempo gasto com manutenções preventivas, tanto para garantir sua realização, quanto para ter indícios de se ela foi realizada no tempo esperado, minimizando as indisponibilidades.

3.2 A necessidade de um novo *software* para gestão de paradas

Segundo Fuentes et al. (2006), com base em dados fornecidos pela Associação Brasileira de Manutenção (ABRAMAN) nos anos de 1997 até 2003, no Brasil, houve um contínuo esforço na capacitação do pessoal na área de manutenção, além de investimentos em equipamentos, no apoio computacional e na melhor organização de recursos. Isso pode ser observado ao se analisar a disponibilidade média das máquinas no Brasil entre 1997, que era de 85.82%, e 2003, havendo um aumento para 89.48%. Entretanto, a disponibilidade dos equipamentos ainda não era satisfatória, uma vez que os melhores indicadores apresentavam valores em torno de 93.3%. Apesar de o documento ser de quase duas décadas atrás, documentos mais recentes da ABRAMAN, do ano de 2017, mostram que não houve melhora significativa nesses indicadores:

Setores	**Disponibilidade dos equipamentos**
Açúcar e álcool, alimentos e bebidas	81%
Aeronáutico e automotivo	82%
Eletroeletrônicos - Energia elétrica	95%
Químico e saneamento	88%
Mineração e siderúrgico	88%
Petróleo e petroquímico	88%
Papel e celulose, e plástico	92%
Predial e prestação de serviços (EQ e MQ)	83%
Máquinas e equipamentos - Metalúrgico	90%
Média geral	**87%**

Tabela 3.1: Dados de disponibilidade do Documento Nacional de 2017. *Fonte:* (ABRAMAN, 2017)

Entre 2003 e 2017 houve uma piora na média geral de, aproximadamente, 2% na disponibilidade média geral dos equipamentos. Ainda segundo Fuentes et al. (2006),

alguns fatos podem justificar esse problema:

1. Muitas empresas adquirem ERPs, programas de computador para processamento de dados de toda a organização. Tais *softwares* costumam possuir um excesso de informações de negócio disponíveis, dificultando análises específicas focadas apenas na área de manutenção;

2. A obtenção de *softwares* específicos para manutenção, chamados de CMMS (*Computer Maintenance Management Systems*), mas sem um plano e um projeto para análise dos dados obtidos, além da ausência de funções administrativas, tarefas e responsabilidades ao lidar com as informações, caracterizando um investimento com baixo aproveitamento.

Por fim, Fuentes et al. (2006) afirma que ao se impor um sistema de gestão sem considerar as particularidades dos usuários, ele é rejeitado e o uso se torna ineficiente.

3.3 Indicadores de desempenho

Muitas vezes, a equipe de gestão de uma empresa recorre apenas a indicadores de desempenho de característica econômico-financeira que revelam informações sobre o mercado e competição, mas deixando de lado os indicadores que representam o processo produtivo (SILVA, 2013).

De forma a auxiliar na gestão do processo e na tomada de decisão, o *software* recebe eventos de funcionamento ou de parada das máquinas e realiza diversos cálculos de indicadores de desempenho, também chamados de *KPIs* (*key performance indicators*), assim permitindo uma visão geral do desempenho da fábrica como um todo e de cada equipamento individual.

A seguir, são explicados os indicadores de desempenho utilizados e calculados pelo *software*.

3.3.1 *MTTR - Mean Time To Repair*

Segundo Kardec e Nascif (2009), o indicador chamado *Mean Time To Repair* (MTTR), ou Tempo Médio Para Reparos (TMPR), possui grande importância para a manutenção, pois está ligado ao desempenho do equipamento, dependendo da organização e planejamento da produção, da sua facilidade de ser mantido e da capacidade profissional do operador responsável pela manutenção.

Ele é calculado pela seguinte fórmula:

$$MTTR = \frac{TR}{NR} \tag{3.1}$$

Em que TR é o tempo total gasto em reparos no equipamento em determinado intervalo, e NR é o número de reparos realizados.

3.3.2 *MTBF - Mean Time Between Failures*

Kardec e Nascif (2009) define o *Mean Time Between Failures* (MTBF), ou Tempo Médio Entre Falhas (TMEF), como uma medida da vida média dos equipamentos, e é calculado por meio da divisão entre o número de horas operantes pela quantidade de falhas, ou seja:

$$MTBF = \frac{TO}{NF} \tag{3.2}$$

Em que TO equivale ao tempo operante no intervalo, e NF ao número de falhas ocorridas.

3.3.3 Disponibilidade

Utilizando os dois indicadores anteriores, Kardec e Nascif (2009) define um indicador chamado Disponibilidade Inerente, que é o percentual do tempo que o equipamento possui disponível para a produção, sem levar em conta atrasos do processo. O termo inerente se deve justamente ao fato de que esse indicador não leva em consideração tempos de logística ou de espera.

Ele é calculado pela seguinte equação:

$$Disponibilidade = \frac{MTBF}{MTBF + MTTR} \tag{3.3}$$

Em que MTBF e MTTR são, respectivamente, os indicadores de Tempo Médio Entre Falhas e Tempo Médio Para Reparos explicados anteriormente. Vale destacar que, para a disponibilidade inerente, o MTTR considera apenas paradas por manutenção corretiva, ignorando as manutenções preventivas.

3.3.4 Eficiência

A taxa de eficiência é o tempo produtivo da máquina em relação ao tempo total de operação. O tempo produtivo pode ser expresso como a taxa ideal de produção da máquina multiplicada pela produção real (ALMEANAZEL, 2010). Por exemplo, a taxa ideal pode ser representada em unidades como partes por hora.

A fórmula da eficiência é dada por:

$$Eficiência = \frac{TI * S}{TO} \tag{3.4}$$

Em que TI é a taxa ideal de produção, S é a saída total de produtos, e TO o tempo operante do processo.

3.3.5 Qualidade

A qualidade é uma taxa que expressa a quantidade de produtos não-defeituosos em relação ao total produzido (ALMEANAZEL, 2010). Ela é expressa pela fórmula:

$$Qualidade = \frac{TP - D}{TP} \tag{3.5}$$

Em que TP é o total produzido, e D é o total defeituoso.

3.3.6 *OEE - Overall Equipment Effectiveness*

O indicador conhecido como *Overall Equipment Effectiveness* (*OEE*), ou Eficácia Global do Equipamento, foi cunhado nas práticas da *Total Productive Maintenance* (TPM), sistema criado no Japão por Seiichi Nakajima, no *Japan Institute of Plant Maintenance* (JIPM). O TPM busca alcançar a *performance* ideal, sem perdas ou defeitos de fabricação, acidentes de trabalho, ou qualquer outro fator que comprometa o processo (ALMEANAZEL, 2010). Por meio da quantificação do poder produtivo total do processo e das perdas ocorridas, é possível calcular o indicador, de forma a permitir uma visão geral do desempenho da fábrica.

Segundo Silva (2013), o OEE, faz a medição "tri-dimensional" do desempenho, pois leva em consideração três fatores, que foram explicados anteriormente:

1. Disponibilidade: tempo útil que o equipamento possui para produzir;

2. Eficiência: capacidade de produzir em relação à cadência nominal da linha produtiva;

3. Qualidade: qualidade do produto obtido pelo processo.

Sendo cada um desses itens um valor entre 0 e 1.

De acordo com Busso e Miyake (2013), o OEE é calculado por meio da identificação de 3 categorias de perdas:

1. Perdas de disponibilidade:

    [a.] Paradas causadas por falhas da máquina;

    [b.] Paradas para a realização do *setup*, tempo gasto para troca da fabricação de um produto para outro, ou ajustes do processo.

2. Perdas de desempenho:

    [a.] Paradas breves geradas por funcionamento indevido da máquina;

    [b.] A máquina produz mais devagar devido ao mau funcionamento de algo.

3. Defeitos e perda de qualidade:

    [a.] Produção que veio com defeito, ou necessidade de retrabalho;

    [b.] Perdas durante o início da produção até o momento em que a máquina se estabiliza.

Por meio da identificação dessas perdas, e posterior cálculo de cada uma das variáveis individuais, o cálculo da OEE é feito por meio da seguinte fórmula:

$$OEE = Disponibilidade \times Eficiência \times Qualidade \tag{3.6}$$

Por fim, segundo (KWON; LEE, 2004 apud BUSSO; MIYAKE, 2013), o aumento do OEE mediante aos esforços para melhoria da disponibilidade do processo converte-se na redução de custos e aumento incremental dos lucros. Além disso, o sistema de manutenção e a gestão dos equipamentos podem ter suas efetividades avaliadas por meio do OEE (CHAND; SHIRVANI, 2000 apud BUSSO; MIYAKE, 2013).

3.4 Princípio de Pareto

Segundo Powell e Sammut-Bonnici (2014), o princípio de Pareto, também conhecido como "regra 80/20", é baseado nas observações realizadas por Vilfredo Pareto, economista italiano do século XIX, que afirmava que havia um desequilíbrio previsível na economia da Itália, sendo 80% da riqueza do país acumulada por 20% da população. Então, pesquisadores passaram a notar esse desequilíbrio em diversos outros setores da economia, como gestão da produção e gestão de finanças.

Assim, surgiu a ideia de que a maior parte dos problemas das empresas é causada por um conjunto pequeno de problemas. A aplicação desse princípio permite encontrar os pontos fortes e fracos das organizações por meio de seus recursos internos e capacidades.

Uma forma de se realizar essa análise é por meio de um Diagrama de Pareto. Nele, são elencadas as diversas ocorrências da organização em formato de barras, ordenadas de forma decrescente na frequência, normalmente acompanhadas de uma curva que representa o percentual do total de cada tipo de ocorrência (CAMARGO, 2018).

Um exemplo desse diagrama, aplicado a uma ferramentaria, é mostrado na figura 3.2.

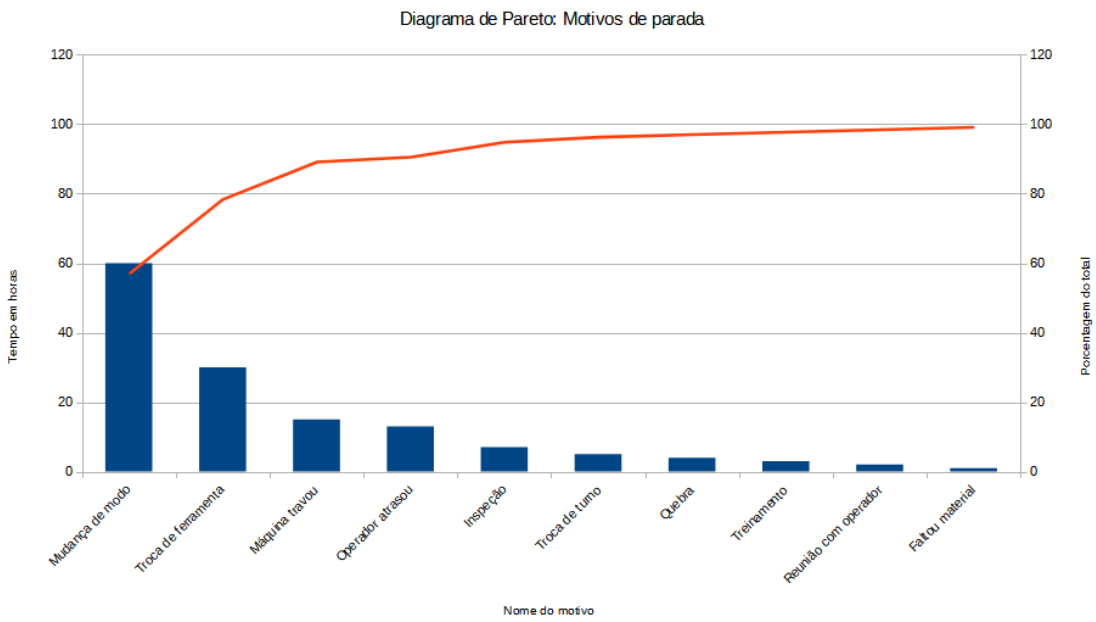


Figura 3.2: Exemplo de Diagrama de Pareto. *Fonte:* elaboração própria

3.5 Presença do tema na literatura

O problema de paradas na indústria é algo vasto, muito trabalhado e estudado, com o intuito de reduzir ao máximo o tempo em que os equipamentos se encontram inoperantes, uma vez que isso representa perda de tempo útil para produção. Entre as soluções já apresentadas no meio acadêmico, observa-se que há pouca presença de um foco voltado para o desenvolvimento de sistemas de gestão, sendo mais comum o estudo dos impactos e apresentação de ideias para minimizar o problema.

Para demonstrar isso, foi realizada uma pesquisa por palavras-chave em alguns dos portais mais conhecidos no meio acadêmico nacional, SciELO e CAPES, e internacional, World Wide Science (WWS), indo de temas mais abrangentes até temas mais específicos, relacionados de forma mais próxima à proposta do projeto desta monografia. Os resultados em português são apresentados na tabela 3.2, e em inglês na tabela 3.3.

Como é possível observar nas tabelas acima, a combinação dos termos relevantes ao projeto causam uma redução drástica na quantidade de material acadêmico em todos os portais consultados. Além disso, apesar de alguns resultados ainda apresentarem números relativamente expressivos, na casa de centenas, uma rápida observação dos resultados mostrou que a maioria dos materiais não estava diretamente relacionada com a ideia de desenvolver um programa de computador para gestão de paradas, ou gestão de manutenção, sendo apenas demonstrações de resultados do uso de algum *software* pronto, ou falhas na busca por parte dos buscadores.

Dessa forma, nota-se uma carência no meio acadêmico de propostas específicas de *softwares* para gestão de paradas, em que a maioria dos produtos relacionados encontrados

Palavras-chave	**Quantidade por portal**		
	SciELO	CAPES	WWS
manutenção	6015	56671	1384
gestão de paradas	4	2020	505
indicadores de desempenho	1390	24815	1433
(*software*) AND (manutenção)	159	7458	672
(*software*) AND (gestão de paradas)	0	485	263
(*software*) AND (indicadores de desempenho)	31	6786	572
(*software*) AND (indicadores de desempenho) AND (gestão de paradas)	0	164	251

Tabela 3.2: Quantidade de resultados para cada combinação de palavras-chave em português. *Fonte:* elaboração própria

Palavras-chave	**Quantidade por portal**		
	SciELO	CAPES	WWS
maintenance	4773	2,32mi	5371
stoppage management	3	12759	2203
key performance indicators	152	696885	2492
(*software*) AND (*maintenance*)	183	660851	1031
(*software*) AND (*stoppage management*)	0	2939	1046
(*software*) AND (*key performance indicators*)	4	257465	1595
(*software*) AND (*key performance indicators*) AND (*stoppage management*)	0	543	611

Tabela 3.3: Quantidade de resultados para cada combinação de palavras-chave em inglês. *Fonte:* elaboração própria

são soluções providas por empresas privadas.

3.6 Trabalhos prévios na literatura

Uma proposta de *software* encontrada foi feita para o Simpósio de Excelência em Gestão e Tecnologia (SEGET), em que foi criado um sistema para gestão de paradas em uma cervejaria, em que máquinas que realizam envase nas linhas de produção necessitavam ter suas interrupções relatadas para que fossem entendidas as razões e áreas responsáveis pelas principais paradas dos equipamentos. Além disso, eram utilizadas planilhas sem o uso de bancos de dados para armazenamento de informações, e isso causava dificuldades nas buscas de dados e promovia possibilidade de erros humanos no momento de inserir os dados (CARVALHO; FERNANDES; CÔBERO, 2012).

Para isso, esse programa de computador foi desenvolvido para sistematizar os dados relacionados a cada parada, como área, setor da indústria, motivo, e turno, para que fossem gerados relatórios para auxiliar na tomada de decisão por parte dos diretores. O *software* apresenta um banco de dados relacional, separando os dados das paradas em

tabelas, uma interface gráfica para cadastro das informações, e para busca, filtragem e visualização dos dados e de relações entre suas diversas categorias, assim como apresentar estatísticas de desempenho do processo.

O projeto teve como pontos fortes a presença de diversos indicadores que permitem a visualização do funcionamento do processo, como disponibilidade, confiabilidade e eficiência global, além de conseguir categorizar os parâmetros dele, como áreas, turnos, produtos e motivos de paradas, diminuindo as chances de erros humanos. Entretanto, o *software* apresentou uma interface com poucos elementos visuais para análise temporal, como gráficos, tendo um foco no uso de tabelas que, apesar de auxiliarem na visualização categórica dos dados, não permitem observar a progressão dos parâmetros e indicadores no tempo (CARVALHO; FERNANDES; CÔBERO, 2012).

Outro projeto que aborda o tema de gestão de paradas foi proposto na revista Inovação, Gestão e Produção (INGEPRO), em que foi implementado um sistema para monitorar paradas de máquinas em uma indústria de usinagem de peças automotivas. Similarmente ao projeto da cervejaria citado anteriormente, seu objetivo foi mitigar o problema de apontamentos manuais das paradas, bem como maior organização dos dados para facilitar buscas e coletas. Além disso, também buscou fornecer indicadores de desempenho para os gestores da indústria. Por fim, o projeto buscou estudar os benefícios causados pela implementação desse sistema.

Para isso, foram categorizadas os tipos de paradas presentes e foi utilizado um *software* proprietário de uma outra empresa, que já disponibiliza diversas ferramentas para inserção, análise e visualização dos dados. Com o sistema implementado e funcionando na indústria, o estudo coletou informações relacionadas à eficiência do processo e dados sobre as paradas. A partir disso, concluiu que, em contrapartida à inserção manual das informações, com o uso do sistema de gestão houve melhora de 14% do indicador *Overall Equipment Effectiveness* (*OEE*), e foi propiciado uma melhor gestão das informações, facilitando tomadas de decisão. O estudo se mostrou bem-sucedido na implementação do sistema, realizando análises estatísticas do processo antes e depois do projeto, demonstrando resultados positivos (RENO et al., 2010).

