



# Plataforma de análise e visualização de dados para sistemas IoT industriais baseada em métodos de *Big Data*

**Gabriel Felipe Nunes Ferreira**

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação.

Trabalho orientado por:

Professor Paulo Alexandre Vara Alves

Professora Simone de Almeida

Bragança

2020

# Agradecimentos

Gostaria de agradecer a minha mãe e ao meu irmão, por todos estes anos de estudos estarem a dar todo o apoio para que eu atingisse meu objetivo, mesmo que com todas as dificuldades que enfrentei.

Aos meus orientadores, Professor Doutor Paulo Alexandre Vara Alves e Professora Doutora Simone de Almeida, por me auxiliarem no trabalho, me aconselhando e me ensinando, que foram essenciais nessa etapa.

Também, aos amigos e pessoas à minha volta que me ajudaram nesta formação, academicamente e psicológicamente.

Gostaria de agradecer também as instituições Universidade Tecnológica Federal do Paraná e Instituto Politécnico de Bragança, que fizeram possível este trabalho.

# Resumo

O presente trabalho é parte do projeto ON-SURF, que propõe um modelo de análise e visualização de dados utilizando sensores *IoT* de temperatura e pressão em máquinas injetoras de plástico. O objetivo é monitorar os dados que são coletados no processo de fabricação de moldes de plástico.

O projeto é composto por três etapas, em que a primeira etapa é responsável pelo condicionamento dos sinais dos sensores efetuando a coleta dos dados. A segunda etapa é composto pela aquisição de sinal, desenvolvendo um módulo de digitalização e envio do sinal para um concentrador através de comunicação *Ethernet*. A terceira etapa, no qual é o foco desta tese, é o concentrador de dados e a plataforma de análise e visualização de dados, no qual é responsável por coletar os dados enviados pela aquisição, armazena-los, e apresenta-los em uma plataforma local ou *Web*.

O estudo apresentou-se como uma boa solução para a análise e visualização de dados, tendo um tempo de resposta eficaz, mesmo quando operando com uma grande carga de dados. Os testes foram eficazes em ambiente local, ou por *Ethernet*, porém, quando dependendo da rede houve um atraso maior na visualização dos dados. O modelo ainda pode ser melhorado, aperfeiçoando o processo de análise, e de limpeza dos dados que não são mais utilizados.

**Palavras-chave:** Visualização de Dados, Sistemas em Tempo Real, Big Data, Internet das coisas.

# Abstract

The present work is a part of the project ON SURF, which proposes a model of analysis and visualization of data using temperature and pressure sensors at plastic injection machines. The objective is to monitor the data collected from the manufacture process of plastic molds.

The project consists of the three steps. The first step is responsible for the signal conditioning of the sensors collecting the data. The second step is responsible for the signal aquisition, developing a scanning and sending signal module to a concentrator through comunication. The third step, which is the main subject of this thesis, is the data concentrator and the analysis and data visualization platform, that collects the data sent by aquisition, store and present them into a local platform.

This study showed itself as a good solution for data analysis and visualization, having an effective time of response even when operating with large data scale. The tests were effective at a local environment, or by Ethernet, yet, depending on the web there was a larger delay on the data visualization. The model can still be improved, perfecting the analysis process, and the data cleaning which are no longer in use.

**Keywords:** Data Visualization, Real-Time Systems, Big Data, Internet of Things.

# Conteúdo

# Lista de Listagens

# Lista de Figuras

# Capítulo 1

# Introdução

O presente capítulo apresenta as informações introdutórias desta pesquisa, seguindo pela identificação do problema, motivação, e estrutura do documento.

## 1.1 Identificação do problema

Existem diversas técnicas de análise e visualização de dados que incluem algoritmos de Big Data e de Ciência dos Dados, tornando possível automatizar a análise de grande volume de dados que provém de redes de sensores industriais IOT, através da criação de modelos analíticos.

O Big Data permite uma maior precisão na análise dos dados, maior rapidez, maior produtividade e menores custos, existindo uma grande diversidade de bibliotecas de código aberto que podem ser usadas para este fim.

A aplicação do Big Data a dados provenientes de uma rede de sensores IOT permite a criação de modelos para o controlo da qualidade e disponibilização de sistemas de alerta quando existem problemas ao nível da qualidade da produção, identificando os principais fatores. Pretende-se que este modelo disponibilizado através de uma plataforma web seja muito relevante para as linhas de produção industrial que utilizam diversos tipos de sensores, mas que não possuem uma plataforma de análise e visualização de dados integrada.

## 1.2 Motivação

O presente trabalho é desenvolvido em contexto com o projeto ON-SURF - Plataforma tecnológica de Engenharia de Superfícies, que envolve diversas empresas nacionais e internacionais de diversos setores de atividade, bem como centros de investigação do sistema científico nacional. Pretende-se desenvolver e aplicar processos de modificação de superfície que promovam soluções avançadas e inovadoras em diferentes sectores de atividades, tais como Automóvel, Aeronáutica, Moldes  Ferramentas, Saúde e Eletrónica [1]. Trata-se de um projeto, que é sediado em parceria com o CEDRI-IPB (Centro de Investigação em Digitalização e Robótica Inteligente), que é uma unidade de pesquisa interdisciplinar aplicada que agrupa pesquisadores ativos das áreas de eletrônica, ciência da computação e matemática, localizado no Instituto Politécnico de Bragança.

O projeto ON-SURF utiliza moldes na produção de peças plásticas, que são criadas a partir de máquinas injetoras de plástico. Estas máquinas tem uma variação de temperatura alta, danificando os componentes da máquina com o tempo, sendo necessário a troca dos componentes.

A motivação para o trabalho é de desenvolver sensores aplicados a esta máquina, no qual, capturam os dados de temperatura e pressão, e apresentam-os em uma plataforma de análise e visualização de dados. Assim, com estes dados coletados, pode-se ter mais controle sobre o processo de produção, analisando se o molde já está em extrema temperatura, e posteriormente, verificar se há danos nos componentes.

O projeto ON-SURF foi dividido em três etapas:

- Condicionamento de sinal;
- Aquisição de sinal;
- Plataforma de análise visualização de dados.

O presente trabalho foi desenvolvido com base na terceira etapa do projeto, no qual apresenta a construção de uma plataforma de análise e visualização de dados dos sensores

que captam os valores de temperatura e pressão. As outras etapas do projeto foram desenvolvidas por outros responsáveis.

## 1.3 Objetivos

O presente trabalho propõe uma solução para a visualização de dados de temperatura e pressão das máquinas injetoras de plástico. A partir dos dados coletados, é efetuado uma análise destes dados, para apresentar os dados da forma mais clara possível.

Esses dados são enviados para uma base de dados local, e que então devem ser visualizados em forma de gráficos em tempo real. Já para consultas posteriores dos dados, também é desenvolvido um *dashboard* para estes dados, dividido-os por intervalos de tempo, no qual são visualizados os dados máximos, médios, e mínimos, da temperatura e pressão, em forma de gráficos de linhas e circulares. Este trabalho deve apresentar uma solução para os seguintes objetivos específicos:

- Desenvolver um servidor para coleta, envio e organização dos dados;
- Organizar dados de temperatura e pressão coletados pelos sensores;
- Enviar dados para as interfaces gráficas;
- Desenvolver um protótipo de visualização de dados utilizando os princípios de Big Data;
- Desenvolver uma aplicação para sistemas de placa única de baixo custo para visualização de dados em tempo real na linha de produção.

A solução pretende ser utilizada não só em ambiente na web, mas também na linha de produção em sistemas de baixo custo de placa única, como por exemplo RaspberryPI, permitindo a visualização em tempo real.

## 1.4 Estrutura do Documento

O presente trabalho foi dividido em seis capítulos que descrevem o trabalho desenvolvido. O capítulo 1 apresenta a introdução ao tema, como a motivação e seus objetivos. O capítulo 2 mostra a revisão da literatura, no qual infere as pesquisas feitas ao longo dos temas relacionados ao trabalho. O capítulo 3 discorre as ferramentas utilizadas no desenvolvimento. O capítulo 4 apresenta o desenvolvimento completo do projeto. No capítulo 5 é mostrado os testes, e enfim no capítulo 6 é apresentado as conclusões do trabalho e os possíveis trabalhos futuros.

# Capítulo 2

# Revisão da Literatura

Este capítulo apresenta os conceitos estudados nas áreas de Big Data, visualização de dados, dados em tempo real, internet das coisas e estudos nestas áreas para o desenvolvimento do trabalho.

## 2.1 *Big Data*

Nos últimos anos, os dados tem crescido em larga escala em várias áreas da informação. Em 2011, a quantidade de informação criada e replicada no mundo era de 1.8 *zettabytes* (1.8 trilhões de *gigabytes*) [2]. Com esse grande aumento de dados, foi necessário um aumento enorme nos estudos relacionados a estes dados, trazendo muitas tecnologias relacionadas ao *Big Data*.

O termo *Big Data* refere-se a grandes conjuntos de dados em crescimento. Estes dados podem ser gerenciados, obtendo mais confiabilidade do que se é apresentado, agregando-se dados de diferentes fontes, e codificando os dados para a segurança e privacidade. O objetivo do gerenciamento de *Big Data* é garantir que os dados confiáveis sejam facilmente acessíveis, gerenciáveis, armazenados e protegidos [3].

À medida que mais e mais campos envolvem problemas de Big Data, que vão da economia global à administração da sociedade e de pesquisas científicas para a segurança nacional, entramos na era do Big Data.

Com previsões diversificadas de dados, como redes de sensores, telescópios, experimentos científicos e instrumentos de alto rendimento, os conjuntos de dados aumentam na taxa exponencial, como demonstrado na Figura 2.1 [4], chegando a ultrapassar as capacidades computacionais.

Em muitas aplicações importantes de Big Data, as técnicas e tecnologias de ponta não podem resolver idealmente os problemas reais, especialmente para análises em tempo real.

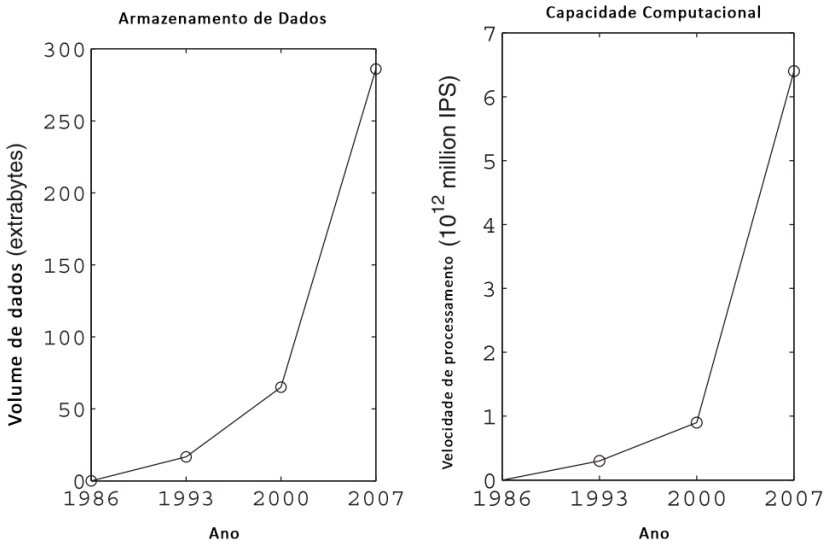


Figura 2.1: O aumento dos dados superou as capacidades de computação (adaptado de Philip Chen et al. [4])

Normalmente, o processo de análise é mostrado na Figura 2.2, onde o conhecimento é descoberto na mineração de dados. Como os tamanhos do conjunto de dados geralmente são muito grandes, as bases de dados atuais do mundo real são severamente suscetíveis a dados inconsistentes e incompletos. Portanto, várias técnicas de pré-processamento de dados, incluindo limpeza de dados, integração, transformação e redução de dados, podem ser aplicados para remover ruídos e corrigir inconsistências [4].

Enquanto o volume do *Big Data* aumenta, o mesmo acontece com a complexidade e os relacionamentos sob os dados. Em uma fase inicial dos sistemas de informação centralizada dos dados, o foco está em encontrar os melhores valores de recursos para


Figura 2.2: Processo de descoberta de conhecimento (adaptado de Philip Chen et al. [4])

representar cada observação. Isso é semelhante ao uso de vários campos de dados, como idade, sexo, renda, formação educacional, etc. para caracterizar cada indivíduo [5].

Então, o *Big Data* teve que se desenvolver para abranger toda esta complexidade. Esta evolução do *Big Data* é definida como "Os três Vs", no qual são: Volume, Velocidade e Variedade [6]. Outros conceitos foram adicionados aos "Vs", conforme os tempos passaram: veracidade e valor.

Como é possível ver na Figura 2.3, os três Vs são vistos como os pilares do *Big Data*, pois quando juntos, se tornam a união de todos os atributos dos mesmos [7].

1. **Volume:** se refere a quantidade de dados que são armazenados. Antes da explosão do mundo informático, empresas e governos coletavam dados, mas o tempo para armazenar os dados era desafiador. Hoje, o volume de dados coletados não é mais um problema. Esses dados também podem ser quantificados contando registros, transações, tabelas ou arquivos [7]. Estas definições de grandes volumes de dados são relativas e variam de acordo com fatores, como tempo e tipo de dados. O que

pode ser considerado um limite hoje, o *Big Data* pode não atingir o limite no futuro, porque capacidades de armazenamento aumentarão, permitindo conjuntos de dados ainda maiores para se capturar [8].

2. **Velocidade:** se refere a velocidade na qual os dados são coletados e armazenados. Atualmente, os dados estão sendo coletados em tempo real a taxas muito altas. A velocidade abrange a geração dos dados, como também a frequência da taxa de entrega dos dados. A coleta dos dados em tempo real já é uma técnica antiga, utilizada por muitas empresas que coletavam dados de cliques de sites, usando os dados para fazer recomendações para visitantes da Web [7]. A quantidade de dispositivos digitais, como *smartphones* e sensores, levou a uma taxa imensa de criação de dados, gerando necessidade para análises em tempo real e planejamento baseado em evidências [8].

3. **Variedade:** se refere aos diferentes tipos de dados que são armazenados. Um dos fatores que torna o *Big Data* realmente grande é a vasta variedade de dados que são armazenados. Muitas das mais recentes são fontes da *web*, incluindo *logs*, cliques em anúncios e redes sociais [7]. Os avanços tecnológicos permitem o uso de vários tipos de estruturas, como dados semiestruturados e não estruturados [8].

   Dados **estruturados** se refere aos dados tabulares encontrados em planilhas ou bases de dados relacionais. Texto, imagens, áudios e vídeos são exemplos de dados **não estruturados**, que carecem de organização estrutural. E dados **semiestruturados** são os dados nos quais não estão em conformidade com padrões rígidos, como XML ou JSON.

Um problema que o *Big Data* traz se diz respeito ao armazenamento, gestão e análise dos dados. Este gerenciamento tradicionalmente é baseado em bases de dados relacionais. No entanto, essas bases de dados relacionais aplicam-se apenas a dados estruturados.

