Politécnico
Castelo Branco
Escola Superior
de Tecnologia

allbesmart

Desenvolvimento de uma *dashboard* para monitorização e configuração remota de uma estação base 5G

Liliana Sofia Ribeiro Monforte

Orientador

Professor Adjunto Doutor Osvaldo Arede dos Santos

Dissertação apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática – Área de Especialização em Desenvolvimento de Software e Sistemas Interativos, realizada sob a orientação científica do Professor Adjunto Doutor Osvaldo Arede dos Santos, do Instituto Politécnico de Castelo Branco.

Julho 2025

Composição do júri

Presidente do júri

Doutor, José Carlos Meireles Monteiro Metrôlho

Vogais

Doutor, Rui Pedro Monteiro Amaro Duarte

Professor Adjunto, Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Viseu

Doutor, Fernando Reinaldo da Silva Garcia Ribeiro

Professor Adjunto, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Doutor, Osvaldo Arede dos Santos

Professor Adjunto, Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco

Dedicatória

À minha mãe.

Agradecimentos

A realização e conclusão deste trabalho não teriam sido possíveis sem o apoio, o auxílio e os contributos de diversas pessoas, a quem deixo aqui o meu profundo agradecimento.

Ao Professor Doutor Paulo Marques e a toda a equipa da empresa Allbesmart, pela oportunidade e pela constante disponibilidade demonstrada ao longo do projeto.

Ao Professor Doutor Osvaldo Santos, pela orientação e pelo apoio contínuo em todo o processo de investigação e elaboração desta dissertação.

Ao meu pai e restantes familiares, aos meus amigos Ana, Mónica, João e Jaime, ao Skander e à Missy, pelo encorajamento e presença durante este percurso.

Resumo

A contínua evolução das tecnologias móveis é conseguida através do desenvolvimento de produtos que permitam o estudo e investigação das mesmas. É neste contexto que se enquadra a OAIBOX™, uma estação base 5G produzida e distribuída pela empresa Allbesmart, cuja plataforma *web* de suporte à sua monitorização e configuração remota representa o objetivo principal da presente dissertação. O projeto requer uma interface gráfica de utilizador intuitiva e responsiva, que disponibilize a visualização de gráficos e métricas em tempo real da estação em funcionamento e também uma comunicação com a mesma para configuração de parâmetros de serviço de rede.

Após uma apresentação de conceitos chave para a compreensão do desenvolvimento de *dashboards* e da tecnologia 5G, foi efetuada uma análise de trabalhos científicos que utilizam plataformas *web* para funcionalidades de monitorização e configuração remota de sistemas e/ou processos. Sendo o foco deste trabalho o desenvolvimento *frontend*, foram estudadas as principais tecnologias *frontend* utilizadas nos últimos anos para este efeito, nomeadamente Angular, React e Vue.js. Acompanhando uma análise teórica, foi ainda concebido um cenário hipotético implementado com as três tecnologias que, suportado por um projeto *backend* desenvolvido em Spring Boot, permitiu uma comparação de métricas estabelecidas. Estas métricas consistiram na análise do número de linhas de código, no volume de dados e tempo decorrido durante o carregamento inicial da página e tempos de comunicação de dados entre o *backend* e *frontend*, incluindo envio de valores contínuos e alteração de parâmetros. A análise aos resultados revelou um melhor desempenho do projeto Vue.js na maioria dos parâmetros, no entanto, a escolha da tecnologia de desenvolvimento *frontend* para o projeto da *dashboard* recaiu sobre Angular, por se tratar de uma *framework* completa e robusta ideal a projetos complexos e de longa longevidade.

Por fim, a modelação do sistema permitiu a identificação dos requisitos e funcionalidades essenciais a incluir na *dashboard* de monitorização e configuração remota da estação base 5G. A apresentação e descrição das funcionalidades e interfaces desenvolvidas demonstrou a utilização desta plataforma no contexto em que se integra, incluindo a comunicação com os restantes componentes do sistema.

Palavras-chave

Dashboard; *frontend*; monitorização; configuração; tempo real.

Abstract

The continuous evolution of mobile communications technology is driven by the development of products that support their study and research. In this context, the OAIBOX™, a 5G base station created and distributed by Allbesmart, plays a central role. The web platform that enables its remote management and monitoring represents the main goal of this work. The project requires an intuitive and responsive graphic user interface (GUI) that provides real-time visualization of charts and metrics from the active station, as well as communication capabilities to configure network service parameters.

Following an introduction to key concepts related to dashboard development and 5G technology, a review of scientific studies that use web platforms for the remote monitoring and configuration of systems and/or processes was conducted. With a focus on frontend development, the main frontend technologies used in recent years were examined, namely Angular, React and Vue.js. Complementing the theoretical analysis, a hypothetical scenario was implemented using the three technologies and supported by a backend project developed with Spring Boot. This setup allowed a comparison based on established metrics: lines of code, data size and loading time during the initial page load, and communication times between backend and frontend, including continuous data transmission and parameter changes. Although the Vue.js project demonstrated better performance in most metrics, Angular was ultimately chosen for the dashboard development, as it is a robust and comprehensive framework, well-suited for complex and long-term projects.

Finally, system modelling was used to identify the essential requirements and features to be included in the dashboard for remote monitoring and management of the 5G base station. The presentation and description of the implemented features and interfaces showed how this platform operates within the broader system, including its communication with other system components.

Keywords

Dashboard; frontend; monitoring; configuration; real-time.

Índice geral

Índice de figuras

Lista de tabelas

1. Introdução

O avanço e a investigação de tecnologias móveis trouxe novos desafios e oportunidades ao desenvolvimento de ferramentas que permitem a monitorização e controlo eficiente destes sistemas de telecomunicações. A gestão remota de redes móveis, recorrendo a interfaces gráficas, é alcançada com a implementação de, por exemplo, *dashboards* cujas customizações são definidas e ajustadas às particularidades de cada projeto.

Neste contexto, torna-se fundamental um estudo do desenvolvimento de *dashboards*, tanto a nível de trabalhos científicos que abordam este tema, como da análise de tecnologias *frontend* disponíveis no mercado. Este último ponto é particularmente importante para a escolha da *framework* a utilizar na implementação.

Outro ponto essencial é a modelação do sistema, nomeadamente ao nível do levantamento de requisitos e da definição de casos de uso, garantido que o produto desenvolvido corresponda às expetativas dos utilizadores. Não descurando, no entanto, as restantes etapas da modelação que auxiliam a implementação de um sistema robusto.

Considerando estes elementos, a implementação de uma *dashboard* confiável exige uma abordagem cuidadosa e adaptada, alinhada com as necessidades específicas do projeto, com especial atenção à integração das ferramentas selecionadas e aos demais componentes do sistema ao qual pertence. Neste contexto, atendendo ao carácter dinâmico e em permanente evolução do setor, é crucial reconhecer que se trata de um produto alvo de atualizações contínuas.

1.1. Enquadramento

A empresa Allbesmart [1], sediada em Castelo Branco e com uma forte ligação ao Instituto Politécnico de Castelo Branco, é especializada no desenvolvimento de soluções baseadas em implementações 3rd Generation Partnership Project (3GPP), contribuindo ativamente para a iniciativa OpenAirInterface™. O seu principal produto é a OAIBOX™, uma estação base desenvolvida para redes 5G, mas que está em atualização para a investigação no âmbito das redes 6G. Nas Figura 1 e Figura 2 encontram-se reproduções das soluções OAIBOX™ 40 e OAIBOX™ MAX, respetivamente.

Figura 1 – OAIBOX™ 40 [2].

Figura 2 – OAIBOX™ MAX [2].

Tratando-se, sobretudo, de um produto direcionado para a investigação e a indústria de desenvolvimento de redes, é essencial um suporte de interface gráfica que facilite a monitorização e configuração da estação base 5G. É neste sentido que se insere a presente dissertação: o estudo e implementação de uma *dashboard* para monitorização e configuração remota de uma estação base 5G.

1.2. Objetivos

O objetivo deste projeto é a implementação de uma *dashboard* para monitorização e configuração remota de uma estação base 5G que será integrada com *backend* desenvolvido pela empresa Allbesmart. Pretende-se uma plataforma *web* com interface gráfica de utilizador (GUI) intuitiva e responsiva, que permita a visualização em tempo real de gráficos e métricas de qualidade de serviço 5G e também a configuração remota de parâmetros como a frequência rádio de transmissão e largura de banda.

Para tal, o trabalho inicia-se com a identificação dos requisitos técnicos e a definição da arquitetura do sistema. Ainda numa fase inicial, algumas das *frameworks* mais em voga para o desenvolvimento *frontend* são investigadas no âmbito deste projeto. O estudo comparativo entre Angular, React e Vue.js, que

também compreende testes de desempenho, permite a escolha fundamentada da *framework* mais adequada à implementação da *dashboard*. Numa fase final, a implementação do *frontend* da *dashboard* é acompanhada da respetiva integração com o *backend* desenvolvido pela Allbesmart.

1.3. Planeamento

O projeto foi planeado para decorrer ao longo de 12 meses, começando por uma identificação dos requisitos e definição da arquitetura do sistema, acompanhado pelo estudo de *frameworks* que melhor se adequa ao trabalho a desenvolver. Procedeu-se, depois, à implementação da *dashboard* e respetiva integração com *backend*. A escrita da presente dissertação foi efetuada durante todo o processo. Para a elaboração do cronograma utilizou-se o *software* Online Gantt [3] e é representado na Figura 3, tendo o projeto sido dividido nas seguintes tarefas:

- Tarefa 1: Identificação dos requisitos do sistema;
- Tarefa 2: Definição da arquitetura de *software*;
- Tarefa 3: Estudo comparativo das *frameworks*;
- Tarefa 4: Desenvolvimento da *Dashboard*;
- Tarefa 5: Integração com o *backend* 5G desenvolvido pela Allbesmart;
- Tarefa 6: Escrita da dissertação.

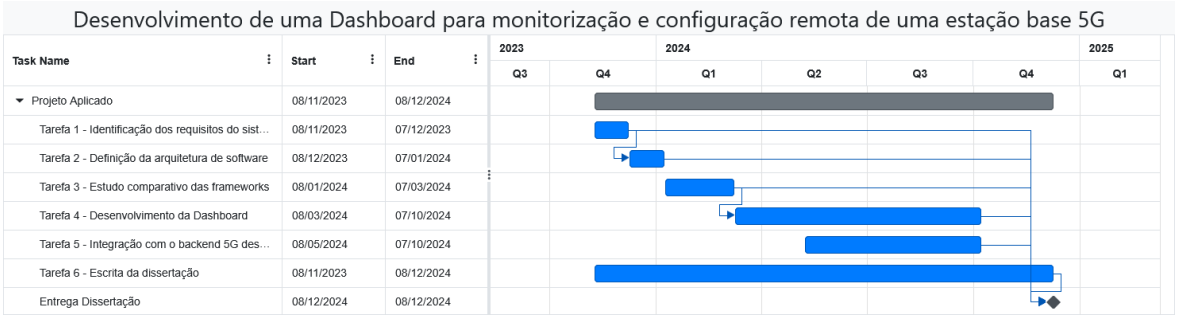


Figura 3 – Cronograma previsto.

1.4. Organização da dissertação

A dissertação está organizada em seis capítulos: *Introdução*, *Estado da arte*, *Tecnologias de desenvolvimento* frontend, *Modelação do sistema*, *Desenvolvimento da* dashboard e *Conclusão*. Cada capítulo é antecedido com uma breve introdução e subdividido em diversos subcapítulos.

No primeiro capítulo, é efetuada uma introdução ao projeto, com foco no seu enquadramento, objetivos e planeamento.

No segundo capítulo, são apresentados vários conceitos considerados relevantes, nomeadamente elementos teóricos relativos ao desenvolvimento de *dashboards* e definições de redes e tecnologia 5G que, apesar de não serem o cerne da dissertação, são importantes para o entendimento do projeto final. Ainda neste capítulo, é realizada uma análise de trabalhos científicos na área do desenvolvimento de *dashboards* seguindo a diretriz PRISMA [4].

No terceiro capítulo, três tecnologias *frontend* são descritas e é elaborado um pequeno cenário hipotético implementado com estas tecnologias. Posteriormente, são efetuados alguns testes considerados relevantes no âmbito geral da dissertação e apresentados os resultados e conclusões.

No quarto capítulo, é realizada a modelação do projeto da *dashboard* para monitorização e configuração remota de uma estação base 5G seguindo vários elementos da metodologia ICONIX, justificando a ausência de outros.

No quinto capítulo, é descrito o processo de implementação da *dashboard*, dando um enquadramento da arquitetura geral do sistema em que se insere e especificando as interfaces e funcionalidades desenvolvidas.

O sexto capítulo é dedicado às conclusões da dissertação, sintetizando os principais resultados e discutindo as limitações encontradas.

2. Estado da arte

Neste capítulo descrevem-se conceitos chave ao tema do projeto e apresentam-se exemplos pertinentes.

2.1. Conceitos e aspetos relevantes no desenvolvimento de *dashboards*

A implementação de *dashboards* tem sido amplamente utilizada com diversos propósitos e em inúmeras áreas, desde a educação, a saúde e a logística empresarial [5]. O conceito de *dashboard* pode ser definido como a visualização de informação importante, consolidada e organizada num único ecrã, para alcançar determinados objetivos, nomeadamente na tomada de decisões. Para tal, a informação deverá estar organizada de modo a facilitar a sua rápida e acessível monitorização e perceção, sendo essencial a visualização de dados em tempo real sobre o estado atual, bem como o registo histórico relevante [5], [6].

O *design* da *dashboard*, que é fundamental para o seu sucesso e eficiência, deve considerar o propósito da mesma, dado que a informação a apresentar e respetivo detalhe depende do nível de gestão que se procura. Assim, tendo por base o nível de gestão e o papel dos utilizadores na organização, podem-se considerar três níveis principais de *dashboards*:

- ***Dashboard* estratégica:** *dashboard* cujo objetivo é a monitorização, comunicação e avaliação do desempenho da implementação de estratégias decretadas por executivos, lidando com informação mais complexa, resultante da manipulação de dados recolhidos a longo termo;
- ***Dashboard* tática:** *dashboard* para monitorização e gestão do desempenho de departamentos ou projetos, representado um alvo mais específico que a estratégica e, por isso, apresenta a informação a um nível mais detalhado;
- ***Dashboard* operacional:** *dashboard* utilizada principalmente por trabalhadores na linha da frente para gestão e controlo a nível operacional dos processos da organização; alguns autores diferenciam as *dashboards* operacionais nos subtipos *detetar-e-responder* (monitorização e otimização de processos) e *incentivar-e-motivar* (métricas de desempenho); das três é a que manipula os dados na sua forma mais simples, detalhada e atualizada [5], [6].

Para além do nível de gestão, a identificação do propósito da *dashboard* é um elemento essencial aquando do *design*, dividindo-se em:

- **Consistência:** indicadores de processos, procedimentos e medidas da organização;
- **Monitorização:** monitorização e avaliação de desempenho para que um evento seja rapidamente detetado e a organização possa reagir adequadamente. Alguns autores referem também a **análise** individual e da organização similar ao de monitorização;
- **Planeamento:** planeamento estratégico das próximas ações baseado nos dados atuais;
- **Comunicação:** partilha da informação atual de desempenho com os *stakeholders* [5], [6].

Após a determinação do propósito e dos objetivos da utilização da *dashboard*, é feito o levantamento das funcionalidades do sistema que podem ser funcionais ou visuais. As primeiras referem-se às operações que a *dashboard* pode efetivamente realizar; já as segundas, incluem a apresentação dos dados que influencia diretamente a eficiência e a eficácia da transmissão da informação e a dificuldade ou a facilidade com que esta é apreendida [5], [6].

As características funcionais correspondem à interação entre o utilizador e a informação da *dashboard*, incluindo a interatividade através das ações com o cursor, filtros, ordenação, entre outros. Estas funcionalidades tornam a *dashboard* interativa, atribuindo ao utilizador um maior papel na análise dos dados [6].

As características visuais são especialmente importantes na apreensão da informação. O uso apropriado de efeitos visuais é um aspeto essencial no desenvolvimento da interface com o utilizador. Por exemplo, o uso excessivo de objetos 3D e cores como o amarelo e vermelho (que são geralmente utilizados para destacar avisos e problemas imediatos) dificultam o processo de compreensão. Já a utilização de grelhas nos gráficos e a redução de componentes não representativos de dados são melhoramentos para a assimilação da informação [6].

Cada nível de *dashboard* tem mais de um propósito e os diferentes níveis podem ter o mesmo propósito entre eles; no entanto, as funcionalidades com que atingem esses propósitos são diferentes, mesmo não existindo uma fronteira clara entre eles. Tomando como exemplo o propósito de monitorização que pode estar presente em todos os níveis, mas efetuado de modo e perspetiva diferente em cada um: a um nível estratégico para a monitorização geral da organização, a um nível tático para a monitorização departamental e a um nível operacional para a monitorização individual. Também algumas das funcionalidades são transversais aos diferentes níveis de *dashboard*: como a utilização de um único ecrã (visual)

para apresentação gráfica da informação (funcional). De facto, o recurso a gráficos é uma das características presentes nos diversos níveis e tipos, e um dos elementos distintivos na visualização de dados [5], [6].

Existem inúmeros tipos de gráficos e a sua escolha depende dos dados a apresentar e do desígnio da sua representação. Alguns dos componentes gráficos mais empregues são o gráfico de barras, o gráfico de linhas, o gráfico de setores, o gráfico *gauge* e o gráfico de dispersão [7], [8], [9], [10].

O gráfico de barras (Figura 4) é utilizado na representação de dados numéricos em barras horizontais; gráficos com barras verticais são designados de gráficos de colunas. O comprimento das barras reflete o valor e podem ser utilizadas diferentes cores para facilitar a comparação de categorias diferentes [7], [8], [9], [10].

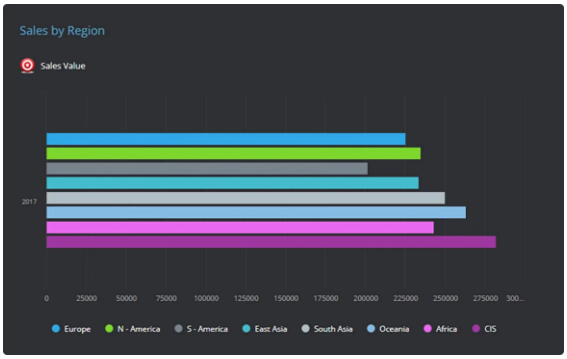


Figura 4 – Exemplo de gráfico de barras [8].

O gráfico de linhas (Figura 5) mostra a correlação entre as variáveis representadas nos dois eixos, sendo muitas vezes empregue para a variação de um dado num determinado período ou em diferentes categorias contínuas. Podem apresentar pontos que são interligados ou apenas a(s) linha(s) e, no caso de apresentação de múltiplas linhas, a utilização de diferentes cores ajuda na análise do gráfico [7], [8], [9], [10].

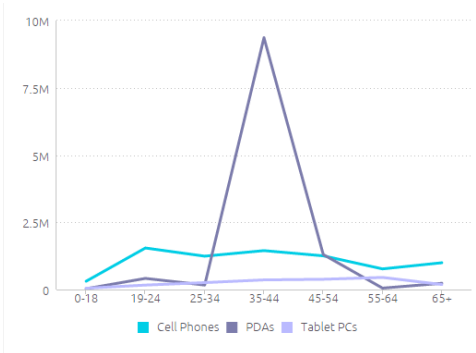


Figura 5 – Exemplo de gráfico de linhas [10].

O gráfico de setores (Figura 6) é um diagrama de setores dividido em diferentes fatias que representam diferentes categorias, cujo tamanho reflete a percentagem dessa categoria no todo. Este gráfico é geralmente utilizado para visualizar e comparar dados percentuais e/ou numéricos [7], [8], [9], [10].

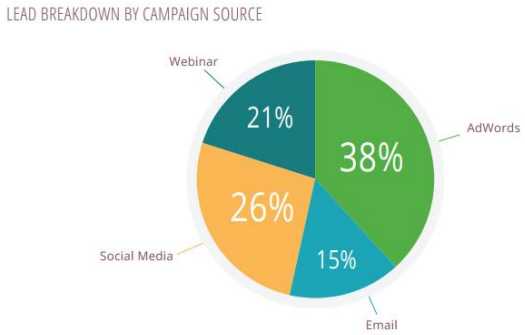


Figura 6 – Exemplo de gráfico de setores [10].

O gráfico *gauge* (Figura 7 e Figura 8) mostra o dado atual num mostrador numa escala por forma a uma rápida avaliação, sendo muito utilizado para monitorização de processos em *dashboards* e como indicador de indicador-chave de desempenho (*Key Performance Indicator* – KPI). A escala é representada de acordo com o que se pretende visualizar e a natureza do dado, podendo, por exemplo, utilizar intervalos de valores ou sinalizar um valor alvo/objetivo para uma análise mais rápida e intuitiva [7], [8], [9], [10].

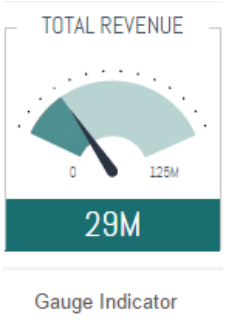


Figura 7 – Exemplo de gráfico *gauge* como indicador de valor [10].

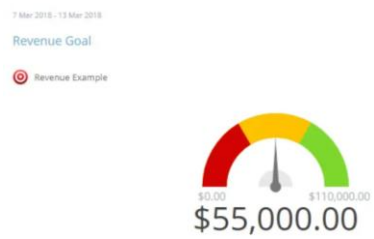


Figura 8 – Exemplo de gráfico *gauge* com intervalos [8].

O gráfico de dispersão (Figura 9) demonstra a correlação entre variáveis em dois eixos utilizando apenas pontos, o que facilita a identificação de padrões e/ou tendências [7], [8], [9], [10].

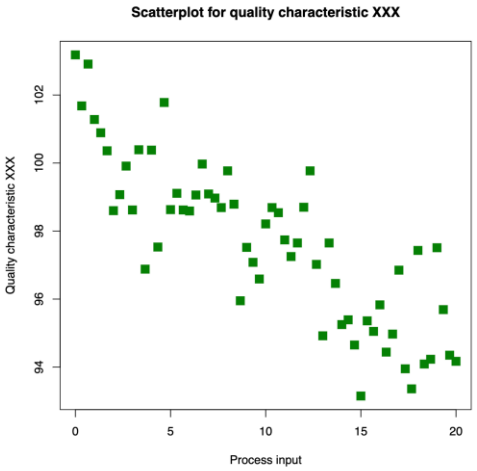


Figura 9 – Exemplo de gráfico de dispersão [7].

Dada a sua importância na visualização de informação, existem bibliotecas de gráficos para auxílio no desenvolvimento de *dashboards*, como Grafana e Apache ECharts [11], [12].

Grafana é uma plataforma *open source* de visualização de dados e respetiva análise e monitorização. Pode ser utilizada diretamente por analistas de dados. através da implementação de *dashboards* com grande diversidade de gráficos e configuração de limites e alertas, ou integrada como uma biblioteca de código, através do recurso a Interfaces de Programação a Aplicações (*Application Programming Interface* – API) e *plugins*. Este *software* pode ser integrado com diversas *frameworks* de *backend* e *frontend*, como Django, Spring Boot, React, Vue.js e Angular [11].

Apache ECharts é uma biblioteca JavaScript *open source* para criação de gráficos interativos e visualização de dados com personalização flexível, sendo apelativa no desenvolvimento de *dashboards* e outros sistemas de visualizações de dados. Disponibiliza uma vasta diversidade de gráficos, com grande interatividade como zoom, dicas e destaque de dados. É direcionada principalmente para programadores de JavaScript, mas providencia APIs e respetiva documentação para integração com várias *frameworks* como React, Vue.js e Angular [12]. Alguns dos componentes e produtos oferecidos pela Grafana recorre a *plugins* de Apache ECharts [11].

2.2. Conceitos chave de monitorização e configuração de estações base 5G

Considerando o caso específico em que se insere o desenvolvimento da *dashboard* – monitorização e configuração de estação base 5G – este subcapítulo aborda alguns termos básicos relacionados com redes privadas e tecnologia 5G que se consideram essenciais para contextualizar elementos que serão implementados. De ressalvar que este tema não será aprofundado e detalhado uma vez que não se trata do foco principal do trabalho.

2.2.1. Tecnologia 5G

5G refere-se à quinta geração de tecnologia móvel desenhada para aumentar a velocidade, reduzir a latência, aumentar a capacidade da rede e a disponibilidade, oferecendo a mais utilizadores uma experiência uniforme e confiável. As especificações da tecnologia 5G foram definidas pela organização 3GPP [13] e, teoricamente, consegue alcançar velocidades de 20 Gbps, contrastando com o máximo de 1Gbps oferecido pelo 4G. As redes 5G são virtualizadas, orientadas ao *software*, retirando grande proveitos das tecnologias *cloud* e melhoram as experiências de *machine learning* e automação em que é necessária uma resposta rápida [13], [14], [15].

A tecnologia 5G introduz alterações na arquitetura de rede que atende a deficiências nas tecnologias sem fios anteriores, nomeadamente a utilização de antenas *Multiple Input, Multiple Output* (MIMO) para uma maior transmissão de dados nos dois sentidos em simultâneo e a possibilidade de utilizar tecnologias sem fios licenciadas e não licenciadas aumentado a largura de banda disponibilizada. As funcionalidades das redes 5G são geridas através de *software* e permite a criação de *slices* que consistem em sub-redes definidas no *software* cujas funcionalidades são atribuídas com base nos utilizadores e dispositivos [14].

A garantia de baixa latência, alta largura de banda e conexões estáveis e seguras tornam as redes 5G ideais para setores que requerem ligações fiáveis. Esta tecnologia é de grande interesse para as áreas como a saúde (cirurgias remotas), a indústria (automação e robots) e o setor automóvel (condução autónoma, transmissão de informação de tráfego, acidentes e outros incidentes entre veículos e/ou estruturas rodoviárias) [14], [15], [16].

2.2.2. Arquitetura de redes 5G

Genericamente, uma rede 5G (Figura 10) consiste numa rede sem fios e numa rede central, sendo os seguintes elementos os principais componentes numa arquitetura de rede 5G:

- Rede central (*Core Network* – 5GC): responsável por tarefas de encaminhamento, autenticação e segurança da rede, que assenta numa arquitetura baseada em serviços (*Service-Based Architecture* – SBA). Neste tipo de arquitetura, os elementos são definidos em termos de funções de rede (*Network Function* – NF) e cada NF oferece os seus serviços a outros NF e/ou consumidores autorizados. Alguns destes NFs são:
    - *Access and Mobility Management Function* (AMF): gere a mobilidade dos equipamentos, nomeadamente o acesso inicial à rede, autenticação e sessão entre diferentes pontos de acesso;
    - *User Plane Function* (UPF): lida com o tráfego de dados do utilizador;
    - *Session Management Function* (SMF): controla a gestão de sessões e fornecimento do serviço 5G;
    - *Policy Control Function* (PCF): gere o controlo de políticas 5G;
    - *NF Repository Function* (NRF): repositório de informação das funções da rede;
    - *Authentication Server Function* (AUSF): funções de autenticação de utilizadores e segurança associada;
    - *Unified Data Management* (UDM): gestão de dados de utilizadores e perfis na rede 5G;
    - *Network Exposure Function* (NEF): interface para aplicações e serviços de terceiros;
- *Radio Access Network* (RAN): responsável pela comunicação sem fios entre a rede central e os dispositivos. Inclui estações base, antenas e outros equipamentos, sendo os principais:
    - Estação base 5G (gNB): responsável pela transmissão e receção dos sinais sem fios com os equipamentos de utilizador;
    - *Next-Generation RAN* (NG-RAN): arquitetura RAN que inclui gNB e outros elementos de suporte e permite a coordenação e gestão de múltiplos gNB para garantir uma conetividade contínua e eficiente;

- *Air interface*: espaço que as ondas de rádio atravessam entre a gNB e os dispositivos;
- Equipamento de utilizador (*User Equipment* – UE): dispositivo, como telemóvel ou computador, equipado com um modem 5G e capaz de comunicar através de 5G [13], [17], [18], [19].

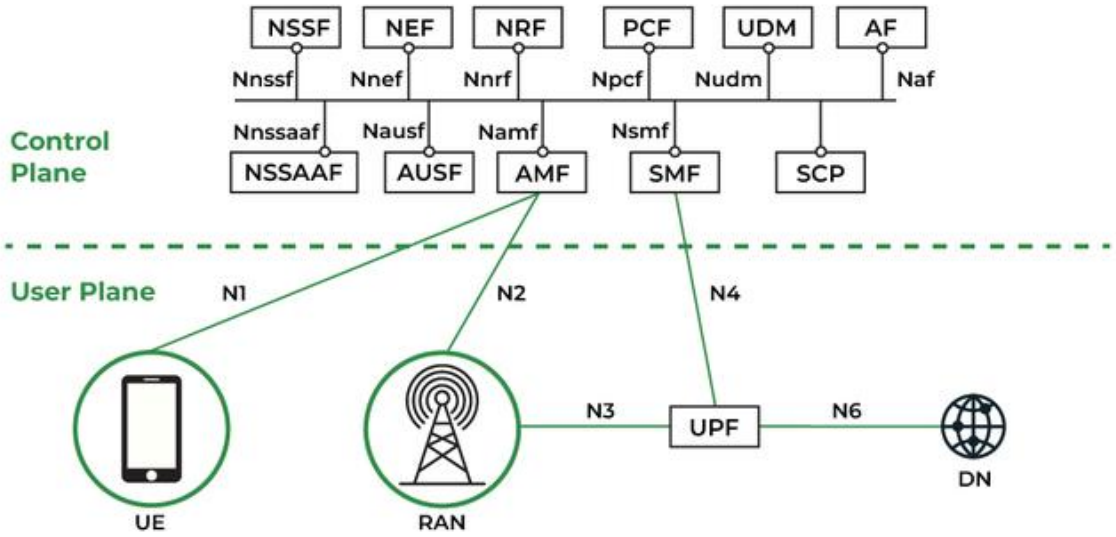


Figura 10 – Arquitetura 5G [19].

Esta arquitetura genérica representa um sistema *standalone* (SA), isto é, uma rede cujos componentes utilizam exclusivamente tecnologia 5G (ver Figura 11). Todavia, numa altura de transição, existiram as chamados sistemas *non-standalone* (NSA) em que parte da arquitetura recorre a tecnologia LTE, nomeadamente a rede central, sendo utilizado ondas rádio 5G entre o UE e a gNB. Nestes sistemas, para além dos componentes mencionados acima, são necessários equipamentos responsáveis pela compatibilidade entre a tecnologia 5G e LTE da rede [13], [17], [20].

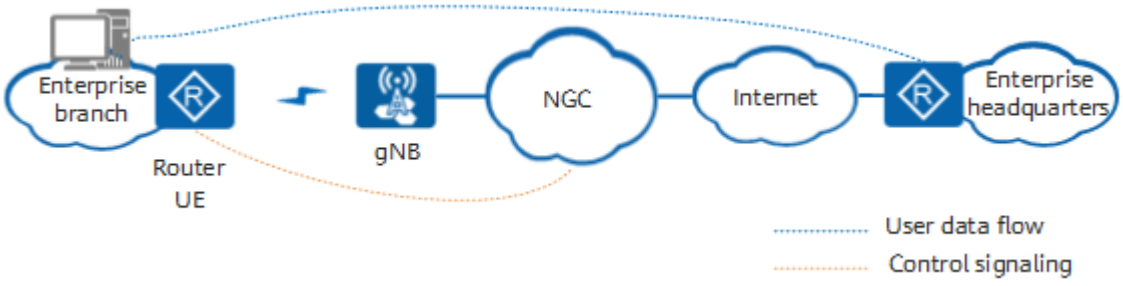


Figura 11 – Arquitetura SA suportada por *routers* [17].

A aposta em sistemas Open RAN permite a utilização de *open standards* que promove a interoperabilidade entre uma vasta diversidade de *hardware* e *software* na implementação dos sistemas 5G. Esta possibilidade de utilização de diversas interfaces e equipamentos permite o aparecimento de novos casos de uso, aplicações e serviços [21].

A nível de *software*, a OpenAirInterface™ é uma iniciativa *open source* que segue as normas de 3GPP para implementar uma solução de core, gNB e UE [22].

