Uma arquitetura para Softwares Móveis orientada à Observabilidade: O caso do Ibricks

Filipe L. Oliveira, Amanda Ferraz de O. Passos, Luis Paulo da S. Carvalho

[1]Instituto Federal de Educação, Ciência e Tecnologia (IFBA)
CEP 45078-900 – Vitória da Conquista – BA – Brasil

filipeelacerda@gmail.com, amandaferraz@ifba.edu.br,

luiscarvalho@ifba.edu.br

Abstract. *Observability refers to interpreting data received from a system to predict errors at the slightest data anomaly. The goal of an observability platform is to find the normal performance baseline of a system and then improve it. To achieve this goal, it's necessary to relate three pillars: metrics (what to measure?), logs, and traces (where to extract metrics?). For effective and strategic observability, these pillars must be incorporated into software system architectures from the design phase. IBricks is a mobile software developed as a mini-world for the disciplines of mobile programming and project management in the Bachelor of Information Systems course at IFBA, Vitória da Conquista campus. The idea behind the application is to enable the rental of civil construction products through mobile devices. As this is a promising application with the potential to serve many customers and users, it's crucial to have automated observability in place. This work aimed to incorporate observability principles into the IBricks architecture. As a contribution, it explores and describes how such principles can be automated and how they can potentially help manage this type of software.*

Resumo. *Observabilidade diz respeito a interpretar os dados recebidos de um sistema de forma que se consiga prever um erro à mínima falha nos dados. O objetivo de uma plataforma de observabilidade é encontrar o ponto de desempenho normal de um sistema e depois melhorá-lo. Para conseguir atingir este objetivo é necessário relacionar três pilares: métricas (o que medir?), logs e traces (de onde extrair as métricas?). Para que a observabilidade ocorra bem, de forma estratégica, se faz necessário que tais pilares sejam incorporados às arquiteturas de sistemas de software desde a fase de concepção. O IBricks é um software móvel que foi desenvolvido como mini-mundo para as disciplinas de programação para dispositivos móveis e gestão de projetos do curso de Bacharelado de Sistemas de Informação do IFBA, campus Vitória da Conquista. A ideia que embasa o aplicativo é possibilitar o aluguel de produtos da construção civil através de dispositivos móveis. Como se trata de algo promissor, com a possibilidade de atender muitos clientes e usuários, se faz necessário que a observabilidade já se encontre automatizada. Este trabalho teve, como objetivo, incorporar à arquitetura do IBricks princípios de observabilidade. Como contribuição, é explorado e descrito como tais princípios podem ser automatizados e como, possivelmente, eles podem ajudar a gerir este tipo de software.*

1. Introdução

Observabilidade diz respeito a interpretar os dados recebidos de um sistema de forma que seja possível prever um erro à mínima falha nos dados. O objetivo de uma plataforma de observabilidade é encontrar o ponto de desempenho normal de um sistema e depois melhorá-lo [Santos Carvalho 2022]. Para conseguir atingir este objetivo é necessário relacionar três pilares: *métricas* (o que medir?), *logs* (o que foi registrado naquela data/hora?) e *traces* (de onde extrair as métricas?) [Santos Carvalho 2022]. Para que a observabilidade ocorra bem, de forma estratégica, se faz necessário que tais pilares sejam incorporados à arquitetura do sistema de software que se deseja observar desde a sua concepção. [Majors et al. 2022] destacam que observabilidade é importante para otimizar sistemas, identificando gargalos de desempenho e antecipando problemas.

A crescente ubiquidade dos dispositivos móveis transformou a maneira como interagimos com a tecnologia, influenciando setores como comunicação, comércio e entretenimento. No entanto, a complexidade dos aplicativos móveis modernos, frequentemente interconectados com serviços externos e executados em uma variedade de dispositivos, cria desafios para os desenvolvedores em termos de qualidade, manutenção e desempenho. Este é um dos contextos em que a observabilidade pode auxiliar, fornecendo elementos para averiguação desses aspectos.

Considerando a arquitetura de software, [Pressman 2005] destaca a sua importância como um projeto estrutural que define a organização do sistema, seus componentes e suas interações. Desta forma, uma arquitetura adequada é crucial para criar aplicativos móveis escaláveis, flexíveis e sustentáveis. Adicionalmente, [Santos Carvalho 2022] propõem que a observabilidade seja um aspecto central da arquitetura, permitindo que os desenvolvedores coletem, processem e analisem dados operacionais para entender o comportamento do sistema.

O Ibricks é um software móvel que foi desenvolvido como mini-mundo para as disciplinas de programação para dispositivos móveis e gestão de projetos do curso de Bacharelado de Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA), *campus* Vitória da Conquista. Esse software permite cadastrar várias lojas que oferecem o aluguel de máquinas voltadas para a construção civil. A ausência de uma arquitetura orientada à observabilidade dificulta a identificação de problemas e a manutenção do sistema, especialmente em cenários onde o aplicativo é utilizado em uma variedade de dispositivos móveis. Isso resulta em desafios como a dificuldade de monitorar o desempenho e a confiabilidade, além de aumentar o tempo necessário para resolver falhas.

A Figura 1 exibe a versão inicial da arquitetura do Ibricks, onde é possível visualizar uma tela de *feed* listando algumas lojas que oferecem o serviço de aluguel de máquinas para construção. Tanto o front-end quanto o back-end são peças fundamentais no desenvolvimento de software, trabalhando juntos para garantir o bom funcionamento de aplicativos modernos. É importante destacar que a observabilidade do back-end é vital, pois ela possibilita monitorar e aprimorar requisitos não funcionais como desempenho, segurança e confiabilidade, assegurando uma experiência rápida, segura e estável para o usuário.

À vista do que foi dito, para este trabalho foi desenvolvida uma arquitetura ori-

entada à observabilidade para um aplicativo móvel, o Ibricks, visando oferecer uma infraestrutura que permita a monitoramento em tempo real, a detecção proativa de falhas e a otimização contínua do desempenho. Ao direcionar esforços para este campo emergente, vislumbra-se a oportunidade de aprimorar a experiência do usuário e proporcionar soluções confiáveis e resilientes para o aplicativo.

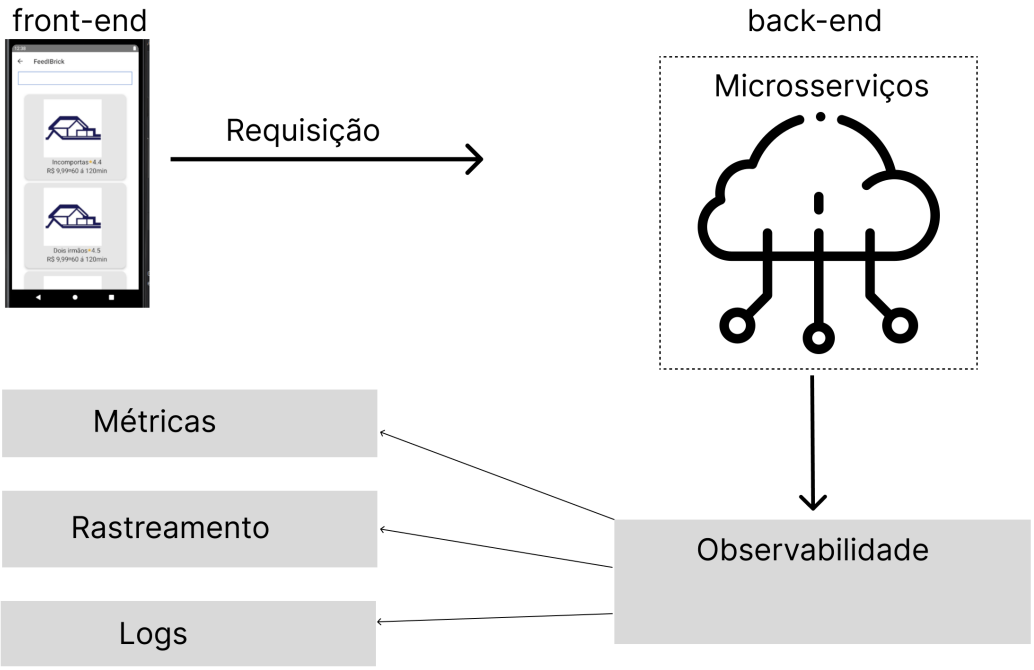

Figura 1. Ilustração front-end e back-end

Resumidamente, o problema considerado por este trabalho corresponde à seguinte pergunta: **considerando um ponto inicial, dado um software móvel como o iBricks, cuja arquitetura ainda não é orientada à observabilidade, de que forma pode ser realizada a aplicação dos princípios de observabilidade visando tornar o aplicativo robusto e resiliente?**