As bases de dados relacionais tradicionais, por muitas vezes, não conseguiram lidar com o enorme volume de dados, devido a robustez nos dados. Para solucionar o problema,

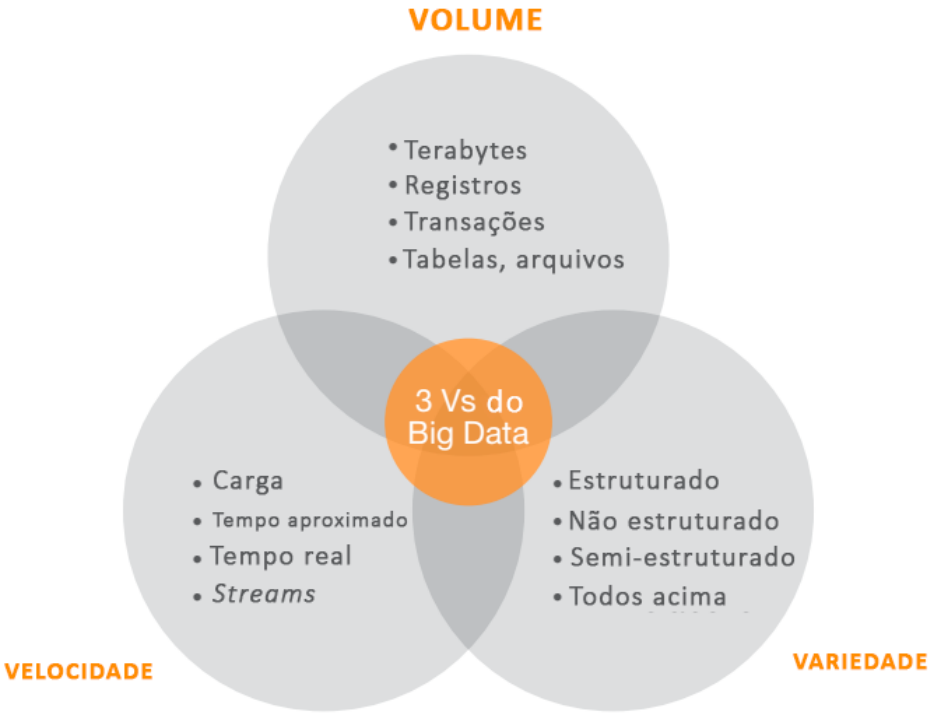


Figura 2.3: Os três Vs do *Big Data* (adaptado de Russom)

utilizar de bases de dados *NoSQL* é uma boa opção para o gerenciamento permanente dos dados.

Tais estruturas obtiveram grande sucesso no processamento de tarefas, especialmente para classificação de páginas web. Vários grandes aplicativos são desenvolvidos com base nessas tecnologias ou plataformas inovadoras [9].

Um problema enfrentado pelos pesquisadores diz respeito às técnicas analíticas e estatísticas disponíveis para analisar os dados. O volume e a variedade de dados são tão grandes e rápidos que os métodos de análise de dados tradicionais não são mais adequados para lidar com o constante aumento de informações [7]. Estudos nos dados são feitos constantemente para que novos métodos sejam desenvolvidos, para tornar mais eficiente a análise dos dados.

## 2.2 Visualização de Dados

Os dados estão presentes em diversas áreas do conhecimento e utilizados por diversas tecnologias, devido a vivermos uma sociedade digital. Seja por mídias, redes sociais, e também por muitas empresas, esses dados são de extrema importância, e devem ser analisados.

Atualmente, existem técnicas de análise de dados, incluindo mineração de dados, análise estatística e aprendizado de máquina. E uma destas técnicas, é a visualização de dados, que permite apresentar os dados importantes e necessários de uma amostra de dados, apresentando de uma maneira clara e consisa [3].

A visualização de dados trata-se de representações visuais de dados quantitativos, representados em sua maioria por gráficos, sejam eles gráficos de "*pizza*", linhas, barras, áreas, entre outros. A visualização de dados, especificamente, é o processo que utiliza tecnologias computacionais para transformar dados abstratos em modelos visuais [10].

Na visualização de dados, o elemento visual de uma apresentação é projetado para garantir que todas as partes de uma tomada de decisão entendam e possam interpretar corretamente as informações que lhes são apresentadas. Em muitas situações, a visualização de dados é a melhor maneira de entender as informações, enquanto para outras a visualização de dados simplesmente ajuda a contextualizar fatos que são apresentados oralmente ou por escrito.

Hahn [11] descreve uma situação em que deve-se imaginar que um pesquisador precise apresentar as diferentes idades em uma comunidade por meio de um gráfico de barras ou gráfico de pizza, no qual 20% da população esteja entre 0 e 18 anos, 20% entre 30 e 40 anos e 60% acima de 60 anos. Um gráfico de barras pode mostrar lacunas significativas na população - como que não há residentes entre 19 e 30 anos na comunidade. No entanto, um gráfico de pizza dividiria o círculo em três grupos, mostrando uma população significativamente maior de residentes com mais de 60 anos de idade, mas não destacaria a falta de moradores entre 19 e 30. Ambas as visualizações de dados mostram a mesma informação, mas a interpretação para o público é diferente. No gráfico de barras, a comunidade é

apresentada como obviamente perdendo uma demografia significativa. Considerando que, no gráfico de setores circulares, a comunidade é apresentada como um círculo completo, que possui um número alto de mais de 60 residentes, mas não está necessariamente ausente de nenhum grupo demográfico. A decisão sobre qual gráfico usar é fundamental para os formuladores de políticas públicas que precisam do público para decidir sobre novas políticas.

Para qualquer visualização de dados, será necessário uma ferramenta para efetuar a criação dos gráficos para visualização. Existem linguagens específicas para esse tipo de tratamento de dados, bem como bibliotecas para visualização de dados.

Existem inúmeras bibliotecas ou aplicações que são desenvolvidas para visualização de dados. No geral, atualmente com ferramentas como *PowerBI* são muito utilizadas no mercado de trabalho, o que chama a atenção de gestores de empresas.

O *PowerBI* é uma ferramenta de visualização de dados para efetuar análises de dados empresariais. Essas análises são focadas em *Bussines Intelligence*, para setores empresariais, o que abrangem tanto pequenas tabelas, como largas escalas de dados [12].

Também, existem outras ferramentas de visualização como o *Grafana*, uma ferramenta de análise e visualização de dados de código aberto e multiplataforma. É uma ferramenta *Web* que fornece tabelas, gráficos e alertas na aplicação [13]. Ainda citando ferramentas de análise e visualização de dados de código aberto, há a plataforma *ThingsBoard*, focado em $IoT$ [14].

O fator determinante no qual estas ferramentas de análise e visualização de dados não são próprias para o projeto é que há muitas limitações quanto a apresentação de dados em tempo real. Suas interfaces não são preparadas para isto, visto que o sistemas geralmente são *Web*, e pela necessidade de uma rede, causa um atraso na visualização dos dados em tempo real.

Outro fator que influencia na integração destas ferramentas para com o projeto é que há a necessidade de integrar o projeto em um ambiente com pouca disponíbilidade de *hardware*. O *software* de visualização em tempo real necessáriamente deve poder rodar em um *Raspberry PI*, o que estas ferramentas prejudicam um pouco, por geralmente serem

*softwares* custosos para o sistema.

Por isso, foi optado por bibliotecas para desenvolvimento de aplicações de análise e visualização de dados que são leves e de fácil integração, como o *MatPlotLib*. Esta biblioteca para *Python* permite desenolver vários tipos de gráficos, para cada necessidade, e permite uma grande integração com outras bibliotecas de programação. A Figura 2.4 apresenta exemplos de gráficos que podem ser utilizados no *MatPlotLib*.

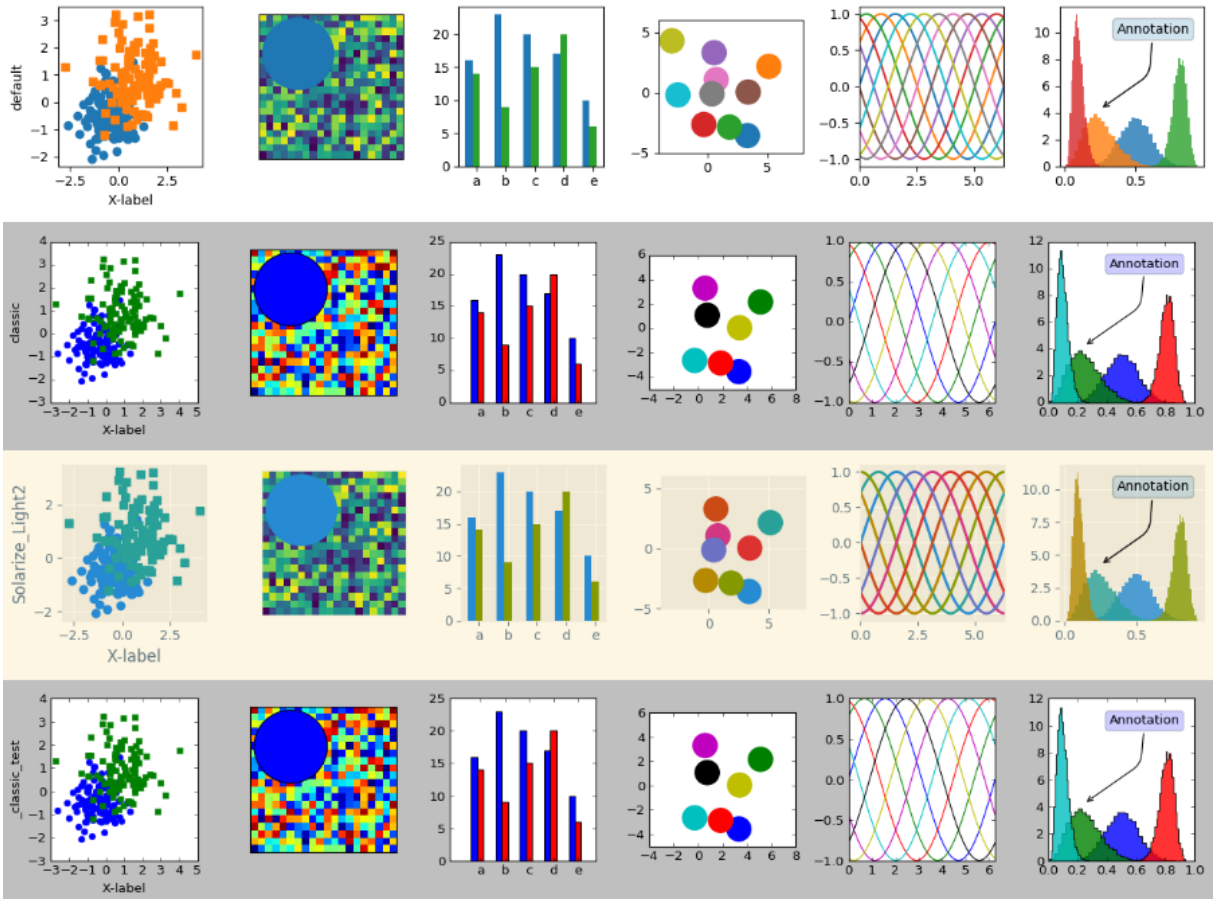


Figura 2.4: Exemplos de gráficos de visualização de dados do *Matplotlib* [15].)

A visualização de dados se trata de um estudo muito importante para o presente trabalho, pois permite auxiliar no processo de criação de gráficos com os dados coletados.

## 2.3 *Internet of Things*

*Internet of things*, ou *IoT*, refere-se ao conjunto de dispositivos e sistemas que interconectam sensores e atuadores do mundo real para a internet. Isso inclui vários sistemas:

- Carros conectados à internet;
- Dispositivos portáteis, incluindo dispositivos de monitoramento de saúde;
- Medidores inteligentes;
- Sistemas de automação residencial e controles de automação;
- Smartphones;
- Redes de sensores sem fio.

Como é possível ver na Figura 2.5, o volume de pesquisas a respeito de *IoT* vem aumentando consistentemente com a tendência de queda das redes de sensores sem fio.

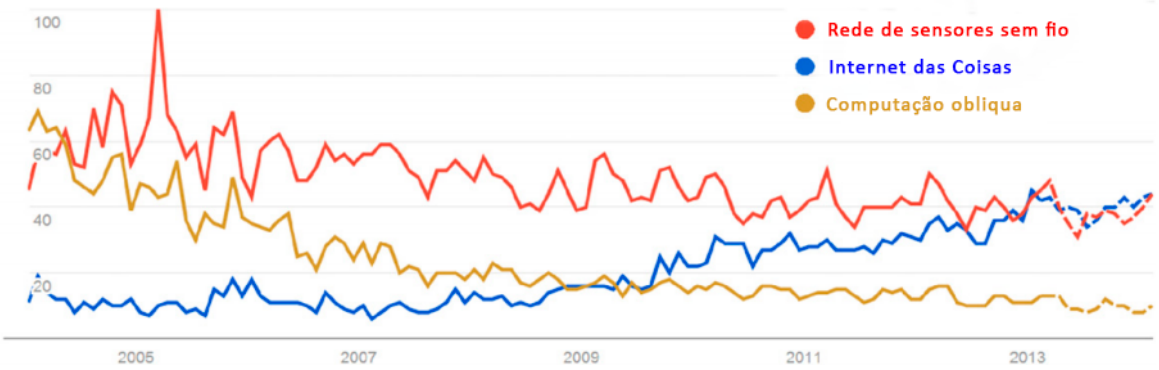


Figura 2.5: Tendências do *Google Search* desde 2004 para termos Internet das Coisas, Redes de Sensores Sem Fio, Computação Ubíqua (adaptado de Gubbi et al. [16])

A visualização é muito importante para uma plataforma de *IoT*, pois isso permite a interação do usuário com o ambiente. Com os avanços recentes das tecnologias, o uso de *tablets* e *smartphones* tornam-se muito intuitivo.

Os dispositivos embarcados tem seu uso mais eficaz em espaços inteligentes. Um espaço pode ser fechado como uma sala de reuniões ou um corredor, ou um local bem definido de área aberta, como um pátio ou um quadrante.

Incorporando a infraestrutura de computação, uma solução inteligente reune dois mundos disjuntos. A fusão destes mundos permite sentir e controlar um mundo pelo outro. Um exemplo simples é o ajuste automático de níveis de aquecimento, refrigeração e iluminação [17].

A extração dos dados brutos abrange tanto a detecção de eventos quanto a visualização dos dados brutos, com informações representadas de acordo com as necessidades do usuário final [16].

No paradigma da *IoT*, uma enorme quantidade de redes de sensores são incorporados em vários dispositivos e máquinas no mundo real. Tais sensores implantados em diferentes campos podem coletar vários tipos de dados, como dados de ambiente, dados geográficos, dados astronômicos, etc. Equipamenteos móveis, instalações de transporte, instalações públicas e eletrodomésticos podem ser equipamentos de aquisição de dados.

O *Big Data* gerado pelo *IoT* tem características diferentes em comparação com o big data geral devido aos diferentes tipos de dados coletados, cujas características mais clássicas incluem heterogeneidade, variedade, característica não estruturada, ruído e alta redundância.

Até 2030, a quantidade de sensores atingirão um trilhão e, em seguida, os dados da IoT serão a parte mais importante do big data, de acordo com a previsão da *HP* [9].

## 2.4 Sistemas em tempo real

Um dos desafios que a análise de dados nos traz são como tratar e coletar dados em tempo real. Conforme os dados se tornam importantes, mais rápidos e eficazes devem ser os sistemas, em muitos casos, os sensores *IoT*, como sensores que coletam dados de temperatura, pressão, umidade, etc.

Estes tipos de sistemas necessitam em alguns contextos, um tempo de resposta imediato, para que possam ser tomadas decisões de operação sobre estes sistemas.

Um sistema em tempo real é um sistema de computador no qual a correção do comportamento do sistema depende não apenas dos resultados lógicos dos cálculos, mas também no instante físico em que esses resultados são produzidos [18]. Significa que estes dados devem ser apresentados com o mínimo de atraso no tempo possível, para garantir a integridade dos dados.

Como apresentado na figura 2.6, um sistema em tempo real pode ser decomposto em subsistemas chamados *clusters*, como por exemplo, o objeto controlado, o sistema em tempo real, e o operador humano.

