Data lake para agregação de dados de produção e ferramentas de visualização na indústria de estampagem

Leonardo Leite Meira dos Santos - 54363

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:

Prof. Paulo Alves

Prof. Kecia Marques

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2022-2023

Data lake para agregação de dados de produção e ferramentas de visualização na indústria de estampagem

Leonardo Leite Meira dos Santos - 54363

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:

Prof. Paulo Alves

Prof. Kecia Marques

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri.

Bragança

2022-2023

Dedicatória

(Facultativo) Dedico este trabalho a ...

Agradecimentos

(Facultativo) Agradeço a ...

Resumo

O resumo (no máximo com 250 palavras), permite a avaliação do interesse de um documento e facilita a sua identificação na pesquisa bibliográfica em bases de dados onde o documento se encontre referenciado.

É recomendável que o resumo aborde, de forma sumária:

- Objetivos principais e tema ou motivações para o trabalho;
- Metodologia usada (quando necessário para a compreensão do relatório);
- Resultados, analisados de um ponto de vista global;
- Conclusões e consequências dos resultados, e ligação aos objetivos do trabalho.

Como este modelo de relatório se dirige a trabalhos cujo foco incide, maioritariamente, no desenvolvimento de software, algumas destas componentes podem ser menos enfatizadas, e acrescentada informação sobre análise, projeto e implementação do trabalho.

O resumo não deve conter referências bibliográficas.

Palavras-chave: termos (no máximo 4), que descrevem o trabalho.

Abstract

Direct translation (maximum of 250 words) to English of the section “Resumo”.

Keywords: direct translation of “Palavras-chave”

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Enquadramento

No cenário industrial contemporâneo, a busca constante por eficiência e inovação tornou-se um pilar essencial para a competitividade e sustentabilidade financeira das empresas. À medida que a tecnologia avança, a produção industrial sofre uma evolução para se manter competitiva perante ao mercado. Nesse contexto, o monitoramento e a otimização das máquinas em linhas de produção tornam-se cruciais para garantir um funcionamento eficaz e prevenir potenciais paralisações ou falhas operacionais.

No entanto, a tradição das práticas industriais muitas vezes é caracterizada por inspeções manuais e sistemas de monitoramento desatualizados, que não conseguem fornecer informações em tempo real ou análises aprofundadas sobre o desempenho das máquinas. Esse descompasso tecnológico pode resultar em perdas significativas, em termos de produção, de recursos financeiros e manutenção de máquinas.

Além disso, com a crescente integração de sistemas de IoT e a proliferação de sensores avançados, existe uma quantidade imensa de dados sendo gerada continuamente. No entanto, sem a infraestrutura adequada para coletar, armazenar e analisar esses dados, as empresas podem se encontrar sobrecarregadas, incapazes de extrair insights significativos que poderiam informar decisões estratégicas e operacionais.

Com esse contexto de mercado, a empresa Catraport buscou a construção de projetos que realizem a sensorização de suas maquinas de estampagem, armazenamento e tratamento dos dados e visualização dessas informações, de forma a se tornar mais competitiva, eficiente e lucrativa.

1.2 Objetivos

Dado o enquadramento anterior, a necessidade identificada é de desenvolver um sistema robusto que possa receber dados de sensores que coletam dados das máquinas em tempo real, armazená-los de maneira eficiente em um data lake e apresentá-los através de um dashboard, transformando a maneira como as empresas monitoram e otimizam suas linhas de produção, garantindo não apenas eficiência, mas também uma abordagem proativa à manutenção e gestão industrial. Esse sistema não apenas forneceria informações em tempo real sobre o status e o desempenho das máquinas, mas também permitiria análises históricas, ajudando gestores e técnicos a identificar tendências e falhas, além de otimizar a produção, minimizando perdas na produção e otimizando os ganhos financeiros.

1.3 Estrutura do Documento

Esta dissertação de mestrado está organizada começando pela Introdução, onde o problema e o escopo são apresentados. Esta seção contextualiza a necessidade de modernização industrial, destacando a importância do monitoramento e otimização de máquinas na indústria. A relevância do estudo é então justificada, com foco na crescente demanda por análise de dados no mercado.

Segue-se com a Revisão Literária, que apresenta uma análise sobre trabalhos e conceitos relacionados ao projeto. Esta seção engloba desde trabalhos semelhantes já realizados, métodos de análise de dados, até discussões sobre visualização de dados e importância da gestão de dados no mercado atual.

A Metodologia discute a escolha de tecnologias específicas usadas no projeto, detalhando o método de armazenamento de dados, e o processo adotado para o desenvolvimento do software. Também são descritas as estratégias de gestão das atividades do projeto e os desafios enfrentados durante a coleta de dados.

No capítulo sobre a Arquitetura do Sistema, é discutido sobre as especificidades técnicas do sistema proposto. Desde o diagrama geral do sistema até a implementação de princípios de codificação, como o SOLID, esta seção visa elucidar o design e a funcionalidade do software criado.

A seção de Implementação aprofunda-se nos aspectos técnicos do sistema. Aqui, cada componente do banco de dados, da API e do módulo de processamento de dados é detalhadamente descrito. Também se discute a reutilização potencial do sistema em outros contextos, e o que seriá necessário para isso.

No que diz respeito às Características do Sistema do ponto de vista funcional, a dissertação tem uma seção que foca em mostrar a aplicação prática do software, utilizando capturas de tela e diagramas para demostrar as funcionalidades.

Em Resultados e Avaliação, são apresentados os resultados obtidos durante o projeto, destacando as principais realizações e benefícios observados na implementação do sistema proposto. A Conclusão e Trabalhos Futuros resume os pontos chave do projeto, identifica as limitações do sistema atual e sugere possíveis direções para continuidade e implementações futuras.

Capítulo 2

Revisão literaria

In this chapter it is expected to have a generic description of the problem and area: scope, concepts and technology and/or a literature review (state-of-the-art). In case of a practical project, there should also be described the tools and the justification for their use.

Usually, this chapter is divided in multiple sections, to complement the topics.

trabalhos semelhantes já realizados, métodos de análise de dados, até discussões sobre visualização de dados e importância da gestão de dados no mercado atual

2.1 Nome do artigo

Texto...

Capítulo 3

Methodology

A seção de metodologia explora uma abordagem para o desenvolvimento do software, iniciando com a definição dos requisitos, passando pela definição do método de recebimento dos dados, e a seleção das tecnologias. O processo de desenvolvimento incorpora a criação de histórias de usuário, organizadas em um quadro Kanban para gerenciamento de tarefas, e à configuração dos repositórios. A documentação e reuniões regulares que acompanham o progresso, são uma constante em todas as fases para garantir que o projeto está ocorrendo como o esperado.

3.1 Definição dos requisitos

A definição precisa dos requisitos é fundamental para garantir que o sistema desenvolvido atenda às necessidades e objetivos. Os requisitos desse projeto foram classificados nas categorias funcionais e não funcionais, para garantir uma compreensão completa do que é esperado do sistema.

3.1.1 Requisitos Funcionais

Os requisitos funcionais desempenham um papel fundamental no desenvolvimento de sistemas, definindo as funções que um sistema ou componente de software deve ser capaz

de executar. Em essência, eles fornecem uma descrição das interações que o sistema terá com seus usuários ou com outros sistemas, especificando os serviços que o sistema deve fornecer.

Para garantir eficácia, os requisitos funcionais devem ser claramente definidos, sem ambiguidades, e serem mensuráveis, rastreáveis, completos e consistentes. Além disso, eles devem ser definidos levando em consideração as necessidades e objetivos do projeto, garantindo que o sistema desenvolvido seja não apenas tecnicamente sólido, mas também útil e relevante para seus usuários finais.

Dentro do contexto do sistema desenvolvido, está listado abaixo os requisitos funcionais do sistema e sua descrição, de forma a deixar claro, de forma funcional, o que o sistema deve fazer.

FR1 - O sistema deve permitir que um usuário acesse o sistema de forma segura com um email e senha

Dado que o sistema é para a visualização dos dados de operação de uma industria de estampagem, as informações disponibilizadas devem ser acessadas apenas por usuários previamente autorizados.

FR2 - O sistema deve permitir que um usuário veja as suas informações pessoais que são armazenadas nos sistema

Cada usuário que tiver acesso ao sistema terá registrado nele alguns de seus dados pessoais, como e-mail, cargo e tipo de acesso. Portanto, cada usuário deve ter acesso a suas informações pessoais que estão salvas no sistema.

FR3 - O sistema deve exibir em tempo real os valores lidos pelos sensores em cada uma das máquinas

Ao receber os dados enviados pelos sensores, os sistema deve exibir em tela os valores lidos, separado por tipo de sensor e máquina.

FR4 - O sistema deve armazenar um valor máximo ideal para cada para tipo de sensor utilizado

Cada sensor deverá ter um valor máximo ideal para o funcionamento. Ele servirá de parâmetro para entender se o valor lido pelo sensor indica um bom ou mal funcionamento da máquina.

FR5 - O sistema deve identificar sempre que um valor lido pelo sensor não estiver abaixo do valor ideal

Esse requisito refere-se à capacidade do sistema de detectar, de forma automática, toda vez que o sensor indicar um valor que não esteja abaixo dO limite pré definido. Isto é, se o valor ideal for X, e o sensor ler um valor maior ou igual a X, o sistema reconhecerá esta situação.

FR6 - O sistema deve registrar sempre que um valor lido pelo sensor não estiver de acordo com o valor ideal

Esse requisito implica que o sistema deve manter um registro de todos os momentos em que o valor detectado pelo sensor não estiver abaixo do valor ideal armazenado.

FR7 - O sistema deve mostrar em tela quando um valor lido pelo sensor não estiver abaixo do ideal

Sempre que o sensor detectar um valor abaixo do padrão ideal, o sistema deverá exibir um alerta na interface de forma que fique sempre visível para o usuário.

FR8 - O sistema deve mostrar no formato de notificação os registros de não funcionamento abaixo do valor ideal

Este requisito estabelece que o sistema deve apresentar aos usuários na forma de notificações quando o sensor ler um valor acima do ideal, para possibilitar que os usuários sejam informados, mesmo que posteriormente, sempre que um alerta for identificado.

FR9 - O sistema deve permitir que o usuário marque uma notificação como lida, de forma que ela não apareça novamente

Após ser notificado, os usuários deverão ter a capacidade de marcar essa notificação como "lida", garantindo que a mesma informação não continue a ser exibida repetidamente.

FR10 - O sistema deve exibir gráficos mostrando os valores lidos pelos sensores nos dias anteriores de forma agregada, separando por máquinas

Esse requisito assegura que os usuários possam visualizar, por meio de representações gráficas, as leituras dos sensores de dias anteriores de forma agregada. Estes gráficos devem ser categorizados por máquina, proporcionando uma análise detalhada do desempenho de cada equipamento ao longo do tempo.

