

**HORUS FACULDADES
SISTEMAS DE INFORMAÇÃO**

**MONITORAMENTO E OBSERVABILIDADE PARA ARQUITETURA DE
SOFTWARE DISTRIBUÍDA**

Odair Belo Sehn

Pinhalzinho – SC

2023

**HORUS FACULDADES
SISTEMAS DE INFORMAÇÃO**

Odair Belo Sehn

**MONITORAMENTO E OBSERVABILIDADE PARA ARQUITETURA DE
SOFTWARE DISTRIBUÍDA**

Monografia de trabalho de graduação de curso apresentado à Horus Faculdades, como parte dos requisitos para obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Ricardo Hendges

Pinhalzinho – SC

2023

“Não é o conhecimento, mas o ato de aprender, não a posse mas o ato de chegar lá, que concede a maior satisfação.”

Carl Friedrich Gauss.

SUMÁRIO

1. INTRODUÇÃO	4
2. JUSTIFICATIVA	5
3. OBJETIVOS	6
3.1 Geral	6
3.2 Específico	6
4. REFERENCIAIS TEÓRICOS	7
4.1 Criação da tecnologia de informação	7
4.2 Evolução da tecnologia de informação	7
4.3 Sistemas de Informação	8
4.4 Sistemas de software na atualidade	9
4.5 Arquitetura de software	11
4.5.1 Cliente Servidor	11
4.5.2 Multicamadas	13
4.5.3 SOA (Arquitetura orientada a serviços)	14
4.5.4 Microsserviços	16
4.6 Observabilidade	17
4.7 Monitoramento	18
4.8 Monitoramento x Observabilidade	19
5. METODOLOGIA DA PESQUISA	20
6. CRONOGRAMA	21
7. ANÁLISE DOS REQUISITOS	21
7.1 UML	22
7.1.1 Diagrama Entidade-Relacionamento	22
7.2 GitHub	23
8. DESENVOLVIMENTO DO BACK-END	23
8.1 Banco de Dados	24
8.2 APIs	24
8.3 JavaScript	25
8.4 Node.js	25
8.5 Express.js	27
8.6 Documentação das APIs	28
8.6.1 Swagger	28
8.7 Back-end finalizado	30
9. DESENVOLVIMENTO DO FRONT-END	32
9.1 Vue.js	33
9.2 Quasar Framework	34
9.3 Vuex	34

	5
9.4 Axios	35
9.5 Front-end finalizado	35
10. DOCKER	37
10.1 Docker Compose	38
11. MONITORAMENTO E OBSERVABILIDADE	38
11.1 Elasticsearch	38
11.2 APM Server	40
11.3 Kibana	43
11.4 Monitoramento e Observabilidade finalizados	43
12. REFERÊNCIAS BIBLIOGRÁFICAS	47

AGRADECIMENTOS

Antes de tudo gostaria de agradecer aos familiares e amigos mais próximos, por todo apoio e ajuda, que com toda certeza tem uma parcela na realização deste trabalho.

Ao meu pai, Jorge Sehn, que é um exemplo de pai e mesmo sozinho, nunca mediu esforços para me proporcionar um ensino de qualidade para ter prosperidade no futuro.

Aos professores, que dentro da sala de aula contribuíram com correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional ao longo do curso, mas que fora da sala de aula também se tornaram grandes amigos.

Ao meu orientador Ricardo Hendges, sempre disponível a compartilhar todo o seu vasto conhecimento, que durante o andamento do curso acabou virando colega de trabalho, me ensinando diariamente. Também ao Juliano Eichelberger, que foi um segundo orientador neste trabalho, pela cumplicidade e pelo apoio sempre que necessário, sem ele o trabalho não seria possível.

Aos meus colegas de curso, com quem pude ter o privilégio de trocar experiências nestes últimos anos, que me fizeram crescer não só como pessoa, mas também como formando.

A todos os alunos da minha turma, pelo ambiente descontraído na sala de aula, estes que criei laços fortes de amizade.

A instituição Horus Faculdades, essencial no meu processo de formação, pela dedicação e por tudo que aprendi ao longo desses anos.

A minha namorada, Kerli Laiz Dill, pelo apoio e companheirismo, que está comigo desde o início dessa trajetória,

A todos aqueles que de alguma forma, em algum momento, fizeram parte dessa trajetória, tornando esse trabalho possível.

LISTA DE FIGURAS

Figura 1 - Funções básicas de um sistema de informação	6
Figura 2 - Tempo gasto em aplicativos no segundo trimestre de 2021	7
Figura 3 - Arquitetura cliente servidor	8
Figura 4 - Arquitetura multicamadas	8
Figura 5 - Arquitetura de microsserviços	10
Figura 6 - Modelo Node.js vs Tradicional	17
Figura 7 - Criação de documentação via Swagger	18
Figura 8 - Apresentação das rotas no Swagger UI	19
Figura 9 - Estrutura de pastas do projeto back-end	20
Figura 10 - Exemplo de API de inserção na pasta Services	20
Figura 11 - Exemplo de API de inserção na pasta Controllers	20
Figura 12 - Interface gráfica desenvolvida no front-end	22
Figura 13 - Estrutura de pastas do projeto front-end	23
Figura 14 - Elasticsearch declarado no docker-compose	24
Figura 15 - APM Server declarado no docker-compose	25
Figura 16 - Instalada dependência do APM no back-end	25
Figura 17 - Configuração do APM no back-end	26
Figura 18 - Kibana declarado no docker-compose	27
Figura 19 - Requisições feitas no front-end	27
Figura 20 - Transações registradas no Kibana	27
Figura 21 - Gráficos ao detalhar transação GET do produto	28
Figura 22 - Amostra de rastreamento de requisição GET	28
Figura 23 - Informações ao detalhar SQL GET	29
Figura 24 - Gráficos do processamento e armazenamento do servidor	29

1. INTRODUÇÃO

Atualmente a população mundial está cada vez mais conectada à internet, os aplicativos fazem parte do dia a dia da grande maioria da população, microempresas, pequenos negócios e grandes corporações empresariais. Os celulares já não são destinados somente para utilizar as redes sociais e trocar mensagens, e se tornaram um grande potencial para alavancar os negócios.

De acordo com a Neotrust/Movimento Compre & Confie, em parceria com o Cômite de Métricas da Câmara Brasileira da Economia Digital (2022), o faturamento de vendas na internet em 2021 teve um aumento expressivo em relação a 2020, tendo uma alta máxima de 48,41%. Considerando o mesmo período de vendas nesses dois anos, o ano de 2021 teve um crescimento de 35,36% em relação ao ano anterior (Mercado e Consumo, 2022).

Com essa grande demanda na utilização de sistemas de software, além de surgirem muitas novas aplicações, a grande maioria também cresceu, onde destaca-se algumas empresas como google, microsoft, amazon, netflix, que dispõem de sistemas de softwares enormes, que são utilizados mundialmente com grande frequência. Para atender essa alta demanda de acessos, e também como um processo natural do meio da tecnologia, existe a evolução dos softwares, sendo na adoção de novos métodos de desenvolvimento, novas arquiteturas de software, utilizando novas linguagens de programação ou novas tecnologias. Isso tudo resulta numa maior complexidade a nível de desenvolvimento, que gera um desafio para os profissionais de TI, que é de manter essas grandes aplicações estáveis e com alta disponibilidade.

Para destacar a importância do bom funcionamento e disponibilidade dos aplicativos, é possível citar o acontecido do dia 04/10/2021, onde houve a queda por aproximadamente 7 horas do uso do Facebook, Instagram e Whatsapp, o que ocasionou uma quantidade massiva de reclamações por parte dos comércios, que ficaram impossibilitados de realizar vendas pela indisponibilidade dos aplicativos. Segundo O Especialista (2021) durante o apagão das redes, Zuckerberg dono desses aplicativos, teve uma perda estimada de US\$ 5,9 bilhões em sua fortuna pessoal, segundo cálculos feitos com base nas ações da empresa.

2. JUSTIFICATIVA

Os sistemas de software são essenciais no cotidiano da população mundial, desempenhando uma função importante na sociedade e economia. Caso aconteça falhas nos softwares, as partes envolvidas sofrem um grande impacto, podendo prejudicar negócios inteiros e causar danos irreversíveis na pior das hipóteses (Jeanderson Cândido, Maurício Aniche, Arie van Deursen, 2021). Recentemente, em 2019, foi evidenciado uma parada global dos serviços da google cloud, após um erro de gerenciamento do data center, onde os mesmos acusam não ter sido ataques cibernéticos, e sim uma cascata de configurações incorretas e bugs de software. Essa interrupção prolongada impossibilitou acesso ao gmail, instabilidades nos aplicativos youtube, snapchat, entre outros que utilizam o armazenamento em nuvem da google cloud (Barrett, 2019).

Para prevenção de eventuais falhas, os testes de software têm um papel de grande relevância, porém, os desenvolvedores e equipes de operações necessitam do monitoramento de sistemas para um entendimento assertivo de como o sistema vai se comportar em produção (Jeanderson Cândido, Maurício Aniche, Arie van Deursen, 2021). Essa interação entre as equipes de desenvolvimento e operação resultou num termo conhecido como DevOps, onde ambas equipes caminham juntos com objetivo de entregar um software de qualidade e de valor na entrega, combinado com uma velocidade adequada. Nesse modelo existe um ciclo contínuo entre as duas equipes, onde a responsabilidade pela entrega é compartilhada e busca-se a automatização de tudo aquilo que for possível, principalmente testes e entregas, para alcançar o objetivo de uma entrega de qualidade. Em uma empresa comum, é normal dispor de um número grande de servidores, o que dá mais destaque para essas medidas de automatização, já que é inviável a atualização e monitoramento desses serviços individualmente. Nestes casos existem ferramentas de orquestração e monitoramento que auxiliam nessas tarefas (Vertigo, 2018).

Com a observabilidade e monitoramento do software é possível prever possíveis falhas, antes que algo prejudicial aconteça, através de monitoramento dos logs e métricas coletadas e dos alertas pré configurados. Além disso, fornece uma visão clara de como o seu sistema funciona, já que fica mais fácil entender o fluxo inteiro de um serviço, o que ganha maior ênfase quando tratamos de um software que dispõe de uma arquitetura distribuída, que geralmente é mais complexa.

3. OBJETIVOS

3.1 Geral

Como principal objetivo, é utilizar um conjunto de ferramentas de código aberto de monitoramento e observabilidade de software com arquitetura distribuída, que oferecem a possibilidade de localizar falhas e auxiliar os profissionais de TI na descoberta de pontos críticos que podem estar afetando o bom funcionamento de um sistema de software, seja ele de pequeno ou grande porte, através da possibilidade de configurar alertas e visualizações gráficas intuitivas. Buscando fornecer uma visão mais clara para a área de TI de como o sistema de software está se comportando, o que pode resultar em maior rapidez na manutenção e prevenir possíveis falhas, fornecendo uma melhor experiência do usuário final. Quando trata-se de bom funcionamento, é abordado todas as questões de segurança da informação, consultas otimizadas e disponibilidade.

3.2 Específico

1. Coletar métricas e logs do sistema, com o armazenamento de forma centralizada;
2. Possibilitar rastreamento da execução de serviços;
3. Proporcionar análise das métricas e logs de modo a prevenir possíveis falhas na infraestrutura;
4. Possibilitar exibição de logs e métricas de forma simplificada, através de visualizações gráficas intuitivas;

4. REFERENCIAIS TEÓRICOS

4.1 Criação da tecnologia de informação

Segundo Laia (2013), inicialmente o processamento dos dados era feito somente no hardware, parte física do computador, o que significa que o computador não tinha nenhum programa instalado e necessitava ser fisicamente mudado conforme a finalidade, com objetivo de atender determinada demanda. Houve uma grande virada de chave quando Leibniz criou o código binário, que foi o fundamental na criação dos computadores modernos, juntamente com o EDVAC, CPU criado por John Von Neumann em 1945 e também os Mark I, da universidade de Harvard. Estes aplicativos só vieram a ser chamados de software em 1958, quando John Hilder Tukey, um cientista americano, utilizou o termo em um artigo.

Na década de 1960, a utilidade da tecnologia entre as organizações era o “processamento de dados”. Nessa época, a maioria das empresas dedicava os recursos para o processamento centralizado desses dados em grandes computadores e para os sistemas de controles operacionais, como folha de pagamento, contabilidade, estoque, faturamento e finanças. O processamento dos dados tinha como objetivo principal a substituição de mão de obra e redução de custos. Os poucos recursos existentes eram totalmente centralizados na área de processamento de dados e as funções de informática praticamente não existiam. Conforme o tempo foi passando, as empresas foram percebendo a importância da informática para gestão de negócios e incorporando-a como ferramenta de gestão empresarial, onde se inicia já pequenas integrações entre sistemas, mesmo que com algumas redundâncias (Rezende, 2002).

4.2 Evolução da tecnologia de informação

Segundo Rezende (2002) na atualidade, a informática se transformou em tecnologia de informação, contando com diversas integrações em modernos recursos. A tecnologia da informação ou TI pode ser definida como um conjunto de recursos tecnológicos e computacionais para armazenamento de dados, para geração e uso da informação e de conhecimentos. A TI possibilita às empresas a

terem diversas aplicações que se comuniquem, com diferentes bases de dados, eliminando dados redundantes. Além disso, a tecnologia de informação contempla os sistemas de informação, que fornecem um apoio para as organizações na tomada de decisões. Portanto, a utilização da TI, em conjunto com os sistemas de informação, são um diferencial competitivo para as empresas, fornecendo facilidade de gestão e inteligência empresarial.

