FACULDADE DE TECNOLOGIA DE SÃO PAULO

NAOMI CRISTINA TABATA

ESTUDO DE ARQUITETURA DE MICRO *FRONTENDS*

SÃO PAULO

2021

FACULDADE DE TECNOLOGIA DE SÃO PAULO

NAOMI CRISTINA TABATA

ESTUDO DE ARQUITETURA DE MICRO *FRONTENDS*

Trabalho de Conclusão de Curso (TCC) apresentado para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Faculdade de Tecnologia de São Paulo – FATEC-SP.

Orientadora: Prof.ª Ms. Vânia Franciscon Vieira

SÃO PAULO

2021

FACULDADE DE TECNOLOGIA DE SÃO PAULO

NAOMI CRISTINA TABATA

ESTUDO DE ARQUITETURA DE MICRO *FRONTENDS*

Trabalho de Conclusão de Curso (TCC) apresentado para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Faculdade de Tecnologia de São Paulo – FATEC-SP.

Parecer do Orientador

_____
_____
_____

Conceito/Nota Final: 9.0

Atesto o conteúdo contido na postagem do ambiente TEAMS pelo aluno e assinada por mim para avaliação do TCC.

Orientadora: Prof.ª Ms. Vânia Franciscon Vieira

SÃO PAULO, __ de _____ de 2021.

Assinatura do Orientador

Assinatura do aluno

RESUMO

Atualmente existe uma grande demanda de desenvolvimento de *software*. Para atender tal demanda, estão surgindo novos modelos de desenvolvimento, que visam entregar rapidamente um produto utilizável, mesmo que limitado, para atender às necessidades de clientes.

Este trabalho é uma pesquisa bibliográfica de arquitetura de micro *frontends*, que tem como objetivo é realizar e documentar um estudo acerca do tópico.

O objeto de estudo em questão se trata de um estilo de arquitetura cujos princípios derivam da arquitetura de microsserviços, onde o *backend* da aplicação é dividido em aplicações menores independentes entre si, porém neste caso é aplicado também ao *frontend*.

Palavras-chave: Arquitetura de *software*, micro *frontends*, desenvolvimento de *software*.

ABSTRACT

There is currently a great demand for software development. To meet this demand, new development models are emerging, which aim to quickly deliver a usable product, even if limited, to meet the needs of customers.

This paper is a bibliographical research on micro frontends architecture, which aims to carry out and document a study on the topic.

The object of the study in question is an architectural style, wich principles derive from the microservices architecture, where the application's backend is divided into smaller applications independent on each other, but in this case it is also applied to the frontend.

Keywords: Software architecture, micro frontends, software development.

LISTA DE FIGURAS

Figura 1 – Modelo de arquitetura monolítica 11
Figura 2 – Divisão de um aplicativo monolítico para microsserviços 13
Figura 3 – Monolitos e microsserviços 14
Figura 4 – Ciclo de *DevOps* 16
Figura 5 – Monolito, *front* & *back* e microsserviços 18
Figura 6 – *End-to-end* com micro *frontends* 19
Figura 7 – Implementação independente em micro *frontends* 21
Figura 8 – Abordagens de integração em micro *frontends* 23
Figura 9 – *Iframes* no aplicativo Spotify 24
Figura 10 – *Web components* 25
Figura 11 – Modelo de site de entrega de comida 26
Figura 12 – Separação dos elementos da página 27
Figura 13 – Página inicial do exemplo finalizado 28

LISTA DE SIGLAS

API – *Application Programming Interface*

DDD – *Domain-Driven Design*

TI – Tecnologia da Informação

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO 8
1.1 Contexto 8
1.2 Objetivo 9
1.3 Resultado esperado 9
1.4 Metodologia de trabalho 9
1.5 Organização do trabalho 10
CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA 11
2.1 Arquitetura monolítica 11
2.2 Arquitetura de microsserviços 13
2.3 DevOps 15
CAPÍTULO 3 – ESTUDO DE ARQUITETURA DE MICRO *FRONTENDS* 18
3.1 Introdução a micro *frontends* 18
3.2 Princípios de micro *frontends* 20
3.2.1 Modelagem em torno de domínio de negócio 20
3.2.2 Cultura da automação 20
3.2.3 Abstração de detalhes de implementação 20
3.2.4 Descentralização acima de centralização 21
3.2.5 Implementação independente 21
3.2.6 Falha isolada 21
3.2.7 Altamente observável 22
3.2.8 Estabelecer prefixos de equipe 22
3.2.9 Favorecer recursos nativos do navegador 22
3.3 Abordagens utilizadas em micro *frontends* 22
3.3.1 Roteamento e transição de página 23
3.3.2 Composição 24
3.4 Exemplo de uso de micro *frontends* 26

CAPÍTULO 4 – VANTAGENS E DESVANTAGENS DA UTILIZAÇÃO DE MICRO *FRONTENDS*.....29
4.1 Estudo de vantagens.....29
4.1.1 Atualizações incrementais.....29
4.1.2 Times autônomos.....29
4.2.3 Códigos menores e desacoplados.....29
4.2 Estudo de desvantagens.....30
4.2.1 Redundância.....30
4.2.2 Complexidade operacional e de governança.....30
CAPÍTULO 5 – CONSIDERAÇÕES FINAIS.....31
REFERÊNCIAS.....32
GLOSSÁRIO.....35

CAPÍTULO 1 – INTRODUÇÃO

1.1 Contexto

Desde a criação do computador e internet, grandes avanços tecnológicos ocorreram nas últimas décadas. Hoje em dia, boa parte da população brasileira possui acesso a pelo menos um computador ou *smartphone.* Segundo o Centro Regional de Estudos para o Desenvolvimento da Sociedade e Informação (Cetic.br), cerca de 74% dos brasileiros com mais de 10 anos possuíam acesso à rede em 2019.

Além da necessidade de as empresas possuírem suas ferramentas de *software* para uso próprio, ficou evidente a importância de uma presença digital para atingir potenciais consumidores, principalmente para produtos e serviços (FREITAS, 2008; HAUSCHILD, 2017).

Com o isolamento social decorrente da pandemia de Covid-19, que se agravou em 2020 e perdura pelo ano de 2021, tornou-se praticamente indispensável que as empresas possuam pelo menos uma página na rede para alcançar seu público-alvo: segundo uma pesquisa realizada pela Mastercard e *Americas Market Intelligence* (AMI), 46% dos brasileiros realizaram mais compras através do *e-commerce*, sendo que 7% compraram pela primeira vez (E-COMMERCE BRASIL, 2020).

Tal demanda pode também ser notada no mercado de trabalho: mesmo com crises sanitária e econômica, a área de Tecnologia da Informação (TI) é uma das mais requisitada atualmente. No entanto, é uma carreira que ainda possui poucos profissionais especialistas no Brasil, e a tendência é que esse cenário se prorrogue (GRANATO, 2020; EXAME, 2020). Por isso, é imprescindível o estudo e registro de novas tecnologias.

No desenvolvimento de *software* e *web*, podem ser utilizadas diversas abordagens, metodologias e tecnologias. Além disso, novas tecnologias surgem todos os dias, e a melhor utilização de cada um desses elementos devem ser avaliados pelas equipes responsáveis.

Atualmente, devido à grande demanda e à necessidade de constantes alterações para melhor atender e adaptar-se aos clientes, popularizou-se a cultura *DevOps* e a utilização de metodologias ágeis como *Scrum* e *Kanban* no desenvolvimento de *software*.

O objeto de estudo deste trabalho propõe uma metodologia para desenvolver o *frontend* de um projeto de forma mais rápida e adaptável a possíveis alterações, sendo uma opção interessante para as práticas atuais.

1.2 Objetivo

Este trabalho tem como objetivo realizar um estudo teórico da arquitetura de micro *frontends*, bem como um breve estudo sobre arquitetura monolítica e arquitetura de micro serviços para auxiliar em uma análise de vantagens e desvantagens sobre o tema proposto.

1.3 Resultado esperado

Espera-se como resultado um estudo sobre arquitetura de micro *frontends* e realizar uma análise de suas vantagens e desvantagens em comparação às arquiteturas monolítica e de microsserviços. Ressalta-se que este trabalho não possui intento de realizar uma análise qualitativa acerca das metodologias estudadas, visto que esta avaliação dependeria da execução do projeto e não necessariamente da arquitetura utilizada.

Como isso, este trabalho visa auxiliar futuros projetos, trazendo um registro acadêmico acerca do objeto de estudo.