FR11 - O sistema deve exibir nos gráficos uma análise estatística do funcionamento das máquinas, junto com o valor máximo de funcionamento ideal

Os gráficos devem oferecer uma análise estatística, mostrando os indicadores estatísticos dos dados agregados média, mediana, percentil 75 e média removendo os outliers. Juntamente com isso, o gráfico também mostrará o valor ideal, servindo como uma referência para avaliar o desempenho.

FR12 - O sistema deve permitir filtrar as informações exibidas em tela por máquinas

Os usuários devem ter a flexibilidade de selecionar e visualizar informações específicas para determinadas máquinas, permitindo que eles se concentrem em equipamentos específicos conforme a necessidade.

FR13 - O sistema deve permitir filtrar os gráficos exibidas em tela por data

O sistema deve oferecer a capacidade de os usuários filtrarem as exibições gráficas por datas específicas, permitindo análises temporais detalhadas e comparações entre diferentes

períodos.

FR13 - O sistema deve exibir os gráficos de parada das máquinas de forma a exemplificar a exibição desses dados

O sistema deve mostrar as paradas de máquina de acordo com os dados passados pelas planilhas com os dados. Dessa forma poderá ser exemplificado como ficaria as informações de parada das maquinas caso o sistema recebesse essas informações.

3.1.2 Requisitos Não Funcionais

Requisitos não funcionais são especificações que determinam as características de desempenho, usabilidade, confiabilidade e outras propriedades que o sistema deve possuir, ao invés de comportamentos específicos que ele deve demonstrar. Enquanto os requisitos funcionais descrevem o que um sistema deve fazer, os requisitos não funcionais especificam como o sistema deve realizar essas funções.

Estes requisitos são cruciais para garantir a satisfação do usuário e a eficácia operacional do sistema, desempenhando um papel fundamental na qualidade e na operação geral de um produto de software.

Os requisitos não funcionais podem ser de vários tipos, como usabilidade, desempenho, segurança, disponibilidade, manutenção e confiabilidade. Dentro do contexto do sistema desenvolvido, está listado abaixo os requisitos não funcionais do sistema e sua descrição, de forma a deixar claro o que foi levado em consideração no momento de desenvolver cada uma das funcionalidades do sistema.

NFR1 - Disponibilidade

O sistema deve possuir mecanismos de reconexão automática que se ativam quando problemas de conexão ou recebimento de dados dos sensores forem detectados, garantindo assim a continuidade no recebimento dos dados.

NFR2 - Segurança no acesso

O sistema deve implementar controles de acesso para que somente colaboradores autorizados tenham permissão para acessar os dados e funcionalidades pertinentes ao seu papel.

NFR3 - Segurança na rede

Para garantir a segurança da transmissão de dados, a conexão ao sistema deve ser estabelecida utilizando o protocolo HTTPS, que incorpora a camada de segurança TLS, protegendo assim os dados contra intercepções e alterações.

NFR4 - Transmissão em tempo real

O sistema deve processar e transmitir os dados enviados pelos sensores em uma arquitetura baseada em streaming. O atraso entre o envio do dado pelo sensor e sua visualização pelo usuário final deve ser inferior a três segundos.

NFR5 - Modularidade

A arquitetura do sistema deve ser modular, possibilitando a integração e a adição de novos componentes ou funcionalidades de maneira eficiente e sem comprometer o funcionamento das partes já existentes.

NFR6 - Manutenibilidade

Priorizando a longevidade e facilidade de manutenção, o sistema deve ser desenvolvido seguindo boas práticas de programação e os princípios do SOLID. Isso facilitará futuras modificações, expansões e a correção de eventuais problemas.

NFR7 - Escalabilidade de sensores e máquinas

O design do sistema deve ser capaz de lidar com um crescente volume de sensores e máquinas, garantindo que não haja degradação de performance ou falhas quando a demanda por recursos aumentar.

NFR8 - Portabilidade

O sistema deve garantir compatibilidade com os principais navegadores web disponíveis no mercado. Além disso, a interface de usuário deve se adaptar bem em telas maiores como televisões, permitindo que o dashboard seja visualizado de forma clara em diferentes ambientes da fábrica.

NFR9 - Usabilidade

A interface do sistema e seus componentes devem ser projetados considerando princípios fundamentais de design de interação, garantindo que os usuários possam compreender e interagir com o sistema de maneira intuitiva e eficiente.

3.2 Método de coleta e armazenamento de dados

Dentro do contexto do projeto, a forma como os dados dos sensores são coletados e armazenados influenciam muito no funcionamento do sistema, pois é a partir deles que todo o sistema é estruturado. Dessa forma foi utilizado como base um protocolo desenvolvido em outro projeto, que transmite todas as informações necessárias para o contexto desse projeto. Dentro do sistema em questão, ficou a responsabilidade de implementar o decodificador para o determinado protocolo.

O protocolo utilizado foi desenvolvido dentro do mesmo contexto desse projeto, dentro do IPB, para atender uma demanda da mesma empresa, portanto ele se tornou a opção mais ideal, otimizando a comunicação entre os sensores e o sistema. O formato é estruturado de forma a representar as informações pertinentes à máquina, ao tipo de comunicação, ao sensor e ao significado dos dados transmitidos, seguindo o seguinte formato:

Machine ID (2 bytes)

O campo *Machine ID* é responsável por identificar a máquina em questão e está dividido em dois subcampos:

- **High (bytes de ordem superior)**: Representa o tipo da máquina. Exemplos de valores possíveis são: prensa, torno, robot, tapete, entre outros.
- **Low (bytes de ordem inferior)**: Identifica o número da máquina.

Type (1 byte)

O campo *Type* indica o tipo de mensagem e pode assumir os seguintes valores:

1. Publish
2. Request to publish

Sensor ID (2 bytes)

O campo *Sensor ID* fornece detalhes sobre o sensor que está transmitindo os dados:

- **High (bytes de ordem superior)**: Representa a quantidade física sendo medida, como temperatura, velocidade, pressão, força, entre outros.
- **Low (bytes de ordem inferior)**: Indica o número do sensor.

Meaning of Data (2 bytes)

O campo *Meaning of Data* fornece informações sobre o tipo e significado dos dados:

- **High (bytes de ordem superior)**: Tipo dos dados:
    1. Not defined
    2. Normal
    3. Raw data

4. Alarm

- **Low (bytes de ordem inferior)**: Significado dos dados, que varia de acordo com o equipamento. Exemplos incluem:
  - Oil critical temperature
  - Check oil temperature
  - Oil pressure

Length (2 bytes)

O campo *Length* indica o número de bytes subsequentes no pacote.

Data

Este campo representa os dados transmitidos pelo sensor. A especificação exata do que os dados representam deve ser definida e normalizada, conforme indicado pela notação (*).

3.3 Processo de desenvolvimento do software

Com a definição clara dos requisitos do sistema e da forma como os dados lidos pelos sensores são transmitidos, foi definido o processo de como o projeto seria desenvolvido. Esse processo de desenvolvimento passa pela definição das histórias de usuário, organização das atividades, organização da documentação, configuração dos repositórios no GitHub e reuniões periódicas com o professor e com a empresa para discutir o andamento.

3.3.1 Histórias de Usuário

No desenvolvimento ágil de software, uma das abordagens mais centradas no usuário entendimento das funcionalidades do sistema e dos requisitos é a utilização de *histórias de usuário* (ou *user stories* em inglês). Estas são descrições curtas, simples e informais

do ponto de vista de um usuário final, capturando o que eles necessitam ou desejam fazer no software.

A estrutura típica de uma história de usuário é: "Como [tipo de usuário], eu quero [uma ação] para que [um benefício/resultado]". Esta estrutura ajuda a manter o foco nas necessidades e desejos do usuário, em vez de mergulhar prematuramente em soluções técnicas ou detalhes de implementação.

Além de serem uma ferramenta de comunicação entre os desenvolvedores e os stakeholders, as histórias de usuário facilitam a priorização das tarefas, ajudam na criação de critérios de aceitação e fornecem uma base para discussões interativas durante as reuniões de revisão e planejamento.

Em suma, as histórias de usuário servem como um meio eficaz de traduzir requisitos complexos em tarefas gerenciáveis e centradas no usuário, garantindo que o produto final atenda às expectativas e necessidades dos seus utilizadores.

Com essa definição de historias de usuário, foi definido os seguintes itens que traduzem os requisitos em tarefas para o desenvolvimento do projeto:

1. As a user, I must be able to log in with my credentials to use the system.

2. As a user, I should be able to view my personal information on a profile page to manage the data the system holds about me.

3. As a user, I should be able to view in real-time values of the machines to detect relevant variations in operation more quickly.

4. As a user, I should be able to view the historical values of the sensors aggregated in charts of the machines to monitor the status over time.

5. As a user, I should be able to filter the dashboard information by machine and by date to view data according to my needs.

6. As a user, I should be alerted when a sensor reads a value that exceeds the ideal parameter so I can take necessary actions as quickly as possible.

7. As a user, I should be able to view system notifications to be alerted about machine operation alerts.

8. As a user, I should be able to view the machine downtime records for a better and more organized view of the recorded machine downtimes.

9. As a user, I should be able to view the system information (dashboards) on different screen sizes so that I can display the information in different contexts.

Cada uma dessas historias de usuário foi detalhada melhor na organização das tarefas, incluindo uma descrição mais completa, possíveis regras de negócio, quais requisitos, funcionais e não funcionais, ela se referencia, e também critérios de aceite. A organização das atividades está detalhada na seção 3.3.2.

3.3.2 Organização das tarefas

O método Kanban, originário do sistema Toyota de produção, tornou-se uma ferramenta popular e eficaz para gerenciamento e organização de fluxos de trabalho. A palavra "Kanban"é de origem japonesa e pode ser traduzida como "cartão visual"ou "sinalização". No contexto de gestão de projetos, Kanban refere-se a um sistema visual de gestão que destaca o fluxo de trabalho e as tarefas em diferentes estágios de processo. A essência do Kanban é visualizar todo o fluxo de trabalho, desde as tarefas que ainda não foram iniciadas até aquelas que foram concluídas. Esta visualização permite identificar gargalos e ineficiências, otimizando assim o processo. Para a organização das atividades desse projeto, o método Kanban foi adotado como uma estratégia para garantir uma progressão eficiente e sistemática do trabalho.

Para a implementação do método Kanban, foi escolhido o Notion como ferramenta. A escolha deste software deve-se à sua flexibilidade e capacidade de personalização, permitindo a criação de um board Kanban que se adapta especificamente às necessidades do projeto. Além disso, o Notion oferece uma interface intuitiva para a construção documentação, que está melhor detalhada na seção 3.3.3.

