UNIVERSIDADE D
COIMBRA

Igor Filipe dos Santos Moreira

FERRAMENTA DE VISUALIZAÇÃO DE MICROSSERVIÇOS

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software orientada pelo Professor Doutor Filipe Araújo e Professor Doutor Raul Barbosa e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Janeiro de 2022

Resumo

Um produto de *software* é desenvolvido com base numa arquitetura que define a organização de todos os seus componentes. Existem várias arquiteturas, como a tradicional arquitetura monolítica e a moderna arquitetura de microsserviços. Atualmente, muitas empresas utilizam a arquitetura de microsserviços no desenvolvimento dos seus produtos. Com o decorrer do tempo, a sua dimensão aumenta à medida que são adicionados novos serviços para satisfazer as necessidades impostas pelo mercado, como por exemplo o aumento de utilizadores ou novas funcionalidades. Estes sistemas podem atingir graus de complexidade elevados e requerem ferramentas de monitorização que permitam vigiar o estado dos seus serviços. Por esta razão, foi realizado um estágio que decorreu no âmbito da unidade curricular Dissertação/Estágio do Mestrado em Engenharia Informática da Universidade de Coimbra, que teve lugar no Centro de Informática e Sistemas da Universidade de Coimbra, onde foi desenvolvida uma solução que, através de dados recolhidos pelo método de *tracing*, oferece quatro formas de visualização destes sistemas. O objetivo desta ferramenta é permitir que os utilizadores observem todos os serviços que compõem o sistema e identifiquem facilmente a origem de anomalias. Este documento relata o trabalho efetuado durante o estágio. Apresenta o estado da arte realizado sobre a arquitetura monolítica, arquitetura de microsserviços e os seus métodos de monitorização. Descreve a fase de planeamento da ferramenta, constituída pelos casos de uso, requisitos funcionais e não funcionais. Apresenta o produto da fase de desenvolvimento e testes efetuados.

Palavras-Chave

Microsserviços. Monitorização. *Tracing*. Visualização.

Abstract

A software product is developed based on an architecture that defines the organization of all its components. There are several architectures such as the traditional monolithic architecture and the modern microservices architecture. Currently, many companies use the microservices architecture in the development of their products. Over time, its size increases as new services are added to meet the needs imposed by the market, such as the increase in users or new features. These systems can reach high degrees of complexity and require monitoring tools to monitor the status of their services. For this reason, an internship was carried out within the scope of the curricular unit Dissertation/Internship of the Master in Computer Engineering at the University of Coimbra, which took place at the Center for Informatics and Systems of the University of Coimbra, where a solution was developed that, through of data collected by the tracing method, offers four ways of visualizing these systems. The purpose of this tool is to allow users to observe all the services that make up the system and easily identify the source of anomalies. This document reports the work done during the internship. It presents the state of the art carried out on monolithic architecture, microservices architecture and its monitoring methods. It describes the tool planning phase, consisting of use cases, functional and non-functional requirements. It presents the product of the development phase and tests performed.

Keywords

Microservices. Monitoring. Tracing. Visualization.

Agradecimentos

Agradeço à minha família por me apoiarem ao longo de todo o meu percurso académico. Agradeço também ao Professor Doutor Filipe Araújo, Engenheiro Jaime Correia e Engenheiro André Bento por toda a sua disponibilidade e acompanhamento durante estágio.

Índice

Acrónimos

CISUC - Centro de Informática e Sistemas da Universidade de Coimbra

API - *Application Programming Interface*

REST - *REpresentational State Transfer*

Lista de Figuras

Lista de Tabelas

Capítulo 1
Introdução

O presente estágio decorreu no âmbito da unidade curricular Dissertação/Estágio do Mestrado em Engenharia Informática da Universidade de Coimbra e a entidade acolhedora é o Centro de Informática e Sistemas da Universidade de Coimbra (CISUC).

É importante para as empresas de desenvolvimento de *software* garantirem que a sua solução é viável e responde às necessidades impostas pelo mercado, como por exemplo o aumento do número de utilizadores e novas funcionalidades. Estes fatores são, por vezes, determinantes para o seu sucesso.

1.1 Problema

Um produto de *software* é desenvolvido com base numa arquitetura que define as regras da estrutura e organização dos seus componentes. As empresas desenvolvem as suas soluções de acordo com a arquitetura que se adeque melhor ao seu produto, considerando as suas vantagens e desvantagens. Existem vários padrões arquiteturais, desde a tradicional arquitetura monolítica à moderna arquitetura de microsserviços.

Na arquitetura monolítica, os componentes que constituem uma aplicação partilham o mesmo *container*. Nos projetos de grande dimensão esta arquitetura não é a mais apropriada. Em produtos formados por vários milhares de linhas de código qualquer alteração pode facilmente causar *bugs* difíceis de localizar e corrigir.

Na arquitetura de microsserviços, os componentes da aplicação são isolados e independentes dos outros. Podemos pensar em cada componente (serviço) como uma miniaplicação que comunica com os restantes formando uma aplicação completa. Esta arquitetura adequa-se melhor do que a monolítica a projetos de grande dimensão. O código fonte de cada serviço é pequeno, o que facilita o seu desenvolvimento (menos linhas, menos *bugs*), torna a testagem de cada unidade mais fácil e simultaneamente acelera a sua entrega ao mercado. Esta arquitetura oferece flexibilidade para realizar atualizações e a escalabilidade necessária para lidar com fatores impostos pelo mercado, como por exemplo o aumento de utilizadores. Com o decorrer do tempo, a dimensão destas aplicações aumenta à medida que são adicionados novos microsserviços com o fim de satisfazer as necessidades que vão surgindo. Este processo aumenta a sua complexidade, dificultando a descoberta da origem de anomalias, compreensão do sistema, a sua visibilidade e monitorização. Estes problemas requerem uma solução.

1.2 Motivação

A arquitetura de microsserviços tem vindo a ser adotada por cada vez mais empresas. Gigantes internacionais como a Amazon, Netflix e eBay utilizam esta arquitetura nos seus produtos. O funcionamento correto das aplicações deve ser assegurado, por isso os serviços

que as constituem são monitorizados constantemente. São necessárias ferramentas para visualizar o seu estado e que permitam identificar facilmente anomalias para uma intervenção rápida e direta. Embora existam ferramentas no mercado para monitorizar estes sistemas, muitas são caras e não se focam na sua visualização.

1.3 Objetivos

O objetivo do estágio foi desenvolver uma ferramenta que ofereça novas formas de visualizar um sistema desenvolvido sob a arquitetura de microsserviços. No primeiro semestre foi adquirido conhecimento sobre a arquitetura de microsserviços, métodos de monitorização e realizado o planeamento do desenvolvimento da solução. No segundo foi executada a sua implementação.

1.4 Plano de trabalhos

Plano de trabalhos proposto para o primeiro semestre:

1. Estudo do estado da arte (1 mês);
2. Análise dos requisitos (1 mês);
3. Definição da arquitetura (1 mês);
4. Início da implementação (1 mês);
5. Escrita do relatório (1 mês).

O plano de trabalhos executado foi semelhante ao proposto. A figura 1 apresenta detalhadamente as tarefas realizadas durante o primeiro semestre, com início a 10 de fevereiro e fim a 30 de junho.

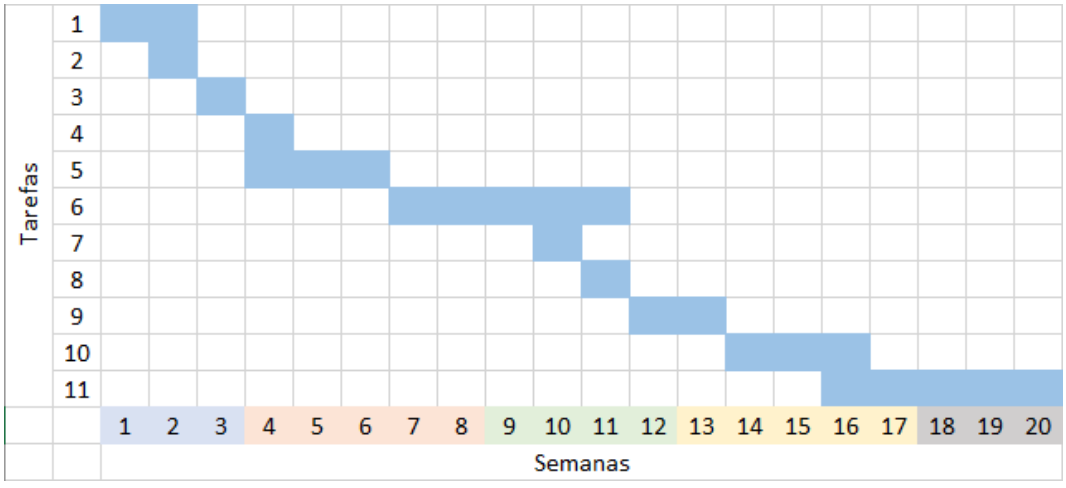

Figura 1 - Plano de trabalhos executado no primeiro semestre

Legenda das tarefas:

1. Estudo de microsserviços e *tracing* distribuído;
2. Estudo de *Javascript*;
3. Estudo de *p5.js*;
4. Estudo de *WebGL*;
5. Estudo de *PixiJS*;

6. Estudo de *React*;
7. Estudo de *D3.js*, *Chart.js* e *Recharts*;
8. Estudo de *React Material UI*;
9. Estudo de algoritmos de grafos e formulação dos casos de uso;
10. Formulação dos requisitos funcionais e não funcionais;
11. Escrita do relatório.

A figura 2 apresenta o plano de trabalhos proposto para o segundo semestre, com início a 20 de setembro e fim a 24 de janeiro.

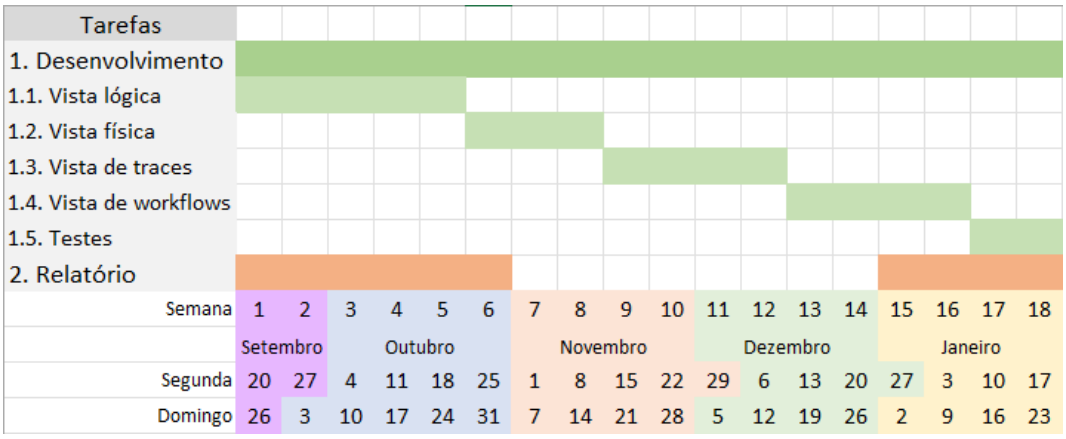

Figura 2 - Plano de trabalhos proposto para o segundo semestre

A figura 3 ilustra o plano de trabalhos executado.

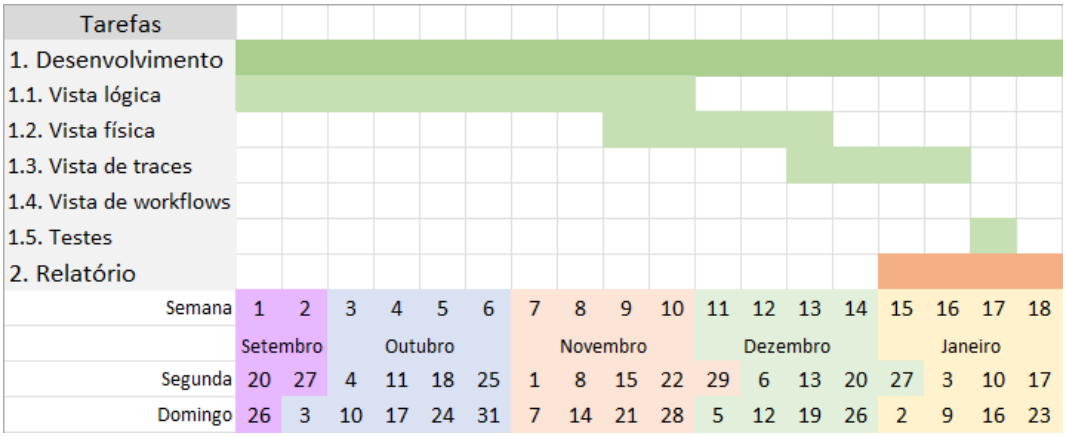

Figura 3 - Plano de trabalhos executado no segundo semestre

1.5 Estrutura do documento

Este documento encontra-se organizado por capítulos.

No primeiro capítulo é dada a introdução, abordado o problema e motivação deste estágio. São também apresentados os objetivos e plano de trabalhos realizado durante os dois semestres.

No segundo capítulo é abordada a arquitetura monolítica, arquitetura de microsserviços e respetivos métodos de monitorização.

O terceiro capítulo apresenta os projetos anteriores de uma ferramenta de visualização de microsserviços, *mockups* e funcionalidades da atual.

No quarto capítulo são mostrados os casos de uso que descrevem a interação do utilizador e a ferramenta.

No quinto capítulo são exibidos os requisitos funcionais que explicam as funcionalidades e também os requisitos não funcionais.

No sexto capítulo são apresentadas as tecnologias e a sua utilidade durante a fase de implementação, um protótipo desenvolvido durante o primeiro semestre, a aplicação final e os testes que foram realizados.

Finalmente, no sétimo capítulo é apresentada a conclusão do trabalho desenvolvido.