2. Método de Pesquisa

Este trabalho adotou uma abordagem predominantemente quantitativa, focando na análise numérica e quantificação de fenômenos. Trata-se também de uma pesquisa descritiva, que buscou detalhar características e fenômenos do contexto investigado. A coleta de dados ocorreu em ambiente controlado de laboratório.

Seguiu-se uma abordagem dedutiva, partindo de teorias estabelecidas para formulação e teste de hipóteses específicas. Em relação a procedimentos, os seguintes passos, exibidos e relacionados na Figura 2, foram seguidos (cada retângulo é um passo):

1. Atividade 1 - foi feita a migração tecnológica do Ibricks de React-Native [1] para Flutter [2], por se entender que este é um framework robusto que vem atraindo atenção de muitos desenvolvedores ultimamente [Gülcüoğlu et al. 2021];

[1]https://reactnative.dev/
[2]https://flutter.dev/

2. Atividade 2 - foram pesquisados os princípios da observabilidade para implementação e conceituação no projeto. Esta atividade foi permanente ao longo de toda a pesquisa;
3. Atividade 3 - foram fomentadas e implementadas novas funcionalidades na aplicação a fim de ter mais cenários para aplicar os princípios e práticas da observabilidade;
4. Atividade 4 - o back-end desenvolvido durante a disciplina de programação para dispositivos móveis, foi atualizado para contemplar novas funcionalidades;
5. Atividade 5 - foi realizado o levantamento de ferramentas voltadas à observabilidade para implementar na arquitetura de software móvel Ibricks;
6. Atividade 6 - foi demonstrado o monitoramento de métricas e o impacto da observabilidade a partir de uma simulação sobre o back-end do Ibricks;
7. Atividade 7 - foram registrados neste trabalho os resultados a partir do uso do IBricks e de teste de cargas simulados sobre sua arquitetura.

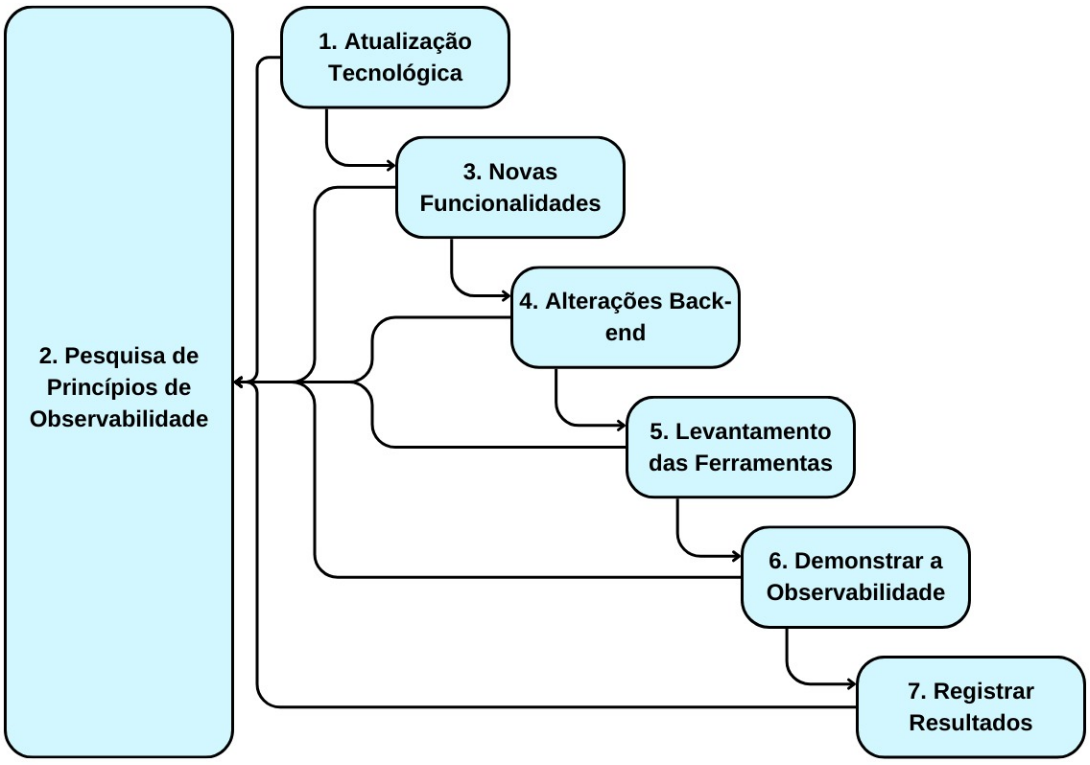

Figura 2. Fluxo de trabalho

3. Fundamentação Teórica e Técnica

A observabilidade é essencial para o entendimento do comportamento interno dos sistemas de software através de suas saídas externas [Majors et al. 2022]. Mais do que apenas monitorar métricas e registros, a observabilidade envolve a capacidade de compreender o estado interno do sistema por meio de ferramentas e práticas que permitem aos profissionais diagnosticar e resolver problemas de forma eficaz. Ao fornecer uma visão detalhada do funcionamento do sistema, a observabilidade é fundamental para identificar problemas complexos, entender o impacto das mudanças no sistema e garantir sua confiabilidade e eficiência operacional.

As próximas seções discutem os princípios relacionados à observabilidade que serão embutidos na arquitetura do Ibricks. Os princípios discutidos foram extraídos da literatura correlata, "*Observability Engineering: Achieving Production Excellence*"[Majors et al. 2022], uma das mais importantes nesta área.