2.2.3. Estação base 5G – gNB

A estação base 5G (*gNodeB* – gNB) é um equipamento numa rede 5G que atua como ponto entre dispositivos móveis e a rede central, fornecendo conetividade aos dispositivos numa determinada área. Os seus principais componentes são:

- *Active Antenna Unit* (AAU): integra a RRU e o conjunto de antenas responsável pela transmissão e receção dos sinais de rádio nas frequências 5G definidas. As antenas estão capacitadas com a tecnologia MIMO para suportar a transferência de múltiplos fluxos de dados;
- *Radio Remote Unit* (RRU): facilita a comunicação *wireless*, sendo responsável pela conversão entre sinais digitais e ondas rádio;
- *Baseband Unit* (BBU): processa dos dados digitais transmitidos e recebidos, efetuando tarefas de modulação, encaminhamento e gerindo os protocolos de comunicação, integridade dos dados e segurança;
- Conexão *Backhaul*: conexão da gNB com a rede central [23].

A gNB pode ser dividida em *Central Unit* (CU) e uma ou mais *Distributed Unit(s)* (DU), interligadas por uma interface F1. A CU gere os protocolos de camada superior entre as várias DUs que funcionam como modem [13], [21], [22].

2.3. Trabalhos científicos na área

Para a análise de trabalhos científicos de interesse para a área em estudo, recorre-se ao método *The Preferred Reporting Items for Systematic reviews and Meta-Analyses* (PRISMA), uma diretriz para a formulação e conceção de revisões sistemáticas. PRISMA é sobretudo um guia de preparação destes trabalhos de modo transparente e replicável, melhorando a credibilidade e limitando o viés do

processo, em que os autores, seguindo a versão PRISMA 2020, disponibilizam uma *checklist* de 27 itens para avaliação do trabalho e um fluxograma de identificação dos artigos selecionados e rejeitados para estudo. PRISMA é essencialmente desenhado para reportar revisões relacionadas com a área da saúde, contudo, tem sido aplicado e adaptado para os mais diversos temas de estudo [24], [25], [26], [27], [28].

Contudo, aqui não são seguidos estritamente todos os pontos sugeridos pelo PRISMA. Com esta pesquisa e análise pretende-se sobretudo uma avaliação qualitativa e, apesar de também auxiliar estudos qualitativos, alguns dos itens deste método são específicos para dados quantitativos. Também alguns pontos são específicos de uma estrutura completa de artigo, que aqui se cinge a um subcapítulo. Ainda de ressalvar que o PRISMA representa, sobretudo, um modelo e guia para auxiliar a pesquisa e investigação, não sendo um método estrito e restritivo, mas um suporte para ser adaptado à área e à circunstância do estudo [24], [25], [26], [27], [28].

2.3.1. Definição da pesquisa

A pesquisa realizou-se nas bases de dados Scopus e IEEE Xplore; a primeira, pertencente ao grupo Elsevier, contém com uma vasta coleção de documentos nas mais diversas áreas de investigação e a segunda é dedicada, sobretudo, a documentos relacionados com tecnologia, ciências da computação, engenharias e outras áreas afins [29], [30].

Como termos de pesquisa utilizou-se *operational dashboard*, *real-time monitoring*, *configuration*, *management* e *web-based system*; combinados recorrendo a operadores booleanos nas entradas “operational dashboard AND real-time monitoring AND (configuration OR management)” e “web-based system AND real-time monitoring AND configuration”. Estas pesquisas foram efetuadas para o campo de dados “Article title, Abstract, Keywords” na base de dados Scopus e “All Metadata” na base de dados IEEE Xplore. Para refinar a pesquisa, estabeleceu-se os seguintes limites:

- Data de publicação desde 2015;
- Escritos em língua inglesa ou portuguesa;
- Conteúdos disponíveis para as subscrições do IPCB (IEEE Xplore).

2.3.2. Seleção dos trabalhos científicos

Como trabalhos científicos na área considera-se o desenvolvimento de *dashboard* ou aplicação *web* que monitorize um processo e/ou atividade e que

permite o controlo e/ou comando remoto através desta mesma *dashboard* ou aplicação *web*. Assim, a seleção dos artigos a analisar foi orientada tendo esta temática como critério principal.

A pesquisa das entradas referidas anteriormente, resultou num total de 179 artigos (119 na base de dados Scopus e 60 na base de dados IEEE Xplore), sendo 36 identificados como duplicados. Assim, 143 artigos passaram para triagem, onde foram analisados com base no título e resumo, considerando o critério geral definido anteriormente. Nesta fase, 105 artigos foram excluídos por não incidirem no desenvolvimento e/ou implementação da *dashboard* ou aplicação *web*, por incidirem apenas em sistemas de visualização e/ou monitorização, por constituírem estudos e/ou avaliações da utilização de *dashboards* ou aplicação *web* (com pouco ênfase no sistema em si), por não se enquadrarem no tema de *dashboard* ou aplicação *web* ou por se tratarem de registos fora do âmbito de artigo científico (nomeadamente, um livro e três coletâneas de artigos no seguimento de conferências). Com isto, 38 artigos foram identificados para obtenção, não sendo a mesma possível em 7 situações por motivos de falta de acesso ou indisponibilidade. Com a leitura do texto completo dos 31 artigos avaliados quanto à elegibilidade, foram excluídos 25 ao concluir que eram trabalhos que não tinham a *dashboard* ou aplicação *web* como alvo principal, trabalhos cuja *dashboard* ou aplicação *web* desenvolvida apenas fornecia funcionalidades de visualização e/ou monitorização, trabalhos fora do âmbito do desenvolvimento de *dashboard* ou aplicação *web* e um trabalho de estudo sobre o *design* de *dashboard* de visualização e/ou monitorização. No final do processo de seleção, foram identificados 6 trabalhos relevantes que são explorados na análise subsequente.

O processo de seleção dos trabalhos incluídos e excluídos está sintetizado no fluxograma da Figura 12, elaborado na ferramenta online disponibilizada através do *website* oficial do PRISMA2020 [31].

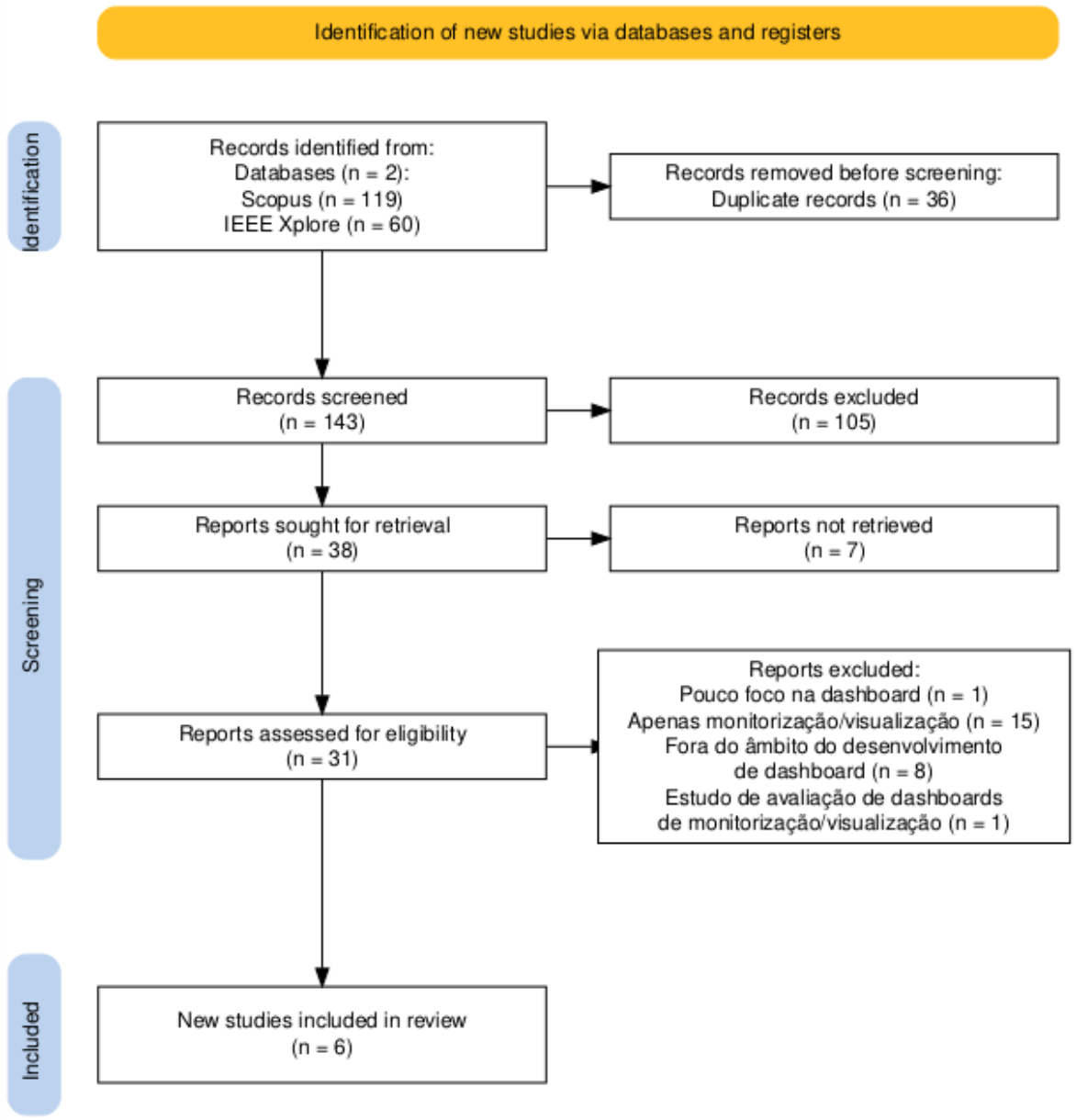


Figura 12 – Fluxograma do processo de seleção de artigos elaborado na ferramenta online disponibilizada em [31].

2.3.3. Análise dos trabalhos científicos

A utilização de *dashboards* está amplamente disseminada nas mais diversas áreas. Apesar de incidir sobretudo na visualização e monitorização de processos ou atividades, também existem estudos que exploram o seu uso como interface para configuração e controlo do sistema a monitorizar. De seguida, apresentam-se alguns casos relevantes que, apesar de não coincidirem diretamente com o contexto específico abordado na presente dissertação, ilustram a integração da monitorização e da configuração numa mesma *dashboard* ou aplicação *web*.

Trabalho científico 1

O sistema ciberfísico (*Cyber-Physical System* – CPS) desenvolvido em [32] tem como objetivo monitorizar e controlar um sistema descentralizado de tratamento e reutilização de águas residuais. Este sistema recorre a sondas de medição de controlo de qualidade de água e partes mecânicas e hidráulicas de abertura e fecho das secções das unidades de tratamento, assegurando a integração dos sistemas preexistentes.

Os dados medidos pelos sensores dos diferentes tanques de tratamento são recolhidos por um dispositivo de registo e controlo de sensores industrial e toda a panóplia de equipamentos de regulação de fluxo e volume de água em cada tanque são programados e operados manualmente através de um *Programmable Logic Controller* (PLC) industrial. Por sua vez, estes dispositivos estão ligados a um microcontrolador que intermedeia a comunicação entre os dispositivos de medição e atuação e as camadas superiores do sistema, destinadas à interação com os operadores [32].

No *middleware*, é incorporado um conjunto de normas Open Geospatial Consortium's Sensor Web Enablement (OGC SWE) que padronizam o acesso, controlo e partilha de dados de sensores via *web*, permitindo a uniformização, representação e transferência de informação entre a camada física e a plataforma *web*. Após o processamento dos dados, estes são enviados para a *dashboard* (ver Figura 13) onde são apresentados em tempo real numa interface gráfica que suporta a análise e a tomada de decisões pelos operadores, permitindo também um controlo dos elementos físicos [32].

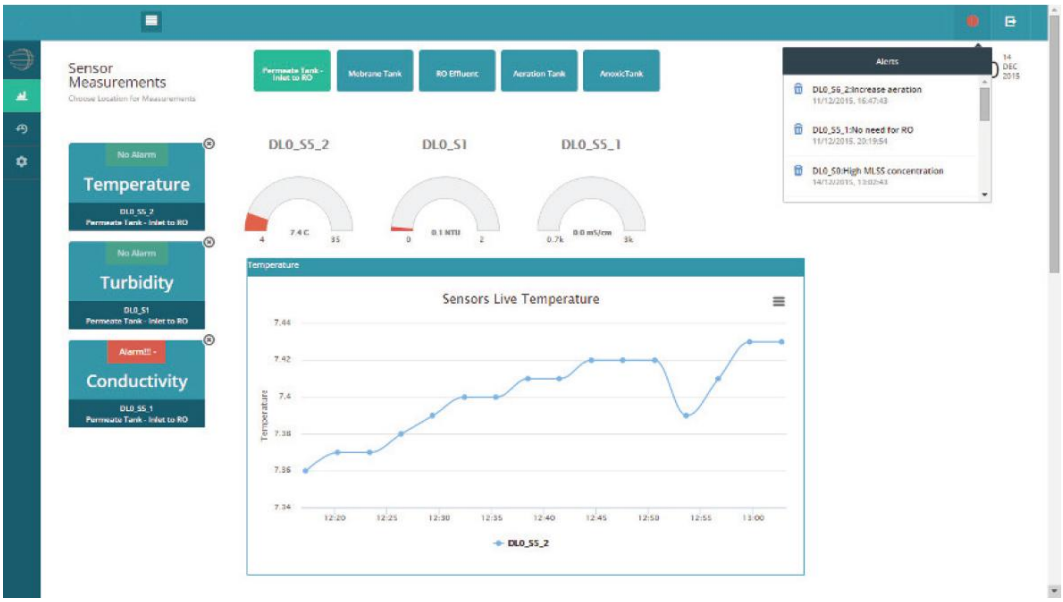


Figura 13 – Exemplo da monitorização de sensores na *dashboard* desenvolvida em [32].

O sistema disponibiliza ainda um mecanismo de alertas e eventos críticos, consulta de dados históricos, exportação de dados e automação de ações. Ainda de referir que utilizadores com perfil de administrador têm acesso a configurações específicas da rede de sensores e controladores [32].

Trata-se, assim, de um exemplo da utilização de *dashboard* para monitorização e controlo remoto de um sistema Internet das Coisas (*Internet of Things* – IoT) em contexto industrial.

Trabalho científico 2

Um outro trabalho relacionado com a indústria é apresentado em [33], onde se descreve o desenvolvimento de uma ferramenta de gestão de nós *edge* em larga escala (*Large-scale Edge node Management* – LEM). Este consiste num *software* de monitorização e configuração de dispositivos *edge* num contexto de Indústria 5.0, permitindo a distribuição de fluxos de trabalho a um grande número de dispositivos.

Esta ferramenta LEM funciona como uma camada de sobreposição à infraestrutura industrial existente, sendo por isso integrada com Node-RED, uma *framework open source* de programação *low code*, baseada em fluxos constituídos por nós escritos em JavaScript. Instalada nos dispositivos *edge*, a plataforma Node-RED permite o desenvolvimento de fluxos numa interface *web* e a sua exportação em formato JSON (*JavaScript Object Notation*) para posterior execução em dispositivos que tenham um ambiente de execução baseado em Node.js. Uma vez desenhado o fluxo original num dos dispositivos *edge*, este pode ser distribuído em larga escala pela ferramenta LEM que, para a comunicação relativa aos fluxos, recorre à API Node-RED HTTP [33].

A LEM também oferece funcionalidades de monitorização do sistema em que se sobrepõe, a gestão de dispositivos *edge*, a gestão de fluxos de trabalho e o controlo da execução das instâncias Node-RED. Para isso, a interface gráfica LEM (ver Figura 14), desenvolvida em Java Spring Boot, está organizada em três secções principais: *Monitoring*, *Deploy* e *Control*. A primeira apresenta os dados de estado dos dispositivos e fluxos, agrupando-os por dispositivo; a segunda permite a criação de novos fluxos num dispositivo *edge* ou o registo de um novo dispositivo; e a terceira possibilita ao operador as funcionalidades de iniciar ou parar instâncias Node-RED num dispositivo selecionado [33].

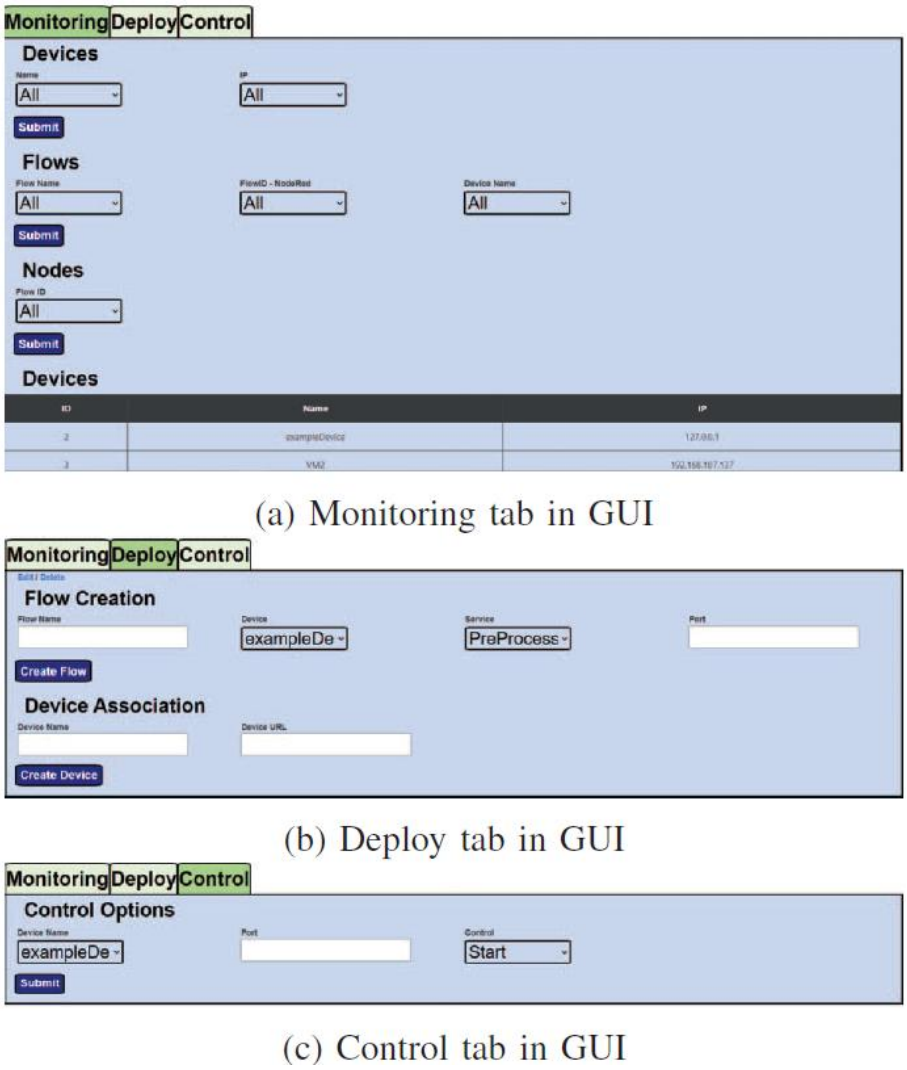


Figura 14 – Interfaces gráficas desenvolvidas em [33].

Este exemplo constitui um cenário de utilização de aplicação *web* de monitorização e gestão num sentido mais abstrato e estratégico, não envolvendo uma atuação direta sobre os processos físicos em causa.

Trabalho científico 3

Os autores de [34] desenvolvem um gémeo digital 3D (*digital twin* – representação virtual de um objeto que emula o seu comportamento em tempo real) integrado numa *dashboard* para monitorizar e controlar sistemas de casas inteligentes.

Os dados recolhidos pelos sensores IoT distribuídos pelo edifício são enviados via MQTT para a plataforma Home Assistant que processa, disponibiliza e armazena os dados numa base de dados VictoriaMetrics. Esta plataforma disponibiliza uma interface de configuração, Lovelace, que permite a criação e

personalização de *dashboards*, recorrendo a elementos de visualização e controlo configurados em YAML [34].

O elemento diferenciador neste trabalho, o gémeo digital, foi conseguido através da criação de um modelo interativo 3D da estrutura alvo, incluindo o mapeamento dos sensores e equipamentos de controlo. O modelo foi construído no programa SweetHome3D e, após exportação em formato OBJ, integrado recorrendo à biblioteca JavaScript Three.js que permite criar e visualizar gráficos interativos 3D em *browsers* [34].

A integração do gémeo digital na plataforma Home Assistant é assegurada pelo mapeamento da *stream* de dados dos sensores para os seus correspondentes visuais, bem como do mapeamento das interfaces de controlo inseridas no ambiente 3D que permitem as ações sobre os respetivos dispositivos físicos (ver Figura 15). Assim, os dados recolhidos são dinamicamente renderizados no gémeo digital, permitindo ao operador monitorizar parâmetros em tempo real – ocupação, humidade, temperatura, consumo de energia e estado da porta (aberta/fechada) – e gerir remotamente o espaço – energia, luz, temperatura, ventilação e ar condicionado – através das interfaces de controlo [34].

Figura 15 – Dashboard desenvolvida em [34] integrada na plataforma IoT.

A utilização de um gémeo digital permite ao operador navegar pelo modelo, rodá-lo, ampliando e diminuindo o zoom, o que facilita a localização e seleção dos sensores e equipamentos de controlo com os quais pretende interagir. De referir ainda a existência de um sistema de alertas configuráveis, a análise de dados históricos e a configuração de ações automatizadas [34].

Este é, assim, um exemplo da utilização de *dashboard* para monitorização e controlo remoto de um sistema IoT para casas inteligentes, cujo objetivo é oferecer uma manutenção proativa, otimizar o consumo de energia e melhorar a gestão geral do edifício alvo em tempo real.

Trabalho científico 4

Ainda no âmbito de casas inteligentes, em [35] é apresentado um sistema que recorre a modelos de *machine learning* (ML) com o objetivo de otimizar o consumo energético, sendo monitorizado e controlado a partir de uma aplicação *web*.

O sistema é composto por várias unidades locais (*local units* – LUs), constituídas por microcontroladores ESP32 ligados a sensores para recolha de dados de temperatura, humidade, luminosidade e ocupação, e uma unidade central (*central unit* – CU), representada por um Raspberry Pi4 que atua como central de processamento e *gateway*. Esta CU incorpora o modelo de ML, que toma decisões com base nos dados obtidos, e a ferramenta Node-RED que permite a comunicação entre os dispositivos físicos e disponibiliza APIs e outros serviços online. Os dados dos sensores são enviados à CU pelas LUs através de MQTT, que é também usado pela CU para envio de comandos de controlo de volta às LUs [35].

A gestão e controlo da CU é efetuado remotamente por meio de uma *dashboard* implementada em Node-RED que faculta uma interface gráfica para funcionalidades de visualização e operação remota em tempo real (ver Figura 16). Na aplicação *web*, o operador pode também alternar o modo de operação da CU entre manual e automático a qualquer momento. O modo automático, ativado por defeito, delega na CU a responsabilidade do controlo dos atuadores, que toma as decisões com base no algoritmo de ML. Por outro lado, o modo manual coloca o sistema em espera por comandos do utilizador que são enviados da *dashboard* à LU correspondente. Existe ainda um sistema de alertas que, por exemplo, notifica o operador de falhas a nível da LU [35].

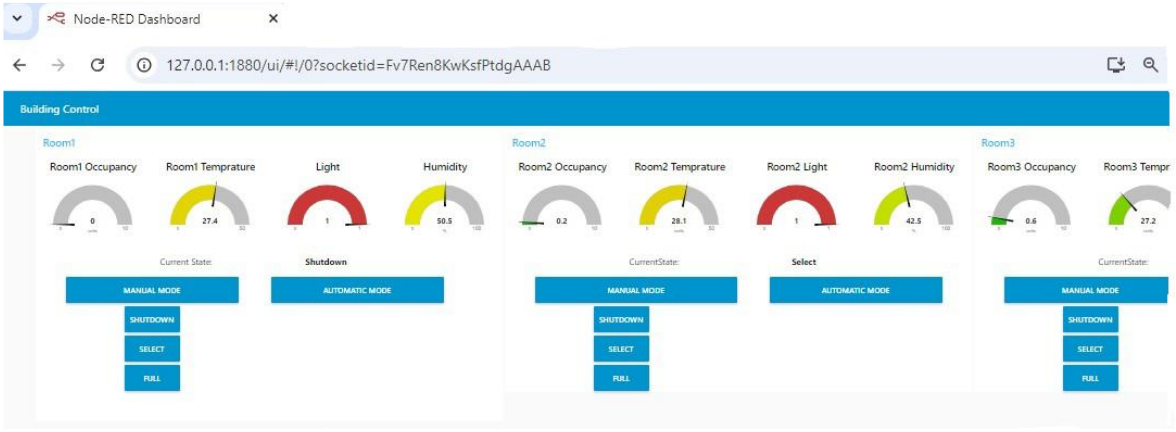


Figura 16 – Aplicação *web* de monitorização e controlo desenvolvido em [35].

Este trabalho encontra-se em fase embrionária, centrando-se na conceção do modelo de ML e nas comunicações entre os diferentes componentes (por exemplo, os atuadores são apenas LEDs para sinalizar o modo de operação) [35]. Contudo, isso não retira a relevância do projeto, constituindo um exemplo da utilização de *dashboards* aplicado à monitorização e controlo de sistemas de casas inteligentes.

Trabalho científico 5

Em [36], é relatado o desenvolvimento de uma plataforma *web* para visualização e controlo remoto de impressoras 3D baseadas em RepRap, um projeto *open source* capaz de imprimir a maioria dos seus componentes para criar réplicas de si mesmo. Neste trabalho, é construído um sistema de controlo remoto leve e embebido, eliminando a necessidade de instalação de *software* num computador, uma vez que o controlo da impressora é efetuado através de uma aplicação *web*.

Os autores montaram um cenário em que a camada física é constituída pela impressora 3D e sensores adicionais que permitem uma monitorização do processo, incluindo, por exemplo, sensores de temperatura e câmara de vídeo. Estes comunicam diretamente com um dispositivo embebido, um Raspberry Pi, responsável pelo controlo global do processo, incluindo a preparação dos ficheiros de código do modelo a imprimir (após indicação do utilizador e com os parâmetros definidos por este) [36].

No Raspberry Pi foi também configurado um servidor *web* recorrendo à *framework* Tornado, escrita em Python, que permite comunicação assíncrona e escalonamento de conexões, sendo ideal para *long polling* e para uso de WebSockets, o protocolo adotado para garantir uma comunicação em tempo real bidirecional entre o dispositivo e a interface *web*. De referir também a utilização do protocolo HTTP em tarefas como o carregamento do ficheiro do modelo STL

(formato do objeto a imprimir) e a configuração de parâmetros de impressão, operações menos exigentes em termos de latência [36].

A aplicação *web* (ver Figura 17), desenvolvida com recurso às tecnologias JavaScript, HTML5 e Bootstrap, permite a visualização e manipulação remota da impressora a partir de um *browser*. Destaca-se ainda a utilização da biblioteca Three.js que possibilita o manuseamento virtual do objeto a imprimir, nomeadamente no ajuste do seu posicionamento e tamanho [36].

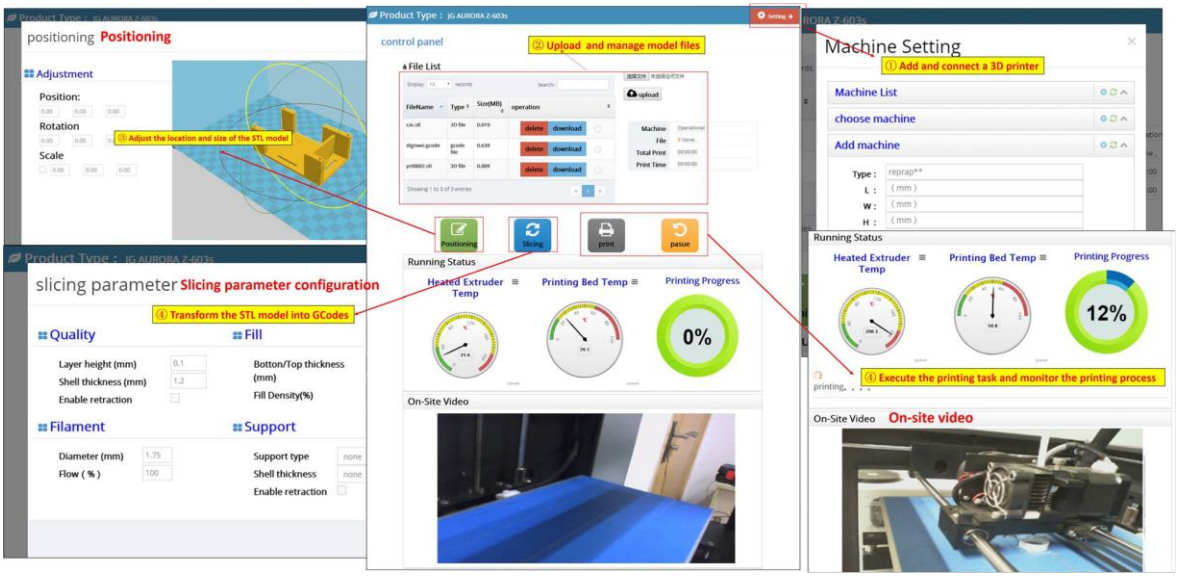


Figura 17 – Exemplo da aplicação *web* para monitorização e controlo remoto da impressora 3D desenvolvida em [36].

Este é um projeto que, recorrendo a uma aplicação *web*, permite a configuração do objeto a imprimir e monitorização do respetivo processo, constituindo uma abordagem acessível e intuitiva a um público mais alargado.

Trabalho científico 6

Num exemplo em contexto académico, os autores de [37] recorrem a modelos 3D para monitorizar e controlar remotamente plataformas de ensaio, aprimorando a *framework networked control system laboratory* – NCSLab. A solução suporta todo o processo de controlo de experimentação remota (monitorização, ajuste, medição, configuração visual e implementação de algoritmo de controlo) através de uma aplicação *web* (ver Figura 18), dispensando o uso de *plug-ins*. Esta independência é possibilitada pelo uso de HTML5 que, através da API WebGL, suporta a renderização de gráficos 2D e 3D interativos diretamente no *browser*.

Figura 18 – Exemplo de uma interface de um sistema de controlo para uma plataforma de ensaio de tanque duplo desenvolvido em [37].

O processo de monitorização e controlo recorre à interatividade 3D, gráficos e vídeo (não apenas do processo, mas também do utilizador), todos sincronizados em tempo real com os processos experimentais que ocorrem no laboratório físico. Os algoritmos de controlo da experiência são gerados em MATLAB/Simulink RTW e previamente associados pelos docentes às respetivas plataformas de ensaio; os utilizadores (alunos), usando a interface gráfica do NCSLab, selecionam depois o algoritmo a executar. Na mesma interface *web*, os utilizadores podem configurar os elementos gráficos (*widgets*) com os quais pretendem interagir, associando-os aos sinais e parâmetros dos algoritmos de controlo, permitindo assim a monitorização e controlo dos processos [37].

A configuração dos *widgets* é viabilizada por uma interface desenvolvida com a biblioteca *open source* Yahoo UI (YUI), escrita em JavaScript, que facilita a criação de páginas dinâmicas com objetos arrastáveis e redimensionáveis. A YUI, responsável pela criação e gestão da página principal, gera um objeto *Document Object Model* (DOM) designado *widget container*, onde são organizados os *widgets* pretendidos (cada um colocado num objeto DOM dedicado). A configuração da interface pelo utilizador é posteriormente guardada em formato JSON no servidor [37].

Para suportar as funcionalidades e características fornecidas pelos *widgets*, foram integradas no sistema várias tecnologias JavaScript, como Three.js (renderização de animações 3D sincronizadas com as plataformas de ensaio remotas), CanvasJS (criação de gráficos para monitorização em tempo real) e Fusioncharts (criação de gráficos *gauge* virtuais). A transmissão e exibição dos vídeos são realizadas através de APIs dedicadas e *iframe*. A comunicação entre os diferentes constituintes do sistema é efetuada utilizando o protocolo HTTPS e um algoritmo de *buffering* para mitigar atrasos no transporte dos pacotes [37].

Complementando a transmissão de vídeo em tempo real pelas câmaras das plataformas de ensaios, a aplicação *web* disponibiliza modelos 3D sincronizados

com o processo físico, fornecendo diferentes ângulos de visualização através de rotação e controlo de zoom do objeto. Estes modelos 3D são essenciais quando não há possibilidade de utilizar uma plataforma de ensaio real e o aluno tem de recorrer a experimentos virtuais, também disponíveis no sistema [37].