1.4 Metodologia de trabalho

Para desenvolver este trabalho serão realizadas as seguintes atividades:

a) Fundamentação Teórica: Nesta atividade serão realizados o levantamento e o estudo dos fundamentos teóricos necessários para o desenvolvimento do trabalho.
b) Estudo de Arquitetura de Micro *Frontends*: Esta atividade é dedicada ao estudo da arquitetura de micro *frontends*.

c) Estudo de Vantagens e Desvantagens da Utilização de Micro *Frontends*: São analisadas as vantagens e desvantagens da utilização de micro *frontends* com base nos estudos realizados.
d) Análise de Resultados: Para esta atividade são realizadas as considerações finais do trabalho.

1.5 Organização do trabalho

Este trabalho é constituído pela Introdução e os seguintes capítulos:

a) Capítulo 2 – Fundamentação teórica: Capítulo dedicado à documentação do estudo teórico de arquitetura monolítica, arquitetura de microsserviços e *DevOps* realizado para este trabalho;
b) Capítulo 3 – Estudo de Arquitetura de Micro *Frontends*: Trata do registro do estudo teórico realizado acerca da arquitetura de micro *frontends*, contemplando uma introdução à arquitetura, seus princípios, abordagens utilizadas e um exemplo de uso;
c) Capítulo 4 – Análise de Vantagens e Desvantagens da Utilização de Micro *Frontends*: Destinado à análise das vantagens e desvantagens da utilização de micro *frontends*, com base nos estudos realizados;
d) Capítulo 5 – Considerações Finais: Considerações sobre o objeto de estudo, o atendimento do objetivo proposto e dificuldades encontradas durante o processo do trabalho.

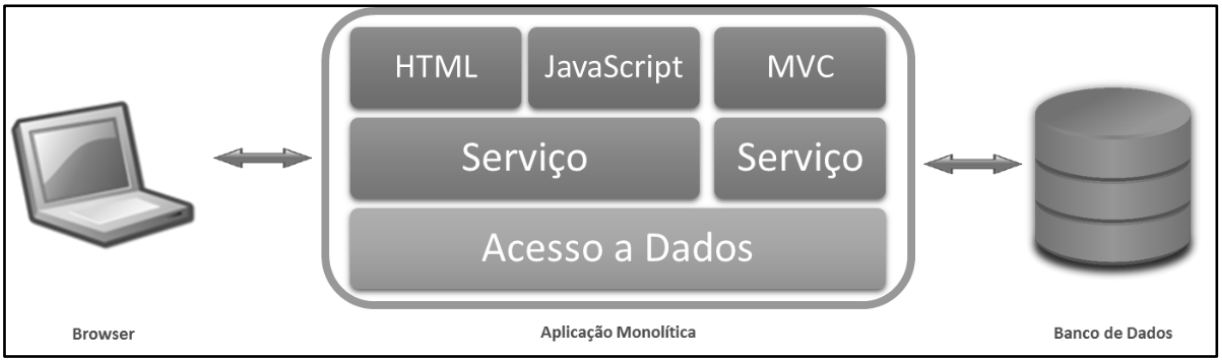
CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA

2.1 Arquitetura monolítica

Segundo Garlan e Shaw (1994), desde o fim da década de 1960 são estudados padrões para promover uma boa arquitetura de *software*. Com o passar dos anos, foram criados modelos, estilos e normas que permitiram um nível maior de organização para a produção de sistemas.

A arquitetura monolítica é um modelo onde todos os processos de um programa são alocados, compilados e executados em um único serviço (AMAZON WEB SERVICE, c2020; ANNET, 2014). Ou seja, em uma aplicação monolítica, as camadas de interface de usuário (*frontend*), processos do sistema (*backend*) e controladores de acesso aos dados ficam em um único código. Um exemplo dessa estrutura pode ser observado na Figura 1 abaixo.

Figura 1 – Modelo de arquitetura monolítica



Fonte: Annet (2014)

Esta abordagem é conhecida e utilizada por ser simplificada, robusta e fácil de desenvolver e implementar, de forma que, mesmo sendo considerada ultrapassada por alguns, poderia boa opção para iniciar um projeto. Ela também tem capacidade de escalar para grandes proporções: a rede social Facebook e a plataforma de comércio eletrônico Etsy, por exemplo, foram criados a fazendo uso de arquitetura monolítica (FOWLER, 2014).

Segundo Souza e Pelissari (2017), a construção de uma aplicação a partir do modelo monolítico segue o “modelo natural de construção de um sistema”. O desenvolvimento segue um processo singular, no qual é possível modular as

funcionalidades, caso a linguagem utilizada disponibilize ferramentas como classes, funções e *namespaces*, a fim de facilitar a visualização dos diferentes componentes do sistema.

A congregação de toda a aplicação em um único código, enquanto pequeno, facilita a visualização por toda equipe, já que não é necessária a navegação por entre diferentes projetos. Também simplifica os processos de compilação e implementação, uma vez que é gerado um único executável.

Caso haja a ampliação da aplicação e, eventualmente, da equipe, alguns pontos negativos da arquitetura vão se tornando evidentes. Se o código se tornar muito extenso, realizar a manutenção e criação de novas funcionalidades no sistema pode acabar se transformando em uma tarefa exaustiva e frustrante, principalmente na ausência do uso de boas práticas e de uma documentação centralizada, atualizada e consistente, já que é necessário analisar o código por completo para compreendê-lo e realizar testes de regressão para garantir que todos os módulos continuem funcionando corretamente após as alterações.

Outro problema que pode ocorrer, caso a equipe esteja desenvolvendo fazendo uso de uma cultura de *DevOps* (assunto do tópico 2.3), é a perda de alterações se mais de uma pessoa estiver atuando no código. Além disso, um *software* muito grande pode acabar se tornando pesado, acarretando a necessidade de um *hardware* mais robusto tanto para a equipe de desenvolvimento quanto para o usuário final.

Segundo Lewis e Fowler (2014), é possível escalar uma aplicação monolítica verticalmente, executando várias instâncias e fazendo uso de um balanceador de carga para gerenciá-las. No entanto, como não é possível executar apenas o módulo desejado, a utilização desta técnica pode não gerar um bom custo-benefício de performance.

Outras desvantagens de uma aplicação monolítica estão relacionadas à implementação: caso algum dos componentes utilizados pelo sistema não funcione, existe a grande possibilidade de que toda a aplicação deixe de funcionar, devido à dependência existente entre elas. Esta característica se torna um problema ainda maior se a aplicação fizer uso de serviços de terceiros, já que a equipe não possuirá controle sobre a disponibilidade e versionamento destes artefatos.

Por fim, por se tratar de uma única unidade de *software*, para efetivar alterações em uma aplicação monolítica, mesmo que em um único componente, é necessário compilar e reimplementar a aplicação inteira (BASTOS, 2020).

As desvantagens da arquitetura monolítica levaram ao desenvolvimento da arquitetura de microsserviços.

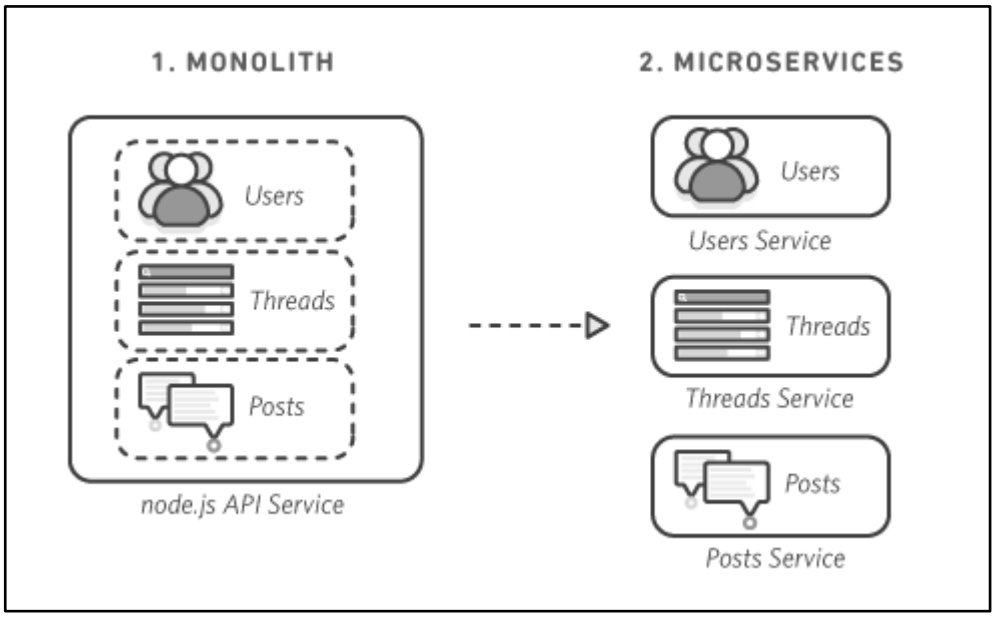
2.2 Arquitetura de microsserviços

A arquitetura de microsserviços tem ganhado popularidade nos últimos anos, já que ela se adequa bem à grande demanda e constante necessidade de atualização e manutenção dos projetos de TI nos tempos atuais, tanto para *softwares* como para sites.