Turban, Velonino (2013) definem que a TI é o conjunto de componentes computacionais utilizados em uma organização, e que são de extrema importância em todas disciplinas de negócios, pois tem impacto nas operações, no comércio eletrônico, na logística, nos recursos humanos e no relacionamento com os consumidores e parceiros de negócios. Portanto, tudo aquilo que envolva negócios ou estratégia corporativa passa em algum momento pela TI.

4.3 Sistemas de Informação

Existe uma diferença entre TI e sistemas de informação, mesmo que estes sejam sinônimos. A TI é uma definição mais ampla, refere-se ao lado tecnológico de um sistema de informação, onde contempla todos os recursos tecnológicos utilizados para geração desses dados, como hardware, software e gestão de dados. Um sistema de informação tem uma função mais específica, onde é responsável por coletar, armazenar, analisar, processar e utilizar essas informações obtidas para uma finalidade específica. Resumindo em quatro funções básicas, seriam a entrada, processamento, armazenamento e saída dos dados, conforme ilustra a figura 01 (Turban, Velonino, 2013).

Figura 1 - Funções básicas de um sistema de informação



Fonte: Turban, Volonino (2013)

Segundo Filho (2015), o conceito de sistemas de informação é um conjunto de componentes da estrutura organizacional, onde o fluxo de informações internas e externas da empresa é processado de forma ordenada. O sistema de informação desempenha também a obtenção de dados para o planejamento e operação de controle da empresa, que posteriormente esses dados organizados se tornaram subsídio para ajuda no processo de tomada de decisões. Onde o mesmo destaca que os sistemas de informações não tratam somente dados externos, e também informações internas decorrentes da operação da própria empresa.

4.4 Sistemas de software na atualidade

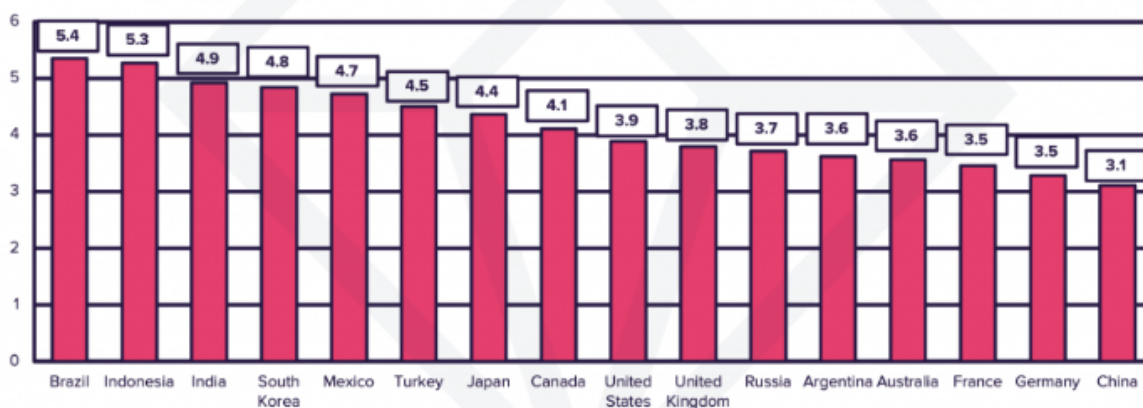
No mundo moderno, tudo é software. Todas empresas, indiferente do seu tamanho, necessitam de sistemas de informação para automatizar seus processos. Os governos utilizam sistemas computacionais para interagir com os cidadãos, como por exemplo, para coletar impostos ou realizar eleições. Empresas utilizam os softwares para realizar vendas e publicar seus produtos diretamente para o consumidor final, através de sistemas de comércio eletrônico. O software também aparece embarcado em diferentes dispositivos e produtos da engenharia, como automóveis, satélites, aviões, robôs, etc. Enfim, o software contribui para renovar indústrias e serviços tradicionais, como hospedagem, telecomunicações, publicidade, transporte em grandes centros urbanos e lazer (Valente, 2021).

Nos últimos três anos, os pequenos negócios no Brasil apostaram na informatização e na utilização de novas ferramentas digitais, em especial nas redes sociais. Atualmente, 72% do segmento utilizam o WhatsApp para se comunicar com clientes e 40% possuem perfil no Facebook. É o que mostrou a pesquisa “Transformação Digital nas MPE”, realizada pelo Sebrae entre abril e junho deste ano, e que identificou a informatização das micro e pequenas empresas. A pesquisa avaliou como o setor está envolvido no processo de mudança para a era digital, confirmando o crescimento do grau de informatização das empresas de micro e pequeno porte. O aplicativo WhatsApp e a rede social Facebook são as ferramentas mais usadas pelas MPE na divulgação de produtos e serviços. Também são aproveitadas para estreitar o relacionamento com os clientes e ampliar vendas (Sebrae, 2018).

“O forte crescimento das vendas online em 2021 revela um novo hábito do consumo, com migração de compras para a internet. No ano passado, não tivemos períodos críticos de confinamento e quarentena como em 2020, e, mesmo assim, os consumidores adotaram em maior proporção às compras remotas. Em novembro de 2021, o e-commerce representou 17,9% das vendas varejistas, um recorde no histórico observado desde janeiro de 2018, quando a penetração era de apenas 4,7%”, analisa o responsável pela divisão de Varejo Online da camara-e.net Gastão Mattos (Mercado e Consumo, 2022).”

Em outra pesquisa realizada pelo App Annie (2021), aponta que o mercado global de aplicativos bateu recorde histórico de gastos no segundo trimestre de 2021, com valor de U\$34 bilhões. A pesquisa também aponta que oito países gastam mais de quatro horas diárias usando aplicativos de celular, inclusive o Brasil lidera o ranking com média de 5.4 horas diárias. A pandemia gera um impacto nesses números, onde foi constatado um considerável aumento de 45% de utilização no número de horas que as pessoas utilizam aplicativos de celular nesse período.

Avg. Daily Hours Spent in Apps Q2 2021 Select Markets



Source: App Annie Intelligence

Note: Android phones.

Fonte: App Annie Intelligence (2021)

4.5 Arquitetura de software

Garlan e Shaw (1994, p. 5) definem a arquitetura de software como uma solução para softwares maiores e mais complexos, que o problema não se resume somente em algoritmos e estrutura de dados, onde tem a principal função de definir a estrutura geral do projeto, questões como organização bruta e estrutura do controle global, protocolos para comunicação, sincronização e acesso de dados, dimensionamento e desempenho, atribuição de funcionalidades aos elementos de design e seleção entre alternativas de projeto.

Segundo Valente (2021), a arquitetura de software preocupa-se com projetos em mais alto nível. Ou seja, o foco não é destinado na organização ou interfaces de classes individuais, mas a unidade de maior tamanho, sejam elas pacotes, componentes, módulos, subsistemas, camadas ou serviços.

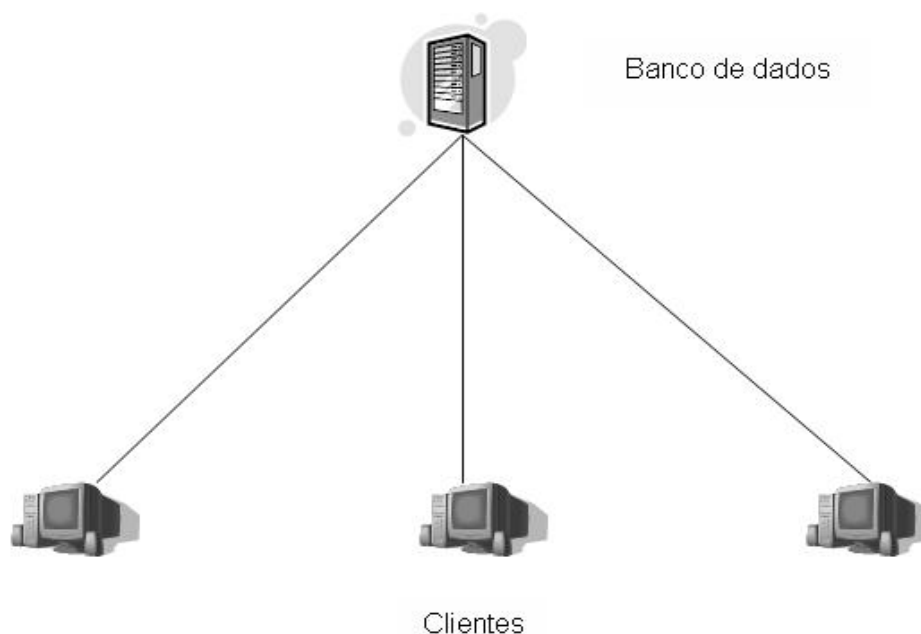
4.5.1 Cliente Servidor

Nessa arquitetura existe um servidor, que representa um processo e fornece serviços a outros processos, que seriam os clientes. Os clientes já conhecem a

identidade do servidor, ou podem descobri-la através de outro servidor, e acessá-la por chamada de procedimento remoto, porém, o servidor normalmente não sabe de antemão quais e quantos clientes irão acessá-lo em tempo de execução (Garlan, Shaw, 1994, p. 14).

A arquitetura cliente servidor, também chamada de duas camadas, é composta por duas partes distintas, uma executada no ambiente do cliente e outra no servidor, com objetivo de dividir o processamento entre as máquinas. A camada do cliente utiliza uma interface para interação entre o usuário e o sistema, onde é inserido e manipulado informações e enviadas ao servidor, através de uma conexão com o banco de dados que deve ser configurada na aplicação. Já a camada do servidor é responsável por processar essas informações recebidas e retornar o resultado para a camada do cliente. De forma resumida, essa arquitetura de software utiliza um computador central para armazenar os dados e receber as requisições da camada do cliente, onde processa e retorna o resultado, conforme demonstra a figura 03 (Jones, 2007).

Figura 3 - Arquitetura cliente servidor



Fonte: Jones (2007)

4.5.2 Multicamadas

Segundo Jones (2007) o método multicamadas é conhecido como a evolução do cliente servidor, onde define como princípio básico que o cliente nunca se comunicará diretamente com o servidor de banco de dados, mas sim com uma camada intermediária, e essa então se comunica com o banco de dados. Essas três camadas podem ser chamadas de apresentação, regra de negócios e banco de dados.

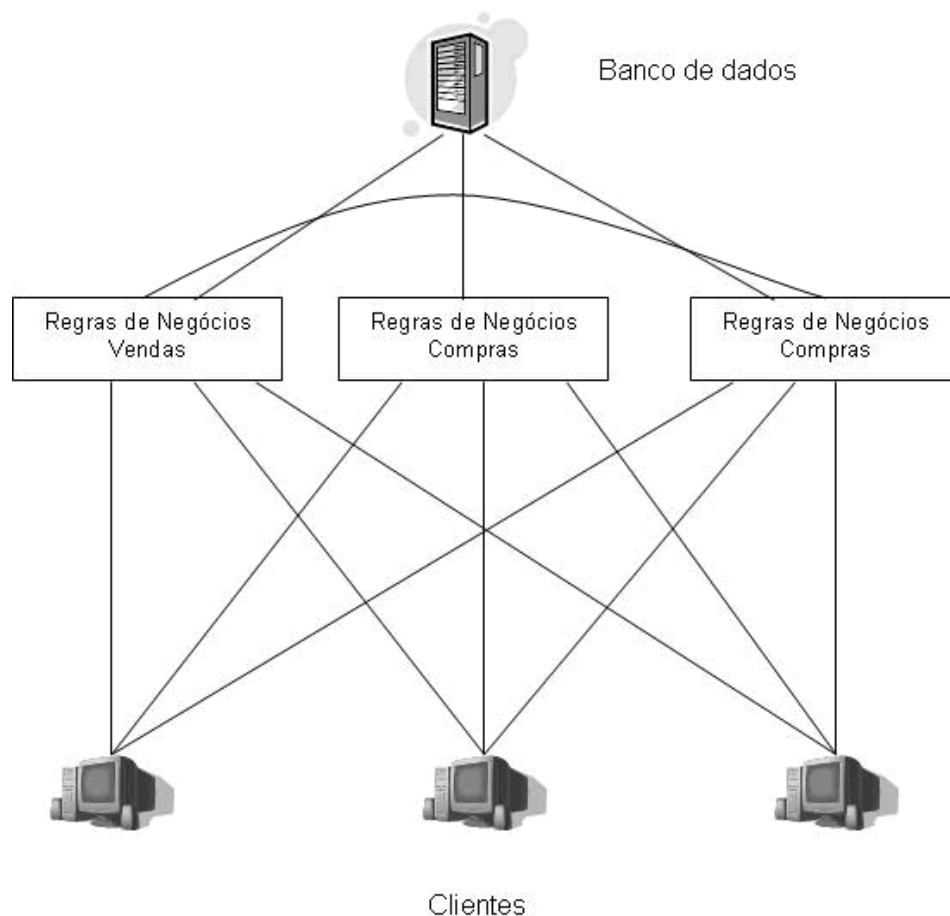
A camada de apresentação é responsável por realizar a interação entre o usuário e o sistema. É uma camada leve, basicamente executa tratamentos de tela e campos e acessa somente a segunda camada de regras de negócios. É conhecida também como camada do cliente, regras de interface do usuário ou camada de interface.

A camada de regra de negócio é um meio campo entre a apresentação e o banco de dados, Após ser chamada pelo cliente, podendo ser várias aplicações cliente, ele faz a requisição para o banco de dados já com todos os tratamentos. Geralmente é uma máquina dedicada com bons recursos de hardware, e é nele que ficam armazenadas as regras de negócios, realizando todo seu tratamento e processamento.