Este trabalho constitui um exemplo representativo da utilização de *dashboards* para monitorização e controlo remoto de procedimentos experimentais em ambiente académico.

Conclusões

Considerando os exemplos apresentados, a utilização de *dashboards* ou aplicações *web* para monitorização e controlo de sistema IoT é um cenário recorrente, tanto a nível industrial como em ambientes de menor escala e complexidade. Destaca-se, ainda, a pertinência destes sistemas em situações onde a presença física nem sempre é possível ou desejável, tornando o controlo remoto um fator essencial para garantir a eficiência, a segurança e a acessibilidade aos processos. A diversidade dos casos apresentados demonstra a versatilidade destas soluções e a sua adaptação a diferentes setores e objetivos.

3. Tecnologias de desenvolvimento *frontend*

Neste capítulo é efetuado um estudo sobre as tecnologias de desenvolvimento *frontend* mais em voga atualmente: Angular, React e Vue.js. Para além de uma descrição geral, cada uma é avaliada de acordo com os requisitos que se consideram importantes para a *dashboard* a implementar. No final do capítulo, é selecionada a *framework* a utilizar de acordo com uma escolha fundamentada.

3.1. Critérios de avaliação

A *dashboard* a implementar visa a monitorização em tempo real, bem como a respetiva configuração, de estações base 5G. Para assegurar uma resposta satisfatória do utilizador, a *dashboard* deverá garantir níveis de desempenho adequados. Apesar da oferta de diferentes tecnologias, *frameworks* e bibliotecas, é necessário analisar qual melhor se adequa e permite o desenvolvimento de uma aplicação que vá ao encontro dos objetivos propostos.

Assim, os elementos fundamentais e que deverão ser considerados para a escolha da tecnologia *frontend* são os seguintes:

- Latência da receção de dados e respetiva representação, incluindo representação gráfica atualizada frequentemente;
- Responsividade fluída face a interações do utilizador, nomeadamente utilização de botões, introdução de texto e números, interatividade de gráficos e exibição de notificações;
- Escalabilidade face a um aumento de dados e/ou utilizadores;
- Disponibilidade 24/7 garantindo o acesso nos diversos fusos horários;
- Garantia de segurança e autenticidade na comunicação de dados, bem como autenticação e controlo de acesso dos diferentes tipos de utilizador;
- Compatibilidade com ferramentas de terceiros que possam ser pertinentes a incluir no desenvolvimento;
- Fácil manutenção e atualização da aplicação, essencial em produtos em expansão.

3.2. Angular

Angular é uma *framework open source* completa para desenvolvimento *frontend*, desenvolvida e mantida pela Google. Lançada em 2016, é baseada em TypeScript, sendo considerada a sequela de AngularJS (2010) que utilizava JavaScript. Angular permite a criação de projetos robustos e tem sido amplamente utilizada no desenvolvimento de aplicações *web* de página única (*Single Page Application* – SPA) [38], [39].

A atualização da página é efetuada no DOM, executada no *browser* (renderização no lado do cliente) e recorre a um mecanismo de deteção de alterações para reduzir o número de renderizações. Este mecanismo é baseado no conceito de *zone* que é executado para um contexto ou zona específica, correspondendo cada componente a uma zona diferente. As alterações detetadas por este mecanismo desencadeiam as atualizações necessárias na página. Utilizando a estratégia *OnPush*, há um aumento do desempenho, uma vez que nestas situações apenas alterações em propriedades de *input* desencadeiam a renderização. Alternativamente, a renderização pode ocorrer no lado do servidor (*server-side rendering* – SSR) de modo a melhorar o desempenho relativamente ao carregamento da página, experiência do utilizador e *Search Engine Optimization* (SEO) [39], [40], [41], [42].

A comunicação de dados entre componentes pai e filho é bidirecional (*two-way databinding*). Isto permite uma atualização simultânea em tempo real dos dados em diferentes componentes através da utilização de eventos. Para comunicação assíncrona, Angular recorre a *Observables* que representam um fluxo de dados e, para evitar a emissão de eventos desnecessário e aumentar o desempenho da aplicação, tem de se garantir o cancelamento da sua subscrição após o término da tarefa. De referir ainda o uso de *Signals*, introduzidos nas versões mais recentes da *framework* e que permitem um aumento da eficiência da comunicação de dados entre os componentes. [39], [43], [44], [45].

O mecanismo de injeção de dependência promove a modularização dos projetos para uma melhor organização dos mesmos. Esta modularização aumenta também a sua eficiência, tanto a nível de desenvolvimento como de teste do código. A reutilização destes módulos facilita a manutenção e escalabilidade dos projetos [39].

De referir a técnica de compilação *Ahead of Time* (AoT) que permite a otimização da aplicação uma vez que compila o código do projeto e bibliotecas aquando da compilação ao invés de ocorrer ao nível do *browser*. Esta é a estratégia predefinida desde a versão 9 e melhora a eficiência ao converter o código TypeScript e HTML em JavaScript antes do *download* no *browser*, trazendo benefícios na velocidade de renderização, pedidos assíncronos, tamanho do *download*, deteção de erros e segurança [39], [46].

A documentação detalhada e simples, com diversos exemplos e tutoriais, bem como a escalabilidade oferecida e consistência do código, torna Angular uma *framework* popular no desenvolvimento de projetos de tamanho médio/grande. Para além disso, como é mantida pela Google, prevê-se a sua constante atualização e manutenção por um extenso período, o que torna esta *framework* confiável para projetos grandes e de longa duração. Todavia, a atualização dos projetos para as últimas versões de Angular pode tornar-se um processo moroso [38], [39].

A curva de aprendizagem associada a esta *framework* é uma das suas limitações: para além do domínio de TypeScript, envolve conteúdos complexos que é necessário dominar para tirar o melhor proveito do seu desempenho. Outra limitação é a necessidade da definição de parâmetros que, para pequenos projetos, pode tornar-se excessivo [38], [39].

3.3. React

React é uma biblioteca JavaScript *open source* para o desenvolvimento de interfaces de utilizador baseado em componentes, funcionando cada componente como um bloco reutilizável e independente. Foi criada por Jordan Walke em 2011 e, desde 2013, é desenvolvida e mantida pela Facebook, tendo também uma ampla comunidade. De mencionar a biblioteca React Native que permite a criação de aplicações móveis nativas utilizando os mesmos conceitos e princípios de React [38], [39].

Ao invés do DOM convencional, utiliza um DOM virtual (JS *structure*, estrutura de dados em JavaScript) que é uma representação leve em memória do DOM real. A atualização do DOM virtual em cada alteração de um componente permite ao React verificar as diferenças entre esta representação virtual e o DOM real, calcular o mínimo de alterações realmente necessárias e, posteriormente, atualizar o DOM. Esta estratégia reduz o número de renderizações da página e, consequentemente, um melhor desempenho e aumento na velocidade dos programas. A renderização é efetuada no lado do cliente após a receção dos ficheiros que contém a lógica e os componentes, no entanto, recorrendo a *frameworks* como Gatsby e NextJS ou à utilização de *React Server Components* a renderização pode ocorrer no lado do servidor [38], [39], [40], [47].

A comunicação de dados em React é unidirecional (*one-way databinding*) do componente pai para os componentes filhos e é efetuada através da utilização de *props* (propriedades) e *state* (estado), os tipos de “modelo” de dados em React. A informação passada de componente pai para os componentes filhos recorre a *props* que funcionam como argumentos passados em funções e, sendo *read-only*, não podem ser modificadas. A atualização dos dados passa pela gestão de estado. O *state* permite guardar dados e alterá-los face a interações em

componentes pai e/ou filhos: *state* é inicializado no componente pai, podendo ser depois modificado noutros componentes [39], [48], [49].

Para otimização dos programas, React tem disponível algumas técnicas como *React Memoization* e estruturas de dados imutáveis. *React Memoization* consiste na salvaguarda de dados resultantes de funções complexas que, caso não haja alterações nas suas dependências e *props*, não são executadas a cada renderização, evitando, assim, o consumo desnecessário de recursos. Já o uso de estruturas de dados imutáveis garante a persistência e consistência dos dados, simplifica a gestão de estado e facilita a comparação aquando da atualização do DOM [39], [50], [51], [52], [53], [54].

A atualização dos projetos para novas versões é facilitada pelo uso de *codemods scripts* que permitem uma migração automatizada na maioria das situações. Aquando de atualizações de React APIs ou de refatorizações, pode-se recorrer à ferramenta React Codemod para transformar o código existente compatível com os novos padrões. [39], [55].

Se por um lado a sua curva de aprendizagem é considerada acessível, pois tem como base HTML e JavaScript, as constantes atualizações requerem uma constante atualização de conceitos. De referir que JSX (JavaScript XML; extensão para escrita de JavaScript e HTML num mesmo ficheiro) e o *inline scripting* são considerados os aspetos mais complexos na aprendizagem de React [38], [39].

Como referido acima, React é uma biblioteca JavaScript e não uma *framework*. No entanto, o uso do termo "*frameworks*" na presente dissertação ao referir as três tecnologias *frontend* justifica-se pela relevância e pelas capacidades de React no desenvolvimento de interfaces *web*, equiparáveis às de Angular e Vue.js.

3.4. Vue.js

Vue.js é uma *framework open source* progressiva para desenvolvimento *frontend*, escrita em JavaScript e lançada em 2014 por Evan You, sendo baseada nos pontos fortes de Angular. Vue.js, tal como as anteriores tecnologias *frontend*, utiliza blocos de código como forma de reutilização e construção do programa, disponibilizando componentes (os principais elementos de blocos reutilizáveis que têm a sua própria representação visual e lógica), *composables* (focados na lógica de estado) e diretivas predefinidas e personalizáveis (atributos HTML que envolvem o acesso de baixo nível ao DOM). Vue.js é considerada uma *framework* leve, sendo amplamente utilizada em SPA e páginas dinâmicas [38], [39], [56], [57].

Tal como em React, a atualização das páginas é auxiliada por um DOM virtual. A renderização ocorre primariamente no lado do cliente, o que pode tornar a

página mais lenta, mas, tal como referido para Angular e React, é possível colocar a renderização a cargo do servidor (SSR). Para obter esta funcionalidade em Vue.js recorre-se à integração com Express ou ao uso da *framework* Nuxt (*framework* desenvolvida sobre Vue.js). Alternativamente, esta *framework* possibilidade também a técnica *Static Site Generation* (SSG) que, para páginas estáticas e iguais para qualquer utilizador, permite a renderização uma única vez durante o processo de compilação (*ahead of time*), sendo enviados apenas ficheiros HTML estáticos aquando do pedido [39], [58], [59].

Para além da utilização do DOM virtual, Vue.js apresenta algumas técnicas que visam aumentar a eficiência dos programas, como *lazy loading* e otimização da gestão de eventos. Em Vue.js a técnica *lazy loading* é especialmente aconselhável em grandes projetos e pode ser efetuada através de Vue Router, que utiliza importes dinâmicos (ao invés de importes estáticos), e/ou *async components*, que permite o carregamento de determinados recursos, componentes, imagens ou estilos, apenas quando necessários. Como os eventos são uma grande fonte de carga, nomeadamente eventos relacionados com ações do rato e *scroll* da janela, é necessário efetuar o seu correto tratamento. Por exemplo, a utilização de *event listeners* apenas nos elementos necessários e renderização condicional, e otimização através de *event modifiers*, *throttling* (limita a frequência a que um *event handler* é chamado) e *debouncing* (atraso na execução do *event handler*) [39], [60], [61], [62], [63].

Em Vue.js a comunicação entre os dados e o DOM pode ser unidirecional (*one-way binding*), bidirecional (*two-way databinding*) ou através de propriedades computadas. Em Vue.js esta comunicação é reativa, resultante da criação de *getters* e *setters* aquando da declaração das propriedades, sendo uma das características mais marcantes desta *framework* e essencial na gestão do estado da aplicação. A comunicação unidirecional é, tipicamente, do modelo para a *view* e é conseguida através de interpolação e da diretiva *v-bind*; já para a comunicação bidirecional recorre-se à diretiva *v-model*. Utilizando propriedades computadas, o resultado computado é atualizado sempre que há uma alteração nas suas dependências [39], [64], [65].

A sua principal vantagem é a curva de aprendizagem acessível, sendo considerada uma das tecnologias mais fáceis de aprender por principiantes. Outra vantagem é o tamanho dos projetos em Vue.js que são relativamente leves, o que traz também benefícios a nível de SEO. Esta *framework* é também reconhecida pela documentação disponibilizada e adaptabilidade (nomeadamente na migração que é facilitada pela compatibilidade de APIs entre versões de Vue.js e disponibilização de ferramentas de migração pela *framework*) [38], [39], [66].

Apesar da vasta comunidade e fóruns que ajuda a manter a *framework* e da sua crescente adoção, comparativamente com Angular e React, Vue.js não é suportada por grandes empresas, o que pode representar um risco para a sua continuidade e afastar a sua utilização em grandes e/ou longos projetos. Esta

framework tem também um ecossistema mais restrito, apesar de oferecer uma gama crescente de ferramentas e bibliotecas, como Vue Router mencionado acima e Vuex para gestão de estados. Outro ponto negativo prende-se com a incompatibilidade das aplicações desenvolvidas com versões antigas de sistemas e *browsers* [38], [39].

3.5. Cenário hipotético como teste de desempenho às *frameworks*

Para uma melhor análise e avaliação e posterior escolha fundamentada da *framework* de desenvolvimento, foi idealizado um cenário hipotético que inclui as caraterísticas essenciais para a *dashboard* final. Este cenário é implementado recorrendo a cada uma das *frameworks*, utilizando as mesmas soluções e verificando os mesmos parâmetros. Complementarmente, é desenvolvido um pequeno projeto auxiliar como forma a representar o *backend* da aplicação em Spring Boot, uma vez que é este o ambiente do *backend* desenvolvido pela empresa Allbesmart.

Como pontos para comparação e avaliação são utilizados como métricas:

- Número de linhas de código escrito;
- Volume de dados enviado durante o carregamento da página inicial;
- Tempo de carregamento da página inicial;
- Tempo decorrido para o envio de dados do servidor e respetiva representação na *dashboard* (simulação de telemetria);
- Tempo decorrido após a alteração de parâmetro na *dashboard* e respetiva atualização no servidor e representação na *dashboard*.

O *backend* desenvolvido inclui o envio contínuo (a cada 5 segundos) de um valor numérico como simulação de telemetria e uma propriedade para o teste de alteração de dados e respetiva notificação. O *frontend*, desenvolvido nas três *frameworks* supracitadas, representa uma *dashboard* simplista e com um *layout* básico. Os valores contínuos recebidos (simulação de telemetria) são apresentados num gráfico e o valor atual (ou último reportado) é também representado em separado. Para avaliação de alteração de valor de propriedade a partir da *dashboard*, inclui-se formulários de preenchimento e representação do valor atual (último reportado).

Os projetos descritos nos próximos pontos foram desenvolvidos e testados numa máquina com as seguintes características:

- Sistema operativo: Windows 11 Home;
- CPU: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz;
- GPU: NVIDIA GeForce RTX 3060 Laptop GPU e Intel® UHD Graphics
- RAM: 16 GB.

3.5.1. Simulação do *backend* utilizando Spring Boot

Para a criação e configuração inicial do projeto *backend* recorreu-se à ferramenta Spring Initializr, que permite a seleção das configurações base para projetos Spring. A Figura 19 apresenta as configurações utilizadas para gerar a estrutura do projeto que inclui as configurações gerais e as dependências necessárias para o problema em questão. Após o *download* do ficheiro ZIP, este foi importado no IntelliJ IDEA 2024.3.1 Community Edition, o Ambiente de Desenvolvimento Integrado (*Integrated Development Environment* – IDE) utilizado para o desenvolvimento desta simulação [67], [68], [69], [70].

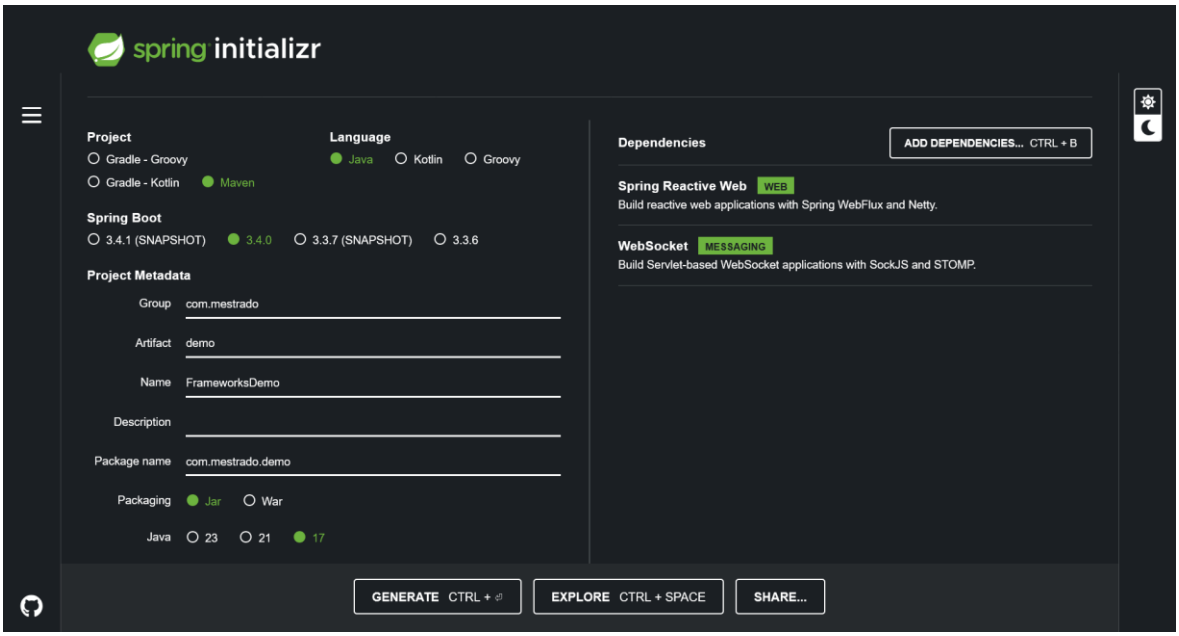


Figura 19 – Configuração utilizada para gerar o projeto Spring Boot recorrendo à ferramenta Spring Initializr.

No projeto para simulação de *backend*, foi desenvolvido um WebSocket com a criação de dois tópicos para subscrição e um tópico para envio de dados:

- '/topic/telemetry': tópico para subscrever a escuta do envio contínuo de telemetria;
- '/topic/telemetry/name': tópico para subscrever a escuta de alterações ao nome da telemetria;
- '/app/telemetry/name': tópico para envio de alterações ao nome da telemetria.

Para tal, foi programado uma configuração de WebSocket (*WebSocketConfig*) para configurar o *message broker* e registar os *endpoints*. O *message broker* permite definir o prefixo utilizado para o envio de mensagens ('topic') e o prefixo associado a método anotados com @MessageMapping ('app') que mapeiam pedidos enviados pelo cliente. O *endpoint* registado ('demo-ws') é utilizado no estabelecimento de conexões com o WebSocket [69], [70], [71].

O controlador para implementação dos métodos de comunicação com o cliente (*TelemetryStreamController*) consiste em dois métodos: uma para envio contínuo de dados e outro para alterar o nome da telemetria. O método *sendTelemetry* foi anotado com @Schedule para que seja chamado a cada 5 segundos e enviar uma mensagem de *broadcast* com um valor aleatório para todos os clientes que tenham subscrito o respetivo tópico ('/topic/telemetry'). O método *setTelemetryName*, anotado com @MessageMapping('/telemetry/name'), recebe pedidos com o novo nome da telemetria ('/app/telemetry/name') e, após processamento, envia uma mensagem de *broadcast* para todos os clientes que tenham subscrito o respetivo tópico ('/topic/telemetry/name') [69], [71], [72].

Adicionalmente, foram implementadas duas classes, *TelemetryMessage* e *TelemetryNameUpdated*, como modelo de dados. A primeira classe é utilizada para a simulação do envio contínuo de dados e a segunda aquando da modificação do nome da telemetria. Ambos os modelos têm como propriedade *timestamp* que guarda a data e hora da criação do objeto, essencial nos testes.

3.5.2. Cenário hipotético utilizando Angular

O desenvolvimento em Angular requer a instalação prévia de Node.js. Após garantir que a máquina de desenvolvimento tem uma versão compatível com os requisitos de Angular, procedeu-se à instalação da *framework*, sendo aconselhável recorrer à sua última versão. A instalação efetuou-se através da linha de comandos pela execução do seguinte comando: npm install -g @angular/cli [73], [74], [75].

Uma vez instalada a *framework*, procedeu-se à criação do projeto. Neste trabalho foram utilizados os seguintes comandos que, para além da criação do projeto, instalam as bibliotecas necessárias para a implementação do projeto:

- ng new angular-demo --no-standalone;
- cd angular-demo;
- npm install --save @angular/flex-layout@6.0.0-beta.16;
- npm install rxjs-compat @stomp/stompjs;
- npm install echarts -S;
- npm install ngx-echarts -S [74], [75], [76].

O IDE elegido para o desenvolvimento foi o Visual Studio Code e, uma vez criado o projeto, utilizou-se o terminal do IDE para correr a aplicação e verificar em tempo real as modificações durante a implementação: ng serve. Para criação de novos componentes, também se utiliza o terminal que adiciona as modificações aos ficheiros de configuração do projeto (app.module.ts) se necessário:

- ng generate component <nome-do-componente>;
- ng generate service <nome-do-serviço> [74], [75].

A *dashboard* implementada consiste na representação do nome e do valor da última telemetria recebida, no desenho de um gráfico dos dados recebidos e do formulário para a introdução e o envio de alteração do nome da telemetria. Seguindo as boas práticas recomendadas, o projeto foi dividido em diversos componentes que, para além de facilitar a programação e compreensão, permite uma construção em bloco, podendo cada um dos elementos ser utilizados em vários ficheiros e/ou projetos. Assim, foi desenvolvido um serviço *WebSocketService* para gerir a ligação ao *backend*, um componente *TelemetryComponent* que constitui a página principal da *dashboard*, um componente *LineChartComponent* responsável pelo desenho do gráfico e duas interfaces como modelo de dados (*TelemetryMessage* e *TelemetryNameUpdated*).

Nesta simulação utilizou-se STOMP (*Simple Text Oriented Messaging Protocol*), um protocolo de alto nível que opera por cima de WebSocket (considerado um protocolo de baixo nível) e permite a estruturação das mensagens, nomeadamente em JSON. A lógica da conexão ao *backend* e todos os métodos de subscrição de tópicos e envio de dados foram implementadas no componente *WebSocketService* que é utilizado no componente *TelemetryComponent* por injeção de dependência [70], [76], [77].

Como referido, o componente *TelemetryComponent* constitui neste exercício a página principal da *dashboard* e, aquando da sua inicialização, é chamado o método *listenTelemetry* do *WebSocketService* para começar a escutar a *stream* de dados de telemetria. Uma vez chegado um novo dado ao componente *TelemetryComponent*, este é adicionado a um *array* que guarda todos os valores recebidos e atualizada a propriedade que conecta os novos dados ao componente *LineChartComponent*. No *TelemetryComponent*, foi ainda programado o formulário para alterar o nome da telemetria que, em situações mais complexas, é recomendado isolar num componente. Para este formulário utilizou-se o módulo *FormsModule* do Angular que providencia diretivas para facilitar a gestão e programação destes elementos [78]. Sempre que uma alteração é pedida, é acionado o método no *WebSocketService* para envio do novo valor. A escuta de alterações a este valor foi estabelecida na inicialização do componente – similarmente ao efetuado para a *stream* de telemetria – e, assim, a alteração na *dashboard* é efetuada logo que o novo valor seja recebido; esta alteração também se reflete no gráfico, uma vez que também é alterada a propriedade enviada para o componente *LineChartComponent*.

O componente *LineChartComponent* é responsável pela configuração, desenho e atualização do gráfico, recorrendo à biblioteca Apache Echarts apresentada anteriormente. Efetivamente, utilizou-se a diretiva ngx-echarts que facilita a aplicação e gestão destes gráficos em projetos Angular. Neste componente não há qualquer referência ao tipo de dados representados, que são tratados pelo componente pai (*TelemetryComponent*), sendo um exemplo claro da programação em bloco característica da *framework* [79], [80], [81].

Os modelos *TelemetryMessage* e *TelemetryNameUpdated* refletem os modelos utilizados no *backend* para facilitar o manuseio destes objetos na *dashboard*.

A Figura 20 representa a *dashboard* desenvolvida em Angular utilizando o *browser* Google Chrome.

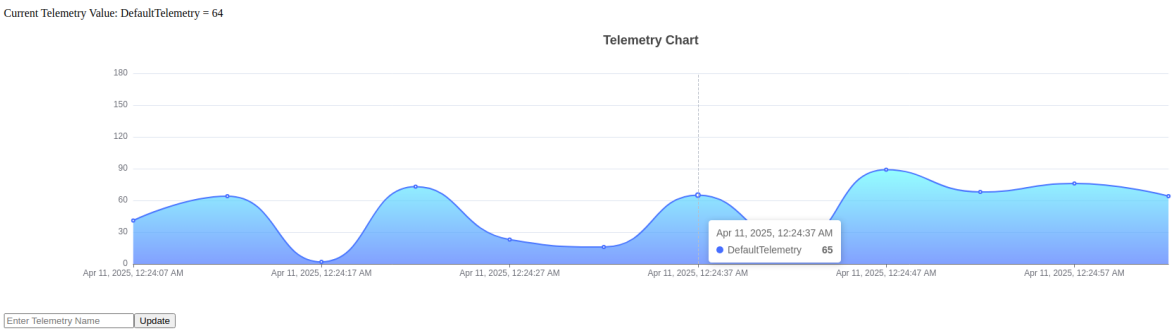


Figura 20 – *Dashboard* desenvolvida em Angular.

3.5.3. Cenário hipotético utilizando React

A programação com React, tal como em Angular, requer a instalação de Node.js. Assim, antes de iniciar um novo projeto é aconselhável verificar se a versão de Node.js e npm na máquina de desenvolvimento são compatíveis com os requisitos de React, através dos seguintes comandos:

- node -v;
- npm -v [82], [83].

Após esta verificação, procedeu-se à criação do projeto através do comando: npx create-react-app react-demo; que gerou uma diretoria "react-demo" com todos os ficheiros e estrutura base de um projeto React. Mais uma vez, o Visual Studio Code foi eleito como IDE de desenvolvimento e, uma vez aberto o projeto, através do terminal do IDE instalaram-se as seguintes dependências:

- npm install @stomp/stompjs;
- npm install --save @types/stompjs;
- npm i echarts echarts-for-react [82], [83], [84], [85], [86], [87].

Também a partir do terminal do Visual Studio Code correu-se a aplicação, o que permite verificar em tempo real o resultado da implementação no *browser*: npm start [82], [83], [85].

Como referido, o pretendido com estas simulações é uma comparação objetiva das três principais *frameworks* atualmente em voga no desenvolvimento *frontend*. Assim, a *dashboard* implementada é em tudo similar ao descrito anteriormente para a *framework* Angular: há a representação do nome e do valor da última telemetria recebida, o desenho dos dados recebidos em gráfico e o formulário para a introdução e o envio da alteração ao nome da telemetria. React também segue uma construção em bloco para facilitar a programação e entendimento através da reutilização de elementos. Como tal, a estrutura seguiu uma divisão similar ao descrito no projeto anterior, existindo, contudo, algumas nuances ao nível do código, uma vez que tecnologias diferentes assim o exigem. Foi, então, desenvolvido um componente *WebSocket* para gestão de toda a lógica relacionada com a ligação ao *backend*, um componente *Telemetry* que representa a *dashboard* e toda a sua representação e um componente *LineChart* responsável unicamente pelo desenho do gráfico.

A implementação da conexão ao *backend* utilizando STOMP é efetuada no componente *WebSocket* que contém os métodos de subscrição de escuta aos tópicos da *stream* da telemetria e de alterações ao nome da mesma e de envio de dados. Este componente é depois chamado no componente *Telemetry* que

recebe as variáveis que contém a *stream* de dados, a função a evocar aquando da alteração por parte de utilizador e a alteração ao nome da telemetria [86].

O componente *Telemetry* corresponde à página principal e apresenta os dados recebidos do componente *WebSocket*. Estes dados são guardados num *array* no componente *WebSocket* e é o *array* de telemetria que chega ao componente *Telemetry* onde é apresentado o último dado recebido e atualizada a propriedade enviada ao componente de desenho do gráfico; esta propriedade também é atualizada aquando da deteção de um novo nome da telemetria. O componente *Telemetry* é ainda responsável pelo formulário de alteração do nome da telemetria que, quando acionado o botão, chama o método de envio de dados do *WebSocket*.

No componente *LineChart* é configurado, desenhado e atualizado o gráfico Apache Echarts através do emprego da biblioteca echarts-for-react que simplifica a sua gestão. Tal como no correspondente no projeto Angular, este componente não tem qualquer conhecimento sobre o tipo de dados real que está a representar [87], [88], [89].

A *dashboard* resultante da implementação do projeto em React utilizando o *browser* Google Chrome encontra-se na Figura 21.

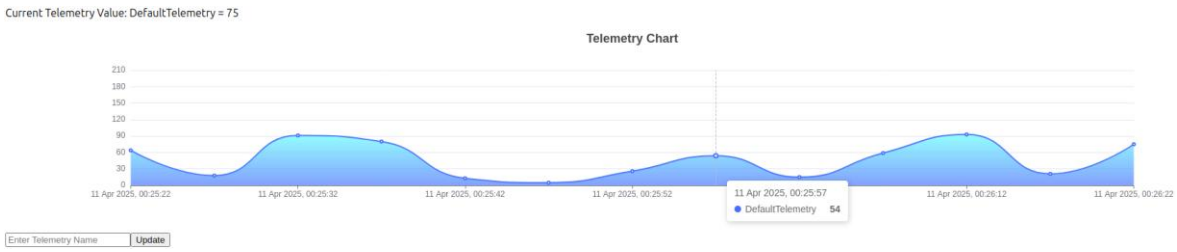


Figura 21 – *Dashboard* desenvolvida em React.

3.5.4. Cenário hipotético utilizando Vue.js

O desenvolvimento com Vue.js requer a instalação prévia de Node.js, tal como nas *frameworks* anteriores. Uma vez verificada a instalação do mesmo, procedeu-se à criação do projeto. Aquando da criação do projeto, a linha de comandos apresenta várias opções de configuração, nomeadamente a nível de utilização de TypeScript (neste projeto, optou-se pelo JavaScript), suporte a testes e ferramentas de auxílio de formatação. Adicionalmente, instalaram-se as dependências necessárias à implementação do projeto. Para este processo, utilizaram-se os seguintes comandos:

- npm create vue@latest;
- cd vue-demo;

- npm install;
- npm install @stomp/stompjs sockjs-client;
- npm install webstomp-client;
- npm install echarts vue-echarts [90], [91], [92], [93], [94], [95].

Mais uma vez, o IDE selecionado foi o Visual Studio Code e através do seu terminal correu-se a aplicação para verificar as alterações em tempo real, utilizando o comando: npm run dev [90], [92].

O projeto desenvolvido é a reprodução do cenário implementado em Angular e React: uma *dashboard* com a representação do nome e do valor da última telemetria recebida, o desenho do gráfico dos dados recebidos e o formulário para a introdução e o envio da alteração ao nome da telemetria. Tal como nas tecnologias anteriores, também em Vue.js há uma programação em bloco para a reutilização de componentes que facilita o trabalho do programador e a legibilidade da aplicação. A estrutura deste projeto é similar à dos anteriores, com as devidas adaptações de código face à tecnologia utilizada. Assim, foi desenvolvimento um componente *WebSocketComponent* para a gestão da lógica da ligação ao *backend*, um componente *TelemetryComponent* que constitui a *dashboard* e um componente *LineChartComponent* com o propósito único de lidar com a configuração e desenho do gráfico.

A conexão ao *backend* através de STOMP foi implementada no *WebSocketComponent* que inclui os métodos para subscrever os tópicos de escuta e envio de dados. A receção de dados é emitida e recebida pelos componentes que estejam à escuta que, neste caso, corresponde ao *TelemetryComponent* no qual o *WebSocketComponent* é integrado [93].

Como nos projetos anteriores, o componente *TelemetryComponent* reproduz a página principal. A integração do *WebSocketComponent* permite a receção dos dados emitidos por esse componente, sendo que a *stream* de telemetria é guardada num *array* e as alterações ao nome da telemetria são atualizadas na *dashboard*; em ambas as situações, há ainda a atualização da propriedade enviada ao componente *LineChartComponent*. Também neste projeto, o formulário para a alteração do nome da telemetria foi incluído neste componente.

