UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

Luiz João Carvalhaes Motta

UMA ANÁLISE DA ARQUITETURA DE MICRO FRONTENDS EM UMA APLICAÇÃO REAL

Florianópolis
2023

Luiz João Carvalhaes Motta

UMA ANÁLISE DA ARQUITETURA DE MICRO FRONTENDS EM UMA APLICAÇÃO REAL

Trabalho de Conclusão de Curso do Curso de Graduação em Ciência da Computação do Campus Florianópolis da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Raul Sidnei Wazlawick, Dr.

Florianópolis

2023

Luiz João Carvalhaes Motta

UMA ANÁLISE DA ARQUITETURA DE MICRO FRONTENDS EM UMA APLICAÇÃO REAL

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Ciências da Computação”e aprovado em sua forma final pelo Curso Ciências da Computação.

Florianópolis, 14 de Junho de 2023.

_____

Álvaro Júnior Pereira Franco, Dr.

Coordenação do Curso

Banca examinadora

_____

Prof. Raul Sidnei Wazlawick, Dr.

Orientador

_____

Profa. Fabiane Barreto Vavassori Benitti, Dra.

Avaliadora

Universidade Federal de Santa Catarina - UFSC

_____

Prof. Elder Rizzon Santos, Dr.

Avaliador

Universidade Federal de Santa Catarina - UFSC

Florianópolis, 2023

AGRADECIMENTOS

A meus amigos, que sem eles a jornada da graduação não teria sido tão divertida.

A minha namorada que me ajudou a deixar os meus dias mais leves.

A meus pais que sempre me apoiaram nas minhas decisões.

RESUMO

Uma das partes mais importantes no desenvolvimento do *software* é a escolha da sua arquitetura. Esta escolha irá ditar todo o relacionamento entre equipes e como essas equipes irão entregar seus requisitos. Muitas vezes essa escolha é negligenciada ou é feita de forma errônea, impactando o desenvolvimento do projeto a longo prazo. Além disso, hoje em dia, os desenvolvedores possuem uma grande variedade de ferramentas, sendo que cada uma dessas ferramentas podem ser mais adequadas para resolver um tipo de problema específico. Este estudo traz uma análise de uma aplicação web que se utiliza da estrutura arquitetônica de micro *frontends*. Esta análise é feita a partir de uma comparação com a arquitetura tradicional monolítica. A análise utiliza o método *Architecture-level Modifiability Analys* (ALMA), no qual se utiliza de cenários para entender qual arquitetura possui as características mais apropriadas para os requisitos em questão. Este trabalho apresenta características do desenvolvimento *web*, uma revisão sobre o que é arquitetura de *software*, uma revisão sobre as arquiteturas monolítica e de micro *frontends*, a aplicação do método ALMA em uma aplicação web e seus resultados.

Palavras-chave: micro frontends; seleção de arquitetura; desenvolvimento *web*

ABSTRACT

One of the most significant parts of software development is choosing its architecture. This choice will dictate the entire relationship between teams and how those teams will deliver their requirements. This choice is often neglected or made incorrectly, impacting the development of the project in the long term. In addition, nowadays, developers have a wide variety of tools, each of which may be better suited to solve a specific type of problem. This study brings an analysis of a web application that uses the architectural structure of micro frontends. This analysis is based on a comparison with the traditional monolithic architecture. The analysis uses the Architecture-level Modifiability Analyzes (ALMA) method, in which scenarios are used to understand which architecture has the most appropriate characteristics for the requirements in question. This work features web development characteristics, a review of what software architecture is, a review of monolithic and micro frontend architectures, and the application of the ALMA method into a web application and its results.

Keywords: micro frontends; architecture selection; web development

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ALMA - *Architecture-Level Modifiability Analysis*

ALPSM - *Architecture Level Prediction of Software Maintenance*

ATAM - *Architecture Tradeoff Analysis Method*

BPMN - *Business Process Model and Notation*

CDN - *Content Delivery Network*

CSS - *Cascading Style Sheets*

DDD - *Domain-Driven Design*

DMN - *Decision Model and Notation*

ESAAMI - *Extending SAAM by Integration in the Domain*

GWT - *Google Web Toolkit*

JS - *JavaScript*

KIE - *Knowledge is Everything*

OWP - *Open Web Platform*

PMML - *Product Model Markup Language*

HTML - *HyperText Markup Language*

HTTP - *HyperText Transfer Protocol*

SAAMCS - *SAAM, Complex Scenarios in Computing*

SACAM - *Software Architecture Comparison Analysis Method*

SBAR - *Scenario-Based Architecture Reengineering*

SUMÁRIO

1 INTRODUÇÃO

Desde a invenção da internet, há um crescimento contínuo de pessoas que a consomem (GLOBAL CHANGE DATA LAB, 2023) e esse crescimento, fez com que os navegadores de internet, os *browsers*, começassem a serem utilizados para executarem as aplicações *web*, na qual independem da plataforma ou sistema operacional utilizado pelo usuário (TAIVALSAARI *et al*., 2008). Uma aplicação *web* pode ser desde um editor de texto, onde este trabalho de conclusão de curso está sendo escrito, como um sistema bancário, uma plataforma de nuvem, entre outros (TAIVALSAARI *et al*., 2008).

A estrutura das aplicações *web* é baseada na arquitetura cliente-servidor (DOYLE; LOPES, 2005), sendo o cliente responsável pela camada de apresentação e o servidor pela camada de aplicação. Para comportar o crescimento da complexidade dessas aplicações foram criados alguns estilos arquitetônicos para a sua organização estrutural. A escolha e adoção desses estilos variou no decorrer dos anos (BETTS e GIL, 2023), e estes podem ser classificados em dois tipos principais: monolítico e distribuído (RICHARDS e FORD, 2020).

As arquiteturas monolíticas acabam sendo a escolha natural quando um projeto está sendo iniciado devido a sua simplicidade. Elas compartilham uma série de características, entre elas estão a facilidade de implantação, reutilização dos códigos e seus custos (RICHARDS e FORD, 2020). Mas todas compartilham os mesmos problemas relacionados a atualização de um componente e na eventual necessidade de escalar a aplicação para uma quantidade maior de usuários (NEWMAN, 2021). Com isso, projetos grandes e mais complexos, acabam adotando uma arquitetura distribuída.

Um dos estilos arquitetônicos do tipo distribuído que ganhou força nos últimos anos é a arquitetura de microsserviços (RICHARDS e FORD, 2020). Sua nomenclatura foi popularizada por Lewis e Fowler (2014). Richardson (2018) explica que a ideia principal deste estilo é quebrar uma arquitetura monolítica em partes menores, chamando cada uma destas partes de microsserviço. Cada um desses microsserviços são mantidos por um pequeno time, que tem um domínio único e uma implantação independente. Em contrapartida, esse desacoplamento fez com

que a complexidade da aplicação crescesse como um todo, através da necessidade de manter a comunicação entre estes microsserviços. Esse estilo arquitetônico não prescreve um modelo para a camada de aplicação, sendo assim, ainda é utilizado uma abordagem monolítica para esta parte (PELTONEN *et al*., 2021).

Mezzalira (2021) traz que a origem da arquitetura de micro *frontends* surgiu pela demanda de estrutura arquitetônica no cliente que acompanhasse a arquitetura de microsserviços, e com isso uma solução similar foi criada. Assim como a arquitetura de microsserviços, ela separa um grande cliente que é desenvolvido por múltiplos times em clientes menores mantidos por pequenos times, tendo em vista um propósito único e que possui uma implantação independente. Todavia, possui o mesmo problema da arquitetura de microsserviços, a complexidade na comunicação entre as partes. Conectar esses micro *frontends* de forma eficaz sem refletir em uma experiência negativa para o usuário é desafiador.

Muito se fala sobre os benefícios e malefícios da arquitetura de micro *frontends* na literatura (PELTONEN *et al*., 2021; GEERS, 2020). A fim de averiguar o uso desta estrutura, este trabalho traz uma análise sobre uma aplicação real que utiliza esta estrutura arquitetural, evidenciando os seus requisitos e por fim, trazendo se a escolha estrutural foi adequada. Para esta análise, este trabalho trouxe a estrutura monolítica a fim de ter uma análise comparativa utilizando o método *Architecture-Level Modifiability Analysis* (ALMA).

1.1 OBJETIVOS

Nesta seção são abordados os objetivos, geral e específicos, deste trabalho.

1.1.1 Objetivo Geral

O objetivo principal deste estudo é avaliar a escolha da estrutura arquitetônica de micro *frontends* de uma aplicação *web*, chamada de KIE Sandbox. Essa avaliação será realizada utilizando o método ALMA, no qual possui uma seção de como realizar a seleção entre arquiteturas candidatas para um mesmo projeto. As estruturas arquitetônicas a serem comparadas são a de micro *frontends* e a

monolítica. Com isto, esta monografia visa evidenciar se a utilização da arquitetura de micro *frontends* foi, ou não, adequada para o caso proposto, auxiliando desenvolvedores em sua escolha arquitetural em projetos futuros ou atuais.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Verificar o estado da arte com relação a micro *frontends* e métodos de análise de arquiteturas;

- Aplicar um método de avaliação de arquitetura, comparando a arquitetura monolítica e de micro *frontends* para a aplicação KIE Sandbox;

- Avaliar se a escolha arquitetural de micro frontends foi adequada para a aplicação analisada.

- Auxiliar desenvolvedores em sua escolha arquitetural em projetos futuros.

1.2 METODOLOGIA

A pesquisa realizada neste trabalho visa estudar a realidade existente de uma aplicação *web* que possui a estrutura arquitetural de micro *frontends*, validando se a sua escolha foi apropriada ou não. Para alcançar esse objetivo foram feitas algumas etapas:

- Realizar uma análise da literatura referente a arquiteturas de *software* focado no desenvolvimento *web*;

- Seleção de uma aplicação *web* que se utiliza da estrutura de micro *frontends* para a sua composição;

- Consulta às pessoas que possuem um alto grau de conhecimento sobre a aplicação no formato de um questionário com perguntas abertas;

- Aplicação do método de análise arquitetural traçando as conclusões.

A pesquisa feita por este trabalho se utilizou das bases de dados do CAPES e da plataforma Scopus.

1.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho se divide em seis capítulos. Este primeiro capítulo apresenta a introdução do trabalho, seus objetivos, e metodologia. O capítulo 2 apresenta a fundamentação teórica, incluindo os conceitos de arquitetura de *software*, projeto de uma aplicação *web*, arquitetura de servidor, arquitetura de cliente, características da arquitetura monolítica e de micro *frontends* e por fim, o método ALMA. O capítulo 3 faz uma análise do estado da arte mostrando os métodos de análise de arquiteturas e análises já realizadas. O capítulo 4 mostra a aplicação que será utilizada na comparação de arquiteturas monolítica e de micro *frontends* e a aplicação do método de análise contextualizado no capítulo 2. No capítulo 5 é mostrada as conclusões tiradas a partir deste trabalho, e por fim, no capítulo 6, temos as referências deste trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo é abordado o referencial teórico deste trabalho.

2.1 ARQUITETURA DE SOFTWARE

Muitos pontos são levantados no que diz respeito à “arquitetura de *software*”. O termo “arquitetura” é bem abrangente, causando divergências entre estudiosos e até hoje não possui uma boa definição (RICHARDS e FORD, 2020). Fowler (2019) define como “o conhecimento compartilhado que os desenvolvedores especialistas possuem do design do sistema”, mas que a indústria de *software* define “arquitetura” como as partes mais importantes de um sistema de *software*. Fowler (2019) continua a discussão sobre a definição citando uma frase de Ralph Johnson:

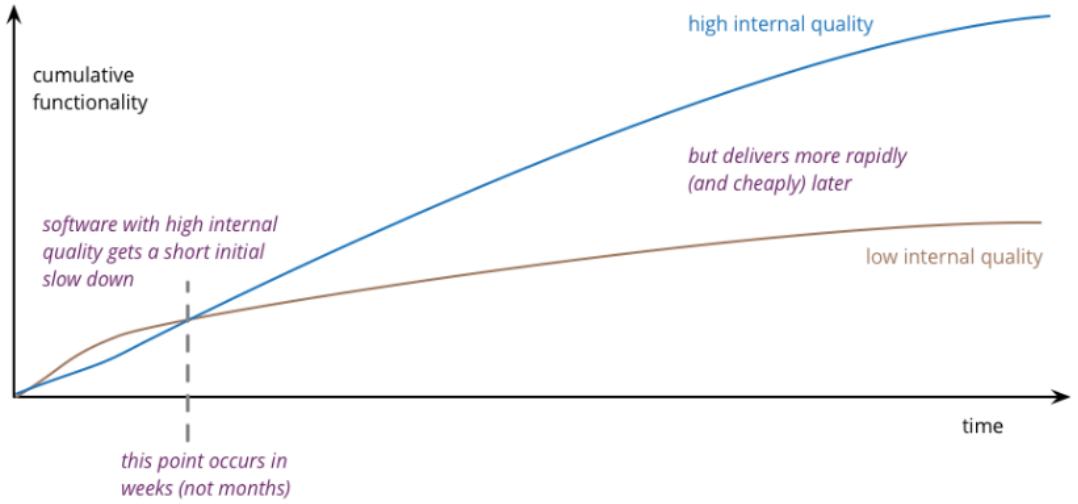
> “Arquitetura é sobre o que é importante. Seja lá o que for.” (FOWLER, 2003; tradução do autor)[1]

E conclui que apesar de ser uma frase banal, pensar sobre arquitetura é pensar sobre o que é importante no *software*, ou seja, o que faz parte dele. A qualidade da arquitetura é dificilmente percebida pelos usuários do *software*, mas não é imperceptível. Uma arquitetura ruim é o maior contribuinte para o crescimento de débito técnico, e que acaba por introduzir elementos que os próprios desenvolvedores possuem dificuldade em entender, deixando o processo de adicionar novas funcionalidades mais lento e complexo (FOWLER, 2019). A Figura 1 exemplifica esse processo de criação de funcionalidades mostrando dois *softwares* iguais, um com uma qualidade interna alta e outro com uma qualidade interna baixa. O de qualidade baixa começa a entregar as funcionalidades mais rapidamente, mas isso se inverte em poucas semanas, mostrando que o custo de se investir em uma alta qualidade interna irá ser compensado rapidamente (FOWLER, 2019).

---

[1] “Architecture is about the important stuff. Whatever that is” (FOWLER, 2003).

Figura 1 - Uma comparação de quantidade de funcionalidades sobre o tempo entre um *software* de alta qualidade e outro de baixa qualidade.



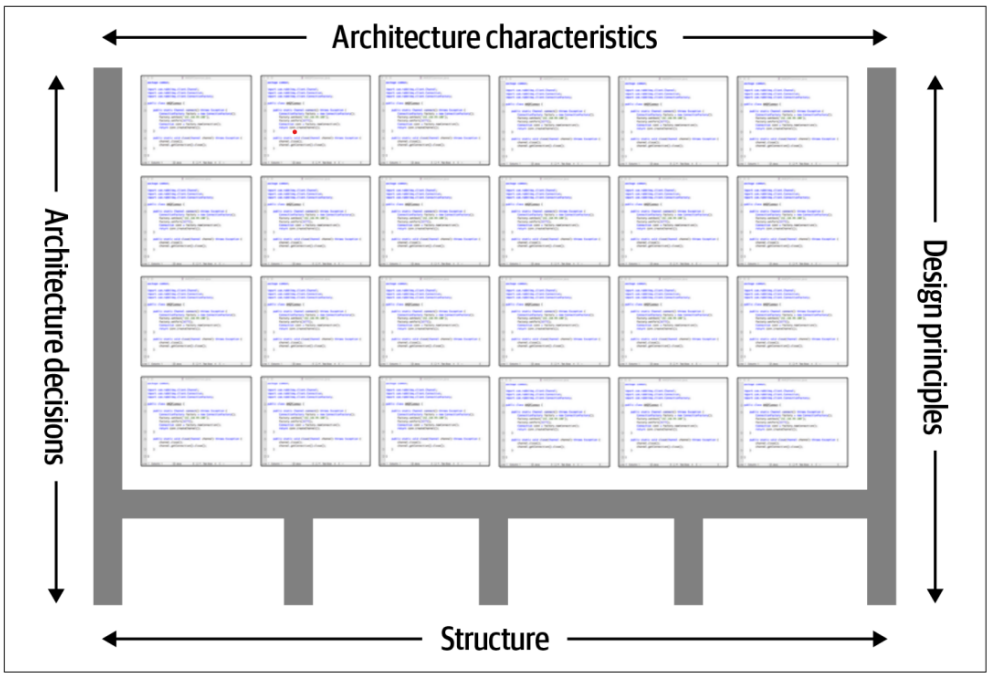
Fonte: (FOWLER, 2019)

Apesar da dificuldade em definir precisamente o que é englobado pela arquitetura de *software*, Richards e Ford (2020) mostram um meio de pensar sobre ela. Como ilustrado da Figura 2, eles separam a arquitetura de *software* em estrutura, características arquiteturais, decisões arquiteturais e princípios de *design*.

