INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Reengenharia de Arquitetura Monolítica para Arquitetura Micro Frontend

FREDERICO RENATO TAIPA COELHO DE SOUSA
Outubro de 2021

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Reengenharia de Arquitetura Monolítica para Arquitetura Micro Frontend

Frederico Sousa

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Nuno Silva

Coorientador: Rui Nascimento

Porto, Outubro, 2021

Dedicatória

Dedico esta dissertação a quem nunca acreditou em mim.

Aos restantes dedico a minha vida.

Resumo

Este projeto tem como objetivo a reengenharia, para a BMW AG, de uma aplicação web focada na monitorização de tráfego no âmbito da condução autónoma, adotando uma arquitetura baseada em Micro Frontend.

As aplicações web têm vindo a sofrer grandes mudanças a todos os níveis do seu desenvolvimento. Com a evolução da tecnologia e o aumento da complexidade que este tipo de aplicações é capaz de suportar, uma arquitetura capaz de se adaptar a um mercado constantemente em evolução é um ponto crucial para um projeto bem-sucedido.

Devido ao aumento significativo da dificuldade de manutenção que a arquitetura monolítica apresenta a longo prazo, muitas das organizações têm optado pela divisão dos seus projetos em partes de tamanho reduzido.

Desta segregação do espectro das funcionalidades em projetos mais reduzidos, introduzida inicialmente no backend e mais recentemente ao nível do frontend, surgem novas possibilidades de organização dos projetos que visam uma possível otimização das soluções.

O conceito de Micro Frontend assoma desta necessidade de divisão e propõe uma nova arquitetura muito similar aos já pré-existente micro serviços adotados em backend. Promovendo baixo acoplamento, aumento da coesão, redução do espectro de possíveis problemas e tornam mais fácil a manutenção.

Arquitetura de Micro Frontends é ainda bastante recente, pelo que a comunidade ainda não está munida de informação suficiente que permita prever o seu impacto em todo o tipo de projetos. Cada projeto deve procurar a solução arquitetural que mais se adapta ao seu negócio e restrições tecnológicas, desta forma obtendo mais benefícios e sofrendo menos prejuízos.

Assim, nesta situação em concreto, deve ser encontrada uma solução arquitetural baseada em Micro Frontend que permita dar resposta aos requisitos de negócio, e ao mesmo tempo que seja de fácil manutenção.

Palavras-chave: Micro Frontend, desenvolvimento web, monólito, arquitetura, necessidades de negócio, capacidades das soluções.

Abstract

This project aims to create, for BMW AG, a web application focused on traffic monitorization in the scope of autonomous driving, adopting an architecture based on Micro Frontend.
Currently, web applications have undergone major changes at all levels of their development. With the evolution of technology and the increasing complexity that this type of application is able to withstand, an architecture capable of adapting to a constantly evolving market is a crucial point for a successful project.
Due to the significant increase in maintenance difficulty and scalability needs that monolithic architecture presents in the long run, many organizations have opted to divide their projects into small parts.
From this segregation of the spectrum of functionalities in smaller projects, initially introduced in the backend and more recently at the frontend level, new possibilities arise for the organization of projects, aimed at a possible optimization of solutions.
The concept of Micro Frontend adds to this need for division and proposes a new architecture, very similar to the already pre-existing micro services adopted in Backend.
This promotes low coupling, increased cohesion, reduce the spectrum of possible problems and make maintenance easier.
Micro Frontends architecture is still quite recent, so the community is not yet provided with enough information to predict its impact on all types of projects. Each project should look for the architectural solution, within Micro Frontend, that most adapts to your business and techological constraints, thus obtaining more benefits and suffering less losses.

Thus, in this specific situation, an architectural solution based on Micro Frontend must be found that makes the solution more flexible, scalable and easy to maintain, always responding to customer needs.

Keywords: Micro Frontend, web development, monolith, architecture, business needs, solution capabilities.

Agradecimentos

A todos os que leram esta dissertação, alguns mais que uma vez, para me guiarem no caminho certo, o meu eterno obrigado.

Um agradecimento especial para a minha Mãe, Irmã e Namorada pela paciência ao longo dos anos.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Código

Acrónimos e Símbolos

Lista de Acrónimos

API	*Application Programming Interface*
AWS	*Amazon Web Services*
BMW	*Bayerische Motoren Werke*
DOM	*Document Object Model*
ESM	*ES Modules*
HCL	*Hashicorp Language*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IaC	*Infrastructure as Code*
MFEA	*Micro FrontEnd Architecture*
MPA	*Multiple Page Application*
MVC	*Model View Controller*
SPA	*Single Page Application*
PoC	*Proof of Concept*
UI	*User Interface*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*

1 Introdução

Esta parte do documento serve para introduzir o contexto, problema e as maiores forças motivadoras deste projeto.

Serão também apresentados os objetivos assim como a abordagem seguida e o planeamento do projeto.

Finalmente, apresenta-se a estrutura do documento de forma a facilitar a consulta do mesmo por parte do leitor.

1.1 Contexto

Desde a invenção da World Wide Web (WWW) por Tim Berners Lee em 1989, da definição das três primeiras tecnologias fundamentais da web de hoje (HTML, URI, HTTP) em 1991 e a sua disponibilização ao público geral em 1993, que a sua expansão ao longo dos anos e o número de utilizadores tem vindo a aumentar drasticamente [1].

Tabela 1 - Uso da Internet no mundo [2]

World regions	Population	Population % of the world	Internet users - 31 dec 2020	Penetration rate	Growth 2000-2020	Internet world %
Africa	1,357,198,684	17.3%	633,8856,924	46.7%	13.941%	12.8%
Asia	4,309,503,789	55%	2,563,603,922	59.5%	2.143%	51.8%
Europe	835,700,837	10.7%	727,848,547	87.1%	593%	14.7%
Latin america/ Caribbean	658,382,700	8.4%	477,824,732	72.6%	2.545%	9.7%
Middle east	263,933,993	3.4%	184,856,813	70.0%	5.528%	3.7%
North america	370,146,066	4.7%	332,910,868	89.9%	208%	6.7%
Oceania/Australia	43,138,089	0.6%	29,066,532	67.4%	281%	0.6%
World total	7,838,004,158	100.0%	4,949,868,338	63.2%	1.271%	100.0%

Em 2008 existiam mais de 1 trilião (1E18) de páginas web públicas e 1,7 biliões (1,7E14) de pessoas já usavam a WWW em 2009 [1]. Em 2020 estimava-se que 63,2% das pessoas do planeta usavam a WWW, o que corresponde a 4,9 mil milhões de pessoas (Tabela 1 – Internet Users 31 dec 2020) [2].

Este número mostra que o mundo mudou desde o lançamento da mesma, denotado diariamente na partilha de dados/informação, educação, comércio, troca de ideias e a saúde [3].

Através destes números é fácil perceber a rápida expansão do uso da World Wide Web e, consequentemente, das necessidades de desenvolvimento de software web. Este desenvolvimento visa a criação de páginas web que cumpram requisitos de um determinado cliente e/ou problema de negócio [4].

Normalmente estas páginas dividem-se em duas partes distintas de codificação:

- Cliente, ou frontend, é a parte do sistema que é usada pelos utilizadores humanos nas suas tarefas de negócio, e daí responsável pelos aspetos visuais e de interação com o utilizador. Esta parte é um cliente (i.e. consome dados e funcionalidades) da parte do servidor;
- Servidor, ou backend, é a parte do sistema que responde aos pedidos do cliente (funcionalidades) com dados de resposta, garantindo a lógica de negócio, processamento de dados e armazenamento.

O software desenvolvido deve cumprir tanto os requisitos funcionais como não funcionais. A definição de uma arquitetura que seja capaz de se adaptar a eventuais necessidades, assim como a aquisição de conhecimento e competências das tecnologias e metodologias a utilizar pode garantir o sucesso futuro da estabilidade da aplicação.

'Para muitos negócios, o frontend é a porta de entrada de muitos dos utilizadores, pelo que se torna natural ter sobre ele uma grande atenção na fase de desenvolvimento' [5].

As aplicações web construídas atualmente têm em conta as espectativas dos utilizadores, que podem ser variadíssimas, desde o nível de desempenho até à necessidade de correrem numa ampla gama de dispositivos, como computadores de secretária, dispositivos móveis em diferentes contextos físicos e utilizações diversas.

Este projeto, começado por outra empresa e equipa, enquadra-se no âmbito da condução autónoma com o objetivo de dotar os veículos da informação necessária para se deslocarem autonomamente. Mais concretamente, na gestão, manutenção e monitorização de dados geográficos colecionados e processados. Estes dados, fornecidos por sistemas terceiros e implantados na nuvem, complementam o sistema e são peças vitais para as várias decisões que o veículo tem de tomar. Este software será utilizado por monitorizadores de tráfego da BMW, espalhados pelo mundo, permitindo a gestão, manutenção e monitorização de dados referida anteriormente.

1.2 Problema

Foi-nos entregue um monólito de frontend, alimentado por (i) uma aplicação servidora (gerido pela mesma empresa, mas por outra equipa) e (ii) outros serviços terceiros fornecedores de dados e funcionalidades.

A aplicação frontend original necessita de melhorias, tendo sido sistematizadas nos seguintes requisitos[1]:

- Adição de novas funcionalidades;
- Melhoria da performance das funcionalidades pré-existentes;
- Maior acessibilidade das funcionalidades;
- Interligação de funcionalidades e o seu desempenho;
- Diferentes funcionalidades por país;
- Criação de testes;
- Mudança do design gráfico da aplicação.

Estas tarefas implicam uma compreensão completa da solução atual de forma a ser possível readaptá-la para uma solução capaz de suportar a informação e funcionalidades necessárias de forma eficaz e eficiente. Para isso torna-se necessário a mudança da maior parte dos componentes da interface do utilizador, desde a autenticação até à forma como a informação é disponibilizada ao utilizador.

Com o evoluir do projeto, a equipa começou a notar o aumento da complexidade e tempo de desenvolvimento, tornando-se cada vez mais custoso para a equipa realizar alterações, devido à rigidez, fragilidade, imobilidade e viscosidade da solução [6], o que é exponenciado pela dimensão da base de código.

[1] No capítulo três serão explicadas com mais detalhe as funcionalidades e a dimensão da aplicação.

1.3 Objetivos

O objetivo deste projeto tem duas dimensões interrelacionadas:

- Adquirir competências na empresa de reengenharia de frontends monolíticos que sofrem dos problemas de manutenibilidade identificados na secção anterior;
- Reengenharia da aplicação frontend identificada na secção anterior de forma a atingir os requisitos identificados, e resolver os problemas de manutenibilidade.

1.4 Abordagem

Na maior parte dos projetos, independentemente da divisão feita do lado do servidor (e.g. arquitetura monolítica, micro serviços), quase todos [5, 7, 8] usam um frontend monolítico. Isso significa que este último tem uma única base de código e que normalmente apenas uma equipa consegue trabalhar em simultâneo de forma adequada no projeto.

Esta generalização do uso de arquitetura monolítica do lado do cliente teve naturalmente as suas vantagens ao longos dos anos [5], mas também problemas como rigidez, fragilidade, imobilidade e viscosidade [5], daí resultando baixa manutenibilidade e instalabilidade [5, 7]. O mesmo pode ser observado atualmente nos monólitos frontend.

De facto, exponenciado pela grande base de código, os problemas de rigidez, fragilidade, imobilidade e viscosidade [6] dificultam cada vez mais perceber o código existente, assim como corrigir erros e criar ou alterar funcionalidades. As mudanças no projeto (backend ou frontend) tornam-se cada vez mais morosas quer a desenvolver quer a instalar e disponibilizar [5, 7, 8]. Podemos, assim, deduzir a necessidade de adoção de uma arquitetura que resolva estes problemas.

Com a adoção bem-sucedida de abordagens arquiteturais baseadas em micro serviços [5] a vários projetos de reengenharia de software monólito do lado do servidor (incluindo projetos na empresa promotora deste projeto), é possível especular que tal abordagem arquitetural também terá bons resultados, pelo que se propõe a reengenharia da arquitetura monolítica da aplicação frontend para uma arquitetura baseada em micro serviços, i.e. Micro Frontend Architecture (MFEA), em que o monólito dá origem a uma aplicação com vários micro frontend interligados e coerentes entre si.

1.5 Gestão do projeto

Apesar de se propor a adoção do estilo MFEA, outras alternativas arquiteturais serão analisadas, pelo que é necessário o estudo, análise, sistematização, seleção e adoção de abordagens arquiteturais e tecnológicas alternativas. Nesse sentido, as tarefas/atividades previstas para o projeto são:

- Estudo da MFEA e de outras arquiteturas que permitam a divisão em módulos;
- Estudo e criação de métodos e tecnologia que assegurem o sucesso da reengenharia:
    - Estudo e sistematização do processo de divisão do monólito em módulos;
    - Estabelecer a forma de comunicação entre módulos;
    - Estudo e aquisição de competências sobre tecnologias de ponta nas várias fases do ciclo de vida de um projeto de software;
- Reengenharia (redesign arquitetural) da aplicação, apresentando alternativas pertinentes e comparando-as;
- Implementação da arquitetura considerada mais promissora;
- Design e implementação do pipeline de CI/CD;
- Testes e avaliação da solução;
- Experiências e Avaliação.

Para a aquisição de conhecimentos é necessário o estudo de MFEA através de fontes de conhecimento significativos, como por exemplo o livro Micro Frontends in Action; Michael Geers, 2018; Manning Publications [5], vídeos de palestras sobre exemplos práticos do mercado referidos no livro e exemplos de tipos de migração desta escala [9].

Após a fase de estudo sobre os pontos em questão e tendo estes sido analisados e sistematizados, será adotado um método para elaborar a prova de conceito que possa comprovar o conhecimento previamente adquirido durante a fase inicial. Depois disso será possível avaliar os resultados obtidos durante o processo de experimentação. Com esta informação torna-se possível averiguar se o uso de MFEA é uma opção viável para o projeto em questão, quais são as suas maiores vantagens e desvantagens.

O planeamento temporal das tarefas pode ser consultado na Figura 1.

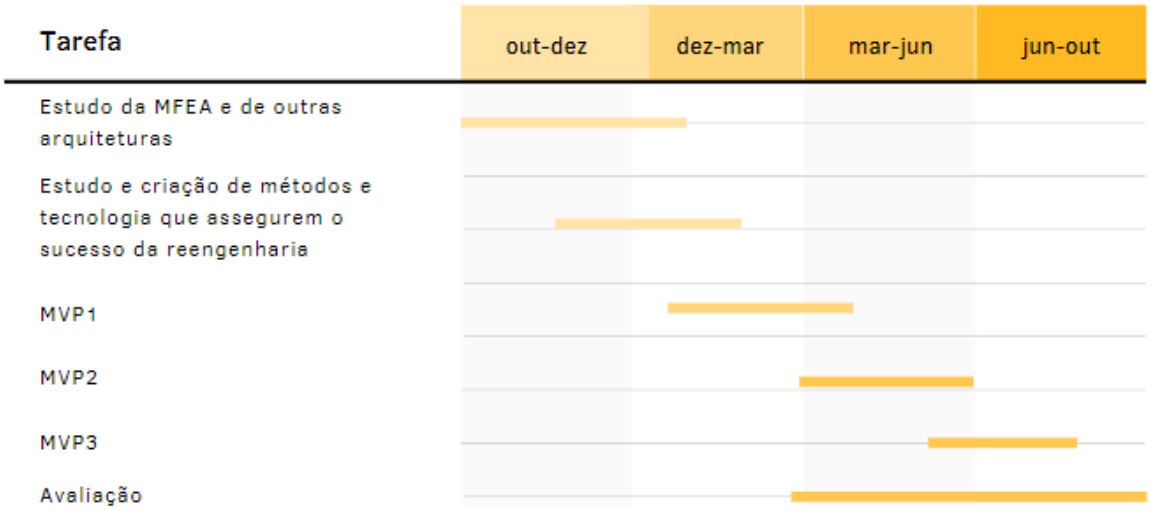

Figura 1 - Diagrama de Gantt

- MVP 1, criação de um MFE, prevê-se que seja o mapa;
- MVP 2, especula-se que venha a ser a criação de pelo menos dois MFE com comunicação entre eles;
- MVP 3, especula-se que venha a ser a implementação do novo design com os respetivos MFE.

1.6 Análise de valor

Esta secção visa apresentar uma análise e valorização do projeto através de perspetivas complementares. Esta versão é uma síntese do conteúdo do Anexo A. Análise de valor.

1.6.1 Segundo o modelo NCD

Tendo em conta o modelo NCD, foram identificados os seguintes fatores:

- **Identificação de oportunidades** - Com a crescente complexidade das páginas web, o uso da arquitetura monolítica pode tornar-se desvantajoso para as equipas e para as empesas. Com o aparecimento dos micro serviços, surge também a MFEA, que visa permitir o uso de uma nova abordagem arquitetural do lado do cliente, através da criação de software modular

com mais baixo acoplamento. Esta arquitetura pretende facilitar tanto o desenvolvimento, implantação e qualidade da solução como também a sua manutenibilidade;

- **Análise de oportunidade** – Tendo o projeto original uma arquitetura monolítica e com potencial para um crescimento de funcionalidades e consequente adaptação do projeto, e tendo em conta vantagens anteriormente observadas na reengenharia de software backend, MFEA pode vir a ser vantajosa para a reengenharia do software de frontend;
- **Geração de ideias -** De forma a enriquecer a ideia, inicialmente foram reunidos os principais conceitos relacionados desenvolvimento web e MFEA assim como de outras arquiteturas que possam ser adequadas ou não à solução (cf.2.3). De seguida, foi realizado o estado da arte em tecnologia relevante, tecnologias de desenvolvimento web, fornecedores *cloud* e tecnologias automação de CI/CD (cf. Anexo B. Abordagens conceptuais);
- **Seleção da ideia** – Foi efetuada a comparação entre as várias tecnologias e os requisitos pré-existentes, descrito no capítulo 4;
- **Definição de conceito**: O objetivo pretendido é obter um software frontend eficaz, eficiente e manutenível, capaz de suportar a solução atual e as novas funcionalidades que possam eventualmente surgir, garantindo implantações independentes.

1.6.2 Valor, valor percecionado e valor para o cliente

Mesmo que MFEA represente mais código do lado do cliente e mais esforço de integração e CI/CD, não é possível dizer que seja desadequada como solução; essa resposta dependerá das necessidades dos desenvolvedores, e em última análise dos utilizadores. Assim sendo, dependendo dessas mesmas necessidades ou do conhecimento sobre elas, pode ser considerada adequada ou desadequada. Assim de seguida pode ser consultada a tabela de benefícios e sacrifícios (Tabela 2) referente a este documento.

Tabela 2 - Benefícios e Sacrifícios da adoção de MFEA

	Serviço	**Relação**
Benefícios	Com o uso desta arquitetura é possível obter projetos mais pequenos, conseguindo desta forma obter um desenvolvimento por funcionalidade. Para além disso existe um menor acoplamento, uma maior flexibilidade e a equipas passam a ser verticais.	Flexibilidade Performance Qualidade Escalabilidade Desacoplamento
Sacrifícios	Para a existência de uma MFEA, é provável que o código existente do lado do cliente aumente, dado cada equipa ser independente.	Mais código Mais esforço de: - Integração - CI/CD

1.6.3 Proposta de valor

MFEA oferece uma nova perspetiva arquitetural do lado do cliente. Através da divisão em pequenos projetos, é capaz de dar à equipa um maior foco nos desenvolvimentos, uma redução da margem de erros, implantações independentes, mais flexibilidade e um maior foco no que é entregue ao cliente [5]. Sendo bastante recente no mercado, torna-se umas das mais prominentes alternativas ao monólito, visto estar diretamente ligada com o principal concorrente do monólito, os micro serviços.

1.6.4 Modelo canvas

O modelo canvas [10] sistematiza um conjunto de informação relevante para a compreensão do projeto, mostrando de forma simples e concisa alguma interligação entre as diversas dimensões (Figura 2).

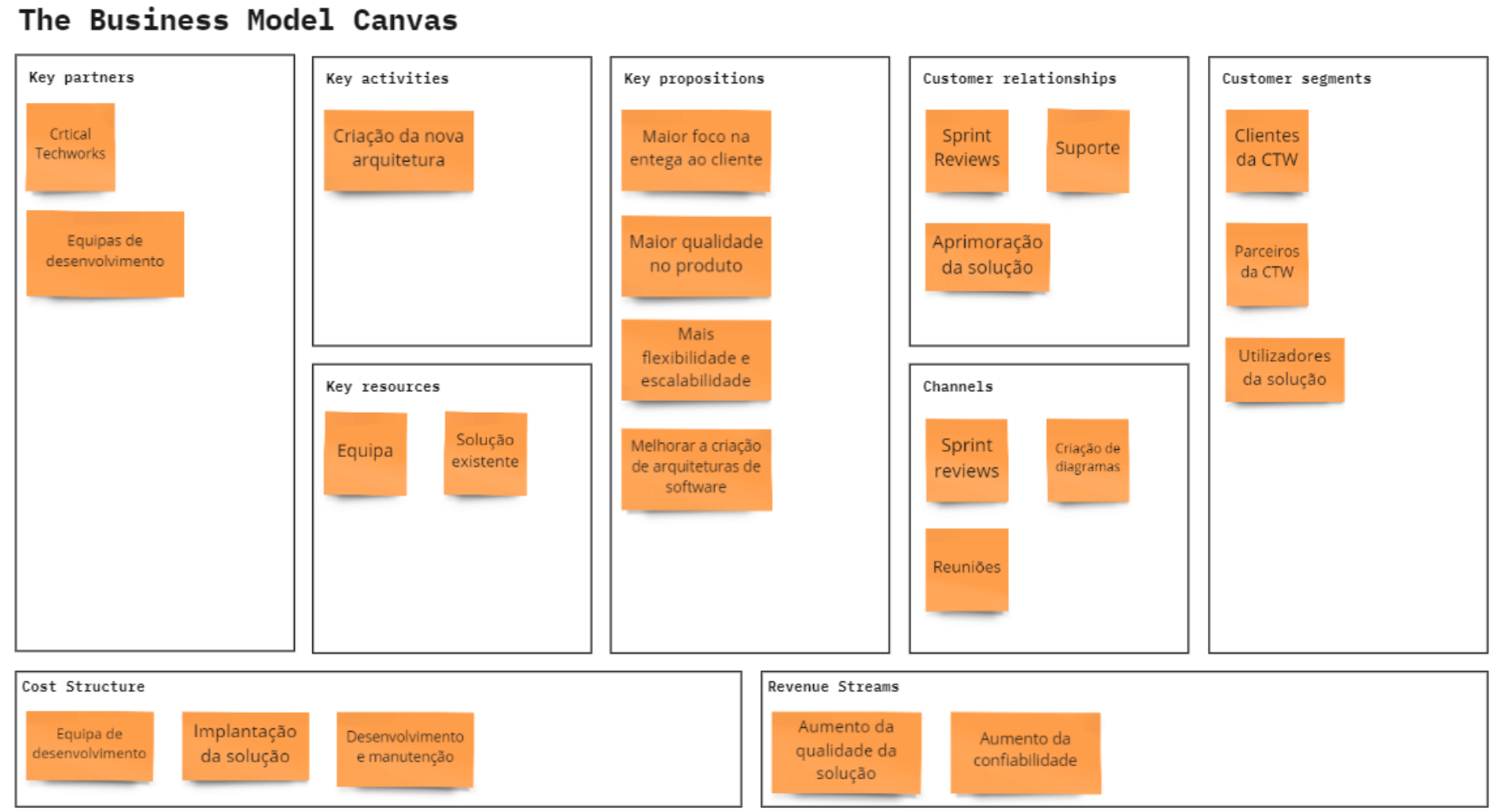

Figura 2 - Modelo Canvas

1.7 Estrutura do documento

Este documento divide-se em sete capítulos e dois anexos:

1. Introdução, este capítulo;
2. Estado da Arte, identifica e analisa abordagens conceptuais e tecnologia existentes relevantes para o projeto;
3. Análise de Sistema, descreve a solução atual, incluindo a arquitetura do sistema, as suas funcionalidades e tecnologias utilizadas;
4. Desenho arquitetural, apresenta várias alternativas arquiteturais, assim como as decisões tomadas;
5. Implementação, descreve a forma como foi implementada a nova arquitetura e mostrados os resultados visuais obtidos;
6. Avaliação e Resultados, descreve o processo de avaliação (incluindo grandezas, hipótese e as metodologias), bem como os resultados obtidos;
7. Conclusões, apresenta uma sistematização crítica do trabalho realizado e objetivos atingidos, bem como uma descrição de linhas de trabalho futuro;
8. Anexo A - Análise de Valor, apresentada uma análise de valor do projeto;
9. Anexo B, abordagens conceptuais e tecnologias relevantes.

2 Estado da arte

Durante este capítulo serão descritas as diferentes arquiteturas concetuais existentes que podem ajudar a colmatar os problemas explicados anteriormente, assim como as tecnologias a serem adotadas para o desenvolvimento da mesma.
O capítulo incluirá secções de análise crítica dos objetos de estudo, nomeadamente comparações, vantagens, desvantagens e principais fatores de adoção.

2.1 Abordagens Conceptuais

Como referido anteriormente as aplicações web atuais utilizam o padrão conhecido como Cliente/Servidor [11]. No desenvolvimento da parte do cliente web existem dois estilos primordiais de design: (i) aplicações web multi-paginadas (MPA) e aplicações web de página única (SPA).

2.1.1 Multiple page application

O padrão MPA é o mais antigo e o seu aparecimento coincide com aparecimento dos primeiros sites web.
Aplicações MPA consistem em várias páginas estáticas compostas por texto, imagens (entre outros) e links para outras páginas com o mesmo tipo de conteúdo. Aquando da mudança de página todo o conteúdo é completamente recarregado. Este “reload” da informação inclui até componentes que sejam comuns a várias dessas páginas, como os cabeçalhos e rodapés.
Segundo este padrão, a parte cliente (frontend) está constantemente a receber novos ficheiros de HTML vindos do servidor (backend) e que respondem às suas necessidades. Esta abordagem torna-se mais aconselhável em aplicações mais simples [12]. Caso exista a necessidade de criação de uma interface mais rica (Rich Internet Application), a performance da aplicação pode ser afetada, aumentando os tempos de espera entre pedidos e prejudicando a experiência do utilizador [12].
O uso de AJAX para a transferência de dados sem necessidade de recarregar a página permite que um MPA seja integrado em vários tipos de aplicações web, melhorando a performance da mesma.

Algumas das vantagens do uso de MPA são [13] [14]:

- Uso de Ajax torna-se fulcral para a performance da página, pois permite carregar apenas um componente da aplicação individualmente;
- Melhor performance em motores de busca, uma vez que cada página pode ser otimizada para uma palavra-chave diferente.

Algumas das desvantagens do uso de MPA são [12]:

- Melhor desempenho em soluções estáticas e com pouco suporte a *Javascript*, dada a natureza do tipo de aplicação possíveis de criar com MPA. *Javascript* é mais indicado/utilizado a SPA [15];
- O desenvolvimento torna-se bastante complexo, aumentando o tempo de desenvolvimento. Quanto maior for mais difícil vai ser conhecer e conseguir dar suporte a toda a solução.

2.1.2 Single-page application

Single-Page Applications (SPA) são aplicações mais recentes que são constituídas por apenas uma página HTML [12]. O frontend em SPA é mais rápido do que as aplicações web tradicionais porque executam a lógica no próprio navegador e não no servidor. Após a abertura da página, apenas dados são trocados ente cliente e servidor [13].

Algumas das vantagens do uso de SPA são [13]:

- Melhor experiência de utilizador, visto não existirem tantos *re-load* e mudanças de página;
- Reduz os *re-loads* desnecessários, pois não é necessário recarregar toda a página;
- O desenvolvimento da aplicação, incluindo debug, torna-se muito mais simples;
- É possível monitorizar operações de rede como pedidos HTTPS e investigar elementos associados à página.

Algumas das desvantagens do uso de SPA são [13] [12]:

- Fraca performance nos motores de busca, porque o seu conteúdo é carregado via AJAX;
- Mais inseguras devidos ao *Cross-Site Scripting,* pois torna-se possível a injeção de scripts do lado do cliente*;*
- O peso das Frameworks utilizadas pode ser prejudicial na performance, porque estas trazem agregadas algumas dependências.

2.1.3 Arquitetura monolítica

Ao longo dos anos, a promoção da arquitetura monolítica dominou a indústria. Este domínio decresceu com o aumento da crescente complexidade dos desenvolvimentos, a consequente maior dificuldade de manutenção dos mesmos e uma maior dificuldade em dimensionar as soluções [7].

As suas caraterísticas são [16]:

- Apenas uma camada, porque apenas existe uma base de código;
- Maior acoplamento, pois existe um elevado grau de dependência entre os módulos;
- Base de código partilhada, pois todo o código está presente no mesmo projeto;
- Responsável pela integração com outras aplicações.

As vantagens deste estilo arquitetural são [17]:

- Simples de desenvolver, durante o início do desenvolvimento;
- Simples de testar, pois basta criar teste e manter a cobertura de apenas uma solução;
- Simples de implantar, porque apenas existe um pacote para ser implantado;
- Simples de escalar horizontalmente, basta correr várias cópias da aplicação usando, por exemplo, um *load balancer*.

As desvantagens deste estilo arquitetural são [16]:

- A manutenção torna-se ineficaz, à medida que a aplicação cresce em complexidade, devido ao seu tamanho e alto acoplamento, tornando difícil efetuar mudanças eficazes;
- Todo o sistema deve ser reimplantado em cada atualização, porque apenas existe uma base de código;
- Qualquer alteração pode afetar o sistema todo, porque um bug criado em qualquer módulo pode afetar toda a aplicação;
- Dificuldade em adotar novas tecnologias, pois essas mudanças afetariam toda a aplicação;
- Dificuldade em escalar, porque diferentes módulos podem ter conflitos em certos recursos.

2.1.4 Arquitetura Micro Serviços

Micro serviços é uma abordagem para desenvolver uma solução como um conjunto de pequenos serviços, cada um executando em seu próprio processo. Esses serviços são construídos em torno

de capacidades de negócios e implementados independentemente por máquinas de implantação totalmente automatizadas.

As suas principais características são [16]:

- Baixo acoplamento, pois os módulos são mais independentes uns dos outros;
- Implantações Independentes, porque é possível implantar cada um deles de forma independente, usando até tecnologias diferentes;
- Organizado em torno de capacidades de negócios;
- Projetos mais pequenos, porque este é um dos objetivos principais deste tipo de arquitetura, divisão entre módulos independentes.

As suas vantagens são:

- Versátil — os micros serviços permitem o uso de diferentes tecnologias e linguagens;
- Fácil de integrar e dimensionar com aplicações de terceiros;
- Os micro serviços podem ser implantados conforme necessário, por isso funcionam bem dentro de metodologias ágeis;
- A manutenção é mais simples e barata — pode fazer-se melhorias ou alterações a um módulo de cada vez, deixando o resto a funcionar normalmente;
- Um projeto modular baseado em micro serviços evolui mais naturalmente, é uma maneira fácil de gerir diferentes desenvolvimentos, utilizando os recursos disponíveis, ao mesmo tempo.

As suas desvantagens são:

- Os componentes são distribuídos, logo os testes globais são mais complicados;
- É preciso controlar o número de micro serviços que estão são geridos, elevados números podem ser impossível de gerir.

2.1.5 Arquitetura Micro Frontend

Nos últimos anos, os micro serviços cresceram bastante em popularidade, muitas organizações adotaram esse estilo arquitetónico para evitar as limitações de grandes monólitos. Embora exista muita informação sobre esse estilo de construção de software, usado do lado do servidor, muitas empresas continuam a ter problemas com bases de código frontend monolíticas [8].

As suas caraterísticas são [16]:

- Projetos mais pequenos, porque este é um dos objetivos principais deste tipo de arquitetura, divisão entre módulos independentes assim como os micro serviços;
- Desenvolvimento por funcionalidade é também um dos principais objetivos deste tipo de arquitetura, de forma a permitir maior foco na funcionalidade entregue;
- Baixo acoplamento, devido à baixa dependência entre módulos;
- Flexibilidade na escolha de tecnologias;
- Equipas verticais, que trabalham apenas numa funcionalidade.

