Estudo aprofundado sobre *micro frontends*

Ricardo Hideki Hangai Kojo
Orientador: Alfredo Goldman vel Lejbman
Coorientadores: Luiz Fernando O. Corte Real,
Renato Cordeiro Ferreira

Agosto 2021

1 Introdução

A arquitetura de microsserviços — surgida por volta de 2012 (Lewis e Fowler, 2014) — tem crescido em popularidade, tanto no campo acadêmico quanto na indústria, sendo adotada por empresas como Amazon, Netflix e Uber (Richardson, 2020). Essa arquitetura consiste em quebrar a aplicação em serviços pequenos, independentes e que trabalham em conjunto. O baixo acoplamento entre eles traz benefícios como manutenibilidade, escalabilidade e resiliência, características essenciais para grandes sistemas, que precisam ser confiáveis — sempre disponíveis e com o mínimo de falhas. No entanto, essa abordagem engloba apenas o *backend* da aplicação, deixando o *frontend* de lado (Peltonen *et al.*, 2021).

Assim, surge a arquitetura de ***micro frontends***, com o objetivo aplicar os conceitos de microsserviços para a camada de apresentação das aplicações. Nesta abordagem, o *frontend* é dividido em pequenas partes, que podem ser desde um componente até uma página. Essas partes são unidas a nível de cliente — navegador do usuário — ou de servidor, produzindo um resultado único e coeso.

Há uma grande intersecção entre os benefícios e *trade-offs* de microsserviços e *micro frontends*. Logo, é desejável adotar as duas arquiteturas, aproveitando o conhecimento e infraestrutura já disponíveis. No entanto, o conceito de *micro frontends* é recente — surgido em 2016 (Geers, 2017) — e existem poucos trabalhos acadêmicos relacionados. Ao contrário de microsserviços, que possui tecnologias e padrões de projeto específicos para esta arquitetura, há poucas ferramentas para *micro frontends*. Sendo assim, há diversas oportunidades a serem exploradas.

Deste modo, o objetivo deste Trabalho de Conclusão de Curso é entender o estado da arte da arquitetura de *micro frontends*, além de analisar e documentar sua implementação e evolução no contexto do **Elo7**, *marketplace* de produtos artesanais.

2 Fundamentação Teórica

2.1 Arquitetura de software

A arquitetura de *software* pode ser definida como "o conjunto de estruturas necessárias para compreender um sistema, que inclui os elementos que o compõe, as relações entre eles, e suas propriedades."(Bass *et al.*, 2013). Assim, fazem parte de uma arquitetura os fundamentos, propriedades, regras e restrições que guiam o projeto.

O objetivo de defini-la é facilitar aspectos como desenvolvimento, implantação, operação e manutenção (Martin, 2019), além de ajudar na análise dos chamados atributos qualitativos: escalabilidade, resiliência, segurança, manutenibilidade, entre outros. Logo, decisões arquiteturais são essenciais para a evolução de um sistema. O crescimento desordenado e extrapolação dos limites da arquitetura levam o sistema a se tornar um *big ball of mud* (ou grande bola de lodo), termo criado por Foote e Yoder (1997).

Outro conceito importante é o de estilos arquiteturais. Um estilo arquitetural determina um conjunto limitado de elementos e relações, dos quais é possível definir uma visão geral do projeto. Um sistema que respeita tais determinações pode ser considerado um membro daquele estilo arquitetural. Por fim, é importante notar que uma mesma aplicação pode adotar múltiplos estilos. (Richardson, 2018).

Nas próximas seções serão apresentados três estilos arquiteturais: monolítico (2.2), microsserviços (2.3) e micro *frontends* (2.4).

2.2 Arquitetura monolítica

Na arquitetura monolítica, a aplicação é estruturada num único bloco que contém toda a lógica necessária, e que resulta num único arquivo executável. Num projeto Java[1], por exemplo, o resultado é um único arquivo WAR.

Essa arquitetura se caracteriza por ser simples e direta, o que a torna ideal para novos projetos. No entanto, com o crescimento da equipe e da base de código, podem surgir problemas técnicos e organizacionais.

