UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

MS-TRICK: ARQUITETURA DE MICROSERVICES APLICADA

ARIEL RAI RODRIGUES

BLUMENAU
2018

ARIEL RAI RODRIGUES

MS-TRICK: ARQUITETURA DE MICROSERVICES APLICADA

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof.ª Simone Erbs da Costa, Especialista – Orientadora

Eduardo Hildebrandt, Especialista – Coorientador

BLUMENAU
2018

MS-TRICK ARQUITETURA DE MICROSERVICES

APLICADA

Por

ARIEL RAI RODRIGUES

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof.ª Simone Erbs da Costa, Especialista – Orientadora, FURB
Membro:	Profº. Mauro Marcelo Mattos, Doutor – FURB
Membro:	Profº. Roberto Heinzle, Doutor – FURB

Blumenau, 12 de julho de 2018

Dedico este trabalho a minha família, em especial minha mãe que nem sempre pode me ajudar da forma que queria, porém sempre me ajudou da forma que eu precisava.

AGRADECIMENTOS

A Deus...

À minha família...

Aos meus amigos...

A minha orientadora...

Ao meu coorientador do pré-projeto...

A empresa pública informática...

There is nothing permanent except change.

—Heraclitus

RESUMO

Este trabalho apresenta uma arquitetura de *microservices*. O objetivo principal deste trabalho é reconstruir partes de uma aplicação existente de maneira escalável, demonstrando os resultados obtidos com esta alteração. De maneira complementar foi desenvolvido um aplicativo móvel para visualização da arquitetura. Os *microservices* desenvolvidos para contemplar a arquitetura foram desenvolvidos em linguagem de programação Java, mais especificamente utilizando Spring Boot. Para armazenamento de dados foi utilizado o banco de dados Mysql. O aplicativo móvel foi desenvolvido utilizando Ionic, *framework* responsável por gerar aplicativos Android e IOS. A ideia para o desenvolvimento deste trabalho foi desenvolvida com a vivência no sistema atual referido, com o crescimento da empresa no mercado Brasileiro demonstrou-se visível que uma arquitetura como a apresentada neste trabalho é viável. O levantamento de informações foi realizado por meio de inspeção do código em conjunto com o coorientador do projeto. Foram utilizados diagramas da Unified Modeling Language (UML) para representar estruturas e fluxos desenvolvidos, bem como foi realizado uma avaliação de usabilidade junto aos especialistas das áreas na empresa para verificar a eficiência e visibilidade do usuário de Tecnologia de Informação (TI) sobre a arquitetura. A implementação do trabalho foi realizada utilizando as ferramentas RedHat Development Suite e Atom, todo seu código fonte foi armazenado no website Github. Por fim, mas não menos importante, foram levantadas melhorias futuras e extensões para este trabalho; bem como conclui-se as vantagens oferecidas pela arquitetura de *microservices*.

Palavras-chave: *Microservices*. Arquitetura. Escalável. Aplicativo. Spring Boot.

ABSTRACT

This project presents a *microservices* architecture. The main goal of this project is rebuilding parts of an existing application in a scalable way, showing what was achieved with the changes. Additionally, a mobile application was developed to show the *microservices* themselves. The *microservices* were developed using Java as the programming language, being more specific: Spring Boot. For storage the database Mysql was used. The mobile application was built using Ionic, a framework used to generate Android and IOS apps. The idea was generated experiencing the current system described, due to the growing of the company in the Brazilian market, an architecture like the presented one turns to be viable. The needed information for this project was collected by inspecting the application code together with the joint supervisor. UML diagrams were used to present the developed structures and flows, as well as a usability evaluation, which was applied to the company specialists to check how efficient and viable the architecture can be for IT users. For the implementation of this Project the tools RedHat Development Suite and Atom were used, all the source code is stored on the website Github. At last, but not least, extensions and future improvements were raised, as well as the advantages of using the *microservices* arquitecture.

Key-words: *Microservices*. Architecture. Scalable. App. Spring Boot.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

AWS – Amazon Web Services

CSS – Cascading Style Sheets

DCU – Diagrama de Casos de Uso

EAR – Enterprise Application Archive

HTML – HyperText Markup Language

HTTP - Hypertext Transfer Protocol

IDE - Ambiente Integrado de desenvolvimento

M3C-URUCAg - Relationship of M3C with User Requirements and Usability and Communicability Assessment in groupware

MER – Modelo Entidade Relacionamento

REST - Representational State Transfer

RF – Requisito Funcional

RNF – Requisito Não Funcional

SaaS – Software como Serviço

TI – Tecnologia de Informação

UC – Caso de Uso

UI – User Interface

UML – Unified Modeling Language

URL – Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

A velocidade com que os sistemas crescem exige que as empresas constantemente evoluam em suas arquiteturas de software, a fim de suportar o fluxo de dados e informações que trafegam em massa pelas redes corporativas e na internet (SILVA, 2015). O mesmo autor (SILVA, 2015) enfrentou esses desafios fazendo uso de arquiteturas mais modernas e sustentáveis, possibilitando que as soluções sejam reutilizáveis e aumentem a produtividade. Gamma (1995, p. 17) ainda sugere que o projeto deve ser apropriado para o problema que visa solucionar, contudo, deve ser generalizável para resolver futuras especificações, minimizando a quantidade de re-projetos.

Nesse cenário, estão as empresas desenvolvedoras de projetos de software, que a fim de obterem vantagem competitiva, buscam continuamente a melhoria de seus sistemas (SILVA, 2015). Apesar de existirem muitos fatores para o sucesso de um sistema, desempenho se torna um dos pontos principais (SOARES, 2012). Segundo Soares (2012) usuários buscam velocidade, e a escalabilidade se torna um dos pontos chaves para se manter um ambiente adequado para utilização.

Gamma (1995) observa que os problemas de escalabilidade são causados por rápidas mudanças e pelo crescimento demasiado dos sistemas, além de causarem sérios problemas de performance, disponibilidade etc. De acordo com Lima (2015), grande parte desses problemas são resolvidos ou melhorados com a arquitetura de *microservices*, uma arquitetura modular e escalável. Além disso, a arquitetura traz benefícios de tolerância a falhas e facilidade na manutenção da aplicação (SILVA, 2015). Entretanto, a arquitetura por si só não traz benefício para as aplicações, é necessário que ela seja aplicada em um contexto condizente com as vantagens trazidas por ela, assim como fazer uso de padrões e projetos de sucesso existentes.

A arquitetura de *microservices* pode ser aplicada em diferentes contextos e aplicações (LIMA, 2015). Contudo, um aplicativo para auxiliar no controle e visualização de métricas pode contribuir com a manutenção da arquitetura de software. Desta forma aplicativos móveis podem ser integrados a essa arquitetura para facilitar o uso e possibilitar que usuários menos experientes possam controlar as aplicações de maneira simplificada e remota.

Diante deste cenário, este trabalho propõe uma arquitetura de *microservices* aplicada ao caso de estudo na empresa Pública Tecnologia. Gerando como objetivo apoiar usuários menos experientes na visualização dessa arquitetura por meio de um aplicativo móvel (app), que foi desenvolvido e integrado com a arquitetura de *microservices* proposta.

1.1 OBJETIVOS

Esse projeto tem como objetivo realizar um estudo de caso da aplicação da arquitetura de *microservices* na empresa Pública Tecnologia. Os objetivos específicos do projeto são:

a) dividir os módulos file-server e portal da transparência da aplicação para criar *microservices*;
b) utilizar de integração contínua para facilitação na liberação de recursos;
c) implementar uma interface móvel para manutenção e visualização dos *microservices*.

1.2 ESTRUTURA

Este trabalho foi desenvolvido em quatro capítulos. O primeiro capítulo apresenta a introdução do trabalho desenvolvido, sua justificativa, os objetivos do trabalho e estrutura do trabalho. No segundo capítulo são abordados os conceitos e fundamentos mais relevantes para o desenvolvimento deste trabalho; o sistema atual é apresentado, bem como são explorados os conceitos de arquitetura de software, recursos tecnológicos, usabilidade da interface de aplicativos móveis e os trabalhos correlatos. O terceiro capítulo traz o desenvolvimento do trabalho, demonstrando os requisitos necessários para elaboração do projeto, diagramação, implementação e execução da arquitetura. Por fim, no quarto capítulo são apresentadas as conclusões, bem como são sugeridas as extensões para serem implementadas no futuro.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os fundamentos e conceitos mais relevantes para o desenvolvimento deste trabalho, estando estruturado da seguinte forma: na seção 2.1 é apresentado o sistema atual; a seção 2.2 traz os conceitos sobre arquitetura de software; a seção 2.3 aborda os recursos tecnológicos necessários; a seção 2.4 apresenta conceitos de usabilidade para aplicativos móveis; e por último, na seção 2.5 os trabalhos correlatos ao desenvolvido são apresentados.

2.1 DESCRIÇÃO DO SISTEMA ATUAL

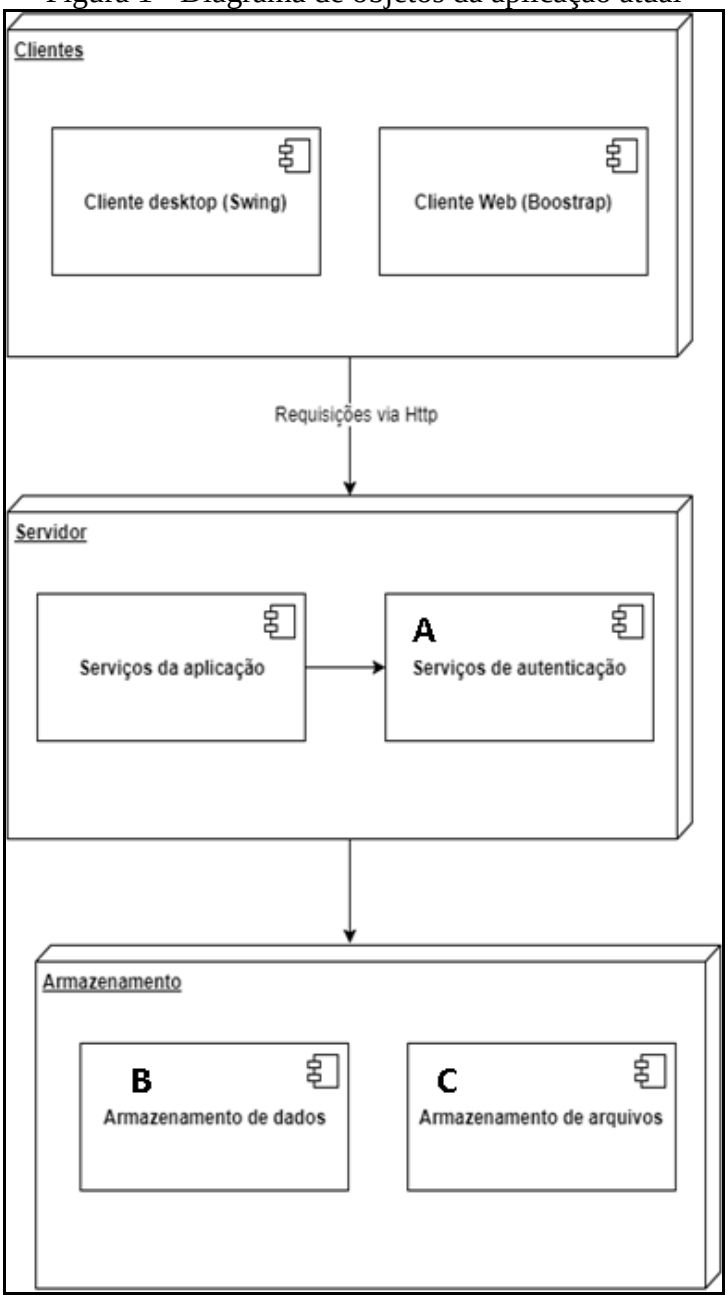
A Empresa Pública Tecnologia possui um sistema de administração pública para municípios no território nacional brasileiro. Esse sistema foca em interatividade, parametrização e rotinas específicas para atender as necessidades dos clientes. Os principais módulos disponibilizados pelo sistema são:

a) planejamento: permite realizar o plano de governo necessário para gestão do município. Este módulo é integrado com a lei de diretrizes orçamentárias, facilitando o acompanhamento de ações do governo;
b) compras: permite o controle de compra de materiais e serviços dentro da entidade pública, mantendo a organização e o processo de aquisição de forma correta e executado pela lei;
c) contabilidade: controla a execução orçamentária, financeira e patrimonial da entidade pública, com regras de negócio, possibilitando que a apreciação seja realizada de maneira integrada com os outros módulos da aplicação;
d) portal da transparência: disponibiliza informações gerais sobre gastos, custos, compras etc., de um determinado município;
e) file-server: módulo praticamente invisível ao usuário, permitindo a realização de *upload* e *download* de arquivos pelos usuários. Qualquer relatório, log do sistema, ou qualquer tipo de arquivo gerado pela aplicação utiliza desse módulo.

O sistema possui outros módulos além dos acima descritos, que são implementados utilizando Interface de Usuário (IU) baseado no *framework* Swing. Ademais, utilizam o método cliente-servidor para comunicação.

Atualmente, encontram-se em desenvolvimento novos módulos web, sendo a interface construída em Bootstrap[1], biblioteca de componentes visuais disponibilizados pela rede social Twitter. Além dos itens de cliente e servidor apresentados, a Figura 1 traz os outros elementos para o que compõem a estrutura da aplicação. Os itens do diagrama em conjunto representam as partes integrantes do sistema estudado.

Figura 1 - Diagrama de objetos da aplicação atual



Fonte: elaborado pelo autor.

[1] Bootstrap: Disponível em: <https://getbootstrap.com/>. Acesso em: 19 jun. 2018.

Na Figura 1 é possível visualizar uma parte desta arquitetura pelo Diagrama de objetos da aplicação atual, dentre eles: serviços de autenticação (letra A): responsáveis por todo o controle de autenticação de uso, apesar desse ser parte integrante do servidor da aplicação, ele pode ser mantido separado no diagrama, pois já possui baixo acoplamento; armazenamento de dados (letra B): responsáveis por armazenar dados da aplicação, todos as unidades diferentes de arquivos são mantidas neste componente; armazenamento de arquivos (letra C): responsável por salvar todo tipo de arquivos, relatórios e logs da aplicação. Este é representado como uma parte individual da arquitetura, sendo o armazenamento externo, porém o código para realizar os procedimentos de armazenamento de arquivos se encontra acoplado ao Monolito.

O mau funcionamento destes componentes pode implicar em uma parada total, representando um ponto de falha na aplicação, não sendo possível provisionar recursos de forma separada em caso de sobrecarga de uso. Ainda que representados separadamente no diagrama da estrutura na Figura 1, o código fonte é único para todos os integrantes da arquitetura. Portanto, é necessário que a compilação da aplicação seja completa, apesar de demorada. Ressalta-se que a liberação de correções em apenas algumas das partes para clientes não é possível, sendo necessário realiza-la de maneira completa. Além disso, o sistema atual utiliza uma arquitetura monolítica, havendo diversos clientes que consomem o mesmo serviço em *cloud*, o que pode ocasionar alguns dos seguintes problemas para o desenvolvimento e manutenção da aplicação:

a) difícil provisionamento de recursos: impossibilidade de liberação de recursos para módulos ou serviços específicos;
b) dificuldade de compilação e desacoplamento de diferentes módulos: adicionando complexidade à compilação da aplicação;
c) demora em tempo de compilação de toda a aplicação: é utilizada a Ferramenta Maven[2] para gerenciamento das dependências e compilação. A lentidão ao realizar o processo de compilação ocorre devido à grande quantidade de módulos;
d) impossibilidade de liberação de correções independentes por módulos: por não se utilizar de conceitos de integração contínua.
e) Crescimento apenas vertical dos recursos da aplicação: necessidade de máquinas potentes, adicionando memória, processamento e armazenamento à mesma máquina (DEVMEDIA, 2014).

Isso se torna necessário à medida que a quantidade de usuários cresce. Esses problemas podem ser amenizados ou até mesmo resolvidos ao fazer uso da arquitetura de *microservices* (DRAGONI et al., 2016). Essa propicia suporte ou melhoria para esses problemas, em consonância, trazem dificuldades e complexidade ao projeto. Os trabalhos correlatos se destacam por trazerem tanto os benefícios como os obstáculos, similares aos encontrados na aplicação atual.

2.2 ARQUITETURA DE SOFTWARE

Arquitetura de software é a especificação de componentes de uma aplicação e como eles se relacionam (LEITE, 2000). O autor (LEITE, 2000) também coloca que essa arquitetura permite especificar, visualizar e documentar componentes de maneira abstrata, indiferente de sua linguagem e meio para execução. Pode-se dizer, que a arquitetura é o que dita os princípios básicos para o sucesso da aplicação.

Lima (2015) exemplifica uma arquitetura monolítica como uma aplicação contendo o servidor sendo uma aplicação única, desta forma, considerada um Monolito e sua arquitetura monolítica. Basicamente, uma arquitetura monolítica é aquela que seus módulos não podem ser executados de maneira individual (DRAGONI et al., 2016), tornando-se um obstáculo para sistemas distribuídos e dificultando a entrega ágil das aplicações (SILVA, 2015).

Para Dragoni et al. (2016), um *microservice* é uma parte independente de uma aplicação, capaz de se comunicar com outras partes via mensagens, possuindo infraestrutura, ferramentas de persistência e módulos isolados (DRAGONI et al., 2016). Desta forma, a arquitetura de *microservices* é formada por várias partes da aplicação (*microservices)*, sendo a soma desses segmentos trabalhando em conjunto uma aplicação distribuída. Para a arquitetura de *microservices* Dragoni et al. (2016) cita os seguintes princípios:

a) contexto limitado;
b) funcionalidades relacionadas são mantidas no *microservice*;
c) tamanho: o contexto das partes deve ser mantido pequeno, ou seja, caso uma parte seja muito grande ela deve ser dividida em partes menores;
d) independência: um *microservice* deve conhecer apenas o necessário para se comunicar com o restante da aplicação. Além disso, um *microservice* deve conseguir executar-se por si.

---

[2] Ferramenta Maven: Disponível em: <https://maven.apache.org/>. Acesso em: 19 jun. 2018.

A escalabilidade é a capacidade que uma aplicação tem de crescer conforme necessidade de utilização. Segundo Dragoni et al. (2016), ela é necessária por motivos de performance, enquanto a quantidade de usuários cresce, a carga de utilização de recursos também cresce significativamente. Pode-se dizer que o conceito de escalabilidade diz respeito principalmente à alocação de recursos de infraestrutura. Desta forma, se uma pequena parte de uma aplicação é o gargalo de performance, caso esteja contida em uma aplicação monolítica, a aplicação como um todo precisará ser escalada (NAMIOT; SNEPS-SNEPPE, 2014).

Em uma arquitetura de *microservices*, cada parte da arquitetura é independente, sendo possível a replicação para cada uma destas partes a medida que se fizer necessário, ou seja, escalabilidade horizontal. Segundo Dragoni et al. (2016), dentre outras características de *microservices*, a disponibilidade é a habilidade de resposta da aplicação, ocorrendo pela facilidade de replicação em diversas geolocalizações. Outro ponto importante da arquitetura é que o tempo de indisponibilidade por atualização, é reduzido para o tempo de atualização de um *microservice* (DRAGONI et al., 2016).

2.3 RECURSOS TECNOLÓGICOS

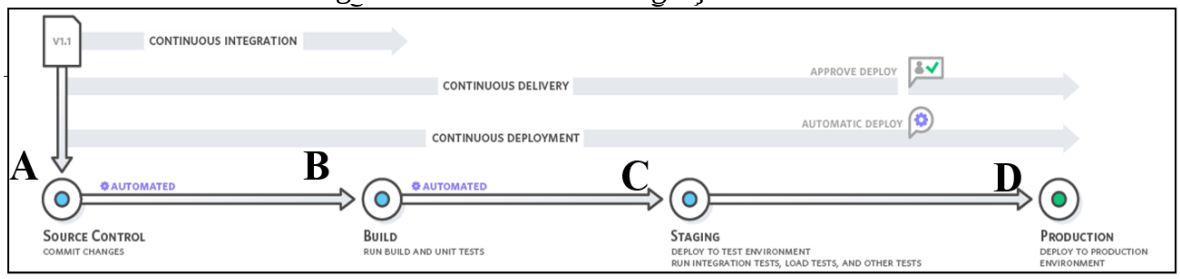
Para o desenvolvimento do projeto foi necessário abordar determinados recursos tecnológicos, como servidores de aplicação e integração contínua. Segundo Soares (2012), um servidor de aplicação disponibiliza o ambiente necessário de software e infraestrutura para execução e instalação de uma aplicação. Servidores de aplicação têm como objetivo abstração de complexidades computacionais para disponibilização das aplicações (SOARES, 2012). Enquanto para Sobral (2002), servidores de aplicação são responsáveis por manter uma lista de serviços disponíveis para acesso via rede.

Para Soares (2012), os servidores de aplicação permitem que os desenvolvedores foquem na solução a ser desenvolvida e nas questões relacionadas ao negócio da empresa. Características computacionais, assim como questões de tolerância a falhas, são facilitadas pela utilização dos servidores de aplicação. Outra característica, diz respeito ao balanceamento de carga, podendo ser alcançada pela distribuição das requisições. Soares (2012) destaca ainda o gerenciamento de componentes realizado por alguma ferramenta como o gerenciamento de transações, a integridade de transações e o console de gerenciamento, disponibilizando uma interface para visualização e configuração.

A aplicação proposta por este trabalho utiliza a Ferramenta Spring Boot[3] para criação dos *microservices*. Segundo Spring (2017), Spring Boot é uma ferramenta pronta para produção, responsável por disponibilizar um ambiente de fácil configuração para desenvolvedores de software. Mesmo considerado um servidor de aplicação por si só, a ferramenta utiliza de outros servidores para execução, sendo estes: Tomcat, Jetty ou Undertow (SPRING, 2017). Outro recurso tecnológico utilizado diz respeito ao trabalho de integração contínua que está ligado ao conceito de DevOps.

O processo de integração contínua, geralmente, faz com que os desenvolvedores mantenham os códigos fontes em um mesmo repositório, possibilitando corrigir falhas e liberar a aplicação de forma rápida, reduzindo o tempo de validação (AMAZON AWS, 2017b). O processo de integração contínua divide-se em quatro etapas que podem ser melhores compreendidas pela Figura 2. A primeira etapa diz respeito ao *source control*, como ilustrado na letra A, referindo-se ao controle e liberação dos códigos fontes da aplicação. A segunda etapa é a compilação da aplicação, *build*, como pode ser visualizado pela letra B. A terceira etapa diz respeito à validação dos códigos fontes da aplicação, *staging*, demonstrado na letra C; e a quarta e última etapa é apresentada pela letra D, designa à liberação ao cliente final, ou seja, o *production*.

Figura 2 - Processo de integração contínua



Fonte: Amazon AWS (2017b).

As etapas ilustradas na Figura 2 ocorrem de forma integrada e automática. Para Amazon AWS (2017a), esse processo além de ser bem definido traz produtividade ao liberar os desenvolvedores de tarefas manuais e busca evitar falhas nas versões finais disponibilizadas aos clientes. Dessa forma, permitem realizar uma investigação de forma rápida e eficaz na detecção dos problemas, assim como possibilitam que a distribuição da aplicação seja realizada de forma rápida e fácil (AMAZON AWS, 2017b).

[3] Ferramenta Spring Boot: Disponível em: <https://spring.io/projects/spring-boot>. Acesso em: 19 jun. 2018.

2.4 USABILIDADE DA INTERFACE DE APLICATIVOS MÓVEIS

O método de avaliação heurística é utilizado em contextos que visam identificar problemas relacionados com a usabilidade, baseando-se em melhores práticas definidas por um conjunto de heurísticas (PRATES; BARBOSA, 2003). O método pode ser realizado por especialistas oriundos do mercado (profissionais especialistas da área) ou da academia, sendo a avaliação realizada por três a cinco especialistas. Para melhor entendimento do procedimento necessário para realização do método, o Quadro 1 traz uma síntese das etapas necessárias; o Quadro 2 mostra o conjunto básico das heurísticas de Nielsen; e o Quadro 3 traz a escala de severidade.

Quadro 1 – Síntese das etapas da avaliação heurística

Etapa	Sessão	Observação	Passos
1	Avaliação individual e por especialista em sessões curtas (1 a 2 horas).	Sessões precisam ser individuais para que um avaliador não influencie a opinião dos outros. Em cada sessão de avaliação, o avaliador deve percorrer a interface mais de uma vez para inspecionar os diferentes elementos de interface e compará-los com a lista de heurísticas de usabilidade.	1.1. julga a conformidade da interface com um determinado conjunto de princípios (“heurísticas”) de usabilidade; 1.2. anota os problemas encontrados e sua localização; 1.3. julga a gravidade destes problemas; 1.4. gera um relatório individual com o resultado de sua avaliação e comentários adicionais.
2	Consolidação da avaliação dos especialistas.	Os avaliadores possuem acesso aos relatórios individuais de todos os avaliadores, podendo realizar considerações referente as considerações realizadas pelos demais avaliadores. O artefato resultante dessa etapa é um relatório unificado e consolidado sobre todos os problemas encontrados.	2.1 novo julgamento sobre o conjunto global dos problemas encontrados 2.2. relatório unificado de problemas de usabilidade
3	Seleção dos problemas a serem corrigidos.	Essa etapa é realizada junto ao cliente ou ao gerente de projeto. É realizado uma análise de custo/benefício das correções aos problemas encontrados na etapa anterior, levando em consideração a gravidade, os prazos e o orçamento do projeto, bem como a capacitação da equipe de desenvolvimento.	

Fonte: Costa (2018a) adaptado de Prates e Barbosa (2003).

Quadro 2 – Síntese do conjunto básico das heurísticas de Nielsen

Nro	Heurística	Característica
H1	Visibilidade do estado do sistema	Manter os usuários informados sobre o que está acontecendo, por meio de feedback adequado e no tempo certo.
H2	Correspondência entre o sistema e o mundo real	Utilizar conceitos, vocabulário e processos familiares aos usuários.
H3	Controle e liberdade do usuário	Fornecer alternativas e "saídas de emergência"; possibilidades de *undo* e *redo.*
H4	Consistência e padronização	Palavras, situações e ações semelhantes devem significar conceitos ou operações semelhantes; caso haja convenções para o ambiente ou plataforma escolhidos, estas devem ser obedecidas.
H5	Prevenção de erro	Evitar que o erro aconteça, informando o usuário sobre as consequências de suas ações ou, se possível, impedindo ações que levariam a uma situação de erro.
H6	Ajuda aos usuários para reconhecerem, diagnosticarem e se recuperarem de erros	Utilizar mensagens de erro em linguagem simples, sem códigos, indicando precisamente o problema e sugerindo de forma construtiva um caminho remediador.
H7	Reconhecimento em vez de memorização	Tornar os objetos, ações e opções visíveis e compreensíveis.
H8	Flexibilidade e eficiência de uso	Oferecer aceleradores e caminhos alternativos para uma mesma tarefa; permita que os usuários customizem ações frequentes.
H9	Design estético e minimalista	Evitar porções de informação irrelevantes. Cada unidade extra de informação em um diálogo compete com as unidades de informação relevantes e reduz sua visibilidade relativa.
H10	Ajuda e documentação	Facilitar as buscas, focadas no domínio e na tarefa do usuário, e devem listar passos concretos a serem efetuados para atingir seus objetivos.

Fonte: Costa (2018a) adaptado de Nielsen (1994).

Quadro 3 - Escala de Severidade

Nro	Severidade
0	Não é considerado um problema de usabilidade.
1	Problema apenas visual, não necessita correção imediata.
2	Representa um pequeno problema, a correção deve possuir baixa prioridade.
3	Problema maior de usabilidade, correção deve ser tratada como alta prioridade.
4	Catástrofe de usabilidade, correção deve ser realizada antes da liberação do aplicativo.