As vantagens deste estilo arquitetural são [5]:

- Otimização de desenvolvimento, pois é necessária pouca coordenação entre equipas *frontend* e *backend* separadas;
- Implantações independentes, pois cada módulo tem uma implantação independente do resto;
- Maior foco no cliente, cada equipa desenvolve diretamente para o cliente. Não existem equipas de operações ou de criação de API;
- Independência da equipa na escolha de tecnologias utilizadas, tendo em conta que sendo módulos separados não existem incompatibilidades;
- Melhor Estratégia de testes, devido ao menor tamanho dos módulos, torna-se mais fácil definir uma estratégia e implementá-la.

As desvantagens deste estilo arquitetural são [5]:

- Maior Redundância, devido há existência de mais código duplicado;
- Mais código do lado do cliente, pois existe pouca partilha de código entre módulos;
- Maior heterogeneidade de tecnologias.

‘Micro frontends descreve uma abordagem alternativa. Divide a aplicação em fatias verticais, construídas desde a base de dados até à interface do utilizador e executada por uma equipa’ [5].
‘Com esta alteração arquitetural, a equipa geral de frontend deixa de ser necessária, e traz algumas vantagens que o monólito não era capaz de proporcionar. Essas vantagens foram a grande forma motivadora para a adoção desta arquitetura por parte de várias empresas’ [5].

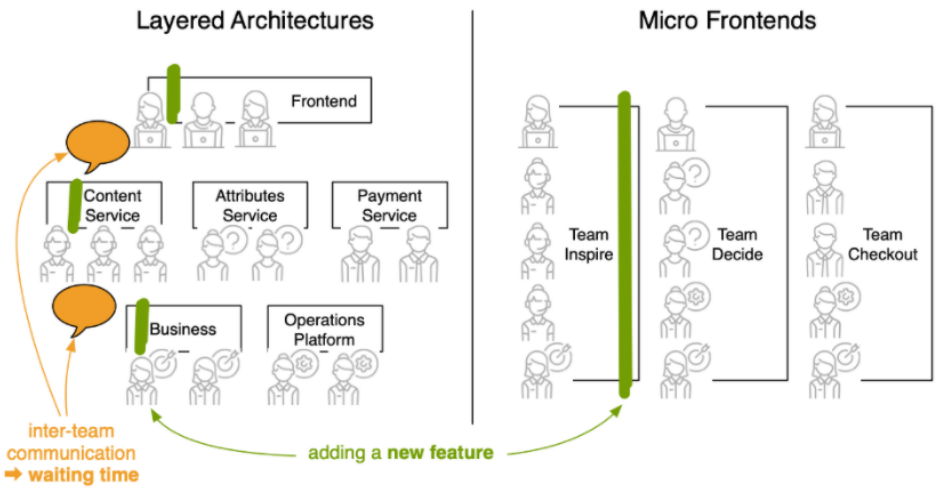

Figura 3 - Comparação entre Layered Architectures e Micro Frontends [5]

As arquiteturas com camadas, explicadas na secção Anexo B. Abordagens conceptuais, em projetos complexos, implicam o envolvimento de várias equipas diferentes no mesmo processo. No entanto, ao utilizar MFEA, todas a pessoas envolvidas num novo desenvolvimento estarão na mesma equipa. Melhorando significativamente a comunicação entre os mesmos sem aumentar a quantidade de trabalho necessária.

Com a mudança para uma arquitetura vertical, algumas das dificuldades que o monólito apresenta são superadas. Com as implantações independentes, o risco de falha reduzido a uma área menor, maior facilidade de entendimento da base de código e facilidade de alteração do mesmo, visto que este passa a ser muito menor com a existência de projetos frontend mais pequenos. Com a divisão do monólito em projetos mais pequenos, as equipas deparar-se-ão com o primeiro desafio que a aplicação desta arquitetura implica. Com esta divisão existe a necessidade da criação de diferentes URL, um para cada projeto criado. Nesta situação as equipa terão de criar um padrão para os URL. Desta forma todas as equipas sabem como comunicar com outros projetos. Um ficheiro JSON/XML partilhado pode ser uma das soluções. Assim qualquer equipa pode atualizar e consultas URL de uma forma simples. Existem também algumas *frameworks* que facilitam esta partilha.

2.1.6 Mapas em páginas web

Sendo a solução referente a este documento, baseada no uso de um mapa, é necessário a escolha de um mapa capaz de suportar as funcionalidades requeridas com uma performance aceitável para

o utilizador. Para fazer uma escolha mais acertada é necessário conhecer um pouco da evolução dos mapas do crescimento da internet e das linguagens de programação, onde vimos passar de simples partilhas e texto, gráficos e imagens para informações muito mais complexas, como informação topográfica. 'O aparecimento do Open GIS Consortium (OGC) como mediador entre o setor público e privado, permitiu uma maior gestão da arquitetura do geoprocessamento de toda a indústria e uma consequente maior interoperabilidade. A visão e o objetivo da OGC passam pela integração completa de dados geoespaciais confiáveis e recursos de geoprocessamento. Isso significa um uso generalizado de software de geoprocessamento interoperável' [18].

Com esta generalização dos dados também os tipos de mapas foram evoluindo. Atualmente os mapas mais popularmente utilizados são conhecidos como *Tiled Web Map*. Estes são capazes de apresentar num *browser* dezenas de informações através de imagens (**Raster**) ou *vector tiles*, que são pacotes de informação geográfica que representam a forma de um quadrado no mapa, conhecidas por Tile.

As maiores vantagens do uso de Tiled Maps com imagens (**Raster**):

- Cada vez que o utilizador muda de zona no mapa com o rato, a maioria das *Tiles* ainda são relevantes, e podem ser mantidas no ecrã, enquanto as necessárias continuam a ser carregadas;
- Funciona em diversos browsers, pois já é usado há bastantes anos e resistiu à evolução dos browsers;
- Renderizado do lado do cliente, desta forma otimizando a performance da aplicação.

As maiores desvantagens do uso de Tield Maps com imagens (**Raster**):

- Não é possível editar o tema do mapa;
- Não é possível adicionar ou esconder informação no mapa sem o uso de outros componentes;

As maiores vantagens do uso de Tiled Maps com vector tiles são [19]:

- Cada vez que o utilizador muda de zona no mapa com o rato, a maioria das *Tiles* ainda são relevantes, e podem ser mantidas no ecrã, enquanto as necessárias continuam a ser carregadas;
- O tema do mapa pode ser alterado de acordo com as necessidades;
- Tiles de menor tamanho, otimizando os mapas *offline*.

As maiores desvantagens do uso de Tield Maps com vector tiles são:

- Renderização lenta em máquinas mais lentas, pois esta é realizada do lado do cliente.

2.2 Estratégias de migração

A estratégia de migração de projetos complexos pode tornar-se custosa e demasiado trabalhosa. Durante esta secção serão apresentadas estratégias alternativas direcionadas à migração de projetos monolíticos para uma MFEA e que se apliquem ao contexto do projeto referido em 0.

2.2.1 Slice By Slice

'Baseado no padrão *Strangler Fig Application*, que visa a construção de uma nova aplicação à volta da anterior, deixando a nova aplicação crescer de forma a 'estrangular' a antiga [20]. Neste caso específico, *Slice by Slice*, é uma estratégia focada em aplicações *frontend* monolíticas serem migradas funcionalidade a funcionalidade' [21].

Após a definição dos diferentes *bounded context* de cada equipa, estas podem começar a migrar as funcionalidades para uma nova aplicação. De seguida é necessário estabelecer mecanismos de integração e finalmente pode-se substituir a funcionalidade na UI do monólito. Este processo repete-se até que o monólito tenha desaparecido. Esta estratégia é representada metaforicamente na Figura 4.

A principal razão para considerar o uso desta estratégia é a reescrita da aplicação ser reduzida em passos mais simples, desta forma apresentando menores riscos. Assim garantem-se implantações frequentes que permitem monitorizar os progressos cuidadosamente. Não existindo um momento big-bang de atualização do sistema, existe sempre uma versão funcional da aplicação, mesmo que decida não continuar a migração [21] [20]. Apesar disso esta estratégia requer uma maior compreensão do sistema existente. Extrair funcionalidades do monólito significa que no mínimo, será necessário adaptar a interface UI do monólito.

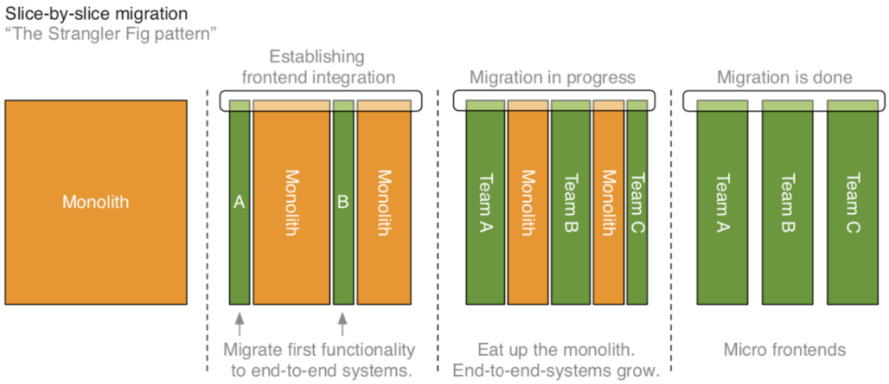

Figura 4 – Metáfora representativa de migração adotando a estratégia Slice by Slice [21]

2.2.2 Greenfield & Big bang

'Esta estratégia é a mais fácil do ponto de vista conceptual [21]. O projeto existente permanece o mesmo, enquanto outro novo é contruído em paralelo.' Com o devido planeamento o novo sistema vai sendo implementado pelas respetivas equipas. Quando estas acabam de implementar as necessárias funcionalidades e configurações, o novo sistema é implantado e disponibilizado. Desta forma torna-se possível remover o projeto antigo. Estes dois sistemas nunca se misturam, como representado metaforicamente na Figura 5 [21].

Começar do zero mostra-se uma óbvia vantagem desta estratégia, evitando ter de lidar com código e dependências pré-existentes. Desta forma as equipas podem focar-se completamente na criação e uma nova arquitetura.

Mesmo assim, existe um nível de risco considerável associado com esta abordagem. O feedback dos utilizadores é nulo durante esta mudança, pelo que a qualidade do software em produção fica sempre questionada durante esse tempo. 'A existência de utilizadores reduz o risco e aumenta a confiança no novo sistema' [21].

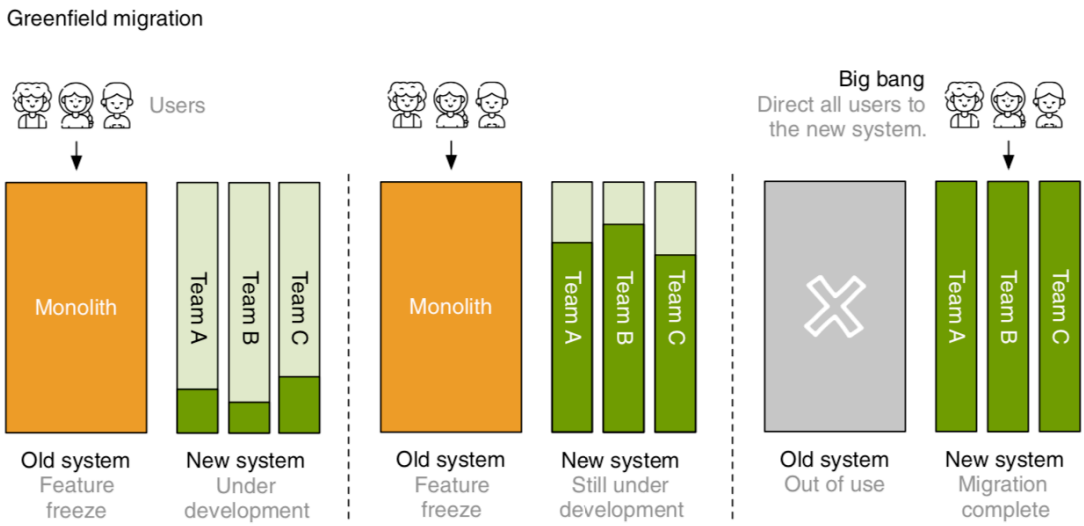

Figura 5 - Exemplo de migração Greenfield & Big Bang [21]

2.2.3 Combinação de Slice by Slice e Greenfield & Big Bang

Tendo em conta as duas abordagens referidas nas secções anteriores e o objetivo deste documento ser a migração arquitetural de um projeto para MFEA, torna-se possível combinar as duas.

Começando por utilizar Greenfield & Big Bang, criando um projeto completamente novo e mantendo o anterior em funcionamento. Mas a migração das respetivas funcionalidades não é feita toda ao mesmo tempo, aqui entra a abordagem de Slice by Slice, que permite a migração funcionalidade a funcionalidade, criando novos MFE que convivem com um monólito, que passa também a ser um MFE, mas cada vez menor.

Esta combinação permite que apenas uma equipa seja capaz de migrar um projeto, apesar de o tempo de implementação seja maior, comparativamente ao que se propõe nas outras abordagens, visto estarem pensadas para mais que uma equipa para otimizar o processo de implementação e encaixarem-se no conceito de equipa vertical da MFEA. Assim sendo é possível conseguir vantagens das duas estratégias anteriores.

2.2.4 Comparação de estratégias de migração

Durante esta secção será realizada uma comparação entre as três diferentes Estratégias de migração explicadas em 2.2.1, 2.2.2 e 2.2.3. Permitindo uma melhor perceção das suas vantagens e desvantagens comparativamente umas às outras, para tal foi realizada a Tabela 3.

Tabela 3 - Comparação de Estratégias de migração

Caraterística/Dimensão	Slice by Slice	Greenfield & Big Bang	Combinação
Rapidez de implementação	x[2]	x	x[3]
Facilidade de implementação	x	x[4]	x[4]
Monitorização	x	x	x
Implantações menores	x	x	x
Feedback do utilizador	x		x
Disponibilidade	x	x	x

[2] Requer maior compreensão do sistema existente
[3] Requer compreensão do sistema existente e de como o migrar
[4] Conhecendo bem as novas tecnologias e abordagens

2.3 Alternativas arquiteturais

Durante esta secção serão explicadas em detalhe as alternativas arquiteturais que permitam a criação de uma MFEA, assim como de metodologias alternativas para migração de projetos para este tipo de arquitetura.

Será também apresentada uma comparação entre as abordagens descritas, tendo em conta as suas vantagens e desvantagens, permitindo assim efetuar uma seleção criteriosa das mesmas.

2.3.1 Client Side Integration

Nesta secção serão explicadas em mais detalhe as técnicas de integração do lado do cliente, que são mais utilizadas em aplicações onde a performance e a user experiency devem ser tidos em conta. Ao utilizar Client Side Integration onde o HTML é produzido e atualizado diretamente no browser, assim são abertas portas a diferentes alternativas.

2.3.1.1 Integração via Links

Ao utilizar a integração via links, cada equipa constrói e disponibiliza a sua página autónoma. Estas aplicações contem o seu próprio HTML, CSS*, scripts* e outros ficheiros estáticos requeridos pelas mesmas.

As equipas não têm de se preocupar com a linguagem de programação, ou framework que as outras equipas utilizam. Desde que as aplicações estejam disponíveis em URL previamente definidos, é possível montar uma solução maior através de outras menores, completamente independentes.

Esta integração funciona quando todas as equipas combinam previamente os seus padrões de URL. Naturalmente os URL podem ter de ser alterados por vários motivos e notificar todas as equipas, em projetos de larga escala pode tornar-se moroso. Nestes casos, um ficheiro JSON partilhado pelas equipas pode ser uma solução para manter todas atualizadas quanto às mudanças de URL.

As suas principais vantagens são [22]:

- Baixo acoplamento, pois as equipas não necessitam de saber qual a linguagem de programação, técnica de implantação ou técnica de aplicação de estilos umas das outras, desde que os URL estejam acessíveis nos locais pré-definidos;
- Alta robustez, pois dificilmente toda a aplicação estaria em baixo no mesmo momento, visto que são independentes umas das outras.

As suas principais desvantagens são [22]:

- Integração via links não é suficiente na maioria dos casos, porque não existe maneira de partilhar dados entre aplicações;
- Maior redundância, pois as partes partilhadas pela aplicação, como por exemplo o header, teriam de ser replicadas e mantidas por todas as equipas.

2.3.1.2 Composição via Iframe

Iframe é usado para incorporar outro documento dentro do documento HTML atual, neste caso significa incorporar MFE. Com os iframes, é possível incorporar uma página noutra página, mantendo as mesmas propriedades e um baixo acoplamento e robustez. Para isso, como exemplificado na Figura 6, basta adicionar ao HTML da página onde queremos adicionar o novo MFE, uma tag onde é especificado o caminho absoluto desse MFE a carregar [22].

```
1 ...
2 <iframe src="http://localhost:3002/recommendations/porsche"></iframe>
3 ...
```

Figura 6 - Exemplo do uso de Iframe

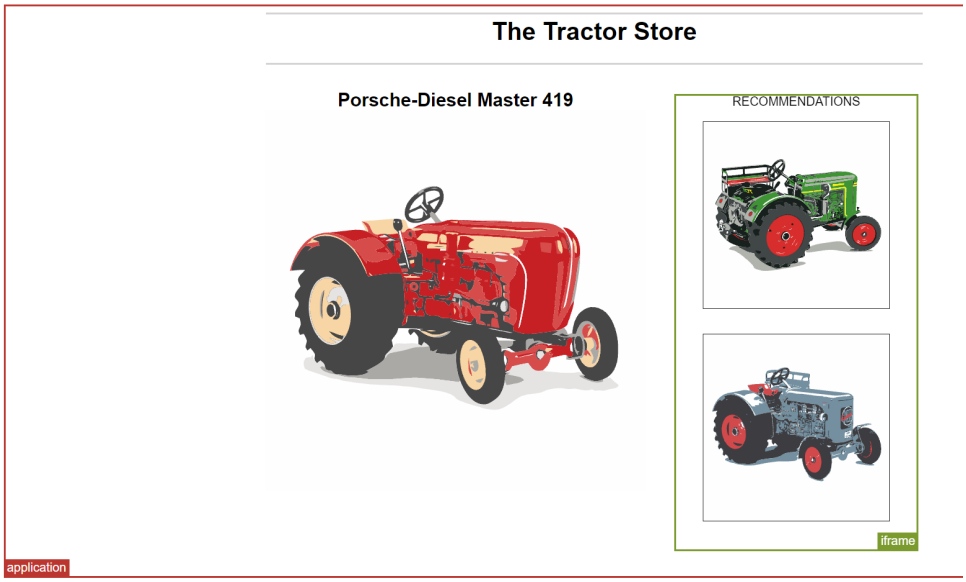

Figura 7 - Resultado teste do uso de IFrame

Mesmo assim, o uso de Iframes apresenta um aumento do acoplamento no que toca ao layout da solução. Pois o documento principal precisa de saber a altura exata do conteúdo do Iframe para evitar problemas relacionados com os estilos da página.

As suas principais vantagens são [22]:

- Isolamento, pois os scripts e estilos não podem vazar para dentro ou fora;
- Funciona numa ampla gama de *browsers*;
- Funcionalidades de segurança disponibilizadas pelo iFrame.

As suas principais desvantagens são [22]:

- Ausência de uma solução fiável para a altura automática;
- Fraca performance, porque cada iFrame cria um novo contexto de browser, criando um maior uso de memoria e CPU;
- Má acessibilidade, porque iFrames quebram a semântica da página;
- Baixa compatibilidade com buscas otimizadas (SEO), pois usando um IFrame o browser identifica duas aplicações.

2.3.1.3 Composição via AJAX

A integração via AJAX é mais direta e robusta e mostra-se também mais simples de implementar. Através de um pedido AJAX é possível obter um fragmento, que de seguida pode ser injetado na DOM da página principal. Ao contrário do iframe, todo o conteúdo é

integrado em apenas um DOM e as equipas que utilizam o fragmento não necessitam de saber com antecedência o seu tamanho. Torna-se assim uma alternativa viável ao uso de iFrames ou Links.

```
window
  .fetch(url)
  .then(res => res.text())
  .then(html => {
    element.innerHTML = html;
  });
```

Figura 8 - Exemplo de injeção de fragmento no DOM via AJAX [69]

Existindo apenas um contexto de navegação, ou seja, uma só aplicação, são também introduzidos alguns desafios de isolamento. Como exemplo, as equipas terem que construir as suas aplicações de forma a não gerar conflitos com as restantes. Se duas equipas criam duas regras de css iguais ou tentam alterar o valor da mesma variável, pode revelar-se um problema. Por isso é crucial introduzir convenções como por exemplo um prefixo de nível de componente e outro nível que represente a equipa, desta forma evitando conflitos de estilos.

As equipas responsáveis por cada projeto podem disponibilizar as suas aplicações através de diferentes plataformas de hospedagem, o que leva à existência de diferentes domínios. Estes domínios diferentes, podem gerar URL demasiado extensos, que dificultarão a perceção do utilizador.

Através da criação de um servidor web partilhado é possível solucionar o problema anteriormente referido. Passando este a ser o ponto central onde chegam os pedidos da aplicação. O servidor irá encaminhar os pedidos para a respetiva aplicação responsável, previamente definidas através de uma tabela de encaminhamento. Para desenvolver uma abordagem deste tipo pode ser utilizado NGIX [23].

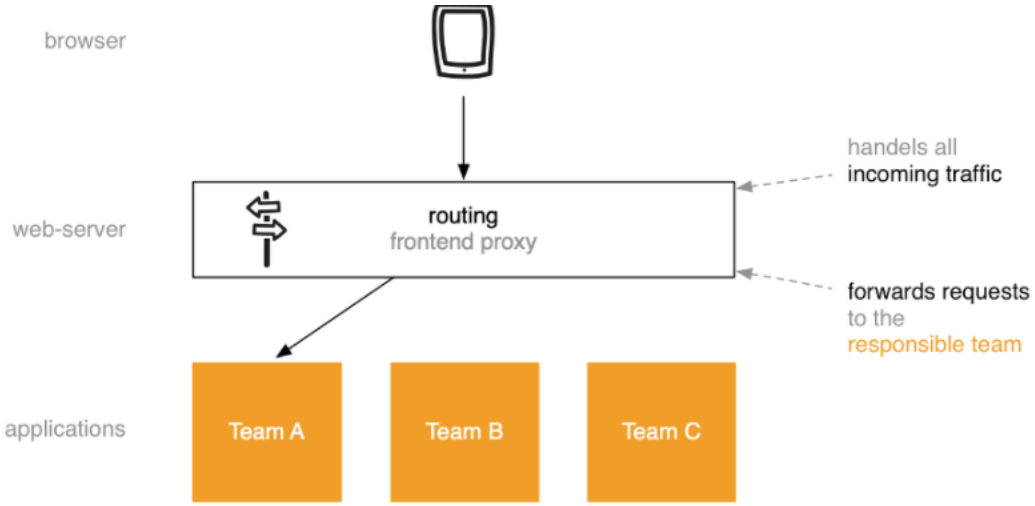

Figura 9 - Exemplo do uso de NGINX

As suas principais vantagens são [22]:

- Mecanismos de busca e acessibilidade, mesmo que todo o HTML naoe steja presente quando a página inicial é carregada, este modelo funciona bem em motores de busca;
- Apenas um DOM, em contrate com Iframe;
- Manipulação de erros flexível, isto devido ao uso de AJAX que permite o alterar o fragmento a mostrar, em caso de erro.

As suas principais desvantagens são [22]:

- Fraco Isolamento, por isso as equipas devem acordar convenções de nomes;
- Cada ação do utilizar espoleta um novo pedido ao servidor, podendo gerar problemas de performance, onde certos fragmentos chegam mais tarde que outros afetando também a experiência do utilizador.

2.3.1.4 Web Components

Web Components são uma combinação de diferentes API que permitem a criação de elementos customizados e reutilizáveis. As suas funcionalidades são separadas do restante código, podendo assim ser reutilizados por toda a aplicação independentemente da *framework* utilizada para a criação da solução [24].

É possível então usufruir facilmente do modelo de componente neutro e padronizado oferecido pelos web components, que se mostra vantajoso ao permitir o encapsulamento e interoperabilidade de elementos HTML individuais.

Estes componentes são baseados em quatro tecnologias diferentes [22, 24]:

- Custom Elements: permite definir elementos customizados e comportamentos adjacentes, que podem ser utilizados de diferentes formas na interface da aplicação;
- Shadow DOM: permite isolar uma árvore independente do DOM principal, evitando colisões de estilos e conflitos com as restantes partes da aplicação;
- Templates HTML: Elementos que permitem a escrita de templates de marcação que não são mostrados na página. Estes facilitam a sua reutilização como modelo de estrutura de um elemento customizado.

As suas principais vantagens são [22]:

- Técnica de integração bastante utilizada;
- Custom Elements e Shadow DOM fornecem funcionalidades de isolamento extra que normalmente se traduz em aplicações mais robustas;
- Maior encapsulamento e interoperabilidade de elementos HTML;
- Podem ser utilizados declarativamente, como por exemplo utilizar uma div.

As suas principais desvantagens são:

- Apesar de o suporte dos browsers atuais no que toca a componentes web ter melhorado drasticamente ao longo dos últimos anos, pode tornar-se complicado dar suporte a browser mais antigos, tornando o uso de Shadow DOM mais complexo;
- Maior criação de código do que usando uma Framework de projetos UI.

2.3.2 Server Side Integration

Server Side Integration é uma abordagem amplamente utilizada e que consiste na assemblagem de diferentes fragmentos e templates do lado do servidor. Normalmente é alcançada através da criação de um serviço agregador entre o browser e os servidores das aplicações, como ilustrado na figura seguinte [25].

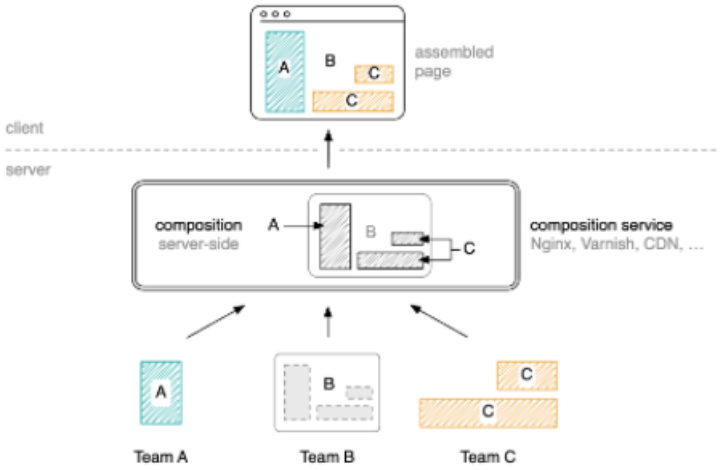

Figura 10 – Diagrama exemplo de Server Side Integration [73]

Um dos benefícios mais significativos é o facto de a página já está totalmente montada quando chega ao browser, aumentando as velocidades do carregamento da primeira página que o utilizar vê [26].

2.3.2.1 Server Side Includes

Server Side Includes (SSI) é uma linguagem usada do lado do servidor, útil para incluir o conteúdo de um ou mais ficheiros numa página web dentro de um servidor web, através da sua diretiva '#include' [27]. Esta linguagem é trocada pelo conteúdo do URL que seja incluído na mesma e pode ser usada como é demonstrado na imagem que se segue.

```
<!--#include virtual="/url/to/include" -->
```

Figura 11 - Exemplo do uso da diretiva '#include'

'Server Side Integration pode ser uma excelente ferramenta para melhorar a performance das páginas web.' Através do uso de múltiplas diretivas é possível obter o carregamento paralelo de fragmentos. Se por exemplo for usado Nginx, este irá em primeiro lugar obter toda a estrutura da página e depois todos os fragmentos em paralelo, diminuindo assim o TTFB (Time To First Bite) para a soma da geração da estrutura e do fragmento mais lento a chegar.

Comparativamente com o AJAX referido em 2.3.1.3, o uso da diretiva 'include' normalmente significa uma melhora significativa da performance, podendo produzir melhores resultados até 40%, dependendo do caso de uso [26]. Por isso o uso de fragmentos dentro de outros fragmentos pode levar à criação de uma arvore de fragmentos, que se tornará prejudicial á performance da aplicação [26]. Também o uso de fragmentos que sejam mais lentos que o desejado para a funcionalidade em questão, podem tornar-se um problema de performance, visto ser o tempo do último fragmento mais lento que influencia o TTFB.

Para a criação de páginas de maior escala, onde existam fragmentos que o seu uso/necessidade não seja instantâneo ao abrir a página, pode ser adotado *lazy-loading,* um padrão que permite adiar a inicialização de um objeto até este ser necessário. Este tipo de carregamento torna possível reduzir o tamanho da marcação inicial que o cliente precisa de carregar, permitindo assim contribuir para a eficiência do funcionamento do programa e para melhorar o tempo de resposta na abertura da página [28].

Mesmo assim, é possível começar a enviar os primeiros fragmentos e dados ainda mais cedo. Usando por exemplo um envio parcial de fragmentos, onde o primeiro fragmento é o modelo da página e os restantes chegam de seguida, otimizando assim o desempenho do browser que começa a mostrar fragmentos mais cedo. Este mecanismo é disponibilizado pela Varnish Enterprise.

Outro exemplo é a ideia de modelos de streaming, que leva este um passo um pouco mais além. Com este modelo é possível sobrepor o carregamento da estrutura da página e dos seus fragmentos, enquanto ao mesmo tempo, os envia para o browser. Assim torna-se possível diminuir ainda mais o TTFB. Tailor e Podium são duas frameworks que permitem o uso de streaming em conjunto com este tipo de composição [26].

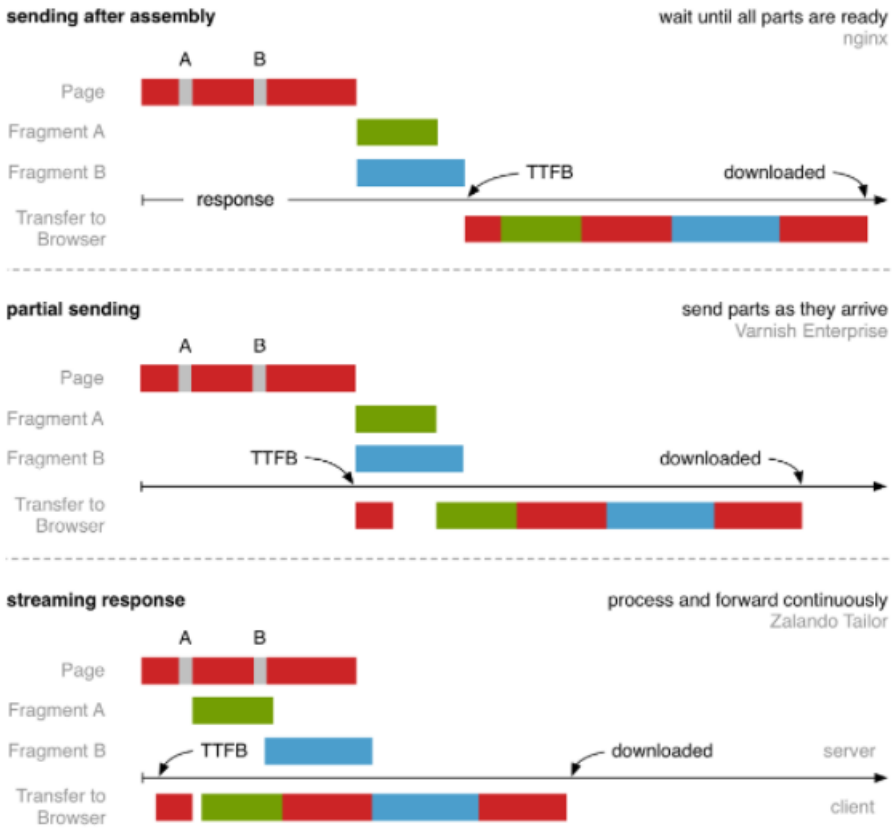

Figura 12 - Comparação entre diferentes abordagens de Server Side Includes [26]

As suas principais vantagens são [26]:

- Excelente performance, reduzindo bastante o TTFB;
- Tecnologia robusta e bem testada;
- Boa compatibilidade com motores de busca.