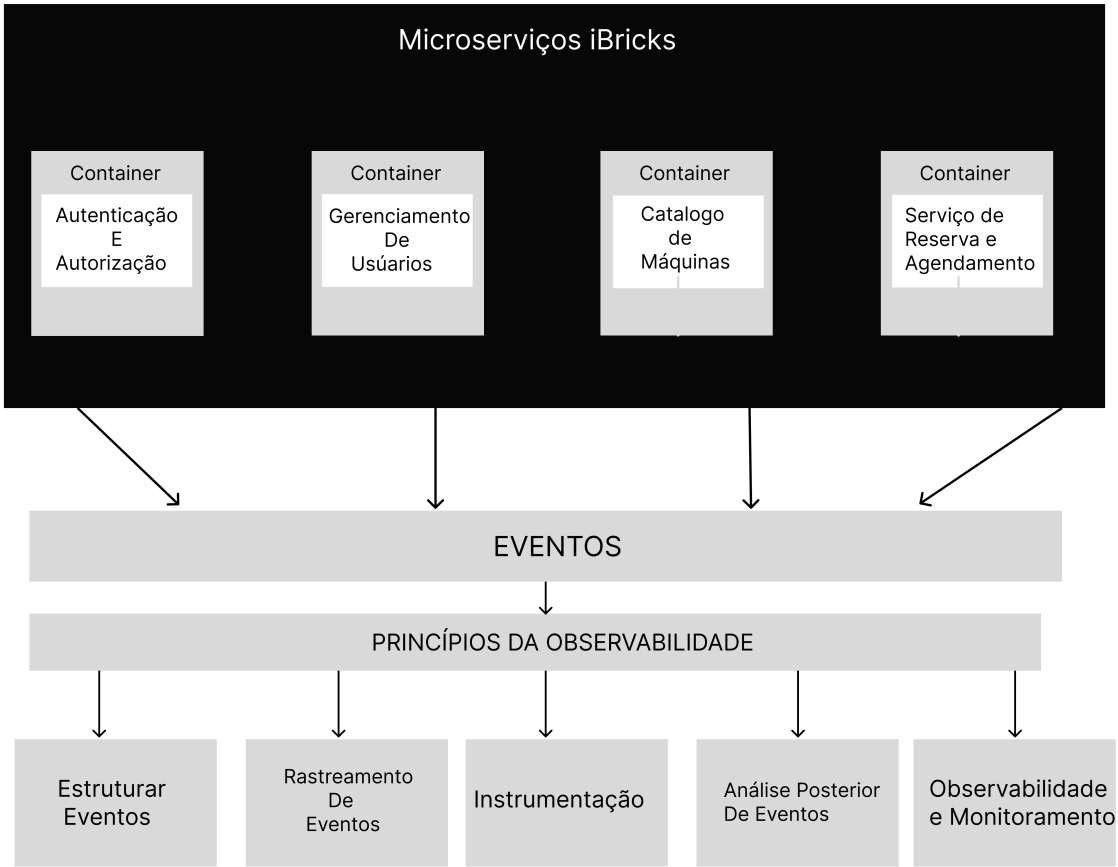

Figura 3. Relacionamento entre o IBricks e Princípios de Observabilidade

3.1. Estruturar eventos

A base essencial da observabilidade reside nos eventos estruturados de amplitude arbitrariamente ampla. Isso implica a capacidade de segmentar dados em inúmeras dimensões,

proporcionando respostas a qualquer indagação durante o processo de depuração interativa.

A amplitude arbitrária desses eventos é crucial, pois necessitam conter dados suficientes para permitir uma compreensão abrangente do que ocorreu quando um sistema executou uma unidade de trabalho. Isso geralmente implica definir o escopo de um evento como o registro completo do que transcorreu ao longo do ciclo de vida de uma solicitação de processamento de dados.

Enquanto os logs não estruturados são legíveis para humanos, sua utilização computacional é desafiadora. Por sua vez, os logs estruturados podem ser analisados por máquinas e se mostram valiosos para os objetivos de observabilidade, desde que sejam reconfigurados para se assemelhar a eventos estruturados.

Um exemplo de log estruturado é exibido na Figura 4. No exemplo, são mostrado logs de monitoramento de processamento consumido por um serviço. O log está estruturado em formato *JavaScript Object Notation* (JSON) para facilitar a identificação dos parâmetros.

3.2. Possibilitar rastreamento de eventos

Os eventos são elementos fundamentais da observabilidade, enquanto os rastreamentos consistem em uma série interconectada de eventos.

Um exemplo de rastreamento é exibido na Figura 5. A ilustração apresenta três pontos finais (*end-points*). Ao acionar o primeiro *end-point*, um identificador de rastreamento, denominado traceId, é gerado. Esse traceId é exclusivamente utilizado para agrupar os diferentes serviços associados a esse evento. Por sua vez, o parentId é empregado para identificar eventos específicos no sistema, facilitando o rastreamento desses de maneira distinta.

3.3. Aplicar à instrumentação

A instrumentação em observabilidade é crucial para monitorar e entender sistemas complexos na infraestrutura digital moderna. Envolve a incorporação de métricas, registros e rastreamento para obter uma visão detalhada e em tempo real do sistema.

Métricas fornecem medidas quantitativas do estado e desempenho, enquanto registros registram eventos relevantes. O rastreamento permite seguir o fluxo de uma solicitação através de componentes distribuídos, identificando gargalos e pontos de falha. Essa prática é essencial para detectar e resolver problemas rapidamente, otimizar a eficiência e proporcionar uma experiência confiável aos usuários em ambientes digitais complexos e em constante evolução.

Um exemplo de instrumentação se encontra na Figura 6. Pontos de instrumentação de código são adicionados para registrar dados relevantes, permitindo entender o desempenho e o comportamento do sistema em tempo real por meio de logs, métricas, rastreamento e distribuição de rastreamento.

3.4. Permitir a análise posterior de eventos

A coleta precisa de dados de telemetria representa o primeiro passo crucial na jornada em direção à observabilidade. No entanto, a simples coleta não é suficiente. É imperativo

```
{
  "help": "Total user CPU time spent in seconds.",
  "name": "node_process_cpu_user_seconds_total",
  "type": "counter",
  "values": [
    {
      "value": 14.252388000000003,
      "labels": {}
    }
  ],
  "aggregator": "sum"
},
{
  "help": "Total system CPU time spent in seconds.",
  "name": "node_process_cpu_system_seconds_total",
  "type": "counter",
  "values": [
    {
      "value": 4.896763000000007,
      "labels": {}
    }
  ],
  "aggregator": "sum"
},
```


Figura 4. Log estruturado

analisar esses dados conforme os princípios fundamentais, a fim de identificar de maneira objetiva e precisa problemas de aplicação em ambientes complexos. Conforme ilustrado na Figura 7, a visualização dos dados coletados, como métricas de serviço, permite aos operadores identificar tendências e anomalias que poderiam indicar problemas subjacentes.

Ao buscar fontes de anomalias, é possível aproveitar os recursos computacionais para examinar conjuntos de dados extensos e identificar padrões relevantes de forma ágil. A apresentação desses padrões a um operador humano, capaz de contextualizá-los e direcionar a investigação, alcança um equilíbrio eficaz que otimiza as capacidades tanto das máquinas quanto dos humanos na resolução rápida de problemas do sistema. Os sistemas de observabilidade são projetados para aplicar essa abordagem analítica aos dados de

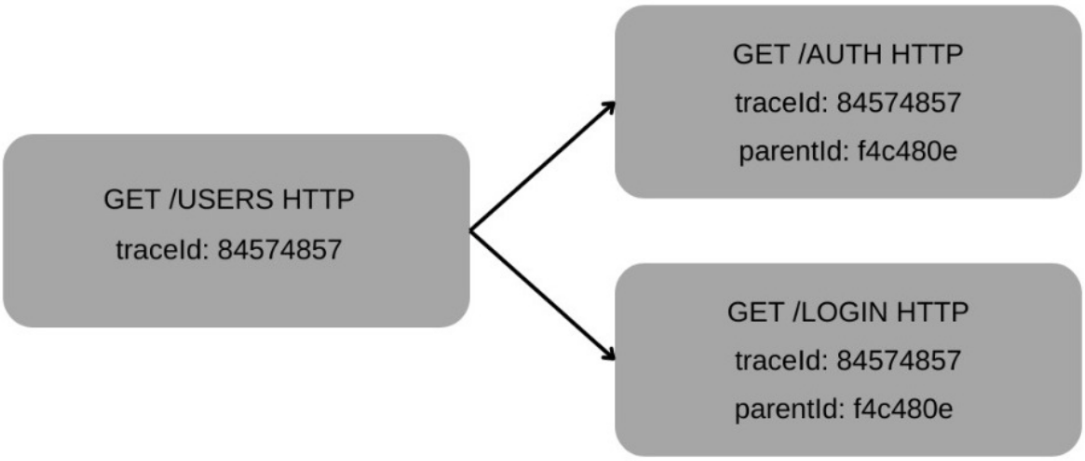

Figura 5. Exemplo de evento, gerando um id para o *trace* e o *parent*

Figura 6. Exemplo de instrumentação

eventos previamente coletados.