2.1.1 Estrutura

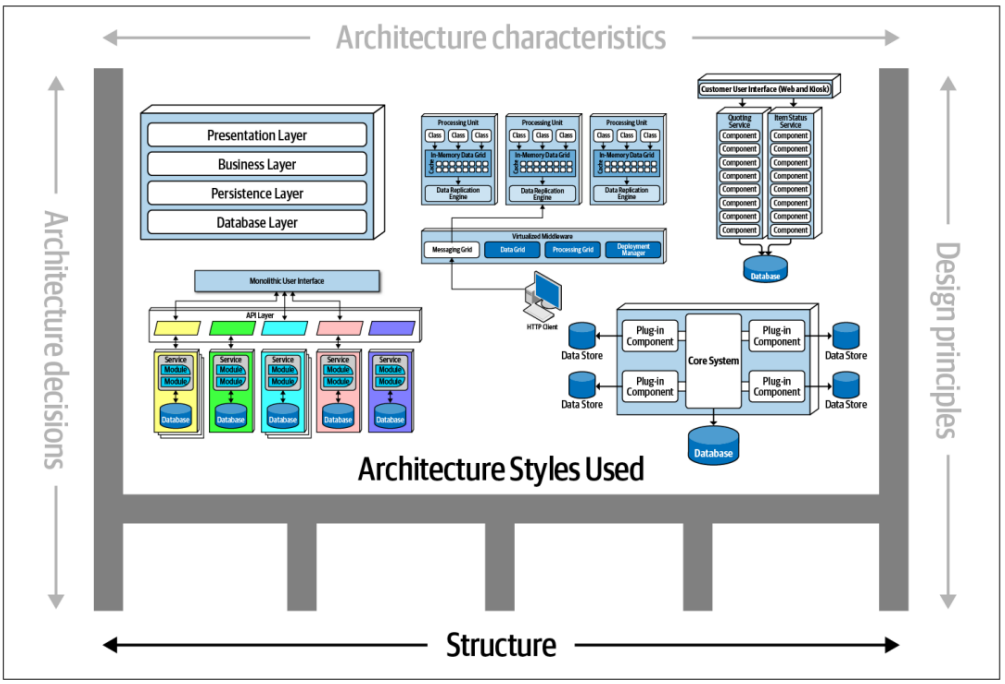
A estrutura do sistema, como ilustrado na Figura 3, se refere ao tipo arquitetural em que o sistema foi feito, como por exemplo microsserviços, em camadas, microkernel, entre outros. Descrever a arquitetura apenas pela estrutura não cobre todos os aspectos dela. Ela dita como será feita a separação dos componentes e como esses componentes se relacionam (RICHARDS e FORD, 2020). Esses tópicos serão cobertos nas próximas seções 2.1.1, 2.1.2, 2.1.3 e 2.1.4.

Figura 2 - Arquitetura é a combinação entre a estrutura, características da arquitetura, decisões arquiteturais, e princípios de design.



Fonte: (RICHARDS e FORD, 2020)

Figura 3 - Estrutura da arquitetura.

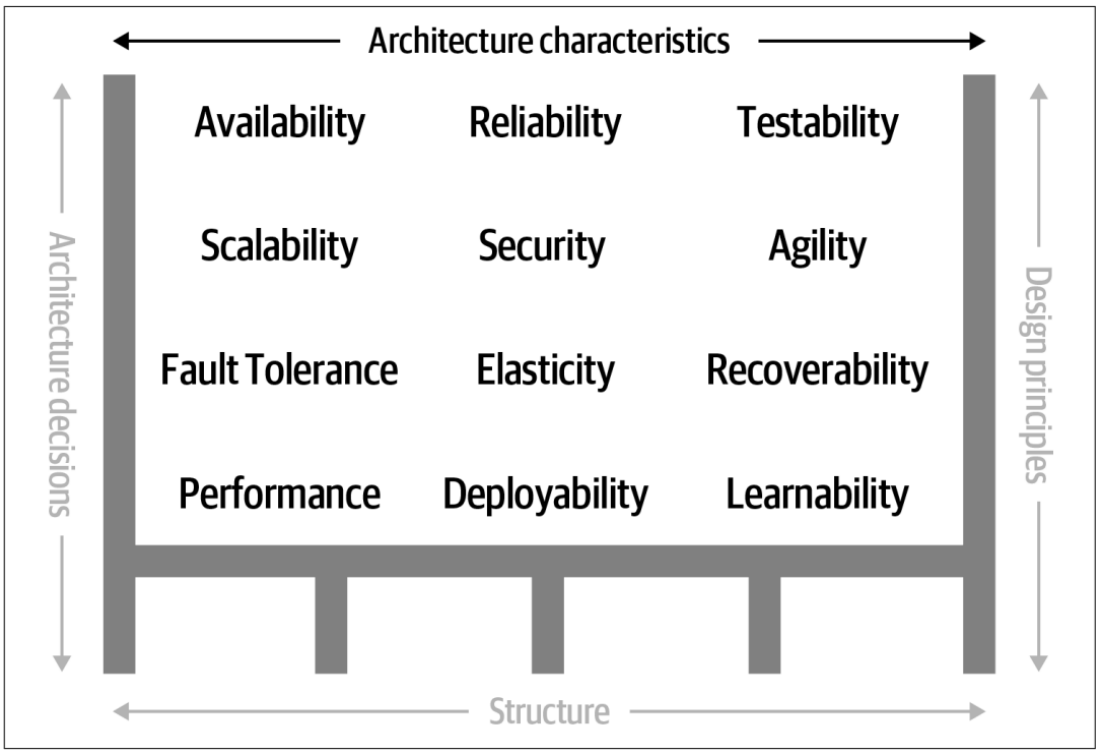


Fonte: (RICHARDS e FORD, 2020)

2.1.2 Características da arquitetura

As características da arquitetura definem o critério de sucesso do sistema. Todas as características presentes na Figura 4 não precisam do conhecimento de como o sistema funciona, mas ainda sim, são necessárias para o sistema funcionar corretamente (RICHARDS e FORD, 2020).

Figura 4 - Características da arquitetura.

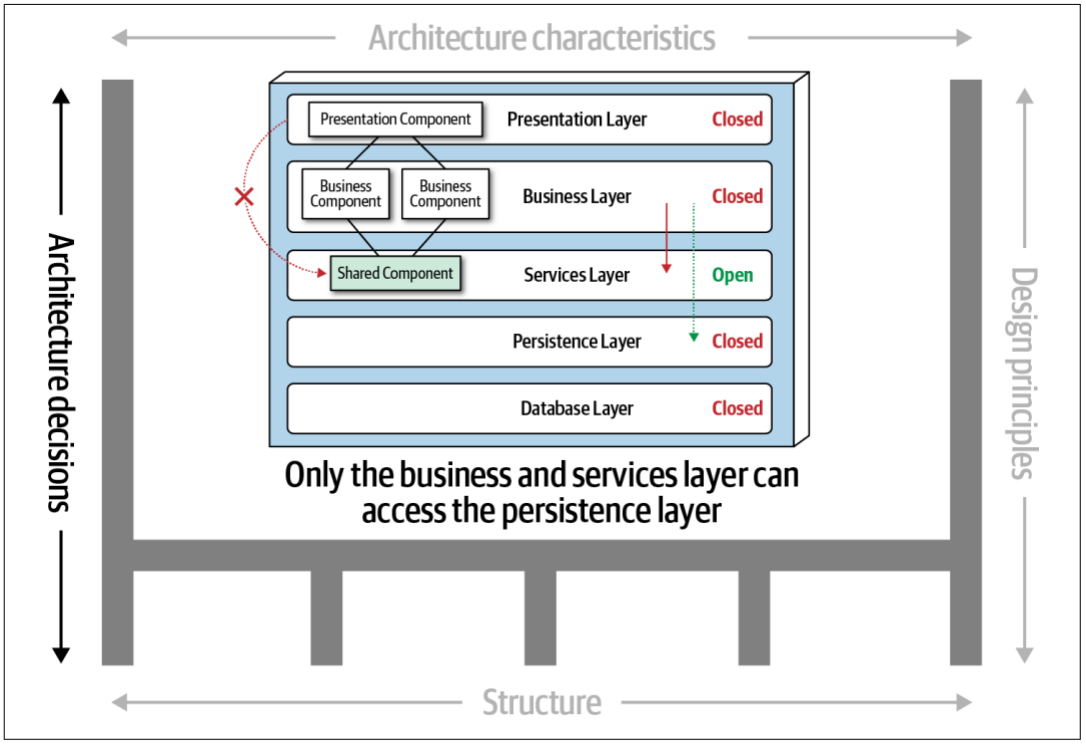


Fonte: (RICHARDS e FORD, 2020)

2.1.3 Decisões arquitetônicas

As decisões arquitetônicas são as regras para a construção do sistema. Como ilustrado na Figura 5, temos uma arquitetura em que a camada de apresentação não pode se comunicar diretamente com a camada de persistência, mas a camada de negócio e de serviço podem. Essas decisões constroem as limitações do sistema e direcionam os times de desenvolvimento para o que é e o que não é permitido (RICHARDS e FORD, 2020).

Figura 5 - Decisões arquitetônicas.

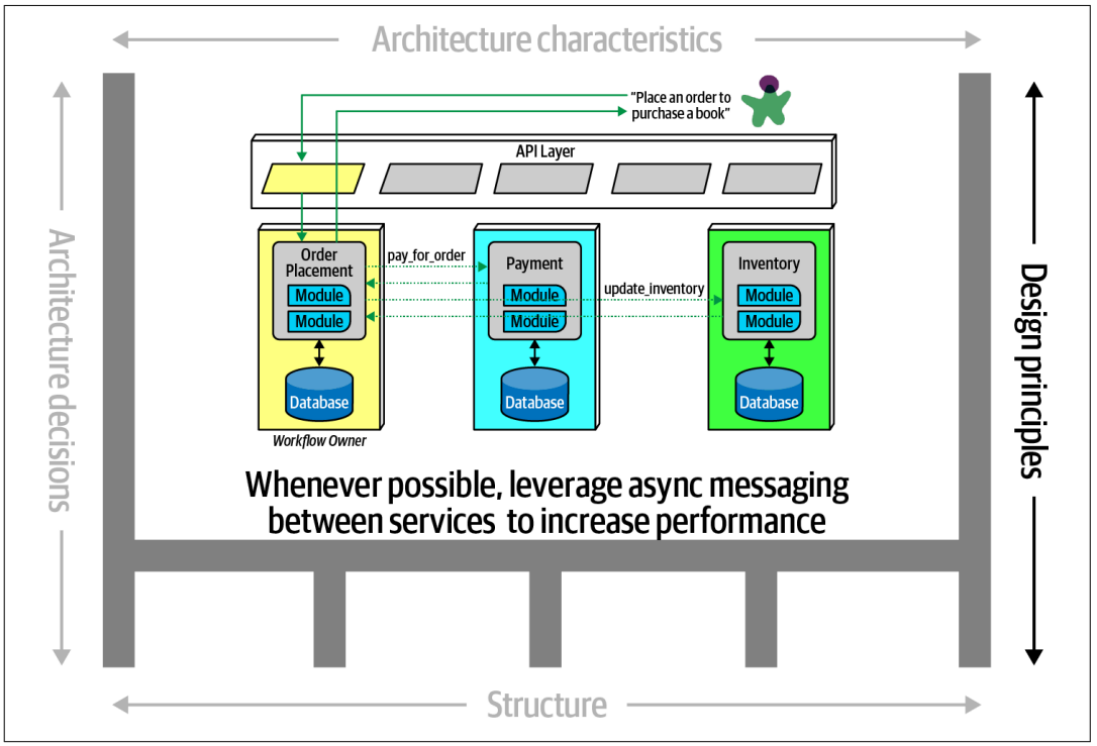


Fonte: (RICHARDS e FORD, 2020)

2.1.4 Princípios de design

Os princípios de design diferem das decisões arquitetônicas por serem um guia em vez de uma regra. A Figura 6 ilustra um princípio de design que todas as equipes devem seguir, que é a comunicação assíncrona entre os serviços de uma estrutura de microsserviços. Uma decisão arquitetural não poderia cobrir todas as condições e opções de comunicação possível, então um princípio de design é usado, deixando o desenvolvedor escolher qual o protocolo de comunicação mais apropriado para uma circunstância específica (RICHARDS e FORD, 2020).

Figura 6 - Princípios de design.



Fonte: (RICHARDS; FORD, 2020)

2.2 ARQUITETURA CLIENTE-SERVIDOR

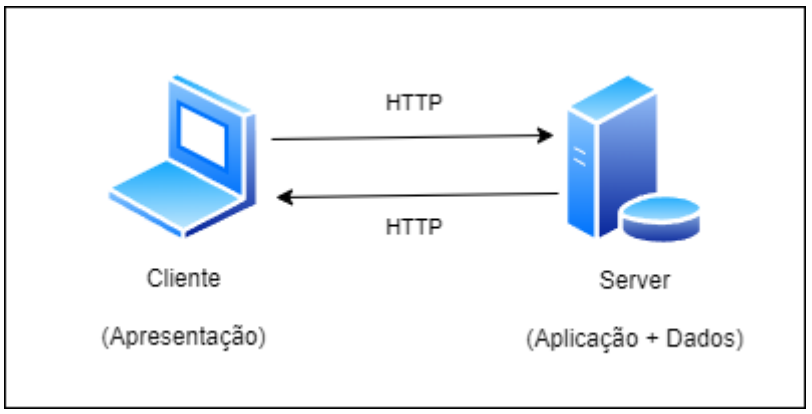
A *Web* é um sistema distribuído, que se utiliza da arquitetura cliente-servidor, onde os clientes se comunicam com os servidores por mensagens. O *HyperText Transfer Protocol* (HTTP) é um protocolo específico para comunicação dessas partes desse sistema, e as aplicações *web*, por sua vez, são implementadas dentro de uma camada de aplicação distribuída em cima do HTTP e, portanto, se utilizam desta arquitetura (DOYLE, LOPES, 2005).

Como o nome desta arquitetura sugere temos uma separação entre o cliente e o servidor, como é exemplificado na Figura 7. O cliente é a parte responsável pela camada de apresentação, onde o usuário irá interagir. Em uma aplicação *web*, o cliente é o navegador de internet, e a parte da aplicação responsável por funcionar no cliente é chamada de *frontend*. O *frontend* utiliza tecnologias que fazem parte da *Open Web Platform* (OWP), como *HyperText Markup Language* (HTML), *Cascading*

Style Sheets (CSS) e *JavaScript* (JS) para a sua composição (LINDLEY, 2019). Para o desenvolvimento *frontend*, é possível se utilizar de *frameworks* e ferramentas que transformam o código feito em outra linguagem de programação em arquivos compatíveis com a OWP. Alguns dos *frameworks* mais utilizados são o React.js, Angular e Vue.js (STACKOVERFLOW, 2022); e dentre as ferramentas mais utilizadas para gerar arquivos compatíveis com a OWP são o *Webpack*, *Rollup* e *Vite* (MOIVA, 2023).

O servidor tem o dever de implementar as funções do protocolo HTTP, que será utilizado para a comunicação das partes (DOYLE, LOPES, 2005) e pela camada de aplicação, onde, muitas vezes, a grande parte da lógica reside (REESE, 2000), essa parte também é chamada de *backend*. A camada de aplicação pode ser dividida entre outras camadas, como será abordado na próxima seção. Não necessariamente a aplicação segue esta exata configuração, sendo possível ter uma permuta, com um cliente com toda, ou grande parte, da lógica da aplicação, fazendo com que não seja necessário um servidor com lógica alguma, e por outro lado, podemos ter até um cliente nulo, sendo usado um terminal para se comunicar diretamente com o servidor (CDW, 2022).

Figura 7 - Exemplificação da arquitetura cliente-servidor.



Fonte: elaborado pelo autor

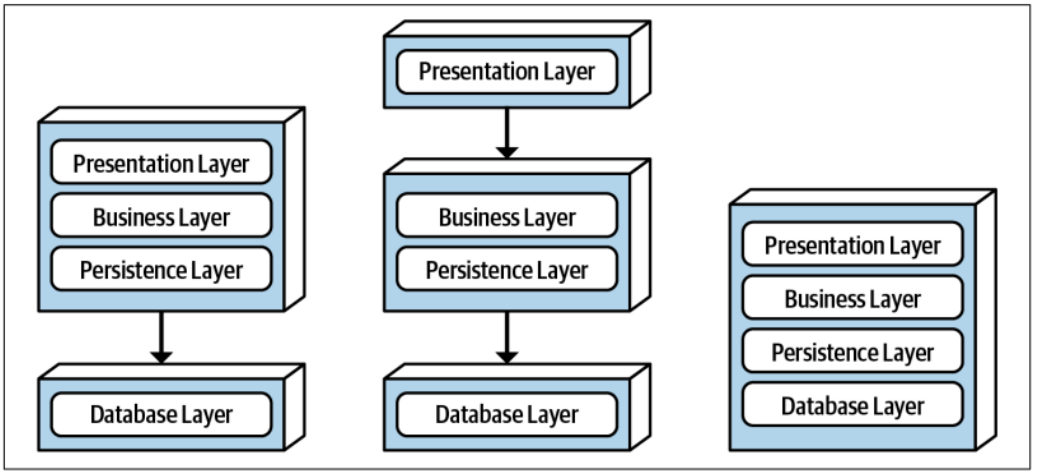
A arquitetura cliente-servidor é inerente a uma aplicação *web*, sendo que existem outros estilos de arquitetura com o intuito de estruturar este tipo de aplicação. Esses estilos possuem duas categorias principais: monolítica e distribuída (RICHARDS e FORD, 2020).