Ao utilizar uma abordagem de microsserviços, normalmente divide-se o *backend* da aplicação em serviços menores independentes que, geralmente, se comunicam através de *Application Programming Interfaces* (APIs). Esses serviços possuem autonomia, tendo pouca ou nenhuma dependência entre os outros do mesmo sistema (AMAZON WEB SERVICE, c2020; FOWLER; LEWIS, 2014).

Uma aplicação monolítica já existente pode ser adaptada à uma estrutura de microsserviços, como representado na Figura 2 abaixo:

Figura 2 – Divisão de um aplicativo monolítico para microsserviços



Fonte: Amazon Web Service (c2020)

No entanto, caso exista a pretensão de utilizar a arquitetura de microsserviços, é recomendado que ela seja utilizada desde o início do projeto, ao invés de começar

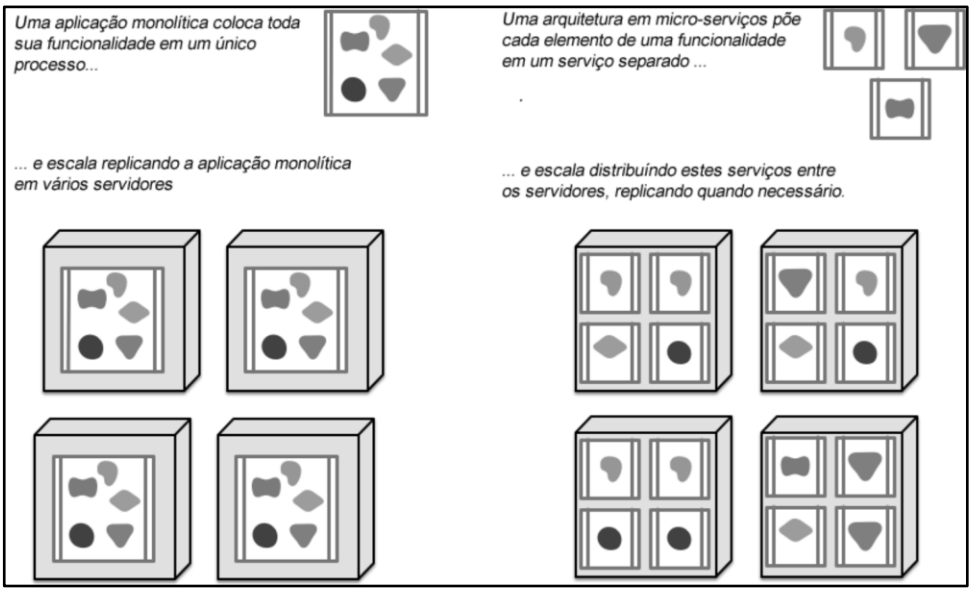
a partir de uma aplicação monolítica para ser refatorada no futuro, evitando assim desperdício de recursos.

Em uma aplicação orientada a microsserviços, cada serviço é especializado para executar uma determinada função no sistema, desta forma, a estrutura organizacional do aplicativo mantém-se inteligível. Caso sejam realizadas muitas alterações em um serviço que acarretem o aumento de sua complexidade, existe a possibilidade de dividi-lo em serviços menores (AMAZON WEB SERVICE, c2020).

Cada serviço pode ser testado, compilado e implementado de forma individual e contínua, conforme a necessidade do negócio, sendo possível ainda designar diferentes equipes para cada uma dessas atividades. Cada equipe é capaz de utilizar ferramentas e linguagens diferentes para desenvolver seu serviço. Devido à independência entre os serviços, a falha de um deles não acarreta a falha de toda a aplicação.

Quanto à escalabilidade, enquanto uma aplicação monolítica exige a replicação de uma instância inteira, a arquitetura de microsserviços permite replicar e distribuir apenas os serviços necessários para cada componente, conforme a Figura 3 abaixo:

Figura 3 – Monolitos e microsserviços



Fonte: Fowler, Lewis (2014). Tradução de Souza, Pelissari (2017).

Desta forma, é possível avaliar com precisão as necessidades de infraestrutura para manter cada serviço funcionando mesmo que haja um pico de demanda (AMAZON WEB SERVICE, c2020).

A independência entre os serviços também favorece a correção de implementações incorretas, pois o serviço apresentando comportamento errôneo é facilmente identificado, podendo ser tratado de maneira rápida, seja voltando para uma versão que não apresentava problemas ou corrigindo e implementando uma correção de forma emergencial.

Apesar da agilidade promovida pela arquitetura de microsserviços, sua implementação pode se tornar complicada, se, por exemplo, as equipes optarem em utilizar diferentes ferramentas e linguagens, a integração entre os serviços torna-se complexa. Além disso, é possível ocorrer a incompatibilidade entre bibliotecas e *frameworks* utilizados por cada componente. A governança das equipes pode também se tornar mais complexa como consequência da individualidade entre elas, por isso, é necessário planejamento para a aplicação dessa arquitetura.

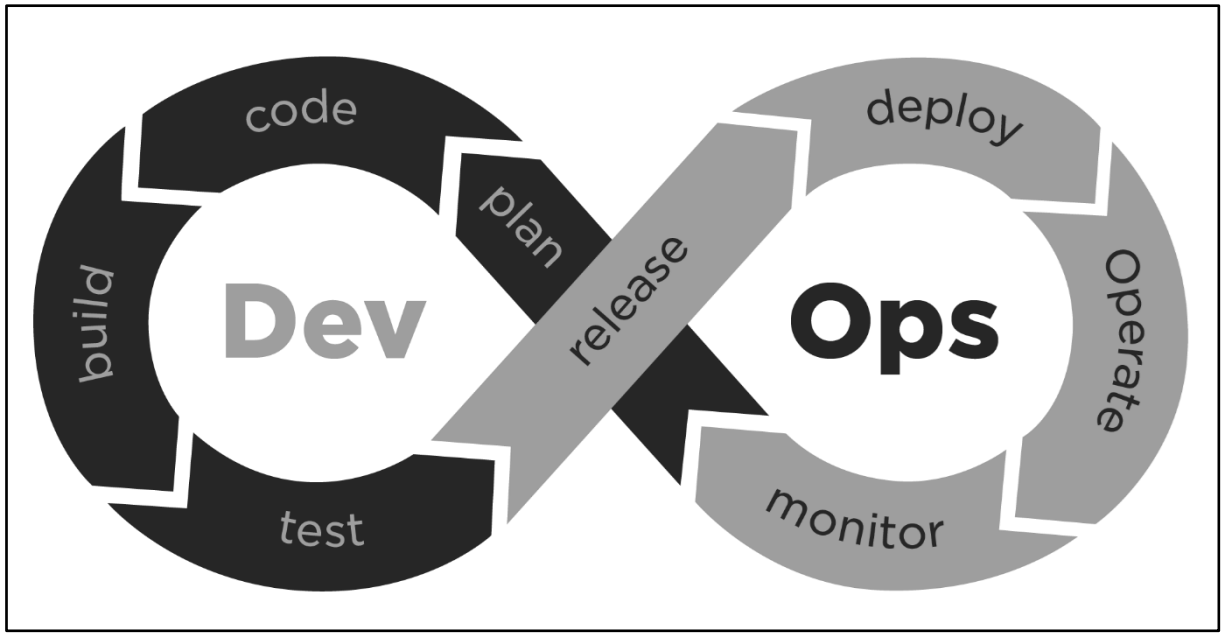
2.3 DevOps

Segundo Gene et al. (2018), o termo *DevOps* deriva das palavras *Development* (desenvolvimento) e *Operations* (operações) e foi citado pela primeira vez em 2009 por Patrick Debois após assistir à palestra “10+ *Deploys per Day: Dev and Ops Cooperation at Flick*”, de John Allspaw e Paul Hammond (apud BRANCO; BARROS, 2020, p. 1). De acordo com Sousa (2019), mesmo tendo ganhado popularidade nos últimos anos, o termo ainda não está bem definido, pois autores utilizam palavras diferentes para descrevê-lo, como abordagem e método de desenvolvimento de *software*, ou cultura organizacional por exemplo.

A AWS (c2021) define *DevOps* como uma “combinação de filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma empresa de distribuir aplicativos e serviços em alta velocidade [...]”.