Fonte: Costa (2018a) adaptado de Moulin (2013).

Segundo Moulin (2013), ao violar uma heurística, é necessário selecionar a severidade da heurística violada, em uma escala de 0 – 4, bem como após a seleção de severidade é necessário definir o local do problema, ou seja, qual interface, podendo ser em uma única localidade, em duas ou mais, ou na estrutura geral de forma sistemática. Além disso, será

necessária a adoção de padrões específicos para aplicativos móveis (COSTA, 2018a; NIELSEN, 1994; MOULIN, 2013; PRATES; BARBOSA, 2003).

Segundo Griffiths (2015), uma aplicação deve seguir os seguintes princípios: **adote**, se refere a remover obstáculos de utilização e adesão ao software; **use**, o aplicativo deve facilitar a utilização de suas funções principais. Se a busca é uma função principal do software, essa deve ser destacada; **facilidade a transação**, cada etapa do software deve ser destacada e com mensagens suficientes para o usuário saber que finalizou a etapa; **retorne,** diz respeito ao foco na utilidade do sistema. Encantamento gera fidelidade e satisfação do usuário; **higiene da usabilidade**, eliminar pontos do sistema que podem promover interrupção ou forçar usuários a pensar sobre coisas que deveriam ser simples (GRIFFITHS, 2015).

Em uma interface móvel ainda existem diversos pontos que precisam ser evitados para o sucesso de uma aplicação (NEIL, 2012), como: elementos de User Interface (UI) de outras plataformas não devem ser copiados, cada plataforma possui seus elementos que devem ser respeitados; links não carecem de ser sublinhados e preferencialmente devem ser evitados, caso seja feita a utilização, não devem redirecionar ao navegador. Griffiths (2015) observa que não é viável pedir a avaliação do aplicativo logo após o download, visto que os usuários precisam utilizá-lo para formar uma opinião e realizar uma avaliação. Por último, Costa (2018b) em seu trabalho maior complementa a observação de Griffiths (2015), que a avaliação não deve ser realizada logo após o download e sim logo após o uso do aplicativo.

2.5 TRABALHOS CORRELATOS

Esta seção apresenta três trabalhos correlatos ao desenvolvido. Os trabalhos descritos referem-se a *microservices* aplicados em diferentes aplicações. A subseção 2.5.1 detalha o caso da arquitetura de *microservices* do serviço de *streaming***,** que significa: transmissão continua de dados pela internet (COUTINHO, 2013) de conteúdo de vídeo (NETFLIX, 2017). A subseção 2.5.2 traz a arquitetura de *microservices* das aplicações da empresa Softplan, que atende segmentos como gestão de obras públicas, gestão de documentos digitais e administração financeira (SILVA, 2015). A subseção 2.5.3 apresenta o estudo de São Miguel e Farina (2016), trazendo a separação de um módulo da aplicação para criação de um *microservice*. Por fim, a subseção 2.5.4 traz a comparação entre os trabalhos relacionados.

2.5.1 Netflix

Netflix é um serviço pago de *streaming* de vídeo que disponibiliza ao usuário grande quantidade de séries, filmes e documentários (NETFLIX, 2017). O serviço chega perto dos

100 milhões de usuários pelo mundo todo (FARINACCIO, 2017). A Netflix possui uma arquitetura escalável e flexível, para que todos possam ser atendidos de maneira satisfatória e tenham uma boa experiência ao fazer uso da aplicação (TONSE, 2014).

Inicialmente, em 2008, sua arquitetura era monolítica, mantendo todos os serviços juntos em uma única webApp (TONSE, 2014), aplicação que essencialmente tem suas funções executadas pela rede (NOURIE, 2006). Em 2012 a arquitetura foi migrada para a *cloud* da Amazon Web Services (AWS), serviço disponibilizado pela empresa especializada em aplicações para arquitetura de *cloud* e dividida em diversos *microservices* distribuídos (TONSE, 2014).

Nesta nova arquitetura a aplicação foi desenvolvida em sua maior parte em linguagem de programação Java, utilizando *frameworks open-source* como: Netflix Eureka[4] (plataforma responsável por descobrir serviços na rede); Netflix Hystrix[5] (isolamento de latência e tolerância a falhas); Netflix Ribbon[6] (comunicação entre *microservices*) (NETFLIX, 2016). Outra prática utilizada na arquitetura da empresa é a distribuição de múltiplas instâncias dos *microservices* da aplicação pelo mundo, esta é realizada de forma variada conforme a necessidade. Esta prática faz com que o tempo de resposta em diferentes regiões seja semelhante, não necessitando de acesso a um *microservice* em uma região distante, considerando a localização do usuário (TONSE, 2014).

Em decorrência desta arquitetura adotada, segundo Tonse (2014), a aplicação passou a ter mais disponibilidade pois conforme necessidade é possível aumentar o número de instâncias da mesma aplicação executada, fazendo com que nenhuma dessas instâncias seja sobrecarregada. A velocidade de desenvolvimento de novas *features*, em consonância com as correções que passaram a ser independentes por serviço. A empresa já fazia uso de alguns métodos e técnicas existentes da arquitetura de *microservices*, todavia ao adiciona-los por completo alcançou uma série de vantagens, salientam-se maior disponibilidade, agilidade de entrega de módulos independentes e DevOps (TONSE, 2014).

Outro ponto a se destacar, é que a Netflix começou a adotar os conceitos de DevOps, um conjunto de práticas e ferramentas as quais permitem que as empresas liberem suas aplicações com mais facilidade e velocidade (AMAZON AWS, 2017a). Segundo Tonse (2014), a adoção dessa arquitetura trouxe novos desafios à aplicação, em função do aumento

[4] Netflix Eureka: Disponível em: <https://github.com/Netflix/eureka>. Acesso em: 19 jun. 2018.
[5] Netflix Hystrix: Disponível em: <https://github.com/Netflix/hystrix>. Acesso em: 19 jun. 2018.
[6] Netflix Ribbon: Disponível em: <https://github.com/Netflix/Ribbon>. Acesso em: 19 jun. 2018.

da complexidade dos sistemas distribuídos, destacando que os conceitos de DevOps tornaram-se necessários para a manutenção da aplicação.

O ecossistema da aplicação necessita de implementação de testes, bem como o tráfego de rede teve um aumento significativo (TONSE, 2014). Ainda para esse autor, *microservices* não trazem disponibilidade instantânea, visto que qualquer dependência da aplicação pode deixar o site indisponível. Fato esse, acrescido aos outros desafios encontrados no desenvolvimento de um sistema nessa arquitetura levam os desenvolvedores da aplicação a recomendarem um meio de tolerância a falhas (TONSE, 2014).

Na literatura, Netflix (2016) é amplamente conhecida e referenciada por se tratar de um caso de sucesso, sendo que o desenvolvimento desta aplicação apresentou significativa contribuição para a comunidade da computação (TONSE, 2014). Destacan ainda a utilização de tecnologias *open-source*, desenvolvidas e disponibilizadas pela empresa, bem como o uso do conceito de DevOps contribuiu para a melhoria da arquitetura de *microservices* como um todo.

2.5.2 Adoção da arquitetura *microservices* em aplicações corporativas

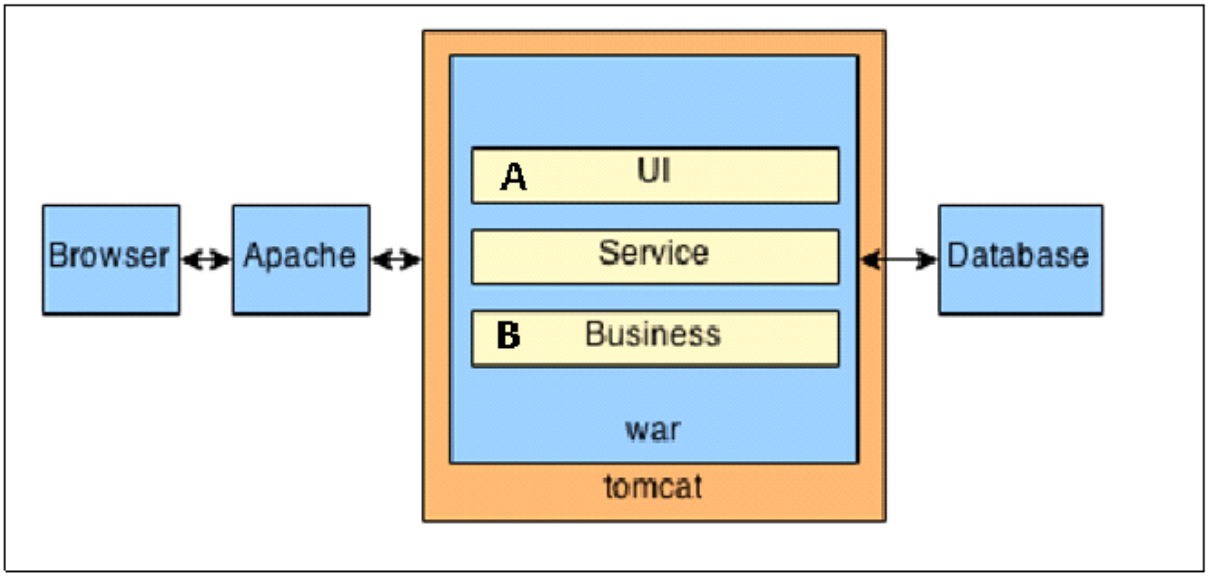
Silva (2015) aplica seu estudo na empresa Softplan que atua no segmento de gestão, no mercado brasileiro, desde 1990, verificando que uma arquitetura monolítica traz obstáculos às grandes aplicações corporativas. Dentre eles, Silva (2015) destaca que a quantidade e complexidade de grandes aplicações dificultam a inserção de novas funcionalidades. Outro ponto destacado é que o amplo acoplamento contribui para geração de código ruim ao longo do tempo, afetando assim atualizações e deteriorando a aplicação ao longo do tempo. Ademais, o agrupamento de todos os componentes dificulta a implantação e atualização do método. (SILVA, 2015, p. 5).

A aplicação era implementada inteiramente na linguagem de programação Java web dentro de uma arquitetura monolítica em um único Enterprise Application Archive (EAR), formato de arquivo para liberação da aplicação. O primeiro passo para adoção da arquitetura de *microservices* foi desenvolver as novas funcionalidades, porém, considerando que grande parte da cartela de clientes ainda não utilizava Softwares como Serviço (SaaS) (SILVA, 2015). Neste contexto, o aumento na complexidade da arquitetura pode impactar no tempo de atualização e manutenção da infraestrutura desses usuários.

A primeira etapa se refere à refatoração da camada de apresentação da aplicação, a User Interface (UI) (letra A da Figura 3), para diminuição do acoplamento. De acordo com Silva (2015), o forte acoplamento da camada de UI era uma dificuldade em virtude da

utilização de um *framework* próprio de componentes visuais da empresa, possuindo acentuada ligação com a camada de negócio (letra B da Figura 3). A segunda etapa se refere ao levantamento de pontos críticos, pois eles poderiam representar uma falha única. (SILVA, 2015).

Figura 3 - Arquitetura monolítica

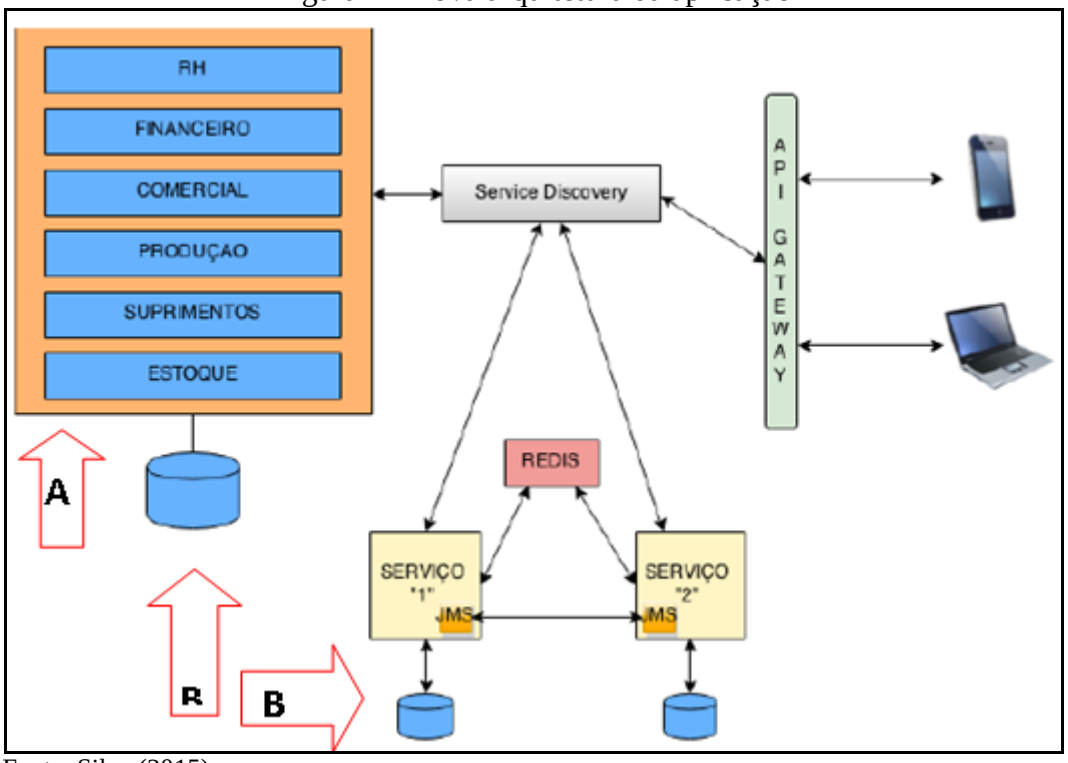


Fonte: Richardson (2014).

Silva (2015) observa que a arquitetura de *microservices* é similar a arquitetura de equipes divididas por módulos da aplicação utilizada pela Softplan. Apesar disso, devido à existência de muitas operações síncronas executadas entre os módulos do software, o grande acoplamento entre módulos foi a maior dificuldade dos projetos de conversão de aplicações monolíticas para *microservices* (SILVA, 2015).

A arquitetura de *microservices* trouxe desafios como de realizar a implantação da arquitetura de maneira gradativa, evitando que situações de necessidade de resolução em curto prazo não atrasassem ou gerassem problemas para a área de negócio (SILVA, 2015). Na Figura 4 pode-se observar a aplicação após sua implementação. A arquitetura pode ser vista de forma mais modular e subdivida, separados por módulos da aplicação (letra A), bem como os serviços de acesso à camada de dados (letra B). Além disso, a visão geral fica simplificada. O resultado desta pesquisa, intitulada MS-Trick se assemelha a imagem da Figura 4.

Figura 4 - Nova arquitetura da aplicação



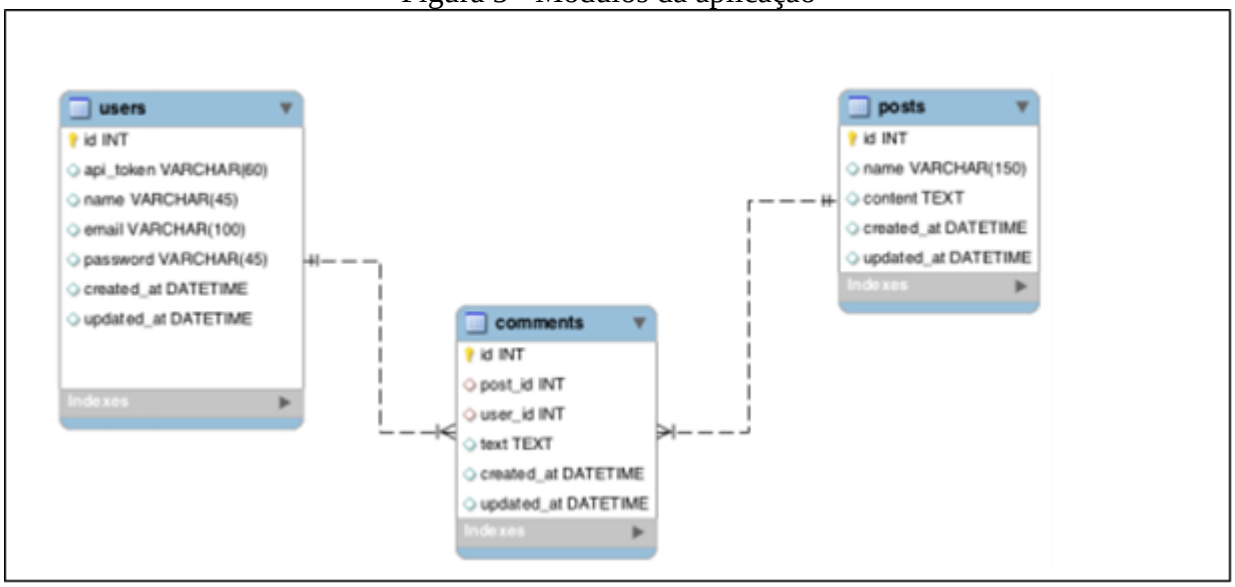
Fonte: Silva (2015).

2.5.3 Integrando *microservices* em uma aplicação web

São Miguel e Farina (2016) trazem em seu estudo uma aplicação monolítica desenvolvida em linguagem PHP, fazendo uso basicamente de três tabelas representadas pela Figura 5: `Users` (letras A), `Comments` (letra B), e `Posts` (letra C). Cada uma das tabelas representa um módulo da aplicação. Como pode ser visualizado pela Figura 5, o módulo `Comments` (letra B), antes parte integrante da aplicação, foi retirado da forma monolítica, tornando-se um *microservice*. Além disso, a aplicação utiliza a biblioteca CURL[7] para PHP, esssa biblioteca o Monolito se comunica com o *microservice* por meio do Representational State Transfer (REST) (SÃO MIGUEL; FARINA, 2016).

[7] Biblioteca CUR: Disponível em: <http://php.net/manual/en/book.curl.php>. Acesso em: 19 jun. 2018.

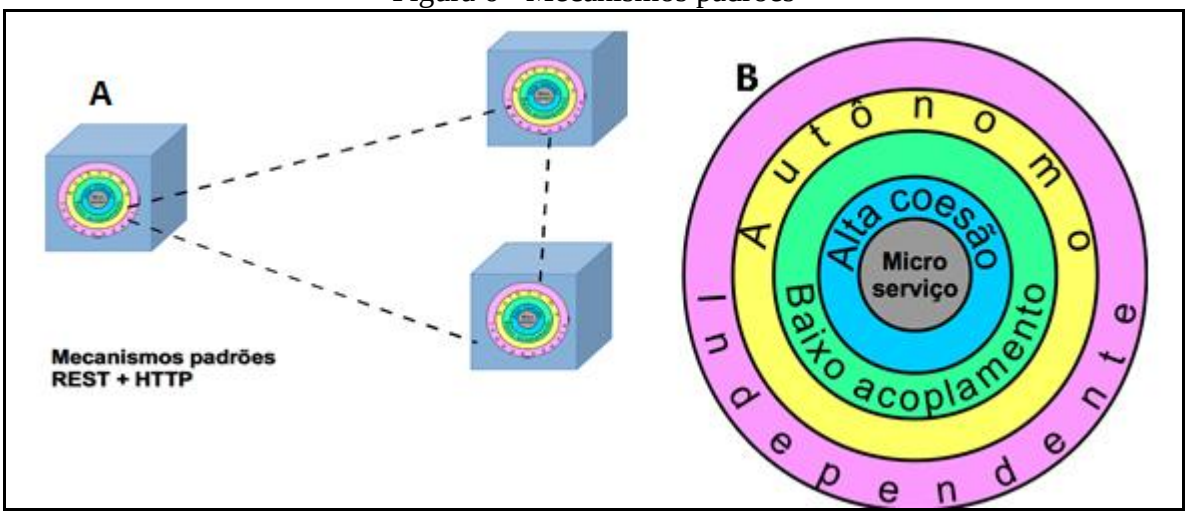
Figura 5 - Módulos da aplicação



Fonte: São Miguel e Farina (2016).

Para comunicação via Rest, São Miguel e Farina (2016) criaram uma implementação padrão, Application Programming Interface (API), para acesso dos métodos do módulo de comentários, como: retornar a lista de comentários de um determinado *post*. Após a implementação da arquitetura a aplicação existente se demonstrou mais escalável (SÃO MIGUEL; FARINA, 2016). Além disso, devido à separação das aplicações foi obtida autonomia de cada uma delas que pode ser verificado pela Figura 6. Ainda na Figura 6 é possível visualizar pela letra A como um *microservice* é representado dentro da arquitetura e pela letra B características obtidas pela utilização da arquitetura.

Figura 6 - Mecanismos padrões



Fonte: Sampaio (2015).

A manutenibilidade mostrou-se menos complexa. Entretanto, a chamada de métodos do módulo de comentários se tornou mais custosa, pois além da requisição Hypertext Transfer Protocol (HTTP), são utilizados dados de rede. Desta forma é possível visualizar que além de

vantagens *microservices* trazem novos desafios as aplicações (SÃO MIGUEL; FARINA, 2016).

2.5.4 Comparação entre os trabalhos correlatos

O Quadro 4 apresenta de forma comparativa características entre os trabalhos correlatos, comparando os trabalhos de Tonse (2014), Silva (2015) e São Miguel e Farina (2016) demonstrando as principais características buscadas com este trabalho.

Quadro 4 - Comparativo da Correlação entre Trabalhos Correlatos

Correlatos / Características	Tonse (2014)	Silva (2015)	São Miguel e Farina (2016)
Escalabilidade	✓	✓	X
Integração contínua	✓	X	X
Disponibilidade	✓	✓	X
Métricas exclusivas	X	X	✓
Modularidade	✓	✓	X
Linguagem de programação	Java	Java	PHP
Melhoria no processo de manutenção	✓	✓	X

Fonte: elaborado pelo autor.

Conforme demonstrado no Quadro 4, os trabalhos correlatos trazem o estudo de Tonse (2014), Silva (2015) e São Miguel e Farina (2016). Tonse (2014) e Silva (2015) se destacam por fazerem uso da arquitetura como vantagem competitiva e melhoria no modelo atual de entrega da aplicação, enquadrando-se assim em diversas características. Dentre as principais estão: escalabilidade, integração contínua e disponibilidade. Enquanto São Miguel e Farina (2016) a arquitetura foi utilizada para modularização, e, sobretudo conseguir métricas exclusivas do *microservice*. A autonomia desse foi uma das métricas adquiridas pela implementação da arquitetura.

Com relação à linguagem de programação, os projetos de Tonse (2014) e Silva (2015), ambos são desenvolvidos em Java, enquanto São Miguel e Farina (2016) utilizam a linguagem de programação PHP, demonstrado que não existe uma única linguagem a ser utilizada para o desenvolvimento da arquitetura. Tonse (2014) e Silva (2015) destacam-se também pelos benefícios relacionados, os quais se ligam como solução às dificuldades encontradas, possibilitando verificar as seguintes correlações entre os problemas encontrados:

a) difícil provisionamento de recursos: Tonse (2014) e Silva (2015) relacionam diretamente com escalabilidade e disponibilidade. A utilização demasiada da aplicação faz com que seja necessário escalar mais recursos para manter a disponibilidade aceitável na aplicação;

b) dificuldade de compilação e desacoplamento de diferentes módulos: Tonse (2014) e Silva (2015) destacam características como módulos separados e serviços sem dependências de partes de outros módulos, ou seja, basicamente a modularidade da aplicação. A modularidade da aplicação faz com que seja possível trabalhar de maneira individual com cada parte da aplicação, não necessitando a compilação completa da aplicação;
c) demora em compilação de toda a aplicação: principalmente em Silva (2015) está relacionado com a modularidade, devido que em projetos mais modulares não se faz necessário realizar a compilação de toda a aplicação constantemente;
d) impossibilidade de manutenção de um único módulo da aplicação: Tonse (2014) e Silva (2015) relacionam com a modularidade e integração contínua. Destacam-se como aplicações modulares e conceitos de integração contínua, também possuem manutenção e liberação mais veloz de módulos únicos da aplicação;
e) crescimento apenas horizontal dos recursos da aplicação: Tonse (2014) e Silva (2015) relacionam com a escalabilidade. Segundo Coelho (2004), aplicações escaladas horizontalmente além de contarem com um custo monetário menor, possuem diversas vantagens como maior disponibilidade. Outro ponto importante para se escalar uma aplicação horizontalmente são métricas, utilizadas em cada módulo, principal característica do trabalho de São Miguel e Farina (2016).

3 DESENVOLVIMENTO

Este capítulo descreve as etapas do desenvolvimento. Na seção 3.1 é apresentado o levantamento de informação necessária para a implementação; a seção 3.2 demonstra os diagramas, bem como os casos de uso da arquitetura. O código implementado para a arquitetura é demonstrado na seção 3.3; e na seção 3.4 os resultados obtidos com o trabalho desenvolvido são apresentados.

3.1 LEVANTAMENTO DE INFORMAÇÕES

Neste trabalho foi desenvolvida uma arquitetura de *microservices*. Como visto na subseção 2.5.1 a arquitetura não se dá somente pela implementação de um *microservice*, mas também, pela integração contínua e posteriormente pelos benefícios adquiridos combinados. Deste modo, serão apresentados os *microservices* desenvolvidos, a integração contínua e os passos para sua execução; e um aplicativo para facilitar a visualização do projeto.

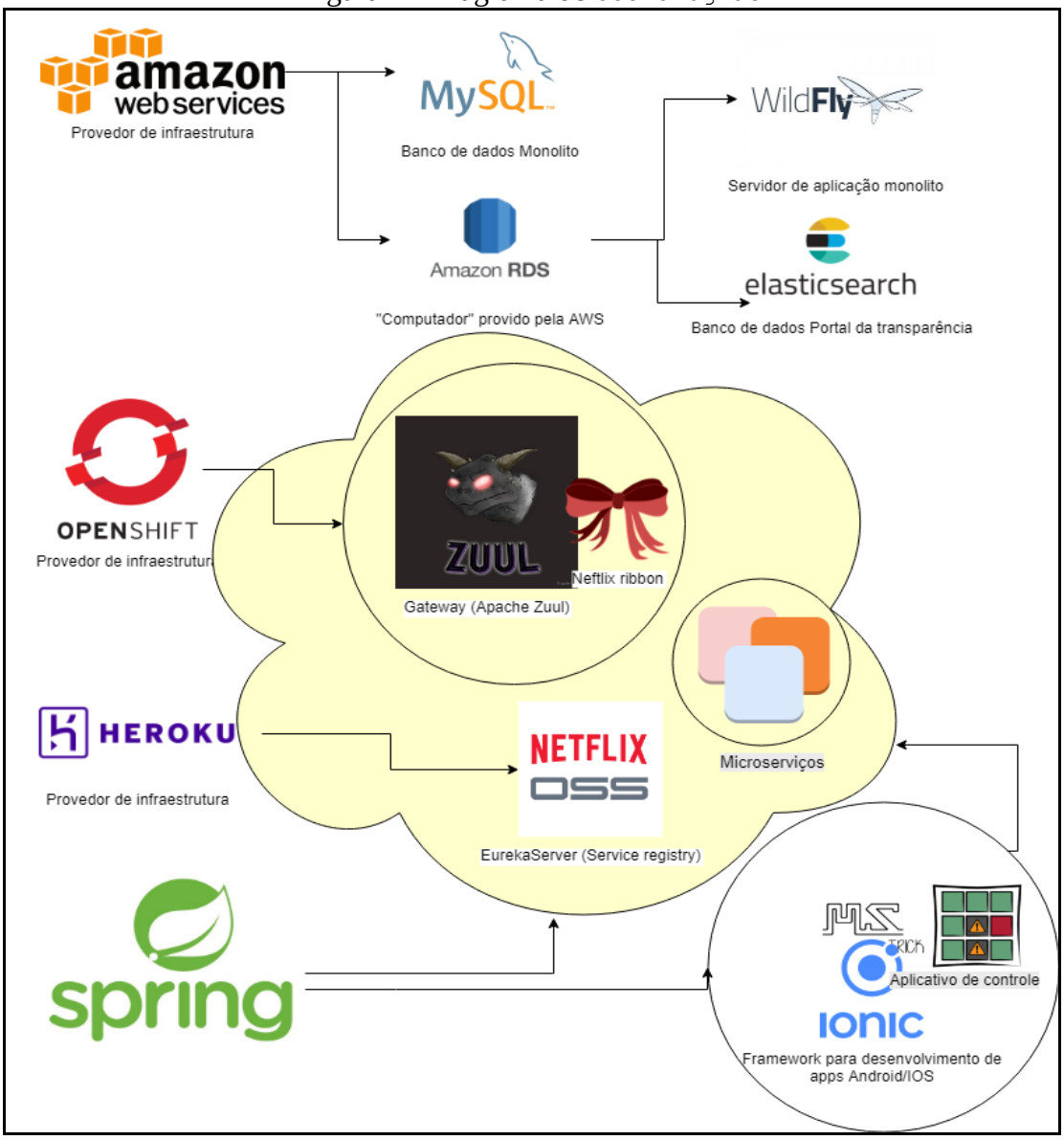
O Monolito estudado se trata de uma aplicação com os módulos descritos na seção 2.1, contudo, a implementação da arquitetura foi realizada em dois módulos, devido ao tamanho do sistema descrito. O primeiro módulo escolhido foi o `portal da transparência`, sendo conhecido não somente por usuários do sistema, mas também pela população brasileira, a qual busca dados e informações sobre o governo. O segundo módulo é o `file-server`, que é praticamente invisível para o usuário, mas que todas as aplicações utilizam. Este módulo é responsável por disponibilizar uma Uniform Resource Locator (URL) para `download` do respectivo arquivo. Além dos dois módulos destacados para desenvolvimento, o *`backend`* do aplicativo móvel foi implementado dentro da mesma arquitetura.