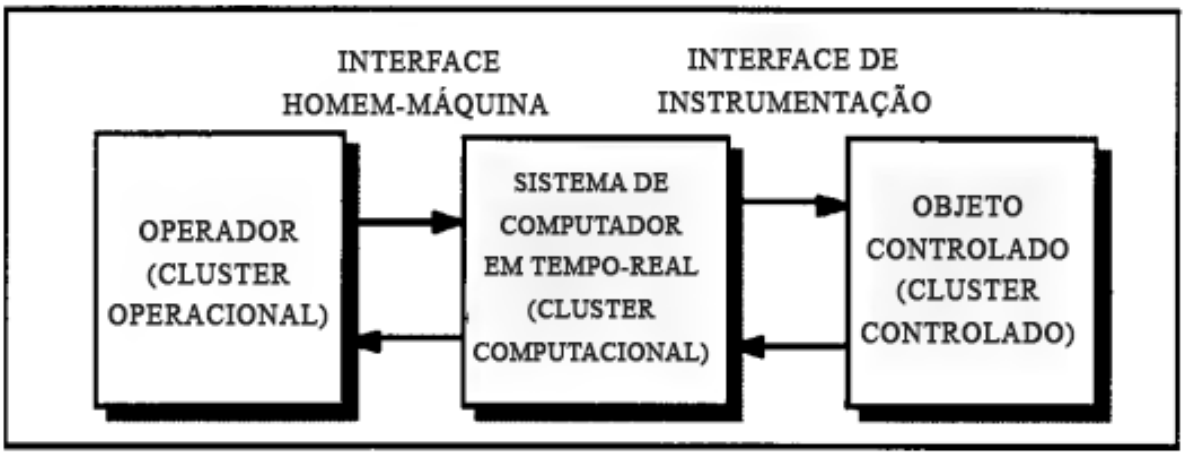


Figura 2.6: Sistema de tempo real (adaptado de Kopetz et al. [18])

A interface entre o operador humano e o sistema em tempo real é chamado de interface homem-máquina (*Man-Machine interface*) e a interface entre o objeto controlado e o objeto sistema em tempo real é chamado de interface de instrumentação (*Instrumental Interface*).

Os sistemas em tempo real são caracterizados pelo fato de que devem reagir a mudanças no estado do sistema e fornecer ação de controle adequada em tempo útil. Para garantir a estabilidade de tal sistema, é essencial determinar o pior tempo de reação do sistema, que compreende o tempo para coletar dados do sensor, o pior dos casos de execução do algoritmo e o tempo necessário para que os atuadores iniciem a ação de controle desejada.

Qualquer latência de computação afeta o desempenho do sistema e deve ser considerado no design do controlador [19].

Estes problemas de estimativa de tempo razoável, e a variável do tempo de resposta podem ser resolvidos usando uma abordagem de caminho único. Este paradigma gera o código que produz em um único caminho de execução do programa, para que não tenha a interferência de outros fatores que possam interferir no tempo de resposta. Quando executado em *hardware* o tempo de execução do código de caminho único é constante independentemente dos dados de entrada.

Existem sistemas em tempo real em que o sistema de controle deve interagir com seu ambiente com base nos dados disponíveis sobre o meio ambiente [20]. Um exemplo é um carro em um cruzamento. Imagine que todos os semárofos devem se comunicar, pois, se houver latência ou variação do tempo de resposta de um dos semáforos, pode ocorrer uma catástrofe. Por este motivo, deve haver um conjunto de comunicação entre todos os componentes. Este conjunto de comunicação é chamado de base de dados em tempo real (*real-time databases*).

A base de dados em tempo real deve ser atualizada sempre que uma de suas entidades muda seu valor. Essas atualizações podem ser realizadas periodicamente, acionadas em tempo real por um período fixo de observação, ou imediatamente após uma mudança de estado de uma das entidades [18].

A necessidade de manter a consistência entre o estado real do ambiente ao estado refletido pelo conteúdo da base de dados nos leva a consistência temporal. A consistência temporal tem dois componentes [20]:

- **Consistência absoluta:** Entre o estado do meio ambiente e sua reflexão na base de dados. Isso decorre da necessidade de manter a visão do sistema de controle do estado do ambiente consistente com o estado real do ambiente.

- **Consistência relativa:** Entre os dados usados para derivar outros dados. Isso surge da necessidade de produzir as fontes de dados derivadas próximas umas das outras.

Por isso, é correto dizer que, o principal fator que determina se um sistema com uma base de dados em tempo real é trivial, é a consistência dos dados e da velocidade em que o dado é transmitido.

Existem duas tendências fundamentais que estão influenciando profundamente a maneira como as novas distribuições em tempo real estão sendo construídas: *Digital commoditization* e *Network-centric paradigm shift*. Os artefatos de hardware e software estão a ficar mais rápidos, mais baratos e melhores. E além disso, os sistemas estão ficando cada vez mais centrados em rede, onde são construídos integrando componentes separados conectando várias formas de serviços de comunicação [21].

## 2.5 Base de dados não relacional

Os sistemas de bases de dados relacionais dominam largamente o mercado, fornecendo um conjunto integrado de serviços que se referem a uma variedade de requisitos, que principalmente incluem suporte ao processamento de transações, mas também se referem ao processamento analítico e apoio à decisão. Do ponto de vista técnico, as principais bases de dados relacionais mostram uma arquitetura muito semelhante, e suportam *SQL* (*Structured Query Language*) como idioma por defeito.

No entanto, algumas preocupações surgiram com as bases de dados relacionais ao longo dos anos. Em algumas situações o desempenho não é adequado, e que, embora seja eficaz para muitas aplicações tradicionais, é considerado muito rígido ou não é útil em outros casos, com argumentos que exigem dados semi-estruturados para certas aplicações [22].

Um dos fatores de que trazem preocupação para as bases relacionais, é que cada vez mais as informações são guardadas diretamente a partir dos aplicativos, o que deixa o aplicativo mais robusto quando utilizado dados semi-estruturados. Ainda assim, a maioria das informações é armazenada na base de dados e não no aplicativo [23]. Porém, é difícil garantir a integridade dos dados com bases de dados relacionais quando compartilhados. Como nenhum aplicativo único tem controle sobre os dados, é muito difícil garantir que

todos os aplicativos estejam operando da mesma forma [24].

As bases de dados não relacionais conseguem resolver estes problemas, com sua velocidade e flexibilidade, porém, acaba perdendo na robustez do dado coletado. Os padrões estão se tornando cada vez mais estabelecidos em aplicativos on-line e arquitetura corporativa, e por isso, torna-se a norma mais importante a rapidez do que se é esperado [24].

Base de dados não relacional, ou *NoSQL* é uma classe de sistemas que gerenciam bases de dados e difere amplamente de bases relacionais de muitas maneiras significativas. Uma de suas caracteristicas mais importantes é que não se utiliza relações e nem tabelas como estrutura de armazenamento [23]. Também não é utilizado por defeito o *SQL* como idioma para as consultas da base de dados.

*NoSQL* são categorizados principalmente pelos seguintes modelos de dado [25]:

1. ***Key-Value Stores:*** Este é o modelo de dados mais simples e popular de bases *NoSQL*. Os dados são gerenciados e representados como pares (chave, valor) armazenados em estruturas de pesquisa eficientes, altamente escaláveis e baseadas em chaves, como tabelas de hash estruturadas com log. Ele permite que o desenvolvedor do aplicativo armazene dados sem esquema para os dados [23].

   O valor é codificado como uma matriz de bytes na qual sua serialização ou desserialização é deixada para o aplicativo cliente. Devido à estrutura sem esquema dos valores armazenados, a indexação e a consulta com base em valores não são suportadas pelo sistema, qualquer cenário que necessite consultar a estrutura interna dos valores de dados precisa ser implementado pelo aplicativo cliente.

   Portanto, os armazenamentos de *key-value* são adequados apenas para aplicativos que apenas use uma única chave para acessar dados, como carrinho de compras on-line, perfil / configuração do usuário e informações da sessão na *web* [25].

2. ***Wide-Column Stores:*** Diferentemente dos bancos de dados de linha, o modelo *wide-column* armazena seus dados na forma de colunas [23]. *Wide-column* é comparativamente eficiente que orientado a linha, pois os dados da coluna podem ser

gravados com eficiência e substituir dados antigos sem alterar nenhuma outra coluna para o linhas [25].

3. ***Document Stores:*** Esses são armazenamentos de *key-value* estendidos nos quais o valor é representado como um documento codificado em formatos semiestruturados por defeito, como XML, JSON ou BSON (Binary JSON). Um documento tem um esquema flexível por meio da adição ou remoção de seus atributos em tempo de execução, quando um atributo tem um nome junto com um ou mais valores.

   Os armazenamentos de documentos permitem consultar dados dentro de um documento sem precisar recuperar todo documento (por meio de sua chave) e depois inspecioná-lo.

4. ***Graph Stores:*** As bases de dados de grafos são bases de dados que usam estruturas de dados de grafos, juntamente com nós, arestas e determinadas propriedades. Nós podem representar qualquer entidade, como pessoa, negócios, ou qualquer item semelhante ao objeto repesentado em qualquer linguagem. Já as arestas relacionam um nó ao outro ou a alguma propriedade [23].

Representações que são eficasez e rápidas são triviais para algoritmos que necessitam de respostas em tempo real ou que exigem uma carga enorme de dados. Por estes motivos, para o presente trabalho, uma estrutura de dados baseada em *NoSQL* é demasiado importante para a eficiência do resultado final.

# Capítulo 3

# Tecnologias

Este capítulo apresenta os materiais e os métodos utilizados no presente trabalho. Espera-se abordar a utilização de cada uma das ferramentas principais para o bom funcionamento do sistema desenvolvido, assim como também as metodologias e os dados utilizados no desenvolvimento do projeto.

## 3.1 Os dados

Os dados utilizados para este trabalho tem como base os sensores *IoT*, que neste contexto, estão a coletar dados de temperatura e pressão em máquinas injetoras de plástico. Estes sensores estão acoplados nestas máquinas, nos quais a temperatura e pressão tem demasiada variação.

Estes dados se conectam com muitas das caracteristicas que é estudado em *Big Data*, e por isso, eles se aplicam em uma plataforma de *Big Data* para visualização e análise dos dados, pois como é visto, os dados devem ser coletados em grandes quantidades, com muita variação, e demasiada velocidade. Por estes motivos que também estes dados são semi-estruturados, em formato *JSON*, e serão armazenados em uma base de dados *NoSQL*, para que possa ocorrer muito mais escalabilidade dos dados. Além disso, a fonte dos dados será única, não necessitando de várias tabelas e relacionamentos.

## 3.2 Python

*Python* é uma linguagem de programação que é muito utilizada atualmente. É uma linguagem com uma sintaxe fácil de se utilizar, e com um ótima curva de aprendizado, principalmente para iniciantes.

A linguagem *Python* também provê muitas bibliotecas que facilitam o esforço para criação de algumas funções [26]. Tais bibliotecas são utilizados em vários tipos de projetos, e serão citadas abaixo algumas das bibliotecas que foram utilizadas no presente trabalho.

Uma das facilidades que a linguagem *Python* tem trazido, é a criação de *API's*, ou Interfaces de Programação de Aplicação. A *API* facilita a criação de um servidor em contextos *Web*, pois utiliza a rede para criar portas que comunicam com o cliente e se conectam com a base de dados da aplicação. Para este segmento, existem algumas bibliotecas que trabalham com requisições na rede em *Python*, como o Flask, que será descrito a seguir.

### 3.2.1 Flask

O *Flask* é uma biblioteca de requisições http em *Python* para criação de *API's*. Ele tem uma micro-estrutra, que significa que ele visa manter o núcleo da aplicação simples, mas extensível. A aplicação é limpa, o que o torna rápido em tempo de execução [27].

Uma de suas vantagens é que este micro-framework não toma muitas decisões pelo desenvolvedor, permitindo-o escolher e adicionar os pacotes que serão necessário para a aplicação. O *Flask* não tem suporte nativo para acessar bancos de dados, validar formulários da Web, autenticação usuários ou outras tarefas de alto nível. Esses e muitos outros serviços importantes que a maioria dos os aplicativos precisam estão disponíveis por meio de extensões que se integram aos pacotes da aplicação [28].

O *Flask* foi selecionado para este trabalho devido a não serem necessárias muitas das funcionalidades das *frameworks* mais completas, e por ser desenvolvido em uma linguagem de alto nível, o que nos permite utilizar hardwares mais simples, permitindo assim focar no desenvolvimento de um servidor para dados em tempo real.

## 3.3 Base de Dados

Em um sistema de *Big Data*, é priorizado a integridade do dado que é analisado, porém, é dificultado pela quantidade e pela velocidade que os mesmos são coletados em uma base de dados. Por isso, é necessário que esta base de dados seja veloz e que consiga comportar uma grande quantidade de dados para serem analisados.

No presente trabalho, foi necessário utilizar uma base de dados não relacional, para que não tivesse problemas pela demanda demasiadamente grande de dados. Como os dados são coletados em massa e rapidamente, foi utilizado uma base que consiga consultar uma grande quantidade de dados.

### 3.3.1 MongoDB

O *MongoDB* é um banco de dados distribuído, baseado em documentos e de propósito geral, orientado para desenvolvedores de aplicativos modernos e para a era da nuvem. Seus dados são armazenados em forma de documentos, o que significa que ele armazena dados em documentos semelhantes a JSON, utilizando o conceito de chave-valor. A sua linguagem tem uma consulta rica e expressiva que permite filtrar e classificar por qualquer campo, independentemente de quão aninhado possa estar em um documento [29].

O *MongoDB* foi projetado para desenvolver rapidamente aplicativos da Web e infraestrutura da Internet. O modelo de dados e as estratégias de persistência foram desenvolvidos para alto rendimento de leitura e gravação e capacidade de fácil escalabilidade. Se um aplicativo requer apenas um nó do banco de dados ou dezenas deles, o MongoDB pode fornecer um desempenho surpreendentemente bom [30].

Um ponto crucial que se deve levar em consideração é o tamanho das cargas de dados e a velocidade em que estes dados devem ser coletados. O *MongoDB* é desenvolvido aplicando estes conceitos e aumentando o desempenho geral do sistema [31]. Como o presente trabalho tarbalhará apenas com uma carga única e linear dos dados, foi priorizado a velocidade e o tamanho que a base de dados consegue suportar, e esta base de dados é capaz de resolver estes problemas, portanto, foi selecionada para o projeto.

## 3.4 Ferramentas de Interface do Usuário

Quando se explora a visualização de dados, obrigatoriamente se necessita de ferramentas que apresentem e criem o escopo dos dados apresentados, seja por meio de interfaces gráficas ou por um *Dashboard* para gerenciamento dos dados. São disponibilizadas tecnologias que nos permitem desenvolver interfaces gráficas para o usuário, agregando valor ao produto final.

No presente projeto, foi necessário desenvolver duas interfaces gráficas com tecnologias diferentes. O *Tkinter* foi utilizado para desenvolver uma interface de usuário de alto nível em *Python*, que permite ser executada em diferentes *hardwares* possíveis, e também, foi utilizado o *Angular* para desenvolvimento de uma aplicação *web*, para que seja possível obter a visualização de dados por meio de um *Dashboard*, a fim de visualizar dados de outros períodos de tempo.

### 3.4.1 Tkinter

*Tkinter* é um pacote desenvolvido para *Python*, que tem como objetivo desenvolver interfaces gráficas para dispositivos locais. Ela é a interface padrão do *Python*, e traz integração com diversos pacotes disponibilizados pela linguagem [32].

A ferramenta é utilizada no projeto em questão, pois possibilita que o *software* seja instalado diretamente em uma máquina com pouca disponibilidade de *hardware*. A utilização de *Python* e *Tkinter* têm a capacidade de fornecer aplicativos viáveis [33]. E para o presente trabalho, o desempenho é um fator determinante para o progresso do projeto.

### 3.4.2 Angular

Angular é uma *framework* para aplicações *web* que utiliza o conceito de componentização e criação de páginas eficientes em *Javascript* [34].

Diferentemente de alguns *frameworks* de desenvolvimento de aplicações web, o *Angular* envia o modelo da página e os dados diretamente para o navegador, para serem apresentados. A função de um servidor passa a servir apenas como recursos estáticos para os

modelos, e servir adequadamente os dados exigidos por estes modelos. Também é possível injetar dependências diretas nos seus componentes, trazendo o conceito de *Dependency Injection* [35].

Esta *framework* foi selecionada para o presente trabalho pela necessidade de utilizar tais conceitos na aplicação, como injeção de dependências, ciclo de vida de componentes e requisições na *API*, e também para uma posterior integração com outros sistemas já desenvolvidos em *Angular* no instituto.

## 3.5 Ferramentas de Visualização de Dados

As ferramentas de visualização foram importantes para o desenvolvimento do projeto para criar as interfaces de visualização nos sistemas desenvolvidos. Foram utilizadas duas bibliotecas que foram utilizadas para cumprir com o objetivo de criar estas interfaces: Matplotlib, e *ChartsJS*.