Capítulo 2
Estado da arte

Neste capítulo são apresentados os padrões arquiteturais utilizados no desenvolvimento de *software* conhecidos por arquitetura monolítica e arquitetura de microsserviços. Um padrão arquitetural é uma forma geral de organizar os elementos que integram um sistema.

2.1 Arquitetura monolítica

A arquitetura monolítica organiza todos os componentes de uma aplicação no mesmo *container*. Estes elementos estão interligados e o seu funcionamento depende dos outros, logo na eventualidade de uma falha num componente, a aplicação será interrompida [1]. Em adição, quando o código de um componente é modificado, é necessário compilar toda aplicação, incluindo os componentes que não foram alterados.

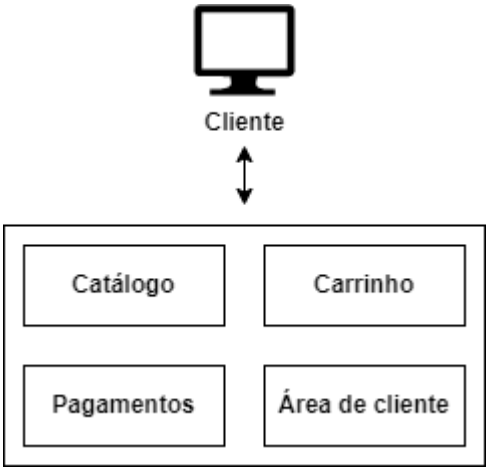

Figura 4 – Exemplo de uma aplicação com arquitetura monolítica

A figura 4 exemplifica a organização dos componentes de um sistema composto por uma aplicação que utiliza a arquitetura monolítica. O *container*, que representa o monólito, é formado por quatro componentes: catálogo, carrinho, pagamentos e área de cliente.

2.2 Arquitetura de microsserviços

Na arquitetura de microsserviços a aplicação é composta por vários serviços, que são componentes pequenas, independentes entre si e responsáveis por desempenhar uma tarefa. Os serviços são isolados do resto do sistema. Podem ser distribuídos em máquinas diferentes ou ser virtualizados na mesma [2]. Comunicam entre si através de uma API (*Application Programming Interface*) que estabelece um conjunto definições e protocolos de comunicação através da internet, como por exemplo REST (*REpresentational State Transfer*) [3]. Nesta

arquitetura, se um serviço for interrompido devido a uma falha, o resto do sistema permanece funcional, salvo se este for indispensável para o seu funcionamento. Esta arquitetura é sobretudo utilizada no âmbito de aplicações *web*.

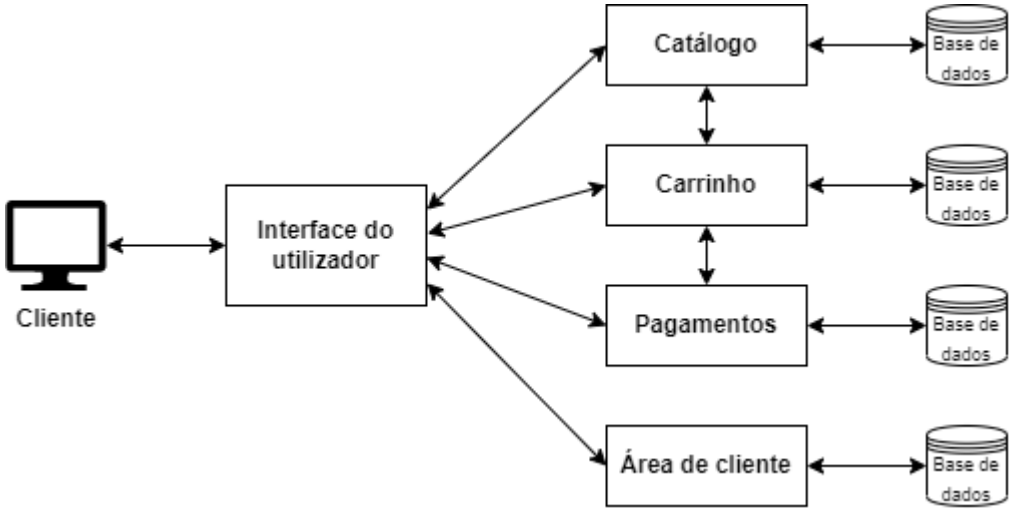

Figura 5 – Exemplo de uma aplicação com arquitetura de microsserviços

A figura 5 exemplifica uma aplicação que utiliza a arquitetura de microsserviços. É composta por cinco serviços: interface do utilizador, catálogo, carrinho, pagamentos e área de cliente. A comunicação entre estes elementos é representada por setas.

Existem aplicações que foram desenvolvidas com a arquitetura monolítica, mas devido a fatores como o aumento de utilizadores e a necessidade de alterar ou adicionar funcionalidades, mudaram para a arquitetura de microsserviços [4]. A Netflix, Amazon e eBay são exemplos de empresas que realizaram esta troca nos seus produtos [5]. O isolamento dos serviços permite escalar facilmente as funcionalidades para responder às necessidades do momento. Esta arquitetura também aumenta a rapidez da distribuição das aplicações no mercado e acelera o seu crescimento uma vez que é possível disponibilizar mais funcionalidades em menos tempo [6].

A tabela 1 compara alguns aspetos que considero importantes para o desenvolvimento de *software* entre a arquitetura de microsserviços e monolítica [5].

Tabela 1 - Comparação entre as arquiteturas microsserviços e monolítica

	Arquitetura de Microsserviços	Arquitetura Monolítico
Escalabilidade	Permite escalar apenas alguns serviços [7].	É necessário escalar toda a aplicação.
Resiliência	Como os serviços são isolados e independentes, na eventualidade de uma falha num deles, o sistema não é comprometido e continua a funcionar devidamente, salvo se o serviço em questão for	Os componentes do sistema partilham o mesmo *container*. Se ocorrer um erro num, toda a aplicação deixa de funcionar.

	essencial para o eu funcionamento [7].	
Manutenção	O código fonte de um serviço é pequeno, logo é mais fácil de modificar e manter [5].	O código fonte de uma aplicação pode ser grande, o que o torna mais complexo e dificulta a sua compreensão e manutenção.
Modificabilidade	A alteração de funcionalidades na aplicação é mais simples porque apenas são atualizados os serviços essenciais [5].	A alteração do código fonte de qualquer componente requer que toda a aplicação seja compilada.
Deploy	O *deploy* dos serviços é executado individualmente [6].	É necessário fazer *deploy* de toda a aplicação.

A arquitetura de microsserviços tem as suas vantagens, no entanto, os sistemas evoluem e crescem ao longo do tempo. A sua complexidade aumenta à medida que são adicionados novos serviços que trazem suporte a novas funcionalidades, o que se traduz num aumento da dificuldade dos administradores destes sistemas para os compreenderem e gerirem. Além disso, faz parte do ciclo de vida de um projeto existirem alterações na equipa responsável pelo seu desenvolvimento, sendo fundamental os novos membros o conhecerem o mais rápido possível, pois na eventualidade de surgir um problema é preciso agir de forma eficaz, identificando a sua origem e resolvendo-o.

O funcionamento correto destes sistemas deve ser assegurado, por isso precisam de ser vigiados e testados sempre que existem alterações. Existem métodos para este fim, como a monitorização através do controlo de recursos, *logs*, métricas e *tracing*.

2.2.1 Métodos de monitorização

1. Controlo de recursos

Este método consiste no controlo da utilização dos recursos dos serviços ou do sistema operativo da máquina hóspede (processador, memória, disco e rede). A deteção de níveis invulgares na utilização de um recurso pode estar relacionada com a existência de algum problema [8].

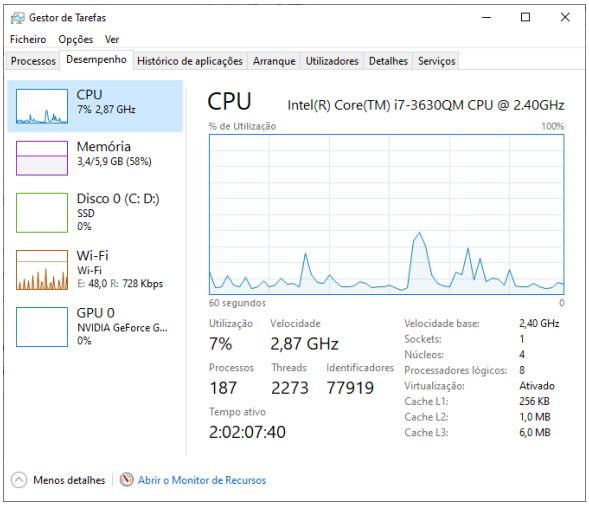

Figura 6 - Monitorização de recursos do sistema operativo

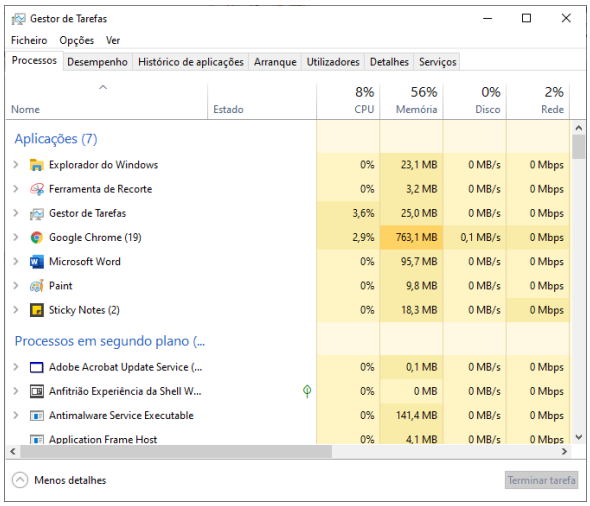

Figura 7 - Monitorização dos recursos de aplicações

2. *Logs*

Os *logs* podem ser produzidos pelas aplicações e sistemas durante a sua execução. São apresentados no formato de texto e mostram informações sobre eventos que ocorrem. A partir dos *logs* é possível visualizar o *flow* da aplicação, erros, exceções e resultados de operações [8].

```
06-18 17:04:09.876 2023-2023/system_process I/AppOps: No existing app ops /data/system/appops.xml; starting empty
06-18 17:04:09.899 2023-2023/system_process W/ProcessCpuTracker: Skipping unknown process pid 2040
06-18 17:04:09.906 2023-2023/system_process I/IntentFirewall: Read new rules (A:0 B:0 S:0)
06-18 17:04:09.913 2023-2045/system_process I/ServiceThread: Enabled StrictMode logging for android.display looper.
06-18 17:04:09.918 2023-2023/system_process D/AppOps: AppOpsService published
06-18 17:04:09.919 2023-2023/system_process I/SystemServiceManager: Starting com.android.server.power.PowerManagerService
06-18 17:04:09.922 2023-2047/system_process I/ServiceThread: Enabled StrictMode logging for PowerManagerService looper.
06-18 17:04:09.924 2023-2023/system_process E/PowerManagerService-JNI: Couldn't load power module (No such file or directory)
06-18 17:04:09.925 2023-2023/system_process W/libsuspend: Error writing 'on' to /sys/power/state: Invalid argument
06-18 17:04:09.925 2023-2023/system_process I/libsuspend: Selected wakeup count
```

Figura 8 - Exemplo de *logs*

3. Métricas

As métricas são dados numéricos quantificáveis utilizadas para avaliar o desempenho da aplicação. Permitem tirar conclusões exatas relativas ao seu comportamento e determinar se é o esperado. No contexto de microsserviços são utilizadas métricas para medir o desempenho através de fatores como o tempo de resposta a pedidos de um serviço, a quantidade de pedidos que recebe e erros que ocorrem num determinado período de tempo. As métricas podem ser apresentadas em gráficos ou texto [8].

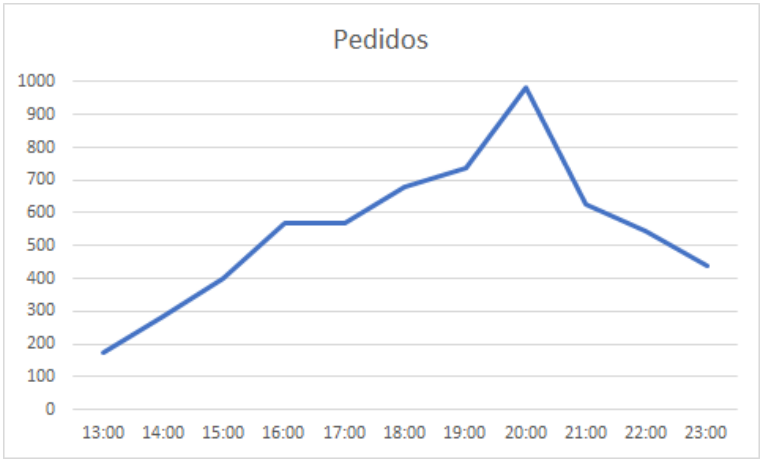

Figura 9 - Exemplos de visualização de métrica para quantidade de pedidos

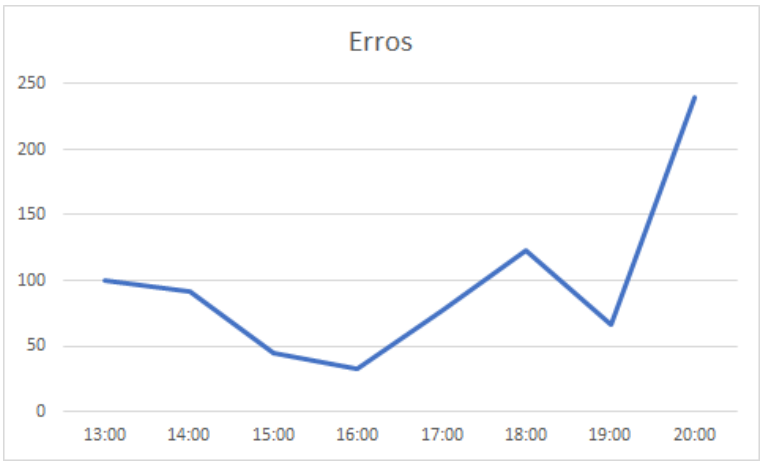

Figura 10 - Exemplos de visualização de métrica para quantidade de erros

4. *Tracing*

O *tracing* é um método que pode ser utilizado na análise e monitorização de aplicações de microsserviços para detetar erros e anomalias nos serviços [9].