Além dos dois projetos previamente citados, também há uma gama de estudos relacionados à gestão de paradas, mas com uma abordagem mais teórica, como um estudo realizado na Universidade Federal Rural do Semi-Árido, que lista e analisa diversos indicadores de desempenho para gestão de paradas programadas de grandes linhas de produção. A pesquisa buscou observar o impacto que o uso de indicadores de desempenho promovem nas linhas de produção, com a justificativa de que o mercado exige que a produção apresente números elevados, que podem ser alcançados com um bom funcionamento dos equipamentos, apresentando boa disponibilidade e confiabilidade (FREITAS, 2019).

A pesquisa cita vários indicadores de desempenho para paradas, como *OEE*, disponibilidade de equipamentos, desempenho, qualidade, desvio de prazo e taxas de frequência de

acidentes. Por fim, a pesquisa realizou a quantificação dos indicadores em uma empresa produtora de cimento, e conclui afirmando que os resultados obtidos tornam possíveis uma melhor análise do processo, permitindo uma melhor tomada de decisões (FREITAS, 2019).

Capítulo 4

Metodologia

Uma vez que o *software* desenvolvido faz muito uso de dados temporais, é de extrema importância a implantação de um banco de dados para os armazenar de forma padronizada, robusta e segura. Além disso, para garantir um projeto bem feito, de simples entendimento e de fácil modificação e expansão futura, foi necessário definir um padrão de arquitetura a ser seguido, baseado em conceitos já bastante fundamentados na literatura da área de engenharia de *software*.

Por isso, o projeto foi proposto com a definição de três planos de ação, nesta ordem:

1. Arquitetura do sistema;
2. Arquitetura do banco de dados;
3. Passo a passo de implementação.

Após esses três itens serem bem definidos, o projeto pôde ser desenvolvido. Porém, antes de explicar cada parte do projeto, é necessário compreender a diferença entre dois conceitos fundamentais de programação, *back-end* e *front-end*.

4.1 O que é *back-end* e *front-end*?

4.1.1 *Back-end*

Segundo FOLDOC (1996), ou *Free On-line Dictionary of Computing*, dicionário de termos de ciências da computação criado em 1985 por Denis Howe, o *back-end* é a camada de *software* que realiza tarefas não visíveis ao usuário do sistema, ou que realiza a parte final de um processo. Um exemplo de *back-end* é um compilador de linguagens de programação, que transforma o código em linguagem de máquina e realiza otimizações para a arquitetura do computador.

Pode-se dizer, então, que o *back-end* é a parte do *software* que realiza o processamento de dados, sem uma interação direta com o usuário. No sistema proposto nesta monografia,

o *back-end* fica responsável por realizar autenticação de usuários, armazenar informações no banco de dados, garantir a validade e consistência dos dados, realizar cálculos e receber requisições do *front-end*.

4.1.2 *Front-end*

Também segundo FOLDOC (s.d), o *front-end* é a camada de *software* que fornece uma *interface* para um outro programa que, normalmente, não é "amigável" ao usuário. Com isso, é possível afirmar que o *front-end* é a parte do *software* responsável pela interação direta com o usuário, fornecendo elementos visuais, como botões, textos, imagens e gráficos, para permitir uma maior facilidade de uso do sistema.

Normalmente, o *front-end* é apresentado como a parte visível de programas de *desktop*, ou como páginas *web*. Ele, geralmente, não se comunica diretamente com o banco de dados, sendo necessário encaminhar os pedidos de interação do usuário, como cliques em botões e inserção ou atualização de dados, para uma camada de *back-end*. O *front-end* também pode solicitar informações ao *back-end*, e este fará a busca no banco de dados, por exemplo, e enviará os dados obtidos de volta para ser exibido na tela do usuário. Há diversas formas de se realizar essa troca de informações, utilizando protocolos de comunicação de *software*, como *Hypertext Transfer Protocol* (HTTP) e *Open Platform Communications* (OPC).

4.2 Arquitetura do sistema

O primeiro passo na definição dos padrões do *software* foi definir toda a arquitetura do sistema. Quais *frameworks*, bibliotecas e plataformas seriam utilizadas, quais suas vantagens e desvantagens em relação a opções concorrentes, e qual estrutura de fluxo de dados a aplicação teria. Após a análise de diversas arquiteturas possíveis, o *solution stack* (conjunto de soluções) da aplicação foi definido pelos seguintes itens, que serão explicados a seguir:

1. *Front-end*:
    [a.] *Single page application*: Angular.
2. *Back-end*:
    [a.] .NET *Core* e ASP.NET *Core*;
    [b.] *Entity Framework Core*;
    [c.] PostgreSQL;
    [d.] Redis;
    [e.] RabbitMQ.

Além disso, também foram definidos padrões de arquiteturas para o desenvolvimento do código:

- *Models*;
- Camada *Service* e padrão *Repository*;
- Princípios SOLID;
- Arquitetura multinível: 3 níveis lógicos;
- Arquitetura REST;

4.2.1 *Single page application vs MVC*

Para a interação direta com o usuário, foi definido o uso de um tipo de aplicação conhecido como *Single Page Application* (SPA), ou aplicação de página única. Segundo Mikowski e Powell (2013), as SPAs são aplicações que não necessitam de recarregar a página enquanto o usuário utiliza e interage com o conteúdo. É uma aplicação inteira entregue ao navegador do cliente, que altera o conteúdo da sua página dinamicamente. Os SPAs mais utilizados são feitos em JavaScript, e apresentam diversas vantagens sobre a abordagem tradicional de enviar páginas estáticas para o cliente a cada interação, combinando benefícios de aplicações de *desktop* com benefícios de *websites*:

- Conseguem renderizar suas partes de forma individual, similarmente a aplicações de *desktop*, em contrapartida a páginas servidas estaticamente, que devem ser completamente redesenhadas na tela a cada atualização no navegador;
- Permitem realizar processamentos localmente, ao invés de somente enviar dados para o servidor e aguardar uma resposta, similarmente a aplicativos de *desktop*, o que acelera a resposta visual ao cliente;
- Pode mostrar seu estado ao usuário, como exibir barras de carregamento durante processamentos, diferentemente de páginas estáticas, que simplesmente ficam em branco até serem carregadas;
- Permitem atualização e distribuição da aplicação similar a de um *website*, bastando atualizar a página no navegador para as receber, sem necessidade de reinstalar nos computadores dos clientes;
- Conseguem ser compatíveis com qualquer sistema operacional na maioria das situações.

É possível exemplificar o funcionamento de uma arquitetura SPA por meio do seguinte diagrama:

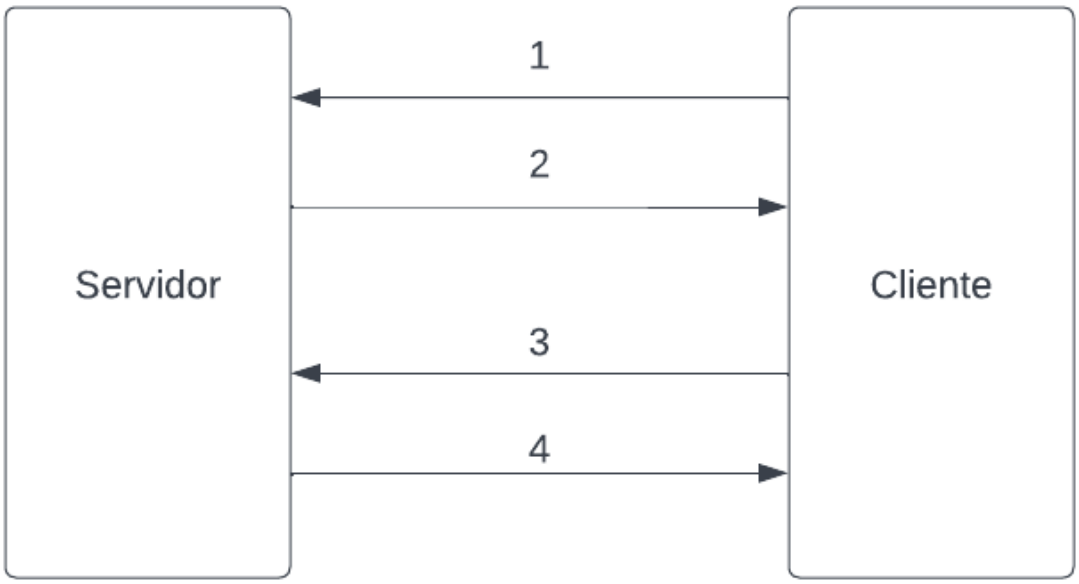


Figura 4.1: Fluxo de dados de uma aplicação SPA. *Fonte:* elaboração própria

Seguindo a ordem da numeração presente na figura 4.1:

1. O cliente acessa o endereço do *website*, o que provoca um envio de uma requisição ao *web server*;

2. O *web server* devolve ao cliente o conjunto de arquivos que constituem a SPA, como HTML, CSS e JavaScript;

3. Durante o uso da aplicação, ao interagir com algum elemento na tela, é disparada uma requisição HTTP para o *web server*, que a encaminha para o *back-end* da aplicação;

4. O *back-end* processa a requisição e envia uma mensagem, normalmente no formato *JavaScript Object Notation* (JSON), para o *web-server*, que, então, encaminha para o cliente. Então, a partir dos dados recebidos, a SPA atualiza alguma parte de sua tela, de forma a refletir o comportamento esperado pela interação inicial do usuário, como carregar uma imagem ou uma tela nova.

Dessa forma, após o carregamento inicial (passos 1 e 2), o uso da aplicação se torna mais dinâmico por meio da troca de mensagens menores (passos 3 e 4), que se repetem

continuamente enquanto o usuário permanece no *website*, devido a não ser mais necessário o servidor reenviar páginas inteiras a cada requisição.

Em contrapartida, seria possível utilizar um outro tipo de aplicação para a interação com o usuário, conhecido como *Model-Viewer-Controller* (*MVC*), inventado pelo professor Trygve Reenskaug em 1979, com o principal objetivo de criar uma forma de conectar o formato dos dados tratados pelo computador com as informações que o usuário final busca visualizar e editar (REENSKAUG, 1979).

De uma forma mais técnica, é uma forma de separar a parte da aplicação que interage com o usuário da parte de processamento de dados, uma vez que, segundo Deacon (2009), as considerações que ditam o *design* do *graphical user interface* (*GUI*), ou a parte visual da aplicação, não devem ser as mesmas considerações que ditam o resto da aplicação.

Ainda segundo Deacon (2009), cada item da sigla MVC possui o seguinte significado:

- *Model*: conjunto de classes que modelam o problema a ser resolvido pela aplicação, definindo o formato dos dados, sendo totalmente agnóstico ao resto da implementação;
- *View*: conjunto de classes que representam elementos visuais na tela do usuário, permitindo visualizar e interagir com o *Model*. São bastante independentes de plataforma, fazendo pouca ou nenhuma diferença em qual sistema operacional elas serão utilizadas;
- *Controller*: objeto que permite a manipulação da *View*. Eles recebem as entradas do usuários e retornam respostas para as *Views*, além de possuírem conhecimento da plataforma utilizada pelo usuário.

Ainda vale destacar que as *Views* devem conhecer o *Model*, mas ser agnósticas aos *Controllers*. Estes, por sua vez, devem conhecer as *Views* e o *Model*.

Seguindo a numeração na figura 4.2, é possível definir o seguinte caminho de fluxo de dados para uma aplicação MVC tradicional:

1. O usuário interage com a *View* e envia algum dado;
2. A *View* envia o dado do usuário ao *Controller*;
3. O *Controller* envia o dado ao *Model*;
4. O *Model* processa o dado, e insere ou busca informações no banco de dados;
5. O banco retorna uma resposta ao *Model*;
6. O *Model* devolve a resposta ao *Controller*;
7. O *Controller* seleciona a *View* e insere os dados da resposta;

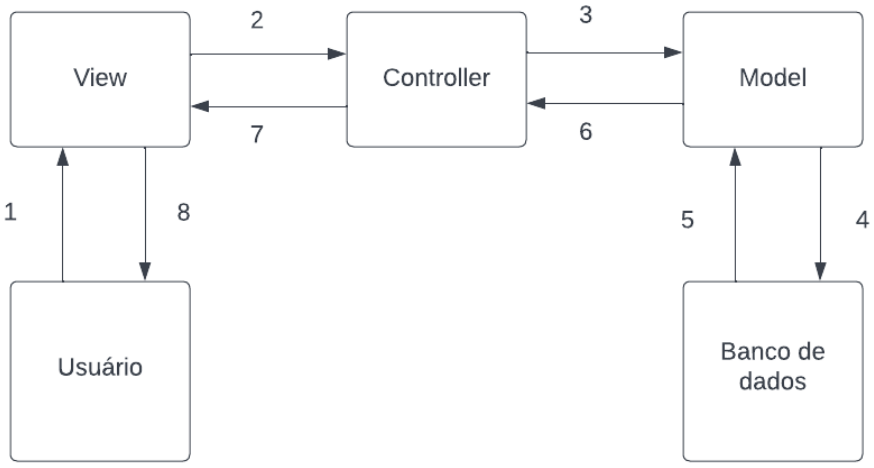


Figura 4.2: Fluxo de dados de uma aplicação MVC. *Fonte:* elaboração própria

8. A *View* atualizada é exibida ao usuário.

Por fim, apesar de o MVC ser um padrão bastante conhecido e utilizado na comunidade, optou-se pelo modelo SPA pelas vantagens já citadas anteriormente, combinando benefícios tanto de aplicações de *desktop* quanto de *websites*, além do fato de que um modelo MVC tradicional foca em servir páginas continuamente durante o uso da aplicação, o que pode provocar lentidão se comparado à abordagem SPA, sendo possível citar, por exemplo, um *website* destinado a galerias de fotos. Ao se clicar num *link* para visualizar uma imagem, toda a página é atualizada, todos os componentes na tela redesenhados, e todas as informações visuais que não se encontram com o cliente sejam buscadas, novamente, no servidor. No modelo SPA, toda a página se manteria carregada, alterando apenas o necessário para que o usuário visualize a imagem (MIKOWSKI; POWELL, 2013).

Angular

Para criar uma SPA no *front-end*, foi utilizada a tecnologia Angular. Ela é uma plataforma para o desenvolvimento de aplicações *web*, próprio para *front-end*. De acordo com a documentação oficial do Angular, ele é uma plataforma para desenvolvimento baseada na linguagem TypeScript, incluindo um *framework* baseado em componentes, uma coleção de bibliotecas para realização de diversas tarefas, como preenchimento de formulários, roteamento e comunicação com servidores, e um conjunto de ferramentas para auxílio no de desenvolvimento do código. Uma das maiores vantagens citadas é a possibilidade de criar aplicações escaláveis (ANGULAR TEAM, 2022).

Em contrapartida, foram cogitadas duas outras possibilidades para o desenvolvimento do *front-end*: React e Vue.js. Ambos apresentam maior velocidade de renderização de componentes na tela que o Angular (SAKS, 2019), e React já apresenta maior popularidade na comunidade de programação, com mais material e componentes prontos, além de maior simplicidade de desenvolvimento. Vue.js, apesar de ser o mais rápido entre os três, ainda é muito novo se comparado aos outros, e não possui tanto material para apoio (WOHLGETHAN, 2018).

Entretanto, optou-se por utilizar Angular devido a sua grande estabilidade e popularidade na comunidade, que, apesar de ter perdido espaço para o React, ainda é muito expressiva. Além disso, por possuir um *framework*, e não uma biblioteca, como é o caso do React, há uma grande gama de ferramentas já disponíveis para auxílio no desenvolvimento, além de fornecer maior segurança no fluxo de dados da aplicação, uma vez que o próprio *framework* já fornece um molde pronto para isso, dando pouca margem a erros.

4.2.2 .NET Core e ASP.NET Core

.NET é uma plataforma gratuita e de código aberto desenvolvida pela Microsoft, utilizada para o desenvolvimento de diversos tipos de aplicações, como, mas não limitado a:

- Aplicativos Web;
- Jogos;
- Aplicativos de celular;
- Programas para *desktop*;
- Serviços Windows.

A premissa da plataforma é que o código do projeto é sempre similar, independente da plataforma de destino da aplicação, tendo sempre as mesmas funções disponíveis para o desenvolvimento. Além disso, permite criar aplicações para diversos tipos de sistemas operacionais diferentes, como Windows, Linux e macOS (MICROSOFT, 2022c).

O .NET Core é uma implementação do .NET, funcionando em todas as plataformas, diferentemente de outras implementações, como .NET Framework e UWP, que funcionam apenas em ambientes Windows (MICROSOFT, 2022c).

Já o ASP.NET Core é um *framework* para o desenvolvimento de *back-end* para aplicações *web*, construído utilizando o .NET Core como base. Ele é a versão de código aberto do ASP.NET e funciona nos sistemas operacionais Windows, Linux e macOS (MICROSOFT, 2022b).

A razão por trás da escolha dessas ferramentas da Microsoft se deve pela grande utilização de sistemas Windows na indústria. Até os dias atuais, há um domínio por parte do *software* da Microsoft no meio industrial, juntamente de produtos da Intel e da Cisco (GAWER; CUSUMANO et al., 2002). Devido a isso, há uma grande quantidade de dispositivos, *drivers* de comunicação e *softwares* já adaptados para o funcionamento com o .NET.

4.2.3 *Entity Framework Core*

O *Entity Framework Core* é um tipo de *software* conhecido como *Object-Relational Mapper* (*ORM*), ou mapeador objeto-relacionamento. É uma biblioteca da Microsoft que cria uma abstração na comunicação com o banco de dados, e permite executar consultas ao banco por meio de funções em C#, sem necessidade de programação direta de SQL. Cada tabela no banco é mapeada para uma classe no código, em que as colunas são propriedades da classe (ABUHAKMEH, 2020). Os objetos instanciados de cada classe são conhecidos como entidades.