As suas principais desvantagens são [26]:

- Aplicações de tamanho elevado podem ter um alto TTFB;
- Desenvolvimento local torna-se complexo, visto ser necessário um web server com SSI a correr localmente;
- Para construir aplicações bastante reativas às necessidades do utilizador é necessário combinar Server Side com Cliente Side Integration.

2.3.2.2 Edge Side Includes

'Edge Side Includes (ESI) é uma linguagem de marcação baseada em XML que fornece um meio de reunir recursos em clientes HTTP [29]. Assim como o SSI o ESI foi projetado para alavancar ferramentas do lado do cliente que permitam melhorar a *user experience*, reduzir o processamento no servidor de origem e aumentar a disponibilidade [26]. A integração de ESI e feita como demonstrado na seguinte imagem e seria necessário usar um dos servidores proxy referidos anteriormente, substituindo o Nginx usado no exemplo da secção anterior.

```
<esi:include src="https://tractor.example/fragment" />
```

Figura 13 - Exemplo do uso de Edge Side Includes

O comportamento, assim como no SSI, varia entre implementações. Por exemplo, se for usado Varnish, obtém-se um envio parcial de fragmentos, como referido em 2.3.2.1 [26]. As vantagens e desvantagens do uso de ESI podem ser consultados na secção anterior 2.3.2.1 pois estes são coincidentes.

2.3.3 Unified Single Page App e App Shell

Durante as secções anteriores, foram explicados em detalhe métodos de composição e comunicação, tanto do lado do cliente como do servidor, com o objetivo de integrar interfaces de diferentes equipas numa só página. Nesta secção, o foco será na integração ao nível da página.

Porque a maior parte das atuais *frameworks* de *javascript* já contém uma solução dedicada ao encaminhamento permite a navegação entre páginas de uma forma simples, sem necessidade de atualizações em cada *click* em links, atualizando apenas as partes que sofreram alterações.

Contudo numa aplicação monolítica, normalmente ou se constrói uma aplicação com páginas carregadas pelo servidor ou se opta por implementar uma SPA [30]. Onde no primeiro caso, as transições de página obrigam a um completo recarregar de página, e no segundo, as transições são realizadas inteiramente do lado do cliente, através de um router. Já num contexto MFEA essa opção binária não é obrigatória. Assim nasce o conceito de App Shell.

App Shell

Uma App Shell é um padrão que permite a construção de aplicações web de forma progressiva, que carrega fragmentos de forma fiável e instantânea, muito semelhante a aplicações a aplicações nativas [31]. Esta atua como uma aplicação pai de todos os micro frontend existentes. Todos os pedidos chegam à App Shell e esta seleciona o fragmento correto a mostrar ao utilizador, como pode ser consultado na Figura 14.

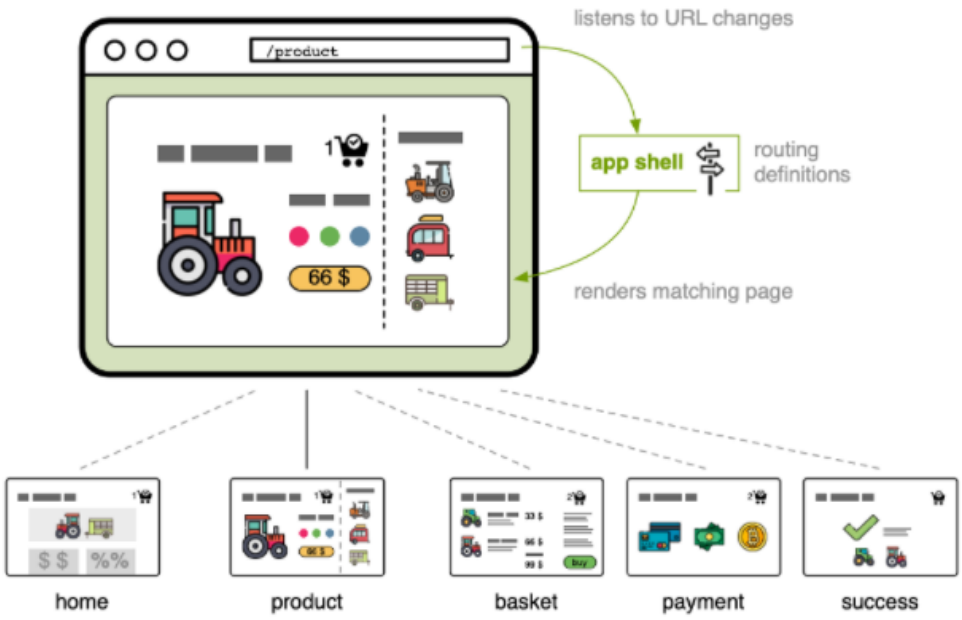

Figura 14 - App Shell [30]

A App Shell é constituída por quatro características essenciais [30] [31]:

- Providenciar uma página HTML partilhada e o CSS básico da aplicação;
- Mapear os URL de todas as páginas existentes;
- Apresentar a página requerida pelo utilizador;
- Inicializar e desligar as próximas ou anteriores páginas.

Para o bom funcionamento da mesma é necessário a existência de um acordo, onde cada equipa publica uma lista de URL que está a gerir, permitindo que outras equipas possam usar estes URL para ligar a uma página específica.

No caso de este *routing* ser apenas de um nível, a Shell terá de conhecer todos os URL existentes na aplicação. Neste caso a adição de um novo URL ou alteração ao mesmo, significa que também será efetuado um reajuste à App Shell, assim como uma nova implantação da mesma. Criando assim um maior acoplamento [30].No caso da Shell de dois níveis, é possível resolver esse acoplamento extra pois a Shell fica apenas responsável por encaminhar para as equipas corretas, deixando as equipas configurar um router interno que decide a página especifica a mostrar, desta feita já dentro do contexto da equipa.

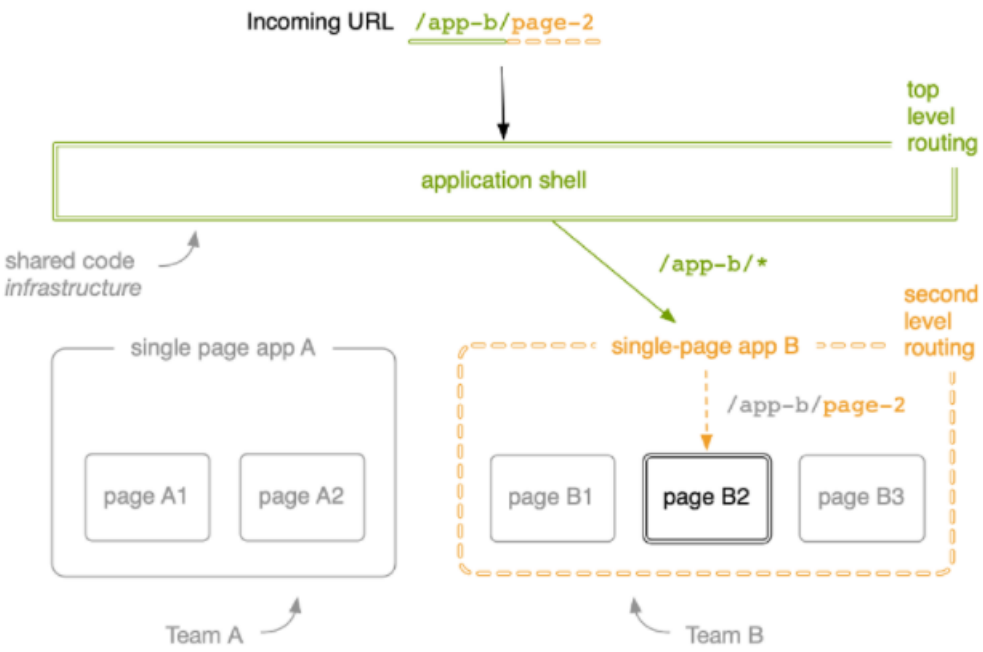

Figura 15 - Exemplo de Routing de dois níveis [30]

Desta forma é possível obter várias SPA por equipa, que ficam contidas dentro de outra SPA, a AppShell, como demonstrado na figura anterior. Assim é reduzindo o esforço de encaminhamento da AppShell, atribuindo a responsabilidade da mostragem da página especifica para o router de cada equipa responsável. As equipas podem, desta forma, adicionar novos URL á sua aplicação sem causar nenhuma necessidade de mudança na App Shell e esta passa apenas a necessitar de saber o prefixo de cada equipa, logo apenas precisa de sofrer alterações, caso exista uma nova equipa [30].

Esta solução é vantajosa para aplicações onde a interatividade é um fator mais importante que o tempo inicial de carregamento. Contudo também apresenta alguns desafios, como a partilha do documento HTML, dificuldade em perceber a origem dos erros e a Shell ser um ponto de falha único, assim como se denota um amento considerável da complexidade. No caso da existência de um erro de um nível mais severo, a aplicação pode ficar completamente inacessível. Para evitar problemas deste género é necessário criar uma AppShell com código de qualidade e bem testado.

2.3.4 Universal Composition

Universal composition consiste na combinação de técnicas de composição do lado do servidor e do lado do cliente. O uso desta técnica foi facilitado pelo amplo suporte de *universal rendering* por parte das frameworks existentes no mercado.

Usando esta técnica, como pode ser consultado na Figura 16, o primeiro pedido usa técnica como ESI ou SSI, que agrupa todo HTML estrutural do lado do servidor, que depois será enviado para o browser. Já no browser, cada micro frontend é responsável por se "hidratar" e ser interativo. A partir deste ponto, todas as interações acontecerão apenas do lado do cliente e o HTML será diretamente atualizado no browser [32].

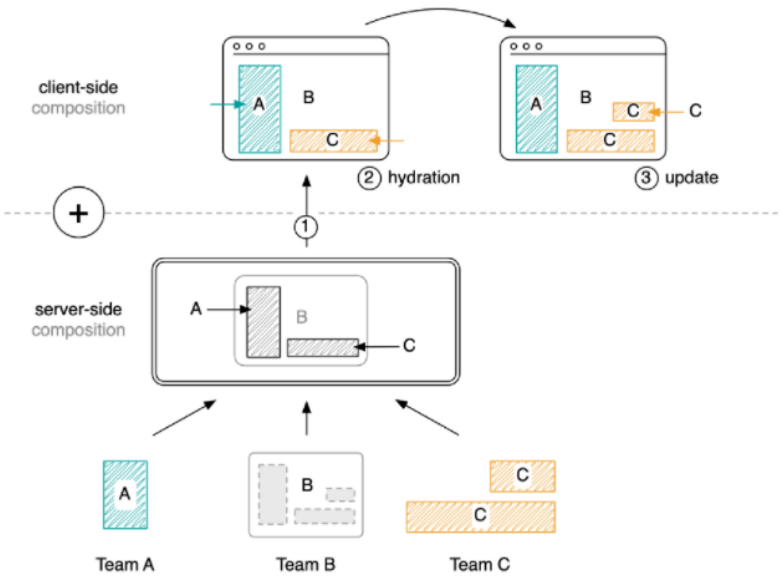

Figura 16 - Exemplo de Universal Composition [32]

Esta técnica mostra-se vantajosa em aplicações onde o primeiro carregamento da página, a interatividade e a comunicação entre micro frontends são pontos cruciais. Uma vez que são utilizadas duas técnicas de integração diferentes, a equipa que fornece o componente micro frontend, deve disponibilizar tanto a definição do elemento HTML costumizado como o endpoint SSI que o contém. Desta forma a equipa que utiliza o micro frontend pode especificar ambos [32].

Apesar de combinar os benefícios da renderização do servidor e do cliente a *universal composition* tem alguns custos extra. Configurar, executar e encontrar a fonte dos erros, pois torna-se mais complicado do que com soluções puramente client ou server-side. Todos os desenvolvedores veem-se obrigados a compreender como a integração no servidor e hidratação no cliente funcionam [32].

2.3.5 Build Time Integration

Esta técnica visa a publicação de MFE como um *package,* que será posteriormente incluído numa aplicação pai que funciona como container, como demonstrado na Figura 17. Isto significa que, após o build da aplicação, será gerado apenas um bundle. Apesar de parecer uma técnica que faz sentido inicialmente, o processo de implantação pode tornar-se mais complexo. Pois cada micro frontend deve ser recompilado e implantando de novo para ser possível aplicar uma alteração [33].

```
{
  "name": "@father/container",
  "version": "1.0.0",
  "description": "the father web app",
  "dependencies": {
    "@childs/browse-child1": "^1.2.3",
    "@childs/order-child2": "^4.5.6",
    "@childs/user-child3": "^7.8.9"
  }
}
```

Figura 17 - Exemplo de Build Time Integration

Na Figura 17 é possível que a aplicação com o nome “father/container”, tem três dependências, “browse-child1”, “order-child2”, “user-child3”, que serão compiladas durante o processo de *build* da mesma.

Assim sendo é possível perceber que esta técnica introduz um acoplamento desvantajoso quando o objetivo passo por desacoplar um monólito em pequenos projetos mais pequenos e onde o processo de implantação se pretende também desacoplado.

2.3.6 Resumo e comparação

Após a elucidação de várias alternativas arquiteturais para MFE feita durante as secções anteriores, foi possível concluir que existem cinco possibilidades diferentes:

- Client side;
- Server Side;
- Unified Single Page Application;
- Universal Composition;
- Build time.

Cada uma das opções apresenta diferentes formas de aplicar uma arquitetura MFE e naturalmente que todas apresentam vantagens e desvantagens, para tornar mais fácil a comparação entre as mesmas, foi realizada a Tabela 4 que pode ser consultada de seguida.

Os parâmetros de comparação serão os seguintes:

1. Baixa infraestrutura inicial necessária;
2. Facilidade de implementação;
3. Baixo acoplamento;
4. Implantações independentes;
5. Isolamento;
6. Flexibilidade;
7. Facilidade de manutenção;
8. Boa performance;
9. Lazy loading;
10. Ui interactiva.

Tabela 4 - Comparação entre alternativas arquiteturais

	1	2	3	4	5	6	7	8	9	10
Links	x	x	x	x	x		x			
Iframe	x	x		x	x		x			
Ajax	x	x	x	x		x	x		x	x
Web components		x[5]	x	x	x	x	x[5]		x[6]	x
Ssi			x	x		x		x	x[5]	
Esi			x	x		x		x	x[5]	
App shell			x	x		x	x		x	x
Build time				x	x	x			x	x

2.4 Tecnologias de integração

Esta secção dedica-se a escrutinar as *frameworks* existentes que permitam a integração dos vários micro frontend. Desta forma, analisando as suas principais características, vantagens e desvantagens.

2.4.1 Build Time Integration

2.4.1.1 Bit

'Bit é uma ferramenta que permite construir aplicações com componentes independentes. Aplicando a premissa dos micro serviços a todo o espectro da aplicação'. Usando Bit é possível uma equipa desenvolver, testar e implantar componentes de forma autónoma, ao mesmo tempo que colaboram continuamente na criação de aplicações de maior escala.

Os principais conceitos do Bit são [34]:

- **Component**: descreve qualquer componente de software que tenha uma funcionalidade de negócio clara e bem definida (Bounded Context Domain Driven

[5] Em web browsers recentes
[6] Combinado com AJAX

Design (DDD)). No desenvolvimento frontend os componentes podem ser componentes UI de pequena escala como botões e filtros ou página inteiras;

- **Workspace**: proporciona uma experiência monolítica para a construção de aplicações distribuídas através de componentes independentes. Permite ainda o desenvolvimento de cada componente isoladamente. Todos os componentes podem ser adicionados dinamicamente, importados ou removidos do workspace;
- **Scope**: conjunto de componentes necessários para criar uma funcionalidade na integra, desde a respetiva UI até ao consumo dos dados.

Através do workspace é possível o desenvolvimento de projetos web modulares, transformando qualquer projeto num mono repositório composto por vários componentes ou scopes. Desta forma as equipas podem trabalhar no código de componentes que não pertencem ao mesmo repositório.

As suas principais vantagens são [34]:

- Isolamento dos componentes criados;
- Implantação independente dos componentes criados, que serão posteriormente integrados durante o build;
- Oferta de componentes personalizados pré feitos.

A principal desvantagem é [34]:

- Integração de componentes ocorre durante o build, criando um maior acoplamento.

2.4.2 Client Side

2.4.2.1 Luigi

Luigi é uma *framework javascript micro frontend,* que permite a criação de interfaces do utilizador, derivadas de fragmentos existentes. Não só facilita a comunicação entre a aplicação principal e os restantes frangmentos, como também oferece configurações de *routing,* navegação, autorização e experiência do utilizador [35].

Composto por duas aplicações base, Luigi Core e Luigi Client, estas permitem estabelecer a ligação entre micro frontends. Sendo o Luigi Core responsável pela disponibilização da criação

da aplicação principal e o Luigi Client responsável por disponibilizar a criação de aplicações fragmento [36].

2.4.3 Server Side

2.4.3.1 Podium

Podium é uma biblioteca para a construção de micro frontend criada pela Finn.no e é baseada em Node.js. Consiste em duas bibliotecas diferentes, biblioteca de *layout* e biblioteca *podlet*. Nesta framework, a composição é feita solicitando cada fragmento independente via HTTP e, de seguida, colocar cada fragmento no seu correto local na página estrutural de HTML [37]. Estes fragmentos Figura 18, denominados *podlets,* consistem num manifesto escrito em JSON e servido via HTTP, que define [37]:

- Endpoint HTTP para o conteúdo do *podlet*;
- Endpoint HTTP para um possível caso de erro;
- Conjunto de endpoints HTTP para os ficheiros de *css*;
- Endepoints HTTP que devam ser públicos.

Este manifesto serve como contrato para a comunicação entre equipas diferentes, através do nome e da versão disponibilizados, como se pode ver na seguinte figura.

Figura 18 – Exemplo de Podlet [26]

Já a estrutura inicial de HTML é denominada por *layout*, coloca cada *podlet* no local apropriado da página e disponibiliza o resultado. Também é responsável por criar e anexar um contexto Podium aos pedidos realizados a cada podlet.

Desta forma a ideia de ter um manifesto que descreve um contrato de integração de cada um dos fragmentos torna-se bastante útil. Podium ainda inclui ambiente de desenvolvimento para podlets e versionamento, assim como esta dotado de uma excelente documentação [26].

2.4.3.2 Tailor

A Zalando, uma empresa alemã, transformou o seu site que, inicialmente era um monólito, para uma arquitetura MFEA. A esse projeto chamou-lhe *Mosaic* [38]. Após esse projeto estar concluído, partes da infraestrutura de integração do lado do servidor do mesmo foram publicadas. Assim nasceu o Tailor, um serviço de layout que usa *streaming* para compor a página web a partir de serviços de fragmentos [39]. A característica mais proeminente do Tailor é o suporte para modelos de *streaming* [26]. O resultado é enviado para o *browser* como *template* básico denominado layout. Este é analisado e os fragmentos começam a aparecer na página principal.

Para além do HTML utilizado, um endpoint de um fragmento, também podem ser especificados estilos e scripts associados a um fragmento [39]. Através de HTTP *headers,* que serão lidos pelo Tailor e adicionados ao documento, como pode ser consultado na seguinte figura.

```
HTTP/1.1 200 OK
Link: <http://localhost:3002/static/fragment.css>; rel="stylesheet",
      <http://localhost:3002/static/fragment.js>;  rel="fragment-script"
Content-Type: text/html
Connection: keep-alive
```

Figura 19 – Fragmento com estilos e scripts associados [26]

Estes fragmentos aceitam ainda outro tipo de atributos que oferecem diferentes funcionalidades como timout e fallback.

As suas principais vantagens são [39]:

- Adota o uso de Server side rendering;
- Assegura um rápido TTFB;
- Alta tolerância a falhas.

As suas principais desvantagens são [26]:

- Código deve ser embrulhado num modulo AMD (Asynchronous Module Defenition), um tipo de modulo *javascript*;
- Não é possível controlar como os scripts/estilos são adicionados ao HTML.

2.4.4 Unified Single Page App com Single spa

Single SPA é uma *framework* que tem como objetivo agrupar vários micro frontend numa só aplicação. É agnóstico quanto às *framework* usada para contruir cada fragmento, permitindo que vários fragmentos contruído utilizando diferentes *frameworks* coexistam na mesma página. Desta forma é possível obter implantações independentes, lazy load de fragmentos e uma maior flexibilidade de mudança de tecnologia utilizada para a criação da aplicação UI [40].

Esta Framework inspira-se nos ciclos de vida dos componentes de outras *frameworks* modernas, abstraindo-os para aplicações inteiras. Baseia-se em dois conceitos principais [41]:

- **Single spa root config:** configuração raiz que permite a renderização do *HTML* base e o *javascript* que permite o registo de outras aplicações;
- **Applications:** podem ser pensadas como aplicações do tipo SPA mas que não contem uma página HTML, capazes de se iniciar, ligar e desligar do DOM, mas sendo capazes de interagir e coexistir com outras aplicações no contexto da Single SPA root config.

As aplicações registadas consistem em três partes principais [42]:

- Nome;
- Função de inicialização da aplicação;
- Função que determina quando a aplicação deve estar ativa ou inativa.

O registo ocorre normalmente dentro da root config, como pode ser consultado no Excerto de Código 1,Excerto de Código 1 - Exemplo de Registo de aplicação Single-spa de forma a garantir hierarquia entre as aplicações. Assim as aplicações do mesmo nível serão iniciadas e desativadas de acordo com as suas próprias funções de atividade, predefinidas no registo.

```
1. registerApplication({
2. name: "@ctw/map",
3. app: () =>
4. System.import(
5. "@ctw/map"
6. ),
7. activeWhen: ['/'],
8. });
```

Excerto de Código 1 - Exemplo de Registo de aplicação Single-spa

Como se pode ver no Excerto de Código 1 na linha 2, é atribuído um 'name' que é um *Import Map Key,* que permite que a aplicação seja reconhecida por esse nome. Na linha 3 'app' que representa a aplicação a importar, esta key esta associada a URL como se pode ver no Excerto de Código 2. Na linha 7 'activeWhen' representa a rota na qual essa aplicação MFE estará visível para os utilizadores.

Param além disto as aplicações podem receber propriedades customizadas, através da key 'customProps', que podem ser utilizadas para por exemplo passar token's de acesso para aplicações filhas ou outra informação de inicialização.

Tendo em conta que devemos ter apenas um root config onde as aplicações são registadas, essas aplicações podem ser de dois tipos com finalidades diferentes [43]:

- **Parcel**: tem como objetivo permitir a reutilização de componentes de UI em diferentes aplicações. Exporta um objeto chamado parcelConfig, que pode ser utilizado por outra qualquer aplicação;
- **Utilities**: permite exportar uma interface pública de funções e variáveis que podem ser utilizadas por outros MFE. São o lugar correto para partilhar lógica comum entre as aplicações, que desta forma permite a redução da duplicação de código.

Desta forma, torna-se ainda mais fácil desacoplar aplicações maiores em partes mais pequenas com funcionalidades especificas. Para aplicações onde o fornecimento de uma elevada interatividade e existência de necessidade de log-in são pontos crucias, este tipo de abordagem revela-se promissor.

Configuração recomendada da Single spa e outras opções

Não existe uma configuração obrigatória para o ambiente de CI/CD ou de desenvolvimento local, mas naturalmente é necessário que seja definido um, para ser possível implementar uma aplicação Single spa. A documentação disponível, indica uma recomendação, que usa *in-*

browser ES modules em combinação com *import maps* (exemplo que pode ser consultado no Excerto de Código 2):

- **In-browser modules:** módulos javascript que nao são resolvidos pela ferramenta de build da aplicação, mas sim pelo browser;
- **Import maps:** são especificações que permitem aos *browsers* controlar o comportamento dos *imports javascript,* através da criação de um *bundle* de dependencias*.*

```
<script type="systemjs-importmap">
    {
      "imports": {
        "@ctw/root-config": "//localhost:9000/ctw-root-config.js",
        "@ctw/map": "//localhost:8080/ctw-map.js",
        "@ctw/info": "//localhost:8080/ctw-info.js"
      }
    }
  </script>
```

Excerto de Código 2 - Exemplo de configuração através de import maps

Para alem destas podem ser usadas opções de configuração como [41]:

- **Module federation:** técnica que recorre ao uso de webpack para partilha de módulos durante o build da aplicação. Cada fragmento é envolvido em conjunto com todas as suas dependências num módulo, inclusive as partilhadas. Ao chegar ao browser, apenas é realizado um download pois todos os restantes fragmentos podem passar a utilizar essa cópia.;
- **SystemJs:** fornece um comportamento parecido com o da recomendação indicada, com in-browser modules e import maps, mas com limitações na área de import maps. Devido a limitações do *javascript* na resolução de especificadores de importação. Por isso terá de ser compilar as aplicações no formato System.register em vez de no formato ESM.

2.4.5 Universal composition com Ara

Esta *framework* criada para integração de microfrontends, através da combinação de *universal rendering e client side rendering.* Funciona qualquer uma das *frameworks* disponíveis (React, Angular, Vue etc), isto através do uso da biblioteca Hypernova, criada pela

Airbnb. No que toca a universal rendering esta frameworks esta baseada em quatro conceitos principais [44]:

- **Nova Bindings:** permite a Hypernova carregar fragmentos construídos com qualquer *framework*;
- **Nova Directive:** este é o componente principal para a integração de micro frontends. renderiza o espaço necessário para um fragmento, para que de seguida a *nova proxy* possa solicitar a visualização do mesmo;
- **Nova proxy and middleware:** é um servidor proxy para do lado servidor que inclui fragmentos através do uso de SSI. Cria o HTML para *Nova Directive* de forma a disponibilizar os fragmentos para o Nova Cluster as poder incluir na página;
- **Nova Cluster:** é um agregador de fragmentos, que permite ao utilizador pedir e utilizar fragmentos independentemente da origem.

Quanto ao client side rendering a Ara disponibiliza uma aplicação SPA anfitriã que contem referencias para cada um dos bundles correspondente a cada micro frontend. Cada um deles será carregado de forma lazy, através do uso de um router. Esta aplicação reserva um espaço (Nova Directive) para cada um dos fragmentos, usando Nova Bridge. Esta permite a integração de Nova views em qualquer Framework como exemplo React ou Vue. Estas emitem eventos 'NovaMount' que permitem a renderização das views no browser por parte de cada micro frontend. Esta arquitetura é descrita no diagrama da Figura 20.

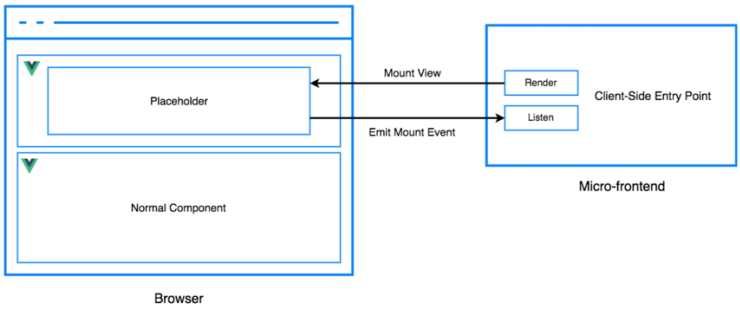

Figura 20 - Client-side rendering Ara framework

2.4.6 Resumo e Comparação de tecnologias de integração

Existem bastantes tecnologias para a criação de uma MFEA, como foi possível dissecar ao longo das seções anteriores. Com o objetivo de comparar as vantagens e desvantagens de cada uma das tecnologias foi elaborada a Tabela 5.

Os parâmetros de comparação serão os seguintes:

1. Curva de aprendizagem alta;
2. Facilidade de implementação;
3. Facilidade de Debug;
4. Permite dependências partilhadas;
5. Flexibilidade;
6. Facilidade de manutenção;
7. Boa performance;
8. Lazy Loading.

Tabela 5 - Comparação entre tecnologias MFE

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Luigi	x							
Podium	x				x		x	
Tailor	x				x		x	
Single-spa		x[7]	x	x	x			x
Ara			x	x	x		x	x
Bit			x	x	x			x

[7] Usando o CLI disponibilizado

3 Análise de Sistema

A solução existente e disponibiliza um mapa onde é possível explorar diferentes tipo de *zoom* geoespacial obtendo diferentes vistas do mesmo, e mostra eventos relativos a características da via, tráfego e outros atributos geoespaciais. Além disso representa as vias onde existe a possibilidade do uso de condução autónoma (*Road Adequacy)*, as suas características e detalhes.

Os utilizadores podem interagir com o mapa e melhorar a informação recebida, através da possibilidade de criação de eventos manuais caso necessário. Esta solução é principalmente usada por utilizadores associados à monotorização dos dados das estradas, pelo que as funcionalidades desenvolvidas são focadas na facilitação desse trabalho.

3.1 Domínio

Esta secção descreve conceitos do modelo de domínio da aplicação, para uma mais fácil perceção do restante documento.

Os conceitos de domínio são:

- Utilizador: monitorizadores de tráfego em diferentes países;
- Coordenadas: pontos geográficos no mapa, que adotam uma norma universal de Latitude e Longitude, por exemplo camara municipal do Porto (lat:41.150160082726565, -lng:8.61092627904805);
- Mapa: mapa mundo em 2D;
- Localizações: Localizações no mapa, como países, cidades ou monumentos;
- Evento: evento de tráfego ou relativo às características da via;
- Evento Draft: evento do tipo teste, pode ser tornado 'real' após submissão;
- Via: estrada ou troço de estrada no mapa;
- Tráfego: circulação/fluxo de veículos;
- Road Adequacy: coordenadas onde é possível conduzir autonomamente;
- Reasoning: razões pelas quais é possível conduzir autonomamente numa determinada estrada;

- Segmento: Pedaço de estrada com um determinada reasoning associado;
- Boundingbox: todas as localizações visíveis dentro do ecrã visível, definidos por duas coordenadas ("northEast":{ "lat":48.176388808753146, "lng":11.713829040527344 }, "southWest":{ "lat":48.09367440979962, "lng":11.450157165527346 }).

3.2 Funcionalidades

Tendo em conta os principais objetivos da solução, referidos anteriormente na secção 1.3 esta encontra-se atualmente dotada das funcionalidades enunciadas na secção 3.2.1 e funcionalidades que irão ser implementadas no futuro ou serão reimplementadas, enunciadas em 3.2.3.

3.2.1 Funcionalidades existentes

Tendo em conta os principais objetivos da solução, referidos anteriormente, esta encontra-se atualmente dotada das seguintes funcionalidades:

- UC1: Efetuar autenticação;
- UC2: Registar utilizador;
- UC3: Visualizar o mapa;
- UC4: Pesquisar localizações;
- UC5: Visualizar eventos de tráfego no mapa;
- UC6: Ver detalhes de eventos;
- UC7: Ver detalhes extra dos eventos;
- UC8: Comentar o evento;
- UC9: Ver histórico do evento;
- UC10: Ver lista de eventos ou drafts*;*
- UC11: Filtrar lista de eventos entre presentes no ecrã ou todos;
- UC12: Adicionar evento Manual;
- UC13: Editar evento;
- UC14: Clonar evento;
- UC15: Gravar novo evento;

- UC16: Guardar novo evento como draft*;*
- UC17: Filtrar eventos por tipo;
- UC18: Filtrar eventos por fonte;
- UC19: Filtrar eventos por disponível/indisponível;
- UC20: Visualizar road adequacy no mapa;
- UC21: Ver detalhes de uma *road adequacy* específica;
- UC22: Configurar diferentes mapas por país;
- UC23: Ver detalhes por segmento;
- UC24: Ver eventos num segmento.

3.2.2 Funcionalidades a implementar

As funcionalidades a implementar são:

- UC25: Ver linha temporal de alterações num dado evento.