Como forma de facilitar o entendimento do projeto, a Figura 7 traz cada uma das tecnologias utilizadas no desenvolvimento, bem como, suas respectivas infraestruturas de execução. Pela referida figura é possível concluir que nenhuma parte da arquitetura está sendo executada localmente e utiliza serviços gratuitos, disponibilizados pelas plataformas.

Dois aspectos merecem destaque: primeiramente, pode-se visualizar que cada parte executável possui um provedor de infraestrutura. Atualmente são utilizados três, todos de forma gratuita, sendo estes: `AWS`, `Openshift` e `Heroku`. Esses provedores foram utilizados para demonstrar que é possível executar partes da arquitetura de maneira individual, em qualquer infraestrutura. Outro ponto a se observar no diagrama é a nuvem (localizada ao centro da Figura 7), representado os *microservices*, bem como as tecnologias para o desenvolvimento deles Ainda se referindo a nuvem, é possível verificar a conexão com a logo

do `framework Spring`, que foi utilizado em todas as partes necessárias para a execução da arquitetura.

Figura 7 - Diagrama de `tecnologias`



Fonte: elaborado pelo autor.

Para a realização deste trabalho cumpriram-se os seguintes requisitos:

a) integração continua deve permitir aos administradores de sistema e desenvolvedores a realização de *deploy* de aplicações (Requisito Funcional – RF);
b) integração continua deve permitir aos administradores de sistema e desenvolvedores realização a compilação de aplicações (RF);
c) integração continua deve permitir aos administradores de sistema e desenvolvedores a avaliação do código das aplicações (RF);
d) arquitetura deve permitir a execução de múltiplas instâncias dos *microservices* pelos administradores de sistema (RF);

e) implementar os *microservices* utilizando Spring Boot como servidor de aplicação (Requisito não funcional – RNF);
f) utilizar Gitlab para desenvolver a integração contínua (RNF);
g) utilizar tecnologias disponibilizadas pelo Netflix: Hystrix, Ribbon, Zuul (RNF);
h) utilizar apenas ferramentas gratuitas para desenvolvimento do projeto (RNF).

Os seguintes requisitos deverão ser contemplados na aplicativo de visualização:

a) aplicativo deve permitir o usuário realizar *login* (RF);
b) aplicativo deve permitir o usuário reiniciar uma instância de *microservice* (RF);
c) aplicativo deve permitir o usuário visualizar *microservices* executados (RF);
d) aplicativo deve permitir o usuário desligar uma instância de *microservice* (RF);
e) aplicativo deve permitir o usuário visualizar métricas de *microservice* (RF);
f) aplicativo deve permitir o usuário visualizar os *endpoints* dos *microservices* (RF);
g) aplicativo deve permitir o usuário visualizar as variáveis de ambiente dos *microservices* (RF);
h) aplicativo deve permitir o usuário visualizar as requisições realizadas aos *microservice* (RF);
i) aplicativo permitir o usuário habilitar ou desabilitar notificações (RF);
j) aplicativo deve permitir o usuário recebimento de notificações (RF);
k) aplicativo deve permitir ao usuário configurar de URLs para Service Registry (RF).
l) *backend* do aplicativo deve permitir ao sistema o envio de notificações (RF).
m) implementar a interface em Ionic (RNF);
n) implementar *backend* do aplicativo utilizando Spring Boot (RNF).

As tecnologias não listadas nos RNFs demonstradas na Figura 7 representam técnicas já utilizadas no sistema atual (seção 2.1). A utilização das demais tecnologias será descrita na seção 3.3.

3.2 ESPECIFICAÇÃO

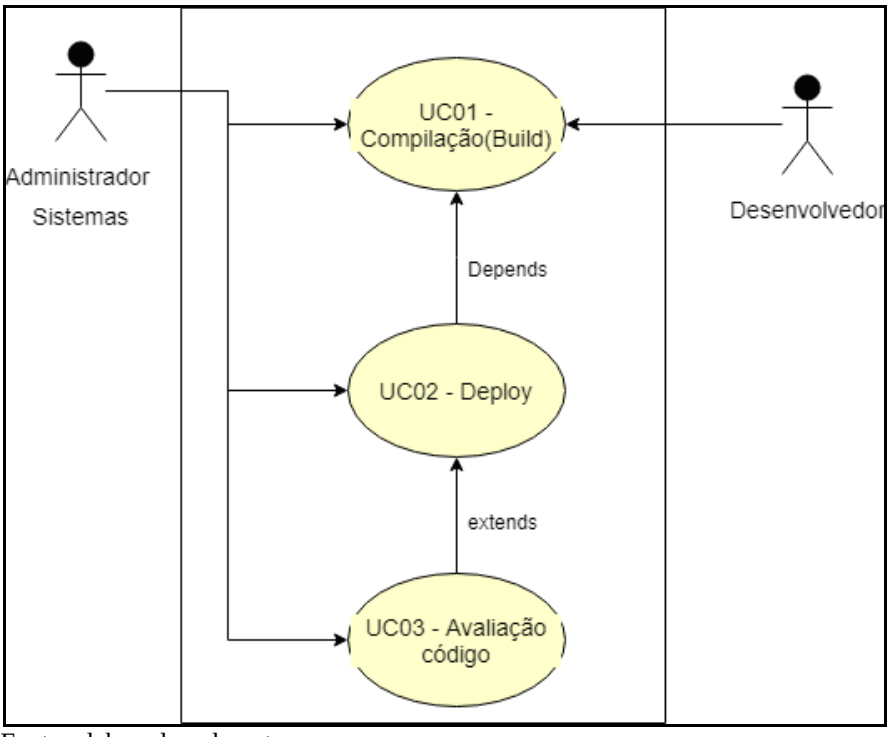
Esta seção contém a especificação do sistema desenvolvido. Apresentam-se os Diagramas de Caso de Uso (DCU) na subseção 3.2.1 e suas respectivas rastreabilidades no Quadro 5. Nas subseções subsequentes são demonstrados os demais diagramas utilizados durante o desenvolvimento da arquitetura.

3.2.1 Diagrama de Casos de Uso

Nesta subseção são apresentados os diagramas de casos de uso elaborados a partir da especificação dos requisitos descritos na seção 3.1 Os casos uso foram distribuídos em dois diagramas, sendo que o primeiro trata dos casos de uso relacionados ao `administrador do sistema` e do `desenvolvedor`; e o segundo diagrama representa o relacionamento entre o `usuário do aplicativo` e o `sistema MS-Trick`.

Na Figura 8 pode-se visualizar o primeiro diagrama descrito (`UC01` à `UC3) e` os `UC04` à `UC16` demonstrados na Figura 9 dizem respeito aos casos de uso relacionados ao aplicativo desenvolvido. O `UC01 – Compilação(Build)` permite que o código fonte das aplicações seja compilado de maneira remota no Gitlab[8]. O `UC02 – Deploy` realiza o *deploy*, disponibilizando a aplicação em um servidor de aplicações específico. Por fim, o caso de uso `UC03 – Avaliação código` permite a visualização de bugs e melhorias possíveis no código das aplicações.

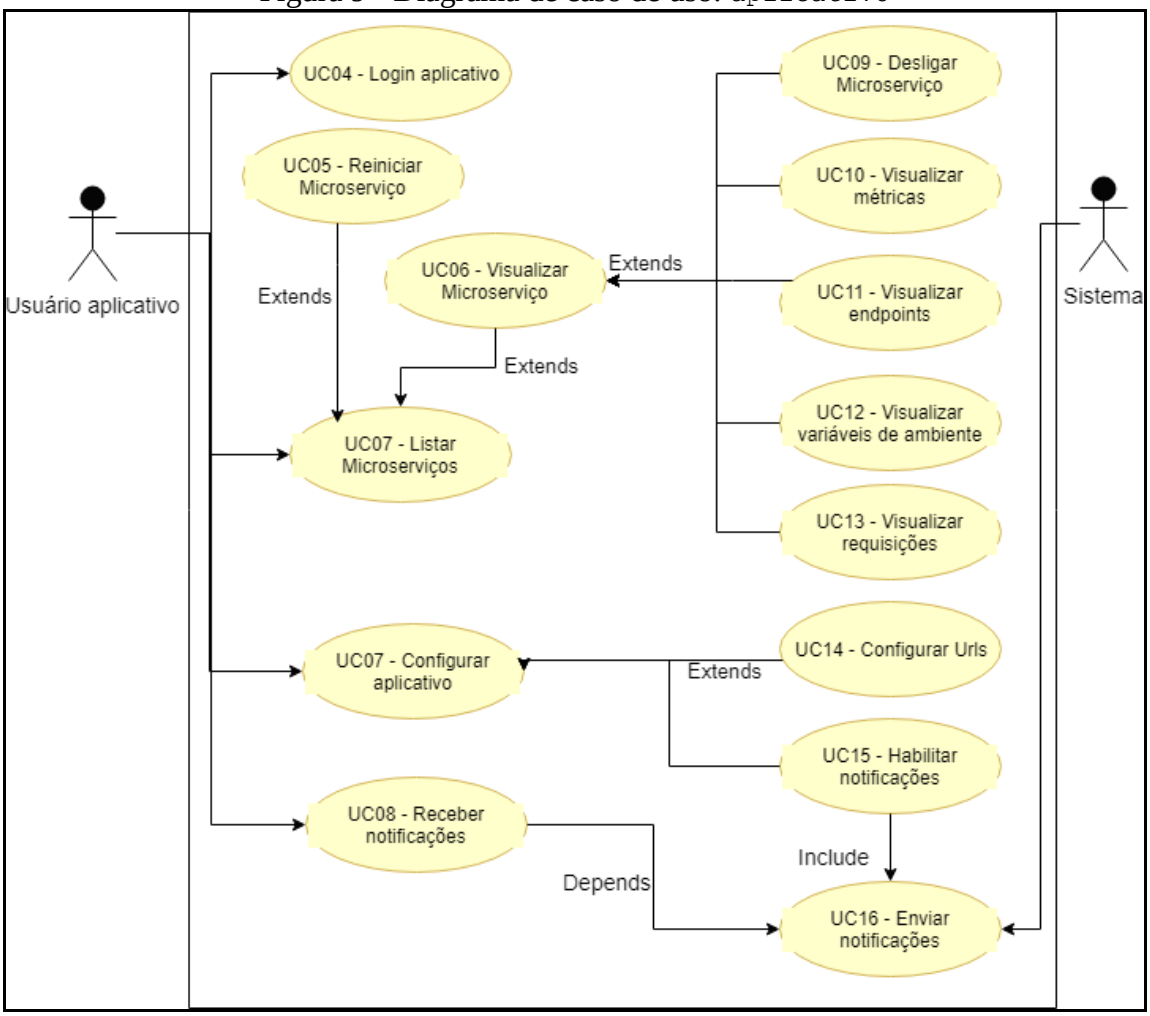
Figura 8 - Diagrama de caso de uso: `integração contínua`



Fonte: elaborado pelo autor.

[8] Gitlab: Disponível em: <https://about.gitlab.com/>. Acesso em: 19 jun. 2018.

Figura 9 - Diagrama de caso de uso: `aplicativo`



Fonte: elaborado pelo autor.

O caso de uso `UC04 – Login aplicativo` possibilita que o usuário realize `login` no aplicativo, verificando se as credencias são válidas. Após realizar o `login`, o caso de uso `UC07 – Listar Microservices` lista todos os *microservices* ativos e por meio dessa listagem permite que o usuário reinicie um *microservice* (`UC05 – Reiniciar Microservice`) ou visualize as informações detalhadas desse *microservice* (`UC06 – Visualizar Microservice`) ao selecionar o *microservice*.

Ao selecionar um *microservice* o usuário terá acesso a algumas opções referentes ao item selecionado, podendo `desligar o micro serviço` (`UC09 – Desligar Microservice`), desativando o *microservice* que deixará de executar as atividades. O `UC10 – Visualizar métricas` permite ao usuário verificar as métricas do *microservice*, ou seja, memória utilizada, memória disponível, número de processadores etc. O `UC11 – Visualizar endpoints,` permite visualizar os pontos de entrada ao *microservice* (*endpoints*);

Enquanto `o UC11 – Visualizar endpoints` permite a visualização das requisições feitas a estes pontos de entrada. O `UC07 – Configurar aplicativo` permite a configuração

do aplicativo; o `UC14 – Configurar URLs` permite a configuração de URLs necessárias para visualização dos *microservices* e suas métricas e o `UC15 – Habilitar notificações` permite habilitar notificações de controle dos *microservices*. Caso as notificações sejam habilitadas o *backend* passa a realizar o `UC16 – Enviar notificações`, enviando notificações aos dispositivos que já executaram o aplicativo, fazendo com que os usuários recebam tais notificações pelo `UC08 – Receber notificações`.

3.2.2 Matriz de rastreabilidade entre os requisitos funcionais e os casos de uso

No Quadro 5 pode ser visualizada a matriz de rastreabilidade relacionando os requisitos funcionais e com os casos de uso levantados para o desenvolvimento da arquitetura.

Quadro 5 – Matriz de rastreabilidade entre os RF e os UC

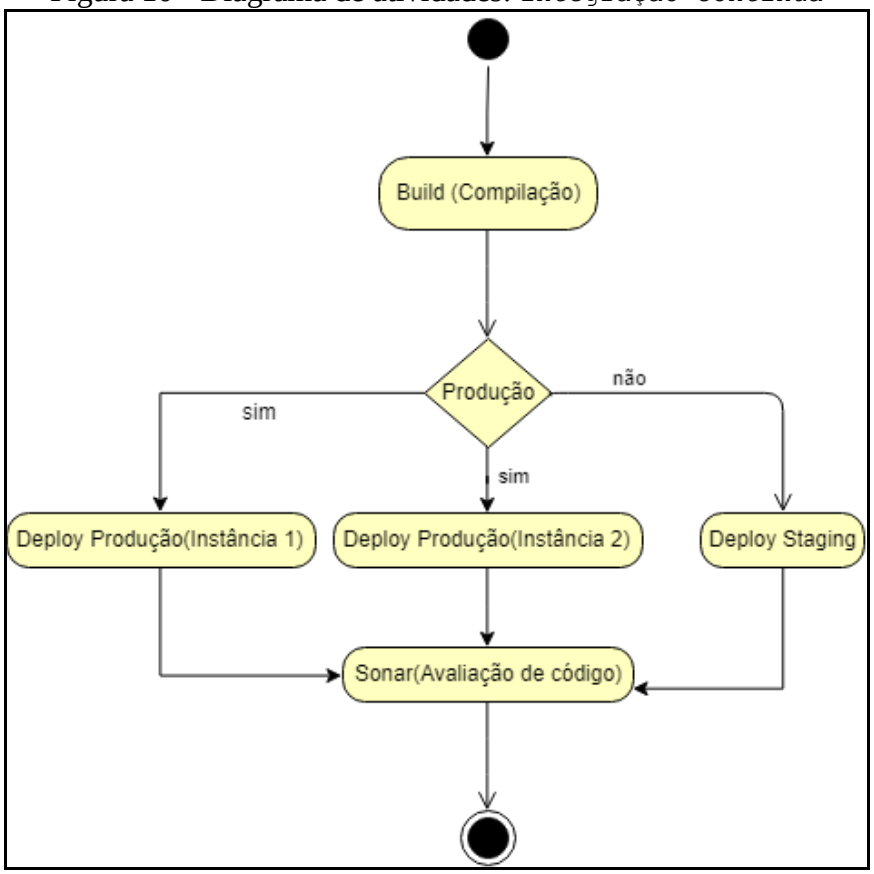
REQUISITOS FUNCIONAIS	**UC**
RF01 - Integração continua deve permitir aos administradores de sistema e desenvolvedores a realização de deploy de aplicações.	UC01
RF02 - Integração continua deve permitir aos administradores de sistema e desenvolvedores realização a compilação de aplicações.	UC02
RF03 - Integração continua deve permitir aos administradores de sistema e desenvolvedores a avaliação do código das aplicações.	UC03
RF04 - Aplicativo deve permitir o usuário realizar *login*.	UC04
RF05 - Aplicativo deve permitir o usuário reiniciar uma instância de *microservice*.	UC05
RF06 - Aplicativo deve permitir o usuário desligar uma instância de *microservice*.	UC06
RF07 - Aplicativo deve permitir o usuário visualizar *microservices* executados.	UC07
RF08 - Aplicativo deve permitir o usuário desligar uma instância de *microservice*.	UC09
RF09 - Aplicativo deve permitir o usuário recebimento de notificações.	UC08
RF10 - Aplicativo deve permitir o usuário visualizar métricas de *microservice*.	UC10
RF11 - Aplicativo deve permitir o usuário visualizar os *endpoints* dos *microservices*.	UC11
RF12 – Aplicativo deve permitir o usuário visualizar as requisições realizadas aos *microservice*.	UC13
RF13 - Aplicativo deve permitir o usuário visualizar as variáveis de ambiente dos *microservices*.	UC12
RF14 - Aplicativo permitir o usuário habilitar ou desabilitar notificações.	UC15
RF15 - Aplicativo deve permitir aplicativo configurar de URLs para Service Registry.	UC14
RF16 - B*ackend* do aplicativo deve permitir ao sistema o envio de notificações.	UC16

Fonte: elaborado pelo autor.

3.2.3 Diagrama de atividades: integração contínua

Esta subseção apresenta o diagrama de atividades da integração contínua desenvolvido na Ferramenta DrawIO[9]. Para esta parte da arquitetura foi desenvolvido um diagrama de atividade, por este ser idealmente representado por tal diagrama, representado na Figura 10. Apesar deste fluxo se demonstrar simples, representa benefício por automatizar atividades repetitivas realizadas diariamente.

Figura 10 - Diagrama de atividades: `integração contínua`



Fonte: elaborado pelo autor.

Na Figura 10 é possível visualizar o fluxo de integração contínua. O fluxo é iniciado pela `compilação` do aplicativo, esta etapa é realizada de forma abstrata, pois a Ferramenta Gitlab permite a criação de qualquer ambiente Linux para realização de procedimentos. O `caso de deploy` possui dois fluxos diferentes: caso este seja executado com a base de código de desenvolvimento será feito *deploy* de uma instância no servidor de testes (`deploy staging`); caso seja executado com a base de código de produção será realizado o *deploy* para duas instâncias de produção - `deploy production` – (servidores). Destaca-se que poderia ser realizado o *deploy* do número necessário de instâncias para suprir a necessidade do

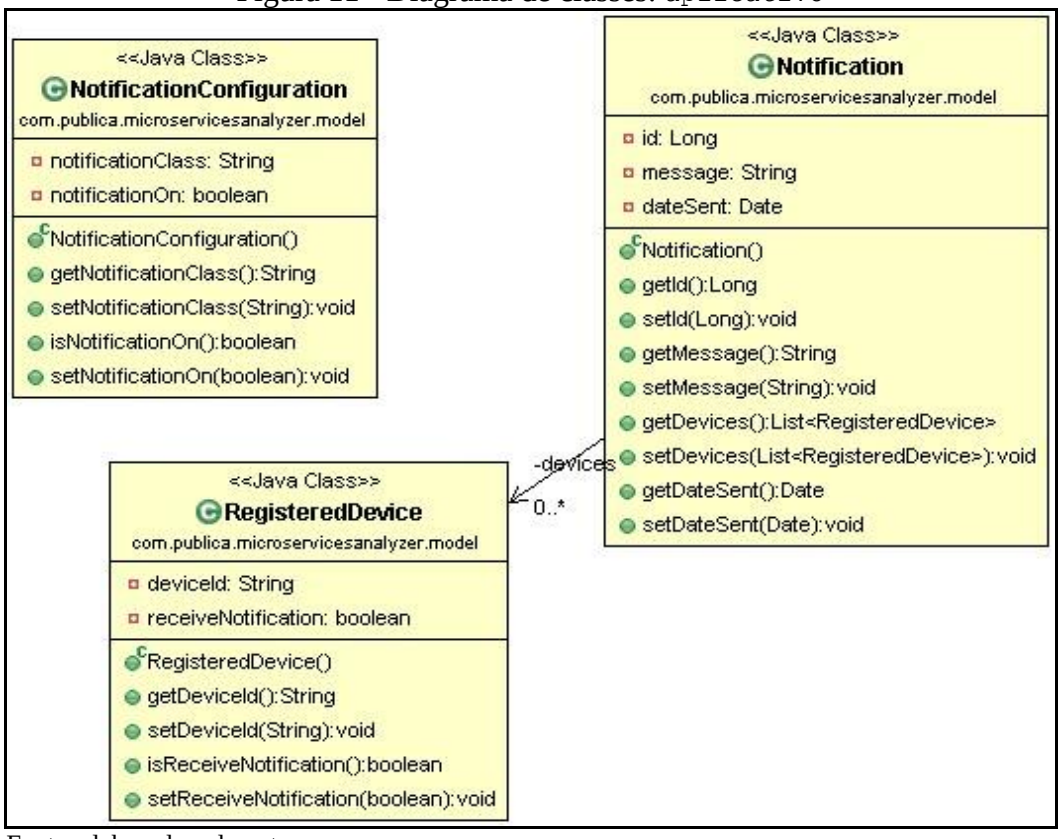
[9] Ferramenta DrawIO: Disponível em: <https://www.draw.io/>. Acesso em: 19 jun. 2018.

usuário. Por último, é realizada a avaliação do código pelo *website* `Sonar`[10], no qual são lidos os arquivos compilados e enviados para o *website* realizar a avaliação.

3.2.4 Diagrama de classes: aplicativo

O diagrama de classes foi desenvolvido dentro do Ambiente Integrado de Desenvolvimento (IDE) RedHat Development Suite[11] pela facilidade de geração de código. A ferramenta utilizada ainda é muito similar a IDE Eclipse[12], ferramenta mais familiar ao desenvolvedor. O diagrama demonstrado na Figura 11 traz as classes do aplicativo, pela necessidade do desenvolvimento de classes persistentes para sua execução.

Figura 11 - Diagrama de classes: `aplicativo`



Fonte: elaborado pelo autor.

A Figura 11 contempla as classes persistentes do aplicativo do `MS-Trick`. Foram desenvolvidas três classes, devido que as demais ações realizadas no aplicativo são diretamente executadas nos *microservices*. As classes são: `NotificationConfiguration`, mantém as configurações de quais notificações serão enviadas para o

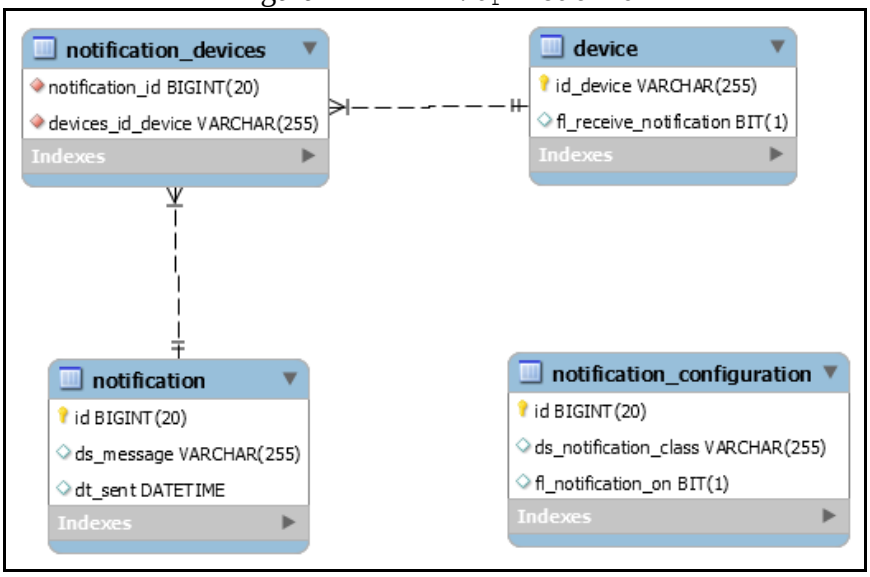
[10] website Sonar: Disponível em: <https://sonarcloud.io>. Acesso em: 19 jun. 2018.
[11] RedHat Development Suite: Disponível em: <https://developers.redhat.com/products/devsuite/overview/>. Acesso em: 19 jun. 2018.
[12] IDE Eclipse: Disponível em: <https://www.eclipse.org/>. Acesso em: 19 jun. 2018.

aplicativo;`Notification`, preserva todas as notificações já enviadas, estando relacionada à `RegisteredDevice`, contendo os respectivos dispositivos que já executaram a aplicação.

3.2.5 Modelo Entidade Relacionamento (MER): aplicativo

O Modelo Entidade Relacionamento (MER) representa as entidades persistentes da aplicação no banco de dados Mysql. Esse foi desenvolvido na Ferramenta Mysql Workbench[13] pela facilidade na criação do banco a partir do diagrama. Pela Figura 12 pode-se perceber a similaridade com o diagrama da Figura 11, porém esse tem os nomes alterados para padronização do banco de dados.

Figura 12 – MER: `aplicativo`



Fonte: elaborado pelo autor.