Além disso, ele possui um conjunto de ferramentas que facilitam o trabalho com bancos de dados. É possível gerenciar históricos de alterações no banco, chamados de *migrations*, bem como realizar operações de engenharia reversa em bancos de dados já existentes, criando as classes que representam cada tabela do banco (ABUHAKMEH, 2020).

A abstração criada pelo Entity Framework Core permite um desenvolvimento muito mais veloz e facilmente testável se comparado ao uso de opções mais diretas, que dependem da programação de SQL, como o ADO.NET, também da Microsoft, às custas de perda de performance da aplicação.

4.2.4 PostgreSQL

Para o banco de dados, foi escolhido o PostgreSQL. Ele é um sistema de banco de dados objeto-relacional de código aberto, em desenvolvimento e uso desde 1986. Ele usa e estende a linguagem SQL, implementando diversas funcionalidades novas para escalar e armazenar dados complexos. Além disso, ele possui grande reputação positiva em meio à comunidade pela sua robustez, arquitetura e confiabilidade, além de funcionar em todos os principais sistemas operacionais do mercado (POSTGRESQL, 2022).

Ele foi escolhido para o projeto justamente por ser uma opção muito robusta e com grande apoio da comunidade *online*, além de possuir bibliotecas de código aberto que permitem grande integração com o Entity Framework Core, facilitado a troca de dados com o banco por meio do código em C#.

Outra opção de banco de dados possível seria o SQL Server, da Microsoft, que também possui grande integração com o Entity Framework Core. Entretanto, o PostgreSQL possui melhor disponibilidade nos mais diversos sistemas operacionais, permitindo maior

liberdade de desenvolvimento e implementação, e um maior controle de acessos concorrentes ao banco, o que facilita o desenvolvimento de aplicações com métodos assíncronos e com o uso de *threads*, além de também ser mais facilmente escalável (PLUS, 2018). Outro ponto é que o SQL Server não é uma solução código aberto, e seu uso criaria uma grande dependência de *softwares* da Microsoft.

4.2.5 Redis

Além do PostgreSQL, um outro tipo de banco de dados foi utilizado. Redis é um *software* de armazenamento em memória RAM para estruturas de dados, de código aberto, e possui diversas funções, como banco de dados, intermediário de mensagens, e *cache* (REDIS, 2022a).

Seu uso teve como objetivo auxiliar na troca de mensagens simples entre partes da aplicação de forma a manter um bom desempenho. Estas mensagens são aquelas que não se encaixam na arquitetura do banco projetada para o PostgreSQL, que sofrem alterações muito recorrentes, de forma que um banco de dados tradicional poderia causar impactos de desempenho na aplicação se precisasse lidar com essas informações com frequência.

Como o principal meio de armazenamento que o Redis utiliza é a memória RAM, buscas e alterações de variáveis são feitas com uma velocidade muito maior que se feitas em disco. Ademais, as mensagens ocupam pouco espaço, com 1 milhão de valores pequenos salvos em memória ocupando cerca de 85 MB, ou 1 milhão de *hashes* ocupando 160 MB (REDIS, 2022b).

4.2.6 RabbitMQ

O último *software* do *solution stack* utilizado é o RabbitMQ, um *software* especializado em enfileiramento de mensagens, também chamado de *message broker* (intermediário de mensagens), que cria filas, e diversas aplicações podem depositar diferentes tipos mensagens nelas, com o principal objetivo de transmitir informações entre programas (JOHANSSON, 2019).

O RabbitMQ permite diminuir a carga de dados transmitida às aplicações durante trocas de mensagens, uma vez que uma comunicação direta entre dois *softwares* exigiria uma conexão entre eles, que pode ser contínua ou não, bem como o processamento imediato das mensagens, ou seu armazenamento na memória do programa, o que pode vir a comprometer seu desempenho em situações de mensagens muito grandes ou em grande número (JOHANSSON, 2019).

Apesar de o Redis também fornecer suporte para sistemas de enfileiramento, o RabbitMQ foi escolhido por ser um aplicativo especializado nessa tarefa, além de ser uma referência muito conhecida e utilizada pela comunidade.

4.2.7 *Models*

Os *Models*, ou modelos, são formas de representação do problema a ser tratado pelo sistema. Eles objetivam abstrair as informações e remover informações desnecessárias, com o intuito de ajudar os desenvolvedores ao lidar com problemas complexos. A construção de *Models*, ou modelagem, se divide em três tipos principais (OPENLEARN, 2022):

- *Domain modelling* (modelagem de domínio): não limitada a apenas soluções de *softwares*, os modelos de domínio buscam representar informações provenientes do mundo real, como regras de negócio;
- *Specification modelling* (modelagem de especificação): utilizada para definir os elementos de *software* que serão utilizados na resolução do problema, e costuma focar nos serviços que a aplicação provê;
- *Design modelling* (modelagem de projeto): define o fluxo de controle da aplicação e como as responsabilidades de cada parte dela serão alocadas.

De forma análoga, no *software* proposto nesta monografia, os modelos de domínio são os dados tratados pela aplicação, que representam, por exemplo, os diversos setores e máquinas da indústria, os indicadores calculados e os eventos registrados. O modelo de especificação define a ideia da aplicação: gestão de paradas, com exibição de telas para visualização de indicadores de qualidade e análise temporal do funcionamento de máquinas. Já o modelo de projeto é toda a regra definida para a arquitetura da aplicação, que está sendo descrita neste capítulo.

4.2.8 Camada *Service* e Padrão *Repository*

Para separar a busca e inserção no banco de dados do processamento dos dados na aplicação, foram adotados dois padrões de arquitetura de *software*. O primeiro, chamado *Repository Pattern*, ou Padrão de Repositório, é um padrão que define um conjunto de classes e métodos especializados na interação com o banco de dados, encapsulando toda a lógica necessária para isso (MICROSOFT, 2022a).

Essencialmente, a *Repository* age como intermediário entre o mapeamento dos dados e o processamento deles, sendo o responsável por executar consultas no banco de dados e devolver respostas (FOWLER, 2012).

Já a camada de serviço, ou *Service Layer*, encapsula toda a regra de negócio da aplicação, que se divide em lógica de domínio (*domain logic*), responsável por resolução de problemas da aplicação, como cálculos financeiros, e lógica de aplicação (*application logic*), que define as responsabilidades da aplicação, como notificar usuários da aplicação sobre determinada informação (FOWLER, 2012).

4.2.9 Arquitetura REST

REST, sigla para *Representation State Transfer* é um padrão de arquitetura criado em 2000 por Roy Fielding, em sua tese de doutorado. De acordo com Fielding (2000), esse padrão provê um conjunto de regras e limitações para a transferência de dados que, ao serem todos aplicados, garantem escalabilidade de interação, generalidade, maior segurança e velocidade e independência no lançamento de sistemas.

Ainda segundo o autor, os componentes REST se comunicam pela transferência da representação de um recurso, que é uma sequência de *bytes* e um conjunto de metadados que descrevem esses *bytes*. Além disso, um dos requisitos do REST é que a comunicação seja *stateless*, que significa o não armazenamento, por parte do servidor, de dados relacionados ao contexto da troca de informações, sendo o cliente responsável por isso.

A vantagem dessa abordagem é a habilidade de o servidor liberar recursos rapidamente, uma vez que não precisa armazenar informações de estados entre requisições, além de promover maior facilidade para a recuperação de falhas. A desvantagem é a necessidade de constantemente inserir dados repetidos a cada requisição, já que os dados do estado não podem persistir no servidor, o que diminui o desempenho da transferência de dados.

Para o projeto desta monografia, a representação dos recursos é feita por meio do formato JSON por ser um padrão com suporte nativo ao uso em JavaScript, ter sua adoção quase onipresente, e possuir uma sintaxe facilmente legível (MASSÉ, 2011).

4.2.10 Arquitetura de 3 níveis lógicos

O tipo de arquitetura conhecido como multinível, ou arquitetura de n-níveis, é o padrão mais popular e utilizado no mercado, sendo conhecido por grande parte dos desenvolvedores, *designers* e arquitetos de *software*. Os níveis são organizados de forma a isolar responsabilidades internamente ao *software*, em que cada nível realiza apenas um tipo de tarefa. Não há um limite definido de número máximo ou mínimo de níveis, mas aplicações menores costumam se limitar a apenas três, enquanto que aplicações complexas podem vir a apresentar até cinco níveis diferentes (RICHARDS, 2015).

Fowler (2012) define 3 subdivisões principais para a arquitetura de sistemas:

1. *Presentation (apresentação)*: Nível responsável pela interação direta com o usuário. Costuma conter uma interface gráfica que envia e recebe dados de um servidor, e sua principal responsabilidade é exibição de informações para o usuário, e interpretar ações do usuário;

2. *Domain Logic* (lógica de domínio): Contém as regras de negócio da aplicação. Essencialmente, processamento de dados, validação e tomadas de decisão relacionadas aos modelos de domínio do *software* são realizados nela;

3. *Data Source* (fonte de dados): Utilizada para a comunicação com outros sistemas que executam tarefas para a aplicação, como bancos de dados e sistemas mensageiros.

Além disso, Fowler também afirma que os níveis *Domain* e *Data Source* nunca devem depender da *Presentation*, o que implica que lógicas da camada de apresentação jamais devem ser invocadas diretamente pelas outras camadas. Essa regra objetiva evitar que alterações visuais ou funcionais relacionadas à interação com o usuário impactem funções dos outros níveis.

Há diversas vantagens em se utilizar uma arquitetura em níveis (FOWLER, 2012):

- Torna possível o entendimento e trabalho em um nível específico sem a necessidade de uma grande compreensão do funcionamento de outros;
- Devido ao isolamento, é possível substituir implementações de um nível sem que isso afete os outros;
- Dependências entre responsabilidades são minimizadas;
- Há uma maior padronização do código e das regras;
- Um nível inferior pode ser utilizado por diversas camadas de nível mais alto.

Uma vez definidos os principais níveis da arquitetura, é natural, então, relacionar os padrões REST, *Service* e *Repository* com eles (YAGHINI, 2020), aplicando-nos em cada um dos níveis presentes:

1. *Presentation*: uma vez que lida com a transferência de dados entre *front-end* e *back-end*, a arquitetura REST e todo o *front-end* são aplicados neste nível;
2. *Domain Logic*: como esse nível lógico é focado em processamento de dados e regras de negócio, a camada *Service* faz parte da lógica de domínio. Além disso, o formato dos dados transferidos pela aplicação, representados pelos *Domain Models* numa estrutura de classes, residem aqui;
3. *Data Source*: o acesso ao banco e execução de *queries* presentes neste nível implicam que o padrão *Repository* deve ser aplicado nesta parte do sistema.

Dessa forma, é possível visualizar as subdivisões do *software* e sua relação com os padrões mencionados por meio da figura 4.3.

Apesar de a organização em níveis possuir as vantagens citadas anteriormente, também há algumas desvantagens, como citado por Fowler (2012):

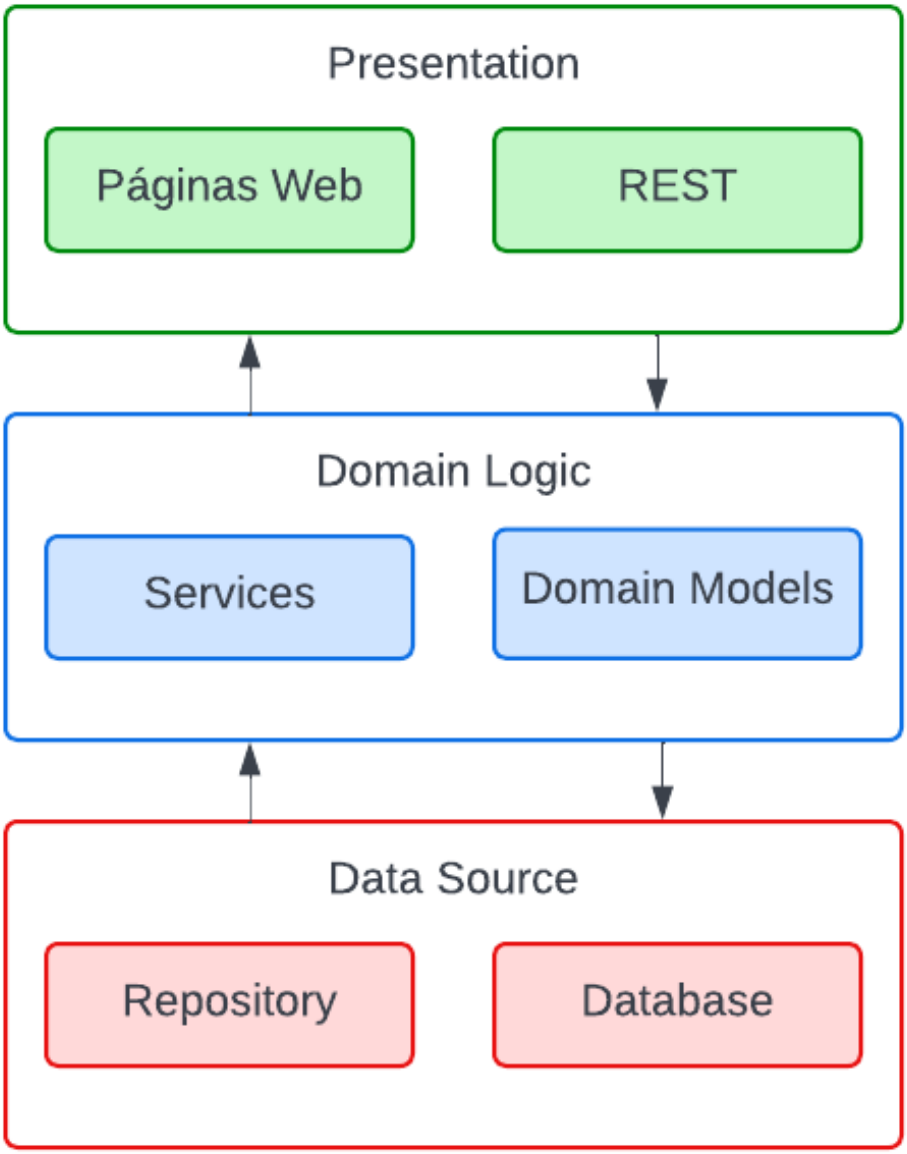


Figura 4.3: Estrutura de uma arquitetura de três camadas. *Fonte:* elaboração própria

- Apesar de haver grande isolamento entre os níveis, ele não é perfeito. O autor cita, por exemplo, a adição de um campo a ser exibido ao usuário. Isso pode implicar na necessidade de processamentos relacionados a ele, e, consequentemente, a adição dessa informação no banco de dados; processo este que afetaria todas os níveis da aplicação;

- Cada nível cria um *overhead*, ou carga extra de informações, na aplicação, pois os dados podem sofrer transformações a cada transição.

Outra possibilidade avaliada para a arquitetura do sistema foi a utilização de microsserviços, que, segundo Richards (2015), é uma arquitetura distribuída, fazendo com que cada parte da aplicação seja seu próprio processo independente. Isso traz algumas vantagens, como:

- Agilidade de desenvolvimento, pois o isolamento entre as partes da aplicação implica que mudanças em uma parte possuem menos chance de impactar outras partes se comparada a outras arquiteturas;

- Por cada aplicação ser uma parte individual do sistema, podem ser lançadas no servidor separadamente;

- Cada unidade do sistema pode ser facilmente testada, devido à independência relativa a outras partes;

- Muito escalável, já que cada parte pode ser escalada individualmente sem afetar outras.

Porém, há uma série de desvantagens nessa arquitetura, como: baixa performance devido à constante necessidade da troca de mensagens entre aplicações; maior número de *softwares* sendo executados, o que implica em maior cuidado com manutenção do sistema; e necessidade de disponibilidade de serviços remotos para autenticação e autorização.

Devido ao pequeno porte do sistema desenvolvido no projeto dessa monografia, uma arquitetura de três níveis se mostra suficiente, por ser uma divisão simples, robusta, organizada e muito centralizada.

4.2.11 SOLID

Na área da Engenharia de *Software*, o acrônimo SOLID é um mnemônico para representar princípios de orientação a objetos que auxiliam no desenvolvimento, flexibilidade e manutenção de códigos. Cunhado por Robert C. Martin, foi introduzido pela primeira vez no ano de 2000 em sua obra *Design Principles and Design Patterns*.

Segundo Martin et al. (2018), SOLID representa cinco princípios de projeto:

- **S**ingle Responsibility Principle (princípio da responsabilidade única);
- **O**pen-Closed Principle (princípio do aberto-fechado);
- **L**iskov Substitution Principle (princípio da substituição de Liskov);
- **I**nterface Segregation Principle (princípio da segregação de interfaces);
- **D**ependency Inversion Principle (princípio da inversão de dependência).

O principal objetivo da aplicação desses princípios é a criação de *softwares* de fácil compreensão, flexíveis a mudanças e que são a base de outros sistemas. Devido a isso, o sistema desenvolvido no projeto desta monografia seguiu rigorosamente a aplicação de cada um deles, que serão explicados a seguir.

Single Responsibility Principle

O primeiro princípio afirma que todo módulo, ou arquivo de código, do sistema deve ser responsável por apenas um ator. A palavra "ator" é utilizada pelo autor no sentido de grupo de usuários do sistema que demandam mudanças.