3.2.3 Funcionalidades a reimplementar

As funcionalidades a reimplementar são:

- UC1: Autenticar utilizador;
- UC2*:* Registo de utilizadores*;*
- UC9: Ver histórico do evento;
- UC20: Visualizar road adequacy no mapa;
- UC5: Visualizar eventos de tráfego no mapa;
- UC7: Ver detalhes extra dos eventos;
- UC13: Editar Evento;
- UC15: Gravar novo evento.

3.2.4 Requisitos de qualidade

Os requisitos de qualidade são restrições definidas exteriormente ao projeto que condicionam como o projeto é desenvolvido. Existem vários modelos de caracterização, nomeadamente o FURPS+ e o ISO25010.

Para este projeto, adotando o modelo FURPS+, identificam-se os seguintes requisitos:

- **Funcionalidade**: Necessidade de autenticação de utilizadores;
- **Confiabilidade:** Necessidade do uso de AWS;
- **Suportabilidade:** Necessidade de testes unitários e testes end-to-end;
- **+ design**:
  - Necessidade de adotar uma solução desenvolvida por outra equipa;
  - Utilização de MFEA;
  - Adoção de CI/CD.

3.3 Arquitetura da solução atual

Nesta secção é descrita a atual arquitetura através da combinação de dois modelos:

- Modelo C4 [45] que defende a necessidade e vantagem de adotar diferentes graus de abstração (zoom) na representação. Os níveis propostos são [45]:
  - Nível 1 (Context ou Sistema): Contexto/imagem geral da aplicação;
  - Nível 2 (Container): Zoom no componente principal, apresentando as aplicações independentes (containers) e responsabilidades entre eles;
  - Nível 3 (Components): Novo zoom, identifica os principais blocos de construção de cada aplicação descrita no nível 2, e as suas interações;
  - Nível 4 (Code): Novo zoom para mostar o design ao nível do código/implementação.
- Modelo baseado em vistas, e.g. 4+1, que sugere a necessidade e vantagem de adotar diversas vistas complementares (e.g. vista lógica, vista de processo, vista de implementação, vista física ou de implantação e vista de cenários), e mapeamento entre elas.

As secções seguintes apresentam diagramas UML representativos das vistas arquitecturalmente relevantes até ao nível 4.

3.3.1 Nível 1

Este é o primeiro nível (Context 3.3) da arquitetura atual e é composto por três secções: vista lógica, vista de processos e vista física.

3.3.1.1 Vista lógica

A Figura 21 apresenta um diagrama UML de componentes relativos à vista lógica de nível 1 do sistema, no qual, para além do sistema se descrevem as interfaces entre o sistema e os sistemas terceiros externos:

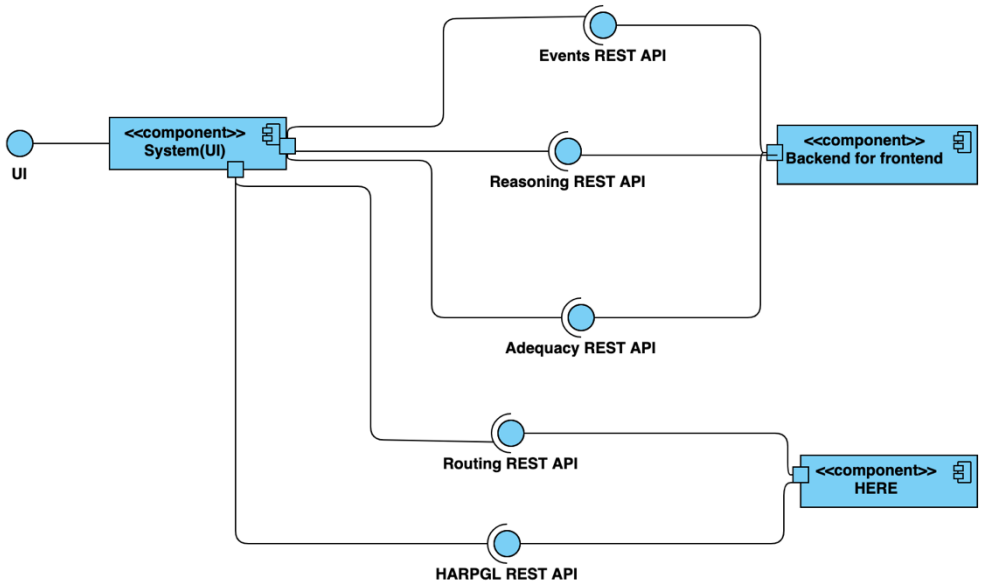

Figura 21 - Vista lógica de nível 1 da solução atual

- Backend for Frontend (BFF): serve como *middleware* para a aplicação frontend, através da conexão com micro serviços fornece os dados necessários para cumprir os requisitos funcionais da aplicação:
    - Informação geográfica relativa a eventos para mostrar no mapa;
    - Informação geográfica relativa a adequacy para mostrar no mapa;
    - Detalhes de eventos;
    - Detalhes da adequacy;
    - Listas de tipos de eventos para propósitos de filtragem.

- HERE: "Provider" do mapa do projeto e serviços relacionados com *Routing*:
    - Cálculo de rotas;
    - Pesquisa de localizações;
    - Disponibilização do mapa.

  Este serviço é pago para ser utilizado, e não é da responsabilidade da equipa.

3.3.1.2 Vista de processos

Esta secção apresenta diagramas de alguns casos de uso atuais, assim como uma breve explicação dos mesmos.

Na Figura 22 podemos ver o diagrama de processos referente ao cálculo de uma rota (extrato do UC 12) e pode ser interpretado da seguinte maneira:

1. Utilizador entra na aplicação;
2. A aplicação UI pede ao BFF via HTTPS os eventos dentro da área atual, através da Events REST API;
3. BFF pede ao HERE para calcular a boudingbox atual;
4. HERE devolve a boudingbox calculada;
5. O BFF filtra os eventos dentro dessa área;
6. O BFF devolve os eventos ao UI;
7. UI cria uma layer para esses eventos;
8. O utilizador pode ver os eventos no mapa.

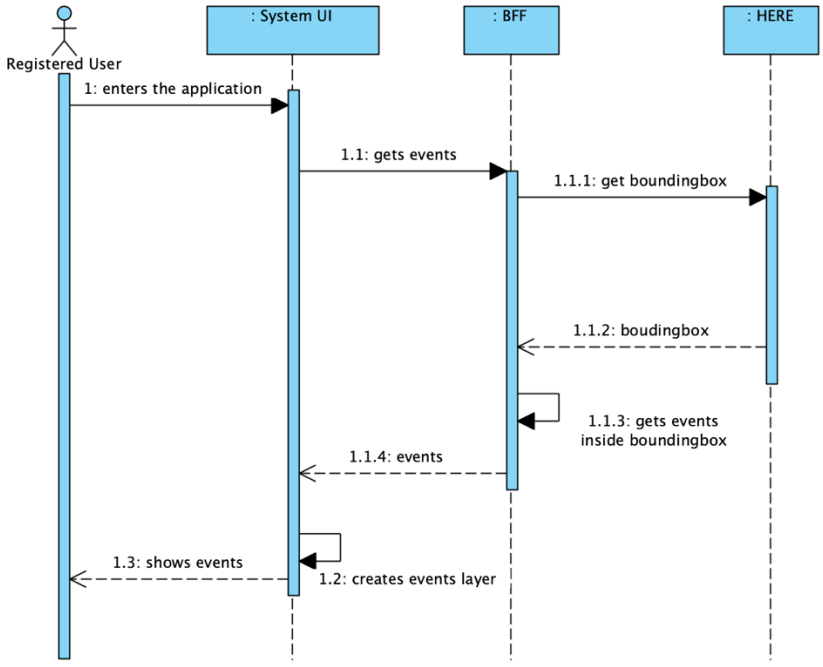

Figura 22 - Diagrama de sequência relativo ao UC5

A Figura 23 apresenta o diagrama de sequência referente ao UC5 relativo ao cálculo de uma rota:

1. O utilizador seleciona 2 pontos no mapa;
2. Via HTTPS o System UI pede ao HERE para calcular as coordenadas entre x e y;
3. Após calcular o HERE devolve a rota pretendida;
4. A rota é atualizada no mapa;
5. O mapa é apresentado ao utilizador.

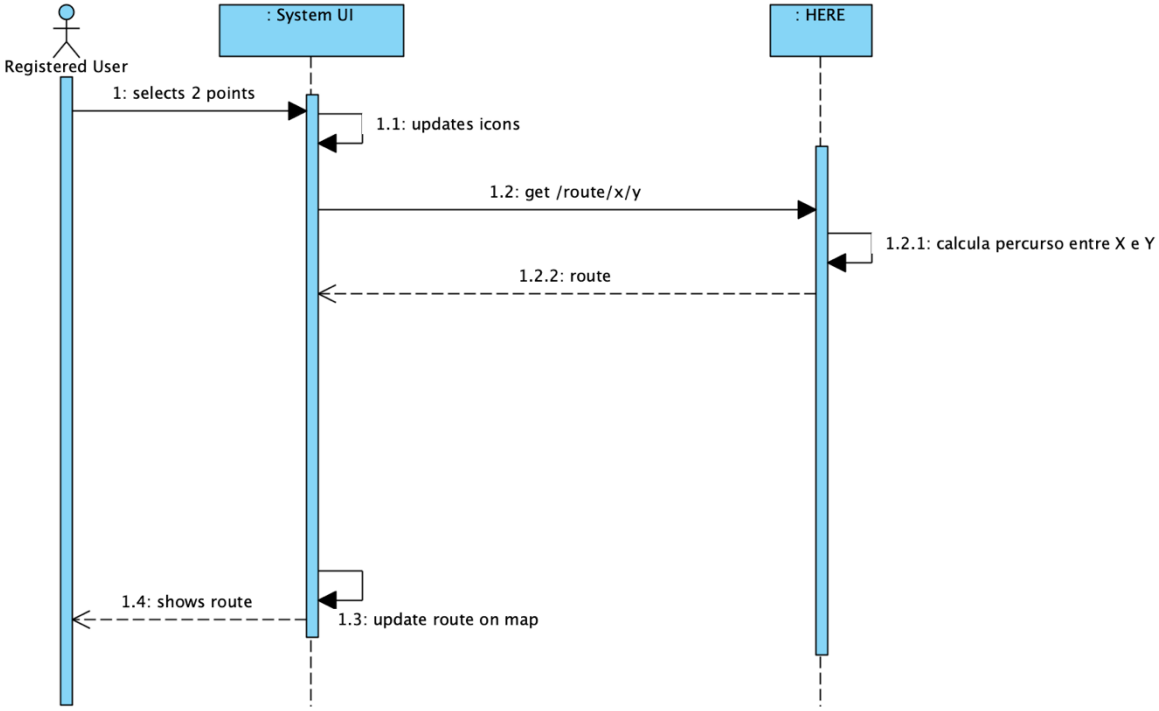

Figura 23 – Diagrama de sequência nível 1 relativo ao extrato de UC12

3.3.1.3 Vista física

A vista física descreve a implantação do software em hardware e os protocolos de comunicação adotados entre os subsistemas (3.3). No diagrama seguinte é possível consultar o diagrama de vista física de nível 1 da aplicação.

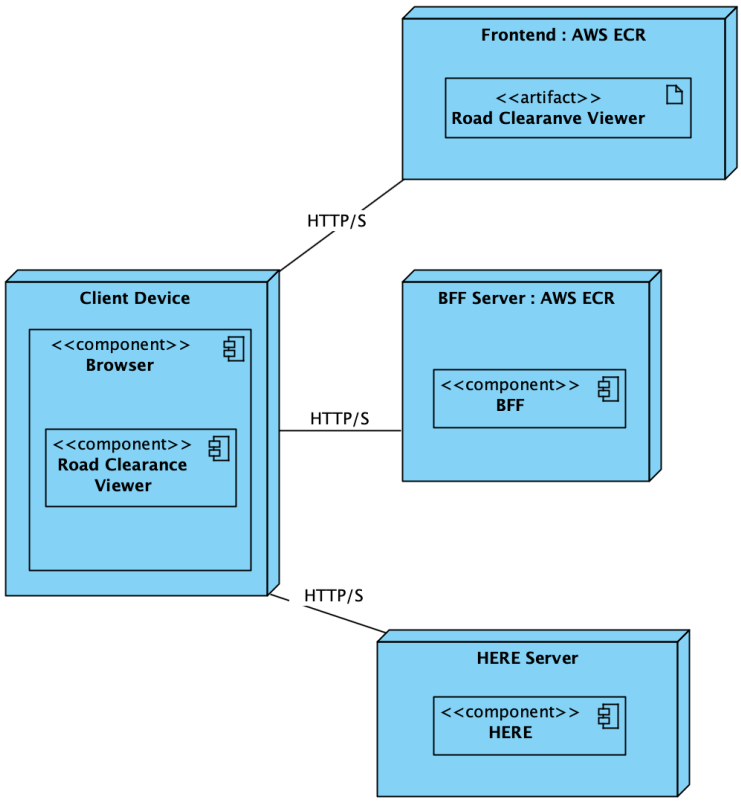

Figura 24 – Diagrama de vista física de nível 1

O processo de implantação da UI foi codificado e automatizado através de Jenkins. Sendo esse processo bem-sucedido, são corridos os testes unitários existentes. De seguida essa imagem Docker é enviada para a um Amazon Elastic Container Registry (ECR) que finalmente é implantado em máquinas virtuais no PaaS da Amazon Web Services (AWS).

3.3.2 Nível 2

Este é o segundo nível (Container 3.3) da arquitetura atual e é composto por três secções: vista lógica, vista de processos e vista de implementação.

3.3.2.1 Vista Lógica

De seguida pode ser consultado o diagrama de componentes de vista lógica de nível 2, que representa as partes que constituem o sistema:

- Components: componentes React e sua lógica associada;
- Client: é o intermediário entre os Components e o sistema BFF, através das API REST do BFF;

- Client HERE: é o intermediário entre os Components e as API REST do sistema HERE;
- DTO: Modelo de dados usado para mapear informação recebida do BFF em formato JSON ou XML, em objetos usados internamente.

Esta arquitetura permite desacoplar Components das API REST de BFF e HERE, promovendo assim a manutenibilidade.

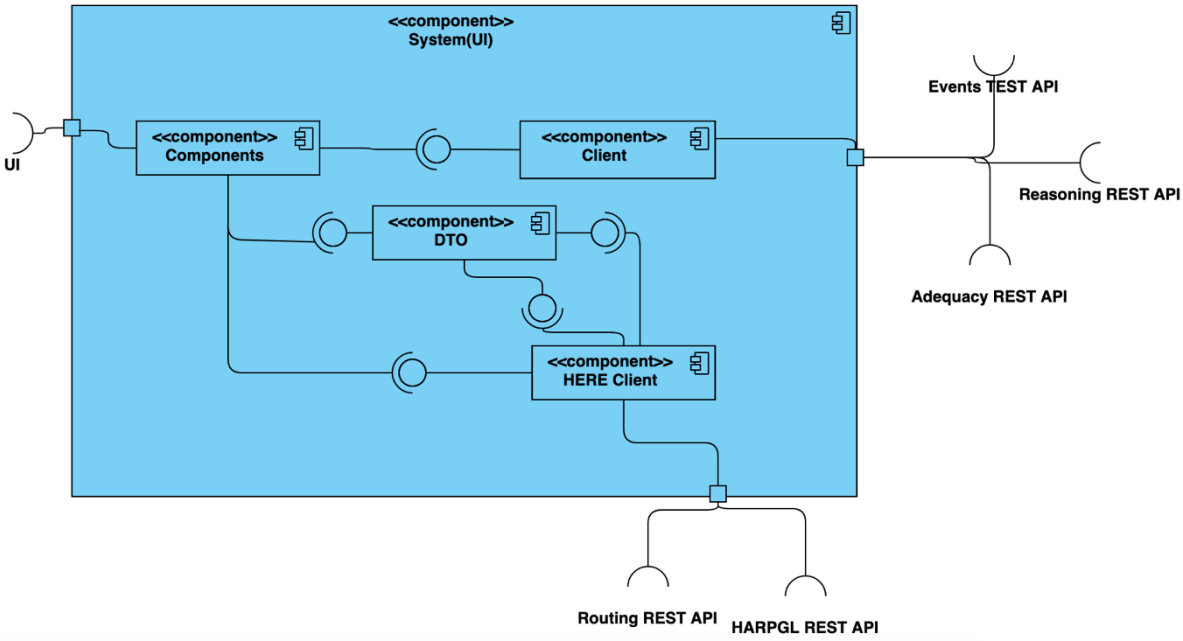

Figura 25 – Vista lógica de nível 2 da solução atual

3.3.2.2 Vista de processos

Durante esta secção serão apresentados alguns exemplos diagramas de processos relativos à solução atual, assim como uma breve explicação dos mesmos.

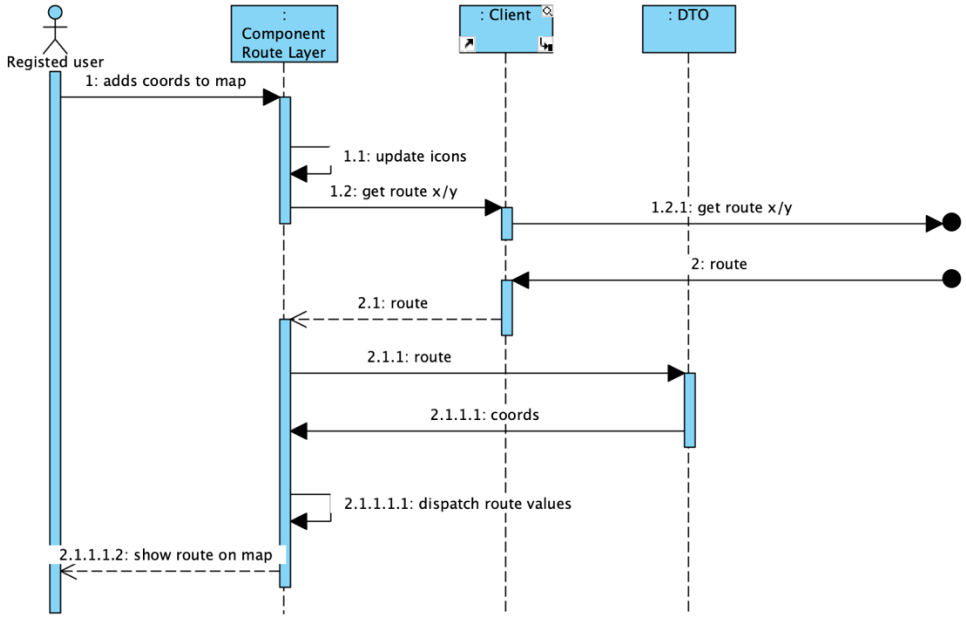

Figura 26 – Diagrama de sequência nível 2 relativo ao extrato de UC12

Na Figura 26 podemos ver o diagrama de processos referente ao cálculo de uma rota e pode ser interpretado da seguinte maneira:

1. Utilizador adiciona duas coordenadas no mapa;
2. ‘Component Route layer‘ mostra as coordenadas via icons;
3. Através de um pedido HTTPS é obtida uma string;
4. A informação recebida é convertida em coordenadas;
5. A rota é mostrada no mapa.

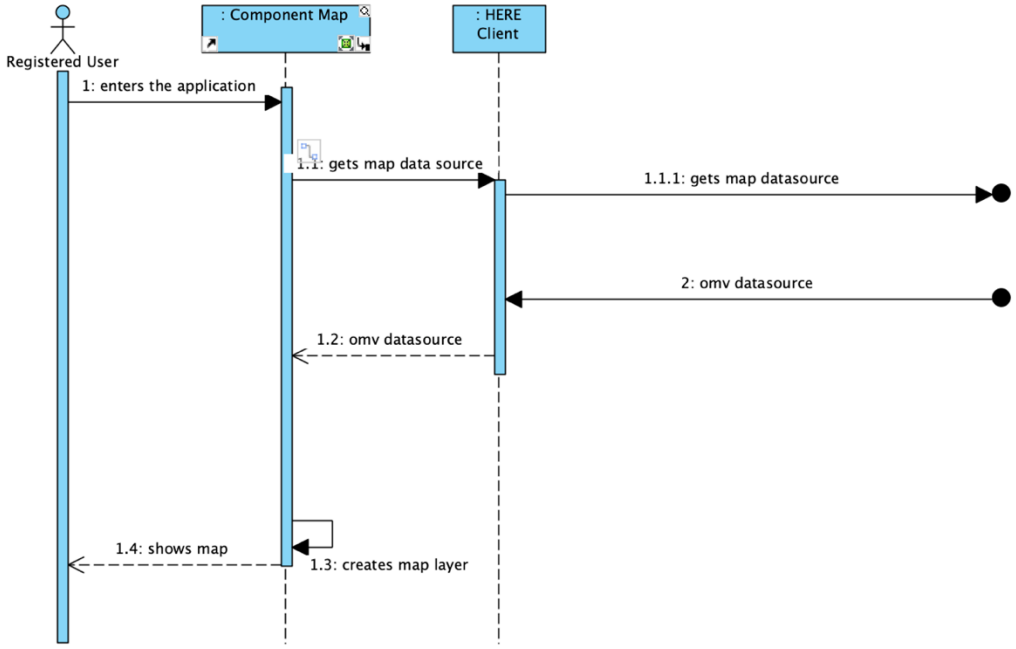

Figura 27 – Diagrama de sequência nível 2 relativo ao UC3

Na Figura 27 podemos ver o diagrama de processos referente à visualização do mapa. Pode ser interpretado da seguinte maneira:

1. O utilizador entra na aplicação;
2. O 'Component do Map' tenta obter o datasource do mapa;
3. Através do HERE Client é obtido o datasource do mapa;
4. Após a obtenção do datasource omv é criada a map layer;
5. Finalmente é possível ver o mapa.

3.3.2.3 Vista de implementação

A Figura 28 descreve a organização do projeto de implementação, que contém as pastas:

- **App:** contém componentes UI e lógica associada aos mesmos. Esta pasta será explicada em detalhe de seguida;
- **Tests:** contém os testes unitários relativos ao projeto. Atualmente o projeto apenas se encontra dotado de 60% de *code coverage*;
- **Cypress:** contém os testes end to end relativos aos fluxos existentes. Atualmente a solução tem todos os fluxos existentes testados e a correr via CI/CD;
- **Node_modules:** Contém as dependências utilizadas neste projeto.

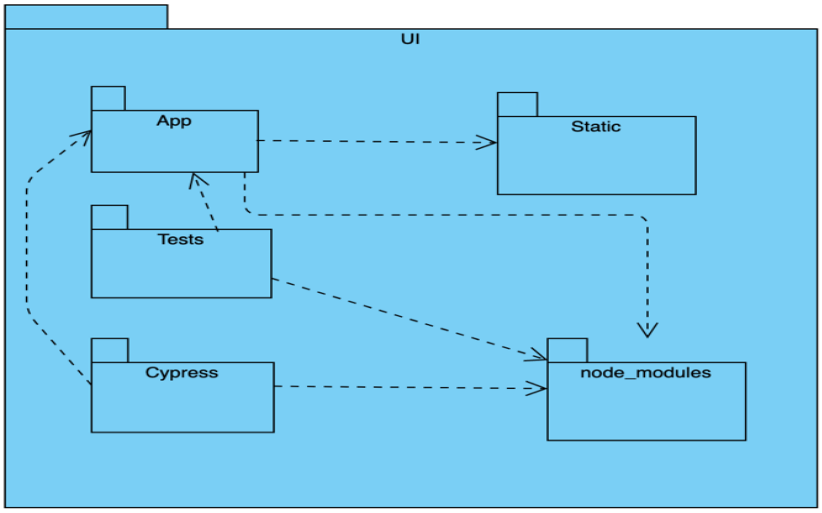

Figura 28 – Diagrama de packages nível 2

Foram utilizadas as seguintes tecnologias, descritas em mais detalhe em Anexo B. Abordagens conceptuais:

- React;
- Redux;
- Typescript;
- Sass;
- Jest;
- Enzyme;
- Material-UI.

3.3.3 Nível 3

Este é o segundo nível (Componentes 3.3) da arquitetura atual e é composto por duas secções: vista de implementação e vista de mapeamento entre vista lógica e de implementação.

3.3.3.1 Vista de implementação

Na pasta designada App, apresentada pela imagem seguinte (Figura 29), podem ser encontradas as seguintes pastas:

- **Components:** contém os componentes UI *react* e *leaflet* assim como alguma da lógica associada aos mesmos;

- **Client:** *service* que contém todos os pedidos de comunicação com o *backend*;
- **Hooks:** funções de gestão de estado e os efeitos colaterais;
- **Shared:** contém reducers (redux) para gestão de estados, gestão de erros e unções úteis e amplamente utilizadas;
- **HERE:** contém um descodificador de informação de geolocalização;
- **Config:** Ficheiros relativos a configurações, como a *store redux e* ficheiro de *logs.*

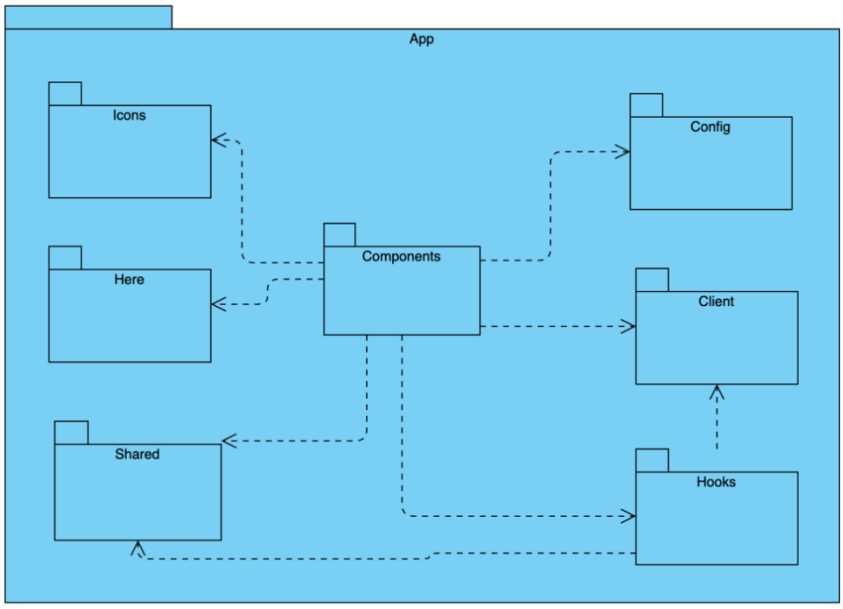

Figura 29 – Diagrama de packages nível3

3.3.3.2 Mapeamento entre vista lógica e de implementação

No diagrama da Figura 30, é possível perceber as ligações entre a vista lógica e a vista de implementação. É possível perceber que o componente lógico Components é implementado por (manifesta-se em) código existente nas pastas:

- Components;
- Hooks;
- Client.

O componente HERE Client associado às pastas:

- Here;
- Hooks.

E finalmente o componente Client associado apenas à pasta Client.

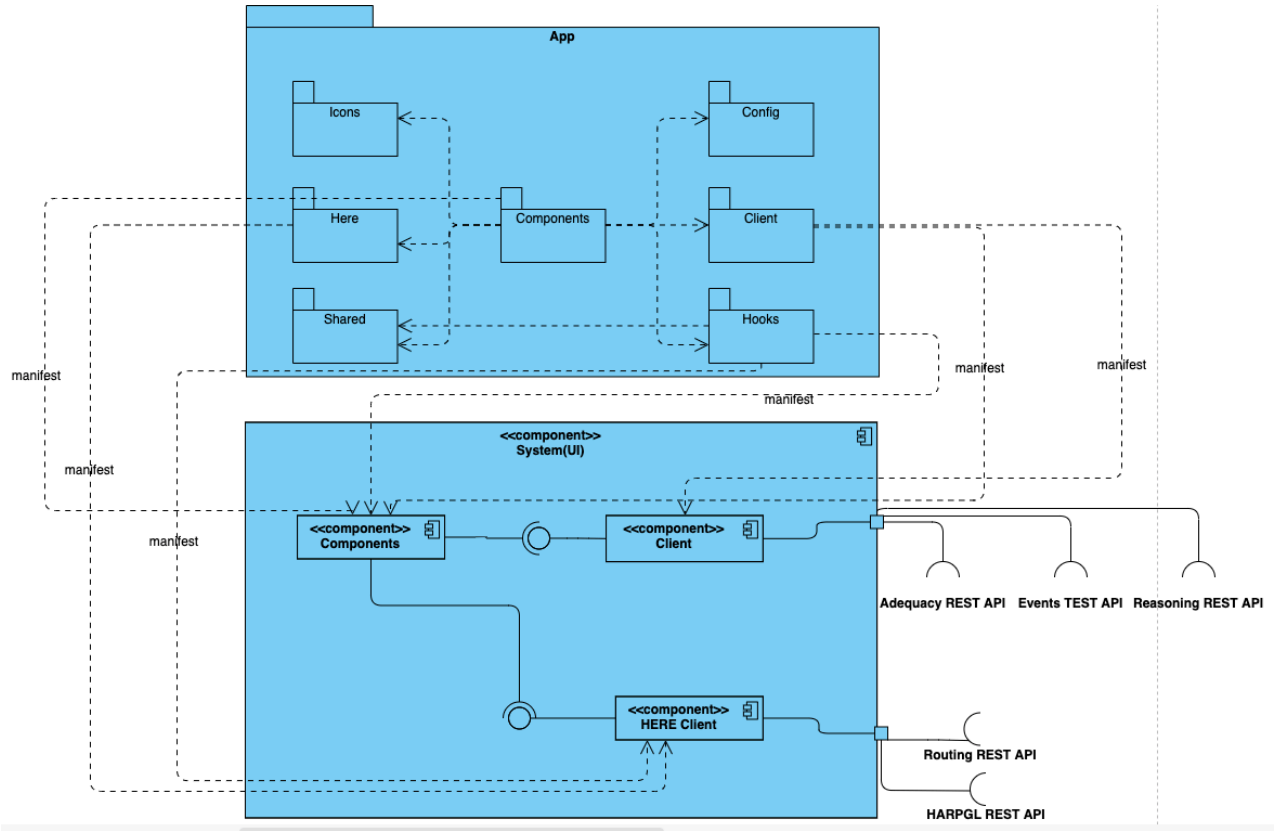

Figura 30 – Diagrama de mapeamento entre vista lógica e de implementação

3.3.4 Nível 4

Este é o segundo nível (Code 3.3) da arquitetura atual e é composto por duas secções: vista de implementação e vista de processos.

3.3.4.1 Vista de implementação

A Figura 31 descreve organização de pastas dentro de Components, assim como as dependências entre elas:

- **MapViewer:** componentes relativos à disponibilização do mapa e dos restantes componentes necessários;
- **Layout:** componentes e lógica associados à disponibilização e organização de informação no mapa;

- **Adequacy:** componentes associados à condução autónoma e a sua disponibilização do mapa;
- **Events:** componentes associados aos eventos relativos a características da via, tráfego e outros atributos geoespaciais, assim como a sua disponibilização no mapa;
- **Authtentication:** componentes relativos ao processo de registo e autenticação.