Dificuldades de manutenção, aumento da quantidade de *bugs*, desenvolvimento de novas funcionalidades lento, problemas de escalabilidade. Estes são alguns exemplos de problemas que podem surgir, e que baixam a moral dos desenvolvedores. Neste contexto, a arquitetura de microsserviços é uma alternativa (Richardson, 2018).

Antes de introduzir a arquitetura de microsserviços, serão apresentadas as motivações, benefícios e *trade-offs* no uso de monolitos.

2.2.1 Motivações

- **Velocidade de desenvolvimento**: a velocidade de entrega de novas funcionalidade é essencial, principalmente para novos projetos e *startups*;
- **Simplicidade**: o projeto deve ser simples de se entender e fácil de ser alterado, de forma que novos membros sejam capazes de se integrar e contribuir

[1] https://www.java.com/pt-BR/

rapidamente.

2.2.2 Benefícios

- **Desenvolvimento simples**: há diversas ferramentas que facilitam o desenvolvimento de aplicações monolíticas;
- ***Deploy* simples**: há apenas um arquivo a ser entregue em cada *deploy*;
- **Simples de escalar**: basta executar múltiplas instâncias da aplicação.

2.2.3 Trade-offs

- **Troca de tecnologias**: a adoção de uma nova ferramenta ou linguagem de programação pode ser muito custosa, o que é problemático caso a tecnologia utilizada se torne obsoleta;
- **Resiliência**: um único erro derruba todo o sistema, o que afeta negativamente a disponibilidade.

2.3 Arquitetura de microsserviços

De acordo com Lewis e Fowler (2014), "em resumo, o estilo arquitetural de microsserviços é uma abordagem na qual uma única aplicação é desenvolvida como um conjunto de pequenos serviços, cada um rodando seu próprio processo e se comunicando por meio de mecanismos leves, comumente APIs HTTP. Estes serviços são construídos sob as regras de negócio e com *deploys* independentes, feitos por um maquinário completamente automatizado. Há o mínimo de controle centralizado entre os serviços, que podem ser escritos em diferentes linguagens de programação e usar diferentes tecnologias para armazenamento de dados."

O desacoplamento entre os microsserviços possibilita escalabilidade, uma das características principais dessa arquitetura. Assim, múltiplos times podem trabalhar de forma independente, cada um com seu serviço. Por serem pequenos, os microsserviços também facilitam a manutenção, testes e *deploys*. Por outro lado, a complexidade de desenvolvimento aumenta consideravelmente. O sistema passa a ser distribuído, o que trás desafios em comunicação, consistência de dados, testes entre serviços, além da necessidade de uma infraestrutura robusta que dê suporte aos múltiplos serviços.

Na literatura, a arquitetura de microsserviços é mencionada como uma alternativa à sistemas monolíticos que enfrentam problemas organizacionais e técnicos. A passagem da arquitetura monolítica para microsserviços pode ser feita através do *Strangler Fig Pattern* (Fowler, 2004), no qual as funcionalidades presentes no monolito são gradualmente movidas para serviços independentes, até que o monolito se torne obsoleto e seja substituído pelo conjunto de microsserviços. Enfatizando: esse processo ocorre de forma gradativa, dado que uma reescrita completa do monolito é normalmente inviável, devido ao seu tamanho e complexidade.

Nas próximas seções, serão detalhadas as motivações, benefícios e *trade-offs* da adoção de microsserviços.

2.3.1 Motivações

- **Crescimento do sistema:** aumento da complexidade, quantidade de *bugs* e código-fonte, problemas organizacionais;
- **Escalabilidade**: o aumento na quantidade de desenvolvedores não resulta num aumento de produtividade ou velocidade de entregas;
- **Resiliência**: o sistema precisa estar disponível 100% do tempo. Eventuais falhas não devem interferir na disponibilidade da aplicação.

2.3.2 Benefícios

- **Diferentes tecnologias**: permite a experimentação e adoção de novas tecnologias. Cada serviço ou banco de dados pode utilizar a tecnologia mais adequada para o problema;
- **Autonomia**: os times passam a funcionar de forma independente, diminuindo a necessidade de ações coordenadas;
- **Independência**: com projetos independentes, vários aspectos são beneficiados: desenvolvimento, testes, manutenção, *deploy*;
- **Escalabilidade**: inclui o âmbito organizacional — é fácil aumentar os times de desenvolvimento — e estrutural — aumentar a capacidade da aplicação baseado na quantidade de usuários;
- **Resiliência**: caso ocorra um erro em um microsserviço, apenas esse serviço cai. Num sistema monolítico, construído completamente num único projeto, um erro derrubaria toda a aplicação.