2.2.1 Arquiteturas Monolíticas

Dentro da categoria monolítica há alguns estilos de arquitetura, como em camadas, pipeline e microkernel. Esses estilos compartilham algumas características, como simplicidade e um baixo custo. Por outro lado, como pontos negativos, o aumento da disponibilidade de uma funcionalidade requer que a aplicação como um todo seja disponibilizada, e também, essa estrutura é ruim com relação à tolerância à falha (RICHARDS e FORD, 2020). Lewis e Fowler (2014) comentam que pela simplicidade dessa abordagem, ela acaba sendo a escolha natural para a estruturação da aplicação, quando é iniciado o seu desenvolvimento, e pode vir a ser a abordagem suficiente, dependendo da complexidade da mesma.

A Figura 8 exemplifica uma arquitetura em camadas, onde há três configurações possíveis: a camada de apresentação, de negócio e de persistência em uma mesma unidade, e o banco de dados separado; a camada de apresentação separada, a camada de negócio e de persistência juntas em uma unidade e o banco de dados separado; e por último, todas as camadas juntas em uma única unidade (RICHARDS e FORD, 2020).

Figura 8 - Arquitetura em camadas.



Fonte: (RICHARDS e FORD, 2020)

A sua estrutura traz alguns benefícios, como por exemplo: a implantação é facilitada, e a simplificação do fluxo de trabalho, da solução de problemas e do teste

da aplicação como um todo. Devido a sua base de código estar em uma mesma estrutura, o reuso de código também é favorecido (NEWMAN, 2021).

Por outro lado, com o crescimento das aplicações, há a possibilidade de ter múltiplos times trabalhando em uma mesma unidade, transformando toda a sua simplicidade em uma dificuldade. Fazer alterações sem impactar outros times se torna um desafio, cada unidade precisa ser implantada como um todo.

Tomando a terceira configuração da Figura 8, uma alteração na camada de apresentação, força a implantar as camadas de negócio, persistência e de banco de dados, mesmo sem alterações presentes. Também não é possível aumentar a disponibilidade de uma parte crítica da aplicação, sem aumentar a disponibilidade da aplicação como um todo. Utilizando a segunda configuração como base, caso seja necessário aumentar a disponibilidade da camada de negócio devido a grande quantidade de pessoas utilizando a aplicação, será necessário aumentar a disponibilidade da camada de persistência (LEWIS e FOWLER, 2014).

2.2.2 Arquiteturas Distribuídas

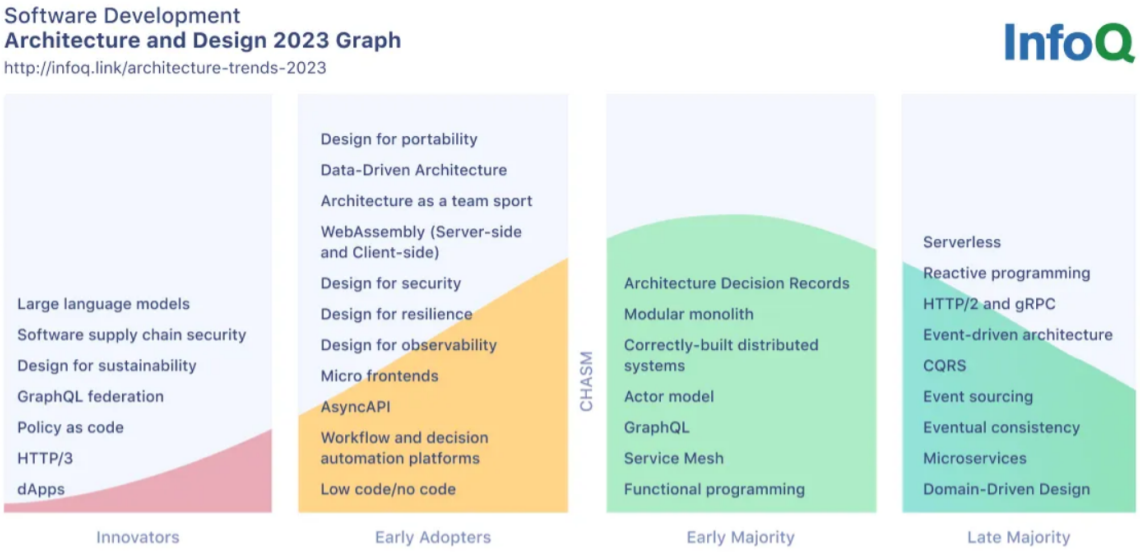
Richards e Ford (2020) discutem as arquiteturas distribuídas, comentando que são mais adequadas para aplicações mais complexas ou aplicações maiores. Algumas das arquiteturas distribuídas são: arquitetura baseada em serviços, arquitetura orientada a eventos, arquitetura baseada em espaço, arquitetura orientada a serviços e a arquitetura de microsserviços. Apesar de serem mais poderosas em performance, escalabilidade e disponibilidade do que as arquiteturas monolíticas, elas também possuem alguns problemas, sendo eles:

- A comunicação entre as partes pode falhar;
- A latência da comunicação entre as partes não é nula;
- A largura de banda não é infinita, então o tamanho do dado influencia na latência da comunicação;

- As comunicações entre os serviços não são sempre seguras;
- A topologia da rede pode mudar, e;
- Há um custo de comunicação entre as partes.

As arquiteturas distribuídas vêm se tornado bem populares e sendo bem adotadas nos últimos anos (BETTS e GIL, 2023), como é possível verificar na Figura 9. Entre elas uma que se destaca é a arquitetura de microsserviços.

Figura 9 - Tendências arquitetônicas e escolhas de design de 2023.



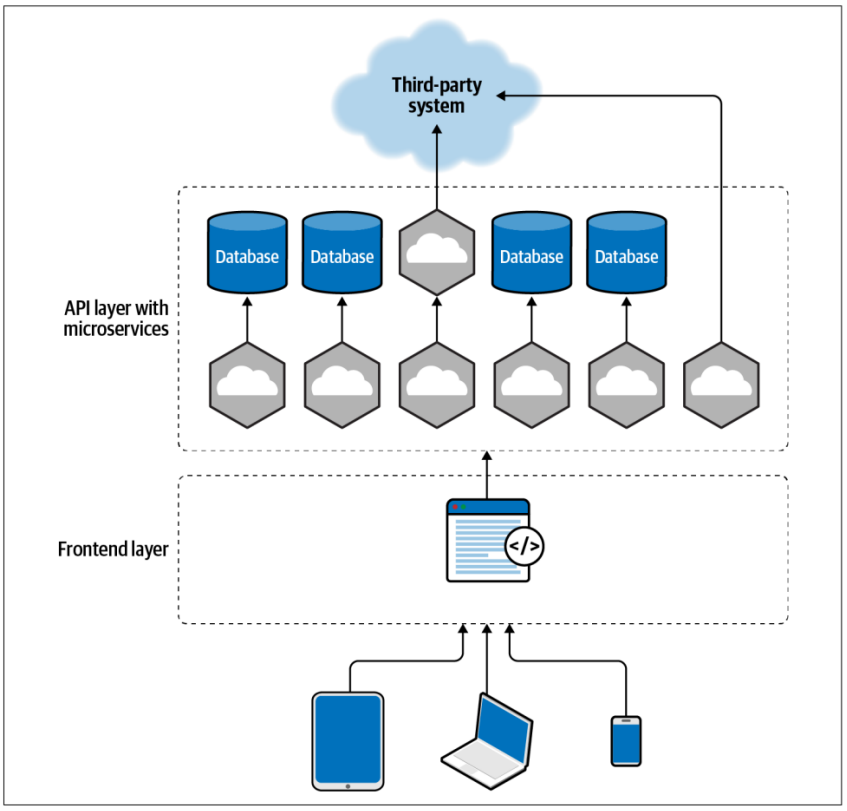
Fonte: (BETTS e GIL, 2023)

2.2.2.1 Microsserviços

Newman (2021) traz que a arquitetura de microsserviços é baseada em serviços que são modelados em torno de um domínio de negócio e que são independentemente implantados. Esses serviços encapsulam funcionalidades, e as deixam disponíveis para outros serviços, permitindo que um sistema complexo seja construído pela junção destes blocos de funcionalidades. A Figura 10 exemplifica essa arquitetura, mostrando a camada de aplicação sendo dividida em uma

quantidade razoável de microsserviços.

Figura 10 - Exemplificação da arquitetura de microsserviços.



Fonte: Adaptado pelo autor (MEZZALIRA, 2021)

Lewis e Fowler (2014) ditam que cada microserviço deve ter apenas um domínio de negócio, sendo que essa filosofia vem das ideias do *Domain-Driven Design* (DDD), que é uma abordagem para modelagem de projetos. Fowler (2014) traz que um dos conceitos base dessa abordagem é o *bounded context*, no qual representa um estilo de desacoplamento. O *bounded context* dita que todas as definições pertencentes a um domínio façam parte dele, sem um compartilhamento das suas informações. Dessa forma, cada microsserviço define em seu código apenas o que ele precisa para o seu domínio, em vez de acomodar informações desnecessárias. Um exemplo da aplicação dessa ideologia, é que esta arquitetura favorece a duplicação de código em vez do reuso, para assim, evitar que sejam feitos acoplamentos desnecessários (RICHARDS e FORD, 2020).

Outro conceito desta arquitetura é a implantação independente. Isto dita que

deve ser possível fazer uma alteração no microsserviço, implantá-lo e disponibilizá-lo para os usuários sem precisar implantar qualquer outro microsserviço. Isso só é possível caso os microsserviços estejam fracamente acoplados. Algumas escolhas de implementação podem tornar essa tarefa difícil, como por exemplo, o compartilhamento de um banco de dados (NEWMAN, 2021).

Esta arquitetura traz também, flexibilidade, pois cada time mantém os seus próprios microsserviços, e por conta do desacoplamento, eles podem escolher quais tecnologias favorecem o seu desenvolvimento. Isso faz com que os times sejam responsáveis pela tomada de decisões referentes à implantação e manutenção dos seus próprios serviços (MEZZALIRA, 2021).

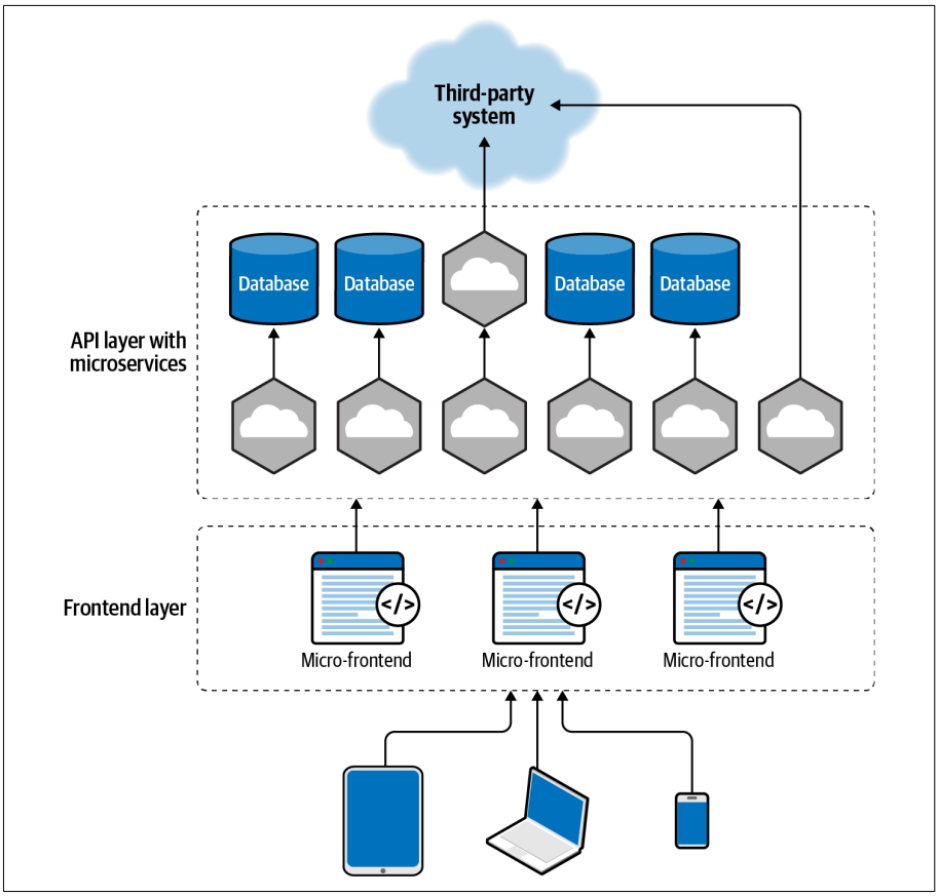
Sendo fiel ao DDD, a arquitetura de microsserviços idealmente abrangeria a camada de apresentação juntamente com a lógica da aplicação, pois a interface do usuário também faz parte do domínio, ou seja, do *bounded context*. Contudo, os aspectos práticos do particionamento exigido pelas aplicações *web*, tornam esse objetivo difícil (RICHARDS e FORD, 2020), como por exemplo a composição de dois *frontends* totalmente desacoplados em uma mesma interface. Desta forma, apesar deste estilo arquitetônico possuir implementações maduras para a camada de aplicação, a camada de apresentação continua monolítica. Devido ao sucesso desta arquitetura, a mesma ideia foi trazida para a camada de apresentação, originando a arquitetura de micro *frontends* (PELTONEN *et al*., 2021).

2.2.2.2 Micro *frontends*

O termo micro *frontend* é relativamente recente sendo trazido pela primeira vez em 2016 (JACKSON, 2019). Peltonen *et al*., (2021) traz que o micro *frontend* estende os conceitos de microsserviços para o desenvolvimento *frontend*, compartilhando seus princípios, benefícios e problemas. Nesta estrutura arquitetural é decomposto um cliente monolítico em clientes menores que são desenvolvidos e implantados de forma independente, como pode ser observado na Figura 11. Eles são modelados em torno de um domínio de negócio, e escondem os detalhes da implementação entre as partes. No entanto, há riscos na comunicação, possíveis problemas de performance e de inconsistências na experiência de usuário

(PELTONEN *et al.*, 2021).

Figura 11 - Exemplificação da arquitetura de micro *frontends*.



Fonte: (MEZZALIRA, 2021)

Para utilizar essa estrutura arquitetônica há princípios que devem ser pensados, referente a divisão, a composição e a comunicação das partes (MEZZALIRA, 2021).

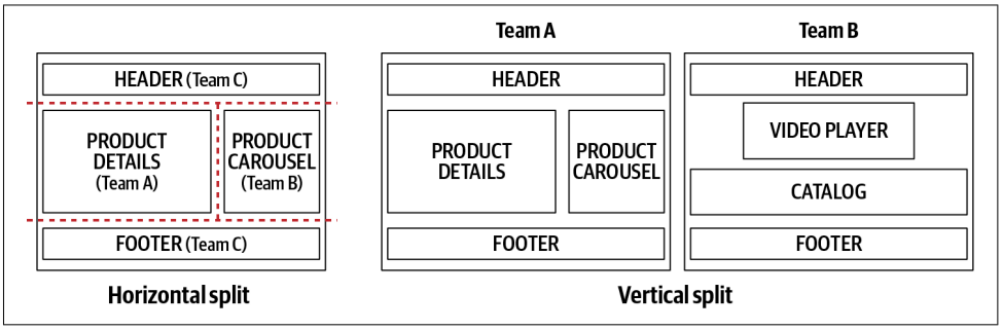
2.2.2.2.1 Divisão dos micro frontends

Peltonen *et al.*, (2021) mostra que a escolha da divisão do micro *frontend* é a primeira decisão e possui um forte impacto sobre as demais escolhas. As duas possíveis escolhas são ilustradas na Figura 12. É necessário definir se a aplicação terá múltiplos micro *frontends* carregados por página, ou seja, uma divisão horizontal, ou cada página será um micro *frontend* por inteiro, com cada time

responsável por um domínio de negócio, tendo uma divisão vertical.

No caso de uma divisão horizontal, um passo importante é a definição da comunicação entre os micro *frontends*. Um meio de se comunicar é a utilização de eventos, assim cada micro *frontend* pode se inscrever para receber os eventos dos outros micro *frontends*. No caso vertical é importante entender como as informações serão trocadas pelos micro *frontends*. Para ambos os casos, é preciso pensar também, em como as páginas irão informar que foram modificadas, podendo ser utilizado o armazenamento temporário ou permanente do navegador, ou pela URL (MEZZALIRA, 2021).

Figura 12 - Divisão horizontal e divisão vertical.