Ao adotar um modelo de *DevOps*, as equipes de desenvolvimento e operações trabalham em conjunto para realizar entregas menores e rápidas seguindo um ciclo (Figura 4). Desta forma, é possível desenvolver um produto funcional, podendo ser parcial ou completo, de forma mais rápida e facilitar a adaptação a eventuais novas necessidades. É comum a utilização de microsserviços na cultura de *DevOps*, uma vez que suas naturezas possuem boa compatibilidade. Neste caso, cada serviço poderia seguir um ciclo.

Figura 4 – Ciclo de *DevOps*



Fonte: Kornilova (2017) apud Sousa (2019)

O ciclo de *Devops* segue as seguintes fases:

1. Planejamento: onde são definidos os requisitos e as ferramentas necessários para a entrega;
2. Codificação: é realizado o desenvolvimento do que foi definido na fase anterior;
3. Compilação: o código é compilado em um artefato, podendo ser um executável ou algum componente necessário para o *software* final;
4. Testes: são realizados testes para verificar a integridade do artefato que será entregue;
5. Disponibilização: o artefato é disponibilizado;
6. Implementação: o artefato é implementado no ambiente definido;
7. Operação: o artefato é utilizado pela equipe usuária;
8. Monitoração: é verificado se o que foi entregue está funcionando corretamente e de acordo com as necessidades.

Caso haja a necessidade de realizar algum ajuste ou melhoria para atender melhor o cliente, pode-se o ciclo iniciar novamente. Não é obrigatório seguir estes processos de forma linear, de modo que, caso seja detectada alguma inconsistência ou possibilidade de melhoria em uma fase, pode-se retornar a alguma fase anterior,

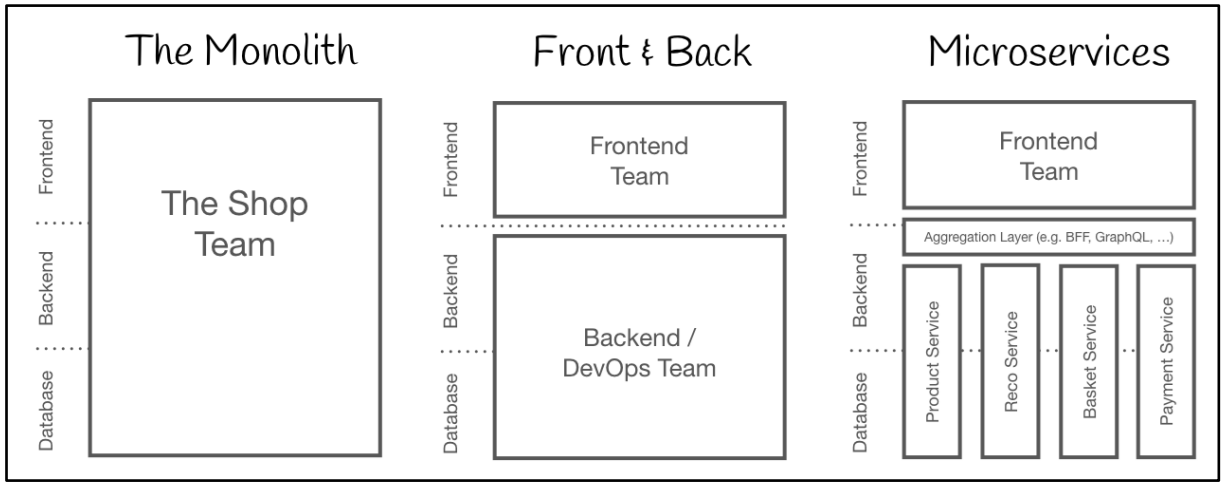
garantindo assim uma melhoria contínua e um produto que atenda melhor às necessidades do cliente.

CAPÍTULO 3 – ESTUDO DE ARQUITETURA DE MICRO *FRONTENDS*

3.1 Introdução a micro *frontends*

Atualmente é comum a utilização de arquitetura em camadas em projetos de TI na qual, geralmente, a aplicação é dividida em camadas de *backend* e *frontend,* podendo incluir outras como camada de controle e de negócios caso seja necessário, as quais podem ser desenvolvidas por uma ou várias equipes distintas e independentes entre si. Nestes casos, as abordagens mais comuns são: as aplicações *backend* e *frontends* serem monolíticas, como o exemplo "*Front* & *Back"* na Figura 5 abaixo, ou a aplicação *backend* utilizar da arquitetura de microsserviços enquanto o *frontend* mantem-se monolítico, conforme o exemplo "Microsserviços" da Figura 5 (BASTOS, 2020).

Figura 5 – Monolito, *front* & *back* e microsserviços



Fonte: Geers (c2021)

A utilização da arquitetura microsserviços popularizou-se nos últimos anos, sendo uma forma de contornar as inconveniências de monolitos *backend* muito extensos (JACKSON, 2019; THOUGHWORKS, 2016). Com isso, naturalmente surgiu a ideia de utilizar os princípios e técnicas de microsserviços também no desenvolvimento *frontend*.

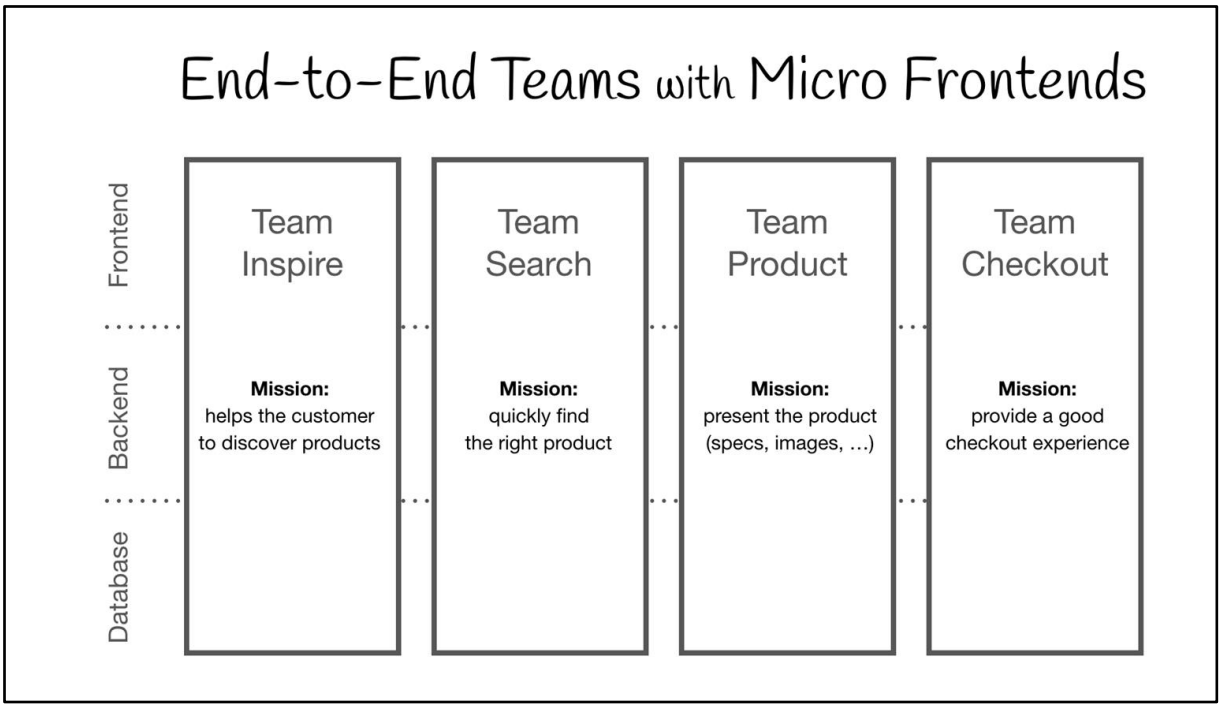
Micro *frontends* é um termo relativamente recente, cujo primeiro registro é datado de novembro de 2016, publicado no *Technology Radar* da *ThoughtWorks* (GEERS, c2021). A publicação introduz o termo micro *frontends*, como uma

abordagem onde uma aplicação *web* é dividida em páginas e funcionalidades de negócio e cada funcionalidade é de responsabilidade de uma equipe isolada (THOUGHTWORKS, 2016).

Jackson (2019), por sua vez, define micro *frontends* como “um estilo de arquitetura onde aplicações *frontend* que podem ser entregues independentemente formam um conjunto maior”.

A construção de uma aplicação com utilizando esta arquitetura um modelo vertical, como na Figura 6, onde cada equipe fica responsável por um componente desde o *frontend* até o banco de dados.