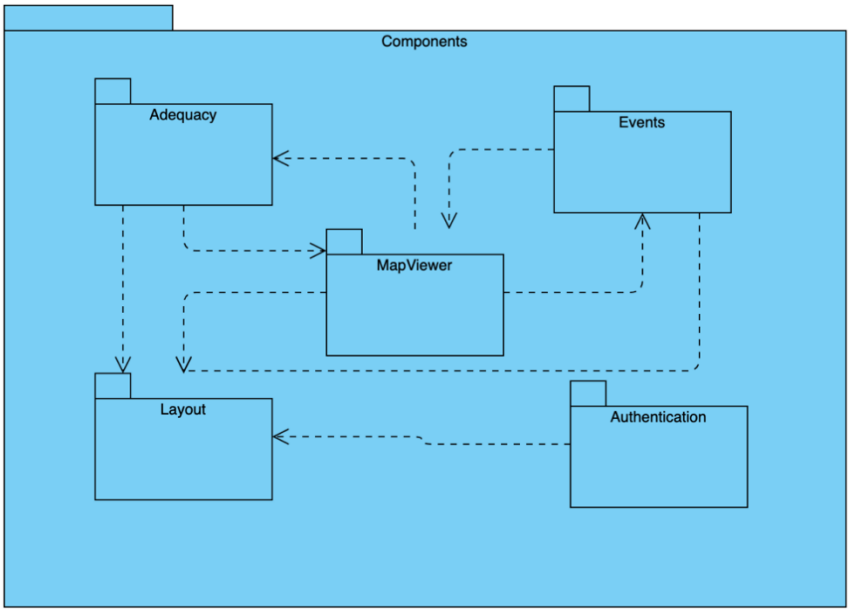

Figura 31 - Diagrama de packages (Nível 4)

3.3.4.2 Vista de processos

Na podemos ver o diagrama de processos referente à visualização de eventos no mapa.

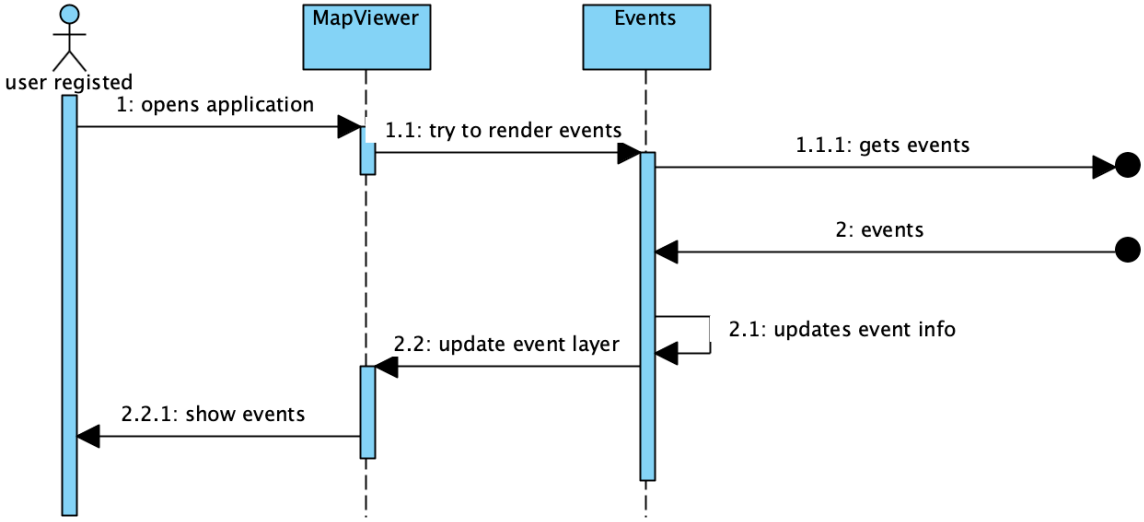

Figura 32 – Diagrama de sequência nível 4 relativo ao UC5

Pode ser interpretado da seguinte maneira:

1. Utilizador entra na aplicação:
2. 'MapViewer' tenta mostrar os eventos;
3. 'Events' espelota um pedido HTTPS para trazer os eventos;
4. Os Eventos são recebidos;
5. A informação dos eventos é atualizada;
6. Os eventos são mostrados ao utilizador.

3.4 Testes

Nesta secção serão explicados em maior detalhe os testes existentes na aplicação.

3.4.1 Unit Tests

Atualmente a aplicação encontra-se dotada de vários testes unitários que se focam nas funcionalidades principais da aplicação acima referidas (cf. 3.2.1). Para a realização destes testes foram utilizadas Jest e Enzyme como tecnologias. Estes consistem no isolamento de componentes UI e da sua lógica, de forma a ser possível testá-los de acordo com os critérios previamente definidos em cada uma das UC.

Atualmente, os testes podem ser caraterizado quantitativamente pelos seguintes indicadores:

- **Statements**: declarações existentes e testadas;
- **Branches**: possibilidades diferentes existentes no código testadas (ex: if);
- **Funções:** funções existentes testadas;
- **Lines:** linhas de código existentes e testadas.

Tabela 6 – Indicadores quantitativos dos testes unitários existentes

Statements	Branches	Functions	Lines
69.77%	50.54%	64.37%	68.9%

3.4.2 End-to-End Tests

O principal objetivo dos testes end-to-end (E2E) é testar as funcionalidades da aplicação, simulando as atividades do utilizador final, simulando um cenário real do utilizador e validando o sistema através de um teste aos seus diversos componentes, para validar a integração e integridade dos dados. Estes testes foram desenvolvidos usando Cypress.

Os casos de uso testados são:

- Detalhes da adequacy/reasoning (UC20, UC21, UC23);
- Detalhes dos eventos dentro de um segmento (UC25);
- Aplicar filtros ao mapa (UC17, UC18, UC19);
- Aplicar filtros à lista de eventos (UC11);
- Criar evento, injeção de dados (UC15, UC12);
- Criar evento *draft*, injeção de dados (UC16);
- Validar histórico dos eventos;
- Validar conteúdo dos filtros (UC11, UC17, UC18, UC19);
- Validar as funcionalidades do mapa (UC3, UC5, UC6, UC12, UC20, UC23, UC24);
- Adicionar evento, preenchimento manual (UC12);
- Validação de notificações (todos UC);
- Atualizar evento (UC13);
- Testar o zoom (sem UC);
- Testar *login* e *logout(UC1)*.

4 Desenho arquitetural

Esta seção descreve o processo de design arquitetural da aplicação, tendo em conta as alternativas estratégicas e de abordagem tática para a criação de uma nova MFEA.

Logo, serão analisadas em primeiro lugar as abordagens existentes, seguindo-se a análise de frameworks de integração e finalmente seleção das respetivas abordagens/frameworks a utilizar tendo em conta a análise/avaliação realizada.

Serão também avaliadas estratégias de migração de projetos para MFEA, de forma a avaliar a melhor maneira de aplicar a reengenharia no projeto referido na seção 3.

Finalmente será apresentado o novo desenho arquitetural adotando MFEA recorrendo a vários níveis e vista de representação.

4.1 Decisão sobre Estratégia

Tendo em conta as estratégias anteriormente explicadas e o contexto empresarial em que a aplicação pré-existente existe, referidos em 2.2 e 0 respetivamente, torna-se mais natural a adoção da combinação de estratégias *Greenfield & Big bang e Slice by Slice.* Aliando as vantagens referidas na Tabela 3 com o uso de *SCRUM* por parte da equipa, tornar-se mais o conceito de entrega contínua de valor acrescentado, pois permite que a equipa comece um projeto novo, mas que seja possível fazer entregas/implantações menores obtendo feedback do utilizador mais cedo.

4.2 Decisão sobre abordagem arquitetural e tecnologia

Para efetuar uma decisão bem fundamentada sobre a abordagem arquitetural a utilizar naturalmente é necessário ter em conta o contexto em que o projeto se insere, assim como o objetivo pretendido com o mesmo. Neste caso aplicam-se dois tipos de aplicações com objetivos diferentes:

- **Content-centric (web sites):** aplicações onde o foco principal reside na disponibilização de conteúdo;
- **Behavior-centric (web apps):** aplicações onde o foco principal é a disponibilização de uma funcionalidade.

Num contexto real é natural que não seja uma decisão binária e por isso será usada a ideologia ilustrada na Figura 33, de forma a tornar-se mais fácil perceber qual o tipo de abordagem a seguir e de seguida a tecnologia a utilizar.

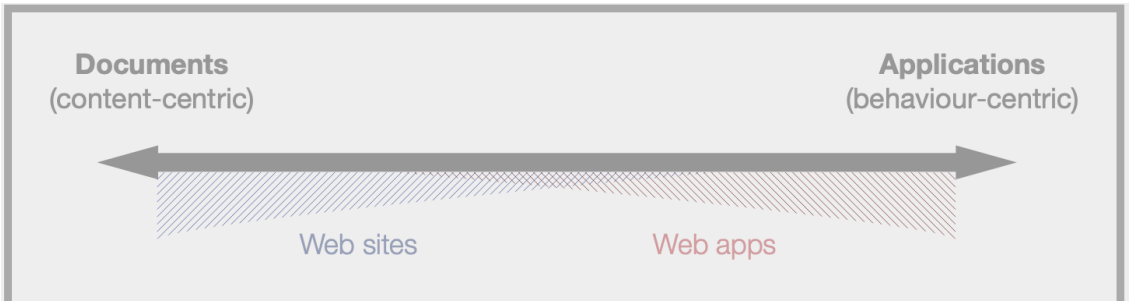

Figura 33 – Web sites vs Web apps

Sendo que a aplicação pré-existente se foca na entrega de várias funcionalidades relacionadas com a monitorização de dados sobre a condução autónoma, recai mais sobre o espectro das aplicações *behavior-centric.*

Na sequência da decisão de classificar o sistema como behaviour-centric, há que escolher a abordagem/padrão arquitetural a adotar. Neste caso, as abordagens mais indicadas seriam as Unified Single Page App (cf.2.3.3) e Universal Composition (cf.2.3.4). Após a avaliação realizada nos capítulos 2.3.6 e 2.4.6, decidiu-se tendo em conta as vantagens avaliadas em 2.3.6 e 2.4.6, utilizar Unified Single Page App e Single-Spa (cf.2.4.4) como tecnologia.

Assim sendo, foi decidido a divisão da aplicação em cinco MFE diferentes (cf.Figura 35):

- **Root config/App Shell MFE**: A configuração raiz permite a renderização do *HTML* base e o *javascript* que define/configura o registo de outras aplicações. Neste caso irá definir 2 aplicações que coexistem ao mesmo tempo na mesma rota (Map MFE e Info MFE) e outra que é usada de forma partilhada por ambas (Utilities MFE);
- **Map MFE:** Este MFE é o componente da UI responsável pela página principal que o utilizador vê e utiliza após a autenticação. Atualmente é a única página que o

utilizador verá, pois, a restante informação a ser mostrada será posicionada por cima do mapa, do lado direito da aplicação;

- **Info MFE**: Este MFE, do tipo Parcel, é responsável pela disponibilização da informação detalhada dos eventos presentes no Map MFE assim como de todo o *reasoning* por detrás das estradas onde é possível a condução autónoma;
- **Autenticação MFE**: Componente disponibilizado pela BMW e utilizado pela equipa;
- **Utilities MFE:** Este MFE foi tem o intuito de disponibilizar funções transversais à aplicação, como conversões de datas, informação geográfica, cálculos geométricos, ícones e também fontes partilhadas pela aplicação.

4.3 Descrição arquitetural

Descreve-se nesta secção dois níveis arquiteturais.

4.3.1 Nível 1

Como descrito no diagrama da Figura 34, a arquitetura de nível 1 não sofre alterações, pelo que as descrições apresentadas na secção 3.3.1 se mantêm válidas na versão orientada a MFEA.

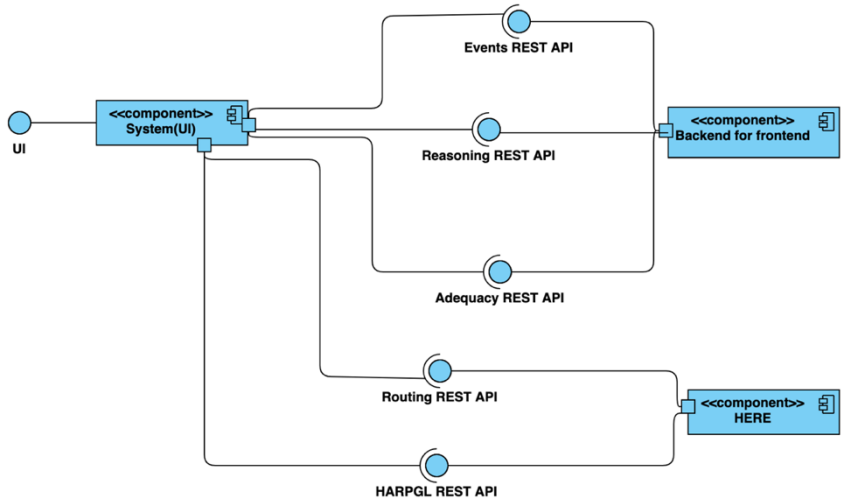

Figura 34 – Vista lógica nível 1 da solução MFE

4.3.2 Nível 2

4.3.2.1 Vista Lógica (Unified Single Page App)

Esta secção mostra a vista lógica para uma Unified Single Page App e consequente justificação.

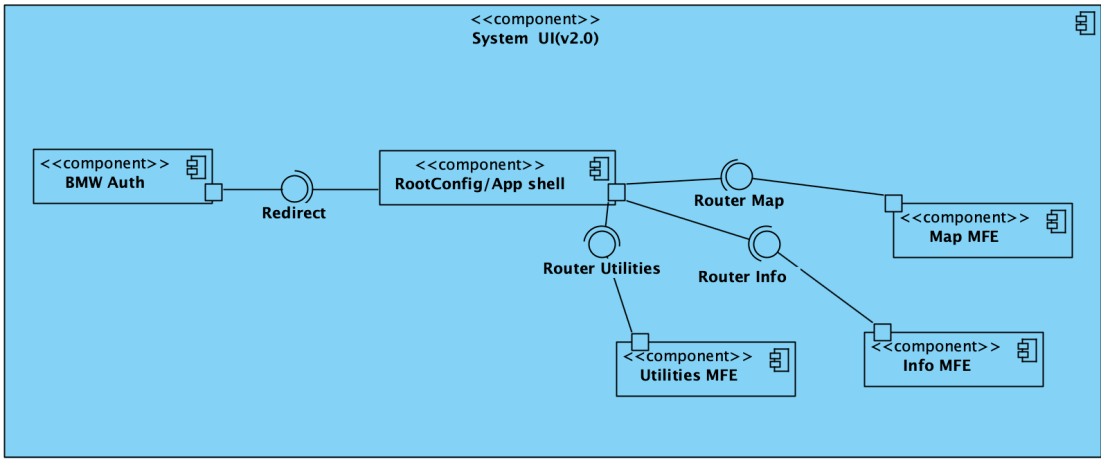

Figura 35 – Vista Lógica Unified Singel Page App

Através desta arquitetura torna-se possível isolar o mapa (Map MFE) numa aplicação própria, permitindo a sua substituição de uma forma teoricamente mais simples. Esta foi uma das principais forças de motivação da divisão da aplicação nos componentes acima representados, pois a aplicação irá ser implantada em diferentes países, nos quais será necessário a mudança de tipo de mapa ou mesmo de Map Provider. Tendo isso em conta, a aplicação MFE do mapa terá de ser capaz de decidir qual desses mapas/Map Provider utilizar. Através do uso de Single-spa usa-se uma App Shell/Root config como orquestrador dos MFE a mostrar ao utilizador.

4.3.2.2 Vista lógica (Universal Composition)

Esta secção mostra a vista lógica para Universal Composition e consequente justificação.

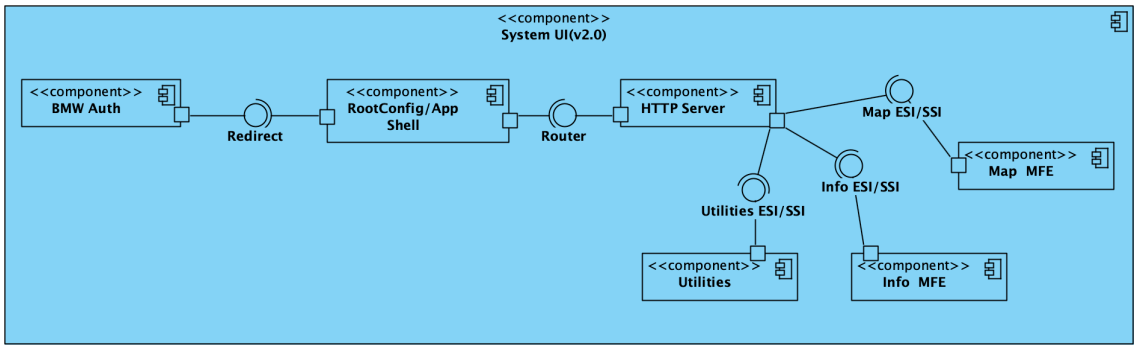

Figura 36 – Vista lógica da alternativa adotando Universal Composition

Nesta arquitetura é possível obter o mesmo isolamento do mapa que na anterior, mas nesta é adicionado um orquestrador do lado do servidor, por exemplo NGINX, para incluir os MFE através do uso de ESI ou SSI (descritos em 2.3.2.1 e 2.3.2.2 respetivamente).

4.3.2.3 Vista de processos

Esta secção foca-se nas vistas de processo de nível 2 para os casos de uso mais importantes desta arquitetura.

'Em engenharia de *software*, um requisito funcional define uma função de um sistema de *software* ou componente. O requisito funcional representa o que o *software* faz, em termos de tarefas e serviços' [46].

Com o início do desenvolvimento do *design* da aplicação, foram definidos os requisitos funcionais da mesma. Estes representam as funcionalidades chave e serão distribuídos pelos demais Casos de Uso ou *Use Case* (UC).

Durante esta secção serão explicados em maior detalhe os casos de uso mais importantes, referentes à nova arquitetura a implementar. Com casos de uso mais importantes quer-se dizer, aqueles que contêm comunicação entre MFE diferentes e consequente partilha de informação entre os mesmos. Para facilitar esta explicação foram elaborados diagramas de sequência para cada um deles.

Os casos de uso são:

- UC1: Adicionar evento;
- UC2: Carregar *boundingbox*;

- UC3: Ver detalhes de *reasoning*;
- UC4: Ver detalhes do evento;
- UC5: Mudar de range;
- UC6: Desligar/Ligar eventos;
- UC7: Desligar/Ligar eventos não bloqueadores;
- UC8: Pesquisar localização.

UC1: Adicionar evento

Na Figura 37 podemos ver o diagrama de sequência referente à adição de eventos.

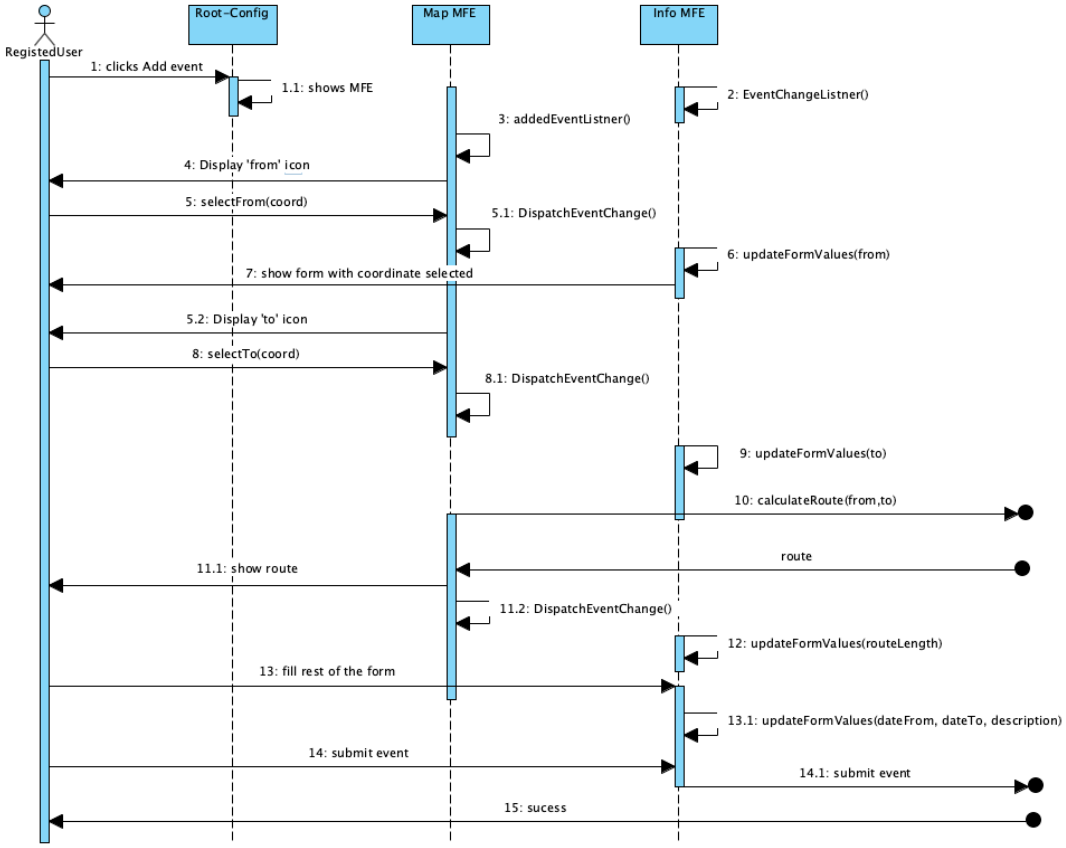

Figura 37 - Diagrama de sequência Adicionar Evento UC1

Pode ser interpretado da seguinte maneira:

1. O utilizador inicia o caso de uso;
2. O MFE Root Config mostra os MFE necessários;
3. Ambos Info MFE e Map MFE adicionam *listeners* para receber as informações que necessitam;

4. O utilizador ve o ícone de da 1ª cordenada e seleciona-a;
5. É acionado um *DispatchEventChange*;
6. Através de um *EventListener* a aplicação mostra um formulário ao utilizador já com o valor da nova coordenada;
7. O processo repete-se para a 2ª coordenada;
8. Através de um pedido HTTPS é calculada a rota selecionada;
9. A rota e mostrada ao utilizador;
10. Utilizador preenche os restantes campos;
11. Utilizador submete o evento ou grava o evento através de um pedido HTTPS.

UC2: Carregar *boundingbox*

Na podemos ver o diagrama de sequência de referente ao carregamento de eventos no mapa.

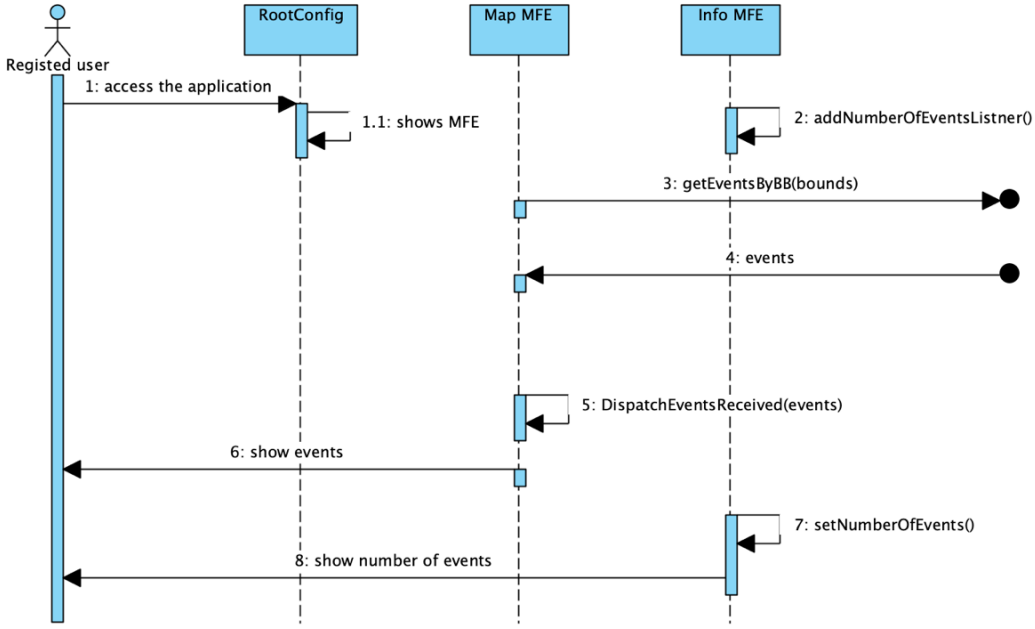

Figura 38 – Diagrama de sequência da funcionalidade Boundingbox UC2

Pode ser interpretado da seguinte maneira:

1. O Utilizador entra na aplicação;
2. O MFE Root Config mostra os MFE necessários;
3. O Info MFE adiciona um *EventListener* para receber a informação que necessita;

4. É espoletado um pedido HTTPS para trazer os eventos existentes dentro de quatro coordenadas;
5. É espoletado um *DispatchEvent* com os respetivos eventos;
6. São mostrados os eventos no mapa;
7. Através de um *EventListener* o Info MFE recebe a informação dos eventos e mostra o número de eventos existentes.

UC3: Ver detalhes de *reasoning*

Na podemos ver o diagrama de sequência referente ao carregamento do *reasoning*.

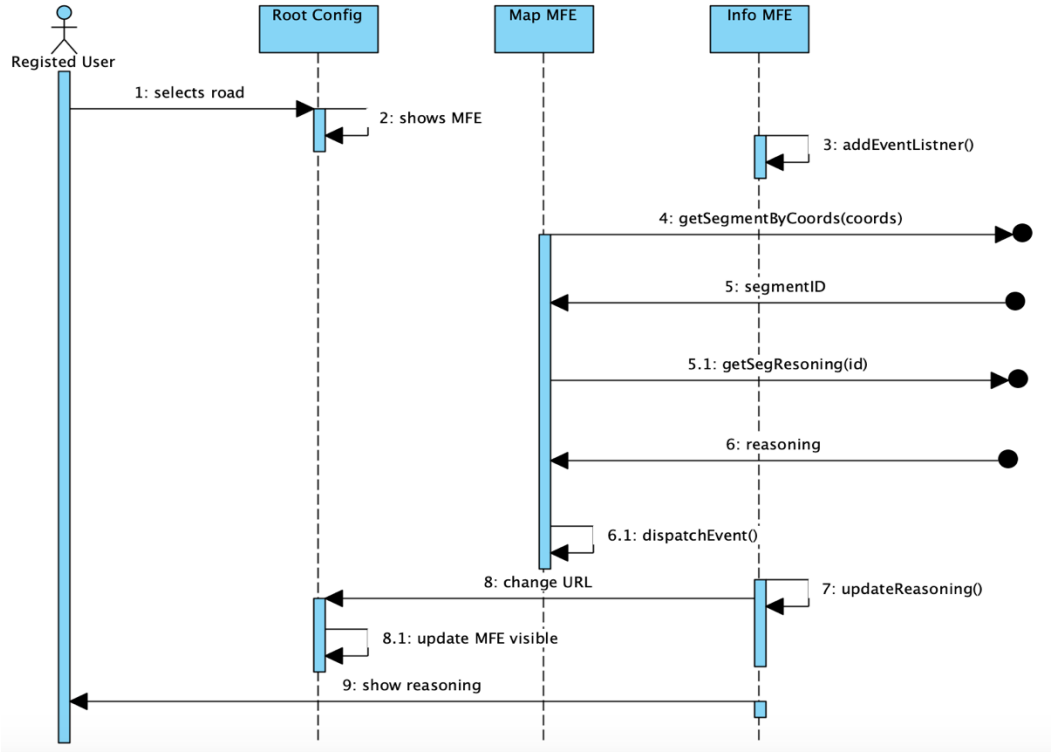

Figura 39 - Diagrama de sequência Detalhes do *reasoning* UC3

Pode ser interpretado da seguinte maneira:

1. Utilizador seleciona uma estrada;
2. O MFE Root Config mostra os MFE necessários;
3. É espoletado um pedido HTTPS que devolve o id do segmento;
4. É espoletado um pedido HTTPS que devolve o *reasoning* desse segmento;
5. Após o pedido é espoletado um *DispatchEvent* com a informação do segmento;

6. Através de um EventListener essa informação é recebida pelo Info MFE e mostrada ao utilizador após a mudança do URL.

UC4: Ver detalhes do evento

Na podemos ver o diagrama de sequência de referente ao carregamento dos detalhes de um evento.

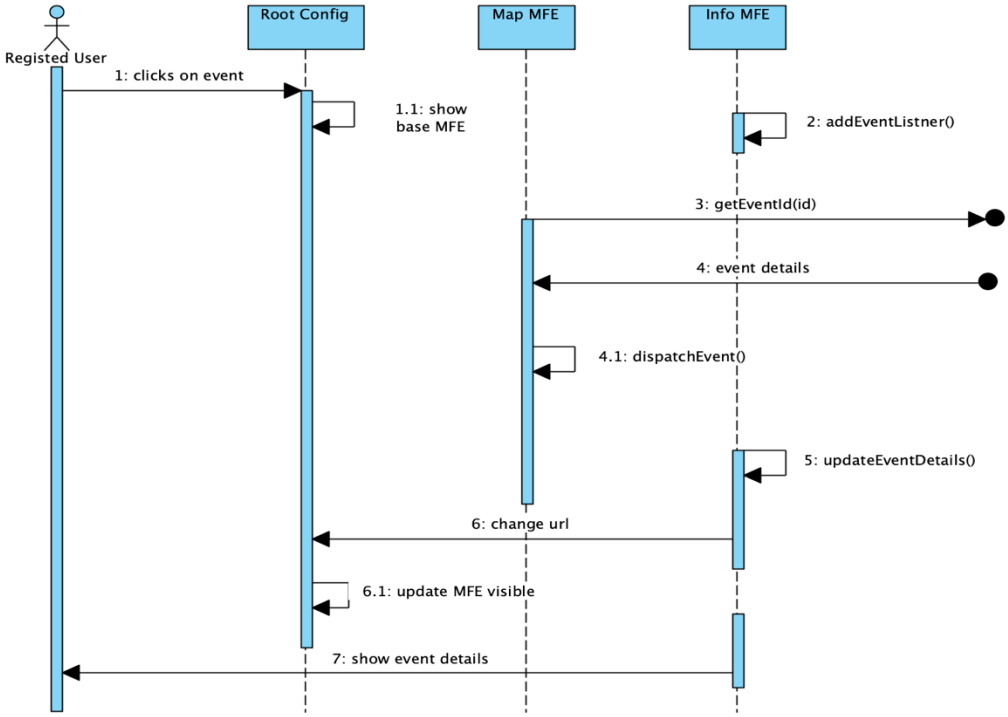

Figura 40 – Diagrama de sequência Detalhes do Evento UC4

Pode ser interpretado da seguinte maneira:

1. O utilizador seleciona um evento visível no mapa;
2. É espoletado um pedido HTTPS referente à busca da informação do evento;
3. Após o pedido é espoletado um DispatchEvent com a informação do evento;
4. Através de um EventListener a informação do evento é recebida pelo Info MFE e mostrada ao utilizador.

UC5: Mudar de range

Na podemos ver o diagrama de sequência de referente à mudança de range dentro de um segmento.

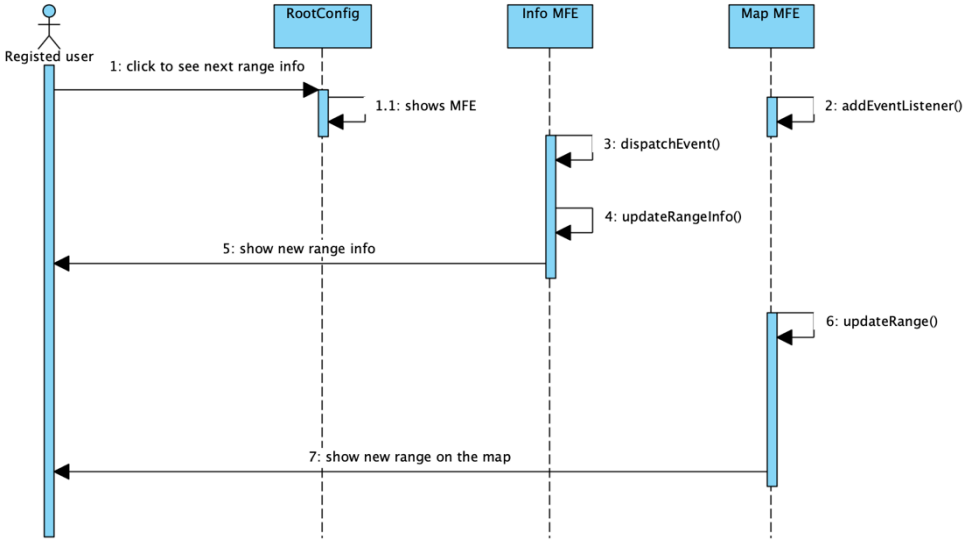

Figura 41 - Diagrama de sequência de mudança de range UC5

Pode ser interpretado da seguinte maneira:

1. Utilizador seleciona o range seguinte;
2. Após é espoletado um DispatchEvent com a informação do range selecionado e atualizada a informação do range;
3. O utilizador pode ver a nova informação;
4. Através de um EventListener a informação do range é recebida pelo Map MFE e mostrada ao utilizador no mapa.

