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

Aplicação para cálculo de indicadores de performance de malhas de controle de processos industriais

Felipe T. C. Ribeiro

Orientador: Prof. Frederico Gualberto Ferreira Coelho.
Supervisor: Eng. Juliana Vieira Martins

Belo Horizonte, Março de 2021

Monografia

Aplicação para cálculo de indicadores de performance de malhas de controle de processos industriais

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

Belo Horizonte, Março de 2021

Resumo

O presente trabalho apresenta o processo de desenvolvimento de uma aplicação para cálculo de índices de performance de malhas de controle, implantada em conjunto a um sistema CPM, sigla do inglês para *Controller Performance Monitoring*, que recebe o nome de LOOP, desenvolvido pela empresa *IHM Stefanini*. O sistema em questão é utilizado por um conjunto de diferentes clientes da indústria nacional como ferramenta para acompanhamento e metrificação dos estados de suas malhas de controle. Para tais avaliações é comum a utilização de índices de performance, que traduzem a números, o desempenho de um sistema de controle. Nesse ponto entra a aplicação alvo desse projeto, que recebe o nome de Kpi-Executor. O Kpi-Executor foi construído para a realização dos cálculos dos índices de performance, do inglês *Key Performance Indicators*, abreviado por KPIs, que são parte essencial do sistema LOOP. É desejado que esta aplicação se integre sem problemas ao sistema existente, em substituição a uma anterior com função semelhante, tendo sua execução realizada a cada hora, gerando assim os valores desses índices utilizados. Nessa monografia serão detalhados o processo de desenvolvimento desse software, além da concepção teórica dos índices de performance os utilizados pelo sistema: Tempo em modo normal, Tempo sem saturação, Erro médio aceitável e Nota global. Como resultados são apresentados documentação de regras de negócio gerada, a estrutura de logs para melhoria na etapa de manutenção e evolução do software, além da estabilização e repetibilidade obtidas pelo Kpi-Executor.

Agradecimentos

"Pra que amanhã não seja só um ontem com um novo nome", Leandro

A Deus pela coragem e força concedida. À minha mãe, Sônia, pelo exemplo de força, resiliência e por todo apoio e ensinamento sobre jornadas. Ao meu pai por sua importância no meu amadurecimento. Às minhas amigas e colegas de curso Lud, Manu e Pena que sabem exatamente o porque de estarem aqui. À expansão da minha família nas pessoas de Gutenberg, Lidiane, Letícia e avós. A todos os amigos de caminhada. Por fim, não menos importante, minha esposa, Larissa.

Mãe! A favela venceu!

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação e Justificativa

É incontestável a incessante busca do setor industrial pelo aumento do desempenho de suas unidades operacionais. Para que as empresas envolvidas nesse ramo possam se manter de forma significativa no mercado, é necessária a constante reavaliação de seus processos, bem como de seus custos com energia, perdas e reprocessamento de material. Independentemente do setor de atuação da indústria a ser avaliada, tais fatores possuem extrema importância, devendo ser monitoradas, controladas e projetadas.

Para elucidar, tomemos como exemplo um processo de beneficiamento de minério de ferro. Pode-se dividi-lo em: Britagem, peneiramento, moagem, classificação, concentração, espessamento e filtragem. Cada uma dessas etapas é composta por variáveis a serem controladas de forma individual, como níveis de tanque, temperatura e densidade de misturas, velocidades de motores, posições de válvulas, etc. Para estabelecer o comportamento desejado para tais processos, se faz necessária a aplicação de estratégias de controle regulatório, avançado ou de outras naturezas. O conjunto composto por variável a ser controlada e estratégia de controle utilizada é chamado de *malha de controle*.

Implementado um sistema de controle, obtém-se então um processo minimamente adequado aos parâmetros configurados para o contexto operacional escolhido. Entretanto, não menos importante que essa adequação técnica individual estabelecida por esse sistema, são os resultados e efeitos colaterais gerados naquela linha do processo, bem como a avaliação do conjunto em situações não previstas nas estratégias de controle. Dessa forma, o acompanhamento e metrificação do comportamento de sistemas de controle industriais ganha notabilidade dentro do setor industrial.

Para este monitoramento são utilizados softwares baseados em CPM (*Controller Performance Monitoring*). Eles possuem ferramental para análise temporal das malhas envolvidas, avaliação de parâmetros de controladores, modelagem de processos e medição de KPIs (*Key Performance Indicators*). Buscando este espaço de mercado, a empresa IHM Stefanini desenvolveu o software LOOP, que se enquadra no tipo citado.

Nos últimos meses o time de desenvolvimento da empresa iniciou, juntamente ao time de analistas de controle que utilizam o sistema, um processo de identificação de pontos de melhoria do software LOOP, envolvendo quesitos de interface, regras de negócio, distribuição de responsabilidades entre os módulos do sistema, arquitetura, integração de código e documentação. Dentre os pontos não satisfatórios listados pela equipe, o módulo responsá-

vel pelo cálculo de KPIs foi apontado como crítico, sendo o primeiro a passar por mudanças significativa. O projeto descrito por esta monografia aborda o processo de reconstrução desse módulo.

1.2 Objetivos do Projeto

O objetivo do projeto é a construção de um módulo para cálculo de índices de performance, utilizando-se de conhecimentos de Engenharia de software e desenvolvimento de sistemas. Como requisitos para aceitação, o módulo deve possuir as seguintes características:

- **Execução cíclica:** O cálculo dos índices de performance é baseado em janelas de dados de uma hora. Dessa forma, ele deve possuir execução cíclica com um intervalos semelhantes;
- **Adequação à estrutura do sistema:** O módulo será utilizado para compor um sistema da empresa parceira, Ihm Stefanini, que já possui outros módulos existentes. Assim é necessário que sejam respeitadas limitações e conceitos do sistema como um todo;
- **Registro de falhas:** É necessário que o módulo apresente registros de suas falhas de cada uma das execuções, não somente listando, mas também identificando quais tipos;

1.3 Local de Realização

O projeto de fim de curso foi desenvolvido em parceria com a empresa IHM Stefanini, no setor de Discovery da mesma, responsável pelo desenvolvimento, evolução e manutenção da arquitetura e código fonte dos sistemas ali desenvolvidos. O posto de trabalho variou entre as dependências da empresa, e regime de *home office*, conforme necessário.

A empresa realiza projetos e desenvolvimentos de Automação, Elétrica, Montagem, bem como serviços de otimização de malhas, projetos de controle avançado e gestão. Atua em diversas áreas da indústria, como Mineração, Siderurgia & Metais, Papel & Celulose, e Óleo & Gás.

A IHM Stefanini foi criada em 1994 inicialmente com o nome de IHM como uma integradora de sistemas, instrumentação, elétrica, e TI Industrial. Com sua expansão, passa a fazer parte do grupo Stefanini em 2015. A partir daí funda o departamento de inovação e começa a atuar na busca por tecnologias disruptivas e elaboração de produtos digitais.

Atualmente a IHM Stefanini é dividida nos seguintes setores técnicos:

- Departamento de TA;
- Departamento de TI Industrial;
- Departamento Inteligência Industrial;
- Departamento de Discovery;
- Departamento de Elétrica;

Este projeto foi desenvolvido no departamento de Discovery que tem como propósito a descoberta sistemática e estruturação de ofertas de alto valor agregado para a IHM Stefanini. Subdividido em:

- *Produtos*: Formação de produtos que resolvam problemas reais de clientes finais;
- *Coder*: Time responsável pelos *softwares* do setor. Estrutura, desenvolve e mantém códigos-fonte e infraestruturas das demandas do setor.

1.4 Estrutura da Monografia

O trabalho está estruturado em cinco capítulos, sendo o Capítulo 1 o conjunto contextualização, relevância e principais objetivos do projeto como um todo. O Capítulo 2 contempla apresentação e revisão de conceitos básicos importantes para melhor entendimento do projeto. Já o Capítulo 3 traz informações sobre os recursos necessários, metodologia de desenvolvimento e implementação do projeto. No Capítulo 4, a apresentação e discussão de dados, bem como sugestões e dificuldades a serem encontradas no projeto. Por fim, no Capítulo 5, são apresentadas as conclusões obtidas no após a elaboração do projeto, e considerações sobre possíveis continuidades.

Capítulo 2

Revisão Literária

Neste capítulo são apresentados os conceitos utilizados neste trabalho baseados na literatura. Na primeira seção são abordadas definições processuais de desenvolvimento de um software. Na segunda são revisados termos de direcionamento técnico para este processo. Na terceira, conceitos de controle e automação importantes, enquanto, na quarta, trabalhos relevantes que abordam assuntos comuns ao projeto aqui relatado.

2.1 Processo de Software

Um processo de software é composto por um conjunto de atividades relacionadas que levam à produção de um produto de software [30]. A utilização deste conjunto de atividades auxilia no aumento da qualidade do produto desenvolvido, bem como sua manutenção. De forma geral, um processo de software é composto de quatro grandes etapas:

- **Especificação de Software:** Devem ser definidas as funcionalidades e limitações de funcionamento do software a ser desenvolvido;
- **Projeto e Implementação do Software:** O software deve ser produzido buscando o atendimento das especificações;
- **Validação de software:** Deve ser validado para garantir o atendimento às demandas solicitadas;
- **Evolução do Software:** Deve ser evoluído e alterado conforme novas necessidades

2.1.1 Modelo em Cascata

Existem diversos modelos de processo de software [30] , dentre eles o modelo cascata, também conhecido como ciclo de vida de software. Este modelo consiste no encadeamento de suas etapas, tendo o início de uma associado necessariamente ao final de outra. São elas:

- **Definição de requisitos:** São definidas restrições e metas do sistema por meio de entrevista com o usuário. Posteriormente são detalhadas a fim de se tornar especificação do sistema;

- **Projeto de Sistema e Software:** Etapa de definição de arquitetura geral do sistema. Envolve identificação e descrição das abstrações, módulos ou componentes, bem como o relacionamento entre eles;
- **Implementação:** O projeto de software é desenvolvido com seus conjuntos ou unidades de programas;
- **Integração e Testes de sistema:** Os módulos individuais do sistema elaborado são integrados e testados em conjunto, buscando assegurar o funcionamento conforme especificado;
- **Operação e manutenção:** O sistema é colocado em uso. São efetuadas possíveis correções de erros não descobertos em etapas anteriores. Expansões e melhorias de implementação das unidades também podem ser realizadas nesta etapa.