Figura 6 – *End-to-end* com micro *frontends*



Fonte: Geers (c2021)

É comum a utilização de micro *frontends* no *e-commerce*: empresas como Amazon e IKEA podem ser destacadas nesta categoria (GEERS, 2020). Contudo, a arquitetura não precisa se restringir apenas a esta área. A corretora Easynvest e o serviço de *streaming* de música Spotify, por exemplo, também utilizam da abordagem (SILVA, 2019).

3.2 Princípios de micro *frontends*

Os princípios de micro *frontends* são basicamente uma adaptação daqueles utilizados na arquitetura de microsserviços, porém aplicados no desenvolvimento *frontend*. Nesta seção, serão utilizados como base os princípios descritos por Mezzalira (c2021), complementando com os sugeridos por Geers (c2021), sendo eles: modelagem em torno de domínio de negócio; cultura de automação; ocultar detalhes de implementação; descentralização acima de centralização; implementação individual; falha isolada; altamente observável; estabelecer prefixos de equipes; favorecer recursos nativos do navegador.

3.2.1 Modelagem em torno de domínio de negócio

Primeiramente, Mezzalira (c2021) traz este conceito derivado do *Domain-Driven Design* (DDD), o qual, aplicado a micro *frontends*, sugere que as aplicações do sistema devem ser baseadas em domínio ou subdomínio de negócio. Neste contexto, domínio refere-se a um conjunto de processos e regras de negócio. Desta forma, a divisão entre as diferentes aplicações fica clara, facilitando a administração do projeto e a inclusão de futuras funcionalidades.

3.2.2 Cultura da automação

Este princípio discorre sobre a necessidade de uma automatizar a implementação. Pelo fato de cada projeto de micro *frontends* conter vários elementos, deve-se assegurar uma automação sólida para realizar implementações, garantindo que todos os elementos estejam corretamente integrados e agilizando os processos de *DevOps*.

3.2.3 Abstração de detalhes de implementação

Está relacionado com o princípio anterior. Ao realizar implementações de maneira automática, deve-se garantir que as equipes possam realizar o desenvolvimento de sua respectiva aplicação sem a necessidade de saber o estado das demais, principalmente se alguma parte da aplicação precisar se comunicar com

outra, com o intuito de diminuir o tempo de espera entre as equipes. Para isso, é preciso que as equipes estabeleçam e sigam um acordo com relação ao desenvolvimento, desta forma, não ficam dependentes umas das outras.

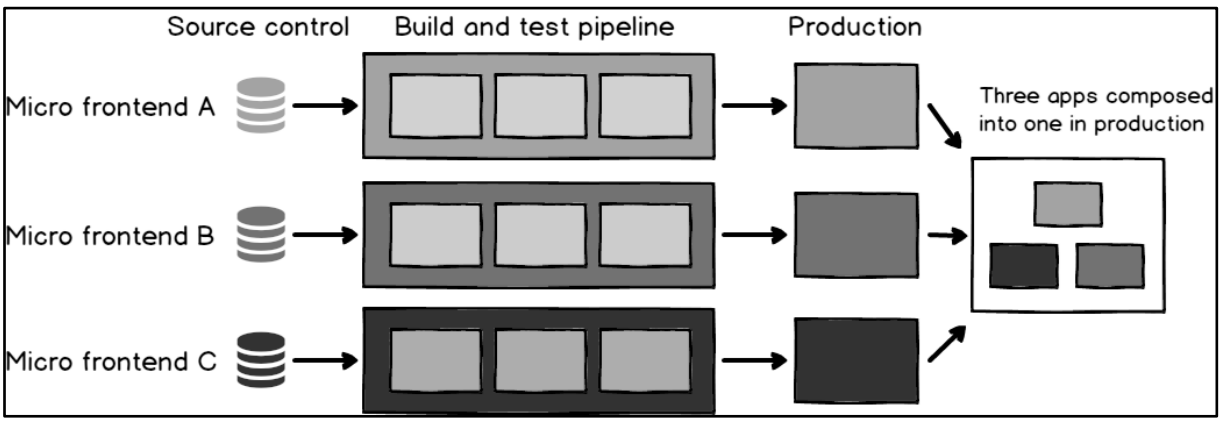
3.2.4 Descentralização acima de centralização

Refere-se a garantir a autonomia de decisão dos times. Assim, cada time pode decidir a abordagem que utilizará para realizar sua aplicação. No entanto, para que este princípio ocorra é necessário que existam diretrizes, minimizando a necessidade de autorizações de outras partes da empresa.

3.2.5 Implementação independente

Complementando a cultura da automação, a implementação independente garante que um artefato possa ser implementado sem precisar aguardar a resolução de dependências externa. A Figura 7 abaixo ilustra o caminho de cada micro *frontend* até a implantação em produção, onde as aplicações se juntam em um único aplicativo.

Figura 7 – Implementação independente em micro *frontends*



Fonte: Jackson (2019)

3.2.6 Falha isolada

Este princípio é contemplado também por Geers (c2021). Deve-se garantir que, mesmo que haja falha em algum componente, o resto da aplicação deve permanecer acessível e funcional para o usuário. No caso dos micro *frontends*, é preciso também

ocultar os componentes defeituosos para os usuários, garantindo uma melhor experiência.

3.2.7 Altamente observável

Manter uma aplicação orientada a microsserviços ou micro *frontends* altamente observável é uma tarefa difícil de se garantir ao comparar com um monolito. A existência de dezenas ou até centenas de objetos torna imprescindível a capacidade de visualizá-los como um todo para, por exemplo, localizar uma aplicação com falha de forma precisa e rápida.

3.2.8 Estabelecer prefixos de equipe

Geers (2021) traz mais princípios interessantes. Um deles é estabelecer prefixos de equipe. Refere-se a nomear arquivos e serviços utilizando um prefixo único, associado a uma equipe. Desta forma, mesmo se não for possível isolar algum elemento da aplicação, é possível visualizar qual equipe é responsável por um determinado componente.

3.2.9 Favorecer recursos nativos do navegador

Ao invés utilizar APIs personalizadas, a fim de facilitar a consistência da comunicação entre os componentes da aplicação. Caso seja necessária a construção de uma API, recomenda-se mantê-la o mais simples possível.

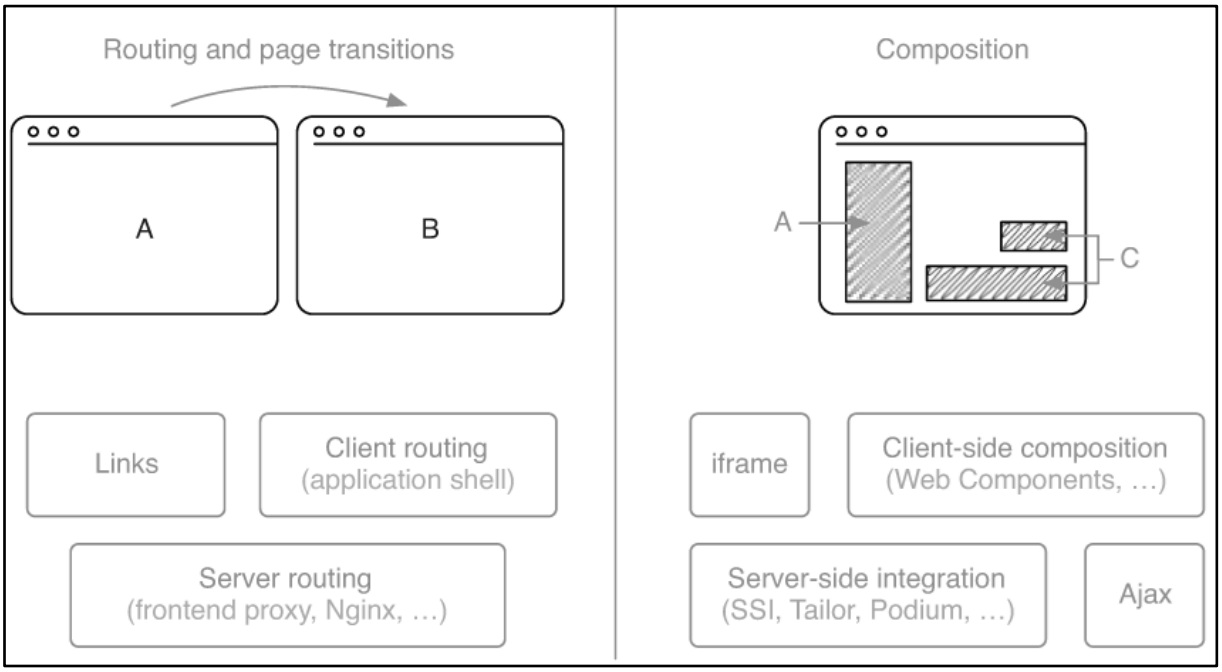
3.3 Abordagens utilizadas em micro *frontends*

Pelo fato de micro *frontends* se tratar de um conjunto de diretrizes para desenvolver uma aplicação, diversas abordagens podem ser utilizadas para integrar os diferentes objetos desenvolvidos, seguindo as necessidades do projeto.