O board Kanban foi estruturado em cinco colunas, cada uma representando um estágio distinto no fluxo de trabalho:

- **Backlog:** Esta coluna contém todas as tarefas e atividades identificadas, mas que ainda não foram iniciadas. É um repositório de tudo o que precisa ser feito, mas que ainda não tem uma data definida para começar.
- **To Do:** As tarefas que estão nesta coluna estão prontas para serem iniciadas. Elas foram retiradas do Backlog, detalhadas e têm prioridade para serem iniciadas em breve.
- **Stopped:** Aqui estão as tarefas que foram iniciadas, mas por algum motivo tiveram que ser interrompidas, desde mudanças de prioridade, necessidade de alguma valição ou limitação técnica.
- **In Progress:** Esta coluna contém as tarefas que estão atualmente em execução. A movimentação para esta coluna indica que o trabalho está ativamente sendo feito na tarefa.
- **Done:** Assim que é finalizado o desenvolvimento de uma tarefa ela é dada como concluída, e portanto, movida para esta coluna. Representa o sucesso na finalização da atividade, e serve como um registro de todos os itens concluídos.

A estruturação destas colunas oferece uma visão clara do status de cada atividade e ajuda a identificar rapidamente onde estão os gargalos, facilitando a tomada de decisões e a priorização das tarefas.

A fim de definir melhor a implementação de cada história de usuário, cada card no board Kanban foi detalhado com uma descrição que inclui regras de negócio relevantes, referências a requisitos funcionais e não funcionais, critérios de aceitação e sub-tarefas. Antes que uma história de usuário possa ser movida para a coluna "In Progress", é essencial que esses campos sejam avaliados para assegurar um entendimento do escopo da tarefa. Os critérios de aceitação desempenham um papel importante na verificação de que uma história atende a todas as exigências estabelecidas antes de ser marcada como concluída.

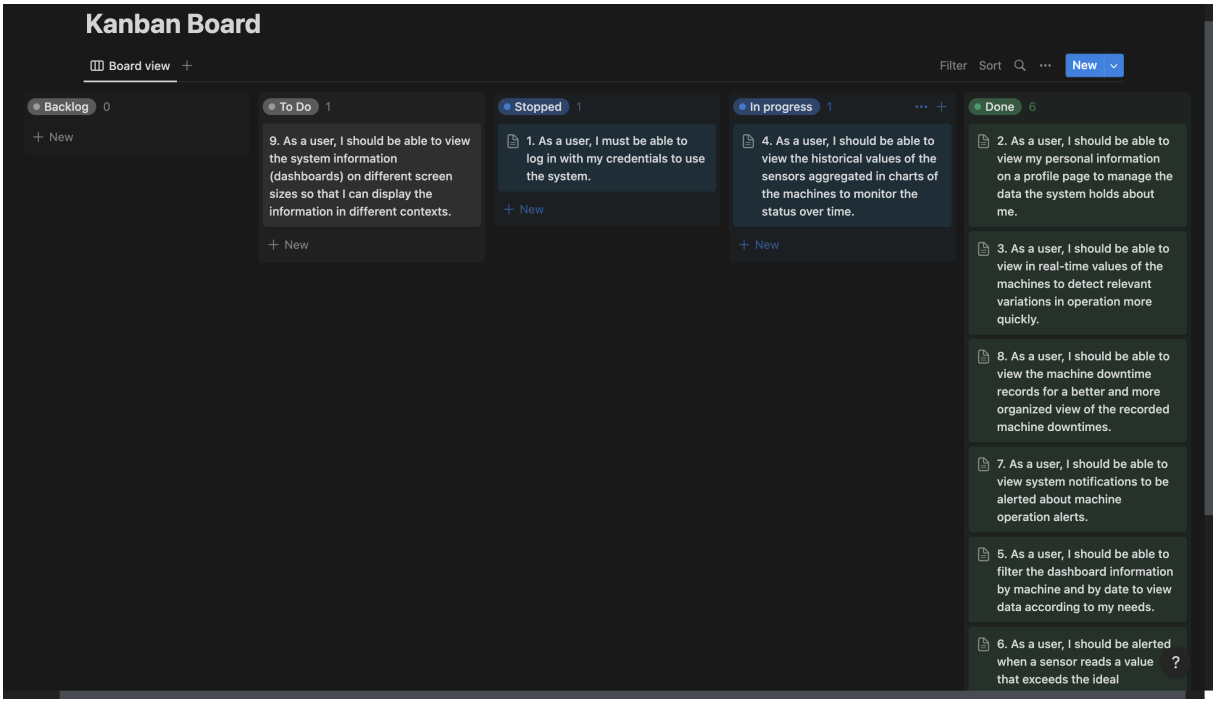

Figura 3.1: Board Kanban used to manager the tasks.

Para ilustrar este processo, é detalhado a história de usuário número 1.

1. As a user, I must be able to log in with my credentials to use the system.

- **Descrição:** Para garantir a segurança e a personalização da experiência do usuário, o sistema deve possuir uma funcionalidade de autenticação. O usuário deve inserir suas credenciais - geralmente um nome de usuário ou endereço de e-mail e uma senha - para acessar sua conta e as funcionalidades associadas a ela.

- **Regras de Negócio Relevantes:**

  – Os usuários não podem acessar o sistema sem autenticação.

  – As tentativas de entrar no sistema devem ficar armazenadas no banco de dados.

  – As senhas devem ser armazenadas de forma segura, utilizando técnicas como hashing.

- **Referências a Requisitos:**

  - **Funcionais:** FR1
  - **Não Funcionais:** NFR2, NFR3, NFR6, NFR9

- **Critérios de Aceitação:**
  - O sistema deve apresentar uma tela de login clara e intuitiva.
  - Após inserir as credenciais corretamente, o usuário deve ser redirecionado para a página inicial do sistema (/dashboard).
  - Se as credenciais estiverem incorretas, o usuário deve receber uma mensagem de erro clara.

- **Sub-tarefas:**
  - Desenhar a interface da tela de login.
  - Construção da interface de login no repositório.
  - Implementar a lógica de criação de3 usuário com hash de senha.
  - Implementar a lógica de autenticação no backend com JWT Token.
  - Testar a segurança e eficácia da funcionalidade de login.
  - Testar se a API está retornando os códigos HTTP corretamente.

3.3.3 Documentação do projeto

A documentação do projeto foi feita na ferramenta Notion. A escolha para este projeto foi fundamentada pois o Notion se destaca por sua flexibilidade, permitindo uma configuração adaptável dos texto para atender a vários tipos de necessidades. A interface intuitiva torna a inserção e atualização de informações um processo simples, que pode ser executado por ser uma plataforma acessível na internet por qualquer navegador.

Dessa forma, a documentação foi elaborada adotando uma abordagem estruturada para garantir que todas as características e funcionalidades fossem devidamente catalogadas. A base dessa documentação é uma tabela, onde cada entrada corresponde a uma

funcionalidade ou a uma específica parte do sistema, sendo denominada de acordo com o nome da característica em questão.

Cada item da tabela se expande em uma página independente, com detalhes descritivos. Esta organização permite o entendimento em cada aspecto do sistema, pois foi sendo construído à medida que o sistema era desenvolvido, refletindo assim as adições e mudanças mais recentes.

Para facilitar a navegação e busca por informações específicas, foi incorporado uma coluna de tags na tabela. Estas tags servem para categorizar as funcionalidades, e também um mecanismo eficiente de busca, permitindo que os usuários identifiquem rapidamente os aspectos relevantes do sistema.

Além disso, o design interno de cada página é fundamentado na estrutura do markdown, para assegurar que o conteúdo da documentação seja apresentado de forma clara, estruturada e esteticamente agradável, facilitando a compreensão e a absorção das informações por parte do leitor.

3.3.4 Configuração dos repositórios

A escolha da plataforma para gerenciamento dos repositórios do projeto foi o GitHub, uma das mais renomadas e amplamente adotadas plataformas de controle de versão baseada em Git disponíveis atualmente. A decisão de utilizar o GitHub foi pautada em alguns fatores. Primeiramente, a plataforma oferece uma interface intuitiva e um conjunto robusto de ferramentas que facilitam o acompanhamento do progresso do código, bem como a colaboração entre diferentes membros da equipe. Além disso, o GitHub é amplamente reconhecido por sua comunidade ativa, o que se traduz em uma vasta gama de recursos, tutoriais e suporte disponível, essencial para resolver possíveis dúvidas e desafios. Ademais, a integração com outras ferramentas e plataformas é facilmente realizável caso fosse necessário, permitindo um fluxo de trabalho contínuo e otimizado. Por fim, o compromisso do GitHub com a segurança, garantindo a integridade do código e dos dados do projeto, reforçou nossa decisão em adotá-lo como a solução de controle de versão para

o projeto desenvolvido.

Nesse contexto, os repositórios criados para esse projeto foram `backend`, que armazena o código referente a API do sistema, outro chamado `frontend`, que armazena o código do dashboard que é exibido na web, e outro chamado `iot_sensors_data_aggregation`, responsável por armazenar o código que realiza a agregação dos dados recebidos pelos sensores.

3.3.5 Reuniões periódicas

O desenvolvimento do projeto foi acompanhado por reuniões para garantir seu alinhamento com os objetivos do projeto. Reuniões semanais com o professor orientador foram estabelecidas, garantindo uma constante revisão, momento para tirar dúvidas, e realizar o detalhamento de atividades. Estes encontros proporcionaram um feedback contínuo, possibilitando a correção de trajetória e o foco no progresso desejado para o projeto. Paralelamente, reuniões mensais foram conduzidas com a empresa financiadora do projeto, para apresentar o que estava sendo construído, pegar feedbacks, orientações sobre funcionalidades desejadas, e também entender melhor como funciona a empresa e suas necessidades.

Esses encontros desempenharam um papel fundamental na integração entre a pesquisa acadêmica e as necessidades práticas da indústria, buscando que as soluções desenvolvidas se mantivessem pertinentes e aplicáveis ao contexto empresarial. Este sistema de supervisão foi crucial para manter o projeto em seu curso, balanceando as necessidades acadêmicas com a aplicabilidade industrial dentro do contexto da empresa.

3.4 Tecnologias

A seleção de tecnologias foi realizada para atender a requisitos específicos de escalabilidade e sustentabilidade a longo prazo. Dado que o sistema é primariamente voltado para armazenamento e gestão de dados, foi antecipada sua utilização como referência para futuros

projetos de características similares. Por isso, a ênfase recaiu sobre tecnologias modernas, amplamente reconhecidas e com robusto suporte na comunidade de desenvolvimento.