O componente *LineChartComponent* é responsável pela configuração, desenho e atualização dos dados do gráfico, utilizando a biblioteca Apache Echarts, nomeadamente através do componente Vue-ECharts que agiliza todo o processo. Como nos projetos anteriores, este componente não lida diretamente com os dados de telemetria [94], [95].

Na Figura 22 representa-se a *dashboard* resultante da implementação do projeto em Vue.js utilizando o *browser* Google Chrome.

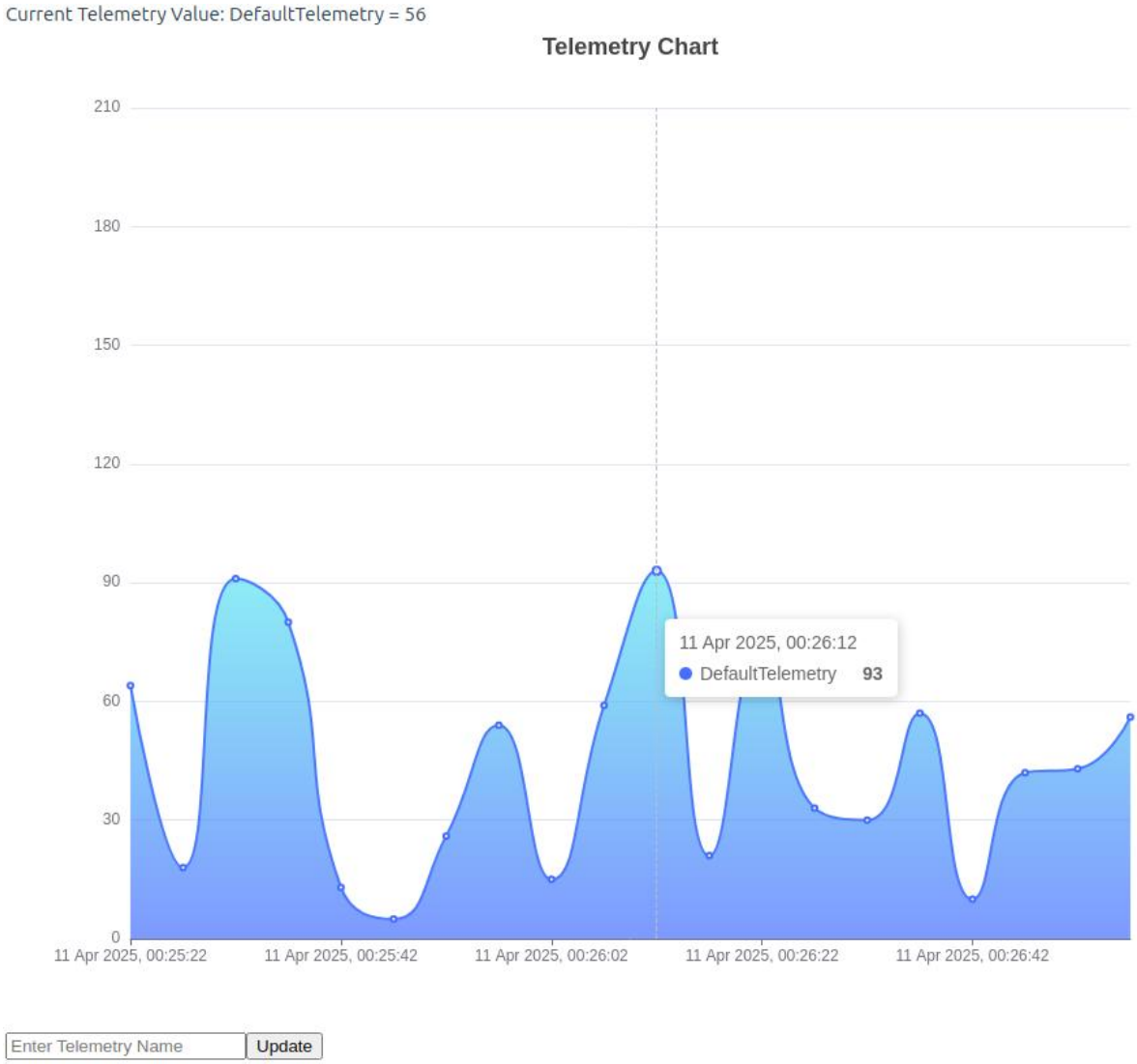


Figura 22 – *Dashboard* desenvolvida em Vue.js.

3.5.5. Preparação e configuração de ambientes de teste

Os testes às *dashboards* desenvolvidas nas três *frameworks* realizaram-se em máquinas virtuais (*Virtual Machine* – VM), tentado simular uma situação real. Para tal, foram criadas três VMs Ubuntu 24.04 LTS utilizando Oracle VirtualBox Manager versão 7.1.6, cada uma com as seguintes configurações:

- 4000 MB de memória base;
- 2 CPU;
- 25GB de disco rígido virtual [96].

Após a instalação do sistema operativo, efetuou-se uma atualização do sistema, desabilitou-se a sincronização do relógio da VM com o da máquina *host*

e instalou-se o *Network Time Protocol* (NTP). O protocolo NTP permite a sincronização dos relógios em rede, prevenindo possíveis desvios dos relógios dos sistemas. Neste contexto, servidores NTP comunicam entre si e relógios de referência para obter valores de data e hora exatos que são depois providenciados aos clientes NTP que requisitam esse serviço. O recurso a este protocolo prende-se com a necessidade de garantir a sincronização dos relógios nas VMs e, embora estas obtenham inicialmente o valor do relógio a partir da máquina *host*, foram verificadas algumas inconsistências. Assim, optou-se por esta estratégia, dado que algumas das métricas a avaliar consistem na comparação de data e hora em diferentes máquinas [97], [98].

Na VM destinada ao papel de servidor, instalou-se os seguintes *softwares* para correr os projetos de *backend* e *frontends*:

- Java: para correr o projeto *backend* .jar;
- Node.js: necessário para a instalação do *software* seguinte;
- http-server: um servidor HTTP estático simples, mas robusto para este cenário [99], [100].

Na VM destinada ao papel de cliente e, como tal, aceder às várias *dashboards*, foi instalado o *browser* Google Chrome, escolhido por disponibilizar a ferramenta de desenvolvimento Lighthouse que oferece um relatório de desempenho, acessibilidade, boas práticas e SEO, associado a sugestões sobre o que pode ser melhorado. Cada secção é avaliada de diferente modo, encontrando-se em baixo as métricas utilizadas e/ou pontos considerados em cada temática:

- Desempenho:
  - *First Contentful Paint* (FCP): mede o tempo que demora para o *browser* renderizar o primeiro conteúdo de DOM;
  - *Speed Index*: mede a rapidez com que o conteúdo é visualizado aquando do carregamento da página;
  - *Total Blocking Time* (TBT): mede o tempo total em que a página fica bloqueada de responder a ações do utilizador, sendo calculada através da soma das partes bloqueantes de todas as tarefas longas entre o FCP e o *Time to Interactive* (TTI, antiga métrica correspondente ao tempo que demora até a página ficar totalmente interativa);
  - *Largest Contentful Paint* (LCP): mede o tempo que o maior conteúdo demora a renderizar e ser visível para o utilizador;

- *Cumulative Layout Shift* (CLS): medida do valor mais elevado de mudanças de *layout* inesperadas resultantes de alterações de posição de elementos e conteúdos que podem ocorrer em carregamentos de recursos assíncronos;
- Os valores são reportados maioritariamente em milissegundos e convertidos numa escala de 0 a 100. O valor final é uma média ponderada, que pode ser ajustada pelo utilizador. Para estes testes mantiveram-se os valores predefinidos que são regularmente avaliados e ajustados pela equipa de desenvolvimento da ferramenta;
- Após a análise das métricas, o relatório fornece um conjunto de oportunidades e diagnósticos com sugestões de melhoria;

- Acessibilidade:
  - A pontuação é obtida através de uma média ponderada da avaliação de impacto de utilizador axe, sendo um diagnóstico da acessibilidade da página;
  - A lista de itens avaliados inclui, por exemplo, todos os botões terem nomes acessíveis, o documento apresentar um elemento <title> e as imagens conterem atributo [alt];
  - Cada auditoria é avaliada em aprovada/reprovada e representa um peso diferente;
- Boas práticas – sinalização de elementos de boas práticas em falta, divididas por:
  - Boas práticas gerais, como falta de *doctype* HTML e erros na consola;
  - Sugestões para tornar a página mais rápida, como a utilização de *listeners* passivos para ações de *scrolling*;
  - Sugestões para tornar a página mais segura, como a não utilização HTTPS e redireccionamento de tráfego HTTP para HTTPS;
  - Sugestões de melhoria da experiência de utilizador, como permitir a ação “colar” em campos de introdução de dados e utilização da *tag* “viewport” para indicação da largura ou escala iniciais;
  - Sinalização da utilização de tecnologias e APIs descontinuadas e obsoletas;
  - Auditorias de diagnóstico, nomeadamente a deteção de bibliotecas JavaScript;

- SEO – indicação de sugestões para otimizar os mecanismos de pesquisa a nível de:
    - Melhoria da compreensão do conteúdo da página por parte de motores de busca através de, por exemplo, descrição na *tag* <meta>;
    - Ajuda nos mecanismos de *web crawl* e indexação ao indicar, por exemplo, a existência de pedidos com retorno de códigos HTTP de falha;
    - Melhoria da compatibilidade com dispositivos móveis, como tamanho de fonte inadequado [101].

Para além desta ferramenta, foi também utilizado o separador Network, nomeadamente os valores relativos ao tempo de carregamento da estrutura DOM (*DOMContentLoaded*), ao tempo de carregamento (*Load*) e à quantidade de dados transferidos (*MB transferred*). O evento *DOMContentLoaded* é espoletado quando o documento HTML inicial é carregado, permitindo a interação com a estrutura básica da página, podendo não estar ainda disponíveis recursos como estilos, imagens e *subframes*. Já o evento *Load* é espoletado quando todos os recursos da página (estilos, imagens, *scripts* e *subframes*) estão completamente carregados, sendo a partir deste momento possível visualizar e interagir com a página no seu todo. Ambos os valores são, geralmente, utilizados para avaliar o desempenho do carregamento inicial da página. Por fim, o valor dos dados transferidos refere-se à quantidade total de dados descarregados de todos os recursos da página, geralmente comprimidos, servindo como métrica para avaliar o volume de dados no carregamento inicial da página [102], [103], [104], [105].

Para a obtenção dos dados de data e hora necessários para analisar as métricas propostas, foram concebidos dois ambientes de teste cujas diferenças se prendem com o modo de obtenção de dados. Numa primeira abordagem, recorreu-se à utilização de *logs* impressos diretamente no separador Console das ferramentas de desenvolvimento do *browser*; e numa segunda análise, implementou-se um sistema centralizado de *logs* na terceira VM que recebe estes *logs* provenientes dos diferentes projetos, *backend* e *frontend*, cujos detalhes são pormenorizados nos próximos pontos.

Ambiente de teste 1

Nesta abordagem, em que são usados apenas duas VMs (uma como servidor e outra como cliente), foi necessário introduzir a impressão de *logs* em locais estratégicos nos projetos *frontend*. Assim, utilizando a variável *timestamp* que guarda a data e hora da criação ou alteração dos modelos de dados e a função de obtenção data e hora do momento, a impressão dos *logs* foi inserida:

- Após a atualização e representação de um novo dado de telemetria recebido é apresentada a data e hora de criação do objeto e a data e hora do momento;
- Imediatamente antes do envio do pedido de alteração de nome da telemetria é apresentada a data e hora do momento;
- Após a receção e atualização na *dashboard* da alteração de nome da telemetria é apresentada a data e hora de atualização do valor (no servidor) e a data e hora do momento.

Para além dos *softwares* e das alterações aos projetos supracitados, não foi necessário a instalação ou configuração de outros elementos neste ambiente que se encontra representado no esquema da Figura 23.

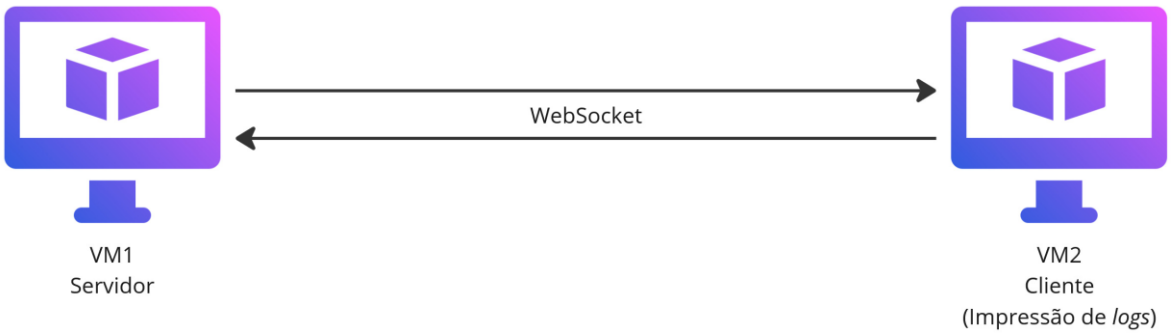


Figura 23 – Esquema do ambiente de teste 1.

Ambiente de teste 2

Num segundo contexto, recorreu-se a um sistema de *logs* externo centralizado que recebe registos do *backend* e dos projetos *frontend*, permitindo uma comparação de valores de data e hora independente.

Os sistemas de *logs* centralizado permitem a recolha de dados de diferentes fontes num único local o que facilita e agiliza a sua gestão e análise, providenciando uma visualização da atividade do sistema especialmente proveitosa durante a resolução de problemas. Genericamente, consistem na recolha e respetivo armazenamento, no processamento e análise, na pesquisa e consulta, na visualização e na monitorização e alertas. Estes sistemas são

particularmente úteis em ambientes distribuídos e existem várias soluções no mercado [106], [107].

Dada a natureza do trabalho, optou-se pela utilização do ELK Stack, uma solução *open source* que suporta dados provenientes de diversas fontes e capacidades de processamento e análise compatíveis com o objetivo da sua aplicação. Esta solução é constituída por três ferramentas, Logstash, Elasticsearch e Kibana, cuja arquitetura está esquematizada na Figura 24. Neste sistema, os *logs* são enviados ao Logstash que é responsável pelo processamento de dados de diferentes fontes e os disponibiliza a Elasticsearch. Por sua vez, a ferramenta Elasticsearch guarda e indexa os dados, permitindo a sua pesquisa e análise. Finalmente, estes dados são visualizados na ferramenta Kibana, uma interface *web* que possibilita a interação com os dados através de diversos relatórios e formatos [108], [109], [110], [111].

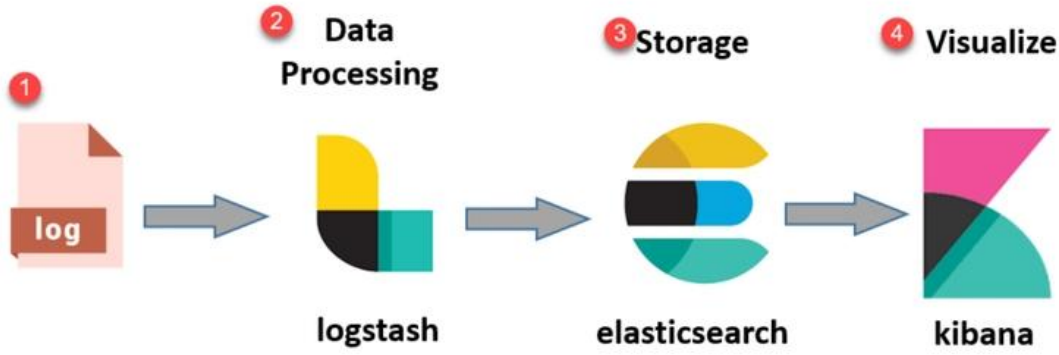


Figura 24 – Arquitetura da solução ELK Stack [108].

A solução ELK Stack foi instalada na terceira VM, bem como o *software* Java, uma dependência necessária, através dos seguintes passos:

- Instalar Java;
- Adicionar o repositório Elasticsearch:
  - Adição da chave do repositório;
  - Adição do repositório Elasticsearch à lista de repositórios do sistema;
- Instalar Elasticsearch:
- Alterar o ficheiro de configuração Elasticsearch:
  - Propriedade “network.host” com o valor 0.0.0.0 (para vincular a todas as interfaces de rede da VM);
  - Propriedade “http.port” com o valor 9200 (porto para envio dos dados a Elasticsearch)

    - Propriedade “discovery.type” com o valor “single-node” (configuração de *cluster* de nó único, adequado para implementações em pequena escala);
- Configurar a alocação de memória para JVM (*Java Virtual Machine*), definindo um máximo de 512MB como valor mínimo e máximo;
- Ativar o serviço Elasticsearch no início do sistema;
- Reiniciar o serviço Elasticsearch;
- Instalar Logstash;
- Criar ficheiro de configuração Logstash para o cenário pretendido:
    - Especificar a receção de dados via HTTP no porto 8081 e formato “json_lines”;
    - Configurar o envio de eventos HTTP para Elastichsearch, vinculando o endereço e porto anteriormente configurados e indexando os dados de acordo com o projeto de origem e data;
- Ativar o serviço Logstash no início do sistema;
- Reiniciar o serviço Logstash;
- Instalar Kibana;
- Alterar o ficheiro de configuração Kibana:
    - Propriedade “server.port” com o valor 5601;
    - Propriedade “server.host” com o valor “localhost”;
    - Propriedade “elasticsearch.hosts” com o valor “[“http://localhost:9200”]”;
- Ativar o serviço Kibana no início do sistema;
- Reiniciar o serviço Kibana [109], [110], [111].

A *dashboard* Kibana pode, então, ser acedida através do *browser*, onde são criados padrões de índice de acordo com o estabelecido no ficheiro de configuração do Logstash. É através destes padrões que são visualizados os eventos recebidos no sistema de *logs* [111].

Para além destas configurações, foi também necessário proceder a alterações nos projetos *backend* e *frontend* implementados unicamente com o intuito de envio de *logs*. De forma a minimizar possíveis atrasos nestes envios, todos os projetos utilizam o protocolo HTTP para comunicar diretamente com Logstash.

Assim, no projeto de *backend* foi necessário criar um ficheiro “logback-spring.xml” na diretoria “resources” que é responsável pela criação de ficheiros *log*. Este é um ficheiro de configuração onde se declaram os *loggers* que

capturam os eventos a registar e passam para os *appenders*, também declarados neste ficheiro de configuração, responsáveis pela documentação e envio dos *logs* para o endereço e porto de destino aqui identificados. Neste cenário, em que se pretende o envio via HTTP, foi necessário implementar um *appender*, definindo o objeto *log* a enviar. No controlador *TelemetryStreamController*, onde estão implementados os métodos de comunicação com o cliente, foi adicionado como propriedade o *logger* que é usado para registar os eventos imediatamente antes do envio de telemetria contínua e imediatamente antes do envio do novo nome de telemetria após modificação da mesma [112].

Nos projetos *frontend*, foi necessário incluir bibliotecas que possibilitam a realização de pedidos HTTP, nomeadamente a adição do módulo *HTTPClient* em Angular e a instalação da biblioteca Axios em React e Vue.js. Respeitando as particularidades de cada tecnologia, foi criado o serviço *LogService*, onde se implementou o pedido HTTP POST para envio de *logs*. Neste pedido, definiram-se a mensagem a enviar e o endereço e porto de destino, incluindo ainda o cabeçalho “Content-Type” com o valor “application/json” [113], [114], [115], [116], [117].

O serviço foi, então, injetado (Angular) ou importado (React e Vue.js) nos componentes e/ou serviços correspondentes à *Telemetry* e ao WebSocket, responsáveis por, respetivamente, implementar a página principal da *dashboard* e comunicar com o *backend*. No componente responsável pela página principal, o envio de *log* foi inserido imediatamente após a atualização da *dashboard* aquando da receção de novo valor de telemetria e da receção e atualização do novo nome de telemetria. No componente responsável pelo WebSocket, o envio de *log* foi introduzido imediatamente antes do envio da alteração do nome da telemetria para o *backend*.

Devido ao mecanismo de segurança CORS (*Cross-Origin Resource Sharing*) presente nos *browsers*, incluindo o Google Chrome, os pedidos efetuados a domínios diferentes são bloqueados. Assim, o envio de *logs* a partir das aplicações *web*, através de pedidos HTTP, para uma máquina externa (a VM configurada com o sistema centralizado de *logs*) é barrado. Existem várias estratégias para resolver esta limitação num contexto real, no entanto, como neste trabalho os *logs* são utilizados para comparação de data e hora, sendo essencial reduzir atrasos adicionais, optou-se pela desativação deste mecanismo de segurança na VM designada como cliente através do comando: chromium-*browser* --disable-web-security --user-data-dir="[directory]" [118], [119].

Na Figura 25 representa-se um esquema deste ambiente, destacando os protocolos utilizados na comunicação entre as diferentes VM intervenientes.

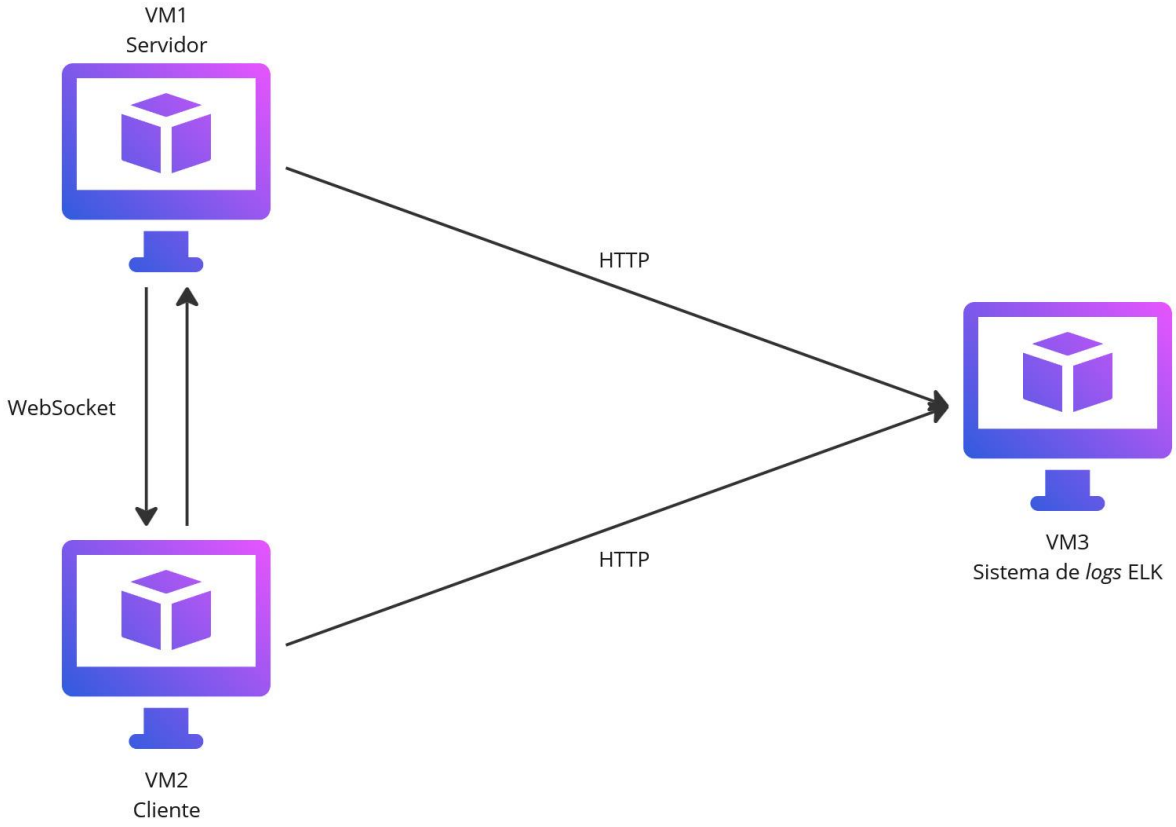


Figura 25 – Esquema do ambiente de teste 2.

3.5.6. Execução dos testes

Para a contagem de linhas de código utilizou-se a extensão VS Code Counter disponível gratuitamente no Marketplace do Visual Studio Code. Esta extensão permite a contabilização de linhas, com diversos relatórios para analisar o número de linhas por diretorias, ficheiros e linguagem de programação [120]. A contagem foi efetuada em cada projeto *frontend* sem incluir as especificações introduzidas em cada ambiente teste.

Com o propósito de simular um ambiente de produção, procedeu-se ao *deployment* dos projetos, cujos ficheiros obtidos foram utilizados nos testes, ao invés de recorrer diretamente aos mecanismos disponibilizados pelos IDEs. De seguida descreve-se como se obtêm estes ficheiros:

- Projeto Spring Boot:
  - Nas opções “Maven”, selecionar “Lifecycle” → “package” (ver Figura 26);
  - Uma vez terminado o processo, no terminal é indicada a localização do ficheiro .jar;
  - Colocar o ficheiro .jar na máquina servidor;

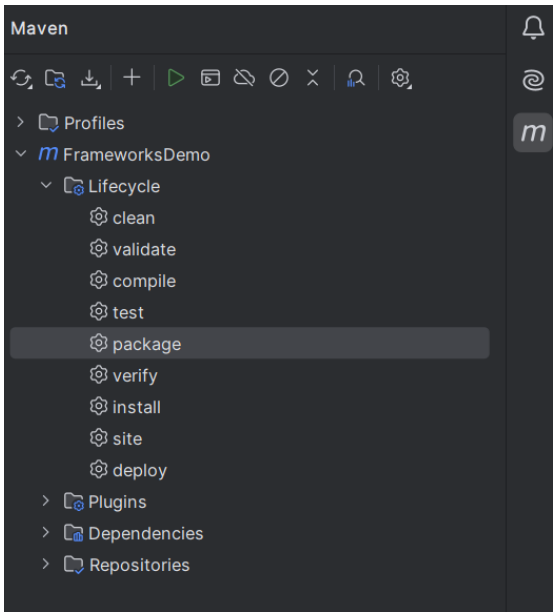


Figura 26 – Obtenção do ficheiro .jar do projeto Spring Boot no IDE IntelliJ.

- Projeto Angular:
  - No terminal do IDE, utilizar o comando: ng build --configuration production;
  - Uma vez terminado o processo, no terminal é indicada a localização da diretoria “dist”;
  - Colocar a diretoria “dist” na máquina servidor [121], [122], [123];
- Projeto React:
  - No terminal do IDE, utilizar o comando: npm run build;
  - Uma vez terminado o processo, no terminal é indicada a localização da diretoria “build”;
  - Colocar a diretoria “build” na máquina servidor [83], [124];
- Projeto Vue:
  - No terminal do IDE, utilizar o comando: npm run build;
  - Uma vez terminado o processo, no terminal é indicada a localização da diretoria “dist”;
  - Colocar diretoria “dist” na máquina servidor [90].

Estes passos foram realizados em três momentos distintos para a obtenção dos ficheiros para os diferentes testes e ambientes, nomeadamente os projetos sem as particularidades de ambiente de teste introduzidas, os projetos com as

especificidades para o ambiente de teste 1 e os projetos com as especificidades para o ambiente de teste 2.

Após a configuração dos ambientes de teste e o *deployment* dos projetos, executou-se o programa de *backend* recorrendo ao comando: java -jar demo-spring-boot.jar; e de seguida, os projetos de *frontend* com os comandos:

- Angular: http-server dist -p 8081;
- React: http-server build -p 8082;
- Vue.js: http-server dist -p 8083.

Inicialmente, obtiveram-se os valores de volume de dados e tempos de carregamento iniciais do separador Network do *browser* e o relatório do separador Lighthouse do *browser*, descritos anteriormente. Para estes testes utilizaram-se os projetos sem especificidade de testes (isto é, sem impressão ou envio de *logs*), recorrendo apenas às VMs servidor e cliente.

Para os testes referentes à data e hora de envio de dados contínuos e à data e hora de alteração de parâmetro, foram utilizados os projetos com as particularidades e as configurações de ambiente detalhados anteriormente.

No ambiente de teste 1, antes de iniciar quaisquer testes, garantiu-se que ambas as máquinas estavam sincronizadas recorrendo ao NTP; esta sincronização foi verificada após cada conjunto de testes por projeto através do comando: ntpq -p. Caso o *offset* do servidor ao qual está sincronizado no momento (identificado pelo asterisco) for superior a 2 ms, reinicia-se o serviço NTP e verifica-se novamente. Repete-se o processo até obter um valor de *offset* inferior a 2 ms em todas as máquinas intervenientes [97], [98].

Com a sincronização assegurada, procedeu-se à recolha dos dados que resultam dos *logs* colocados no código e que são impressos no separador Console do *browser*. Inicialmente, registaram-se os primeiros dez dados recebidos, com indicação da data e hora de criação do objeto e data e hora do momento de apresentação na *dashboard*. De seguida, efetuou-se a alteração do nome da telemetria dez vezes, registando-se a data e hora de envio do pedido ao servidor, data e hora de atualização no servidor e data e hora de atualização da *dashboard*. Esta sequência foi efetuada para cada aplicação *frontend*, sendo verificada a sincronização do relógio com o servidor NTP entre cada conjunto de testes.

No ambiente de teste 2, o processo de obtenção destes dados foi similar, considerando que, nesta situação, os dados de data e hora foram registados no sistema de *logs* centralizado configurado na terceira VM. Neste ambiente, as *dashboards* dos vários projetos *frontend* foram acedidas através do *browser* com a segurança desabilitada, como referido anteriormente. Inicialmente, esperou-se pela receção dos dez primeiros dados e, de seguida, procedeu-se à alteração do

nome da telemetria dez vezes. Este conjunto de testes foi efetuado para cada aplicação *frontend*. Tanto os dados emitidos pelo *backend*, como os dados apresentados e os pedidos de alteração na parte do *frontend* foram registados no sistema centralizado de *logs*, sendo posteriormente recolhidos a partir da *dashboard* Kibana, como descrito anteriormente.

3.5.7. Resultados

No Apêndice A encontram-se os detalhes da contagem de linhas de código, resumidas na Tabela 1.

Tabela 1 – Resumo das contagens totais de linhas de código dos três projetos *frontend*.

Framework	Linhas Código (total)	Linhas Código src
Angular	15501	419
React	18007	300
Vue.js	5358	358

No Apêndice B encontram-se os valores recolhidos no separador Network, correspondente ao volume de dados recebidos e aos tempos do carregamento inicial da página. A Tabela 2 sumariza os valores considerados mais relevantes.

Tabela 2 – Valores dos dados transferidos e dos tempos de carregamento iniciais da página dos três projetos *frontend*.

Framework	Dados transferidos (comprimidos)	Tempo de carregamento da estrutura DOM (*DOMContentLoaded*)	Tempo de carregamento (*Load*)
Angular	1.4 MB	354 ms	357 ms
React	1.3 MB	242 ms	242 ms
Vue.js	1.1 MB	371 ms	385 ms

Nas Figura 27 a Figura 32, encontram-se os dados considerados mais relevantes para o caso em estudo retirados do relatório da ferramenta de desenvolvimento Lighthouse.

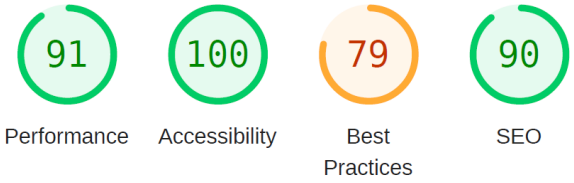


Figura 27 – Pontuações gerais do relatório obtido pela ferramenta Lighthouse do projeto Angular.

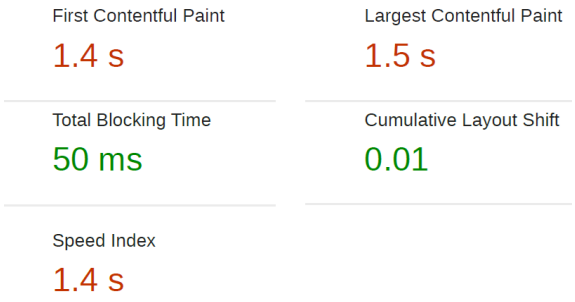


Figura 28 – Métricas de desempenho do relatório obtido pela ferramenta Lighthouse do projeto Angular.