Banco de dados é a última divisão do modelo e é onde fica localizado o sistema gerenciador de banco de dados.

A figura 04 representa a comunicação em uma arquitetura de software multicamadas.

Figura 4 - Arquitetura multicamadas



Fonte: Jones (2007)

Arquitetura de multicamadas ou três camadas é comum na construção de sistemas de informação corporativos. Consiste em uma arquitetura distribuída, onde conta com três camadas, cliente (interface gráfica), aplicação (lógica de negócio) e banco de dados. A camada de interface executa na máquina dos clientes. A camada de negócio executa em um servidor, muitas vezes chamado de servidor de aplicação. E, por fim, temos o banco de dados (Valente, 2021, cap.7).

4.5.3 SOA (Arquitetura orientada a serviços)

SOA tem como princípio tornar componentes de software reutilizáveis por meio de interfaces de serviços. Essas interfaces utilizam padrões de comunicação comuns, o que possibilita acrescentá-las em novos aplicativos sem a necessidade

de sempre fazer uma integração, também fornecem acoplamento fraco, ou seja, podem ser solicitadas com pouco ou nenhum conhecimento sobre a integração que está sendo feita nos bastidores. No SOA cada serviço é responsável por realizar uma função de negócios completa, como por exemplo, verificar crédito de um cliente, calcular um pagamento mensal de um empréstimo ou processar uma proposta de hipoteca. Estes serviços se comunicam através de protocolos como SOAP/HTTP ou JSON/HTTP, para enviar solicitações ou alterar dados (IBM, 2019).

“A arquitetura orientada a serviços (SOA) é uma evolução da computação distribuída baseada no paradigma de design de solicitação/resposta para aplicativos síncronos e assíncronos. A lógica de negócios de um aplicativo ou funções individuais são modularizadas e apresentadas como serviços para aplicativos de consumidor/cliente. A chave para esses serviços é sua natureza fracamente acoplada; ou seja, a interface de serviço é independente da implementação (Kodali, 2015).”

Segundo Abrams e Schulte (2008), um aplicativo SOA adota cinco princípios, sendo eles:

1. O sistema deve ser modular, ou seja, um conjunto de pequenos componentes que funcionam juntos e resolvem problemas complexos;
2. Os módulos devem ser distribuídos, o que possibilita a comunicação entre diferentes computadores através de uma rede, em tempo de execução;
3. As interfaces de módulo devem ser documentadas e definidas de forma clara, o que auxilia na localização e reutilização dessa interface por outro desenvolvedor;
4. Um módulo que fornece um serviço pode ser substituído por outro módulo que fornece o mesmo serviço e interface, pois a interface projetada é separada do módulo. Isso possibilita a manutenção e aprimoramentos incrementais, característica de acoplamento fraco;
5. Os módulos do provedor de serviços devem ser compartilháveis, projetados e implantados de uma forma a ser chamados posteriormente por outros módulos com objetivo relacionado.

4.5.4 Microsserviços

A arquitetura de microsserviços é um conceito com escopo definido pelo aplicativo. Ela possibilita que o aplicativo seja construído em pequenos pedaços que podem ser alterados, escalonados e administrados de forma independente (IBM, 2019).

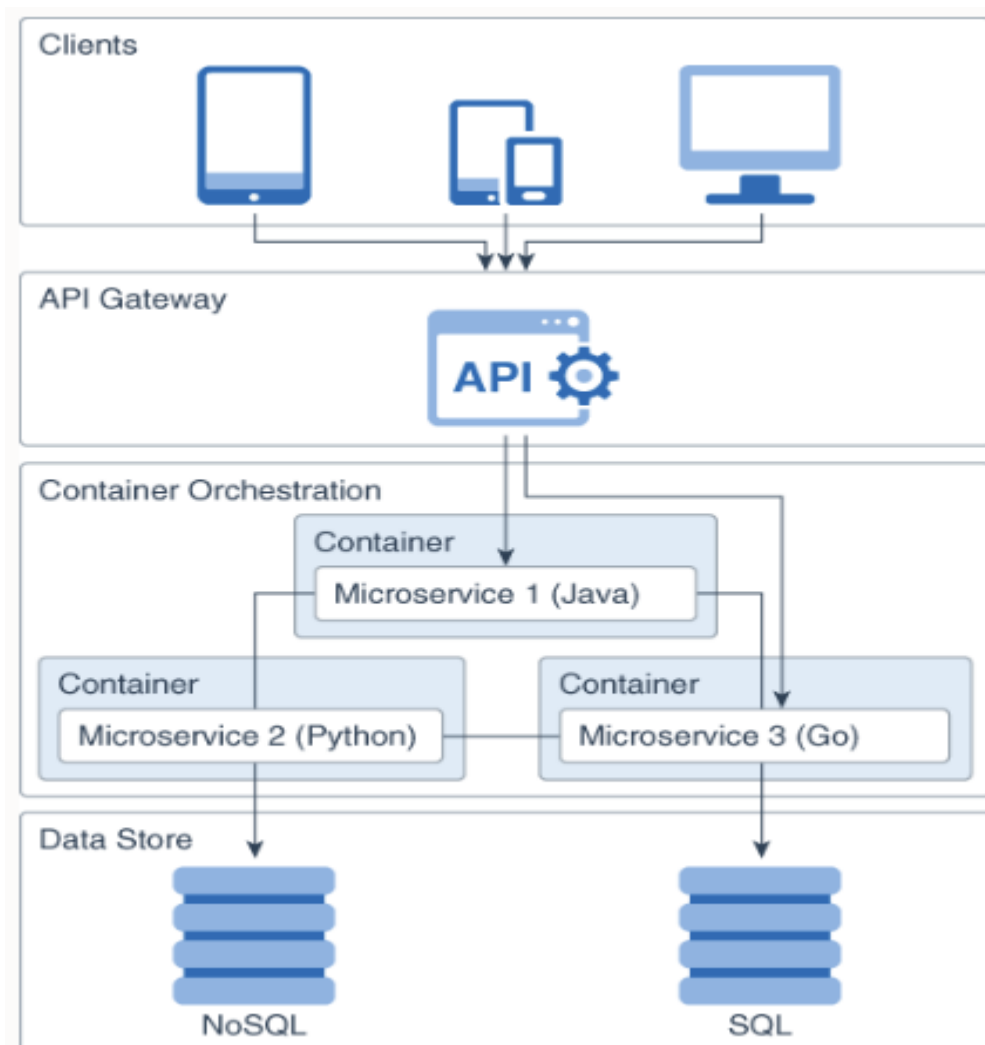
Segundo Fowler (2017), o micro serviço basicamente é uma pequena aplicação que executa uma única tarefa com eficiência. É um pequeno componente desenvolvido e implantado de forma individual e que pode ser facilmente substituído. É uma pequena parte de um sistema maior, sendo executado junto com outros microsserviços, que tem como finalidade executar tarefas que seriam tratadas por uma grande aplicação autônoma.

A arquitetura de microsserviços possibilita uma aplicação ser facilmente escalada tanto horizontalmente, quanto verticalmente, a produtividade e velocidade do desenvolvedor aumentam drasticamente, e antigas tecnologias podem ser substituídas facilmente. Esse formato vem sendo adotado por grandes empresas como Amazon, Twitter, Netflix, eBay e Uber, empresas que executam aplicações em milhares, até mesmo em centenas de milhares de servidores, cuja necessidade foi pelo motivo que essas aplicações eram monolíticas e passaram por dificuldades de escalabilidade (Fowler, 2017).

Segundo a Oracle (2021), a arquitetura de microsserviços deve ser utilizada se o objetivo é desenvolver uma aplicação com possibilidade de suportar diferentes linguagens de programação, ser facilmente escalável, fácil de manter e implantar, altamente disponível e que minimize falhas.

O diagrama a seguir mostra a arquitetura de microsserviços de um aplicativo:

Figura 5 - Arquitetura de microsserviços



Fonte: Oracle, 2021

4.6 Observabilidade

O termo observabilidade foi criado em 1960 pelo engenheiro Rudolf E. Kálmán, quando publicou um artigo caracterizando a observabilidade para descrever sistemas de controle matemático. Na teoria do controle, a observabilidade é definida como uma medida de quão bem os estados internos de um sistema podem ser inferidos a partir do conhecimento de suas saídas externas (Majors, Jones e Miranda, 2022)

Em sistemas de software a definição de observabilidade é definida como uma

medida de quão bem você pode entender e explicar cada estado em que seu sistema entra. Não é sobre tipos de dados ou entradas, nem equações matemáticas, é sobre como as pessoas interagem e tentam entender seus sistemas complexos. Ou seja, requer o conhecimento da interação entre pessoas e tecnologia, para entender como esses sistemas complexos funcionam juntos (Majors, Jones e Miranda, 2022).

A observabilidade demonstra o quão bem é possível entender um software, ou o que está acontecendo nele, através de ferramentas que frequentemente vão coletando métricas, logs ou rastreamentos. As soluções de observabilidade permitem a coleta e análise de dados de aplicativos e infraestrutura para melhor entendimento do estado interno do software, e com isso ser alertado, com objetivo de resolver problemas de disponibilidade e desempenho do aplicativo para melhorar a experiência de usabilidade do usuário final (Amazon, 2022).

Segundo Carey (2021), o termo de observabilidade na engenharia de software ganhou maior relevância por volta de 2018, por uma evolução natural das práticas de monitoramento do software. Ao utilizar a observabilidade os desenvolvedores tiveram a possibilidade de obter dados em tempo real de como o software se desempenhava e onde estava ocorrendo os problemas, apenas reunindo saídas brutas de métricas, eventos, logs e rastreamentos. Carey reforça que a utilização de observabilidade nunca foi tão importante quanto agora, pelo fato da mudança da utilização de sistemas de software distribuídos por meio de microsserviços e containers.

4.7 Monitoramento

Segundo Tebaldi (2020) monitoramento possibilita a observabilidade e entendimento do estado do sistema, através de um conjunto predefinido de métricas e registros. O monitoramento é realizado para detectar falhas conhecidas. Ele é crucial para analisar tendências a longo prazo, construir painéis com dados coletados e alertar, possibilitando entender como os aplicativos funcionam, como estão crescendo e sendo utilizados.

Para Majors, Jones e Miranda (2022) os sistemas de monitoramento tem função de coletar, agrupar e analisar métricas, onde filtram por padrões já

conhecidos que indicam que algo está ocorrendo com tendências preocupantes. Como exemplo pode ser citado o processamento de um servidor, supondo que a utilização normal dele seja os 90%, pode ser configurado um alerta para notificar quando a porcentagem ultrapassar esse número, e caso baixe dos 90%, o sistema de monitoramento pode ser configurado para declarar que a condição ideal foi recuperada.

Quando o monitoramento é combinado com alertas há a possibilidade de identificar o que está com problema, ou está prestes a ter um problema. Com esses dados é possível identificar possíveis falhas facilmente, que podem ser resolvidas antes que os usuários sejam afetados, como por exemplo o armazenamento de um servidor, caso esteja ficando sem espaço em disco. O monitoramento apenas não é eficaz quando existem incógnitas desconhecidas ou falhas imprevistas, o que torna necessário a capacidade de identificar com precisão um conjunto principal de métricas que indica a integridade de um sistema ou um conjunto de modos de falha de um sistema (Tebaldi, 2020).

4.8 Monitoramento x Observabilidade

O monitoramento trata-se de um modelo reativo, mais adequado para detectar problemas conhecidos e padrões previamente identificados. A observabilidade é mais adequada para entender sistemas mais complexos e descobrir explicitamente a fonte de qualquer problema, ao longo de qualquer dimensão ou combinação de dimensões, sem precisar prever onde e como o problema está ocorrendo (Majors, Jones e Miranda, 2022).

Segundo Tebaldi (2020), a observabilidade é como um superconjunto de monitoramento no sentido de que se um sistema é observável, ele pode ser monitorado. A capacidade de observação ajuda o monitoramento com fornecimento de informações, ou seja, o monitoramento é feito após um sistema ser observável, sem nenhum nível de observabilidade não seria possível o monitoramento.

“A observabilidade é uma *propriedade de um sistema* .

Você pode monitorar um sistema usando vários instrumentos, mas se o sistema não exteriorizar seu estado o suficiente para que você possa descobrir o que

realmente está acontecendo lá, então você está preso (Mueller, 2018).”

5. METODOLOGIA DA PESQUISA

A metodologia deste trabalho será aplicar observabilidade e monitoramento a um software. Para atingir o objetivo, será necessário um conjunto entre estudo sobre as principais ferramentas de código aberto de observabilidade e monitoramento de software, buscando escolher a que melhor resolve os desafios que foram destacados neste trabalho, e então a integração destas ferramentas a um grupo de microsserviços.

Escolhido as ferramentas, em seguida será criado alguns microsserviços (APIs) de serviços comuns, como de clientes, fornecedores, produtos, que terão a função de inserir, editar, atualizar e excluir registros, juntamente com o seu banco de dados, que armazenará somente aquilo que é pertinente do próprio microsserviço. Os microsserviços serão desenvolvidos com a tecnologia NodeJS, que permite a execução de códigos javascript fora do navegador, e orquestrados com o Docker, que é uma ferramenta de código aberto que possibilita o empacotamento de uma aplicação dentro de um container com virtualização, ou seja, será possível testar e implementar aplicações em um ambiente separado da máquina original.