3.5. Integrar observabilidade e monitoramento

A integração entre observabilidade e monitoramento é fundamental para garantir a integridade e eficiência dos sistemas de software. O monitoramento desempenha um papel central na avaliação da integridade dos sistemas, concentrando-se nos aspectos operacionais e de desempenho, enquanto a observabilidade foca na capacidade de entender e diagnosticar o comportamento interno do software.

A proporção ideal entre essas duas abordagens varia de acordo com o nível de terceirização da infraestrutura para provedores externos, como os provedores de serviços em nuvem. Equilibrar monitoramento e observabilidade de forma complementar permite uma visão mais completa do ambiente de software, facilitando a detecção precoce de problemas e a implementação de melhorias contínuas. Além disso, a integração dessas práticas não se restringe apenas a considerações tecnológicas, mas também requer

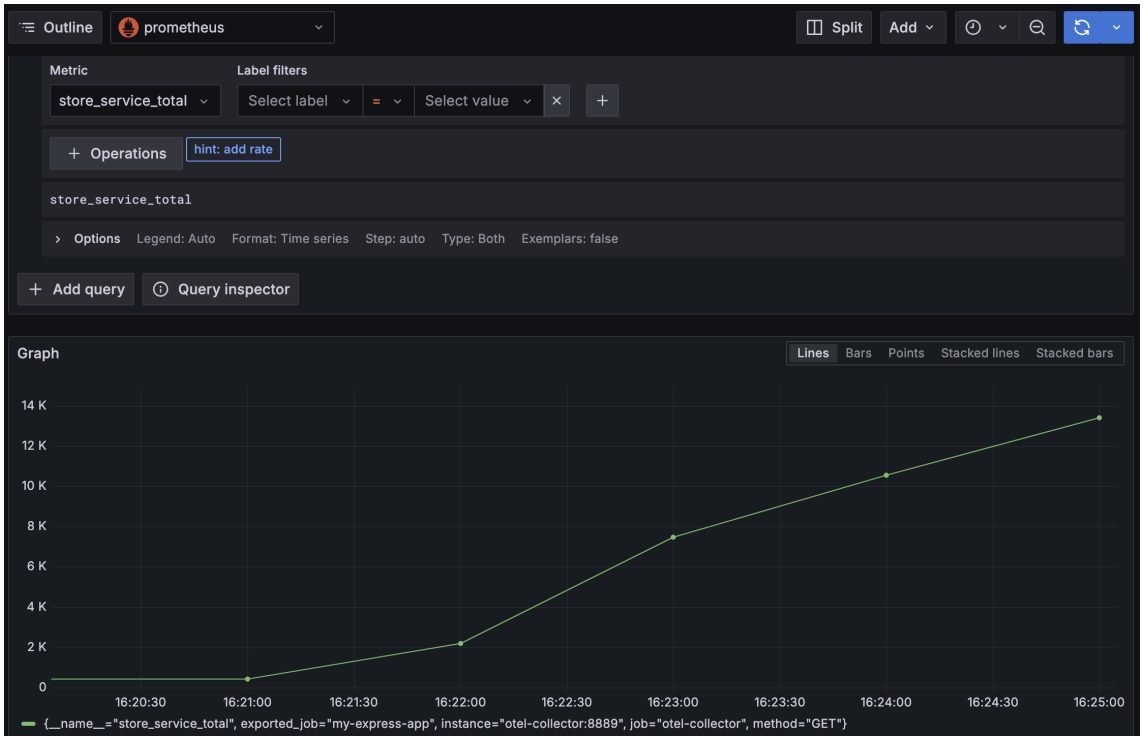

Figura 7. Visualização de métricas de serviço coletadas no Prometheus

transformações culturais para promover sua adoção efetiva em todas as equipes envolvidas no processo de desenvolvimento e operação.

Na Figura 8, extraída de [Lopes 2023], podemos ver um exemplo de monitoramento de uma métrica chamada “Mensagens por Minuto”.

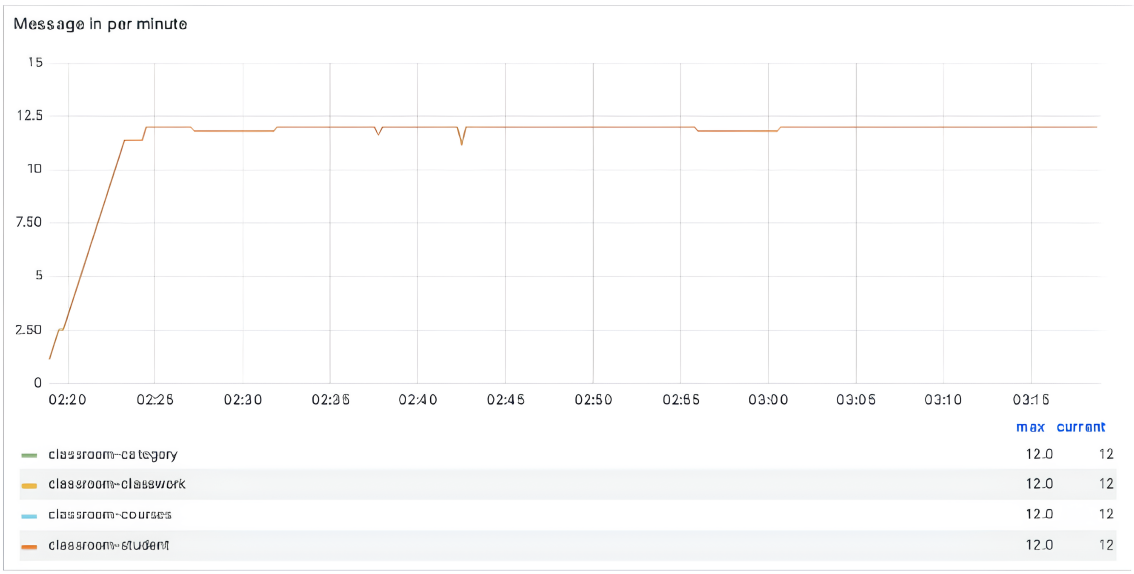

Figura 8. Exemplo de monitoramento

4. O Ibricks

O aplicativo Ibricks é uma solução para o mercado/comércio da área de engenharia civil que visa simplificar o processo de aluguel de máquinas essenciais para projetos de infraestrutura.

O Ibricks proporciona uma plataforma para conectar usuários e fornecedores, agilizando o acesso a equipamentos e materiais necessários para projetos de infraestrutura. Na Tabela 1 estão os requisitos e escopo principal do aplicativo Ibricks. A primeira coluna, "Id", para indentificação do requisito. A segunda coluna "Descrição", significa uma visão de todos os pontos do requisito e, por fim, a terceira, e quarta coluna, significa o que foi implementado de fato do requisito.

4.1. Arquitetura do Ibricks

A arquitetura adotada para o desenvolvimento do Ibricks envolveu vincular um front-end a um back-end. [Renaud 1994] apresenta tal arquitetura como uma estrutura que oferece vantagens significativas, tais como, por exemplo, descentralização de processamento e facilidade de manutenção. Essa abordagem permite uma clara divisão de responsabilidades entre o back-end, que lida com o processamento de dados nos servidores, e o front-end, que cuida da interface do usuário nos clientes. Essa separação eficiente contribui para sistemas mais escaláveis e robustos, melhorando a experiência do usuário de forma geral.

O front-end refere-se à interface visível de um sistema, com a qual os usuários interagem diretamente. Ele é desenvolvido utilizando tecnologias como HTML, CSS e JavaScript, sendo responsável por apresentar informações de maneira clara e interativa. O back-end processa as requisições do front-end, realiza operações no banco de dados e retorna os dados necessários. É composto por servidores, bancos de dados e lógica de negócios [Bass et al. 2012].

Para realizar a atualização tecnológica do Ibricks, foi adotada uma arquitetura baseada em um front-end escrito em Flutter e um back-end orientado a microsserviços para promover a modularidade e escalabilidade do sistema. Essa abordagem permitiu maior flexibilidade no desenvolvimento e manutenção, facilitando futuras atualizações e expansões do projeto.