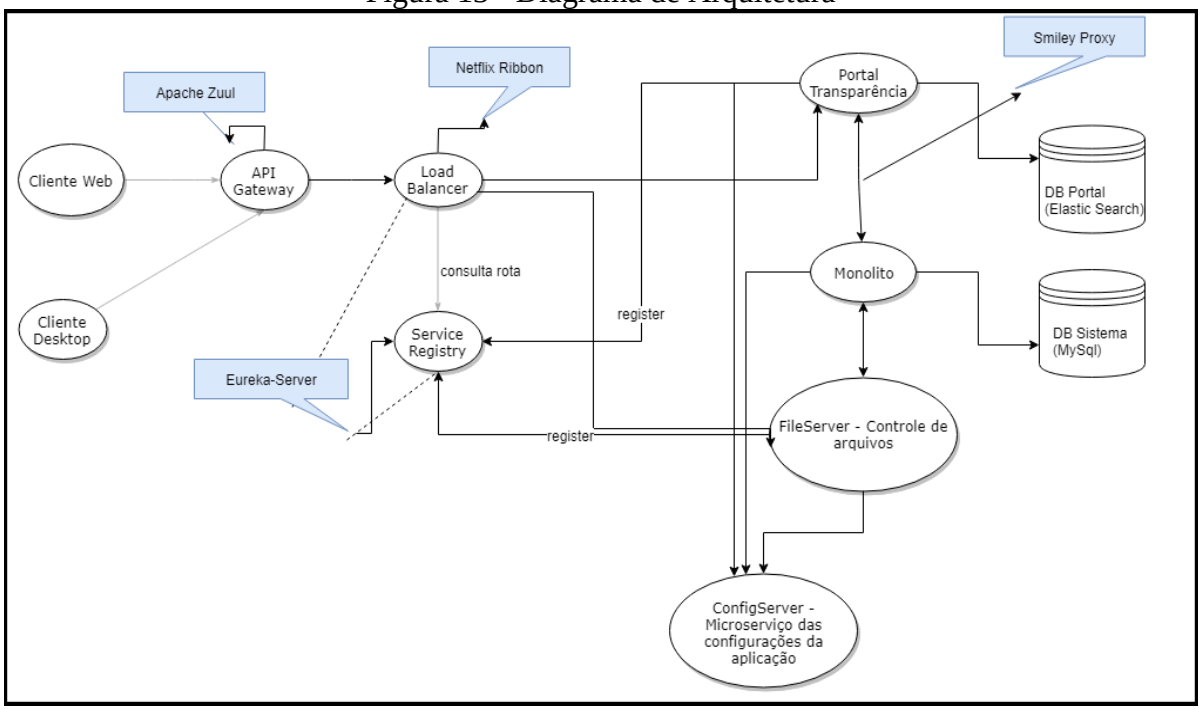
O diagrama representado na Figura 12 contempla as seguintes entidades: `notification_devices`, `notification`, `device` e `notification_configuration`. A entidade `notification_devices` demostra o relacionamento entre `device` e `notification`, pois uma notificação pode ser enviada para múltiplos dispositivos, da mesma forma que um equipamento pode receber diversas notificações. A entidade `device` representa os dispositivos que já realizaram *login* no aplicativo. A entidade `notification` representa as notificações que já foram enviadas aos dispositivos. Já a entidade `notification_configuration` representa as notificações que poderão ser enviadas futuramente aos dispositivos.

[13] Mysql Workbench: Disponível em: <https://www.mysql.com/ >. Acesso em: Acesso em: 19 jun. 2018.

3.2.6 Diagrama de arquitetura

A Figura 13 apresenta a arquitetura do projeto como um todo. É possível visualizar os componentes desenvolvidos, bem como os já existentes adaptados. Na referida figura é possível visualizar o relacionamento de cada uma um dos componentes, assim como pode-se ver a parte das tecnologias que foram empregadas para o desenvolvimento do MS-Trick.

Figura 13 - Diagrama de Arquitetura



Fonte: elaborado pelo autor.

Na Figura 13 é possível visualizar os componentes da arquitetura e a seção 3.3 descreve em maiores detalhes os componentes, bem como as tecnologias utilizadas no desenvolvimento, sendo:

a) `Cliente Web e Cliente Desktop`: pontos de requisição à arquitetura. Requisições `HTTP/HTTPS` são feitas para arquitetura por meio destes clientes;

b) `API Gateway e Load Balancer`: ambos desenvolvidos na mesma aplicação. O `API Gateway` é o ponto de entrada da arquitetura, os clientes fazem as requisições para o `API Gateway` e o `Load Balancer` faz o balanço entre os *microservices* e as requisições são distribuídas entre as instâncias dos *microservices* disponíveis;

c) `Service Registry`: para que o `Load Balancer` possa balancear as requisições é necessário que ele tenha conhecimento dos *microservices* existentes executados. O `Service Registry` disponibiliza uma interface para armazenamento das instâncias em execução;

d) `ConfigServer`: as configurações entre os *microservices* são basicamente as

mesmas, possibilitando a sua centralização, desta forma múltiplos *microservices* podem acessar esta configuração ao mesmo tempo;

e) `Portal da Transparência` e `File Server`: módulos separados do `Monolito`, criando-se *microservices* a partir destes módulos;

f) `Monolíto`, `DB Portal da Transparência` e `DB Sistema`: fazem parte da arquitetura antiga da aplicação, porém estes estão no diagrama da arquitetura porque essa se comunica com estes componentes.

3.3 IMPLEMENTAÇÃO

Nesta seção são descritas ferramentas e técnicas utilizadas no desenvolvimento da arquitetura (subseção 3.3.1), em adição a implementação da arquitetura e sua operacionalidade (subseção 3.3.4).

3.3.1 Técnicas e ferramentas utilizadas

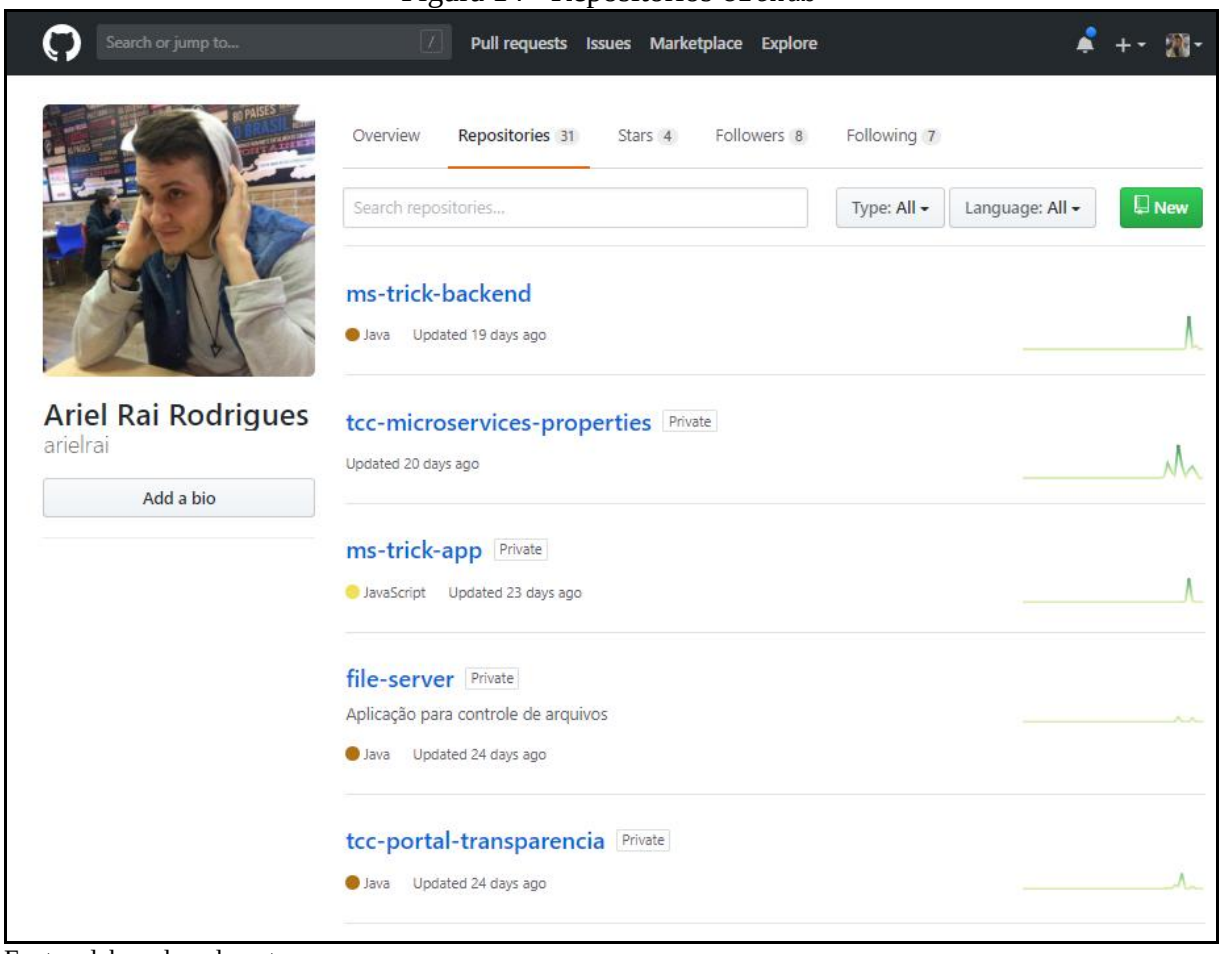
Para desenvolver o projeto intitulado MS-Trick foram utilizados como linguagem de programação Java, para alterar os serviços existentes e desenvolver novos serviços necessários para arquitetura e Ionic[14] combinado com Javascript e Cascading Style Sheets (CSS) para desenvolvimento do aplicativo. Para desenvolvimento Java foi utilizado o IDE RedHat Development Suite; para desenvolvimento do aplicativo foi utilizado o editor de texto Atom. Dois *frameworks* foram utilizados para o desenvolvimento do MS-Trick pela facilidade de integração: Spring e Netflix.

Para armazenamento de dados do aplicativo foi utilizado o banco de dados Mysql[15] e estão sendo executados em servidores providos pelos *websites* Amazon AWS, Heroku e Openshift, permitindo a execução completa da arquitetura em servidores remotos. O código implementado está disponível no *website* GitHub[16], plataforma de desenvolvimento gratuita utilizada por outros desenvolvedores (GITHUB, 2018). Na Figura 14 é possível visualizar alguns dos repositórios, responsáveis por armazenar o código fonte das diferentes partes da arquitetura.

---

[14] Ionic: Disponível em: <https://ionicframework.com/>. Acesso em: 19 jun. 2018.
[15] *website* GitHub: Disponível em: <https://www.mysql.com/>. Acesso em: 19 jun. 2018
[16] Mysql: Disponível em: <https://github.com/>. Acesso em: 19 jun. 2018

Figura 14 - Repositórios `GitHub`



Fonte: elaborado pelo autor.

Na Figura 14 são demonstrados quatro dos repositórios do projeto e a seguir a listagem completa dos repositórios utilizados no projeto:

a) `ms-trick-backend`: *backend* do aplicativo de visualização;
b) `tcc-`*`microservices`*`-properties`: propriedades compartilhadas dos aplicativos;
c) `ms-trick-app`: aplicativo de visualização;
d) `file-server`: servidor de arquivos;
e) `tcc-portal-transparencia`: portal da transparência;
f) `tcc-`*`microservices`*`-config-server`: aplicação responsável por disponibilizar as propriedades compartilhadas;
g) `tcc-`*`microservice`*`-service-registry`: propriedades compartilhadas dos aplicativos;
h) `tcc-gateway-server`: aplicação responsável por centralizar as requisições dos *microservices*;
i) `tcc-`*`microservice`*`-service-registry`: aplicação responsável pelo registro dos *microservices*.

3.3.2 Desenvolvimento do sistema

Para demonstrar como a arquitetura foi implementada será apresentada a parte da arquitetura, bem como as partes integrantes necessárias para a sua execução.

3.3.2.1 Partes integrantes da arquitetura

Foram implementados *microservices* para as seguintes partes do Monolito: `Portal da Transparência` e `File-server` (servidor de arquivos). O *backend* do aplicativo de visualização mesmo não fazendo parte da arquitetura original da aplicação foi desenvolvido dentro da arquitetura de *microservices*, trazendo os benefícios da arquitetura ao aplicativo. A ideia de um *microservice* é que além de ser uma parte menor de uma aplicação, este possa ser replicado à medida que for necessário. Desta forma é necessário a existência de uma aplicação para registro de *microservices* executantes.

Esse é o `Service registry` da aplicação e seu código fonte encontra-se no repositório `tcc-microservice-service-registry`. Foi utilizado Spring Boot como servidor e Netflix Eureka como *framework* responsável por disponibilizar o registro de *microservices*. No Quadro 6 é possível verificar que a codificação do aplicativo é simples, devido a facilidade de integração dos *frameworks*. Cabe ressaltar, que ao final da inicialização ele se auto registra.

Quadro 6 – Código do `Service registry`

15 lines (11 sloc) | 410 Bytes

```
1  package com.publica;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7  @SpringBootApplication
8  @EnableEurekaServer
9  public class EurekaServerApplication {
10
11         public static void main(String[] args) {
12                 SpringApplication.run(EurekaServerApplication.class, args);
13         }
14 }
```

Fonte: elaborado pelo autor.

Como pode ser visualizado no Quadro 6, o fonte do aplicativo de registro de *microservices* se dá por um aplicativo Java comum, possuindo método `main` (`Linha 11`) e mais duas anotações, sendo elas:

a) `@SpringBootApplication: linha 7,` anotação existente em toda aplicação Spring Boot. Ao executar a aplicação como um aplicativo Java qualquer, a anotação criará um servidor Tomcat para executar o aplicativo;

b) `@EnableEurekaServer: linha 8,` cria o ambiente para registro dos *microservices*. Disponibiliza *endpoints* e uma interface simplificada para visualização dos *microservices*.

Para que faça sentido à execução de múltiplas instâncias de uma aplicação é necessário um ponto único de entrada para os usuários (Quadro 7). Pode-se utilizar o seguinte exemplo, mesmo que existam `X instâncias` de cada parte do `Netflix` sendo executadas, o aplicativo faz a requisição para um local único chamado de `Api Gateway`. `Api Gateway` recebe as requisições e tem como objetivo distribuí-las entre os *microservices* registrados no `Service registry`. Para o desenvolvimento desta aplicação o `ApiGateway` foram utilizados Spring Boot e três tecnologias disponibilizadas pela Netflix: Apache Zuul, Ribbon e Hystrix.

Quadro 7 - Código `Api Gateway`

```
1  package com.publica;
2
3  import org.springframework.boot.autoconfigure.SpringBootApplication;
4  import org.springframework.boot.builder.SpringApplicationBuilder;
5  import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
6  import org.springframework.cloud.sleuth.sampler.AlwaysSampler;
7  import org.springframework.context.annotation.Bean;
8  import org.springframework.web.bind.annotation.RestController;
9
10 @SpringBootApplication
11 @EnableZuulProxy
12 public class GatewayServerApplication {
13
14         public static void main(String[] args) {
15                 new SpringApplicationBuilder(GatewayServerApplication.class).web(true).run(args);
16         }
17
18 }
```

Fonte: elaborado pelo autor.

Um ponto a salientar da aplicação, vê-se que em suas propriedades de configuração na `Linha 2` do Quadro 8 está a url para o `Service Registry`, possibilitando encontrar os *microservices* disponíveis. A implementação deste aplicativo é similar ao do `Service registry` disponibilizado no Quadro 6, porém adicionando a anotação `@EnableZuulProxy` visível na `linha 11` do Quadro 8, sendo essa responsável por ativar:

a) `Apache Zuul`: responsável por receber as requisições numa única URL;

b) `Ribbon`: faz o balanço das requisições entre os *microservices* disponíveis, desta forma mantém a carga relativamente semelhante em todas as instâncias;

c) `Hystrix`: não permite que requisições fiquem travadas para sempre, após um tempo determinado, configurável nas propriedades do aplicativo, as requisições são canceladas e um erro é retornado.

Quadro 8 - Propriedades de configuração: Api Gateway

```
2 lines (2 sloc)  106 Bytes
1  spring.application.name: gateway-server
2  spring.cloud.config.uri: https://tcc-config-service.herokuapp.com/
```

Fonte: elaborado pelo autor.

O `gateway` também possui configurações manuais que podem ser visualizadas pelo Quadro 9, sendo possível visualizar a URL de entrada na `linha 28`, e seu respectivo aplicativo redirecionado `linha 32`, `linha 35` e `linha 38`; exemplo: ao requisitar o `gateway/portal`, as requisições serão redirecionadas ao serviço `portal-transparencia`.

Quadro 9 - Configuração `Api Gateway`

```
27  zuul:
28    prefix: /gateway
29    routes:
30      portal:
31        path: /portal/**
32        serviceId: portal-transparencia
33      file-server:
34        path: /epublica-file-server/**
35        serviceId: file-server
36      ms-trick:
37        path: /ms-trick/**
38        serviceId: ms-trick
```

Fonte: elaborado pelo autor.

Outro ponto importante para arquitetura são as configurações dos aplicativos. Como o aplicativo pode ser replicado X vezes, faz sentido manter estas configurações em um único local. Para isso foi criado o `ConfigServer` representado pelos repositórios: `tcc-`*`microservice`*`-config-server`, aplicativo responsável por disponibilizar as propriedades em si e `tcc-`*`microservices`*`-properties`, responsável por armazenar as configurações dos aplicativos. No Quadro 10 é possível visualizar a implementação do aplicativo e no Quadro 11 da `linha 2` a `linha 4` são demonstradas a `URL`, `usuário` e `senha` respectivamente para acesso das configurações dos aplicativos; por último, no Quadro 11 são demonstrados arquivos de configurações existentes.

Quadro 10 - Código `ConfigServer`

15 lines (11 sloc) | 418 Bytes

```
1  package com.publica;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.config.server.EnableConfigServer;
6
7  @SpringBootApplication
8  @EnableConfigServer
9  public class DeliveryConfigServerApplication {
10
11         public static void main(String[] args) {
12                 SpringApplication.run(DeliveryConfigServerApplication.class, args);
13         }
14 }
```

Fonte: elaborado pelo autor.

Quadro 11 - Configuração `ConfigServer`

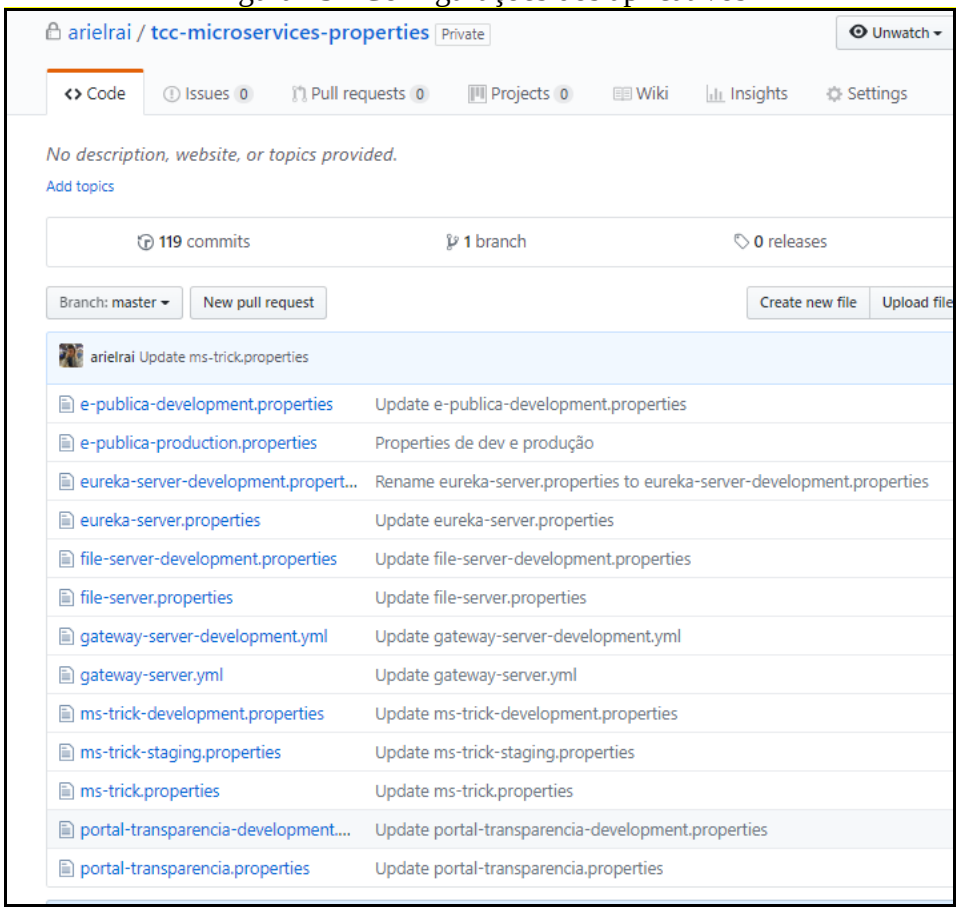
5 lines (4 sloc) | 216 Bytes

```
1  server.port: 9090
2  spring.cloud.config.server.git.uri: https://github.com/arielrai/tcc-microservices-properties
3  spring.cloud.config.server.git.username=
4  spring.cloud.config.server.git.password=
```

Fonte: elaborado pelo autor.

É possível visualizar na `linha 8` do Quadro 10 a anotação `@EnableConfigServer`, fazendo com que a aplicação faça a leitura da sua configuração visível no Quadro 11, criando um servidor Rest responsável por disponibilizar os arquivos de propriedade listados na Figura 15.

Figura 15 - Configurações dos aplicativos

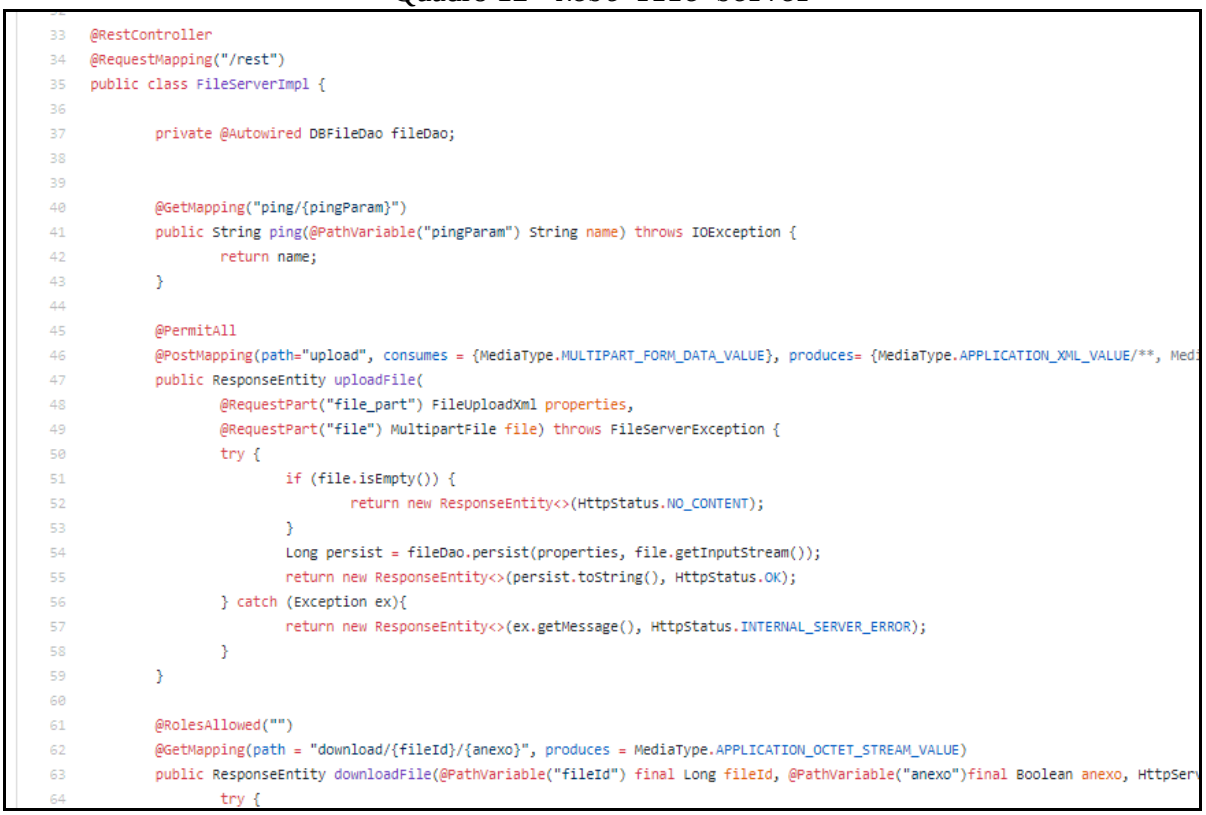


Fonte: elaborado pelo autor.

3.3.2.2 *Microservices*

O primeiro dos *microservices* é o `file-server`, dificilmente visto ou reconhecido pelo usuário final por se tratar de parte integrante da arquitetura do sistema e por disponibilizar os relatórios do sistema. O `file-server` antes do desenvolvimento deste trabalho não era uma aplicação `Spring`, sendo necessário adaptações em seu código, além da utilização de `Spring Boot` como servidor. Nas `linhas 47` e `63` do Quadro 12 é possível visualizar dois dos *endpoints* de entrada adaptados.

Quadro 12 - `Rest file-server`



```
33  @RestController
34  @RequestMapping("/rest")
35  public class FileServerImpl {
36
37          private @Autowired DBFileDao fileDao;
38
39
40          @GetMapping("ping/{pingParam}")
41          public String ping(@PathVariable("pingParam") String name) throws IOException {
42                  return name;
43          }
44
45          @PermitAll
46          @PostMapping(path="upload", consumes = {MediaType.MULTIPART_FORM_DATA_VALUE}, produces= {MediaType.APPLICATION_XML_VALUE/**, Medi
47          public ResponseEntity uploadFile(
48                  @RequestPart("file_part") FileUploadXml properties,
49                  @RequestPart("file") MultipartFile file) throws FileServerException {
50                  try {
51                          if (file.isEmpty()) {
52                                  return new ResponseEntity<>(HttpStatus.NO_CONTENT);
53                          }
54                          Long persist = fileDao.persist(properties, file.getInputStream());
55                          return new ResponseEntity<>(persist.toString(), HttpStatus.OK);
56                  } catch (Exception ex){
57                          return new ResponseEntity<>(ex.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
58                  }
59          }
60
61          @RolesAllowed("")
62          @GetMapping(path = "download/{fileId}/{anexo}", produces = MediaType.APPLICATION_OCTET_STREAM_VALUE)
63          public ResponseEntity downloadFile(@PathVariable("fileId") final Long fileId, @PathVariable("anexo")final Boolean anexo, HttpServ
64                  try {
```

Fonte: elaborado pelo autor.

O código disponibilizado no Quadro 12 já existia, todavia este foi alterado para utilizar anotações do framework Spring `@RestController` na `linha 33`, `@PostMapping` na `linha 46 e @GetMapping` na `linha 62` sendo estas responsáveis por disponibilizar os métodos utilizando `Rest`. O `file-server` também passou a utilizar `Spring Boot` como servidor de aplicação, possibilitando visualizar um código similar as demais aplicações no Quadro 13.

Quadro 13 - Código `Spring Boot file-server`

```
1  package com.file.server;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.boot.context.properties.EnableConfigurationProperties;
6  import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
7  import org.springframework.context.annotation.ComponentScan;
8
9  @SpringBootApplication
10 @EnableDiscoveryClient
11 @EnableConfigurationProperties
12 @ComponentScan(lazyInit = true)
13 public class FileServerApplication {
14
15         public static void main(String[] args) {
16                 SpringApplication.run(FileServerApplication.class, args);
17         }
18
19 }
```

Fonte: elaborado pelo autor

Além da anotação `@SpringBootApplication` visível na `linha 9`, é possível visualizar a anotação `@EnableDiscoveryClient` na `linha 10`, esta anotação realiza o registro do aplicativo no `Service registry` assim que iniciado. As demais anotações: `@EnableConfigurationProperties` (`Linha 11`) e `@ComponentScan` (`Linha 11`); são anotações responsáveis por controlar as classes dentro de uma aplicação Spring, estas anotações não se mostraram nas demais aplicações por se tratarem de apenas uma classe.

Para implementação do *microservice* do portal da transparência o processo foi diferente do file-server. O código fonte do portal da transparência já se tratava de uma aplicação Spring, porém esta possuía diversas dependências de outras aplicações. Removeram-se grande parte das dependências, mantendo-se apenas algumas necessárias para execução do *website*. Mesmo existindo a dependência, a forma de acoplamento foi alterada, as requisições ao portal que ainda dependem do Monolito, são redirecionadas diretamente ao mesmo pelo `Smiley Proxy` (visível na na Figura 13, subseção 3.2.6).