### 3.5.1 Matplotlib

*Matplotlib* é uma biblioteca desenvolvida em *Python* que nos permite plotar gráficos 2D para visualização de dados. Embora tenha origem em emular comandos gráficos do *MATLAB*, é independente do *MATLAB*, e também faz muito uso do *NumPy*, que também é uma biblioteca em *Python* [36].

O *Matplotlib* foi desenvolvido com a filosofia de que em poucos comandos, é possível criar qualquer tipo de gráfico 2d, como histogramas, gráficos de linha, de barra, dispersão, etc. Se por exemplo, é necessário visualizar um histograma com um vetor de dados, não é necessário instanciar objetos, chamar métodos e nem definir propriedades [15].

Esta biblioteca permite também integrar com diversas outras bibliotecas relacionadas com a matemática, que necessitam de grandes cálculos para análise de dados. Isso facilita o desenvolver no momento em que precisa apresentar grandes cargas de dados em uma interface.

Os dados apresentados podem ser atualizados em uma frequência alta de tempo, o que possibilita que os dados sejam também apresentados em tempo real, dependendo da disponibilidade de hardware, o que acaba fazendo com que esta biblioteca seja muito utilizada em aplicações em tempo real.

### 3.5.2 Chart.js

*Chart.js* é uma biblioteca desenvolvida em *Javascript* para a criação de gráficos animados e responsivos para aplicações *Web*, que nos permite gerar até 8 tipos diferentes de gráficos animados e customizáveis, e podem ser facilmente criados a partir de algumas *tags* utilizando *HTML5*[37].

Esta biblioteca é uma das mais populares em visualização de dados, e também uma das mais fáceis de se usar. O *Chart.js* fornece visualizações interativas prontas para uso de seus dados com mínima codificação. Depois de carregar seus dados em um *array* padrão em *Javascript*, é possível adicionar estilos e outras configurações usando estruturas declarativas simples baseadas em objetos [38].

O *Chart.js* escala automaticamente seus dados, gera linhas de grade e cria e se encaixa no espaço disponível, tornando o gráfico responsivo automaticamente. Também oferece uma customização mais básica para caso necessite um desempenho melhor na atualização e apresentação da visualização dos dados. No presente trabalho, esta ferramenta é essencial para a análise e visualização dos dados em tempo real, e também para o *Dashboard* que apresentará o histórico dos dados já coletados.

## 3.6 Comunicação Cliente-Servidor

Para se desenvolver uma comunicação cliente-servidor, é necessário utilizar métodos para criação de comunicação em tempo real. Isso é um problema, pois a instabilidade da conexão pode ser um fator que atrapalha a comunicação. Além disso, outros fatores influenciam a dificultar uma comunicação em tempo real, como perda de pacotes e oscilação da rede.

Para se resolver este problema, existem metodologias para envios de pacotes que podem garantir a chegada destes pacotes, como TCP e UDP [39]. Porém, o que é analisado de diferença entre os dois, é que no protocolo *UDP*, a prioridade é a velocidade da mensagem, mas não a integridade. Já o protocolo TCP garante a chegada da mensagem, porém, pode haver demora na comunicação.

Existem métodos eficazes em que o cliente se conecta a um servidor, e que garanta a chegada destes pacotes sem uma abordagem muito fora de contexto. O *Socketio* é uma ferramenta muito útil para resolver este problema de garantir a integridade e a coleta dos dados em tempo real, como será descrito a seguir.

### 3.6.1 Socketio

*Socketio* é uma biblioteca que permite trabalhar com sistemas em que exigem uma comunicação cliente-servidor em que não se pode haver perda de pacotes. O servidor quando inicializado, libera uma porta, que a partir desta porta, vários clientes podem se conectar a ela, e assim sempre que houver uma comunicação do cliente ou do servidor, a mensagem consegue ser recebida por qualquer receptor [40].

Estas conexões permitem que dados que entram no servidor possam ser enviados para diversas interfaces, como por exemplo, dados de um gráfico em tempo real. Além disso, podem também ser compartilhados documentos, e também, comunicações de *chat* entre os próprios clientes conectados.

# Capítulo 4

# Desenvolvimento

Este capítulo apresenta as etapas de desenvolvimento do projeto utilizando os conhecimentos adquiridos na revisão da literatura, e os materiais e métodos. O capítulo apresentará desde a definição da estrutura do modelo, como também os níveis de acesso, desenvolvimento do servidor e o desenvolvimento das interfaces gráficas.

## 4.1 Estrutura do modelo proposto

Seguindo os objetivos do presente trabalho, existiu a necessidade de organizar quais as partes que devem se conectar no projeto, para que se possa fazer a visualização dos dados corretamente. Como este trabalho é desenvolvido na última camada do projeto, ou seja, na camada de visualização, foi apenas necessário das outras partes receber os dados de forma que sejam tratados pela camada de visualização, e coletado na base de dados.

### 4.1.1 Arquitetura da plataforma de análise e visualização de dados

Como apresentado na seção de motivação do projeto, no capítulo de apresentação, o projeto se desenvolve na terceira etapa do projeto, no qual é efetuado o tratamento dos dados por meio de uma plataforma de análise e visualização. Portanto, neste capítulo

será abordado a arquitetura no qual o projeto foi desenvolvido.

Como apresentado na Figura 4.1, conforme os dados são captados pelos sensores, estes dados devem ser coletados pelo servidor, e apresentados em uma plataforma que apresenta gráficos em tempo-real. Estes mesmos dados coletados devem ser enviados para a base de dados, onde servirão para posteriormente analisar os dados destes sensores, assim, verificando possíveis problemas nos componentes das máquinas injetoras de plástico.

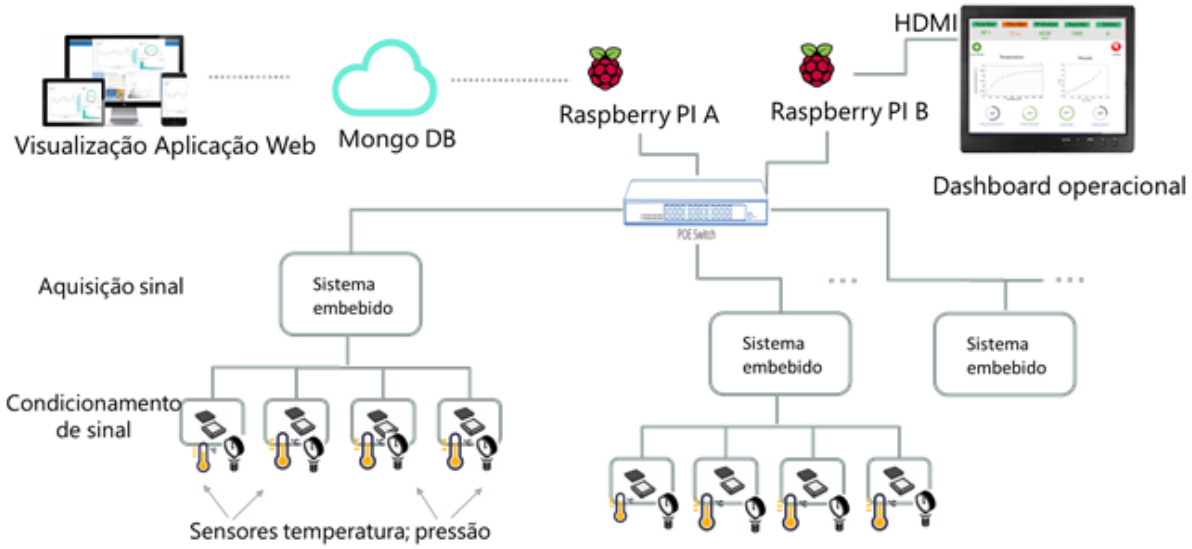


Figura 4.1: Etapas do projeto.

Após todo o processo de condicionamento de sinal e aquisição de sinal ser efetuado, os dados são enviados para um *Raspberry PI*, onde é iniciado o processo de coleta de dados, e posteriormente, de análise e visualização dos dados. Portanto, a arquitetura da plataforma se dividirá entre coleta de dados e visualização de dados.

Este servidor terá muitas funções, como tratar os dados, enviar para a base de dados os dados coletados, além de também receber requisições HTTP. Assim, conforme os dados são recebidos pelo servidor, os mesmos já são apresentados no *Dashboard* em tempo real, e posteriormente, requisitados na aplicação web.

Como pode-se notar na Figura 4.2, o servidor fica responsável por qualquer coleta, consulta e envio dos dados. A primeira ação é quando o dado é enviado da camada do projeto de coleta dos dados (1), e então, ao receber este dado, o servidor efetua duas

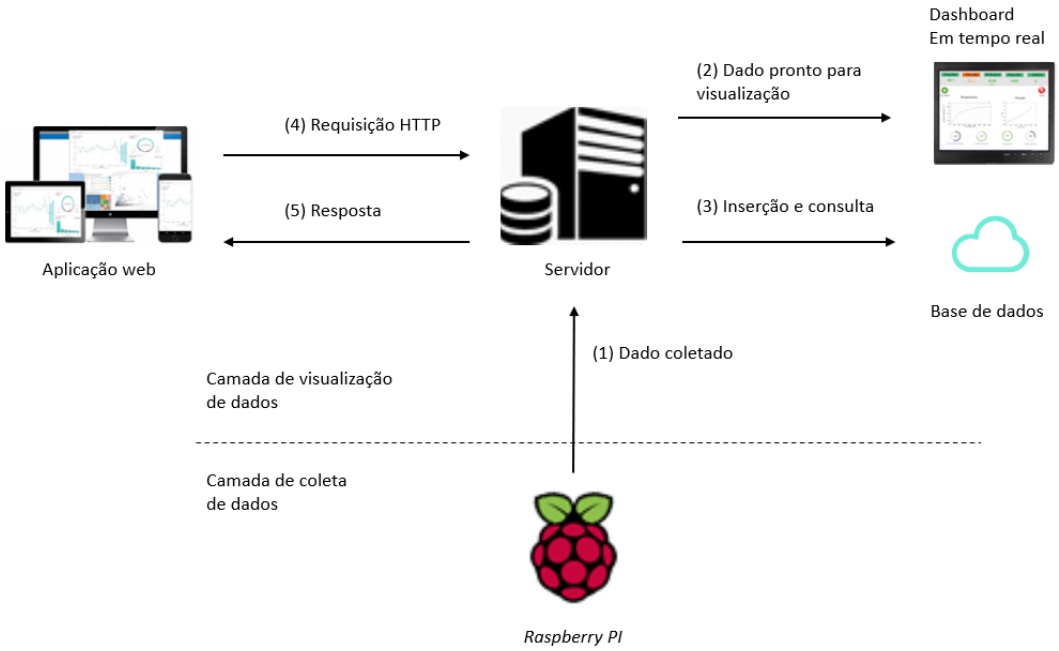


Figura 4.2: Diagrama da estrutura do projeto.

ações quase que simultaneamente, no qual se envia o dado atual para todas as interfaces gráficas que estiverem conectadas ao servidor (2), e insere o dado na base de dados local do *MongoDB* (3).

A partir destes passos, ao se conectar na aplicação web e o usuário efetuar uma consulta no *Dashboard* dos dados já coletados, será feita uma requisição ao servidor (4), e o servidor responderá à requisição com um *array* de dados, que serão apresentados na interface.

Como os dados são centralizados pelo servidor, o desenvolvimento do servidor foi uma das primeiras etapas do projeto para que depois pudessem ser desenvolvidas as interfaces de visualização de dados.

### 4.1.2 Servidor

O servidor tem como papel principal os processos de CRUD da aplicação. Estes processos tiveram que ser divididos em tarefas diferentes, porém, na mesma aplicação do projeto. O que deve se apontar são cada uma das tarefas, que serão dividas em inserção, envio e

requisição dos dados.

As tarefas de inserção e envio sempre acontecerão em conjunto, porém, o envio dos dados pode não acontecer, visto que dependerá que tenha algum cliente conectado ao servidor para receber o dado. Já a inserção sempre acontecerá, independente de ter algum cliente conectado ou não, o dado deve ir para a base de dados.

Como se pode notar na Figura 4.3, após inicializar o servidor e efetuar suas configurações, o servidor aguarda uma requisição para efetuar alguma ação. Caso ele receba uma requisição para coleta de dados, ele primeiramente irá verificar se há algum cliente conectado ao servidor. Caso haja, ele irá enviar esta mensagem em formato *JSON*, que será recebida por uma interface e apresentada para sua visualização. E independente de haver um cliente conectado ou não, ele irá posteriormente inserir este dado na base de dados.

O processo em que o cliente se conecta ao servidor em uma porta e consegue receber todas as mensagens que são enviadas é feito com a ferramente *Socketio*. Como falado anteriormente, esta ferramenta cria uma porta em que, por meio dela, o servidor e o cliente conseguem se comunicar por meio de mensagens em tempo real, por meio de *sockets*.

Estes *sockets* são como uma referência do endereço do cliente que é criada no servidor, e vice-versa. Por esta razão, o servidor consegue se comunicar com dezenas de clientes ao mesmo tempo, sem precisar que todos eles efetuem uma requisição.

Como pode-se ver na Figura 4.4, o servidor consegue criar uma conexão unificada onde se percebe todos os diferentes clientes conectados, como um sistema distribuído. Desta forma, se consegue enviar a mesma mensagem ao mesmo tempo para todos os clientes conectados, não permitindo que estes dados se percam.

Ainda assim, há também a tarefa de requisição de dados, no qual foi abordado anteriormente. Esta tarefa permite que o servidor efetue respostas com os dados no qual foram solicitados por alguma interface.

Estas requisições foram desenvolvidas no padrão de *REST API*, no qual o servidor deve efetuar operações de criação, remoção, edição e busca nos dados da base de dados.

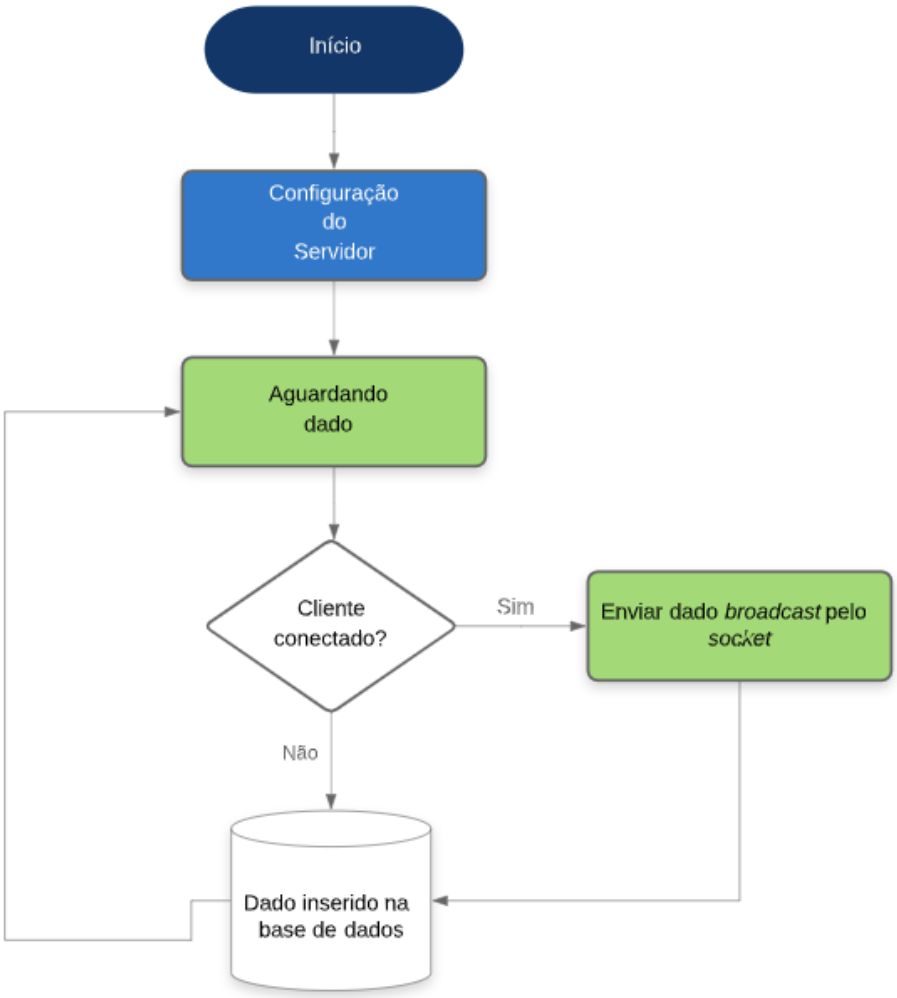


Figura 4.3: Fluxograma do processo de coleta e envio dos dados.