Este modelo dá grande importância à documentação gerada ao final de cada uma das etapas, geralmente amarrando o início da etapa seguinte à aprovação destes documentos. Ele também é consistente com outros modelos de processos de engenharia, tornando o processo mais visível para o nível gerencial.

2.1.2 Desenvolvimento Ágil

O desenvolvimento ágil se baseia na entrega de pequenos incrementos do produto de software disponibilizando versões para o cliente, a cada período de tempo especificado (em geral, duas ou três semanas) [30]. Esse envolvimento do cliente durante o processo de construção possibilita o acompanhamento da evolução dos requisitos do sistema. Tem como outra forte característica a utilização de conversas informais, minimização da documentação, gerando mais responsividade do processo a possíveis mudanças nos requisitos do produto de software.

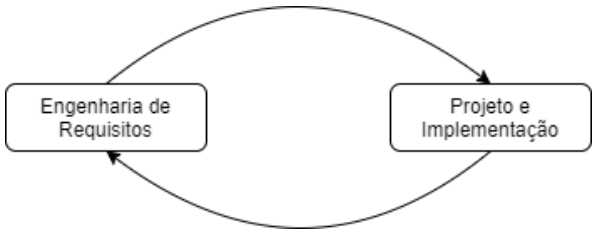

Figura 2.1: Métodos Ágeis (Adaptado de [30])

2.2 Arquitetura de Software

Arquitetura de Software é um conjunto de decisões significativas sobre a organização de um sistema de software, a seleção dos elementos estruturais e suas interfaces, a composição destes em subsistemas progressivamente maiores, bem como o estilo de arquitetura que orienta a organização e interação dos módulos envolvidos. A definição da arquitetura de um software passa por etapas e conhecimentos prévios necessários para uma estruturação correta do mesmo [22].

2.2.1 Modelagem orientada a Dados

Modelos construídos sob o paradigma de orientação a dados trazem a sequência de ações envolvidas no processamento daqueles dados, desde sua entrada, até sua saída. Pode ser realizada através de um modelo de sequências[22] destacando a movimentação dos dados daquele módulo ali representado. A figura 2.2 traz um exemplo desse tipo de modelo.

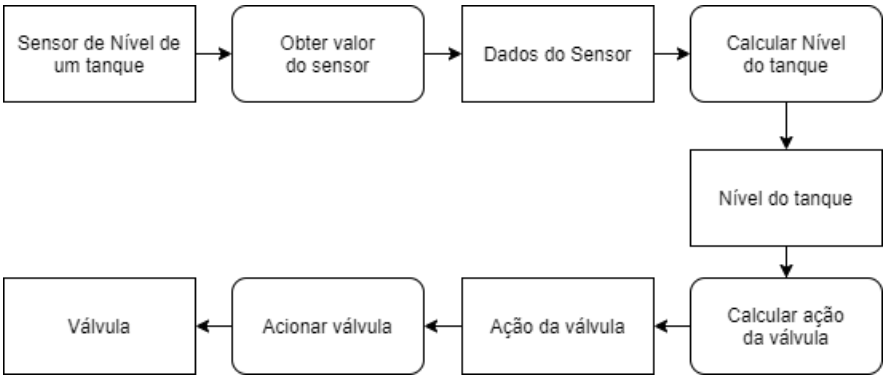

Figura 2.2: Modelo de sequência (Fonte: Autor)

2.2.2 Arquitetura em Camadas

Arquitetura que organiza o sistema em um grupo de camadas, onde cada uma delas oferece um conjunto de serviços. Essa noção de separação e independência é fundamental para que haja a possibilidade de alterações localizadas, facilitando as etapas de manutenção e sustentação do produto de software. A figura 2.3 traz uma representação genérica deste tipo de arquitetura.

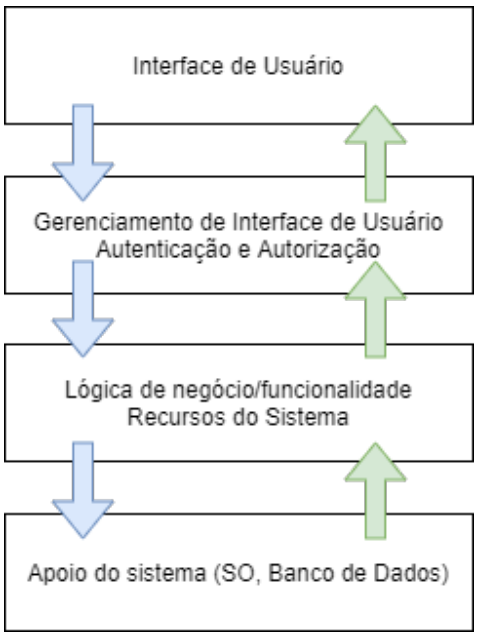

Figura 2.3: Arquitetura genérica em camadas (Fonte: Autor)

O relacionamento entre as camadas se dá de forma bem definida. Possui protocolos que indicam como a comunicação entre elas se dá. Uma camada só solicita serviços à inferior, e somente fornece à superior. Esse tipo de divisão facilita e colabora para o processo de desenvolvimento incremental. Uma arquitetura deste tipo muito utilizada e conhecida é a *MVC*, sigla do inglês para *Model-View-Controller*.

2.2.3 Orientação a Objetos

O conceito de OO (Orientação a objetos) consiste na estruturação de um código ou programa utilizando abstrações para conectar o espaço do problema ao da solução. Os vários objetos construídos nessas abstrações se relacionam através de comandos. Cada um desses objetos possui uma *classificação* que o determina, chamada de *classe*.

Para ficar claro como os objetos se relacionam, os conceitos que seguem são importantes:

- **Encapsulamento:** Estratégia utilizada para garantir que detalhes internos do funcionamento de uma classe. Dessa forma, o conhecimento sobre a implementação interna de uma classe é desnecessária do ponto de vista da instância daquela classe;
- **Herança:** Quando uma classe pode ser um *tipo* de uma outra classe, o conceito de herança pode ser aplicado. Se existe uma classe *veículo* e se deseja construir uma representação *motocicleta*, a segunda (classe filha) pode herdar características da primeira (classe pai), visto que *motocicleta* é um tipo de *veículo*;
- **Polimorfismo:** É o princípio pelo qual duas ou mais classes derivadas de uma mesma classe pai podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse.

2.2.4 Modelo Relacional de Banco de Dados

Banco de dados é um mecanismo de armazenamento que permite persistência de dados de uma aplicação, programa ou produto de software. O modelo relacional busca o aumento da independência de dados nos sistemas. O Modelo relacional revelou-se ser o mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação da base de dados. A estrutura fundamental do modelo relacional é a relação. Ela é constituída por um ou mais atributos que traduzem o tipo de dados a armazenar [31].

2.2.5 Containers

Container é uma unidade de software para desenvolvimento e implantação que empacota código e suas dependências para a execução de um aplicativo de forma rápida, escalável e confiável, independentemente do ambiente de computação que o hospede. No projeto descrito por essa monografia foram utilizados *containers docker* [18].

2.2.5.1 Containter e Máquinas Virtuais

Containers e máquinas virtuais possuem benefícios de isolamento e alocação de recursos, porém contemplam um funcionamento distinto. O primeiro utiliza-se da virtualização do sistema operacional, enquanto o segundo virtualiza o hardware.

Os containers são uma abstração na camada do aplicativo que busca o agrupamento de código e suas dependências. Podem haver múltiplos containers em execução , compartilhando *kernel* [19] do sistema operacional com outros containers, todos isolados em processos diferentes.

Em contrapartida, as máquinas virtuais abstraem o hardware, transformando um servidor em vários servidores. Cada máquina virtual inclui uma cópia completa do sistema operacional, do aplicativo e suas dependências.

2.2.5.2 Orquestração de containers - Kubernetes

A orquestração consiste na manipulação de containers buscando sua organização, escalabilidade e robustez. Ela permite a automatização de gerenciamento, rede, escala e implantação de softwares construídos sob este paradigma. Sua utilização também permite suporte a estruturas de *DevOps* [8] que integram fluxos de CI/CD (*Continuous Integration/Continuous Delivery*) [7].

2.3 Key Performance Indicators - KPIs

KPIs, sigla do inglês para *Key Performance Indicators*, são indicadores projetados para medir objetivos críticos de um sistema, indo de uma simples malha de controle regulatório, a uma inteira linha de montagem automotiva. O monitoramento desse tipo de indicadores promove o aumento no entendimento do sistema a ser analisado, auxiliando em possíveis tomadas de decisões estratégicas, e aumento de produtividade [29]. O controle desses índices busca geralmente melhorar qualidade e produtividade, alcançando pontos ótimos de operação, bem como aumento da lucratividade da cadeia produtiva em questão. Um exemplo de aplicação de controle de KPIs pode ser visto em [28].

2.3.1 Controller Performance Monitoring - CPM

CPM, sigla do inglês para *Controller Performance Monitoring* é uma técnica das mais usadas para a medição de *KPIs* direcionados a sistemas de automação, tendo T. J Harris [26, 27] como um dos pioneiros no assunto. Sua utilização, nos últimos anos, tem se tornado mais comum como ferramenta para programas de melhoria de operações promovidas por indústrias e universidades. *CPM* tem como principal objetivo o aumento da confiabilidade de uma planta industrial, sendo um auxiliar para o descobrimento de modos de falhas ocultos, que normalmente não são ligados à sintonia de controle, e sim a outras causas, como defeitos em atuadores (bombas, válvulas) ou formas de operação [33].

2.3.2 KPIs Universais para Plantas Industriais

Como dito, os *KPIs* visam, ao final de seu uso, melhorias nos resultados da empresa que os utilizar, podendo estar diretamente (de Negócio) ou indiretamente (de Controle e Processo) ligados aos objetivos da empresa [33]. Os *KPIs* de negócio costumam fornecer medidas valiosas quanto às evoluções ligadas à empresa, tendo caráter de longo prazo. Entretanto, nem sempre estão ligados a ações diretas em uma planta. Lucros, Qualidade e Custos totais estão nessa classificação.

Para que seja possível alcançar sucesso nos *KPIs* de Negócio (longo prazo), é necessária anteriormente a análise de *KPIs* com caráter de curto prazo. Exemplos deste tipo de *KPIs* são: Eficiência, confiabilidade, tempo de acomodação e tempo saída saturada. Esses, por sua vez, podem mais facilmente gerar ações diretas na planta [33].