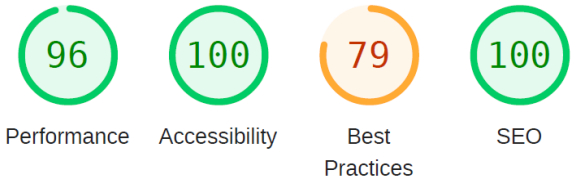


Figura 29 – Pontuações gerais do relatório obtido pela ferramenta Lighthouse do projeto React.

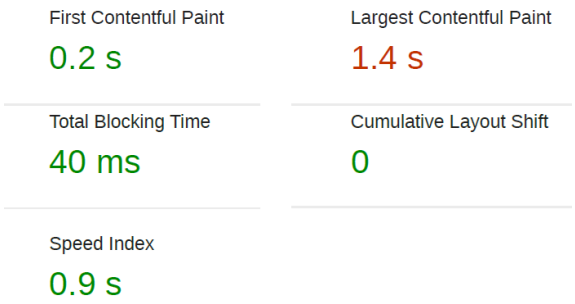


Figura 30 – Métricas de desempenho do relatório obtido pela ferramenta Lighthouse do projeto React.

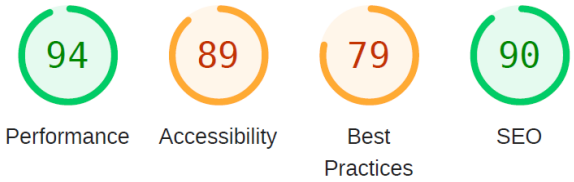


Figura 31 – Pontuações gerais do relatório obtido pela ferramenta Lighthouse do projeto Vue.js.

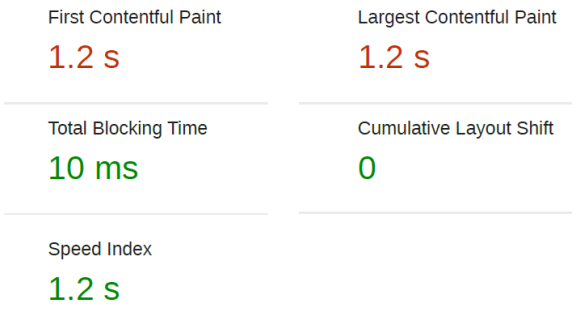


Figura 32 – Métricas de desempenho do relatório obtido pela ferramenta Lighthouse do projeto Vue.js.

Relativamente ao ambiente de teste 1, os valores recolhidos dos *logs* introduzidos no código para obtenção de data e hora encontram-se no Apêndice C, sendo as médias das latências calculadas nas Tabela 3 e Tabela 4.

Tabela 3 – Média de latência do envio de dados do servidor e respetiva representação na *dashboard* no ambiente de teste 1.

Framework	Latência (ms)
Angular	9.6
React	8.8
Vue.js	4.4

Tabela 4 – Média de latências após a alteração de parâmetro na *dashboard*, respetiva atualização no servidor e sequente representação na *dashboard* no ambiente de teste 1.

Framework	Latência cliente-servidor (ms)	Latência (ms)
Angular	5.7	12.1
React	3.5	14
Vue.js	3.7	9.8

Quanto ao ambiente de teste 2, os valores de data e hora obtidos pelos registos no sistema centralizado de *logs* encontram-se no Apêndice D e as respetivas médias das latências calculadas nas Tabela 5 e Tabela 6.

Tabela 5 – Média de latência do envio de dados do servidor e respetiva representação na *dashboard* no ambiente de teste 2.

Framework	Latência (ms)
Angular	13.8
React	20.6
Vue.js	10.4

Tabela 6 – Média de latências após a alteração de parâmetro na *dashboard*, respetiva atualização no servidor e sequente representação na *dashboard* no ambiente de teste 2.

Framework	Latência cliente-servidor (ms)	Latência (ms)
Angular	11.8	19.2
React	8.5	27.1
Vue.js	7.6	15.6

3.6. Justificação sobre a escolha da *framework*

A *framework* de desenvolvimento *frontend* escolhida foi Angular, uma vez que se trata de uma *framework* completa e facilmente se consegue escalar o projeto que, neste trabalho, se prevê de longa duração e em contínua atualização e expansão durante os próximos anos, tornando-se cada vez mais complexo. De referir também que a equipa de desenvolvimento tem uma maior experiência com esta tecnologia *frontend*, o que elimina o tempo de aprendizagem.

No entanto, considerando os resultados, o projeto Vue.js apresentou um melhor desempenho na maioria dos parâmetros. Por isso, sugere-se um seguimento da evolução desta *framework* e pesquisa de vantagens e desvantagens face a Angular, com maior detalhe às especificidades do projeto.

Apesar de tudo, identificaram-se várias limitações aos testes que devem ser mencionadas. A principal prende-se com o facto de se tratar de uma simulação utilizando máquinas virtuais, as quais não replicam os constrangimentos a nível de rede e o desempenho de máquinas reais.

Ainda no âmbito dos ambientes de teste, importa referir a necessidade de proceder a estratégias alternativas, nomeadamente no ambiente de teste 2, que não são as mais adequadas num contexto real:

- A integração de *logs* provenientes de aplicações Spring Boot com ELK Stack é geralmente efetuada através da dependência Logstash Logback Encoder. Esta disponibiliza *appenders* que possibilitam o envio via TCP e agilizam a implementação do projeto Spring Boot. Contudo, para garantir uma maior similaridade entre projetos *backend* e *frontend* e evitar discrepâncias devido a diferentes protocolos de transporte, optou-se por uniformizar e recorrer a HTTP [112], [125];
- A utilização de ELK Stack está muitas vezes associada ao envio dos *logs* através de Beats, como é o caso da ferramenta Filebeat. Esta ferramenta é utilizada como um intermédio entre a aplicação que produz o *log* e a solução ELK Stack, reduzindo a necessidade de processamento por parte do Logstash. No entanto, dada a simplicidade do cenário, optou-se por não utilizar este *software* [110], [112];

- Uma solução mais sofisticada para contornar a limitação do mecanismo de segurança CORS consiste na utilização de um servidor NGINX como *proxy* reverso. Neste contexto, seria configurado um servidor NGINX na terceira VM para o qual seriam reencaminhados os pedidos HTTP; posteriormente, este servidor encaminharia os *logs* para a solução ELK Stack. Todavia, como se trata de um cenário simples e considerando a necessidade de evitar interferências que não ocorrerão no projeto final, optou-se pela desativação temporária deste mecanismo [126].

Para além de limitações a nível de ambiente de teste, é também importante referir que a *dashboard* do projeto é muito mais complexa que o cenário hipotético. Os dados enviados têm origem em equipamentos externo (OAIBOX™) e o sistema não tem uma arquitetura simples *backend-frontend* constituída por um único serviço Spring Boot utilizando apenas WebSocket e uma *dashboard*. Também o envio contínuo foi aqui efetuado numa frequência constante que poderá não ser observado numa situação real.

Para finalizar, espera-se que durante a evolução do projeto surjam novas funcionalidades não testadas, e a utilização de uma *framework* robusta, como o Angular, será uma mais-valia.

4. Modelação do sistema

Neste capítulo é efetuada a modelação do sistema seguindo a metodologia ICONIX esquematizada na Figura 33. Esta metodologia divide-se em quatro fases, com a utilização de diversos diagramas UML. Na primeira fase, Análise de Requisitos, procede-se ao levantamento de requisitos e identificação dos casos de uso, também é desenhado o modelo de domínio e *storyboards* que auxiliam a visualização das funcionalidades que se procuram. Na segunda fase, Análise e Projeto Preliminar, são descritos os casos de uso e desenhados os diagramas de robustez, enquanto o modelo de domínio é atualizado. Na terceira fase, Projeto, há um maior detalhe do projeto, recorrendo a diagramas de sequência que, mais uma vez, auxiliam no aprimoramento do modelo de domínio que se vai transformando no diagrama de classes. A quarta fase corresponde à Implementação, na qual são também desenhados os testes unitários e de integração [127].

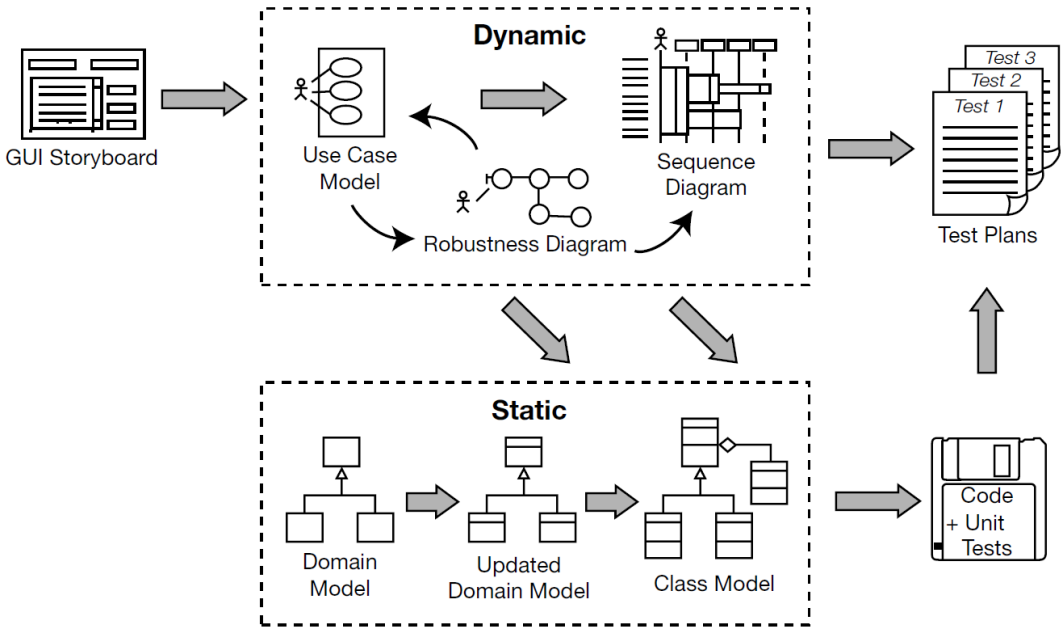


Figura 33 – Esquema do processo ICONIX.

Contudo, neste trabalho não são desenhados todos os diagramas, recorrendo-se apenas aos que se consideram mais indicados e com maior utilidade prática para o projeto. Assim, neste capítulo são apresentados os requisitos do sistema, o diagrama de casos de uso, a descrição de casos de uso, os diagramas de robustez, os diagramas de sequência e o diagrama de classes. De ressalvar que são apenas apresentados a descrição e os diagramas de robustez e de sequência de três casos de uso para efeitos demonstrativos.

4.1. Requisitos do sistema

O levantamento dos requisitos do sistema visa corresponder às necessidades do utilizador comum da aplicação, sendo a caracterização do público-alvo um ponto essencial neste processo. A *dashboard* a implementar neste trabalho destina-se à monitorização e configuração remota da estação base 5G OAIBOX™, um produto criado e desenvolvido pela empresa Allbesmart. Como tal, o público-alvo da aplicação são os utilizadores da OAIBOX™, principalmente indivíduos familiarizados com a utilização e configuração de redes 5G e dados e propriedades associados à respetiva configuração e monitorização. Estes utilizadores são sobretudo:

- Universidades: tanto a nível de ensino como de investigação;
- Instituições de investigação;
- Empresas e indústria ligada ao desenvolvimento de redes 5G [2].

Os utilizadores são, portanto, conhecedores dos dados e das informações relatados na *dashboard* e, apesar da necessidade de uma linguagem clara e concisa requerida em todas as aplicações, os textos, designações, siglas, acrónimos, ícones, símbolos, gráficos, entre outros serão empregues numa linguagem profissional e técnica.

Conhecendo o utilizador comum do produto e, consequentemente, da *dashboard*, os requisitos do sistema podem ser, então, enumerados. Geralmente, os requisitos do sistema são categorizados em Requisitos Não Funcionais e Requisitos Funcionais. Os primeiros especificam critérios de operação do sistema, por exemplo, a nível de desempenho; por outro lado, os requisitos funcionais incluem funcionalidades e comportamentos que o sistema deve efetuar e/ou apresentar [128].

Os requisitos não funcionais são os seguintes:

- Baixo tempo de resposta, com responsividade fluída em resultado das interações do utilizador;
- Baixa latência na receção de dados e respetiva atualização gráfica;
- Escalabilidade da aplicação aquando de um aumento de comunicação de dados e/ou utilização;
- Disponibilidade garantida 24/7;
- Segurança e autenticidade dos dados transmitidos entre a OAIBOX™ e a *dashboard*, incluindo autenticação e controlo de acesso dos diferentes tipos de utilizador;

- Fácil manutenção: dada a constante atualização do produto, a programação deverá seguir regras de boas práticas e a *framework* e outras bibliotecas utilizadas deverão facilitar a atualização e manutenção do projeto.

Dada a natureza da *dashboard*, os requisitos funcionais incidem sobre a monitorização e configuração da estação base 5G OAIBOX™. Estas funcionalidades podem estar disponíveis para todos os utilizadores e/ou tipos de OAIBOX™ ou serem apenas específicas para determinado tipo de utilizador e/ou modelo de produto. Considerando estes pontos, os requisitos funcionais são os seguintes:

- Criação de novo utilizador;
- Iniciar sessão e autenticar o utilizador;
- Terminar sessão;
- Seleção da OAIBOX™ a monitorizar/configurar;
- Aceder ao manual;
- *Download* de recursos para configuração, como *drivers* de Quectel;
- Sistema de alertas e notificações;
- Visualização de detalhes da OAIBOX™:
    - Tipo de OAIBOX™;
    - Componentes e respetivos modelos (*Software Defined Radio* – SDR);
    - Identificadores do cliente e da máquina;
    - IP;
    - Data e hora da última receção de dados;
- Monitorização da OAIBOX™:
    - Receção de dados:
        - Valores atuais reportados pela(s) OAIBOX™(es) e/ou componentes;
        - Estado atual da OAIBOX™ e componentes;
        - Estado de funções de rede;
        - Estado de funções *core*;
    - Visualização dos dados recebidos e representação de estados com cores representativas;
    - Visualização de gráficos com a evolução dos dados recebidos;

  - Visualização de *logs* dos componentes;
  - Visualização de mensagens de rede, nomeadamente inspeção de pacotes;
  - Execução de testes de rede, como teste de velocidade (OpenSpeedTest) e *iPerf test*;
  - Visualização de parâmetros, métricas e KPIs do gNB;
  - Visualização de UE ligados e respetivo estado;
  - Visualização de outras métricas e KPIs de UE;
- Configuração da OAIBOX™ (através de envio de *actions*):
  - Reiniciar OAIBOX™;
  - Ligar/desligar/reiniciar componentes;
  - Repor propriedades;
  - Alterar propriedades de rede;
  - Compilar código de componentes alterados pelo utilizador;
  - Configuração de parâmetros do gNB;
  - Outras definições avançadas.

Como se trata de um produto em constante desenvolvimento, melhoramento e expansão, é expetável a inclusão de novos requisitos que deverão ser incluídos na *dashboard* em momento oportuno.

4.2. Diagrama de caso de uso

Um caso de uso representa um cenário de utilização que cumpra requisitos que tenham sido definidos, demonstrando a interação entre o utilizador e o sistema. No diagrama de casos de uso são apresentados os casos de uso associados aos atores intervenientes e, possivelmente, a outros casos de uso a que estejam intimamente relacionados [127].

Os atores correspondem, geralmente, a utilizadores com um papel (*role*) específico e, sendo externos ao sistema, são desenhados como uma figura em boneco (*stick figure*) e colocados fora do retângulo que representa o sistema. Estes são associados aos casos de uso que podem executar e que se encontram no interior do sistema. Por vezes, há ligações entre casos de uso, como é o caso de relações *includes* (comportamento que é sempre executado, caso o principal seja executado) e *extends* (comportamento que pode ser executado, durante a execução do caso de uso principal) [127].

O diagrama de casos de uso está representado nas Figura 34, Figura 35, Figura 36 e Figura 37.

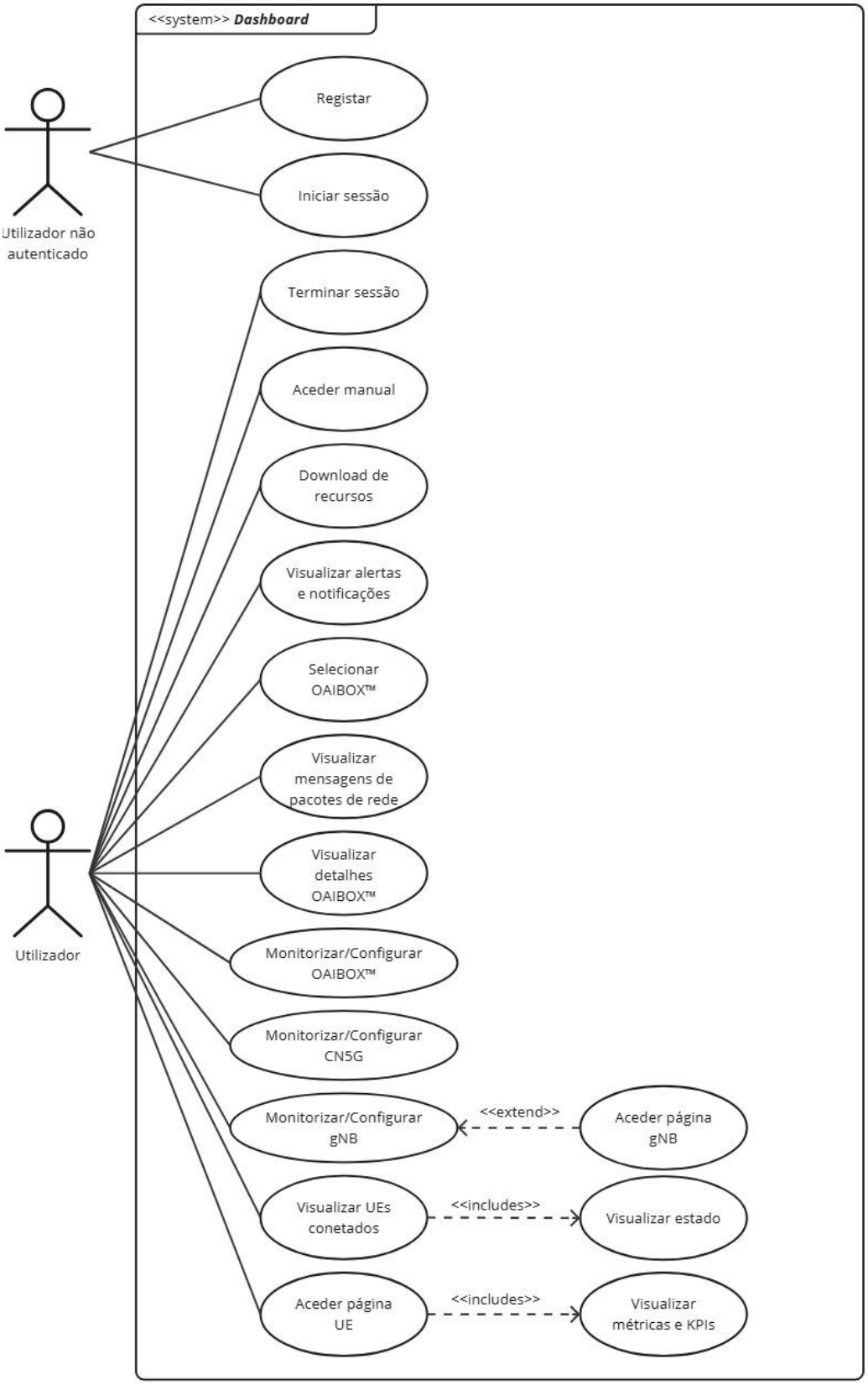


Figura 34 – Diagrama de casos de uso.

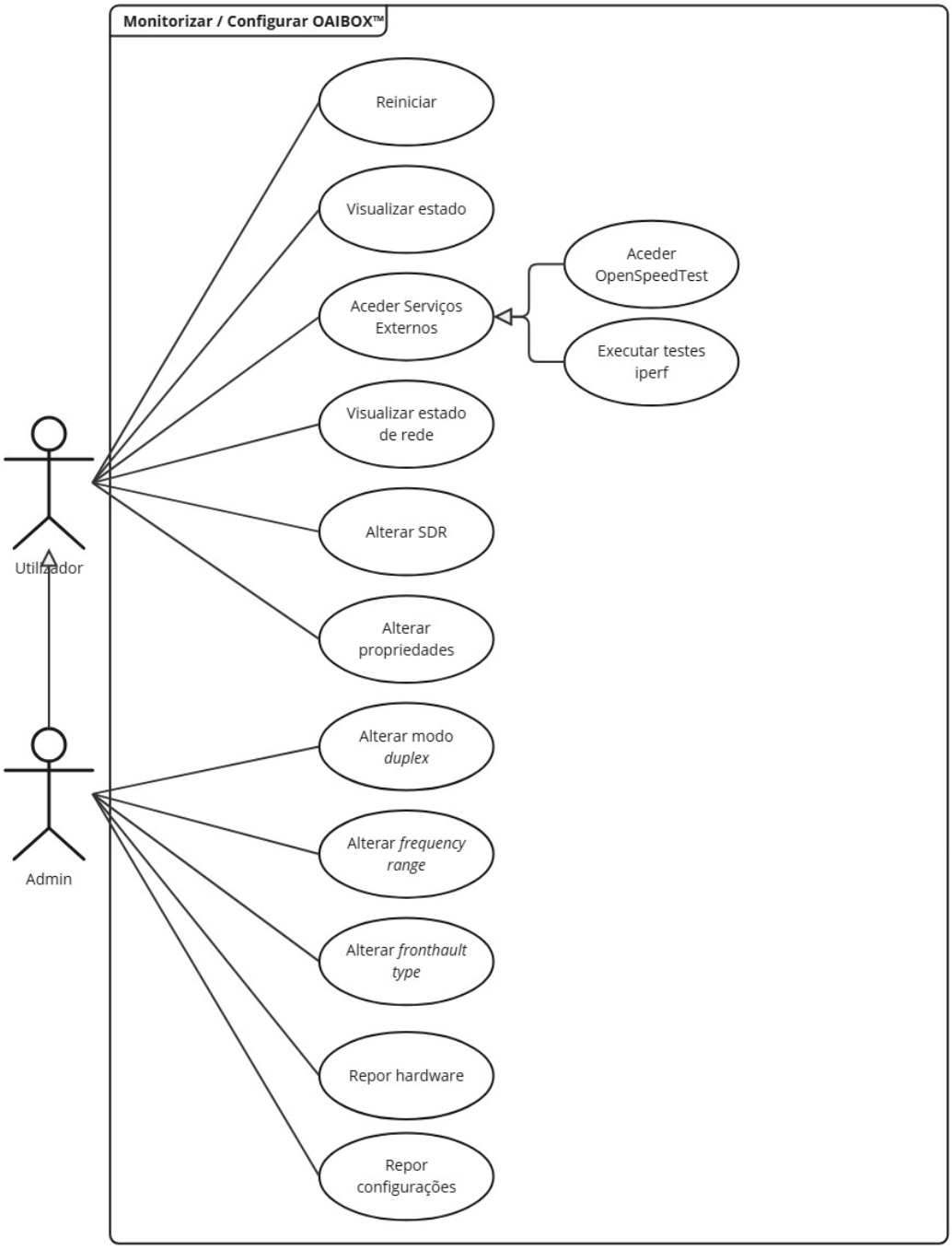


Figura 35 – Diagrama de casos de uso: funcionalidade Monitorizar/Configurar OAIBOX™.

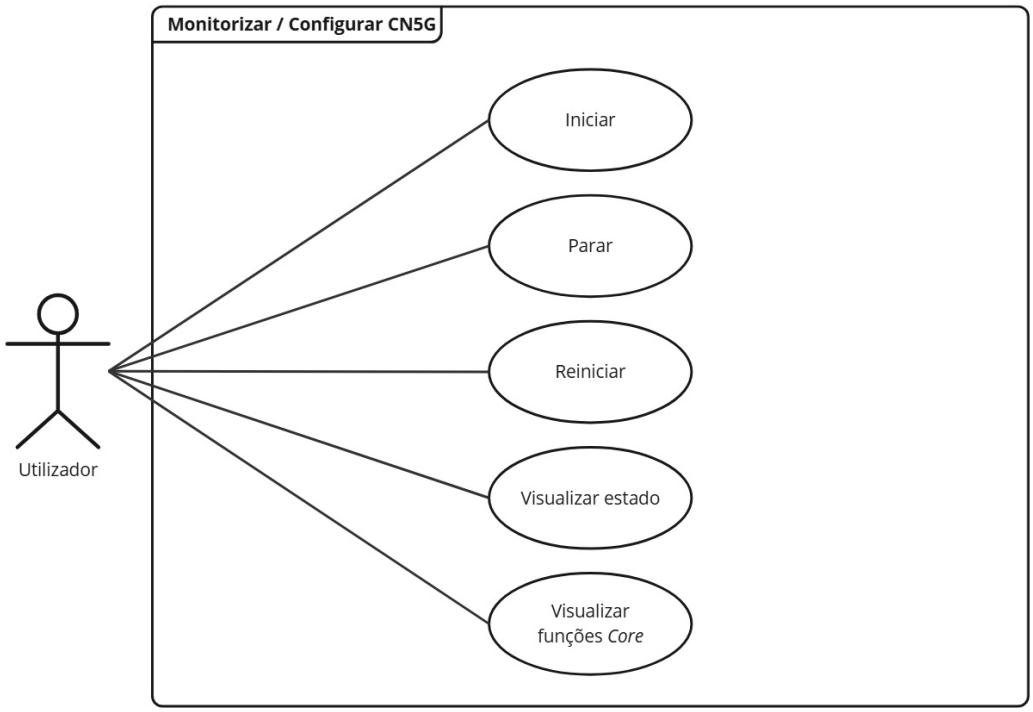


Figura 36 – Diagrama de casos de uso: funcionalidade Monitorizar/Configurar CN5G.

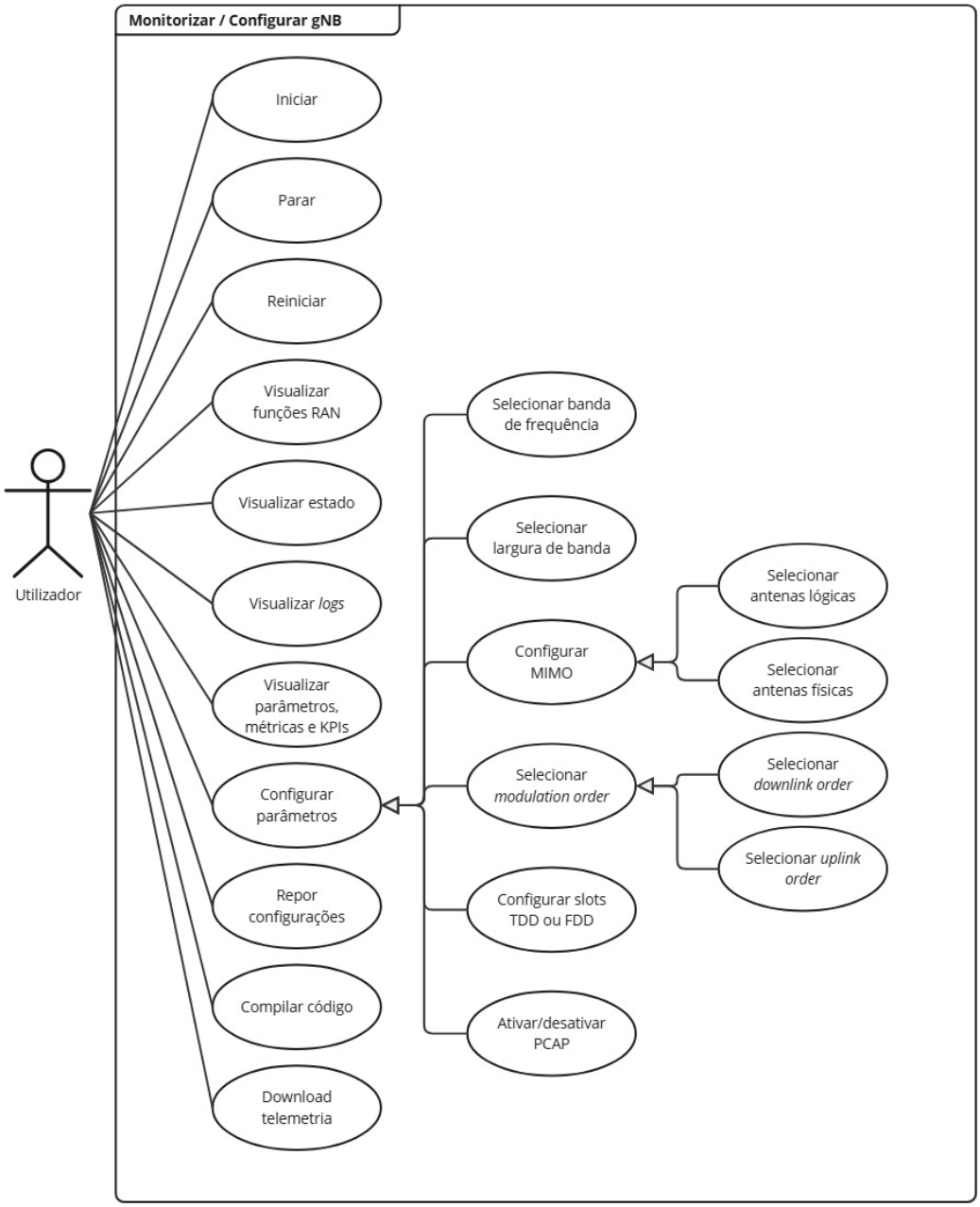


Figura 37 – Diagrama de casos de uso: funcionalidade Monitorizar/Configurar gNB.

4.3. Descrição de caso de uso

A descrição do caso de uso detalha o processo que deverá ser seguido para a concretização do cenário, especificando os atores envolvidos e os passos necessários para o seu sucesso e situações de erro, entre outros [127].

Para demonstração, foram selecionados os casos de uso “Selecionar OAIBOX™”, “Iniciar [gNB]” e “Visualizar métricas e KPIs [do UE]”, cuja descrição de se encontra em baixo.

Descrição detalhada – Exemplo 1

Título: Selecionar OAIBOX™;

Ator primário: Utilizador;

Stakeholders: Utilizador;

Precondição: Utilizador autenticado no sistema cuja organização possui uma ou mais OAIBOX™(es);

Garantia mínima: Sistema permanece na página *Overview* da OAIBOX™ predefinida;

Sucesso garantido: Sistema apresenta página *Overview* da OAIBOX™ selecionada;

Trigger: Utilizador seleciona o botão de seleção da OAIBOX™;

Cenário principal de sucesso:

1. Sistema apresenta lista de OAIBOX™(es) da organização do utilizador;
2. Utilizador seleciona a OAIBOX™ pretendida;
3. Sistema apresenta página *Overview* da OAIBOX™ selecionada;

Extensões:

1.a. Sistema não apresenta lista de OAIBOX™(es) da organização:

1.a.1. O utilizador clica em *refresh* e a lista é atualizada;

1.a.2. O utilizador clica em *refresh* e a lista não é atualizada; o utilizador contacta o suporte;

3.a. Sistema não redireciona para a página da OAIBOX™ selecionada:

3.a.1. O utilizador seleciona novamente a OAIBOX™ e a página é atualizada;

3.a.2. O utilizador seleciona novamente a OAIBOX™ e a página não é atualizada; o utilizador clica em *refresh* e tenta novamente.