Fonte: (MEZZALIRA, 2021)

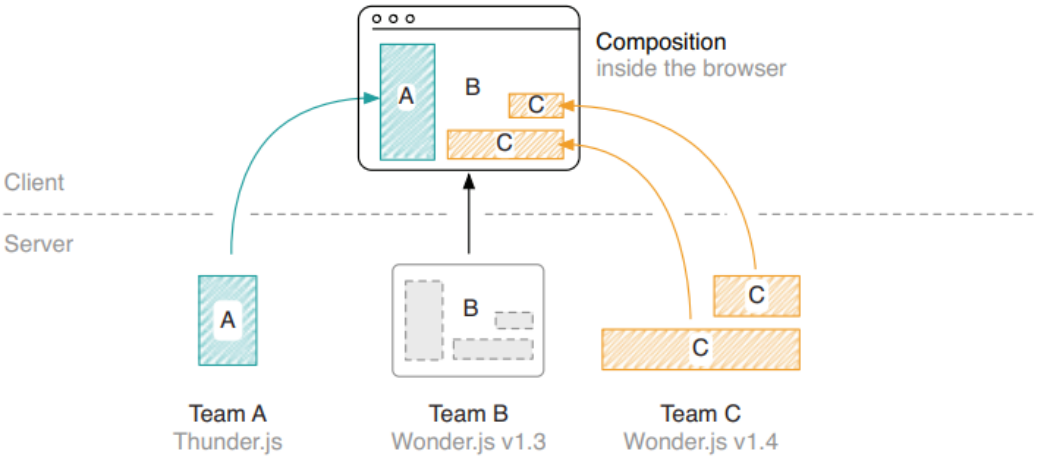
2.2.2.2.2 Composição dos micro frontends

A composição dita como as partes dessa estrutura arquitetônica serão montadas. Para compor uma aplicação micro frontend há duas opções mais comuns: composição pelo cliente e composição pelo servidor (GEERS, 2019). Em ambos os casos, a *application shell* está presente, uma estrutura que serve como um parente em comum para todas as partes, responsável pelo gerenciamento da comunicação entre os micro *frontends* e por montá-los corretamente (GEERS, 2019).

Em uma composição feita pelo cliente, a *application shell* pode carregar múltiplos micro *frontends*, como ilustrado na Figura 13. Neste caso, o micro *frontend* deve disponibilizar um arquivo JS ou HTML, que será utilizado pela *application shell* para adicionar dinamicamente ao *frontend*. Isso pode ser feito usando *iframes* ou um

mecanismo de transclusão, no qual substitui uma tag HTML reservada por um componente complexo (MEZZALIRA, 2021). *Iframes* permitem a criação de um novo contexto dentro de uma página do navegador, possibilitando a adição de uma nova página dentro da página existente, de forma totalmente isolada (MDN, 2023a).

Figura 13 - Composição pelo cliente.

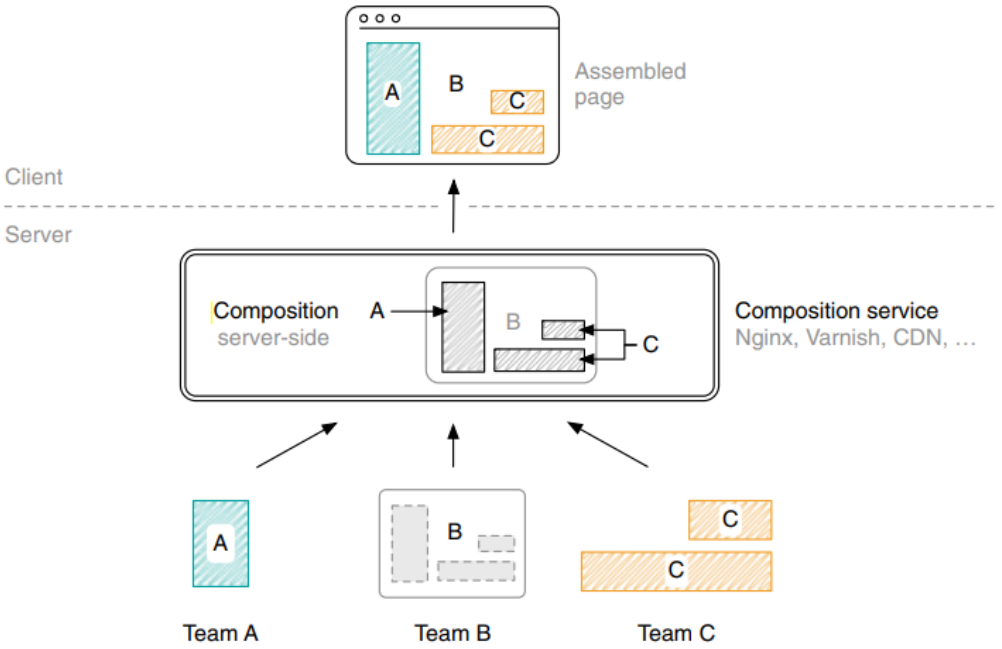


Fonte: (GEERS, 2019)

A composição feita pelo servidor pode ser feita tanto em tempo de compilação como em tempo de execução. Esse processo é ilustrado na Figura 14. O servidor compõe a página com os micro *frontend* e disponibiliza para um *Content Delivery Network* (CDN)[2]. Se essa página for cacheada, ela pode ficar disponível por um longo período de tempo, deixando essa solução bem performática. Por outro lado, se cada página for personalizada por usuário, esse tipo de solução pode ser bem oneroso, sendo necessário ter um plano relativo à escalabilidade dos servidores para manter a aplicação funcionando (MEZZALIRA, 2021).

[2] Rede de distribuição de conteúdo

Figura 14 - Composição pelo servidor.



Fonte: (GEERS, 2019)

2.2.2.2.3 Comunicação dos micro fontends

Independente da divisão escolhida, horizontal ou vertical, é preciso decidir como será feita a comunicação entre a *application shell* e os micro *frontends*. É importante manter o desacoplamento, sem as partes terem conhecimento dos detalhes da implementação, preservando sua independência e os princípios do *bounded context* (MEZZALIRA, 2021).

Um dos meios de se comunicar é utilizando um *EventBus*[3] (GEERS, 2019; PELTONEN *et al.*, 2021), que consiste de um mecanismo que permite que componentes desacoplados se comuniquem entre eles através de eventos por um barramento. Os micro *frontends* se subscrevem a esse barramento e então enviam notificações através de mensagens, que serão recebidos por todos os componentes que se inscreveram neste barramento.

[3] Barramento de eventos

Similar ao *EventBus,* há a opção de comunicação por *custom events*[4] (MEZZALIRA, 2021). Esse tipo de evento estende os eventos do navegador, permitindo adicionar uma mensagem customizada (MDN, 2023b) e são enviados para um objeto presente na aplicação, que todos os micro *frontends* possuem acesso. Um desses objetos é chamado de *window*, que é a representação da janela do navegador (MDN, 2023c). Caso seja utilizado uma abordagem utilizando *iframes* é preciso ressaltar que cada *iframe* possui o seu objeto *window*, e a comunicação entre estes objetos só pode ser feita por um método chamado de "*post message*" (MDN, 2023d).

Por fim, é possível também, a comunicação através do armazenamento do navegador ou por cookies. É possível salvar uma informação sensível em um armazenamento local para que outro micro *frontend* consuma-a (MEZZALIRA, 2021).

2.3 ALMA

ALMA é um método de análise arquitetural proposta por Bengtsson *et al*., (2004), que tem por objetivo observar a modificabilidade da arquitetura, ou seja, a capacidade que a arquitetura consegue suportar mudanças. Esse método pode ser utilizado para previsão de manutenção, avaliação de risco e para ajudar a selecionar uma arquitetura que tem uma maior cobertura dos requisitos propostos pelas partes interessadas. A análise é feita em cinco passos a partir de uma análise de impacto na mudança de cenários.

2.3.1 Os Cinco Passos do método ALMA

O método ALMA possui uma estrutura de cinco passos:

- Definição da meta: determinar qual o objetivo da análise;
- Descrição da arquitetura do *software*: escrever uma descrição das partes

[4] Eventos customizáveis

relevantes da arquitetura de *software*;

- Listagem de cenários: encontrar uma lista de cenários relevantes;
- Avaliação dos cenários: determinar o efeito da lista de cenários, e;
- Interpretação do resultado: tirar as conclusões do resultado da análise.

As técnicas apresentadas nos passos não podem ser escolhidas aleatoriamente, pois possuem uma relação entre elas.

2.3.1.1 Definição da Meta

O primeiro passo do método ALMA está relacionado a determinar a meta da análise. A meta da análise pode ser uma das seguintes:

- Previsão de custos de manutenção: estima o esforço requerido para modificar um sistema para acomodar futuras modificações;
- Avaliação de risco: identifica os tipos de mudanças que a arquitetura de *software* é inflexível, e;
- Seleção de uma arquitetura de *software*: compara duas ou mais possíveis arquiteturas de *software*, e escolhe a que melhor suporta os requisitos levantados.

2.3.1.2 Descrição da Arquitetura de *Software*

De um modo geral, para se fazer uma avaliação da modificabilidade do *software* o analista precisa das informações arquitetônicas, sendo necessário uma

descrição detalhada.

A análise em nível arquitetural é feita para identificar elementos que foram influenciados por uma mudança de cenário. Isso inclui os componentes que foram afetados diretamente e indiretamente. Os efeitos dos cenários são expressados usando alguma escala de medida que depende da meta definida.

Como mostrada na seção 2.1, a arquitetura de *software* não pode ser mostrada em um único modelo, é necessário analisar vários elementos do sistema. Para realizar a análise de impacto em nível arquitetural é necessário ter informações sobre:

- Decomposição do sistema em componentes;
- Relação entre os componentes, e;
- Relação do sistema com o ambiente.

As relações entre os componentes e a relação do sistema com o ambiente podem vir de formas diferentes e normalmente são implícitas, mas essa informação é crucial para a análise, pois evidenciam se haverá, ou não, um efeito cascata na mudança em um componente. Algumas relações são difíceis de serem conhecidas, pois são introduzidas no nível de implementação, sendo assim podem causar efeitos em cascata inesperados, comprometendo a análise.

2.3.1.3 Listagem de Cenários

Neste passo é feito o processo de encontrar e selecionar as mudanças de cenários que serão utilizados no próximo passo. A listagem envolve encontrar as partes interessadas a serem entrevistadas e a documentação dos cenários propostos por eles. O critério de seleção das mudanças de cenários está ligado a meta da análise:

- Se a meta é estimar o custo de manutenção, os cenários que correspondem à

maior probabilidade de ocorrer durante a vida do sistema são os desejados;

- Caso a meta seja a análise de risco, a escolha favorece cenários que expõe esses riscos;

- E por fim, se a meta é comparar arquiteturas diferentes, podem ser utilizadas qualquer uma das duas opções anteriores, focando em cenários que destaquem as diferenças entre as arquiteturas;

2.3.1.4 Avaliação dos Cenários

O próximo passo da ALMA é avaliar os efeitos das mudanças de cenários na arquitetura. Neste passo, a pessoa que está realizando a análise coopera com os arquitetos ou partes interessadas para determinar o impacto da mudança de cenário e expressa o resultado em uma forma compatível com a meta. Esse procedimento é chamado de análise do impacto em nível de arquitetura, e possui os seguintes passos:

- O primeiro passo é determinar quais componentes precisam ser modificados para implementar a mudança de cenário;

- O segundo passo é encontrar as funcionalidades dos componentes que foram afetados pelas mudanças. As mudanças podem propagar para as bordas do sistema; mudanças no ambiente podem impactar o sistema; ou mudanças no sistema podem afetar o ambiente;

- O terceiro passo é encontrar os efeitos cascata causados pelas modificações. As ocorrências de efeitos em cascatas podem ser cíclicas, podendo desencadear outros efeitos em cascata. Para saber se uma mudança irá causar um efeito em cascata nos componentes é preciso confiar nos arquitetos e nos desenvolvedores que foram entrevistados.

Os resultados devem ser expressados quantitativamente, descrevendo as alterações necessárias para a mudança de cenário, ou qualitativamente usando uma medida, essa podendo ser um ranque entre os cenários, linhas de código, pontos de função, entre outros.

Para avaliar e expressar os resultados da meta da comparação entre arquiteturas, pode ser escolhido um dos seguintes meios:

1. Para cada cenário determinar qual arquitetura possui um melhor suporte, sendo que os resultados são na forma de uma lista de cenários e as respectivas arquiteturas;
2. Para cada cenário é feito um ranqueamento de qual arquitetura suporta melhor;
3. Para cada cenário determinar qual o efeito nas arquiteturas que estão sendo comparadas. Podendo ser utilizado a quantidade de linhas de código sendo afetadas.

2.3.1.5 Interpretação do Resultado

Neste último passo a análise foi finalizada, sendo necessário interpretar os resultados e traçar as conclusões. O resultado depende inteiramente da meta definida e dos requisitos do sistema.

3 ESTADO DA ARTE

A análise do estado da arte foi separada em três partes. Primeiramente foram analisados os trabalhos que mencionaram a estrutura arquitetural alvo deste trabalho. Posteriormente foi feito o levantamento dos métodos de análise arquitetural e por último foi feita a pesquisa de análises já realizadas, utilizando um dos métodos de análise.

3.1 MICRO FRONTENDS

A seguinte *string* de busca foi utilizada no portal CAPES da biblioteca universitária:

“micro frontend” OU “micro-frontend” OU “micro frontends” OU “micro-frontends”

O mesmo termo em várias grafias diferentes foi utilizado para remover a chance de possíveis divergências na escrita do termo. Esta string de busca obteve 13 resultados, dentro deles, 4 sendo conjunto de dados, e 3 sendo o mesmo artigo, totalizando 7 referências distintas, dessas nenhuma abordando uma comparação entre a arquitetura de micro *frontends* e a arquitetura monolítica. Na plataforma Scopus foi feita uma busca com uma *string* de busca equivalente:

“(TITLE-ABS-KEY ("micro frontend") OR TITLE-ABS-KEY ("micro frontend") OR TITLE-ABS-KEY ("micro-frontend") OR TITLE-ABS-KEY ("micro-frontend"))”

Resultou em 19 documentos, entre artigos de conferências, artigos de periódicos e conferências, todos datados a partir de 2019. Devido uma quantidade maior de documentos, foi utilizada essa base de dados para as demais pesquisas.

3.2 MÉTODOS DE ANÁLISE ARQUITETURAL

Para encontrar métodos de análise arquitetural de *software* foi utilizada a seguinte *string* de busca na plataforma Scopus:

“TITLE-ABS-KEY ("software architecture evaluation")”

Resultando em 165 documentos datados desde 1997. Devido a grande quantidade de resultados, e pela quantidade de métodos, foi adicionada a busca a palavra “*survey*” e “*review*” para encontrar estudos que sumarizem esses métodos de análise.

“(TITLE-ABS-KEY ("software architecture evaluation") AND TITLE-ABS-KEY (survey OR review))”

Essa nova string de busca resultou em 27 documentos, e fazendo uma busca manual por eles, foi selecionado um documento que englobava diversos métodos: “*A survey on software architecture evaluation methods*” (PATIDAR, SUMAN, 2015). Nele é mencionado alguns métodos de análise: “*Simulation Analysis and Modeling”* (SAAM), “*Architecture Tradeoff Analysis Method”* (ATAM), ALMA, *“SAAM for Complex Scenarios”* (SAAMCS), “*Scenario-Based Architecture Reengineering*” (SBAR), “*Architecture Level Prediction of Software Maintenance*” (ALPSM), “*Extending SAAM by Integration in the Domain*” (ESAAMI), “*Software Architecture Comparison Method*” (SACAM).

SAAM é um método de análise que foi descrito em 1993, vindo de uma tendência da busca de um melhor entendimento sobre os conceitos gerais da arquitetura, com o intuito de validar se o software entrega, ou não, os seus requisitos (DOBRICA; NIEMELÄ, 2002). Dessa forma, o sistema poderia ser corrigido nas etapas iniciais, mitigando custos elevados de reestruturação (DOBRICA; NIEMELÄ, 2002).

Dobrica e Niemelä (2002) trazem que os objetivos específicos deste método são: verificar as suposições e princípios arquitetônicos escolhidos em relação a

documentação que descrevem as propriedades desejadas de uma aplicação; análise de riscos inerentes à arquitetura; levanta possíveis conflitos de requisito ou design incompleto, a partir do ponto de vista de uma parte interessada, e comparar duas arquiteturas se baseando nos seus requisitos.

ATAM é um método de avaliação focado em múltiplos atributos de qualidade, como por exemplo, manutenibilidade, disponibilidade, segurança e outros (PATIDAR, SUMAN, 2015). Esse método é tido como uma evolução do método SAAM (BENGTSSON *et al*. 2004), e tem como objetivo mostrar as compensações entre as qualidade de software para uma arquitetura, e analisar quanto uma arquitetura de *software* satisfaz os requisitos de forma eficiente (PATIDAR, SUMAN, 2015).

SAAMCS é um método que é uma extensão do método SAAM, e tem como objetivo específico a análise de risco durante a modificação da arquitetura (PATIDAR, SUMAN, 2015).

Assim como ATAM, SBAR é focado em atributos de qualidade, mas especificamente em performance e confiabilidade. Estima o potencial do design da arquitetura de alcançar os requisitos de qualidade de software (DOBRICA; NIEMELÄ, 2002).

O método ALPSM possui como principal objetivo analisar a manutenibilidade de um sistema a partir do impacto de cenários no nível da arquitetura (DOBRICA; NIEMELÄ, 2002).

Assim como SAAMCS, ESAAMI é uma extensão do método SAAM. Neste método é integrado o método SAAM em um processo de desenvolvimento baseado no reuso. Assim, esse método tem todas as características do método SAAM, mas possui o intuito de criar uma base de conhecimento que possa ser reutilizada (DOBRICA; NIEMELÄ, 2002).

SACAM é um método que provém um meio para a seleção de arquitetura comparando possíveis arquiteturas candidatas, sendo esse o seu principal objetivo. Este método se utiliza das metas de negócio da aplicação para então derivar um critério. O critério é transformado em atributos de qualidade que são posteriormente refinados utilizando um método, como por exemplo a ATAM, em cenários de atributos de qualidade (STOERMER *et al.* 2003).