Nesse contexto, o MongoDB foi escolhido como nossa solução de banco de dados não relacional, devido à sua flexibilidade e performance. Python foi adotado como linguagem para o backend, devido à sua versatilidade e ampla biblioteca. O framework FastAPI, por sua vez, foi empregado para a elaboração da API, graças à sua eficiência e facilidade de integração. No âmbito do frontend, a linguagem JavaScript foi complementada pelo framework NextJs, reconhecido por sua otimização e recursos avançados. Para garantir uma integração fluida e modulada dos componentes do sistema, foi utilizado ao uso de containers Docker, enquanto o gerenciamento eficiente do servidor web foi assegurado pelo NGINX.

3.4.1 MongoDB

Ao selecionar uma plataforma de banco de dados, optou-se pelo MongoDB, um sistema de banco de dados não relacional projetado para adaptar-se com flexibilidade às mudanças no formato dos dados que são armazenados. O MongoDB proporciona facilidade na manipulação do data lake em variados contextos. Sua documentação meticulosamente estruturada, aliada à vasta gama de conteúdo disponível online, provou ser inestimável para a aquisição de conhecimento.

Distintivamente, o MongoDB apresenta vantagens como a capacidade de suportar consultas distribuídas e paralelas, otimizando o processamento de requisições intrincadas em cenários com densidade significativa de dados. Adicionalmente, sua compatibilidade com uma ampla variedade de ferramentas de análise de dados estabelece um precedente promissor para evoluções futuras do projeto.

3.4.2 Python

No desenvolvimento do backend, optou-se pela utilização da linguagem Python, amplamente reconhecida por sua versatilidade, legibilidade e adaptabilidade em diversos contextos de aplicação. Python, sendo uma das linguagens mais populares e amplamente aceitas no mundo acadêmico e industrial, apresenta uma vasta biblioteca padrão e suporte comunitário robusto. A rica gama de materiais educativos, que abrange desde tutoriais detalhados até extensos fóruns de discussão, foi essencial para o aprendizado.

A sintaxe intuitiva de Python favorece a rápida prototipagem e desenvolvimento, ao passo que a ampla gama de frameworks e bibliotecas disponíveis potencializa sua aplicação em diversos aspectos, desde análise de dados até desenvolvimento web. Essas características intrínsecas, somadas à flexibilidade e eficiência da linguagem, consolidam a decisão de adotar Python como a linguagem central para o backend neste projeto de mestrado.

3.4.3 FastAPI

Na etapa de implementação do backend, decidiu-se empregar a linguagem Python, aliada ao framework FastAPI. Esta escolha foi motivada, em grande parte, pela eficácia e elevado desempenho oferecido pelo FastAPI. Este framework se destaca por incorporar a biblioteca ASGI (Asynchronous Server Gateway Interface), uma interface que otimiza a gestão de solicitações, ao aproveitar ao máximo a execução assíncrona, garantindo respostas mais ágeis e precisas. Uma característica importante do FastAPI é sua documentação abrangente e bem elaborada, que ser como uma ferramenta fundamental ao processo de aprendizado e desenvolvimento.

3.4.4 NextJs

Para a arquitetura do frontend, optou-se pelo uso do Next.js, um framework que aprimora significativamente a interação com o React, tal como enfatizado na própria documentação oficial do React. O suporte comunitário é uma de suas principais características, sendo

amplamente complementado por uma oferta de materiais educativos disponíveis na internet - desde tutoriais a artigos de blog e vídeos instrucionais - os quais contribuíram de maneira essencial para o processo de aprendizado.

No escopo desse projeto, paralelamente ao Next.js, incorporou-se o TypeScript, que por sua natureza de tipagem estática, proporciona uma manutenção do código mais intuitiva, incrementando sua legibilidade, simplificando sua compreensão e gerenciamento.

A coerência entre Next.js e TypeScript estabelece um ambiente de desenvolvimento altamente eficaz. Enquanto o Next.js promove uma experiência de desenvolvimento mais fluída e de alto desempenho, o TypeScript fortalece a segurança e a produtividade, graças à sua tipagem rigorosa. Estes fatores justificam a escolha da combinação de Next.js e TypeScript para a concretização deste projeto.

3.4.5 Docker

Para a orquestração e gerenciamento do ambiente de desenvolvimento e produção, adotou-se o Docker como ferramenta de containerização. O Docker, amplamente reconhecido no universo de desenvolvimento de software, possibilita encapsular aplicações e suas dependências em containers, garantindo uniformidade, reprodutibilidade e isolamento entre os ambientes. Esta abordagem simplifica significativamente os processos de integração, teste e implantação, uma vez que os containers podem ser movidos de forma transparente entre diferentes ambientes e plataformas.

A ampla documentação disponível, juntamente com uma comunidade ativa, proporcionou um entendimento claro e facilitou a adoção desta tecnologia. Além disso, a flexibilidade e eficiência proporcionadas pelo Docker, ao minimizar os conflitos de dependências e garantir que a aplicação funcione consistentemente em diversos contextos, foram fatores decisivos para sua escolha neste projeto.

3.4.6 NGINX

Para a parte de gestão das solicitações web, adotou-se o NGINX como servidor web. O NGINX é reconhecido por sua alta performance, confiabilidade e flexibilidade, sendo uma escolha adequada em ambientes de produção que demandam baixa latência, tratamento eficiente de um grande número de conexões simultâneas, e com capacidade de servir conteúdo estático de maneira extremamente rápida. Essas características tornam-no particularmente adequado para sistemas que visam escalabilidade e robustez.

A vasta documentação e os extensos recursos da comunidade foram essenciais para aprofundar o entendimento e aplicar as melhores práticas no contexto do projeto. Ao se considerar a necessidade de uma entrega consistente e otimizada do conteúdo ao usuário final, bem como uma configuração segura e eficaz do proxy, o NGINX emergiu como a escolha preeminente para esta dissertação de mestrado.

Capítulo 4

Arquitetura do sistema

Neste capítulo, é abordado em detalhe a arquitetura e a estrutura adotada para a construção dos componentes do sistema. É importante compreender que o sistema foi concebido como um conjunto de módulos, com cada um desempenhando funções específicas, e quando operados em conjunto, esses módulos resultam na realização dos propósitos pensados para o sistema.

O sistema é estruturado fundamentalmente em camadas distintas, o backend, o frontend, e o banco de dados.

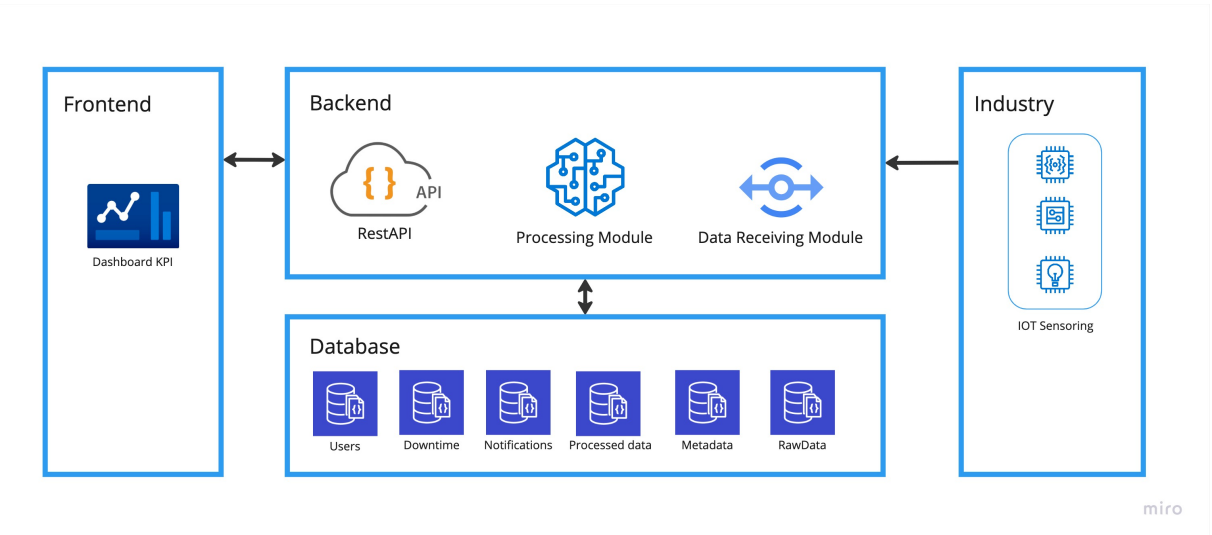

Figura 4.1: System architecture.

O backend funciona como o núcleo do sistema. Sua principal função é a de receber os dados, processá-los conforme as regras estabelecidas nos requisitos e histórias de usuários e, armazená-los de maneira segura no banco de dados. Além das funções de armazenamento e processamento, ao backend também é atribuída a responsabilidade de disponibilizar esses dados por meio de uma interface de programação de aplicações (API), que pode ser acessada utilizando métodos HTTP. Esta API age como um intermediário entre a lógica central do sistema e as interfaces com as quais o usuário final interage, o frontend.

Por outro lado, o frontend é caracterizado como a interface visual que o usuário final acessa. Serve como meio pelo qual os usuários interagem com o sistema, enviando e recebendo informações. Esta camada é projetada para acessar, recuperar e apresentar os dados processados e armazenados pelo backend de uma maneira intuitiva e amigável, utilizando princípios de design e data visualization.

Como pode ser visto na figura 4.1, na planta industrial, onde as máquinas com os sensores se encontram, os sensores enviam os dados para o sistema, que recebe eles por meio do modulo de recebimento de dados e os armazena no banco de dados. O modulo de processamento acessa os dados armazenados para realizar a agregação, e a API gerencia o acesso ao banco de dados, disponibilizando as informações para os usuários no frontend.

Nas seções seguintes, cada um desses componentes será explorado mais profundamente, passando por suas especificidades e interações.

4.1 Arquitetura backend

Nessa seção é abordado o funcionamento do backend. Ele está dividido em três partes, o module de recebimento dos dados dos sensores, o modulo de processamento onde é feita a agregação e analise estatística dos dados, e a API que gerencia o acesso as informações por meio de requisições HTTP.

Em relação a organização dos repositórios, a API e o modulo de recebimento de dados

ficam no mesmo repositório, facilitando a comunicação entre eles. Já o modulo de processamento está em um repositório a parte, sendo a sua única função sendo ler o banco de dados, processar os dados, e armazenar os resultados.

4.1.1 Modulo de Recebimento de dados

Para o modulo de recebimento de dados, inicialmente, destaca-se a classe `SensorConnection`, cuja principal função é gerenciar a conexão. Esta classe foi projetada para lidar com tarefas essenciais, como a inicialização e manutenção da conexão. Essa classe faz a transmissão dos dados recebidos à uma função designada `save_data_func`, assegurando que os dados sejam encaminhados para manipulação apropriada.