Martin exemplifica com uma classe chamada Empregado, que contém métodos para calcular pagamentos de uma empresa (usado pelo setor de finanças), gerir horas (usados pelo setor de recursos humanos) e salvar informações no banco de dados (usado pelos administradores do banco de dados, os DBAs). Se os métodos de cálculo de pagamentos e gestão de horas compartilharem um algoritmo que utilizam horas de trabalho não-extras, se o setor de finanças necessitar de um ajuste nesse algoritmo, inevitavelmente o setor de recursos humanos será afetado.

Outro problema surgiria se os DBAs necessitassem alterar o formato de dados sobre empregados no banco, ao mesmo tempo que um desenvolvedor precisasse alterar algo nos métodos da classe Empregado. Isso causaria um problema de mescla de alterações, o que pode gerar riscos para a aplicação e, por consequência, para o negócio.

A solução proposta é separar cada uma das responsabilidades em sua própria classe, e a classe do Empregado apenas armazena os dados de cada método de cada classe.

Open-Closed Principle

Este princípio implica na ideia de que um sistema deve permitir extensões, mas não mudanças de arquitetura. Martin afirma que, caso uma pequena ampliação do sistema implica em reformulações da arquitetura, é um sinal de que os arquitetos do sistema falharam ao projetá-lo.

Quando uma nova demanda de alteração do sistema surge, códigos antigos devem ser alterados o mínimo possível. A solução para este problema se encontra no uso de

interfaces ou classes abstratas, em que novos comportamentos podem ser invocados por meio de substituição com polimorfismo.

É possível exemplificar o problema da seguinte forma. Se queremos simular a habilidade "comunicar" de um animal, podemos criar uma classe "Animal" que recebe um tipo e possui um método que executa condicionalmente:

```
public enum TipoAnimal{
    Gato,
    Cachorro,
    Passaro
}

public class Animal {
    public TipoAnimal Tipo { get; set; }

    public void Comunicar() {
        if(Tipo == TipoAnimal.Gato)
            Console.WriteLine("Miado");
        else if(Tipo == TipoAnimal.Cachorro)
            Console.WriteLine("Latido");
        else if(Tipo == TipoAnimal.Passaro)
            Console.WriteLine("Canto");
    }
}
```

Código 4.1: Código em C♯ para representar uma violação do princípio aberto-fechado

Caso um dia seja necessário adicionar um novo animal, como "Boi", seria preciso modificar tanto a função "Comunicar" quanto a enumeração "TipoAnimal", o que viola o *open-closed principle*. Para garantir que o princípio seja seguido, podemos utilizar uma classe abstrata:

```
public abstract class Animal {
    public abstract void Comunicar();
}

public class Gato : Animal {
    public override void Comunicar() {
        Console.WriteLine("Miado");
    }
}

public class Cachorro : Animal {
```

```
    public override void Comunicar() {
        Console.WriteLine("Latido");
    }
}

public class Passaro : Animal {
    public override void Comunicar() {
        Console.WriteLine("Canto");
    }
}
```

Código 4.2: Código em C♯ que segue o princípio aberto-fechado

Dessa forma, para implementarmos um "Boi", basta criar uma nova classe:

```
public class Boi : Animal {
    public override void Comunicar() {
        Console.WriteLine("Mugido");
    }
}
```

Código 4.3: Código em C♯ da implementação da classe "Boi"

Utilizando polimorfismo, bastando instanciar um objeto do novo tipo na classe mãe "Animal", é possível invocar o método "Comunicar", que automaticamente o código executará o algoritmo respectivo da classe filha. Dessa forma, nenhum código antigo foi alterado, e o comportamento do sistema foi expandido.

Liskov Substitution Principle

De acordo com (LISKOV, 1987 apud MARTIN et al., 2018), este princípio implica, de forma resumida, que uma classe filha deve ser substituível pela sua classe pai sem perda de generalidade.

Martin cita um exemplo clássico de uma classe "Retângulo", que possui uma classe herdeira chamada "Quadrado".

```
public class Retangulo {
    private double _altura;
    private double _largura;

    public void DefinirAltura(double valor) {
        _altura = valor;
    }

    public void DefinirLargura(double valor) {
```

```
        _largura = valor;
    }

    public double CalcularArea() {
        return Altura * Largura;
    }
}

public class Quadrado : Retangulo {
    public void DefinirAltura(double valor) {
        _altura = valor;
        _largura = valor;
    }

    public void DefinirLargura(double valor) {
        _altura = valor;
        _largura = valor;
    }
}
```

Código 4.4: Código em C♯ de uma violação do princípio de substituição de Liskov

Uma vez que quadrados devem ter sempre altura e largura iguais, os métodos "DefinirAltura" e "DefinirLargura" automaticamente definem ambos os atributos de lado como iguais. Entretanto, em um código que utiliza polimorfismo, os resultados podem ser diferentes dependendo de qual classe foi utilizada:

```
Retangulo retangulo = new Retangulo();
retangulo.DefinirAltura(5);
retangulo.DefinirLargura(10);
Console.WriteLine(retangulo.CalcularArea()); // Produz 50

Retangulo retangulo = new Quadrado();
retangulo.DefinirAltura(5);
retangulo.DefinirLargura(10);
Console.WriteLine(retangulo.CalcularArea()); // Produz 100
```

Código 4.5: Código em C♯ de uma violação do princípio de substituição de Liskov

Isso ocorre porque quadrados não possuem o mesmo comportamento de retângulos, não sendo uma substituição correta. Dessa forma, o programador precisa sempre se preocupar em garantir que a classe correta será utilizada para evitar que resultados incorretos sejam exibidos. Isso gera condicionais extras e um aumento da complexidade do código.

Uma forma de solucionar o problema seria, ao invés de utilizar "Retangulo" como classe pai, criar uma nova classe chamada "Forma", contendo apenas um método para cálculo de área, cujas filhas são "Retangulo" e "Quadrado". Dessa forma, cada filha implementa o cálculo de área seguindo suas próprias regras e, para alterar as dimensões da forma, os utilizadores devem utilizar as classes concretas diretamente, tornando possível assumir o comportamento do cálculo de área.

Interface Segregation Principle

Em algumas linguagens estaticamente tipadas, como Java, cada classe é compilada em seu próprio pacote. Portanto, dependências diretas entre classes podem gerar recompilações desnecessárias, como exemplificado a seguir. Supondo que as Classes 1 e 2 dependam da classe Operações, como mostra a figura 4.4, caso haja qualquer modificação na última, as primeiras deverão ser recompiladas.

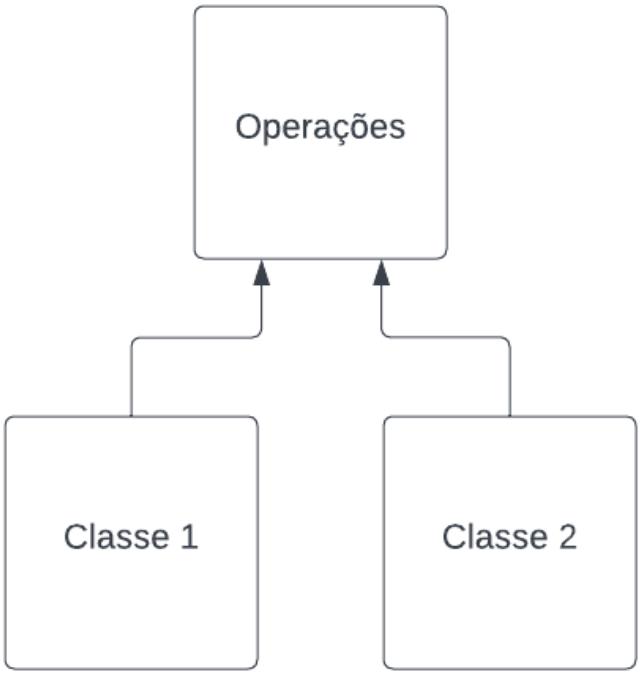


Figura 4.4: Estrutura de dependências diretas entre classes. *Fonte:* elaboração própria

Numa nova situação, implementando interfaces para cada uma das Classes 1 e 2, como mostra a figura 4.5, esse problema é evitado.

Entretanto, este problema como um todo não ocorre em C#, portanto, este princípio não é uma preocupação de projeto.

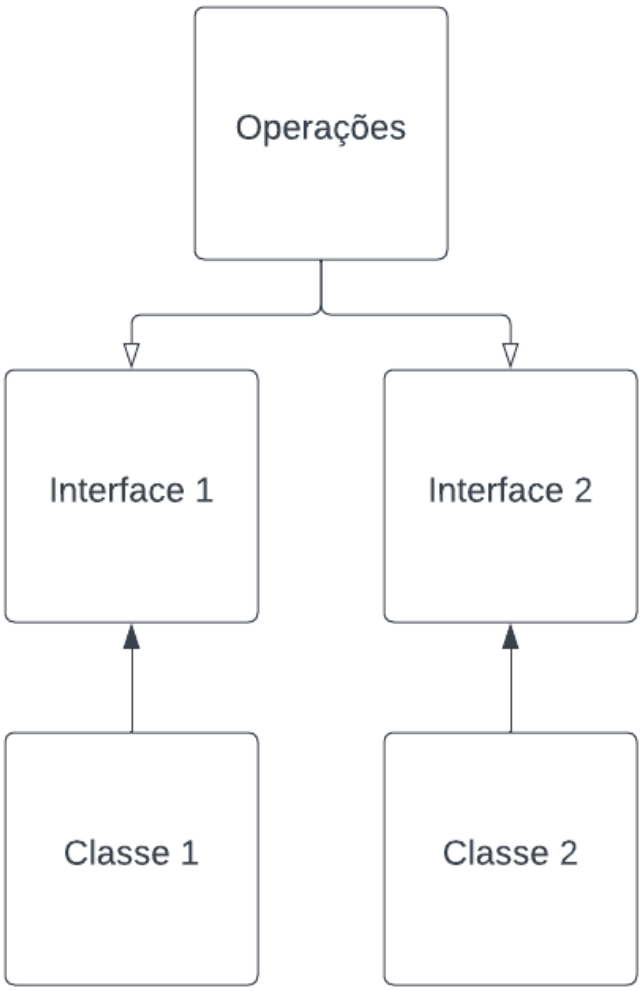


Figura 4.5: Estrutura de dependências entre classes e interfaces. *Fonte:* elaboração própria

Dependency Inversion Principle

O último princípio do acrônimo SOLID é referente a um conceito conhecido como inversão de dependência. Martin et al. (2018) afirma que o princípio diz que sistemas flexíveis não utilizam códigos que dependem de implementações, mas sim de abstrações. Na prática, isso é inviável em algumas situações, mas é uma boa prática tentar aplicar o princípio sempre que possível.

Isso se justifica pelo fato de que mudanças numa interface sempre implicam em alterações na implementação, mas a recíproca nem sempre é verdadeira. Devido a isso, interfaces apresentam característica menos dependente em relação ao mundo externo, sendo mais estáveis.

É possível aplicar este princípio por meio de diversos padrões e técnicas, entre eles injeção de dependência, que é uma técnica que permite ser possível invocar métodos por meio de interfaces (LARKIN; SMITH; DAHLER, 2022), e *Factory Pattern*, ou padrão de fábrica, que define uma interface para criação de objetos, mas deixa as subclasses executarem os métodos (Refactoring Guru, 2022).

4.3 Arquitetura do banco de dados

Uma vez definida a arquitetura do sistema, é necessário definir a estrutura dos dados que serão armazenados no banco e estabelecer as relações lógicas entre eles. Dessa forma, foram elencados 7 tipos de entidades diferentes, que serão mapeados em tabelas no banco de dados:

- Máquinas: cada máquina da empresa deve ter suas próprias informações, como nome e identificador. Sua tabela se chama *machines*;
- Categorias de máquinas: de forma a resumir todo o comportamento de um conjunto de máquinas de um mesmo tipo, é importante agrupá-las em categorias. Por isso, há uma tabela *machine_ categories* para isso;
- Eventos: cada evento de funcionamento ou parada deve ser armazenado para posterior cálculo de indicadores de desempenho, e para análise no gráfico de Pareto;
- Motivos de evento: cada evento possui alguma justificativa, seja para dizer que a máquina está funcionando, seja para explicar um motivo de parada. Por isso, é criada uma tabela contendo esses motivos, chamada *event_ reasons*. Ela permite análises do gráfico de Pareto, bem como obtenção de percentuais, tempos totais e tempos médio, para cada justificativa individualmente;
- Indicadores de desempenho: os diversos indicadores discutidos previamente se baseiam no tempo que a máquina não funcionou e no tempo de produção ideal. Seria

possível calculá-los sob demanda para cada vez que alguém desejasse visualizá-los. Entretanto, isso seria um processo oneroso, uma vez que uma máquina, ou categoria de máquina, pode vir a ter centenas, em alguns casos milhares, de interrupções em poucas horas. Então, buscas contínuas e cálculos sob demanda poderiam gerar quedas de desempenho do sistema, principalmente pelo fato de que dados históricos seriam sempre iguais. Por isso, optou-se por realizar os cálculos e salvar os resultados no banco de dados, atualizando-nos quando ocorrer alterações em eventos antigos ou chegada de eventos novos. A tabela dos indicadores se chama *kpis*;

- Tipos de indicadores: pelo fato de que cada indicador é apenas um valor após uma série de cálculos, é necessário ter uma forma de controlar o que os valores significam. Por isso, se ele equivale a um cálculo de OEE, MTTR, MTBF, ou outro, é salvo em uma tabela específica, *kpi_types*, e referenciado pela tabela de indicadores;

- Turnos de trabalho: para garantir maior poder de análise do processo, foram separados os turnos de trabalho da empresa, de forma que os KPIs e os eventos podem ser analisados por turno, também. Por isso, as informações de turno são salvas numa tabela chamada *work_shifts*.

O diagrama do banco e de suas relações de chaves é mostrado na figura 4.6.
Para melhor análise da figura, é necessário afirmar alguns pontos:

- Toda máquina possui uma categoria, havendo uma relação de chave estrangeira *machines.category_id->machine_categories.id*;

- Cada evento possui data de início e fim, e, por consequência, um tempo total. Além disso, todo evento é de um turno específico, possui um motivo e pertence a uma máquina. Logo, há três relações de chave estrangeira:

  1. *events.work_shift_id->work_shifts.id*;
  2. *events.event_reason_id->event_reasons.id*;
  3. *events.machine_id->machines.id*.

- Cada indicador possui um valor e um tipo, uma data de referência para dizer qual dia, mês ou ano esse indicador equivale, e pode ser referente a uma máquina ou a uma categoria de máquina. Além disso, pode ser referente a um turno específico ou não. Dessa forma, há quatro relações de chave estrangeira:

  1. *kpis.work_shift_id->work_shifts.id*;
  2. *kpis.kpi_type_id->kpi_types.id*;
  3. *kpis.machine_id->machines.id*;
  4. *kpis.machine_category_id->machine_categories.id*.

Figura 4.6: Diagrama do banco de dados. *Fonte:* elaboração própria

- Os motivos de evento podem ser motivos de funcionamento, ou motivos de parada, representado pela variável do tipo *bool is_stoppage*, que representa se o motivo é de parada. Similarmente, o motivo pode ser de uma parada para manutenção ou não, representado pela variável *is_maintenance* do tipo *bool*, que representa se é um motivo de manutenção. Esse controle de variáveis verdadeiro/falso serve para facilitar o cálculo dos indicadores.

Os campos *name* e *description* das tabelas são meramente para visualização por parte do usuário ao se exibir as informações na tela.

4.4 Implementação

Uma vez que todos os tópicos de arquitetura foram bem definidos, pôde-se dar início ao desenvolvimento do projeto. Os passos seguidos foram:

1. Primeiramente, foi realizada uma visita à empresa cliente para conhecimento do processo e dos procedimentos utilizados para análise de dados;

2. Foram discutidos os pontos focais de necessidade do sistema, quais demandas ele deveria atender e qual o público alvo. Dessa forma, o escopo do projeto foi levantado, as demandas foram priorizadas e as telas foram desenhadas;

3. Estudou-se os diversos conceitos apresentados relacionados à gestão de manutenção, paradas e produtividade, como indicadores de desempenho e tipos de gráfico (por exemplo, Pareto);

4. A arquitetura do projeto foi debatida, idealizada, experimentada e repensada, até que se chegasse num patamar de aceitação geral por parte da equipe da empresa fornecedora (Vitau *Automation*);

5. O banco de dados foi projetado, com as necessidades iniciais de representação de dados pensadas e descritas num *script* SQL;

6. Deu-se início ao desenvolvimento do código, sempre seguindo todo o padrão pré-definido. O *back-end* foi desenvolvido simultaneamente ao *front-end*;

8. Foi feita, então, a integração com o banco de dados do cliente para recebimento dos dados reais do processo;

9. Diversas reuniões com o cliente foram realizadas no decorrer do desenvolvimento do código para avaliação, recebimento de sugestões e melhorias, e para debater novas ideias;

10. Com os algoritmos e telas implementadas, foram feitas diversas validações, por meio de comparação de dados, com as planilhas do cliente;

11. O código foi lançado num servidor em nuvem, e disponibilizado para testes por parte do cliente. O acesso é realizado por meio de um endereço *web* que leva à plataforma. Antes de utilizar os recursos, o usuário deve se autenticar numa página de *login*. A estrutura e algoritmos do sistema de autenticação são externos à plataforma e não são abordados pelo projeto desta monografia;

12. Após a realização desses testes e aprovação do cliente, foi realizada uma nova visita à empresa para treinamento dos utilizadores.