Com os microsserviços rodando, será integrado às ferramentas de observabilidade e monitoramento com os microsserviços. Após identificado que o monitoramento e observabilidade estão coletando dados e métricas dos microsserviços, será configurado alertas específicos, como de alertar quando o processamento do servidor está em 90%, ou então o banco de dados já atingiu 80% do seu espaço, e também será feito inserções, edições, atualizações e exclusões de registros, forçar erros, utilizar grande parte do processamento do servidor, derrubar alguns microsserviços, enfim, o software irá passar por uma simulação real do uso de suas ferramentas no dia a dia e comportamentos que poderiam surgir na utilização do mesmo.

Todos esses testes feitos serão apresentados em interfaces gráficas intuitivas, onde será possível identificar os pontos críticos, anomalias e acompanhar em tempo real como o software está se comportando durante a sua utilização.

6. CRONOGRAMA

O cronograma é referente ao tempo que será reservado e desempenhado para o desenvolvimento do trabalho, iniciando desde a pesquisa do tema até o trabalho pronto com os objetivos esperados.

Atividades	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov
Pesquisa do tema	X	X								
Pesquisa bibliográfica		X	X	X						
Coleta de Dados (se for o caso)								X	X	
Apresentação e discussão dos dados									X	X
Elaboração do trabalho		X	X	X	X	X	X	X	X	

7. ANÁLISE DOS REQUISITOS

Nessa etapa são abordados todos os passos necessários para realizar com sucesso o desenvolvimento da solução proposta.

Iniciando-se pela análise dos requisitos, serão utilizadas ferramentas para auxiliar no levantamento desses requisitos. Como a solução proposta não vem de requisitos de um cliente, o levantamento dos requisitos neste caso pode ser resumido no planejamento de tudo que vai compor o projeto, antes de efetivamente iniciar o desenvolvimento. Essa etapa é importante pois ficará claro quais tecnologias utilizar, estruturação do banco de dados, entre outros detalhes que deverão surgir em tempo de desenvolvimento.

7.1 UML

UML é uma linguagem de notação para uso de projeto de sistemas, resume-se em uma forma de ilustrar a estrutura e comunicação do software. Para realizar a ilustração, são utilizados diagramas, cada diagrama é composto por elementos e possuem relação entre si (Ventura, 2019).

Um grande problema no desenvolvimento de projetos é a má comunicação entre os envolvidos, o que torna o UML de grande utilidade, pois deixa o escopo do projeto mais claro, e o mais importante, centralizado numa única visão, utilizando uma linguagem que todos entendem.

Existem diversos modelos de diagramas dentro do UML, cada um atendendo uma necessidade. Esses modelos são divididos entre dois grandes grupos: diagramas estruturais e diagramas comportamentais.

“Diagramas estruturais devem ser utilizados para especificar detalhes da estrutura do sistema (parte estática), por exemplo: classes, métodos, interfaces, namespaces, serviços, como componentes devem ser instalados, como deve ser a arquitetura do sistema etc. Diagramas comportamentais devem ser utilizados para especificar detalhes do comportamento do sistema (parte dinâmica), por exemplo: como as funcionalidades devem funcionar, como um processo de negócio deve ser tratado pelo sistema, como componentes estruturais trocam mensagens e como respondem às chamadas etc. (Ventura, 2019)”

7.1.1 Diagrama Entidade-Relacionamento

Os diagramas de classes são fundamentais para a modelagem de objetos e modelam a estrutura estática de um sistema. No estágio de análise, um diagrama de classe pode ajudá-lo a compreender os requisitos do domínio do problema e a identificar seus componentes. (IBM, 2021)

Neste projeto especificamente foi utilizado apenas o diagrama de classes para montar toda nossa estrutura do banco, quais as tabelas, seus campos, o tipo dos seus campos e neste caso sem relações entre tabelas, pois não é o foco do projeto e não irá afetar no monitoramento, se tornando uma estrutura de banco de dados bem simples.

7.2 GitHub

Antes de iniciar todo o desenvolvimento do projeto é importante organizar a área de trabalho, para isso foi criada a estrutura de pastas necessárias, e tudo que ia sendo feito, foi sendo subido para a nuvem, para em caso de eventualidades, nada ser perdido, e para atender essa necessidade foi utilizado o GitHub.

O GitHub é uma ferramenta com muita popularidade mundialmente, e basicamente essencial para engenheiros de software. Atualmente ela acomoda mais de 25 milhões de usuários. Resumidamente o GitHub é um serviço baseado na nuvem que hospeda um sistema de controle de versão. O sistema de controle de versão ajuda a acompanhar todas as alterações feitas, quando e por quem foram feitas, além de ser possível restaurar o código removido ou editado (Longen, 2022).

Sendo assim, já foram criados os repositórios no GitHub, possibilitando o desenvolvimento de qualquer máquina, sempre com a versão atualizada e mantendo um histórico de tudo que havia sido feito.

8. DESENVOLVIMENTO DO BACK-END

O desenvolvimento back-end é focado principalmente em como um site funciona, concentrado na lógica e funcionalidade do sistema, mantém o sistema alimentado de dados, mas não é vista diretamente pelos usuários. As tecnologias utilizadas no back-end são uma combinação de servidores, aplicativos e banco de dados. Os desenvolvedores do back-end são responsáveis geralmente pela criação de APIs, comunicação com banco de dados, arquitetura de dados, criação de bibliotecas, entre outros. (Krystal, 2022)

O back-end de uma aplicação pode ser resumida como os bastidores de um show, mantém as coisas funcionando, mas não é vista.

8.1 Banco de Dados

Baseado na estrutura criada no diagrama entidade-relacionamento foram criadas as tabelas no banco de dados. O banco de dados é utilizado em conjunto com as APIs, onde acontece a comunicação por parte da API que consulta e manipula essa coleção dados (inserção, edição e exclusão)

O SGBD escolhido foi o MySQL, por tratar-se de um banco gratuito, de código de fonte aberto, leve, rápido, seguro e prático, já utilizado por muitos desenvolvedores mundialmente, e geralmente utilizado em conjunto com aplicações web, que se encaixa perfeitamente com o escopo desse projeto.

8.2 APIs

As APIs são um conjunto de padrões que fazem parte de uma interface que tem como objetivo facilitar a criação de plataformas pelos desenvolvedores. A sigla API deriva da expressão inglesa Application Programming, que traduzida para português, significa interface de programação de aplicação. Basicamente é um conjunto de normas que possibilita a comunicação entre plataformas através de diferentes padrões e protocolos, além de garantirem uma segurança maior para a aplicação, pois são capazes de filtrar e bloquear permissões a dados de software e hardware que a aplicação pode ter acesso (Fabro, 2020).

Pode ser citado alguns exemplos de APIs que são utilizados no cotidiano, por exemplo, quando uma pessoa está navegando em um e-commerce, adiciona um produto no seu carrinho e calcula o valor de frete do correios para a entrega, apenas digitando o seu CEP, por trás existe uma API do correios que é requisitada neste momento, onde recebe o CEP informado pelo usuário e retorna o valor daquela entrega automaticamente, sem precisar que todos os CEP de todos os municípios estejam cadastrados no banco de dados da aplicação e-commerce. Se não existisse essa API, cada aplicação teria que fazer um desenvolvimento à parte para calcular o frete das entregas.

8.3 JavaScript

De forma breve, o JavaScript é uma linguagem de programação de alto nível, criada em 1996 pelo programador Brendan Eich. Originalmente o JavaScript foi criado para funcionar em um navegador em específico, chamado Netscape Navigator, e tinha como objetivo facilitar processos dentro da página web, tornando a programação das animações e alertas mais simples. Com o passar do tempo essa linguagem se popularizou muito, o que veio junto com grandes evoluções, se tornando mais versátil e completa já pode ser utilizada no desenvolvimento não só de aplicações de navegadores, mas também desktop e mobile. (Estrella, 2022)

A escolha da utilização do JavaScript no projeto é decorrente de ser uma linguagem com uma comunidade enorme, onde existem muitas funções e bibliotecas criadas que podem ser utilizadas, o que facilita o desenvolvimento, além de apresentar uma curva de aprendizado baixa, ser compatível com várias plataformas e navegadores, ser rápida e leve e tornar os sites mais interativos. Além disso, como já dito, a linguagem JavaScript se tornou muito flexível, o que o torna possível utilizar tanto no server-side (back-end) como no lado client-side (front-end), sendo necessário no back-end um compilador que interprete a linguagem, já no front-end nada é necessário, pois os navegadores o interpretam como HTML.

8.4 Node.js

O Node.js pode ser definido como um ambiente de execução da linguagem JavaScript no server-side. O que possibilita criar aplicações JavaScript e subir essas aplicações sem depender de um browser para a execução, diferente de como foi criado o JavaScript, onde era exclusivo para os browsers.

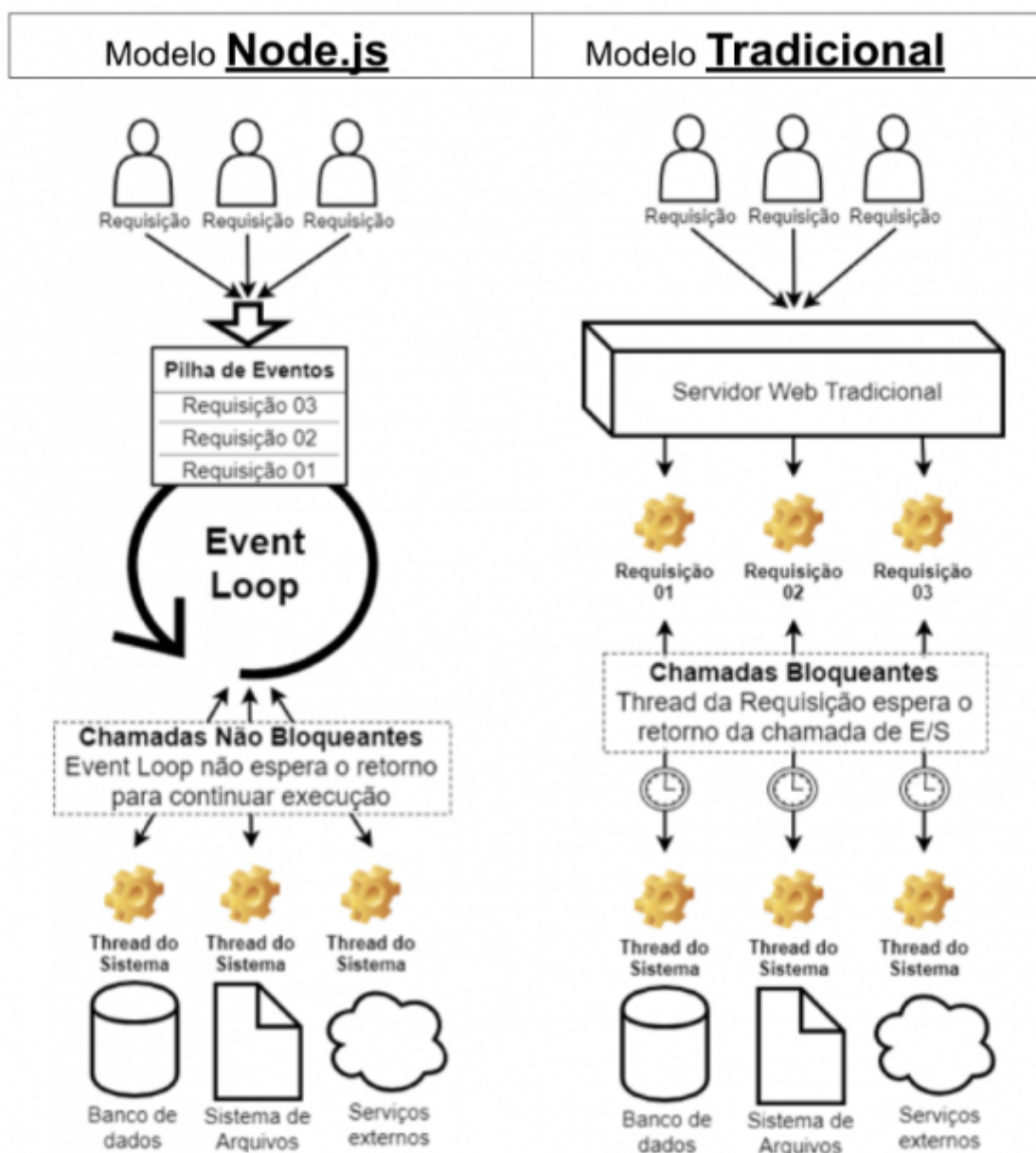
A utilização do JavaScript no server-side com o Node.js é um tanto quanto recente (em torno de 10 anos), porém, já é utilizada por grandes empresas do mercado de tecnologia, como Netflix, Uber e LinkedIn. O principal motivo de adoção é a alta capacidade de escala, flexibilidade e baixo custo. (Lenon, 2018)

A principal característica que faz o Node.js se diferenciar das demais tecnologias bastante conhecidas como PHP, Java e C#, é o fato da sua execução ser single-thread. Isso significa que apenas uma thread é responsável por executar o código JavaScript da aplicação, enquanto nas outras linguagens são várias, sempre

criando uma a cada nova a cada requisição, o que acaba demandando recursos computacionais para criação destas threads. Apesar de ser *single-threaded*, o node consegue tratar requisições concorrentes para não ocasionar uma espera enorme de tempo da primeira para a última requisição. Enquanto o servidor tradicional utiliza o sistema *multi-thread* para tratar requisições concorrentes, o Node.js consegue o mesmo efeito através de chamadas de entrada e saída não-bloqueantes. Isso significa que estas operações de entradas e saídas (ex: acesso a banco de dados e leitura de arquivos do sistema) são assíncronas e não bloqueiam a *thread*. Diferentemente dos servidores tradicionais, a *thread* não fica esperando que essas operações sejam concluídas para continuar sua execução. (Lenon, 2018)

A figura a seguir apresenta a diferença entre o funcionamento de um servidor web tradicional (PHP, Java, C#) e o funcionamento do Node.js:

Figura 6 - Modelo Node.js vs Tradicional



Fonte: Lenon, 2018

8.5 Express.js

O Express.js foi lançado em 2010, trata-se de uma framework de código aberto, ou seja, sem custo de uso, que tem como objetivo otimizar a construção de aplicações web e APIs. Se tornou um dos frameworks mais populares da comunidade de tecnologia e utiliza o Node.js para execução do JavaScript no back-end (Andrade, 2020).