Na próxima parte da arquitetura, é utilizada a classe `IotSensorConnection`, que se origina da interface `IotSensorConnectionInterface`. Esta interface foi criada para garantir a adaptabilidade do sistema, facilitando a integração de diferentes tipos de recebimentos de dados, como por exemplo, uma classe destinada a gerar dados dos sensores em um ambiente de desenvolvimento, onde não há acesso ao sensor real. A classe `IotSensorConnection`, quando instanciada, é encarregada de estabelecer a conexão, e criar uma nova thread que opera como um ouvinte ativo, monitorando a chegada de novas informações. Ao perceber a recepção de novos dados, a classe direciona estas informações para uma terceira entidade, a qual detém a responsabilidade de aplicar as regras de negócio.

Esta terceira entidade é a classe `SensorsRepository`, que quando acionada com dados oriundos dos sensores, tem a responsabilidade de avaliar a informação com base nos parâmetros estabelecidos, decidindo se é imperativo acionar um alerta, e tornar os dados do sensor acessíveis via API, garantindo que esses dados estejam disponíveis para serem transmitidos em tempo real, via stream, para todos os usuários conectados. Além do mais, o dado é salvo no banco de dados, especificamente na coleção de dados brutos do data lake, `Raw Data`. Uma vez salvos no banco de dados, estes dados brutos estão disponíveis para serem processados pelo módulo de agregação.

A disponibilização dos dados pela classe `SensorsRepository` acontece por meio da instancia da classe `SensorValue`, que com o método `update_current_sensor_value` atualiza os dados em memoria que são acessados pelos usuários conectados.

O diagrama que mostra a organização dessas classes pode ser visto na figura 4.2. Este design assegura que os dados brutos dos sensores sejam efetivamente recebidos, avaliados e armazenados.

No capítulo 5, é aprofundado nos detalhes de implementação desse módulo.

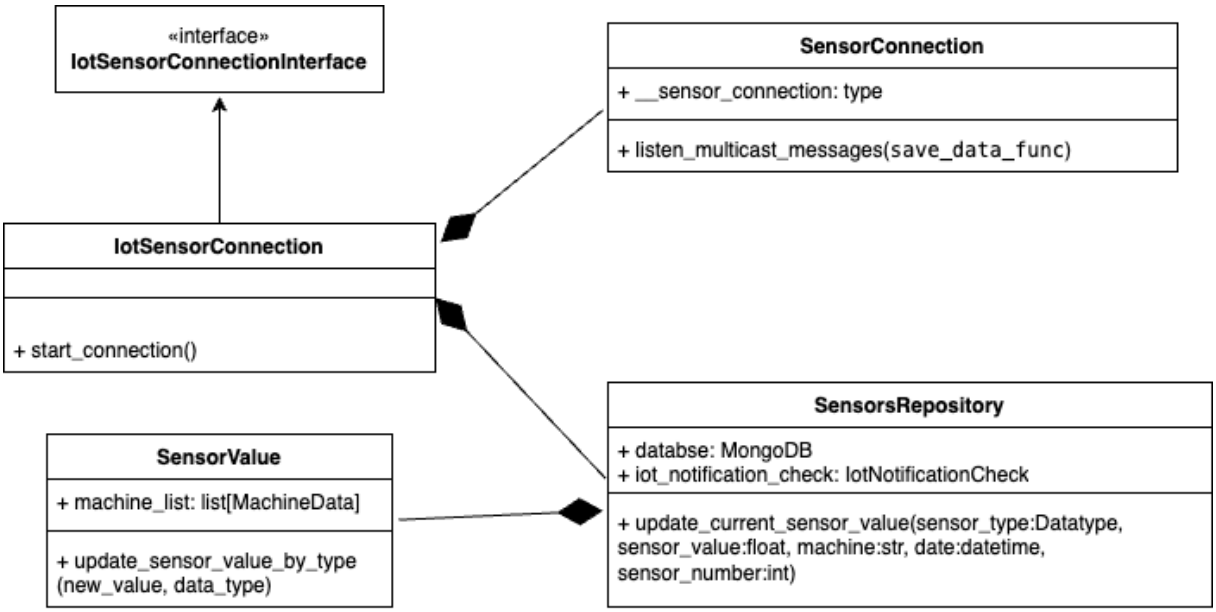

Figura 4.2: Module to receive sensor data.

4.1.2 Modulo de processamento

O módulo de processamento de dados foi desenvolvido para garantir que os dados brutos coletados sejam processados, fornecendo a análise estatística que é exibida para os usuários.

O código é executado quando é invocada uma função específica encarregada de realizar uma série de operações. Primeiramente, uma lista é constituída contendo as coleções do banco de dados responsáveis pelo armazenamento tanto dos dados brutos quanto dos

dados já processados. Simultaneamente, uma segunda lista é gerada, representando as máquinas que enviaram informações para o sistema.

Com essas listas em mãos, inicia-se um procedimento iterativo. Para cada máquina identificada, os dados disponíveis são lidos, submetidos a um processo de análise estatística, após o qual os resultados obtidos são registrados na coleção de dados processados. Essa análise estatística adota o método do Box Plot.

O Box Plot, também conhecido como diagrama de caixa, é uma ferramenta gráfica utilizada para representar a variação de dados observados de uma variável numérica por meio de quartis. Na figura 4.3, pode ser visto o retângulo formado pelo primeiro quartil (Q1), mediana e terceiro quartil (Q3), que fornecem uma noção sobre a centralidade e dispersão dos dados, enquanto as "antenas"estendem-se para mostrar a amplitude completa dos dados, ajudando assim na identificação de possíveis outliers.

Ao adotar o Box Plot, o sistema garante uma compreensão robusta da distribuição dos dados, identificando não apenas tendências centrais, mas também variações e potenciais anomalias.

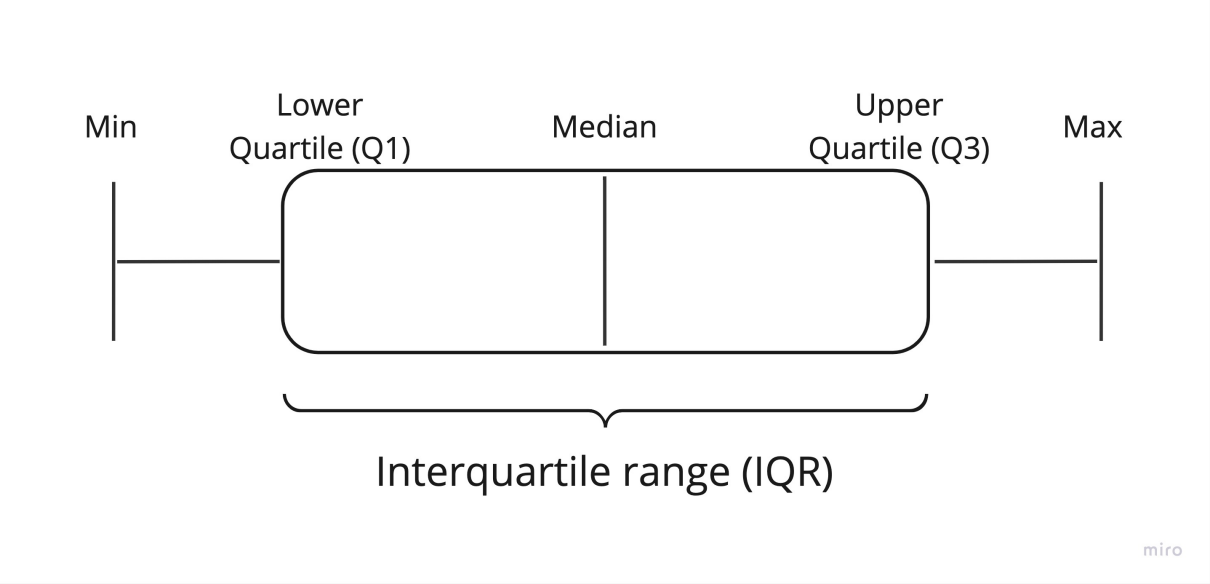

Figura 4.3: BoxPlot.

4.1.3 API

A API foi estruturada em pequenos sub-módulos, cada um focado em um contexto específico. Esta modularização assegura que cada parte da API tenha uma única responsabilidade. Em cada módulo, há uma segmentação composta por: a camada de *controller*, destinada a receber e gerenciar as requisições HTTP; a camada de serviço, que serve para processar a informação e aplicar as respectivas regras de negócio; e a camada de repositório, cujo papel é estabelecer uma ponte com o banco de dados, acessando e disponibilizando os dados necessários. A figura 4.4 representa a organização de pastas que foi utilizada para essa arquitetura.

Quando uma solicitação é enviada à API, a primeira interação acontece com a camada de *controller*. Uma vez recebida, essa requisição é direcionada à camada de serviço, onde as regras de negócio são aplicadas. A camada de serviço se comunica estreitamente com a camada de repositório, que detém a responsabilidade de acessar o banco de dados e trazer informações precisas, adequadas às demandas do módulo em questão.

Os módulos implementados na API com a lógica descrita são:

- **Downtime:** Responsável por gerenciar o acesso aos dados de paragem armazenados para teste no sistema.
- **IOT Sensors:** Responsável por gerenciar o acesso aos dados referentes aos sensores das maquinas na fabrica.
- **Notification:** Responsável por gerenciar o acesso as notificações geradas pelo sistema, e também as conexões web sockets para envio de notificações.
- **User:** Responsável por gerenciar o acesso aos dados dos usuários, assim como realizar as operações de login e logout.