No Quadro 14 pode ser observado como o proxy foi configurado. Requisições ao endereço `/public` (`Linha 2`), com qualquer terminação, e exatamente `/menu` (`Linha 1`) são redirecionadas ao Monolíto sendo executado na URL demonstrada na `linha 3`. Ainda para execução do portal como uma aplicação Spring Boot foi implementado um código semelhante ao demonstrado no Quadro 13. Pequenos ajustes como nome ou pacote foram necessário por se tratar de outra aplicação.

Quadro 14 - Configuração `Smiley Proxy`

```
50 lines (39 sloc)  1.36 KB
1  proxy.epublica.menu_path: /menu
2  proxy.epublica.attr_path: /public/*
3  proxy.epublica.target: http://52.14.48.79:8080/epublica/rest
```

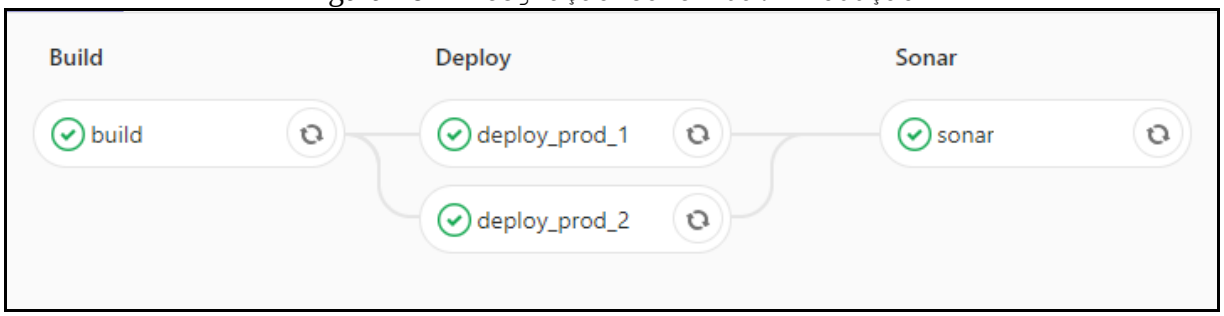
Fonte: elaborado pelo autor.

3.3.2.3 Integração contínua

A integração contínua realiza a compilação, testes se existirem, *deploy* e avaliação do código fonte da aplicação compilada, sendo parte essencial pela quantidade relativamente alta de aplicações trazidas pela arquitetura de *microservices*. A integração contínua foi implementada para um dos aplicativos, o *backend* do aplicativo de visualização.

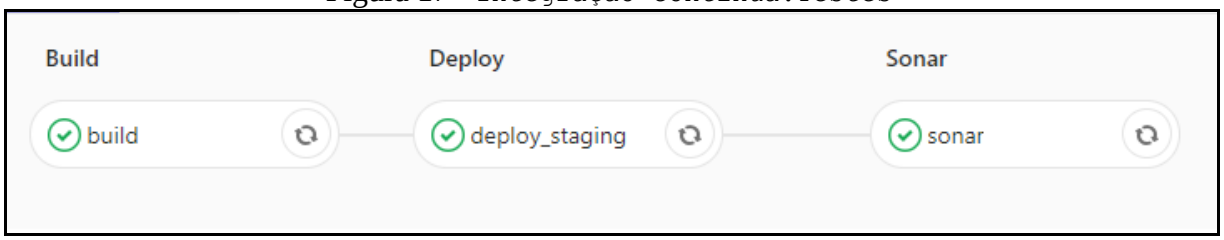
Para realização da integração contínua, descrita na seção 2.3 foi utilizada a Ferramenta Gitlab, que permite utilizar qualquer ambiente Linux para realização de procedimentos necessários para liberação do aplicativo., sendo necessário criar um `script .yml`, no formato aceito pelo Gitlab. No Quadro 15 é possível visualizar partes do script desenvolvido. Ainda no mesmo quadro é possível visualizar entre as `linhas 6 e 8` os estágios de execução do script: `build`, compilação da aplicação; `deploy`, distribuição da aplicação; e `sonar`, avaliação do código. Estes estágios são demonstrados na Figura 16 e Figura 17 respectivamente. Observa-as nas figuras que o script adota comportamentos diferenciados dependendo do repositório de fontes compilado. Na `linha 14` é demonstrado o ambiente utilizado para compilação, sendo está uma máquina Linux com a Ferramenta Maven instalada.

Figura 16 - `Integração contínua: Produção`



Fonte: elaborado pelo autor.

Figura 17 - Integração contínua:Testes



Fonte: elaborado pelo autor.

Quadro 15 - Script Gitlab

Branch: master ▾ ms-trick-backend / .gitlab-ci.yml

arielrai no message

1 contributor

73 lines (66 sloc) | 1.41 KB

```
1  cache:
2   paths:
3   - /root/.m2/repository
4
5  stages:
6   - build
7   - deploy
8   - sonar
9
10 build:
11  stage: build
12  image: maven:3.3.9-jdk-8
13  script:
14  - mvn clean install -DskipTests
15  tags:
16  - docker
17
18 sonar:
19  stage: sonar
20  image: maven:3.3.9-jdk-8
21  script:
22  - mvn clean install -DskipTests sonar:sonar -Dsonar.organization=arielrai-github -Dso
23  tags:
24  - docker
25
26 deploy_staging:
27  stage: deploy
```

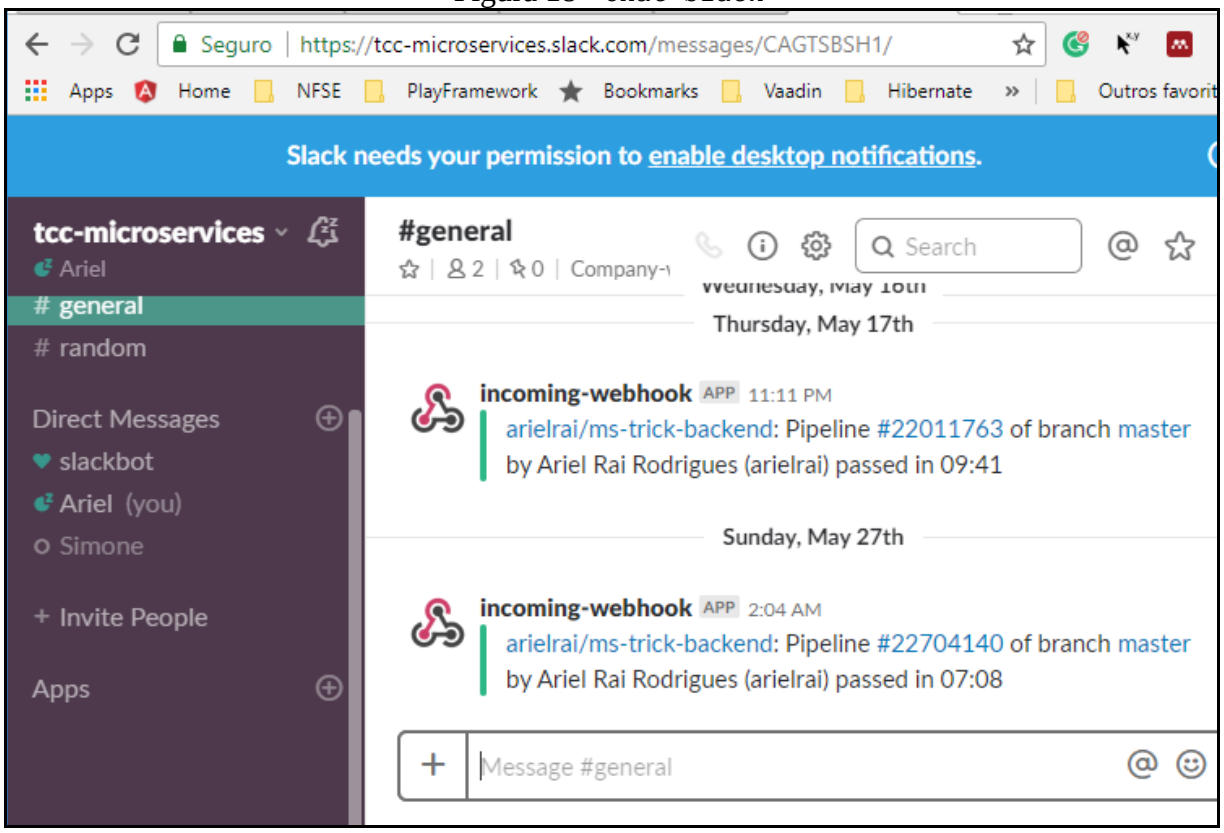
Fonte: elaborado pelo autor.

Ainda para maior facilidade de visualização de usuários da arquitetura, a cada compilação requisitada é enviada uma mensagem em caso de falha ou sucesso a um chat criado no Slack[17]. A Figura 18 demonstra algumas das notificações enviadas no chat. Estas

[17] Slack: Aplicação on-line de Chat. Disponível em: <https://slack.com/>. Acesso em: 19 jun. 2018

notificações podem demonstrar uma notificação de sucesso ou falha da compilação, bem como a base de código utilizada na compilação.

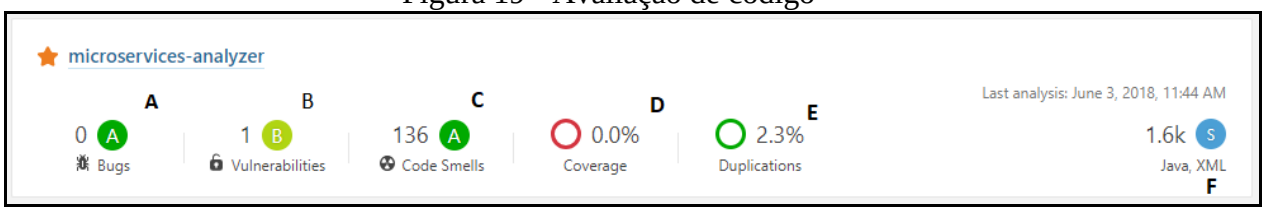
Figura 18 - `Chat Slack`



Fonte: elaborado pelo autor.

Por fim, todo fluxo de integração realiza a avaliação do código fazendo upload dos resultados ao website `sonarcloud.io`. O website mostra de forma gráfica características importantes para implementação de um bom código. Estas características são visíveis na Figura 19. Ainda é faz-se importante destacar que cada uma das métricas possui uma visualização mais detalhada clicando sob a mesma.

Figura 19 - Avaliação de código



Fonte: elaborado pelo autor.

Na Figura 19 é possível visualizar `letras` para cada uma das métricas disponibilizadas pelo *website*, sendo:

a) `letra A`: Representa falhas no código, partes que ao serem executadas geram exceções. Como é possível visualizar, não há nenhum código deste tipo;

b) `letra B`: Representa alguma vulnerabilidade no código, estas podem ser logs

suprimidos, códigos não executados entre outros. Há um log suprimido na aplicação;

c) `letra C:` São códigos que poderiam ser melhorados, alguma condicional que poderia ser feita de forma mais eficiente, um comentário que poderia explicar algum método etc.;

d) `letra D:` Cobertura de testes. Não foram implementados testes para esta aplicação;

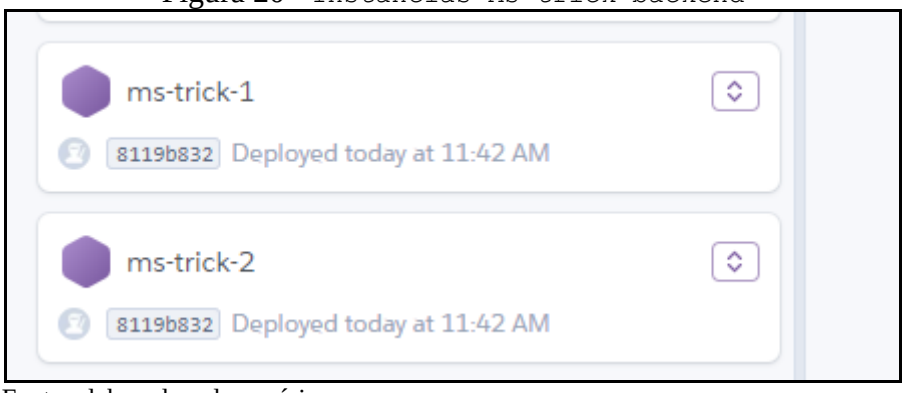
e) `letra E:` Códigos duplicados;

f) `letra F:` Linguagem de desenvolvimento da aplicação. Foram utilizados Java e XML.

3.3.3 Aplicativo de visualização: MS-Trick

O aplicativo foi desenvolvido com intuito de facilitar e centralizar a visualização dos *microservices*. O *backend* deste aplicativo utiliza a implementação Spring Boot para sua inicialização, como pode ser visto pelo Quadro 13, aproveitando a arquitetura de *microservices*. Como pode ser visualizado na Figura 20 foram utilizadas duas instâncias do *backend* da aplicação para teste do aplicativo.

Figura 20 - `Instâncias Ms-trick backend`



Fonte: elaborado pelo usuário

A primeira instância `ms-trick-1` recebe grande parte das requisições, pois a segunda instância `ms-trick-2` possui uma carga maior enviando notificações aos dispositivos registrados, quando o usuário faz *login* na aplicação. O código do envio de notificações pode ser visualizado no Quadro 16, este é responsável por criar uma requisição de notificação `linha 61`, buscar os dispositivos na `linha 69` e por fim enviar as notificações na `linha 82`.

Quadro 16 - Código `envio notificações Ms-trick`

```
59          public void sendNotification(String serviceName, String text) {
60
61                  HttpClient client = HttpClientBuilder.create().build();
62                  HttpPost post = new HttpPost(API_URL_FCM);
63                  post.setHeader("Content-type", "application/json");
64                  post.setHeader("Authorization", String.format("key=%s", AUTH_KEY_FCM));
65
66                  if (getDevices() != null && !getDevices().isEmpty()) {
67                          JSONObject message = new JSONObject();
68                          try {
69                                  message.put("registration_ids", new JSONArray(getDevices()));
70                                  message.put("priority", "high");
71
72                                  JSONObject notification = new JSONObject();
73                                  notification.put("title", serviceName);
74                                  notification.put("body", text);
75                                  notification.put("sound", "notification");
76                                  notification.put("vibrate", 1);
77                                  notification.put("icon", "ionic");
78
79                                  message.put("notification", notification);
80
81                                  post.setEntity(new StringEntity(message.toString(), "UTF-8"));
82                                  HttpResponse response = client.execute(post);
83
84                                  Notification sentNotification = new Notification();
85                                  sentNotification.setDevices(repo.findAll());
86                                  sentNotification.setMessage(String.format("Title: %s; Message: %s", serviceName, text));
87                                  sentNotification.setDateSent(new Date());
88                                  notificationRepo.save(sentNotification);
89
90                                  System.out.println(response);
91                                  System.out.println(message);
92                          } catch (JSONException | IOException e) {
93                                  logger.error("Parsing", e);
94                          }
95                  }
```

Fonte: elaborado pelo autor.

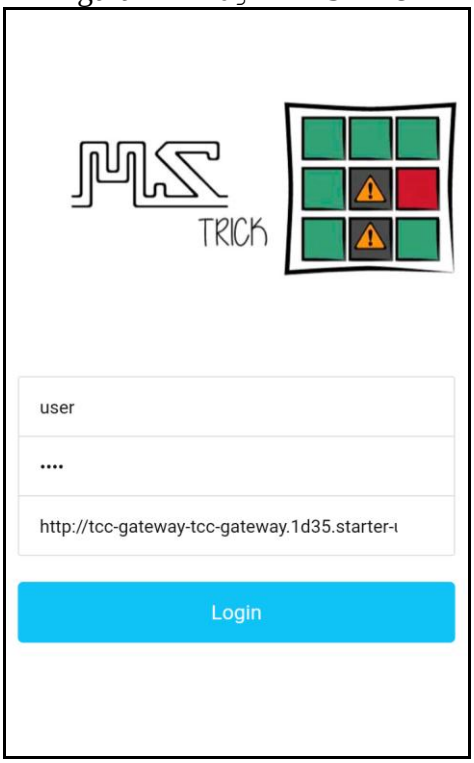
Para que fosse possível este comportamento diferenciado entre instâncias, foi criada uma configuração que pode ser adicionada nas variáveis de ambiente da aplicação. Como pode ser visualizado na Figura 21, ao adicionar a variável de ambiente `SERVERNOTIFY` com o valor `false` a instância não enviará mais notificações, com o valor `true` serão enviadas notificações aos dispositivos. Desta forma, ainda que aplicações com códigos iguais, é possível comportamentos diferenciados por instância.

Figura 21 - Configuração de notificações MS-trick

Fonte: elaborado pelo autor.

O aplicativo foi desenvolvido com o *framework* Ionic, responsável por gerar aplicativos Android e IOS. As requisições deste aplicativo ao *backend* são realizadas via HTTP e realizadas por Api Gateway e a URL do Gateway pode ser configurada na tela de *login* sendo possível visualizar na Figura 22. Na tela de login também pode-se o usuário e senha, como atualmente a aplicação não possui nenhum comportamento específico por usuário, há apenas um usuário sendo esse configurado nas configurações da aplicação pelo `ConfigServer`.

Figura 22 - `Login` MS-Trick



Fonte: elaborado pelo autor.

No Quadro 17 da tela de instância de um aplicativo é demonstrada que a implementação se dá apenas pelo desenvolvimento de HyperText Markup Language (HTML), Javascript e CSS como um *website* qualquer, este código é transformado num aplicativo durante a compilação. Ainda visualizando o Quadro 17 é possível perceber que a tela se dá por dois arquivos `app.html` e `controlers.js`. Toda tela do aplicativo é composta desta forma, o arquivo .html sendo a tela em si e o arquivo.js as `requisições` e ações desta tela. Na Figura 23 pode-se visualizar a tela gerada por este código, sendo possível visualizar algumas das Heurísticas de Nielsen, destaque para H2, H7, H8 e H9, descritas na seção 2.4.

Quadro 17 - Código tela de `instância de um aplicativo`

app.html

ion-view | ion-content.has-header | div | div.item.item-body

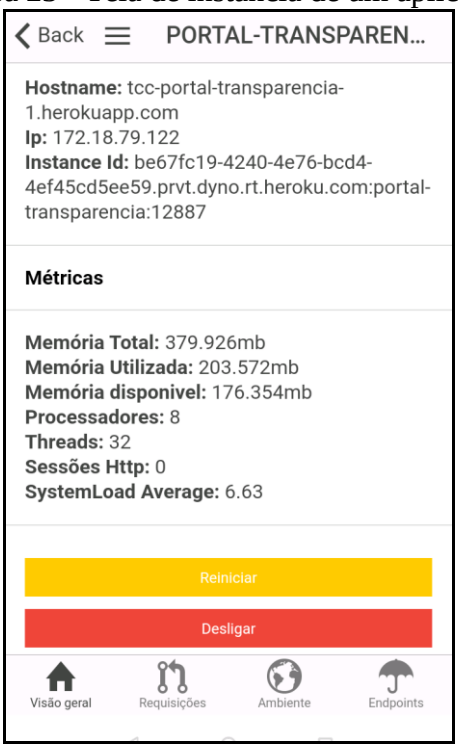
```
<ion-view title="{{app.applicationName}}">
  <ion-nav-buttons side="left">
    <button menu-toggle="left" class="button button-icon icon ion-navicon"></buttor
  </ion-nav-buttons>
  <ion-content class="has-header" style="margin-bottom: 40px">
    <div ng-if="menu == 'home'">
      <div class="item">
        <h2><b>{{app.applicationName}}</b></h2>
        <p><u>Ativo desde: {{app.upSince | date : 'dd/MM/yyyy - H:ss'}}</u></p>
      </div>

      <div class="item item-body">
        <b>Hostname:</b> {{app.host}}<br>
        <b>Ip:</b> {{app.ipAdress}}<br>
        <b>Instance Id:</b> {{app.instanceId}}
      </div>
```

app.js | controllers.js

```
.controller('AppCtrl', function ($scope, $state, $http, URLService, $stateParams,
  $scope.getKeys = function (object) {
    return Object.keys(object);
  }
  $scope.toggleGroup = function (group) {
    group.show = !group.show;
  };
  $scope.isGroupShown = function (group) {
    return group.show;
  };
  $scope.app = JSON.parse($stateParams.app);
  $scope.menu = 'home';
  $scope.restart = function (app) {
    $ionicLoading.show();
    $http.post(URLService.buildUrl('/thirdPartyPost'), buildThirdPartyRequest(wir
      setTimeout(function () {
```

Fonte: elaborado pelo autor.

Figura 23 - Tela de instância de um aplicativo



Fonte: elaborado pelo autor.

Na Figura 23 é possível visualizar algumas das métricas disponibilizadas no aplicativo, são estas:

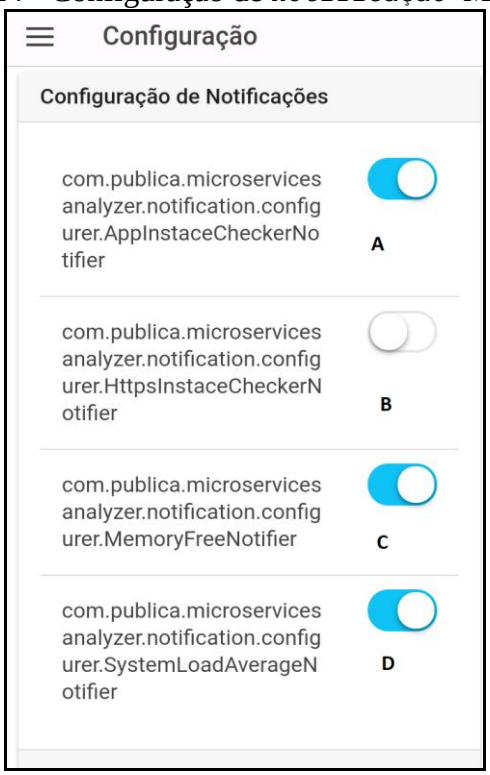
a) `Memória total`: representa a memória total que a instância possui para ser executada;
b) `Memória utilizada`: memória atualmente utilizada pela instância;
c) `Memória disponível`: (`Memória total`) – (`Memória utilizada`);
d) `Processadores`: representa o número de processadores que a máquina onde a instância está sendo executada possui;
e) `Threads`: número de threads máximo que pode ser utilizado pela aplicação;
f) `Sessões HTTP`: requisições ativas no momento;
g) `System Load Average`: métrica disponível em sistemas operacionais Linux, caso este número dividido pelo número de processadores seja superior a um (1), medidas devem ser tomadas.

As notificações podem ser configuradas no servidor, porém além da configuração no servidor, o usuário final pode optar por não receber tais notificações. Por esse motivo dentro das configurações do aplicativo visível na Figura 24 foi adicionada a possibilidade de habilitar e desabilitar notificações específicas ou todas elas. Caso elas não sejam desabilitas, o usuário receberá notificações no seu dispositivo. Pode-se visualizar na Figura 24 as possíveis notificações enviadas pela aplicação, sendo essas:

a) `letra A` – O `AppInstanceCheckNotifier` verifica o número de instâncias ativas do *microservice*, caso o número de instâncias seja inferior a 2, será enviada uma notificação aos dispositivos;
b) `letra B` – O `HttpsInstanceCheckNotifier` verifica se o *microservice* está sendo executando num domínio *HTTPS*, será enviada uma notificação aos dispositivos caso não esteja;
c) `letra C` – O `MemoryFreeNotifier` envia uma notificação aos dispositivos caso a memória do *microservice* seja inferior a 200mb;
d) `letra D` – O `SystemLoadAverageNotificer` verificar a métrica *System Load Average* do sistema operacional *Linux*, caso esta apresente um número superior ao número de processadores disponíveis uma notificação será enviada.

Figura 25.

Figura 24 - Configuração de `notificação` MS-Trick



Fonte: elaborado pelo autor.

Pode-se visualizar na Figura 24 as possíveis notificações enviadas pela aplicação, sendo essas:

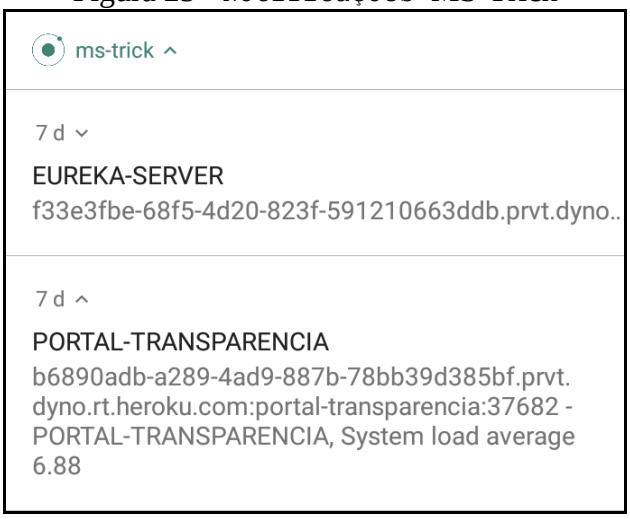
e) `letra A` – O `AppInstanceCheckNotifier` verifica o número de instâncias ativas do *microservice*, caso o número de instâncias seja inferior a 2, será enviada uma notificação aos dispositivos;

f) `letra B` – O `HttpsInstanceCheckNotifier` verifica se o *microservice* está sendo executando num domínio *HTTPS*, será enviada uma notificação aos dispositivos caso não esteja;

g) `letra C` – O `MemoryFreeNotifier` envia uma notificação aos dispositivos caso a memória do *microservice* seja inferior a 200mb;

h) `letra D` – O `SystemLoadAverageNotificer` verificar a métrica *System Load Average* do sistema operacional *Linux*, caso esta apresente um número superior ao número de processadores disponíveis uma notificação será enviada.

Figura 25 - `Notificações` MS-Trick

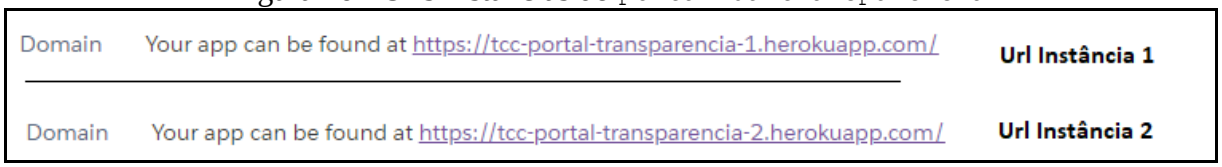


Fonte: elaborado pelo autor.

3.3.4 Operacionalidade correlacionada da arquitetura

Nesta subseção será demonstrada a arquitetura executada como um todo, sendo possível visualizar o que estava sendo buscado com este trabalho. Como exemplo será utilizado o portal da transparência. O processo se inicia com integração contínua, assim que algum desenvolvedor submete qualquer código a algum repositório o aplicativo é compilado. Logo em seguida é realizado o *deploy* da aplicação. O *deploy* pode ser realizado para uma ou infinitas instâncias, dependendo da necessidade dos usuários. Neste exemplo utilizamos duas instâncias como pode ser visualizado na Figura 26, cada uma das instâncias possui um endereço diferente.