Descrição detalhada – Exemplo 2

Título: Iniciar [gNB]

Ator primário: Utilizador;

Stakeholders: Utilizador;

Precondição: Utilizador autenticado no sistema na página de *Overview* ou na página do gNB de uma OAIBOX™ cujo *Core Network* 5G (CN5G) e respetivas funções *core* se encontram no estado *Running*;

Garantia mínima: Sistema apresenta mensagem de erro;

Sucesso garantido: Estado do gNB passa a *Running*;

Trigger: Utilizador seleciona o botão *Start* do gNB, após verificar que CN5G e funções *core* estão no estado *Running*;

Cenário principal de sucesso:

1. Sistema apresenta *modal* para confirmar a ação;
2. Utilizador seleciona o botão de confirmação;
3. Sistema envia *action* para espoletar a ação na OAIBOX™;
4. Sistema apresenta a notificação de que o gNB está a iniciar e a *dashboard* atualiza a apresentação do estado nos locais necessários;
5. Sistema apresenta a notificação de que o gNB está *Running* e a *dashboard* atualiza a apresentação do estado nos locais necessários;

Extensões:

1.a. Modal não é apresentado:

1.a.1. O utilizador seleciona novamente o botão *Start* do gNB e o modal é apresentado;

1.a.2. O utilizador seleciona novamente o botão *Start* do gNB e o modal não é apresentado; o utilizador clica em *refresh* e tenta novamente;

3,4,5.a. OAIBOX™ não altera estado:

3,4,5.a.1 O utilizador confirma que a OAIBOX™, CN5G e funções *core* estão no estado *Running* e tenta novamente;

3,4,5.a.2 O utilizador clica em *refresh* e tenta novamente;

Descrição detalhada – Exemplo 3

Título: Visualizar métricas e KPIs [do UE]

Ator primário: Utilizador;

Stakeholders: Utilizador;

Precondição: Utilizador autenticado no sistema na página de *Overview* ou na página do gNB de uma OAIBOX™ com CN5G e gNB no estado *Running* e UE(s) conectado(s);

Garantia mínima: Sistema permanece na página inicial;

Sucesso garantido: Sistema apresenta gráficos de métricas e KPIs na página correspondente ao UE selecionada;

Trigger: Utilizador seleciona o UE pretendido;

Cenário principal de sucesso:

1. Sistema direciona o utilizador para a página de visualização de gráficos com as métricas e KPIs do UE;

Extensões:

1.a. Erro no redireccionamento e sistema apresenta mensagem de erro:

1.a.1. O utilizador clica em *refresh* e a página é carregada;

1.a.2. O utilizador clica em *refresh* e a página não é carregada; o utilizador retorna à página anterior e tenta novamente.

4.4. Diagramas de robustez

Um diagrama de robustez corresponde a um caso de uso e associa os passos do caso de uso a objetos (classes) e respetivos atributos a implementar, estabelecendo uma ponte entre os casos de uso e os diagramas de sequência e permitindo, principalmente, um *design* de programação orientada a objetos a partir do caso de uso. É, assim, uma representação pictorial do comportamento descrito no caso de uso, mostrando os objetos (classes) participantes, sem, todavia, associar as respetivas responsabilidades [127].

Nos diagramas de robustez são desenhados os atores, que interagem e/ou espoletam o caso de uso, os objetos de fronteira, que correspondem a interfaces como ecrãs e páginas *web*, objetos entidade, que representam dados persistentes do sistema ou classes, e controladores, que ligam os objetos entre si [127].

Para demonstração, foram selecionados os casos de uso “Selecionar OAIBOX™”, “Iniciar [gNB]” e “Visualizar métricas e KPIs [do UE]”, cujos diagramas de robustez se encontram nas Figura 38, Figura 39 e Figura 40.

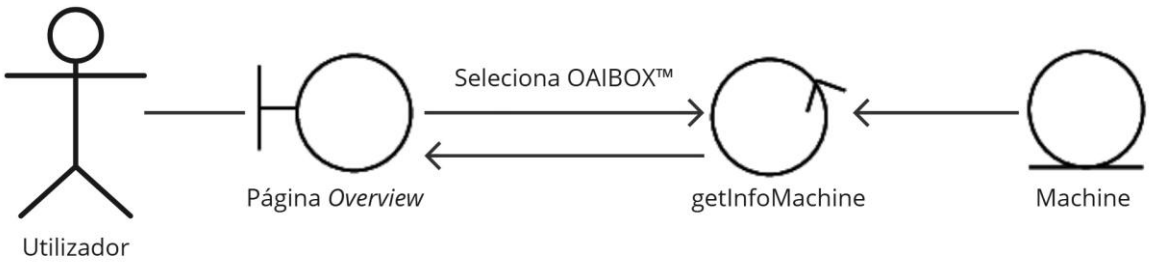


Figura 38 – Diagrama de robustez do caso de uso “Selecionar OAIBOX™”.

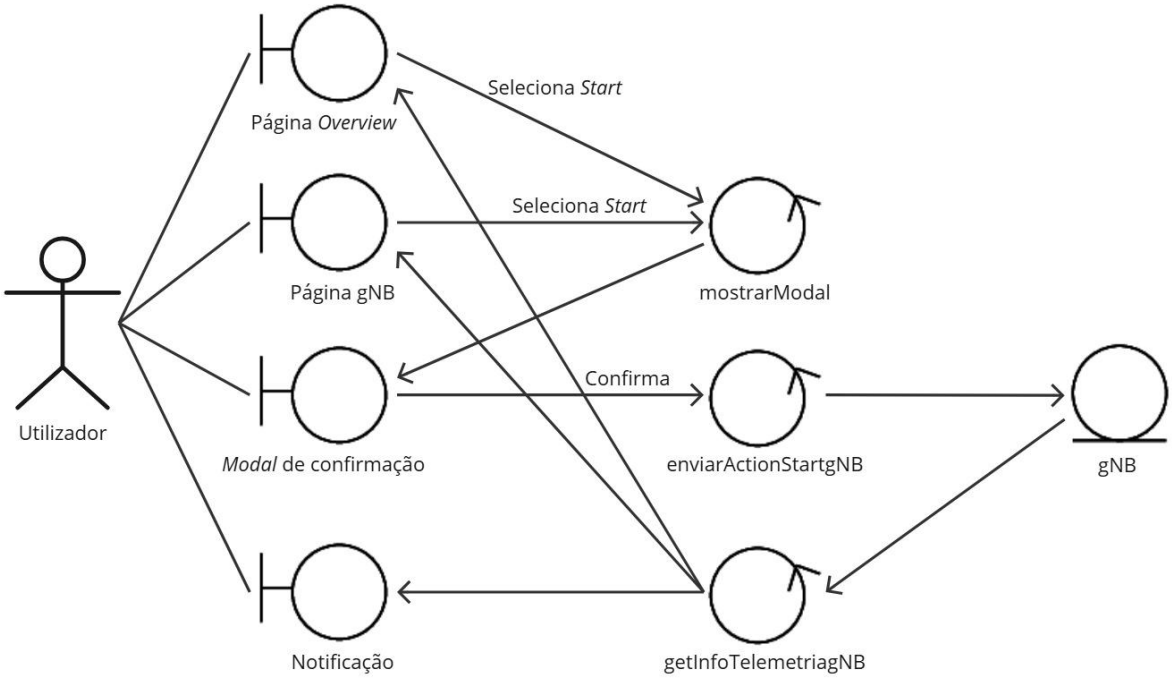


Figura 39 – Diagrama de robustez do caso de uso “Iniciar [gNB]”.

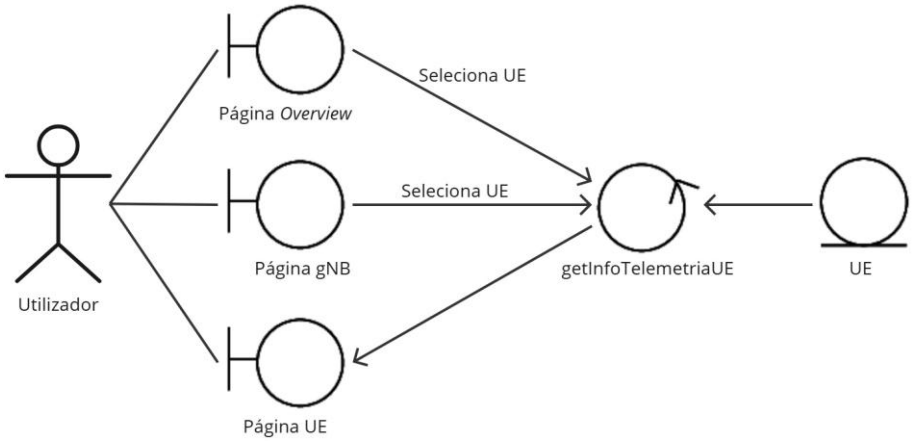


Figura 40 – Diagrama de robustez do caso de uso “Visualizar métricas e KPIs [do UE]”.

4.5. Diagramas de sequência

Um diagrama de sequência corresponde a um caso de uso e pretende demonstrar em detalhe o comportamento do sistema e como esse caso de uso será implementado. Estes diagramas permitem uma exploração detalhada do sistema, facilitando a compreensão do comportamento do mesmo, principalmente em programação orientada a objetos [127].

Nos diagramas de sequência são desenhados os atores, que iniciam e/ou interagem no caso de uso, os objetos de fronteira e entidade, que foram

identificados nos diagramas de robustez, linhas de vida verticais, que indicam o tempo em que os objetos existem e/ou estão ativos, e linhas horizontais direcionais, que representam as mensagens trocadas entre cada um dos intervenientes no caso de uso (atores e objetos) e cujo desenho detalha a ordem temporal das mesmas [127].

Para demonstração, foram selecionados os casos de uso “Selecionar OAIBOX™”, “Iniciar [gNB]” e “Visualizar métricas e KPIs [do UE]”, cujos diagramas de sequência se encontram nas Figura 41, Figura 42 e Figura 43.

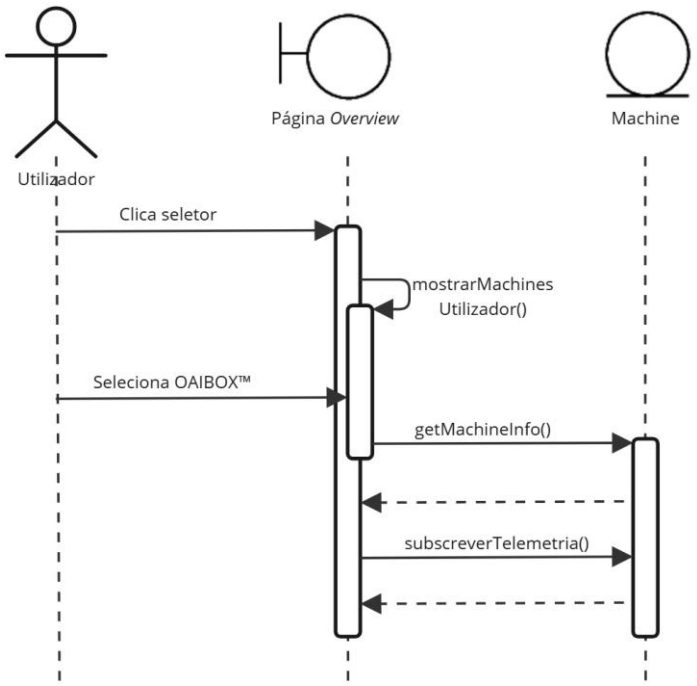


Figura 41 – Diagrama de sequência do caso de uso “Selecionar OAIBOX™”.

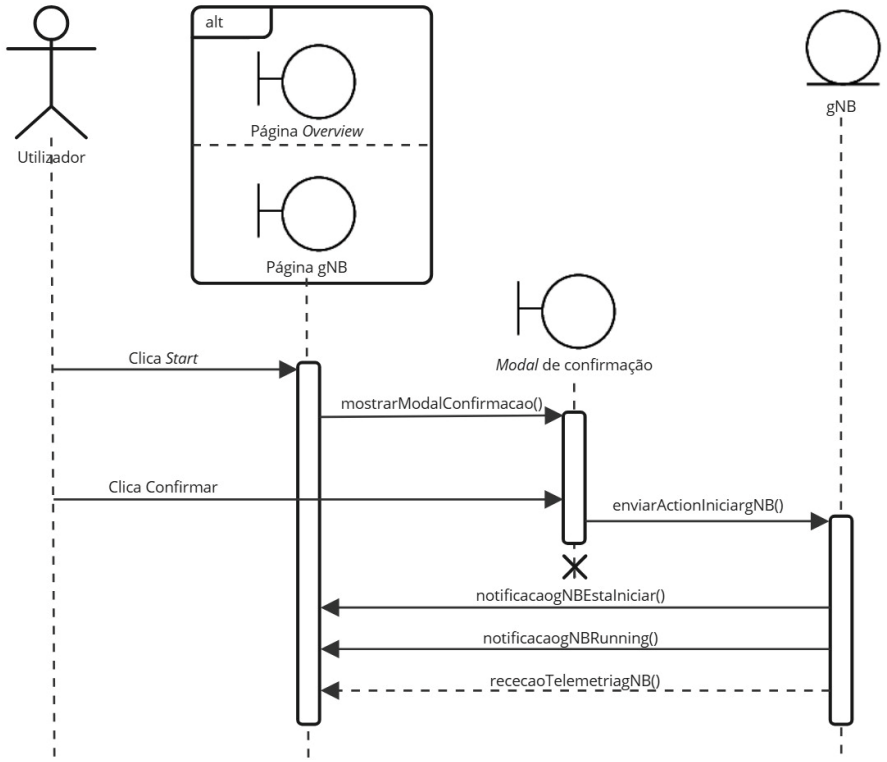


Figura 42 – Diagrama de sequência do caso de uso “Iniciar [gNB]”.

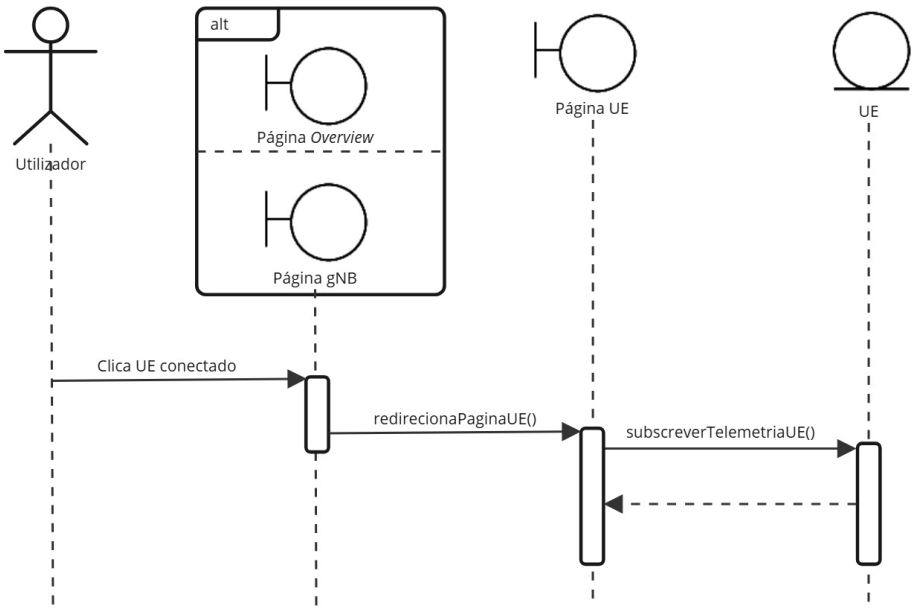


Figura 43 – Diagrama de sequência do caso de uso “Visualizar métricas e KPIs [do UE]”.

4.6. Diagrama de classes

Em ICONIX, o diagrama de classes resulta de uma evolução iterativa ao longo das fases da metodologia, sendo progressivamente refinado a partir do modelo de

domínio. Este diagrama é constituído pelas classes, e respetivos atributos e métodos, com ligações entre elas refletindo as relações entre elas [127].

O ICONIX está fortemente relacionado com a programação orientada a objetos, no entanto, este paradigma não é tão relevante no desenvolvimento *web* moderno, incluindo em *frameworks* como Angular, cuja arquitetura é baseada em componentes, sendo os diagramas de sequência mais relevantes na implementação [127], [129]. Assim, o diagrama de classes da Figura 44 representa sobretudo o modelo de dados a implementar que não é, contudo, o foco principal do projeto.

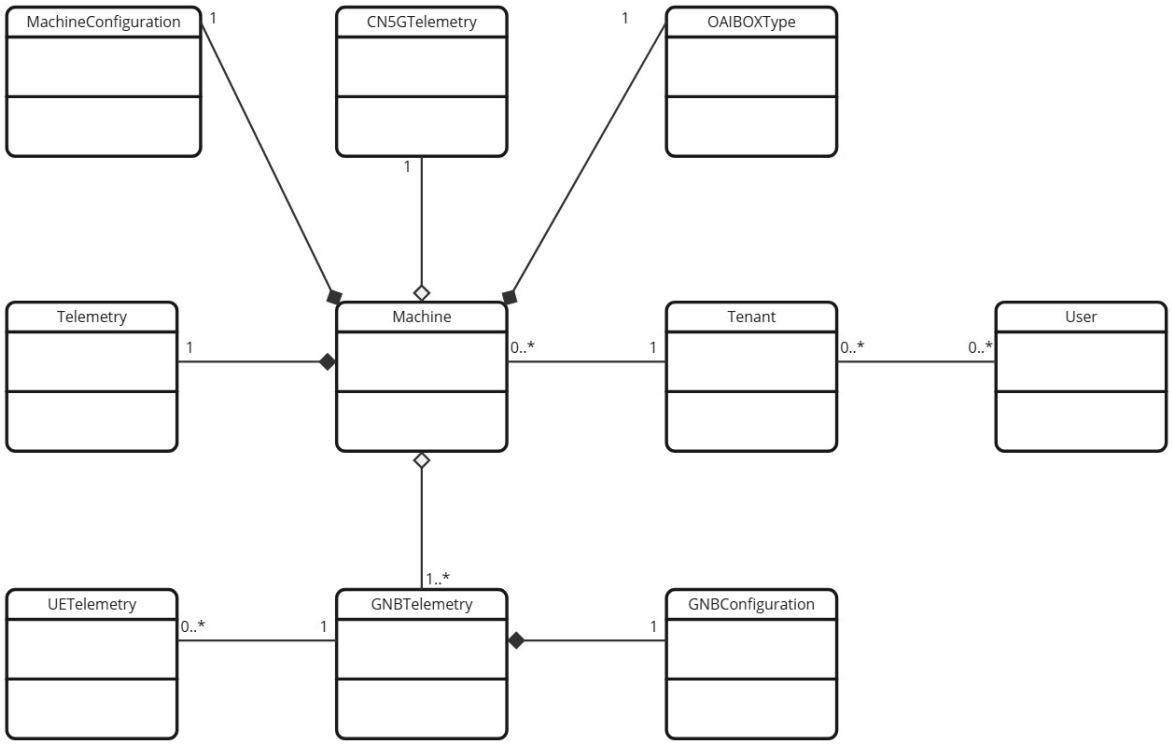


Figura 44 – Diagrama de classes**.**

5. Desenvolvimento da *dashboard*

Neste capítulo é efetuada a descrição da *dashboard* implementada, com foco nas funcionalidades desenvolvidas e um breve resumo da arquitetura geral em que se insere este projeto.

5.1. Arquitetura do sistema

A *dashboard* implementada insere-se num projeto cuja arquitetura geral se encontra representada na Figura 45. O sistema pretende monitorizar e configurar a OAIBOX™ construída e distribuída pela empresa Allbesmart. Na figura há uma reprodução da solução OAIBOX™ MAX, no entanto, a Allbesmart oferece uma maior variedade de soluções [2] que, apesar das particularidades inerentes a cada uma, o descrito neste trabalho é comum a todas.

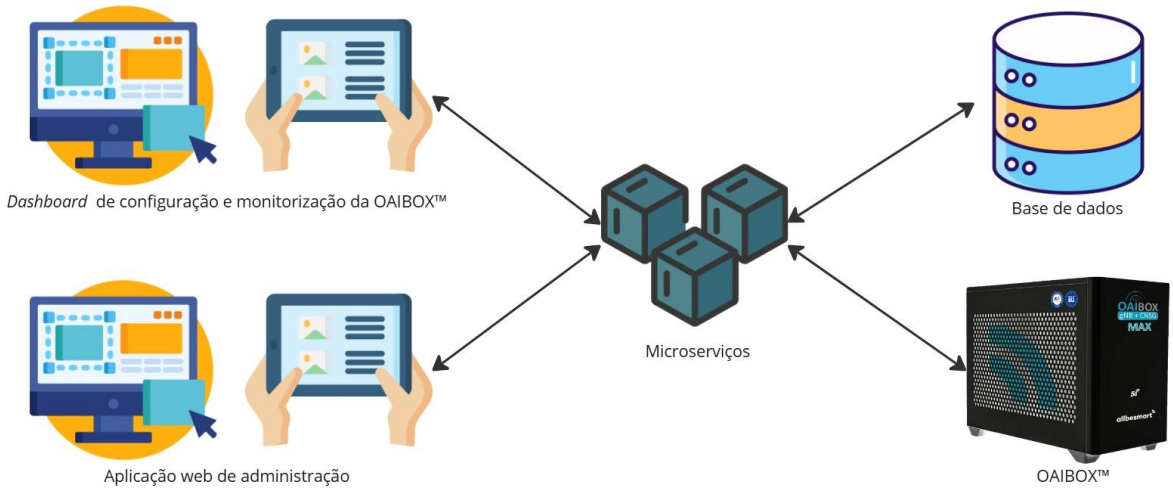


Figura 45 – Arquitetura do sistema.

A comunicação entre os diferentes elementos do sistema é efetuada através de microserviços, que constituem aplicações independentes com um propósito definido: por exemplo, um microserviço responsável pela comunicação com a OAIBOX™ e um microserviço encarregue da gestão de *tenants*. No que diz respeito à *dashboard* de configuração e monitorização, o foco desta dissertação, a comunicação com o *backend* é efetuada através de WebSockets, para, por exemplo, receção de telemetria, e pedidos HTTP para, por exemplo, gestão de *tenants*.

Adicionalmente, foi implementada uma outra aplicação *web* com um propósito administrativo, que não é alvo desta dissertação.

De referir também que o acesso às aplicações *web* é efetuado via HTTPS e a segurança e autenticação das comunicações são asseguradas através da utilização do *software* Keycloak. Este é um produto *open source* que permite a autenticação e a segurança de serviços e aplicações, encarregando-se de toda a gestão dos utilizadores. Providencia suporte a protocolos como OpenID Connect, OAuth 2.0 e SAML, e disponibiliza também uma consola de administração onde possibilita a configuração das políticas de autorização de um dado projeto [130].

5.2. Implementação da *dashboard*

O objetivo principal deste trabalho é a implementação de uma *dashboard* para monitorização e configuração remota de uma estação base 5G, especificamente OAIBOX™, um produto da empresa Allbesmart. O projeto é fruto do trabalho da equipa *web* da empresa, sendo o *frontend* implementado o tema essencial deste trabalho. Sendo um trabalho de equipa, é importante clarificar que a minha contribuição no projeto correspondeu a um papel algo secundário, de manutenção do produto. O *backend* que sustenta a *dashboard* é igualmente um produto da empresa, contudo, a sua análise não constitui o foco desta dissertação.

Como referido anteriormente, a aplicação *frontend* foi desenvolvida com recurso à *framework* Angular. De mencionar também a utilização da biblioteca Apache ECharts nos componentes para desenho de gráficos, as tecnologias WebSocket e pedidos HTTP na comunicação entre a *dashboard* e o *backend*, e o recurso à *framework frontend* Bootstrap que agiliza a utilização de CSS e HTML e, consequentemente, a construção de interfaces [131].

A *dashboard* é acedida através do URL https://my.oaibox.com. Caso o utilizador não esteja autenticado, é mostrada a página de início de sessão representada na Figura 46. Através desta página é possível aceder à página de registo de utilizador. Na eventualidade de ter sessão iniciada ou após iniciar sessão, é apresenta a página *Overview* exemplificada na Figura 47.

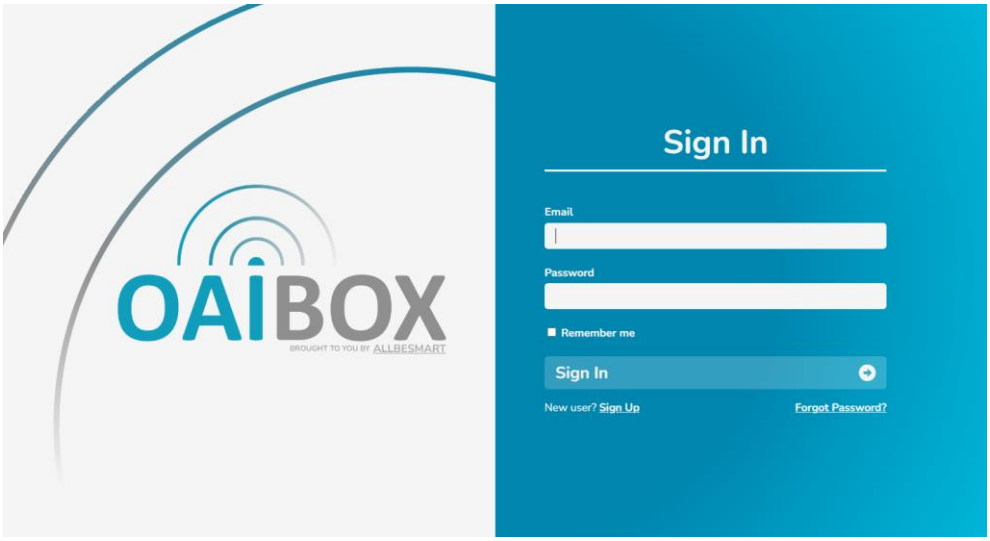


Figura 46 – Página de início de sessão.

A página *Overview* representa a página inicial e principal da *dashboard*. No topo encontra-se a barra de navegação e em baixo a página *Overview* propriamente dita. Esta última pode ser dividida em três partes distintas: à esquerda uma secção com os detalhes da OAIBOX™ e seus componentes, bem como o acesso às respetivas configurações, no centro uma zona com as funções *core* e outros elementos de rede interligados num esquema em árvore, e à direita uma área de visualização de mensagens de rede e inspeção de pacotes. O exemplo da Figura 47 é uma reprodução da página *Overview* de uma OAIBOX™ 40 *Online*, com o respetivo *core* e gNB no estado *Running* e um UE ligado. As *dashboard* de outras soluções OAIBOX™ poderão apresentar elementos diferentes do que aqui apresentados, devido à natureza do próprio *hardware*.

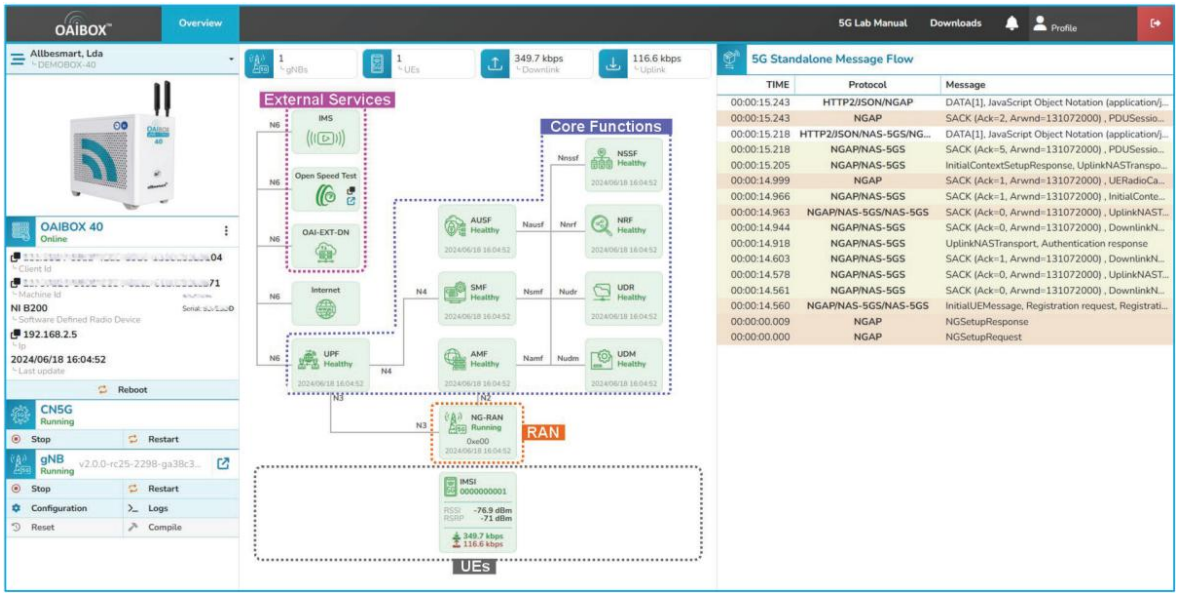


Figura 47 – Página *Overview*.

A barra de navegação no topo da página é um componente presente em toda a aplicação, com detalhes particulares dependendo da rota e página em questão: por exemplo, na Figura 47, o botão *Overview* está destacado, indicado o ponto de navegação em que o utilizador se encontra. À direita, há hiperligações para aceder às páginas de *download* do manual da OAIBOX™ e de *download* de outros recursos, como *drivers* para configuração, ícone de alertas e notificações enviadas pelo sistema, identificação de utilizador e botão para terminar sessão.

Na parte superior da secção à esquerda, encontra-se o seletor de OAIBOX™: quando o utilizador pode monitorizar e configurar mais de uma máquina, ao clicar neste botão aparecerá uma lista com todas as OAIBOX™es que poderão ser selecionadas. Uma vez selecionada, o utilizador é redirecionado para a página *Overview* da respetiva OAIBOX™.

Abaixo deste seletor, há uma representação do modelo da OAIBOX™, que no exemplo da Figura 47 é uma OAIBOX™ 40, e abaixo desta existem três secções: uma para a OAIBOX™, outra para o CN5G e uma última respeitante ao gNB. Relativamente à área da OAIBOX™, são listados alguns detalhes da máquina em questão, nomeadamente o seu estado (*Online* ou *Offline*), a identificação do cliente (*tenant*), a identificação da máquina, SDR em utilização (e possível alteração quando aplicável), endereço local IP da máquina e data e hora da última comunicação da OAIBOX™. Também aqui se encontra o botão para reiniciar a máquina e um botão para abrir o menu de configurações avançadas (Figura 48). Estas definições avançadas poderão não estar disponíveis para todos os utilizadores e/ou modelos e incluem as opções de repor configurações, modificar o modo *duplex*, o *fronthault type* e *frequency range*, alterar propriedades, acesso a uma linha de comandos, entre outros.

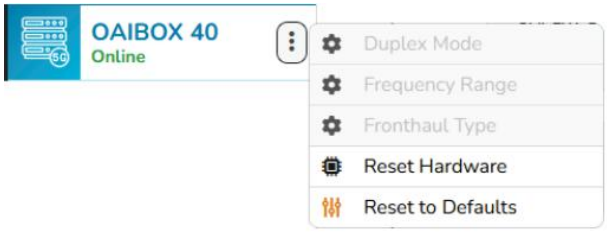


Figura 48 – Exemplo do menu de definições avançadas da OAIBOX™ 40.

Na secção respeitante ao CN5G, encontra-se o estado do componente (*Stopped*, *Starting*, *Running* ou *Upgrading*) e um botão de *Start*, caso o *core* esteja parado, ou os botões *Stop* e *Restart*, quando o *core* está iniciado, como no exemplo apresentado.

Na área referente ao gNB é indicado o seu estado (*Stopped*, *Starting*, *Running*, *Upgrading* ou *Compiling*), o seu código de lançamento e uma hiperligação para a

página dedicada à monitorização e configuração do gNB (Figura 54) que será detalhada posteriormente. Tal como descrito no componente anterior, aqui também se disponibiliza um botão de *Start*, se o gNB estiver parado, ou os botões *Stop* e *Restart*, se o gNB estiver iniciado, como no exemplo. Adicionalmente, são incluídos um botão *Configuration*, para alterar a configuração de início do gNB, um botão *Logs*, para visualização de *logs* quando o gNB está iniciado ou em ações de compilação ou atualização, um botão *Reset*, que apenas repõe as configurações referentes ao gNB, e um botão *Compile*, que compila o código do gNB modificado pelo utilizador. Estes dois últimos estão apenas disponíveis caso o gNB esteja no estado *Stopped*.

Os diferentes botões mencionados até agora, com exceção da hiperligação para a página dedicada ao gNB, abrem *modals* (elemento que é mostrado à frente da página, mantendo-se esta visível) onde as visualizações e/ou configurações são efetuadas. Para além disso, todas as operações carecem de confirmação, como é o caso dos processos de iniciar, reiniciar e parar a máquina ou os seus componentes. Nestas situações, também é apresentado um modal como o exemplo da Figura 49. Somente após confirmação, é enviada à OAIBOX™ a respetiva *action* – comando com uma estrutura definida e conhecida pelos diferentes constituintes do sistema, que consiste nas instruções a executar pela máquina. De referir ainda que as cores dos *modals* diferem de acordo com a natureza da tarefa.

Figura 49 – *Modal* de confirmação de paragem do gNB.

Na parte superior da zona central da página *Overview*, são mostrados alguns indicadores de estado em tempo real, especificamente o número de gNBs conectados ao *core* da OAIBOX™, o número de UEs que por sua vez estão conectados ao(s) gNB(s) e aos valores em tempo real dos débitos agregados de *downlink* e *uplink*.