4.1.1. Front-end

O front-end do aplicativo Ibricks foi desenvolvido utilizando o framework Flutter, que oferece uma abordagem eficaz para criar interfaces de usuário dinâmicas e responsivas para dispositivos Android e IOS. Neste contexto, optamos por adotar o padrão modular, baseado no estilo de programação nativo do Flutter.

A abordagem de programação modular, conforme proposta por [Meyer 1997], destaca a importância de dividir sistemas de software em partes independentes e interconectadas, conhecidas como módulos. Cada módulo possui uma interface claramente definida, o que facilita a comunicação e a colaboração entre eles, sem a necessidade de compreender os detalhes de implementação interna. Esta prática, não apenas estimula a reutilização de código, mas também contribui para a manutenção e testes do sistema.

Id	Descrição	Implementado	
		Sim	Não
RQ1	Os usuários podem criar contas individuais ou empresariais, fornecendo informações como: CPF/CNPJ, nome ou razão social/nome fantasia, endereço, número de telefone e e-mail. Empresas devem fornecer também o escopo operacional.	X	
RQ2	O aplicativo oferece um catálogo completo de máquinas fornecidas pelas lojas disponíveis para aluguel, cada uma com informações detalhadas, imagens e preços por período (diário, semanal, mensal)	X	
RQ3	Os usuários podem selecionar uma máquina e especificar o período desejado para aluguel. O aplicativo verifica a disponibilidade em tempo real e permite a confirmação da reserva	X	
RQ4	Os usuários podem refinar a busca por tipo de máquina, lojas, máquinas específicas, descrição das máquinas e marcas.		X
RQ5	Opções de pagamento incluem métodos de pagamento online		X
RQ6	Os usuários podem deixar avaliações e comentários sobre as máquinas e a experiência de aluguel. Essas avaliações ajudam a construir a reputação dos fornecedores de máquinas		X
RQ7	Os usuários podem visualizar um histórico completo de todas as suas transações de aluguel. Inclui detalhes como datas, máquinas alugadas e valores pagos		X
RQ8	O aplicativo envia notificações sobre status de reservas, confirmações		X
RQ9	Os fornecedores podem cadastrar suas máquinas, gerenciar disponibilidade e atualizar informações sobre os equipamentos	X	

Tabela 1. Implementação de Requisitos do Ibricks

Ao implementar uma arquitetura modular, mostrado na Figura 9, assegura-se uma estrutura de código organizada do front-end, o que pode simplificar a manutenção e desenvolvimento de novas funcionalidades do aplicativo. Ao mesmo tempo, favorece a clara distinção de responsabilidades entre os diversos componentes da aplicação.

O módulo LIB contém o código principal da aplicação, incluindo widgets e lógica

de negócio. Widgets [3] compõem a estrutura visual e funcional da aplicação. A pasta *Services* encapsula as chamadas ao back-end, garantindo a comunicação entre o front-end e o back-end. Em *Models*, encontram-se as classes que representam os objetos de dados da aplicação, servindo como uma ponte entre os dados e a lógica de negócio. Por fim, *Views* abriga os arquivos que definem as diferentes telas e interfaces do usuário, utilizando os widgets e interagindo com os modelos de dados para apresentar informações ao usuário de forma interativa.

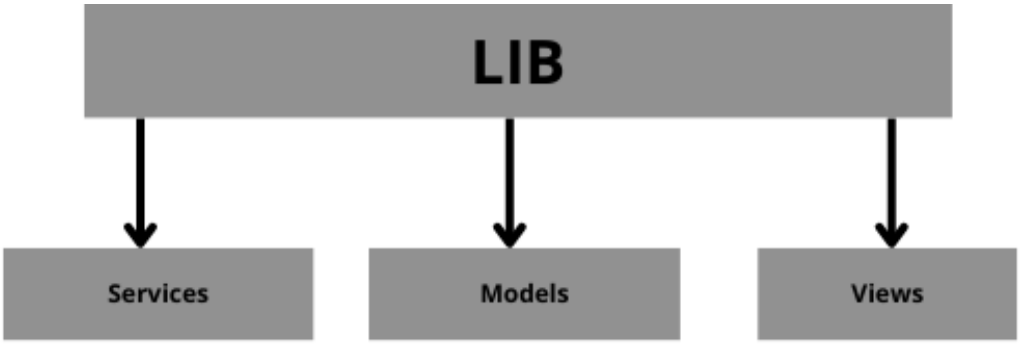

Figura 9. Módulos do front-end

4.1.2. Back-End

O back-end do aplicativo Ibricks foi construído sobre uma arquitetura de microsserviços, utilizando as tecnologias: Node.js [4], Express.js [5], JWT [6] e Prisma ORM [7], todos encapsulados em contêineres Docker [8].

A arquitetura de microsserviços é uma abordagem que se caracterizada pela composição de uma aplicação em pequenos serviços independentes, cada um focado em uma função específica e interconectados por meio de APIs [Fowler et al. 2014]. A importância atribuída a esta abordagem arquitetônica tem sido crescente na indústria de desenvolvimento de software, visto que oferece benefícios como flexibilidade, escalabilidade e capacidade de resposta a mudanças no mercado [Newman 2015].

Os serviços foram encapsulados em contêineres Docker individuais, o que simplifica a implantação, escalabilidade e manutenção. Isso garante um ambiente de execução coeso e isolado. A comunicação entre esses serviços é estabelecida por meio de requisições HTTP. Esta abordagem promove uma integração fluida e eficaz entre os diversos componentes [Vase 2015].

[3] Widgets são elementos de interface gráfica que compõem a estrutura visual e funcional de um aplicativo. Eles podem representar desde botões simples até *layouts* complexos, permitindo a criação de interfaces interativas e dinâmicas. Widgets em Flutter são construídos de forma hierárquica e podem ser personalizados e combinados para criar a aparência desejada e a interação do aplicativo.

[4] https://nodejs.org/en

[5] https://expressjs.com/

[6] https://jwt.io/introduction

[7] https://www.prisma.io/

[8] https://www.docker.com/

Os seguintes microsserviços foram adicionados ao back-end, de acordo com o que está exibido na Figura 10:

1. **Serviços de Autenticação e Autorização:** Responsável pela autenticação e geração de tokens JWT (JSON Web Tokens). Utiliza Node.js e Express.js para criar uma API segura que gerencia a autenticação de usuários e a autorização para acessar os recursos do aplicativo.
2. **Serviço de Gerenciamento de Usuários:** Lida com operações relacionadas aos usuários, como criação, leitura, atualização e exclusão de contas. Utiliza o Prisma ORM para interagir com a base de dados, permitindo a persistência de informações de usuários.
3. **Serviço de Catálogo de Máquinas:** Dedicado ao catálogo de máquinas disponíveis para aluguel. Utiliza o Prisma ORM para gerenciar os dados das máquinas, além de disponibilizar uma API para consultas e atualizações no catálogo.
4. **Serviço de Reservas e Agendamentos:** Responsável por gerenciar as reservas de máquinas feitas pelos usuários. Interage com o serviço de catálogo para verificar a disponibilidade das máquinas e registra as reservas correspondentes.