Capítulo 5

Desenvolvimento

Uma vez estabelecidos todos os passos a serem seguidos para a realização do projeto, seu desenvolvimento pôde ser iniciado. Após as visitas ao cliente e conhecimento do processo e da coleta de dados atual, foram levantadas as seguintes necessidades por parte do cliente:

- Tabela para visualização de todos os eventos de cada máquina, com datas de início e fim, turno e tempo total;
- Gráfico de Pareto para resumir os principais motivos de parada dos equipamentos, permitindo uma maior análise das principais causas de problemas na empresa;
- Gráficos de OEE, permitindo análise por máquina ou categoria, bem como visualizações separadas em ano, mês, dia e turno;
- Gráficos de manutenção, de forma a ser possível ver os indicadores MTTR, MTBF e Disponibilidade Inerente;
- Um painel para visualização resumida de todos os dados de indicadores simultaneamente de forma tabular, permitindo comparação direta de desempenho entre ativos;
- Uma página para resumir informações sobre um ativo e uma categoria em um determinado intervalo de tempo.

Uma vez definidos esses pontos de necessidade, foram esboçadas as páginas presentes na plataforma, apresentadas a seguir.

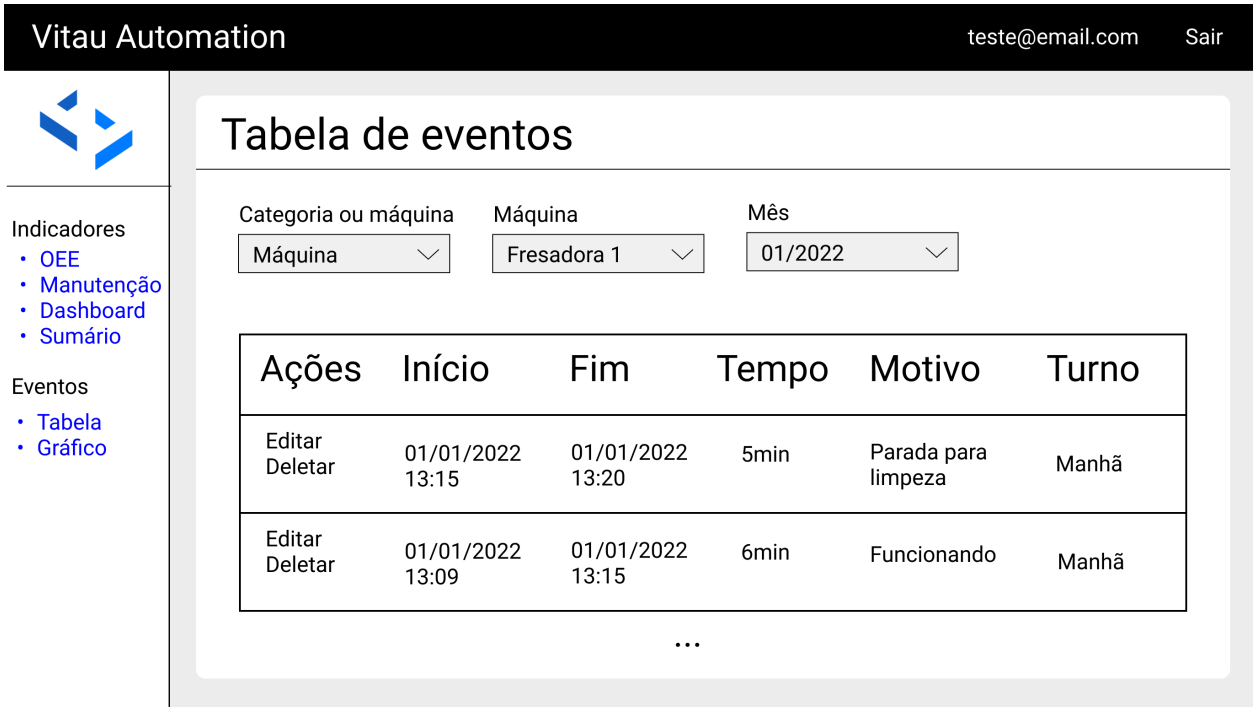


Figura 5.1: Esboço de página da tabela de eventos. *Fonte:* elaboração própria

5.1 Esboço das páginas

5.1.1 Tabela de eventos

Nesta página, serão exibidas as informações sobre cada evento individual, organizados por máquina, similarmente a como era feito na planilha do cliente. Dessa forma, é necessário exibir informações sobre datas e horas de início e fim do evento, tempo total, o motivo de apontamento e qual o turno do evento.

Além disso, é necessário permitir edição manual de eventos, para os casos de haver alguma inconsistência dos dados proveniente do *software* ERP, ou se houver algum apontamento de motivo incorreto por parte do operador. Também é preciso permitir a deleção de eventos para o caso de sinais espúrios provenientes dos sensores de coleta, de forma que o gestor remova o evento espúrio e expanda o evento real prévio ou posterior.

O esboço da página pode ser visto na figura 5.1.

5.1.2 Diagrama de Pareto

Aqui, é possível visualizar os motivos de parada de cada máquina em determinado mês no formato de um diagrama de Pareto. Dessa forma, a gestão da empresa consegue facilmente localizar os principais motivos que estão gerando gargalos no processo, permitindo melhor tomada de decisão para a redução dos tempos improdutivos gastos.

O esboço da página pode ser visto na figura 5.2. Nela, o eixo horizontal representa os

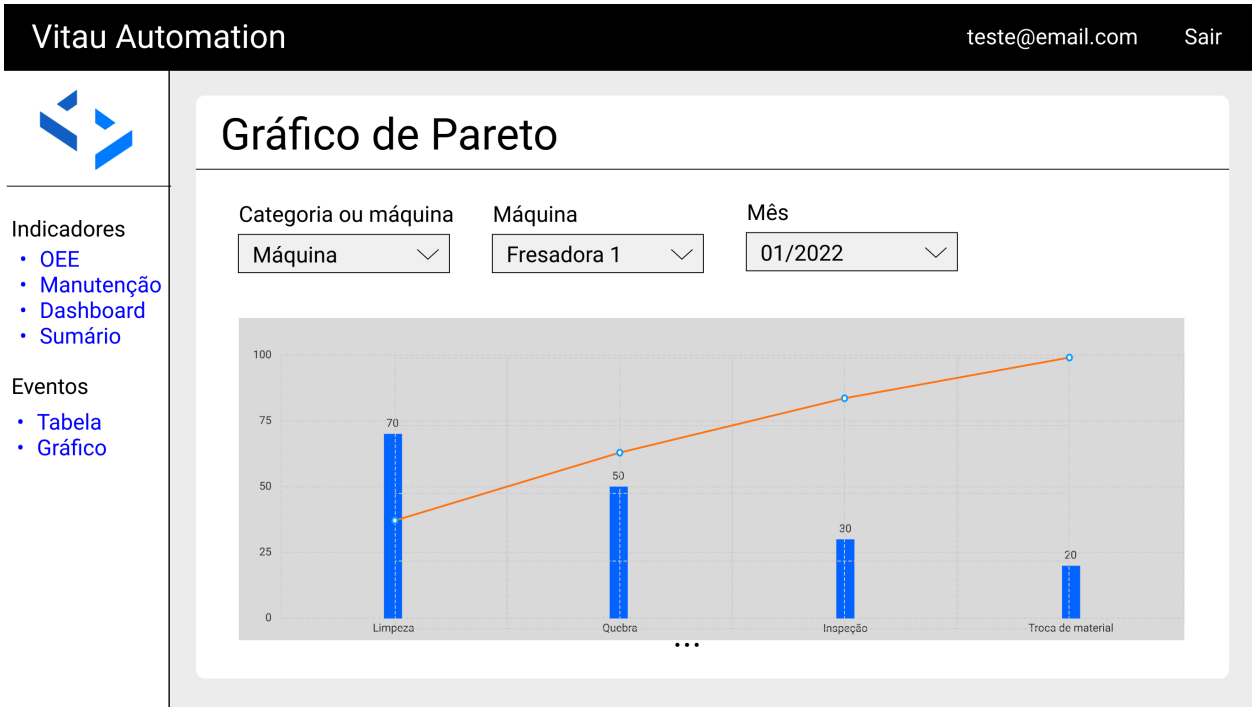


Figura 5.2: Esboço de página do diagrama de Pareto. *Fonte:* elaboração própria

motivos de parada das máquinas, e o eixo vertical o tempo em horas de parada para cada motivo.

5.1.3 Gráficos de OEE

Para a visualização dos indicadores de OEE, uma página contendo gráficos e diversos critérios de seleção foi esboçada. Dessa forma, é possível que o usuário selecione se deseja visualizar a OEE por categoria de máquinas ou por máquina individual, e por ano, mês ou dia.

O esboço da página pode ser visto na figura 5.3. O eixo horizontal representa os meses do ano, e o vertical a porcentagem de OEE para cada mês.

5.1.4 Gráficos de manutenção

Similarmente aos gráficos de OEE, esta página exibe informações dos indicadores MTTR, MTBF e Disponibilidade Inerente separadas por categoria ou por ativo, além de por mês. Nesta página não faz sentido permitir a visualização por dia, uma vez que, muitas vezes, os tempos de reparo ou de quebra são muito maiores que 24 horas.

O esboço da página pode ser visto na figura 5.4. Nela, as barras vermelhas equivalem ao MTTR, as azuis ao MTBF, e a linha acima das barras à Disponibilidade Inerente.

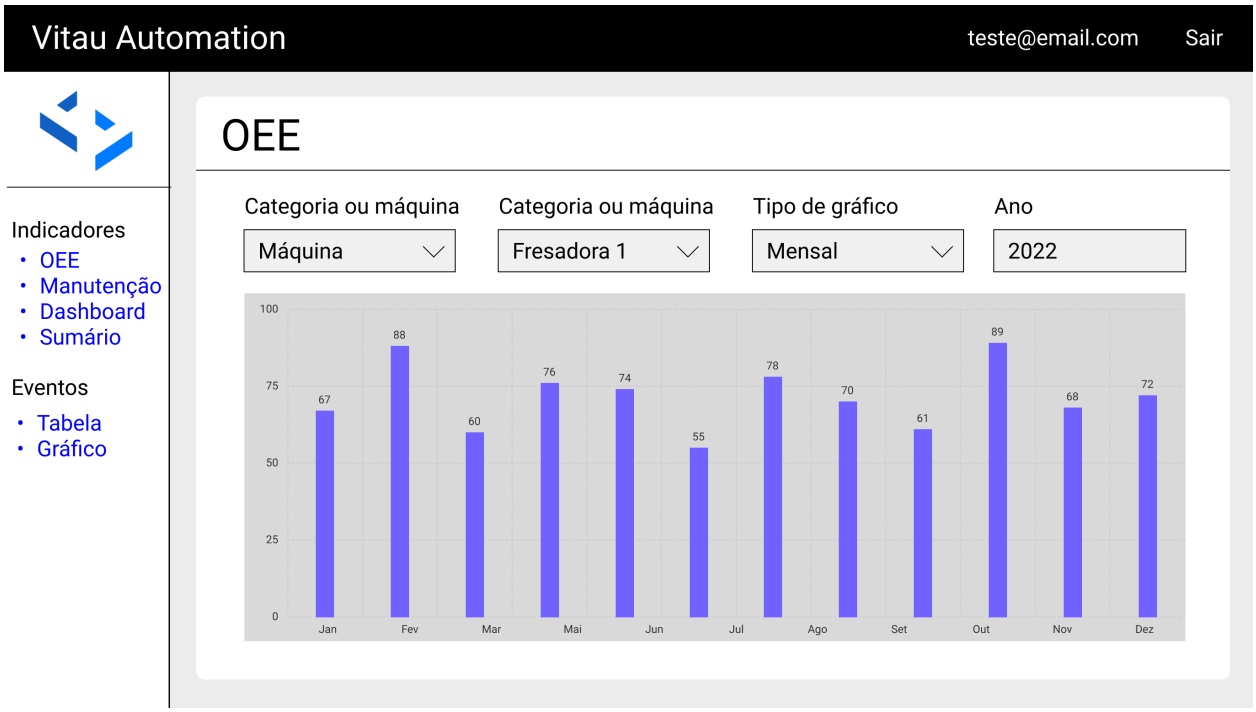


Figura 5.3: Esboço de página contendo gráficos de OEE. *Fonte:* elaboração própria

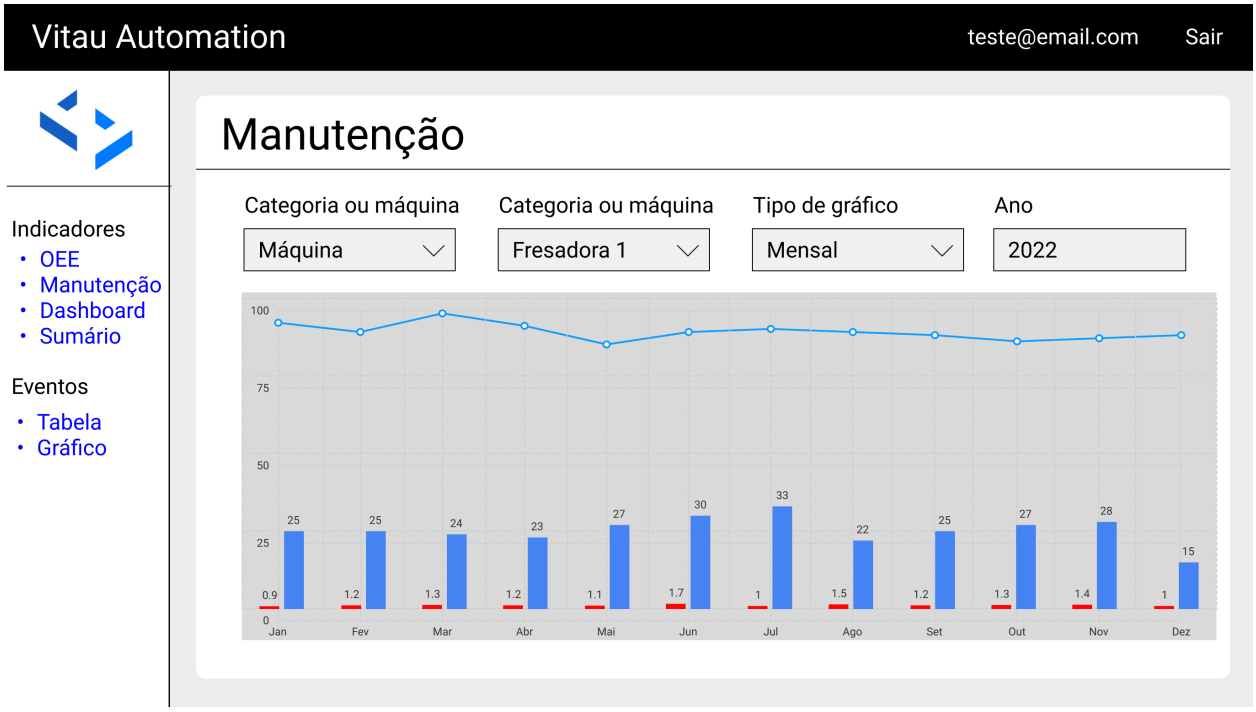


Figura 5.4: Esboço de página contendo gráficos de manutenção. *Fonte:* elaboração própria

5.1.5 *Dashboard*

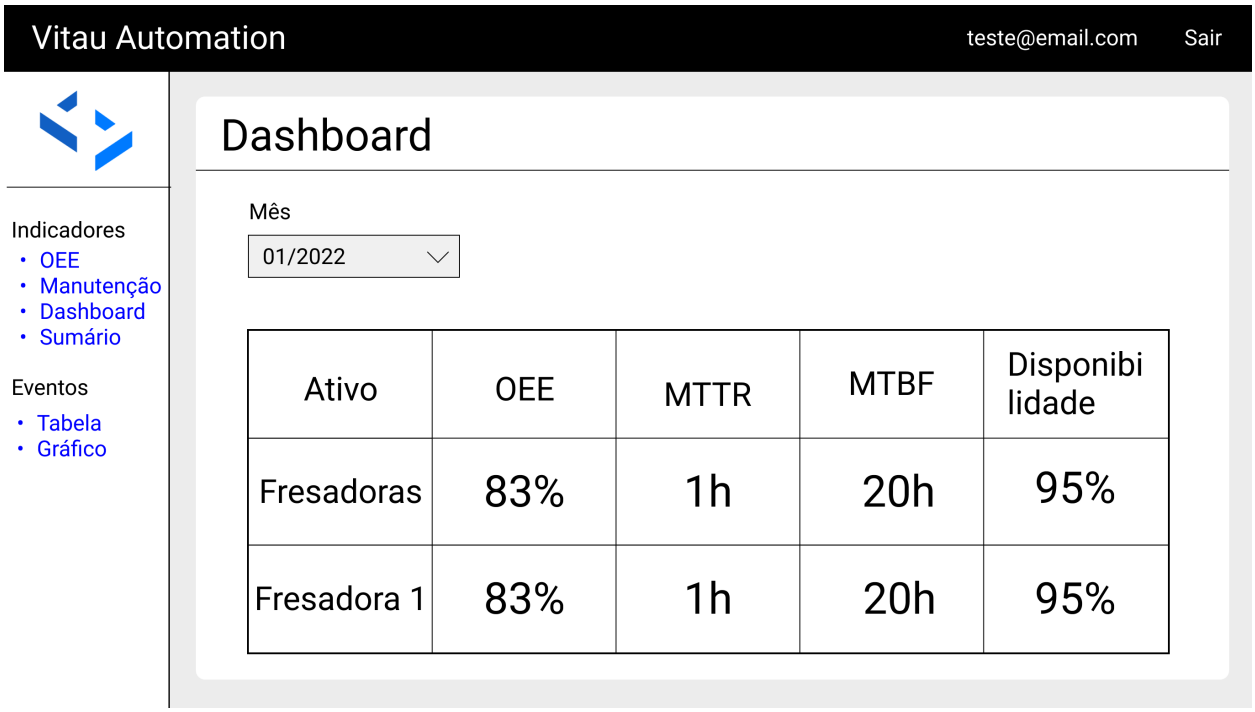


Figura 5.5: Esboço de página contendo um painel de indicadores. *Fonte:* elaboração própria

De forma a se obter um comparativo entre os diversos ativos da empresa, sejam máquinas, ou categorias, e encontrar *outliers*, foi esboçada uma página contendo um painel tabular que exibe informações sobre diversos indicadores simultaneamente.