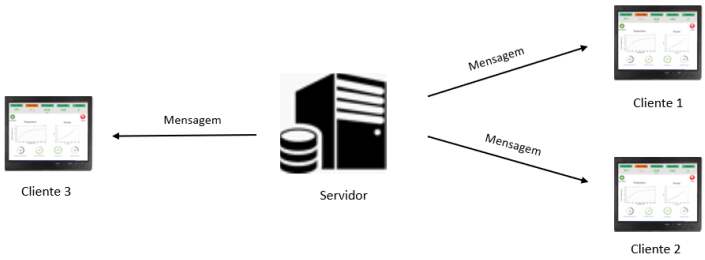


Figura 4.4: Diagrama do servidor com as conexões utilizando *Socketio*.

Estas requisições servem para que um cliente utilizando uma interface possa interagir com estes dados sem que se possa corromper a segurança destes dados, utilizando o conceito de portas no servidor, sendo determinado quais portas se conectam com a aplicação, e pelo endereço de IP.

A Figura 4.5 exemplifica como será feita as requisições no servidor. O cliente envia uma requisição que será respondida com os dados solicitados, como resposta, depois de um processo de análise nos documentos em questão.

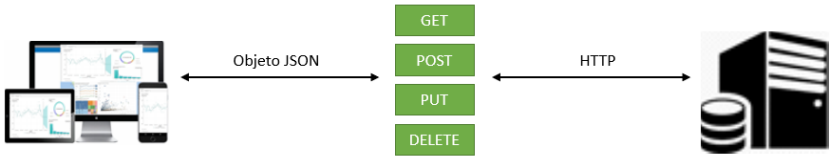


Figura 4.5: Diagrama das requisições *HTTP*.

O que se foi utilizado no presente trabalho sobre requisição de dados são as requisições *GET* e *POST*, para que os dados sejam enviados para a interface gráfica.

### 4.1.3 Cliente

O cliente representa qualquer usuário final que irá aceder ao backend, que terá acesso pelas interfaces gráficas do projeto. A função do cliente será analisar os dados que são apresentados, e por isso, é necessário desenvolver uma interface gráfica que utilize de métodologias de *User Interface* e *User Experience*.

Este usuário será capaz de se conectar a aplicação *Web* utilizando seus dados cadastrados, efetuando *Login* na aplicação, como é notado na Figura 4.6.

Após logado, o usuário será capaz de consultar os dados por meio de um filtro no formato de relógio, que efetuará a consulta por meio do servidor já descrito. A resposta será mostrada no *Dashboard*, que apresentará os dados do momento em questão. Esta etapa de apresentação dos dados será mais detalhada na seção de *Aplicação Web*.

As ações do cliente seguirão o princípio de poder filtrar os dados que serão necessários para a sua consulta. No *Dashboard*, ele poderá filtrar por dados específicos de um momento

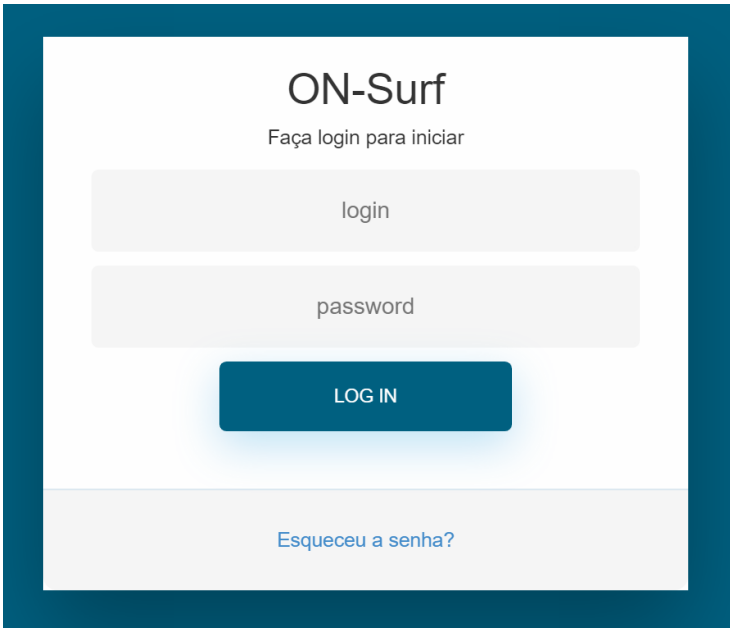


Figura 4.6: *Modal* de Login.

em que a máquina efetuou uma coleta de dados.

O cliente, após conectado no sistema, após filtrado os dados, receberá uma resposta se os dados existem ou não. Caso não existam, será necessário filtrar por outro momento, pois não há dados daquele período. Caso os dados existam, estes são apresentados nos gráficos do *Dashboard*, como é apresentado no fluxograma da Figura 4.7.

### 4.1.4 Estrutura dos dados

Para poder-se apresentar os dados, na transmissão entre servidor e cliente, deve-se saber o padrão da mensagem, que no caso, é um documento contendo todas as informações de uma leitura dos dados.

Cada leitura é representada por este documento, que é salvo em uma coleção, e posteriormente, consultado na base de dados. Como no presente projeto é utilizado a base de dados em *MongoDB*, então, os dados são estruturados em formato *JSON*. Este padrão de dado facilita a leitura do *Client-side* da aplicação, pois facilita a leitura os dados em objeto.

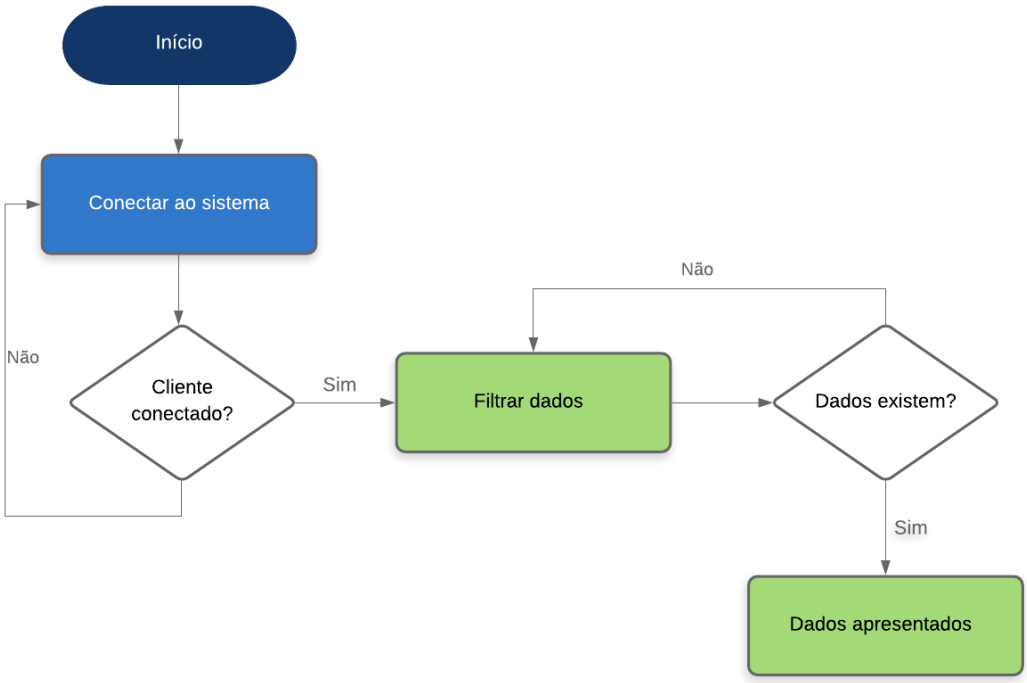


Figura 4.7: Fluxograma das ações do cliente.

A estrutura final dos dados coletados são representados na Listagem 4.1, no qual apresenta os dados ordenados por um *Timestamp*, no qual indicará qual foi o momento da leitura do dado. As seguintes informações também são apresentadas neste mesmo documento, o *ModuleAddress* que indica o endereço do módulo de rede, o *Data Counter* que apresenta uma contagem dos documentos, e os respectivos dados dos sensores, totalizando até 8 sensores diferentes.

```
{
    "Timestamp": timestamp ,
    "ModuleAddress": IPaddress ,
    "DataCounter": count_id ,
    "Sensor1": data1 ,
    "Sensor2": data2 ,
    "Sensor3": data3 ,
    "Sensor4": data4 ,
    "Sensor5": data5 ,
```

```
    "Sensor6": data6 ,
    "Sensor7": data7 ,
    "Sensor8": data8
}
```

Listagem 4.1: Estrutura dos dados coletados

Esta estrutura de dados foi projetada na camada 2 do desenvolvimento do projeto, no qual é coletado os dados do sensor e enviado para a base de dados.

### 4.1.5 Nível de Acesso

Para que se tenha uma estrutura de usuário na aplicação *Web*, para questões de regras de limitações do conteúdo abordado, foi desenvolvido um esquema de nível de acesso para que se tenha controle total e segurança sobre os dados coletados.

A partir disso, foi desenvoldido um campo na entidade usuário no qual limita a interação com a aplicação. Este campo é chamado de role, no qual o mesmo pode assumir dois valores: *User* e *Admin*.

A diferença entre estes dois tipos de usuário se limita a apenas ao gerenciamento dos usuários pelo *Admin*, no caso, o administrador do sistema. Ele terá acesso total a quem está acessando o sistema, e a quem também poderá acessar o sistema, já que é uma aplicação privada e não será qualquer usuário que poderá acessar. Então, o administrador tem total acesso a efetuar configurações gerais nos usuários, como criar novos usuários, edita-los ou excluí-los.

Na Figura 4.8, é apresentado um diagrama de caso de uso para a aplicação, no qual apresenta as funções permitidas pelos usuários. Como pode-se verificar, as outras interações que poderão ser feitas no sistema serão filtrar os dados por intervalo de tempo no *Dashboard*, e visualizar os dados em tempo real, na tela de tempo real. Ficando limitado ao administrador apenas gerenciar os usuários como complemento das funções.

Os dados acessados pelo perfil de acesso de usuário se limitam a dados de leitura, ou seja, isto significa que os dados importantes não poderão ser editados ou deletados. Isto

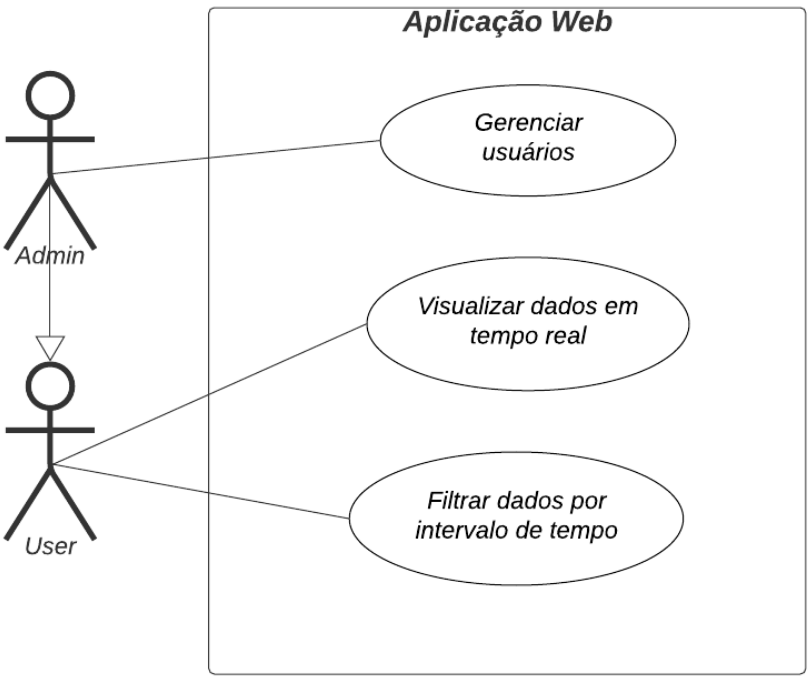


Figura 4.8: Diagrama de caso de uso da aplicação *Web*.

assegura que não sejam feitas mudanças nos dados já existentes, criando inconsistencias nas análises posteriores dos dados.

### 4.1.6 Organização dos arquivos

A organização foi desenvolvida de forma em que os arquivos de cada parte do projeto se encontram divididos por camadas funcionais. Para que seja feita a aquisição dos dados, foi utilizado conceitos de padrões de projeto para manter uma padrão na aplicação. O padrão em que se tornou base do projeto é o padrão *MVC*, um padrão para desenvolvimento de *Software* para facilitar o reuso e utilização do projeto.

O padrão divide o projeto em três partes, que são a camada de visão, controle e modelo. Como é visto na Figura 4.9, A camada de controle recebe a requisição da camada de visão, e efetua uma ação na camada de modelo. Após receber esta tarefa, a camada de modelo responde a requisição para a camada de visão.

Estas camadas foram dividas no projeto com o intuito de dividir as requisições em

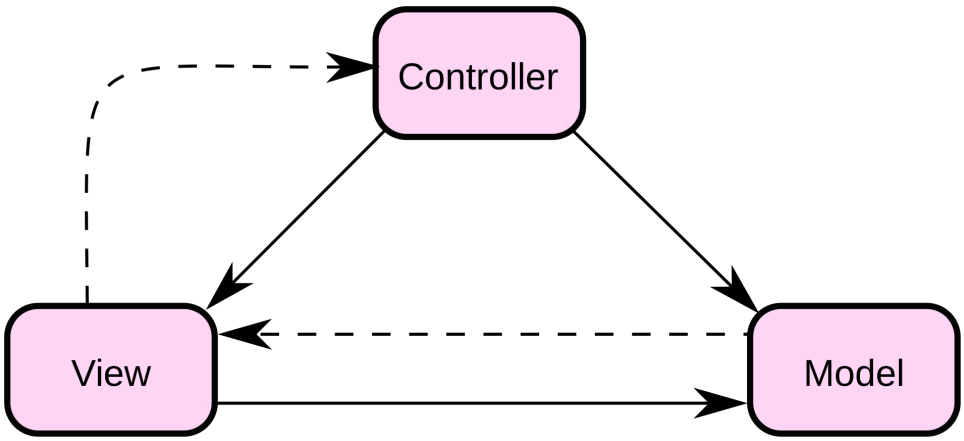


Figura 4.9: Diagrama simples do modelo *MVC*.

rotas da aplicação. A camada de visão é toda a implementação das interfaces gráficas do projeto. Já a camada de controle efetua todas as requisições e todo o tratamento dos dados da aplicação, e então faz uma chamada na camada de modelo. Quando efetuada uma chamada, a camada de modelo é quem efetua toda a comunicação com a base de dados, desde criação de novo dado, como também consultar algum dado.

As *Views* do projeto são toda a parte desenvolvida no projeto separadas das *Controllers* e das *Models*, pela necessidade de se ter um servidor centralizado e as interfaces serem acessadas por qualquer parte, mas se tem a necessidade de conectividade, seja por rede ou local. Por este motivo que as *Views* foram desenvolvidas em *frameworks* voltadas para desenvolvimento de interfaces gráficas, como o *Angular* e o *Tkinter*.

Estas interfaces foram desenvolvidas utilizando-se classes e componentes, que podem ser usados em interfaces gráficas para melhor separar seus conjuntos de informações. No caso dos componentes, foram separados por partes da tela que podiam ser reaproveitados, caso se estivesse em outra rota da aplicação, por exemplo, a barra fixa de menu.