Contudo, o foco desta secção é dominado pela estrutura em árvore que representa em tempo real o estado de rede dos diferentes constituintes. Esta estrutura é composta por diferentes blocos que fornecem uma visão geral e rápida do estado do sistema. De um modo genérico, estes blocos apresentam o estado, a data e hora da última telemetria recebida e, no caso de UE, um pequeno sumário de KPIs selecionados. Tal como destacado no exemplo da Figura 47, pode-se agrupar os diferentes blocos em quatro áreas: acesso a serviços externos, estado das funções *core*, estado da RAN (que inclui o gNB) e a lista de

UEs ligados (se aplicável). Há ainda um bloco referente ao estado da conexão à Internet.

Ao posicionar o cursor sobre estes blocos é apresentado um balão de informação (*tooltips*) com detalhes e/ou a nomenclatura do que está a ser representado. Por exemplo, na Figura 50 está representado o bloco de conexão à Internet, cujos detalhes em tempo real são apresentados no balão de informação. Há também outros blocos mais interativos, como por exemplo o bloco respeitante ao serviço externo de teste de velocidade (OpenSpeedTest). Como demonstrado na Figura 51, neste bloco existem dois botões: um para copiar a hiperligação para o teste e outro para abrir a página de teste numa aba em separado que, neste exemplo, é assinalada pelo balão informativo.

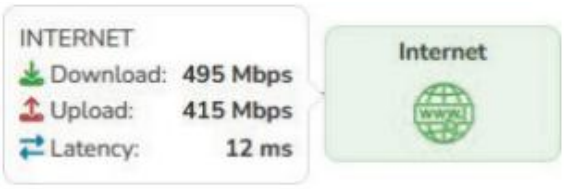


Figura 50 – Bloco de informações da conexão à Internet.

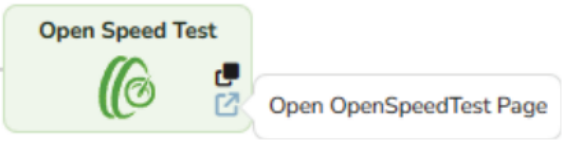


Figura 51 – Bloco de acesso à página do OpenSpeedTest.

Uma outra funcionalidade acedida através destes blocos é a execução de testes *iPerf*. Ao clicar no bloco OAI-EXT-DN a *dashboard* abre um *modal* que permite configurar e executar estes testes. Como se constata na Figura 52, é possível realizar testes de *uplink* (em que o UE funciona como cliente e envia dados para a máquina, o recetor) ou *downlink* (processo inverso, em que é a OAIBOX™ a responsável pelo envio de dados). Os campos com os dados para executar os testes estão pré-configurados, no entanto, o utilizador pode efetuar as customizações pertinentes de acordo com o seu cenário. De ressalvar que no exemplo da Figura 52 há um teste *downlink* em progresso que pode ser terminado. Esta secção representa uma lista e, caso ainda não haja qualquer teste executado, o elemento não é visível.

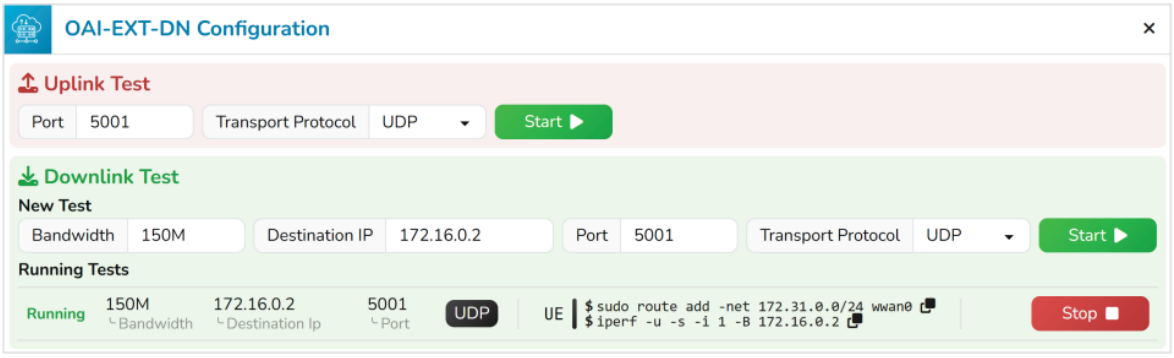


Figura 52 – *Modal* de configuração de testes *iPerf* com um teste em curso.

É importante referir que os blocos NG-RAN e os blocos referentes ao(s) UE(s) são clicáveis e redirecionam o utilizador para as páginas dedicadas à monitorização e configuração do gNB e à página de visualização de métricas e KPIs do UE, respetivamente. Estas páginas serão detalhas mais adiante.

Para finalizar a descrição deste esquema em árvore, é essencial referir que a cor dos blocos varia consoante o estado dos componentes a que está associado e que está em constante atualização refletindo a telemetria recebida na *dashboard*. Daí que tenha sido vincado a relevância deste elemento na monitorização do panorama geral da OAIBOX™ num dado momento. No exemplo da Figura 47, a OAIBOX™ está *Online*, todos os seus componentes encontram-se no estado *Running* e o UE está ativamente conectado, e como tal, todos os blocos estão desenhados a verde. No entanto, caso a OAIBOX™ esteja *Offline* ou algum dos componentes no estado *Stopped*, o bloco é representado a vermelho, como exemplificado na Figura 53, e representado a laranja caso se apresente num estado *Starting*. Estas cores são também utilizadas nas descrições de estado presentes nas secções à esquerda da *dashboard* explanadas anteriormente; no caso de estados *Upgrading* e *Compiling*, o estado é apresentado a azul.

Figura 53 – Exemplo de bloco no estado *Stopped*.

Para terminar a exposição da página *Overview*, a área à direita é dedicada à monitorização de mensagens de rede e de inspeção de pacotes. Estes dados são obtidos através do *software* Wireshark embutido na OAIBOX™ e apresentados apenas a data e hora, o protocolo e um resumo do conteúdo da mensagem.

Como referido anteriormente, o acesso à página de monitorização e configuração do gNB é efetuado através do botão na secção do gNB à esquerda na página de *Overview* da dashboard ou através do bloco NG-RAN do esquema em árvore que se encontra na mesma página. A Figura 54 exemplifica esta página, na qual se constata que o componente da barra de navegação no topo é o mesmo do anteriormente descrito. De realçar a zona da esquerda da barra, onde é indicado o ponto na rota de navegação em que o utilizador se encontra: aqui é destacado a azul o gNB e o botão *Overview* é uma hiperligação para a página correspondente.

Figura 54 – Página de monitorização e configuração do gNB.

À esquerda da página dedicada ao gNB, há um menu expansível/contrátil (no exemplo da Figura 54 está expandido) no qual se encontra um botão para *download* da telemetria. O ficheiro descarregado está em formato JSON e inclui diversos KPIs que poderão auxiliar o utilizador a ter uma melhor perceção do desempenho, detetar e resolver problemas e otimizar a eficiência da rede.

Na zona principal da página do gNB são apresentados no topo diversos indicadores atualizados em tempo real relacionados com o mesmo, nomeadamente, o estado e o identificador, o nome, os débitos de *downlink* e *uplink*, o número de UEs conectadas e a data e hora da última telemetria recebida na *dashboard*. O estado do gNB e respetiva cor do ícone e do texto aqui mostrados são similares ao anteriormente especificado.

Imediatamente abaixo há duas secções distintas: à esquerda dedicada à configuração de diversos parâmetros do gNB e à direita com detalhes referentes ao mesmo. Este segundo são dados estáticos, não editáveis, e refletem a configuração do gNB num dado momento.

Relativamente à configuração do gNB nesta página, constata-se que as operações acessíveis na página *Overview* também estão aqui disponíveis: os botões para iniciar, reiniciar e parar o gNB, o botão para repor definições, o botão para compilar o código modificado pelo utilizador e o botão de visualização de *logs*. Mais uma vez, a disponibilidade dos botões depende do estado do gNB: o botão *Start* aparece quando o gNB está parado e os botões *Restart* e *Stop* quando está no estado *Running*. Adicionalmente, há o botão para ativar/desativar PCAP, o seletor de banda de frequência, o seletor de largura de banda, os seletores da configuração MIMO (configuração das antenas lógicas e físicas), os seletores da *modulation order* (para *downlink* e *uplink*) e o seletor de *slots Time Division Duplex* – TDD – (ou *Frequency Division Duplex* – FDD –, dependendo do modelo de OAIBOX™). Ao clicar nestes seletores, é apresentado uma *dropdown* que lista as opções disponíveis que dependem do tipo de OAIBOX™ e que também podem ser configuradas especificamente para uma máquina. Estas operações de personalização de configurações são efetuadas na aplicação de administração de suporte à *dashboard* que não é o alvo desta dissertação. É ainda importante salientar que todos os botões e seletores de configuração apenas estão editáveis quando o gNB está parado.

Abaixo destes elementos, é desenhado o gráfico de *bitrate* no qual é possível analisar os débitos agregados de *downlink* e *uplink* dos últimos 10 minutos. Como demonstrado na Figura 55, ao posicionar o cursor sobre o gráfico, é apresentado um balão de informação com os valores do ponto em cima do qual o cursor se encontra. Como se trata de um gráfico de linhas múltiplas (especificamente duas linhas correspondendo a duas variáveis), este balão de informação indica a data e hora do ponto (eixo horizontal) e os valores de débitos agregados de *downlink* e *uplink* (eixo vertical) incluindo a unidade de medida e sinalização da cor do traço correspondente.

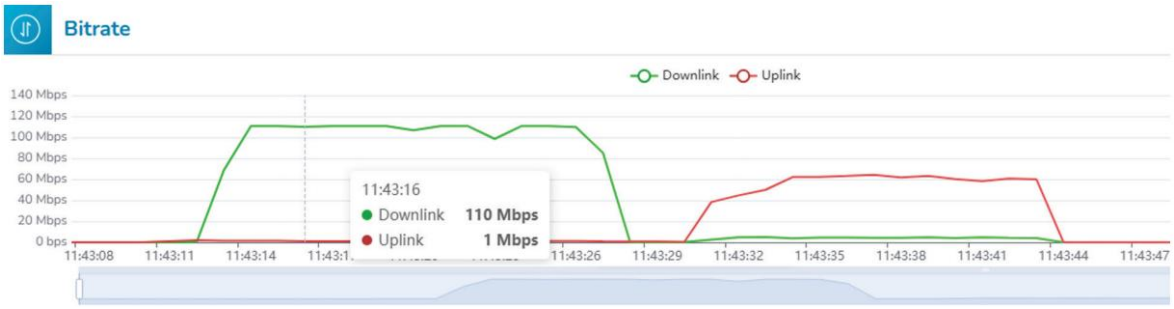


Figura 55 – Exemplo de gráfico, neste caso, referente ao *Bitrate.*

No fundo da página, caso haja UEs ligados ao gNB, estes são listados em cartões individuais. Estes componentes constituem um pequeno resumo das métricas e KPIs do UE em tempo real e são clicáveis, providenciando

hiperligações que redirecionam o utilizador para uma página dedicada à monitorização das métricas e KPIs do UE.

A Figura 56 exemplifica a página de monitorização de um UE que, como especificado anteriormente, é acedida ao clicar no bloco do UE no esquema em árvore na página *Overview* ou no cartão deste na página do gNB a que o UE está conectado. Ao contrário das outras páginas, esta é exclusivamente para visualização e monitorização; apresenta gráficos que exibem métricas e KPIs em tempo real e o histórico desde o acesso à página, atualizando-se constantemente com a telemetria recebida na *dashboard*.

Figura 56 – Página de monitorização de um UE.

Mais uma vez, o componente da barra de navegação está presente no topo da página, indicando a rota de navegação atual com destaque para o UE. Os botões *Overview* e gNB na barra de navegação redirecionam o utilizador,

respetivamente, para a página *Overview* da OAIBOX™ e para a página dedicada ao gNB ao qual o UE está ligado.

A parte superior da página exibe num pequeno resumo dos dados do UE, incluindo a data e hora da última telemetria recebida na *dashboard*, recorrendo a gráficos *gauge*, uma vez que apenas é apresentado um valor a dado momento. De seguida, são apresentados os gráficos para as diferentes métricas e KPIs, sendo estes essencialmente gráficos de linhas, podendo ser múltiplos em conformidade com a natureza da métrica ou KPI a apresentar. No fundo da página são disponibilizados dois gráficos que permitem a sobreposição de métricas e KPIs, facilitando a análise de correlação através da seleção personalizada dos dados pelo utilizador.

6. Conclusão

Neste último capítulo é apresentado um resumo do trabalho realizado e respetivas conclusões. São também discutidas as limitações encontradas e efetuada uma análise a melhorias e trabalho futuro no âmbito da *dashboard* implementada.

6.1. Resumo do trabalho

O setor das tecnologias móveis encontra-se em larga expansão, sendo essencial o desenvolvimento de produtos que auxiliem o estudo e a investigação de redes móveis, no qual se inclui o 5G. É neste âmbito que se enquadra o produto desenvolvido pela empresa Allbesmart, a OAIBOX™, uma estação base 5G que requer uma plataforma *web* com uma GUI intuitiva e responsiva, que permita a visualização de gráficos e métricas em tempo real, bem como a configuração remota de parâmetros relacionados com o serviço de rede. É esta aplicação *web* o foco do trabalho apresentado na presente dissertação.

Numa primeira fase do trabalho foram identificados e apresentados conceitos tanto a nível do desenvolvimento de *dashboards* e aplicações *web*, bem como relacionados com as tecnologias de redes móveis 5G, com o intuito de contextualizar o leitor com os termos e os assuntos abordados. De seguida, procedeu-se ao levantamento e respetiva análise de trabalhos científicos que demonstram a utilização de *dashboards* e/ou aplicações *web* para monitorização e configuração em vários contextos, desde sistemas IoT em ambientes domésticos e industriais a projetos com propósitos lúdicos e didáticos.

Avançando com uma abordagem mais direcionada ao trabalho proposto, efetuou-se uma análise das tecnologias *frontend* mais em voga, sendo estas Angular, React e Vue.js. Inicialmente, foram descritos, do ponto de vista teórico, os elementos considerados mais importantes de cada framework. Posteriormente, idealizou-se um cenário hipotético que permitisse avaliar os projetos desenvolvidos com as três tecnologias. Após avaliação dos resultados dos testes, verificou-se que a *dashboard* implementada em Vue.js obteve um melhor desempenho na maioria dos parâmetros. Contudo, a escolha recaiu sobre a *framework* Angular, uma vez que se trata de uma *framework* completa e robusta e, considerando a longevidade do projeto, é particularmente adequada para projetos grandes e em contínua expansão, devido a um melhor suporte a médio/longo prazo. Outro fator que influenciou a decisão é a familiaridade da equipa de desenvolvimento com Angular.

Uma vez definida a tecnologia, procedeu-se à modelação do sistema recorrendo à metodologia ICONIX. No entanto, ICONIX é um procedimento moroso e, por uma questão de real utilidade prática no projeto, não foram

utilizados todos os diagramas, tendo sido selecionados os considerados mais proveitosos para análise e auxílio à implementação desta *dashboard*.

Para terminar, foi apresentada a *dashboard* desenvolvida, situando-a no contexto do sistema mais amplo no qual se integra. Para além de descrever as funcionalidades e interfaces do projeto, foi igualmente clarificada a comunicação deste elemento com os restantes componentes do sistema, demonstrando, assim, como se processa a monitorização e configuração remota de uma estação base 5G, a OAIBOX™, através desta plataforma *web* em tempo real.

6.2. Principais conclusões

O estudo das tecnologias de desenvolvimento *frontend* trouxe uma oportunidade de aplicação e comparação destas tecnologias, não só a nível das métricas avaliadas no contexto do presente trabalho, como também das similaridades e diferenças práticas da implementação de um mesmo projeto. Assim, verificaram-se semelhanças na estruturação do projeto e nas bibliotecas utilizadas. As diferenças prendem-se sobretudo com as particularidades na codificação inerentes a cada tecnologia, nomeadamente na comunicação entre os diferentes componentes do projeto.

Relativamente às métricas definidas para avaliação e comparação do desempenho dos projetos, verificou-se que a aplicação desenvolvida em Vue.js obteve melhores resultados na maioria dos testes. Observou-se, contudo, que o projeto desenvolvido em React necessitou de um menor número de linhas de código escrito (apesar de, no total dos ficheiros, apresentar um número substancialmente maior relativamente aos restantes projetos), teve um tempo de carregamento inicial de página menor e um melhor desempenho no relatório Lighthouse. Todavia, a escolha da tecnologia de desenvolvimento *frontend* recaiu sobre Angular que, apesar de ter resultados geralmente medianos, consiste numa *framework* completa e robusta que se coaduna com grandes projetos e de grande longevidade. Assim, a escolha da tecnologia *frontend* deve também ter em consideração o ambiente em que o projeto se insere.

O desenvolvimento desta *dashboard* insere-se num sistema maior que inclui como ponto central a OAIBOX™, uma estação base 5G desenvolvida pela empresa Allbesmart. Assim sendo, este projeto é acima de tudo um trabalho de equipa em contínua melhoria e adaptação às novas exigências, incluindo diferentes versões do produto. Como tal, é importante salientar que o trabalho discutido nesta dissertação se refere às particularidades dos produtos OAIBOX™ 40 e OAIBOX™ MAX [132].

Por fim, é importante reforçar que as funcionalidades implementadas vão ao encontro aos requisitos identificados e correspondem às exigências dos utilizadores que a utilizam rotineiramente.

6.3. Limitações e trabalho futuro

Apesar dos objetivos definidos terem sido alcançados, foram encontradas algumas limitações e, como tal, é importante refletir sobre o seu impacto no trabalho.

As limitações relativas aos testes às *frameworks* de desenvolvimento *frontend* foram discutidas no capítulo respetivo e prendem-se, sobretudo, com os constrangimentos associados à utilização de máquinas virtuais que não replicam perfeitamente um cenário real. Para além disso, o cenário hipotético não tem uma dimensão nem uma complexidade comparável com o projeto final desenvolvido.

No que diz respeito à modelação, apesar de se ter seguido a metodologia ICONIX, foram excluídos alguns elementos considerados pouco relevantes para o projeto a desenvolver. Todavia, uma alternativa mais adequada poderia passar pelo uso de uma outra metodologia que melhor se enquadrasse nas necessidades específicas desta implementação.

A nível do desenvolvimento da *dashboard* são identificadas como limitações a não realização de testes. Assim, como trabalho futuro, considera-se que seja importante a introdução e a utilização de testes unitários que permitem melhorar o processo de implementação de novas funcionalidade e de manutenção do *software*. No que diz respeito aos testes de usabilidade, importa referir que, sendo este um projeto empresarial e que já se encontra no mercado em utilização ativa, não foram realizados testes formais. No entanto, os utilizadores finais mantêm um contacto regular com a equipa de suporte, o que permite uma recolha contínua de *feedback* essencial para a correção e o melhoramento do produto.

Como foi sendo referido ao longo da dissertação, o produto OAIBOX™ encontra-se em constante evolução, acompanhando e auxiliando a igual evolução da investigação nas tecnologias de redes móveis. Assim, a *dashboard* aqui apresentada tem necessariamente de evoluir com o produto que visa monitorizar e configurar, não apenas a nível de melhorias de funcionalidades já existentes, como também adaptar-se a novas necessidades identificadas.

Para terminar, relembrar que este projeto é, acima de tudo um trabalho da equipa de desenvolvimento *web* da empresa Allbesmart, sendo que a minha participação e contribuição incidiu, principalmente, em processos de manutenção e evolução.

7. Referências Bibliográficas

[1] «Allbesmart». Acedido: 21 de Maio de 2025. [Em linha]. Disponível em: https://allbesmart.pt/

[2] «OAIBOX: The Ultimate Open Source 5G Platform for Academic and Industrial Research». Acedido: 14 de Outubro de 2024. [Em linha]. Disponível em: https://www.oaibox.com/

[3] «Online Gantt Chart Software». Acedido: 19 de Novembro de 2024. [Em linha]. Disponível em: https://www.onlinegantt.com/#/gantt/cloud

[4] «PRISMA statement». Acedido: 24 de Março de 2025. [Em linha]. Disponível em: https://www.prisma-statement.org/

[5] A. A. Rahman, Y. B. Adamu, e P. Harun, «Review on dashboard application from managerial perspective», em *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, 2017, pp. 1–5. doi: 10.1109/ICRIIS.2017.8002461.

[6] A. Kruglov, G. Succi, e I. Ishbaev, «Metrics Representation and Dashboards», em *Developing Sustainable and Energy-Efficient Software Systems*, A. Kruglov e G. Succi, Eds., Cham: Springer International Publishing, 2023, pp. 47–59. doi: 10.1007/978-3-031-11658-2_5.

[7] E. Nassirova, «Data Visualization Dashboard: Types, Examples, And Templates». Acedido: 21 de Novembro de 2024. [Em linha]. Disponível em: https://blog.coupler.io/what-is-data-visualization-dashboard/

[8] «Dashboard Charts and Graphs». Acedido: 21 de Novembro de 2024. [Em linha]. Disponível em: https://www.simplekpi.com/Resources/Dashboard-Charts-And-Graphs

[9] M. Yi e M. Sapountzis, «Essential Chart Types for Data Visualization». Acedido: 21 de Novembro de 2024. [Em linha]. Disponível em: https://www.atlassian.com/data/charts/essential-chart-types-for-data-visualization

[10] Sisense Team, «10 Useful ways to visualize your data (with Examples)». Acedido: 21 de Novembro de 2024. [Em linha]. Disponível em: https://www.sisense.com/blog/10-useful-ways-visualize-data-examples/

[11] «Grafana: The open observability platform | Grafana Labs». Acedido: 18 de Fevereiro de 2024. [Em linha]. Disponível em: https://grafana.com/

[12] «Apache ECharts». Acedido: 18 de Fevereiro de 2024. [Em linha]. Disponível em: https://echarts.apache.org/en/index.html

[13] «5G System Overview». Acedido: 23 de Agosto de 2024. [Em linha]. Disponível em: https://www.3gpp.org/technologies/5g-system-overview

[14] Cisco, «What Is 5G? - How Does 5G Network Technology Work». Acedido: 14 de Julho de 2024. [Em linha]. Disponível em: https://www.cisco.com/c/en/us/solutions/what-is-5g.html

[15] Qualcomm, «What is 5G? | Everything You Need to Know». Acedido: 14 de Julho de 2024. [Em linha]. Disponível em: https://www.qualcomm.com/5g/what-is-5g

[16] Cisco, «What is Private 5G Network?» Acedido: 14 de Julho de 2024. [Em linha]. Disponível em: https://www.cisco.com/c/en/us/solutions/private-5g-networks.html

[17] «5G Network Architectures and Technologies». Acedido: 21 de Agosto de 2024. [Em linha]. Disponível em: https://support.huawei.com/enterprise/en/doc/EDOC1100112349/cfb76756/5g-network-architectures-and-technologies

[18] «5G Network Architecture». Acedido: 23 de Agosto de 2024. [Em linha]. Disponível em: https://telcomaglobal.com/p/5g-network-architecture

[19] «5G Network Architecture». Acedido: 23 de Agosto de 2024. [Em linha]. Disponível em: https://www.geeksforgeeks.org/5g-network-architecture/

[20] E. Kim e Y. Choi, «Traffic monitoring system for 5G core network», em *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, 2019, pp. 671–673. doi: 10.1109/ICUFN.2019.8806155.

[21] «5G ORAN BASE STATION». Acedido: 23 de Agosto de 2024. [Em linha]. Disponível em: https://www.faststreamtech.com/products/5g-oran-base-station/

[22] F. Kaltenberger, G. de Souza, R. Knopp, e H. Wang, «The OpenAirInterface 5G New Radio Implementation: Current Status and Roadmap», em *WSA 2019; 23rd International ITG Workshop on Smart Antennas*, 2019, pp. 1–5.

[23] «What is a 5G Base Station?» Acedido: 20 de Agosto de 2024. [Em linha]. Disponível em: https://5gstore.com/blog/2024/06/21/what-is-a-5g-base-station/

[24] M. J. Page *et al.*, «The PRISMA 2020 statement: an updated guideline for reporting systematic reviews», *BMJ*, vol. 372, p. n71, Mar. 2021, doi: 10.1136/bmj.n71.

[25] M. J. Page *et al.*, «PRISMA 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews», *BMJ*, vol. 372, p. n160, Mar. 2021, doi: 10.1136/bmj.n160.

[26] A. Trifu, E. Smîdu, D. O. Badea, E. Bulboacă, e V. Haralambie, «Applying the PRISMA method for obtaining systematic reviews of occupational safety issues in literature search», *MATEC Web Conf.*, vol. 354, 2022, [Em linha]. Disponível em: https://doi.org/10.1051/matecconf/202235400052

[27] M. K. Swartz, «The PRISMA Statement: A Guideline for Systematic Reviews and Meta-Analyses», *Journal of Pediatric Health Care*, vol. 25, n. 1, pp. 1–2, Jan. 2011, doi: 10.1016/j.pedhc.2010.09.006.

[28] P. V Torres-Carrión, C. S. González-González, S. Aciar, e G. Rodríguez-Morales, «Methodology for systematic literature review applied to engineering and education», em *2018 IEEE Global Engineering Education Conference (EDUCON)*, 2018, pp. 1364–1373. doi: 10.1109/EDUCON.2018.8363388.

[29] «What is Scopus about?» Acedido: 1 de Abril de 2025. [Em linha]. Disponível em: https://service.elsevier.com/app/answers/detail/a_id/15100/supporthub/scopus/related/1/

[30] «IEEE Xplore». Acedido: 1 de Abril de 2025. [Em linha]. Disponível em: https://ieeexplore.ieee.org

[31] N. R. Haddaway, M. J. Page, C. C. Pritchard, e L. A. McGuinness, «PRISMA2020: An R package and Shiny app for producing PRISMA 2020-compliant flow diagrams, with interactivity for optimised digital transparency and Open Synthesis», *Campbell Systematic Reviews*, vol. 18, n. 2, p. e1230, Jun. 2022, doi: https://doi.org/10.1002/cl2.1230.

[32] L. Karagiannidis, M. Vrettopoulos, A. Amditis, E. Makri, e N. Gkonos, «A CPS-enabled architecture for sewer mining systems», em *2016 International Workshop on Cyber-physical Systems for Smart Water Networks, CySWater 2016*, 2016, pp. 1–6. doi: 10.1109/CySWater.2016.7469056.

[33] R. Reis, P. M. Santos, M. J. Sousa, N. Martins, J. Sousa, e L. Almeida, «LEM: a Tool for Large-Scale Workflow Control in Edge-Based Industry 5.0 Applications», em *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, 2023, pp. 317–323. doi: 10.1109/DCOSS-IoT58021.2023.00059.

[34] F. Wibowo, K. T. Putra, M. Syamsudin, Suheri, e C. Vasseur, «A 3D Digital Twin Dashboard for Enhanced IoT-based Smart Building Monitoring and Control», em *Proceedings - ICE3IS 2024: 4th International Conference on Electronic and Electrical Engineering and Intelligent System: Leading-Edge Technologies for Sustainable Societies*, 2024, pp. 438–443. doi: 10.1109/ICE3IS62977.2024.10775397.

[35] M. M. Talib e M. S. Croock, «Implementation of an Intelligent Power Management System for Building Using Machine Learning Model», *Ingenierie des Systemes d'Information*, vol. 30, n. 2, pp. 335–347, 2025, doi: 10.18280/isi.300205.

[36] C. Liu, P. Jiang, e W. Jiang, «Embedded-web-based remote control for RepRap-based open-source 3D printers», em *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017, pp. 3384–3389. doi: 10.1109/IECON.2017.8216573.

[37] W. Hu *et al.*, «Plug-in free web-based 3-D interactive laboratory for control engineering education», *IEEE Transactions on Industrial Electronics*, vol. 64, n. 5, pp. 3808–3818, 2017, doi: 10.1109/TIE.2016.2645141.