UC6: Desligar/Ligar eventos

Na podemos ver o diagrama de sequência de referente à remoção de eventos do mapa.

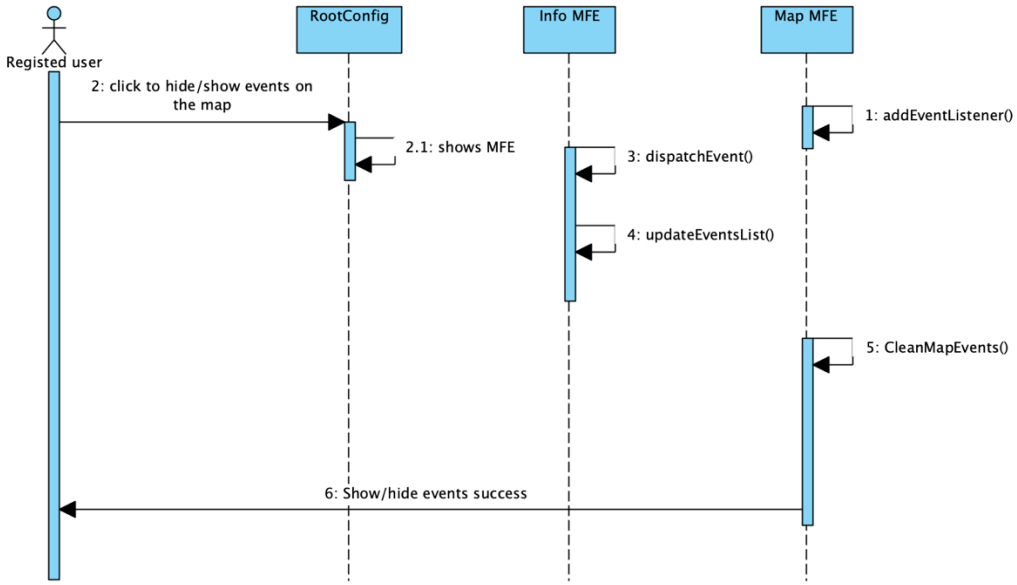

Figura 42 - Diagrama de sequência desligar eventos UC6

Pode ser interpretado da seguinte maneira:

1. O utilizador desliga/liga os eventos através de um click;
2. É espoletado um DispatchEvent relativo ao toggle de eventos do mapa;
3. Através de um EventListener o mapa sabe que deve remover os eventos visíveis;
4. Utilizador deixa de ver/vê os eventos no mapa.

UC7: Limpar/mostrar Eventos não bloqueadores

Na podemos ver o diagrama de sequência de referente ao *toggle* de eventos não bloqueadores do mapa.

Pode ser interpretado da seguinte maneira:

1. O utilizador desliga/liga os eventos não bloqueadores através de um click;
2. É espoletado um DispatchEvent relativo ao toggle de eventos bloqueadores do mapa;
3. Através de um EventListener o mapa sabe que deve remover/mostrar os eventos não bloqueadores visíveis;

4. Utilizador deixa de ver/vê os eventos não bloqueadores no mapa.

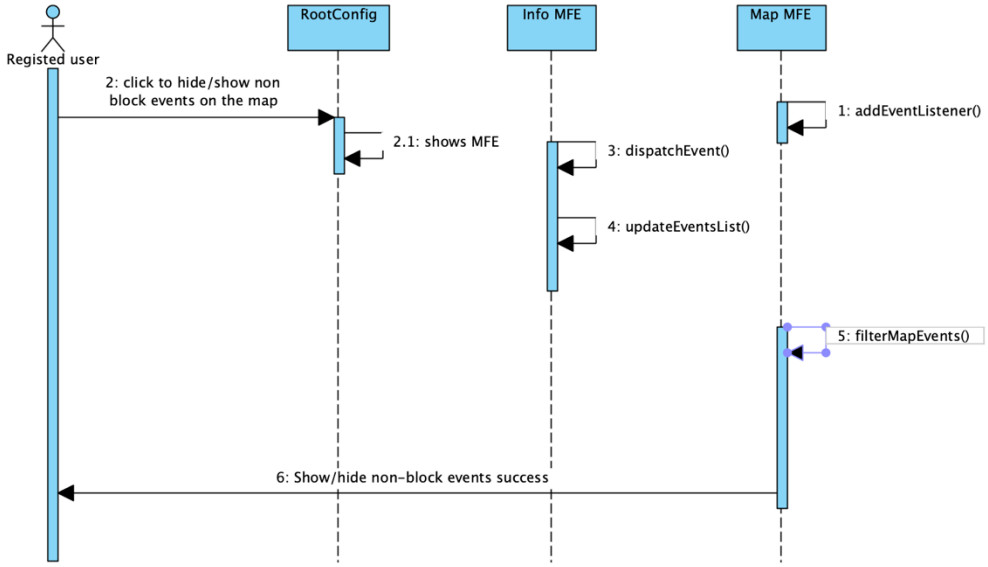

Figura 43 - Diagrama de sequência limpar/mostrar eventos nao bloqueadores UC7

UC8: Pesquisa de nova localização

Na podemos ver o diagrama de sequência de referente à pesquisa de novas localizações.

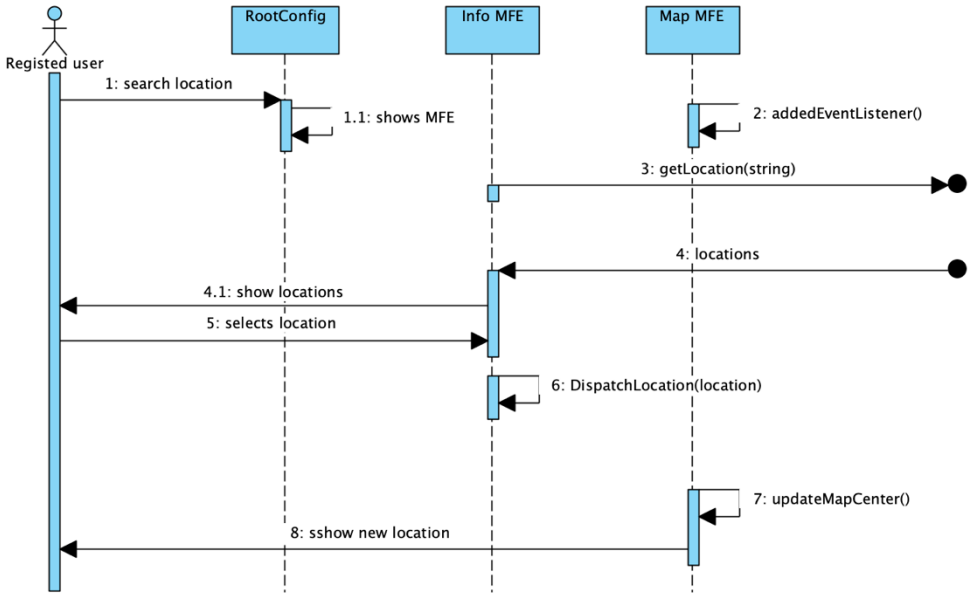

Figura 44 - Diagrama de sequência pesquisa de localização

Pode ser interpretado da seguinte maneira:

1. O utilizador pesquisa uma nova localização;
2. É espoletado um pedido HTTPS referente à busca de novas localizações;
3. Após o pedido, é mostrado ao utilizador uma lista de localizações;
4. O utilizador seleciona a localização pretendida;
5. É espoletado um DispatchEvent com as coordenadas da nova localização;
6. Através de um EventListener as coordenadas da localização são recebidas pelo Map MFE;
7. O Map MFE muda a localização para a selecionada pelo utilizador.

5 Implementação

Este capítulo descreve a implementação da prova de conceito do design descrito e selecionado no capítulo anterior, bem como a seleção da tecnologia adotada.

5.1 Tecnologia utilizada

Tendo em conta a avaliação feita na secção 2.4.6, foi selecionado para o desenvolvimento a criação e uma Unified Single Page Application através do uso de Single Spa (2.4.4). Para além desta foram utilizadas as seguintes tecnologias:

- React;
- Jest;
- Enzyme;
- Typescript;
- SCSS;
- Webpack;
- Leaflet;
- Import maps e Systemjs;
- Material-UI;
- Cypress.

Estas tecnologias visam ir de encontro às necessidades do projeto que foram descritas ao longo do documento, como também nos conhecimentos pré-existentes da equipa de forma a facilitar a migração e manutenção futura do mesmo.

5.2 Micro Frontend criados

Como pode ser consultado na Figura 35 na secção 4.3.1, a nova solução divide-se em cinco aplicações MFE distintas, onde apenas quatro são controladas pela equipa. Estes MFE criados estão associados a uma *Import Map Key* que permite o uso dos respetivos microfrontends. As keys associadas são as seguintes:

- Map MFE: @ctw-leaflet-map;
- Info MFE: @ctw-info-displayer;
- Utilities MFE: @ctw-utilities;
- Root Config: @ctw-root-config.

Como já foi referido, atualmente todos os MFE criados são geridos apenas por apenas uma equipa, mas futuramente, podem ser distribuídos por diferentes equipas de forma a distribuir responsabilidades e não tornar impossível a manutenção por parte da equipa atual.

5.2.1 Root-Config MFE

A configuração raiz permite a renderização do *HTML* base e o *javascript* que define/configura o registo de outras aplicações. Neste caso irá definir 2 aplicações que coexistem ao mesmo tempo na mesma rota (Map MFE e Info MFE) e outra que é usada de forma partilhada por ambas (Utilities MFE). A app Info MFE contem *routing* interno, de forma a ser possível mostrar diferentes vistas.

```
<script type="systemjs-importmap">
    {
      "imports": {
        "@ctw/root-config": "//localhost:9000/ctw-root-config.js",
        "@ctw/leaflet-map": "//localhost:8080/ctw-leaflet-map.js",
        "@ctw/info-displayer": "//localhost:8081/ctw-info-
displayer.js",
       "@ctw/utilities": "//localhost:8082/ctw-utilities.js",

      }
    }
  </script>
```

Excerto de Código 3 - Configuração de MFE via import maps

```
registerApplication({
name: "@ctw-leaflet-map",
app: () =>
System.import(
""@ctw-leaflet-map"
),
activeWhen: ['/'],
});
```

Excerto de Código 4 - Registo da app Map MFE

```
name: "@ctw-info-displayer",
app: () =>
System.import(
"@ctw-info-displayer"
),
activeWhen: ['/'],
});
```

Excerto de Código 5 - Registo da app Info MFE

```
name: "@ctw-utilities",
app: () =>
System.import(
"@ctw-utilities"
),
activeWhen: ['/'],
});
```

Excerto de Código 6 - Registo da app Utilities MFE

5.2.2 Map MFE

Este MFE é o componente responsável pela página principal que o utilizador vê e utiliza. A restante informação a ser mostrada, será posicionada por cima do mapa, do lado direito da aplicação. Como pode ser consultado na Figura 45, o mapa irá conter dois botões que permitirão ao utilizador fazer zoom no mapa. No Excerto de Código 7 é possível consultar o código necessário para configurar o mapa.

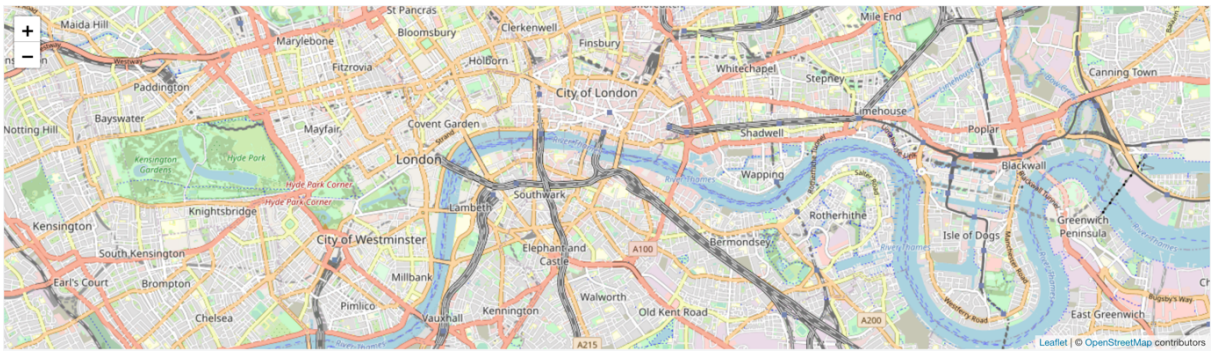

Figura 45 – Map MFE Leaflet Map

```
<MapContainer center={position} zoom={13} scrollWheelZoom={false}>
    <TileLayer
      attribution='&copy; <a
href="http://osm.org/copyright">OpenStreetMap</a> contributors'
      url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
  </MapContainer>
```

Excerto de Código 7 - Código de configuração do mapa

Esta aplicação será também responsável por apresentar os diferentes tipos de eventos de tráfego e caraterísticas da via no mapa assim como as zonas onde os carros poderão conduzir autonomamente como pode ser consultado nas Figura 46 e Figura 47 respetivamente. Ambos os tipos de informação terão ainda detalhes adicionais, que podem ser consultados através de um *click* no respetivo componente *leaflet* que os caracteriza. O código que permite a configuração destes eventos pode ser consultado no Excerto de Código 8 e o código de configuração da adequacy no Excerto de Código 9.

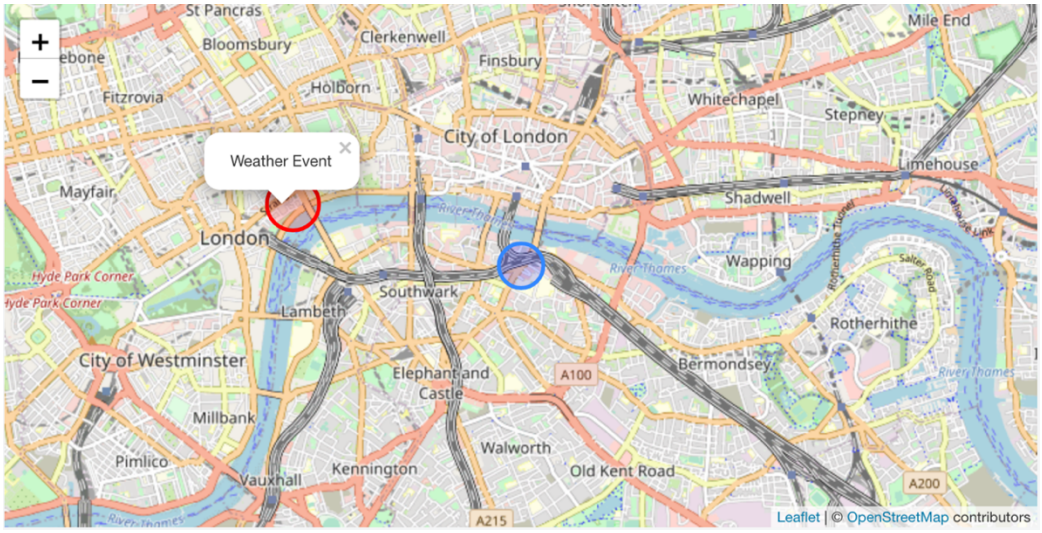

Figura 46 - Eventos no mapa

```
<Circle center={center} pathOptions={fillBlueOptions} radius={200} />
    <CircleMarker center={position} pathOptions={redOptions}
radius={20}>
      <Popup>{typeOfEvent}</Popup>
    </CircleMarker>
```

Excerto de Código 8 - Código da configuração dos eventos

```
<Polyline pathOptions={fillBlueOptions} positions={polyline} />
```

Excerto de Código 9 - Código de configuração e adequacy

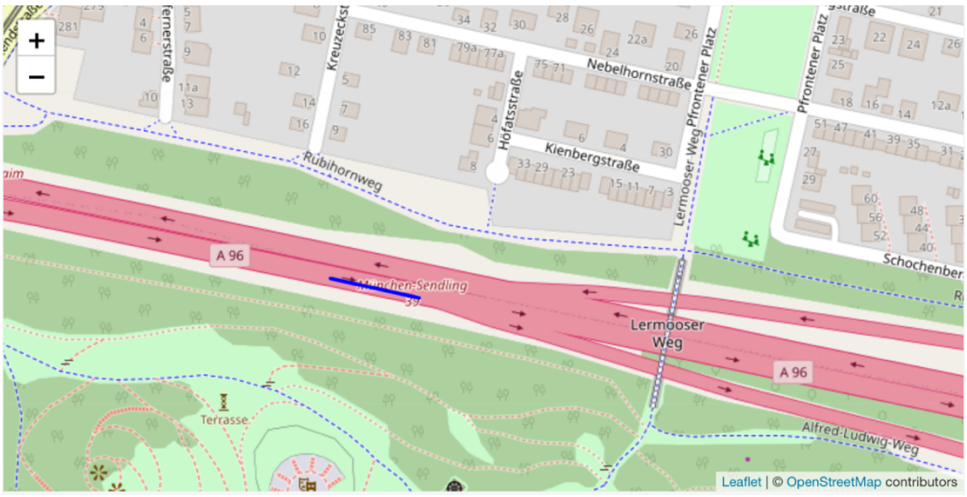

Figura 47 - Troço de estrada adequado à condução autónoma

5.2.3 Info MFE

Este MFE, do tipo Parcel, é responsável pela disponibilização da informação detalhada dos eventos referidos no MFE anterior: os detalhes relativos a eventos como de todo o *reasoning* por detrás das estradas onde é possível a condução autónoma. Toda esta informação é mostrada ao utilizador através de um componente Material-UI customizado, posicionado do lado direito da aplicação, como pode ser consultado na Figura 48 e 49 respetivamente.

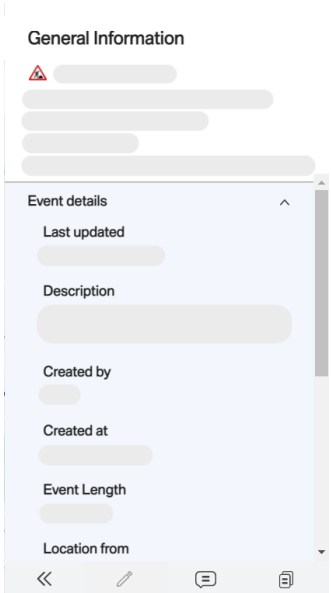

Figura 48 - Aba detalhes dos eventos

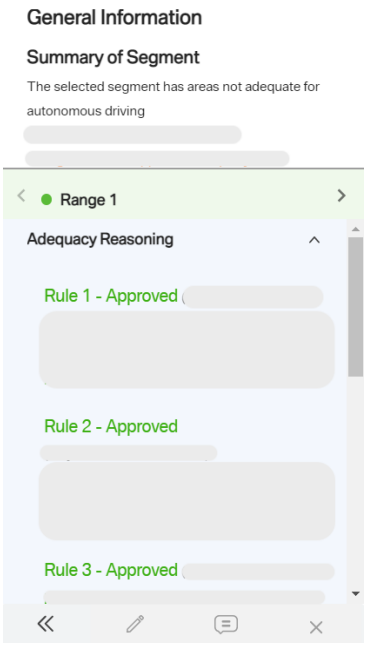

Figura 49 - Aba de detalhes da adequacy reasoning

Para além destas abas laterais, este MFE é também responsável pela renderização de uma top bar que mostrará informação adicional, como o número total de eventos visíveis no mapa. Disponibiliza também opção de pesquisa de localizações, a opção de desligar eventos, de apenas ver eventos bloqueadores de *adequacy* e disponibiliza o botão que permite adicionar novos eventos, como pode ser consultado na Figura 50.

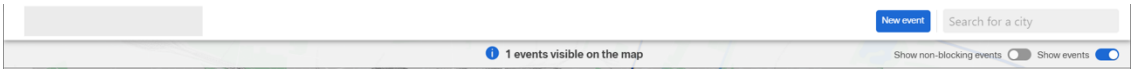

Figura 50 - Top bar Info MFE

No caso de utilizador querer adicionar um novo evento, através do respetivo botão, a aplicação mostrará uma aba diferente. Disponibiliza um formulário ao utilizador que lhe permite a adição de um novo evento, como pode ser consultado na Figura 51.

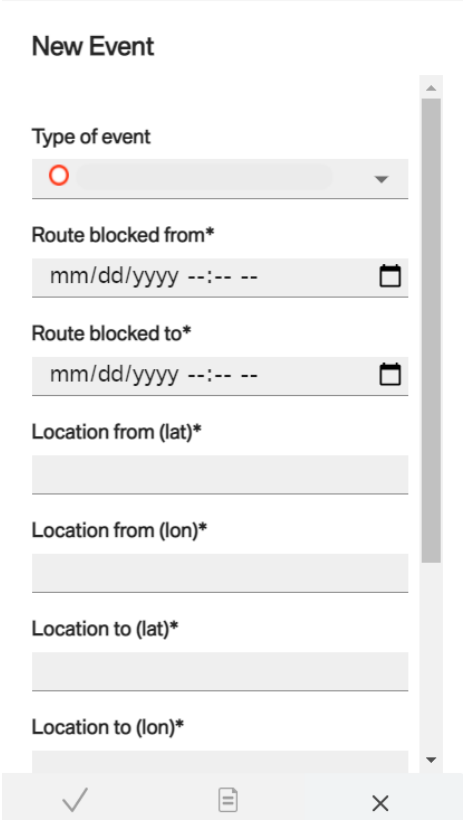

Figura 51 - Formulário para adição de novos eventos

Para adicionar um novo evento, o utilizador apenas precisa de preencher os campos obrigatórios e de seguida submeter a informação.

5.2.4 Utilities MFE

Este MFE foi criado com o intuito de disponibilizar funções transversais à aplicação, como conversões de datas, informação geográfica, cálculos geométricos, ícones e também fontes partilhadas pela aplicação. Esta aplicação é do tipo Utilities como referido em 2.4.4, de forma a partilhar as funções pelos restantes MFE criados.

```
1. export const formatDateTimeForEventForm = (d?:string) =>
   moment(d).format('YYYY-MM-DDTHH:mm')
```

Excerto de Código 10 - Exemplo de utilities

5.3 Testes

Como referido em 3.4, o monólito encontra-se dotado de vários testes unitários e testes end-to-end. Isto e a divisão do projeto em MFE, permitiu que se procedesse a uma migração dos mesmos, de uma forma faseada, para os respectivos MFE.

5.3.1 Testes unitários

Na nova solução, os testes foram divididos consoante o MFE para onde o(s) componente(s) que estes testavam, foram mudados. Permitindo continuar a ser possível testá-los de acordo com os critérios previamente definidos, mas dentro do respetivo MFE. O foco dos testes manteve- nas funcionalidades principais dos MFE acima referidas, como os detalhes de eventos e reasoning. Para a realização destes testes mantiveram-se as tecnologias Jest e Enzyme.

Desta forma foi possível manter a cobertura existente previamente (69%), visto que os testes existentes já se focavam nas funcionalidades referidas. Apenas deixando sem testes a aplicação ctw-utilities e ctw-root-config.
A primeira porque esta planeado pela equipa realizar estes testes no futuro e a segunda porque existe como configuração dos restantes MFE, não necessitando de testes unitários.

5.3.2 Testes end to end

O monólito encontra-se dotado de vários testes end-to-end, realizados usando Cypress, como já foi referido em 3.4.2. Estes testam todos os fluxos principais na aplicação e estes foram movidos para aplicação ctw-root-config. Destes, foram retirados os relativos ao login e logout e de filtragem, de forma a ser possível corrê-los com toda a aplicação a funcionar, visto necessitar de mais que um MFE paras ser possível testar os fluxos.

Os testes existentes já foram enumerados em 3.4.2, mas para facilitar a leitura do documento serão enumerados de novo nesta secção:

- Detalhes da adequacy/reasoning;
- Detalhes dos eventos dentro de um range;
- Aplicar filtros ao mapa;

- Criar evento, injeção de dados;
- Criar evento *draft*, injeção de dados;
- Validar histórico dos eventos;
- Validar conteúdo dos filtros;
- Validar as funcionalidades do mapa;
- Adicionar evento, preenchimento manual;
- Validação de notificações;
- Atualizar evento;
- Testar o zoom.

6 Avaliação e Resultados

Este capítulo tem como objetivo esclarecer o processo de avaliação de resultados relativos a este caso de estudo. Dividindo-se em secções que especificam a hipótese, identificam os indicadores e fontes de informação e finalmente a descrição da metodologia de avaliação utilizada.

6.1 Hipótese

Este caso de estudo específico tem como objetivo a averiguação da capacidade de MFEA resolver os problemas identificados em 1.2 em particular na reengenharia do software descrito.

6.2 Identificação de indicadores e fontes de informação

Nesta secção serão enunciadas grandezas que permitam avaliar o resultado da utilização de MFEA. As grandezas que serão utilizadas são as seguintes:

- **Satisfação do cliente:** tem como objetivo analisar se o cliente sente melhorias na solução provenientes da alteração arquitetural;
- **Satisfação da equipa:** tem como objetivo analisar a satisfação da equipa perante a nova arquitetura;
- **Qualidade de código:** tem como objetivo avaliar se a nova arquitetura contribui para o aperfeiçoamento do código gerado e adoção de boas práticas;
- **Performance da solução:** tem como objetivo avaliar a diferença da performance da aplicação, com a nova arquitetura;
- **Deteção de erros:** tem como objetivo a medição da facilidade de deteção da origem dos erros e correção dos mesmos;
- **Facilidade de desenvolvimento:** tem como objetivo medir a facilidade de implementar novas funcionalidades na solução;
- **Facilidade de manutenção:** tem como objetivo medir a facilidade da equipa em aplicar manutenção na solução.

Para cada uma destas grandezas são esperados os seguintes resultados:

- **Satisfação do cliente:** espera-se que o cliente se sinta satisfeito com a aplicação desenvolvida com a nova arquitetura, e que não sinta nenhum impacto negativo na solução frontend;
- **Satisfação da equipa:** espera-se um aumento da satisfação da equipa perante a nova arquitetura, assim como uma assimilação e adoção da mesma de forma a obter uma maior produtividade e foco na entrega ao cliente;
- **Qualidade de código:** espera-se um aumento da qualidade do código existente, um maior desacoplamento, separação de conceitos e qualidade da solução no geral;
- **Performance da solução:** espera-se que, no mínimo, a solução mantenha a performance esperada pelo cliente atualmente, com possibilidade do aumento da mesma através do aumento da qualidade de código e redução do tamanho dos módulos;
- **Deteção de erros:** espera-se um aumento da facilidade da deteção da origem de erros na solução por parte da equipa;
- **Facilidade de desenvolvimento:** espera-se um aumento da facilidade de implementação de novas funcionalidades por parte da equipa de desenvolvimento e uma maior facilidade de gestão da fase desenvolvimento pelo SCRUM Master;
- **Facilidade de manutenção:** espera-se um amento na facilidade de manutenção da solução produzida por parte da equipa de desenvolvimento.

6.3 Metodologia de avaliação

6.3.1 Inquéritos de satisfação

Para a avaliar as grandezas de ordem subjetiva serão utilizados inquéritos de satisfação. Para este projeto em concreto, que envolve a alteração de uma arquitetura, a equipa, o cliente e restantes intervenientes no processo serão quem irá responder (anonimamente) a esses inquéritos. Estes questionários serão compostos por várias perguntas e cinco tipos de resposta possíveis, sendo estes os seguintes [47]:

1. Discordo completamente;
2. Discordo;

3. Neutro;
4. Concordo;
5. Concordo completamente.

Descreve-se de seguida as questões e os intervenientes para cada fator em avaliação.

6.3.1.1 Satisfação do cliente

Questões

1. Vê vantagem na implementação de MFE no projeto?
2. A implementação de MFE é mais vantajosa que o anterior monólito?
3. A atual equipa tem capacidade para a implementação e manutenção do novo design?
4. Micro frontends é a melhor arquitetura para a solução?
5. Sente que MFEA aumenta a complexidade do projeto?

Intervenientes

Tabela 7 - Avaliação Satisfação do Cliente

	Questão 1	**Questão 2**	**Questão 3**	**Questão 4**	**Questão 5**
Interveniente 1	4	4	2	3	1

O único interveniente deste questionário é o *Product Owner* da equipa, que analisou as vantagens e desvantagens oferecidas pelo software que adotou MFEA, e nota a dificuldade da equipa para a implementação e manutenção desta arquitetura.

6.3.1.2 Satisfação da equipa

Questões

1. Vê vantagem na implementação de MFE no projeto?
2. A implementação de MFE é mais vantajosa que o anterior monólito?
3. É mais simples trabalhar com esta arquitetura?
4. Esta arquitetura aumenta o foco na entrega ao cliente?
5. Existe menos acoplamento na aplicação com o novo design?
6. Vê um aumento na qualidade de código?
7. Tornou-se mais fácil a deteção de erros?
8. A manutenção da solução tornou-se mais fácil?

Intervenientes

Tabela 8 - Avaliação Satisfação da Equipa

	Questão 1	Questão 2	Questão 3	Questão 4	Questão 5	Questão 6	Questão 7	Questão 8
Interveniente 1	2	3	2	2	4	3	2	2
Interveniente 2	3	3	2	4	4	4	2	2
Interveniente 3	4	4	2	4	4	4	4	2
Interveniente 4	4	4	3	4	5	5	5	3
Interveniente 5	4	4	5	5	5	5	5	5

A equipa consegue ver as vantagens na implementação de uma MFEA em comparação ao monólito atual, mas nota que seria mais difícil trabalhar no projeto com esta arquitetura assim como seria mais complexa a manutenção desta solução.

6.3.2 Ferramentas de análise de código e performance

As ferramentas de análise de código facilitam a deteção de anomalias, erros de codificação e vulnerabilidades de segurança existentes numa aplicação, entre outros. Estas ferramentas potenciam a qualidade do código produzido. Nesta situação em concreto será usado o *TSLint*. Para a avaliação de performance da aplicação será utilizado o *devtools* do *Chrome*, mais especificamente a opção de *performance*, onde é possível medir tempos de *scripting*, *rendering*, *painting*, *other*, *idle* entre outros.

6.3.3 Testes estatísticos

O teste de hipóteses estatísticas clássicas envolve o teste de uma hipótese nula. Na maioria dos casos, significa que não há efeito. Por exemplo, hipótese nula significa que não há diferença entre os meios das duas populações das quais as amostras são extraídas. Tendo isto em conta, existem duas perguntas que permitem distinguir os dois tipos de testes existentes quando não existe hipótese nula. As perguntas são:

- O resultado é maior (ou menor) que um certo valor?
- O resultado está dentro (ou fora) de uma certa gama de valores?

No primeiro caso estamos a falar de um teste **paramétrico**, que por sua vez pode ser [48]:

- **à esquerda**: quando a hipótese alternativa afirma que o valor real do parâmetro especificado na hipótese nula é menor do que a hipótese nula.
- **à direita:** quando a hipótese alternativa afirma que o valor real do parâmetro especificado na hipótese nula é maior do que as alegações de hipótese nula.

No segundo caso estamos a falar de testes **não-paramétricos** que ao contrário do anterior contém dois pontos críticos, ou seja, um intervalo de confiança que deve permanecer constante.

Tendo em conta o objetivo deste documento e após analise das duas opções anteriores, fará sentido neste projeto usar os dois tipos de teste.

6.4 Resultados finais

Esta secção permitirá avaliar os resultados obtidos durante o processo de desenvolvimento deste projeto, tendo em conta os fatores acima referidos.

6.4.1 Qualidade de código

A qualidade do código foi avaliada pela equipa no questionário realizado aos mesmos referido na secção 0. Para além disso, foi integrado do projeto *TSLint* nos novos projetos criados de forma a manter as boas práticas de programação que eram mantidas no monólito.