Segundo Geers (2020), pode-se dividir as abordagens utilizadas nesta arquitetura em duas categorias: *roteamento e transição de página* e *composição*. A

Figura 8 apresenta técnicas que podem ser utilizadas, dentro nas categorias mencionadas.

Figura 8 – Abordagens de integração em micro *frontends*



Fonte: Geers (2020)

3.3.1 Roteamento e transição de página

Refere-se às técnicas que controlam internamente as transições entre as diferentes páginas/aplicações, ou seja, a forma que o usuário será direcionado de uma página para outra.

3.3.1.1 *Links*

Faz-se o uso de *hyperlinks* para o redirecionamento da página. É uma técnica simples, efetiva e de fácil execução, onde a aplicação simplesmente utiliza o endereço das outras para direcionar o consumidor (GEERS, 2020).

3.3.1.2 *Shell*

Utiliza-se uma aplicação *shell* compartilhada entre os micro frontends, que

observa o endereço da aplicação e redireciona o usuário para a aplicação correta quando o endereço é alterado (GEERS, 2020).

3.3.2 Composição

Englobam o encapsulamento das aplicações, integrando-as de forma que aparentem ser uma única aplicação. Podem ser divididas em *server-side integration*, onde os artefatos são integrados antes de chegar ao navegador do consumidor, e *client-side integration*, que permite que haja a integração diretamente no navegador.

3.3.2.1 IFrame

Promove um forte encapsulamento e controle das aplicações. A Figura 9 abaixo ilustra a separação das diferentes aplicações apresentadas no aplicativo do Spotify, realizadas a partir da utilização de *Iframes* (SILVA, 2019).

Figura 9 – *Iframes* no aplicativo Spotify

Fonte: Silva (2019)

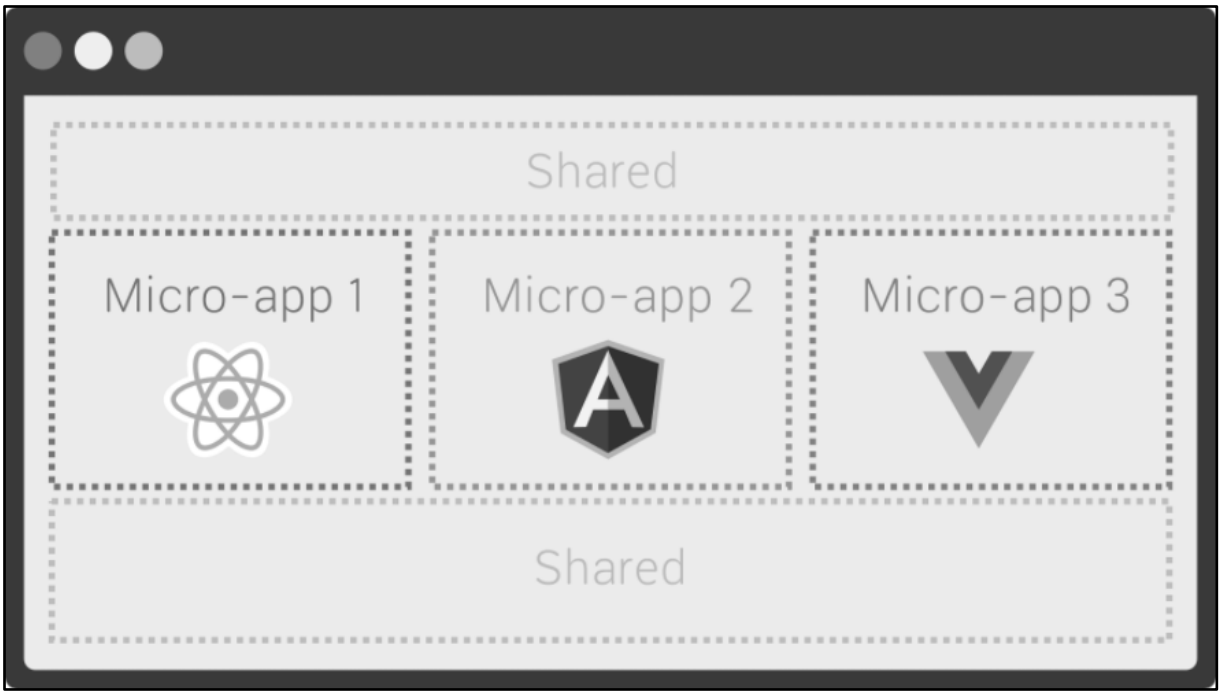
3.3.2.2 Micro aplicações

São criadas diferentes micro aplicações independentes, que são executadas em contextos distintos isolados por rotas, onde a comunicação pode ser realizada através de um *proxy* reverso (SILVA, 2020).

3.3.2.3 *Web components*

Promove maior individualidade entre as equipes, com um forte encapsulamento e permitindo o desenvolvimento de componentes personalizados, onde uma aplicação não influenciará em outra. É uma estratégia bastante utilizada, porém requer atenção, pois alguns navegadores podem não suportar alguns padrões (BASTOS, 2020). A Figura 10 representa uma aplicação onde existem três micro aplicações desenvolvidas em *frameworks* diferentes, com elementos compartilhados.

Figura 10 – *Web components*

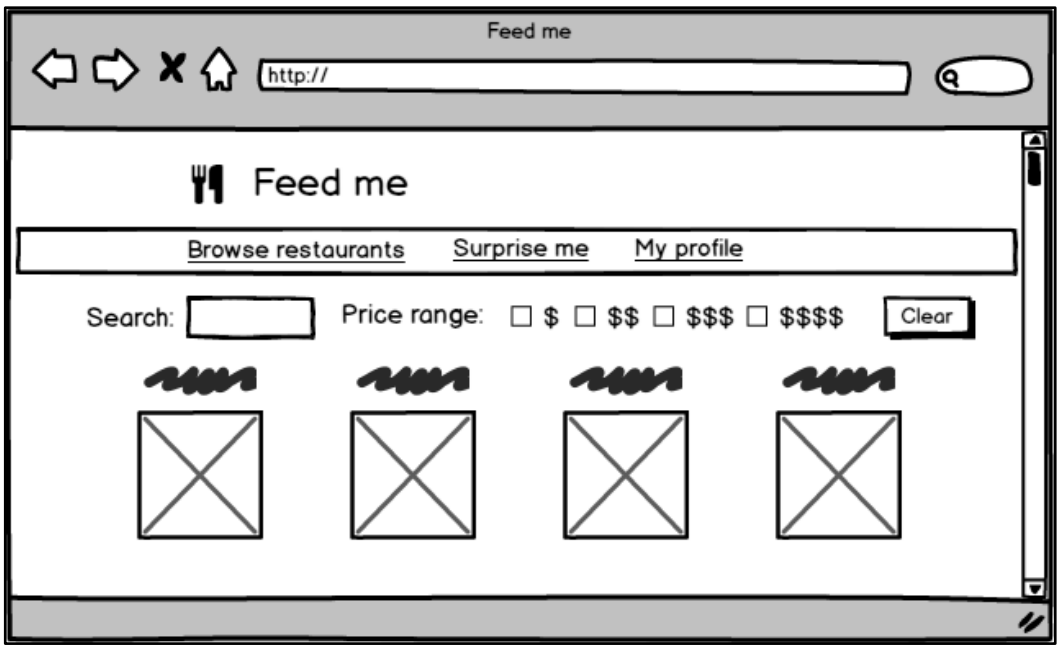


Fonte: Silva (2020)

3.4 Exemplo de uso de micro *frontends*

Jackson (2019) traz em seu artigo um exemplo de demonstração da arquitetura de micro *frontends*. Trata-se de um site de entrega de comida que possui as seguintes especificações: apresentar uma página principal com filtros de pesquisa para os consumidores procurarem restaurantes; cada restaurante precisa ter sua própria página com o cardápio, onde o consumidor pode escolher seu pedido; possuir uma página com o perfil do consumidor, na qual pode-se consultar histórico de pedidos, acompanhar a entrega e realizar manutenção das formas de pagamento. A Figura 11 abaixo apresenta um modelo do site.