Desses *KPIs* citados no último parágrafo (e outros presentes em [33]), alguns possuem característica universal, sendo entendidos pelos níveis de negócio e técnicos. Dentre eles, alguns apresentados em [33] se destacam para este tipo de análise:

- **Tempo em modo anormal:** Medição de tempo em que o sistema de controle não está funcionando;
- **Tempo em saturação:** Medição do tempo em que um atuador fica em seu máximo de atuação. Geralmente associado a mal-dimensionamento do atuador;

2.4 Trabalhos Relacionados

Foram encontrados diferentes trabalhos envolvendo *KPIs*: modelagem e controle desses indicadores [32], seu relacionamento com *PSS*, sigla do inglês para *Product-service system* [29], bem como seu uso direcionado a performance de sistemas de controle [33]. Dentre eles, destacou-se o último, devido a relação direta com os índices de performance utilizados para a elaboração do módulo descrito por essa monografia. Os índices citados na seção 2.3.2 são utilizados como base para a construção dos *KPIs* deste trabalho. Na seção 3.2.1.3.4 serão descritas suas escolhas e elaborações.

Quanto às referências relacionadas aos conceitos de software utilizados para a construção deste trabalho, Lázaro Marques e Dilson Júnior aplicaram o uso de *ORM* [3] para abstração de um banco de dados relacional no trabalho na UNA-SUS/UFMA (Universidade Aberta do SUS/Universidade Federal do Maranhão) [24]. Além disso, José Rocha aponta o uso de arquitetura e desenvolvimento de software orientado a objetos como fator importante para a garantia de qualidade e escalabilidade do um software.

Capítulo 3

Materiais e Métodos

Neste capítulo são descritos os recursos de hardware e software necessários para o desenvolvimento deste projeto. São listados IDE utilizada para desenvolvimento, arquitetura existente a consumir dados da aplicação e escolha de algoritmos de cálculo de KPIs. Também são apresentados os processos de desenvolvimento de software utilizados e funcionamento da equipe dentro da qual foi o projeto foi construído.

3.1 Descrição de Componentes

Esta seção apresenta a arquitetura existente que receberá a aplicação desenvolvida, o Kpi-Executor, bem como os recursos de hardware e software utilizados no seu desenvolvimento.

3.1.1 Software

3.1.1.1 IDE, Linguagem e Banco de Dados

Para o desenvolvimento da aplicação foi escolhida a linguagem *Python* [15], em sua versão 3.8, devido a seu caráter de alto nível e sua grande gama de bibliotecas disponíveis. Para a codificação foi escolhida a IDE *Visual Studio Code* [2], devido à sua multiplicidade de extensões e também por já ser amplamente utilizada pela comunidade de desenvolvedores de software.

Ambas, linguagem e IDE, tem característica *opensource*, sendo ativamente utilizadas e exploradas dia a dia por sua ampla comunidade, além de terem seus módulos e extensões gratuitos.

Quanto ao armazenamento dos dados, foi utilizada uma estrutura relacional de banco de dados, considerando sua praticidade, bem como a estrutura existente descrita nas próximas seções. Foi utilizada uma instância de um banco de dados *PostgresSQL* [14] , uma ferramenta gratuita para a persistência dos dados gerados pela aplicação.

3.1.1.2 Orquestradores e Containers

Como a aplicação desenvolvida foi acoplada a um sistema existente, foi necessário que ela respeitasse o conjunto anteriormente elaborado, bem como se adequasse à arquitetura em nuvem [25] já em funcionamento. Para tal, essa aplicação deveria possuir certo encapsulamento.

Uma forma amplamente utilizada para o encapsulamento e execução de aplicações de grande porte é o formato de *containers*. É uma abordagem de desenvolvimento de software onde um serviço ou aplicativo é empacotado com suas dependências e configurações [1]. Dessa forma, são facilitados os processos de Testes de unidade e integração [30], bem como etapas de evolução [30] e versionamento daquele conjunto. Para o processo de construção de *container* do módulo *Kpi-Executor* foi utilizada a plataforma *Docker* [9], gerando uma imagem de um SO (Sistema Operacional) Linux contendo as dependências e instalações necessárias para a execução do *Kpi-Executor*.

Uma vez empacotada a aplicação, é necessário utilizar uma ferramenta para operar esse pacote, o executando sempre que necessário. Esta utilidade é chamada de *orquestrador*. A ferramenta deste tipo utilizada foi o *Kubernetes* [11]. Este tipo de ferramenta possibilita a automação de operações *containers*, como implantações ou atualizações das aplicações, o gerenciamento de serviços de forma declarativa garantindo a repetibilidade do sistema, a separação de *containers* em diferentes *hosts*, bem como a verificação de integridade e autorrecuperação das aplicações [20].

3.1.2 Estrutura Existente

A aplicação construída funcionará como um módulo de um sistema de CPM existente chamado LOOP. Ele é composto por um conjunto de módulos, cada um com sua responsabilidade bem definida. A figura 3.1 traz o esquema arquitetural do sistema, bem como o relacionamento entre os módulos.

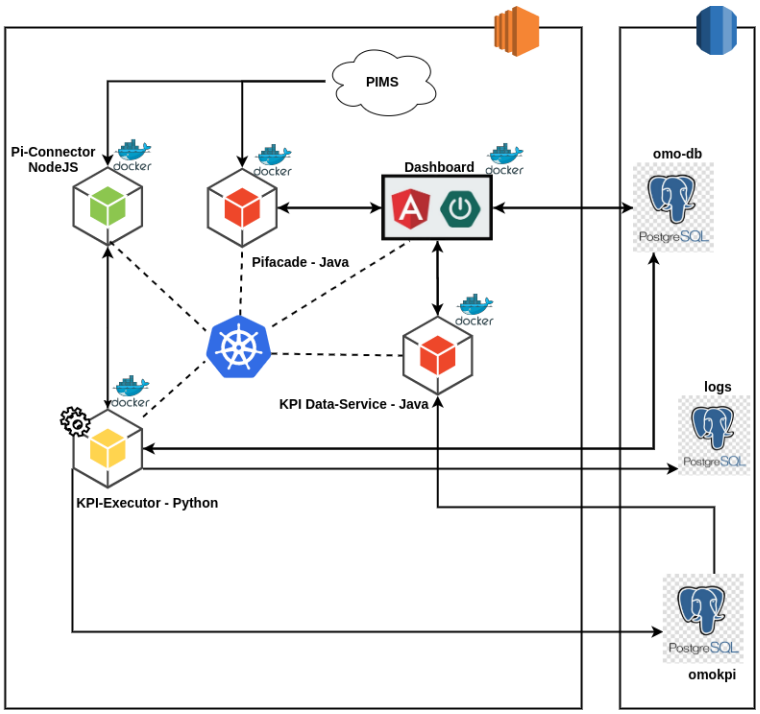

Figura 3.1: Arquitetura Loop

Neste esquema são identificadas as linguagens e tecnologias de cada módulo. São eles:

- **Dashboard:** Aplicação web acessível ao usuário do sistema LOOP composta por um *backend* em *Spring Boot* [16] e *frontend* em *AngularJS* [5];
- **Bancos de Dados:** São três instâncias de banco de dados do sistema. A nomeada como *omo-db* carrega as informações de cadastro e configuração das entidades do LOOP. A nomeada *omokpi* armazena os dados calculados pelo *Kpi-Executor*, enquanto a *logs* registra os logs de funcionamento da aplicação de cálculos. Todas as instâncias são do tipo PostgreSQL [14];
- **Data-Service:** Módulo do tipo API construído em *Spring Boot* [16] responsável por prover dados registrados no banco *omokpi* à aplicação web *Dashboard*. Formata pesquisa e retorna dados sob demanda;
- **PIMS:** Um sistema PIMS da *OsiSoft* chamado *PI System* [13]. Nele são armazenados os dados de processo dos clientes que utilizam o sistema LOOP. Estes dados são utilizados na realização dos cálculos (Kpi-Executor);
- **Pi-Facade:** Módulo do tipo API construído em *Spring Boot* [16] responsável por prover dados registrados e cadastrar novas informações no PIMS sob demanda da aplicação web *Dashboard*;
- **Pi-Connector:** Módulo do tipo API construído em *NodeJS* [12] responsável pelo fornecimento de dados registrados no PIMS à aplicação de cálculos (*Kpi-Executor*);
- **Kpi-Executor:** Módulo desenvolvido descrito por esta monografia. Construído em *Python 3.8* [15] e responsável por calcular os índices de performance (*KPIs*) do sistema LOOP. Opera de forma cíclica, realizando seus cálculos a cada hora.

3.1.3 Hardware

Para realização do desenvolvimento do Kpi-Executor, foi necessário um notebook com 16Gb de memória e processador intel i7. Também foi necessário um ambiente de Testes e validação semelhante ao de produção. São servidores hospedadas na *Amazon Web Services* [6] estruturadas e acompanhadas pelo time de *DevOps*[8] de Discovery.

3.2 Metodologia

Para a construção dessa aplicação foi utilizado o um processo de desenvolvimento de software composto por características de um cascata e um incremental [30]. Como o projeto foi entendido como uma demanda associada ao fluxo de trabalho da equipe de Coder, descrita na seção 1.3, ele estava submetido a um fluxo de *Kanban*, o que aponta para o quesito de entrega incremental quando associado ao sistema LOOP como um todo. Entretanto, como se tratava de uma tarefa grande e complexa, foram necessárias etapas que coincidem com o processo cascata. Neste capítulo serão listadas as etapas de projeto, desenvolvimento e testes do Kpi-Executor, bem como as escolhas relacionadas aos cálculos por ele realizados.

3.2.1 Processo de Desenvolvimento

O processo de desenvolvimento da aplicação iniciou-se com um escopo pouco descritivo. Ela deveria realizar de forma cíclica um conjunto de cálculos de índices de performance que são utilizados pelo sistema LOOP, registrá-los em um banco de dados existente, com estrutura também definida, e possibilitar um rastreio de falhas no fluxo de cálculos, identificando o qual tipo de erro, e em que momento este acontecera. Tendo essas informações, ainda eram necessárias mais definições de casos específicos não mapeados pela descrição inicial.

3.2.1.1 Levantamento do Requisitos

3.2.1.1.1 Variáveis do Sistema

Conceitos importantes para a descrição e detalhamento do módulo construído são os seguintes:

- ***Setpoint* (SP):** Valor-alvo de um sistema de controle regulatório;
- ***Process Value* (PV):** Variável de processo a ser controlada por uma malha de controle. O objetivo de uma malha é fazer com que a *PV* atinja o valor escolhido como *Setpoint*;
- ***Manipulated Value* (MV):** Em contextos de processo, MV é a variável que uma malha de controle utiliza para atuar no valor final da PV. Exemplos são aberturas de válvulas e velocidades de bomba;
- **Modo:** Malhas de controle costumam possuir modos de operação que habilitam ou não um tipo de controle específico. Exemplos de modo são automático, manual e cascata.