Um *trace* representa o caminho percorrido por um pedido realizado no sistema, ao longo dos vários serviços que o constituem. É constituído por unidades de trabalho individuais designadas *spans*.

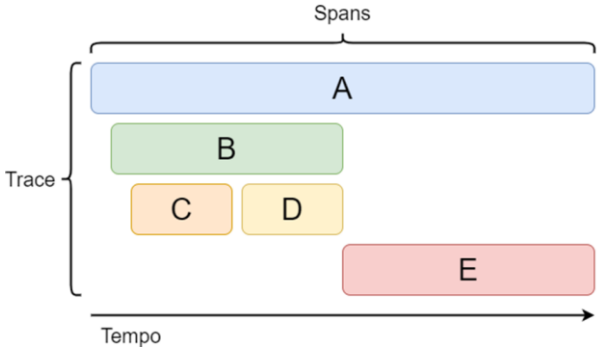

Figura 11 - Exemplo da visualização de um *trace*

A figura 11 ilustra a anatomia de um *trace* constituído por cinco *spans*: A, B, C, D e E.

Um *span* é uma estrutura de dados que encapsula informações sobre a operação que foi realizada. Existem várias *frameworks* de *tracing*, por isso a estrutura de um *span* não é necessariamente a mesma entre todas. A especificação de *span* apresentada faz parte da *framework* OpenTelemetry [10]. Nesta *framework*, as propriedades que caracterizam um *span* são o nome da operação realizada, *timestamp* do início e fim, um objeto *SpanContext*, um conjunto de atributos e uma lista de eventos.

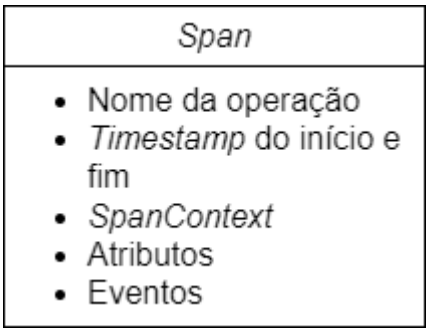

Figura 12 - Estrutura de um *span* da *framework* OpenTelemetry

O objeto *SpanContext* é propagado de um *span* pai para o seu filho. É constituído pelo identificador do *span* e do *trace* a que pertence. A relação pai-filho entre *spans* permite construir um *trace*. Os atributos são estruturas no formato *key-value*. Um atributo qualifica os dados permitindo que sejam agrupados ou filtrados. Informações como o endereço do serviço da sua origem podem ser guardadas como um atributo. Os eventos são compostos por um nome e *timestamp*. Identificam uma ocorrência durante a execução do *span* num determinado momento [10].

O *tracing* é um método que gera uma grande quantidade de *traces* e por isso é necessário o uso de ferramentas para os filtrar e visualizar. Exemplos de ferramentas de visualização e análise de *tracing* disponíveis:

1. Jaeger [11]
2. SigNoz [12]
3. Zipkin [13]
4. Dynatrace [14]
5. New Relic [15]
6. Honeycomb [16]
7. Lightstep [17]
8. Instana [18]
9. Datadog [19]
10. Splunk [20]

Capítulo 3
Ferramenta de visualização de microsserviços

Como foi referido no final do Capítulo 2, uma aplicação de microsserviços monitorizada através do método de *tracing* requer o uso de ferramentas que permita filtrar e visualizar *traces*. Existem algumas aplicações no mercado que satisfazem esta necessidade, no entanto não oferecem o formato de visualização ideal para a sua exploração.

μViz é a solução proposta para visualizar e monitorizar aplicações construídas com a arquitetura de microsserviços. Neste capítulo descrevo as abordagens anteriores deste projeto e apresento os *mockups* das vistas e funcionalidades da nova versão.

3.1 Projetos anteriores

O primeiro projeto foi desenvolvido pelo aluno Joel Fernandes e surge para investigar a ideia de desenvolver uma ferramenta para visualizar sistemas construídos sob a arquitetura de microsserviços. A aplicação final não possuía muitas funcionalidades e foi tomada a decisão de descontinuar esta versão e criar outra.

O segundo projeto foi desenvolvido pelo aluno André Melo. Foi considerado bem-sucedido, no entanto não foi implementada uma das funcionalidades fundamentais alusiva à forma de como o utilizador observa o sistema. Esta funcionalidade consistia em aumentar o *zoom* num microsserviço desenhado e surgirem desenhadas as suas instâncias.

3.2 Projeto atual

A nova versão, μViz, é independentes das anteriores. A aluna Sara Silva descreveu as suas funcionalidades e desenvolveu *mockups* das vistas da aplicação [21].

A tabela 2 clarifica as suas contribuições e as minhas para este projeto.

Tabela 2 - Quadro de tarefas

Tarefa	Responsável
Mockups	Sara Silva
Descrição das funcionalidades	Sara Silva
Casos de Uso	Igor Moreira
Requisitos Funcionais	Igor Moreira
Requisitos Não Funcionais	Igor Moreira

Implementação	Igor Moreira
Testes	Igor Moreira

A solução atual propõe quatro vistas que possibilitam a análise e visualização dos microsserviços de um sistema, através da exploração dos dados de *traces* obtidos pelo método de *tracing* [21].

3.2.1 Vista de microsserviços

Como foi referido no Capítulo 2, um *trace* é composto por *spans* que identificam o serviço da sua origem e incluem referências para outros *spans*, estabelecendo uma relação entre estes elementos que representa a comunicação entre serviços.

A vista de microsserviços [22] apresenta a abstração de alto nível da arquitetura do sistema através dos seus microsserviços e respetivas ligações. Estes elementos são representados num grafo. Nesta estrutura, os nós representam microsserviços e as arestas a comunicação entre estes elementos.

Os nós do grafo apresentam-se coloridos de acordo com o estado atribuído do microsserviço que representam:

- Cinzento: não existe informação do estado do microsserviço;
- Verde: não existem erros nas comunicações em que o microsserviço participa nem problemas detetados nas suas métricas;
- Amarelo: possível existência de problemas no funcionamento do microsserviço;
- Vermelho: existem problemas no microsserviço.

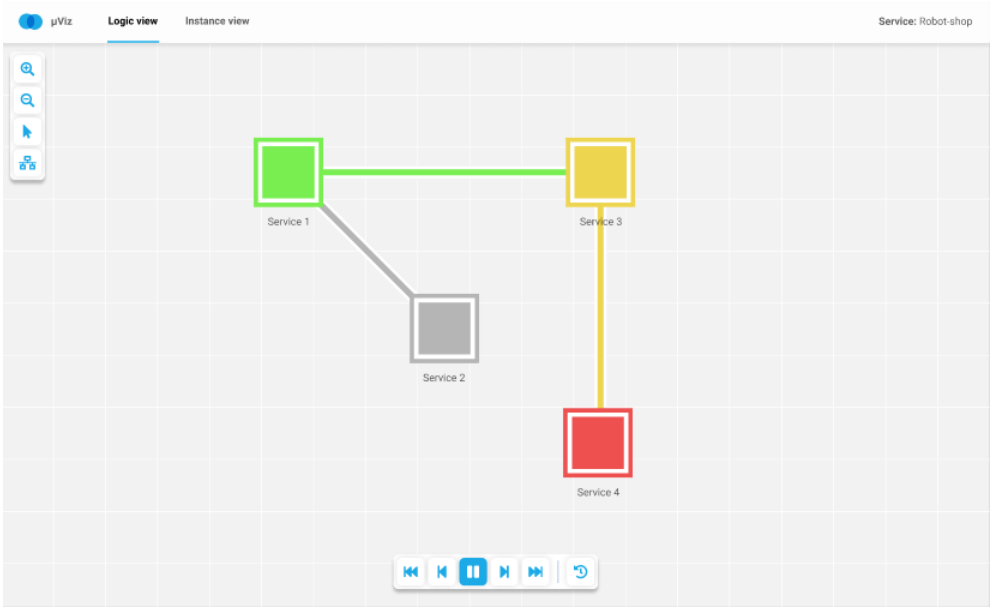

Figura 13 - *Mockup* da vista lógica

Como é possível observar na figura 13, a disposição dos elementos nesta vista permite visualizar arquitetura do sistema, a comunicação entre microsserviços e identificar facilmente a origem de anomalias que requerem intervenção.

O utilizador pode clicar nos microsserviços para aceder a mais informações que são exibidas numa janela como a figura 14 ilustra.

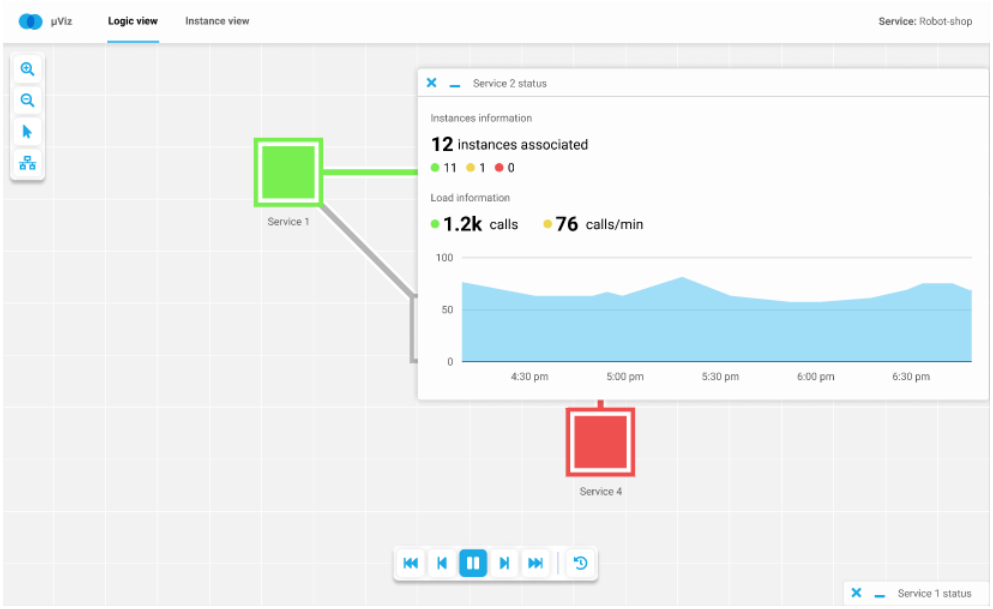

Figura 14 - *Mockup* da vista lógica com exibição de detalhes de um microsserviço

3.2.2 Vista de instâncias

A vista de instâncias [23] apresenta as instâncias de um microsserviço. Esta está integrada na vista de microsserviços. A transição entre vistas é realizada através de *zoom in* e *zoom out*. Nesta vista os elementos desenhados seguem o mesmo esquema de cores utilizado na vista de microsserviços, com uma ligeira alteração no seu significado:

- Cinzento: não existe informação relativa à instância;
- Verde: não existem erros nas comunicações em que a instância participa nem problemas detetados nas suas métricas;
- Amarelo: possibilidade de existirem anomalias na instância;
- Vermelho: existem problemas na instância que afetam pedidos e o sistema.

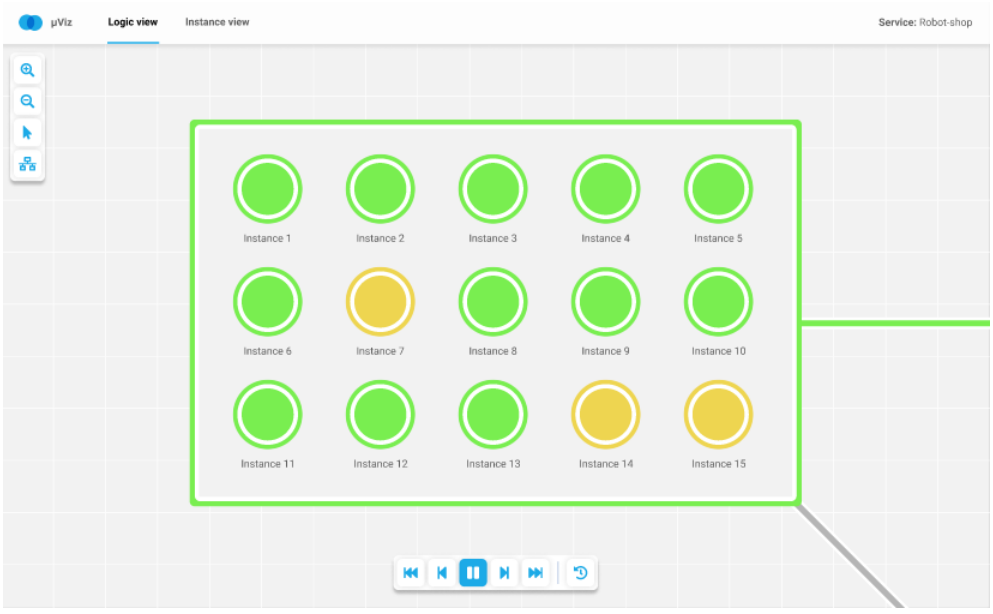

Figura 15 - *Mockup* da vista física

A figura 15 apresenta o *mockup* desta vista. Esta organização permite ao utilizador observar o estado das instâncias e identificar rapidamente quais as que requerem intervenção.

O utilizador pode aceder a mais informações relativas a uma instância ao clicar nela, como mostra a figura 16.