[38] M. Joshi, «Angular vs React vs Vue: Core Differences». Acedido: 23 de Agosto de 2024. [Em linha]. Disponível em: https://www.browserstack.com/guide/angular-vs-react-vs-vue

[39] V. Shah, «Angular Vs React Vs Vue: Which One To Choose». Acedido: 23 de Agosto de 2024. [Em linha]. Disponível em: https://www.tatvasoft.com/blog/angular-vs-react-vs-vue/

[40] H. Malhis, «DOM Rendering and how React and Angular optimize it - in short.» Acedido: 29 de Agosto de 2024. [Em linha]. Disponível em: https://www.linkedin.com/pulse/dom-rendering-how-react-angular-optimize-short-hiba-malhis/

[41] «Angular Zone.js & Change Detection: Understanding the Core Concepts». Acedido: 29 de Agosto de 2024. [Em linha]. Disponível em: https://javascript.plainenglish.io/angular-zone-js-change-detection-understanding-the-core-concepts-7c78b8aa8818

[42] «Server-side Rendering». Acedido: 29 de Agosto de 2024. [Em linha]. Disponível em: https://angular.dev/guide/ssr

[43] «Two-way binding». Acedido: 29 de Agosto de 2024. [Em linha]. Disponível em: https://angular.dev/guide/templates/two-way-binding

[44] «Using observables for streams of values». Acedido: 30 de Agosto de 2024. [Em linha]. Disponível em: https://v17.angular.io/guide/observables

[45] «Signals • Overview». Acedido: 30 de Agosto de 2024. [Em linha]. Disponível em: https://angular.dev/guide/signals

[46] «Ahead-of-time (AOT) compilation». Acedido: 30 de Agosto de 2024. [Em linha]. Disponível em: https://angular.dev/tools/cli/aot-compiler

[47] «React Server Components». Acedido: 5 de Setembro de 2024. [Em linha]. Disponível em: https://react.dev/reference/rsc/server-components

[48] «Thinking in React». Acedido: 12 de Setembro de 2024. [Em linha]. Disponível em: https://react.dev/learn/thinking-in-react

[49] A. Mittal, «Understanding data binding in React». Acedido: 12 de Setembro de 2024. [Em linha]. Disponível em: https://handsontable.com/blog/understanding-data-binding-in-react

[50] P. L. Gunawardhana, «What is Memoization in React?» Acedido: 11 de Setembro de 2024. [Em linha]. Disponível em: https://www.syncfusion.com/blogs/post/what-is-memoization-in-react

[51] «memo». Acedido: 11 de Setembro de 2024. [Em linha]. Disponível em: https://react.dev/reference/react/memo

[52] H. Bloch, «The Power of Immutable Data Structures in React: A Deep Dive». Acedido: 11 de Setembro de 2024. [Em linha]. Disponível em: https://medium.com/@Blochware/the-power-of-immutable-data-structures-in-react-a-deep-dive-c1d8ea7f11ff

[53] «Immutable.js». Acedido: 11 de Setembro de 2024. [Em linha]. Disponível em: https://immutable-js.com/

[54] «How ReactJS Handles Immutability to Improve Performance». Acedido: 11 de Setembro de 2024. [Em linha]. Disponível em: https://www.codewalnut.com/learn/how-reactjs-handles-immutability-to-improve-performance

[55] R. Purohit, «How to Streamline Your Development with React Codemod». Acedido: 4 de Setembro de 2024. [Em linha]. Disponível em: https://www.dhiwise.com/post/how-to-streamline-your-development-with-react-codemod

[56] «Vue Directives». Acedido: 28 de Setembro de 2024. [Em linha]. Disponível em: https://www.w3schools.com/vue/vue_directives.php

[57] «Custom Directives». Acedido: 18 de Setembro de 2024. [Em linha]. Disponível em: https://vuejs.org/guide/reusability/custom-directives

[58] «Server-Side Rendering (SSR)». Acedido: 28 de Setembro de 2024. [Em linha]. Disponível em: https://vuejs.org/guide/scaling-up/ssr.html

[59] «Vue.js Development ». Acedido: 3 de Outubro de 2024. [Em linha]. Disponível em: https://nuxt.com/docs/guide/concepts/vuejs-development

[60] «Lazy Loading Routes». Acedido: 1 de Outubro de 2024. [Em linha]. Disponível em: https://router.vuejs.org/guide/advanced/lazy-loading.html

[61] «Async Components». Acedido: 1 de Outubro de 2024. [Em linha]. Disponível em: https://vuejs.org/guide/components/async.html

[62] «Event Handling». Acedido: 1 de Outubro de 2024. [Em linha]. Disponível em: https://vuejs.org/guide/essentials/event-handling

[63] «6 tips for ensuring fast Vue.js performance: The essential guid». Acedido: 1 de Outubro de 2024. [Em linha]. Disponível em: https://alokai.com/blog/good-performance-with-vue-js

[64] A. Zia, «Vue.js Data Binding». Acedido: 2 de Outubro de 2024. [Em linha]. Disponível em: https://matifzia.medium.com/vue-js-data-binding-762690a08bfe

[65] «Understanding Vue.js Data Binding: A Comprehensive Guide». Acedido: 2 de Outubro de 2024. [Em linha]. Disponível em: https://dev.to/chintanonweb/understanding-vuejs-data-binding-a-comprehensive-guide-4ll0

[66] «Migration Build». Acedido: 30 de Setembro de 2024. [Em linha]. Disponível em: https://v3-migration.vuejs.org/migration-build

[67] D. Boddu, «Create Spring Boot application using initializr in 5 minutes». Acedido: 17 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/railsfactory/create-spring-boot-application-using-initializr-in-5-mins-c70fc62fd7b0

[68] «Spring Initializr Reference Guide». Acedido: 17 de Fevereiro de 2025. [Em linha]. Disponível em: https://docs.spring.io/initializr/docs/current/reference/html/

[69] «Using WebSocket to build an interactive web application». Acedido: 17 de Fevereiro de 2025. [Em linha]. Disponível em: https://spring.io/guides/gs/messaging-stomp-websocket#header

[70] P. Rajalingam, «Introduction to Web-sockets using Spring Boot and Angular». Acedido: 17 de Fevereiro de 2025. [Em linha]. Disponível em:

https://medium.com/@parthiban.rajalingam/introduction-to-web-sockets-using-spring-boot-and-angular-b11e7363f051

[71] «Annotated Controllers». Acedido: 18 de Fevereiro de 2025. [Em linha]. Disponível em: https://docs.spring.io/spring-framework/reference/web/websocket/stomp/handle-annotations.html

[72] E. Paraschiv, «The @Scheduled Annotation in Spring». Acedido: 18 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.baeldung.com/spring-scheduled-tasks

[73] «Your first Angular app ». Acedido: 19 de Fevereiro de 2025. [Em linha]. Disponível em: https://angular.dev/tutorials/first-app

[74] «How to Create an Angular Project from Scratch ?» Acedido: 19 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.geeksforgeeks.org/how-to-create-an-angular-project-from-scratch/

[75] D. Gongora, «How to Create an Angular Project from Scratch». Acedido: 19 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/@dgongoragamboa/how-to-create-an-angular-project-from-scratch-b4031abeb4de

[76] J. S. Avadāta, «WebSocket Application With Spring Boot And Angular [Part — 3, Angular WebSocket Client]». Acedido: 19 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/@zayarthant/websocket-application-with-spring-boot-and-angular-part-3-angular-websocket-client-b8d2ce3efd20

[77] «stompj». Acedido: 21 de Fevereiro de 2025. [Em linha]. Disponível em: https://github.com/stomp-js/stompjs

[78] «FormsModule». Acedido: 21 de Fevereiro de 2025. [Em linha]. Disponível em: https://angular.dev/api/forms/FormsModule

[79] «NGX-ECHARTS - Angular directive for Apache ECharts ». Acedido: 19 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.npmjs.com/package/ngx-echarts

[80] A. Prajapati, «Data Visualization with ECharts in Angular using ngx-echarts». Acedido: 19 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.ngdevelop.tech/data-visualization-with-echarts-in-angular-using-ngx-echarts/

[81] «ngx-echarts-demo documentation». Acedido: 19 de Fevereiro de 2025. [Em linha]. Disponível em: https://xieziyu.github.io/ngx-echarts/api-doc/

[82] K. Gupta, «Building a React App from Scratch: A Step-by-Step Guide». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/womenintechnology/building-a-react-app-from-scratch-a-step-by-step-guide-2a42a4be41fc

[83] «How to Build a React Project with Create React App in 10 Steps». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.freecodecamp.org/news/how-to-build-a-react-project-with-create-react-app-in-10-steps/

[84] «Creating a React App». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://react.dev/learn/creating-a-react-app

[85] «React Getting Started». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.w3schools.com/react/react_getstarted.asp

[86] Samuelgbenga, «Building a Real-Time Messaging App with Spring Boot, WebSocket, StompJs, and React (Part II) ». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/@samuelgbenga972/building-a-real-time-messaging-app-with-spring-boot-websocket-stompjs-and-react-part-ii-67ac5979c252

[87] «echarts-for-react». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.npmjs.com/package/echarts-for-react

[88] M. Usman, «Data Visualization with React & ECharts». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/analytics-vidhya/data-visualization-with-react-echarts-1fa5c765e523

[89] N. Barudwale, «ReactJS : Simple way to use Echart in React». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/@noffybarudwale/simple-way-to-use-echart-in-react-9c4267ab4a95

[90] «Quick Start». Acedido: 23 de Fevereiro de 2025. [Em linha]. Disponível em: https://vuejs.org/guide/quick-start

[91] D. Gongora, «How to Create a Vue Project from Scratch». Acedido: 23 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/@dgongoragamboa/how-to-create-a-vue-project-from-scratch-520dcc293dd2

[92] «Getting started with Vue». Acedido: 23 de Fevereiro de 2025. [Em linha]. Disponível em: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Vue_getting_started

[93] «WebSocket Configuration Between VueJs And Java Spring Boot». Acedido: 23 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/@yagmur.sahin/websocket-configuration-between-vuejs-and-java-spring-boot-74e200e1b074

[94] «Vue Echarts». Acedido: 23 de Fevereiro de 2025. [Em linha]. Disponível em: https://vuejsprojects.com/vue-echarts

[95] «Vue-ECharts». Acedido: 23 de Fevereiro de 2025. [Em linha]. Disponível em: https://github.com/ecomfe/vue-echarts?tab=readme-ov-file#readme

[96] «How to run an Ubuntu Desktop virtual machine using VirtualBox 7». Acedido: 24 de Fevereiro de 2025. [Em linha]. Disponível em: https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#1-overview

[97] V. Pimentel e B. G. Nickerson, «Communicating and Displaying Real-Time Data with WebSocket», *IEEE Internet Comput*, vol. 16, n. 4, pp. 45–53, 2012, doi: 10.1109/MIC.2012.64.

[98] D. K. Kumar, «The Ultimate Guide: Syncing Your Linux Time with NTP Server». Acedido: 28 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.fosslinux.com/126132/the-ultimate-guide-syncing-your-linux-time-with-ntp-server.htm

[99] «Install the Java Runtime Environment». Acedido: 25 de Fevereiro de 2025. [Em linha]. Disponível em: https://ubuntu.com/tutorials/install-jre#1-overview

[100] «http-server: a simple static HTTP server». Acedido: 24 de Fevereiro de 2025. [Em linha]. Disponível em: https://www.npmjs.com/package/http-server

[101] «Introduction to Lighthouse». Acedido: 25 de Fevereiro de 2025. [Em linha]. Disponível em: https://developer.chrome.com/docs/lighthouse/overview

[102] «Inspect network activity». Acedido: 27 de Fevereiro de 2025. [Em linha]. Disponível em: https://developer.chrome.com/docs/devtools/network/

[103] K. Basques e S. Emelianova, «Network features reference». Acedido: 30 de Abril de 2025. [Em linha]. Disponível em: https://developer.chrome.com/docs/devtools/network/reference

[104] «Document: DOMContentLoaded event». Acedido: 30 de Abril de 2025. [Em linha]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event

[105] «Window: load event». Acedido: 30 de Abril de 2025. [Em linha]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event

[106] P. Sonpatki e A. Udasi, «The Developer's Handbook to Centralized Logging». Acedido: 16 de Abril de 2025. [Em linha]. Disponível em: https://last9.io/blog/the-developers-handbook-to-centralized-logging/

[107] «Centralized Logging Systems – System Design». Acedido: 16 de Abril de 2025. [Em linha]. Disponível em: https://www.geeksforgeeks.org/centralized-logging-systems-system-design/

[108] Waji, «Setting up ELK Stack in Linux». Acedido: 17 de Abril de 2025. [Em linha]. Disponível em: https://dev.to/waji97/setting-up-elk-stack-in-linux-18od

[109] Cyber Tool Guardian, «What is ELK and Installing ELK stack (elasticsearch, logstash, kibana) in Ubuntu». Acedido: 17 de Abril de 2025. [Em linha]. Disponível em: https://medium.com/@cybertoolguardian/what-is-elk-and-installing-elk-stack-elasticsearch-logstash-kibana-in-ubuntu-376d60c445b3

[110] RedSwitches, «How to Install ELK Stack on Ubuntu 20.04 / 22.04». Acedido: 17 de Abril de 2025. [Em linha]. Disponível em: https://medium.com/@redswitches/how-to-install-elk-stack-on-ubuntu-20-04-22-04-2c4f13a08c63

[111] U. Iroshan, «How to configure Logstash to receive http events and visualize in kibana». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://dgujitha.medium.com/how-to-configure-logstash-to-receive-http-events-and-visualize-in-kibana-83eff28a86e0

[112] A. Tyagi, «Spring Boot Logs Aggregation and Monitoring Using ELK Stack». Acedido: 3 de Maio de 2025. [Em linha]. Disponível em: https://auth0.com/blog/spring-boot-logs-aggregation-and-monitoring-using-elk-stack/

[113] C. Lüdemann, «Logging with Angular». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://medium.com/@christianlydemann/logging-with-angular-9ef9c4ca8ffa

[114] «Making HTTP requests». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://test-utils.vuejs.org/guide/advanced/http-requests

[115] A. Sawant, «Creating a Modular HTTP Service in React: Step-by-Step Guide». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://medium.com/@aditya.sawant122/creating-a-modular-http-service-in-react-step-by-step-guide-9b8d88747678

[116] M. Ahamad, «Architecting HTTP clients in Vue.js applications for efficient network communication». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://dev.to/localeai/architecting-http-clients-in-vue-js-applications-for-effective-network-communication-1eec

[117] «Making HTTP requests». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://angular.dev/guide/http/making-requests

[118] B. Hamdi, «Run Chrome browser without CORS». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://medium.com/@beligh.hamdi/run-chrome-browser-without-cors-872747142c61

[119] «Cross-Origin Resource Sharing (CORS)». Acedido: 1 de Maio de 2025. [Em linha]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS

[120] Kentaro Ushiyama, «VS Code Counter». Acedido: 28 de Fevereiro de 2025. [Em linha]. Disponível em: https://marketplace.visualstudio.com/items?itemName=uctakeoff.vscode-counter

[121] «Angular - Deployment». Acedido: 24 de Fevereiro de 2025. [Em linha]. Disponível em: https://v17.angular.io/guide/deployment

[122] «Configuring application environments». Acedido: 24 de Fevereiro de 2025. [Em linha]. Disponível em: https://angular.dev/tools/cli/environments

[123] «Build and Deployment with Angular CLI Part 1: Configuring and Executing the Build ». Acedido: 24 de Fevereiro de 2025. [Em linha]. Disponível em: https://medium.com/@david_94373/build-and-deployment-with-angular-cli-part-1-configuring-and-executing-the-build-89704397807f

[124] «Deployment». Acedido: 22 de Fevereiro de 2025. [Em linha]. Disponível em: https://create-react-app.dev/docs/deployment/

[125] L. Veronesi, «Spring Boot & ELK Stack», 2024. Acedido: 4 de Maio de 2025. [Em linha]. Disponível em: https://medium.com/@luiz.veronesi/spring-boot-elk-stack-7ea34fa652d4

[126] M. Pena, «Demystifying CORS: A Practical Guide to Using Nginx Reverse Proxy». Acedido: 2 de Maio de 2025. [Em linha]. Disponível em: https://medium.com/@markpena737/demystifying-cors-a-practical-guide-to-using-nginx-reverse-proxy-dcf5aeea7ca6

[127] D. Rosenberg e M. Stephens, *Use case driven object modeling with UML: theory and practice*. Apress, 2007. doi: https://doi.org/10.1007/978-1-4302-0369-8.

[128] «ICONIX Process for Embedded Systems». Acedido: 18 de Outubro de 2024. [Em linha]. Disponível em: http://www.iconixsoftware.net/ICONIX_Embedded_Systems.html

[129] «What is Angular?» Acedido: 6 de Março de 2025. [Em linha]. Disponível em: https://v17.angular.io/guide/what-is-angular

[130] «Keycloak». Acedido: 18 de Março de 2025. [Em linha]. Disponível em: https://www.keycloak.org/

[131] «Bootstrap · The most popular HTML, CSS, and JS library in the world.» Acedido: 19 de Março de 2025. [Em linha]. Disponível em: https://getbootstrap.com/

[132] Allbesmart Lda, «OAIBOX TM 5G LAB Manual», Jan. 2025.

Apêndice A – Resultados da contagem de linhas de código

Angular

Total : 29 files, 15368 codes, 33 comments, 100 blanks, all 15501 lines

Directories

path	files	code	comment	blank	total
.	29	15,368	33	100	15,501
. (Files)	7	15,046	6	30	15,082
src	22	322	27	70	419
src (Files)	5	55	25	13	93
src\app	17	267	2	57	326
src\app (Files)	8	98	2	21	121
src\app\components	6	118	0	23	141
src\app\components\line-chart	3	57	0	11	68
src\app\components\telemetry	3	61	0	12	73
src\app\models	2	9	0	2	11
src\app\services	1	42	0	11	53

Languages

language	files	code	comment	blank	total
JSON	5	14,985	4	5	14,994
TypeScript	14	299	26	60	385
Markdown	1	36	0	24	60
JSON with Comments	1	25	2	1	28
HTML	4	23	0	6	29
CSS	4	0	1	4	5

Files

filename	language	code	comment	blank	total
README.md	Markdown	36	0	24	60
angular.json	JSON	109	0	1	110
package-lock.json	JSON	14,794	0	1	14,795
package.json	JSON	52	0	1	53
src/app/app-routing.module.ts	TypeScript	8	0	3	11
src/app/app.component.css	CSS	0	0	1	1
src/app/app.component.html	HTML	1	0	1	2
src/app/app.component.spec.ts	TypeScript	31	0	5	36
src/app/app.component.ts	TypeScript	10	0	2	12
src/app/app.module.server.ts	TypeScript	12	0	2	14
src/app/app.module.ts	TypeScript	29	2	5	36
src/app/app.routes.server.ts	TypeScript	7	0	2	9
src/app/components/line-chart/line-chart.component.css	CSS	0	0	1	1
src/app/components/line-chart/line-chart.component.html	HTML	1	0	1	2
src/app/components/line-chart/line-chart.component.ts	TypeScript	56	0	9	65
src/app/components/telemetry/telemetry.component.css	CSS	0	0	1	1
src/app/components/telemetry/telemetry.component.html	HTML	8	0	3	11
src/app/components/telemetry/telemetry.component.ts	TypeScript	53	0	8	61
src/app/models/telemetryMessage.ts	TypeScript	5	0	1	6
src/app/models/telemetryNameUpdated.ts	TypeScript	4	0	1	5
src/app/services/web-socket.service.ts	TypeScript	42	0	11	53
src/index.html	HTML	13	0	1	14
src/main.server.ts	TypeScript	1	0	1	2
src/main.ts	TypeScript	6	0	2	8
src/server.ts	TypeScript	35	24	8	67
src/styles.css	CSS	0	1	1	2
tsconfig.app.json	JSON	17	2	1	20
tsconfig.json	JSON with Comments	25	2	1	28
tsconfig.spec.json	JSON	13	2	1	16

React

Total : 16 files, 17897 codes, 30 comments, 80 blanks, all 18007 lines

Directories

path	files	code	comment	blank	total
.	16	17,897	30	80	18,007
. (Files)	3	17,602	0	35	17,637
public	2	45	23	2	70
src	11	250	7	43	300
src (Files)	8	86	7	19	112
src\components	3	164	0	24	188

Languages

language	files	code	comment	blank	total
JSON	3	17,589	0	3	17,592
JavaScript	8	204	7	35	246
CSS	2	45	0	8	53
Markdown	1	38	0	33	71
HTML	1	20	23	1	44
XML	1	1	0	0	1

Files

filename	language	code	comment	blank	total
README.md	Markdown	38	0	33	71
package-lock.json	JSON	17,516	0	1	17,517
package.json	JSON	48	0	1	49
public/index.html	HTML	20	23	1	44
public/manifest.json	JSON	25	0	1	26
src/App.css	CSS	33	0	6	39
src/App.js	JavaScript	8	0	3	11
src/App.test.js	JavaScript	7	0	2	9
src/components/LineChart.js	JavaScript	56	0	3	59
src/components/Telemetry.js	JavaScript	75	0	11	86
src/components/WebSocket.js	JavaScript	33	0	10	43
src/index.css	CSS	12	0	2	14
src/index.js	JavaScript	12	3	3	18
src/logo.svg	XML	1	0	0	1
src/reportWebVitals.js	JavaScript	12	0	2	14
src/setupTests.js	JavaScript	1	4	1	6

Vue.js

Total : 16 files, 5287 codes, 3 comments, 68 blanks, all 5358 lines

Directories

path	files	code	comment	blank	total
.	16	5,287	3	68	5,358
. (Files)	8	4,970	1	29	5,000
src	8	317	2	39	358
src (Files)	2	10	0	5	15
src\assets	3	102	2	20	124
src\components	3	205	0	14	219

Languages

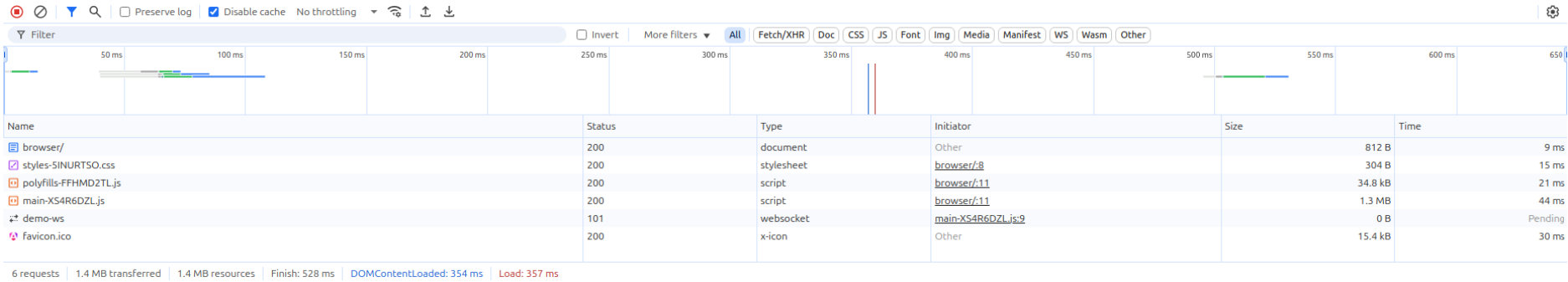
language	files	code	comment	blank	total
JSON	3	4,893	0	4	4,897
vue	4	211	0	16	227
CSS	2	101	2	19	122
JavaScript	3	38	1	10	49
Markdown	1	22	0	14	36
HTML	1	13	0	3	16
JSON with Comments	1	8	0	1	9
XML	1	1	0	1	2

Files

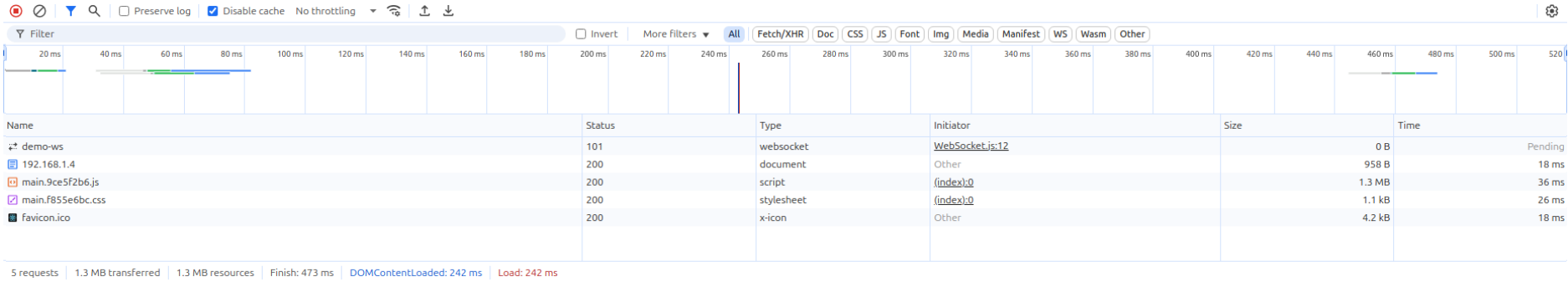
filename	language	code	comment	blank	total
.prettierrc.json	JSON	6	0	2	8
README.md	Markdown	22	0	14	36
eslint.config.js	JavaScript	16	0	4	20
index.html	HTML	13	0	3	16
jsconfig.json	JSON with Comments	8	0	1	9
package-lock.json	JSON	4,854	0	1	4,855
package.json	JSON	33	0	1	34
src/App.vue	vue	6	0	2	8
src/assets/base.css	CSS	71	2	14	87
src/assets/logo.svg	XML	1	0	1	2
src/assets/main.css	CSS	30	0	5	35
src/components/LineChartComponent.vue	vue	86	0	7	93
src/components/TelemetryComponent.vue	vue	77	0	4	81
src/components/WebSocketComponent.vue	vue	42	0	3	45
src/main.js	JavaScript	4	0	3	7
vite.config.js	JavaScript	18	1	3	22

Apêndice B – Resultados obtidos no separador Network

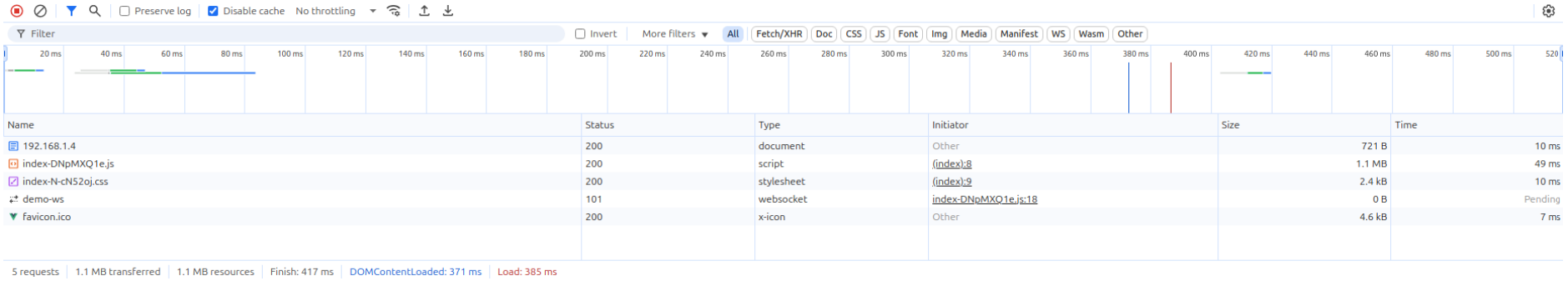
Angular



React



Vue.js


Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
WS
Wasm
Other
20 ms
40 ms
60 ms
80 ms
100 ms
120 ms
140 ms
160 ms
180 ms
200 ms
220 ms
240 ms
260 ms
280 ms
300 ms
320 ms
340 ms
360 ms
380 ms
400 ms
420 ms
440 ms
460 ms
480 ms
500 ms
520
Name
Status
Type
Initiator
Size
Time
192.168.1.4
200
document
Other
721 B
10 ms
index-DNpMXQ1e.js
200
script
(index):8
1.1 MB
49 ms
index-N-cN52oj.css
200
stylesheet
(index):9
2.4 kB
10 ms
demo-ws
101
websocket
index-DNpMXQ1e.js:18
0 B
Pending
favicon.ico
200
x-icon
Other
4.6 kB
7 ms
5 requests
1.1 MB transferred
1.1 MB resources
Finish: 417 ms
DOMContentLoaded: 371 ms
Load: 385 ms

Apêndice C – Resultados de data e hora obtidos no ambiente de teste 1

Angular

Teste	Timestamp Servidor (criação)	Timestamp Cliente (apresentação)	Latência (ms)
Angular 1	1745354028918	1745354028962	44
Angular 2	1745354033917	1745354033920	3
Angular 3	1745354038922	1745354038925	3
Angular 4	1745354043921	1745354043940	19
Angular 5	1745354048918	1745354048929	11
Angular 6	1745354053918	1745354053920	2
Angular 7	1745354058917	1745354058920	3
Angular 8	1745354063942	1745354063947	5
Angular 9	1745354068919	1745354068922	3
Angular 10	1745354073918	1745354073921	3

Teste	Timestamp Cliente (pedido)	Timestamp Servidor (processar pedido)	Timestamp Cliente (apresentação)	Latência cliente-servidor (ms)	Latência (ms)
Angular 1	1745354198049	1745354198060	1745354198067	11	18
Angular 2	1745354200989	1745354200992	1745354200995	3	6
Angular 3	1745354203510	1745354203514	1745354203517	4	7
Angular 4	1745354205559	1745354205562	1745354205566	3	7
Angular 5	1745354207250	1745354207253	1745354207255	3	5
Angular 6	1745354209132	1745354209152	1745354209184	20	52
Angular 7	1745354213449	1745354213453	1745354213456	4	7
Angular 8	1745354216070	1745354216073	1745354216075	3	5
Angular 9	1745354218279	1745354218282	1745354218285	3	6
Angular 10	1745354221220	1745354221223	1745354221228	3	8

React

Teste	Timestamp Servidor (criação)	Timestamp Cliente (apresentação)	Latência (ms)
React 1	1745354433919	1745354433934	15
React 2	1745354438932	1745354438935	3
React 3	1745354446248	1745354446269	21
React 4	1745354448918	1745354448924	6
React 5	1745354453918	1745354453927	9
React 6	1745354458920	1745354458926	6
React 7	1745354463918	1745354463925	7
React 8	1745354468918	1745354468923	5
React 9	1745354473917	1745354473928	11
React 10	1745354478918	1745354478923	5

Teste	Timestamp Cliente (pedido)	Timestamp Servidor (processar pedido)	Timestamp Cliente (apresentação)	Latência cliente-servidor (ms)	Latência (ms)
React 1	1745354606170	1745354606174	1745354606188	4	18
React 2	1745354608016	1745354608021	1745354608027	5	11
React 3	1745354609776	1745354609780	1745354609788	4	12
React 4	1745354611557	1745354611559	1745354611566	2	9
React 5	1745354613586	1745354613589	1745354613600	3	14
React 6	1745354615447	1745354615452	1745354615456	5	9
React 7	1745354617417	1745354617419	1745354617429	2	12
React 8	1745354619827	1745354619830	1745354619846	3	19
React 9	1745354627017	1745354627020	1745354627029	3	12
React 10	1745354675877	1745354675881	1745354675901	4	24

Vue.js

Teste	Timestamp Servidor (criação)	Timestamp Cliente (apresentação)	Latência (ms)
Vue.js 1	1745354908917	1745354908920	3
Vue.js 2	1745354913918	1745354913923	5
Vue.js 3	1745354918943	1745354918952	9
Vue.js 4	1745354923918	1745354923922	4
Vue.js 5	1745354928923	1745354928927	4
Vue.js 6	1745354933939	1745354933942	3
Vue.js 7	1745354938923	1745354938925	2
Vue.js 8	1745354943920	1745354943926	6
Vue.js 9	1745354948922	1745354948926	4
Vue.js 10	1745354953917	1745354953921	4

Teste	Timestamp Cliente (pedido)	Timestamp Servidor (processar pedido)	Timestamp Cliente (apresentação)	Latência cliente-servidor (ms)	Latência (ms)
Vue.js 1	1745355027947	1745355027951	1745355027954	4	7
Vue.js 2	1745355030007	1745355030011	1745355030013	4	6
Vue.js 3	1745355031950	1745355031952	1745355031955	2	5
Vue.js 4	1745355034040	1745355034043	1745355034067	3	27
Vue.js 5	1745355035908	1745355035910	1745355035912	2	4
Vue.js 6	1745355037657	1745355037663	1745355037667	6	10
Vue.js 7	1745355039410	1745355039413	1745355039425	3	15
Vue.js 8	1745355041287	1745355041292	1745355041299	5	12
Vue.js 9	1745355043567	1745355043571	1745355043573	4	6
Vue.js 10	1745355045948	1745355045952	1745355045954	4	6

Apêndice D – Resultados de data e hora obtidos no ambiente de teste 2

Angular

Teste	Timestamp Servidor (criação)	Timestamp Cliente (apresentação)	Latência (ms)
Angular 1	1745525569632	1745525569681	49
Angular 2	1745525574632	1745525574642	10
Angular 3	1745525579633	1745525579642	9
Angular 4	1745525584639	1745525584650	11
Angular 5	1745525589634	1745525589644	10
Angular 6	1745525594633	1745525594639	6
Angular 7	1745525599634	1745525599643	9
Angular 8	1745525604633	1745525604642	9
Angular 9	1745525609632	1745525609648	16
Angular 10	1745525614637	1745525614646	9

Teste	Timestamp Cliente (pedido)	Timestamp Servidor (processar pedido)	Timestamp Cliente (apresentação)	Latência cliente-servidor (ms)	Latência (ms)
Angular 1	1745525616772	1745525616850	1745525616858	78	86
Angular 2	1745525618880	1745525618884	1745525618892	4	12
Angular 3	1745525621171	1745525621175	1745525621183	4	12
Angular 4	1745525623782	1745525623788	1745525623796	6	14
Angular 5	1745525625602	1745525625605	1745525625614	3	12
Angular 6	1745525627334	1745525627335	1745525627342	1	8
Angular 7	1745525629250	1745525629256	1745525629261	6	11
Angular 8	1745525631239	1745525631245	1745525631252	6	13
Angular 9	1745525633202	1745525633207	1745525633215	5	13
Angular 10	1745525636610	1745525636615	1745525636621	5	11

React

Teste	Timestamp Servidor (criação)	Timestamp Cliente (apresentação)	Latência (ms)
React 1	1745525644635	1745525644675	40
React 2	1745525649750	1745525649761	11
React 3	1745525654633	1745525654651	18
React 4	1745525659633	1745525659644	11
React 5	1745525664633	1745525664648	15
React 6	1745525669633	1745525669647	14
React 7	1745525674633	1745525674647	14
React 8	1745525679634	1745525679675	41
React 9	1745525684636	1745525684668	32
React 10	1745525689632	1745525689642	10

Teste	Timestamp Cliente (pedido)	Timestamp Servidor (processar pedido)	Timestamp Cliente (apresentação)	Latência cliente-servidor (ms)	Latência (ms)
React 1	1745525691542	1745525691547	1745525691562	5	20
React 2	1745525693733	1745525693738	1745525693751	5	18
React 3	1745525695812	1745525695851	1745525695867	39	55
React 4	1745525697624	1745525697635	1745525697648	11	24
React 5	1745525699906	1745525699910	1745525699929	4	23
React 6	1745525702603	1745525702608	1745525702622	5	19
React 7	1745525705161	1745525705165	1745525705186	4	25
React 8	1745525707504	1745525707508	1745525707522	4	18
React 9	1745525709765	1745525709769	1745525709813	4	48
React 10	1745525712582	1745525712586	1745525712603	4	21

Vue

Teste	Timestamp Servidor (criação)	Timestamp Cliente (apresentação)	Latência (ms)
Vue.js 1	1745525719633	1745525719644	11
Vue.js 2	1745525724636	1745525724645	9
Vue.js 3	1745525729634	1745525729643	9
Vue.js 4	1745525734634	1745525734650	16
Vue.js 5	1745525739632	1745525739640	8
Vue.js 6	1745525744633	1745525744639	6
Vue.js 7	1745525749634	1745525749647	13
Vue.js 8	1745525754633	1745525754647	14
Vue.js 9	1745525759637	1745525759645	8
Vue.js 10	1745525764635	1745525764645	10

Teste	Timestamp Cliente (pedido)	Timestamp Servidor (processar pedido)	Timestamp Cliente (apresentação)	Latência cliente-servidor (ms)	Latência (ms)
Vue.js 1	1745525769045	1745525769048	1745525769056	3	11
Vue.js 2	1745525771905	1745525771908	1745525771914	3	9
Vue.js 3	1745525773754	1745525773763	1745525773771	9	17
Vue.js 4	1745525775575	1745525775577	1745525775585	2	10
Vue.js 5	1745525778014	1745525778018	1745525778025	4	11
Vue.js 6	1745525779950	1745525779957	1745525779971	7	21
Vue.js 7	1745525781975	1745525781997	1745525782003	22	28
Vue.js 8	1745525783944	1745525783962	1745525783970	18	26
Vue.js 9	1745525786405	1745525786409	1745525786417	4	12
Vue.js 10	1745525788955	1745525788959	1745525788966	4	11