Primeiramente, a categoria é exibida numa linha, e logo abaixo dela, todos os seus ativos são mostrados, cada um com seus indicadores de desempenho. Além disso, é possível especificar qual mês será visualizado.

O esboço da página pode ser visto na figura 5.5.

5.1.6 Página de resumo

A última página referente aos indicadores é a de resumo. Nela, há um componente em formato de velocímetro que mostra a eficiência global da categoria selecionada no período especificado, variando de 0 a 100%, com subdivisões categorizando valores ruins (vermelho), medianos (laranja) e bons (verde).

Além disso, também há seções textuais que resumem a máquina com pior OEE no período e seu respectivo valor, além de mostrar os indicadores de manutenção e o tempo total que a máquina esteve parada.

O esboço da página pode ser visto na figura 5.6.

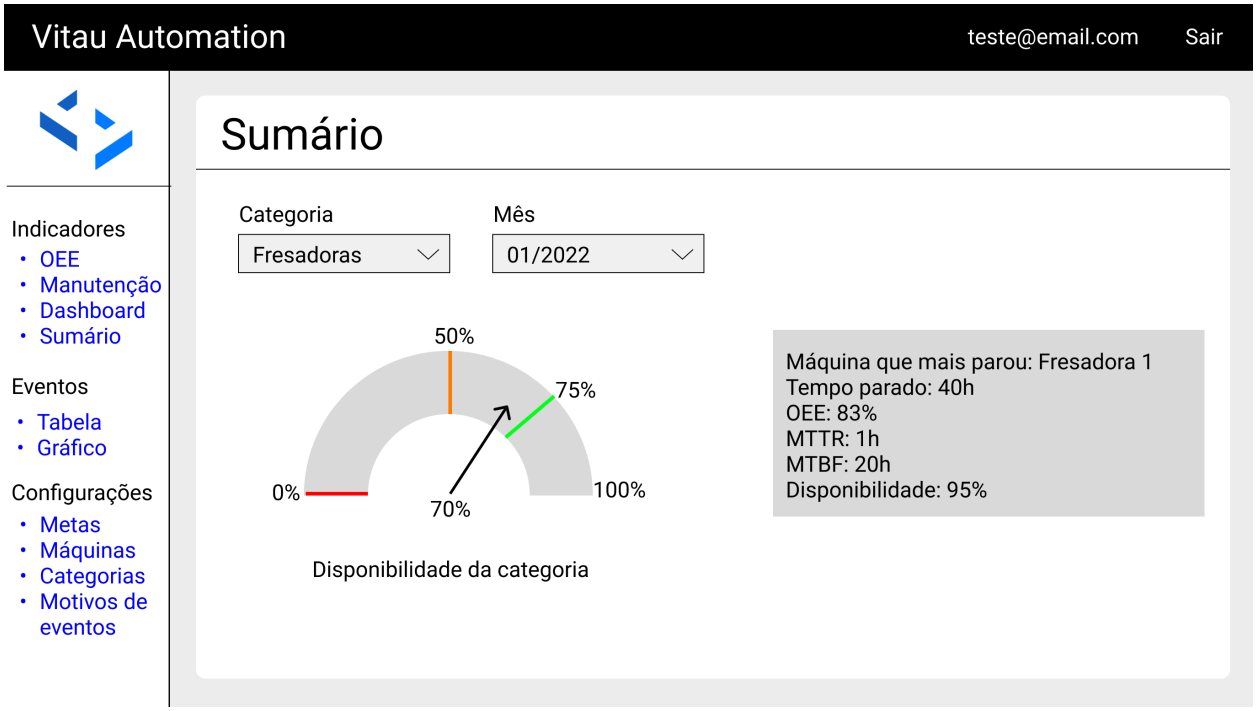


Figura 5.6: Esboço de página de resumo dos indicadores. *Fonte:* elaboração própria

5.2 Obtenção dos dados

Os dados provenientes do cliente são coletados por meio de um *software* independente da plataforma principal, proprietário da Vitau Automation, mas também feito como parte do projeto para a ferramentaria. Este programa se conecta diretamente ao banco de dados do cliente, busca os eventos registrados de cada ativo, gera uma mensagem JSON contendo todos os dados obtidos e a envia para uma fila do RabbitMQ.

Ao se utilizar filas, o *software* consumidor consegue processar as mensagens de acordo com sua própria disponibilidade, buscando elas na fila quando puder. Um envio direto de mensagens entre os *softwares* poderia gerar atrasos no processo caso o consumo de uma mensagem dependesse do final do processamento da mensagem anterior. Em contrapartida, caso as mensagens fossem armazenadas no consumidor enquanto ele processa mensagens anteriores, isso causaria consumo elevado de memória do consumidor se a velocidade de armazenamento fosse maior que a de consumo.

Em seguida, o sistema da plataforma, em uma *thread* dedicada a isso, capta a mensagem da fila, converte o JSON para o formato do *Model* de eventos, e insere os dados convertidos no banco por meio do *Entity Framework Core*, além de usá-los para calcular KPIs. Então, insere, ou atualiza, uma variável no Redis que informa qual foi o identificador (ID) do último evento inserido para cada máquina individual. Essa informação é usada, então, pelo *software* que busca dados diretamente do cliente, que só buscará eventos contendo um ID maior que o disponível no Redis, dessa forma evitando que toda

a base de dados já processada seja buscada novamente. O uso do Redis neste ponto do fluxo tem objetivo de agilizar as atualizações de informação no banco de dados, uma vez que ele utiliza a memória RAM para isso, e não o disco. Realizar essa operação num banco relacional como o PostgreSQL seria muito mais lento.

Este processo pode ser representado pelo diagrama na figura 5.7. Seguindo a numeração da figura, é possível exemplificar o fluxo cíclico executado na obtenção dos dados:

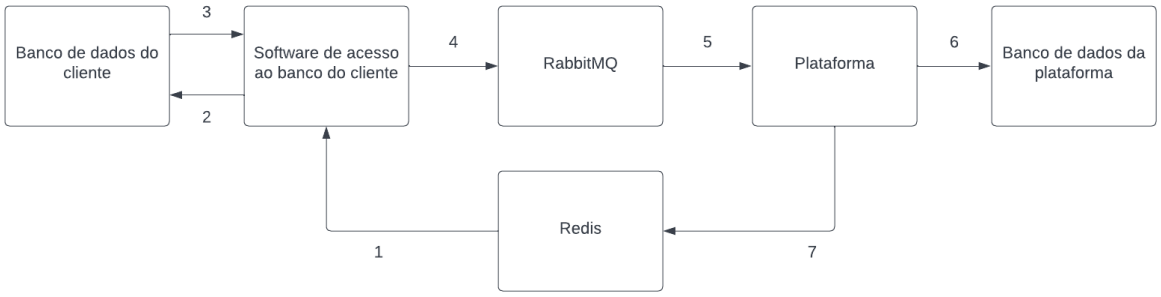


Figura 5.7: Diagrama representativo da obtenção de dados de eventos. *Fonte:* elaboração própria

1. O *software* que lê informações do banco do cliente checa qual o último ID inserido no Redis, equivalente ao último evento captado e inserido no banco de dados da plataforma;
2. O *software*, então, realizada uma busca dos dados no banco;
3. O banco devolve os dados solicitados, seguindo os critérios de ID especificados;
4. Os dados são convertidos em JSON e enviados ao RabbitMQ;
5. A plataforma recebe os dados do RabbitMQ e converte em *Models* de eventos;
6. Os dados convertidos são utilizados para o cálculo de indicadores de desempenho e são inseridos no banco da plataforma;
7. O ID do último evento inserido de cada máquina é atualizado no Redis.

Esse fluxo é executado de forma cíclica a cada 5 minutos. Dessa forma, é possível obter as informações sobre o desempenho das máquinas praticamente em tempo real.

5.3 Código

Após estabelecido o formato de cada uma das páginas por meio de esboços, deu-se início ao desenvolvimento do código. O *back-end* e o *front-end* foram desenvolvidos

concorrentemente, de forma que o funcionamento conjunto pudesse ser testado a todo momento.

Seguindo os padrões de arquitetura definidos previamente, estabeleceu-se que cada uma das tabelas do banco de dados equivaleria a um *Model*, num formato de classe, permitindo-se, assim, trabalhar com o *Entity Framework Core*. Além disso, cada *Model* possuiria uma classe para *Service*, e outra para *Repository*, separando suas responsabilidades de forma bem organizada e garantindo um fluxo padronizado para cada dado a ser tratado.

Por exemplo, a tabela *events* do banco de dados possui uma classe *Model* chamada *Event*, uma classe *EventService*, equivalente à sua *Service*, e *EventRepository*, que representa sua *Repository*. O fluxo de transmissão de dados sempre segue o padrão de três camadas, em que o dado é captado pela camada *Presentation* por meio de uma rota, transferido para a *EventService* para processamento e validação, que então é encaminhado para a *EventRepository* para que seja feita uma inserção ou busca no banco.

Todas as mensagens trocadas entre *back-end* e *front-end* foram definidas seguindo as regras *REST*, no formato JSON, exemplificado na figura 5.8. Esta mensagem mostra o formato de uma solicitação do usuário para atualização de um evento, gerada e enviada pelo *front-end*.


```
{
    "machineId": 1,
    "eventReasonId": 10,
    "workShiftId": 1,
    "datetimeStart": "2022-01-01T13:15:00.000Z",
    "datetimeEnd": "2022-01-01T13:20:00.000Z",
    "id": 1
}
```


Figura 5.8: Mensagem em formato JSON. *Fonte:* elaboração própria

5.4 Resumo do capítulo

Durante o desenvolvimento, foram levantadas as necessidades do cliente, e então esboçadas as páginas *web* da plataforma. Após isso, foram desenvolvidos os códigos para processamento de dados.

Também foi feito um *software* para a coleta de dados no banco do cliente. O *back-end* e o *frontend* da aplicação principal foram desenvolvidos simultaneamente, e a arquitetura definida previamente foi seguida, em que os códigos seguem o fluxo em três camadas de forma estrita.

Capítulo 6

Resultados

Após o desenvolvimento de todas as páginas propostas nos esboços e do *software* de coleta de dados, o sistema foi implementado na indústria cliente, e os indicadores de todos os dados históricos presentes no banco dela foram calculados. As figuras nas seções subsequentes apresentam dados ilustrativos do resultado obtido.

A arquitetura de sistema proposta foi implementada com sucesso, garantindo, assim, um fluxo de dados padronizado e um código de simples desenvolvimento e entendimento, assumindo que o desenvolvedor conheça a teoria por trás dos princípios de arquitetura.

6.1 Páginas *Web*

6.1.1 Tabela de eventos

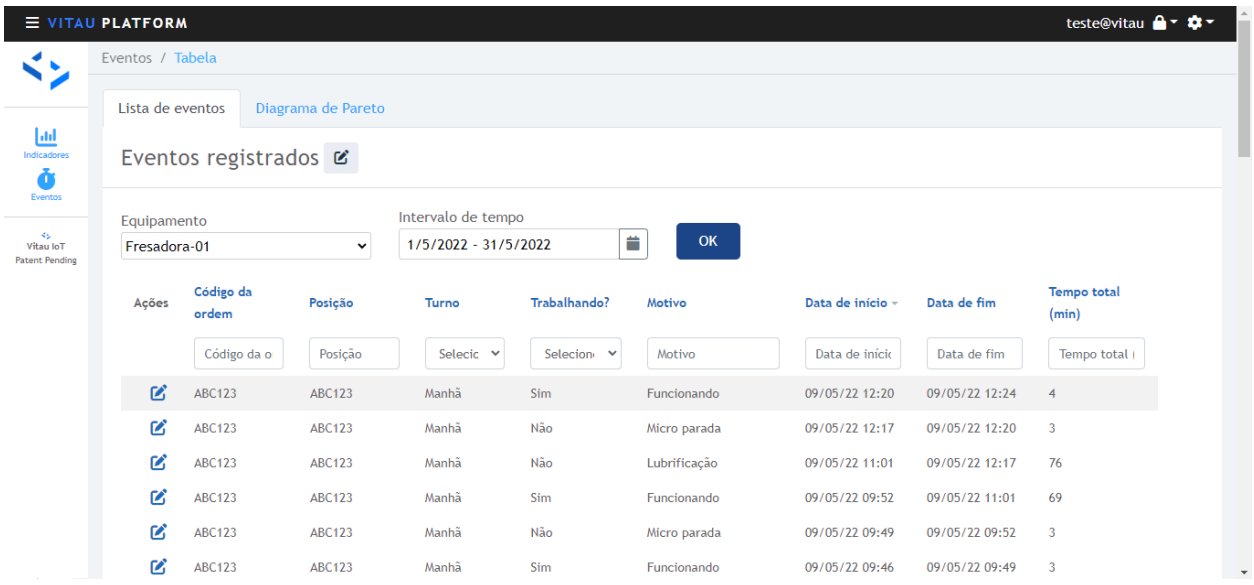
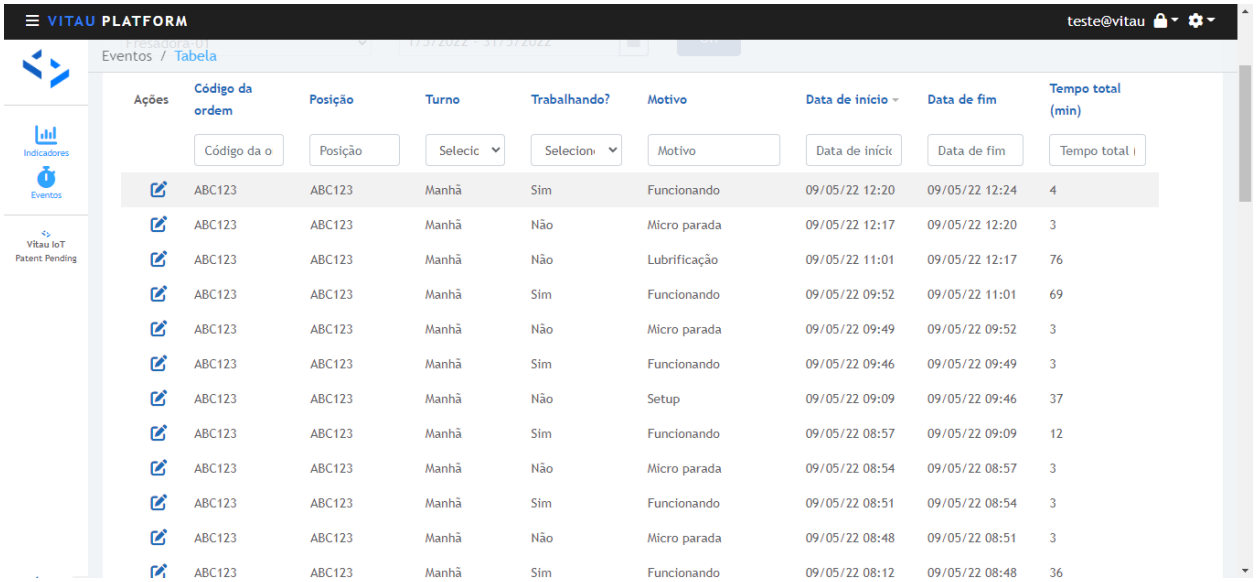


Figura 6.1: Página da tabela de eventos. *Fonte:* Vitau Automation (2022)



Ações	Código da ordem	Posição	Turno	Trabalhando?	Motivo	Data de início	Data de fim	Tempo total (min)
	Código da o	Posição	Selecic	Selecion	Motivo	Data de início	Data de fim	Tempo total
	ABC123	ABC123	Manhã	Sim	Funcionando	09/05/22 12:20	09/05/22 12:24	4
	ABC123	ABC123	Manhã	Não	Micro parada	09/05/22 12:17	09/05/22 12:20	3
	ABC123	ABC123	Manhã	Não	Lubrificação	09/05/22 11:01	09/05/22 12:17	76
	ABC123	ABC123	Manhã	Sim	Funcionando	09/05/22 09:52	09/05/22 11:01	69
	ABC123	ABC123	Manhã	Não	Micro parada	09/05/22 09:49	09/05/22 09:52	3
	ABC123	ABC123	Manhã	Sim	Funcionando	09/05/22 09:46	09/05/22 09:49	3
	ABC123	ABC123	Manhã	Não	Setup	09/05/22 09:09	09/05/22 09:46	37
	ABC123	ABC123	Manhã	Sim	Funcionando	09/05/22 08:57	09/05/22 09:09	12
	ABC123	ABC123	Manhã	Não	Micro parada	09/05/22 08:54	09/05/22 08:57	3
	ABC123	ABC123	Manhã	Sim	Funcionando	09/05/22 08:51	09/05/22 08:54	3
	ABC123	ABC123	Manhã	Não	Micro parada	09/05/22 08:48	09/05/22 08:51	3
	ABC123	ABC123	Manhã	Sim	Funcionando	09/05/22 08:12	09/05/22 08:48	36

Figura 6.2: Página da tabela de eventos. *Fonte:* Vitau Automation (2022)

Na página da tabela de eventos, é possível, além de visualizar todos os eventos de cada máquina, ordenar e filtrar os dados na própria tabela, por meio das ferramentas disponíveis no cabeçalho dela.

Há algumas diferenças em relação aos esboços traçados previamente: há colunas de ordem de serviço e posição, que correspondem, respectivamente, ao código da ordem que a máquina estava trabalhando no momento em que o evento foi registrado, e a posição exata da peça que estava sendo usinada. Além disso, há uma coluna que informa se o evento é de parada ou de trabalho.