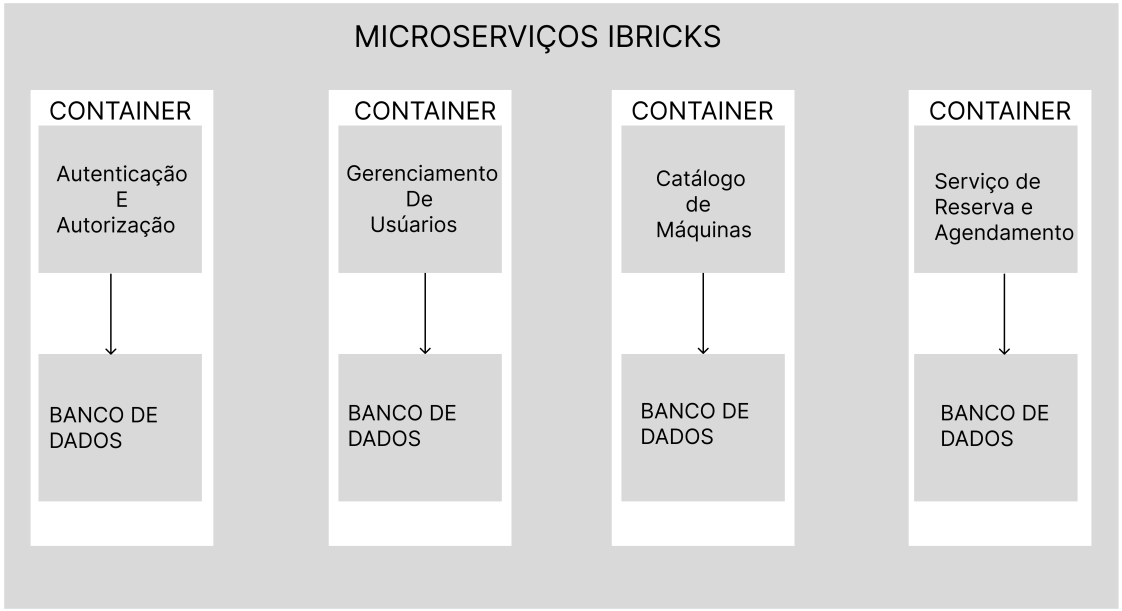

Figura 10. Microsserviços do back-end

5. Aplicando Observabilidade ao Ibricks

Nesta seção, abordaremos as etapas específicas para implementar a observabilidade em nossa arquitetura, utilizando ferramentas como OpenTelemetry [9], Prometheus [10], Grafana Loki [11] e Zipkin [12] como mostra a Figura 11.

Nas subseções seguintes, cada item da figura se encontra descrito.

[9]https://opentelemetry.io/
[10]https://prometheus.io/
[11]https://grafana.com/docs/loki/latest/
[12]https://zipkin.io/

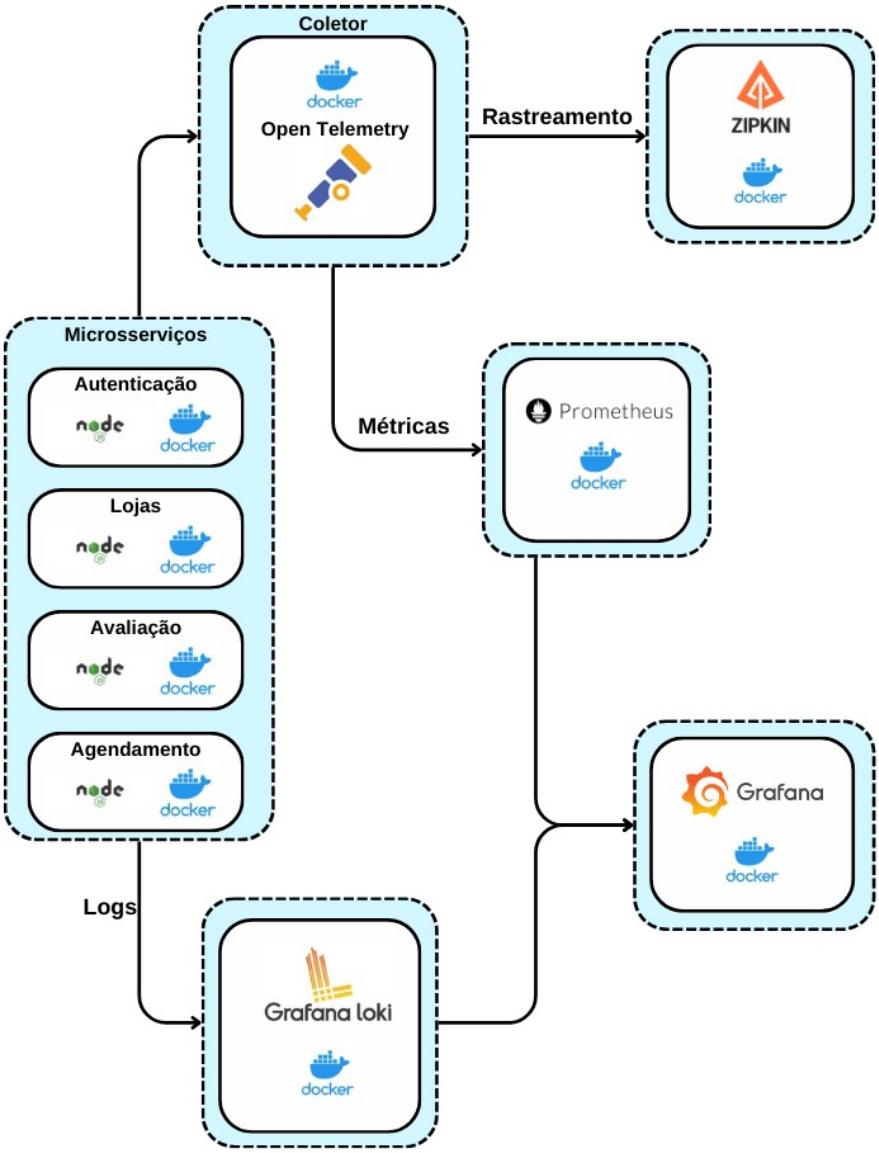

Figura 11. Observabilidade aplicada ao Ibricks

5.1. Microserviços Node.js em Containers Docker

Inicialmente, cada microsserviço da solução é encapsulado em um container Docker. Isso proporciona um ambiente isolado e portátil para cada componente, facilitando a escalabilidade e a implantação consistente em diferentes ambientes de produção.

5.2. OpenTelemetry para Coleta de Dados de Telemetria

O OpenTelemetry desempenha um papel central ao coletar e enviar dados de telemetria para as outras ferramentas do ambiente observável. A configuração adequada do

OpenTelemetry é essencial para uma observabilidade completa do sistema, permitindo a identificação rápida de problemas e a tomada de medidas proativas para garantir a estabilidade e o desempenho da aplicação.

Ao ser integrado em cada microserviço, o OpenTelemetry coleta informações detalhadas sobre chamadas de API, tempos de resposta e fluxos de dados entre os microsserviços. As informações são enviadas ao Prometheus para monitorar métricas em tempo real e ao Zipkin para rastrear solicitações distribuídas.

5.3. Prometheus para Monitoramento de Métricas

Utilizamos o Prometheus para coletar e armazenar métricas em tempo real de cada microserviço. Métricas como uso de CPU, memória e tempos de resposta são monitoradas e podem ser visualizadas por meio do Prometheus, permitindo a identificação de problemas de desempenho.

5.4. Grafana Loki para Agregação de Logs

O Grafana Loki é utilizado para centralizar e analisar os logs de todos os containers Docker. Ou seja, em relação ao IBricks, ele é responsável por registrar os logs de cada microserviço Node.js, enquanto fornece uma capacidade de pesquisa e filtragem que pode facilitar a identificação de problemas e tomada de medidas corretivas contribuindo para a estabilidade do sistema.

Conforme ilustrado na Figura 12, em um aviso vinculado à sua página oficial, o OpenTelemetry não oferece suporte para o envio de logs a microserviços escritos em Node.js, resultando no envio direto dos logs da aplicação para o container do Grafana Loki. Isto significa que, na arquitetura vista na Figura 11, o envio de informações de log não é gerenciado pelo OpenTelemetry por enquanto, embora isto fosse o ideal.