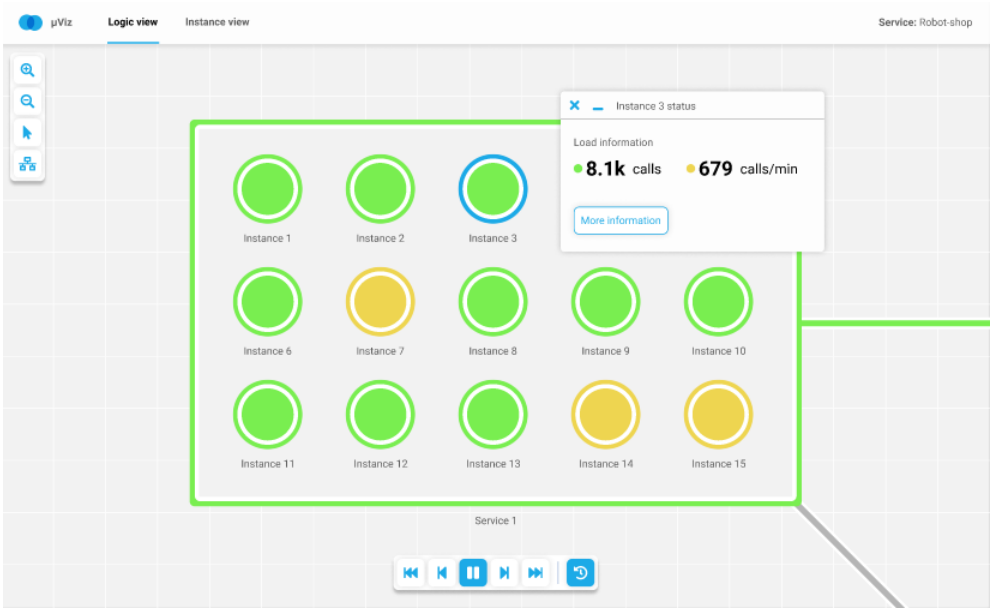

Figura 16 - *Mockup* da vista física com exibição de detalhes de uma instância

3.2.3 Vista de *traces*

A vista de *traces* [24] apresenta a união dos *traces* da aplicação através dos seus *spans*. O utilizador pode analisar o estado de cada *span* e identificar facilmente a existência de algum problema no sistema. Os *traces* são desenhados num grafo onde um nó representa uma instância de um microsserviço e uma aresta a relação entre instâncias (*spans*). As cores das instâncias nesta vista têm o mesmo significado das cores utilizadas na vista física. A figura 17 ilustra o *mockup* desta vista.

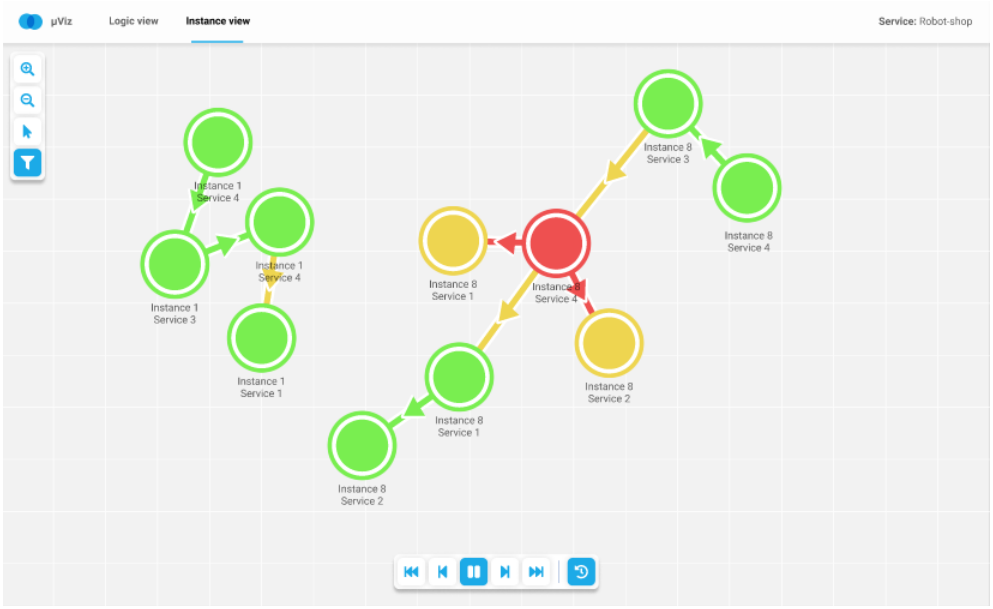

Figura 17 - *Mockup* da vista de *traces*

O utilizador pode filtrar os *traces* por microsserviços ou instâncias e apenas são apresentados *traces* que contêm *spans* de acordo com o filtro estabelecido. Para aceder ao filtro, o utilizador deve clicar no botão do menu lateral esquerdo (assinalado com o retângulo vermelho na figura 18).

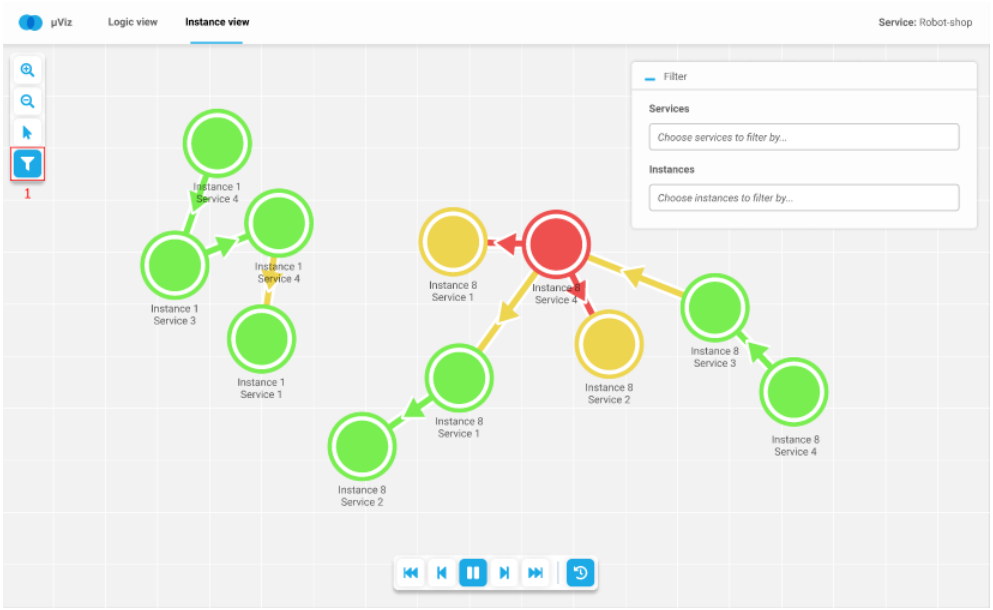

Figura 18 - *Mockup* da vista de *traces* com filtro

3.2.4 Vista de *workflows*

Esta vista [25] apresenta os *workflows* do sistema. Um *workflow* representa os *traces* do sistema agrupados por *spans* comuns, ou seja, os *traces* que percorrem as mesmas instâncias de microsserviços em diferentes instantes. A figura 19 ilustra o conceito de *workflow*.

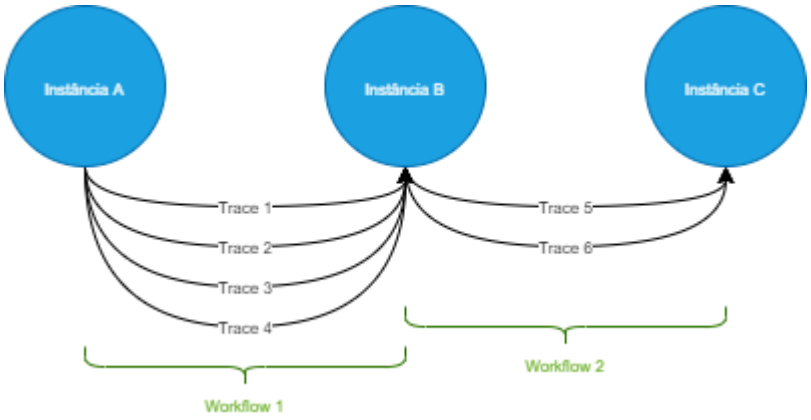

Figura 19 – *Workflow*

A figura 20 apresenta o *mockup* desta vista. Para a ativar, o utilizador deve clicar no botão do menu lateral esquerdo (assinalado com o retângulo vermelho na imagem). Quando é ativada surge em cima de cada microsserviço a quantidade de *workflows* que têm início nele.

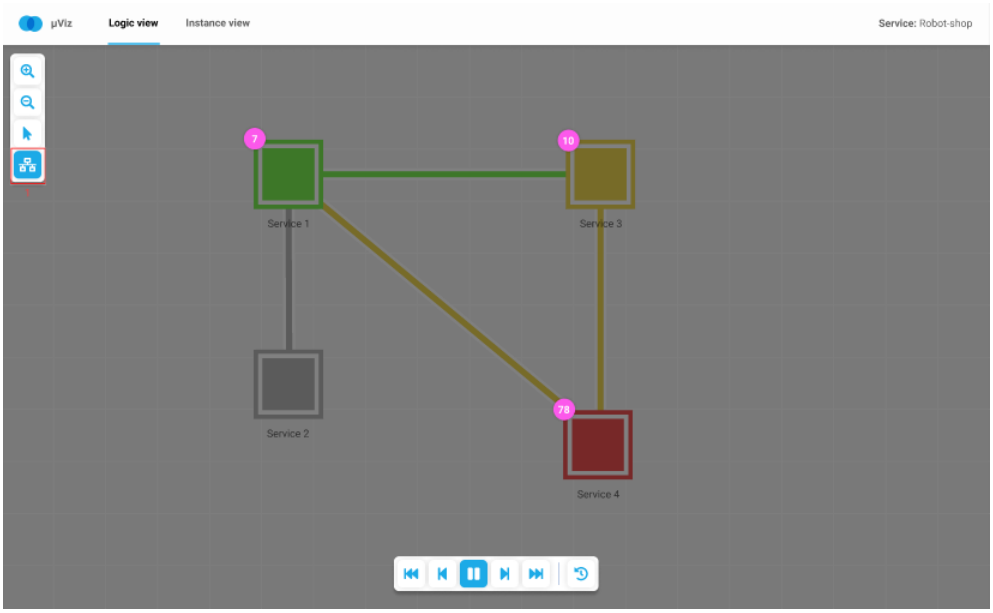

Figura 20 - *Mockup* da vista de *workflows*

Quando o utilizador clica num microsserviço, surge uma janela modal que lista os *workflows* que se iniciam a partir dele. No campo "*filter by*" o utilizador pode filtrar os *workflows* mostrados introduzindo o nome de microsserviços.

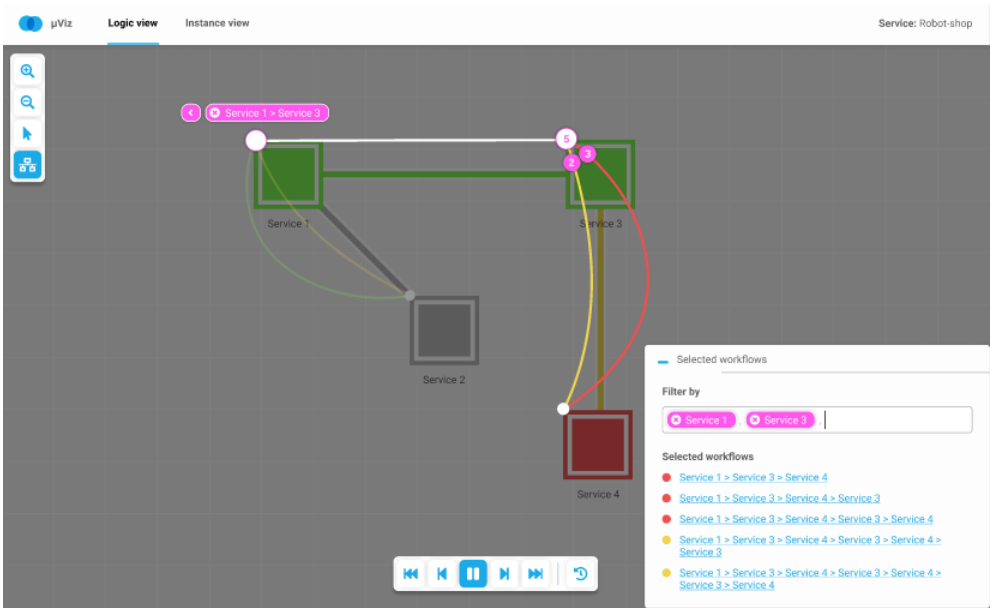

Figura 21 - *Mockup* de filtro na vista de *workflows*

Capítulo 4
Casos de uso

Esta secção aborda os casos de uso da aplicação µViz. Estes apresentam as ações que podem ser efetuadas por uma entidade no sistema e descrevem o comportamento deste perante as mesmas. Os casos de uso foram concebidos com base na descrição das funcionalidades apresentadas no trabalho da aluna Sara Silva.