Além dessas camadas modulares, existe uma área especifica na API para o armazenamento de códigos comuns a todos os módulos. Esta seção engloba diversas funções úteis, modelos de classes, valores constantes e configurações padrão. Tais elementos garantem uma maior coesão e reduzem a repetição de código, otimizando o desempenho

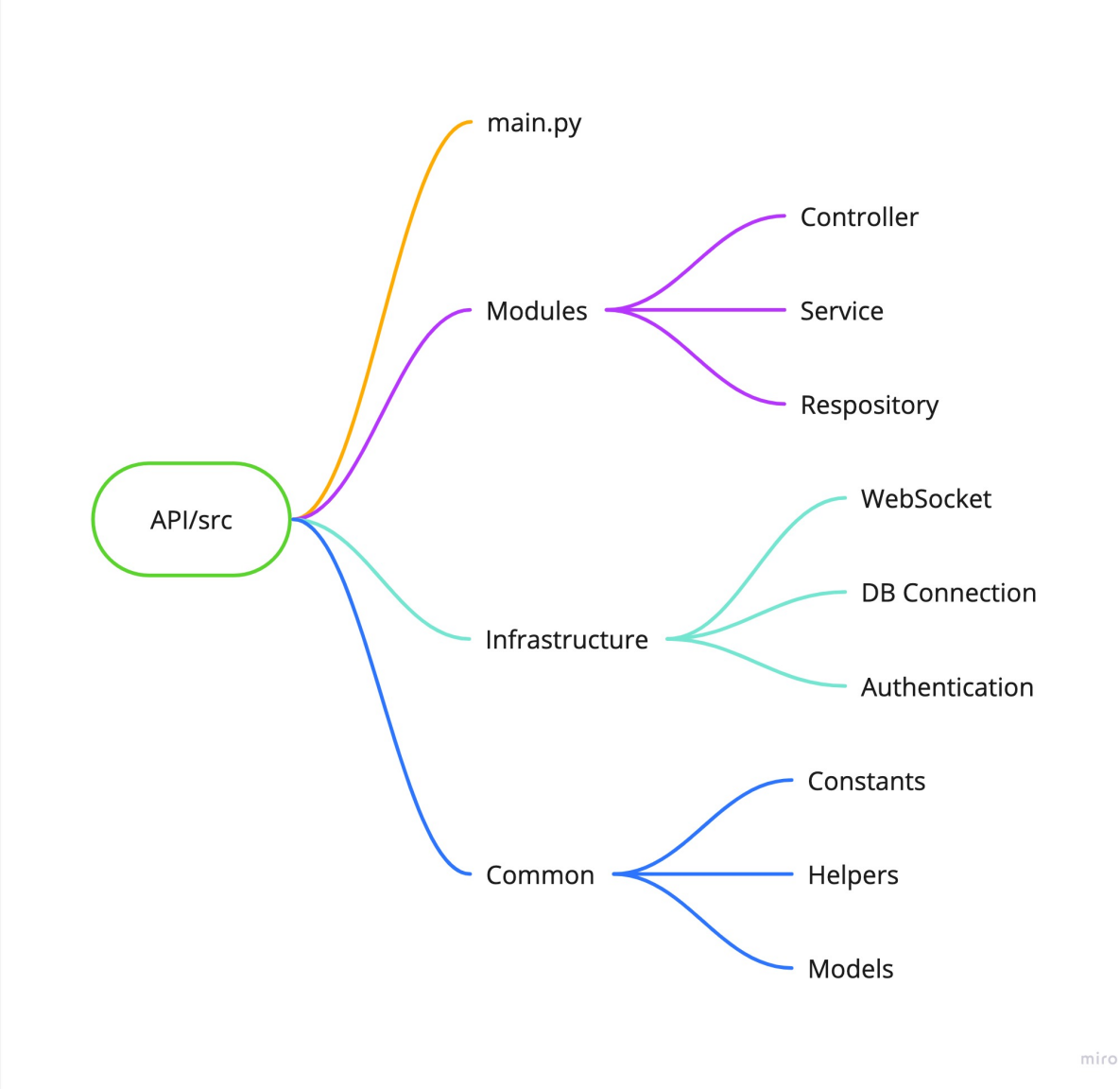

Figura 4.4: API Organization.

geral. Dentre as configurações padrão, merecem destaque o inicializador que estabelece o acesso ao banco de dados, middleware de autenticação, conexão web socket para envio de notificações, e o inicializador de novas *threads*. Este último é utilizada para operações assíncronas que são executadas em paralelo a operação da API, como aquelas executadas pelo módulo de recebimento de dados.

4.2 Arquitetura do frontend

Utilizando o *Next.js* como framework, o frontend segue uma estrutura básica já estabelecida pelo mesmo.

As rotas do sistema residem na pasta `pages`, alinhadas com as diretrizes do framework. Já os layouts que servem de base para cada página estão localizados na pasta `layouts`.

Os componentes *React* são a fundação de cada página e layout e estão organizados em uma camada específica, permitindo que sejam reutilizados em várias partes da aplicação.

Com a adoção do *Typescript*, modelos definem os tipos de estrutura de dados utilizados. Estes são mantidos na pasta `types`, estabelecendo contratos de formato de dados para o frontend. Isso minimiza erros e potencializa a eficiência no desenvolvimento.

A *Context API* do React é empregada para gerir dados nos componentes, permitindo o compartilhamento centralizado de informações, como pode ser visto na figura 4.5. Esta abordagem otimiza a maneira como os dados são acessados e distribuídos no sistema, otimizando a organização da arquitetura.

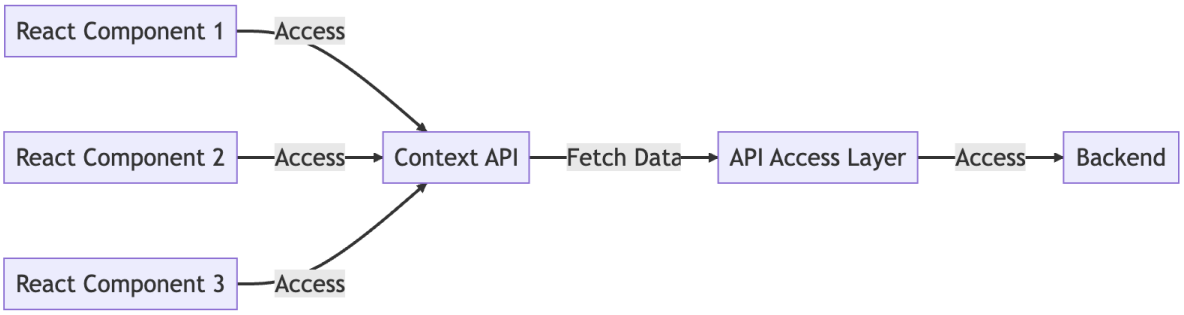

Figura 4.5: Frontend organization.

Existe uma camada específica para acesso externo, que administra a comunicação com a API e as conexões *WebSocket*. Esta é acessada apenas pelos contextos para atualização e recuperação de dados.

Por fim, há uma pasta dedicada para armazenar códigos recorrentes, contendo funções auxiliares, temas e *assets*, facilitando o desenvolvimento e manutenção ao proporcionar uma estrutura clara e coesa.

4.3 Containers

Containers são tecnologias que permitem isolar aplicações em ambientes específicos com todas as suas dependências, bibliotecas e configurações necessárias, sem a sobrecarga de máquinas virtuais completas. Isso garante que a aplicação funcione de maneira idêntica em diferentes ambientes, desde o desenvolvimento até a produção. Na figura 4.6 é possível visualizar como é o funcionamento dos containers dentro do sistema operacional do host.

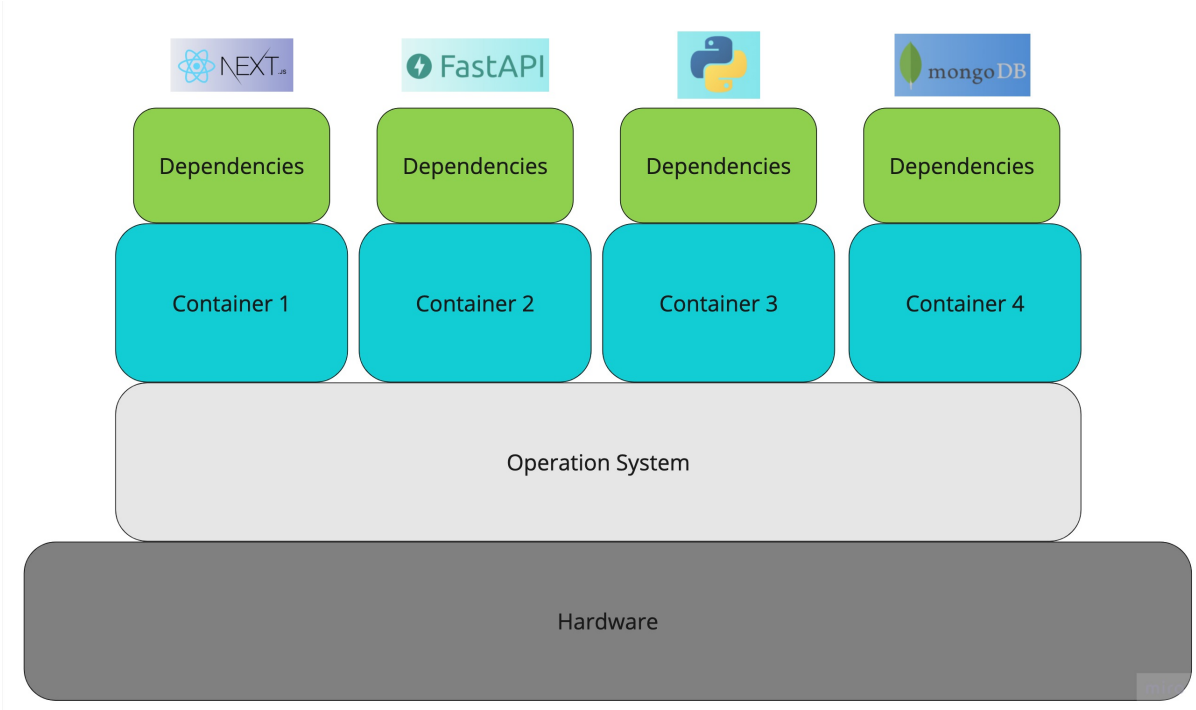

Figura 4.6: How container works.

Dentro do universo dos containers, o *Docker* foi a ferramenta selecionada para este projeto. Diversos fatores influenciaram essa decisão, incluindo uma documentação abrangente, uma comunidade ativa e a presença de uma ampla variedade de conteúdos disponíveis. Além disso, o Docker simplifica a definição, criação e execução de containers, tornando-se uma solução robusta para a implantação de aplicações.

O sistema adota diversos containers para organizar e gerenciar as várias partes da aplicação:

- **Frontend**: Um container dedicado ao frontend, construído com *NextJs*.
- **Backend**: Dividido em dois containers distintos, um abrange a API e o módulo de recebimento de dados, enquanto o segundo é voltado especificamente para o módulo de processamento.
- **Banco de Dados**: Um container para o banco de dados MongoDB, garantindo isolamento e eficiência na gestão dos dados.

A comunicação entre os containers é viabilizada através de uma rede *bridge* providenciada pelo Docker. Esta rede é uma interface de software criada no host que permite que containers comuniquem entre si e com o host, assegurando a conectividade necessária entre os diferentes módulos da aplicação. Com isso é adicionada uma camada a mais de segurança na aplicação, já que toda conexão externa deve ser feita por meio dessa rede. A conexão com rede externa é feita por meio de um web sever, explicado na seção 4.4.