Outra diferença se encontra no fato de não haver possibilidade de deleção direta de eventos, de forma a não permitir o comprometimento do histórico de forma arbitrária.

O resultado final da página pode ser visto nas figuras 6.1 e 6.2.

6.1.2 Diagrama de Pareto

No diagrama de Pareto disponível na página, é possível definir o intervalo de tempo desejado para se analisar a quantidade de horas paradas, separadas por motivo, bem como o impacto percentual na eficiência global do equipamento ou categoria no período. Essas porcentagens não são exibidas em sua forma acumulada, o que normalmente ocorre em diagramas de Pareto, mas a informação fornecida se mantém igual a caso estivesse em seu formato tradicional.

É possível, também, escolher a quantidade de motivos de parada exibidos simultaneamente, uma vez que, em casos que há muitos motivos de parada diferentes, a visualização do gráfico pode ser prejudicada. Por isso, o padrão foi definido como até 5 motivos, permitindo a extensão até a quantidade de razões total que há no intervalo.

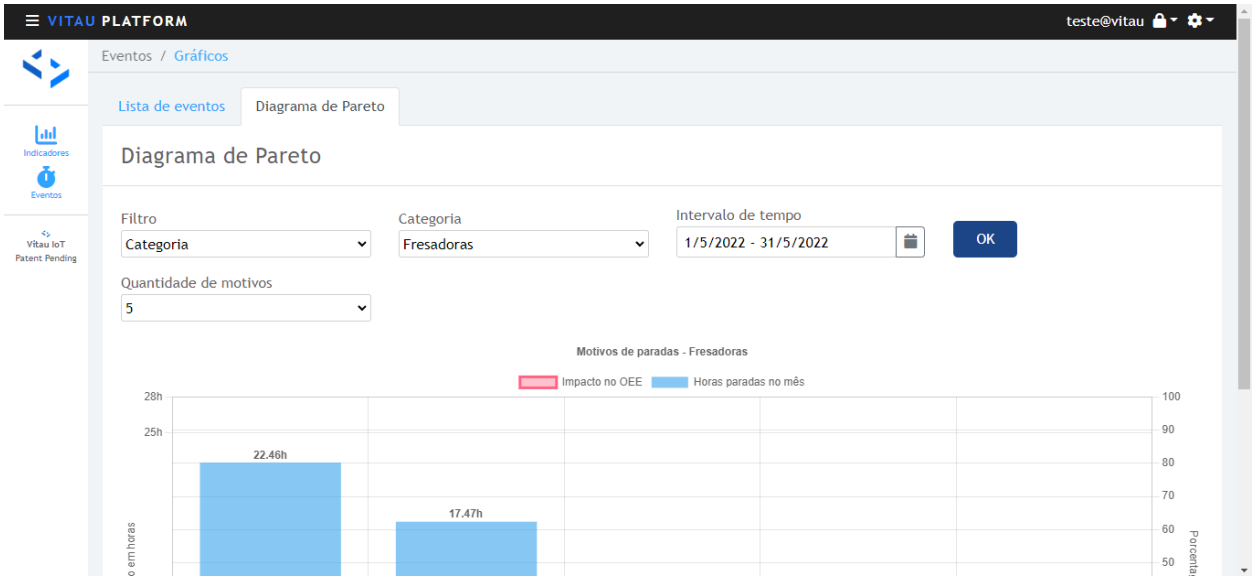


Figura 6.3: Página do diagrama de Pareto. *Fonte:* Vitau Automation (2022)

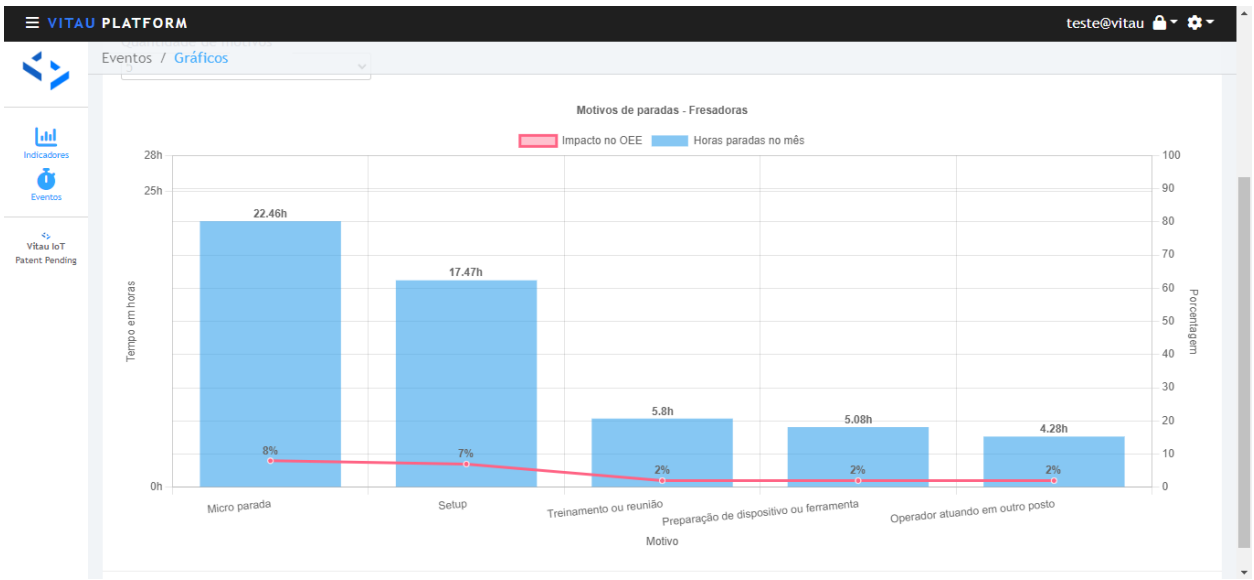


Figura 6.4: Página do diagrama de Pareto. *Fonte:* Vitau Automation (2022)

O resultado final da página pode ser visto nas figuras 6.3 e 6.4.

6.1.3 Gráficos de OEE

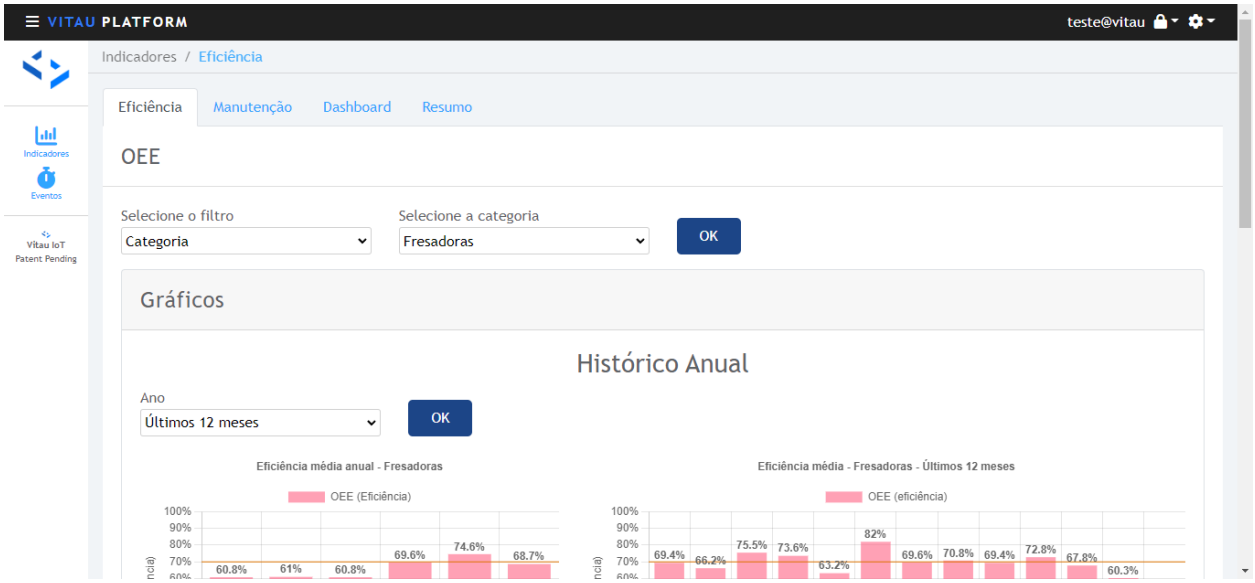


Figura 6.5: Cabeçalho da página dos gráficos de OEE. *Fonte:* Vitau Automation (2022)

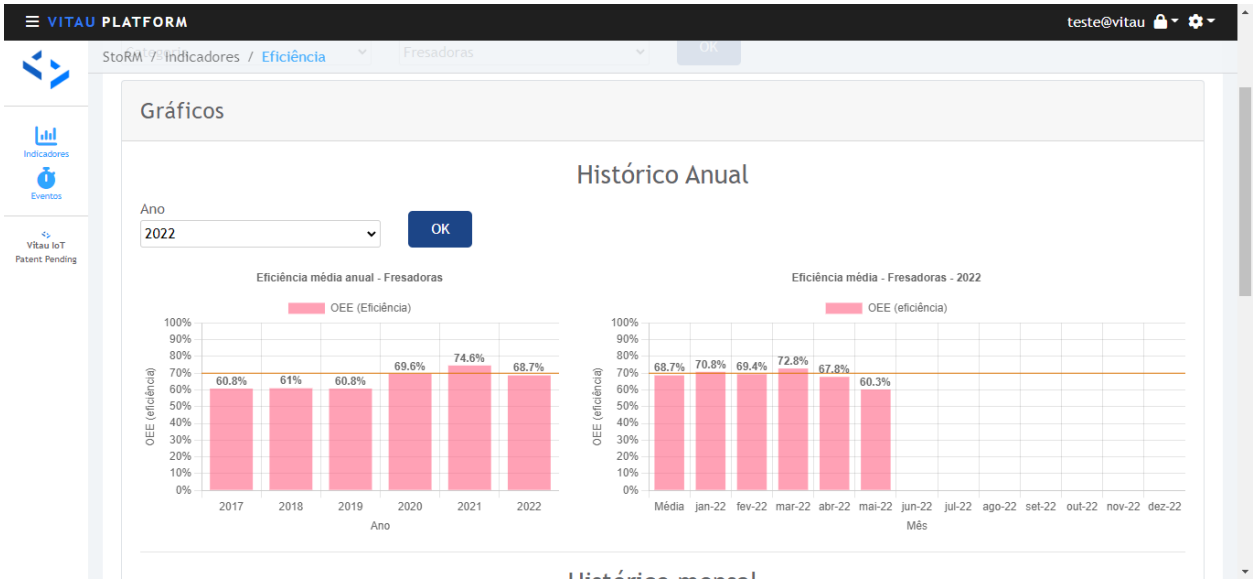


Figura 6.6: Gráficos anuais e mensais de OEE. *Fonte:* Vitau Automation (2022)

A página das informações de OEE contém diversos gráficos simultaneamente, que podem ser modificados para exibir dados sobre uma categoria de máquinas ou uma máquina individual.

Figura 6.7: Gráfico de OEE diária. *Fonte:* Vitau Automation (2022)

Figura 6.8: Gráfico de OEE diária separada por turnos. *Fonte:* Vitau Automation (2022)

Os dois primeiros gráficos, visíveis na figura 6.6, são anuais e mensais. Eles informam, respectivamente, a OEE de cada ano dos últimos 5 anos, e a OEE de cada mês no ano selecionado.

O terceiro gráfico, presente na figura 6.7, mostra a OEE diária, permitindo a seleção de um mês específico. Por último, o quarto gráfico, na figura 6.8, mostra, também, a OEE diária, mas separada em turnos.

Há também um traço laranja em cada gráfico, que indica a meta de OEE da fábrica, valor definido pelo cliente.

6.1.4 Gráficos de manutenção

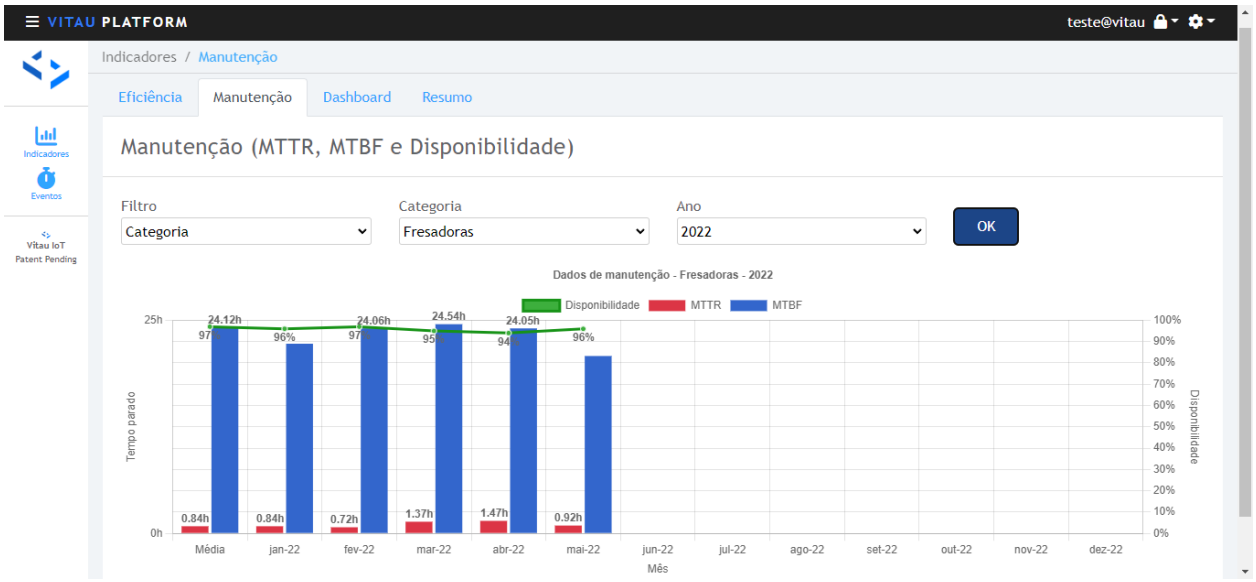


Figura 6.9: Gráficos de manutenção de uma categoria. *Fonte:* Vitau Automation (2022)

O gráfico desta página exibe os indicadores MTTR, MTBF e Disponibilidade Inerente. Similarmente às outras páginas, é possível visualizar os dados de uma categoria, como mostra a figura 6.9, ou de um ativo específico, figura 6.10.

É possível, também, selecionar o ano para visualizar os dados.

6.1.5 *Dashboard*

O *dashboard* mostra informações de OEE do mês totais e separadas por turno, bem como os indicadores de manutenção.

Durante o desenvolvimento do projeto, também surgiu a necessidade de se incluir metas no *dashboard*, de forma a permitir comparar as informações dos indicadores com os valores mínimos ideais. A página pode ser vista na figura 6.11.

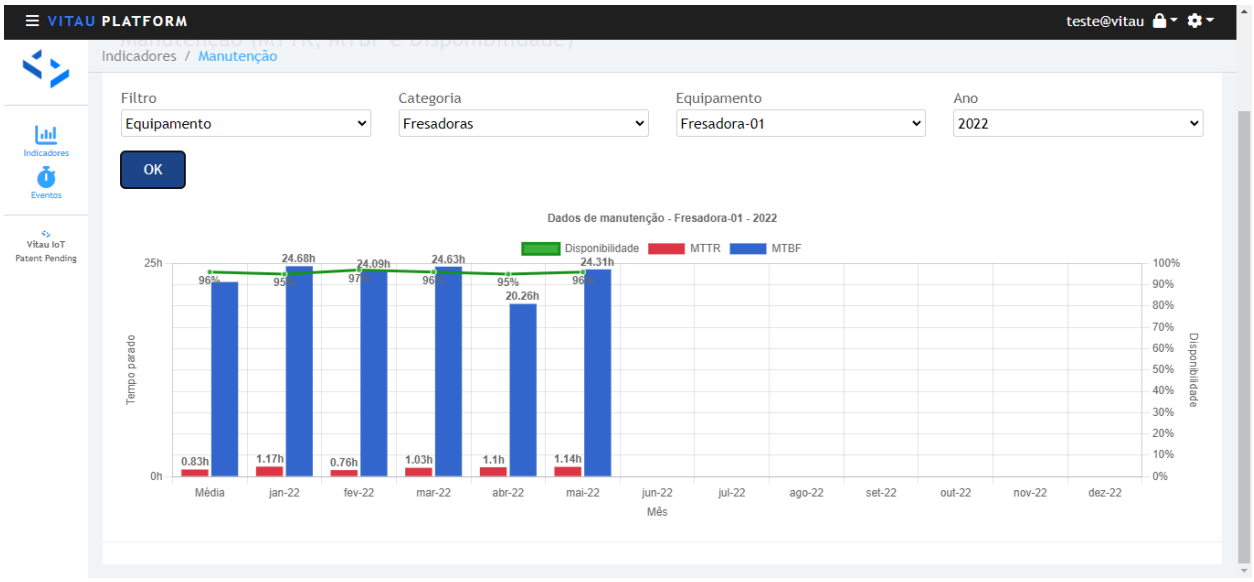


Figura 6.10: Gráficos de manutenção de uma máquina específica. *Fonte:* Vitau Automation (2022)