2.3.3 Trade-offs

- **Complexidade**: num sistema distribuído, surgem novos obstáculos: como garantir consistência entre múltiplos bancos de dados? Como abordar comunicação assíncrona entre serviços? Como testar a interação entre múltiplos serviços de forma consistente?
- **Infraestrutura**: é necessário configurar e manter uma infraestrutura que dê suporte à nova arquitetura, com múltiplos serviços funcionando simultaneamente;
- **Divisão de limites de cada serviço**: para usufruir dos benefícios da arquitetura, é essencial evitar acoplamento entre serviços. Para isso, é necessário compreender e respeitar os limites de cada serviço. No entanto, encontrar esses limites também é um desafio;
- **Decisão de adotar microsserviços**: não existem balas de prata no desenvolvimento de software. A arquitetura de microsserviços, apesar da popularidade,

não é uma solução para todos os casos. Há uma grande curva de aprendizado para lidar com os *drawbacks* citados acima, de forma que a adoção dessa arquitetura pode trazer mais malefícios do que benefício.

2.4 Arquitetura de *micro frontends*

Micro frontends é "um estilo arquitetural no qual aplicações *frontend* independentes são unidas numa singularidade maior". O projeto é decomposto em pedaços menores e mais simples, que podem ser desenvolvidos, testados e entregues de forma independente, enquanto mantém a aparência de um único produto para o usuário final (Jackson, 2019).

De acordo com Peltonen *et al.* (2021), "*micro frontends* extende a ideia da arquitetura de microsserviços e muitos dos princípios de microsserviços se aplicam para *micro frontends*". Logo, é natural propor a implementação das duas arquiteturas simultaneamente (Mezzalira, 2021). Dado que os domínios são diferentes, *backend* e *frontend*, é possível aproveitar os conhecimentos sobre sistemas distribuídos e a infraestrutura existente, sem que haja conflitos entre os diferentes serviços. Por esses motivos, a abordagem de *micro frontends* é vista como o próximo passo após a implementação de microsserviços (Yang *et al.*, 2019).

As motivações, benefícios e *trade-offs* elencados por Peltonen *et al.* (2021) e Geers (2020) serão descritos nas seções abaixo.

2.4.1 Motivações

- **Crescimento do *frontend***: o aumento da complexidade, quantidade de código e problemas organizacionais são os principais motivos para a adoção;
- **Escalabilidade de desenvolvimento**: inclui necessidade de *deploys* e times independentes, velocidade de entrega, falta de inovação e *onboarding* lento de novos desenvolvedores;
- **Seguir as tendências**: algumas empresas implementam microsserviços e *micro frontends* apenas para seguir as tendências do mercado.

2.4.2 Benefícios

Vale notar que os benefícios observados são similares aos de microsserviços. São esses:

- **Diferentes tecnologias**: cada *micro frontend* pode adotar as tecnologias mais adequadas para cada situação;
- **Times multidisciplinares**: com o sistema completamente dividido em serviços, tanto *backend* e *frontend*, é possível organizar os times de forma multidisciplinar. Times especializados executam apenas uma única função: infraestrutura, *backend*, *frontend*, design, marketing etc. Já os times multidisciplinares contém pessoas com diferentes especialidades, trabalhando em conjunto numa área do negócio: busca, recomendação, pagamento etc;

- **Independência em diversos aspectos**: tanto os projetos quanto os times passam a ser independentes, o que facilita o desenvolvimento, testes e entrega de novas funcionalidades;
- **Escalabilidade**: inclui o âmbito organizacional — é fácil aumentar os times de desenvolvimento — e estrutural — aumentar a capacidade da aplicação baseado na quantidade de usuários;
- **Resiliência**: caso ocorra um erro em um *micro frontend*, apenas esse serviço cai. Num *frontend* monolítico, construído completamente num único projeto, um erro derrubaria toda a aplicação.