Muito popular nas comunidades de tecnologia e também nas grandes empresas, o Express.js facilita a criação de aplicações que usam o conjunto de

Node.js com JavaScript no back-end, tendo como principais características um sistema de rotas completo, tratamento de exceções dentro da aplicação, gerenciamento de diferentes requisições HTTP e rápida criação de aplicações, já sendo utilizado por grande empresas como Fox Sports, PayPal, IBM, Uber, entre outras (Andrade, 2020).

8.6 Documentação das APIs

Pensando em uma boa organização das APIs criadas, é essencial a documentação do que foi desenvolvido. Como já dito, as APIs podem ser reutilizadas por outras plataformas, e para que isso seja possível, é inevitável ao menos uma descrição do que se trata determinada API, ou seja, qual sua função, e como deve ser utilizada, caso contrário, pode comprometer sua adoção em outras aplicações.

Para organizar e expor de forma clara o catálogo de APIs disponíveis e como utilizá-las, foi utilizado uma tecnologia chamada Swagger.

8.6.1 Swagger

Swagger é uma ferramenta open source que pode ser utilizada em várias linguagens de programação, com objetivo de auxiliar a criar a documentação dos serviços fornecidos pelas APIs. O framework do Swagger implementa a especificação OpenAPI, que é uma linguagem para descrição de contratos de APIs REST, onde define-se um formato JSON com campos padronizados, através de um JSON Schema, para que seja descrito as descrições, modelos de dados, recursos, métodos HTTP aceitos, URLs e códigos de retornos esperados (Moura, 2021).

O Swagger auxiliou para documentar as APIs desenvolvidas, além de possibilitar reunir as informações de todas APIs desenvolvidas em um local só, pois através do próprio Swagger, é disponibilizado o Swagger UI, que permite criar uma interface para o usuário.

Na figura a seguir é apresentado um exemplo da criação da documentação dos endpoints de consulta, inserção, edição e exclusão da API de produtos, escrito em linguagem OpenAPI:

Figura 7 - Criação de documentação via Swagger

```
You, 13 seconds ago | 2 authors (Odair Sehn and others)
const productController = require('../controllers/product'); Odair Sehn, la

module.exports = (app) => {
  app.get('/product', productController.getProduct
    /**region Documentação
     /* #swagger.tags = ['Produto']
     #swagger.summary = 'Retorna lista de produtos'
     */
    /**endregion
  )
  app.post('/product', productController.insertProduct
    /**region Documentação
     /* #swagger.tags = ['Produto']
     #swagger.summary = 'Insere produto'
     #swagger.parameters['json'] = {
       in: 'body',
       description: 'Dados para inserir um produto (obrigatório)',
       type: 'json',
       schema: {
         pro_descricao: "Camiseta Algodão",
         pro_tamanho: "M",
         pro_custo: 60,
         pro_valor: "100"
       }
     }
     */
    /**endregion
  )
  app.patch('/product', productController.updateProduct
    /**region Documentação
     /* #swagger.tags = ['Produto']
     #swagger.summary = 'Atualiza informações do produto'
     #swagger.parameters['json'] = {
       in: 'body',
       description: 'Dados para atualizar um produto',
       type: 'json',
       schema: {
         pro_codigo: 1,
         pro_descricao: "Camiseta Algodão",
         pro_tamanho: "M",
         pro_custo: 60,
         pro_valor: "100"
       }
     }
     */
    /**endregion
  )
  app.delete('/product/:pro_codigo', productController.deleteProduct
    /**region Documentação
     /* #swagger.tags = ['Produto']
     #swagger.summary = 'Deleta Produto'
     */
    /**endregion)
}
}
```

Autor: Elaborado pelo autor.

Na figura a seguir, é apresentado o resultado do desenvolvimento dessa documentação já em interface para o usuário (Swagger UI):

Figura 8 - Apresentação das rotas no Swagger UI

The image shows the Swagger UI interface for a resource named 'Produto'. It displays four endpoints:

- GET** /product: Retorna lista de produtos
- POST** /product: Insere produto
- PATCH** /product: Atualiza informações do produto
- DELETE** /product/{pro_codigo}: Deleta Produto

The **POST** endpoint is expanded, showing the following details:

- Parameters:** A table with columns 'Name' and 'Description'. The description is 'Dados para inserir um produto (obrigatório)'. Below the table, there is an 'Example Value | Model' section with a JSON object:

```
{  "pro_descricao": "Teste",  "pro_tamanho": "M",  "pro_custo": 5,  "pro_valor": "12"}
```

. A 'Parameter content type' dropdown is set to 'application/json'.
- Responses:** A table with columns 'Code' and 'Description'. The response codes and descriptions are: 201 (Criado), 400 (Solicitação Inválida), and 500 (Erro Interno do Servidor). A 'Response content type' dropdown is set to 'application/json'.

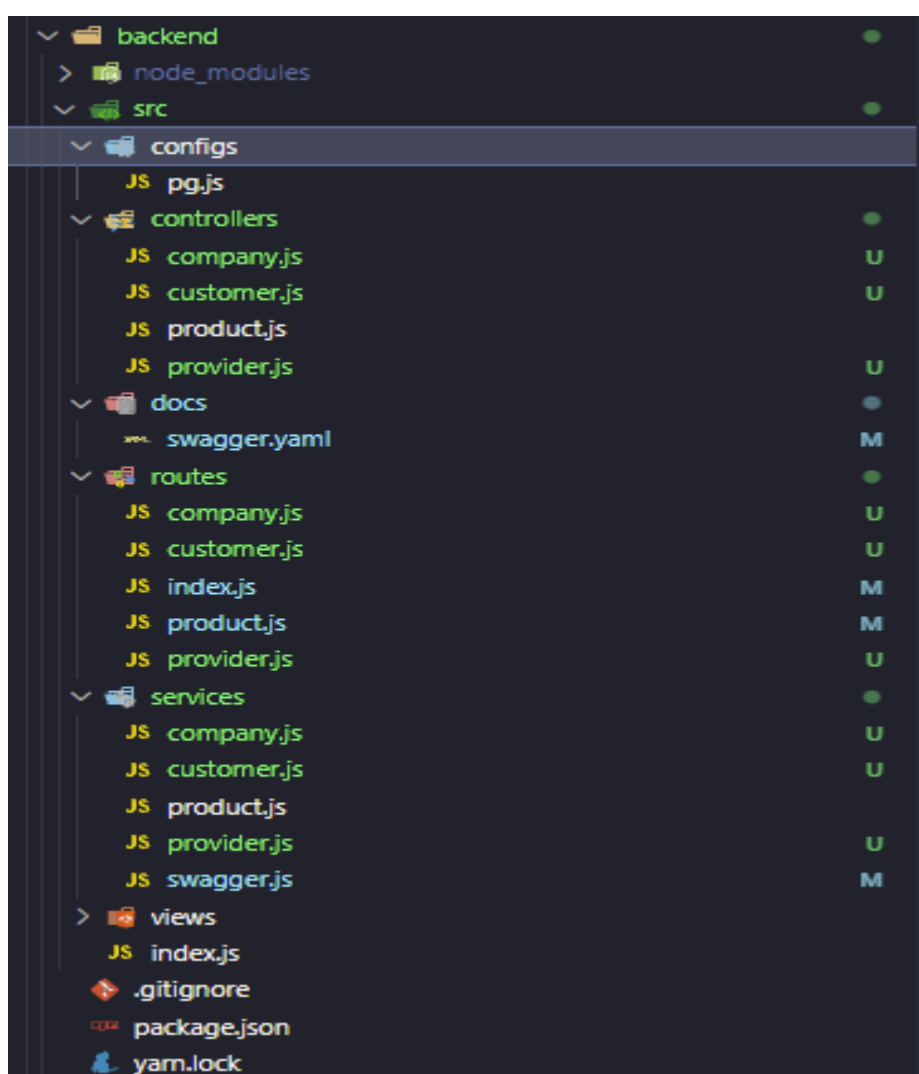
Fonte: Elaborado pelo autor.

8.7 Back-end finalizado

Utilizando boas práticas de programação, organização de pastas, e as demais tecnologias já abordadas, como banco MySQL, JavaScript, Node.js, Express.js e Swagger foram desenvolvidas as APIs necessárias e configurado a conexão delas com o banco de dados, sendo assim, já é possível consultar e manipular os dados do banco de dados através da APIs.

A figura a seguir demonstra toda a estrutura de pastas criadas no projeto, onde na pasta configs foi configurado a conexão com o banco MySQL, pasta docs, routes e views são configurações do Swagger, services responsável pela carga de dados, execução de regra de negócio, formatação dos campos e retorno para quem chama a API e controllers responsável pelo tratamento de parâmetros de entrada e retorno da chamada:

Figura 9 - Estrutura de pastas do projeto back-end



Autor: Elaborado pelo autor.

E nas seguintes figuras, é exemplificado o desenvolvimento de uma API que faz a inserção de um produto na tabela T_PRODUTO no banco de dados, onde a primeira imagem indica o services responsável pela carga de dados, conexão com o

banco e regra de negócio, seguido do controller que trata os parâmetros de entrada e retorno da chamada:

Figura 10 - Exemplo de API de inserção na pasta Services

```

Odair Sehn, 4 weeks ago | 1 author (Odair Sehn)
const db = require('../configs/pg');

const sql_insert =
  `insert into t_produto (pro_codigo, pro_descricao, pro_tamanho, pro_custo, pro_valor)
   values (nextval('gn_produto'), $1, $2, $3, $4)`

async function insertProduct (params) {
  const {pro_descricao, pro_tamanho, pro_custo, pro_valor} = params
  const result = await db.query(sql_insert, [pro_descricao, pro_tamanho, pro_custo, pro_valor])
  if (result.rowCount > 0) {
    return result.rows.msg = 'Inserção realizada com sucesso'
  }
}

```

Autor: Elaborado pelo autor.

Figura 11 - Exemplo de API de inserção na pasta Controllers

```

const productService = require('../services/product');

const insertProduct = async (req, res) => {
  try {
    let params = req.body
    if (params.pro_descricao && params.pro_tamanho) {
      const rows = await productService.insertProduct(params)
      res.status(201).json(rows)
    } else {
      res.status(400).json({
        tpe: 'API',
        message: 'MISSING_BODY_FIELDS',
        detail: 'Obrigatório informar os campos pro_descricao e pro_tamanho'
      })
    }
  } catch(err){
    res.status(500).json(err)
  }
}

```

Autor: Elaborado pelo autor.

9. DESENVOLVIMENTO DO FRONT-END

Também conhecido como “lado do cliente”, o front-end é responsável por toda interação da aplicação com o usuário e interface gráfica, ou seja, o design, conteúdo, comportamento, desempenho e capacidade de resposta de uma aplicação (Machado, 2021).

Usando o mesmo exemplo do back-end, em um show, onde o back-end seria os bastidores, o front-end seria toda a apresentação. No front-end é desenvolvida toda a parte de interação com o usuário, interface gráfica, navegação da aplicação, responsividade para diferentes dispositivos, ou seja, um conjunto de criação de elementos visuais com funcionalidades.

No front-end foi desenvolvido interfaces gráficas para requisitar as APIs já criadas anteriormente, as habilidades necessárias para o desenvolvimento de aplicações web são focadas em três principais linguagens, que são elas:

1. HTML: Tem o significado de linguagem de marcação de hipertexto, é um componente básico de aplicações web. Ele descreve a estrutura através de forma semântica, utilizando tags para renderizar elementos como botões, textos, inputs, links, imagens, entre outros. Sem o HTML, o navegador não saberia exibir textos como elementos ou carregar ou outros conteúdos.
2. CSS: Cascading Style Sheet, ou traduzido, Folha de Estilo em Cascatas, é utilizado em conjunto com o HTML para dar forma ao documento criado. Com ele é possível estilizar os elementos, definindo diferentes fontes, tamanho das fontes, espaçamentos, cores, bordas, posicionamento dos elementos e outros elementos visuais.
3. JavaScript: Diferente do HTML e CSS, é uma linguagem de programação. No front-end ele é responsável por dar "vida" a aplicação, utilizando em conjunto com o HTML e CSS ele torna os elementos mais dinâmicos, ou seja, quando o usuário executa alguma ação na aplicação vai resultar em algum comportamento, neste momento, é acionado o JavaScript, pois é ele quem permite execução de scripts na nossa aplicação. Com ele também conseguimos fazer as funções que fazem as requisições para as APIs.

9.1 Vue.js

O Vue.js é um framework Javascript open source bastante conhecido pela sua reatividade, é um dos mais populares por conta dos fatores como flexibilidade, escalabilidade e baixa curva de aprendizado. Usado para construir SPA (Single Page

Applications) e interfaces de usuário, tornou-se uma excelente opção, também, pelo fato de ter componentes reutilizáveis e proporcionar o desenvolvimento ágil (Pinheiro, 2021).

Trata-se de um facilitador para o front-end, proporcionando mais agilidade, escalabilidade e melhor experiência do usuário, contando com diversos plugins e padrões. Sua estrutura não afeta o desenvolvimento de componentes, que ainda utilizam como base HTML, CSS e JavaScript, ele apenas traz uma melhor arquitetura, com sintaxes padrões para criação de telas e diversas diretivas.