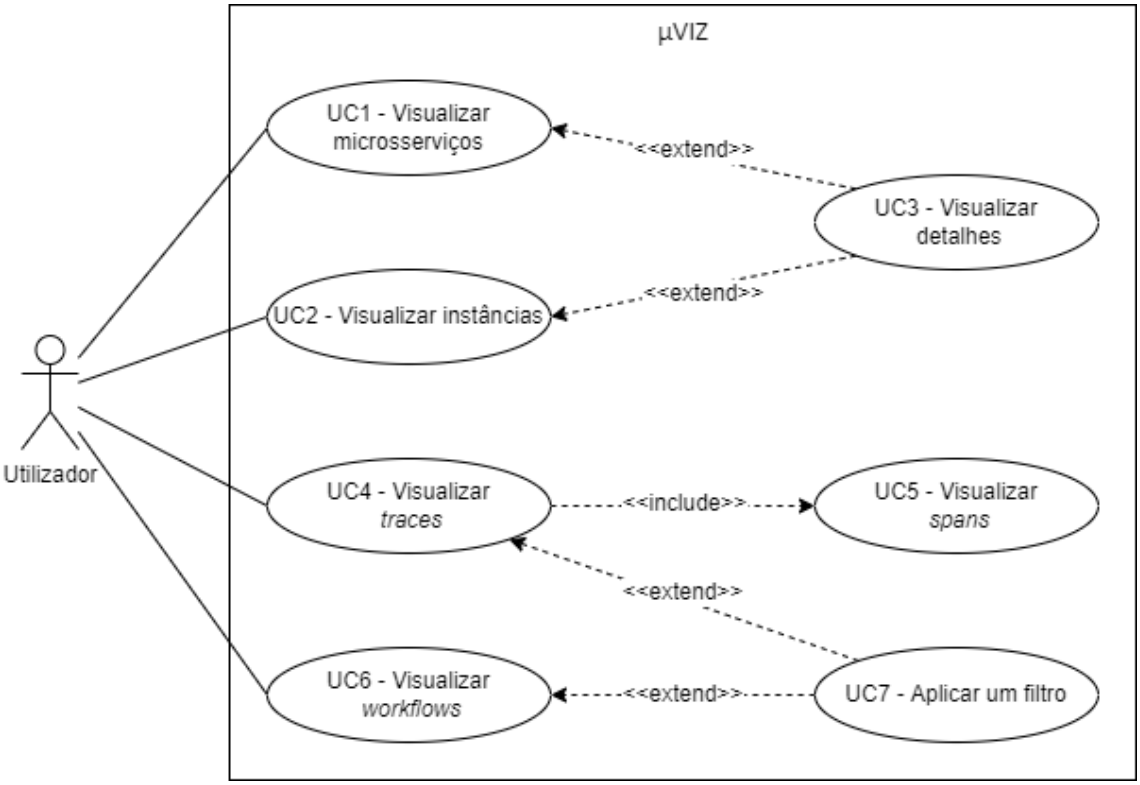

Figura 22 - Diagrama de casos de uso

Tabela 3 - Caso de uso UC1: Visualizar microsserviços

Nome	UC1: Visualizar microsserviços
Atores	Utilizador
Âmbito	O utilizador interage com o sistema para observar os microsserviços da aplicação
Pré-condições	O utilizador tem acesso a um computador com *browser* instalado e ligação à internet
Pós-condições	O sistema apresenta os microsserviços da aplicação ao utilizador

Cenário principal de sucesso	1 - O utilizador entra no sistema 2 - O sistema apresenta os microsserviços (página inicial) 3 - O utilizador vê os microsserviços
Cenários alternativos	2a - Não existe informação relativa aos microsserviços do sistema 1 - O sistema apresenta uma mensagem que informa o utilizador do problema

Tabela 4 - Caso de uso UC2: Visualizar instâncias

Nome	UC2: Visualizar instâncias
Atores	Utilizador
Âmbito	O utilizador interage com o sistema para observar as instâncias de um microsserviço
Pré-condições	O utilizador tem acesso a um computador com *browser* e ligação à internet
Pós-condições	O sistema apresenta as instâncias do microsserviço ao utilizador
Cenário principal de sucesso	1 - O utilizador entra no sistema 2 - O sistema apresenta os microsserviços (página inicial) 3 - O utilizador efetua *zoom in* num microsserviço 4 - O sistema apresenta as instâncias do microsserviço 3 - O utilizador vê as instâncias
Cenários alternativos	4a - Não existe informação relativa às instâncias do microsserviço 1 - O sistema apresenta uma mensagem que informa o utilizador do problema

Tabela 5 - Caso de uso UC3: Visualizar detalhes

Nome	UC3: Visualizar detalhes
Atores	Utilizador
Âmbito	O utilizador interage com o sistema para observar mais detalhes relativos a um microsserviço ou às suas instâncias
Pré-condições	O utilizador tem acesso a um computador com *browser* e ligação à internet
Pós-condições	O sistema apresenta detalhes ao utilizador
Cenário principal de sucesso	1 - O utilizador entra no sistema

	2 - O sistema apresenta os microsserviços (página inicial) 3 - O utilizador seleciona um microsserviço 4 - O utilizador clica no botão que surge em cima do microsserviço selecionado 5 - O sistema mostra os detalhes do microsserviço 6 - O utilizador vê os detalhes
Cenários alternativos	3a - O utilizador efetua *zoom in* num microsserviço 1 - O sistema apresenta as instâncias do microsserviço 2 - O utilizador seleciona uma instância 3 - O sistema mostra os detalhes da instância 4 - O utilizador vê os detalhes

Tabela 6 - Caso de uso UC4: Visualizar *traces*

Nome	UC4: Visualizar *traces*
Atores	Utilizador
Âmbito	O utilizador interage com o sistema para observar os *traces* da aplicação
Pré-condições	O utilizador tem acesso a um computador com *browser* e ligação à internet
Pós-condições	O sistema apresenta os *traces* da aplicação ao utilizador
Cenário principal de sucesso	1 - O utilizador entra no sistema 2 - O sistema apresenta os microsserviços (página inicial) 3 - O utilizador clica no botão do menu para navegar para a vista de *traces* 4 - O sistema apresenta os *traces* da aplicação 5 - O utilizador vê os *traces*
Cenários alternativos	4a - Não existe informação relativa a *traces* 1 – O sistema apresenta uma mensagem que informa o utilizador do problema

Tabela 7 - Caso de uso UC5: Visualizar *spans*

Nome	UC5: Visualizar *spans*
Atores	Utilizador
Âmbito	O utilizador interage com o sistema para observar os *spans* dos *traces*
Pré-condições	O utilizador realiza o caso de uso UC4 – Ver *traces*

Pós-condições	O sistema apresenta os *spans* dos *traces* ao utilizador
Cenário principal de sucesso	1 - O sistema apresenta os *spans* dos *traces* 2 - O utilizador vê os *spans*
Cenários alternativos	Não tem

Tabela 8 - Caso de uso UC6: Visualizar *workflows*

Nome	UC6: Visualizar *workflows*
Atores	Utilizador
Âmbito	O utilizador interage com o sistema para observar os *workflows* da aplicação
Pré-condições	O utilizador tem acesso a um computador com *browser* e ligação à internet
Pós-condições	O sistema apresenta os *workflows* da aplicação ao utilizador
Cenário principal de sucesso	1 - O utilizador entra no sistema 2 - O sistema apresenta os microsserviços (página inicial) 3 - O utilizador clica no botão do menu para ativar a vista de *workflows* 4 - O sistema apresenta em cima de cada microsserviço um botão com o número de *workflows* que têm início nele 5 - O utilizador clica no botão 6 - O sistema apresenta uma janela modal com a lista *traces* que fazem parte do *workflow* 7 - O sistema apresenta os *workflows*
Cenários alternativos	3a - Não existe informação relativa a *workflows* 1 - O sistema apresenta uma mensagem que informa o utilizador do problema

Tabela 9 - Caso de uso UC7: Aplicar um filtro

Nome	UC7: Aplicar um filtro
Atores	Utilizador
Âmbito	O utilizador interage com o sistema para filtrar *traces* ou *workflows* apresentados
Pré-condições	O utilizador tem acesso a um computador com *browser* e ligação à internet
Pós-condições	O sistema apresenta o conteúdo filtrado ao utilizador

Cenário principal de sucesso	1 - O utilizador entra no sistema 2 - O sistema apresenta os microsserviços (página inicial) 3 - O utilizador clica no botão do menu para ativar a vista de *workflows* 4 - O sistema apresenta em cima de cada microsserviço um botão com o número de *workflows* que têm início nele 5 - O utilizador clica no botão 6 - O sistema apresenta uma janela modal com a lista *traces* que fazem parte do *workflow* 7 - O utilizador escreve um *span* no campo de filtro da janela modal 8 - O sistema apresenta os *workflows* filtrados
Cenários alternativos	3a - O utilizador clica no botão do menu para navegar para a vista de *traces* 1 - O sistema apresenta os *traces* da aplicação 2 - O utilizador clica no botão para filtrar no menu 3 - O sistema apresenta uma janela modal com opções de filtro 4 - O utilizador filtra por microsserviços ou instâncias 5 - O sistema apresenta os *traces* filtrados 7a - O utilizador seleciona iterativamente microsserviços do grafo para criar um *span* 1 - O sistema apresenta os *workflows* filtrados

Capítulo 5
Requisitos

Neste capítulo são apresentados os requisitos funcionais e não funcionais da aplicação μViz.

5.1 Requisitos funcionais

Os requisitos funcionais descrevem as funcionalidades disponibilizadas pela aplicação. Os requisitos descritos nas próximas tabelas foram baseados na descrição de funcionalidades e *mockups* produzidos pela aluna Sara Silva.

As tabelas apresentam cada requisito com um identificador, nome, descrição e prioridade. O método de priorização utilizado é o *MoSCoW*, que atribui um nível de prioridade a um requisito de acordo com a sua importância num sistema. Existem quatro níveis:

- *Must Have*: é obrigatório cumprir o requisito. A funcionalidade é essencial e deve ser implementada;
- *Should Have*: a funcionalidade é importante e adiciona valor à aplicação, no entanto o seu cumprimento não é obrigatório. Se não for implementado na versão atual, deve ser na próxima;
- *Could Have*: a funcionalidade é opcional;
- *Won't Have This Time*: a funcionalidade não é importante para versão atual da aplicação. Pode ser implementada no futuro.

Tabela 10 - Requisito funcional RF01: Vista de microsserviços

ID	RF01
Nome	Vista de microsserviços
Descrição	O sistema deve apresentar os microsserviços da aplicação num grafo, em que um nó representa um microsserviço e uma aresta a ligação entre um ou vários microsserviços que comunicam entre si. Os elementos devem ser desenhados com uma cor de acordo com o seu estado como se segue: 1. cinzento: não existe informação; 2. verde: não existem problemas; 3. amarelo: podem existir problemas; 4. vermelho: existem problemas.
Prioridade	*Must Have*

Tabela 11 - Requisito funcional RF02: Seleção de microsserviços

ID	RF02

Nome	Seleção de microsserviços
Descrição	Quando o utilizador clica num ou mais microsserviços, o sistema deve apresentar uma modal com as seguintes informações: 1. quantidade total de instâncias associadas; 2. quantidade de instâncias agrupadas pela sua saúde; 3. informação de carga: quantidade total de pedidos e quantidade de pedidos efetuados por minuto; 4. gráfico com a quantidade de pedidos efetuados por minuto.
Prioridade	*Must Have*

Tabela 12 - Requisito funcional RF03: Vista de instâncias

ID	RF03
Nome	Vista de instâncias
Descrição	O sistema deve apresentar as instâncias de um microsserviço. Cada instância é ilustrada com uma cor de acordo com o seu estado. Ao clicar numa instância, deve surgir uma janela modal com a quantidade total de pedidos e quantidade de pedidos por minuto.
Prioridade	*Must Have*

Tabela 13 - Requisito funcional RF04: Transição entre a vista de microsserviços e a vista de instâncias

ID	RF04
Nome	Transição entre a vista de microsserviços e a vista de instâncias
Descrição	O sistema deve transitar da vista de microsserviços para a vista de instâncias quando o utilizador executa a ação *zoom in* e o inverso através de *zoom out*.
Prioridade	*Must Have*

Tabela 14 - Requisito funcional RF05: Vista de *traces*

ID	RF05
Nome	Vista de *traces*
Descrição	O sistema deve apresentar uma vista com a união de todos *traces* da aplicação representados num grafo em que cada nó representa um *span* e aresta a relação entre *spans*. Os elementos devem ser desenhados com uma cor de acordo com o seu estado como se segue: 1. cinzento: não existe informação; 2. verde: não existem problemas; 3. amarelo: podem existir problemas; 4. vermelho: existem problemas.
Prioridade	*Must Have*

Tabela 15 - Requisito funcional RF06: Filtro na vista de *traces*

ID	RF06
Nome	Filtro na vista de *traces*
Descrição	O sistema deve permitir que o utilizador filtre *traces* que contêm um determinado microsserviço e/ou instância.
Prioridade	*Must Have*

Tabela 16 - Requisito funcional RF07: Vista de *workflows*

ID	RF07
Nome	Vista de *workflows*
Descrição	O sistema deve apresentar ao utilizador uma vista de *workflows* (grupo de *traces* com *spans* em comum).
Prioridade	*Should Have*

Tabela 17 - Requisito funcional RF08: Filtro na vista de *workflows*

ID	RF08
Nome	Filtro na vista de *workflows*
Descrição	O sistema deve permitir que o utilizador filtre *workflows* por microsserviços e/ou instâncias.
Prioridade	*Should Have*

Tabela 18 - Requisito funcional RF09: Seletor de data

ID	RF09
Nome	Seletor de data
Descrição	O sistema deve apresentar em todas as vistas um menu na parte inferior do ecrã que permite ao utilizador visualizar o estado da aplicação numa determinada data.
Prioridade	*Should Have*

Tabela 19 - Requisito funcional RF10: *Zoom*

ID	RF10
Nome	*Zoom*
Descrição	O sistema deve apresentar em todas as vistas um menu lateral esquerdo com opções de *zoom in* e *zoom out* que permitem aumentar ou diminuir o tamanho dos elementos apresentados.
Prioridade	*Must Have*

5.2 Requisitos não funcionais

Os requisitos não funcionais descrevem o comportamento do sistema e a experiência do utilizador.