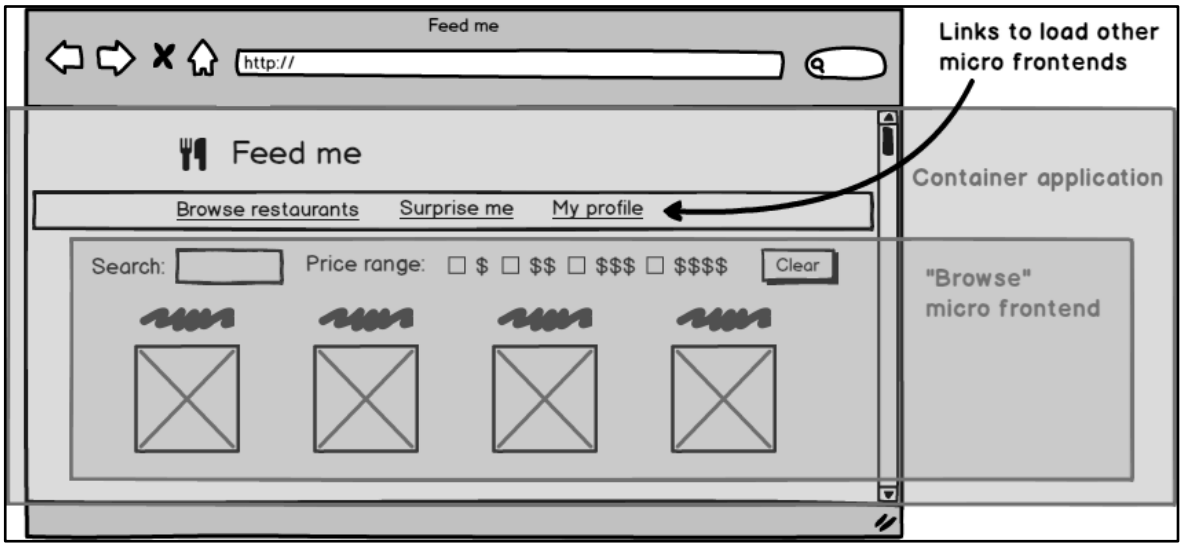
Figura 11 – Modelo de site de entrega de comida



Fonte: Jackson (2019)

Neste exemplo, Jackson (2019) cria um micro *frontend* para cada página e os introduz em uma única aplicação contêiner, que contém os elementos comuns entre as páginas (cabeçalho e menu) e realiza o gerenciamento dos micro *frontends*, controlando elementos de autenticação e navegação entre os diferentes componentes. A Figura 12 a seguir ilustra a separação entre a aplicação contêiner e a área onde o micro *frontend* será renderizado.

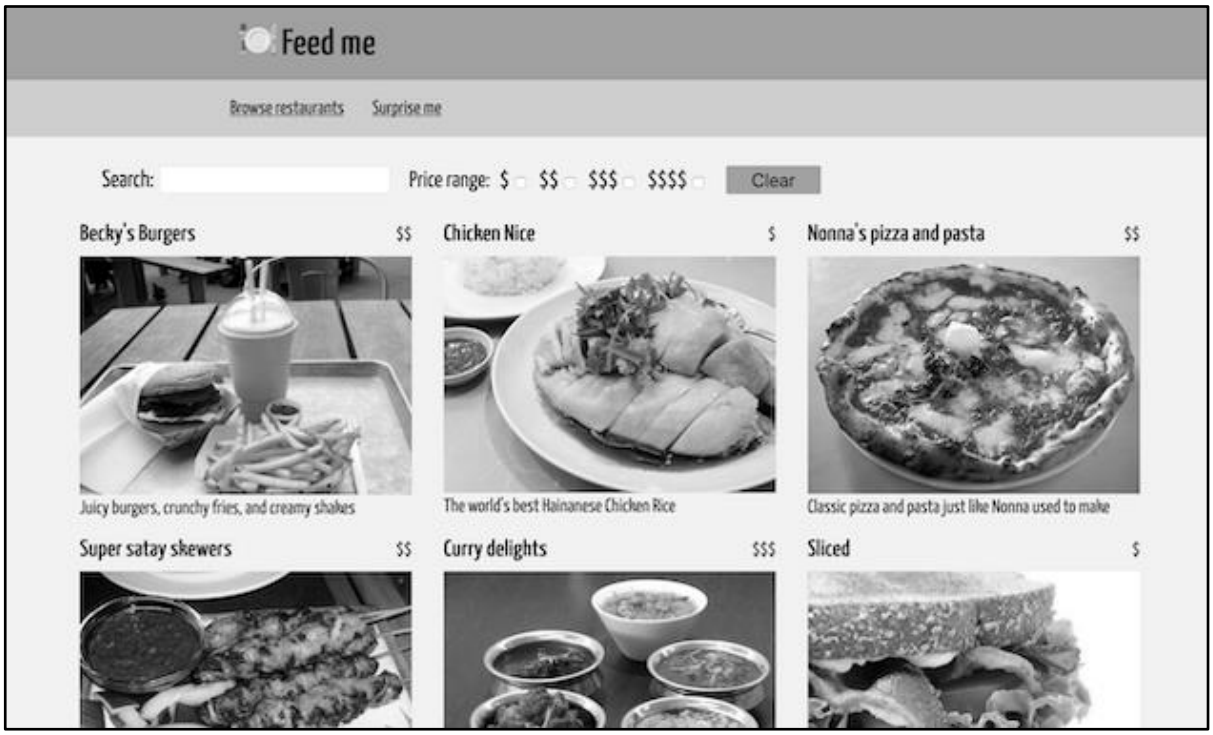
Figura 12 – Separação dos elementos da página



Fonte: Jackson (2019)

Desta forma, uma equipe pode ficar responsável pela criação da aplicação contêiner, enquanto outras, de forma autônoma e independente, desenvolvem, testam e implementam cada uma das páginas necessárias utilizando as ferramentas que forem mais adequadas. É importante que a abordagem que será utilizada esteja bem definida, pois ao implementar o projeto, o resultado deve ser um único site coerente e visualmente consistente, onde o consumidor não consegue visualizar os diferentes componentes, como na Figura 13.

Figura 13 – Página inicial do exemplo finalizado



Fonte: Jackson (2019)

CAPÍTULO 4 – VANTAGENS E DESVANTAGENS DA UTILIZAÇÃO DE MICRO *FRONTENDS*

A arquitetura de micro *frontends*, pelo fato de ser derivada da arquitetura de microsserviços, compartilha de muitas de suas vantagens e desvantagens, dependem tanto da execução das técnicas na aplicação, quanto das abordagens e tecnologias utilizadas. Neste capítulo, serão apresentados alguns pontos de atenção ao realizar um projeto orientado a micro *frontends*.

4.1 Estudo de vantagens

4.1.1 Atualizações incrementais

Característica fundamental para a cultura de *DevOps*. Permite que atualizações e correções sejam implementadas de forma mais ágil, aplicando a modificação apenas no componente necessário (JACKSON, 2019).

É uma das maiores vantagens obtidas ao deixar de utilizar uma arquitetura monolítica, possibilitando ainda a migração de uma aplicação monolítica para uma orientada a micro *frontends*. Esta migração pode ser realizada de forma gradual, utilizando o monolito já existente como base, acrescentando novas funcionalidades e movendo as antigas para novas aplicações (JACKSON, 2019; SILVA, 2020).

4.1.2 Times autônomos

Após definir o escopo do projeto e as abordagens que serão utilizadas, as equipes podem construir suas respectivas funcionalidades, fazendo uso das tecnologias que julgarem mais adequadas, desenvolvendo, testando, compliando e implementando-as sem depender de fatores externos. Desta forma, a velocidade com que a aplicação é produzida é aumentada (GEERS, 2020).

4.2.3 Códigos menores e desacoplados

Uma aplicação orientada a micro *frontends* consiste em aplicações menores,

desta forma, o código fonte torna-se menor e mais simples, se comparado com o de uma aplicação monolítica, facilitando sua manutenção e desenvolvimento de novas funcionalidades. Caso a aplicação torne-se complicada demais, ela pode ser dividida novamente. (JACKSON, 2019).

Esta característica pode também baratear a infraestrutura: como os projetos compilados serão menores, será necessário menos poder de processamento e armazenamento (GEERS, 2020).

4.2 Estudo de desvantagens

4.2.1 Redundância

Pelo fato de cada equipe desenvolver sua aplicação por inteiro, existe a necessidade de gerar um ambiente completo para desenvolvê-la, desde o servidor até as esteiras para integração. Consequentemente podem ocorrer redundâncias tanto de código, como também de bibliotecas, funções e ambientes (GEERS, 2020)

4.2.2 Complexidade operacional e de governança

Apesar do aumento da velocidade de desenvolvimento, a integração dos diferentes micro *frontends* em um único objeto, dependendo das tecnologias utilizadas, pode se tornar um pouco complexa, mesmo que existem poucas aplicações. Quanto mais aplicações forem desenvolvidas, mais difícil se torna a administração e governança do projeto. Além disso, pode surgir a incompatibilidade entre as diferentes tecnologias e *frameworks* utilizados pelas equipes (GEERS, c2021; JACKSON, 2019). Podem ocorrer também inconsistências entre os ambientes, tanto visuais como de comportamento (GEERS, 2020).