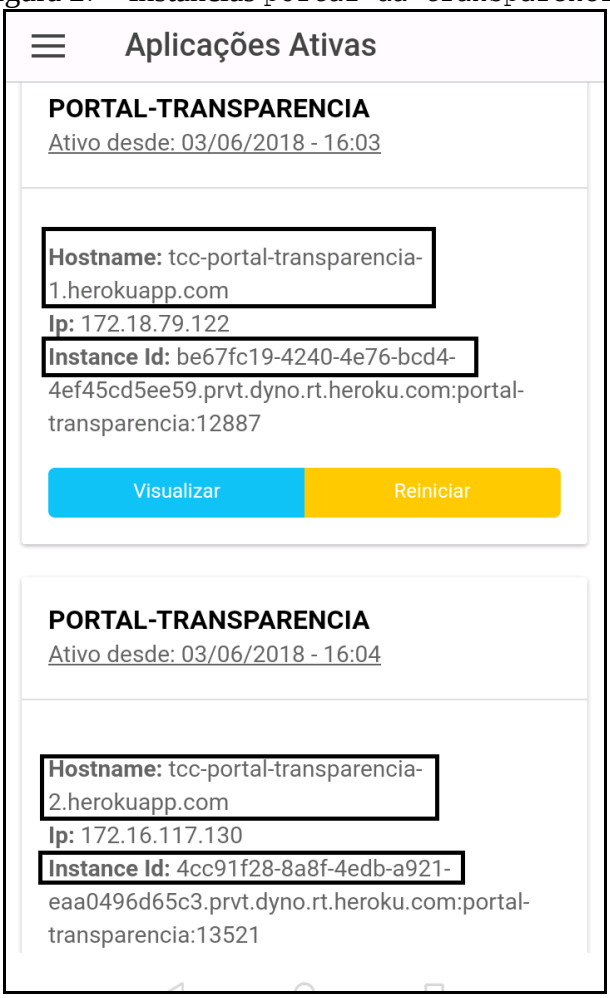
Figura 26 - Urls instâncias do `portal da transparência`



Fonte: elaborado pelo autor.

Durante a inicialização do *microservice* cada instância se auto registra no `Service Registry` da aplicação. Durante o mesmo processo instância seleciona suas propriedades do `ConfigServer` de acordo com seu ambiente de execução. Após a inicialização a aplicação está registrada no `Service registry`, sendo possível visualiza-la no aplicativo de controle conforme a Figura 27.

Figura 27 - Instâncias `portal da transparência`

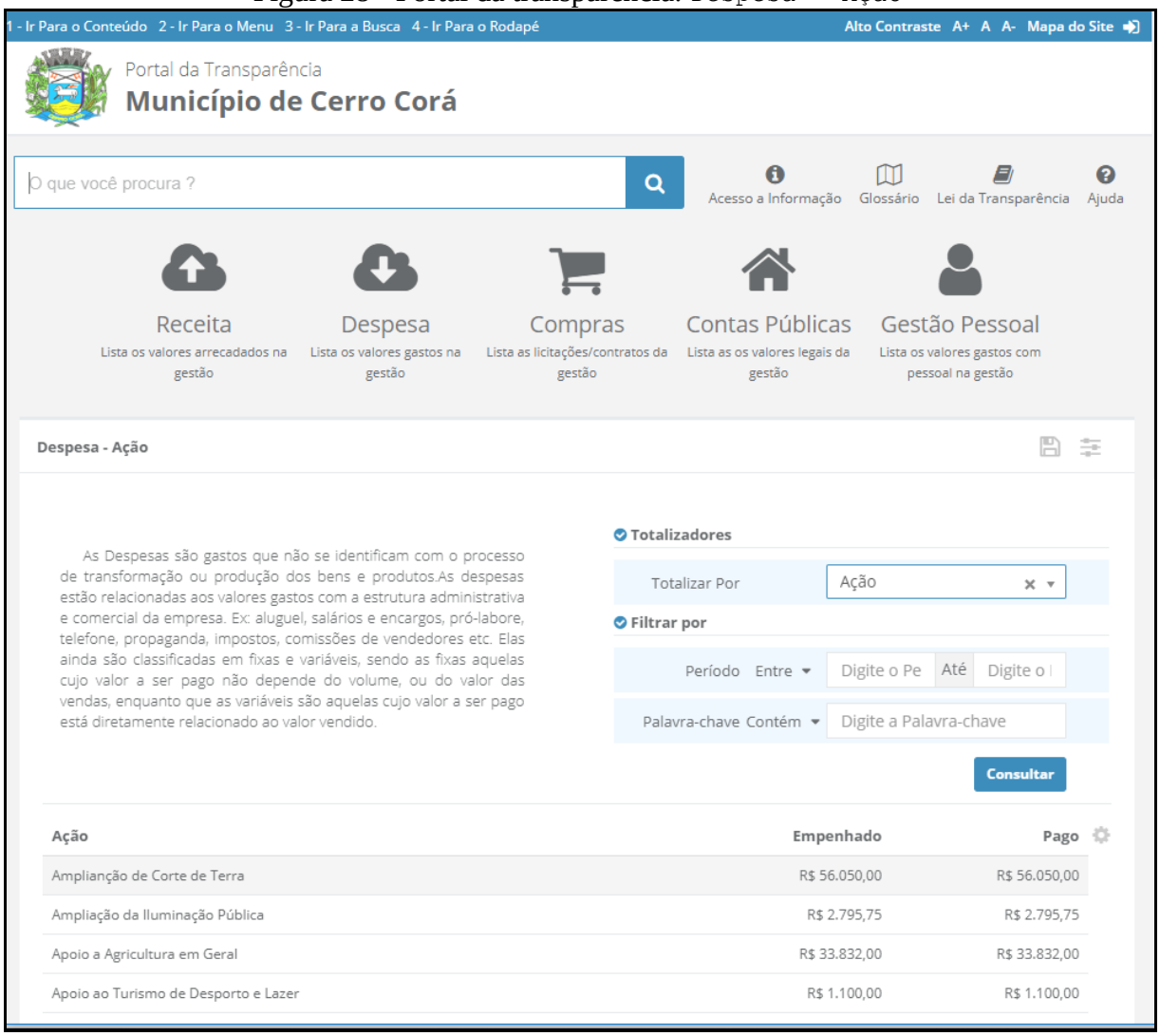


Fonte: elaborado pelo autor

Na Figura 27 é possível visualizar ambas instâncias, cada qual com seu `Instance Id`, identificação gerada pelo `Service Registry;` e sua URL (`Hostname`), possibilitando acessar o *microservice* diretamente. Como nenhuma das instâncias é acessada diretamente, ambas são acessadas pelo Api Gateway disponível na URL[18]. Pode-se dizer que ao acessar qualquer tela do portal da transparência, utilizaremos `Despesa – Ação`, visível na Figura 28.

[18] Api Gateway disponível na URL: <http://tcc-gateway-tcc-gateway.1d35.starter-us-east-1.openshiftapps.com/gateway/portal>.

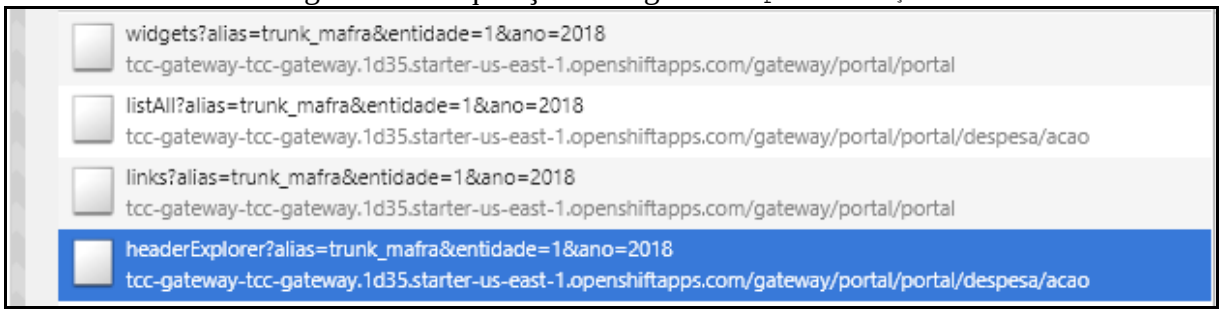
Figura 28 - Portal da transparência: `Despesa - Ação`



Fonte: elaborado pelo autor.

Para demonstrar tal listagem de Despesa - Ação da Figura 28 são realizadas diversas requisições, demonstradas na Figura 29. Todas são realizadas para o mesmo host do Api Gateway. Além da URL do gateway todas as requisições adicionam o *endpoint* requisitado como visível na linha selecionada na Figura 29, essa adicionando `/portal/despesa/acao`.

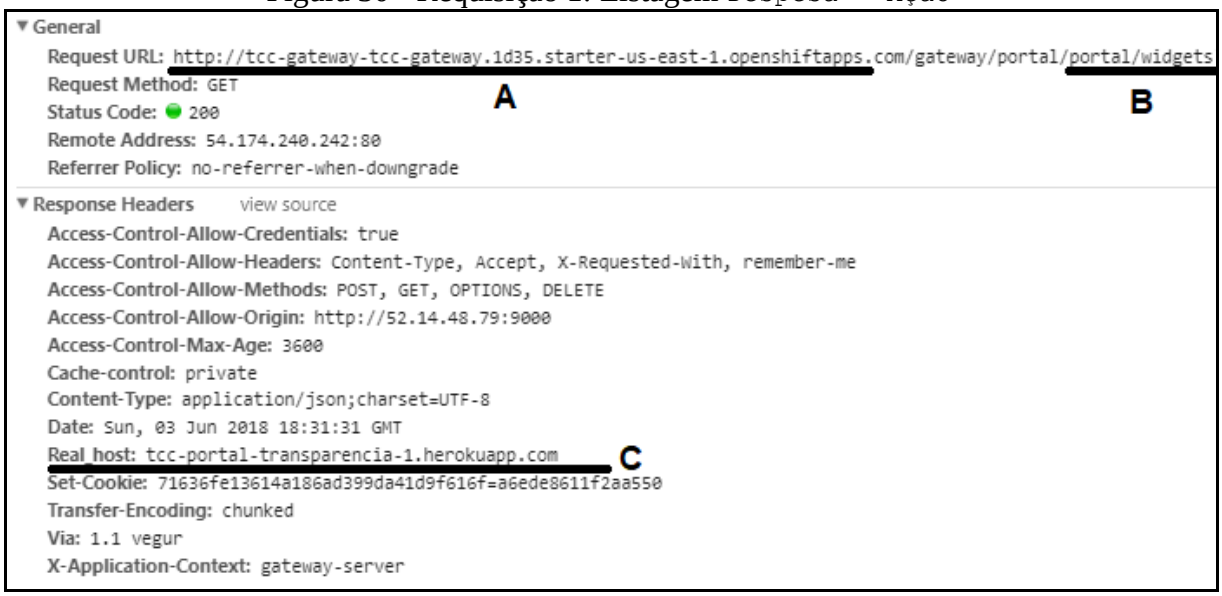
Figura 29 - Requisições listagem: `Despesa - Ação`



Fonte: elaborado pelo autor.

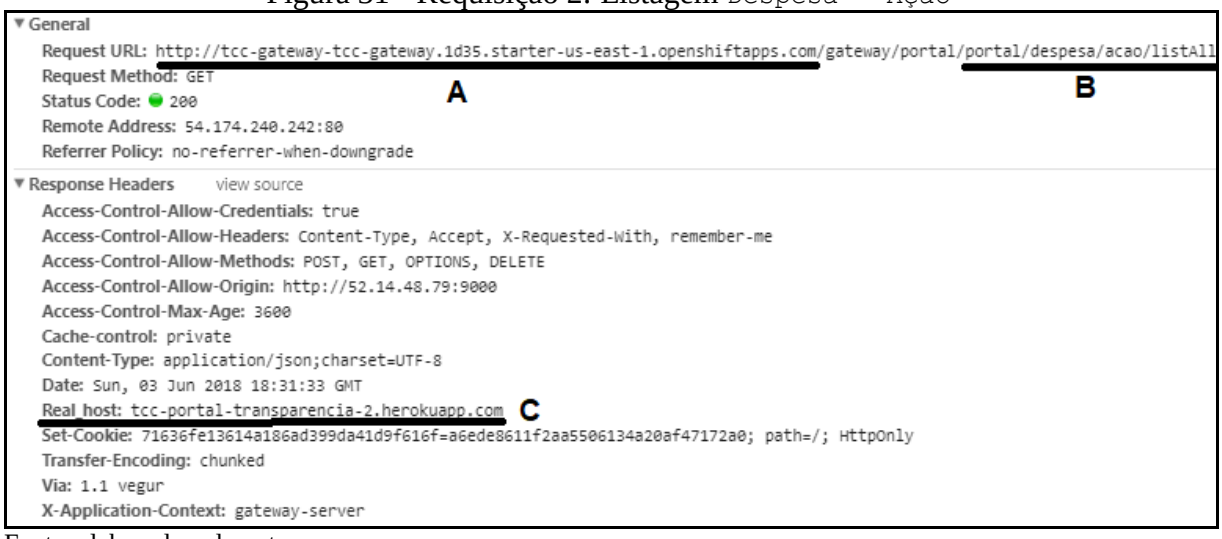
Ainda que as requisições da Figura 29 sejam feitas para um mesmo Host, cada uma delas pode ter sido processada por uma instância diferente. Nas Figura 30 e Figura 31 são demonstradas as duas primeiras requisições da Figura 29, ambas realizadas para o host do `Api Gateway`, porém cada uma possui um `Real_host` diferente.

Figura 30 - Requisição 1: Listagem `Despesa - Ação`



Fonte: elaborado pelo autor.

Figura 31 - Requisição 2: Listagem `Despesa - Ação`



Fonte: elaborado pelo autor.

O `Real_host` das Figura 30 e Figura 31 (`letra C`) representa a instância que processou a requisição, pode-se visualizar que cada uma das respostas possui uma URL das instâncias demonstradas na Figura 26. A `letra A` demonstra o host requisitado, sendo este o `Api gateway`; a letra B demonstra o *endpoint* requisitado. Ainda é possível observar por este balanceamento realizado, pela execução de duas instâncias, nenhuma delas é sobrecarregada, e ainda, se alguma das duas instâncias vier a apresentar alguma falha de execução, a segunda instância estará lá de forma totalmente independente.

3.4 RESULTADOS E DISCUSSÕES

Na seção 3.4.1 será realizada a comparação do trabalho desenvolvido com os trabalhos correlatos, bem como a avaliação das partes da arquitetura na subseção 3.4.2.

3.4.1 Comparação entre o trabalho desenvolvido (MS-TRICK) e os trabalhos correlatos

É realizada nesta subseção uma comparação entre os trabalhos correlatos apresentados na seção 2.5 e a arquitetura desenvolvida neste trabalho, sendo possível visualizar pelo Quadro 18 as características dos trabalhos correlatos relacionadas com a arquitetura desenvolvida no MS-Trick. Observa-se pelo quadro comparativo que o trabalho alcançou seus objetivos, em virtude de todas as características destacadas dos trabalhos correlatos terem sido implementadas, bem como a linguagem de programação utilizada, Java, a mais comum entre os trabalhos correlatos.

Quadro 18 - Comparativo da correlação entre os trabalhos correlatos e o MS-Trick

Trabalhos / Características	Tonse (2014)	Silva (2015)	São Miguel e Farina (2016)	MS-Trick
Escalabilidade	✓	✓	X	✓
Integração contínua	✓	X	X	✓
Disponibilidade	✓	✓	X	✓
Métricas exclusivas	X	X	✓	✓
Modularidade	✓	✓	X	✓
Linguagem de programação	Java	Java	PHP	Java
Melhoria no processo de manutenção	✓	✓	X	✓

Fonte: elaborado pelo autor.

Cada uma das características destacadas pôde ser alcançada de diferentes formas com o trabalho desenvolvido. A `Escalabilidade` foi alcançada com a possibilidade de execução do número de instâncias necessárias para suprir a necessidade dos usuários, está é realizada nos trabalhos de Tonse (2014) e Silva (2015); enquanto a `Integração contínua` foi alcançada da mesma forma que o trabalho de Tonse (2014), implementação de conceitos de DevOps; pela implementação de um script para Ferramenta Gitlab, é possível realizar a compilação, o *deploy* e a avaliação do código da aplicação.

A `Disponibilidade` está diretamente relacionada com a `Escalabilidade`; como no trabalho de Tonse (2014), no MS-Trick é possível aumentar o número de instâncias executantes, tornando a aplicação cada vez mais disponível. Como no trabalho de São Miguel e Farina (2016) `Métricas exclusivas` foram adquiridas pela implementação da arquitetura, estas foram disponibilizadas no aplicativo móvel implementado para melhor visualização da

arquitetura. A `Modularidade` como nos trabalhos de Tonse (2014) e Silva (2015), foi alcançada pela implementação de módulos independentes das demais partes do sistema.

Como no trabalho de Tonse (2014) e Silva (2015) foi utilizada Java como `Linguagem de programação` para adaptação do código existente e criação do novo código; por fim a `Melhoria no processo de manutenção`, como de Tonse (2014) e Silva (2015), é trazida pela a possibilidade de diferentes desenvolvedores trabalharem nos módulos individuais da aplicação, bem pela velocidade trazida pela `Integração contínua`. Cabe destacar, as configurações centralizadas pelo `ConfigServer` no `MS-Trick`, permitindo uma configuração única para todos os *microservices*.

3.4.2 Avaliação de usabilidade

A avaliação de usabilidade teve como intuito comprovar a eficiência da aplicação MS-Trick, assim como garantir que ela tenha sido implementada de acordo com objetivos estabelecidos nesta monografia, tendo como base a experiência do usuário em sua utilização. A avaliação foi realizada por meio de questionário elaborado de forma a obter respostas de forma quantitativa e qualitativa; e de maneira on-line. A ferramenta utilizada para realização do questionário foi o Google Formulários[19] e disponibilizada para usuários especialistas da avaliação pela Ferramenta Skype[20], com acompanhamento do desenvolvedor do trabalho.

A avaliação utilizou princípios do método Relationship of M3C with User Requirements and Usability and Communicability Assessment in groupware (M3C-URUCAg), que aborda: como as perguntas devem ser elaboradas, relacionando os requisitos do sistema com as heurísticas de Nielsen (seção 2.4), bem como ao momento que a pesquisa deve ser aplicada. Neste sentido, no estudo maior de Costa (2018b) é observado que o grau de satisfação do usuário pode ser melhor capturado após ele ter ocorrido. Destaca-se, que para que os usuários especialistas tivessem um melhor entendimento do que se tratava tanto a avaliação quanto o sistema, foi disponibilizado um roteiro a ser seguido na avaliação, guiando os participantes no uso do sistema MS-Trick.

Foi utilizada uma amostra de três usuários especialistas da empresa Pública informática, familiarizados com a linguagem empregada nesta aplicação. A forma de disponibilizar a pesquisa para os usuários especialistas foi por meio da Ferramenta Skype com um link para acesso a uma página do Google Formulários, contendo: o Termo de

---

[19] Ferramenta Google Formulários: Disponível em: <https://docs.google.com/forms/>. Acesso em: 19 jun. 2018.
[20] Ferramenta Skype: Disponível em <https://www.skype.com>. Acesso em: 25 jun. 2018.

Consentimento Livre e Esclarecido (TCLE) (Apêndice A), convidando o usuário especialista a colaborar com a pesquisa, os possíveis danos e os direitos ao participar da avaliação. Ademais, o TCLE contém uma orientação que caso ele prosseguisse para a seção do roteiro (Apêndice B), aceitava o termo estabelecido. Ao prosseguir, o formulário exibia um roteiro disponibilizando o link da aplicação MS-Trick e os passos para serem seguidos para melhor conhecerem melhor a aplicação. No final do roteiro, caso o usuário especialista se dirigisse para a próxima seção ele aceitava responder o questionário de avaliação.

A avaliação foi dividida em duas partes: a primeira parte dizendo respeito a identificação dos usuários especialistas e a segunda parte relacionada a avaliação de usabilidade, referente a arquitetura de *microservices* e das características do aplicativo desenvolvido. Ainda na primeira parte são buscadas vantagens visualizadas pelo usuário especialista, relacionadas a aplicação da arquitetura no contexto existente.

A segunda parte do questionário diz respeito a avaliação de usabilidade e experiência do usuário, composta por dez perguntas. O objetivo das perguntas de usabilidade do sistema foi identificar a satisfação do usuário em relação a utilização dos mecanismos fornecidos pela aplicação. As perguntas foram elaboradas utilizando os fundamentos das heurísticas de Nielsen (1994), dispostas na seção 2.4, bem como está disponível a escala de severidade para as heurísticas. No Quadro 19 é possível visualizar a relação das perguntas relacionadas com sua respectiva heurística.

Quadro 19 - Relação das perguntas com as Heurísticas de Nielsen

NRO	**PERGUNTAS DA AVALIAÇÃO**
H1	É possível visualizar o estado atual do sistema (Processos, consumos de recursos, etc.)?
H2	Aplicação utiliza linguagem do usuário (usuário de TI)?
H3	Aplicação permite reverter e/ou saídas alternativas?
H4	Aplicação utiliza padrões capaz de familiarizar o usuário com sistema?
H5	Aplicação utiliza padrões capaz de prevenir erros do usuário?
H6	Aplicação facilita a lembrança do usuário, não necessitando de lembrança de passos ou ações?
H7	Aplicação eficiente e eficaz evitando frustração do usuário?
H8	Aplicação tenta reduzir demonstração de informação irrelevante?
H9	Aplicação apresenta documentação e ajuda?
H10	Aplicação demonstrada de forma facilitada para interação do usuário, permitindo flexibilidade e eficiência no uso?

Fonte: elaborado pelo autor.

Tratando-se da `vantagem trazida pela utilização da arquitetura de microservices` todos os participantes conseguiram visualizar as vantagens trazidas pela utilização da arquitetura, conforme pode ser observado pela Figura 32. Ficou claro que a

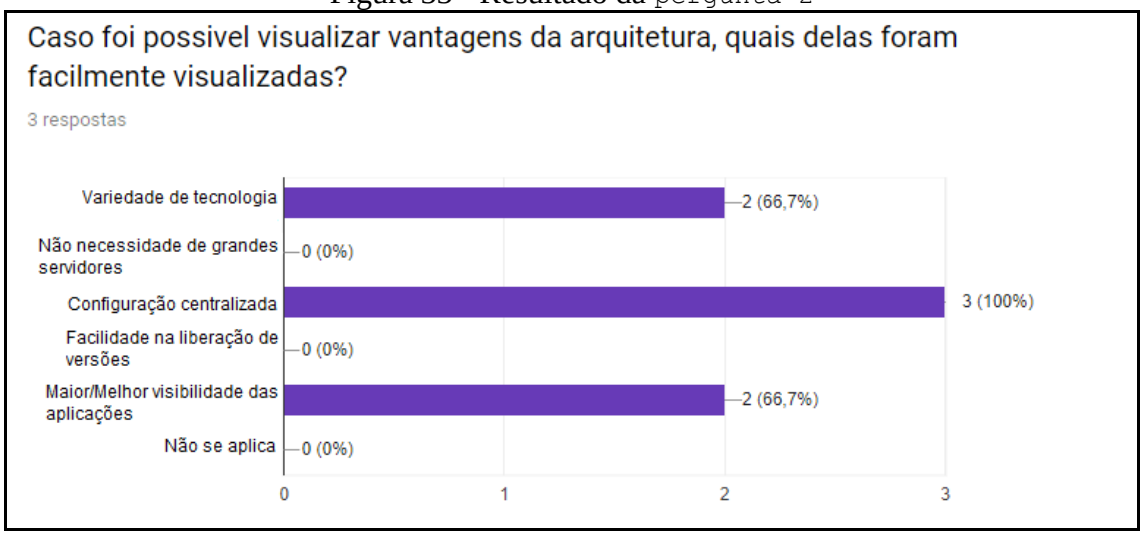
arquitetura contribui para resoluções de problemas anteriormente encontrados no sistema atual.

Figura 32 - Resultado da `pergunta 1`

Fonte: elaborado pelo autor.

Como a arquitetura ainda não está sendo executada em produção algumas das características como `escalabilidade` e `disponibilidade` apresentam dificuldade de verificação. Ademais na Figura 33 são demonstradas algumas das vantagens visualizadas pelos usuários da avaliação. Como foram apresentadas partes da arquitetura, as características mais visualizadas pelos usuários foram `Variedade de tecnologia`, `Configuração centralizada`, e `Maior/Melhor visibilidade das aplicações`. Destaque para `configuração centralizada`, este não foi levantado como um grande problema na seção 2.1, porém todos os participantes demonstraram interesse na implementação do mesmo justificando ser muito útil para problemas rotineiros relacionados a configurações.

Figura 33 - Resultado da `pergunta 2`

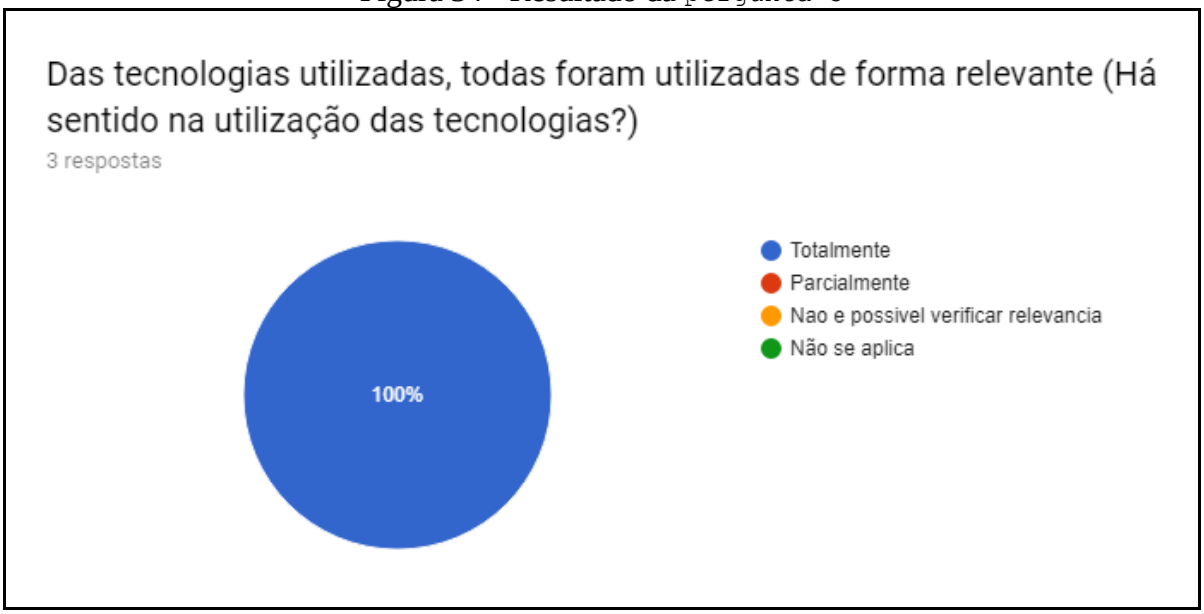


Fonte: elaborado pelo autor.

A arquitetura como um todo necessita de diversas tecnologias para sua execução. Por esse motivo, a pergunta demonstrada na Figura 34 foi realizada, fazendo-se visível que os

participantes puderam visualizar a relevância na utilização das tecnologias. Todas as tecnologias foram utilizadas de maneira a agregar à arquitetura segundo os participantes.

Figura 34 - Resultado da `pergunta 3`



Fonte: elaborado pelo autor.