9.2 Quasar Framework

Nas empresas de tecnologia é normal as equipes de desenvolvimento de aplicações no front-end utilizarem algum framework de componentes UI, isto é, utilizar um framework que já conta com diversos componentes visuais, funcionalidades e plugins já criados e documentados, sendo facilmente utilizáveis, criando uma identidade única, com maior agilidade nas entregas.

O Quasar é uma das várias frameworks que existem no mercado para suprir essa necessidade, ele conta com muitos componentes e plugins fáceis de usar, sem ter nenhum custo, preparado para utilizar com Vue.js. Os componentes já estilizados e bem documentados como devem ser implementados resultam numa agilidade muito grande para o desenvolvimento do front-end, além de auxiliar também no layout padrão das páginas e posicionamento dos elementos.

9.3 Vuex

Em aplicações desenvolvidas em Vue.js é comum termos comunicações entre componentes pais e filhos, e em alguns casos, componentes que utilizam os mesmos dados entre eles. Caso a aplicação cresça, o gerenciamento para manter esses componentes atualizados com os dados mais recentes pode se tornar uma dor de cabeça, ou se tornará complexo e demorado ficar atualizando dados entre vários componentes cada vez que existir uma ação em um deles, pensando em resolver este problema o Vuex foi criado.

O Vuex é um padrão de gerenciamento de estado para o Vue.js. Serve como um centralizador de informações onde todos os componentes da aplicação têm

acesso, possibilitando que o estado deles só possam ser alterados de forma previsível, através de regras criadas pelo mesmo. O Vuex extrai o estado compartilhado dos componentes e o gerencia em um singleton global. Com isso, a árvore de componentes se torna uma grande “view”, e qualquer componente pode acessar o estado ou acionar ações, não importando onde elas estejam na árvore (Monteiro, 2019).

Portanto, quando um componente altera seu estado, é alterada aquela propriedade de forma global, e qualquer outro componente que compartilha aquela propriedade será atualizado também.

9.4 Axios

Para ser possível fazer requisições HTTP, foi utilizado o Axios, que trata-se justamente de uma biblioteca para comunicação via protocolo HTTP e é compatível com Node.js, possibilitando fazer as requisições para as APIs já desenvolvidas.

O Axios no lado do servidor usa requisição HTTP, no lado do cliente (navegador) utiliza XMLHttpRequests. Algumas características importantes dele é que suporta API de promises, intercepta requisições e respostas, cancela requisições, transforma dados de requisições e respostas e automaticamente transforma dados para json.

9.5 Front-end finalizado

Utilizando as tecnologias mencionadas anteriormente, finalizamos o front-end. Isso significa que foi feita uma interface gráfica para interação com o usuário, onde as ações do usuário tomadas na aplicação já estão requisitando nossas APIs e gravando no banco de dados. Para exemplificar o processo e manter uma lógica dos processos anteriores do back-end, foi continuado os exemplos em cima das rotas de produtos, onde é possível consultar, inserir, editar e excluir produtos.

A figura a seguir demonstra a interface gráfica que foi desenvolvida no front-end para as ações relacionadas a produtos.

Figura 12 - Interface gráfica desenvolvida no front-end

Código	Descrição	Tamanho	Custo	Valor de venda
18	Casaco Moletom	M	70.00000	130.00
19	Casaco Moletom	G	70.00000	130.00
20	Camiseta Algodão	P	60.00000	100.00
21	Camiseta Algodão	M	60.00000	100.00
22	Camiseta Algodão	G	60.00000	100.00
23	Camiseta Algodão	GG	60.00000	100.00
24	Jaqueta	M	80.00000	150.00

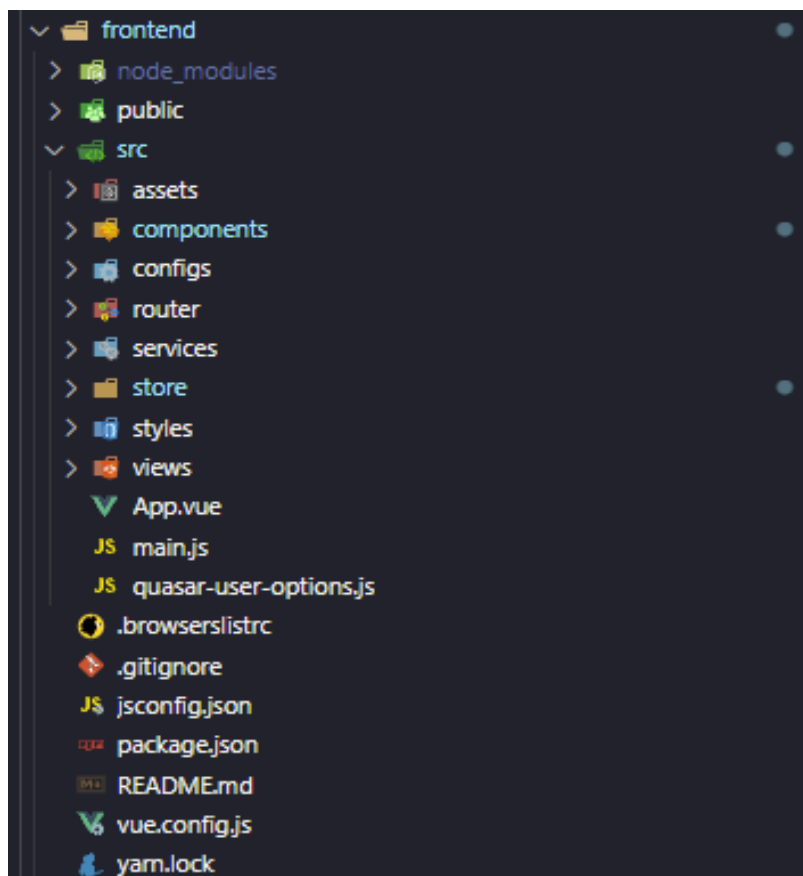
Fonte: Elaborado pelo autor.

A figura a seguir demonstra a estrutura criada no projeto do front-end, onde cada pasta tem sua responsabilidade:

- Assets: Imagens da aplicação
- Components: Todos os componentes desenvolvidos no front-end
- Configs: Configuração do Axios para fazer requisições HTTP
- Router: Navegação de páginas dentro da aplicação
- Services: Requisição para as APIs (chama o Axios)
- Store: Centralização de estado de componentes (Vuex)
- Styles: Classes de css que podem ser utilizadas por qualquer componente
- Views: Páginas principais, junção dos componentes criados
- App: Layout padrão da aplicação, neste caso, menu lateral e superior, pois indiferente da tela navegada, sempre será o mesmo
- main: Criação do app e importações necessárias (Quasar, store, etc)
- quasar-user-options: Opções customizadas do framework Quasar
- jsconfig.json: Arquivo de configuração para auxiliar o editor de texto

- vue.config.js: Arquivo de configuração do vue

Figura 13 - Estrutura de pastas do projeto front-end



Fonte: Elaborado pelo autor.

10. DOCKER

O Docker é uma tecnologia que possibilita virtualizar aplicações no conceito de “containers”, trazendo da web ou de seu repositório interno ou uma imagem completa, essa virtualização também baixa todas as dependências necessárias para executar a aplicação (Neto, 2019)

Trata-se de uma tecnologia open source que facilita a criação e administração de ambientes isolados, possibilitando o empacotamento de uma aplicação ou ambiente dentro de um container.

O container contém um conjunto de processos que são executados a partir de uma imagem que fornece todos os arquivos necessários. A imagem é um modelo de somente leitura que é utilizada para subir um container, também é possível construir nossas próprias imagens e utilizá-las como base para os containers.

Entendendo as possibilidades que o docker disponibiliza, foi utilizado para subir containers para o banco de dados, back-end, front-end e posteriormente para as imagens necessárias de monitoramento e observabilidade.

10.1 Docker Compose

Na utilização no docker, ao construir os containers são necessários informar alguns parâmetros para cada container individualmente, como porta, variáveis de ambiente, entre outros. Quando trata-se de vários containers, o trabalho acaba sendo demorado e repetitivo, pois cada vez que precisar subir aquele serviço, precisará informar novamente aqueles parâmetros.

Para resolver esse problema existe o Docker Compose, que basicamente é uma solução multi-container. Utilizando um arquivo é possível declarar todos os serviços que você quer rodar, juntamente com suas propriedades, porta, nome, variáveis de ambiente, etc. Basicamente toda aquela construção que anteriormente era feita individualmente é declarada em um arquivo, e através desse arquivo é possível executar todos os containers de uma só vez.

Na figura a seguir apresenta o arquivo docker-compose.yaml com os três containers configurados, do banco de dados, back-end e front-end.

11. MONITORAMENTO E OBSERVABILIDADE

O monitoramento e observabilidade na aplicação tem um papel importante para auxiliar na identificação de vários pontos que podem ser críticos, causando uma má experiência do usuário final, como por exemplo, performance da aplicação, serviços indisponíveis, processamento ou memória do servidor em limites preocupantes, erros que não foram previstos em momento de desenvolvimento, entre outros.

Para ser possível aplicar o monitoramento e observabilidade na nossa aplicação foram usados três serviços, APM Server, Kibana e Elasticsearch, todas essas tecnologias trabalham em conjunto para atender o objetivo proposto do projeto, também é importante destacar que seu uso é gratuito, tendo funcionalidades

exclusivas para versão paga. Os serviços foram executados de forma local, porém, é possível subir o serviços na nuvem.

11.1 Elasticsearch

“O Elasticsearch é um mecanismo distribuído de pesquisa e análise de código aberto criado no Apache Lucene e desenvolvido em Java. Ele começou como uma versão escalável da estrutura de pesquisa de código aberto Lucene e, em seguida, adicionou a capacidade de dimensionar índices Lucene horizontalmente. O Elasticsearch permite armazenar, pesquisar e analisar grandes volumes de dados rapidamente e quase em tempo real e fornecer respostas em milissegundos. É capaz de obter respostas de pesquisa rápidas porque, em vez de pesquisar o texto diretamente, ele pesquisa um índice. Ele usa uma estrutura baseada em documentos em vez de tabelas e esquemas e vem com extensas APIs REST para armazenar e pesquisar os dados. Basicamente, você pode pensar no Elasticsearch como um servidor que pode processar solicitações JSON e devolver dados JSON (Abueg, 2020). “

O Elasticsearch quando subido inicializa um cluster, e é nele que serão criados os índices do APM, assim armazenando as métricas coletadas. Para subir o Elasticsearch, como as demais aplicações, será via docker, declarado no docker-compose.

A figura a seguir apresenta o serviço do Elasticsearch declarado no docker-compose.yaml.

Figura 14 - Elasticsearch declarado no docker-compose

```
docker-compose.yml x
E-commerce > docker-compose.yml > version
28
29   elasticsearch:
30     image: docker.elastic.co/elasticsearch/elasticsearch:7.11.1
31     environment:
32       - bootstrap.memory_lock=true
33       - cluster.name=docker-cluster
34       - cluster.routing.allocation.disk.threshold_enabled=false
35       - discovery.type=single-node
36       - ES_JAVA_OPTS=-XX:UseAVX=2 -Xms1g -Xmx1g
37     ulimits:
38       memlock:
39         hard: -1
40         soft: -1
41     volumes:
42       - esdata:/usr/share/elasticsearch/data
43     ports:
44       - 9200:9200
45     networks:
46       - elastic
47     healthcheck:
48       interval: 20s
49       retries: 10
50       test: curl -s http://localhost:9200/_cluster/health | grep -vq "status":"red"
51
```

Fonte: Elaborado pelo autor.

11.2 APM Server

Com o APM Server é possível capturar e analisar todas as transações feitas na aplicação, sendo possível investigar cada camada da requisição, desde a chamada no client-side até o retorno da API requisitada. O detalhe dessas transações resulta em um grande ganho de agilidade para identificar um eventual problema, sendo possível minimizar o tempo de indisponibilidade de determinado serviço, resultando em uma melhor experiência da aplicação para o usuário.

Para implementar o APM Server são necessários três passos: subir o serviço do APM Server, incluir um agente APM como dependência no back-end e configurar o serviço do APM no index.js que faz a inicialização do projeto back-end.

1. Subir o serviço do APM, para isso foi utilizado o docker, já adicionado esse serviço no docker-compose, para quando subir o back-end e o front-end, já subir também o APM Server.

A figura a seguir apresenta o serviço do APM Server declarado no docker-compose.yaml. É possível observar que já foi informada a porta do Elasticsearch que já está configurada, e também do Kibana, que será apresentado em seguida, isso se dá pelo motivo que os três trabalham juntos para o funcionamento.

Figura 15 - APM Server declarado no docker-compose

```
docker-compose.yml X
E-commerce > docker-compose.yml > version
docker-compose.yml (compose-spec:json)
1 version: '2.2'
2 services:
3   apm-server:
4     image: docker.elastic.co/apm/apm-server:7.11.1
5     depends_on:
6       elasticsearch:
7         condition: service_healthy
8       kibana:
9         condition: service_healthy
10    cap_add: ["CHOWN", "DAC_OVERRIDE", "SETGID", "SETUID"]
11    cap_drop: ["ALL"]
12    ports:
13      - 8200:8200
14    networks:
15      - elastic
16    command: >
17      apm-server -e
18        -E apm-server.rum.enabled=true
19        -E setup.kibana.host=kibana:5601
20        -E setup.template.settings.index.number_of_replicas=0
21        -E apm-server.kibana.enabled=true
22        -E apm-server.kibana.host=kibana:5601
23        -E output.elasticsearch.hosts=["elasticsearch:9200"]
24    healthcheck:
25      interval: 10s
26      retries: 12
27      test: curl --write-out 'HTTP %{http_code}' --fail --silent --output /dev/null http://localhost:8200/
```

Fonte: Elaborado pelo autor.