2.4.3 *Trade-offs*

- **Complexidade**: lidar com uma arquitetura distribuída, com múltiplos projetos e diferentes tecnologias;
- **Consistência de interfaces**: uma das premissas de *micro frontends* é que o resultado da união das partes deve ser coeso e indiscernível. Assim, todos os serviços devem compartilhar um padrão estético;
- **Governança**: definir qual time é responsável por um serviço pode se tornar um desafio quando múltiplos times o usam ou trabalham na mesma área de negócio;
- **Desempenho**: é necessário lidar com duplicação de código, dependências compartilhadas, quantidades de dados enviados ao usuário, no momento em que os serviços são unidos.

3 Proposta de Trabalho

3.1 Objetivos

- Estudar trabalhos acadêmicos e literatura cinza sobre *micro frontends*;
- Documentar a implementação de *micro frontends* feita pelo Elo7, durante o período deste trabalho;
- Entrevistar desenvolvedores do Elo7 sobre os impactos no dia a dia, trazidos pela adoção desta arquitetura;
- Disseminar o conhecimento sobre arquitetura de software e *micro frontends* dentro da empresa;
- Analisar entrevistas e implementação no Elo7. Verificar quais foram os benefícios e *trade-offs* observados;
- Analisar criticamente pontos levantados na literatura: todos os benefícios citados foram observados? Existe algum *trade-off* não mapeado? Quais adaptações de *micro frontends* são possíveis?

- Analisar criticamente decisão arquitetural do Elo7: a adoção de *micro frontends* foi correta? Existem outras alternativas?

3.2 Cronograma

Atividade	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Revisão bibliográfica	X	X	X	X	X	X	X	X	X	X
Participação em *meetups* e palestras				X	X	X				
Coleta e transcrição de depoimentos						X	X			
Análise dos depoimentos							X	X		
Escrita da monografia					X	X	X	X	X	X

Referências

Bass *et al.***(2013)** Len Bass, Paul Clements e Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 3a edição. Citado na pág. 2

Foote e Yoder(1997) B. Foote e J. Yoder. Big ball of mud. Citado na pág. 2

Fowler(2004) Martin Fowler. Stranglerfigapplication. https://martinfowler.com/bliki/StranglerFigApplication.html, 2004. Último acesso em 14/08/2021. Citado na pág. 3

Geers(2017) Michael Geers. Micro frontends. https://micro-frontends.org/, 2017. Último acesso em 21/06/2021. Citado na pág. 1

Geers(2020) Michael Geers. *Micro Frontends in Action*. Manning, 1a edição. Citado na pág. 5

Jackson(2019) Cam Jackson. Micro frontends. https://martinfowler.com/articles/micro-frontends.html, Junho 2019. Último acesso em 21/06/2021. Citado na pág. 5

Lewis e Fowler(2014) James Lewis e Martin Fowler. Microservices. https://www.martinfowler.com/articles/microservices.html, Março 2014. Último acesso em 21/06/2021. Citado na pág. 1, 3

Martin(2019) Robert C. Martin. *Arquitetura Limpa: O Guia do Artesão para Estrutura e Design de Software*. Alta Books Editora. Citado na pág. 2

Mezzalira(2021) Luca Mezzalira. *Building Micro Front-ends*. O'Reilley Media, Inc. Citado na pág. 5

Peltonen *et al.***(2021)** Severi Peltonen, Luca Mezzalira e Davide Taibi. Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, 136:106571. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2021.106571. URL https://www.sciencedirect.com/science/article/pii/S0950584921000549. Citado na pág. 1, 5

Richardson(2020) Chris Richardson. Who is using microservices. https://microservices.io/articles/whoisusingmicroservices.html, 2020. Último acesso em 30/06/2021. Citado na pág. 1

Richardson(2018) Chris Richardson. *Microservices Patterns*. Manning, 1a edição. Citado na pág. 2

Yang *et al.***(2019)** Caifang Yang, Chuanchang Liu e Zhiyuan Su. Research and application of micro frontends. *IOP Conference Series: Materials Science and Engineering*, 490:062082. doi: 10.1088/1757-899x/490/6/062082. URL https://doi.org/10.1088/1757-899x/490/6/062082. Citado na pág. 5