As *Controllers* e *Models* foram desenvolvidas no mesmo projeto, estando diretamente relacionadas. Uma camada é totalmente dependentende da outra, visto que os dados que são tratados pela camada de controle devem ser armazenados pela camada de modelo. Nestas camadas, também se utilizou *frameworks* para este tipo de trabalho, e neste caso

utilizou-se o *Flask* para a criação do projeto, que dispõe de ferramentas para desenvolvimento de *API's* focadas em *Controllers* e *Models*. As *Models* também foram responsáveis pela realização do *CRUD* da aplicação.

Por muitas vezes, como a carga de dados é muito grande, foi exigida a necessidade de efetuar vários tratamentos nos dados, e por isso, a divisão das camadas foi extremamente necessária para que não haja perca no desempenho do projeto.

## 4.2 Sistema de Aquisição de Dados em Tempo Real

O presente projeto focou na apresentação e visualização dos dados, e por isso, foi necessário desenvolver uma estrutura que efetuasse a aquisição dos dados para que estes dados sejam apresentados.

Os dados coletados e a apresentação destes dados acontecem simultaneamente, e por isso, é necessário que estes dados sejam tratados e enviados para a interface gráfica o mais eficiente possível.

O foco que foi dado para o projeto é que pode haver muitas interfaces conectadas ao mesmo tempo, e todas elas serão supridas com os dados igualmente. Por esta razão que foi desenvolvido de forma que o cliente efetua a requisição para que possa iniciar a receção de dados, e a partir daí, o cliente é provido com a apresentação dos mesmos.

O cliente apresentar que está conectado significa que o servidor apenas enviará um dado para os clientes quando há algum cliente conectado, não exigindo a necessidade de que os dados sejam enviados quando não há nenhum cliente conectado. Isto facilita na tomada de decisões que o servidor tem com o cliente, e não o sobrecarrega quando há um alto número de requisições simultâneas.

De todas as formas, para que os dados sejam apresentados com precisão, é necessário que seja feito um tratamento destes dados para caso ocorra algum dado incorreto na aplicação. Este processo de tratamento e análise de dados será abordado no tópico seguinte.

## 4.3 Sistema de Análise de Dados

Como a amostra de dados é muito grande, e se necessita que apresente o maior número de informação nas interfaces, deve-se efetuar um tratamento destes dados, antes que eles sejam inseridos na base de dados, e também depois, quando é efetuada uma consulta na base. Por isto que há o processo de análise de dados, no qual ele trata alguns dados que podem ser descartados na coleta, e também, ao consultar os dados, poder utilizar funções de agregação para apresentar apenas dados importantes para a visualização.

Quando são enviados os dados da camada de aquisição, periodicamente podem ser enviados dados incorretos ou redundantes. Então, ao coletar um dado, ele é analisado para verificar se este é um dado válido ou não.

Por exemplo, quando se é verificado o valor do campo de temperatura, se deve encontrar um valor acima dos 20º, que é o valor padrão da máquina em temperatura ambiente, que foi considerado no projeto. Por isso, quando um dado que apresenta menos que este valor é recebido no servidor, este dado é descartado para não ser considerado nas análises posteriores.

Quando, posteriormente, o usuário efetua uma requisição no servidor, é necessário que esse dado seja agregado antes que seja apresentado. A agregação funcionará como o processo da consulta dos dados na base de dados, efetuando cálculos de média nos dados.

Por exemplo, o usuário requisita uma amostra de dados com 5.000 documentos, o que seria muitos dados para serem apresentados em um gráfico apenas. Então, a agregação agrupa estes dados por intervalo de tempo - neste caso, millisegundos - e efetua um cálculo de média para cada agrupamento dos dados, diminuindo a quantidade de pontos do gráfico que serão mostrados.

Na Listagem 4.2 pode-se observar uma agregação do *MongoDB* do tipo *Match*, no qual ele filtra na base de dados todos os dados que estejam em um intervalo. O dado é requisitado com dois parâmetros, o *Timestamp* inicial e o *Timestamp* final. Este *Timestamp* é um valor numérico que representa a data exata de um intervalo de tempo. Esta marcação efetua uma consulta na base de apenas os dados que são maiores que o intervalo inicial, e

os valores menores que o intervalo final. Também, é inserido outra marcação que deve ser respeitada, que a temperatura retornada deve ser maior que 20 graus, para que não sejam considerados valores não importantes para a aplicação, como a temperatura ambiente.

```
{
    "$match": {
        "date": {
            "$gte": float(initial_timestamp),
            "$lte": float(final_timestamp)
        },
        "temp": {"$gte": 20}
    }
}
```

Listagem 4.2: Exemplo de uma consulta em intervalo de tempo

Conforme o intervalo de tempo de agrupamento é menor, mais documentos serão enviados para o usuário. Este intervalo é muito importante para manter a integridade dos dados, pois, caso o intervalo seja muito grande, será considerado uma quantidade muito grande de dados para apenas um agrupamento, então, a média do dado perderá sua precisão.

Na Listagem 4.3, pode-se visualizar o estilo de dado após analisado e transformado que é possível coletar. Esta agregação é do tipo *Group*, que prepara os dados no formato que serão utilizados de uma melhor forma. Este documento representa o retorno de um agrupamento que foi efetuado dos dados. Os dados contém as chaves de *seconds* e *milliseconds* para representar a divisão de cada grupo. Já nas chaves inseridas dentro dos dados do sensor, pode-se visualizar a temperatura e a pressão média do grupo, no qual são os dados mais importantes para a análise, pois representa a junção de todos os dados do conjunto, e divididos pela mesma quantidade.

```
{
    "$group": {
        "seconds": timestamp.seconds,
        "milliseconds": timestamp.milliseconds,
```

```
        "sensor_data": {
            "average_temperature": data.avgTemp,
            "average_pression": data.avgPress,
            "maximum_temperature": data.maxTemp,
            "maximum_pression": data.maxPress,
            "minimum_temperature": data.minTemp,
            "minimum_pression": data.minPress,
        }
        "date": timestamp
    }
}
```

Listagem 4.3: Exemplo de transformação e agrupamento dos dados

Também pode-se notar que existem dados máximos e mínimos, tanto em temperatura como em pressão. Estes dados também são importantes para a análise dos dados, pois representam qual foi o total da variação que foi obtida na apresentação média dos dados.

Estas análises são muito utilizadas no conceito de busca dos dados, pois quando fala-se de uma amostra muito grande de dados, não há como se ter uma quantidade tão grande de dados eficientemente. Na visualização dos dados, é preciso que os gráficos sejam claros e limpos, para que a percepção dos dados seja exata.

Todas estas análises foram inseridas neste contexto pois serão utilizados nos *Dashboards* de visualização dos dados. Sempre que o usuário efetuar requisições, os dados passam por estes processo, pois todos estes dados são importantes, tais como os valores de variação dos dados, como também a contagem de leituras de um determinado intervalo. Esta contagem é para verificarmos qual foi a quantidade de leituras que o sensor pôde fazer em um determinado tempo.

Para poder-se visualizar o processo como um todo, é necessário que as interfaces estejam implementadas para visualizar os dados. Esta etapa do presente projeto será abordada no próximo tópico.

## 4.4 Aplicações para visualização de dados

Para que a visualização dos dados seja feita, é necessária uma interface sólida e que contenha gráficos de visualização de dados nesta interface. Neste presente projeto, foram desenvolvidas duas interfaces gráficas que serão necessárias para a visualização dos dados: uma aplicação local que será utilizada em *hardwares* mais simplificados, e uma aplicação web que não terá a necessidade do tempo real para que se possa acessar dados já coletados.

### 4.4.1 Aplicação Local

Esta aplicação local foi desenvolvida em Python, utilizando a biblioteca Tkinter. A intenção principal é que a interface seja simples e que necessite pouco processamento do hardware, para que possa ser utilizado em ambientes com pouca disponibilidade de componentes, como computadores *Desktop*. A ideia é que seja implementado para ser utilizado diretamente no *Raspberry PI* do projeto, onde são coletados os dados dos sensores.

A interface foi implementada para necessáriamente aplicar os conceitos de visualização de dados em tempo real. Para isso, foi desenvolvido dois gráficos que representam, respectivamente, os dados de temperatura e os de pressão.

Alguns dos campos apresentam os dados máximos que foram alcançados desde a execução da aplicação, como os dados máximos de temperatura e de pressão, para que seja documentado o pico mais alto dos dados no presente momento. Estes campos servem para que o usuário fique informado sobre qual a temperatura ou pressão da máquina que está a receber no seu pico mais alto.

A aplicação permite que se selecione qual conjunto de sensores será abordado na amostragem. Por exemplo, em cada documento da base de dados, temos até quatro conjuntos de sensores conectados, que representam no total oito sensores, quatro para temperatura e quatro para pressão. Esta seleção é disponibilizada com um campo de seleção, no campo de sensores, como se pode visualizar na Figura 4.10. Esta é uma representação da interface em sua inicialização, enquanto ainda não recebeu dados para visualização.

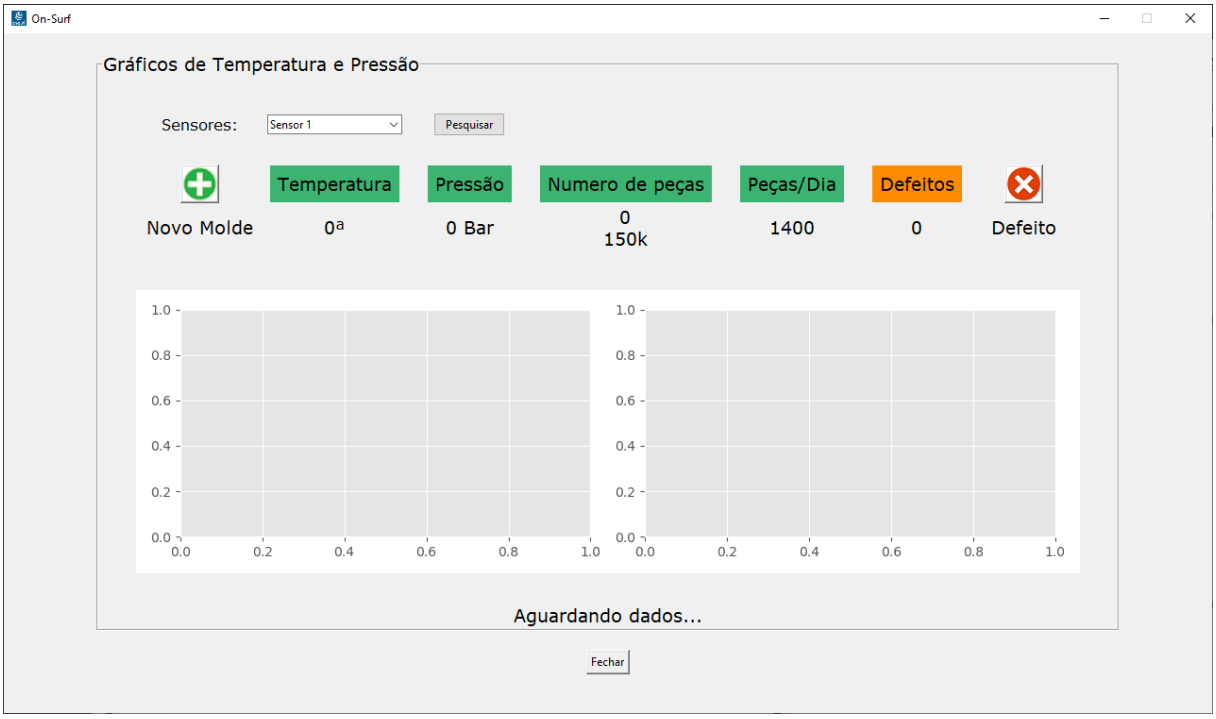


Figura 4.10: Interface em sua inicialização.

Quando os dados começam a serem enviados, a interface recebe uma requisição com os dados para serem apresentados nos gráficos. A interface então é atualizada com os gráficos de linha em sua apresentação, como podemos ver na Figura 4.11. Já também é possível visualizar o número de peças criadas pela máquina injetora de plástico, por meio de uma previsão pela temperatura. Sempre que a temperatura chega a um pico no gráfico, é incrementado o número de peças.

Os campos de temperatura e pressão que apresentam os dados máximos em execução da aplicação apresentam cores diferentes para melhor visualização. Quanto mais as cores se aproximam do vermelho, mais significa que os dados aumentaram nas últimas amostras. Os dados que foram projetados são com base no que foi definido pelo projeto, que são os limites da máquina injetora de plástico. Esta mudança de coloração é apresentada na Figura 4.12.

O que foi definido como requisito no projeto é que a máquina de injeção de plástico permite chegar até 280º de temperatura máxima, e 150 Bar de pressão. Quando

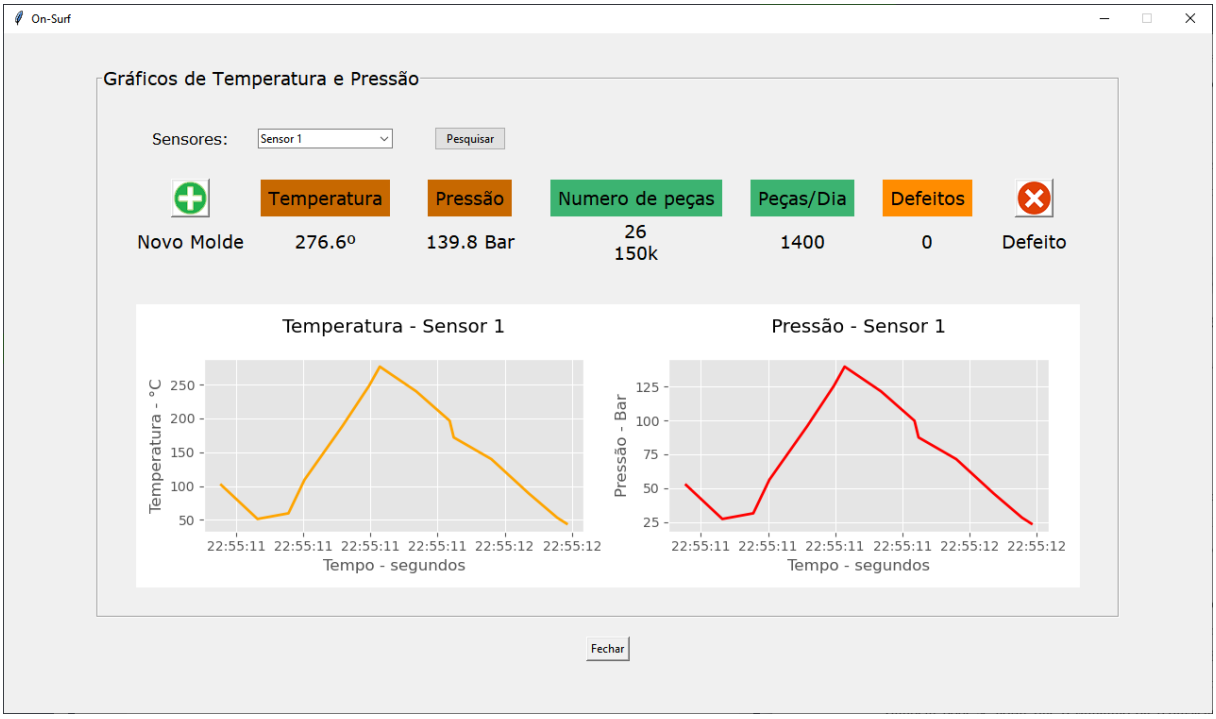


Figura 4.11: Interface após recebimento dos dados.

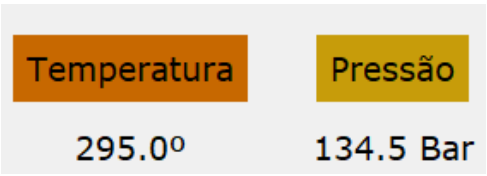


Figura 4.12: Campos de temperatura e pressão quando em execução.

ultrapassados estes dados, significa que a máquina está com irregularidades, e por isso a necessidade dos campos de avisarem o usuário que está operando a máquina. O ideal é que sempre a máquina não ultrapasse este valor, para que não seja gerado nenhum defeito nos moldes.