Para garantir a persistência dos dados e evitar a perda de informações vitais, foi empregado o conceito de *volumes* do Docker na arquitetura do sistema. Volumes são espaços designados no sistema host que podem ser acessados e utilizados pelos containers. No contexto deste projeto, um volume foi especificamente configurado para o banco de dados MongoDB. Assim, mesmo que o container do banco de dados seja reiniciado ou removido, a base de dados se mantém intacta e disponível, devido à sua armazenagem no volume, que opera independentemente do ciclo de vida do container.

Com a necessidade de gerenciar múltiplos containers, configurações de rede e volumes de forma coesa e simplificada, foi adotado o *Docker Compose* na arquitetura do sistema. O

Docker Compose permite a definição e execução de aplicações multi-container usando um arquivo YAML. Esse arquivo contém todas as configurações necessárias para inicializar e interconectar os containers. Assim, ao invés de executar uma série de comandos para iniciar cada container individualmente, é possível, através do Docker Compose, iniciar todo o sistema com um único comando. Essa abordagem não apenas simplifica o processo de deploy e desenvolvimento, mas também garante que as configurações de rede e volume sejam consistentemente aplicadas em cada execução.

A utilização de containers no projeto trouxe vantagens. Primeiramente, garantiu a consistência entre os ambientes de desenvolvimento e produção. Adicionalmente, a modularização proporcionada pelos containers facilita a escalabilidade e manutenção do sistema, permitindo atualizações e alterações de forma ágil e segura a medida que o sistema for crescendo. Por último, a utilização de containers facilita a portabilidade do sistema, podendo ser executado em diversos tipos de servidores e sistemas, bastando ter a instalação do docker.

4.4 Web Server

Dentro da arquitetura proposta, com containers executando diferentes partes da aplicação, foi utilizado o *NGINX* para ser o intermediário no tráfego de requisições, assegurando a distribuição correta para cada container.

O método empregado para tal é o de proxy reverso. Em termos simples, o proxy reverso atua como uma interface entre o cliente e vários servidores, direcionando as solicitações dos clientes ao servidor adequado (no contexto desse projeto, os containers), e assim, otimizando o uso dos recursos e garantindo uma resposta mais rápida e eficiente.

No que se refere a requisições específicas, aquelas que envolvem retorno em formato de stream ou estabelecem uma conexão *WebSocket*, as configurações específicas foram feitas na configuração do *NGINX*, sendo essas detalhadas no capítulo 5, dedicado à implementação. Ao receber uma requisição, o servidor *NGINX* identifica, com base nela, qual container é o responsável pelo atendimento. Após essa identificação, são aplicadas as

configurações adequadas, e a requisição é direcionada ao container correspondente para obter a resposta. Esse workflow pode ser visto na figura 4.7.

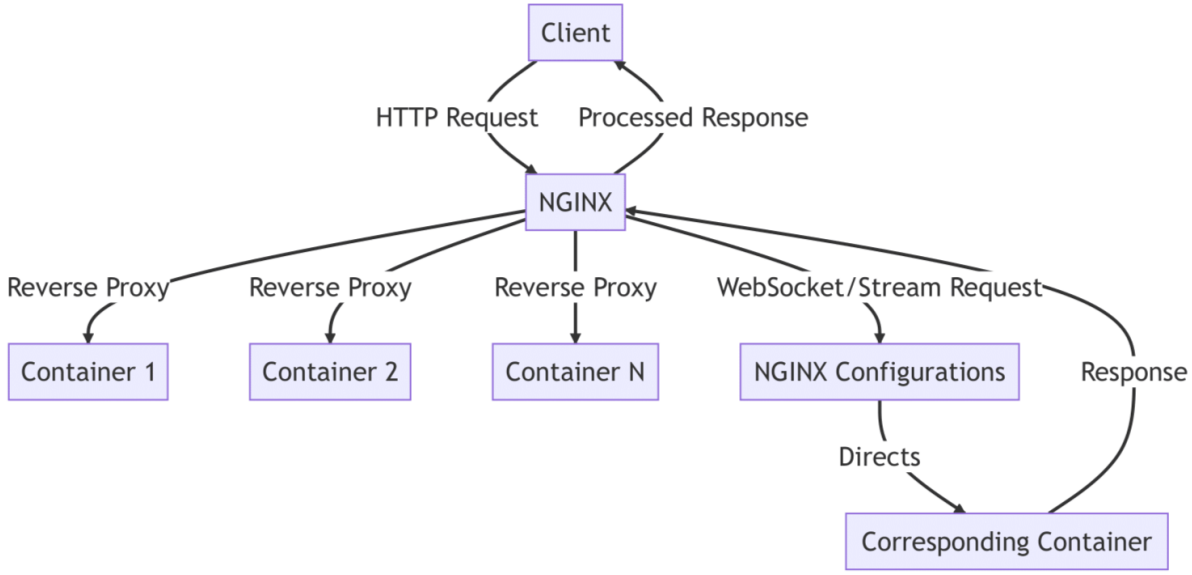

Figura 4.7: NGINX workflow.

A incorporação do *NGINX* trouxe alguns benefícios ao projeto. Um deles é a camada adicional de segurança: o *NGINX* limita o acesso direto aos containers, servindo como uma barreira contra tentativas de acesso não autorizado. Adicionalmente, com o NGINX, o processo de escalabilidade torna-se mais simples e eficiente, graças à capacidade inerente do servidor em atuar como um balanceador de carga. Este balanceador de carga distribui o tráfego de entrada entre vários servidores, assegurando que nenhum servidor fique sobrecarregado. Esta funcionalidade não só melhora a performance geral como também proporciona uma maior disponibilidade do sistema, já que, em caso de falha de um servidor, o tráfego pode ser direcionado a outro que esteja operacional.

Capítulo 5

Implementação

Após o capítulo anterior, onde a arquitetura do software foi detalhada, este capítulo é focado em explicar como tal arquitetura foi implementada. A distinção entre a concepção e a efetiva construção do sistema é crucial, pois enquanto o primeiro descreve a estrutura e a organização, este foca nas ações técnicas e metodologias adotadas para fazer essa estrutura funcionar.

Para uma análise mais estruturada e detalhada, este capítulo foi dividido em seções específicas para cada componente do sistema. São elas:

- **Implementação do banco de dados**: Esta seção abordará os detalhes técnicos do design do banco de dados, esquemas adotados e como as informações são armazenadas e recuperadas.
- **Implementação da API**: Aqui, a estrutura da API será discutida, incluindo os endpoints fornecidos, a lógica por trás de cada um e como eles interagem com o resto do sistema.
- **Implementação do módulo de recebimento de dados**: Esta seção detalhará como os dados são recebidos, validados e processados antes de serem armazenados e processados.
- **Implementação do módulo de processamento de dados**: Será abordado o

tratamento e transformação dos dados, garantindo que as informações sejam interpretadas e utilizadas corretamente pelo sistema.

- **Implementação do frontend**: Por fim, a interface com o usuário será discutida, explicando como os dados são estruturados apresentados e apresentados em tela.

5.1 Implementação do banco de dados

5.1.1 Organização do banco de dados

Dentro da implementação do sistema, o MongoDB foi usado para armazenar todas as informações do sistema. Este banco de dados, orientado a documentos, permitiu uma organização flexível dos dados, facilitando o armazenamento de diferentes dados que podem ser recebidos pelo modulo de recebimento de dados, e facilitando a criação de camadas de processamento. A estruturação dos bancos de dados e suas respectivas coleções foi pensada para facilitar tanto a inserção quanto a consulta de informações.

Em relação à organização dos dados, os seguintes bancos de dados foram criados:

- **Users**: Armazena informações referentes aos usuários. Possui coleções que registram tentativas de login, detalhes pessoais dos usuários e tokens associados a eles.
- **Notification**: Destinado às notificações do sistema. Atualmente, este banco contém apenas notificações associadas aos alertas das máquinas, gerados pelos dados recebidos dos sensores junto com os parâmetros armazenados.
- **Downtime**: Armazena duas coleções, uma com os dados lidos das planilhas de parada das máquinas, e outro com esses dados tratados. Esse banco de dados com essas coleções são apenas para simular como ficaria os dados de parada das maquinas, caso eles fosses inseridos no sistema.
- **Raw Data**: Este banco é dedicado ao armazenamento de dados brutos oriundos de diferentes sensores. Cada tipo de sensor, como os sensores de pressão, tem sua própria coleção, garantindo uma categorização intuitiva dos dados.

- **Processed Data**: Como o próprio nome sugere, armazena dados que já passaram por uma etapa de processamento. Assim, dados interpretados de diferentes sensores são separados em coleções específicas, como os de pressão em uma e os de voltagem em outra.

- **Metadados**: Dedicado à armazenagem de metadados do sistema. Até o momento, a única coleção presente é a "AlertParameter", que reúne parâmetros utilizados para gerar alertas associados a cada sensor.

Com esta estruturação, busca-se não apenas organizar de forma lógica os dados, mas também otimizar operações de consulta e garantir uma expansão simplificada à medida que novas necessidades de armazenamento emergem no sistema.

5.1.2 Acesso ao banco de dados

No processo de implementação do sistema, para estabelecer uma conexão eficiente com o banco de dados foi utilizado a biblioteca "motor"foi adotada como mecanismo.

No centro da estratégia de conexão está uma classe base, denominada `BaseDB`. Esta classe tem a responsabilidade não apenas de estabelecer a conexão com o MongoDB, mas também de definir uma série de operações básicas para a manipulação dos dados armazenados. A estrutura dessa classe é apresentada a seguir:

```
import motor
class BaseDB:
    def __init__(self):
        self.client = motor.motor_tornado.MotorClient(url, port)
```

Algumas das operações fundamentais implementadas por `BaseDB` incluem:

- `insert_one (database, collection, data)`: Insere um documento na coleção especificada.

- `insert_many (database, collection, data)`: Insere vários documentos na coleção especificada.

- `read_data_with_pagination (database, collection, query, page_number, limit, sort_descending_field, projection)`: Recupera dados com paginação, permitindo uma leitura mais organizada.

- `read_data_with_limit (database, collection, query, limit)`: Lê dados com um limite predefinido de documentos retornados.

- `read_data (database, collection, query)`: Realiza uma leitura simples de dados baseada em uma query.

- `get_distinct_property (database, collection, property)`: Obtém propriedades distintas de uma coleção, verificando todos os documentos presentes.

- `list_collections_by_db (database)`: Lista todas as coleções presentes em um banco de dados específico.

- `add_item_into_lists_by_filter (database, collection, filter, list_properties, new_data)`: Adiciona um item em listas específicas baseado em um filtro.

- `update_item (database, collection, data, filter)`: Atualiza um documento específico.

- `update_many_items (database, collection, data, filter)`: Atualiza vários documentos que atendam a um filtro.