Como objetivo deste trabalho é realizar uma comparação entre arquiteturas

candidatas para uma aplicação, dos métodos apresentados, SAAM, ALMA e SACAM satisfazem esse critério. O método SACAM precisa da utilização de outro método para derivar os cenários, sendo assim, sua utilização seria mais onerosa para este trabalho. Ambos os métodos SAAM e ALMA são focados em cenários e na modificabilidade da aplicação, contudo o método SAAM não prescreve de forma explícita quais técnicas devem ser utilizadas em suas etapas e deixa a cargo da experiência do analista (BENGTSSON *et al*. 2004). Como o autor do presente trabalho não possui experiência na área de análise arquitetural e o método ALMA possui etapas bem detalhadas das técnicas a serem utilizadas, foi escolhido este método para realizar a análise da escolha arquitetural de micro *frontends* da aplicação *KIE Sandbox*.

3.3 TRABALHOS CORRELATOS

Foi feita a junção de ambos os resultados para encontrar trabalhos que já fizeram uma análise da arquitetura de micro *frontends* utilizando um dos métodos mencionadas.

“(TITLE-ABS-KEY ("micro frontend") OR TITLE-ABS-KEY ("micro frontend") OR TITLE-ABS-KEY ("micro-frontend") OR TITLE-ABS-KEY ("micro-frontend") AND (TITLE-ABS-KEY ("atam") OR TITLE-ABS-KEY ("alma") OR TITLE-ABS-KEY ("saam") OR TITLE-ABS-KEY ("saamcs") OR TITLE-ABS-KEY ("sbar") OR TITLE-ABS-KEY ("alpsm") OR TITLE-ABS-KEY ("esaami") OR TITLE-ABS-KEY ("sacam"))) “

A string de busca acima retornou apenas um resultado na plataforma Scopus na escrita do presente trabalho: “*Experiences on a Frameworkless Micro-Frontend Architecture in a Small Organization*” (MÄNNISTÖ *et al*., 2023).

O estudo traz uma retrospectiva de micro frontends, uma contextualização da aplicação que será analisada, a aplicação de micro frontends nesta aplicação e a avaliação da arquitetura da aplicação contextualizada.

A arquitetura de micro frontends foi primeiramente validada a partir de dados

estatísticos da utilização da aplicação, após isso foi feita uma análise com o método ATAM de forma comprobatória. A pessoa responsável pela aplicação do método era um arquiteto, líder técnico e que já possuía um treinamento sobre o método. Este estudo envolveu dez pessoas de diversos cargos e responsabilidades diferentes. Neste estudo foram levantados um total de 23 cenários e dentre estes cenários, foram escolhidos os 16 mais importantes para posteriormente serem avaliados pelo arquiteto responsável. Os resultados da análise do método ATAM mostraram um alinhamento com os resultados da análise estatística previamente feita, além de mostrar cenários que a aplicação deveria cobrir e que ainda não estavam implementados, mostrando a importância deste tipo de análise. Foi observado também, que foi feito um esforço grande na aplicação do método, sendo utilizado cerca de 50 horas de trabalho distribuído entre todos os participantes.

4 DESENVOLVIMENTO

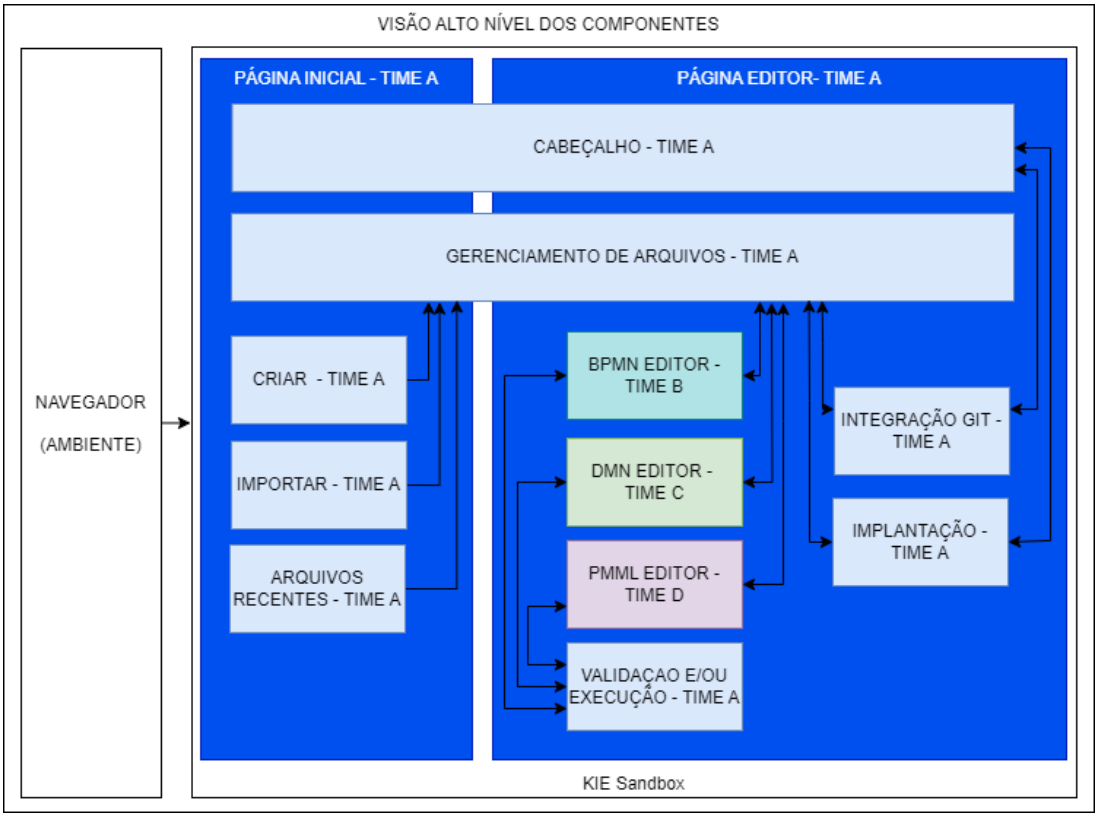
Neste capítulo é abordado o desenvolvimento deste trabalho.

4.1 A APLICAÇÃO

A aplicação escolhida para realizar a comparação se chama *KIE Sandbox*. Esta aplicação foi escolhida pelo autor, pois o mesmo participou do seu desenvolvimento nos últimos três anos, sendo assim, não foi necessário estudar outras aplicações para aplicar o método ALMA. Ela é uma aplicação *web open source*, que funciona totalmente no cliente, não possuindo assim um servidor, e se utiliza da arquitetura de micro *frontends* para a composição de uma de suas páginas. O projeto não possui uma documentação oficial, sendo assim todas as características cobertas nessa seção são de observação do autor deste trabalho. O projeto foi feito utilizando um *framework* de desenvolvimento *web* chamado de *React.js* e a ferramenta *Webpack* para gerar os arquivos HTML, CSS e JS. Os micro *frontends* foram feitos tanto utilizando o *framework* React.js quanto em um *framework* chamado de *Google Web Toolkit* (GWT). O GWT auxilia na criação de *frontends* para aplicações feitas na linguagem de programação Java (GWT, 2023).

O projeto é uma aplicação *web* focada em automação de negócios, e tem como público alvo analistas de negócio e desenvolvedores. Essa aplicação permite ao usuário criar, importar e editar graficamente arquivos *Business Process Model And Notation* (BPMN), *Decision Model and Notation* (DMN) e *Product Model Markup Language* (PMML), possui um gerenciamento de arquivos, integração com um sistema de versionamento Git, um sistema para realizar a implantação em um cluster Kubernetes e um sistema para validar os diagramas e executar arquivos DMN. A provável divisão de times, juntamente com a visão de alto nível dos componentes, e a suas relações podem ser observadas na Figura 15.

Figura 15 - Visão de alto nível dos componentes da aplicação *KIE Sandbox*.



Fonte: elaborado pelo autor

A aplicação conta com duas páginas. A página inicial pode ser vista na Figura 16, e é onde o usuário pode iniciar um novo projeto ou escolher abrir um projeto recente. Caso o usuário deseje criar um novo projeto, na parte “Criar” ele pode escolher criar um arquivo em branco ou abrir o arquivo exemplo do tipo BPMN, DMN ou PMML. O usuário pode também, criar um novo projeto na parte “Importar”, abrindo um arquivo ou projeto através de link de um projeto hospedado no *GitHub*[5] ou fazendo o upload do mesmo. Na parte “Arquivos Recentes” o usuário pode abrir um projeto previamente editado. E por último, o cabeçalho possui um informativo sobre os projetos que foram implantados e as credenciais do cluster de implantação e do provedor Git.

[5] https://github.com/

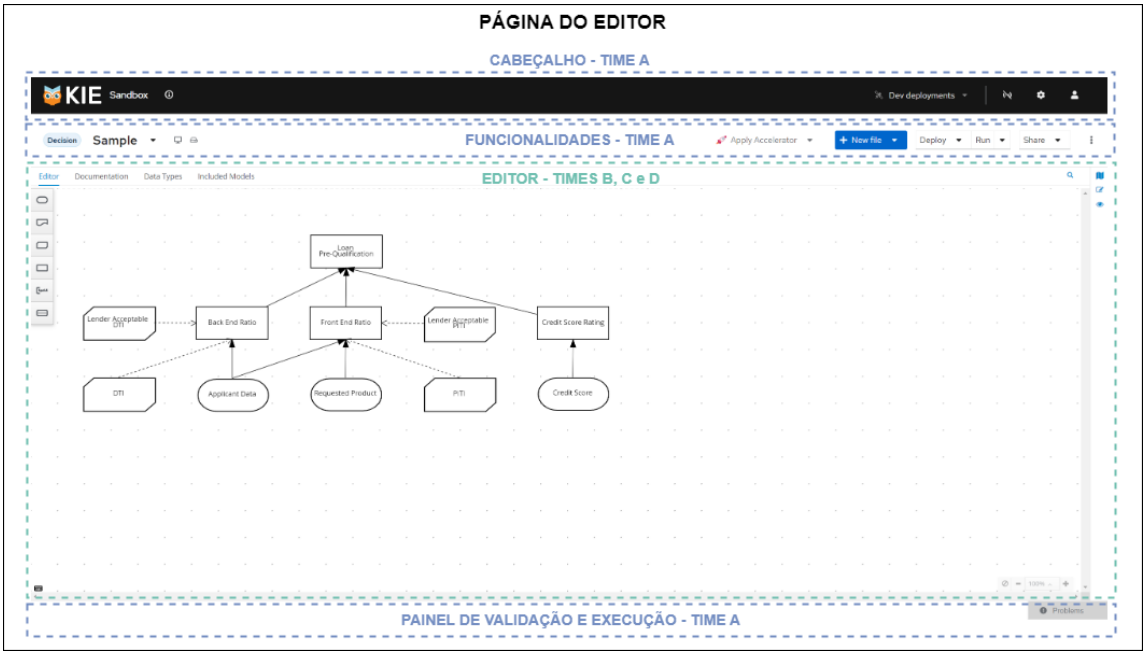
Figura 16 - Página inicial da aplicação *KIE Sandbox* separada por componentes.

Fonte: elaborado pelo autor

A página do editor pode ser vista na Figura 17, ela conta com um editor gráfico que varia com o tipo de arquivo escolhido. Na parte das “Funcionalidades”, há opção de navegar em outros arquivos do projeto, criar um novo arquivo em branco ou um exemplo, importar novos arquivos, implantar o projeto na nuvem, compartilhar o arquivo, salvar o estado atual em um provedor Git, e por fim, caso seja um arquivo DMN, há a opção de executá-lo. O “Painel de validação e execução” mostra a validação do diagrama atual, assim como erros de execução, caso seja um arquivo do tipo DMN. E por fim, cabeçalho apresenta as mesmas funcionalidades da página inicial.

Atualmente há três editores gráficos na aplicação *KIE Sandbox*, os editores de arquivos BPMN e DMN foram desenvolvidos utilizando o *framework* GWT e o editor de arquivo PMML foi desenvolvido utilizando o *framework* React.js. Os editores foram adicionados na aplicação no formato de micro *frontends*, assim o seu desenvolvimento, não impactou no desenvolvimento da aplicação *KIE Sandbox* como um todo.

Figura 17 - Página do editor da aplicação *KIE Sandbox* separada por componentes.



Fonte: elaborado pelo autor

A estrutura arquitetural de micro *frontends* pode ser observada na construção da aplicação. Na Figura 17 há uma divisão horizontal, entre os editores e as demais estruturas, que fazem parte da *application shell*. Por ser uma aplicação *web* que só possui a parte do cliente, a composição de micro *frontends* é trivial, sendo feita no cliente, mais especificamente utilizando *iframes*. A comunicação das partes é feita através de um barramento de eventos, em que há uma interface tanto para os editores quanto para *application shell*, tornando possível uma comunicação bi-direcional. Caso um novo editor tenha que ser adicionado à aplicação o único requisito é a implementação da interface dos editores. É possível adicionar novas estruturas, desde que a interface de ambas as partes sejam implementadas.

4.2 CONSULTA AOS ESPECIALISTAS

Como trazido por Bergston *et al*. (2004) na seção 2.3, o método ALMA requer que as partes interessadas e/ou as pessoas que possuem um alto grau de conhecimento sobre a aplicação sejam consultadas sobre as características da arquitetura da aplicação e dos possíveis cenários que podem acontecer com a

aplicação. Essa consulta se deu por meio de um questionário individual. Tanto as perguntas quanto as respostas podem ser consultadas no Apêndice A do presente trabalho. Com exceção do especialista número 4, todos os outros três foram responsáveis pelo início do desenvolvimento da aplicação, sendo assim, escolheram a arquitetura da aplicação. Além disso, é importante ressaltar que as pessoas que foram consultadas não tiveram acesso às respostas dos outros participantes, para evitar um possível viés.

As primeiras perguntas, da primeira a quarta, tiveram o intuito de traçar o perfil dos entrevistados. Foi questionado qual o cargo do especialista, o tempo de experiência no cargo, o tempo de experiência na área e a sua escolaridade. Suas respostas podem ser verificadas na Tabela 1.

Tabela 1 - Relação dos especialistas com seus perfis.

Fonte: elaborado pelo autor

Especialistas	Pergunta	Respostas
Especialista 1	Cargo	Arquiteto de software e Engenheiro de software especialista sênior
	Tempo de experiência no cargo	2 anos no cargo
	Tempo de experiência na área	15 anos de experiência
	Formação	Mestre em engenharia elétrica e Bacharel em Ciência da Computação
Especialista 2	Cargo	Arquiteto de software
	Tempo de experiência no cargo	2 anos no cargo
	Tempo de experiência na área	10 anos de experiência
	Formação	Bacharel em Ciência da Computação
Especialista 3	Cargo	Engenheiro de software especialista
	Tempo de experiência no cargo	1 anos no cargo
	Tempo de experiência na área	13 anos de experiência
	Formação	Bacharel em Ciência da Computação
Especialista 4	Cargo	Engenheiro de software
	Tempo de experiência no cargo	6 anos no cargo
	Tempo de experiência na área	8 anos de experiência
	Formação	Bacharel em Ciência da Computação

A quinta pergunta teve o objetivo de entender e confirmar possíveis mudanças que seriam necessárias caso a estrutura arquitetônica da aplicação fosse a monolítica em vez da estrutura de micro *frontends*.

E por fim, na sexta pergunta, é questionado quais os cenários que eles consideravam importantes quanto a modificação da aplicação. Após a coleta das respostas de todos os especialistas, foi realizada uma segunda interação, como

sugerida por Bergston *et al*. (2004), em que as respostas foram compartilhadas entre eles, de forma anônima, para que fosse eleito os cenários que melhor representassem uma possível modificação na aplicação.

4.3 APLICAÇÃO DO MÉTODO ALMA

4.3.1 Definição da Meta

A meta será escolhida com base no objetivo deste trabalho. O objetivo é avaliar a escolha atual da estrutura arquitetônica de micro *frontends*, comparando-a com a estrutura monolítica. Com isso, a meta escolhida é a "seleção de uma arquitetura de *software*", para que seja possível traçar conclusões se a escolha da estrutura foi de fato a mais adequada.

4.3.2 Descrição da Arquitetura

A estrutura arquitetônica atual, juntamente com a relação dos componentes, foi descrita na seção 4.1. A escolha estrutural de micro *frontends* para esse sistema foi uma solução natural visto que os editores DMN e BPMN fazem parte de outros projetos e são feitos em uma tecnologia que não é compatível com a tecnologia da *application shell*.

Como a meta escolhida é "seleção de uma arquitetura de *software*" é necessário pelo menos mais uma estrutura arquitetônica candidata para poder-se fazer a análise. Com isso em mente, a outra estrutura que será utilizada na comparação será a monolítica. Como foi trazido na Seção 2.2.1, esta estrutura é a escolha natural no início do desenvolvimento de aplicações, devido a sua simplicidade e baixo custo, e por esses motivos será escolhida para esta análise.

A aplicação não possui uma contraparte desenvolvida com a estrutura monolítica, sendo assim, é preciso entender quais são as mudanças necessárias para que esta aplicação seja feita nesta estrutura, para então ser possível realizar a avaliação dos cenários no quarto passo.