O seguinte campo ilustrado na Figura 4.13 apresenta a quantidade de defeitos que ocorreram nos moldes em tempo de execução da aplicação. Cada vez que um molde acaba por desenvolver um defeito ou problema, o usuário pode clicar no botão para registrar o atual momento em que foi encontrado um molde de plástico com defeito. Estes dados poderão servir para um trabalho futuro que avalie quais os níveis de temperatura e pressão que mais ocorrem defeitos nos moldes.


Figura 4.13: Campo de defeito de peças de plástico produzidas.

O botão de novo molde ilustrado na Figura 4.14 representa quando um novo molde é inserido na máquina para que seja injetado o plástico e criadas novas peças. Sempre que um molde é trocado na máquina, deve-se cadastrar este novo molde, o que reseta os dados atuais da aplicação, para que seja feita novas contagens nos sensores, e para conseguir efetuar posteriormente uma previsão da quantidade de tempo em que o molde suporta as elevadas temperaturas e pressões.

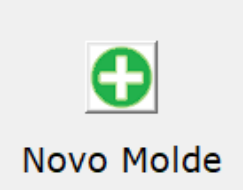


Figura 4.14: Botão de novo molde.

Esta aplicação não é necessária que a máquina tenha uma quantidade grande de processamento, e por isso, foi ajustado uma taxa de atualização para que os gráficos sejam atualizados. Os gráficos trabalham numa taxa de atualização de cerca de 1 segundo, pois é o mesmo tempo em que os sensores conseguem trabalhar na taxa de atualização dos dados para a base de dados.

### 4.4.2 Aplicação *Web*

O objetivo principal da Aplicação *Web* é o desenvolvimento de um dashboard que será utilizado para efetuar uma consulta profunda em dados que já foram cadastrados, sejam eles de dias ou semanas anteriores. É utilizado por padrão um filtro que gera um intervalo de um minuto para as amostras de dados. Estas amostras representam todos os documentos coletados em um intervalo de tempo para serem apresentados nos gráficos.

Já que a quantidade de dados é muito grande, e não se pode utilizar um intervalo muito grande devido as médias de valores utilizados, se é trabalhado com um intervalo de cerca de 1 minuto para que os dados sejam apresentados.

Ao consultar uma amostra de dados, os dados são analisados e a resposta é como é apresentado na Figura 4.15. Cada gráfico apresenta três linhas que são respectivamente, as funções desenvolvidas que geraram os dados máximos, médios e mínimos. A amostragem e variação dependerá da quantidade de dados em um mesmo intervalo. Ali se pode notar que os valores médios permanecem no meio do gráfico, enquanto os dados mínimos, na parte inferior, e os máximos, na parte superior.

Pode-se notar que em cada gráfico de linha há uma cor de fundo que representa os intervalos em que os dados estão em boa forma, na cor verde, e na parte superior, uma cor vermelha de cor de fundo para representar que estes dados estão atingindo valores acima do permitido.

Na parte superior, pode-se ver campos onde são apresentadas as temperaturas e pressões, máximas e mínimas, deste intervalo de tempo. Foi inserido ícones na espécie de "termômetro" que mudam conforme os dados são mais altos ou baixos. Isso facilita a

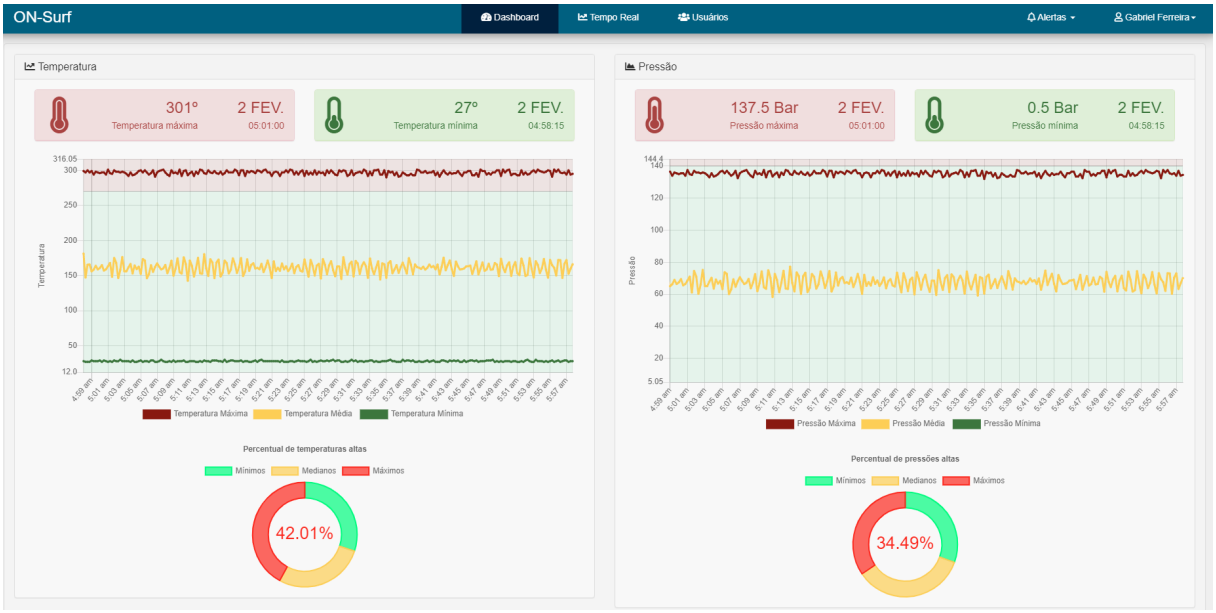


Figura 4.15: *Dashboard* em sua inicialização.

visualização e verificação dos dados conforme a necessidade.

Os gráficos circulares na parte inferior apresentam o total de contagens que o sensor efetuou em um determinado intervalo de tempo. Esses dados são determinados com valores fixos que representam os dados médios, máximos e mínimos, de acordo com os limites inseridos no projeto para classifica-los, que por exemplo, foram classificados para os dados de temperatura: maior que 280º é classificado como temperatura alta, entre 280º e 70º é classificado como temperatura média, e abaixo de 70º é classificado como temperatura baixa.

Os dados de pressão também tiveram seus valores classificados para uma melhor visualização. Os dados de pressão mais altos chegam acima dos 130 Bar. Já os mais baixos atingem entre 0 Bar e 30 Bar. Qualquer outro valor contado neste intervalo é classificado como pressão média na aplicação. Estes limites de temperatura e pressão foram estipulados pelos gestores do projeto.

Os gráficos possuem uma ampla flexibilidade de manipulação, como função de *zoom*, e efetuar seleções de quais linhas é preciso visualizar. Na Figura 4.16, ao selecionar os campos do filtro de data, pode-se visualizar os gráficos em *zoom* para melhor visualização

dos dados.

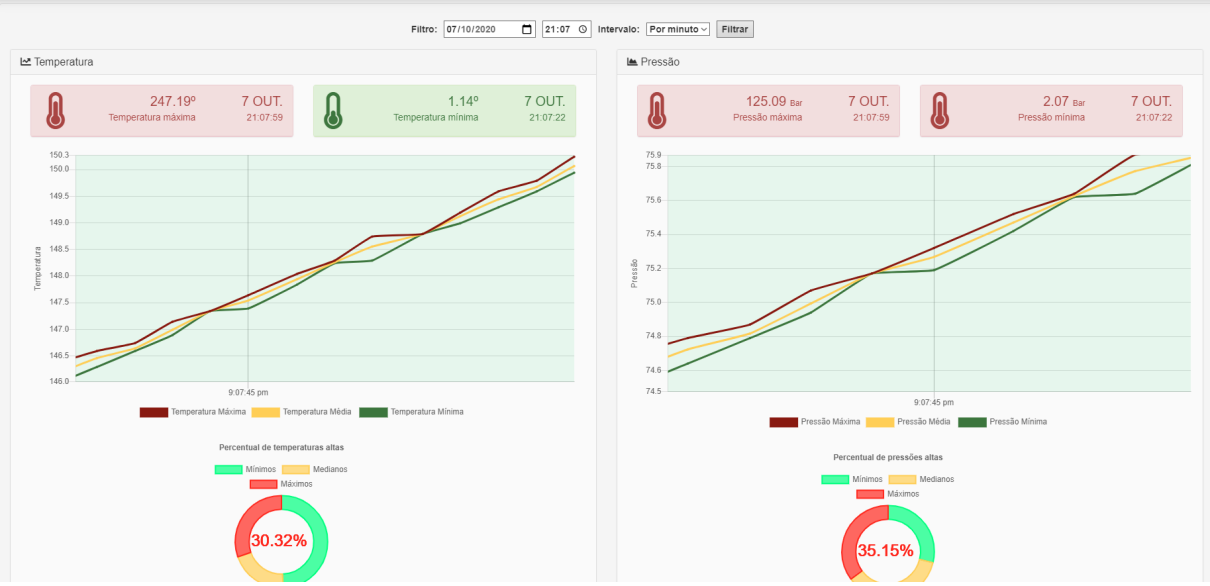


Figura 4.16: *Dashboard* com os gráficos em *zoom*.

Neste exemplo da Figura 4.17, é utilizada uma quantidade de leituras dos sensores que foram classificados como temperaturas mais altas que a média. No total, são cerca de 40.000 contagens neste intervalo em questão, e que destas 40.000 contagens, 17.023 foram classificadas como temperaturas mais altas que a média, ou seja, temperaturas altas. A porcentagem no centro do círculo representa este número de temperaturas altas, que neste contexto, é o dado mais importante para visualização.

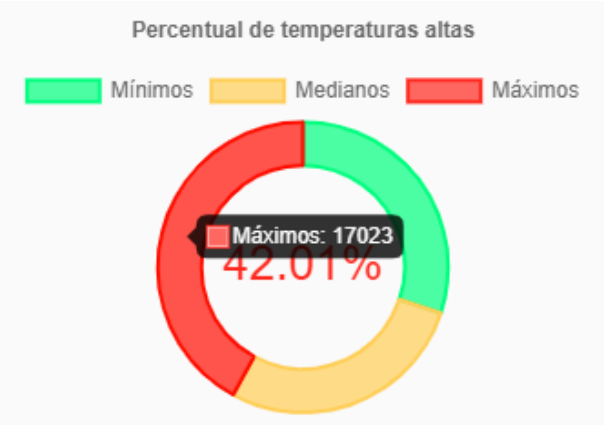


Figura 4.17: Gráfico circular e seu número de contagens.

Esta aplicação *Web* também conta com uma tela que funciona igualmente a aplicação local em tempo real desenvolvida em *Python*, porém, ela é disponibilizada em formato *Web* para o acesso remoto. Porém, diferentemente da aplicaão local, esta não dispõe de botões de novo molde e de defeito, pois não é o foco da aplicação *Web*.

A aplicação *Web* foca no histórico de todas as amostras de dados que já foram coletados, podendo assim, acessar de qualquer lugar e efetuar uma consulta de dados já inseridos anteriormente.

Quando um usuário administrador efetua *Login* desta aplicação, é possível também acessar a tela de usuários, que foi descrito no Tópico 4.1.5. Este usuário pode criar novos usuários para acessar a aplicação, e também efetuar edições nos usuários já anteriormente cadastrados.

# Capítulo 5

# Análise e Discussão de Resultados

Visto que as outras etapas do projeto ainda não tiveram conclusão, e não se conseguiu incluir um sensor real para os testes, estes testes foram efetuados com dados aleatórios.

## 5.1 Análise de desempenho

Para podermos visualizar os dados, a precisão e velocidade em que os dados são trazidos e tratados deve ser observada. Foi efetuado testes no desempenho da aplicação para garantir, que as requisições feitas são respondidas com um tempo mais otimizado possível.

Foram efetuados testes de desempenho em duas requisições, no qual são as mais importantes do sistema: As requisições em tempo real e as requisições dos dados analisados. As requisições em tempo real não necessitam de muito tratamento dos dados, pois o que importa é a velocidade e a precisão. Já os dados analisados exigem mais tempo para tratar os dados, pois é feito uma análise de um intervalo dos dados, sendo assim, capturando uma média geral dos dados deste intervalo.

### 5.1.1 Requisições em tempo real

Os dados das requisições em tempo real mostraram um desempenho rápido quanto as requisições efetuadas no sistema. O tamanho da amostra apresenta pouca quantidade de

*bytes*, facilitando a transferência destes dados para a interface gráfica.

Como é apresentado na Figura 5.1, percebe-se que as requisições tem um desempenho de cerca de 300 milissegundos, ou 0.3 segundos, para que os dados sejam recebidos. Nos testes, apresentou uma média de 300 *Kilobytes* de tamanho. Estes testes não conseguem apresentar o desempenho da interface renderizar os dados nos gráficos, pois isto depende do *Hardware* utilizado.


Figura 5.1: Resultados dos testes de desempenho das requisições em tempo real.

### 5.1.2 Requisições dos dados analisados

Como já citado, as requisições dos dados analisados necessitam de um tratamento maior dos dados, atrasando um pouco o desempenho das requisições. Porém, estes dados apresentam mais informações em cada documento, pois são calculados os valores médios dos dados de um determinado intervalo de tempo.

Nos dados analisados, pode-se notar na Figura 5.2, que as requisições apresentaram uma média de desempenho de cerca de 4 segundos por requisição. Estas requisições não necessitam de uma velocidade tão grande, pois estes dados serão aplicados no *Dashboard* com os dados já coletados anteriormente, e não em tempo real.


Figura 5.2: Resultados dos testes dos dados analisados.

Também pode-se notar que o tamanho da requisição é de cerca de 160 *Kilobytes*, o que apresenta uma menor quantia de dados, pois estes dados foram tratados.

## 5.2 Análise de carga de dados

Conforme os dados são coletados, a base de dados fica mais pesada e isto pode influenciar no sistema. Por isso, foram efetuados testes com uma grande carga de dados para que seja garantido a precisão dos dados e a velocidade das consultas. Também, podemos verificar o tamanho médio dos documentos que estão sendo coletados.

Na Figura 5.3, é apresentado os resultados da base de dados utilizada. Foram coletados até 11.7 milhões de dados, o que equivaleu a 1.7gb. O tamanho médio dos dados é de 158 *bytes*. A interface e as requisições não apresentaram problemas com esta carga de dados, o que é um resultado positivo, pois sabe-se então que a base de dados pode suportar uma grande quantia de dados.


Figura 5.3: Resultados dos testes dos dados analisados.

A base de dados é contínuamente utilizada para coletar os dados, portanto, este volume de dados pode aumentar com o tempo, e com isto, tornar base de dades mais volumosa.

## 5.3 Análise das interfaces

Visto que os dados coletados são dados simulados, não há uma precisão concreta em relação aos dados reais. Portanto, por meio de simulações, é possível verificar os dados que são coletados, e verificar se as respostas estão sendo apresentadas corretamente. A análise dos dados foi feita diretamente nos gráficos da aplicação.

### 5.3.1 Interface em tempo real

Os dados apresentados na Figura 5.4 foram coletados dos gráficos em tempo real. Após a requisição ser efetuada, a aplicação renderiza os campos e os gráficos, assim, apresentando os dados. Os dados são apresentados com precisão, e em um volume que não interfira na renderização.

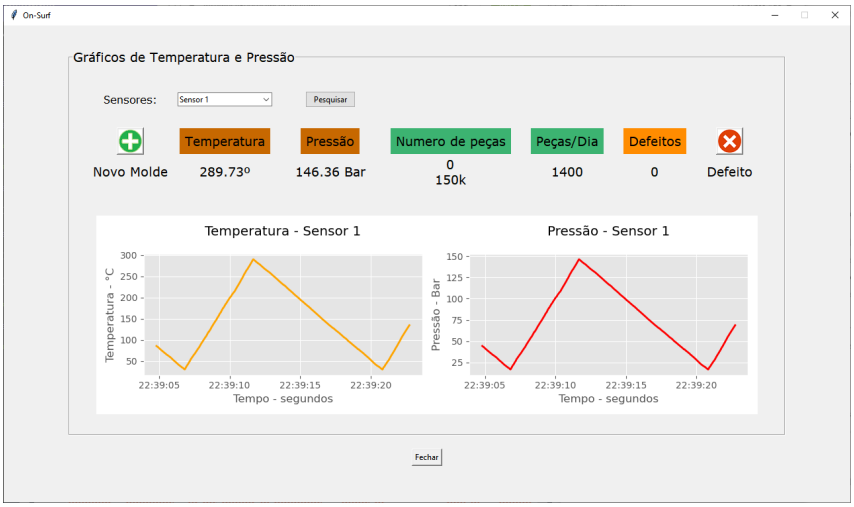


Figura 5.4: Análise da aplicação em tempo real.