3.2.1.1.2 Fluxo desejado

Considerando a necessidade de um mapeamento detalhado dos casos de cálculo, foi realizado um processo de entrevista com o PO (*Product Owner*) do sistema LOOP, a fim de elucidar o fluxo de cálculo em seu estado ideal, possíveis tratativas em casos não usuais e demais requisitos funcionais e não-funcionais. Deste levantamento, foi elaborado o diagrama de sequência (seção 2.2.1) tratando o fluxo a ser executado pelo *Kpi-Executor*, exibido na figura (3.2).

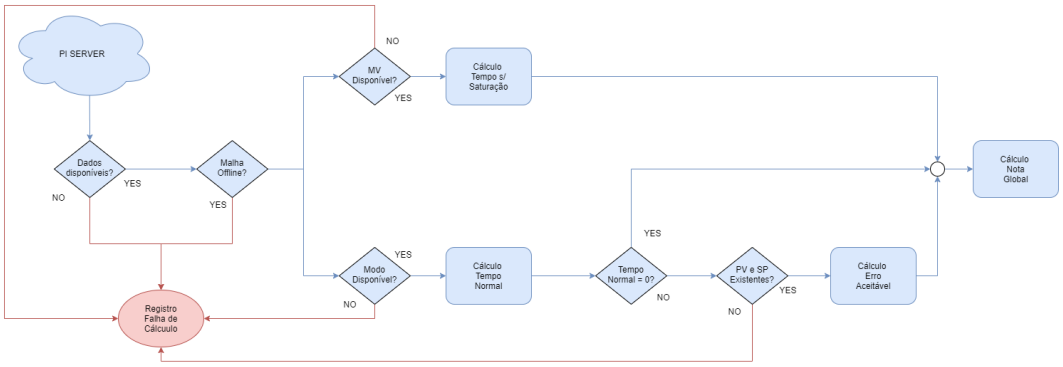

Figura 3.2: Diagrama de Sequência - Kpi-Executor

Após construção deste diagrama de alto nível, foi realizada o nova reunião o validando como fluxo correto de cálculos, incluindo as tratativas de casos específicos. Assim pôde-se caminhar para a próxima etapa do processo.

3.2.1.2 Projeto da Aplicação

Tendo requisitos funcionais e não-funcionais especificados, foi possível então o início do projeto da aplicação. Inicia-se pela definição arquitetural do módulo Kpi-Executor. Foi escolhida uma arquitetura em camadas (seção 2.2.2) devido à familiaridade da equipe com o tipo de construção e a seu caráter de atuação sob demanda e persistência em banco de dados.

Para persistência dos dados em banco, foi escolhida a tecnologia SQLAlchemy [4], amplamente utilizada para acesso via Python a bancos de dados relacionais. Ela possibilita uma abordagem de orientação a objetos (seção 2.2.3) em sua utilização.

3.2.1.3 Desenvolvimento da Aplicação

3.2.1.3.1 Classes auxiliares

Considerando a estrutura do sistema da empresa parceira (Loop) no momento do desenvolvimento do módulo tratado por essa monografia, era tido como de suma importância a possibilidade de utilização de diferentes bancos de dados relacionais, como Microsoft SQL Server [17] e PostgresSQL [14]. Assim, foi necessária a utilização de uma *ORM* (Object Relacional Mapper) [3] *SQLAlchemy* [4], possibilitando que o módulo em questão apresentasse características agnósticas a banco de dados. Além da facilitação no processo de substituição de bancos de dados, devido à objetificação das *queries*, adiciona segurança ao conjunto do sistema, impedindo tentativas de invasão do tipo *SQL Injection* [21].

Uma vez levantados os pontos citados, foram utilizados os conceitos de orientação a objetos para o desenvolvimento (2.2.3) para a elaboração de duas classes que juntas tornariam o processo de construção das consultas e inserções a banco de dados mais práticas e robustas:

- **Database:** Classe que contempla métodos para conexão e desconexão com o banco de dados, bem como as informações de conexão, como *host*, nome do banco, usuário, senha e *driver*. Seu uso consiste na representação de um banco de dados como um objeto. O *SQLAlchemy* [4] faz gerenciamento automático de múltiplas conexões, possibilitando o instanciamento de diversos objetos dessa classe;
- **Repository:** Classe que contempla os métodos básicos relacionados a consultas, inserções, atualizações e deleções. Seu uso consiste em tê-la como pai para cada nova classe com funções de acesso a dados. Possui um atributo do tipo *Database* que aponta para o banco de dados a ser utilizado.

Uma vez elaboradas tais classes auxiliares, tem-se a etapa seguinte que contempla o conceito de divisão de camadas (2.2.2), novamente orientação a objetos (2.2.3) e modelagem orientada a dados (2.2.1).

3.2.1.3.2 Divisão de Responsabilidades

Considerando uma arquitetura de camadas, o *Kpi-Executor* foi modularizado dividindo as responsabilidades entre estes módulos. Foram separadas entre os componentes as tarefas de se consumir e escrever em diferentes bases de dados, operar o fluxo de cálculos, registrar exceções e falhas de cálculo, etc. A figura 3.3 traz o diagrama de componentes que descreve tal modularização.

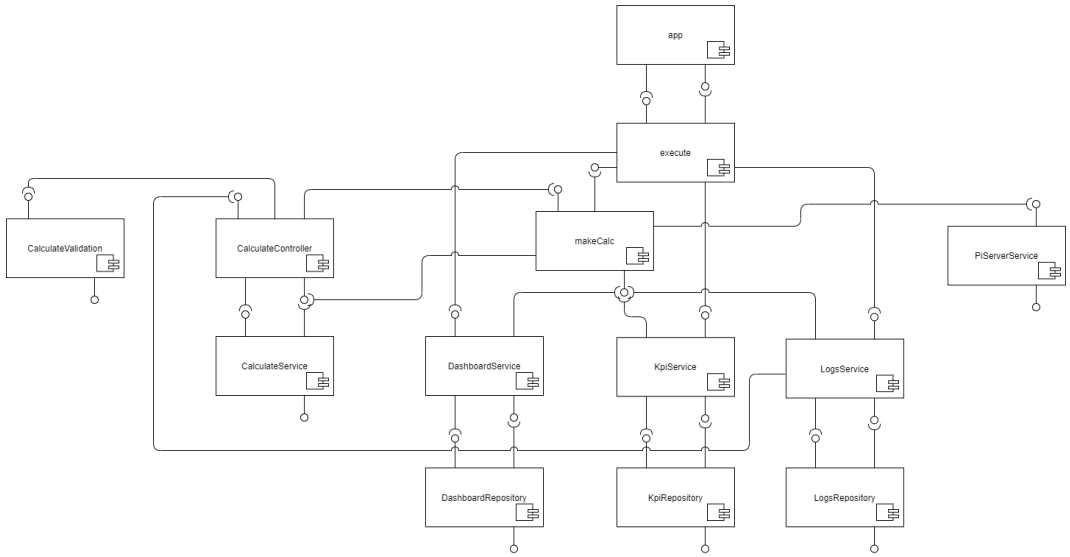

Figura 3.3: Diagrama de Componentes - Kpi-Executor

Cada componente descrito no diagrama é composto por um arquivo independente em Python [15], cada um com suas responsabilidades.

- **app:** Arquivo principal. Ele é o módulo invocado pelo orquestrador Kubernetes [11]. Responsável pela configuração de logs de manutenção e carregamento de variáveis de ambiente do sistema, configurando a aplicação para sua forma de execução local, em ambiente de validação ou produção;
- **execute:** Responsável por instanciar os objetos de banco de dados utilizados pelos próximos componentes. Também invoca DashboardService para montar o objeto com os dados de cadastro para as etapas de cálculos de KPIs;
- **makeCalc:** Opera o fluxo operacional da aplicação. Percorre o objeto de dados de cadastro fornecido pelo componente *execute* e solicitando, a cada iteração, ao *PiServerService* dados de processo cadastrados no PIMS, malha a malha;
- **DashboardService:** Possui regras de negócio relacionadas ao banco *omo-db*. Formata os dados lidos e para inserção neste banco. Também possui os métodos para instanciar os objetos de banco de dados e para solicitar escrita no banco de dados;
- **DashboardRepository:** Possui os métodos correspondentes às queries de inserção e consulta de dados em banco. Tem relação de herança [30] com a classe *Repository* (seção 3.2.1.3.1);
- **PiServerService:** Contempla regras de negócio relacionadas à consulta de dados no PIMS. Formata e realiza requisições *Http* para o módulo externo *Pi-Connector* (figura 3.1). Também possui rotinas para tratamento das respostas com os dados do PIMS;
- **KpiService:** Contempla as regras para formatação de dados calculados para inserção no banco *omokpi*;
- **LogsService:** Possui métodos para conexão com o banco de dados de logs de controle;
- **LogsRepository:** Possui métodos relacionados a inserção e leitura em dos logs em banco de dados. Tem relação de herança com a classe *Repository* (seção 3.2.1.3.1);
- **CalculateController:** Componente chave para o funcionamento da aplicação. Contempla o fluxo descrito pelo levantamento de requisitos na figura 3.2. Também invoca os módulos *KpiService* e *DashboardService* para registro de dados calculados e *LogsService* para persistência dos logs de controle;
- **CalculateService:** Possui os métodos para a realização dos cálculos dos índices de performance. É invocado pelo *calculateController*;
- **CalculateValidation:** Possui métodos auxiliares para a validação do fluxo de cálculos. É invocado pelo *CalulateController*;

3.2.1.3.3 Registro de Falhas de Cálculo

O fluxo de cálculo do módulo apresentado na seção 3.2.1.1.2 contempla o registro das falhas no processo de cálculo. Um dos requisitos levantados foi a necessidade do armazenamento desses casos, de forma organizada e escalável, viabilizando o consumo futuro dessas informações.

Uma boa prática para tratamento e organização de falhas, erros e casos inesperados é o tratamento de exceções. Ele é o mecanismo responsável pelo tratamento de condições que não respeitam o fluxo normal de funcionamento de um programa de computador. Considerando esses conceitos, foram criados diferentes tipos de exceção para diferenciar as falhas em cada etapa descrita pelo diagrama 3.2:

- **Exceções do Fluxo**: São lançadas em caso de problemas relacionados a ausência de dados necessários para os cálculos. Identificam qual dado está ausente, e correspondem às avaliações de condicionais do fluxo. No diagrama de sequência (3.2) são representadas pelos losangos.