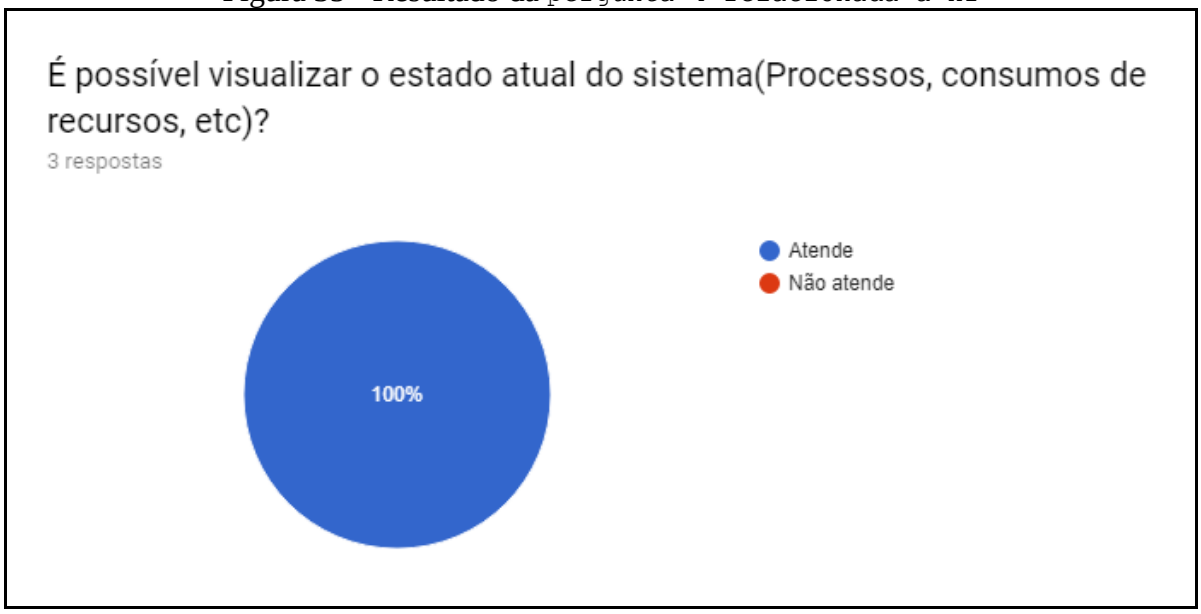
A parte das perguntas da avaliação a seguir estão relacionadas ao aplicativo implementado para melhor visualização da arquitetura, estando relacionadas com as heurísticas descritas na seção 2.4. Na coleta de informações baseadas nas respostas do questionário relacionada as heurísticas, os usuários especialistas levantaram alguns problemas relacionados as heurísticas `H4`, `H6`, `H9` e `H10`. Elencada por todos os participantes, a heurística `H10`, `o sistema possui opção de ajuda`, poderia ser melhor trabalhada na aplicação, podendo ter talvez alguma tela em específico com informações gerais sobre as funcionalidades da aplicação. Relacionada pelos participantes `Usuário Final` e `Analista de Sistema`, a heurística `H6`, `o sistema possui instruções, ações e opções visíveis ou facilmente recuperáveis` (sempre que apropriado para o uso), poderia dispor algumas descrições de funcionalidades, visando facilitar o entendimento delas.

Referente a heurística `H4`, `o sistema possui padrões e estilos consistentes`, um dos `Usuários especialistas` comentou que a aplicação poderia explorar mais a estética, deixando uma aparência mais harmônica. Ademais, relacionada pelos participantes `Usuários especialistas`, a `heurística H9`, `o sistema possui mensagens de erros com linguagem simples`, a aplicação poderia apresentar mensagens um pouco mais amigáveis para o usuário, facilitando seu entendimento.

Em seguida serão demonstrados detalhadamente os resultados obtidos por meio da avaliação. Para cada pergunta existe outra pergunta, relacionada a violação da Heurística, caso

a resposta seja `Não atende`. A primeira pergunta pode ser visualizada na Figura 35, sendo essa diretamente relacionada a `Heurística 1`, demonstrando que os estados da aplicação são claros aos usuários.

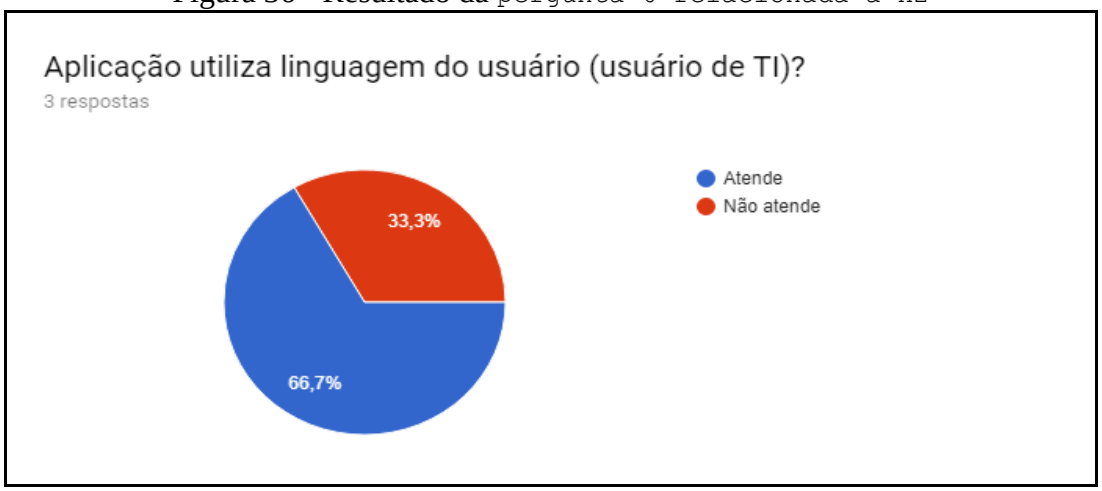
Figura 35 - Resultado da `pergunta 4 relacionada a H1`



Fonte: elaborado pelo autor.

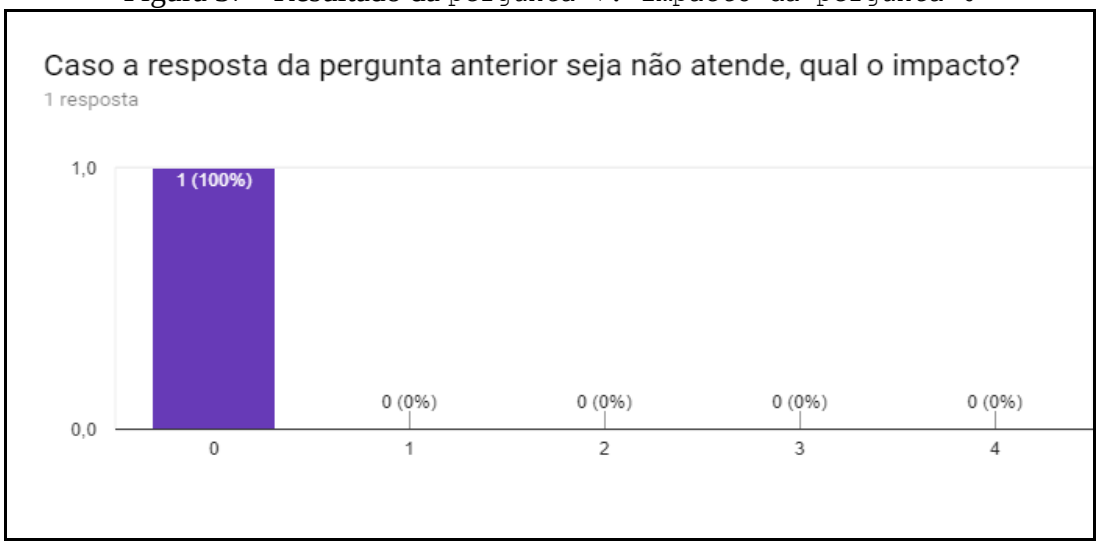
A segunda pergunta representa a `Heurística 2`, Figura 36, relacionada a `linguagem utilizada para o aplicativo`. Essa heurística se faz importante pois todo título ou conteúdo demonstrado deve ser compreendido pelo usuário. Dois dos usuários especialistas puderam visualizar a linguagem como comum entre usuários de TI, porém um dos usuários especialistas não se sentiu totalmente familiarizado. Apesar disso, um usuário considerou o problema como sem importância, demonstrado na Figura 37.

Figura 36 - Resultado da `pergunta 6 relacionada a H2`



Fonte: elaborado pelo autor.

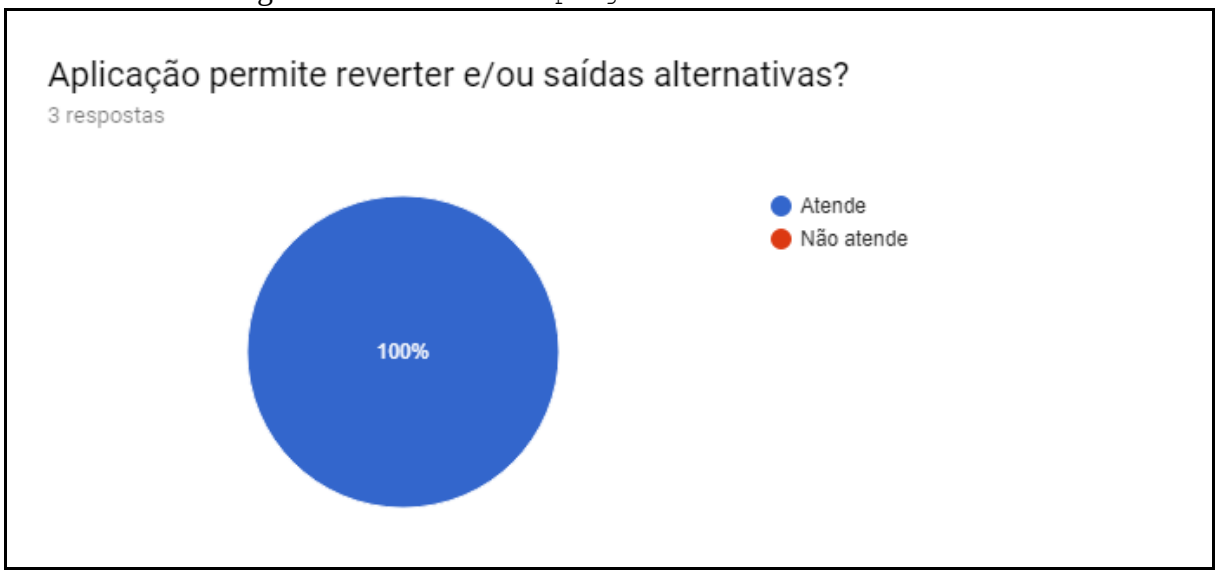
Figura 37 - Resultado da `pergunta 7: impacto da pergunta 6`



Fonte: elaborado pelo autor.

A pergunta relacionada à `Heurística 3` pode ser visualida na Figura 38. Nenhum dos usuários teve problemas em `refazer ou desfazer suas ações`. Alterações de configurações ou ações tomadas puderam ser facilmente revertidas. Demonstrando que aplicação permite saídas de emergência e *undo*/*redo*.

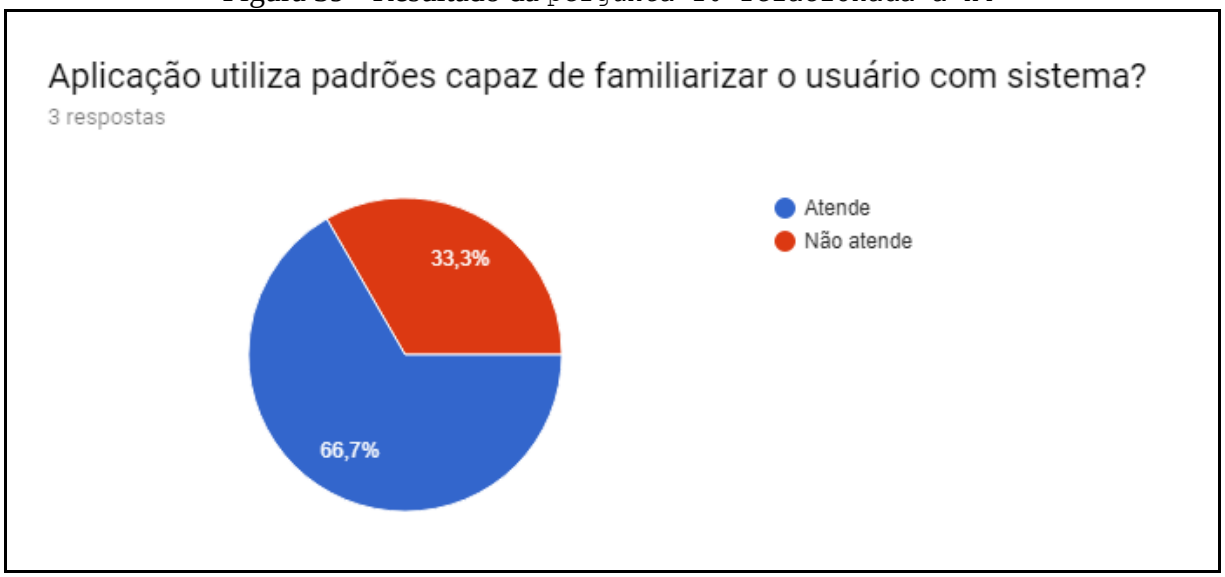
Figura 38 - Resultado da `pergunta 8 relacionada a H3`



Fonte: elaborado pelo autor.

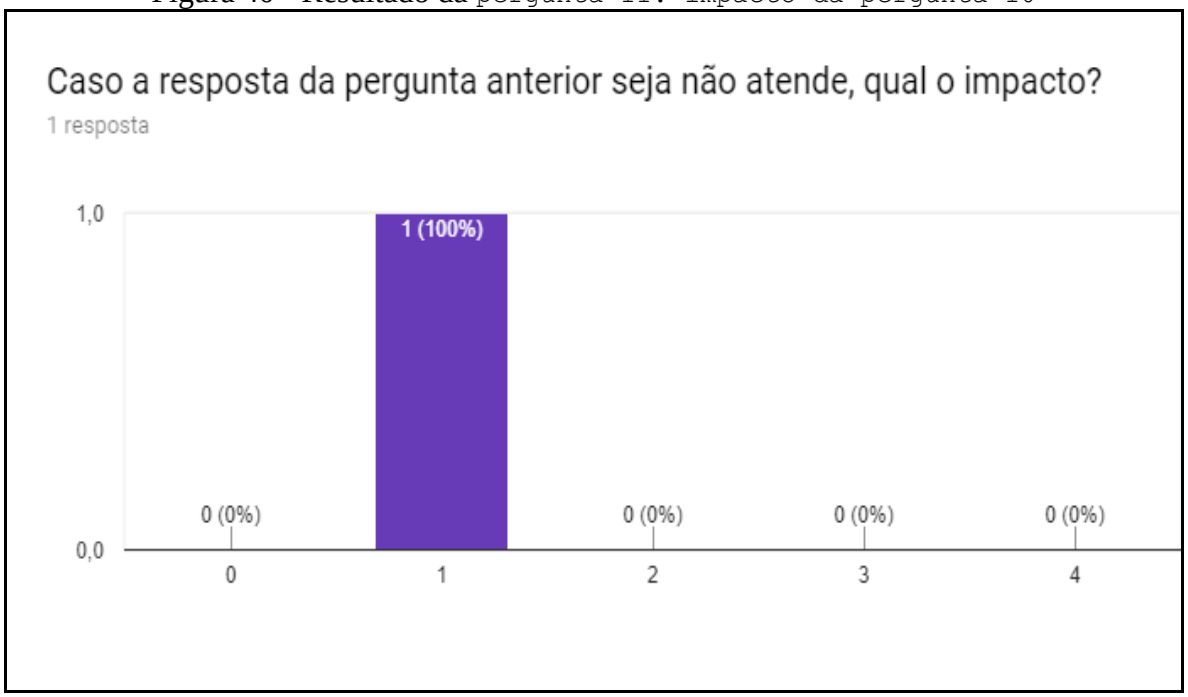
A `Heurística 4` é demonstrada na Figura 39, essa relacionada a padrões a `familiarizar os usuários com a utilização da aplicação`. Dois dos usuários especialistas se sentiram familiarizados com a aplicação, um dos usuários, porém, não se sentiu totalmente familiarizado. Mesmo que o usuário não tenha se sentido totalmente familiarizado, este considerou o problema apenas de interface, sendo visível na Figura 40.

Figura 39 - Resultado da pergunta 10 relacionada a H4



Fonte: elaborado pelo autor.

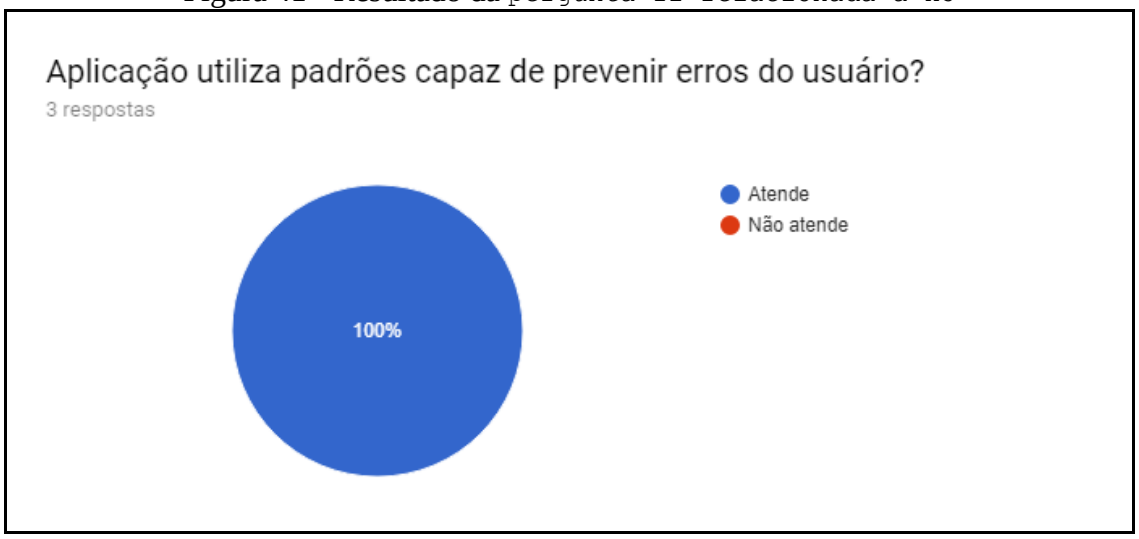
Figura 40 - Resultado da pergunta 11: impacto da pergunta 10



Fonte: elaborado pelo autor.

A Heurística 5 é demonstrada na Figura 41, sendo possível visualizar que nenhum dos usuários especialistas obteve problemas relacionadas a prevenção de erros. Sabendo que erros na arquitetura podem impactar usuários finais, a aplicação visa simplificação afim de prevenir erros. Desta forma os caminhos para realizar procedimentos do aplicativo não levam o usuário a cometer erros.

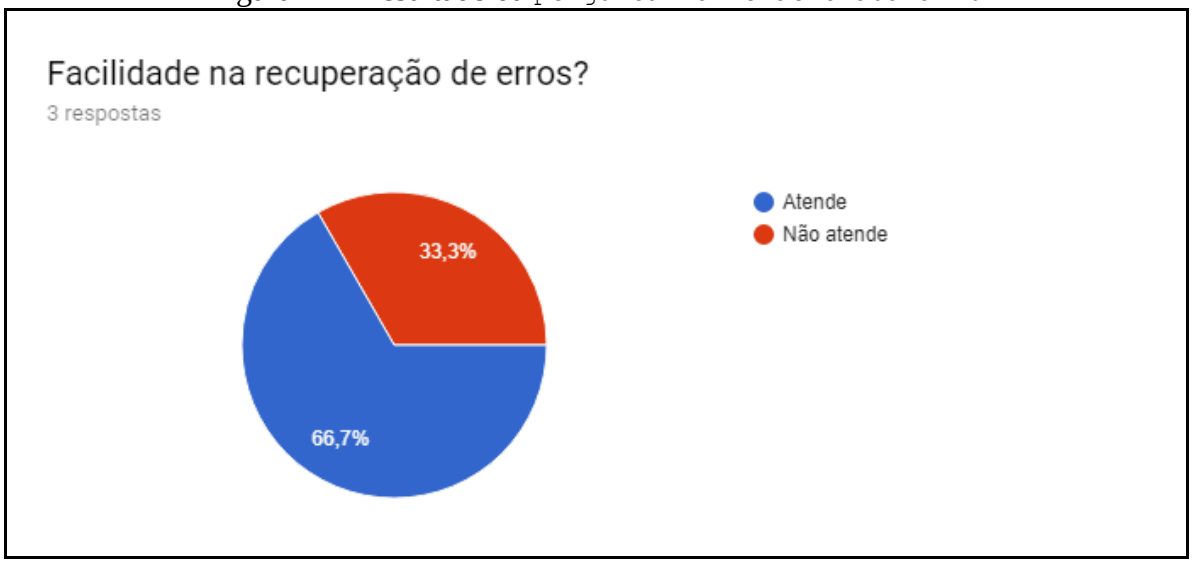
Figura 41 - Resultado da `pergunta 12 relacionada a H5`



Fonte: elaborado pelo autor.

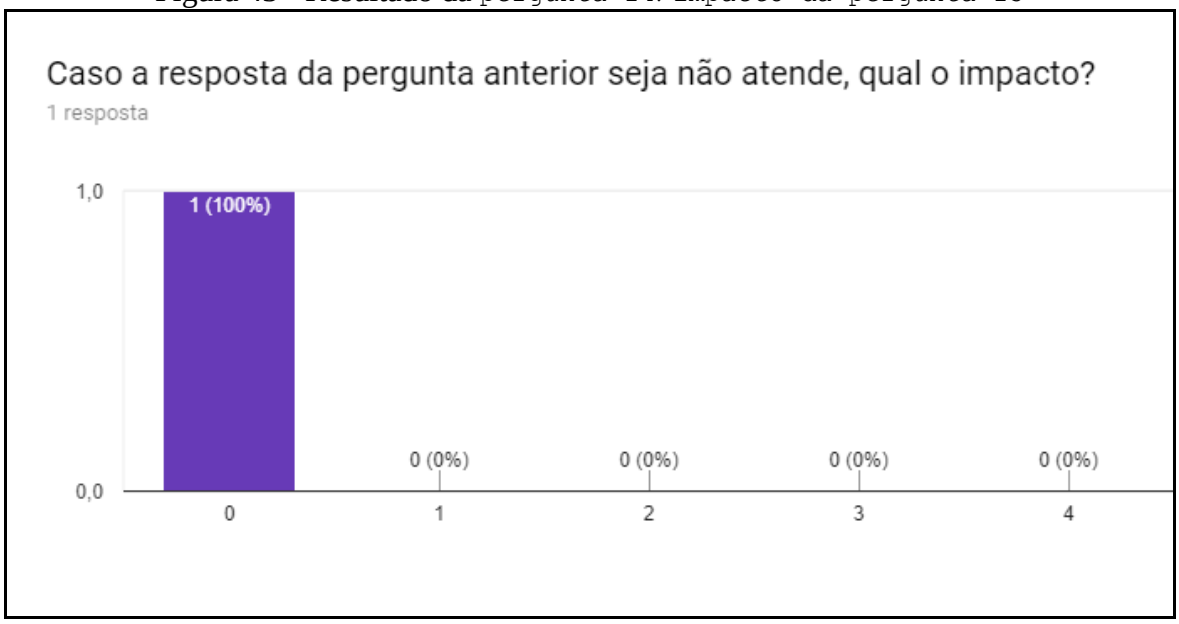
A `Heurística 6` é representada na Figura 42, sendo esta Heurística relacionada a `facilidade na recuperação de erros` realizados durante a utilização do aplicativo. Dois dos usuários especialistas não tiveram problemas com essa Heurística, porém um dos usuários especialistas relatou problemas. Mesmo encontrando problemas o usuário considerou o problema sem importância (Figura 43).

Figura 42 - Resultado da `pergunta 13 relacionada a H6`



Fonte: elaborado pelo autor.

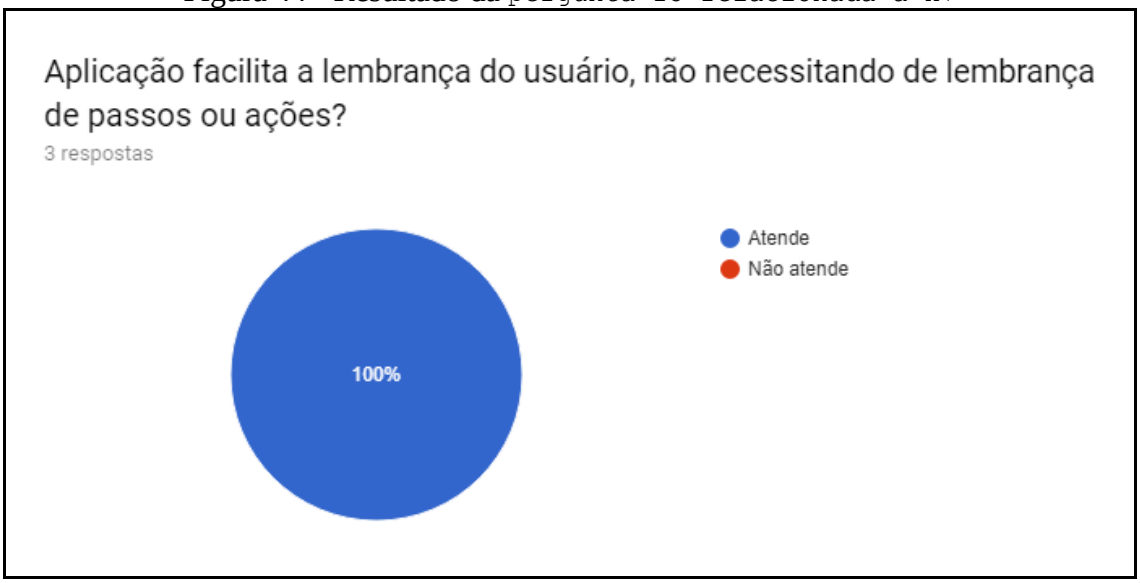
Figura 43 - Resultado da pergunta 14: impacto da pergunta 13



Fonte: elaborado pelo autor.

A Heurística 7 é demonstrada na Figura 44, essa Heurística está relacionada a lembrança e facilidade na realização de passos. A aplicação é facilitada a ponto que nenhum dos usuários especialistas teve problema com a realização dos passos necessárias para utilização da aplicação, bem como dificuldade de lembrança de passos e ações para avançar no uso.

Figura 44 - Resultado da pergunta 15 relacionada a H7



Fonte: elaborado pelo autor.

A Heurística 8 é demonstrada na Figura 45, essa relacionada com a eficiência da aplicação. É possível visualizar que todos os usuários especialistas visualizaram a aplicação eficiente e flexível para seu propósito. Ademais não foram apresentados gargalos ou lentidões durante os testes com os usuários. Listagens de *microservices* e notificações enviadas, por

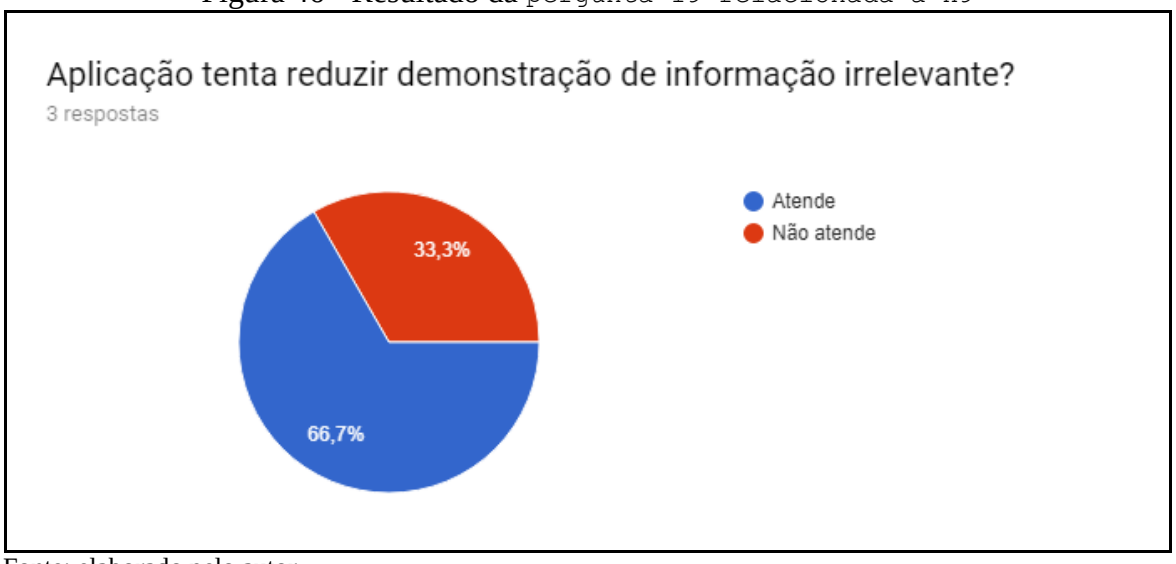
demonstrarem diversas informações poderiam representar problemas, porém essas se mostraram eficiente agradando a todos os participantes da avaliação.