Para melhor abstração, pode-se notar a Figura 5.5 em como os dados são analisados. Este gráfico de temperatura apresenta todas as contagens de documentos do sensor de temperatura. O eixo Y apresenta o valor da contagem, que no caso, é o valor da temperatura em graus. Já no eixo Y, é apresentado o exato momento da leitura. O gráfico consegue apresentar que o maior dado foi perto dos 300 graus, dentro de um intervalo de 20 segundos.

Assim como pode-se notar na Figura 5.6, as *labels* apresentam os maiores valores de temperatura e pressão da última contagem. Aqui é apresentado que nesta contagem, a temperatura alcançou a 287.73º e a pressão alcançou a 146.36 Bar. Estes dados são atualizados em cada rendererização da aplicação, para manter a veracidade dos dados. O número de peças também é incrementado a cada pico do gráfico, pois significa que uma nova peça foi criada.

O novo molde zera o número de peças, o que significa uma nova contagem de peças

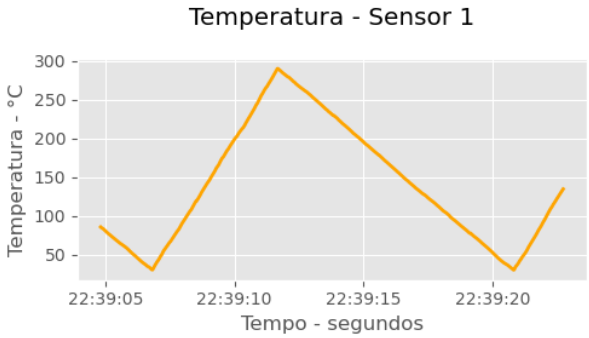


Figura 5.5: Resultados da análise dos dados do sensor de temperatura.

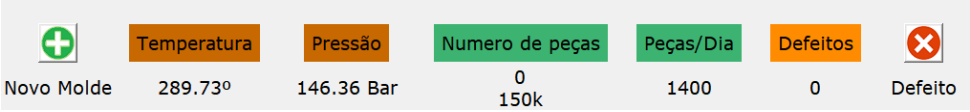


Figura 5.6: Resultados das *labels* da aplicação.

para este novo molde. Já para cada vez que é clicado em defeito, é incrementado um defeito na base de dados com o momento exato do defeito, para posteriores consultas.

### 5.3.2 Interface web

A aplicação *Web* renderiza todos os dados que são analisados, e por isso, há uma quantidade maior de informações no *Dashboard*. Os gráficos de temperatura e pressão são apresentados da mesma forma que na aplicação em tempo real, com a diferença que há duas linhas a mais que apresentam os dados máximos e mínimos da contagem.

Cada ponto do gráfico são renderizados três pontos, no qual os três são apresentados com o mesmo valor de tempo. O dado da linha superior normalmente ficará em um formato mais linear, pois sempre apresenta a maior contagem, o que significa que sempre apresentará o maior valor das três retas. Já o dado coletado médio apresentará um valor variável, pois pega todos os dados de uma amostra e efetua uma média total da amostra.

Na Figura 5.7 é apresentado de maneira precisa cada valor no gráfico, bem como também os dados máximos e mínimos da amostra. As *labels* conseguem apresentar o valor máximo e mínimo de cada gráfico, no intervalo selecionado. Pode-se notar então

que os dados no intervalo são de até 290.19º de temperatura, e 146.6 Bar de pressão.

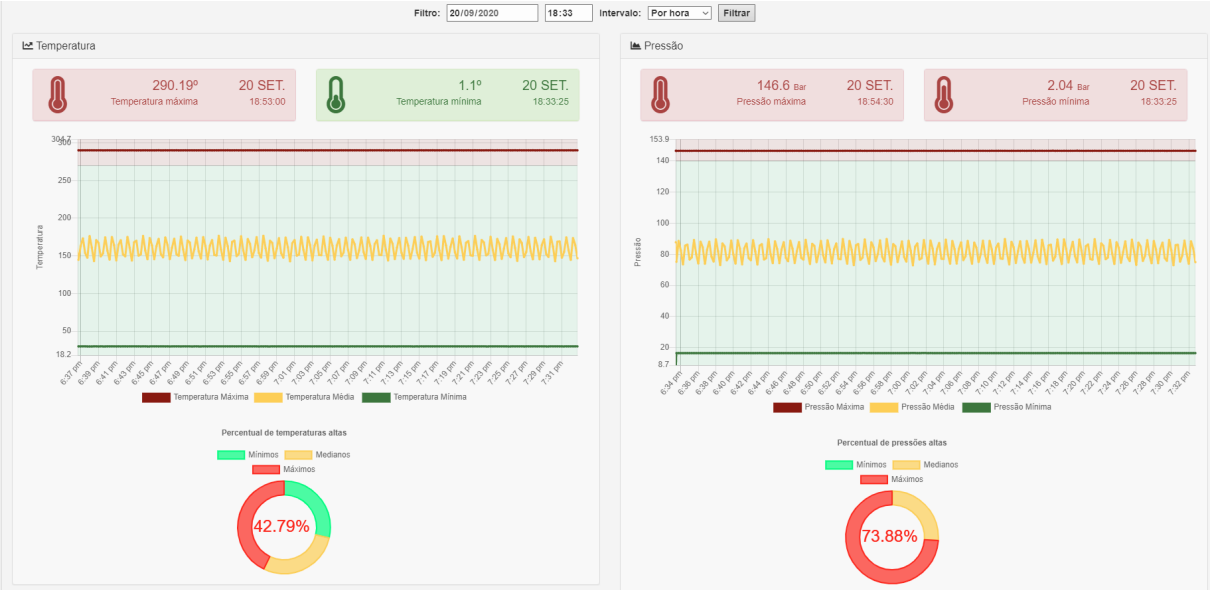


Figura 5.7: Resultados das *labels* da aplicação.

Os gráficos de *pizza* abaixo apresentam todas as contagens de cada intervalo. É concluído destes gráficos que 42.79% por cento das contagens de temperatura foram muito altas, e 73.88% das contagens de pressão também foram muito altas, o que pode ser preocupante caso esteja sendo contado estes dados de uma máquina injetora.

A aplicação *Web* também conta com uma página que simula os dados em tempo real, porém, a velocidade dos dados acaba sendo um pouco prejudicada pelo tempo de tráfego dos dados pela rede. Porém, no acesso remoto, mesmo que haja este atraso, não influencia nos dados coletados pelos sensores, pois o tratamento dos dados é o mesmo que ocorre com a aplicação local.

Outra página que a aplicação web contém, é uma página de monitoramento dos usuários, pois para aceder a aplicação *Web*, é necessário um cadastro. Como explicado no tópico 4.1.5, existem níveis de usuário, e este controle de usuários deve ser feito por um usuário administrador do sistema. Os usuários são cadastrados e editados por uma *modal* que apresenta seus dados em campos, e assim, efetuam uma requisição de cadastro. Novos usuários podem apenas ser cadastrados e editados pelo administrador do sistema.

# Capítulo 6

# Conclusões

Neste capítulo será apresentado as conclusões dos estudos para o presente trabalho, dando importância para os tópicos mais importantes para o trabalho, e uma breve descrição dos trabalhos futuros sugeridos para serem desenvolvidos com base no sistema.

## 6.1 Considerações finais

Neste trabalho foram apresentados dois modelos de análise e visualização de dados, aplicados para devidos fins, detalhando a trajetória do trabalho, que por conta do grande volume de dados e da velocidade que os dados devem ser coletados, e pela quantidade de abordagens que o *Big Data* abrange, implicou um estudo para desenvolver uma plataforma em que estes dados sejam tratados da melhor forma possível. Desta forma, o projeto foi focado nas específicações da terceira etapa do contexto do projeto ON-Surf, no qual será aplicado. Com base nos objetivos do estudo, o projeto se adequou às necessidades apresentadas, e apresentou uma perfomance desejável ao que foi proposto principalmente, no contexto de visualização de dados em tempo real.

Como pode-se perceber na análise dos dados, e nos testes efetuados no sistema, o sistema apresentou um resultado consideravelmente bom no tempo de resposta dos dados, no qual é o fator mais importante para aplicações em tempo real. A carga grande de dados foi um problema a ser solucionado, pois quanto maior a carga de dados por segundo, mais

o sistema apresenta um atraso na visualização. Porém, com o tratamento destes dados, a aplicação consegue fornecer os dados em uma velocidade ótima mesmo quando a carga de dados é mais alta do que o normal.

A carga de dados não resultou também em problema nos testes, visto que a visualização dos dados não foi prejudicada por este fator. Na aplicação *Web*, pode-se perceber que mesmo com a influência da rede de *internet* a prejudicar, foi cumprida a necessidade de poder visualizar um histórico dos dados anteriores com base em um filtro de data e hora. Assim, o usuário ou gestor do sistema poderá visualizar um determinado período de tempo e verificar quais os dados são importantes para sí.

Infelizmente, um fator que prejudicou nos estudos, e nos testes com sensores reais foi devido a pandemia do Covid-19, que afetou o mundo inteiro, o que ocasionou um afastamento social no qual influenciou nos prazos propostos. Porém, mesmo com toda a situação, o modelo não deixou de ser testado com diferentes dados para simular diferentes situações.

## 6.2 Trabalhos futuros

Levando-se em consideração o modelo proposto, para trabalhos futuros, propõe-se o desenvolvimento de métricas para os dados reais, pois, com o passar do tempo, o sistema coletará diversos dados de diferentes valores, o qual pode ser desenvolvido filtros para estes dados para que sejam inseridos apenas dados relevantes para a aplicação. Isto fará com que a validade dos dados seja maior, pois poderá ser feito análises dos dados e desconsiderar, por exemplo, medições coletadas da temperatura ou pressão ambientes.

A intenção deste trabalho é também desenvolver posteriormente e com dados reais, um modelo de *Machine Learning* que possa analisar todos os dados coletados e os que estão sendo coletados, e assim, efetuar previsões a respeito dos moldes das máquinas, prevendo caso os moldes já estão em tempo de manutenção, e fazendo outros tipos de métricas que poderão ser previstas.

Outro trabalho proposto é que, com o passar do tempo, a quantidade de dados ficará

um tanto absurda, o que faltará espaço de *hardware* para armazenamento. Por isso, uma forma de se adequar a isso é desenvolvendo um módulo de *Data Cleaning*, efetuando uma limpeza periódica na base de dados, guardando dados históricos agregados, e só os mais relevantes para análises futuras.

# Bibliografia


[1] *ON-Surf*, `http://onsurf.teandm.pt/`, Accessed: 2020-10-04.

[2] J. Gantz e D. Reinsel, "Extracting Value from Chaos", *IDC iView*, pp. 1–12, 2011.

[3] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen e S. Belfkih, "Big Data technologies: A survey", *Journal of King Saud University - Computer and Information Sciences*, n.º 30, pp. 431–448, 2018.

[4] C. P. Chen e C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data", *Information sciences*, vol. 275, pp. 314–347, 2014.

[5] X. Wu, X. Zhu, G.-Q. Wu e W. Ding, "Data mining with big data", *IEEE transactions on knowledge and data engineering*, vol. 26, n.º 1, pp. 97–107, 2013.

[6] D. Laney, "3D Data Management: Controlling Data Volume, Velocity and Variety", *META Group Research Note*, 2001.

[7] P. Russom, "Big Data Analytics", *TDWI RESEARCH*, 2001.

[8] A. Gandomi e M. Haider, "Beyond the hype: Big data concepts, methods, and analytics", *International journal of information management*, vol. 35, n.º 2, pp. 137–144, 2015.

[9] M. Chen, S. Mao e Y. Liu, "Big data: A survey", *Mobile networks and applications*, vol. 19, n.º 2, pp. 171–209, 2014.

[10] D. M. Ribeiro et al., "Visualização de dados na Internet", tese de doutoramento, Dissertação (Mestrado em Tecnologias da Inteligência e Design Digital), 2009.

[11] A. Hahn, "Data visualization", *Salem Press Encyclopedia*, 2018.

[12] *Microsoft PowerBI*, `https://powerbi.microsoft.com/pt-pt/`, Accessed: 2020-10-07.

[13] *Grafana*, `https://grafana.com/`, Accessed: 2020-10-07.

[14] *Thingsboard*, `https://thingsboard.io/`, Accessed: 2020-10-04.

[15] *Matplotlib*, `https://matplotlib.org`, Accessed: 2020-07-11.

[16] J. Gubbi, R. Buyya, S. Marusic e M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions", *Future generation computer systems*, vol. 29, n.º 7, pp. 1645–1660, 2013.

[17] M. Satyanarayanan, "Pervasive computing: Vision and challenges", *IEEE Personal communications*, vol. 8, n.º 4, pp. 10–17, 2001.

[18] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[19] M. Platzer e P. Puschner, "A Real-Time Application with Fully Predictable Task Timing", em *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, 2020, pp. 43–46.

[20] K. Ramamritham, "Real-time databases", *Distributed and parallel databases*, vol. 1, n.º 2, pp. 199–226, 1993.

[21] D. C. Schmidt, M. Deshpande e C. O'Ryan, "Operating System Performance in Support of Real-Time Middleware.", em *WORDS*, 2002, pp. 199–206.

[22] P. Atzeni, F. Bugiotti e L. Rossi, "Uniform access to non-relational database systems: The SOS platform", em *International Conference on Advanced Information Systems Engineering*, Springer, 2012, pp. 160–174.

[23] N. Jatana, S. Puri, M. Ahuja, I. Kathuria e D. Gosain, "A survey and comparison of relational and non-relational database", *International Journal of Engineering Research & Technology*, vol. 1, n.º 6, pp. 1–5, 2012.

[24] U. Bhat e S. Jadhav, "Moving towards non-relational databases", *International Journal of Computer Applications*, vol. 1, n.º 13, pp. 40–47, 2010.

[25] J. Han, E. Haihong, G. Le e J. Du, "Survey on NoSQL database", em *2011 6th international conference on pervasive computing and applications*, IEEE, 2011, pp. 363–366.

[26] *Python*, `https://www.python.org`, Accessed: 2020-07-25.

[27] *Flask*, `https://flask.palletsprojects.com/en/1.1.x/`, Accessed: 2020-07-25.

[28] M. Grinberg, *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.

[29] *MongoDB*, `https://www.mongodb.com`, Accessed: 2020-07-27.

[30] K. Banker, *MongoDB in action*. Manning Publications Co., 2011.

[31] K. Chodorow, *MongoDB: the definitive guide: powerful and scalable data storage*. "O'Reilly Media, Inc.", 2013.

[32] *Tkinter*, `https://docs.python.org/3/library/tkinter.html`, Accessed: 2020-07-28.

[33] J. E. Grayson, *Python and Tkinter programming*. Manning, 2000.

[34] *Angular*, `https://angular.io`, Accessed: 2020-07-28.

[35] B. Green e S. Seshadri, *AngularJS*. "O'Reilly Media, Inc.", 2013.

[36] J. D. Hunter, "Matplotlib: A 2D graphics environment", *Computing in Science & Engineering*, vol. 9, n.º 3, pp. 90–95, 2007. DOI: `10.1109/MCSE.2007.55`.

[37] *Chart.js*, `https://www.chartjs.org`, Accessed: 2020-07-27.

[38] H. Da Rocha, *Learn Chart. js: Create interactive visualizations for the web with chart. js 2*. Packt Publishing Ltd, 2019.

[39] J. He e S.-H. G. Chan, "TCP and UDP performance for Internet over optical packet-switched networks", *Computer Networks*, vol. 45, n.º 4, pp. 505–521, 2004.

[40] *Socketio*, `https://socket.io`, Accessed: 2020-07-25.