2. O agente APM: Responsável por observar as requisições recebidas do front-end para o back-end, coletar as métricas da requisição e enviar para o Elasticsearch com índice que vem do APM. Para adicionar a dependência no back-end é necessário executar o seguinte comando no terminal: `npm install elastic-apm-node --save`

A figura a seguir apresenta a inclusão da dependência do agente APM no back-end feito em Node.js.

Figura 16 - Instalada dependência do APM no back-end

```
Odair Sehn, last week | 1 author (Odair Sehn)
1  {
2    "name": "api",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "dev": "nodemon ./src/index.js",
8      "test": "echo \\\"Error: no test specified\\\" && exit 1",
9      "doc": "node ./src/services/swagger.js",
10     "prd": "node ./src/index.js"
11   },
12   "keywords": [],
13   "author": "",
14   "license": "ISC",
15   "dependencies": {
16     "cors": "^2.8.5",
17     "elastic-apm-node": "^3.40.0",
18     "express": "^4.17.1",
19     "nodemon": "^2.0.12",
20     "pg": "^8.7.1",
21     "swagger-autogen": "^2.11.2",
22     "swagger-ui-express": "^4.1.6"
23   }
24 }
```

Fonte: Elaborado pelo autor.

3. A configuração no index.js do back-end (que faz a inicialização da aplicação) é feita para configurar a conexão com a imagem do APM que foi buildado via Docker.

A figura a seguir apresenta a configuração do APM no index.js, onde primeiramente foi iniciado o elastic-apm-node através do start, e depois definido o nome do serviço (servicename), que neste caso, é a api(back-end) que está sendo monitorado e qual é a URL (serverUrl) que está rodando o APM Server. É importante destacar que a inicialização do APM deve ser declarado antes de precisar qualquer outro módulo no aplicativo Node.js, que neste caso, usamos o Express.js.

Figura 17 - Configuração do APM no back-end

```
JS index.js M X
E-commerce > backend > src > JS index.js > ...
You 6 minutes ago 12 authors (Odair Sehn and others)
1 const apm = require('elastic-apm-node').start({
2   // Use o nome do serviço de package.json
3   serviceName: 'api',
4
5   // Use se o APM Server exigir um token
6   secretToken: '',
7
8   // Use se APM Server usa chaves de API para autenticação
9   apiKey: '',
10
11   // Definir URL do servidor APM
12   serverUrl: 'http://localhost:8200',
13 })
14
15 const express = require('express')
16 const app = express()
17 const cors = require("cors")
18
19 require('./services/swagger')
20 app.use(express.json())
21
22 let domains = ['http://localhost:8080', 'http://localhost:3000']
23 const corsOptions = {
24   origin: function (origin, callback) {
25     if (domains.indexOf(origin) !== -1 || !origin) {
26       callback(null, true)
27     } else {
28       callback(new Error('Not allowed by CORS? ${origin} // ${domains}'))
29     }
30   },
31   methods: "GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS",
32   credentials: true
33 }
34
35 app.use(cors(corsOptions))
36 require('./routes')(app)
37 app.use('/v1/docs', express.static('src/views'));
38 app.use('/docs/swagger.yaml', express.static('src/docs/swagger.yaml'));
39
40 app.get('/', (req, resp) => resp.send('Server on'));
41 app.listen(3000, () => console.log('Servidor rodando na porta 3000')); Odair Sehn
```

Fonte: Elaborada pelo autor.

11.3 Kibana

O Kibana é o último serviço necessário para termos nossa aplicação monitorada, trata-se de uma ferramenta de visualização e gerenciamento de dados disponibilizado em diversos formatos, como gráficos de pizza, gráficos de linha, histograma, entre outros. É no Kibana que torna-se possível a visualização das métricas coletadas pelo APM Server, da mesma forma que as outras aplicações, ele é subido via docker, declarado no docker-compose.

A figura a seguir apresenta o serviço do Kibana declarado no `docker-compose.yml`. Onde é possível observar que foi declarado uma dependência do Elasticsearch e a porta exposta é a 5601, isso significa que é permitido acesso ao Kibana através do navegador.

Figura 18 - Kibana declarado no docker-compose

```
docker-compose.yml X
E-commerce > docker-compose.yml > version
51
52 kibana:
53   image: docker.elastic.co/kibana/kibana:7.11.1
54   depends_on:
55     elasticsearch:
56       condition: service_healthy
57   environment:
58     ELASTICSEARCH_URL: http://elasticsearch:9200
59     ELASTICSEARCH_HOSTS: http://elasticsearch:9200
60   ports:
61     - 5601:5601
62   networks:
63     - elastic
64   healthcheck:
65     interval: 10s
66     retries: 20
67     test: curl --write-out 'HTTP %{http_code}' --fail --silent --output /dev/null http://localhost:5601/api/status
68
```

Fonte: Elaborado pelo autor.

11.4 Monitoramento e Observabilidade finalizados

Após configurado todo o back-end, front-end, APM Server, Elasticsearch e Kibana, torna-se possível fazer uma requisição no front-end, que chamará o back-end e em tempo real observar essa transação no Kibana.

Em seguida será apresentado o caminho completo de uma requisição no front-end até a visualização das métricas no Kibana.

1. Na figura a seguir foram feitas duas requisições no front-end, uma na criação da tela, onde chamou a `GET/product` e outra para deletar um produto, que foi requisitada a `DELETE/product`.

Figura 19 - Requisições feitas no front-end

The screenshot shows a table with columns: Código, Descrição, Tamanho, Custo, Valor de venda, and Ações. A modal dialog titled 'Confirmação' is open, asking 'Deseja confirmar a exclusão do produto?' with 'Cancelar' and 'Confirmar' buttons.

Código	Descrição	Tamanho	Custo	Valor de venda	Ações
18	Casaco Moletom	M	70.00000	130.00	[Edit] [Delete]
19	Casaco Moletom	G	70.00000	130.00	[Edit] [Delete]
20	Camiseta Algodão	P	60.00000	100.00	[Edit] [Delete]
21	Camiseta Algodão	M	60.00000	100.00	[Edit] [Delete]
22			60.00000	100.00	[Edit] [Delete]
23			60.00000	100.00	[Edit] [Delete]
24			80.00000	150.00	[Edit] [Delete]

Fonte: Elaborado pelo autor.

2. A figura a seguir apresenta que no Kibana, de forma imediata, é atualizado duas novas transações, que foram as chamadas no front-end, com informações iniciais referentes a latência, taxa de transferência e taxa de erro.

Figura 20 - Transações registradas no Kibana

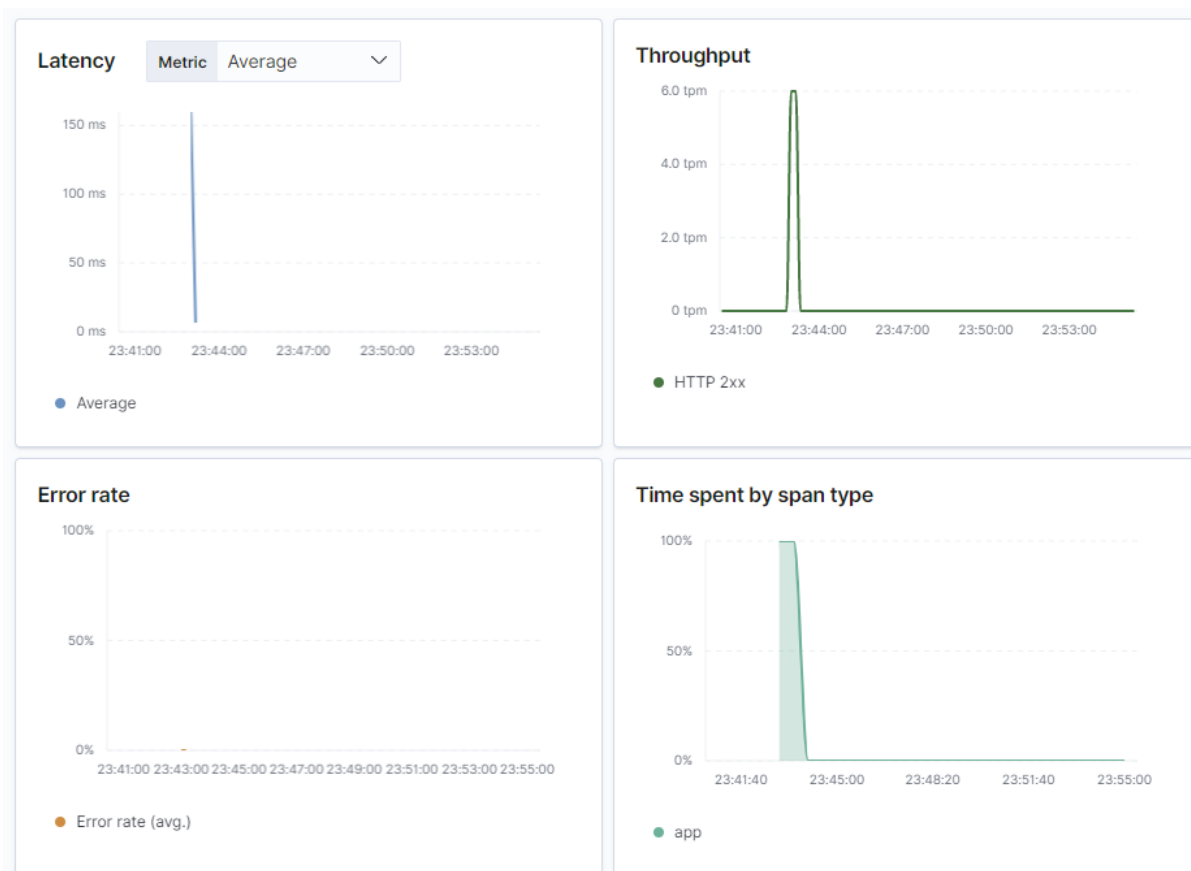
The screenshot shows the Kibana Transactions view with columns: Name, Latency (avg.), Throughput, Error rate, and Impact. Two transactions are listed: GET /product and DELETE /product/:pro_c....

Name	Latency (avg.)	Throughput	Error rate	Impact ↓
GET /product	83 ms	0.4 tpm	0%	[Bar chart]
DELETE /product/:pro_c...	37 ms	0.2 tpm	0%	[Bar chart]

Fonte: Elaborado pelo autor.

3. A figura a seguir apresenta que ao detalhar a transação do GET/product é possível visualizar essas mesmas informações em formato de gráficos, com o horário da transação.

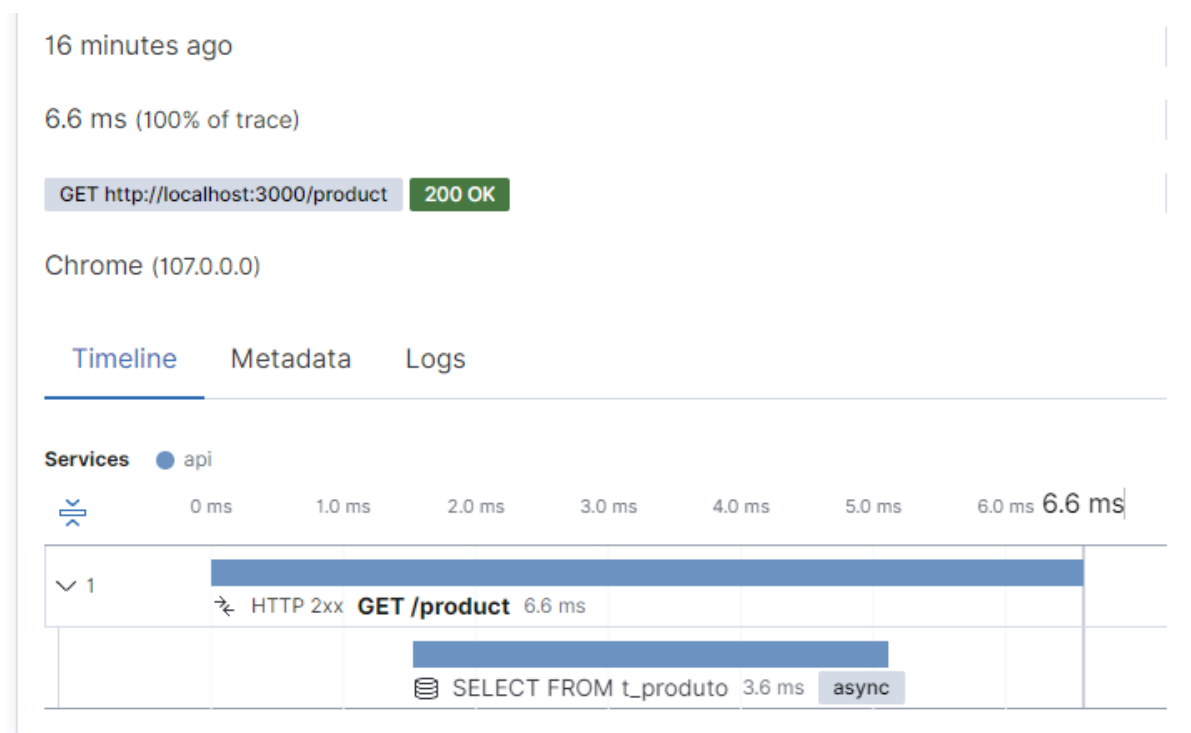
Figura 21 - Gráficos ao detalhar transação GET do produto



Fonte: Elaborado pelo autor.