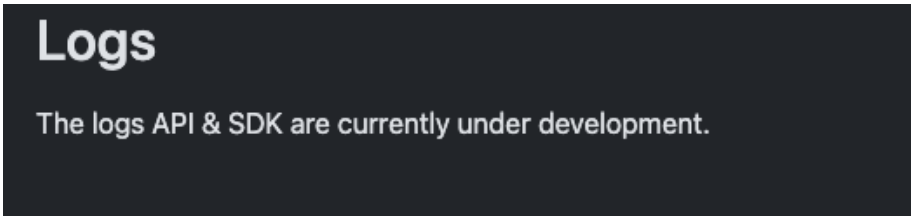

Figura 12. Impossibilidade de envio de logs do Node.js para o OpenTelemetry

5.5. Zipkin para Tracing de Requisições

Zipkin é utilizado para rastrear o fluxo de uma requisição através dos microserviços envolvidos. Com o Zipkin, é possível identificar gargalos de desempenho, analisar a latência de cada chamada e otimizar o fluxo de dados na aplicação, melhorando a experiência do usuário final.

Essas ferramentas em conjunto proporcionam uma visão abrangente e detalhada do ambiente de microserviços, permitindo uma gestão eficiente e proativa da observabilidade para garantir a estabilidade e o desempenho da aplicação.

6. Testando o Ibricks

Com a observabilidade implementada na arquitetura do Ibricks, o próximo passo crucial foi submeter o aplicativo a testes para validar a efetividade das ferramentas e técnicas empregadas. A Tabela 2 contém vídeos demonstrando cenários de utilização do IBricks e de sua infra-estrutura de Observabilidade.

Vídeo	Link	QRCode
Front-end	`https://youtu.be/p9i5KjUr8xk`	
Configuração do back-end	`https://youtu.be/3BEnPACHcCg`	
Demonstrando observabilidade	`https://youtu.be/xzK-TvgkqyE`	

Tabela 2. Testando o IBricks e a sua Observabilidade

7. Trabalhos Correlatos

A proposta de arquitetura baseada em microsserviços e eventos, apresentada por [Lopes 2023], oferece uma perspectiva para a integração entre Ambientes Virtuais de Aprendizagem (AVAs) e instituições educacionais. O autor explorou algumas ferramentas de monitoramento de serviços de um back-end, porém, nota-se uma lacuna no estudo que poderia ser investigada de forma mais aprofundada: a questão da observabilidade utilizando as ferramentas apresentadas. Tal trabalho, desenvolvido para concluir o Curso de Bacharelado em Sistemas de Informação do IFBA, campus Vitória da Conquista, foi uma das motivações para a realização deste.

O trabalho de [Rodrigues et al. 2022] contém uma análise aprofundada sobre a importância da observabilidade em sistemas complexos, destacando a relevância dos logs e métricas na monitorização e na resolução de problemas. No entanto, ocorre uma falta de detalhamento sobre a importância dos traces. Traces são fundamentais para entender o fluxo de execução em ambientes distribuídos e para identificar gargalos, correlacionar eventos e realizar análises mais granulares. Portanto, ao abordar a observabilidade, é crucial considerar não apenas os logs e métricas, mas também traces, para uma visão completa e precisa do sistema em questão, o que foi devidamente considerado para o IBricks.

[Lima 2022] foca no rastreamento e não na ênfase em logs e métricas. Assim, ocorre uma deficiência em reconhecer que a observabilidade abrange uma gama mais ampla de ferramentas e práticas que incluem, também, a análise de logs para compreender o contexto e identificar padrões, bem como a utilização de métricas para medir o desempenho e detectar anomalias. Tais dimensões foram consideradas ao longo deste trabalho.

[Santos Carvalho 2022] explora todos os pilares da observabilidade. Ao destacar a importância do rastreamento, ele enfatiza a necessidade de compreender o comportamento em tempo real dos sistemas. Além disso, ao discutir a análise de logs, o artigo reconhece a importância de registrar eventos e atividades para um entendimento mais profundo do contexto operacional. Da mesma forma, ao abordar métricas, o texto destaca a importância de medir o desempenho e detectar tendências ao longo do tempo.

[Tamburri et al. 2018] também merece reconhecimento por sua visão abrangente e perspicaz sobre a observabilidade e a sua integração eficaz no contexto do padrão de arquitetura MVC (Model-View-Controller). A análise detalhada dos benefícios da observabilidade dentro do MVC é notável, destacando como essa prática pode melhorar significativamente a capacidade de monitoramento e diagnóstico de sistemas complexos.

8. Conclusão

Este trabalho apresentou uma arquitetura para softwares móveis orientada à observabilidade, utilizando o aplicativo Ibricks como caso de estudo. A observabilidade, crucial para a gestão de sistemas complexos, foi incorporada à arquitetura do Ibricks através da implementação de ferramentas como OpenTelemetry, Prometheus, Grafana Loki e Zipkin. Essa abordagem permite um monitoramento em tempo real do desempenho, identificação de gargalos e resolução de problemas de forma proativa, garantindo a qualidade e a estabilidade do aplicativo.

A arquitetura proposta, baseada em microsserviços e containers Docker, tem potencial para fornecer flexibilidade, escalabilidade e manutenibilidade ao sistema. A modularização do front-end em Flutter e a utilização do OpenTelemetry para coletar dados de telemetria permitem uma visão do comportamento do aplicativo. O Prometheus e o Grafana Loki auxiliam na coleta e análise de métricas e logs, enquanto o Zipkin fornece informações sobre o rastreamento de requisições distribuídas.

As ferramentas de observabilidade e a arquitetura modular implementada no Ibricks demonstram a viabilidade técnica da aplicação desses princípios em softwares móveis. Todavia, a jornada de implementação da observabilidade no Ibricks revelou um desafio crucial: a escassez de recursos de aprendizado e uso prático. Documentações incompletas e informações ambíguas sobre a configuração de ferramentas como OpenTelemetry e Grafana Loki dificultaram a integração dessas tecnologias. A ausência de exemplos concretos e guias detalhados para cenários específicos, como o uso do OpenTelemetry em microserviços Node.js, exigiu uma pesquisa extensiva e experimentação para encontrar soluções e superar as lacunas de informação. Tal experiência evidencia a necessidade de iniciativas que promovam a produção e o compartilhamento de conhecimento prático sobre observabilidade.

A necessidade de contornar as lacunas de documentação estimulou a exploração de novas tecnologias. O sucesso da pesquisa e da implementação da observabilidade no Ibricks demonstra a importância da experimentação e da busca por soluções práticas no campo da observabilidade e de arquiteturas de software voltadas a seus princípios.

8.1. Contribuições

Os resultados deste trabalho podem servir como um guia para outros desenvolvedores que buscam implementar observabilidade em seus próprios projetos, fornecendo um modelo

prático e detalhado de como essa tecnologia pode ser utilizada em diferentes cenários. Para tanto, são disponibilizados, na Tabela 3, o código-fonte completo dos dois módulos desenvolvidos, o front-end e o back-end.

Módulo	Link	QRCode
Back-end	`https://abre.ai/kbSc`	
Front-end	`https://abre.ai/kbSk`	

Tabela 3. Código-fonte do IBricks

8.2. Trabalhos Futuros

Como trabalho futuro, há oportunidades para aprofundar a integração do Elastic Search [13] na observabilidade, explorando técnicas avançadas de análise de dados em ambientes distribuídos. O Elasticsearch se destaca na observabilidade por sua capacidade de indexar e pesquisar rapidamente grandes volumes de dados de logs, métricas e traces, permitindo uma análise profunda e em tempo real. Sua busca de texto completo facilita a identificação de padrões e anomalias, enquanto a capacidade de agrupamento e filtragem permite uma análise granular dos dados.