CAPÍTULO 5 – CONSIDERAÇÕES FINAIS

Pelo seu crescente reconhecimento, a arquitetura de micro *frontends* pode parecer a solução ideal para todos os problemas no desenvolvimento do *frontend*, mas devido sua natureza que facilita a escalabilidade, é mais bem aproveitada em projetos de porte médio ou grande, principalmente onde há influência da cultura de *DevOps*. Geers (2020) recomenda também que seja utilizada em projetos direcionados à web.

Para equipes pequenas, a abordagem não necessariamente trará vantagens, e pode acabar gerando desgaste devido à demanda de planejamento e as complexidades que pode acabar gerando. Então, é preciso considerar as vantagens e desvantagens da utilização de micro *frontends* bem como o escopo do projeto para avaliar se é adequada a aplicação da técnica.

Com relação a este trabalho, foi estudado um tema que, apesar de ter ganhado popularidade desde sua primeira introdução, por ser relativamente novo, ainda não possui muitos trabalhos acadêmicos dedicados a ele, principalmente de origem nacional, além da limitação de registros bibliográficos. Existem publicações em blogs e apresentações disponíveis na internet, no entanto, costumam ter boa parte de seu conteúdo derivado das mesmas referências utilizadas neste trabalho.

Este trabalho tinha como objetivo realizar um estudo teórico acerca da arquitetura de micro *frontends* e realizar uma análise das suas vantagens e desvantagens. Dentro do proposto, considera-se que o objetivo foi alcançado.

Deve-se lembrar que o objeto de estudo deste trabalho trata de um estilo de arquitetura que pode ser adaptado de diversas maneiras por quem o usa, as técnicas e abordagens mencionadas são passíveis de evoluir e/ou ser substituídas por outras, que poderiam ser desenvolvidas em futuros trabalhos.

REFERÊNCIAS

AMAZON WEB SERVICE. **O que é DevOps?**. c2021. Disponível em <https://aws.amazon.com/pt/devops/what-is-devops/>. Acesso em 30 mai. 2021.

_____. **O que são microsserviços?**. c2020. Disponível em <https://aws.amazon.com/pt/microservices/>. Acesso em: 10 nov. 2020.

ANNETT, Robert. What is a monolith? **Coding the Architecture**, 19 nov. 2014. Disponível em: <http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html>. Acesso em: 10 nov. 2020.

BASTOS, João Pedro da Silva. **Desenvolvimento web orientado a micro frontends**. 2020. Dissertação (Mestrado em Engenharia Informática - Engenharia de Software) – Curso de Engenharia em Informática, Instituto Superior de Engenharia do Porto, Porto, jul. 2020. Disponível em: <https://recipp.ipp.pt/handle/10400.22/16235>. Acesso em: 01 out. 2020.

BRANCO, Murilo Shinohata; BARROS, Rodolfo Miranda de. **Um estudo sobre DevOps**. Departamento de Computação, Universidade Estadual de Londrina, Londrina, ago. 2020. Disponível em: <http://www.uel.br/cce/dc/wp-content/uploads/ProjetoTCC_MURILO_SHINOHATA_BRANCOpdf.pdf>. Acesso em 30 mai. 2021.

CENTRO REGIONAL DE ESTUDOS PARA O DESENVOLVIMENTO DA SOCIEDADE DA INFORMAÇÃO. **TIC domicílios - 2019**. 2020. Disponível em: <https://cetic.br/pt/pesquisa/domicilios/indicadores/>. Acesso em: 01 nov. 2020.

E-COMMERCE BRASIL. **46% dos brasileiros fizeram mais compras online na pandemia, indica Mastercard**, 26 nov. 2020. Disponível em: <https://www.ecommercebrasil.com.br/noticias/brasileiros-compras-online-pandemia-coronavirus/>. Acesso em 20 abr. 2021.

EXAME. **Carreiras em TI seguem contratando na crise, diz professor da Let's Code**, 16 jul. 2020. Disponível em: <https://exame.com/carreira/programacao-sera-essencial-como-falar-ingles-diz-professor-da-lets-code/>. Acesso em: 01 nov. 2020.

FOWLER, Martin. Microsservices and the first law of distributed objects. **Martin Fowler**, 13 ago. 2014. Disponível em <https://martinfowler.com/articles/distributed-objects-microservices.html>. Acesso em 15 mai. 2021.

FOWLER, Martin; LEWIS, James. Microservices a definition of this new architectural term. **Martin Fowler**, 25 mar. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 10 nov. 2020.

FREITAS, Weber. a importância da sua empresa ter um site. **Administradores**, 20 fev. 2008. Disponível em: <https://administradores.com.br/artigos/a-importancia-da-sua-empresa-ter-um-site>. Acesso em: 01 nov. 2020.

GARLAN, David; SHAW, Mary. **An introduction to software architeture**. 1994. School of Computer Science, Carnegie Mellon University, Pittsburgh, jan.1994. Disponível em: <http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf>. Acesso em: 10 nov. 2020.

GEERS, Michael. **Micro Frontends in Action**. Manning, 2020. 296 p. Disponível em: <https://livebook.manning.com/book/micro-frontends-in-action/about-this-meap/v-4/>. Acesso em: 06 jun. 2021.

_____. **Micro Frontends**: Extending the Microservice Idea to Frontend Development. Disponível em: <https://micro-frontends.org/>. Acesso em: 06 jun. 2021.

GRANATO, Luísa. 15 Cargos em alta na área de tecnologia com salários de até R$ 45 mil. **Exame**, 28 jan. 2020. Disponível em: <https://exame.com/carreira/15-cargos-em-alta-na-area-de-tecnologia-com-salarios-de-ate-r-45-mil/>. Acesso em: 01 nov. 2020.

HAUSCHILD, Tatiana. **A influência da presença digital das empresas nas decisões de compra dos consumidores**. 2017. Monografia (Graduação em Administração - LFE Administração de Empresas) – Universidade do Vale do Taquari - Univates, Lajeado, 27 nov. 2017. Disponível em: <http://hdl.handle.net/10737/1969>. Acesso em: 01 nov. 2020.

JACKSON, Cam. Micro Frontends, **MartinFowler.com**, 19 jun. 2019. Disponível em: <https://Martinfowler.com/articles/micro-frontends.html>. Acesso em: 03 jun. 2021.

MEZZALIRA, Luca. **Building Micro-Frontends** (Early Release). O'Reilly, 2021.

SILVA, Celso Henrique. Micro frontends - Uma abordagem de microservices para o front-end. In: QCON SÃO PAULO 2019. 2019, São Paulo. **Palestra...**. São Paulo, mai. 2019. Disponível em:<https://www.infoq.com/br/presentations/micro-frontends-microservice-front-end/?utm_source=qconsp2019&utm_medium=palestras>. Acesso em 06 jun. 2021.

SOUSA, Leandro Filipe Ribeiro. **DevOps: estudo de caso**. 2019. Trabalho de projeto (Mestrado em Sistemas de Informação de Gestão) – Instituto Superior de Contabilidade e Administração de Coimbra, Coimbra, out. 2019. Disponível em: <http://hdl.handle.net/10400.26/31932>. Acesso em 18 mai. 2021.

SOUZA, Ítalo Andrade de; PELISSARI, William Roberto. (2017). Microserviços como alternativa de arquitetura monolítica. **Revista de Pós-Graduação Faculdade Cidade Verde**. Maringá, v.3, n.2. 2017. Disponível em <https://revista.unifcv.edu.br/index.php/revistapos/article/view/66/79>. Acesso em 20 mai. 2021.

THOUGHTWORKS. **Micro Frontends**. Disponível em: <https://www.thoughtworks.com/radar/techniques/micro-frontends>. Acesso em: 11 jun. 2021.

GLOSSÁRIO

API: Conjunto de rotinas e padrões estabelecidos por um *software* que possibilita o acesso externo controlado a este artefato.

E-commerce: Em português “comércio eletrônico”, trata-se de transação comercial realizada através de um equipamento eletrônico com acesso à internet.

Kanban: Modelo de metodologia ágil utilizado para controlar o andamento de um trabalho ou projeto, visando maximizar sua eficiência.

Refatorar: Processo de modificar um código a fim de melhorar sua performance e/ou facilitar sua interpretação, de forma que não altere seu resultado.

Scrum: Abordagem de gerenciamento de projeto, onde o trabalho é realizado em pequenos ciclos de atividades.