4. Na figura a seguir é apresentado a amostra de rastreamento, outra informação relevante da transação, que informa detalhes como tempo de execução da transação, URL chamada, retorno obtido e até qual o SQL que foi executado

Figura 22 - Amostra de rastreamento de requisição GET



Fonte: Elaborado pelo autor.

5. Na figura a seguir é exemplificado que ao clicar no SELECT é possível observar com mais detalhe qual o SQL executado, qual a porcentagem de tempo que ele utilizou em relação a toda a transação e qual o banco de dados.

Figura 23 - Informações ao detalhar SQL GET

Span details [View span in Discover](#)

Name	Service	Transaction
SELECT FROM t_produto	api	GET /product
21 minutes ago 3.6 ms (55% of transaction)	DB postgresql query	async

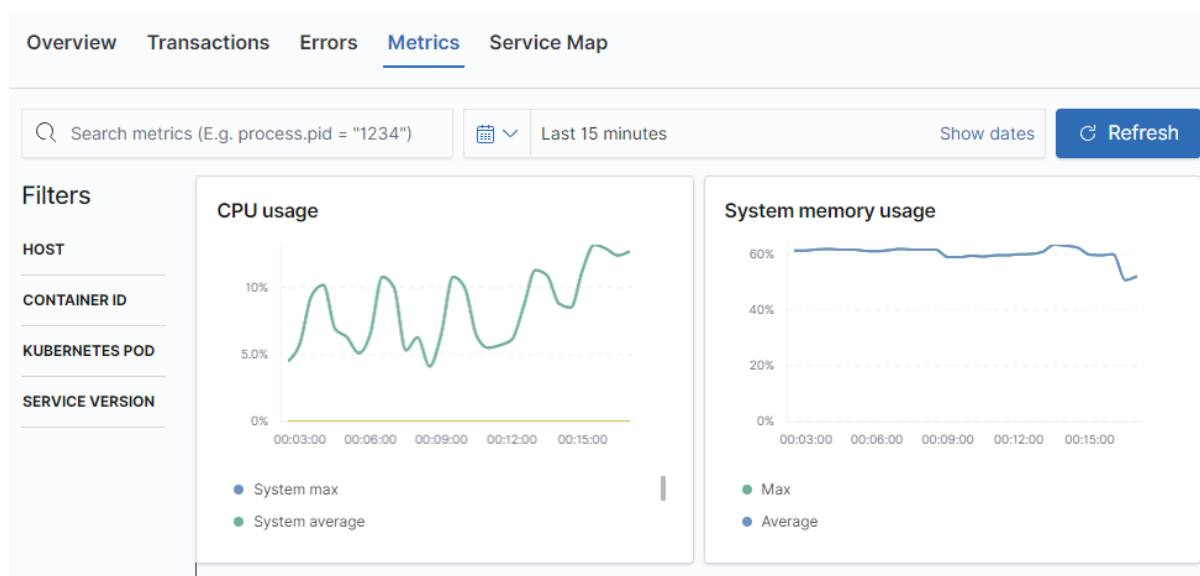
Database statement

```
select pro_codigo,  
       pro_descricao,  
       pro_tamanho,  
       pro_custo,  
       pro_valor  
from t_produto
```

Fonte: Elaborado pelo autor.

6. Ainda é possível detalhar o metadata, onde disponibiliza toda informação referente a requisição HTTP, host da aplicação, requisição na API, agente configurado no back-end, URL requisitada e ao usuário agente que fez a requisição.
7. Também é possível analisar dados referentes ao servidor, sendo fácil identificar possíveis gargalos de processamento ou falta de espaço de armazenamento, conforme apresenta a figura a seguir.

Figura 24 - Gráficos do processamento e armazenamento do servidor



Fonte: Elaborado pelo autor.

12. CONSIDERAÇÕES FINAIS

Com o avanço do uso da tecnologia como uma dependência da maioria das tarefas do dia a dia, seja pessoal, comercial, ou empresarial, é natural que estas aplicações se tornem maiores, mais difíceis de serem controladas e gerenciadas. O trabalho apresenta um auxiliar que foi implementado do zero, sendo o Elastic, juntamente com uma aplicação criada do zero, para melhor controle da mesma, possibilitando a captura de forma fácil e com clareza das execuções executadas na aplicação, mantendo um registro destas, sendo possível exibi-las em dashboards.

O projeto entregue atende o objetivo de monitorar e observar a aplicação, rastreando todas as ações executadas pelos usuários. Porém, como sugestão de melhoria, conforme o tamanho da aplicação for crescendo, outros pontos críticos vão ser identificados e podem se tornar interessantes serem monitorados, estes vão depender da necessidade. Como por exemplo, subir esse serviço na nuvem por limitações de hardware, personalizar alertas, criar dashboards personalizados, monitorar componentes específicos no front-end, etc. O principal para isso se tornar possível já foi realizado, que é a integração da aplicação com o serviço do Elasticsearch trabalhando com o Kibana. Para adicionar mais formas de monitoramentos, pode ser feito facilmente incluindo novas dependências no projeto, inclusive disponíveis pelo Elastic.

13. REFERÊNCIAS BIBLIOGRÁFICAS

O Especialista. **Maior apagão da história das redes faz Zuckerberg perder US\$ 6 bilhões.** Acesso em: 02/05/2022. Disponível em: <https://oespecialista.com.br/whatsapp-facebook-e-instagram-voltam-apos-6-horas-de-apagao/>.

App Annie. **Daily Time Spent in Apps Rise as Much as 45% in Two Years as Pandemic Bolsters the Mobile Habit.** Acessado em: 02/05/2022. Disponível em: <https://www.data.ai/en/insights/market-data/q2-2021-market-index-regional-rankings/>

IBM Cloud Education. **Microserviços.** Acesso em: 15/05/2022. Disponível em: <https://www.ibm.com/br-pt/cloud/learn/microservices>

El-Masri, Petrillo, Guéhéneuc, Hamou-Lhadj, Bouziane. **systematic literature review on automated log abstraction techniques.** Acesso em: 16/05/2022. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0950584920300264>

Ryan Black. **23 métricas de desenvolvimento de software que devem ser monitoradas.** Acesso em 17/05/2022. Disponível em: <https://www.computerweekly.com/br/tip/23-metricas-de-desenvolvimento-de-software-que-devem-ser-monitoradas#:~:text=M%C3%A9tricas%20de%20desempenho%20de%20software.n%C3%A3o%20o%20que%20ele%20faz>

Brian Barrett, **The Catch-22 That Broke the Internet.** Acesso em 23/05/2022. Disponível em: <https://www.wired.com/story/google-cloud-outage-catch-22/#:~:text=12%3A26%20PM-.The%20Catch%2D22%20That%20Broke%20the%20Internet.Google%20needed%20to%20fix%20it.&text=Five%20days%20ago%2C%20the%20internet.around%20the%20globe%2C%20YouTube%20sputtered>

Tecnologia Vertigo, **O que é DevOps?**. Acesso em: 23/05/2022. Disponível em: <https://vertigo.com.br/o-que-e-devops/>

Jeanderson Cândido, Maurício Aniche, Arie van Deursen, **Log-based software monitoring: a systematic mapping study.** Acesso em: 23/05/2022. Disponível em: <https://peerj.com/articles/cs-489/#fig-1>

Denis Alcides Rezende, **Evolução da Tecnologia da Informação nos Últimos 45 Anos.** Fonte: revista FAE BUSINESS: <https://img.fae.edu/galeria/getImage/1/16578659447373246.pdf>

Wilson Laia, **A evolução do software**. Acesso em: 23/05/2022. Disponível em: <https://www.tiespecialistas.com.br/a-evolucao-do-software/>

Efraim Turban, Linda Volonino, **Tecnologia da Informação para Gestão**. Acesso em: 25/05/2022. Disponível em: <https://docero.com.br/doc/csxvnx>

Alberto Oliveira de Lima Filho, **Sistema de informações**. Acesso em: 25/05/2022. Disponível em: <https://www.scielo.br/j/rae/a/J3vT3wZMCNdyQdvtngQrQpH/?lang=pt#:~:text=O%20sistema%20de%20informa%C3%A7%C3%B5es%2C%20por,processo%20de%20tomada%20de%20decis%C3%B5es>

Marco Tulio Valente, **Engenharia de Software Moderno**. Acesso em: 25/05/2022.

Disponível em: <https://engsoftmoderna.info/cap1.html>

Marco Tulio Valente, **Engenharia de Software Moderno**. Acesso em: 25/05/2022.

Disponível em: <https://engsoftmoderna.info/cap7.html>

David Garlan, Mary Shaw, **An Introduction to Software Architecture**. New Jersey, World Scientific Publishing Company, 1993.

Jones, **Introdução ao Modelo Multicamadas**. Acesso em: 29/05/2022. Disponível em: <https://www.devmedia.com.br/introducao-ao-modelo-multicamadas/5541#3>

Raghu R. Kodali, **What is service-oriented architecture?**. Acesso em 29/05/2022. Disponível em: https://www.pcworld.idg.com.au/article/133882/what_service-oriented_architecture_/

IBM Cloud Education, **SOA (Arquitetura Orientada a Serviços)**. Acesso em: 29/05/2022. Disponível em: <https://www.ibm.com/br-pt/cloud/learn/soa>

Charles Abrams, Roy W. Schulte, **Service-Oriented Architecture Overview and Guide to SOA Research**. Stamford - EUA, Gartner, 2008.

Susan J. Fowler, **Microserviços prontos para a produção**, São Paulo - SP, Novatec Editora Ltda, 2017.

Oracle, **Saiba mais sobre a arquitetura de microserviços**. Acesso em: 29/05/2022. Disponível em: <https://docs.oracle.com/pt-br/solutions/learn-architect-microservice/index.html#GUID-BDCEFE30-C883-45D5-B2E6-325C241388A5>

Charity Majors, Liz Fong-Jones, George Miranda, **Observability Engineering**. Gravenstein Highway North, O'Reilly Media, 2022.

Scott Carey, **What is observability? Software monitoring on steroids**. Acesso em 07/06/2022. Disponível em: <https://www.infoworld.com/article/3607980/what-is-observability-software-monitoring-on-steroids.html>

Pedro César Tebaldi, **Qual a diferença entre monitoramento e observabilidade?**. Acesso em: 07/06/2022. Disponível em: <https://www.opservices.com.br/diferenca-entre-monitoramento-e-observabilidade/#:~:text=Voc%C3%AA%20pode%20pensar%20em%20observabilidade,que%20um%20sistema%20%C3%A9%20observ%C3%A1vel>

Amazon Web Services, **Observabilidade**. Acesso em: 07/06/2022. Disponível em: <https://aws.amazon.com/pt/products/management-and-governance/use-cases/monitoring-and-observability/?whats-new-cards.sort-by=item.additionalFields.postDateTim e&whats-new-cards.sort-order=desc&blog-posts-cards.sort-by=item.additionalFields.createdDate&blog-posts-cards.sort-order=desc>

Ernest Mueller, **Monitoring and Observability**. Acesso em: 07/06/2022. Disponível em: <https://theagileadmin.com/2018/02/16/monitoring-and-observability/>

Plínio Ventura, **O que é UML (Unified Modeling Language)**. Acesso em: 26/10/2022. Disponível em: <https://www.ateomomento.com.br/diagramas-uml/>

IBM Cloud Education, **Diagramas de Classes**. Acesso em: 26/10/2022. Disponível em: <https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=structure-class-diagrams>

Krystal, **O guia do iniciante para desenvolvimento de back-end**. Acesso em: 29/10/2022. Disponível em: <https://learntocodewith.me/posts/backend-development/>

Carlos Estrella, **O que é JavaScript?**. Acesso em: 08/11/2022. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-javascript>

Lennon, **Node.js – O que é, como funciona e quais as vantagens**. Acesso em: 08/11/2022. Disponível em: <https://www.opus-software.com.br/node-js/#>

Clara Fabro, **O que é API e para que serve? Cinco perguntas e respostas**. Acesso em: 17/11/2022. Disponível em: <https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml>

Ana Paula Andrade, **O que é o Express.js?**. Acesso em: 17/11/2022. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-o-express-js>

Cícero Joasyo Mateus de Moura, **Criando Documentação para a sua API com o Swagger.** Acesso em 18/11/2022. Disponível em: <https://fcnuvem.com.br/home/blog-criando-documentacao-para-a-sua-api-com-o-swagger/>

Amanda Machado, **Qual a diferença entre front-end e back-end?.** Acesso em: 20/11/2022. Disponível em: <https://tecnoblog.net/responde/qual-a-diferenca-entre-front-end-e-back-end/>

Rafael Pinheiro, **Vue.js: tudo sobre o framework para trabalhar com mídias interativas.** Acesso em 20/11/2022. Disponível em: <https://rockcontent.com/br/talent-blog/vue-js/>

Patrick Pinheiro Monteiro, **Vuex: padronizando o gerenciamento de estado da sua aplicação Vue.js.** Acesso em: 20/11/2022. Disponível em: <https://imasters.com.br/desenvolvimento/vuex-padronizando-o-gerenciamento-de-estado-da-sua-aplicacao-vue-js>

Ralf Abueg, **Elasticsearch: What It Is, How It Works, And What It's Used For.** Acesso em: 21/11/2022. Disponível em: <https://www.knowi.com/blog/what-is-elastic-search/>

Paulo Frugis Neto, **Docker: O que é? Como usar? Serve para minha empresa?.** Acesso em: 22/11/2022. Disponível em: <https://www.microserviceit.com.br/docker/>