- `count_documents (database, collection, query)`: Conta o número de documentos em uma coleção que atendem a uma consulta.

- `get_data_between_dates (database, collection, query)`: Recupera dados entre duas datas específicas.

Com a base de acesso estabelecida, outras classes foram desenvolvidas, herdados de `BaseDB`, para atender contextos específicos do sistema. Essas classes, seguindo o padrão singleton, garantem que apenas uma instância da conexão seja criada para um contexto específico, otimizando a gestão dos recursos. Um exemplo é a classe `MongoDBIOT` destinada ao módulo de recebimento de dados:

```
class MongoDBIOT(BaseDB, metaclass=Singleton):
    def __init__(self):
        super().__init__()
```

Classes semelhantes, seguindo o mesmo formato, foram criadas para outros contextos, como o acesso ao banco de dados pela API, garantindo uma estrutura organizada e eficiente de conexão e manipulação dos dados.

5.2 Implementação do modulo de recebimento de dados

No processo de implementação do sistema, uma das etapas essenciais foi o desenvolvimento de um módulo destinado ao recebimento de dados provenientes dos sensores IoT. Este recebimento é realizado por meio de uma conexão multicast, uma abordagem eficiente para lidar com a transmissão de mensagens a vários destinatários simultaneamente.

5.2.1 Conexão e recebimento dos dados

No centro deste módulo encontra-se a classe `SensorConnection`. Esta classe tem como principal responsabilidade criar um socket, manter-se conectada para receber mensagens e processá-las. A estrutura e o funcionamento desta classe são detalhados a seguir.

A classe `SensorConnection` é iniciada com a criação de um socket IPv4 e UDP:

```
class SensorConnection:
    def __init__(self):
```

```
self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Para garantir que o sistema esteja constantemente ouvindo mensagens multicast dos sensores, a função `listen_multicast_messages` foi definida. Esta função invoca a criação da conexão e inicia o processo de leitura de mensagens, gerenciando ainda possíveis desconexões e reestabelecendo a ligação quando necessário:

```
async def listen_multicast_messages(self, save_data_func):
    self.__create_connection()
    while True:
        await self.__start_read_messages(save_data_func)
        self.sock.close()
        time.sleep(1)
        self.__reconnect()
```

A função `__create_connection` tem um papel fundamental na classe `SensorConnection`, sendo responsável por estabelecer e configurar a conexão inicial com o grupo multicast.

Inicialmente, o socket é configurado para permitir várias conexões em um único endereço. A opção `SO_REUSEADDR` é definida com o valor 1, permitindo que mais de um socket se ligue a um mesmo endereço, o que é especialmente útil em contextos de conexões multicast:

```
self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

Após isso, o socket é vinculado a um endereço e porta multicast específicos. É importante ressaltar que o primeiro argumento na definição do endereço do servidor é deixado vazio. Esta abordagem garante que o sistema esteja conectando-se com todas as interfaces de rede disponíveis, proporcionando uma ampla cobertura de conexão:

```
server_address = ('', SENSOR_MULTICAST_PORT)
self.sock.bind(server_address)
```

Por fim, para se juntar efetivamente ao grupo multicast, algumas etapas são realizadas. O endereço IP multicast é primeiramente convertido para o formato binário. Em seguida, este endereço e o endereço local (representado por `socket.INADDR_ANY`) são empacotados em uma estrutura de dados. Esta estrutura é usado para especificar ao socket que ele deve se juntar a um grupo multicast. E então, a opção `IP_ADD_MEMBERSHIP` é definida e a estrutura previamente criada é passada como argumento, concluindo a conexão com o grupo multicast:

```
multicast_group = SENSOR_MULTICAST
group = socket.inet_aton(multicast_group)
mreq = struct.pack('4sL', group, socket.INADDR_ANY)
self.sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
```

Essas operações garantem que o socket esteja configurado e conectado ao grupo multicast, pronto para receber mensagens de múltiplas fontes simultaneamente.

Após as configurações realizads, as mensagens são continuamente lidas e processadas pela função `__start_read_messages`. Durante este processo, cada mensagem é processada, e se estiver no formato correto, é passada para uma função que irá salvar e disponibilizar para API enviar por streaming para os usuários conectados.

```
async def __start_read_messages(self, save_data_func):
while True:
    try:
        data, address = self.sock.recvfrom(1024)
        result = self.__parse_multicast_message(data)
        if not type(result) == str:
            await save_data_func(result)
    except Exception as e:
        print(f"Error: {e}")
        break
```

Em situações em que a conexão com os sensores é interrompida, o método `__reconnect` é chamado para tentar estabelecer novamente a conexão, criando uma nova instância do socket e chamando novamente a função `__create_connection`:

```
def __reconnect(self):
    try:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.__create_connection()
    except Exception as e:
        print(f"Error to reconnect: {e}")
```

Para interpretar e extrair informações da mensagem recebida do multicast, é crucial decodificar adequadamente a mensagem de acordo com o protocolo definido anteriormente. A implementação dessa decodificação é feita pelo método `__parse_multicast_message`. A função auxiliar `__parse_bytes` é utilizada para essa tarefa, dada uma sequência de bytes, a função interpreta os bytes utilizando a ordem big-endian (onde os bytes mais significativos vêm primeiro):

```
def __parse_bytes(self, bytes):
    data = int.from_bytes(bytes, byteorder='big')
    high_data = (data >> 8) & 0xFF
    low_data = data & 0xFF


    return (high_data,low_data)
```

Aqui, `data` contém o valor inteiro dos bytes fornecidos. O byte de ordem superior (High) é extraído deslocando o valor 8 bits para a direita e aplicando uma operação "E"(&), e o byte de ordem inferior (Low) é simplesmente obtido aplicando a operação "E"com `0xFF`.

Com a capacidade de interpretar os bytes, a função principal `__parse_multicast_message` pode começar a decodificação:

- Primeiro, ela extrai o tipo de máquina e o número da máquina dos dois primeiros bytes da mensagem.
- O terceiro byte da mensagem é então interpretado como o tipo da mensagem. Se o tipo da mensagem for 2, a função retornará diretamente uma solicitação para publicar.
- Os bytes 4 e 5 são interpretados como o ID do sensor, que contém a quantidade física sendo medida e o número do sensor.
- Os bytes 6 e 7 são usados para extrair o tipo de dados e seu significado.
- Finalmente, os bytes 8 e 9 são usados para determinar o comprimento dos dados que seguem.

A informação extraída é então organizada em um dicionário para representação clara e fácil acesso aos componentes individualmente:

```
message_dict = {
    'Machine': {
        ...
    },
    'Type': ...,
    'Sensor': {
        ...
    },
    'MeaningOfData': {
        ...
    }
}
```

Esta estrutura permite uma representação clara e modular da mensagem decodificada, tornando fácil a integração e utilização em outras partes do sistema.

5.2.2 Interpretação dos dados

- explicar a criação da classe - Salvamento no banco - Disponibilizar para streaming para o front

5.3 Implementação do módulo de processamento de dados

- Leitura dos dados de acordo com o processamento realizado anteriormente - Uso da biblioteca pandas - Funcionamento do BoxPlot para realizar analise estatística

5.4 Implementação da API

- Uvicorn usado pelo FastAPI e sua forma asyncrona - Biblioteca Motor usada para acesso ao mongoDB - Biblioteca Pydantic para criação dos modelos e tipos - Web socket para envio de notificações - Biblioteca Jose para autenticação - Comunicação entre as camadas com a classe Result - Contratos de interfaces - Tratamento de erros com classes personalizadas

5.5 Implementação do frontend

- Criação de paginas componentes de layouts - Recharts - Material UI - Days JS - Criação da camada de dados com o Context API - Acesso externo a API - Axios e fetch

5.6 Adaptando a implementação para outros contextos

Discussão sobre a reutização do sistema para outros contextos.... - como fazer - alterações necessarias

Capítulo 6

Características do Sistema do ponto de vista funcional

Adicionar capturas de tela e diagramas para exemplificar o uso

6.0.1 Monitoramento em tempo real

6.0.2 Sistema de alertas e nnotificações

e como isso ajuda a empresa

6.0.3 Dados historicos

6.0.4 Interface do usuário

- Exibição no computador - Exibição na televisão

Capítulo 7

Resultados e Avaliação

This chapter presents and describes the tests that were developed to check if the project fulfills the objectives and solves the problem described in Analysis/Methodology.

To better understand, the results of each test should be preceded by a description of the test and the expected results.

The work results are commented, including:

- What can be learned from the results?
- What could be done differently?
- What was beyond initial objectives?
- What are the objectives there were not met and why?

Apresentação dos resultados obtidos - Discutir as principais realizações do projeto, incluindo a capacidade de monitorar em tempo real as máquinas da planta e a análise histórica do funcionamento das máquinas.

- Sistema que ajuda no monitoramento das maquinas da planta - Exibir para os funcionários como as maquinas estão funcionando - Gerar insights a partir dos dados gerados

Capítulo 8

Conclusão e Trabalhos Futuros

The conclusions should synthesize and provide a single view to the work developed. It can be done a brief reference to similar work of others and to the knowledge that emerged from it, as well as future work suggestions. The consistency of the document implies that the conclusions should be coherent with the main ideas in the introduction.

8.0.1 Resumo

8.0.2 Limitações do sistema

(Adicionar novas camadas de dados e paginas no dashboard pode ser demorado)?

8.0.3 Sugestões para trabalhos futuros

- melhorias de código - Monitoramento do funcionamento do sistema e logs - Log para recebimento dos dados e quando para de receber - Log para a analise estatistica - Log para erros no recebimento dos dados - Generalização para outros contextos - destacando a tendência crescente de coleta e análise de dados na indústria - Melhoria da performance para grande numero de maquinas e graficos exibidos na tela ao mesmo tempo. - Upgrade para next 13 e 14. Pois com isso podemos ter o beneficio dos server componentes, o que melhoraia a performace da aplicação - Os graficos podem ser carregados como server

componentes com cache equivalente ao intervalo que o modulo de processamento roda, otimizando assim o frontend da aplicação

Apêndice A

Proposta Original do Projeto

Curso de Licenciatura em Engenharia Informática

Projeto 3º Ano - Ano letivo de 2016/2017

<Título do projeto>

Orientador: <Nome do orientador>

Coorientador: <Nome do coorientador>

1 Objetivo

<Objetivo do projeto>

2 Detalhes

<Detalhes que julguem ser necessários>

3 Metodologia de trabalho

<Eventual metodologia de trabalho>

Dimensão da equipa:

Recursos necessários:

Apêndice B

Outro(s) Apêndice(s)

Listagens de código fonte, texto/imagens produzidos por testes complementares, etc.