Tabela 20 - Requisito não funcional RNR01: Usabilidade

ID	RNR01
Nome	Usabilidade
Objetivos	1 - A aplicação deve ser fácil de utilizar: o utilizador deve atingir sem dificuldade os seus objetivos utilizando as funcionalidades disponibilizadas. 2 - A interface gráfica deve apresentar uma estrutura clara: não deve apresentar muitas opções para não confundir o utilizador. 3 - As funcionalidades devem ser facilmente acessíveis pelos utilizadores. 4 - O utilizador deve ser informado quando surgem erros no sistema.

Tabela 21 - Requisito não funcional RNR02: Desempenho

ID	RNR02
Nome	Desempenho
Objetivos	1 - A aplicação deve apresentar resultados dentro de intervalos de tempo aceitáveis (não deve demorar mais que um minuto). 2 - A aplicação não sem apresentar quebras notáveis no seu desempenho, como diminuição da taxa de *frames* por segundo durante a sua utilização.

Capítulo 6
Implementação

Este capítulo apresenta as tecnologias utilizadas na implementação da ferramenta μViz, um protótipo criado no primeiro semestre, a aplicação final e respetivos testes.

6.1 Tecnologias

O ambiente de execução da aplicação μViz é um *browser* e a linguagem utilizada é *JavaScript*. As tecnologias apresentadas são *open source* e possibilitam a implementação das funcionalidades pedidas.

6.1.1 Tecnologias para a interface do utilizador

React

React é uma biblioteca *JavaScript* utilizada no desenvolvimento *front-end* das aplicações *web*. A sua popularidade tem crescido e é suportada por uma comunidade que contribui com melhorias constantemente. Esta biblioteca foi utilizada para desenvolver a interface gráfica da aplicação. A figura 23 apresenta um *mockup* de uma vista em que todos os elementos gráficos são criados com *React* [26].

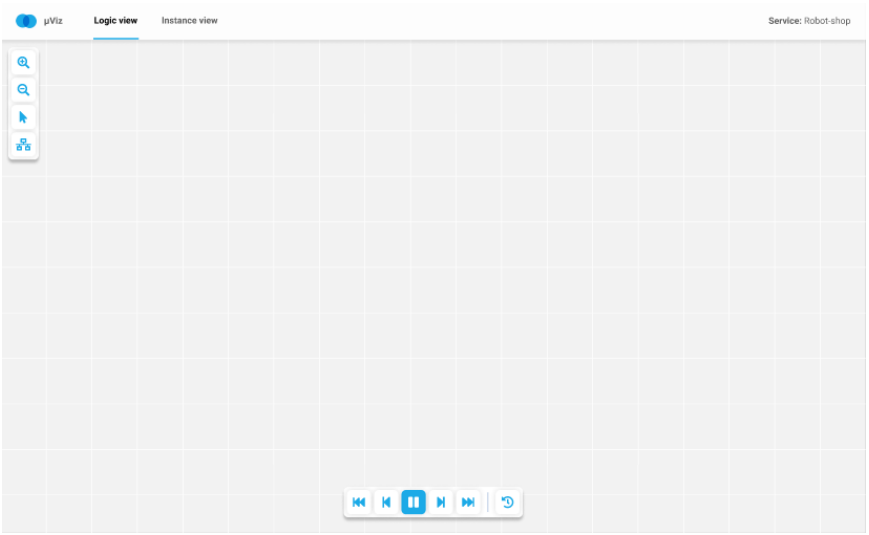

Figura 23 - *Mockup* base da aplicação

6.1.2 Tecnologias para desenho em 2D

A aplicação é composta por vistas que apresentam grafos. Foi realizada uma pesquisa de algumas tecnologias que possibilitam o desenho desta estrutura num plano 2D.

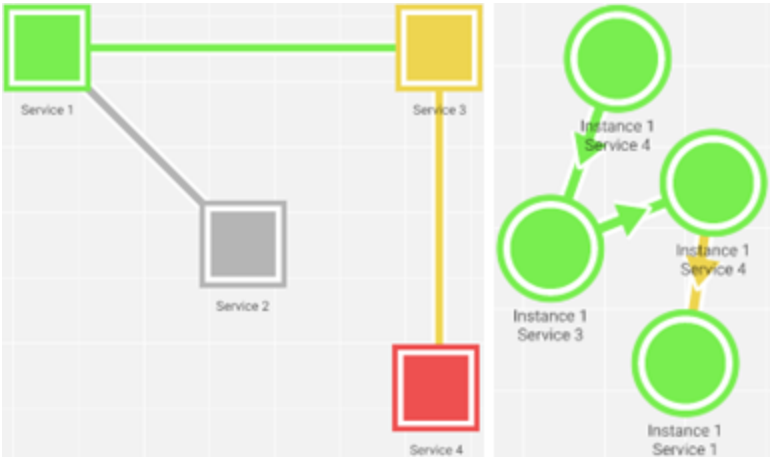

Figura 24 - Exemplo da utilização de grafos na aplicação

WebGL

WebGL oferece bastante controlo ao programador, no entanto o processo de desenho de elementos é considerado bastante complexo. A sua utilização não foi ponderada devido ao tempo requerido para a sua aprendizagem [27].

p5.js

p5.js é uma biblioteca simples e considerada ideal para introduzir programadores ao desenho de elementos gráficos em aplicações *web*. Não foi utilizada por não oferecer suporte completo a determinados eventos, como por exemplo o registo de cliques nos elementos desenhados [28].

PixiJS

PixiJS possibilita o desenho de elementos e suporta vários eventos de interação sobre os eles. É uma biblioteca rica em funcionalidades que a tornam a escolha ideal para a construção da ferramenta. Esta biblioteca foi utilizada no projeto. [29].

6.1.3 Tecnologias para apresentar dados em gráficos

Alguns dados são exibidos em gráficos, por isso é necessário uma tecnologia que o possibilite.

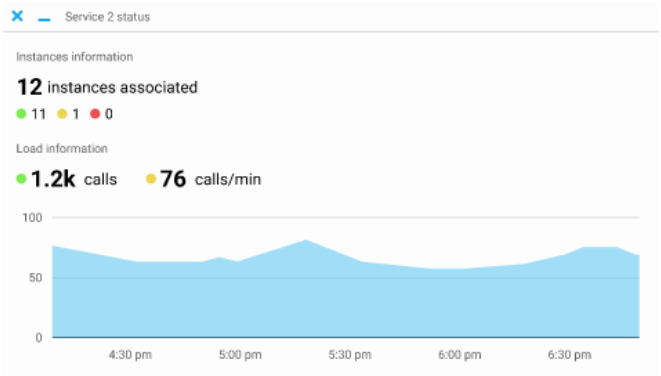

Figura 25 - Exemplo da utilização de gráficos na aplicação

D3.js, Charts.js e Recharts

Estas tecnologias permitem mostrar dados em vários tipos de gráficos diferentes. Qualquer uma destas opções é considerada viável e fácil de implementar. *D3.js* é a biblioteca mais completa e oferece várias funcionalidades que não são necessárias [30]. *Charts.js* e *Recharts* são mais compactas, mas possuem os tipos de gráficos necessários [31] [32]. *Recharts* foi a biblioteca escolhida.

6.2 Protótipo

No primeiro semestre foi também desenvolvido um simples protótipo para explorar a linguagem *JavaScript* e as bibliotecas *React* e *PixiJS*.

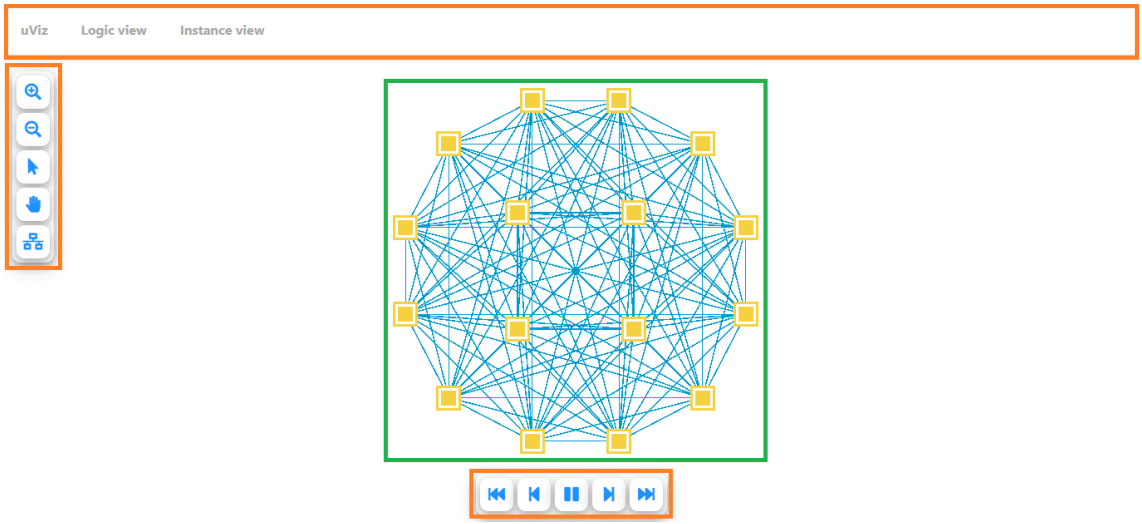

Figura 26 - Protótipo desenvolvido

A figura 26 apresenta uma captura de ecrã do protótipo desenvolvido. Os componentes assinalados com retângulos a laranja foram desenvolvidos com *React*. O elemento na parte superior do ecrã é o menu de navegação e serve para o utilizador navegar entre as vistas da aplicação. O menu lateral esquerdo possui as opções de *zoom in/out* (primeiro e segundo ícone com forma de lupa) e mover o grafo (quarto ícone com forma de mão). Estas funcionalidades estão implementadas.

O grafo desenhado no centro da figura, assinalado com o retângulo verde, foi criado com *PixiJS*. As posições dos elementos do grafo no ecrã são calculadas com recurso a um algoritmo de desenho de grafos. Este algoritmo (*Force-Directed Drawing Algorithm*) [2] trata o grafo como um sistema físico de molas. As ligações entre os nós do grafo (arestas) funcionam como molas. Num sistema físico real, uma mola sujeita os elementos ligados às suas extremidades a uma força que os puxa para o seu centro. Este algoritmo aplica este conceito num grafo: uma aresta "puxa" os nós para o seu centro. Num grafo com múltiplos nós e múltiplas ligações entre eles, este processo é iterativo, ou seja, o algoritmo calcula as forças que as arestas exercem nos nós e atualiza as suas posições a cada iteração. Eventualmente, o sistema chega o seu estado final quanto atinge um ponto de equilíbrio e estabiliza quando não existem mais forças a aplicar.

6.3 Aplicação final

A tabela 22 mostra o sumário do cumprimento dos requisitos funcionais. Dado à falta de tempo, nem todas as funcionalidades foram implementadas.

Tabela 22 - Sumário da implementação dos requisitos funcionais

RF01: Vista de microsserviços	Implementado
RF02: Seleção de microsserviços	Implementado
RF03: Vista de instâncias	Implementado
RF04: Transição entre a vista de microsserviços e a vista de instâncias	Implementado
RF05: Vista de *traces*	Implementado
RF06: Filtro na vista de *traces*	Não implementado
RF07: Vista de *workflows*	Não implementado
RF08: Filtro na vista de *workflows*	Não implementado
RF09: Seletor de data	Não implementado
RF10: *Zoom*	Implementado

6.3.1 Capturas de ecrã

As figuras 27 a 31 apresentam o estado final da aplicação desenvolvida.