  NoLoopDataException: Exceção lançada caso os dados do PIMS não estejam disponíveis por algum motivo;

  OfflineLoopException: Lançada caso a malha analisada naquela iteração esteja em estado *offline*, ou seja, sem o monitoramento do sistema;

  MVException: Semelhante à primeira, porém relativa somente aos dados de MV (seção 3.2.1.1.1). Caso não haja dados, a exceção é lançada;

  ModoException: Relativa à ausência de dados de Modo (seção 3.2.1.1.1). Caso não existam dados, é lançada uma exceção desse tipo;

  PVSPException: Relativa à ausência de dados de PV e/ou SP (seção 3.2.1.1.1). Caso não existam dados, é lançada uma exceção desse tipo;

- **Exceções nos cálculos:** São lançadas caso ocorra alguma falha durante as operações de cálculos. No diagrama de sequência (3.2) são representadas pelos retângulos.

  NoSatCalculateException: Falha no processo de cálculo do KPI *Tempo sem saturação*. É lançada uma exceção desse tipo com as informações da falha capturada;

  TimeNormalCalculateException: Falha no processo de cálculo do KPI *Tempo em modo normal*.

  EMACalculateException: Falha no processo de cálculo do KPI *Erro Médio Aceitável*;

  GlobalPerformanceCalculateException: Falha do processo de cálculo do KPI *Performance Global*.

Uma vez criados os tipos de exceções, para a possibilidade de consumo futuro destes dados, bem como para análise de possíveis gargalos no sistema, tais falhas foram registradas em uma tabela no banco de dados relacional PostgresSQL [14], conforme indicado na figura 3.1. A tabela responsável por esse armazenamento possui as seguintes informações:

- **message:** Mensagem que pode ser personalizada em cada caso;
- **file_name:** Arquivo do projeto que apresentou a exceção em questão;
- **line_number:** Linha do arquivo que apresentou a exceção;
- **exception:** Classe da exceção ocorrida;

- **documentation:** Informação nativa da exceção, caso não seja de uma das classes criadas;
- **web_id:** Identificador único da malha que operava o cálculo de KPI;
- **timestamp:** *Timestamp* com informação de quando ocorreu o problema;

3.2.1.3.4 Índices de Performance - KPIs

Como relatado na seção 2.3.2, os índices de performance calculados pelo módulo construído, se baseiam em índices frequentemente utilizados para a medição de qualidade de malhas de controle regulatório. Tais cálculos são realizados periodicamente, utilizando os dados de processo que foram armazenados no PIMS na hora anterior à sua execução. Dessa forma, todos os cálculos aqui descritos são relacionados a um período de uma hora. Nessa seção são descritos cada um deles.

- **Tempo em modo normal:** Para cada malha cadastrada no sistema LOOP (seção 3.1.2) é cadastrado um modo como estado normal. Este *KPI* consiste na avaliação da porcentagem do tempo em que tal malha esteve neste estado.

$$T_{normal}[\%] = \frac{n_{modoNormal}}{n_{totalAmostras}} \times 100 \tag{3.1}$$

  É utilizado pelo analista para inferências sobre possíveis problemas relacionados a uma má parametrização de um sistema de controle regulatório. Gera perguntas como "Porque esta malha não tem se mantido no modo de operação que deveria ser o normal?";

- **Tempo sem saturação:** Consiste na avaliação da porcentagem de tempo em que a MV se manteve dentro dos limites normais de operação previamente cadastrados

$$T_{semSaturacao}[\%] = \frac{n_{dentroLimite}}{n_{totalAmostras}} \times 100 \tag{3.2}$$

  Esse índice também diz sobre um sistema de controle mal parametrizado. Uma malha que possui um tempo sem saturação baixo pode apresentar defeitos em seus atuadores com uma frequência alta.

- **Erro médio aceitável:** O cálculo deste *KPI* passa por algumas etapas:

  Cálculo de erro: O erro é dado pela diferença absoluta entre o SP e a PV. Tal cálculo é realizado para cada amostra dos vetores de dados.

$$erro = |SP - PV| \tag{3.3}$$

  Assim é gerado um vetor com os erros de cada amostra.

  Boxplot: Do inglês *gráfico de caixa* é utilizado para a avaliação da distribuição empírica dos dados [23]. Nele são utilizados primeiro (Q_1), segundo (Q_2) e terceiro (Q_3) quartis para sua construção. Esta etapa é realizada com o vetor de dados gerado no item anterior.

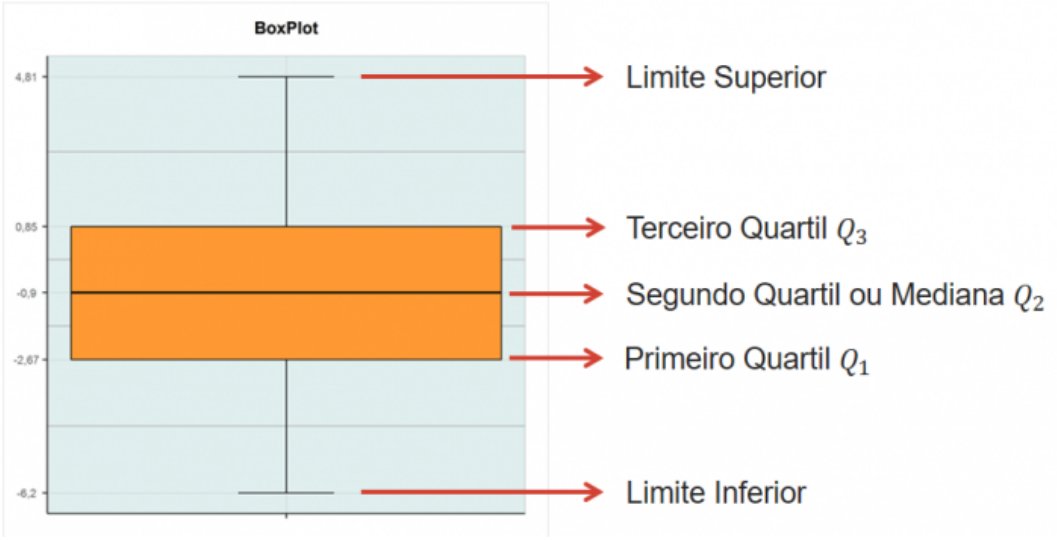

Figura 3.4: Formação de Boxplot (Adaptação de [10])

Para o cálculo de cada um dos quartis é utilizado o conceito de *percentil*. Ele divide o conjunto de dados em 100 partes iguais e indica a porcentagem de dados que estão abaixo daquela parte. Para a avaliação do percentil é necessário que os dados estejam ordenados de forma crescente. As relações matemáticas que o definem são dadas pelas equações 3.4 e 3.5.

$$L = \frac{k}{100} \times n \tag{3.4}$$

Sendo L a posição do percentil desejado no conjunto de dados, k o percentil desejado e n o tamanho do vetor de dados analisado.

$$P_k = \begin{cases} \frac{v_L + v_{L+1}}{2} & \text{caso } L \text{ seja inteiro} \\ v_{round(L)} & \text{caso } L \text{ não seja inteiro} \end{cases} \tag{3.5}$$

Sendo v o vetor de dados a ser analisado, L o valor obtido na equação 3.4 e P_k o k-ésimo percentil a ser obtido. Existe uma ligação direta entre percentis e os quartis utilizados para o *Boxplot*. As equações 3.6 a 3.8 trazem tais relações.

Sendo n o número de amostras, i o percentil desejado

$$Q_1 = P_{25} \tag{3.6}$$

$$Q_2 = P_{50} \tag{3.7}$$

$$Q_3 = P_{75} \tag{3.8}$$

Normalização: Uma vez obtidos os quartis Q_1 a Q_3, é obtido o valor de iqr, dado pela diferença entre Q_3 e Q_1.

$$iqr = Q_3 - Q_1 \tag{3.9}$$

Em seguida são inseridos os conceitos de *meta* e *limiar*. Ambos se apoiam nos quatis calculados até este ponto. Podem ser vistos como os limites inferior e superior, respectivamente, destacados na figura 3.4.

$$meta = Q_1 - Gap \times iqr \tag{3.10}$$

$$limiar = Q_3 + Gap \times iqr \tag{3.11}$$

Tendo $Gap = 0,1$ como padrão para o sistema. Em seguida é definido o *divisor*, dado pela diferença entre *limiar* e *meta*.

$$divisor = limiar - meta \tag{3.12}$$

Também é realizado o cálculo do chamado *Erro Médio Absoluto*, identificado como EMA. Consiste, como o nome explicita, na média dos valores absolutos de erro nos dados analisados.

$$EMA = \frac{\sum_1^{n_{amostras}} erro_i}{n_{amostras}} \tag{3.13}$$

Finalmente, após as etapas descritas é possível obter o *KPI* de *Erro Médio Aceitável*, identificado como EMA_{norm}. Vale ressaltar que esse *KPI* é, além de normalizado, complementar ao cálculo EMA. Isso ocorre pois dessa forma há uma condição vista como "boa"no processo apresentaria um valor mais próximo de 100%.

$$EMA_{norm}[\%] = 1 - \frac{EMA - meta}{divisor} \tag{3.14}$$

- **Nota Global:** É dada pela média dos *KPIs* calculados naquele ciclo. Caso o cálculo de um índice seja aplicável ao ciclo avaliado, ele entra para o cálculo deste.

$$NotaGlobal = \frac{\sum_i^n KPI_i}{n_{KPI}} \tag{3.15}$$

3.2.2 Testes e Implantação

Para implantar um sistema que será utilizado por um cliente, é necessário garantir seu pleno funcionamento. Dessa forma, a etapa de testes, automatizados ou não, de um software é indispensável [30]. Nessa seção são relatadas as etapas de testes, bem como o processo de implantação do módulo junto à estrutura existente do sistema LOOP.

3.2.2.1 Validação dos Cálculos

A primeira etapa de validações consistiu na execução manual de um único ciclo do sistema. Após o término, foram coletados os resultados dos cálculos para as malhas escolhidas. Para apoio e referência, foi utilizada uma planilha de *excel* para realização dos cálculos das mesmas malhas, no período de tempo de 06/04/2020 15:00 a 06/04/2020 16:00.

Inicialmente foram avaliados os valores obtidos para os cálculos intermediários que são realizados para o *KPI* de *Erro médio aceitável*. Assim foram avaliadas as etapas desse *KPI*, ponto a ponto. A tabela 3.2.2.1 traz o comparativo dos valores obtidos pelo *Kpi-Executor* e pela planilha auxiliar. Para essa etapa foi escolhida a malha indicada na mesma tabela.