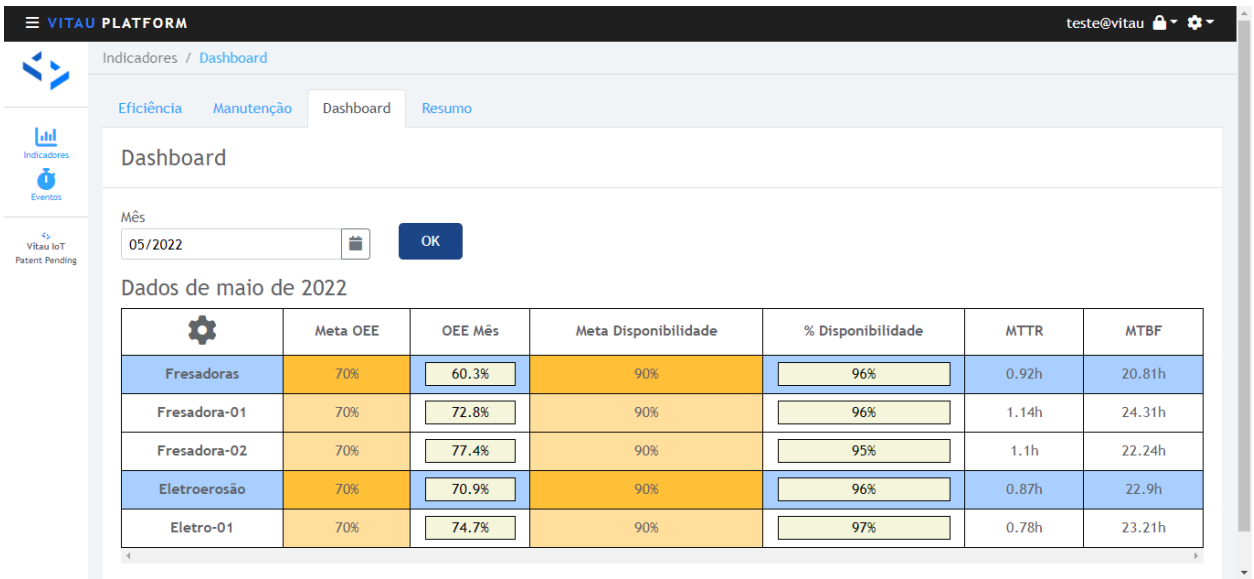


Figura 6.11: Painel de indicadores. *Fonte:* Vitau Automation (2022)

6.1.6 Página de resumo

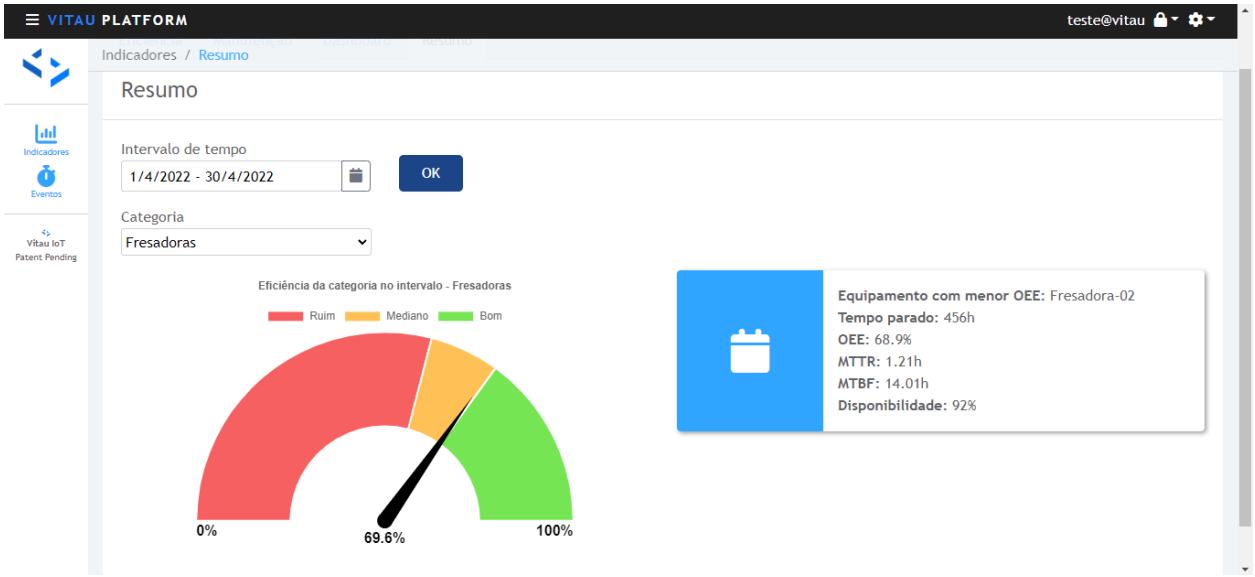


Figura 6.12: Página de resumo dos indicadores. *Fonte:* Vitau Automation (2022)

Por fim, a página de sumário apresenta forma similar à do esboço feito no início do projeto. Há um seletor de período, um gráfico em formato de velocímetro para indicar a OEE da categoria selecionada e um cartão com textos indicando informações e indicadores do ativo com menor eficiência global no intervalo.

A página pode ser visualizada na figura 6.12.

6.2 Resumo do capítulo

Todas as páginas propostas foram desenvolvidas, algumas com modificações em relação aos esboços devido a demandas por parte do cliente. No total, seis páginas foram criadas:

- Tabela de eventos, permitindo a visualização em intervalos;
- Diagrama de Pareto para motivos de eventos em determinado período;
- Gráficos de OEE anual, mensal e diários, em que os diários são separados por turnos;
- Gráficos de indicadores de manutenção, exibindo MTTR, MTBF e Disponibilidade numa visão mensal;
- *Dashboard*, página que resume todos os indicadores de cada máquina e de cada categoria simultaneamente para determinado mês;

- Resumo, página que mostra informações de gargalo para a categoria escolhida, exibindo indicadores da máquina com menor desempenho, bem como motivos de parada mais comuns e a eficiência da categoria.

Com todos as páginas desenvolvidas, o sistema foi implementado na empresa cliente.

Capítulo 7

Conclusões

7.1 Considerações finais

Após finalizado o projeto e implementado na indústria cliente, é possível afirmar que ele atingiu todos os objetivos propostos. As páginas *web* disponíveis permitem uma análise ampla de todos os indicadores mencionados (OEE, MTTR, MTBF e Disponibilidade Inerente), além de permitir uma visualização dos principais problemas da indústria cliente por meio de um Diagrama de Pareto.

Além disso, todos os cálculos com eventos, antes realizados manualmente em planilhas, agora são automatizados, e o histórico de eventos se torna disponível para consulta de forma simples e completa, sem a necessidade de uma busca pelos arquivos de planilha de um período específico. Todos esses pontos garantem uma melhoria na eficiência da gestão da indústria, que agora não necessita mais gastar tanto tempo com essas tarefas manuais, podendo direcionar seus esforços a outras áreas.

Por fim, a arquitetura do sistema seguiu todos os critérios especificados, garantindo um formato de simples desenvolvimento, escalável e fácil de ser entendido uma vez que o desenvolvedor conheça a estrutura teórica por trás.

7.2 Propostas de continuidade

Apesar de o sistema desenvolvido já garantir melhoria na qualidade da gestão, ainda é possível aprimorá-lo. Algumas possíveis expansões seriam:

- Coleta de relatórios de melhoria de desempenho da fábrica após um período de uso do sistema, de forma a concluir, de forma quantitativa, os benefícios trazidos pela plataforma;
- Adicionar funções para impressão dos gráficos gerados, pois os gestores, em diversas situações, optam por armazenar cópias dos dados de forma física para uso em

situações que recursos virtuais não estão presentes;

- Implementar metodologias para realização de testes do *software*, como testes de integração e testes unitários. Dessa forma, é possível garantir o bom funcionamento da integração entre as diversas partes do sistema e o fluxo correto dos dados.

Referências Bibliográficas

ABRAMAN. *Documento Nacional: A situação da manutenção no Brasil*. [S.l.], 2017.

ABUHAKMEH, K. *Basics of Entity Framework Core*. JetBrains, 2020. Acesso em: 30-05-2022. Disponível em: <https://www.jetbrains.com/dotnet/guide/tutorials/basics/entity-framework-core/>.

ALCOFORADO, F. *Globalização*. [S.l.]: NBL Editora, 1997.

ALMEANAZEL, O. T. R. Total productive maintenance review and overall equipment effectiveness measurement. *Jordan Journal of Mechanical and Industrial Engineering*, Department of Industrial Engineering, Hashemite University Zarqa, Jordan, v. 4, n. 4, 2010.

ANGULAR TEAM. *What is Angular?* 2022. Disponível em: <https://angular.io/guide/what-is-angular>.

BUSSO, C. M.; MIYAKE, D. I. Análise da aplicação de indicadores alternativos ao overall equipment effectiveness (oee) na gestão do desempenho global de uma fábrica. *Production*, SciELO Brasil, v. 23, n. 2, p. 205–225, 2013.

CAMARGO, R. *Diagrama de Pareto: o que é e quando você deve usá-lo?* 2018. Acesso em: 01-06-2022. Disponível em: <https://robsoncamargo.com.br/blog/Diagrama-de-Pareto-o-que-e-e-quando-voce-deve-usa-lo>.

CARVALHO, B. A. de; FERNANDES, L. A.; CÔBERO, C. Sistema de informação para o gerenciamento das paradas de produção em uma cervejaria. 2012. Acesso em: 22-01-2022. Disponível em: <https://www.aedb.br/seget/arquivos/artigos12/38016353.pdf>.

CCV INDUSTRIAL. *O que é CNC?* 2017. Acesso em: 01-06-2022. Disponível em: <https://ccvindustrial.com.br/o-que-e-cnc/>.

CHAND, G.; SHIRVANI, B. Implementation of tpm in cellular manufacture. *Journal of materials Processing technology*, Elsevier, v. 103, n. 1, p. 149–154, 2000.

CIMM. *Processos de Usinagem*. 2022. Acesso em: 28-05-2022. Disponível em: <https://www.cimm.com.br/portal/material_didatico/3350-sistemas-e-processos-de-fabricacao>.

DEACON, J. *Model-View-Controller (MVC) Architecture*. 2009. Acesso em: 30-05-2022. Disponível em: <https://web.archive.org/web/20161020010917/http://www.battersea-locksmith.co.uk/briefings/MVC.pdf>.

EROMINAS. *Ferramentaria: tudo o que você precisa saber sobre isso*. 2019. Acesso em: 28-05-2022. Disponível em: <https://www.erominas.com.br/usinagem/ferramentaria-tudo-o-que-voce-precisa-saber-sobre-isso>.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. [S.l.]: University of California, Irvine, 2000.

FOGLIATO, F.; RIBEIRO, J. L. D. *Confiabilidade e manutenção industrial*. [S.l.]: Elsevier Brasil, 2009.

FOLDOC. *back-end*. 1996. Acesso em: 29-05-2022. Disponível em: <http://foldoc.org/backend>.

FOLDOC. *front-end*. s.d. Acesso em: 29-05-2022. Disponível em: <http://foldoc.org/frontend>.

FOWLER, M. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*. [S.l.]: Addison-Wesley, 2012.

FREITAS, V. W. B. d. Indicadores de desempenho para o gerenciamento de manutenções em grandes paradas preventivas. Universidade Federal Rural do Semi-Árido, 2019. Acesso em: 22-01-2022. Disponível em: <https://repositorio.ufersa.edu.br/bitstream/prefix/4586/1/V\%c3\%adncentteWBF_MONO.pdf>.

FUENTES, F. F. E. et al. Metodologia para inovação da gestão de manutenção industrial. Florianópolis, SC, 2006.

GAWER, A.; CUSUMANO, M. A. et al. *Platform leadership: How Intel, Microsoft, and Cisco drive industry innovation*. [S.l.]: Harvard Business School Press Boston, 2002. v. 5.

JOHANSSON, L. *Part 1: RabbitMQ for beginners - What is RabbitMQ?* RabbitMQ, 2019. Acesso em: 30-05-2022. Disponível em: <https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html>.

KARDEC, A.; NASCIF, J. A. *Manutenção: função estratégica*. 3. ed. [S.l.]: Qualitymark, 2009.

KAUER, M. 2019. Acesso em: 28-05-2022. Disponível em: <https://pixabay.com/pt/photos/m%C3%A1quina-ferramenta-plaina-ind%C3%BAstria-3985455/>.

KWON, O.; LEE, H. Calculation methodology for contributive managerial effect by oee as a result of tpm activities. *Journal of quality in Maintenance Engineering*, Emerald Group Publishing Limited, 2004.

LARKIN, K.; SMITH, S.; DAHLER, B. *Dependency injection in ASP.NET Core*. 2022. Acesso em: 30-05-2022. Disponível em: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0>.

Leonard G. 2006. Acesso em: 28-05-2022. Disponível em: <https://commons.wikimedia.org/wiki/File:EDMWorkpiece.jpg>.

LISKOV, B. Keynote address-data abstraction and hierarchy. In: *Addendum to the proceedings on Object-oriented programming systems, languages and applications (Addendum)*. [S.l.: s.n.], 1987. p. 17–34.

MARTIN, R. C. et al. *Clean architecture: a craftsman's guide to software structure and design*. [S.l.]: Prentice Hall, 2018.

MASSÉ, M. *REST API design rulebook: designing consistent RESTful web service interfaces*. [S.l.]: O'Reilly Media, Inc., 2011.

MICROSOFT. *Design the infrastructure persistence layer*. 2022. Acesso em: 30-05-2022. Disponível em: <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design>.

MICROSOFT. *What is ASP.NET Core?* 2022. Acesso em: 29-05-2022. Disponível em: <https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core>.

MICROSOFT. *What is .NET? Introduction and overview*. 2022. Acesso em: 29-05-2022. Disponível em: <https://docs.microsoft.com/en-us/dotnet/core/introduction>.

MIKOWSKI, M.; POWELL, J. *Single page web applications: JavaScript end-to-end*. [S.l.]: Simon and Schuster, 2013.

NETO, J. C. da S.; LIMA, A. Gonçalves de. Implantação do controle de manutenção. *Revista Club de Mantenimiento*, n. 10, 2002.

OPENLEARN. *Modelling and the UML*. 2022. Acesso em: 30-05-2022. Disponível em: <https://www.open.edu/openlearn/science-maths-technology/introduction-software-development/content-section-6>.

PIXABAY. 2016. Acesso em: 28-05-2022. Disponível em: <https://pixabay.com/photos/milling-machining-industry-1151358/>.

PIXABAY. 2016. Acesso em: 28-05-2022. Disponível em: <https://pixabay.com/pt/photos/cnc-fresamento-girando-usinagem-1270101/>.

PLUS, T. *PostgreSQL vs SQL Server - A quick database comparison*. 2018. Acesso em: 30-05-2022. Disponível em: <https://tableplus.com/blog/2018/10/postgresql-vs-sql-server-database-comparison.html>.

POPOV, M. *Reading metrics dashboard in Superset*. 2020. Acesso em: 26-05-2022. Disponível em: <https://commons.wikimedia.org/wiki/File:Reading_metrics_dashboard_in_Superset.png>.

POSTGRESQL. *What is PostgreSQL?* 2022. Acesso em: 30-05-2022. Disponível em: <https://www.postgresql.org/about/>.

POWELL, T.; SAMMUT-BONNICI, T. Pareto analysis. John Wiley & Sons, 2014.

PRECISMEC. 2021. Acesso em: 28-05-2022. Disponível em: <https://precismec.com.br/o-que-e-o-processo-de-usinagem/>.

RALPH. 2018. Acesso em: 28-05-2022. Disponível em: <https://pixabay.com/photos/milling-cutters-metal-shavings-tool-3738903/>.

RALPH. 2019. Acesso em: 28-05-2022. Disponível em: <https://pixabay.com/photos/lathe-metal-processing-machine-4021097/>.

REDIS. *Introduction to Redis*. 2022. Acesso em: 30-05-2022. Disponível em: <https://redis.io/docs/about/>.

REDIS. *Redis FAQ - What's the Redis memory footprint?* 2022. Acesso em: 30-05-2022. Disponível em: <https://redis.io/docs/getting-started/faq/#whats-the-redis-memory-footprint>.

REENSKAUG, T. M. H. *MVC*. 1979. Acesso em: 30-05-2022. Disponível em: <http://wayback.archive-it.org/10370/20180425071111/http://folk.uio.no/trygver/themes/mvc/mvc-index.html>.

Refactoring Guru. *Factory Method em C#*. 2022. Acesso em: 30-05-2022. Disponível em: <https://refactoring.guru/pt-br/design-patterns/factory-method/csharp/example>.

RENO, G. et al. *Sistema de Monitoramento de Paradas de Máquina em uma Linha de Usinagem–Um Estudo de Caso*. ENEGEP, 2010. Acesso em: 22-01-2022. Disponível em: <http://www.ingepro.com.br/Publ_2011/Abr/438\%20pg\%2097-108.pdf>.

RIBEIRO, A. de F. Taylorismo, fordismo e toyotismo. *Lutas Sociais*, v. 19, n. 35, p. 65–79, 2015.

RICHARDS, M. *Software architecture patterns*. [S.l.]: O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA ..., 2015. v. 4.

SAKS, E. Javascript frameworks: Angular vs react vs vue. 2019.

SCHWAB, K. *A quarta revolução industrial*. [S.l.]: Edipro, 2019.

SCHWARZENBERGER, M. 2014. Acesso em: 28-05-2022. Disponível em: <https://pixabay.com/photos/drill-milling-milling-machine-tool-444510/>.

SILVA, J. da. Oee–a forma de medir a eficácia dos equipamentos. *Sites J. 20Th Century Contemp. French Stud.*, p. 1–15, 2013.

Vitau Automation. 2022. Acesso em: 01-06-2022. Disponível em: <https://vitauautomation.com/>.

WOHLGETHAN, E. *SupportingWeb Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js*. Tese (Doutorado) — Hochschule für Angewandte Wissenschaften Hamburg, 2018.

YAGHINI, V. *Organize your application code in three-tier architecture*. 2020. Acesso em: 30-05-2022. Disponível em: <https://openclassrooms.com/en/courses/5684146-create-web-applications-efficiently-with-the-spring-boot-mvc-framework/6156961-organize-your-application-code-in-three-tier-architecture/>.