6.4.2 Performance da solução

Para avaliar a performance da solução, foi escolhido o carregamento das tiles do mapa como exemplo, sendo este um dos novos MFE e onde é possível comparar a diferença de carregamento da tiles carregadas para o ecrã. As seguintes Figuras 52 e 53, mostram a diferença entre a velocidade na arquitetura monolítica e MFEA. Como se pode verificar é

consideravelmente mais rápida a MFEA, visto ter um TTFB mais rápido cerca de 333 milisegundos por cada tile carregada.

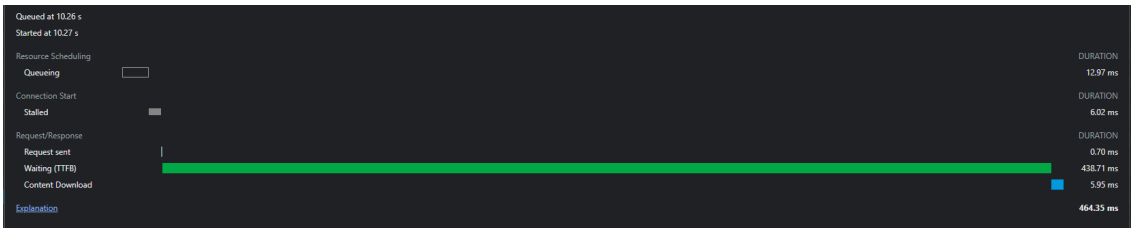

Figura 52 - Performance Monólito

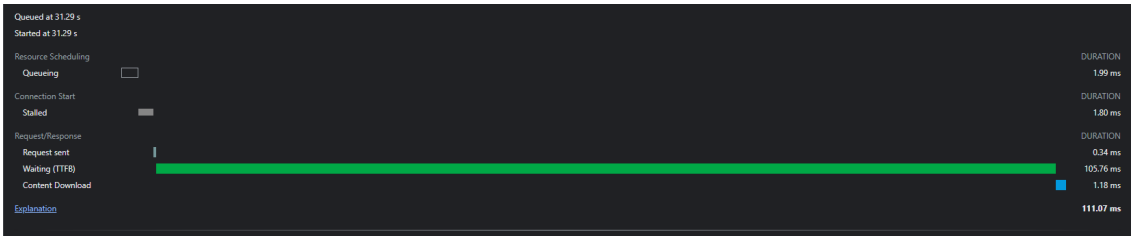

Figura 53 - Performance MFE

6.4.3 Deteção de erros

A facilidade de deteção de erros foi avaliada pela equipa no questionário realizado aos mesmos referido na secção 0, mais concretamente na questão 7.

6.4.4 Facilidade de implementação

Com a divisão da aplicação, torna-se mais fácil a implementação de novas funcionalidades, apenas aumentando a complexidade de comunicação entre cada equipa. Mas tendo apenas uma equipa responsável pelas várias aplicações, esse aumento de complexidade não se torna um problema para a solução.

6.4.5 Facilidade de manutenção

A facilidade de manutenção de erros foi avaliada pela equipa no questionário realizado aos mesmos referido na secção 6.3.1.2, mais concretamente na questão 8.

7 Conclusões

Durante este capítulo serão explicadas as conclusões relativas ao projeto realizado, incluindo objetivos alcançados, limitações e trabalho futuro.

7.1 Objetivos alcançados

Este projeto consistiu em dois objetivos principais, referidos em 1.3 e presentes no diagrama da .

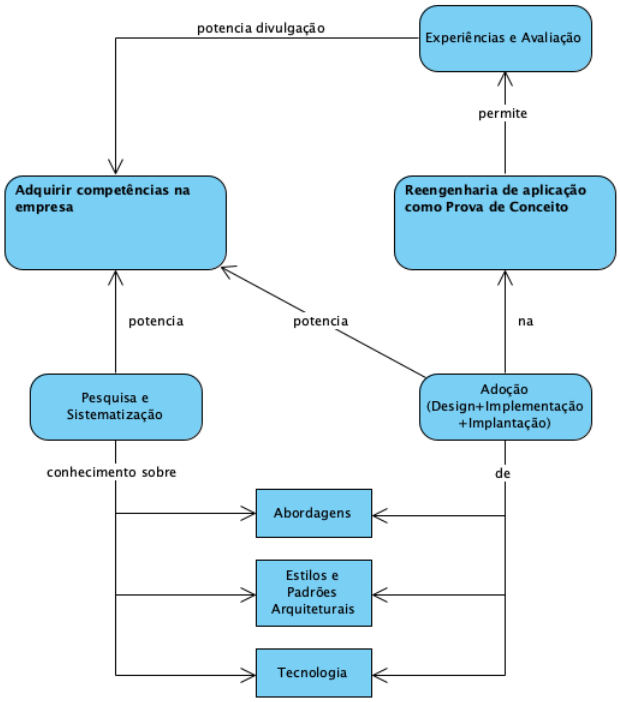

Figura 54 - Diagrama de objetivos

Ambos os objetivos foram atingidos e tanto a equipa como o cliente conseguiram ver as vantagens presentes na nova arquitetura. Naturalmente também lhes foi possível identificar os principais pontos negativos da arquitetura, como por exemplo o aumento de complexidade e dificuldade de manutenção. Para a obtenção destes resultados, foi necessária uma pesquisa e sistematização de conhecimentos sobre abordagens, estilos, padrões arquiteturais, estratégias de migração e tecnologias alternativas ao desenvolvimento de aplicações web

monolíticas, e em particular, propôs-se a adoção de Micro Frontend Architecture (MFEA). Após vários desenvolvimentos e experiências foi possível atingir um novo design e implementação para a solução referida, assim como adquirir competências e conhecimentos sobre MFE na empresa, potenciando assim a conclusão bem-sucedida dos objetivos propostos.

Na Tabela 9 serão detalhados os sub-objetivos e os resultados atingidos.

Tabela 9 – Sub-objetivos alcançados

Sub-objetivo	**RESULTADO**
Análise e comparação de abordagens conceptuais pertinentes	Realizado
Análise e comparação de estratégias de migração MFE pertinentes	Realizado
Análise e comparação de tecnologias MFE pertinentes	Realizado
Decisão sobre Estratégia de migração a utilizar durante a implementação	Realizado
Decisão sobre alternativas e consequente tecnologia a utilizar no redesign arquitetural	Realizado
Reengenharia (redesign arquitetural) da aplicação	Realizado
Implementação da nova arquitetura proposta (redesign arquitetural)	Realizado maioritariamente, pois nem toda a aplicação foi migrada
Migração dos unit e end-to-end testes para a nova arquitetura	Realizado, no caso de end-to-end tests apenas para todos os fluxos migrados
Design e implementação do pipeline de CI/CD para suportar a arquitetura/implementação	Não realizado, indicado em trabalho futuro

7.2 Limitações e trabalho futuro

Independentemente do empenho colocado no desenvolvimento deste projeto, nem tudo foi terminado com sucesso. Como se pode averiguar em 7.1 o trabalho que fica por realizar é o design e implementação do pipeline CI/CD.

Apesar de averiguada a maior parte das vantagens e desvantagens da MFEA, como por exemplo obter vantagens a nível de performance e um acoplamento mais baixo que o monólito, foi também possível identificar algumas das suas limitações de manutenção e complexidade de gestão de demasiados projetos por parte de equipas pequenas.

Ficam ainda por averiguar as diferenças de design no processo CI/CD e implementar as consequentes diferenças desenhadas. Faltou também comparar as diferenças de tempo das execuções de testes end-to-end no novo pipeline e consequente qualidade de implantação.

7.3 Apreciação final

Este projeto foi bom para aprofundamento de conhecimentos de arquiteturas alternativas ao desenvolvimento de aplicações web monolíticas, principalmente para arquiteturas Micro frontend.

Toda a documentação criada ao longo do projeto fornece bases para uma implementação deste tipo de arquitetura, principalmente para o projeto em questão. Mas espera-se que também o seja para outros projetos que se encaixem no mesmo espectro.

Referências bibliográficas

[1] W3C, “How It All Started,” [Online]. Available: https://www.w3.org/2004/Talks/w3c10-HowItAllStarted.

[2] I. W. Stats, “Internet World Stats,” [Online]. Available: https://www.internetworldstats.com/stats.htm.

[3] Y. B. &. F. Casalegno, 19 key essays on how internet is changing our lives, BBVA, 2014.

[4] U. b. o. l. statistics, “Web Developers and Digital Designers,” [Online]. Available: https://www.bls.gov/ooh/computer-and-information-technology/web-developers.htm.

[5] M. Geers, in *Micro Frontends In Action*, 2018.

[6] F. Stephan, “Top 4 Symptoms of Bad Code,” 2016. [Online]. Available: https://www.excella.com/insights/top-4-symptoms-of-bad-code.

[7] AWS, “Substituindo o Monólito com a AWS, Micro Frontends e Modyo,” 2020. [Online]. Available: https://aws.amazon.com/pt/blogs/aws-brasil/substituindo-o-monolito-com-a-aws-micro-frontends-e-modyo/.

[8] C. Jackson, “Micro Frontends,” 2019. [Online]. Available: https://www.martinfowler.com/articles/micro-frontends.html.

[9] L. Mezzalira, Front-End Reactive Architectures, 2018.

[10] S. Nicola, “Aula 1 - Análise de valor,” 2020.

[11] “Web Development,” 2020. [Online]. Available: https://www.techopedia.com/definition/23889/web-development.

[12] P. Skólski, “Single-page application vs. multiple-page application,” 2016. [Online]. Available: https://neoteric.eu/blog/single-page-application-vs-multiple-page-application/.

[13] G. Benedict, “Single Page Applications vs Multiple Page Applications,” 2018. [Online]. Available: https://medium.com/@goldybenedict/single-page-applications-vs-multiple-page-applications-do-you-really-need-an-spa-cf60825232a3.

[14] A. Brothers, “Single Page Application (SPA) vs Multi Page Application (MPA) – Two Development Approaches,” [Online]. Available: https://asperbrothers.com/blog/spa-vs-mpa/. [Accessed 15 03 2020].

[15] Microsoft, “Choose Between Traditional Web Apps and Single Page Apps (SPAs),” 2020. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps.

[16] S. Krishnamurthy, “What are Micro Frontends,” 2019. [Online]. Available: https://site.ieee.org/indiacouncil/files/2019/12/p105-p109.pdf.

[17] S. u. Haq, “Introduction to Monolithic Architecture and MicroServices Architecture,” 2018. [Online]. Available: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63.

[18] C. A. Kottman, “The Open GIS Consortium and Progress Toward Interoperability in Gis”.

[19] Wikipedia, “Tiled web map,” [Online]. Available: https://en.wikipedia.org/wiki/Tiled_web_map.

[20] M. fowler, “StranglerFigApplication,” 29 06 2004. [Online]. Available: https://martinfowler.com/bliki/StranglerFigApplication.html. [Accessed 12 05 2021].

[21] M. Geers, “Migration, local development, and testing,” [Online]. Available: https://livebook.manning.com/book/micro-frontends-in-action/chapter-14. [Accessed 12 05 2020].

[22] M. Geers, “Client Side Composition,” in *Micro Frontends in Action*, 2018.

[23] NGINX, “What is a Reverse Proxy vs. Load Balancer?,” [Online]. Available: https://www.nginx.com/resources/glossary/reverse-proxy-vs-load-balancer/. [Accessed 25 04 2021].

[24] MDN, “Web Components,” [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/Web_Components. [Accessed 25 04 2020].

[25] C. jackson, “Server-side template composition,” [Online]. Available: https://martinfowler.com/articles/micro-frontends.html#Server-sideTemplateComposition. [Accessed 25 04 2021].

[26] M. Geers, “Server Side Composition,” in *Micro Frontends in Action*, 2018.

[27] Wikipedia, “Server Side Includes,” [Online]. Available: https://en.wikipedia.org/wiki/Server_Side_Includes. [Accessed 25 04 2021].

[28] wikipedia, “Lazy loading,” [Online]. Available: https://en.wikipedia.org/wiki/Lazy_loading. [Accessed 26 04 2021].

[29] W3C, “ESI Language Specification 1.0,” [Online]. Available: https://www.w3.org/TR/esi-lang/. [Accessed 04 26 2021].

[30] M. Geers, “Client-side Routing & The Application Shell,” [Online]. Available: https://livebook.manning.com/book/micro-frontends-in-action/chapter-7/v-4/15. [Accessed 01 05 2021].

[31] A. Osmani, “App Shell Model,” [Online]. Available: https://developers.google.com/web/fundamentals/architecture/app-shell. [Accessed 01 05 2021].

[32] M. Geers, “Composition & Universal Rendering,” [Online]. Available: https://livebook.manning.com/book/micro-frontends-in-action/chapter-8/v-4/11. [Accessed 02 05 2021].

[33] C. Jackson, “Build Time Integration,” [Online]. Available: https://martinfowler.com/articles/micro-frontends.html#Build-timeIntegration. [Accessed 01 05 2021].

[34] H. Beta, “What is Bit,” [Online]. Available: https://harmony-docs.bit.dev/essentials/what-is-bit/. [Accessed 4 05 2021].

[35] SAP, “Luigi,” [Online]. Available: https://github.com/SAP/luigi. [Accessed 04 05 2021].

[36] “Luigi Documentation,” [Online]. Available: https://github.com/SAP/luigi/blob/master/docs/README.md. [Accessed 4 05 2021].

[37] Podium, “Podium conceptual overview,” [Online]. Available: https://podium-lib.io/docs/podium/conceptual_overview. [Accessed 1 05 2021].

[38] Zalando, “Project Mosaic Microservices for the frontend,” [Online]. Available: https://www.mosaic9.org/. [Accessed 4 05 2021].

[39] Zalando, “Zalando/Tailor,” [Online]. Available: https://github.com/zalando/tailor. [Accessed 4 05 2021].

[40] G. S. w. single-spa. [Online]. Available: https://single-spa.js.org/docs/getting-started-overview. [Accessed 07 05 2021].

[41] S. spa, “Configuring single-spa,” [Online]. Available: https://single-spa.js.org/docs/configuration. [Accessed 7 05 2021].

[42] S. spa, “Building single-spa applications,” [Online]. Available: https://single-spa.js.org/docs/building-applications. [Accessed 7 05 2021].

[43] S. spa, “single-spa Microfrontend Types,” [Online]. Available: https://single-spa.js.org/docs/module-types. [Accessed 7 05 2021].

[44] Ara, “The Nova architecture (Universal Rendering),” [Online]. Available: https://ara-framework.github.io/website/docs/nova-architecture. [Accessed 17 05 2021].

[45] C4, “The C4 model for visualising software architecture,” [Online]. Available: https://c4model.com/. [Accessed 13 09 2021].

[46] Wikipédia, “Requisitos funcionais,” [Online]. Available: https://pt.wikipedia.org/wiki/Requisito_funcional. [Accessed 06 08 2021].

[47] A. R. A. Gail M. Sullivan, Analyzing and Interpreting Data FromLikert-Type Scales.

[48] N. University, “One-tailed and Two-tailed Tests,” [Online]. Available: https://internal.ncl.ac.uk/ask/numeracy-maths-statistics/statistics/hypothesis-testing/one-tailed-and-two-tailed-tests.html#One-tailed%20Tests.

[49] P. Bellivau, PDMA toolbook.

[50] G. Lancaster, Implementing value strategy through the value chain, 2000.

[51] T. Woodall, “Conceptualising 'Value for the Customer,” *Conceptualising 'Value for the Customer': An Attributional, Structural,* p. 44.

[52] D. A. AAKER, Construindo marcas fortes, 2007.

[53] A. O. &. Y. Pigneur, Business Model Generation.

[54] Microsoft, MicrosoftApplication Architecture Guide 2, 2009.

[55] M. Richards, “Layered Architecture,” in *Software Architecture Patterns*.

[56] J. Palermo, “The Onion Architecture : part 3,” 2008. [Online]. Available: https://jeffreypalermo.com/2008/08/the-onion-architecture-part-3/.

[57] M. v. S. a. L. F. P. Cléver Ricardo Guareis de Farias, “A SYSTEMATIC APPROACH FOR COMPONENT-BASED”.

[58] E. Evans, Domain Driven Design.

[59] M. Fowler, “UbiquitousLanguage,” 2006. [Online]. Available: https://martinfowler.com/bliki/UbiquitousLanguage.html.

[60] M. Fowler, “Bounded Context,” 2014. [Online]. Available: https://martinfowler.com/bliki/BoundedContext.html.

[61] E. Evans, Domain Driven Design Quickly, 2006.

[62] T. D. D. Design, “What is Tactical Design ?,” 2019. [Online]. Available: https://thedomaindrivendesign.io/what-is-tactical-design/. [Accessed 26 2 2021].

[63] A. Soft, 2 2020. [Online]. Available: https://www.altexsoft.com/blog/typescript-pros-and-cons/. [Accessed 28 08 2021].

[64] w3techs, “Usage statistics of JavaScript as client-side programming language on websites,” [Online]. Available: https://w3techs.com/technologies/details/cp-javascript. [Accessed 28 08 2021].

[65] D. Prasanjith, “5 Reasons to Use TypeScript with React,” 2020. [Online]. Available: https://blog.bitsrc.io/5-strong-reasons-to-use-typescript-with-react-bc987da5d907.

[66] M. W. Docs, “Window.localStorage,” 2019. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Web/API/Window/Window.localStorage.

[67] N. Obaseki, “localStorage in JavaScript: A complete guide,” 2020. [Online]. Available: https://blog.logrocket.com/localstorage-javascript-complete-guide/#sessionstoragevslocalstorage.

[68] Wikipedia, “Here,” [Online]. Available: https://pt.wikipedia.org/wiki/Here#cite_note-19.

[69] H. Technologies, “HERE - Pioneers of location technology,” [Online]. Available: https://www.here.com/company/about-us.

[70] Wikipedia, “Leaflet (software),” [Online]. Available: https://en.wikipedia.org/wiki/Leaflet_(software)#cite_note-2.

[71] Leaflet, “Leaflet,” 2020. [Online]. Available: https://leafletjs.com/.

[72] N. Ighodaro, “Why use Redux? A tutorial with examples,” 2020. [Online]. Available: https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/.

[73] D. Ianakiara, “10 Motivos Para Usar Redux em 2018,” 2018. [Online]. Available: https://blog.getty.io/10-motivos-para-usar-redux-em-2018-96664e124425.

[74] MobX, “MobX,” [Online]. Available: https://mobx.js.org/.

[75] A. Ravichandran, “A definitive guide to Redux vs. MobX,” 2019. [Online]. Available: https://blog.logrocket.com/redux-vs-mobx/. [Accessed 13 2 2021].

[76] Webpack, “Webpack Concepts,” [Online]. Available: https://webpack.js.org/concepts/.

[77] C. Arsenault, “A Comprehensive Overview of webpack,” 2018. [Online]. Available: https://www.keycdn.com/blog/webpack.

[78] A. Baptista, “The Problem with Webpack and Why It Is (Kind of) Our Fault,” 2018. [Online]. Available: https://medium.com/hurb-labs/the-problem-with-webpack-8a025268a761.

[79] T. Point, “Gulp - Overview,” [Online]. Available: https://www.tutorialspoint.com/gulp/gulp_overview.htm. [Accessed 29 08 2021].

[80] D. Bhattacharjee, “What is Cypress for Test Automation?,” 2020. [Online]. Available: https://www.tutorialspoint.com/what-is-cypress-for-test-automation.

[81] A. Rustamzadeh, “Why Cypress?,” [Online]. Available: https://docs.cypress.io/guides/overview/why-cypress.html#In-a-nutshell.

[82] P. H. JILL ANTWEILER, “Pros and Cons of Cypress,” 2020. [Online]. Available: https://blog.testery.io/pros-and-cons-of-cypress/.

[83] Altexsoft, “The Good and the Bad of Selenium Test Automation Software,” [Online]. Available: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-selenium-test-automation-tool/. [Accessed 29 08 2021].

[84] Angular, “Introduction to Angular concepts,” [Online]. Available: https://angular.io/guide/architecture.

[85] P. Sight, “ANGULAR 101: PROS, CONS, FEATURES AND MORE,” 2019. [Online]. Available: https://www.pluralsight.com/blog/software-development/angular-101.

[86] Vue.js, “What is Vue.js?,” [Online]. Available: https://vuejs.org/v2/guide/.

[87] AltexSoft, “The Good and the Bad of Vue.js Framework Programming,” 2019. [Online]. Available: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/.

[88] S. Buna, “Yes, React is taking over front-end development. The question is why.,” 2017. [Online]. Available: https://www.freecodecamp.org/news/yes-react-is-taking-over-front-end-development-the-question-is-why-40837af8ab76/.

[89] JavaTPoint, “React Features,” [Online]. Available: https://www.javatpoint.com/react-features.

[90] DA14, “Top 10 Advantages of using REACT.JS,” 2017. [Online]. Available: https://da-14.com/blog/its-high-time-reactjs-ten-reasons-give-it-try.

[91] AltexSoft, “The Good and the Bad of ReactJS and React Native,” 2020. [Online]. Available: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/.

[92] Microsoft, “O que é a computação na cloud?,” [Online]. Available: https://azure.microsoft.com/pt-pt/overview/what-is-cloud-computing/.

[93] ClearBridge, “Cloud Apps vs. Web Apps: Understanding the Benefits and Differences,” [Online]. Available: https://clearbridgemobile.com/cloud-apps-vs-web-apps/.

[94] C. Editor, “Pros and Cons of Amazon Web Services,” 2020. [Online]. Available: https://cogitogroup.net/pros-and-cons-of-amazon-web-services/.

[95] S. J. B. Clive Longbottom, “infrastructure (IT infrastructure),” [Online]. Available: https://searchdatacenter.techtarget.com/definition/infrastructure.

[96] IBM, “Infrastructure as Code,” 2019. [Online]. Available: https://www.ibm.com/cloud/learn/infrastructure-as-code#toc-what-is-in-jDhTgmQ-.

[97] Hashicorp, “Introduction to Terraform,” [Online]. Available: https://www.terraform.io/intro/index.html.

[98] V. Fedak, “What is Terraform and why it rocks,” 2017. [Online]. Available: https://itsvit.com/blog/what-is-terraform-and-why-it-rocks/.

[99] C. Camp, “Jenkins Review,” [Online]. Available: https://comparecamp.com/jenkins-review-pricing-pros-cons-features/.

[100] K. France, “Pros and Cons of Bitbucket Pipelines,” [Online]. Available: https://blog.idrsolutions.com/2018/03/pros-cons-bitbucket-pipelines/. [Accessed 30 08 2021].

[101] Docker, “Docker overview,” [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed 30 08 2021].

[102] Nick, “Docker Containers: Advantages and Disadvantages,” [Online]. Available: https://blog.iron.io/docker-containers-the-pros-and-cons-of-docker/. [Accessed 30 08 2021].

[103] M. H. Nick Rich, “Value Analysis,” p. 33, janeiro 2000.

[104] Wikipédia, “Requisitos não funcionais,” [Online]. Available: https://pt.wikipedia.org/wiki/Requisito_n%C3%A3o_funcional. [Accessed 07 08 2021].

[105] Wikipédia, “FURPS+,” [Online]. Available: https://pt.wikipedia.org/wiki/FURPS. [Accessed 07 08 2021].

[106] B. A. T. i. H. -. Coepd, “What is FURPS+?,” [Online]. Available: https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/. [Accessed 07 08 2021].

[107] Single-spa, “Deployment and Continuous Integration,” [Online]. Available: https://single-spa.js.org/docs/recommended-setup/#deployment-and-continuous-integration-ci. [Accessed 2021 08 09].

[108] j. denning, “docker-import-maps-mfe-server,” [Online]. Available: https://github.com/single-spa/docker-import-maps-mfe-server. [Accessed 2021 09 09].

Anexo A. Análise de valor

Neste capítulo serão explicados conceitos teóricos relativos à análise de valor e processo de inovação. Na secção 0 apresentam-se os conceitos fundamentais adotados, e nas secções seguintes a sua adoção no projeto. Se o leitor já estiver familiarizado com os conceitos, pode avançar para a secção 0.

Contextualização teórica

Análise de valor

'A Análise de Valor é um processo sistemático, formal e organizado de análise e avaliação para melhorar a rentabilidade das aplicações de produtos. As técnicas que apoiam as atividades de VA incluem técnicas usadas para todos os exercícios de AV e alguns que são apropriados apenas sob certas condições' **[10]**.

A análise diz respeito à função de um produto para atender às necessidades ou aplicações pretendidas por um cliente. Para atender a essa exigência funcional, o processo de revisão deve incluir uma compreensão da finalidade para a qual o produto é utilizado' **[10]**.

Consiste nas seguintes fases:

- Orientação;
- Identificação funcional;
- Análise funcional;
- Alternativas criativas;
- Análise e avaliação;
- Implementação.

A aplicação deste processo ao projeto é descrita na secção 0.

Processo de inovação

Este processo divide-se em três subprocessos [10], através dos quais as organizações transformam ideias em produtos, serviços, novos processos e criam melhorias, com a finalidade de evoluir, competir e diferenciar-se com sucesso no mercado.

Os subprocessos são:

- Fuzzy Front End (FFE);
- Desenvolvimento de um novo produto (NPD);
- Comercialização;

A primeira parte (FFE), é considerada um dos maiores subprocessos de melhoria e comercialização de todo o processo de inovação. Já o subprocesso de desenvolvimento de novo conceito (NCD) foi criado para gerir eficazmente o FFE, e será explicado de seguida.

Modelo de desenvolvimento de novo conceito

Fornece uma linguagem comum e definição dos componentes-chave FFE. O modelo NCD () consiste em três partes-chave [49]:

- **Motor** é a liderança, cultura e negócios estratégicos da organização que impulsiona os cinco elementos-chave que são controláveis pela empresa: elemento-chave 1, 2, 3, 4 e 5.
- **Roda**, área interna de raios define os cinco elementos de atividade controláveis (identificação de oportunidades, análise de oportunidades, geração de ideias e enriquecimento, seleção de ideias e definição de conceito).
- **O aro,** fatores de influência consistem em capacidades organizacionais, no mundo exterior (canais de distribuição, lei, política governamental, clientes, concorrentes, e clima político e econômico), as ciências capacitantes (internas e externas) que podem estar envolvidas.

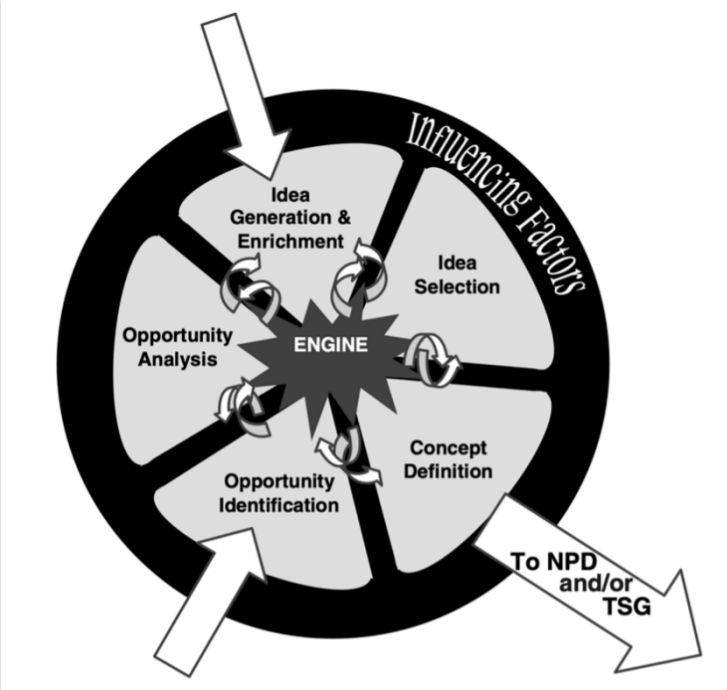

Figura 55 - Modelo NCD

Além disso, o modelo tem uma forma circular, sugerindo que as ideias devem fluir, circular e iterar entre todos os cinco elementos. O fluxo pode abranger os elementos em qualquer ordem ou combinação e pode usar um ou mais elementos mais do que uma vez.

A aplicação deste processo ao projeto é descrita na secção 0.

Valor, valor percecionado e valor para o cliente

Valor

'O valor foi definido em diferentes contextos teóricos como necessidade, desejo, interesse, padrão/critérios, crenças, atitudes e preferências' [10].

'Alguns autores consideram a criação de valor uma troca entre benefícios e sacrifícios percebidos pelos clientes durante a oferta de um fornecedor' [50].

Valor percecionado

Na literatura de marketing, o termo "valor do cliente" é usado para ilustrar um cenário derivado pelo cliente, e também pelo fornecedor [10].

Valor para o cliente

O termo "valor do cliente" é usado dentro da literatura de marketing para representar tanto o que o cliente recebe como o que o cliente pode entregar [51].

A aplicação destes conceitos ao projeto é descrita na secção 1.6.2.

Proposta de valor

'Uma proposta de valor é compreendida como a afirmação dos benefícios oferecidos pela marca ao cliente e que lhe proporcionam valor' [52].

Proposta de valor define 4 pilares:

- 'o que'
- 'quem'
- 'como'
- 'quanto'

A proposta de valor deve deixar claro qual o produto ou serviço que vai ser fornecido, clientes-alvo, valor fornecido e o valor exclusivo.

A aplicação deste processo ao projeto é descrita na secção 1.6.3.

Modelo Canvas

O modelo canvas é uma ferramenta de negócio usada para gerar e capturar valor e representar a estrutura da empresa. E divide-se em nove pontos [53]:

- **Propostas de valor:** resoluções de problemas e necessidades;
- **Atividades-chave:** principais atividades da empresa;
- **Parceiros-chave:** entidades parceiras e produtos adquiridos a outrem;
- **Recursos-chave:** recursos necessários para a entrega do produto;
- **Relação com o cliente:** relação mantida com o cliente e como se integra no processo;
- **Segmentos de Clientes:** para quem esta a ser criado o valor;
- **Estrutura de custos:** maiores custos inerentes à solução;
- **Fontes de receita:** resultante da proposta oferecida ao cliente;
- **Canais:** canais onde são mantidas as relações com o cliente.

A aplicação deste modelo ao projeto é descrita na secção 1.6.4.

Análise de valor do projeto

Contexto Atual

Nesta secção serão abordadas as práticas e metodologias adotadas pela organização e pela equipa. Desta forma contextualizando melhor o ambiente onde o caso de estudo se insere.

A Critical TechWorks, adotou práticas e metodologias que permitem à equipa uma maior gestão dessas. Neste caso em particular metodologias que permitam um processo desenvolvimento mais controlado e com maior qualidade. Isto é conseguido através da adoção de SCRUM.

A equipa adotou o SCRUM e tira proveito de todas as práticas do mesmo, realizando todas as cerimónias ao longo das *Sprint's*, que duram normalmente duas semanas. Todo o trabalho realizado é acompanhado por um SCRUM Master e as tarefas e funcionalidades são planeadas por um *Product Owner* (PO).

Anexo B. Abordagens conceptuais

Arquitetura em camadas

'A arquitetura em camadas baseia-se na junção de funcionalidades relacionadas, da mesma solução, em camadas distintas que se organizam verticalmente em cima umas das outras' [54]. As funcionalidades dentro de cada camada estão relacionadas por uma função ou responsabilidade comum.

'A comunicação entre camadas é explícita e vagamente acoplada' [54]. Esta arquitetura proporciona uma forte separação de conceitos (*Separation of Concerns*) que, por sua vez proporciona maior flexibilidade e manutenção.

As suas principais características são [54]:

- Abstração, porque cria uma 'imagem' abstrata do sistema como um todo, oferecendo detalhes suficientes para entender os papéis e responsabilidades de cada camada;
- Encapsulamento, pois todas as características de cada camada são expostas nos seus limites;
- Camadas funcionais claramente definidas, porque as funcionalidades de cada camada bem definidas. As camadas superiores, enviam comandos para as camadas inferiores que por si reagem em conformidade, permitindo um fluxo de dados nos dois sentidos;
- Alta coesão, sendo as responsabilidades e funcionalidades bem definidas em cada camada ajuda a maximização da coesão dentro de cada camada;
- Reutilização, não existindo dependências dentre camadas altas e baixas torna-se possível a reutilização das camadas mais baixas.