Tabela 3.1: Validação de cálculos intermediários

*PID_16_*	**Q1**	**Q3**	**IQR**	**Limiar**	**Meta**	**Divisor**	**EMA**
Excel	8,9786	9,0175	0,0389	9,0214	8,9747	0,0468	9,0002
Kpi-Executor	8,9786	9,0175	0,0389	9,0214	8,9747	0,0468	9,0002

Uma vez validadas as etapas do cálculo do *KPI Erro médio aceitável*, foram avaliados os demais índices. A tabela 3.2.2.1 traz os valores obtidos em cada uma das operações, bem como os nomes das malhas analisadas.

Tabela 3.2: Validação de cálculos manuais

Malha	**Tempo S/ Saturação**		**Tempo Normal**		**Erro Médio Aceitável**		**Nota Global**	
-	*Excel*	*Kpi-Exec*	*Excel*	*Kpi-Exec*	*Excel*	*Kpi-Exec*	*Excel*	*Kpi-Exec*
PID_16_	0%	0%	100,00%	100,00%	45,57%	45,57%	48,52%	48,52%
PID_9_	100,00%	100,00%	100,00%	100,00%	26,67%	26,67%	75,56%	75,56%

Analisando as tabelas 3.2.2.1 e 3.2.2.1, observa-se o atendimento dos requisitos relacionados aos cálculos. Os valores obtidos pelo *Kpi-Executor* se adequaram à referência (cálculos manuais realizados em Excel) de forma satisfatória e precisa, sem erros para quatro casas decimais, na tabela 3.2.2.1 e com duas casas para a tabela 3.2.2.1. Vale ressaltar a utilização de quatro e duas casas respectivamente para os cálculos: No primeiro caso a escolha foi feita para uma obtenção melhor precisão nos cálculos sequentes, enquanto no segundo caso os dados são apresentados conforme a precisão exibida nas telas do sistema LOOP, que serão mostradas nas sub-seções que se seguem.

3.2.2.2 Validação em Ambiente de Testes

Posteriormente à validação dos cálculos descrita na seção anterior, iniciou-se a etapa de implantação do *Kpi-Executor* em ambiente de testes. Esse ambiente é composto por uma estrutura semelhante à do sistema final, porém não está diretamente ligada à dados de clientes. Dessa forma, é possível realizar testes em escalas e quantidades reais, porém não possibilitando o acesso do cliente final antes que todas as validações sejam realizadas.

Das etapas da validação em ambiente de testes foram realizadas as seguintes:

- **Implantação no ambiente de teste:** Construção de *Dockerfile* e configuração de orquestrador de containers (Kubernetes). Essa etapa foi realizada pelo time de *DevOps* [8] de *Discovery*;

- **Validação ciclicidade:** Foi avaliado o funcionamento cíclico do sistema, sendo observadas, através de consultas ao banco de dados no qual são registrados os cálculos dos *KPIs*, novas execuções do *Kpi-Executor* a cada hora. A figura 3.5 traz os registros em banco de dados de cada uma das execuções. Em destaque os registros de fim de ciclo de cálculo de cada execução das horas seguintes, comprovando a execução cíclica do *Kpi-Executor*;

malha character varying (50)	erro_medio_aceitável numeric (10,4)	tempo_modo_normal numeric (10,4)	tempo_sem_saturação numeric (10,4)	nota_global numeric (10,4)	fim_calculo timestamp without time zone
PID_16_	45.5714	100.0000	0.0000	48.5238	2020-04-06 19:00:00
PID_16_	84.4601	100.0000	0.0000	61.4867	2020-04-06 20:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-06 21:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-06 22:00:00
PID_16_	90.7228	100.0000	0.0000	63.5743	2020-04-06 23:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 00:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 01:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 02:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 03:00:00
PID_16_	11.7525	100.0000	0.0000	37.2508	2020-04-07 04:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 05:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 06:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 07:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 08:00:00
PID_16_	91.2063	100.0000	0.0000	63.7354	2020-04-07 09:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 10:00:00
PID_16_	50.9147	100.0000	0.0000	50.3049	2020-04-07 11:00:00
PID_16_	36.8826	100.0000	0.0000	45.6275	2020-04-07 12:00:00
PID_16_	[null]	100.0000	0.0000	50.0000	2020-04-07 13:00:00

Figura 3.5: Ciclos de execução - *Kpi-Executor*

- **Testes de Integração:** Como dito, o *Kpi-Executor* foi criado como uma reconstrução de um módulo anteriormente existente, que não atendia mais às especificações necessárias. Sendo assim, foi necessária a avaliação das telas do Loop que consomem dados referentes aos índices de performance, e de consultas de módulos adjacentes que pudessem possuir acoplamento garantindo assim que a integração entre os módulos ocorreu com sucesso. Os testes deste item foram realizados pelo *Product Owner* do Loop, utilizando o período de 06/04/2020 15:00 a 06/04/2020 16:00, validando o funcionamento dos pontos a seguir.

  Tela de Dashboard: Tela principal do sistema LOOP. Ela apresenta um overview baseado nos estados das malhas do cliente correspondente ao usuário logado, baseado nos índices de performance calculados para o período de acesso. A figura 3.6 traz os detalhes das validações descritas a seguir.

  1. O número de malhas *offline* deveria se mostrar coerente ao registrado no banco para aquele período em questão;
  2. O número de malhas em automático deveria se mostrar coerente ao registrado no banco para aquele período em questão;
  3. O número de malhas em modo manual deveria se mostrar coerente ao registrado em banco para aquele período em questão;
  4. Os valores de *KPIs* exibidos na tabela indicada deveriam estar coerentes com os registros inseridos pelo *Kpi-Executor* no banco de cálculos. Para essa validação, foram escolhidas malhas arbitrariamente, seus *KPIs* foram calculados de forma manual para o período de tempo em questão, e esses valores foram comparados aos exibidos nesse ponto da tela.

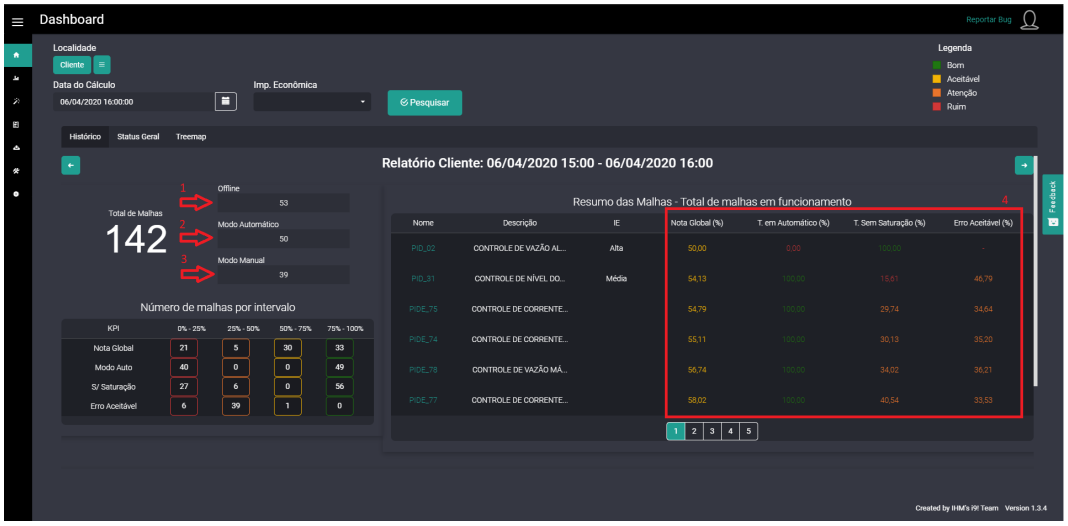

Figura 3.6: Tela de Dashboard - Loop

Tela de Diagnóstico: Nessa tela são exibidos os dados de processo e índices de performance da malha selecionada em um determinado período de tempo. Foi escolhida, de forma aleatória para avaliação da funcionalidade.

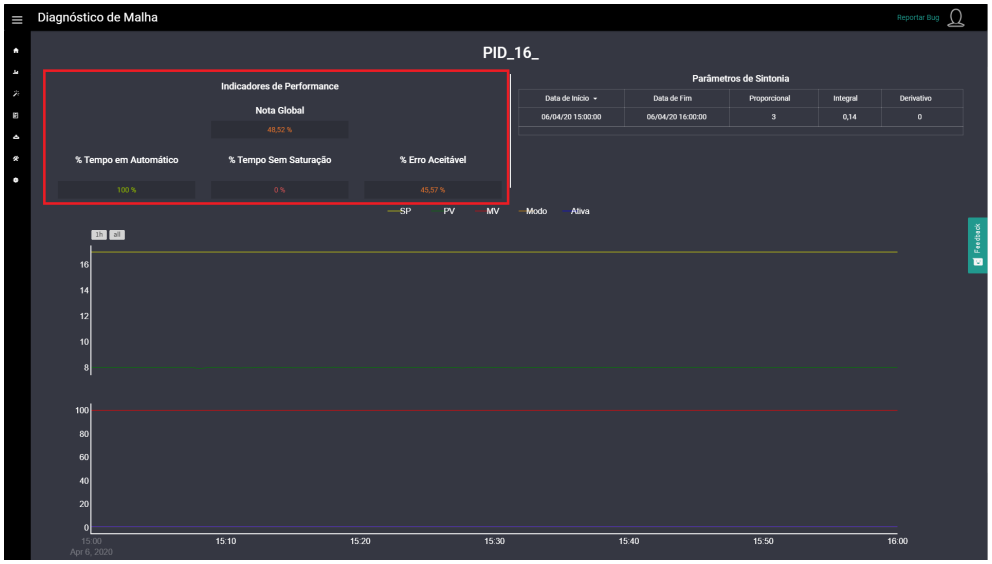

Figura 3.7: Tela de Diagnóstico - Loop

Tela de KPIs de Desempenho: Nessa tela podem ser comparados *KPIs* calculados para diferentes malhas, dado um período de tempo. Foi escolhida mais uma malha para o mesmo período de tempo. Em detalhes os pontos validados nesta tela:

1. Nomes das malhas selecionadas para comparação;
2. Legenda dos *KPIs* selecionados para comparação;
3. Lista dos *KPIs* selecionados.