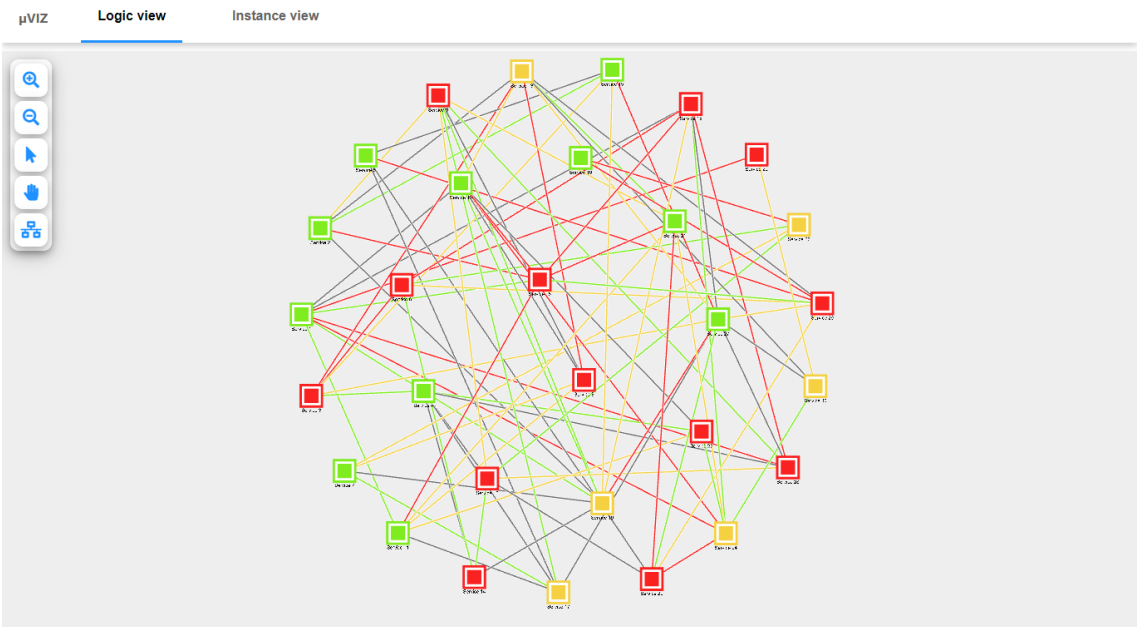

Figura 27 - Captura de ecrã da vista de microsserviços

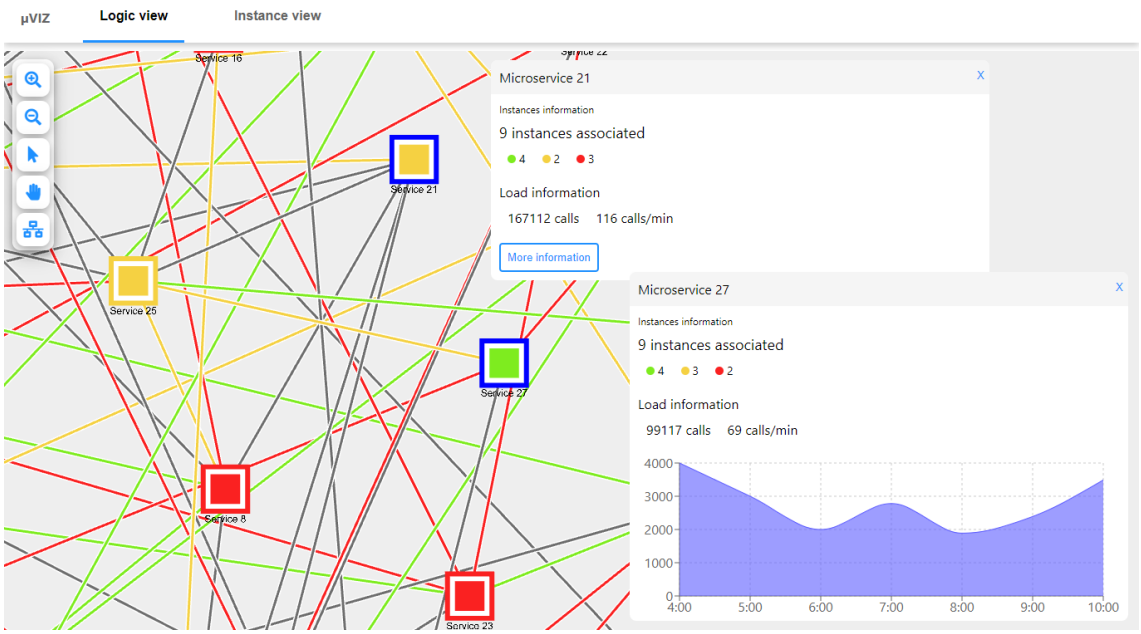

Figura 28 - Captura de ecrã da seleção simples de serviços na vista de microsserviços

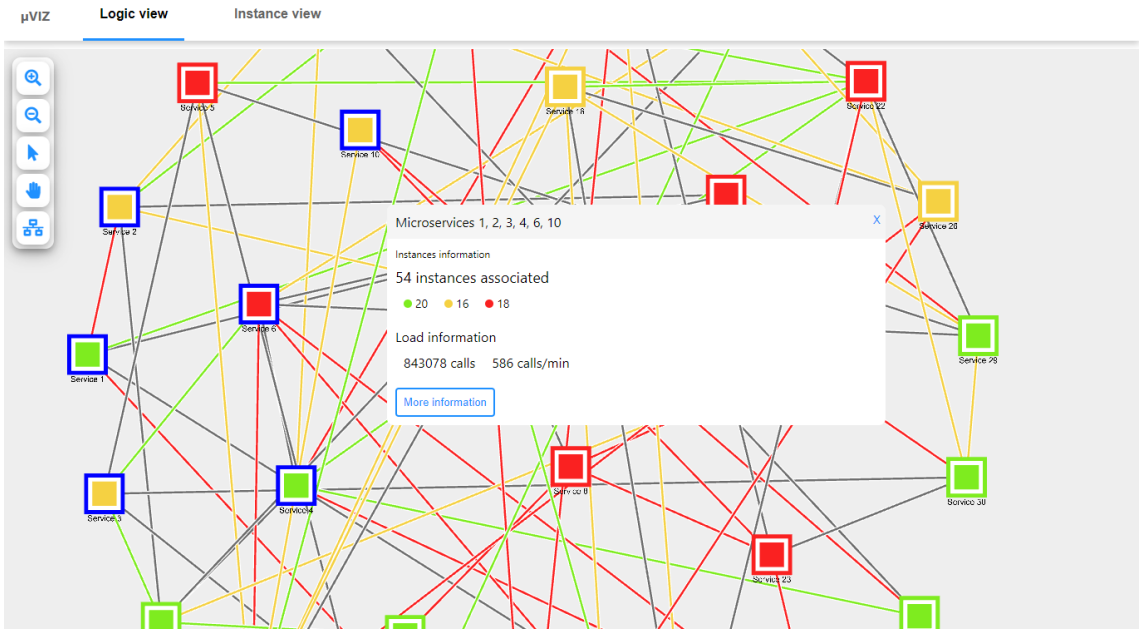

Figura 29 - Captura de ecrã da seleção múltipla de serviços vista de microsserviços

Figura 30 - Captura de ecrã da vista de instâncias com seleção

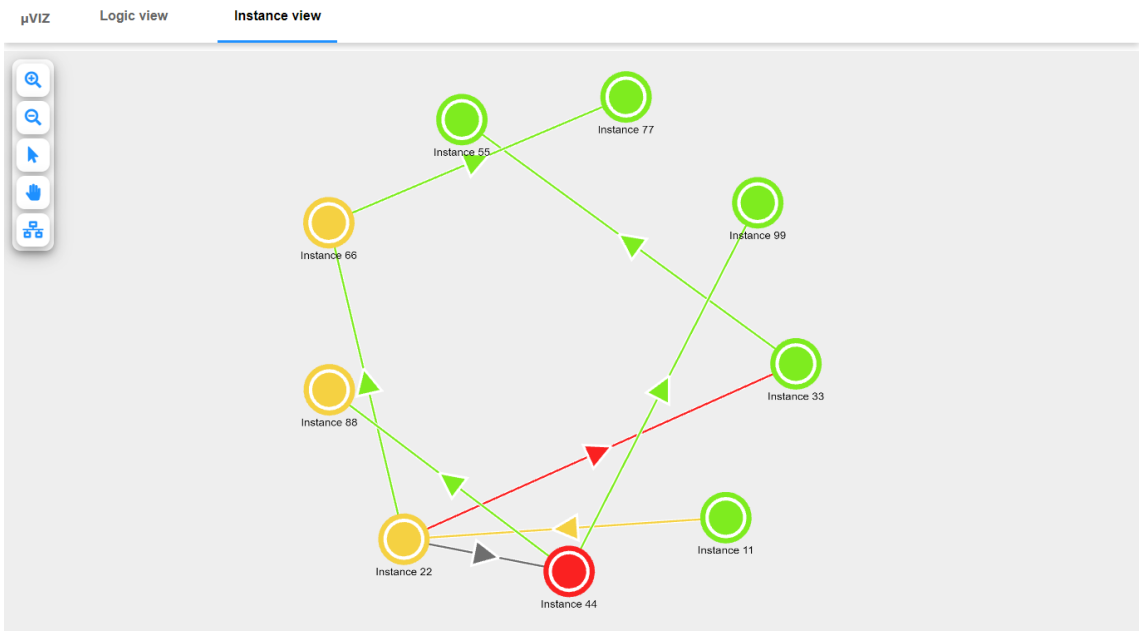

Figura 31 - Captura de ecrã da vista de *traces*

6.4 Testes

Depois da fase de desenvolvimento foram realizados testes a alguns requisitos funcionais e não funcionais da aplicação.

6.4.1 Testes aos requisitos funcionais

Os testes aos requisitos funcionais foram efetuados com recurso ao método *Black Box Testing*. O objetivo deste processo é verificar se a aplicação responde corretamente a um

determinado *input*. Um caso de teste é bem-sucedido quando o *output* obtido corresponde ao esperado de acordo com o *input*.

Tabela 23 – T01: Teste ao requisito funcional RF01

ID	T01
Nome	Teste ao requisito funcional RF01: Vista de microsserviços
Objetivo	Verificar se o grafo desenhado representa corretamente os microsserviços e as suas ligações.
Input	Microservices[ ] Microservice[0] ID: 1; Status: green; Connections[ ] Connection[0] { MicroserviceID: 2; Status: green; } Microservice[1] ID: 2; Status: green; Connections[ ] Connection[0] { MicroserviceID: 3; Status: green; } Microservice[2] ID: 3; Status: green; Connections[ ] Connection[0] { MicroserviceID: 4; Status: green; } Connection[1] { MicroserviceID: 5; Status: green; } Microservice[3] ID: 4; Status: green; Connections[ ] Connection[0] { MicroserviceID: 5; Status: green; } Microservice[4] ID: 5; Status: green; Connections[ ] Connection[0] { MicroserviceID: 6; Status: green; }

	Microservice[5] ID: 6; Status: green;
Resultado	Aprovado.

Tabela 24 - T02: Teste de limites ao requisito funcional RF01

ID	T02
Nome	Teste de limites ao requisito funcional RF01: Vista de microsserviços
Objetivo	Verificar se a aplicação desenha grafos com o número de nós no intervalo [2, 40].
Input	*Arrays* de microsserviços gerados aleatoriamente com tamanhos no intervalo [2, 40]
Resultado	Aprovado.

Tabela 25 - T03: Teste de limites ao requisito funcional RF01

ID	T03
Nome	Teste de limites ao requisito funcional RF01: Vista de microsserviços
Objetivo	Verificar se a aplicação desenha grafos com o número de nós superior a 40.
Input	*Arrays* de microsserviços gerados aleatoriamente com tamanhos superior a 40.
Resultado	Reprovado. Observação: a aplicação bloqueou.

Tabela 26 – T04: Teste ao requisito funcional RF02

ID	T04
Nome	Teste ao requisito funcional RF02: Seleção de microsserviços
Objetivo	Verificar se a aplicação apresenta uma janela modal com a informação correta do microsserviço selecionado.
Input	Microservices[ ] Microservice[0] ID: 1; NumberOfCalls: 110000; Instances[ ] Instance[0] { ID: 11; Status: green; } Instance[1] { ID: 12; Status: green; }

	Instance[2] { ID: 13; Status: green; } Connections[ ] Connection[0] { MicroserviceID: 2; Status: green; } Microservice[1] ID: 2; NumberOfCalls: 220000; Instances[ ] Instance[0] { ID: 21; Status: green; } Instance[1] { ID: 22; Status: yellow; } Connections[ ] Connection[0] { MicroserviceID: 3; Status: green; } Microservice[2] ID: 3; NumberOfCalls: 330000; Instances[ ] Instance[0] { ID: 31; Status: green; } Instance[1] { ID: 32; Status: yellow; } Instance[2] { ID: 33; Status: red; }
Resultado	Aprovado.

Tabela 27 – T05: Teste ao requisito funcional RF03

ID	T05
Nome	Teste ao requisito funcional RF03: Vista de instâncias
Objetivo	Verificar se os elementos desenhados no interior dos nós dos grafos que representam as instâncias de um microsserviço são mostrados corretamente.
Input	Microservices[ ] Microservice[0] Instances[ ] Instance[0] { ID: 11; Status: green; } Instance[1] { ID: 12; Status: yellow; } Instance[2] { ID: 13; Status: red; } Microservice[1] Instances[ ] Instance[0] { ID: 21; Status: green; }

	Microservice[2] Instances[ ] Instance[0] { ID: 31; Status: yellow; }
Resultado	Aprovado.

Tabela 28 – T06: Teste ao requisito funcional RF04

ID	T06
Nome	Teste ao requisito funcional RF04: Transição entre a vista de microsserviços e a vista de instâncias
Objetivo	Verificar se os elementos desenhados no interior dos nós dos grafos que representam as instâncias de um microsserviço surgem ou desaparecem ao efetuar *zoom in* e *zoom out*.
Input	*Scroll wheel*
Resultado	Aprovado.

Tabela 29 – T07: Teste ao requisito funcional RF05

ID	T07
Nome	Teste ao requisito funcional RF05: Vista de *traces*
Objetivo	Verificar se o grafo desenhado representa corretamente a união dos *traces* e relações entre *spans*.
Input	Traces[ ] Trace[0] Spans[ ] Span[0] { ID: 1; InstanceID: 11; ParentSpanID: null; Status: green; } Span[1] { ID: 2; InstanceID: 22; ParentSpanID: 1; Status: yellow; } Span[2] { ID: 3; InstanceID: 33; ParentSpanID: 2; Status: red; } Span[3] { ID: 4; InstanceID: 44; ParentSpanID: 2; Status: grey; } Span[4] { ID: 5; InstanceID: 55; ParentSpanID: 3; Status: green; } Trace[1] Spans[ ] Span[0] { ID: 2; InstanceID: 22; ParentSpanID: null; Status: yellow; } Span[1] { ID: 6; InstanceID: 66; ParentSpanID: 2; Status: green; } Span[2] { ID: 7; InstanceID: 77; ParentSpanID: 6; Status: green; } Trace[2] Spans[ ]

	Span[0] { ID: 4; InstanceID: 44; ParentSpanID: null; Status: yellow; } Span[1] { ID: 8; InstanceID: 88; ParentSpanID: 4; Status: green; } Span[2] { ID: 9; InstanceID: 99; ParentSpanID: 4; Status: green; }
Resultado	Aprovado.

Tabela 30 – T08: Teste ao requisito funcional RF10

ID	T08
Nome	Teste ao requisito funcional RF10: *Zoom*
Objetivo	Verificar se o tamanho dos elementos desenhados num grafo aumenta ou diminui de acordo com a funcionalidade de *zoom in* e *zoom out*.
Input	*Scroll wheel*
Resultado	Aprovado.

6.4.2 Testes aos requisitos não funcionais

Teste ao requisito não funcional RNR02: Desempenho

O desempenho da aplicação foi calculado com recurso à ferramenta Chrome DevTools que vem incorporada no *browser* Google Chrome. Este utensílio oferece várias funcionalidades que auxiliam os programadores na testagem das aplicações, inclusive uma para medir o tempo de carregamento de uma página *web*. O tempo é divido em cinco períodos: *loading, scripting, rendering, painting e system*. O período *scripting* corresponde ao tempo que o código da aplicação demorou a executar, por isso é o único que considero relevante para este teste.