Como foi levantado na seção 4.2, a resposta da quinta pergunta do

questionário do Apêndice A trouxe o conhecimento dos especialistas sobre quais as mudanças necessárias para que a aplicação fosse monolítica. Algumas alterações foram propostas no enunciado, e que foram confirmadas pelos especialistas. São elas:

- A reescrita dos editores utilizando a mesma tecnologia da *application shell,* e;

- A não utilização de *iframes* para encapsular os editores.

A Tabela 2 mostra a relação dos especialistas e suas respectivas respostas com as alterações adicionais trazidas pelos mesmos. Há uma grande similaridade nas respostas entre os especialistas, sendo que todos trouxeram que a comunicação entre as partes seria simplificada. Isso se dá, por conta de não ser mais necessário ter uma comunicação via mensagens por um barramento de eventos, sendo este, um requisito da estrutura de micro *frontends*, como foi visto na seção 2.2.2.2. Em segundo, o isolamento das classes CSS seriam perdidas, pois os editores não seriam mais encapsulados por *iframes*. E por último, um dos especialistas trouxe que seria necessário ter uma cautela adicional para alinhar as dependências entre os times.

Tabela 2 - Relação de especialista e seu parecer sobre as modificações necessárias para ter uma estrutura monolítica para a mesma aplicação.

Especialista	Respostas da questão 5
Especialista 1	A comunicação entre as partes seria simplificada. Perda do isolamento das classes CSS. Alinhamento de dependências entre times.
Especialista 2	A comunicação entre as partes seria simplificada. Perda do isolamento das classes CSS.
Especialista 3	A comunicação entre as partes seria simplificada.
Especialista 4	A comunicação entre as partes seria simplificada. Perda do isolamento das classes CSS.

Fonte: elaborado pelo autor

Sumarizando, temos três alterações necessárias para termos uma aplicação monolítica no *KIE Sandbox*.

- Reescrita dos editores utilizando a mesma tecnologia da *application shell*, tendo assim, a aplicação como um todo, utilizando a mesma tecnologia para o seu desenvolvimento;

- Remoção do encapsulamento dos editores via *iframes* e como consequência a simplificação da comunicação com os editores e a perda do isolamento das classes CSS;

- Alinhamento de dependências na aplicação como um todo.

Todas as alterações trazidas pelos especialistas são detalhes de implementação, e não influenciam nos componentes e as suas relações, desta forma, a Figura 15 continua sendo válida para esta estrutura. Com isso, a análise dos cenários levará em conta a seção 2.2.1 na qual é feita a revisão sobre a estrutura monolítica e a opinião dos especialistas.

4.3.3 Listagem dos Cenários

Como levantado na seção 4.2, a resposta da sexta pergunta do questionário presente no Apêndice A trouxe os cenários que os especialistas julgaram serem importantes para a aplicação. Na Tabela 3 há a relação das perguntas e respostas da primeira interação.

Tabela 3 - Relação de especialista e seu parecer sobre os cenários relacionados a modificação da aplicação.

Especialistas	Cenário	Respostas da questão 6
Especialista 1	1	Foi descoberta uma nova vulnerabilidade crítica em uma das dependências da aplicação
	2	É encontrado um erro crítico em um editor, sendo necessário atualizá-lo o quanto antes.
	3	Reescrever uma parte da aplicação usando uma nova biblioteca ou framework.
Especialista 2	4	É preciso atualizar o sistema de implantação automático do projeto.
	5	Uma nova vulnerabilidade pública foi encontrada numa das dependências do framework de desenvolvimento
	6	Um dos editores possui uma nova funcionalidade que precisa ser implantada o quanto antes.
	7	Um novo desenvolvedor precisa ser ensinado sobre a topologia do projeto.
Especialista 3	8	Desenvolvedores de uma das equipes dos editores precisam ser re-alocados para outra parte do projeto
	9	Uma extensão de arquivo que possui um editor gráfico desenvolvido usando tecnologias diferentes da aplicação deve ser suportado
	10	É necessário ensinar um novo desenvolvedor sobre a estrutura do projeto.
Especialista 4	11	Uma dependência em comum entre os editores precisa ser atualizada para uma nova versão, com alterações que podem quebrar a aplicação
	12	Um dos editores apresenta uma falha em produção e precisa ser atualizado ou desabilitado.
	13	É necessário realizar um teste A-B com duas versões diferentes de um mesmo editor.

Fonte: elaborado pelo autor

É possível observar que alguns dos cenários propostos por diferentes especialistas se sobrepuseram. Esses cenários são:

- Cenário 1 e o Cenário 5;
- Cenário 2 e o Cenário 12, e;
- Cenário 7 e o Cenário 10.

Na segunda interação, os especialistas fizeram uma eleição na qual votaram nos quatro cenários que achavam mais relevantes, que podem ser observados na Tabela 4.

Tabela 4 - Segunda interação dos especialistas.

Especialistas	Respostas	Cenários escolhidos
Especialista 1	1	Cenário 1
	2	Cenário 2
	3	Cenário 3
	4	Cenário 9
Especialista 2	1	Cenário 1
	2	Cenário 2
	3	Cenário 3
	4	Cenário 4
Especialista 3	1	Cenário 1
	2	Cenário 2
	3	Cenário 3
	4	Cenário 9
Especialista 4	1	Cenário 2
	2	Cenário 3
	3	Cenário 9
	4	Cenário 11

Fonte: elaborado pelo autor

Sumarizando, os cenários mais votados, os quatro cenários podem ser observados na Tabela 5.

Tabela 5 - Cenários mais votados pelos especialistas.

Cenários	
1	Foi descoberta uma nova vulnerabilidade crítica em uma das dependências da aplicação
2	É encontrado um erro crítico em um editor, sendo necessário atualizá-lo o quanto antes.
3	Reescrever uma parte da aplicação usando uma nova biblioteca ou framework.
4	Uma extensão de arquivo que possui um editor gráfico desenvolvido usando tecnologias diferentes da aplicação deve ser suportado

Fonte: elaborado pelo autor

4.3.4 Avaliação dos Cenários

Para este trabalho, foi escolhido o segundo método descrito na seção 2.3.1.4 para a meta de “seleção de uma arquitetura de *software*”, com o ranqueamento das estruturas em uma escala de -1 a 1. Após isso ser definido, foi feita uma terceira interação com os especialistas, perguntando qual nota eles atribuem para cada cenário, sendo -1 para a estrutura que não suporta o cenário, 0 para a estrutura que tem um pior suporte, e 1 para a que possui um melhor suporte. Além disso, foi utilizado o referencial teórico para entender o impacto do cenário sobre cada uma das estruturas propostas.

4.3.4.1 Cenário 1

Caso se torne de conhecimento público uma nova vulnerabilidade crítica em uma das dependências externas da aplicação, será necessário atualizá-la, ou se não for possível, substituí-la. Após isso a aplicação terá que ser implantada novamente (LEWIS, 2022).

Como visto na seção 2.2.2.2 do referencial teórico, a estrutura de micro *frontends* apresenta uma vantagem nesse quesito. Caso a vulnerabilidade esteja presente em um dos micro *frontends*, a atualização e implantação não requer que a aplicação como um todo seja implantada. Da mesma forma, se a dependência estiver presente na *application shell*, apenas a *application shell* precisará ser atualizada e implantada. Por outro lado, na estrutura monolítica, a aplicação como um todo terá que ser reimplantada de qualquer modo. Dessa forma foi atribuído o valor 1 a estrutura de micro *frontends* e 0 para a estrutura monolítica. É possível observar na Tabela 6, que essa nota se confirma com a opinião da maioria dos especialistas, com exceção do especialista 3 que escolheu atribuir 1 para ambos os casos. Foi considerado 0 para a estrutura monolítica devido a majoritariedade das opiniões e devido ao embasamento teórico.

Tabela 6 - Opinião dos especialistas sobre o cenário 1

Cenário 1	Micro frontend	Monolito
Especialista 1	1	0
Especialista 2	1	0
Especialista 3	1	1
Especialista 4	1	0

Fonte: elaborado pelo autor

4.3.4.2 Cenário 2

Assim como no primeiro cenário, a estrutura de micro *frontends* é favorecida pela sua implantação independente. A implantação do monolito é mais onerosa, visto que o projeto inteiro precisa ser reimplantado. Dessa forma foi atribuído o valor 1 a estrutura de micro *frontends* e 0 para a estrutura monolítica. É possível observar na Tabela 7, que essa nota se confirma com a opinião dos especialistas.

Tabela 7 - Opinião dos especialistas sobre o cenário 2

Cenário 2	Micro frontend	Monolito
Especialista 1	1	0
Especialista 2	1	0
Especialista 3	1	0
Especialista 4	1	0

Fonte: elaborado pelo autor

4.3.4.3 Cenário 3

Neste caso, considerando que o barramento de eventos da estrutura de micro *frontends* já existe, adicionar uma nova parte a aplicação utilizando outro *framework* de desenvolvimento é trivial. No caso do monolito essa mudança não é possível, desta forma foi atribuído o valor -1, pois seria necessário reescrever a aplicação como um todo, e o valor 1 para a estrutura de micro *frontends*. É possível observar na Tabela 8, que essa nota se confirma com a opinião dos especialistas.

Tabela 8 - Opinião dos especialistas sobre o cenário 3

Cenário 3	Micro frontend	Monolito
Especialista 1	1	-1
Especialista 2	1	-1
Especialista 3	1	-1
Especialista 4	1	-1

Fonte: elaborado pelo autor

4.3.4.4 Cenário 4

Assim como no terceiro cenário, caso ocorra o requisito de inclusão de um novo editor na aplicação, a única mudança necessária na arquitetura de micro frontends é por parte do time responsável pelo novo editor, sendo a implementação de uma interface já definida. No caso do monólito o único modo desta inclusão ocorrer é caso o editor fosse feito na mesma tecnologia da aplicação, dessa forma, esse cenário, também não é suportado pela estrutura monolítica, com isso temos o valor de -1 para a estrutura monolítica e 1 para a estrutura de micro *frontends*. É possível observar na Tabela 9, que essa nota se confirma com a opinião dos especialistas.

Tabela 9 - Opinião dos especialistas sobre o cenário 4

Cenário 4	Micro frontend	Monolito
Especialista 1	1	-1
Especialista 2	1	-1
Especialista 3	1	-1
Especialista 4	1	-1

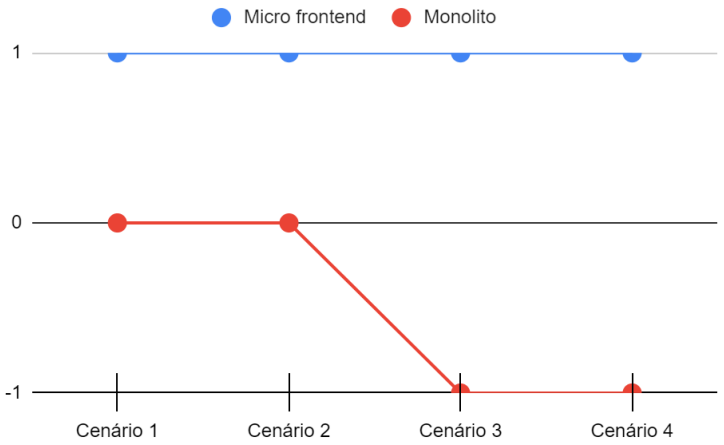
Fonte: elaborado pelo autor

4.3.5 Resultados da avaliação de cenários

De acordo com Bengtsson *et al*. (2004), a arquitetura que suporta a maioria

dos cenários é considerada a mais adequada. O gráfico da Figura 18 foi feito a fim de se fazer uma análise visual da etapa anterior.

Figura 18 - Página do editor da aplicação *KIE Sandbox* separada por componentes.



Fonte: elaborado pelo autor

Seguindo a escala proposta na avaliação, a estrutura monolítica teve um desempenho pior nos primeiro e segundo cenários, e não apresentou suporte ao terceiro e quarto cenários, dessa forma, não se sobressaiu em nenhum caso. Desta forma os resultados da avaliação mostram que a escolha estrutural de micro *frontends* é mais adequada para a aplicação *KIE Sandbox*.

Atualmente a estrutura da aplicação *KIE Sandbox* é a de micro *frontends*, sendo assim, foi confirmado que a sua escolha estrutural foi feita de forma apropriada com base nos cenários levantados pelos especialistas.

5 CONCLUSÃO

A análise do estado da arte detectou uma lacuna de estudos que abordaram a arquitetura de *micro frontends*. Essa lacuna pode ser oriundo da sua novidade, ou pela sua baixa adoção. Seja por um, ou por outro, este trabalho traz uma análise utilizando um método conhecido, a *Architecture-Level Modifiability Analysis*, mitigando possíveis dúvidas sobre a adoção da estrutura de micro *frontends*, e com isso podendo auxiliar desenvolvedores na escolha estrutural de suas aplicações.

O presente trabalho buscou realizar uma análise de uma aplicação real, chamada *KIE Sandbox* que utiliza a estrutura de micro *frontends*. Essa análise foi feita utilizando o método ALMA com a meta de seleção de uma arquitetura de software. A análise utilizou-se de duas possíveis estruturas arquitetônicas para uma mesma aplicação *web*, a monolítica e a de micro *frontends.* Como conclusão do levantamento bibliográfico contra a aplicabilidade profissional dada diante da vivência dos especialistas, foi observado que a escolha da estrutura arquitetônica de micro *frontends* na aplicação *KIE Sandbox* foi mais adequada do que a monolítica. Apesar desta aplicação funcionar totalmente no cliente, não impacta os resultados encontrados, visto que a comparação arquitetural é feita apenas no cliente.

Outra visibilidade que o estudo trouxe, foi que os pontos negativos da arquitetura de micro frontends que foram apresentados na revisão teórica, não foram confrontados, uma vez que essas pontuações não foram levantadas na consulta. Como por exemplo a complexidade, a comunicação e a experiência do usuário.

O método ALMA não se limita a seleção entre apenas duas arquiteturas, podendo ter sido feito um levantamento de uma quantidade maior de estruturas, tendo assim, uma análise mais rica. Isso possibilitaria uma resposta mais completa e talvez levando a uma resposta diferente do presente trabalho. É importante ressaltar também, que este trabalho utilizou a estrutura monolítica de forma genérica, sem analisar a fundo possíveis variações arquitetônicas para o desenvolvimento *frontend*. Este método se utiliza de cenários que foram descritos especificamente para a aplicação *KIE Sandbox*, sendo assim, os resultados encontrados devem ser usados com cautela na avaliação de outras aplicações, e além disto, os especialistas 1, 2 e 3 foram responsáveis pela escolha da estrutura arquitetônica atual da aplicação,

podendo assim, ter um viés na escolha dos cenários.

Outra ressalva importante, é referente ao número limitado de especialistas consultados, sendo este, devido a um universo pequeno de pessoas que poderiam responder às perguntas. Uma quantidade maior de respostas poderiam ter levado a resultados diferentes, assim como a diversificação de área dos mesmos, como por exemplo a consulta a um gerente. Somando-se a isso, a consulta não foi conduzida por um analista com experiência na aplicação do método ALMA, podendo não ter sido capaz de extrair os melhores cenários dos especialistas.

5.1 TRABALHOS FUTUROS

Para trabalhos futuros, outros métodos de análise arquitetural podem ser empregados nesta aplicação, para ser feita uma análise mais completa e de outros aspectos desta arquitetura.

Pode ser feito também a adição de outras estruturas arquitetônicas na aplicação do método, deixando assim a análise mais rica.

6 REFERÊNCIAS

BENGTSSON, Per Olof; LASSING, Nico; BOSCH, Jan; VLIET, Hans; Architecture-Level modifiability analysis (ALMA). The Journal of Systems and Software. vol 69. p129-147. 2004.

BETTS, Thomas; GIL, Blanca G. InfoQ, 2023, Software Architecture and Design InfoQ Trends Report - April 2023, Disponível em <https://www.infoq.com/articles/architecture-trends-2023/>. Acesso em: 28 de Maio de 2023.

CDW. **Research Hub**, 2022, Thick vs. Thin vs. Zero Clients: Which Model Is Right for You? When deploying devices, organizations must consider cost, IT management burdens and - above all - the user experience. Disponível em: <https://www.cdw.com/content/cdw/en/articles/hardware/thick-vs-thin-vs-zero-clients.html>. Acesso em: 25 de Maio de 2023.

DOYLE, Barry; LOPES, Cristina V. Survey of Technologies for Web Application Development. ACM Journal Name, vol 2. no 3. p1-43. 2005.

FOWLER, Martin. Who Needs an Architect? **IEEE Computer Society**, 2003. Disponível em: <https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf>. Acesso em: 20 de Maio de 2023.

FOWLER, Martin. **Martin Fowler**, 2014. BoundedContext, Disponível em <https://martinfowler.com/bliki/BoundedContext.html>. Acesso em: 10 de Junho de 2023.

FOWLER, Martin. **Martin Fowler**, 2019. Software Architecture Guide, Disponível em <https://martinfowler.com/architecture/>. Acesso em: 20 de Maio de 2023.

GEERS, Michael. Micro Frontends in Action. Manning Publication. 2020

GLOBAL CHANGE DATA LAB. Our World in Data, 2023. Number of People using the Internet, Disponível em: <https://ourworldindata.org/grapher/number-of-internet-users>. Acesso em: 08 de Maio de 2023.

GWT. **GWT Project**, 2023. Overview, Disponível em: <https://www.gwtproject.org/overview.html>. Acesso em: 06 de Junho de 2023.