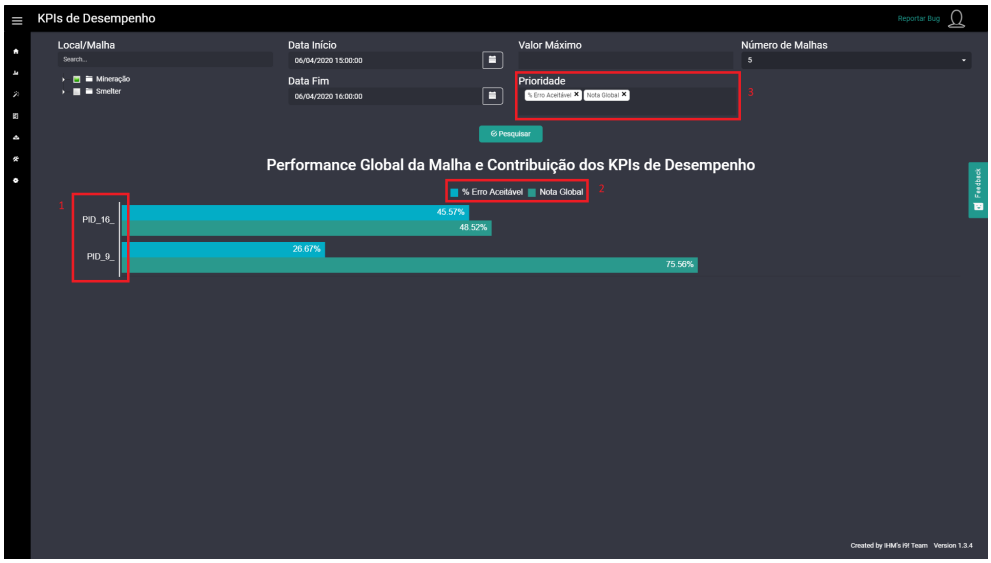

Figura 3.8: Tela de Desempenho de KPIs- Loop

Capítulo 4

Resultados

Anteriormente havia um módulo construído com propósito semelhante ao atual, entretanto, este começou a apresentar fragilidades à medida em que cresceram o número de clientes e malhas. Seu planejamento esperava um perfeito cadastro das entidades do sistema LOOP, assim apresentando problemas com frequência, comprometendo sua estabilidade.

O módulo citado não contemplava conceitos importantes para um software robusto, escalável e de boas condições para manutenção e evolução, como tratamento de exceções devidamente escolhido, *logs* informativos e objetivos em relação a seu funcionamento, nem mesmo registro dos possíveis problemas ocorridos. Não menos importante, o módulo também não possuía documentação mínima de suas regras de negócio, dificultando sua manutenção ou evolução, além de manter o conhecimento do sistema somente na cabeça de seus criadores.

Tendo em vista o impacto dos pontos citados, além de um sistema funcionando plenamente de acordo com as regras de negócio desenhadas na etapa de levantamento de requisitos (3.2.1.1.2), apresentado a seguir na sub-seção Sistema em Produção, foram avaliados como resultados importantes outros itens também apresentados nessa seção: Documentação das regras de negócio e Condições de Manutenção do Software.

4.1 Documentação de regras de negócio

A documentação das regras de negócio de um sistema é de grande importância para sua manutenção e evolução. No início do projeto, ainda na etapa de levantamento de requisitos (3.2.1.1.2), foram elaborados os diagramas correspondentes à estrutura existente , figura 3.1 e ao fluxo de cálculos, figura 3.2. Tais documentos são tradicionalmente utilizados para embasamento do desenvolvimento de um software. Entretanto, apesar de serem um ferramental para a elaboração de um software, no contexto do sistema existente, a elaboração deste documento foi considerada um resultado entregue. O documento elaborado contempla as seguintes informações:

- ***Overview* do sistema:** Descrição do *Kpi-Executor*, seu fluxo de funcionamento, conforme descrito pela figura 3.2, e os *links* para os demais repositórios que compõem o sistema LOOP;
- **Arquitetura:** Possui uma breve descrição e diagrama que relaciona o *Kpi-Executor* com os demais módulos, conforme figura 3.1. Também possui informações sobre as bases de dados utilizadas pelo sistema LOOP, e suas relações com cada módulo;

- ***Development:*** Possui informações para facilitação de integração de novos desenvolvedores ao time. Contempla passo a passo de instalação de dependências necessárias, etapas de configuração do sistema e diferentes formas de execução disponíveis;

O documento citado foi adicionado junto ao repositório de códigos *Gitlab* utilizado pelo time de *Discovery* da *IHM Stefanini*.

4.2 Condições de Manutenção do Software

A etapa de manutenção e evolução de um software é considerada a etapa mais cara de um processo de software [30]. Dessa forma, a facilitação dessa etapa tem grande valia para um time de software. Ela consiste na resolução de possíveis *bugs* que ocorram, além do atendimento de melhorias e evoluções do sistema.

A resolução de um *bug* de um software se inicia pela constatação e réplica do problema relatado, seguidos da identificação da causa do problema e posteriormente o entendimento e a implementação da solução. Das etapas relatadas, a de identificação da causa se mostra das mais trabalhosas, quando tratamos de um sistema complexo. Para auxílio e facilitação desta etapa foram gerados os processos de registros de exceções do *Kpi-Executor* (seção 3.2.1.3.3).

Tais registros de falhas possibilitam ao desenvolvedor, através de baixo esforço, a identificação de um problema ocorrido no cálculo dos índices de performance de uma malha específica. Todas as falhas são registradas em banco de dados, com informações detalhadas da origem do problema. A figura 4.1 traz um exemplo dessa identificação.

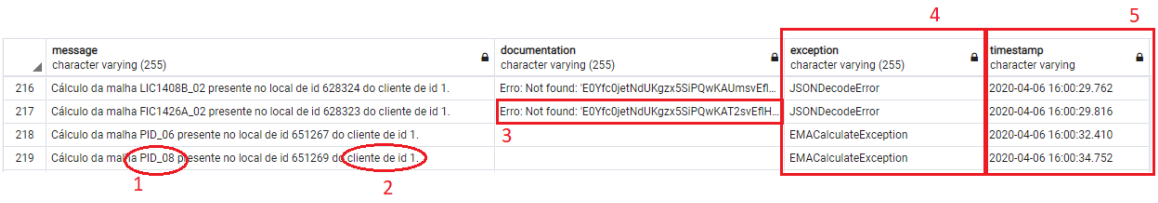

Figura 4.1: Registro de Falhas - Logs

Em destaque:

1. Identificação do nome da malha que apresentou falhas no cálculo de seus índices de performance;

2. Identificação de cliente a quem a malha pertence;

3. Detalhamento do problema quando este não é uma exceção prevista no conjunto de *Exceções de Fluxo* ou *Exceções de Cálculo* (seção 3.2.1.3.3);

4. Identificação de tipo de exceção. Nessa coluna são registradas as previstas no fluxo do sistema, mas também as geradas por outros motivos.

5. Registro de quando a falha ocorreu.

Com esses registros em banco de dados, a investigação de possíveis problemas relacionados ao *Kpi-executor* pode ser feita de forma orientada à falha, sendo possível a busca pelo nome da malha (presente na coluna *message*), tipo de falha (coluna *exception*) ou até mesmo por ciclo de execução (coluna *timestamp*).

4.3 Estabilização do Sistema

Considerando a estrutura existente, descrita na seção 3.1.2, a avaliação de estabilidade do sistema só pôde ser realizada após a garantia de uma boa integração aos demais módulos, em conjunto ao correto funcionamento cíclico, bem como a obtenção dos *KPIs* de forma correta, como demonstrado pela seção 3.2.2.2. A entrega de um módulo (*Kpi-Executor*) com caráter estável foi dos resultados mais importantes para o sistema LOOP como um todo, gerando mais confiança tanto aos usuários do sistema, quanto ao time responsável pela manutenção e evolução do sistema LOOP.

Para a avaliação da estabilidade do sistema foi utilizado o orquestrador de *containers Kubernetes* [11]. Tal estrutura configurada pelo time de *DevOps* do setor *Discovery* da *IHM Stefanini*, além de garantir a disponibilidade dos módulos do sistema LOOP e integrar fluxos de CI/CD, permite também a observação do funcionamento individual de cada módulo.

O *Kpi-Executor* foi estruturado com sua execução como *Job* no orquestrador de *containers*. Esse tipo de execução consiste em disparos cíclicos de uma instância do código escolhido. Na ferramenta também é possível avaliar o estado de *sucesso* ou *insucesso* de cada instância gerada.

Em detalhe:

1. Lista de nomes dos *jobs* executados junto ao estado da execução;
2. Tempo desde a execução daquela instância de *Kpi-Executor*

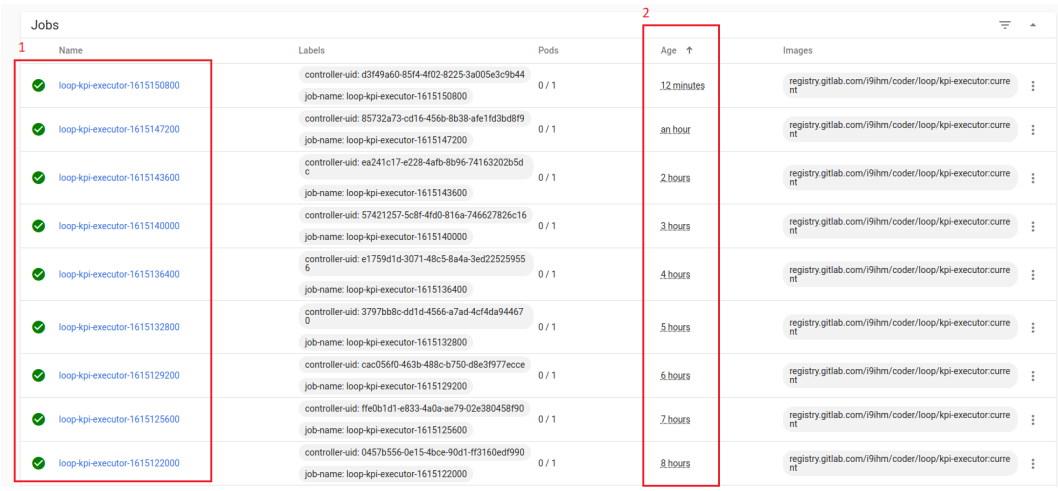

Figura 4.2: Status Execuções - Kubernetes

Estão sempre disponíveis junto ao orquestrador de *containers* os dados das últimas vinte e quatro execuções dos *jobs* do *Kpi-Executor*, bem como outras informações pertinentes à estabilidade do sistema LOOP. A figura 4.3 traz tais dados.