A melhoria contínua na montagem de *dashboards* (ou painéis) na observabilidade pode envolver a adoção de abordagens de design mais inovadoras e personalizadas, visando uma experiência ainda mais intuitiva para os usuários finais. Explorar novos *dashboards* no Grafana é essencial para obter *insights* mais profundos e personalizados sobre dados, além de encontrar novas perspectivas e padrões que podem passar despercebidos com o uso de *dashboards* pré-definidos. A personalização permite visualizar as métricas mais relevantes para suas necessidades, facilitando a identificação de problemas e tendências, além de melhorar a tomada de decisões e a otimização do desempenho de seus sistemas.

Embora a aplicação da observabilidade já esteja integrada à arquitetura proposta, sugerimos que em futuras iterações seja realizado um levantamento detalhado das métricas necessárias para o monitoramento eficaz do sistema. Este levantamento deve considerar como essas métricas podem ser representadas de forma clara e informativa nos dashboards, para garantir uma visualização completa e precisa do desempenho e comportamento do sistema. Essa abordagem permitirá um refinamento contínuo das estratégias de observabilidade e uma melhor adequação às necessidades dos usuários.

A pesquisa e implementação de estratégias para automação de alertas e respostas a anomalias, algo não realizado por este trabalho, também representam áreas de interesse para otimizar a eficiência operacional e a confiabilidade dos sistemas monitorados utilizando a arquitetura aqui apresentada.

[13] https://www.elastic.co/pt/observability

Referências

[Bass et al. 2012] Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice: Software Architect Practice_c3*. Addison-Wesley.

[Fowler et al. 2014] Fowler, M., Lewis, J., and et al. (2014). Microservices. Acesso em: 16 out. 2023.

[Gülcüoğlu et al. 2021] Gülcüoğlu, E., Ustun, A. B., and Seyhan, N. (2021). Comparison of flutter and react native platforms. *İnternet Uygulamaları ve Yönetimi Dergisi*, 12(2):129–143.

[Lima 2022] Lima, M. d. A. (2022). Análise de soluções de rastreamento open source no contexto de aplicações baseadas em microsserviços. B.S. thesis.

[Lopes 2023] Lopes, E. H. S. (2023). Uma arquitetura de microserviços baseada em eventos para sincronização entre ambientes escolares e avas. Bacharelado em Sistemas de Informação.

[Majors et al. 2022] Majors, C., Fong-Jones, L., and Miranda, G. (2022). *Observability Engineering*. "O'Reilly Media, Inc.".

[Meyer 1997] Meyer, B. (1997). *Object-oriented software construction*, volume 2. Prentice hall Englewood Cliffs.

[Newman 2015] Newman, S. (2015). Building microservices: designing fine-grained system. *Oâ€™ Reilly Media, Inc., California*, page 2.

[Pressman 2005] Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave macmillan.

[Renaud 1994] Renaud, P. E. (1994). *Introducao aos sistemas cliente/servidor. Um guia pratico pra profissionais de sistemas*. Infobook.

[Rodrigues et al. 2022] Rodrigues, K. B. C., Bruno, G. Z., Cardoso, K. V., Corrêa, S. L., and Both, C. (2022). Uma investigação empírica sobre observabilidade em sistemas 5g nativos de nuvem. In *Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 252–265. SBC.

[Santos Carvalho 2022] Santos Carvalho, A. P. (2022). Observabilidade e telemetria em arquiteturas de micro-serviços. Master's thesis, Universidade do Porto (Portugal).

[Tamburri et al. 2018] Tamburri, D. A., Bersani, M. M., Mirandola, R., and Pea, G. (2018). Devops service observability by-design: Experimenting with model-view-controller. In *Service-Oriented and Cloud Computing: 7th IFIP WG 2.14 European Conference, ESOCC 2018, Como, Italy, September 12-14, 2018, Proceedings 7*, pages 49–64. Springer.

[Vase 2015] Vase, T. (2015). Advantages of docker. B.S. thesis.

APÊNDICE

Os conceitos adquiridos durante o curso de Bacharelado em Sistemas de Informação (BSI) pelo Instituto Federal de Educação, Ciência e Tecnologia da Bahia contribuíram para a construção do projeto, desde o levantamento teórico até a aplicação prática. A Figura 13 demonstra todas as disciplinas que contemplam a ementa do curso de BSI e que se relacionam de forma direta e indireta com este trabalho.

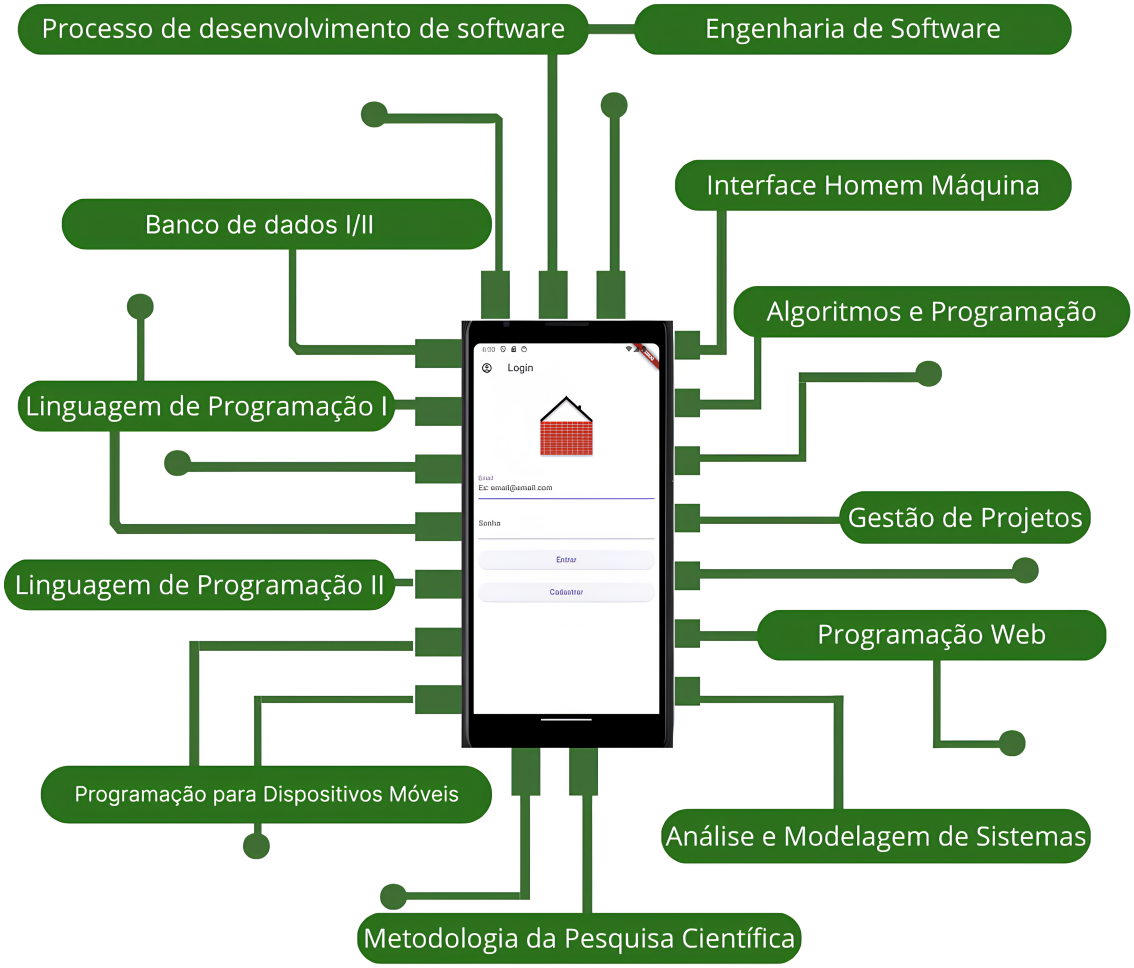

Figura 13. Disciplinas do Curso de BSI relacionadas à realização deste trabalho