JACKSON, Cam. **Martin Fowler,** 2019. Micro frontends. Disponível em: <https://martinfowler.com/articles/micro-frontends.html>. Acesso em: 24 de Maio de 2023.

LEWIS, James; FOWLER, Martin. **Martin Fowler**, 2014. Microservices, a definition of this new architectural term. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 10 de Junho de 2023.

LEWIS, Terry. CVE Vulnerabilities Found - What To Do Next. **RoboShadow**, 2022. Disponível em: <https://www.roboshadow.com/blog/cve-vulnerabilities-found-what-to-do-next>. Acesso em: 09, Junho, 2023

LINDLEY, Cody. **Front-end Developer Handbook 2019**, Frontend Masters, 2019 Disponível em <https://frontendmasters.com/guides/front-end-handbook/2019/>. Acesso em: 24 de Maio de 2023.

MÄNNISTÖ, Jouni; TUOVINEN, Antti-Pekka; RAATIKAINEN, Mikko. Experiences on a Frameworkless Micro-Frontend Architecture in a Small Organization. 2023 IEEE

20th International Conference on Software Architecture Companion (ICSA-C). 2023.

MEZZALIRA, Luca. Building Micro-Frontends: Scaling Teams and Project, Empowering Developers. O'Reilly. 2021.

MDN. **mdn web docs**, 2023a. <iframe>. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe>. Acesso em: 10 de Junho de 2023.

MDN. **mdn web docs**, 2023b. CustomEvent. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/CustomEvent>. Acesso em: 10 de Junho de 2023.

MDN. **mdn web docs**, 2023c. Window. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Window>. Acesso em: 10 de Junho de 2023.

MDN. **mdn web docs**, 2023d. Window.postMessage. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>. Acesso em: 10 de Junho de 2023.

MOIVA. **Moiva**, 2023, JavaScript Build Tools, Module Bundlers, Disponível em: <https://moiva.io/?npm=@parcel/core+esbuild+microbundle+rollup+vite+webpack>. Acesso em: 06 de Junho de 2023.

NEWMAN, Sam, Building Microservices: Designing Fine-Grained Systems. O'Reilly. Segunda Edição. 2021

PATIDAR, Anil; SUMAN, Ugrasen. A survey on software architecture evaluation methods. 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom). 2015

PELTONEN, Severi; MEZZALIRA, Luca; TAIBI, Divided. Motivations, benefits and issues for adopting Micro-Frontends: A Multivocal Literature Review. Information and Software Technology. vol 136. 2021.

REESE, George. Database Programming with JDBC and Java. Segunda Edição. O'Reilly. p. 126-145. 2000.

RICHARDS, Mark; FORD, Neal. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly. 2020.

RICHARDSON, Chris. Microservices Patterns: With examples in Java. Manning Publication. 2018.

STACKOVERFLOW, **StackOverflow**, 2022, Developer Survey, Disponível em: <https://survey.stackoverflow.co/2022/>. Acesso em: 03 de Junho de 2023.

STOERMER, Christoph; BACHMANN, Felix; VERHOEF, Chris. SACAM: The Software Architecture Comparison Analysis Method. Software Engineering Institute. 2003.

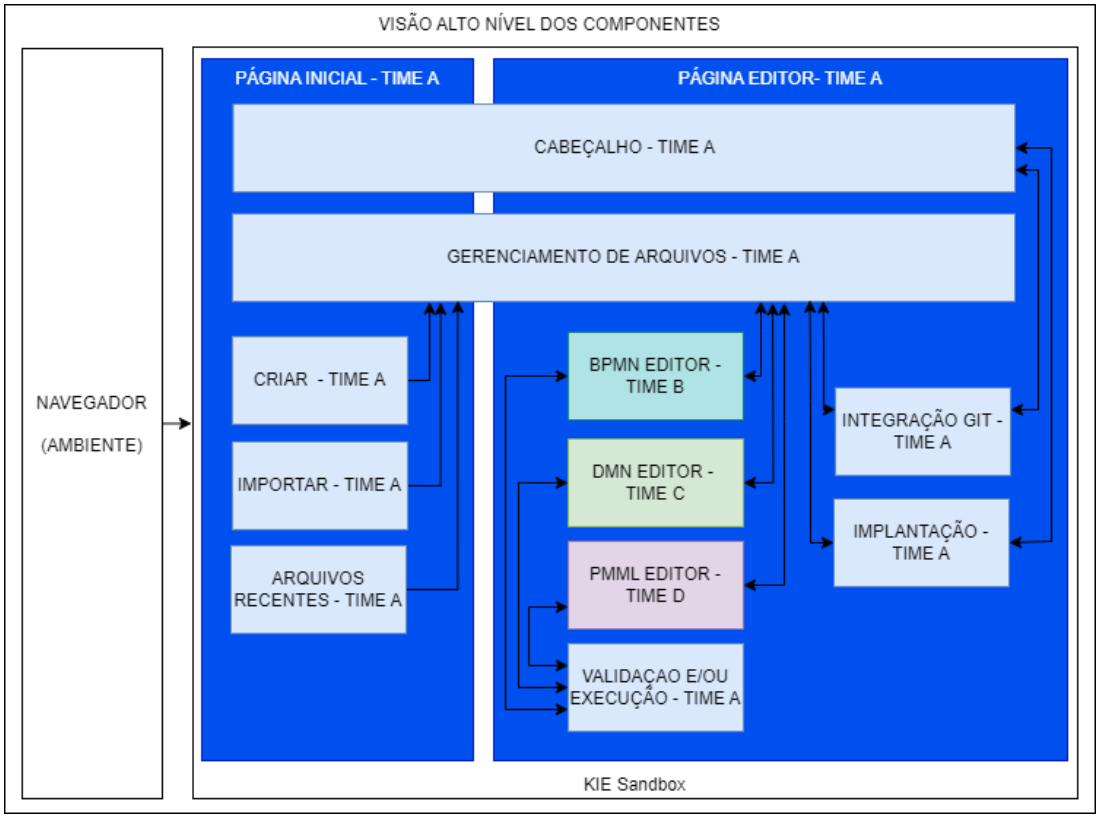
TAIVALSAARI, Antero; MIKKONEN, Tommi; INGALLS, Dan; PALACZ Krzysztof. Web Browser as an Application Platform. 34th Euromicro Conference Software Engineering and Advanced Applications. p293-302. 2008

APÊNDICE A - CONSULTA AOS ESPECIALISTAS

1 INTRODUÇÃO

A consulta a seguir será sobre o projeto o *KIE Sandbox*, um projeto open source que está disponível na plataforma *GitHub*[6]. Sumarizando, o projeto atual usa uma estratégia de micro *frontends* com *iframes*, carregando editores gráficos que foram feitos utilizando frameworks de desenvolvimento *web* diferentes da aplicação atual e/ou feitos por equipes diferentes. A Figura 1 ilustra a visão de alto nível dos componentes desta aplicação, mostrando os componentes, suas relações e a possível divisão de times.

Figura 1 - Visão alto nível dos componentes do da aplicação KIE Sandbox



Fonte: elaborada pelo autor

[6] https://github.com/kiegroup/kie-tools

2 *Architecture-Level Modifiability Analysis* (ALMA)

ALMA é um método de análise arquitetural proposto por Bengtsson *et al.* (2004) que utiliza cenários para verificar a modificação da arquitetura a ser analisada. Com este método, é possível realizar um estudo comparativo entre arquiteturas candidatas para uma mesma aplicação, mostrando assim, qual arquitetura responde melhor aos cenários levantados.

Um cenário é uma situação que pode vir a ocorrer com a aplicação. Os cenários propostos devem trazer situações reais que evidenciem a capacidade da aplicação de sofrer possível modificação. Em uma análise entre arquiteturas candidatas, quando possível, os cenários devem fortalecer a escolha de uma das arquiteturas, mas podem fortalecer ambas ou nenhuma. Possíveis cenários seriam: ocorreu uma atualização na API do navegador; um novo componente precisa ser adicionado na aplicação; entre outros.

Com a lista de cenários em mãos, é analisado o impacto destes cenários sob a arquitetura. No caso, é visto quais componentes de ambas as arquitetura serão afetados por estes cenários, de forma direta ou indireta.

Essa análise pode ser feita, pelos próprios especialistas, definindo assim, qual arquitetura tem um suporte melhor ao cenário proposto; ou para cada cenário é feito um ranqueamento do qual é visto qual arquitetura suporta melhor; ou é utilizada uma escala ordinária como por exemplo, linhas de código.

3 PERGUNTAS

1. Qual é o seu cargo atual?
2. A quanto tempo de experiência você possui neste cargo?
3. Qual o tempo de experiência na área?
4. Qual a sua formação?
5. Caso o *KIE Sandbox* fosse uma aplicação monolítica, isso é, se não fosse utilizado uma arquitetura de micro *frontends*, além dos editores terem que serem feitos na mesma tecnologia que a aplicação (React), seria necessário mais alguma alteração arquitetural?
6. Considerando a aplicação *KIE Sandbox*, quais os cenários relevantes que podem acontecer referente a sua modificação, que pode, ou não, ser facilitada pela escolha arquitetural da arquitetura monolítica ou da arquitetura de micro *frontends*? Por exemplo: Uma nova extensão de arquivo que possui um editor gráfico desenvolvido usando tecnlogias diferentes da aplicação deve ser suportado;

3.1 RESPOSTAS ESPECIALISTA 1

1. Cargo: Engenheiro de Software Especialista Sênior / Arquiteto de Software
2. Tempo de experiência no cargo: 2 anos
3. Tempo de experiencia na area: 15 anos
4. Formação: Mestre em Engenharia Elétrica e Bacharel em Ciência da Computação
5. A não adoção de uma arquitetura baseada em micro frontends no KIE Sandbox, acarretaria em uma série de mudanças arquitetônicas. Entre esta mudanças podemos destacar:

Alinhamento de dependências: na arquitetura de micro frontends, cada módulo pode gerir e evoluir suas dependências de forma independente, o que não acontece em arquiteturas monolíticas. Por exemplo no seguinte cenário:

- Editor A depende da biblioteca X 0.1, que contém a dependência transitiva Z na versão 0.1;
- Editor B depende da biblioteca Y 0.1, que contém a dependência transitiva Z na versão 0.1;
- Editor B deseja utilizar uma nova versão da biblioteca Y 0.2, contém a dependência transitiva Z na versão 0.2;

Se o time responsável pelo Editor B, deseja atualizar a biblioteca Y para versão 0.2 que teve um upgrade na sua dependência transitiva Z para versão 0.2, em uma arquitetura monolítica, o time responsável pelo Editor A, também precisará atualizar a sua dependência X, devido ao conflito entre diferentes versões de dependências Z.

Em muitos cenários, ainda não está disponível uma versão da biblioteca X que utilize a versão mais atualizada da Z (comum entre X e Y), bloqueando o upgrade do time responsável pelo editor B. Em uma arquitetura de micro frontends, cada time poderia realizar o upgrade em momentos distintos.

Isolamento de CSS: O uso de micro frontends e iframes possibilitam isolamento de estilo de forma natural, evitando que definições de classes em componentes independentes colidam em tempo de execução. Em uma arquitetura monolítica, para realizar um isolamento de CSS efetivo, teríamos que empregar técnicas como: pré processadores de CSS, namespacing, CSS Modules, Styled Componentes ou Shadow DOM.

Comunicação entre componentes: a comunicação entre componentes isolados através de um iframe é mais complexa e menos eficiente, pois depende do uso de *postMessage* para troca de dados entre os componentes. Uma arquitetura monolítica tornaria esta comunicação mais simples, eficiente e aderente a bibliotecas como React.

6.

a. Cenário: Se tornou de conhecimento público uma nova vulnerabilidade crítica em uma das dependências da aplicação.

Como falado na questão anterior, a independência de gerenciamento de dependências em arquiteturas micro frontend, torna mais simples a atualização das dependências de cada micro frontend, seja em relação a novas versões quanto a uma resposta rápida a vulnerabilidades públicas.

b. Cenário: É encontrado um erro crítico na aplicação, sendo necessário atualizá-lo o quanto antes.

   A aplicação em produção de um patch de correção em um micro frontend pode ser aplicada de forma isolada em um único micro frontend, não requerendo o rebuild de toda a aplicação. Isto torna o processo muito mais eficiente.

c. Cenário: Reescrever uma parte da aplicação usando uma nova biblioteca ou framework.

   Devido ao isolamento de escopo de cada micro frontend, a exploração e adoção de novas tecnologias e frameworks é simplificada em relação a arquiteturas monolíticas. Devido a adoção de iframes, bibliotecas incompatíveis podem coexistir na mesma aplicação.

3.2 RESPOSTAS ESPECIALISTA 2

1. Cargo: Arquiteto de Software
2. Tempo de experiência no cargo: 2 anos
3. Tempo de experiencia na area: 10 anos
4. Formação: Bacharel em Ciência da Computação
5. Sim. Todas as interfaces de comunicação entre os Editores e o KIE Sandbox teriam que ser reformuladas, para que respeitassem o modelo de programação da tecnologia escolhida, no caso, React. Ter os Editores “envelopados” por iframes criou a necessidade de desenvolvermos soluções customizadas para que essa comunicação fosse transparente, utilizando de abstrações como Promises e RPC, porém, dada a natureza das APIs disponíveis para interação entre múltiplas “windows” em uma página do browser, não é possível usar o modelo de programação declarativo, como dita o React. A utilização de iframes também faz um isolamento natural do estilo das partes, permitindo que cada parte tenha suas próprias classes CSS. Ao fazermos o KIE Sandbox ser uma aplicação monolítica, estaríamos ditando toda a arquitetura dos editores.
6.
    a. É preciso atualizar o sistema de implantação automático do projeto.
    b. Uma nova vulnerabilidade pública foi encontrada numa das dependências do framework de desenvolvimento do editor.
    c. Um dos editores possui uma nova funcionalidade que precisa ser implantada o quanto antes.
    d. Um novo desenvolvedor precisa ser ensinado sobre o projeto.

3.3 RESPOSTAS ESPECIALISTA 3

1. Cargo: Engenheiro de Software Especialista
2. Tempo de experiência no cargo: 1 anos
3. Tempo de experiencia na area: 13 anos
4. Formação: Bacharel em Ciência da Computação
5. Seriam necessários alguns ajustes na maneira como a comunicação entre os componentes em diferentes windows (main window e iframes) é feita, para que as mensagens enviadas e recebidas fossem tratadas apenas para o componente de destino, mesmo com vários componentes residindo na mesma window. Algumas simplificações também poderiam ser feitas, já que várias abstrações relacionadas a essa troca de mensagens passariam a ser desnecessariamente complexas, e poderiam ser trocadas por parâmetros e callbacks mais simples.
6.
    a. Facilitada por monolítica: Desenvolvedores de uma das equipes dos editores precisam ser re-alocados para outra parte do projeto.
    b. Facilitada por micro frontends: Uma extensão de arquivo que possui um editor gráfico desenvolvido usando tecnologias diferentes da aplicação deve ser suportado.
    c. Facilitada por monolítica: Um desenvolvedor que acabou de entrar na equipe precisa ser ensinado sobre as tecnologias e conceitos da arquitetura do projeto.

3.4 RESPOSTAS ESPECIALISTA 4

1. Cargo: Engenheiro de Software
2. Tempo de experiência no cargo: 6 anos
3. Tempo de experiencia na area: 8 anos
4. Formação: Bacharel em Ciência da Computação
5. O principal fator que precisaria ser repensado é a "comunicação" ou transmissão de dados entre os editores e a aplicação. No caso de iFrames pode-se assumir que existe uma API construída em cima da Web API da window (utilizando o método postMessage por exemplo, enviando dados serializados de forma assíncrona e bi-direcional). Ao trazer os editores para dentro de uma aplicação monolítica utilizando a mesma tecnologia (React, no caso), utilizar a window para enviar mensagens serializadas deixa de fazer sentido, pois passa a ser possível utilizar as próprias APIs de gerenciamento de estado e manipulação do DOM que o React oferece. Outro ponto importante a ser considerado é que agora os editores não funcionam em ambientes "isolados" da aplicação, fazem parte da mesma árvore do DOM, de forma que passam a compartilhar o CSS e event listeners com a aplicação, talvez sendo necessário uma reestruturação das classes de HTML utilizadas para evitar o "vazamento" dos estilos de um para o outro.
6. .
    a. Uma dependência em comum entre os editores precisa ser atualizada para uma versão nova, com alterações que podem quebrar a aplicação. Em uma aplicação monolítica bastaria uma única modificação (seguida de adaptações e testes, claro), enquanto que com micro frontends seria necessário atualizar os micro frontends um a um, testando a compatibilidade entre eles.
    b. Um dos editores apresenta uma falha no ambiente de produção e precisa ser atualizado ou desabilitado. No caso de micro frontends, bastaria uma atualização no micro frontend deste editor, republicando a versão atualizada. Com um monolito seria necessário uma atualização na aplicação inteira.

c. É necessário realizar um teste A-B com duas versões diferentes de um mesmo editor. Com micro frontends esse processo tende a ser mais fácil, pois é possível publicar várias versões de um mesmo micro frontend e a aplicação escolhe se deve utilizar a versão A ou B para certo usuário. Com monólitos isso também é possível, porém ambas as implementações A e B devem estar disponíveis na mesma versão publicada, gerando um código mais complexo e propenso a erros.