Figura 4.3: Histórico Execuções - Kubernetes

Tendo em detalhes:

1. Uso de CPU e Memória da máquina que hospeda o sistema;
2. As últimas vinte e quatro execuções dos *jobs* de *Kpi-Executor*, tendo todas em estado de sucesso;

Com os dados exibidos tem-se uma avaliação positiva da estabilidade do *Kpi-Executor*, este se mantendo com execuções consistentes, respeitando seu agendamento e não apresentando grandes problemas de indisponibilidade ou falhas de execução.

Capítulo 5

Conclusões

5.1 Considerações Finais

Os objetivo do trabalho de implementação de uma aplicação para cálculos de *KPIs* foi alcançado com sucesso. Durante as etapas de validação de cálculos, seção 3.2.2.2, foram obtidos dados precisos, quando comparados aos cálculos manuais realizados via *Excel*, assim atendendo as especificações de cada um dos *KPIs*. Juntamente a essas validações, a execução cíclica proposta foi atendida, tendo tal afirmação validada pela seção 3.2.2.2 e reafirmada pelo monitoramento de estabilidade do sistema, na seção 4.3.

Quanto à integração do *Kpi-Executor* com a estrutura existente, também pode ser destacada a seção 3.2.2.2 que apontou o sucesso obtido nos testes realizados pelo *Product Owner* do sistema, atendendo assim, de forma satisfatória, o requisito de integração ao sistema existente. O *Kpi-Executor* segue em execução sem situações não esperadas nesse quesito.

Outro ponto de destaque, não menos importante, é o registro de falhas dos cálculos. Este registro se expandiu para uma classificação das falhas sendo essas divididas em *Exceções de Fluxo* e *Exceções de Cálculos* (seção 3.2.1.3.3), com a adição do detalhamento de cada falha prevista no levantamento de requisitos, além do registro de outras falhas não previstas. Esse registro se tornou ferramenta indispensável para o time de manutenção e evolução de *Discovery*, facilitando a etapa investigação de possíveis *bugs* no sistema.

Também associada à melhoria das condições de manutenção e evolução do software está a documentação elaborada para o *Kpi-Executor*. Ela apresenta a arquitetura geral do sistema LOOP e fluxo de operação da aplicação, pontos importantes para o entendimento do funcionamento do sistema, sendo também um auxiliar importante na manutenção do módulo. Além destes pontos, a documentação também possui passo-a-passo de instalação de dependências e execução do módulo, sendo ferramenta para facilitação do futuro ingresso de novos membros ao time.

Com o projeto foi possível avaliar a complexidade envolvida no processo de desenvolvimento de um software, desde sua concepção arquitetural até sua disponibilização em ambiente de produção. Também foi um desafio notável a construção de um módulo que deveria se encaixar como uma peça de quebra-cabeça em uma estrutura existente complexa e em operação. O time de *DevOps* teve papel muito importante para que a entrega final do módulo acontecesse com sucesso. Também vale ressaltar a dificuldade na construção da monografia. A busca por referências aconteceu de forma bastante dispersa, já que o projeto envolveu conceitos, de um lado diretamente ligados à Engenharia de Controle e Processos Industriais, e

de outro ligados a Engenharia e Desenvolvimento de Software. Ao final, tem-se um trabalho que aborda amplamente áreas de grande relevância para um Engenheiro (a) de Controle e Automação, integrando conhecimentos de processos, controle regulatório e sua qualidade, em conjunto com arquitetura e desenvolvimento de softwares.

5.2 Propostas de Continuidade

5.2.1 Cálculo de Baseline Parametrizável

Os parâmetros de *limiar* e *meta* que são utilizados para o cálculo do *KPI* de *Erro médio aceitável*, descrito na seção 3.2.1.3.4, foram posteriormente ao desenvolvimento do processo denominados como *parâmetros de baseline*. Tais informações são calculadas de forma automática, sem interação com o usuário do sistema. Entretanto, foi previsto como evolução para o *Kpi-Executor* a inserção manual desses parâmetros, através da interface do sistema LOOP.

5.2.2 Monitoramento de métricas

O módulo principal do sistema LOOP denominado *Dashboard* (3.1.2) possui um painel de monitoramento de métricas utilizando o software *Grafana*, que consiste em uma poderosa plataforma aberta para observação de informações de funcionamento de aplicações em geral. Nesse painel (5.1) são exibidas métricas importantes para o acompanhamento de um software. Em detalhe:

1. ***Uptime:*** Porcentagem de tempo em que o sistema esteve disponível;
2. **MTBF - *Mean Time Between Failures*:** Tempo entre falhas ocorridas no sistema;
3. **MTTR - *Mean Time To Recovery*:** Tempo de resposta para a retomada de funcionamento do sistema;
4. ***Events*:** Eventos ligados à queda do sistema;
5. ***Availability*:** Estado de disponível ou indisponível do sistema;

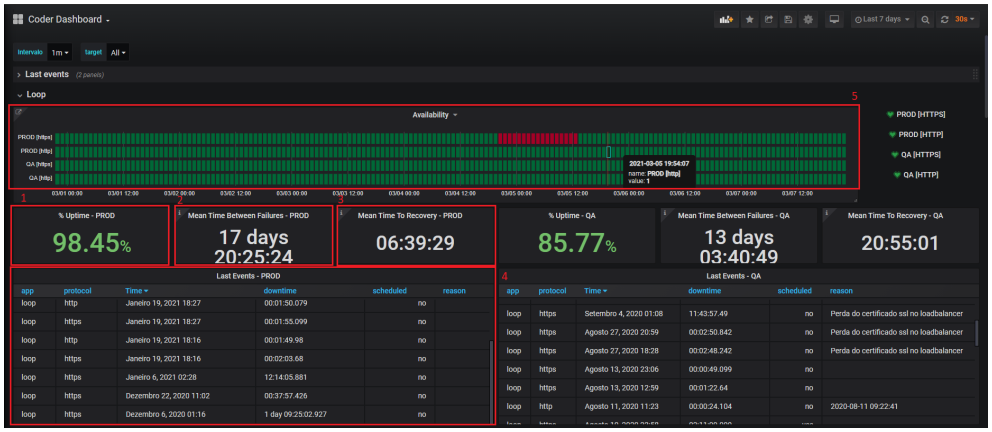

Figura 5.1: Métricas LOOP - Grafana

Tal monitoramento é de muita valia para softwares e posteriormente será implementado junto ao *Kpi-Executor*.

Referências Bibliográficas

[1] Introdução aos containers e ao Docker, 2018. URL https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/container-docker-introduction/.

[2] Página Oficial Visual Studio Code, 2019. URL https://code.visualstudio.com/.

[3] ORM: Object relational mapper, 2020. URL https://www.devmedia.com.br/orm-object-relational-mapper/19056.

[4] Página Oficial SQLAlchemy, 2020. URL https://www.sqlalchemy.org/.

[5] Página Oficial AngularJS, 2020. URL https://angularjs.org/.

[6] Página Oficial AWS, 2020. URL https://aws.amazon.com/pt/.

[7] Integração e entrega contínuas: pipeline ci/cd, 2020. URL https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd.

[8] O que é devops?, 2020. URL https://www.redhat.com/pt-br/topics/devops.

[9] Página Oficial Docker, 2020. URL https://www.docker.com/.

[10] Portal Action - Boxplot, 2020. URL http://www.portalaction.com.br/estatistica-basica/31-boxplot.

[11] Página Oficial Kubernetes, 2020. URL https://kubernetes.io/pt/.

[12] Página Oficial NodeJS, 2020. URL https://nodejs.org/en/.

[13] Página Oficial PI System, 2020. URL https://www.osisoft.pt/pi-system/.

[14] Página Oficial PostgreSQL, 2020. URL https://www.postgresql.org/.

[15] Página Oficial Python, 2020. URL https://docs.python.org.

[16] Página Oficial Spring Boot, 2020. URL https://spring.io/projects/spring-boot.

[17] Microsoft SQL Server, 2020. URL https://www.microsoft.com/pt-br/sql-server/sql-server-downloads.

[18] What is a container?, 2020. URL https://www.docker.com/resources/what-container.

[19] O que é kernel?, 2020. URL https://www.tecmundo.com.br/macos/1636-o-que-e-kernel-.htm.

[20] Kubernetes e a tecnologia de containers, 2020. URL https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes.

[21] SQL injection, 2021. URL https://www.devmedia.com.br/sql-injection/6102.

[22] G. BOOCH, J. RUMBAUGH, and I. JACOBSON. *UML, Guia do Usuario*. 2ª edition, 2005.

[23] Capela, Marisa Veiga and Capela, Jorge M V. Elaboração de Gráficos Box-Plot em Planilhas de Cálculo. page 5.

[24] L. H. C. M. e Dilson José L. R. Júnior. Aplicação do ORM Doctrine para abstração de banco de dados no desenvolvimento de software em PHP na UNASUS/UFMA.

[25] A. G. Hospedagem Cloud, 2020. URL https://www.hostinger.com.br/tutoriais/o-que-e-hospedagem-cloud/.

[26] T. J. Harris. Recent Developments in Controller Performance Monitoring and Assessment Techniques. .

[27] T. J. Harris. Assesment of Control Loop Performance. .

[28] H. Luo, S. X. Ding, A. Haghani, H. Hao, S. Yin, and T. Jeinsch. Data-driven design of kpi-related fault-tolerant control system for wind turbines. In *2013 American Control Conference*, pages 4465–4470, 2013. doi: 10.1109/ACC.2013.6580528.

[29] D. Mourtzis, S. Fotia, and E. Vlachou. Lean rules extraction methodology for lean pss design via key performance indicators monitoring. *Journal of Manufacturing Systems*, 42:233 – 243, 2017. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2016.12.014. URL http://www.sciencedirect.com/science/article/pii/S0278612516301017.

[30] I. SOMMERVILLE. *Engenharia de Software*. 9ª edition, 2011.

[31] O. K. TAKAI. *Introdução a Banco de Dados*. 2005.

[32] C. Wang and S. Zhou. Control of key performance indicators of manufacturing production systems through pair-copula modeling and stochastic optimization. *Journal of Manufacturing Systems*, 58:120 – 130, 2021. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2020.11.003. URL http://www.sciencedirect.com/science/article/pii/S0278612520301904.

[33] J. Zevenbergen, J. Gerry, and G. Buckbee. AUTOMATION KPIs CRITICAL FOR IMPROVEMENT OF ENTERPRISE KPIs. page 4.