Figura 45 - Resultado da `pergunta 17 relacionada a H8`

Fonte: elaborado pelo autor.

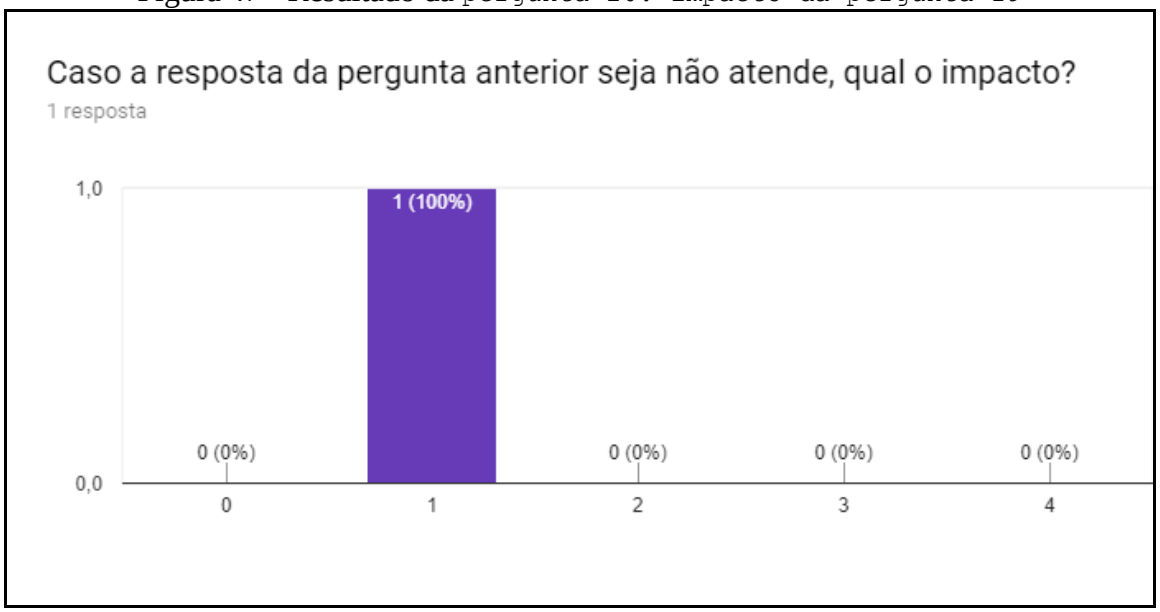
A `Heurística 9` pode ser visualizada na Figura 46, essa Heurística relacionada a `relevância das informações`. É possível visualizar Figura 46 que um dos usuários especialista considerou algumas das informações irrelevantes. Mesmo encontrando problemas nessa Heurísticas, o usuário reconheceu que isto acontece apenas devido a problemas na interface conforme demonstrado na Figura 47. A violação desta Heurística conforme conversa com o participante se deu pelo não compreendimento de uma das informações, fazendo com que a documentação para certos termos e ações se faça ainda mais importante

Figura 46 - Resultado da `pergunta 19 relacionada a H9`



Fonte: elaborado pelo autor.

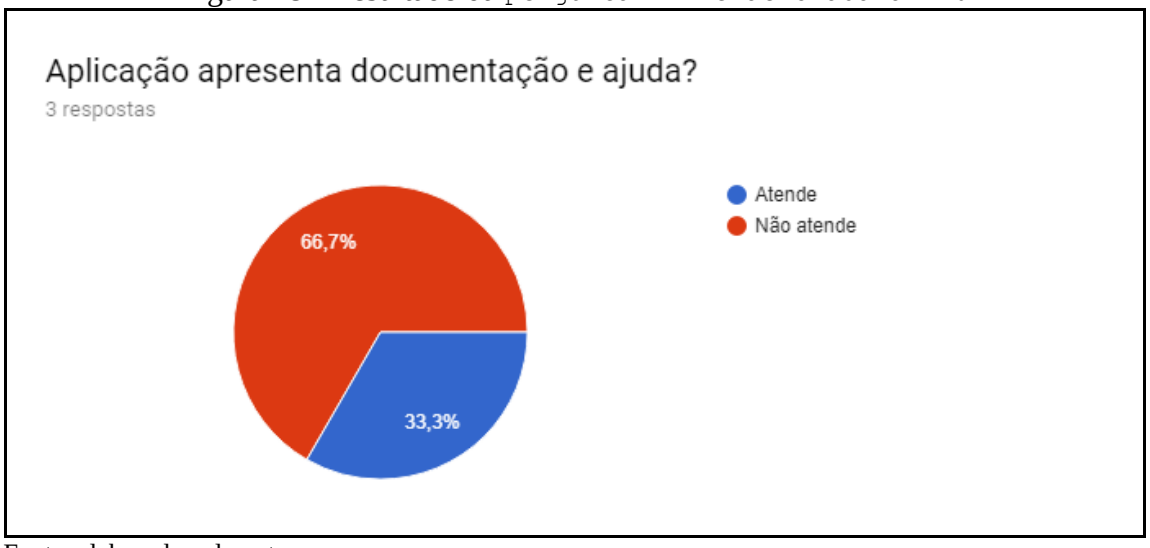
Figura 47 - Resultado da pergunta 20: impacto da pergunta 19



Fonte: elaborado pelo autor.

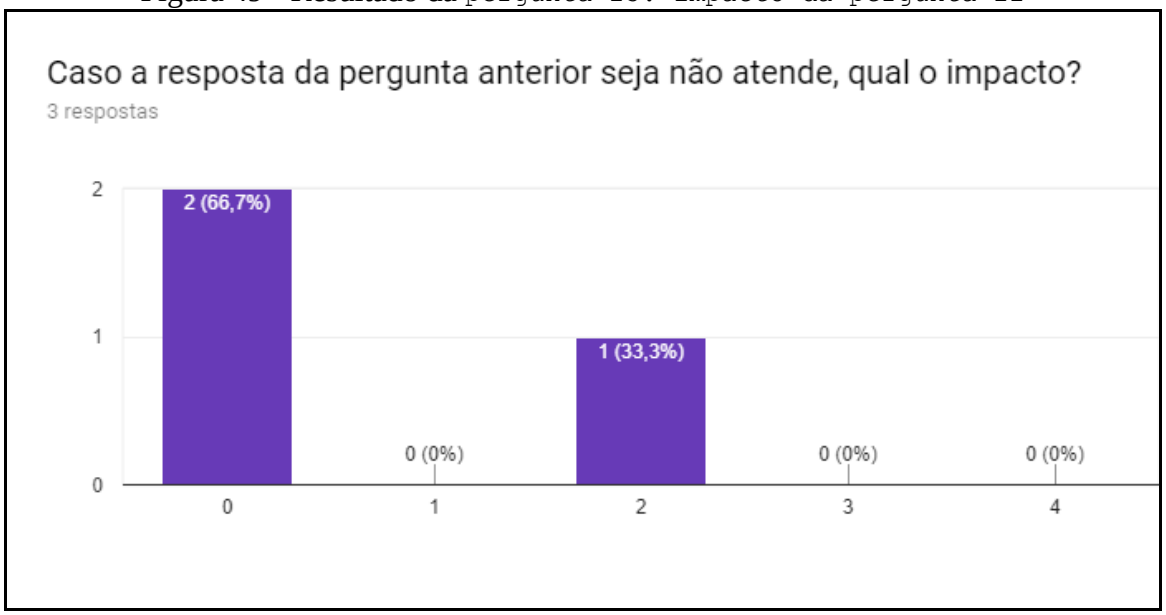
A Heurística 10 demonstrada na Figura 48, relacionada a documentação, foi a que mais apresentou problemas. Como é possível visualizar na Figura 49, dois dos usuários especialistas visualizaram problemas relacionados a essa Heurística. Este problema foi visualizado durante o desenvolvimento, desta forma, uma das extensões propostas para este trabalho é a resolução deste problema pela criação de um glossário dentro do aplicativo. Este glossário traria termos relevantes para entendimento da arquitetura, bem como documentação das informações demonstradas no aplicativo.

Figura 48 - Resultado da pergunta 22 relacionada a H10



Fonte: elaborado pelo autor.

Figura 49 - Resultado da `pergunta 23: impacto da pergunta 22`



Fonte: elaborado pelo autor.

Com base na análise dos resultados obtidos é possível verificar que a aplicação possui um alto grau de satisfação por todos os usuários especialistas na avaliação referente a usabilidade da arquitetura, todos os usuários visualizaram vantagens na utilização da mesma, bem como facilidade ao utilizar o aplicativo de visualização, trazendo facilidade e agilidade no controle e visualização da arquitetura. Outro ponto positivo foram as configurações centralizadas disponibilizadas pela arquitetura, esta destacada pelos usuários por facilitar o processo de manutenção das aplicações.

4 CONCLUSÕES

Neste trabalho é apresentado o MS-Trick, uma arquitetura de *microservices* aplicada no contexto da empresa Pública informática. Trata-se da refatoração de uma aplicação de maneira escalável e mais disponível, bem como aplicação de conceitos para facilitar a manutenção dessa arquitetura.

O objetivo geral descrito na seção 1.1 para este trabalho foi desenvolver uma arquitetura de *microservices*, sendo este cumprido conforme feedback dos usuários especialistas apresentados pelos resultados demonstrados na seção 3.4. Ademais, foram desenvolvidos todos os objetivos específicos descritos para este trabalho.

Referente ao objetivo específico de dividir módulos para criar os *microservices*, dois módulos foram separados do Monolito para criação de *microservices*, file-server e portal da transparência. O objetivo específico utilizar de integração contínua para facilitação na liberação de recursos, foi realizado com a implementação de um fluxo de liberação de aplicações na Ferramenta Gitlab, permitindo que as aplicações sejam compiladas e executadas apenas com a liberação do código fonte dos desenvolvedores. O objetivo específico implementação de aplicativo para manutenção e visualização dos *microservices* foi realizado pela implementação de um aplicativo móvel responsável por demonstrar a arquitetura como um todo.

A arquitetura desenvolvida atingiu as expectativas da empresa Pública Informática, por demonstrar e evidenciar pontos da arquitetura que podem vir a beneficiar a empresa. Os benefícios trazidos, bem como o aplicativo para controle se mostraram eficientes para as necessidades dos participantes da pesquisa.

A fundamentação teórica deste trabalho representa parte essencial do desenvolvimento deste trabalho, pois esta trouxe conhecimento, e com este foi possível materializar a arquitetura de *microservices* dentro de uma aplicação existente. A fundamentação teórica deste trabalho serve também como base para trabalhos futuros referentes a *microservices*. Ainda por se tratar de uma arquitetura, muito do que se imagina são apenas conceitos, porém estes puderam ser aplicados e demonstram vantagens na utilização.

A contribuição tecnológica deste trabalho é apresentar uma arquitetura muitas vezes pouco conhecida ou não cogitada, mesmo em aplicações que representam um caso de uso possível para utilização da mesma. A arquitetura além de trazer benefícios para o usuário final da aplicação, traz melhorias no processo de liberação das aplicações pelos desenvolvedores. A

contribuição acadêmica é demonstrar como a arquitetura pode ser aplicada numa aplicação existente, e ainda demonstrar os benefícios que esta pode trazer.

Este trabalho justifica-se pela necessidade da aplicação da arquitetura no contexto estudado. A necessidade da implementação da arquitetura se dá pelas empresas sempre buscarem a melhoria de seus sistemas, redução de custos e maior satisfação do usuário final. Desta forma é possível visualizar a arquitetura aplicada neste contexto trazendo essas vantagens aos usuários finais e o processo de desenvolvimento como um todo.

As dificuldades encontradas com o desenvolvimento deste trabalho foram a complexidade de entendimento de todas as tecnologias, e utilização delas de forma conjunta; a máquina que foi utilizada no desenvolvimento, por se tratarem de diversas partes sendo executadas ao mesmo tempo, foram enfrentadas dificuldades em relação aos recursos; e por último e considerada a mais importante, demonstrar a arquitetura executando como um todo de forma possível a demonstrar seus benefícios.

4.1 EXTENSÕES

Como extensões para este trabalho, sugere-se:

a) implementação de glossário de termos técnicos dentro do aplicativo;
b) utilização de Docker para implementação dos *microservices*, permitindo a execução ser ainda mais facilitada;
c) utilização de Kubernetes para administração dos *microservices*;
d) implementação de scripts do Gitlab para todas as aplicações desenvolvidas;
e) externalização dos demais módulos da aplicação.

REFERÊNCIAS

Amazon AWS. **O que é devops?**. Amazon AWS, 2017a. Disponível em: <https://aws.amazon.com/pt/devops/what-is-devops/>. Acesso em: 17 set. 2017.

_____. **O que significa integração contínua?**. Amazon AWS, 2017b. Disponível em: <https://aws.amazon.com/pt/devops/continuous-integration/>. Acesso em: 20 set. 2017.

COELHO, Otavio Pecego. **Arquitetura: Princípios para alcançar Desempenho e Escalabilidade em Aplicações**. [S.l.], 2004. Disponível em: <https://msdn.microsoft.com/pt-br/library/cc518051.aspx>. Acesso em: 19 jun. 2018.

COSTA, Simone Erbs da. **Avaliação de Usabilidade e Comunicabilidade com a Experiência do Usuário pelo Método M3C-URUCAg**. iLibras Collaborative, 2018a. Disponível em: <https://www.ilibrascollaborative.com/post-unico/2018/06/20/Avalia%C3%A7%C3%A3o-de-Usabilidade-e-Comunicabilidade-pelo-m%C3%A9todo-M3C-URUCAg>. Acesso em: 01 jul. 2018.

_____. **iLibras Collaborative**. iLibras Collaborative, 2018b. Disponível em: <https://www.ilibrascollaborative.com/noticias-e-recursos>. Acesso em: 19 jun. 2018.

COUTINHO, Mariana. **Saiba mais sobre streaming, a tecnologia que se popularizou na web 2.0.** [S.l.], 2013. Disponível Falta local em: <http://www.techtudo.com.br/artigos/noticia/2013/05/conheca-o-streaming-tecnologia-que-se-popularizou-na-web.html>. Acesso em: 10 maio 2018.

DEVMEDIA**. Simplificando a computação distribuída**. [S.l.], [2014?]. Disponível em: <http://www.devmedia.com.br/hazelcast-simplificando-a-computacao-distribuida/31628>. Acesso em: 17 de set. 2017.

DRAGONI, Nicola et al. ***Microservices*****:** yesterday, today, and tomorrow. Cornell University, 2016. Disponível em: <http://arxiv.org/abs/1606.04036>. Acesso em: 01 de maio. 2018.

FARINACCIO, Rafael. Falta local antes do ano. Tecmundo, [2017?]. Disponível em: <https://www.tecmundo.com.br/netflix/113583-netflix-chega-100-milhoes-usuarios-tem-crescimento-recorde.htm >. Acesso em: 21 out. 2017.

GAMMA, Erich et al. **Padrões de Projeto**: Soluções reutilizáveis de software orientado a objetos. ed. Porto Alegre: Bookman, 1995.

GITHUB. **Built for developers**. [S.l.], 2018. Disponível em: <https://github.com/>. Acesso em: 5. maio. 2018.

GRIFFITHS, Stephen. **Os Princípios de UX para Aplicativos em Dispositivos Móveis**. [S.l.], jun. 2015. Disponível em: <https://www.thinkwithgoogle.com/intl/pt-br/marketing-resources/ux-e-design/mobile-app-ux-principles/>. Acesso em: 17 set. 2017.

LEITE, Jair C. **Notas de aula de engenharia de software**. UFRN, 2000. Disponível em: <https://www.dimap.ufrn.br/~jair/ES/c7.html>. Acesso em: 17 set. 2017.

LIMA, Leonardo Ross Torquato**. Implementação de uma Arquitetura baseada em Micro serviços.** Marília, 2015. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília.

MANZANO, Débora. **Os Softwares precisam ter flexibilidade e escalabilidade**. [S.l.], 2016. Disponível em: <http://saudebusiness.com/noticias/os-softwares-precisam-ter-flexibilidade-e-escalabilidade-roberto-ferrarini/>. Acesso em: 30 out. 2017.

MOULIN, Robson. **O que é e como fazer uma Avaliação Heurística**. [S.l.], 2013. Disponível em: <http://www.designinterativo.etc.br/user-experience/o-que-e-e-como-fazer-uma-avaliacao-heuristica>. Acesso em: 01 jun. 2018.

NAMIOT, Dmitry; SNEPS-SNEPPE, Manfred. On Micro-services Architecture. **International Journal of Open Information Technologies**, v. 2, n. 9, 2014. Disponível em: <http://injoit.org/index.php/j1/article/view/139/104>. Acesso em: 03 de jun. 2018.

NETFLIX. **Como funciona a netflix?** [S.l.], [2017?]. Disponível em: <https://help.netflix.com/pt/node/412>. Acesso em: 17 set. 2017.

NETFLIX. **Netflix open source software center.** [S.l.], [2016?]. Disponível em: <https://netflix.github.io/>. Acesso em: 17 set. 2017.

NIELSEN, Jakob. Usability inspection methods. In: **Conference companion on Human factors in computing systems**, 414, 1994, Boston. **Proceedings.** Boston, 1994. p. 413-414.

FRALEIGH, Arnold. The Algerian of independence. In: ANNUAL MEETING OF THE AMERICAN SOCIETY OF INTERNATIONAL LAW, 61, 1967, Washington. **Proceeding**Washington: Society of International Law, 1967. p. 6-12.

NOURIE, Dana. **Java technologies for web applications**. [S.l.], 2006. Disponível em: <http://www.oracle.com/technetwork/articles/javase/webapps-1-138794.html#webapp>. Acesso em: 17 set. 2017.

PRATES, Raquel Oliveira; DINIZ, Simone; BARBOSA, Junqueira. **Capítulo 6 Avaliação de Interfaces de Usuário – Conceitos e Métodos**. [S.l.], 2003. Disponível em: <http://homepages.dcc.ufmg.br/~rprates/ge_vis/ >. Acesso em: 30 jun. 2018.

RICHARDSON, Chris. **Decomposing Applications for Deployability and Scalability.** [S.l.], 2014. Disponível em: <www.infoq.com/articles/*microservices*-intro>. Acesso em: 21 out. 2017.

SAMPAIO, Cleuton. **Curadoria de micro serviços**. [S.l.], 2015. Disponível em: <http://www.obomprogramador.com/2015/06/curadoria-de-micro-servicos.html>. Acesso em: 30 out. 2017.

SÃO MIGUEL, Camargo Erick; FARINA, Renata Mirella. **Integrando Micro serviços em uma Aplicação Web.** 2016.5 f. Projeto de pesquisa (Iniciação Científica) – CCA, Uniara.

SILVA, S. S. da. **Adoção da Arquitetura de Micro serviços em Aplicações Corporativas.** 2015.1-26 f.. Trabalho de conclusão de curso (Bacharelado de Sistemas de Informação) – Universidade Federal da Paraíba, Rio Tinto, Paraíba.

SOARES, Fabrízzio A M N**. Especialização em Desenvolvimento de Aplicações Web com Interfaces Rica.** UFG, 2012. Disponível em: <http://www.inf.ufg.br/~fabrizzio/publication/>. Acesso em: 17 set. 2017.

SOBRAL, João Bosco Mangueira. S**ervidores de Aplicações Web**. UFSC, 2002. Disponível em: <http://www.inf.ufsc.br/~bosco.sobral/old_page/downloads/>. Acesso em: 17 set. 2017.

SPRING**. Spring Boot**. [S.l.], 2017. Disponível em: <https://spring.io/projects/spring-boot>. Acesso em: 17 set. 2017.

TONSE, Sudhir. ***Microservices* at netflix - challenges of scale.** [S.l.], 2014. Disponível em: <https://pt.slideshare.net/stonse/*microservices*-at-netflix>. Acesso em: 17 set. 2017.

APÊNDICE A – TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO (TCLE)

O(a) senhor(a) está sendo convidado a participar de uma pesquisa de graduação, intitulada “MSTRICK - ARQUITETURA DE *MICROSERVICES* APLICADA” que fará uma breve avaliação com o objetivo geral da aplicação de uma arquitetura de *microservices* no caso de estudo da empresa Pública Tecnologia. Esta avaliação será realizada Pública Informática.

O pesquisador responsável fará a explicação da forma que será aplicado a avaliação. Como esta é uma participação voluntária, o(a) Senhor(a) e seu/sua acompanhante não terão despesas e nem serão remunerados pela participação na pesquisa. Todas as despesas decorrentes de sua participação serão ressarcidas. Em caso de danos, decorrentes da pesquisa será garantida a indenização. Os possíveis desconfortos e riscos decorrentes do estudo, levando-se em conta que é uma pesquisa, e os resultados positivos ou negativos somente serão obtidos após a sua realização. Assim, estou sujeito a realização de tarefas pré-definidas e especificadas no formulário de avaliação. Além disso, a minha avaliação poderá ou não ser considerada no resultado final do aplicativo, dependendo da forma que eu estarei respondendo minha avaliação.

Estou ciente que minha privacidade será respeitada, ou seja, meu nome ou qualquer outro dado ou elemento que possa, de qualquer forma, me identificar, será mantido em sigilo. Também fui informado que eu posso me recusar a participar do estudo, ou retirar meu consentimento a qualquer momento, sem precisar justificar, e que, por desejar sair da pesquisa, não sofrerei qualquer prejuízo.

Os pesquisadores envolvidos no estudo são, Ariel Rai Rodrigues, que posso entrar em contato pelo e-mail arielrairodrigues@gmail.com e da orientadora da pesquisa Simone Erbs da Costa, que posso entrar em contato pelo e-mail simoneerbsdacosta@gmail.com. Além disso, é assegurada toda assistência durante toda a pesquisa, bem como me é garantido o livre acesso a todas as informações e esclarecimentos adicionais sobre o estudo e suas consequências, ou seja, tudo que eu queria saber antes, durante e depois da minha participação.

Dessa forma, tendo sido orientado quanto ao teor de todo aqui mencionado e compreendido a natureza e o objetivo do referido estudo, manifesto meu livre consentimento em participar, estando totalmente ciente de que não existe nenhum valor econômico, a receber ou a pagar, por minha participação. Caso exista algum dano decorrente a minha participação no estudo, serei devidamente indenizado conforme determina a lei.

Em caso de reclamação ou qualquer outra denúncia sobre esse estudo devo entrar em contato com a orientadora da pesquisa Simone Erbs da Costa pelo e-mail simoneerbsdacosta@gmail.com. Os benefícios e vantagens em participar deste estudo estão relacionados a contribuir para o uso de novas metodologias em salas de aula, tornando o ambiente de ensino aprendizagem mais próximo da realidade que estou inserido. A pessoa que estará acompanhando os procedimentos será o professor regente da disciplina que o questionário está sendo aplicado. O(a) senhor(a) poderá se retirar do estudo a qualquer momento, sem qualquer tipo de constrangimento. Solicitamos a sua autorização para o uso de seus dados para a produção de artigos técnicos e científicos.

A sua privacidade será mantida por meio da não-identificação do seu nome. Caso esteja de acordo, com o termo basta seguir o roteiro da pesquisa abaixo e responder as perguntas do questionário de avaliação logo após o roteiro.

APÊNDICE B – ROTEIRO

Casos de testes

Após cada fluxo será demonstrada uma imagem com o resultado

Caso de teste integração contínua - backend do aplicativo

0 - Acessar Gitlab pelo link:
https://gitlab.com/arielrai/ms-trick-backend
1 - Acessar aba CI/CD
2 - Clicar botão Run Pipeline

Fluxo 1(Neste fluxo, será realizado a compilação, deploy):
2.1 - No combobox selecionar: ms-trick-development
2.2 - Clicar botão create pipeline (Aguardar finalização da pipeline)
- build: Compilação da aplicação
- deploy-staging: Deploy no ambiente de testes, pois a branch selecionada no passo 2.1 não é a branch de desenvolvimento.
- sonar: avaliação do código
2.3 - Acessar https://dashboard.heroku.com/apps/ms-trick-backend-staging
2.4 - Selecionar aba activity
É possível visualizar o último deploy nesta aba
resultado: https://imgur.com/a/RliZn2X

Fluxo 2(Neste fluxo, será realizado a compilação, deploy):
2.1 - No combobox selectionar:master
2.2 - Aguardar finalização da pipeline
2.3 - Acessar https://dashboard.heroku.com/apps/ms-trick-2
2.4 - Selecionar aba activity
É possível visualizar o último deploy nesta aba
resultado: https://imgur.com/a/RliZn2X

Também é possível visualizar a qualidade do código em:
https://sonarcloud.io/dashboard?id=com.publica%3A*microservices*-analyzer

---

Caso de testes aplicativo controle

(Configuração de usuário, senha e tempo de notificações para aplicativo de controle)

0 - Configurar usuário e senha em https://github.com/arielrai/tcc-*microservices*-properties/blob/master/ms-trick.properties

0.1 - Alterar variáveis do arquivo: app.user, app.pass; Com respectivos usuários e senha

0.2 - Alterar variaveis do arquivo: "scheduler.minutes" com o respectivo valor para aguardar por notificações(tempo em minutos)

0.2 - Executar caso de testes 2(Irá atualizar as propriedades)

(Visualizar métricas, requisições, variáveis de ambiente, enpoints)

1 - Acessar aplicativo como usuário atualizado

2 - Selecionar algum dos aplicativos e "visualizar"

2.1 - Selecionar aba visão geral

2.1.1 - é possível visualizar métricas do aplicativo, reiniciar e ou desligar a instância

2.2 - Selecionar aba requisições

2.2.1 - é possível visualizar todas as requisições do aplicativo

2.2.2 - utilizar a buscar

2.2.2.1 - escrever "/apps"

2.2.2.2 - é possível visualizar os resultados filtrados

2.3 - Selecionar aba ambiente

2.3.1 - é possível visualizar as variáveis de ambiente da instância

2.4 - Selecionar aba endpoints

2.4.1 - é possível visualizar todos os endpoints da aplicação

(Visualizar configurações do aplicativo e configurar notificações ativas e receber notificações)

3 - Acessar Menu Configuração

3.1 - Alterar switch Notificações Ativas para ativo

3.2 - Salvar configuração

3.3 - Alterar MemoryFreeNotifier switch para ativo

3.4 - SystemLoadAverageNotificer switch para ativo

3.5 - Salvar configuração de notificação

Aguarde o tempo informado na variável "scheduler.minutes" e será possível receber notificações.

(Visualizar notificações enviadas)

4 - Acessar Menu Notificações

Será possível visualizar todas as notificações já enviadas, os respectivos dispositivos e a data de envio;

4.1 - Realizar a busca com o texto "memória".

Será possível visualizar todas as notificações contendo o texto "memória"

Caso de teste microservice

(Visualizar os *microservices* respondendo requisições)

0 - Acessar o portal da transparência pelo link: http://52.14.48.79:9000/#/portal?alias=trunk_mafra

1 - Abrir console de desenvolvedor(Pressionar F12)

2 - Selecionar menu Despesa - Detalhada

3 - Pesquisar Por palavra-chave(qualquer filtro)

4 - Alterar console para aba network

4.1 - Selecionar alguma das requisições feitas

4.2 - Alterar para aba Headers

4.3 - Verificar Response Headers

Dentre os requests é possível verificar a Header Real_host

O valor dele representa o servidor real de resposta

Breve explicação sobre os *microservices* sendo executados:

https://www.dropbox.com/s/vedza3xuuamsauw/Breve%20explica%C3%A7%C3%A3o%20microservi%C3%A7os.pdf?dl=0