O objetivo deste teste foi analisar o tempo de execução do código da aplicação consoante o número de nós de um grafo. Foram considerados cinco cenários diferentes em que apenas foi variado o número de nós do grafo. Os resultados apresentados nas figuras 32 a 36 resultam da média do tempo de dez execuções para cada experiência.

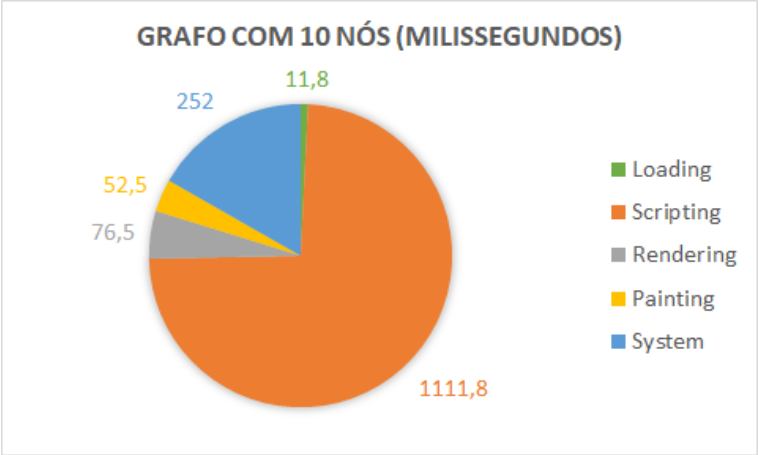

Figura 32 – Tempo de carregamento de um grafo com 10 nós

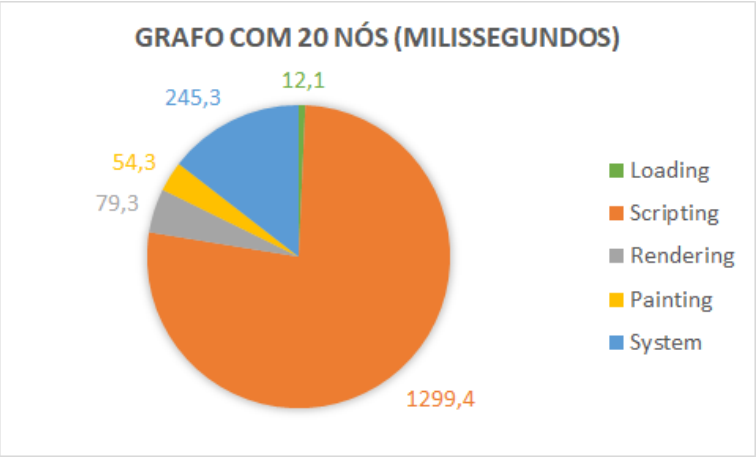

Figura 33 - Tempo de carregamento de um grafo com 20 nós

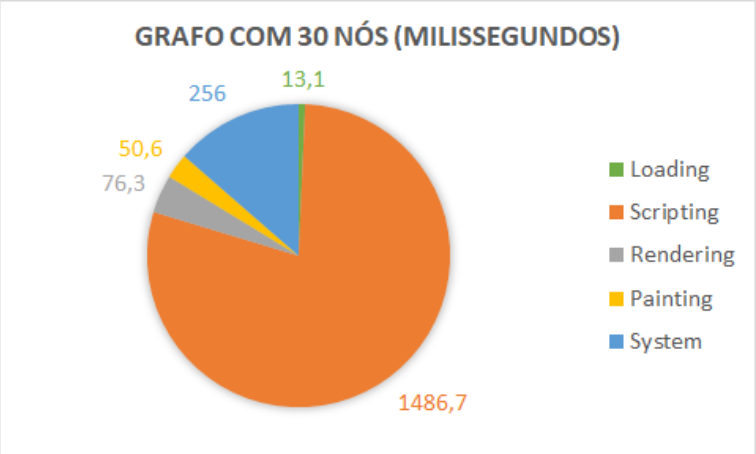

Figura 34 - Tempo de carregamento de um grafo com 30 nós

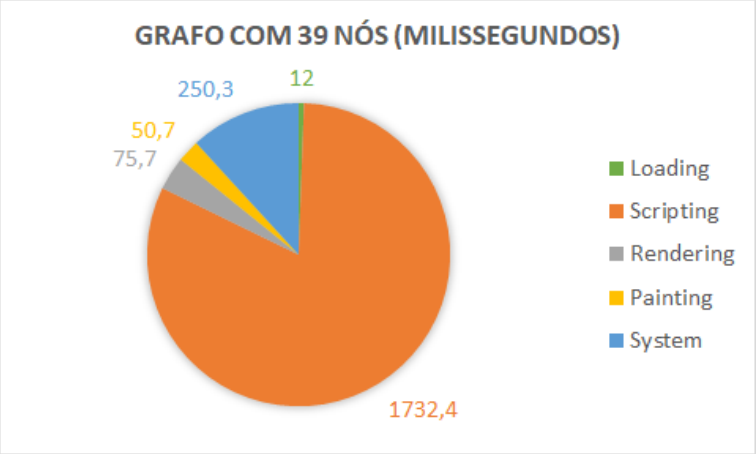

Figura 35 - Tempo de carregamento de um grafo com 39 nós

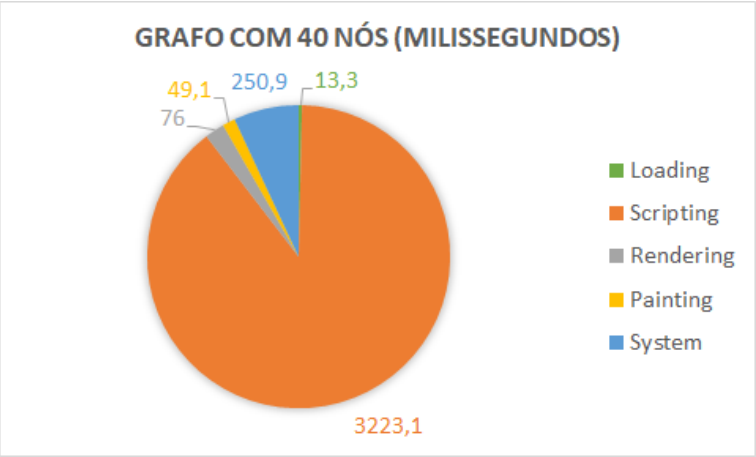

Figura 36 - Tempo de carregamento de um grafo com 40 nós

Através da análise das figuras 32 a 36 é possível observar que o tempo do fator *scripting* aumenta de acordo com a quantidade de nós do grafo. Este aumento é expectável e deve-se à função de cálculo das posições dos nós do grafo: quanto mais nós existem no grafo, mais tempo demora o cálculo das suas posições.

É também notável uma discrepância entre o tempo de *scripting* entre um grafo de 39 e 40 nós. A diferença entre o tempo de um grafo com 30 e 39 nós é bastante inferior comparativamente à de um grafo com 39 e 40 nós, o que indica que provavelmente existe um erro na implementação da função que calcula a posição dos nós do grafo. Este problema tem um impacto negativo no desempenho da aplicação, uma vez que o diminui significativamente.

Os resultados obtidos mostram que num cenário em que o grafo possui 40 nós, o tempo de execução do código médio é de aproximadamente 3.2 segundos. Este tempo é aceitável do ponto de vista do utilizador.

Capítulo 7
Conclusão

Durante o estágio tive a oportunidade de adquirir conhecimento sobre a arquitetura monolítica, arquitetura de microsserviços e os seus métodos de monitorização (controlo de recursos, *logs*, métricas e *tracing*). Aprendi a linguagem de programação *Javascript* e a utilizar ferramentas como *React*, *PixiJS* e *Recharts*.

O planeamento da aplicação desenvolvida incorporou casos de uso, requisitos funcionais e não funcionais.

Relativamente à fase de implementação do projeto, os requisitos funcionais mais importantes (de acordo com a priorização dada) foram cumpridos. Nem todos os requisitos foram cumpridos, o que significa que algumas funcionalidades não foram implementadas. Durante esta fase surgiram alguns *bugs* inesperados que necessitaram de tempo para a sua resolução, acabando por não ser executado o planeamento inicialmente proposto. Das quatro vistas planeadas, foram implementadas a vista de microsserviços, vista de instâncias e vista de *traces*, ficando por implementar a vista de *workflows*. Por fim, foram realizados testes aos requisitos funcionais e não funcionais.

Dada a aplicação final, existem alguns aspetos que podem ser melhorados como a adição de um filtro na vista de *traces*, implementação da vista de *workflows*, a possibilidade do utilizador observar o estado do sistema numa determinada data e correção da função de cálculo da posição de nós do grafo.

Para finalizar, considero que aprendi muito com este estágio, e apesar das dificuldades encontradas fui capaz de desenvolver uma ferramenta que inclui grande parte do que foi solicitado, por isso foi um sucesso.

Referências

[1] I. Wigmore, "What is monolithic architecture? - Definition from WhatIs.com," [Online]. Available: https://whatis.techtarget.com/definition/monolithic-architecture. [Acedido em 9 Janeiro 2022].

[2] R. Tamassia, em *Handbook of Graph Drawing and Visualization*, CRC Press, 2013, pp. 385-386.

[3] M. Richards, "Microservices Architecture Pattern - Software Architecture Patterns," [Online]. Available: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch04.html. [Acedido em 5 Janeiro 2022].

[4] "8 Steps for Migrating Existing Applications to Microservices," [Online]. Available: https://insights.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/. [Acedido em 12 junho 2021].

[5] "Microservice Architecture pattern," [Online]. Available: https://microservices.io/patterns/microservices.html. [Acedido em 5 junho 2021].

[6] S. Newman, em *Building Microservices: Designing Fine-Grained Systems*, Sebastopol, O'Reilly Media, Inc, 2015, pp. 6-7.

[7] S. Newman, em *Building Microservices: Designing Fine-Grained Systems*, Sebastopol, O'Reilly Media, Inc, 2015, pp. 5-6.

[8] S. Nayak, "How to Use Metrics to Monitor Your Microservices," [Online]. Available: https://www.freecodecamp.org/news/microservice-observability-metrics/. [Acedido em 13 junho 2021].

[9] "What is Distributed Tracing," [Online]. Available: https://opentracing.io/docs/overview/what-is-tracing/. [Acedido em 16 junho 2021].

[10] "Spans in OpenTelemetry," [Online]. Available: https://opentelemetry.lightstep.com/spans. [Acedido em 10 Janeiro 2022].

[11] "Jaeger," [Online]. Available: https://www.jaegertracing.io/. [Acedido em 11 Janeiro 2022].

[12] "SigNoz," [Online]. Available: https://signoz.io/. [Acedido em 11 Janeiro 2022].

[13] "OpenZipkin," [Online]. Available: https://zipkin.io/. [Acedido em 11 Janeiro 2022].

[14] "Dynatrace," [Online]. Available: https://www.dynatrace.com/. [Acedido em 11 Janeiro 2022].

[15] "New Relic," [Online]. Available: https://newrelic.com/. [Acedido em 11 Janeiro 2022].

[16] "Honeycomb," [Online]. Available: https://www.honeycomb.io/. [Acedido em 11 Janeiro 2022].

[17] "Lightstep," [Online]. Available: https://lightstep.com/. [Acedido em 11 Janeiro 2022].

[18] "Instana," [Online]. Available: https://www.instana.com/. [Acedido em 11 Janeiro 2022].

[19] "Datadog," [Online]. Available: https://www.datadoghq.com/. [Acedido em 11 Janeiro 2022].

[20] "Splunk," [Online]. Available: https://www.splunk.com/. [Acedido em 11 Janeiro 2022].

[21] S. Silva, J. Correia, A. Bento, F. Araujo e R. Barbosa, "uViz: Visualization of Microservices".

[22] S. Silva, J. Correia, A. Bento, F. Araujo e R. Barbosa, "uViz: Visualization of Microservices," p. 5.

[23] S. Silva, J. Correia, A. Bento, F. Araujo e R. Barbosa, "uViz: Visualization of Microservices," p. 6.

[24] S. Silva, J. Correia, A. Bento, F. Araujo e R. Barbosa, "uViz: Visualization of Microservices," pp. 6-7.

[25] S. Silva, J. Correia, A. Bento, F. Araujo e R. Barbosa, "uViz: Visualization of Microservices," pp. 6-8.

[26] "React – A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org/. [Acedido em 17 junho 2021].

[27] "Getting Started - WebGL Public Wiki," [Online]. Available: https://www.khronos.org/webgl/wiki/Getting_Started. [Acedido em 17 junho 2021].

[28] "home | p5.js," [Online]. Available: https://p5js.org/. [Acedido em 17 junho 2021].

[29] "GitHub - pixijs/pixijs: The HTML5 Creation Engine: Create beautiful digital content with the fastest, most flexible 2D WebGL renderer.," [Online]. Available: https://github.com/pixijs/pixijs. [Acedido em 17 junho 2020].

[30] "D3.js - Data-Driven Documents," [Online]. Available: https://d3js.org/. [Acedido em 17 junho 2021].

[31] "Chart.js | Open source HTML5 Charts for your website," [Online]. Available: https://www.chartjs.org/. [Acedido em 17 junho 2021].

[32] "Recharts," [Online]. Available: https://recharts.org/en-US/. [Acedido em 17 junho 2021].

[33] S. Newman, em *Building Microservices: Designing Fine-Grained Systems*, Sebastopol, O'Reilly Media, Inc, 2015, pp. 2-3.

[34] "Introduction to Monolithic Architecture and MicroServices Architecture | by Siraj ul Haq | KoderLabs | Medium," [Online]. Available:

https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63. [Acedido em 7 junho 2021].

[35] "Spans," [Online]. Available: https://opentracing.io/docs/overview/spans/. [Acedido em 16 junho 2021].