4 REFERÊNCIAS

BENGTSSON, Per Olof; LASSING, Nico; BOSCH, Jan; VLIET, Hans; Architecture-Level modifiability analysis (ALMA). The Journal of Systems and Software. vol 69. p129-147. 2004.

APÊNDICE B - ARTIGO NO FORMATO SB

Uma análise da arquitetura de micro frontends em uma aplicação real

Luiz João Carvalhaes Motta

Departamento de Informática e Estatística (INE) – Universidade Federal de Santa Catarina (UFSC) Caixa Postal: 476 - CEP: 88040370 – Florianópolis – SC – Brasil

luiz.motta@ufsc.br

Abstract. *One of the most significant parts of software development is choosing its architecture. This choice will dictate the entire relationship between teams and how those teams will deliver their requirements. This choice is often neglected or made incorrectly, impacting the development of the project in the long term. This study brings an analysis of a web application that uses the architectural structure of micro frontends. This analysis is based on a comparison with the traditional monolithic architecture. The analysis uses the Architecture-level Modifiability Analysis (ALMA) method, in which scenarios are used to understand which architecture has the most appropriate characteristics for the application requirements. This article presents a review of what software architecture is, a review of monolithic and micro frontend architectures, the application of the ALMA method in a web application, and its results.*

Resumo. *Uma das partes mais importantes no desenvolvimento do software é a escolha da sua arquitetura. Esta escolha irá ditar todo o relacionamento entre equipes e como essas equipes irão entregar seus requisitos. Muitas vezes essa escolha é negligenciada ou é feita de forma errônea, impactando o desenvolvimento do projeto a longo prazo. Este estudo traz uma análise de uma aplicação web que se utiliza da estrutura arquitetônica de micro frontends. Esta análise é feita a partir de uma comparação com a arquitetura tradicional monolítica. A análise utiliza o método Architecture-level Modifiability Analys (ALMA), no qual se utiliza de cenários para entender qual arquitetura possui as características mais apropriadas para os requisitos da aplicação. Este artigo apresenta uma revisão sobre o que é arquitetura de software, uma revisão sobre as arquiteturas monolítica e de micro frontends, a aplicação do método ALMA em uma aplicação web e seus resultados.*

1. Introdução

Um dos estilos arquitetônicos do tipo distribuído que ganhou força nos últimos anos é a arquitetura de microsserviços [Richards e Ford 2020]. Richardson (2018) explica que a

ideia principal deste estilo é quebrar uma arquitetura monolítica em partes menores, chamando cada uma destas partes de microsserviço. Cada um desses microsserviços são mantidos por um pequeno time, que tem um domínio único e uma implantação independente. Esse estilo arquitetônico não prescreve um modelo para a camada de aplicação, sendo assim, ainda é utilizado uma abordagem monolítica para esta parte [Peltonen et al. 2021].

Mezzalira (2021) traz que a origem da arquitetura de micro frontends surgiu pela demanda de estrutura arquitetônica no cliente que acompanhasse a arquitetura de microsserviços, e com isso uma solução similar foi criada. Assim como a arquitetura de microsserviços, ela separa um grande cliente que é desenvolvido por múltiplos times em clientes menores mantidos por pequenos times, tendo em vista um propósito único e com uma implantação independente.

Muito se fala sobre os benefícios e malefícios da arquitetura de micro frontends na literatura [Peltonen et al. 2021; Geers 2020]. A fim de averiguar o uso desta estrutura, este trabalho traz uma análise sobre uma aplicação real que utiliza esta estrutura arquitetural, evidenciando os seus requisitos e por fim, trazendo se a escolha estrutural foi, ou não, adequada. Este trabalho trouxe a estrutura monolítica a fim de performar uma análise comparativa utilizando o método Architecture-Level Modifiability Analysis (ALMA).

2. Micro frontends

O termo micro frontend é relativamente recente sendo trazido pela primeira vez em 2016 (Jackson 2019) e até a escrita do presente artigo, ainda não existem muitos estudos científicos nesta área. Peltonen et al. (2021) traz que o micro frontend estende os conceitos de microsserviços para o desenvolvimento frontend, compartilhando seus princípios, benefícios e problemas. Mezzalira (2021) divide o desenvolvimento de micro frontends em algumas partes fundamentais. Dentre elas temos a definição da divisão, que pode ser horizontal, com múltiplos micro frontends numa mesma página, ou vertical, com um micro frontend por página. Temos também como o micro frontend é montado, tendo dois modos mais comuns: no cliente ou no servidor. Por fim, é importante entender como será feita a comunicação entre as partes, podendo ser através de um barramento de eventos, de custom events ou do armazenamento do browser.

3. Contextualização do KIE Sandbox

A aplicação KIE Sandbox é uma aplicação web open source focada em automação de negócios, e tem como público alvo analistas de negócio e desenvolvedores. Essa aplicação permite ao usuário criar, importar e editar graficamente arquivos Business Process Model And Notation (BPMN), Decision Model and Notation (DMN) e Product Model Markup Language (PMML), possui um gerenciamento de arquivos, integração com um sistema de versionamento Git, um sistema para realizar a implantação em um cluster Kubernetes e um sistema para validar os diagramas e executar arquivos DMN.

O projeto como um todo foi feito utilizando um framework de desenvolvimento web chamado de React.js e a ferramenta Webpack para gerar os arquivos HTML, CSS e JS. Ele funciona inteiramente no cliente, e se utiliza da arquitetura de micro frontends para a composição da página dos editores, possibilitando assim, a utilização de editores gráficos desenvolvidos com a utilização de outros frameworks. Os editores foram feitos tanto utilizando o framework React.js quanto em um framework chamado de Google

Web Toolkit (GWT), sendo essa a maior razão para a adoção de micro frontends.

A provável divisão de times, juntamente com a visão de alto nível dos componentes, e a suas relações podem ser observadas na Figura 1.

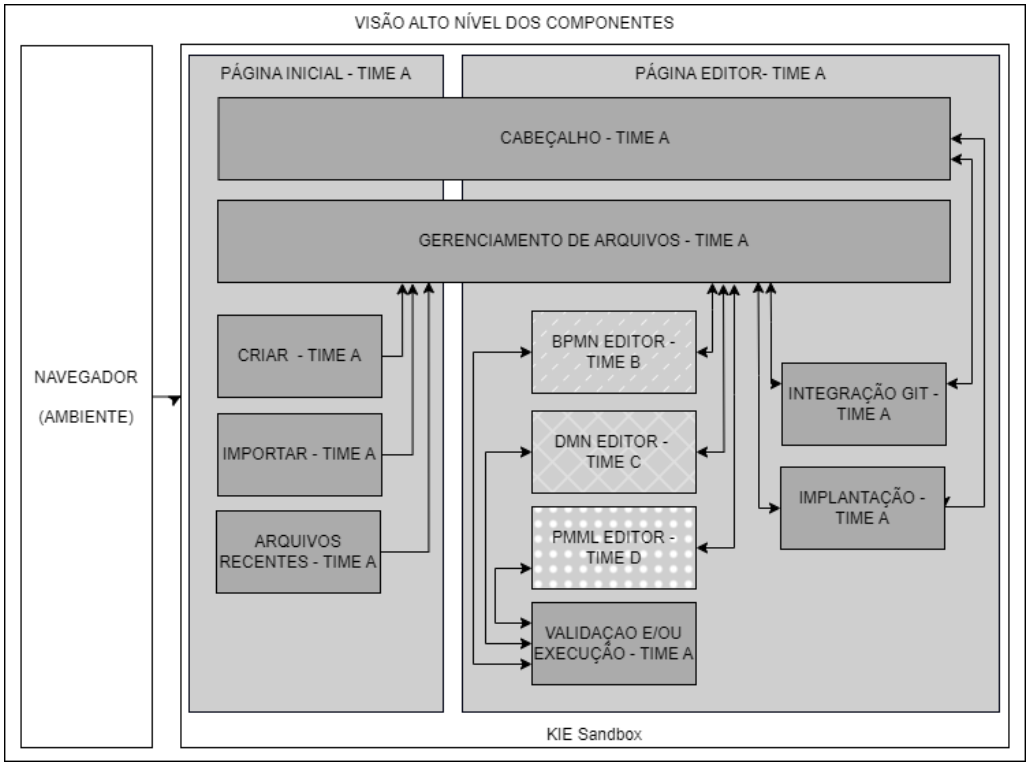


Figura 1. Visão de alto nível dos componentes

Há uma divisão horizontal na página do editor, esta que funciona como uma application shell, onde os editores são montados utilizando iframes. Sendo assim, cada editor deve disponibilizar um arquivo para carregá-lo neste iframe. Para a comunicação das partes, foi desenvolvido um framework utilizando a linguagem TypeScript, no qual é criado interfaces, tanto para os editores quanto para a application shell, no qual ambos subscrevem e há uma comunicação por mensagens, sendo essa comunicação bi-direcional. Caso um novo editor gráfico tenha que ser adicionado à aplicação o único requisito é que o editor implemente a interface de comunicação. É possível também, adicionar novos micro frontends, desde que a interface de ambas as partes sejam implementadas.

4. Análise arquitetural

Como descrito na contextualização da aplicação KIE Sandbox, a escolha da arquitetura de micro frontends foi natural, devido a natureza da aplicação em comportar editores gráficos que foram implementados utilizando outras ferramentas de desenvolvimento. Para analisar se esta foi a escolha adequada com relação aos requisitos da aplicação foi feita uma análise arquitetural utilizando a metodologia Architecture Level Modifiability Analysis (ALMA) [Bengtsson et al. 2004]. Esse método de avaliação foi escolhido por possuir a capacidade de determinar qual arquitetura é mais adequada para um sistema e por exemplificar as técnicas utilizadas em cada etapa de modo a permitir que o autor

deste trabalho consiga aplicá-la, mesmo sem possuir experiências prévias em análises de arquitetura de software. Esse método de análise é dividido em cinco etapas: definição da meta, descrição da arquitetura, listagem dos cenários, avaliação dos cenários e resultados.

4.1 Definição da meta

A meta será escolhida com base no objetivo deste trabalho. O objetivo é avaliar a escolha atual da estrutura arquitetônica de micro frontends. Isso será alcançado a partir da comparação com a estrutura tradicional monolítica. Com isso, a meta escolhida é a "seleção de uma arquitetura de software", para que seja possível traçar conclusões se a escolha da estrutura foi de fato a mais adequada.

4.2 Descrição da Arquitetura

A arquitetura atual da aplicação foi descrita na sua contextualização, sendo assim, ainda é necessário traçar a descrição da arquitetura desta aplicação na sua forma monolítica. Para isso foi requisitado que membros do time de desenvolvimento da aplicação respondessem quais mudanças seriam necessárias para termos uma versão monolítica da aplicação. Os membros que auxiliaram nesta pesquisa foram dois arquitetos de software, um engenheiro de software especialista e um engenheiro de software pleno. Dentre as mudanças levantadas tivemos:

1. A comunicação entre as partes seria simplificada.
2. Perda do isolamento das classes CSS.
3. Alinhamento de dependências entre times.

Essas mudanças não afetam a visão de alto nível dos componentes, mas sim os detalhes de implementação, dessa forma, para a comparação será utilizada a teoria por trás de micro frontends e da arquitetura monolítica, e a opinião do time.

4.3 Listagem dos Cenários

A listagem de cenários foi realizada através de uma pesquisa aos mesmos membros da equipe da etapa anterior. A pesquisa teve o intuito de coletar os cenários que são considerados importantes para a aplicação KIE Sandbox. Nesta pesquisa foram coletados 13 cenários, e após isso, foi feita uma segunda interação com as mesmas pessoas, na qual foram compartilhados os cenários entre elas, para que fosse eleito os 4 cenários mais importantes. Os 4 cenários que foram considerados mais importantes para a aplicação KIE Sandbox foram:

1. Foi descoberta uma nova vulnerabilidade crítica em uma das dependências da aplicação.
2. É encontrado um erro crítico em um editor, sendo necessário atualizá-lo o quanto antes.
3. Reescrever uma parte da aplicação usando uma nova biblioteca ou framework.
4. Uma extensão de arquivo que já possui um editor gráfico, que foi desenvolvido usando tecnologias diferentes da aplicação deve ser suportado.

4.4 Avaliação dos Cenários

Para avaliar os cenários foi utilizado umas das técnicas sugeridas por Begtsson et al.

(2004), na qual é feito um ranqueamento das estruturas que suportam melhor cada cenário. Esse ranqueamento foi feito de -1 a 1, sendo -1 para a arquitetura que não suporta o cenário, 0 para a arquitetura que tem o pior suporte ao cenário e 1 para a arquitetura que tem o melhor suporte ao cenário. Esse ranqueamento foi feito com base na teoria por trás da arquitetura de micro frontends e da arquitetura monolítica, juntamente com a opinião dos entrevistados.

Todos os cenários tiveram uma ranque 1 para a arquitetura de micro frontends. Para a avaliação da arquitetura monolítica, o primeiro e segundo cenário tiveram um ranque 0, por se saíram pior que a arquitetura de micro frontends, e -1 para terceiro e quarto cenário por não possuírem suporte.

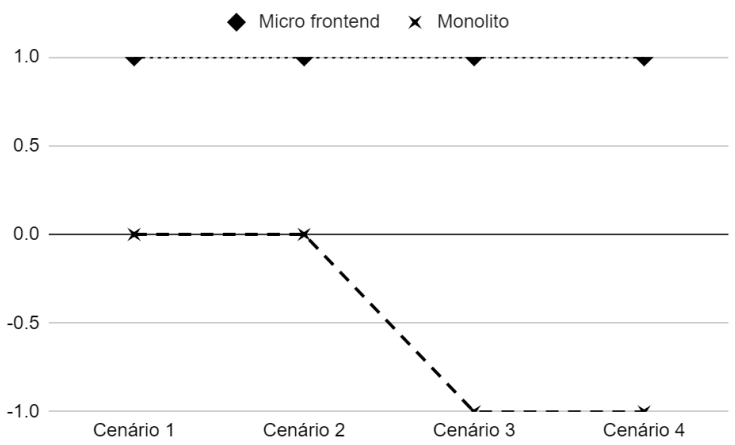


Figura 2. Avaliação dos cenários

4.5 Resultados da avaliação de cenários

De acordo com Bengtsson et al. (2004), a arquitetura que suporta a maioria dos cenários é considerada a mais adequada. Desta forma os resultados da avaliação mostram que a escolha estrutural de micro frontends é mais adequada para a aplicação KIE Sandbox.

5. Conclusão

O presente trabalho buscou realizar uma análise de uma aplicação real, chamada KIE Sandbox que utiliza a estrutura de micro frontends. Essa análise foi feita utilizando o método ALMA com a meta de seleção de uma arquitetura de software. A análise utilizou-se de duas possíveis estruturas arquitetônicas para uma mesma aplicação web, a monolítica e a de micro frontends. Como conclusão foi observado que a escolha da estrutura arquitetônica de micro frontends na aplicação KIE Sandbox foi mais adequada do que a monolítica. Apesar desta aplicação funcionar totalmente no cliente, não impacta os resultados encontrados, visto que a comparação arquitetural é feita apenas no cliente.

Outra visibilidade que o estudo trouxe, foi que os pontos negativos da arquitetura de micro frontends que foram apresentados na revisão teórica, não foram confrontados, uma vez que essas pontuações não foram levantadas na consulta. Como por exemplo a complexidade, a comunicação e a experiência do usuário.

O método ALMA não se limita a seleção entre apenas duas arquiteturas, podendo ter sido feito um levantamento de uma quantidade maior de estruturas, tendo assim, uma análise mais rica. Isso possibilitaria uma resposta mais completa e talvez levando a uma resposta diferente do presente trabalho. É importante ressaltar também, que este trabalho utilizou a estrutura monolítica de forma genérica, sem analisar a fundo possíveis variações arquitetônicas para o desenvolvimento frontend. Este método se utiliza de cenários que foram descritos especificamente para a aplicação KIE Sandbox, sendo assim, os resultados encontrados devem ser usados com cautela na avaliação de outras aplicações, e além disso, três dos entrevistados foram responsáveis pela escolha da estrutura arquitetônica atual da aplicação, podendo assim, ter um viés na escolha dos cenários.

Outra ressalva importante, é referente ao número limitado de especialistas consultados, sendo este, devido a um universo pequeno de pessoas que poderiam responder às perguntas. Uma quantidade maior de respostas poderiam ter levado a resultados diferentes, assim como a diversificação de área dos mesmos, como por exemplo a consulta a um gerente. Somando-se a isso, a consulta não foi conduzida por um analista com experiência na aplicação do método ALMA, podendo não ter sido capaz de extrair os melhores cenários dos especialistas.

Referências

BENGTSSON, Per Olof; LASSING, Nico; BOSCH, Jan; VLIET, Hans; Architecture-Level modifiability analysis (ALMA). The Journal of Systems and Software. vol 69. p129-147. 2004.

GEERS, Michael. Micro Frontends in Action. Manning Publication. 2020

JACKSON, Cam. Martin Fowler, 2019. Micro frontends. Disponível em: <https://martinfowler.com/articles/micro-frontends.html>. Acesso em: 24 de Maio de 2023.

MEZZALIRA, Luca. Building Micro-Frontends: Scaling Teams and Project, Empowering Developers. O'Reilly. 2021.

PELTONEN, Severi; MEZZALIRA, Luca; TAIBI, Divided. Motivations, benefits and issues for adopting Micro-Frontends: A Multivocal Literature Review. Information and Software Technology. vol 136. 2021.