As vantagens deste estilo arquitetural são [54]:

- Abstração, pois permite que alterações realizadas na mesma sejam feitas a um nível abstrato;

- Isolamento, permite isolar atualizações tecnológicas em cada camada, de forma a minimizar o impacto no sistema;
- Fácil gestão, através da separação de conceitos é possível identificar dependências e organizar o código em secções mais fáceis de gerir;
- Performance, pois a distribuição de camadas em níveis físicos ajuda a melhorar a escalabilidade, tolerância de falhas e desempenho;
- Reutilização, porque a distribuição de responsabilidade promove a reutilização, como por exemplo MVC;
- Testabilidade, permitida através da divisão entre camadas. Através desta divisão torna-se mais simples a construção de objetos exemplo que permitam imitar/testar o comportamento esperado da aplicação.

As desvantagens deste estilo arquitetural são [55]:

- Custo de gestão se houver muitas camadas, porque usando este padrão tende-se a criar novas camadas com o crescimento dos projetos, isto gera uma maior dificuldade de escalar o projeto;
- O desempenho baixa à medida que mais camadas adicionadas, pois torna-se cada vez maior o número de camadas que é preciso atravessar para ser possível responder a uma necessidade do negócio;
- Demasiada abstração pode perturbar a intenção das camadas.

Arquitetura por camadas sobrepostas

No caso de camadas sobrepostas, existe uma organização horizontal das camadas existentes na solução. Para a comunicação entre as mesmas existem duas opções diferentes dentro desta arquitetura [55]:

- **Camadas fechadas:** Uma camada fechada significa que um pedido se move camada em camada e deve passar pela camada imediatamente abaixo à sua até chegar à camada pretendida, ou seja, os componentes só podem interagir com componentes na mesma camada ou com componentes da camada diretamente abaixo. Isto significa que a nível de isolamento este tipo de camadas raramente impactam componentes noutras camadas;

- **Camadas abertas:** neste caso os pedidos podem evitar passar por certas camadas, então torna-se possível que componentes de uma camada interajam com componentes na mesma camada ou com componentes em qualquer camada inferior.

Arquitetura por camadas concêntricas

Como a arquitetura de camadas sobrepostas abertas referida no capítulo anterior 0, a arquitetura por múltiplas camadas concêntricas que aceitam a comunicação com camadas inferiores. Organiza-se numa forma circular e a comunicação é feita em direção ao centro, que representa o domínio. Este é um dos pontos principais que diferencia este tipo de arquitetura. No centro encontra-se a base de dados enquanto a infraestrutura e a lógica de negócio adjacente á sua volta [56].

Arquitetura baseada em componentes

'A arquitetura baseada em componentes descreve uma abordagem de engenharia de software, baseada no design de componentes funcionais ou lógicos. Estes expõem interfaces de comunicação bem definidas contendo métodos, eventos e outras propriedades'.

Estes componentes podem ser de interface de utilizador, como listas e botões, e componentes que expõem um subconjunto específico de funções usadas por outros componentes [54].

As suas principais características são [54]:

- Reutilização de componentes, porque os componentes são criados para ser reutilizados criando uma maior modularidade, apesar de alguns serem criados para problemas específicos;
- Componentes substituíveis, pois estes podem ser facilmente substituídos por outros componentes similares;
- Encapsulamento, porque os componentes expõem interfaces que permitem que as suas funcionalidades sejam facilmente atualizadas, sem revelar os seus processos internos;
- Independência, porque os componentes são criados com o mínimo de dependências possíveis.

A arquitetura por camadas pode e deve adotar uma arquitetura baseada em componentes, em que que cada camada é um componente (eventualmente composto por vários outros), apresentando as caraterísticas descritas nesta secção.

As vantagens deste estilo arquitetural são [54]:

- Fácil implantação, porque é possível disponibilizar novas versões, assim que estas estejam disponíveis, sem impactar o sistema;
- Custo reduzido, pois o uso de componentes externos ajuda a reduzir o custo de desenvolvimento e manutenção;
- Fácil desenvolvimento, devido às suas interfaces bem definidas facilita o desenvolvimento, sem impactar outras partes da aplicação;
- Reutilização, porque usando componentes reutilizáveis é possível utilizá-los em diferentes partes de um sistema ou até diferentes aplicações;
- Mitigação da complexidade técnica, porque estes estão contidos e incluem a própria gestão de ciclo de vida, tratamento de eventos e transações.

As desvantagens deste estilo arquitetural são [57]:

- Componentes de médio a grande porte, pois estes tornam-se mais difíceis de desenvolver;
- Tamanho da solução final, pois os componentes podem tornar-se demasiado grandes ou serem demasiados;
- Composição de componentes, porque pode causar problemas de integração entre os mesmos.

Domain Driven Design (DDD)

'O DDD é uma abordagem orientada a objetos, para projetar softwares com base no seu domínio de negócios [54]'. As maiores vantagens são provenientes da união da equipa para aplicar uma abordagem de design orientada por domínio e mover o modelo de domínio para o centro de discurso do projeto, como explicado na Arquitetura em camadas **Erro! A origem da referência não foi encontrada.**. 'Ao fazer isso, os membros da equipa passam a compartilhar uma linguagem que enriquece sua comunicação e a mantém conectada ao software (*Ubiquitus Language)'* [58].

As suas principais características são [54]:

- O modelo de domínio pode ser visto como uma estrutura a partir da qual as soluções podem então ser racionalizadas;
- Necessário ter uma boa compreensão do domínio de negócios que deseja modelar ou ser qualificado para adquirir esse conhecimento de negócios;
- Criação de *Ubiquitus Language*: Linguagem comum e rigorosa entre desenvolvedores e utilizadores. Essa linguagem deve ser baseada no Modelo de Domínio usado no software. Esta linguagem (e o modelo) devem evoluir à medida que a compreensão da equipa sobre o domínio cresce [59]. O vocabulário dessa linguagem comum inclui os nomes das classes e operações proeminentes, por exemplo [58].

As suas vantagens são [54]:

- Comunicação, pois todas as partes envolvidas usam o mesmo modelo de domínio, tornando-se este a maneira de comunicar conhecimento de negócio e requerimentos;
- Extensível, porque o modelo de domínio é normalmente modular e flexível para ser expandido;
- Testável, devido ao seu baixo acoplamento e alta coesão torna-se facilmente testável.

As suas desvantagens são [54, 60]:

- Necessário conhecer o domínio, o que pode tornar-se uma desvantagem para novos elementos das equipas;
- 'Junção total do modelo de domínio para um sistema grande não será viável ou econômico'.

Entities, Value Objects e Aggregates

Entities

Entidades são objetos que tem uma identidade que se mantém em todos os estados do software. Estes objetos podem ser mutados, mas a sua identidade mantém-se sempre [61].

Value Objects

Value objects são objetos imutáveis que são usados para descrever certos aspectos de um domínio, e que ao contrário das anteriormente referidas Entities, não tem identidade [61].

Aggregates

Os Aggregates têm como base as Entities e os Value Objects. É um agregado de uma ou mais Entidades, e também pode conter Objetos de Valor. Contem uma Entitie que representa a raiz do Aggregate [62].

Bounded Context

Bounded Context é um padrão central no DDD. É o foco da seção de design estratégico do DDD que se foca sobre lidar com grandes modelos. O DDD lida com modelos grandes dividindo-os em diferentes *Bounded Context* [60].Delimita a aplicabilidade de um determinado modelo para que a sua compreensão compartilhada seja clara. Dentro desse contexto, o objetivo passar para manter o modelo logicamente unificado, sem preocupações com a aplicabilidade fora dos limites do mesmo [58].

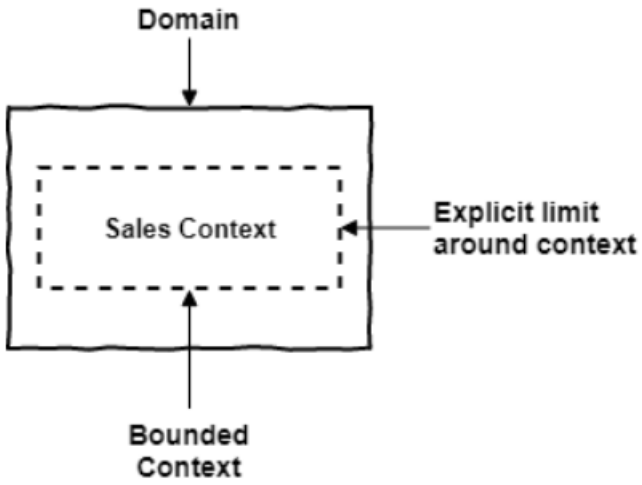

Figura 56 - Bounded Context

Context Map

'Um *Context Map* está na sobreposição entre gestão de projetos e design de software' [58]. As pessoas que trabalham perto diariamente compartilham um contexto. No entanto pessoas em equipas diferentes, dividem-se em contextos diferentes. Quando aparecem ambiguidades torna-se necessária uma forma para conscientizar a equipa da existência de dois contextos diferentes dentro da aplicação. A melhor maneira de fazer isso é através de um *Context Map* (Figura 57).

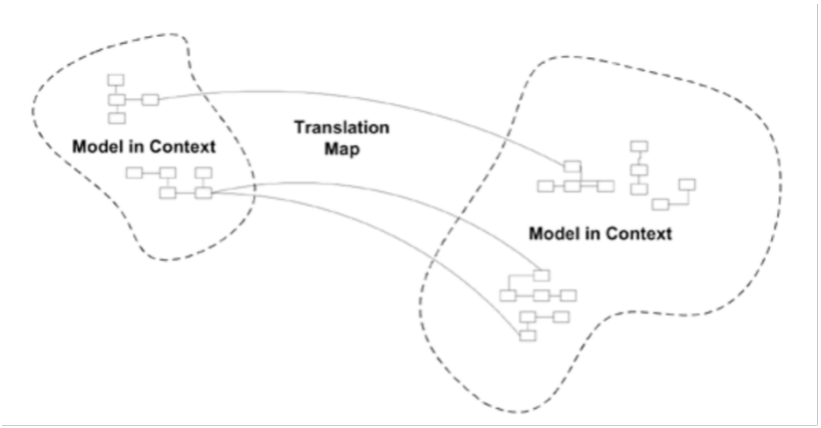

Figura 57 - Context Map

Tecnologia Relevante

A liberdade de escolha de tecnologias que MFEA oferece torna-se uma vantagem, não obrigando necessariamente a mudar a espectro de tecnologia pré-existente.

Sendo este um projeto que se já se encontra em desenvolvimento e, que o pretendido é uma alteração para uma arquitetura mais vantajosa para o futuro da mesma, algumas das tecnologias já se encontravam previamente escolhidas. Essas mesmas tecnologias serão enunciadas de seguida, assim como também serão explicadas tecnologias que sejam opções às mesmas e que possam eventualmente mostrar-se vantajosas para o projeto.

Javacript e Typescript

Javascript é a línguagem número um *de* desenvolvimento web durante vários anos e foi lançada em 1995 [63]. Esta trouxe uma nova interatividade para as páginas web e atualmente 95% dos sites são construídos utilizando *javascript* [64]. Nos últimos anos o *typescript* ganhou bastante popularidade no desenvolvimento frontend. *typescript* é um superconjunto de *javascript* que contém todas as funcionalidades do javascript. Logo qualquer programa escrito em *javascript* também será executado de igual forma em *typescript*.

As suas vantagens do *typescript* são [65]:

- Maior consistência do código criado, pois este torna-se mais expressivo e fácil de entender;
- Suporte a maior número de navegadores, pois maior parte dos sites é construído usando estas linguagens;
- Melhor verificação de tipos estáticos, no caso do uso de *typescript*, ajdua a encontrar erros com maior antecedência;
- Menos erros indefinidos, sendo os erros encontrados mais cedo, isto torna-se mais provável;

As suas desvantagens do *typescript* são [63]:

- Mais uma curva de aprendizagem usando *typescript*, apesar da semelhança com *javascript* é necessário tempo para aprender as diferenças entre ambos;
- Translação para *javascript, como os browsers* não interpretam *typescript* é necessário a sua tranlação para *javascript,* o que leva tempo;
- Mais código, as diferenças entre as duas linguagens, que tornam o *typescript* mais expressivo, trazem um aumento de código criado;

Local Storage

É uma propriedade que permite que aplicações javascript guardem pares de chaves/valores em browsers Web, sem data de validade. Isso significa que os dados continuarão persistidos na página mesmo após um reload ou fecho da mesma. É uma das maneiras mais eficazes armazenar dados do lado do cliente [66].

As suas características são [67]:

- Guardar pares de chave/valor em browsers web;
- Leve;
- O mecanismo localStorage é disponibilizado disponível através da propriedade Window. *Window.localStorage* faz parte da interface Janela em javascript, que representa uma janela contendo um documento do tipo DOM.
- É uma propriedade de leitura que retorna uma referência ao objeto de armazenamento local usado para armazenar dados que só são acessíveis à origem que o criou.

As suas vantagens são:

- O seu uso possibilita a melhora de performance da aplicação, no caso da existência de necessidade de armazenar dados, que por exemplo se encontrem catalogados e tenham uma baixa necessidade de mudança.

As suas desvantagens são:

- Memória limitada.

Mapas na Web

HERE Maps

É uma empresa fornecedora de dados geoespaciais, anteriormente conhecida por Smart2Go, Nokia Maps e OviMaps. Criado originalmente para dispositivos Nokia, mas que atualmente é propriedade de uma parceria de alguns gigantes do mercado como a BMW e a AUDI [68].

As suas principais características são [69]:

- Fornecem uma aplicação *cloud-based* do mapa mundo digital, que atualmente é usada em mais de 150 milhões de veículos, navegação orientada por voz a pedestres e motoristas e localizações 3D;
- Coletam dados de mais de 100.000 fontes e com 80 biliões de chamadas às suas API por mês, são capazes de fornecer a representação digital mais recente e precisa do mundo.

As suas principais vantagens:

- Disponibilização de um mapa constantemente atualizado;
- Diferentes tipos de navegação;
- Veracidade da informação
- Parcerias com Gigantes do mercado.

As suas principais desvantagens são:

- Peso do mapa enviado para o browser.

Leaflet

O Leaflet é uma biblioteca *javascript open-source*, usada para criar aplicações web de geolocalização. Lançado pela primeira vez em 2011, suporta a maioria das plataformas móveis e desktop [70].

As suas características são [71]:

- Disponibiliza também uma lista de componentes web que representam eventos relacionados com informação geográfica. Como por exemplo eventos em áreas e estradas e sinalização de lugares através de marcadores de localização;
- Disponibiliza um mapa base que permite a integração com *Tiled Web Map.*

As suas vantagens são [71]:

- Elevado número de componentes personalizáveis;
- Integração com Here Maps;
- Elevada usabilidade.

As suas desvantagens são:

- Peso dos componentes pode ser um problema quando criados aos milhares.

Orquestradores de estados

Redux

Redux é um gestor de estados projetado para ajudar na construção de soluções javascript e *typescript*. Esta gestão de estados é essencial na facilitação da comunicação e partilha de dados entre os componentes da solução.

As suas caraterísticas são [72]:

- Estrutura de dados Tangível;
- Consulta e alteração de dados;
- Ligação com componentes;
- Partilha horizontal de dados.

As suas vantagens são [73]:

- Fluxo de dados invisível;

- Otimização da solução;
- *Store* central*;*
- Aumenta a velocidade do desenvolvimento;
- Implementação simples.

As suas desvantagens são [72]:

- Frameworks/librarias já possuem partilha de dados vertical entre componentes.

Neste caso, o React já possui uma maneira nativa de gerir estados nos seus componentes. Esta funcionalidade também é usada no projeto em questão, mas nem sempre se mostra a melhor opção.

MobX

A MobX é um orquestrador de gestão de estados. Torna impossível produzir um estado inconsistente. A estratégia para conseguir isso é certificar-se de que tudo o que pode ser derivado do estado de aplicação, será derivado automaticamente [74].

As principais características desta ferramenta são [9]:

- Criação de estrutura de dados;
- Consulta e alteração de dados (Computed Values & Reactions);
- Partilha horizontal de dados;
- Alteração e gestão de estados (*Observables, Actions*).

As suas vantagens são [9]:

- Compatível com *react* e *view.js*;
- Fluxo de dados invisível e reativo;
- Implementação simples;
- Separação de conceitos.

As suas desvantagens são [75]:

- Permitir a criação de múltiplas *stores*, pode ser prejudicial;
- Comunidade mais pequena que o *redux*;
- Menos escalável.

Javascript Bundlers

Webpack

O *webpack* é um empacotador de módulos estáticos para soluções *javascript*/*typescript*, ou seja, o *webpack* transforma todo o código existente nas aplicações em conjunto com as suas dependências e transforma-as em ficheiros estáticos que representam esses módulos [76].

As suas características são [76, 77]:

- Compilação de código e dependências;
- Criação de ficheiros estáticos;
- Modular;
- Permite a integração de bibliotecas de terceiros.

As suas vantagens são [77, 76]:

- Oferece personalização, porque disponibiliza bastantes *plugins* e formas de personalizar o *bundle* criado;
- Divide árvores de dependências, permitindo que sejam carregados quando necessário;
- Permite que cada ficheiro estático seja um módulo;
- Permite a criação de diferentes ambientes desenvolvimento/produção, pois disponibiliza várias configurações para tal;

As suas desvantagens são [78]:

- Adaptação a constantes mudanças, pois estas são normais acontecerem no webpack;
- Peso das migrações, por vezes podem tornar-se custosas tanto de implementar como de entender.

Gulp

Gulp usa o Node.js como plataforma base. Usando apenas código *javascript,* permite criar e executar tarefas em aplicações frontend. Estas tarefas automatizadas permitem por exemplo a minificação de CSS e HTML, concatenar ficheiros de bibliotecas e compilar os ficheiros SASS. Ap´s a criação destas tarefas, para as executar basta correr um comando Shell ou Bash.

As suas características são [79]:

- Simples e rápido;

- Usa SASS e LESS como pré-processador de CSS;
- Permite atualizar automaticamente a página após a edição dos ficheiros de origem;
- Fácil de entender e construir, por usar *javascript*.

As suas vantagens são [79]:

- Vantagem de velocidade por ser bastante leve;
- Fácil de entender e construir, por usar *javascript;*
- Plugins são fáceis de utilizar;

As suas desvantagens são:

- Os plugins não conseguem executar múltiplas tarefas;
- Mais difícil de configurar em comparação a outros Bundlers.

Testes

Cypress

Cypress é uma ferramenta de testes direcionada a aplicações frontend web. Tem como objetivo ajudar os engenheiros de software a superar as dificuldades existentes em testar automaticamente e regressivamente o lado do cliente da aplicação [80].

As suas características são [81]:

- Configurar ambiente de testes;
- Construir testes;
- Correr testes;
- Fazer *debug* a testes.

As suas vantagens são [80]:

- *Cypress* dá a funcionalidade de capturar *snapshots* durante a execução;
- Existência de *Logs* descritivos durante a evolução dos testes;
- Pode ser corrido através de uma interface do utilizador ou pela linha de comandos;
- Tem a capacidade de guardar capturas de ecrã aquando de uma falha. Também pode gravar vídeos de toda a execução do conjunto de testes mesmo correndo pela linha de comando;

- Recarrega todas as atualizações feitas nos testes automaticamente;
- Representação visual da percentagem de testes bem-sucedidos e com erros.

As suas desvantagens são [82]:

- Apenas dá suporte a *javascript*;
- Suporte a poucos navegadores;
- Não tem versão mobile.

Selenium

'O Selenium é um conjunto de ferramentas de automatização de software de *open source* que se tornaram um produto de garantia de qualidade'. Com uma lista de várias linguagens de programação, todos os principais sistemas operativos e bastantes browsers suportados, o Selenium é atualmente utilizado por bastantes empresas do mercado [83].

As suas características são:

- Providencia soluções de testes automatizados;
- Open source;
- Garantir qualidade de produtos;
- Lartgamente utilizado.

As suas vantagens são:

- Grátis para utilizar;
- Segue as práticas de devOps e Agile;
- Suporta testes em dispositivos móveis;
- É bastante compatível, tanto a nível de browsers como linguagens.

As suas desvantagens são:

- Curva de aprendizagem complexa;
- Falta de relatórios gerados automaticamente, apenas possível através de snapshot aquando do erro.

Frameworks e bibliotecas de implementação de SPA

Angular

Angular é uma *framewok* para a construção de interfaces web e SPA, que usa *typescript*. A arquitetura de uma aplicação Angular tem como base alguns conceitos fundamentais [84]:

- Os módulos básicos de construção da estrutura são componentes organizados em *NgModules*;
- Os *NgModules* são constituídos por código de uma parte funcional da aplicação;
- A aplicação tem sempre pelo menos um módulo raiz que permite iniciar a mesma;
- Os componentes definem as vistas, que são conjuntos de elementos que aparecem no ecrã do utilizador.

As suas vantagens são [84, 85]:

- Robustez da *framewok,* pois esta é suportada pela Google e está dotada de uma documentação bastante detalhada;
- Usa *typescript* nativamente, trazendo algumas das vantagens já referidas anteriormente sobre esta linguagem.
- Ampla gama de librarias externas, estas podem ser facilmente integradas por parte dos desenvolvedores;
- Uso de *dependency injection,* pode tornar-se uma grande vantagem em projetos de larga escala, apesar consumir algum tempo extra.

As suas desvantagens são [85]:

- Dificuldade de migração ente versões, pois as versões disponibilizadas pelo Angular incluíram mudanças drásticas entre elas;
- Indicado para projetos de maior escala;
- Curva de aprendizagem, devido à existência de diversos módulos, linguagem de programação(*typescript*), integrações e possibilidades de personalização.

Vue

Vue é uma *famework* que usa *javascript* para criar interfaces web, SPA, aplicações *desktop* e *mobile*. Foi projetado para ser incrementalmente adotável e de fácil de integração com outras bibliotecas ou projetos existentes [86].

As suas características são [87]:

- Arquitetura baseada em componentes;
- Simplicidade;
- Leve.

As suas vantagens são [87]:

- Tamanho pequeno, rápida de descarregar e instalar, o que impacta a experiência do utilizador;
- Virtual *DOM rendering,* o *vue* através desta funcionalidade, que consiste numa cópia do DOM original, consegue descobrir quais partes a atualizar sem atualizar toda a DOM;
- Performance, o uso da virtual DOM aumenta bastante a performance das aplicações;
- Componentes de ficheiro único, que permitem uma maior reutilização, melhor leitura de código e facilidade de testar;

As suas desvantagens são [87]:

- Recursos limitados, porque embora existam bastantes opções disponíveis o *vue* ainda não esta no mesmo nível do Angular e do React;
- Risco vs Flexibilidade, pois a existência de demasiadas opções pode ser problemática dentro da equipa;
- Falta de desenvolvedores experientes, pois ainda é uma tecnologia bastante jovem.

React

React [88] é uma biblioteca utilizada para a construção de interfaces com o utilizador (UI). Basicamente o *react* dá aos desenvolvedores a capacidade de trabalhar com um navegador virtual (Virtual DOM) mais simples de utilizar. O navegador virtual da *react* age como um agente entre o desenvolvedor e o navegador final.

As suas caraterísticas são [89]:

- Simplicidade;
- *DOM* virtual;
- Performance*;*
- Componentes*.*

As vantagens desta biblioteca são [90]:

- Facilidade de aprendizagem, pois é uma biblioteca que já esta bem assente no mercado e está dotada de uma boa documentação;
- Componentes de ficheiro único, que permitem uma maior reutilização, melhor leitura de código;
- Melhor performance, o uso da virtual DOM aumenta bastante a performance das aplicações;
- Facilita a escrita de código, porque é fácil reutilizar código e organizá-lo de forma percebível;
- React Hooks, introduzido pelo React em 2019 que permite partilhar a logica de estados entre componentes sem grandes mudanças no código.

As desvantagens desta biblioteca são [91]:

- Velocidade de mudança, pois o ambiente muda constantemente, e é necessário reaprender novas formas de fazer as coisas;
- Mistura de *javascript* e HTML no mesmo ficheiro, pois pode tornar-se confuso inicialmente;

Computação na nuvem vs aplicação web

A computação na nuvem é o fornecimento de serviços informáticos, que podem ser servidores, armazenamento, bases de dados, rede, software, análises e inteligência, através da Internet (*cloud*) [92].

As suas características são [92]:

- Utilização da internet;
- Disponibilização de recursos;
- Escalabilidade;
- Flexibilidade.

As suas vantagens são [93]:

- Maior velocidade de inovação e disponibilização de recursos;

- Criação de infraestrutura dimensionável;
- Lógica e armazenamento de dados processados na *cloud*;
- Do ponto de vista do utilizador, a aplicação comporta-se como qualquer outra página web normal;
- O utilizador não precisa de ter a aplicação instalada na máquina local;
- Maior escalabilidade e flexibilidade;

As suas desvantagens são:

- Perda de controlo total sobre a aplicação;
- Nem todas as funcionalidades estão disponíveis globalmente;

Amazon web services

A AWS fornece uma grande panóplia de serviços, desde tecnologias de infraestrutura, computação, armazenamento e base de dados até tecnologias mais recentes, como *machine learning* e inteligência artificial.

As suas características são [94]:

- Baseado na *cloud*;
- Fornecimento de serviços;
- Acessibilidade.

As suas vantagens são [94]:

- Está entre os mais confiáveis na indústria, tendo em conta a sua utilização e ampla visibilidade no mercado;
- Fornece velocidade e agilidade;
- Grande número de serviços disponíveis;
- Apenas são pagos os serviços que se utilizam.

As suas desvantagens são [94]:

- Serviços disponíveis por país, o que pode tornar-se problemático em aplicações que tenham de ser disponibilizadas em vários países;
- Serviços menos seguros, que devem ser evitados;
- Vítima de ataques informáticos.

Infraestrutura via código

Neste caso, infraestrutura é a estrutura que suporta um sistema. Na computação, a infraestrutura de tecnologia da informação é composta por recursos físicos e virtuais que suportam o fluxo, armazenamento, processamento e análise de dados. Pode ser centralizada dentro de um *data center*, ou pode ser descentralizada e espalhada por vários data *centers* através da *cloud* [95]. Infraestrutura via código (Infrastructure as Code – IaC) visa o uso de uma linguagem de configuração alto nível, para automatizar a criação e alteração da infraestrutura necessária.

As suas principiais características são:

- Automatização do processo de criação de infraestrutura, através de código;
- Uso da *cloud*;
- Uso de HCL.

As suas vantagens são [96]:

- Atualmente não é incomum para uma empresa implantar centenas de aplicações em produção todos os dias fazendo com que a infraestrutura esteja em constante mudança, assim é essencial que uma organização automatize a criação da infraestrutura dos seus projetos.
- Menor a necessidade de manutenção da infraestrutura;
- Facilita-se a disponibilização da solução;
- Maior resposta a necessidades de mudança;
- Promoção do foco n desenvolvimento/qualidade da solução;
- Evita inconsistências entre versões da solução.

As suas desvantagens são [96]:

- Necessidade de aprender HCL ou necessidade de alguém especializado na construção deste tipo de arquitetura.

Infraestrutura mutável Vs. Imutável

Uma **infraestrutura mutável** pode ser alterada após ter sido originalmente implantada.

As suas vantagens são [96]:

- Este tipo de infraestrutura dá às equipas uma flexibilidade de fazer alterações para cumprir requisitos de desenvolvimento ou como resposta a eventuais casualidades, após a implantação da mesma.

As suas desvantagens são [96]:

- Pode tonar-se prejudicial ao uso de IaC, na medida em que se tornará mais difícil de manter a consistência entre as diferentes implantações.
- Necessidade de aprender HCL ou necessidade de alguém especializado na construção deste tipo de arquitetura.

A **infraestrutura imutável**, não pode ser alterada uma vez provisionada. No caso da existência de necessidade de mudança da mesma, esta deve ser completamente substituída por uma nova. Assim otimiza a IaC, garantindo ainda mais os benefícios que oferece.

As suas vantagens são [96]:

- Remove a necessidade de diferentes configurações e torna ainda mais fácil manter a consistência entre o ambiente de testes e implantação;
- Facilita a manutenção e a manutenção de versões da infraestrutura e tona mais fácil reverter alterações.

As suas desvantagens são:

- Necessidade de aprender HCL ou necessidade de alguém especializado na construção deste tipo de arquitetura.

Terraform

O *Terraform* é uma ferramenta que permite escrever uma infraestrutura na *cloud* ou localmente, através de uma linguagem de configuração alto nível conhecida por HCL [97].

As suas principais características são [98]:

- Uso de ficheiros de configuração;
- Arquitetura *cloud* imutável;
- Adaptável;

- Compatibilidade.

As suas vantagens são [98]:

- Compatível com vários fornecedores *cloud,* pois é possível criar as mesmas funcionalidades utilizando o mesmo 'código';
- Popularidade, pois é utilizada em larga escala;
- Sintaxe simples e com larga documentação.

As suas desvantagens são [98]:

- Framework recente e não aperfeiçoada;
- Implantações apenas *cloud*.

CI/CD

Jenkins

Jenkins é uma ferramenta com base em Java que oferece serviços de *continuous delivery e integration*. Esta solução é um sistema hospedado por servidor que opera em um *contêiner* de *servlet*, como o Apache Tomcat [99].

As suas características são [99]:

- Oferta de serviçõe*s* de *continuous integration;*
- Gratuito*;*
- Suporte robusto e *alargado;*
- Fácil utilizaçã*o;*

As suas vantagens são [99]:

- Fácil configuração;
- Funcionalidades estáveis;
- Disponibilidade;
- Alta disponibilidade de *plugins*.

As suas desvantagens são:

- As instâncias de Jenkins geridas por um utilizador com privilégios administrativos. Isso pode levar a vários problemas quando se trata de auditorias/prestação de contas;
- Por padrão, Jenkins executa todas as compilações no mesmo ambiente que o próprio servidor de compilação, o que pode levar a inúmeros problemas. Alguns *plugins* resolvem este problema, apesar de terem de ser instalados manualmente.

Bitbucket Pipelines

Bitbucket Pipelines é um serviço de *continuous delivery,* disponibilizado pela Atlassian que permite a integração de testes e processos de *build* sobre o repositório de um determinado projeto. Quando uma destas pipelines é adicionada a um repositório, torna-se possivel carregar uma imagem docker, clonar o repositório para esta imagem para de seguida realizar os processos necessários.

As suas características são [100]:

- Útil e seguro;
- Fácil de utilizar e configurar;
- Diferentes opções de configuração;
- Incorporado com bitbucket.

As suas vantagens são [100]:

- A imagem docker criada não é afetada por *commits* posteriores;
- Cada *commit* tem um estado de pipeline independente;
- Escalabilidade, pois as imagens docker criadas para cada *commit* não dependem do hardware disponível.

As suas desvantagens são [100]:

- Projetos *maven* multi modulo são mais difíceis de configurar;
- Não é possível armazenar dados para de forma fiável, porque a não ser que a informação fique em memória, pois é criada uma nova imagem docker a cada *commit*.

Docker

Docker é uma plataforma que foi projetada para ser leve e simples. 'Docker *contêiner* é uma unidade padronizada de software usado para implantar aplicativos. Estes embalam todo o código e dependências de uma aplicação para que ele possa ser usado em qualquer servidor ou sistema operacional' [101].

As suas características são:

- Isolamento;
- Independência;
- Consistência.

As suas vantagens são [102]:

- Automatização, permitem que sejam agendadas tarefas para ocorrer quando elas são necessárias;
- Estabilidade, pois é baseado em Linux e corre Linux Kernel em qualquer máquina;
- Poupança de espaço de memoria, por usar *containers* e não máquinas virtuais que consumem mais espaço.

As suas desvantagens são [102]:

- Mudanças continuas, de forma a tornar a as suas implementações mais eficientes, o que pode ser um problema para os projetos se manterem atualizados;
- Curva de aprendizagem.